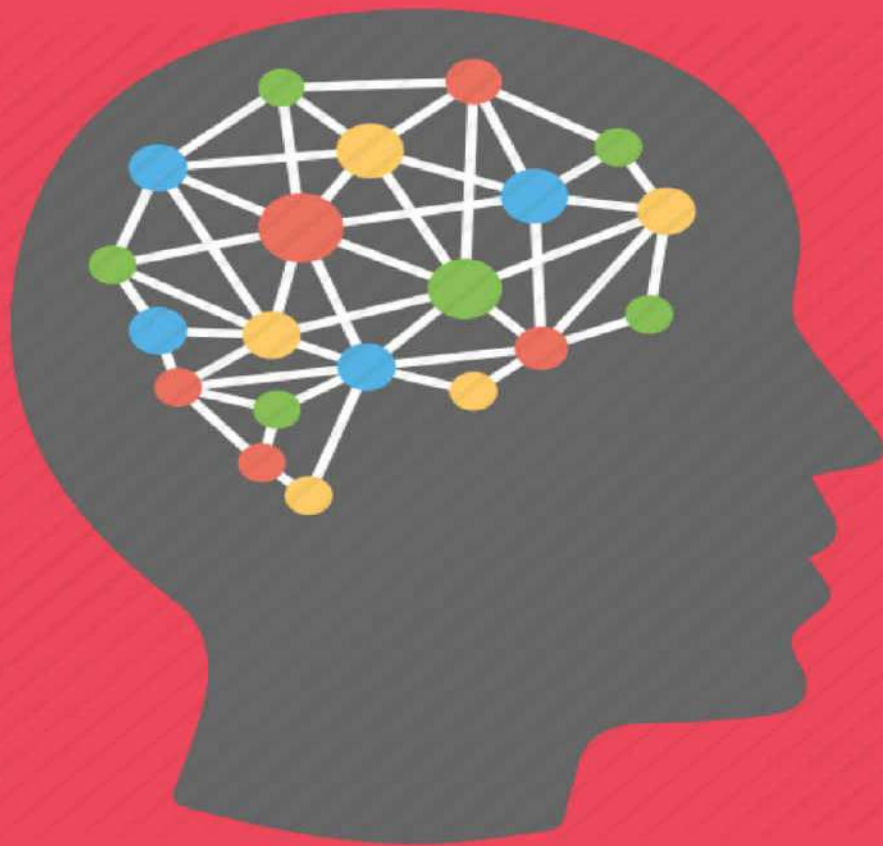


TensorFlow



LEARN IN 1 DAY

KRISHNA RUNGTA

TensorFlow in 1 Day: Make your own Neural Network

By Krishna Rungta

Copyright 2018 - All Rights Reserved – Krishna Rungta

ALL RIGHTS RESERVED. No part of this publication may be reproduced or transmitted in any form whatsoever, electronic, or mechanical, including photocopying, recording, or by any informational storage or retrieval system without express written, dated and signed permission from the author.

Table Of Content

Chapter 1: What is Deep learning?

1. [What is Deep learning?](#)
2. [Deep learning Process](#)
3. [Classification of Neural Networks](#)
4. [Types of Deep Learning Networks](#)
5. [Feed-forward neural networks](#)
6. [Recurrent neural networks \(RNNs\)](#)
7. [Convolutional neural networks \(CNN\)](#)

Chapter 2: Machine Learning vs Deep Learning

1. [What is AI?](#)
2. [What is ML?](#)
3. [What is Deep Learning?](#)
4. [Machine Learning Process](#)
5. [Deep Learning Process](#)
6. [Automate Feature Extraction using DL](#)
7. [Difference between Machine Learning and Deep Learning](#)
8. [When to use ML or DL?](#)

Chapter 3: What is TensorFlow?

1. [What is TensorFlow?](#)
2. [History of TensorFlow](#)
3. [TensorFlow Architecture](#)
4. [Where can Tensorflow run?](#)
5. [Introduction to Components of TensorFlow](#)

6. [Why is TensorFlow popular?](#)
7. [List of Prominent Algorithms supported by TensorFlow](#)

Chapter 4: Comparison of Deep Learning Libraries

1. [8 Best Deep learning Libraries /Framework](#)
2. [MICROSOFT COGNITIVE TOOLKIT\(CNTK\)](#)
3. [TensorFlow Vs Theano Vs Torch Vs Keras Vs infer.net Vs CNTK Vs MXNet Vs Caffe: Key Differences](#)

Chapter 5: How to Download and Install TensorFlow Windows and Mac

1. [TensorFlow Versions](#)
2. [Install Anaconda](#)
3. [Create .yaml file to install Tensorflow and dependencies](#)
4. [Launch Jupyter Notebook](#)
5. [Jupyter with the main conda environment](#)

Chapter 6: Jupyter Notebook Tutorial

1. [What is Jupyter Notebook?](#)
2. [Jupyter Notebook App](#)
3. [How to use Jupyter](#)

Chapter 7: Tensorflow on AWS

1. [PART 1: Set up a key pair](#)
2. [PART 2: Set up a security group](#)
3. [Launch your instance \(Windows users\)](#)
4. [Part 4: Install Docker](#)
5. [Part 5: Install Jupyter](#)
6. [Part 6: Close connection](#)

Chapter 8: TensorFlow Basics: Tensor, Shape, Type, Graph, Sessions & Operators

1. [What is a Tensor?](#)
2. [Representation of a Tensor](#)
3. [Types of Tensor](#)
4. [Shape of tensor](#)
5. [Type of data](#)
6. [Creating operator](#)
7. [Variables](#)

Chapter 9: Tensorboard: Graph Visualization with Example

Chapter 10: NumPy

1. [What is NumPy?](#)
2. [Why use NumPy?](#)
3. [How to install NumPy?](#)
4. [Mathematical Operations on an Array](#)
5. [Shape of Array](#)
6. [np.zeros and np.ones](#)
7. [Reshape and Flatten Data](#)
8. [hstack and vstack](#)

Chapter 11: Pandas

1. [What is Pandas?](#)
2. [Why use Pandas?](#)
3. [How to install Pandas?](#)
4. [What is a data frame?](#)
5. [What is a Series?](#)
6. [Concatenation](#)

Chapter 12: Scikit-Learn

1. [What is Scikit-learn?](#)
2. [Download and Install scikit-learn](#)
3. [Machine learning with scikit-learn](#)
4. [Step 1\) Import the data](#)
5. [Step 2\) Create the train/test set](#)
6. [Step 3\) Build the pipeline](#)
7. [Step 4\) Using our pipeline in a grid search](#)

Chapter 13: Linear Regression

1. [Linear regression](#)
2. [How to train a linear regression model](#)
3. [How to train a Linear Regression with TensorFlow](#)
4. [Pandas](#)
5. [Numpy Solution](#)
6. [Tensorflow solution](#)

Chapter 14: Linear Regression Case Study

1. [Summary statistics](#)
2. [Facets Overview](#)
3. [Facets Deep Dive](#)
4. [Install Facet](#)
5. [Overview](#)
6. [Graph](#)
7. [Facets Deep Dive](#)

Chapter 15: Linear Classifier in TensorFlow

1. [What is Linear Classifier?](#)

2. [How Binary classifier works?](#)
3. [How to Measure the performance of Linear Classifier?](#)
4. [Linear Classifier with TensorFlow](#)

Chapter 16: Kernel Methods

1. [Why do you need Kernel Methods?](#)
2. [What is a Kernel in machine learning?](#)
3. [Type of Kernel Methods](#)
4. [Train Gaussian Kernel classifier with TensorFlow](#)

Chapter 17: TensorFlow ANN (Artificial Neural Network)

1. [What is Artificial Neural Network?](#)
2. [Neural Network Architecture](#)
3. [Limitations of Neural Network](#)
4. [Example Neural Network in TensorFlow](#)
5. [Train a neural network with TensorFlow](#)

Chapter 18: ConvNet(Convolutional Neural Network): TensorFlow Image Classification

1. [What is Convolutional Neural Network?](#)
2. [Architecture of a Convolutional Neural Network](#)
3. [Components of Convnets](#)
4. [Train CNN with TensorFlow](#)

Chapter 19: Autoencoder with TensorFlow

1. [What is an Autoencoder?](#)
2. [How does Autoencoder work?](#)
3. [Stacked Autoencoder Example](#)

4. [Build an Autoencoder with TensorFlow](#)

Chapter 20: RNN(Recurrent Neural Network) TensorFlow

1. [What do we need an RNN?](#)
2. [What is RNN?](#)
3. [Build an RNN to predict Time Series in TensorFlow](#)

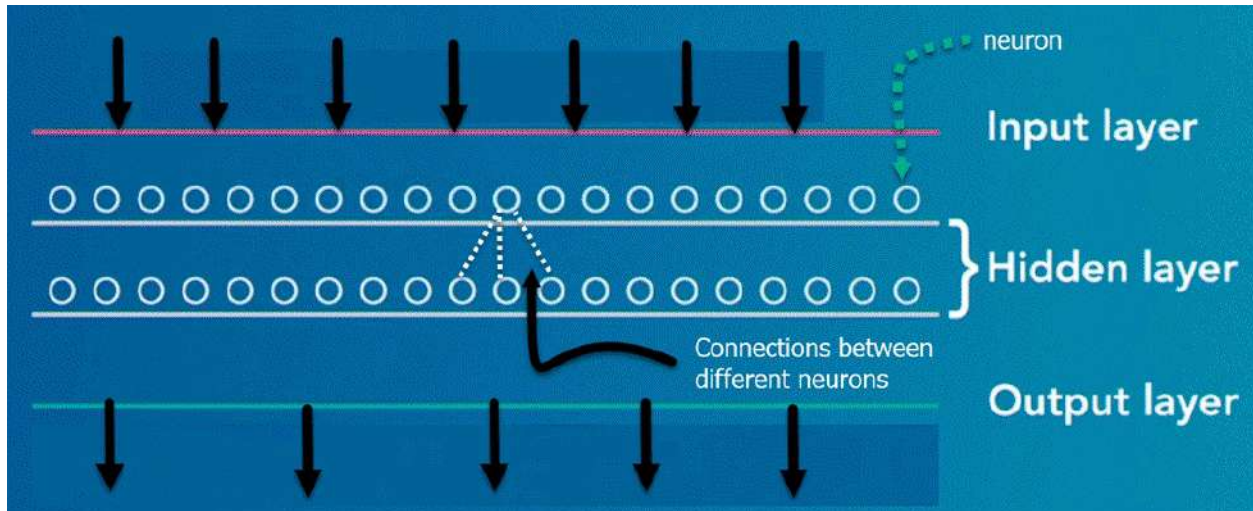
Chapter 1: What is Deep learning?

What is Deep learning?

Deep learning is a computer software that **mimics the network of neurons in a brain**. It is a subset of machine learning and is called deep learning because it makes use of deep **neural networks**.

Deep learning algorithms are constructed with connected layers.

- The first layer is called the Input Layer
- The last layer is called the Output Layer
- All layers in between are called Hidden Layers. The word deep means the network join neurons in more than two layers.



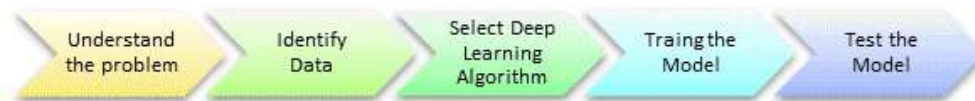
Each Hidden layer is composed of neurons. The neurons are connected to each other. The neuron will process and then propagate the input signal it receives the layer above it. The strength of the signal given the neuron in the next layer depends on the weight, bias and activation function.

The network consumes large amounts of input data and operates them through

multiple layers; the network can learn increasingly complex features of the data at each layer.

Deep learning Process

A deep neural network provides state-of-the-art accuracy in many tasks, from object detection to speech recognition. They can learn automatically, without predefined knowledge explicitly coded by the programmers.



To grasp the idea of deep learning, imagine a family, with an infant and parents. The toddler points objects with his little finger and always says the word 'cat.' As its parents are concerned about his education, they keep telling him 'Yes, that is a cat' or 'No, that is not a cat.' The infant persists in pointing objects but becomes more accurate with 'cats.' The little kid, deep down, does not know why he can say it is a cat or not. He has just learned how to hierarchies complex features coming up with a cat by looking at the pet overall and continue to focus on details such as the tails or the nose before to make up his mind.

A neural network works quite the same. Each layer represents a deeper level of knowledge, i.e., the hierarchy of knowledge. A neural network with four layers will learn more complex feature than with that with two layers.

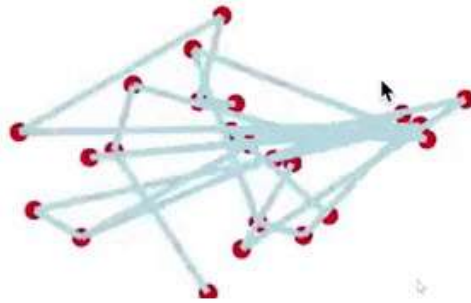
The learning occurs in two phases.

- The first phase consists of applying a nonlinear transformation of the input and create a statistical model as output.
- The second phase aims at improving the model with a mathematical method known as derivative.

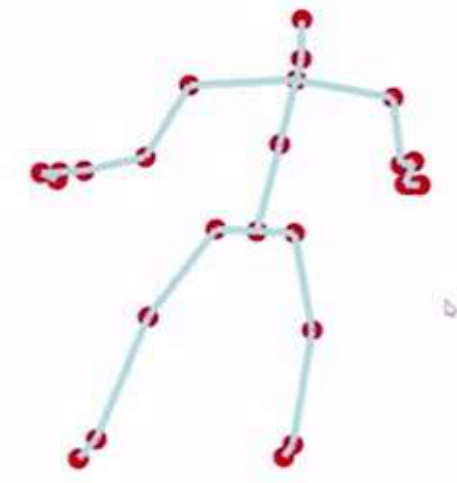
The neural network repeats these two phases hundreds to thousands of time until it has reached a tolerable level of accuracy. The repeat of this two-phase is called

an iteration.

To give an example, take a look at the motion below, the model is trying to learn how to dance. After 10 minutes of training, the model does not know how to dance, and it looks like a scribble.



After 48 hours of learning, the computer masters the art of dancing.



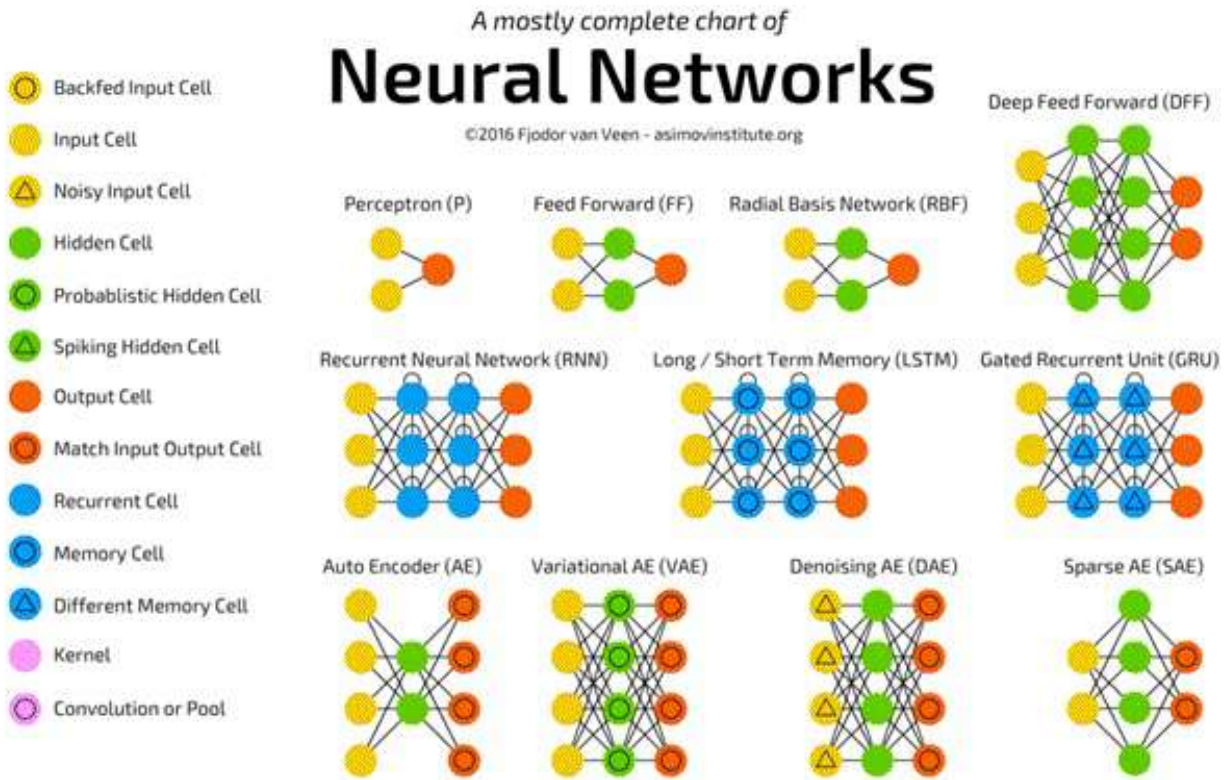
Classification of Neural Networks

Shallow neural network: The Shallow neural network has only one hidden layer between the input and output.

Deep neural network: Deep neural networks have more than one layer. For instance, Google LeNet model for image recognition counts 22 layers.

Nowadays, deep learning is used in many ways like a driverless car, mobile phone, Google Search Engine, Fraud detection, TV, and so on.

Types of Deep Learning Networks



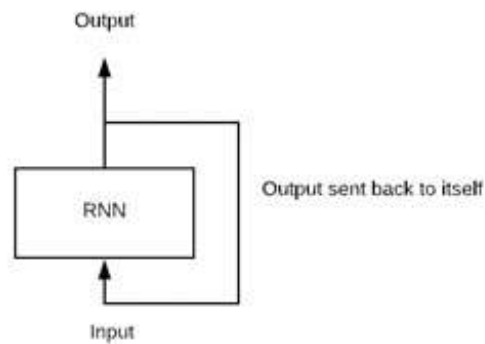
Feed-forward neural networks

The simplest type of artificial neural network. With this type of architecture, information flows in only one direction, forward. It means, the information's flows starts at the input layer, goes to the "hidden" layers, and end at the output layer. The network

does not have a loop. Information stops at the output layers.

Recurrent neural networks (RNNs)

RNN is a multi-layered neural network that can store information in context nodes, allowing it to learn data sequences and output a number or another sequence. In simple words it an Artificial neural networks whose connections between neurons include loops. RNNs are well suited for processing sequences of inputs.



Example, if the task is to predict the next word in the sentence "Do you want a.....?"

- The RNN neurons will receive a signal that point to the start of the sentence.
- The network receives the word "Do" as an input and produces a vector of the number. This vector is fed back to the neuron to provide a memory to the network. This stage helps the network to remember it received "Do" and it received it in the first position.
- The network will similarly proceed to the next words. It takes the word "you" and "want." The state of the neurons is updated upon receiving each word.
- The final stage occurs after receiving the word "a." The neural network will

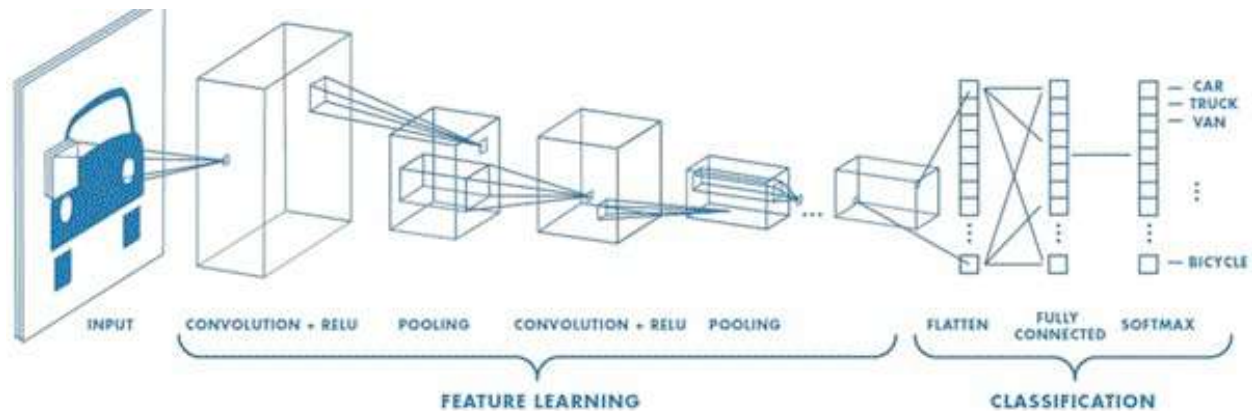
provide a probability for each English word that can be used to complete the sentence. A well-trained RNN probably assigns a high probability to "café," "drink," "burger," etc.

Common uses of RNN

- Help securities traders to generate analytic reports
- Detect abnormalities in the contract of financial statement
- Detect fraudulent credit-card transaction
- Provide a caption for images
- Power chatbots
- The standard uses of RNN occur when the practitioners are working with time-series data or sequences (e.g., audio recordings or text).

Convolutional neural networks (CNN)

CNN is a multi-layered neural network with a unique architecture designed to extract increasingly complex features of the data at each layer to determine the output. CNN's are well suited for perceptual tasks.



CNN is mostly used when there is an unstructured data set (e.g., images) and the practitioners need to extract information from it

For instance, if the task is to predict an image caption:

- The CNN receives an image of let's say a cat, this image, in computer term, is a collection of the pixel. Generally, one layer for the greyscale picture and three layers for a color picture.
- During the feature learning (i.e., hidden layers), the network will identify unique features, for instance, the tail of the cat, the ear, etc.
- When the network thoroughly learned how to recognize a picture, it can provide a probability for each image it knows. The label with the highest probability will become the prediction of the network.

Reinforcement Learning

Reinforcement learning is a subfield of machine learning in which systems are trained by receiving virtual "rewards" or "punishments," essentially learning by trial and error. Google's DeepMind has used reinforcement learning to beat a human champion in the Go games. Reinforcement learning is also used in video games to improve the gaming experience by providing smarter bot.

One of the most famous algorithms are:

- Q-learning
- Deep Q network
- State-Action-Reward-State-Action (SARSA)
- Deep Deterministic Policy Gradient (DDPG)

Applications/ Examples of deep learning applications

AI in Finance: The financial technology sector has already started using AI to save time, reduce costs, and add value. Deep learning is changing the lending industry by using more robust credit scoring. Credit decision-makers can use AI for robust credit lending applications to achieve faster, more accurate risk assessment, using machine intelligence to factor in the character and capacity of applicants.

Underwrite is a Fintech company providing an AI solution for credit makers company. underwrite.ai uses AI to detect which applicant is more likely to pay back a loan. Their approach radically outperforms traditional methods.

AI in HR: Under Armour, a sportswear company revolutionizes hiring and modernizes the candidate experience with the help of AI. In fact, Under Armour Reduces hiring time for its retail stores by 35%. Under Armour faced a growing popularity interest back in 2012. They had, on average, 30000 resumes a month. Reading all of those applications and begin to start the screening and interview process was taking too long. The lengthy process to get people hired and on-boarded impacted Under Armour's ability to have their retail stores fully staffed, ramped and ready to operate.

At that time, Under Armour had all of the 'must have' HR technology in place such as transactional solutions for sourcing, applying, tracking and onboarding but those tools weren't useful enough. Under armour choose **HireVue**, an AI provider for HR solution, for both on-demand and live interviews. The results were bluffing; they managed to decrease by 35% the time to fill. In return, the hired higher quality staffs.

AI in Marketing: AI is a valuable tool for customer service management and

personalization challenges. Improved speech recognition in call-center management and call routing as a result of the application of AI techniques allows a more seamless experience for customers.

For example, deep-learning analysis of audio allows systems to assess a customer's emotional tone. If the customer is responding poorly to the AI chatbot, the system can be rerouted the conversation to real, human operators that take over the issue.

Apart from the three examples above, AI is widely used in other sectors/industries.

Why is Deep Learning Important?

Deep learning is a powerful tool to make prediction an actionable result. Deep learning excels in pattern discovery (unsupervised learning) and knowledge-based prediction. Big data is the fuel for deep learning. When both are combined, an organization can reap unprecedented results in terms of productivity, sales, management, and innovation.

Deep learning can outperform traditional methods. For instance, deep learning algorithms are 41% more accurate than machine learning algorithms in image classification, 27% more accurate in facial recognition and 25% in voice recognition.

Limitations of deep learning

Data labeling

Most current AI models are trained through "supervised learning." It means that humans must label and categorize the underlying data, which can be a sizable and error-prone chore. For example, companies developing self-driving-car technologies are hiring hundreds of people to manually annotate hours of video feeds from prototype vehicles to help train these systems.

Obtain huge training datasets

It has been shown that simple deep learning techniques like CNN can, in some cases, imitate the knowledge of experts in medicine and other fields. The current wave of machine learning, however, requires training data sets that are not only labeled but also sufficiently broad and universal.

Deep-learning methods required thousands of observation for models to become relatively good at classification tasks and, in some cases, millions for them to perform at the level of humans. Without surprise, deep learning is famous in giant tech companies; they are using big data to accumulate petabytes of data. It allows them to create an impressive and highly accurate deep learning model.

Explain a problem

Large and complex models can be hard to explain, in human terms. For instance, why a particular decision was obtained. It is one reason that acceptance of some AI tools are slow in application areas where interpretability is useful or indeed required.

Furthermore, as the application of AI expands, regulatory requirements could also drive the need for more explainable AI models.

Summary

Deep learning is the new state-of-the-art for artificial intelligence. Deep learning architecture is composed of an input layer, hidden layers, and an output layer. The word deep means there are more than two fully connected layers.

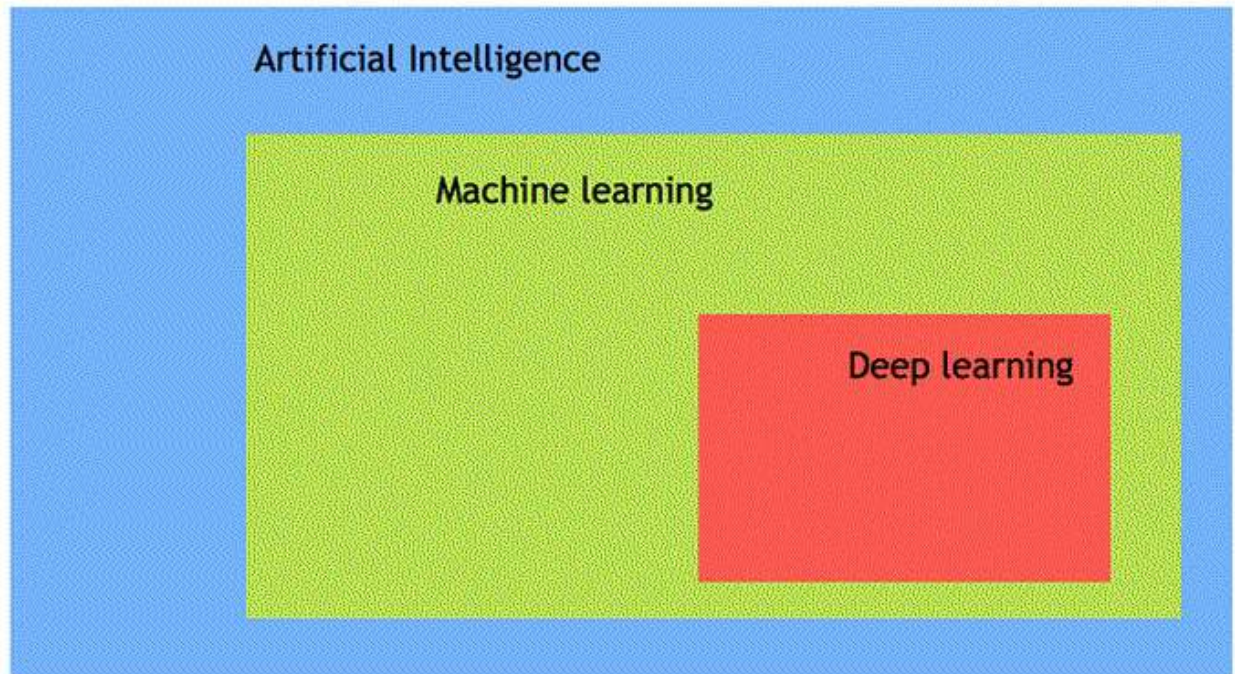
There is a vast amount of neural network, where each architecture is designed to perform a given task. For instance, CNN works very well with pictures, RNN provides impressive results with time series and text analysis.

Deep learning is now active in different fields, from finance to marketing, supply chain, and marketing. Big firms are the first one to use deep learning because they have already a large pool of data. Deep learning requires to have an extensive training dataset.

Chapter 2: Machine Learning vs Deep Learning

What is AI?

Artificial intelligence is imparting a cognitive ability to a machine. The benchmark for AI is the human intelligence regarding reasoning, speech, and vision. This benchmark is far off in the future.



AI has three different levels:

1. **Narrow AI:** A artificial intelligence is said to be narrow when the machine can perform a specific task better than a human. The current research of AI is here now
2. **General AI:** An artificial intelligence reaches the general state when it can perform any intellectual task with the same accuracy level as a human

would

3. **Active AI:** An AI is active when it can beat humans in many tasks

Early AI systems used pattern matching and expert systems.

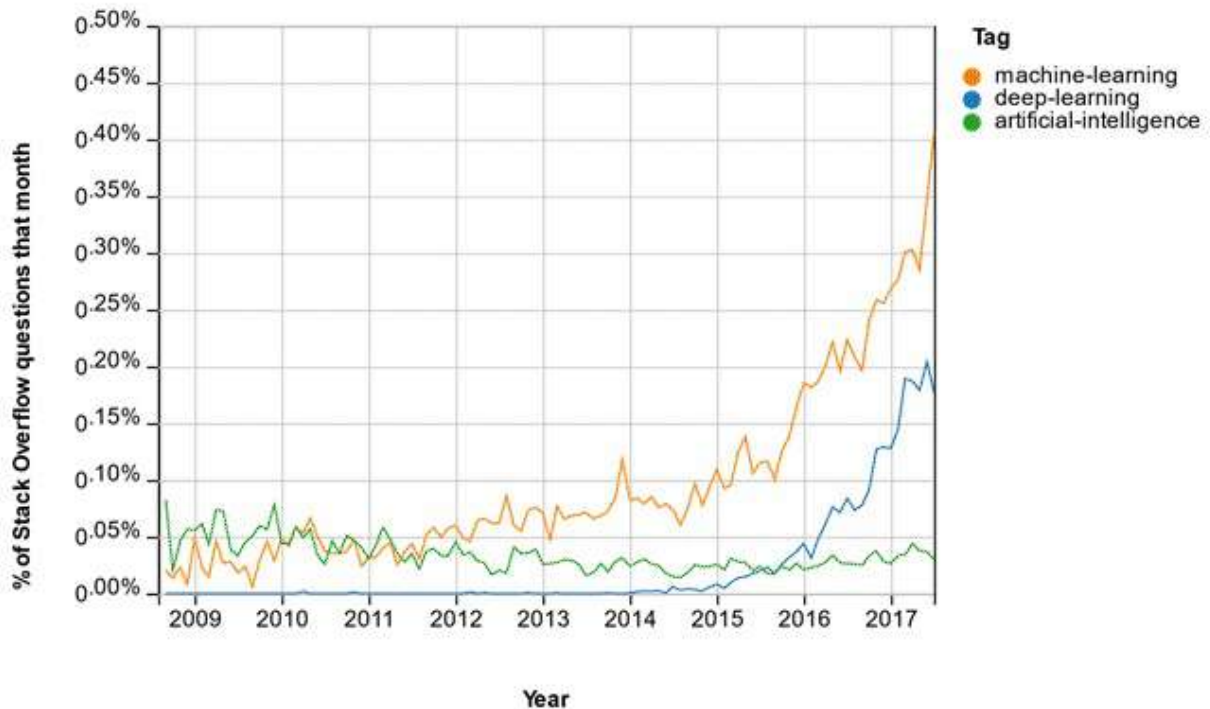
What is ML?

Machine learning is the best tool so far to analyze, understand and identify a pattern in the data. One of the main ideas behind machine learning is that the computer can be trained to automate tasks that would be exhaustive or impossible for a human being. The clear breach from the traditional analysis is that machine learning can take decisions with minimal human intervention.

Machine learning uses data to feed an algorithm that can understand the relationship between the input and the output. When the machine finished learning, it can predict the value or the class of new data point.

What is Deep Learning?

Deep learning is a computer software that mimics the network of neurons in a brain. It is a subset of machine learning and is called deep learning because it makes use of deep neural networks. The machine uses different layers to learn from the data. The depth of the model is represented by the number of layers in the model. Deep learning is the new state of the art in term of AI. In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other



Machine Learning Process

Imagine you are meant to build a program that recognizes objects. To train the model, you will use a **classifier**. A classifier uses the features of an object to try identifying the class it belongs to.

In the example, the classifier will be trained to detect if the image is a:

- Bicycle
- Boat
- Car
- Plane

The four objects above are the class the classifier has to recognize. To construct a classifier, you need to have some data as input and assigns a label to it. The algorithm will take these data, find a pattern and then classify it in the corresponding class.

This task is called **supervised learning**. In supervised learning, the training data you feed to the algorithm includes a label.

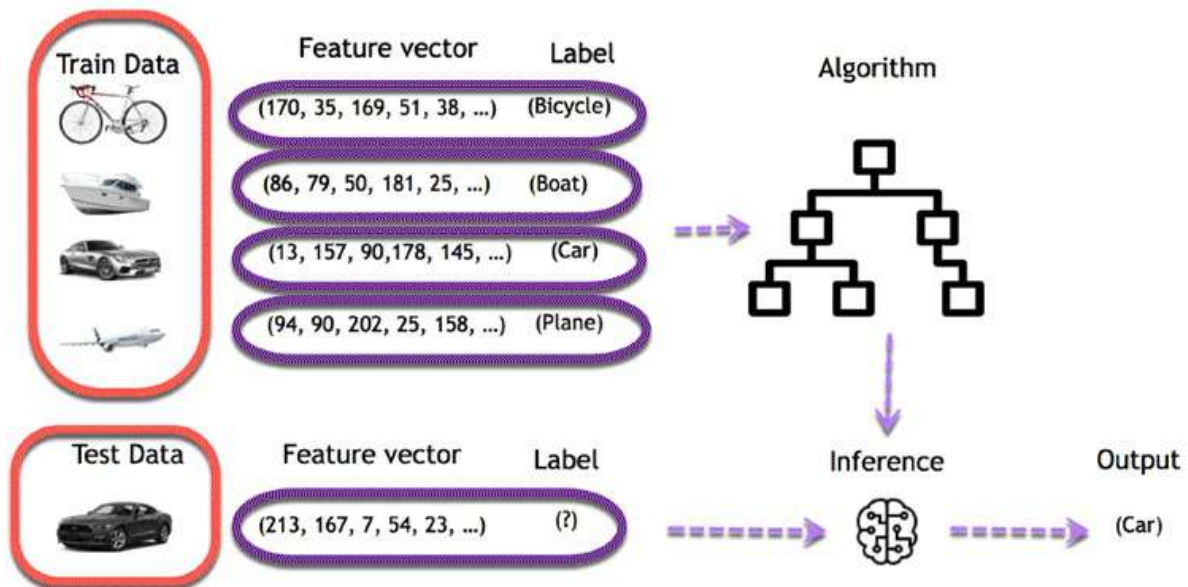
Training an algorithm requires to follow a few standard steps:

- Collect the data
- Train the classifier
- Make predictions

The first step is necessary, choosing the right data will make the algorithm success or a failure. The data you choose to train the model is called a **feature**. In the object example, the features are the pixels of the images.

Each image is a row in the data while each pixel is a column. If your image is a 28x28 size, the dataset contains 784 columns (28x28). In the picture below, each

picture has been transformed into a feature vector. The label tells the computer what object is in the image.



The objective is to use these training data to classify the type of object. The first step consists of creating the feature columns. Then, the second step involves choosing an algorithm to train the model. When the training is done, the model will predict what picture corresponds to what object.

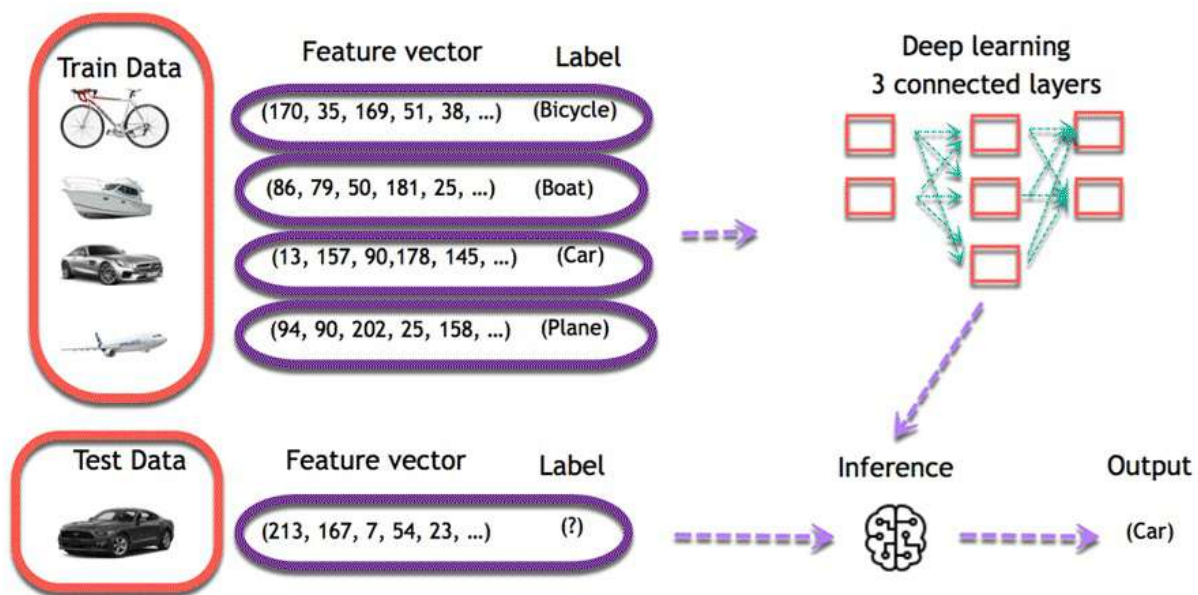
After that, it is easy to use the model to predict new images. For each new image feeds into the model, the machine will predict the class it belongs to. For example, an entirely new image without a label is going through the model. For a human being, it is trivial to visualize the image as a car. The machine uses its previous knowledge to predict as well the image is a car.

Deep Learning Process

In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other.

Consider the same image example above. The training set would be fed to a neural network

Each input goes into a neuron and is multiplied by a weight. The result of the multiplication flows to the next layer and become the input. This process is repeated for each layer of the network. The final layer is named the output layer; it provides an actual value for the regression task and a probability of each class for the classification task. The neural network uses a mathematical algorithm to update the weights of all the neurons. The neural network is fully trained when the value of the weights gives an output close to the reality. For instance, a well-trained neural network can recognize the object on a picture with higher accuracy than the traditional neural net.



Automate Feature Extraction using DL

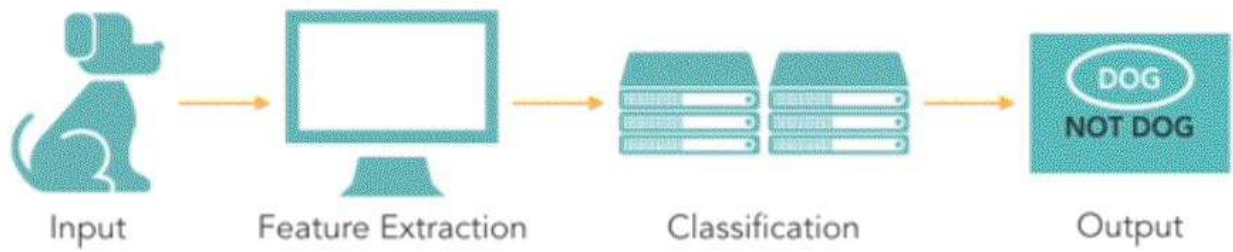
A dataset can contain a dozen to hundreds of features. The system will learn from the relevance of these features. However, not all features are meaningful for the algorithm. A crucial part of machine learning is to find a relevant set of features to make the system learn something.

One way to perform this part in machine learning is to use feature extraction. Feature extraction combines existing features to create a more relevant set of features. It can be done with PCA, T-SNE or any other dimensionality reduction algorithms.

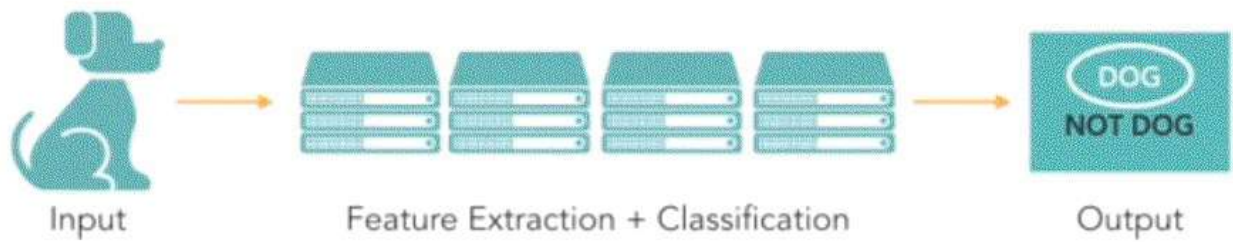
For example, in image processing, the practitioner needs to extract the features manually in the image like the eyes, the nose, lips and so on. Those extracted features are fed to the classification model.

Deep learning solves this issue, especially for a convolutional neural network. The first layer of a neural network will learn small details from the picture; the next layers will combine the previous knowledge to make more complex information. In the convolutional neural network, the feature extraction is done with the use of the filter. The network applies a filter to the picture to see if there is a match, i.e., the shape of the feature is identical to a part of the image. If there is a match, the network will use this filter. The process of feature extraction is therefore done automatically.

TRADITIONAL MACHINE LEARNING



DEEP LEARNING



Difference between Machine Learning and Deep Learning

	Machine Learning	Deep Learning
Data Dependencies	Excellent performances on a small/medium dataset	Excellent performance on a big dataset
Hardware dependencies	Work on a low-end machine.	Requires powerful machine, preferably with GPU: DL performs a significant amount of matrix multiplication
Feature engineering	Need to understand the features that represent the data	No need to understand the best feature that represents the data
Execution time	From few minutes to hours	Up to weeks. Neural Network needs to compute a significant number of weights
Interpretability	Some algorithms are easy to interpret (logistic, decision tree), some are almost impossible (SVM, XGBoost)	Difficult to impossible

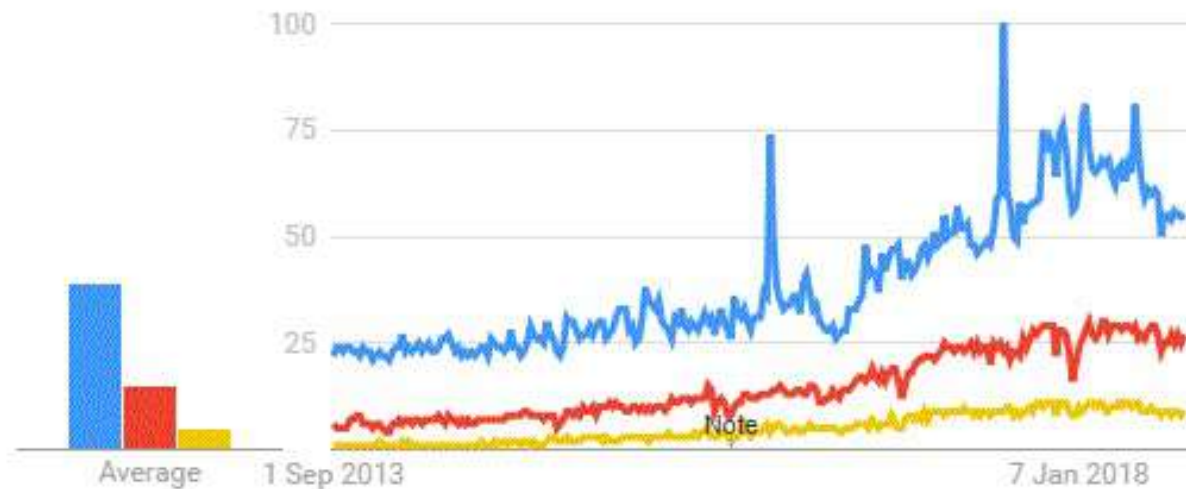
When to use ML or DL?

In the table below, we summarize the difference between machine learning and deep learning.

	Machine learning	Deep learning
Training dataset	Small	Large
Choose features	Yes	No
Number of algorithms	Many	Few
Training time	Short	Long

With machine learning, you need fewer data to train the algorithm than deep learning. Deep learning requires an extensive and diverse set of data to identify the underlying structure. Besides, machine learning provides a faster-trained model. Most advanced deep learning architecture can take days to a week to train. The advantage of deep learning over machine learning is it is highly accurate. You do not need to understand what features are the best representation of the data; the neural network learned how to select critical features. In machine learning, you need to choose for yourself what features to include in the model.

● Artificial intelligence ● machine learning ● deep learning



Summary

Artificial intelligence is imparting a cognitive ability to a machine. Early AI systems used pattern matching and expert systems.

The idea behind machine learning is that the machine can learn without human intervention. The machine needs to find a way to learn how to solve a task given the data.

Deep learning is the breakthrough in the field of artificial intelligence. When there is enough data to train on, deep learning achieves impressive results, especially for image recognition and text translation. The main reason is the feature extraction is done automatically in the different layers of the network.

Chapter 3: What is TensorFlow?

What is TensorFlow?

Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

To give a concrete example, Google users can experience a faster and more refined the search with AI. If the user types a keyword a the search bar, Google provides a recommendation about what could be the next word.



Google wants to use machine learning to take advantage of their massive datasets to give users the best experience. Three different groups use machine learning:

- Researchers
- Data scientists
- Programmers.

They can all use the same toolset to collaborate with each other and improve their efficiency.

Google does not just have any data; they have the world's most massive computer, so TensorFlow was built to scale. TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.

It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

History of TensorFlow

A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:

- Gmail
- Photo
- Google search engine

They build a framework called **Tensorflow** to let researchers and developers work together on an AI model. Once developed and scaled, it allows lots of people to use it.

It was first made public in late 2015, while the first stable version appeared in 2017. It is open source under Apache Open Source license. You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.

TensorFlow Architecture

Tensorflow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

Where can Tensorflow run?

TensorFlow can hardware, and software requirements can be classified into

Development Phase: This is when you train the mode. Training is usually done on your Desktop or laptop.

Run Phase or Inference Phase: Once training is done Tensorflow can be run on many different platforms. You can run it on

- Desktop running Windows, macOS or Linux
- Cloud as a web service
- Mobile devices like iOS and Android

You can train it on multiple machines then you can run it on a different machine, once you have the trained model.

The model can be trained and used on GPUs as well as CPUs. GPUs were initially designed for video games. In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix multiplication. TensorFlow is very fast at computing the matrix multiplication because it is written in C++. Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.

Finally, a significant feature of TensorFlow is the TensorBoard. The TensorBoard enables to monitor graphically and visually what TensorFlow is doing.

Introduction to Components of TensorFlow

Tensor

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.

The graph outlines the ops and connections between the nodes. However, it does not display the values. The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

Graphs

TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:

- It was done to run on multiple CPUs or GPUs and even mobile operating system
- The portability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together
 - A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoints outputs. The edges the edges

explain the input/output relationships between nodes.

Why is TensorFlow popular?

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to built at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation; it allows the developer to visualize the construction of the neural network with Tensorboard. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.

Tensorflow attracts the largest popularity on GitHub compare to the other deep learning framework.

List of Prominent Algorithms supported by TensorFlow

Currently, TensorFlow 1.10 has a built-in API for:

- Linear regression: `tf.estimator.LinearRegressor`
- Classification: `tf.estimator.LinearClassifier`
- Deep learning classification: `tf.estimator.DNNClassifier`
- Deep learning wide and deep: `tf.estimator.DNNLinearCombinedClassifier`
- Booster tree regression: `tf.estimator.BoostedTreesRegressor`
- Boosted tree classification: `tf.estimator.BoostedTreesClassifier`

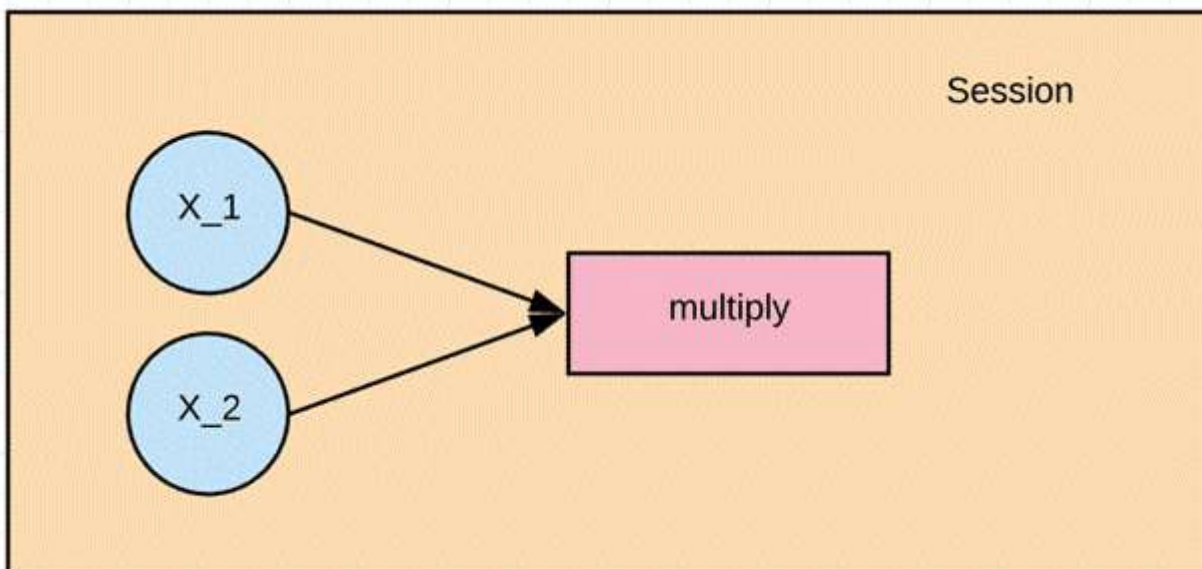
Simple TensorFlow Example

```
import numpy as np
import tensorflow as tf
```

In the first two line of code, we have imported tensorflow as tf. With Python, it is a common practice to use a short name for a library. The advantage is to avoid to type the full name of the library when we need to use it. For instance, we can import tensorflow as tf, and call tf when we want to use a tensorflow function

Let 's practice the elementary workflow of Tensorflow with a simple example. Let 's create a computational graph that multiplies two numbers together.

During the example, we will multiply X_1 and X_2 together. Tensorflow will create a node to connect the operation. In our example, it is called multiply. When the graph is determined, Tensorflow computational engines will multiply together X_1 and X_2.



Finally, we will run a TensorFlow session that will run the computational graph with the values of X_1 and X_2 and print the result of the multiplication.

Let's define the X_1 and X_2 input nodes. When we create a node in Tensorflow, we have to choose what kind of node to create. The X1 and X2 nodes will be a placeholder node. The placeholder assigns a new value each time we make a calculation. We will create them as a TF dot placeholder node.

Step 1: Define the variable

```
X_1 = tf.placeholder(tf.float32, name = "X_1")  
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

When we create a placeholder node, we have to pass in the data type will be adding numbers here so we can use a floating-point data type, let's use tf.float32. We also need to give this node a name. This name will show up when we look at the graphical visualizations of our model. Let's name this node X_1 by passing in a parameter called name with a value of X_1 and now let's define X_2 the same way. X_2.

Step 2: Define the computation

```
multiply = tf.multiply(X_1, X_2, name = "multiply")
```

Now we can define the node that does the multiplication operation. In Tensorflow we can do that by creating a tf.multiply node.

We will pass in the X_1 and X_2 nodes to the multiplication node. It tells tensorflow to link those nodes in the computational graph, so we are asking it to pull the values from x and y and multiply the result. Let's also give the multiplication node the name multiply. It is the entire definition for our simple computational graph.

Step 3: Execute the operation

To execute operations in the graph, we have to create a session. In Tensorflow, it is done by tf.Session(). Now that we have a session we can ask the session to run operations on our computational graph by calling session. To run the

computation, we need to use run.

When the addition operation runs, it is going to see that it needs to grab the values of the X_1 and X_2 nodes, so we also need to feed in values for X_1 and X_2. We can do that by supplying a parameter called feed_dict. We pass the value 1,2,3 for X_1 and 4,5,6 for X_2.

We print the results with print(result). We should see 4, 10 and 18 for 1x4, 2x5 and 3x6

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")

multiply = tf.multiply(X_1, X_2, name = "multiply")

with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:
[4,5,6]})
    print(result)
```

```
[ 4. 10. 18.]
```


Options to Load Data into TensorFlow

The first step before training a machine learning algorithm is to load the data. There are two common ways to load data:

1. Load data into memory: It is the simplest method. You load all your data into memory as a single array. You can write a Python code. These lines of code are unrelated to TensorFlow.
2. TensorFlow data pipeline. TensorFlow has a built-in API that helps you to load the data, perform the operation and feed the machine learning algorithm easily. This method works very well especially when you have a large dataset. For instance, image records are known to be enormous and do not fit into memory. The data pipeline manages the memory by itself.

What solution to use?

Load data in memory

If your dataset is not too big, i.e., less than 10 gigabytes, you can use the first method. The data can fit into the memory. You can use a famous library called Pandas to import CSV files. You will learn more about Pandas in the next tutorial.

Load data with TensorFlow pipeline

The second method works best if you have a large dataset. For instance, if you have a dataset of 50 gigabytes, and your computer has only 16 gigabytes of memory then the machine will crash.

In this situation, you need to build a TensorFlow pipeline. The pipeline will load the data in batch, or small chunk. Each batch will be pushed to the pipeline and be ready for the training. Building a pipeline is an excellent solution because it

allows you to use parallel computing. It means Tensorflow will train the model across multiple CPUs. It fosters the computation and permits for training powerful neural network.

You will see in the next tutorials on how to build a significant pipeline to feed your neural network.

In a nutshell, if you have a small dataset, you can load the data in memory with Pandas library.

If you have a large dataset and you want to make use of multiple CPUs, then you will be more comfortable to work with Tensorflow pipeline.

Create Tensorflow pipeline

In the example before, we manually add three values for X_1 and X_2. Now we will see how to load data to Tensorflow.

Step 1) Create the data

First of all, let's use numpy library to generate two random values.

```
import numpy as np
x_input = np.random.sample((1,2))
print(x_input)
```

```
[[0.8835775 0.23766977]]
```

Step 2: Create the placeholder

Like in the previous example, we create a placeholder with the name X. We need to specify the shape of the tensor explicitly. In case, we will load an array with only two values. We can write the shape as shape=[1,2]

```
# using a placeholder
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

Step 3: Define the dataset method

next, we need to define the Dataset where we can populate the value of the placeholder x. We need to use the method tf.data.Dataset.from_tensor_slices

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

Step 4: Create the pipeline

In step four, we need to initialize the pipeline where the data will flow. We need to create an iterator with make_initializable_iterator. We name it iterator. Then

we need to call this iterator to feed the next batch of data, `get_next`. We name this step `get_next`. Note that in our example, there is only one batch of data with only two values.

```
iterator = dataset.make_initializable_iterator()  
get_next = iterator.get_next()
```

Step 5: Execute the operation

The last step is similar to the previous example. We initiate a session, and we run the operation iterator. We feed the `feed_dict` with the value generated by numpy. These two value will populate the placeholder `x`. Then we run `get_next` to print the result.

```
with tf.Session() as sess:  
    # feed the placeholder with data  
    sess.run(iterator.initializer, feed_dict={ x: x_input })  
    print(sess.run(get_next)) # output [ 0.52374458  0.71968478]
```

```
[0.8835775  0.23766978]
```

Summary

TensorFlow is the most famous deep learning library these recent years. A practitioner using TensorFlow can build any deep learning structure, like CNN, RNN or simple artificial neural network.

TensorFlow is mostly used by academics, startups, and large companies. Google uses TensorFlow in almost all Google daily products including Gmail, Photo and Google Search Engine.

Google Brain team's developed TensorFlow to fill the gap between researchers and products developers. In 2015, they made TensorFlow public; it is rapidly growing in popularity. Nowadays, TensorFlow is the deep learning library with the most repositories on GitHub.

Practitioners use Tensorflow because it is easy to deploy at scale. It is built to work in the cloud or on mobile devices like iOS and Android.

Tensorflow works in a session. Each session is defined by a graph with different computations. A simple example can be to multiply to number. In Tensorflow, three steps are required:

1. Define the variable

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

2. Define the computation

```
multiply = tf.multiply(X_1, X_2, name = "multiply")
```

3. Execute the operation

```
with tf.Session() as session:
    result = session.run(multiply, feed_dict={X_1:[1,2,3], X_2:
    [4,5,6]})
    print(result)
```

One common practice in Tensorflow is to create a pipeline to load the data. If you follow these five steps, you'll be able to load data to TensorFlow

1. Create the data

```
import numpy as np
x_input = np.random.sample((1,2))
print(x_input)
```

2. Create the placeholder

```
x = tf.placeholder(tf.float32, shape=[1,2], name = 'X')
```

3. Define the dataset method

```
dataset = tf.data.Dataset.from_tensor_slices(x)
```

4. Create the pipeline

```
iterator = dataset.make_initializable_iterator() get_next =  
iteraror.get_next()
```

5. Execute the program

```
with tf.Session() as sess:  
sess.run(iterator.initializer, feed_dict={ x: x_input })  
print(sess.run(get_next))
```

Chapter 4: Comparison of Deep Learning Libraries

Artificial intelligence is growing in popularity since 2016 with, 20% of the big companies using AI in their businesses (McKinsey report, 2018). As per the same report AI can create substantial value across industries. In banking, for instance, the potential of AI is estimated at \$ 300 billion, in retail the number skyrocket to \$ 600 billion.

To unlock the potential value of AI, companies must choose the right deep learning framework. In this tutorial, you will learn about the different libraries available to carry out deep learning tasks. Some libraries have been around for years while new library like TensorFlow has come to light in recent years.

8 Best Deep learning Libraries /Framework

In this list, we will compare the top Deep learning frameworks. All of them are open source and popular in the data scientist community. We will also compare popular ML as a service providers

Torch

Torch is an old open source machine learning library. It is first released was 15 years ago. Its primary programming language is LUA, but has an implementation in C. Torch supports a vast library for machine learning algorithms, including deep learning. It supports CUDA implementation for parallel computation.

Torch is used by most of the leading labs such as Facebook, Google, Twitter, Nvidia, and so on. Torch has a library in Python names Pytorch.

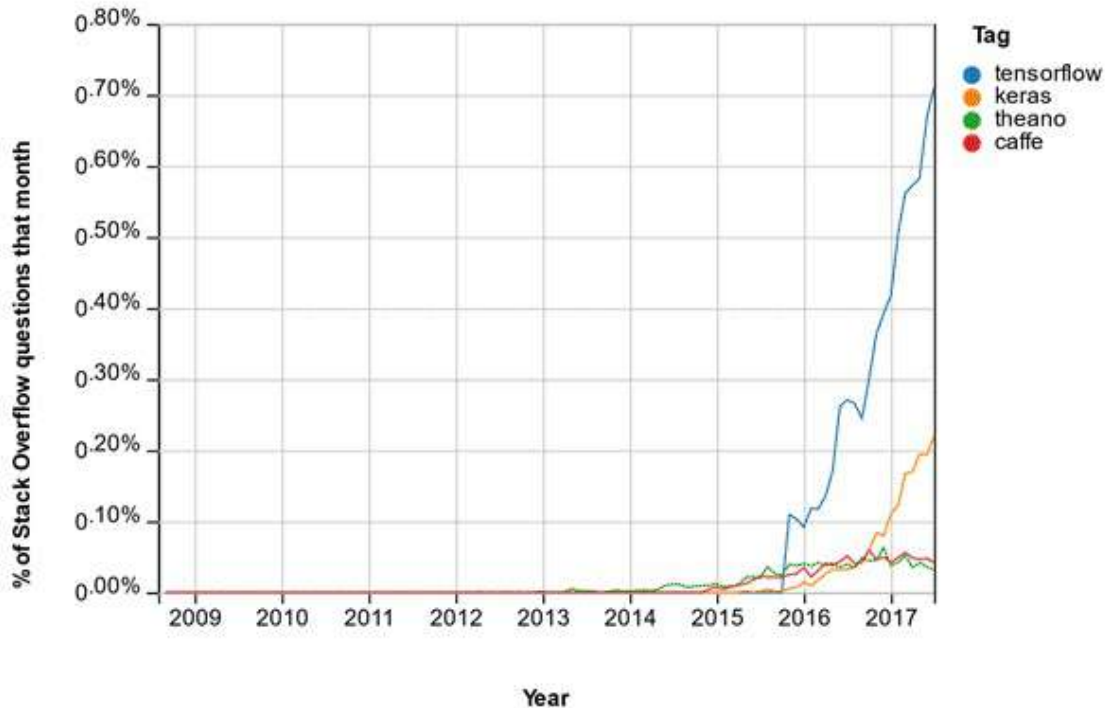
Infer.net

Infer.net is developed and maintained by Microsoft. Infer.net is a library with a primary focus on the Bayesian statistic. Infer.net is designed to offer practitioners state-of-the-art algorithms for probabilistic modeling. The library contains analytical tools such as Bayesian analysis, hidden Markov chain, clustering.

Keras

Keras is a Python framework for deep learning. It is a convenient library to construct any deep learning algorithm. The advantage of Keras is that it uses the same Python code to run on CPU or GPU. Besides, the coding environment is pure and allows for training state-of-the-art algorithm for computer vision, text recognition among other.

Keras has been developed by François Chollet, a researcher at Google. Keras is used in prominent organizations like CERN, Yelp, Square or Google, Netflix, and Uber.



Theano

Theano is deep learning library developed by the Université de Montréal in 2007. It offers fast computation and can be run on both CPU and GPU. Theano has been developed to train deep neural network algorithms.

MICROSOFT COGNITIVE TOOLKIT(CNTK)

Microsoft toolkit, previously known as CNTK, is a deep learning library developed by Microsoft. According to Microsoft, the library is among the fastest on the market. Microsoft toolkit is an open-source library, although Microsoft is using it extensively for its products like Skype, Cortana, Bing, and Xbox. The toolkit is available both in Python and C++.

MXNet

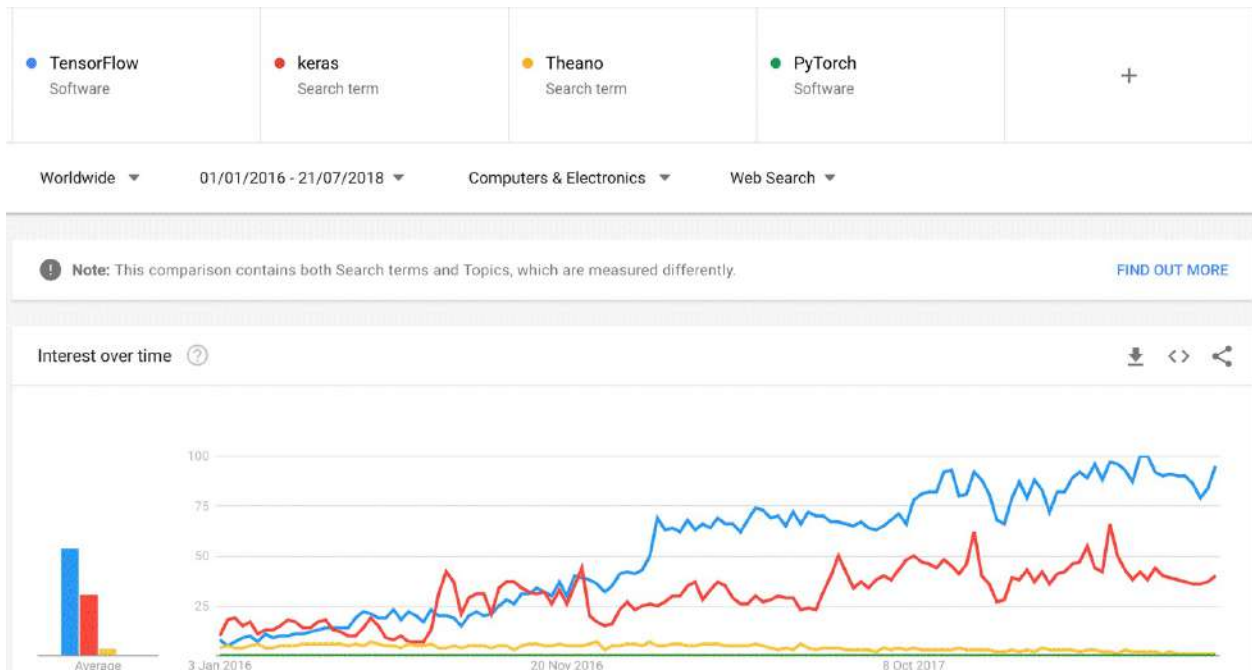
MXNet is a recent deep learning library. It is accessible with multiple programming languages including C++, Julia, Python and R. MXNet can be configured to work on both CPU and GPU. MXNet includes state-of-the-art deep learning architecture such as Convolutional Neural Network and Long Short-Term Memory. MXNet is built to work in harmony with dynamic cloud infrastructure. The main user of MXNet is Amazon.

Caffe

Caffe is a library built by Yangqing Jia when he was a PhD student at Berkeley. Caffe is written in C++ and can perform computation on both CPU and GPU. The primary uses of Caffe is Convolutional Neural Network. Although, in 2017, Facebook extended Caffe with more deep learning architecture, including Recurrent Neural Network. Caffe is used by academics and startups but also some large companies like Yahoo!.

TensorFlow

TensorFlow is Google's open source project. TensorFlow is the most famous deep learning library these days. It was released to the public in late 2015.



TensorFlow is developed in C++ and has convenient Python API, although C++ APIs are also available. Prominent companies like Airbus, Google, IBM and so on are using TensorFlow to produce deep learning algorithms.

TensorFlow Vs Theano Vs Torch Vs Keras Vs infer.net Vs CNTK Vs MXNet Vs Caffe: Key Differences

Library	Platform	Written in	Cuda support	Parallel Execution	Has trained models	RNN	CNN
Torch	Linux, MacOS, Windows	Lua	Yes	Yes	Yes	Yes	Yes
Infer.Net	Linux, MacOS, Windows	Visual Studio	No	No	No	No	No
Keras	Linux, MacOS, Windows	Python	Yes	Yes	Yes	Yes	Yes
Theano	Cross-platform	Python	Yes	Yes	Yes	Yes	Yes
TensorFlow	Linux, MacOS, Windows, Android	C++, Python, CUDA	Yes	Yes	Yes	Yes	Yes
MICROSOFT COGNITIVE TOOLKIT	Linux, Windows, Mac with Docker	C++	Yes	Yes	Yes	Yes	Yes
Caffe	Linux, MacOS, Windows	C++	Yes	Yes	Yes	Yes	Yes
MXNet	Linux, Windows, MacOS, Android, iOS, Javascript	C++	Yes	Yes	Yes	Yes	Yes

Verdict:

TensorFlow is the best library of all because it is built to be accessible for everyone. Tensorflow library incorporates different API to built at scale deep learning architecture like CNN or RNN. TensorFlow is based on graph computation, it allows the developer to visualize the construction of the neural network with Tensorboard. This tool is helpful to debug the program. Finally, Tensorflow is built to be deployed at scale. It runs on CPU and GPU.

Tensorflow attracts the largest popularity on GitHub compare to the other deep

learning framework.

Comparing Machine Learning as a Service

Following are 4 popular DL as a service providers

Google Cloud ML

Google provides for developer pre-trained model available in Cloud AutoML. This solution exists for a developer without a strong background in machine learning. Developers can use state-of-the-art Google's pre-trained model on their data. It allows any developers to train and evaluate any model in just a few minutes.

Google currently provides a REST API for computer vision, speech recognition, translation, and NLP.



Using Google Cloud, you can train a machine learning framework build on

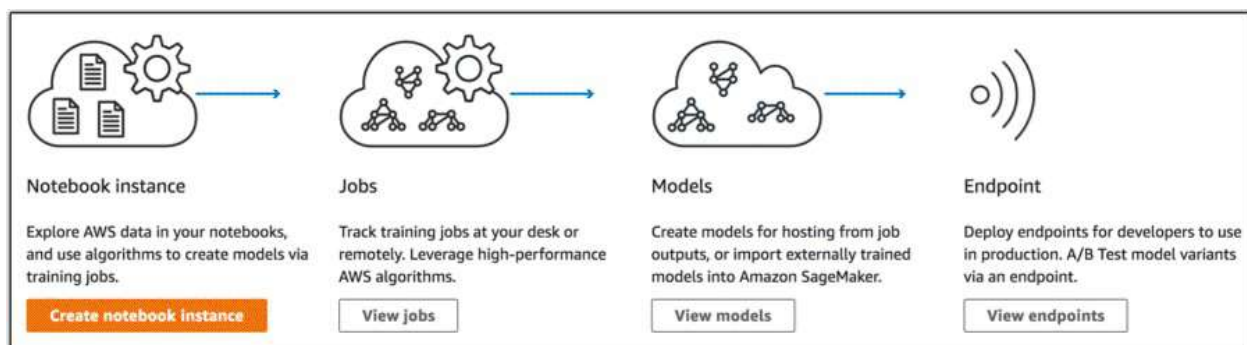
TensorFlow, Scikit-learn, XGBoost or Keras. Google Cloud machine learning will train the models across its cloud.

The advantage to use Google cloud computing is the simplicity to deploy machine learning into production. There is no need to set up Docker container. Besides, the cloud takes care of the infrastructure. It knows how to allocate resources with CPUs, GPUs, and TPUs. It makes the training faster with paralleled computation.

AWS SageMaker

A major competitor to Google Cloud is Amazon cloud, AWS. Amazon has developed Amazon SageMaker to allow data scientists and developers to build, train and bring into production any machine learning models.

SageMaker is available in a Jupyter Notebook and includes the most used machine learning library, TensorFlow, MXNet, Scikit-learn amongst others. Programs written with SageMaker are automatically run in the Docker containers. Amazon handles the resource allocation to optimize the training and deployment.



Amazon provides API to the developers in order to add intelligence to their applications. In some occasion, there is no need to reinventing-the-wheel by building from scratch new models while there are powerful pre-trained models in the cloud. Amazon provides API services for computer vision, conversational

chatbots and language services:

The three major available API are:

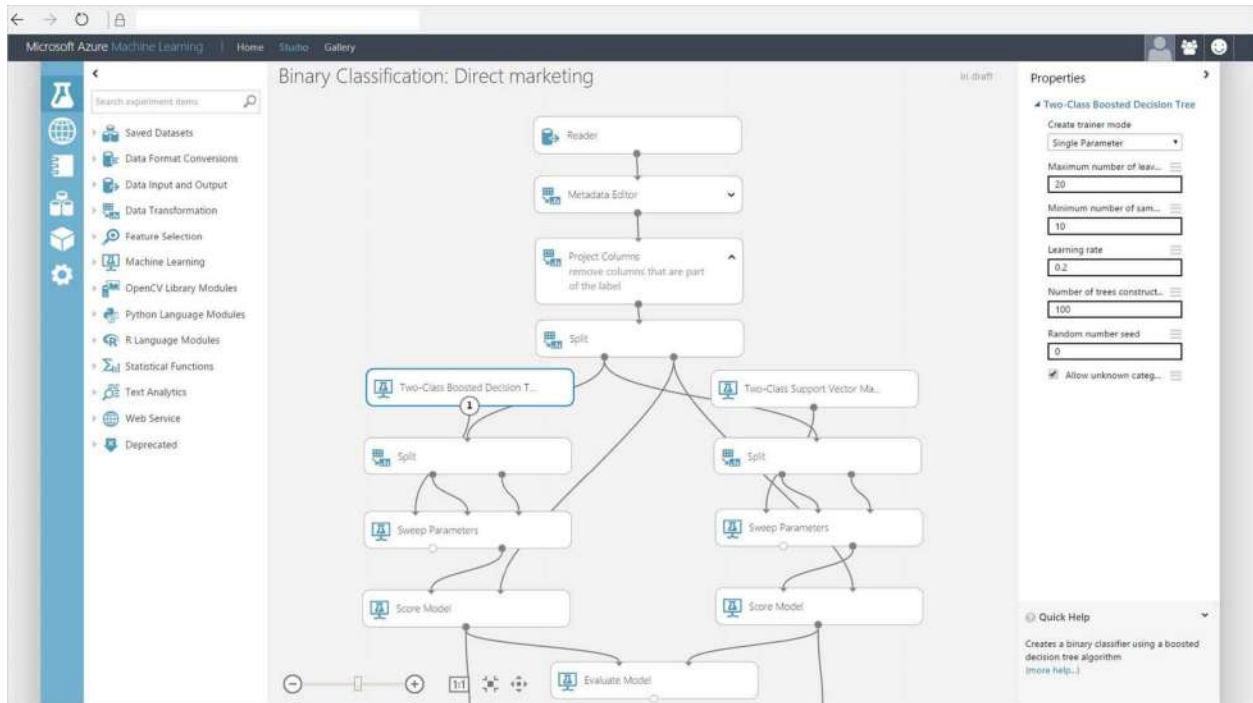
- Amazon Rekognition: provides image and video recognition to an app
- Amazon Comprehend: Perform text mining and neural language processing to, for instance, automatize the process of checking the legality of financial document
- Amazon Lex: Add chatbot to an app

Azure Machine Learning Studio

Probably one of the friendliest approaches to machine learning is Azure Machine Learning Studio. The significant advantage of this solution is that no prior programming knowledge is required.

Microsoft Azure Machine Learning Studio is a drag-and-drop collaborative tool to create, train, evaluate and deploy machine learning solution. The model can be efficiently deployed as web services and used in several apps like Excel.

Azure Machine learning interface is interactive, allowing the user to build a model just by dragging and dropping elements quickly.

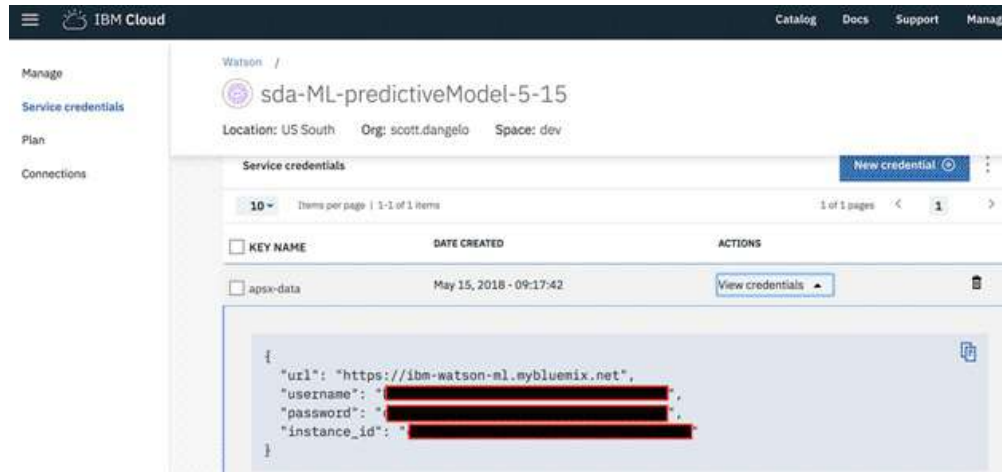


When the model is ready, the developer can save it and push it to Azure Gallery or Azure Marketplace.

Azure Machine learning can be integrated into R or Python their custom built-in package.

IBM Watson ML

Watson studio can simplify the data projects with a streamlined process that allows extracting value and insights from the data to help the business to get smarter and faster. Watson studio delivers an easy-to-use collaborative data science and machine learning environment for building and training models, preparing and analyzing data, and sharing insights all in one place. Watson Studio is easy to use with a drag-and-drop code.



Watson studio supports some of the most popular frameworks like Tensorflow, Keras, Pytorch, Caffe and can deploy a deep learning algorithm on to the latest GPUs from Nvidia to help accelerate modeling.

Verdict:

In our point of view, Google cloud solution is the one that is the most recommended. Google cloud solution provides lower prices the AWS by at least 30% for data storage and machine learning solution. Google is doing an excellent job to democratize AI. It has developed an open source language, TensorFlow, optimized data warehouse connection, provides tremendous tools from data visualization, data analysis to machine learning. Besides, Google Console is ergonomic and much more comprehensive than AWS or Windows.

Chapter 5: How to Download and Install TensorFlow Windows and Mac

In this tutorial, we will explain how to install **TensorFlow** with **Anaconda**. You will learn how to use TensorFlow with Jupyter. Jupyter is a notebook viewer.

TensorFlow Versions

TensorFlow supports computations across multiple CPUs and GPUs. It means that the computations can be distributed across devices to improve the speed of the training. With parallelization, you don't need to wait for weeks to obtain the results of training algorithms.

For Windows user, TensorFlow provides two versions:

- **TensorFlow with CPU support only:** If your Machine does not run on NVIDIA GPU, you can only install this version
- **TensorFlow with GPU support:** For faster computation, you can use this version of TensorFlow. This version makes sense only if you need strong computational capacity.

During this tutorial, the basic version of TensorFlow is sufficient.

Note: TensorFlow does not provides GPU support on MacOS.

Here is how to proceed

MacOS User:

- Install Anaconda
- Create a .yaml file to install Tensorflow and dependencies
- Launch Jupyter Notebook

For Windows

- Install Anaconda
- Create a .yaml file to install dependencies
- Use pip to add TensorFlow
- Launch Jupyter Notebook

To run Tensorflow with Jupyter, you need to create an environment within Anaconda. It means you will install Ipython, Jupyter, and TensorFlow in an appropriate folder inside our machine. On top of this, you will add one essential library for data science: "Pandas". The Pandas library helps to manipulate a data frame.

Install Anaconda

Download Anaconda version 4.3.1 (for Python 3.6) for the appropriate system.

Anaconda will help you to manage all the libraries required either for Python or R. Refer this tutorial to install Anaconda

Create .yml file to install Tensorflow and dependencies

It includes

- Locate the path of Anaconda
- Set the working directory to Anaconda
- Create the yml file (For MacOS user, TensorFlow is installed here)
- Edit the yml file
- Compile the yml file
- Activate Anaconda
- Install TensorFlow (Windows user only)

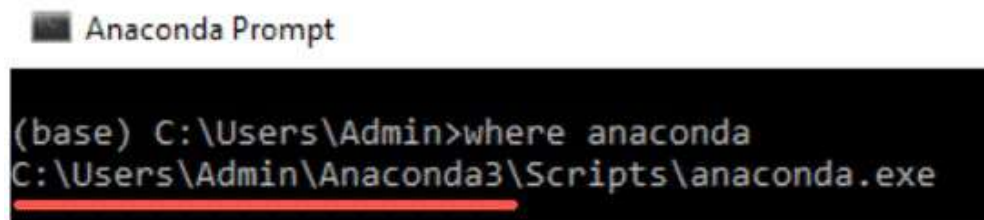
Step 1) Locate Anaconda

The first step you need to do is to locate the path of Anaconda. You will create a new conda environment that includes the necessary libraries you will use during the tutorials about TensorFlow.

Windows

If you are a Windows user, you can use Anaconda Prompt and type:

```
C:\>where anaconda
```



```
Anaconda Prompt  
(base) C:\Users\Admin>where anaconda  
C:\Users\Admin\Anaconda3\Scripts\anaconda.exe
```

We are interested to know the name of the folder where Anaconda is installed because we want to create our new environment inside this path. For instance, in

the picture above, Anaconda is installed in the Admin folder. For you, it can be the same, i.e. Admin or the user's name.

In the next, we will set the working directory from `c:\` to `Anaconda3`.

MacOS

for MacOS user, you can use the Terminal and type:

```
which anaconda
```

```
Thomass-MacBook-Pro:~ Thomas$ which anaconda  
/Users/Thomas/anaconda3/bin/anaconda
```



Path of
Anaconda

You will need to create a new folder inside Anaconda which will contain **Ipypthon, Jupyter** and **TensorFlow**. A quick way to install libraries and software is to write a `yml` file.

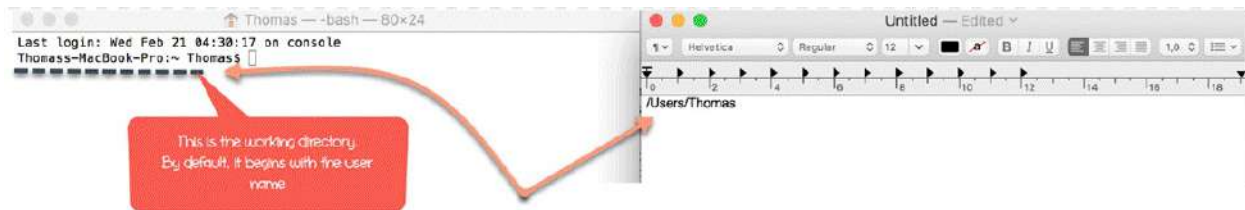
Step 2) Set working directory

You need to specify the working directory where you want to create the `yml` file. As said before, it will be located inside Anaconda.

For MacOS user:

The Terminal sets the default working directory to **Users/USERNAME**. As you can see in the figure below, the path of `anaconda3` and the working directory are identical. In MacOS, the latest folder is shown before the `$`. The Terminal will install all the libraries in this working directory.

If the path on the text editor does not match the working directory, you can change it by writing `cd PATH` in the Terminal. `PATH` is the path you pasted in the text editor. Don't forget to wrap the `PATH` with `'PATH'`. This action will change the working directory to `PATH`.



Open your Terminal, and type:

```
cd anaconda3
```

For Windows user (make sure of the folder before Anaconda3):

```
cd C:\Users\Admin\Anaconda3
```

or the path "where anaconda" command gives you

```
base) C:\Users\Admin>cd C:\Users\Admin\Anaconda3
base) C:\Users\Admin\Anaconda3>
```

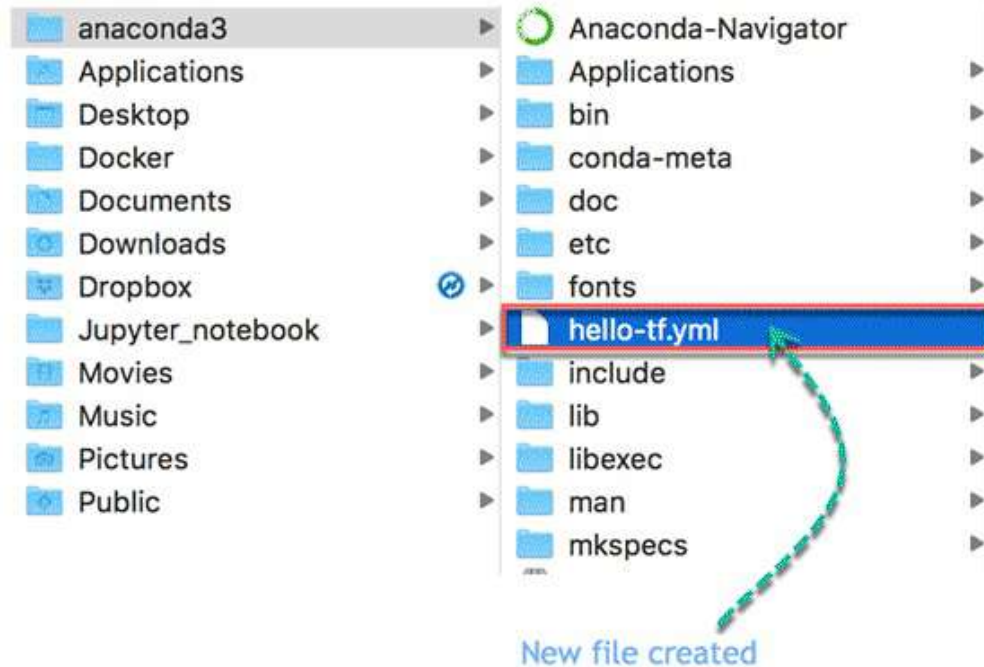
Step 3) Create the yml file

You can create the yml file inside the new working directory. The file will install the dependencies you need to run TensorFlow. Copy and paste this code into the Terminal.

For MacOS user:

```
touch hello-tf.yml
```

A new file named `hello-tf.yml` should appear inside `anaconda3`



For Windows user:

```
echo.>hello-tf.yml
```

A new file named hello-tf.yml should appear

This PC > Local Disk (C:) > Users > Admin > Anaconda3 >

Name	Date modified	Type	Size
hello-tf.yml	05-04-2018 02:38 ...	YML File	1 KB
.nonadmin	21-03-2018 12:26 ...	NONADMIN File	0 KB
Uninstall-Anaconda3.exe	20-02-2018 02:26 ...	Application	296 KB

Step 4) Edit the yml file

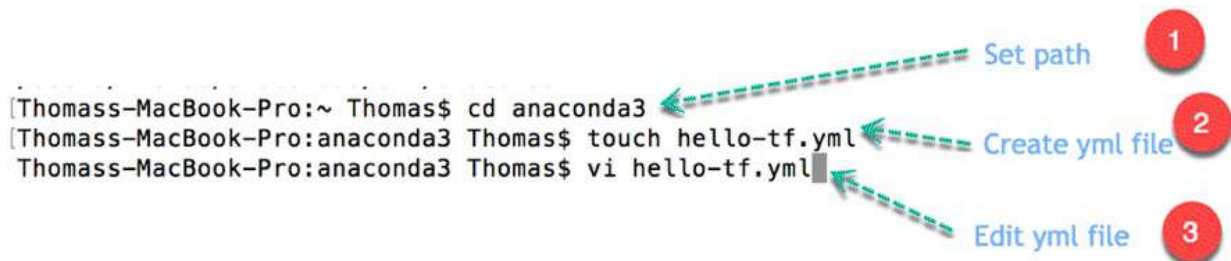
For MacOS user:

You are ready to edit the yml file. You can paste the following code in the Terminal to edit the file. MacOS user can use **vim** to edit the yml file.

```
vi hello-tf.yml
```

So far, your Terminal looks like this

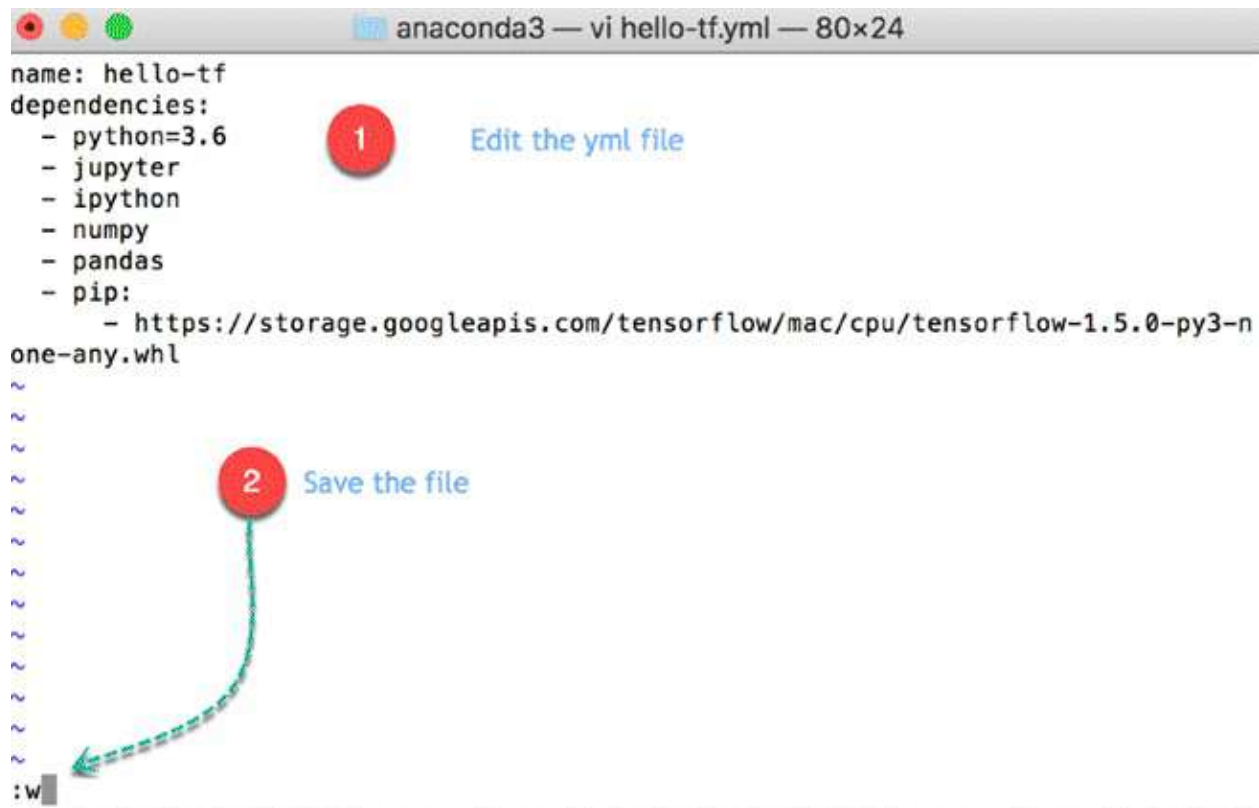
```
[Thomass-MacBook-Pro:~ Thomas$ cd anaconda3
[Thomass-MacBook-Pro:anaconda3 Thomas$ touch hello-tf.yml
[Thomass-MacBook-Pro:anaconda3 Thomas$ vi hello-tf.yml
```



You enter an **edit** mode. Inside this mode, you can, after pressing esc:

- Press i to edit
- Press w to save
- Press q! to quit

Write the following code in the edit mode and press esc followed by :w



```
name: hello-tf
dependencies:
  - python=3.6
  - jupyter
  - ipython
  - numpy
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/mac/cpu/tensorflow-1.5.0-py3-n
one-any.whl
~
~
~
~
~
~
~
~
~
~
~
:w
```

Note: The file is case **and** intend sensitive. 2 spaces are required after each intend.

For MacOS

```
name: hello-tfdependencies:
  - python=3.6
  - jupyter
  - ipython
  - pandas
  - pip:
    - https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl
```

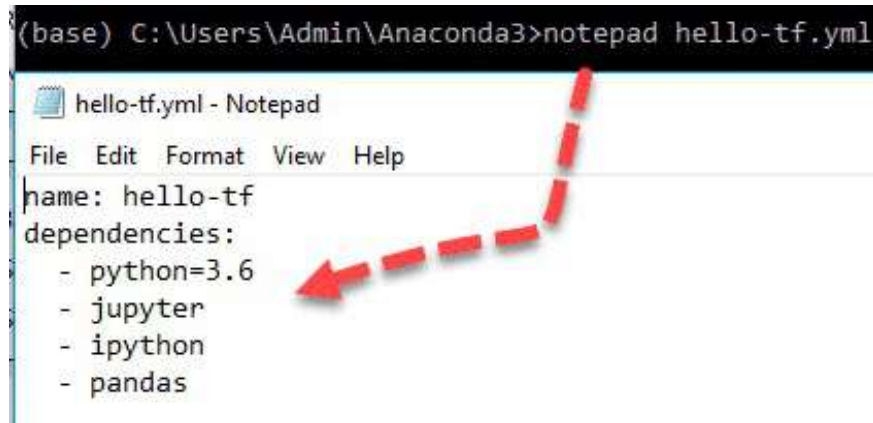
Code Explanation

- name: hello-tf: Name of the yml file
- dependencies:
- python=3.6
- jupyter
- ipython
- pandas: Install Python version 3.6, Jupyter, Ipython, and pandas libraries
- pip: Install a Python library
 - <https://storage.googleapis.com/tensorflow/MacOS/cpu/tensorflow-1.5.0-py3-none-any.whl>: Install TensorFlow from Google apis.

Press esc followed by :q! to quite the edit mode.

- python=3.6
- jupyter
- ipython
- pandas: Install Python version 3.6, Jupyter, Ipython, and pandas libraries

It will open the notepad, you can edit the file from here.



```
(base) C:\Users\Admin\Anaconda3>notepad hello-tf.yml
```

```
hello-tf.yml - Notepad
File Edit Format View Help
name: hello-tf
dependencies:
- python=3.6
- jupyter
- ipython
- pandas
```

Note: Windows users will install TensorFlow in the next step. In this step, you only prepare the conda environment

Step 5) Compile the yml file

You can compile the .yml file with the following code :

```
conda env create -f hello-tf.yml
```

Note: For Windows users, the new environment is created inside the current user directory.

It takes times. It will take around 1.1gb of space in your hard disk.

Installing libraries



```
Downloading and Extracting Packages
python-dateutil 2.7.2: ##### | 100%
openssl 1.0.2o: ##### | 100%
imageio 1.0.0: ##### | 100%
pytz 2018.3: ##### | 100%
cffi 1.11.5: ##### | 100%
tornado 5.0.1: ##### | 100%
setuptools 39.0.1: ##### | 100%
jupyter_client 5.2.3: ##### | 100%
python 3.6.5: ##### | 100%
zeromq 4.2.3: ##### | 100%
pexpect 4.4.0: ##### | 100%
bleach 2.1.3: ##### | 100%
babel 2.5.3: ##### | 100%
asn1crypto 0.24.0: ##### | 100%
libsodium 1.0.16: ##### | 100%
typing 3.6.4: ##### | 100%
sphinx 1.7.2: ##### | 100%
jedi 0.11.1: ##### | 100%
pip 9.0.3: ##### | 100%
packaging 17.1: ##### | 100%
ipywidgets 7.1.2: ##### | 100%
pysocks 1.6.8: ##### | 100%
ipykernel 4.8.2: ##### | 100%
send2trash 1.5.0: ##### | 100%
pyopenssl 17.5.0: ##### | 100%
widgetsnbextension 3.1.4: ##### | 100%
cryptography 2.2.1: ##### | 100%
pyzmq 17.0.0: ##### | 100%
notebook 5.4.1: ##### | 100%
```

In Windows

```
(base) C:\Users\Admin\Anaconda3>conda env create -f hello-tf.yml
Solving environment: done .....

==> WARNING: A newer version of conda exists. <==
  current version: 4.4.10
  latest version: 4.5.0

Please update conda by running

  $ conda update -n base conda

Downloading and Extracting Packages
bleach 2.1.3: #####
numpy 1.14.2: #####
sip 4.19.8: #####
jedi 0.11.1: #####
notebook 5.4.1: #####
mkl 2018.0.2: #####1
```

Step 6) Activate conda environment

We are almost done. You have now 2 conda environments. You created an isolated conda environment with the libraries you will use during the tutorials. This is a recommended practice because each machine learning project requires different libraries. When the project is over, you can remove or not this environment.

```
conda env list
```

```
-----
Thomas-Macbook-Pro:~$ conda env list
# conda environment:
#
base          * /Users/Thomas/anaconda3
hello-tf      /Users/Thomas/anaconda3/envs/hello-tf
```

The asterisk indicates the default one. You need to switch to hello-tf to activate the environment

For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
activate hello-tf
```

```
#
# To activate this environment, use:
# > source activate hello-tf
#
# To deactivate an active environment, use:
# > source deactivate
#
```

You can check all dependencies are in the same environment. This is important because it allows Python to use Jupyter and TensorFlow from the same environment. If you don't see the three of them located in the same folder, you need to start all over again.

For MacOS user:

```
which python
which jupyter
which ipython
```

```
Thomass-MacBook-Pro:anaconda3 Thomas$ which python
/Users/Thomas/anaconda3/bin/python
Thomass-MacBook-Pro:anaconda3 Thomas$ which jupyter
/Users/Thomas/anaconda3/bin/jupyter
Thomass-MacBook-Pro:anaconda3 Thomas$ which ipython
/Users/Thomas/anaconda3/bin/ipython
```

Optional: You can check for update.

```
pip install --upgrade tensorflow
```

Step 7) For Windows user only: Install TensorFlow

For windows user:

```
where python
where jupyter
where ipython
```

```
(hello-tf) C:\Users\Admin\Anaconda3>where python
C:\Users\Admin\Anaconda3\python.exe
C:\Users\Admin\Anaconda3\envs\hello-tf\python.exe

(hello-tf) C:\Users\Admin\Anaconda3>where jupyter
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\jupyter.exe
C:\Users\Admin\Anaconda3\Scripts\jupyter.exe

(hello-tf) C:\Users\Admin\Anaconda3>where ipython
C:\Users\Admin\Anaconda3\envs\hello-tf\Scripts\ipython.exe
C:\Users\Admin\Anaconda3\Scripts\ipython.exe

(hello-tf) C:\Users\Admin\Anaconda3>
```

As you can see, you now have two Python environments. The main one and the newly created one i.e. hello-tf. The main conda environment does not have tensorflow installed only hello-tf. From the picture, python, jupyter and ipython

Launch Jupyter Notebook

This part is the same for both OS.

You can open TensorFlow with Jupyter.

Note: Each time you want to open TensorFlow, you need to initialize the environment

You will proceed as follow:

- Activate hello-tf conda environment
- Open Jupyter
- Import tensorflow
- Delete Notebook
- Close Jupyter

Step 1) Activate conda

For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
conda activate hello-tf
```

```
Thomass-MacBook-Pro:anaconda3 Thomas$ source activate hello-tf  
(hello-tf) Thomass-MacBook-Pro:anaconda3 Thomas$
```

Step 2) Open Jupyter

After that, you can open Jupyter from the Terminal

```
jupyter notebook
```

```
Thomass-MacBook-Pro:anaconda3 Thomas$ source activate hello-tf
(hello-tf) Thomass-MacBook-Pro:anaconda3 Thomas$ jupyter notebook
[W 16:20:59.106 NotebookApp] Error loading server extension jupyter_tensorboard
Traceback (most recent call last):
  File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-packages/notebook/notebookapp.py", line 1451, in init_server_extensions
    mod = importlib.import_module(modulename)
  File "/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/importlib/_init_.py", line 126, in import_t_module
    return _bootstrap._gcd_import(name[level:], package, level)
  File "<frozen importlib._bootstrap>", line 994, in _gcd_import
  File "<frozen importlib._bootstrap>", line 971, in _find_and_load
  File "<frozen importlib._bootstrap>", line 953, in _find_and_load_unlocked
ModuleNotFoundError: No module named 'jupyter_tensorboard'
[I 16:20:59.113 NotebookApp] Serving notebooks from local directory: /Users/Thomas/anaconda3
[I 16:20:59.113 NotebookApp] 0 active kernels
[I 16:20:59.113 NotebookApp] The Jupyter Notebook is running at:
[I 16:20:59.113 NotebookApp] http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.113 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:20:59.114 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time, to login with a token:
http://localhost:8888/?token=9ee3a11fea254e6982421366001678d5a6c01cebc8d5658e
[I 16:20:59.378 NotebookApp] Accepting one-time-token-authenticated connection from ::1
[W 16:21:00.612 NotebookApp] 404 GET /api/tensorboard?_=1522570860085 (::1) 24.15ms referer=http://localhost:8888/tree
```

If the browser does not open automatically, copy this URL.

Your browser should open automatically, otherwise copy and paste the url provided by the Terminal. It starts by `http://localhost:8888`

Inside the Jupyter Notebook, you can see all the files inside the working directory. To create a new Notebook, you simply click on **new** and **Python 3**

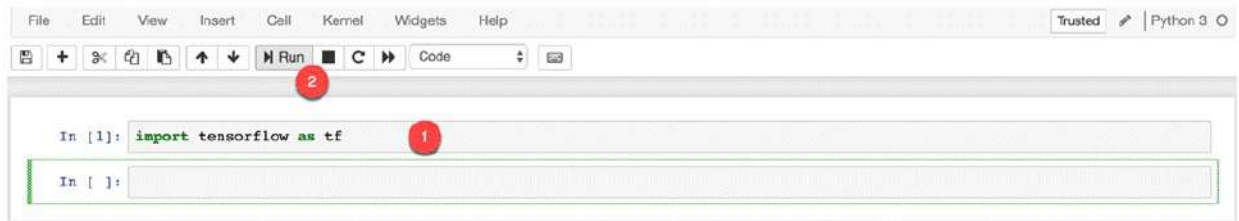
Note: The new notebook is automatically saved inside the working directory.



Step 3) Import Tensorflow

Inside the notebook, you can import TensorFlow with the `tf` alias. Click to run. A new cell is created below.

```
import tensorflow as tf
```



Let's write your first code with TensorFlow.

```
hello = tf.constant('Hello, Guru99!')  
hello
```

A new tensor is created. Congratulations. You successfully installed TensorFlow with Jupyter on your machine.



Step 4) Delete file

You can delete the file named Untitled.ipynb inside Jupyter.



Step 5) Close Jupyter

There are two ways of closing Jupyter. The first way is directly from the notebook. The second way is by using the terminal (or Anaconda Prompt)

From Jupyter

In the main panel of Jupyter Notebook, simply click on **Logout**



You are redirected to the log out page.



From the terminal

Select the terminal or Anaconda prompt and run twice `ctr+c`.

The first time you do `ctr+c`, you are asked to confirm you want to shut down the notebook. Repeat `ctr+c` to confirm

```
^C([I 12:38:40.238 NotebookApp] interrupted
Serving notebooks from local directory: /Users/Thomas
1 active kernel
The Jupyter Notebook is running at:
http://localhost:8888/?token=79edffd9c156eac054cf668c6576cb074716c2182a391076
Shutdown this notebook server (y/[n])? No answer for 5s: resuming operation...
```

Use twice
Ctrl+c
to close Jupyter

```
[I 12:38:57.737 NotebookApp] Shutting down 1 kernel
[I 12:38:58.043 NotebookApp] Kernel shutdown: 2a34c1da-9b2e-40ea-aa20-2971ac33de
1c
(hello-tf) Thomass-MacBook-Pro:~ Thomas$ █
```

You have successfully logged out.

Jupyter with the main conda environment

If you want to launch TensorFlow with jupyter for future use, you need to open a new session with

```
source activate hello-tf
```

If you don't, Jupyter will not find TensorFlow

```
In [1]: import tensorflow as tf
```

```
-----  
ModuleNotFoundError                                Traceback (most recent call last)  
<ipython-input-1-41389fad42b5> in <module>()  
----> 1 import tensorflow as tf  
  
ModuleNotFoundError: No module named 'tensorflow'
```


Chapter 6: Jupyter Notebook Tutorial

What is Jupyter Notebook?

A Jupyter notebook is a web application that allows the user to write codes and rich text elements. Inside the Notebooks, you can write paragraph, equations, title, add links, figures and so on. A notebook is useful to share interactive algorithms with your audience by focusing on teaching or demonstrating a technique. Jupyter Notebook is also a convenient way to run data analysis.

Jupyter Notebook App

The Jupyter Notebook App is the interface where you can write your scripts and codes through your web browser. The app can be used locally, meaning you don't need internet access, or a remote server.



Each computation is done via a kernel. A new kernel is created each time you launch a Jupyter Notebook.

How to use Jupyter

In the session below, you will learn how to use Jupyter Notebook. You will write a simple line of code to get familiar with the environment of Jupyter.

Step 1) You add a folder inside the working directory that will contains all the notebooks you will create during the tutorials about TensorFlow.

Open the Terminal and write

```
mkdir jupyter_tf
jupyter notebook
```

Code Explanation

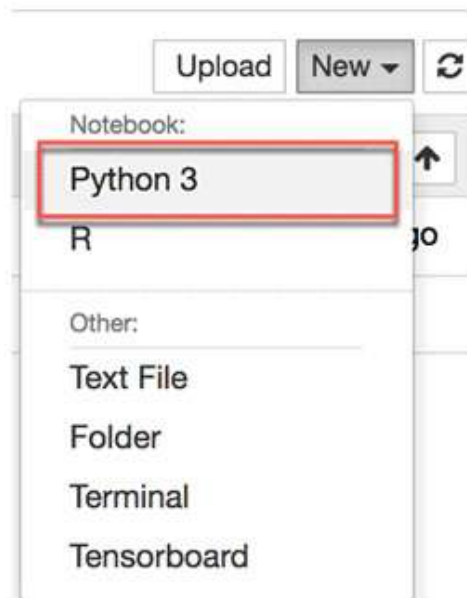
- `mkdir jupyter_tf`: Create a folder names `jupyter_tf`
- `jupyter notebook`: Open Jupyter web-app



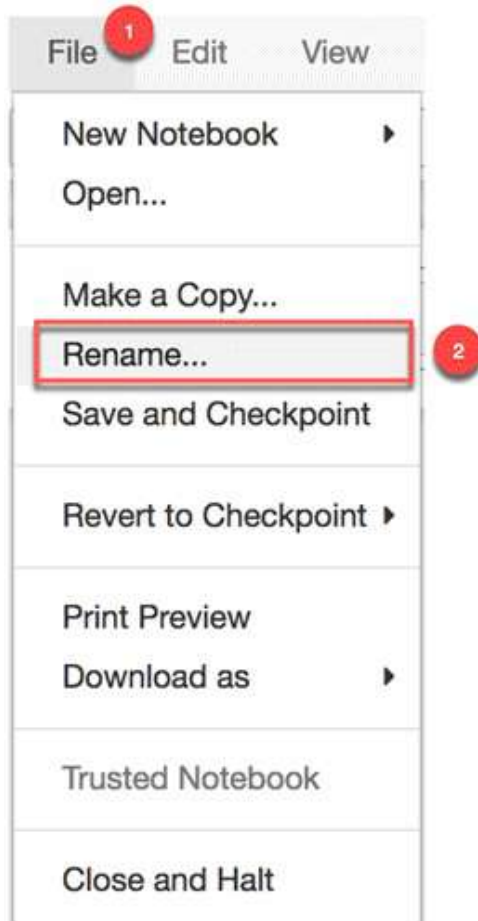
Step 2) You can see the new folder inside the environment. Click on the folder `jupyter_tf`.



Step 3) Inside this folder, you will create your first notebook. Click on the button **New** and **Python 3**.



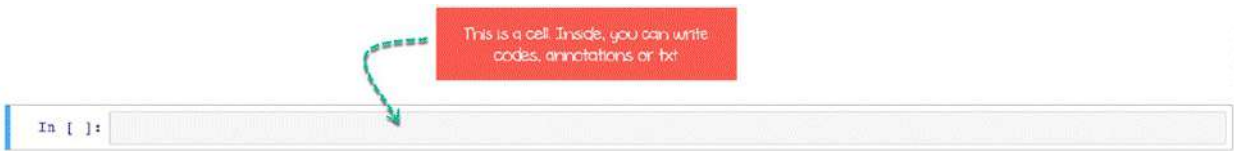
Step 4) You are inside the Jupyter environment. So far, your notebook is called Untitled.ipynb. This is the default name given by Jupyter. Let's rename it by clicking on **File** and **Rename**



You can rename it `Introduction_jupyter`



In Jupyter Notebook, you write codes, annotation or text inside the cells.



Inside a cell, you can write a single line of code.

```
In [1]: # Single line
import matplotlib.pyplot as plt
```

or multiple lines. Jupyter reads the code one line after another.

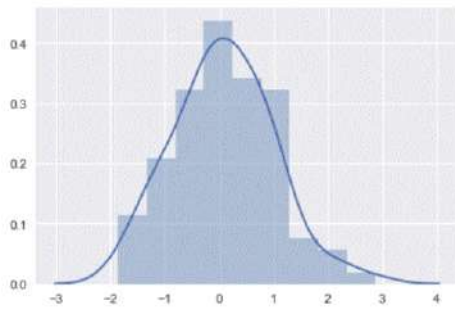
```
In [2]: # Multiple line
import numpy as np
import pandas as pd
from scipy import stats, integrate
```

For instance, if you write following code inside a cell.

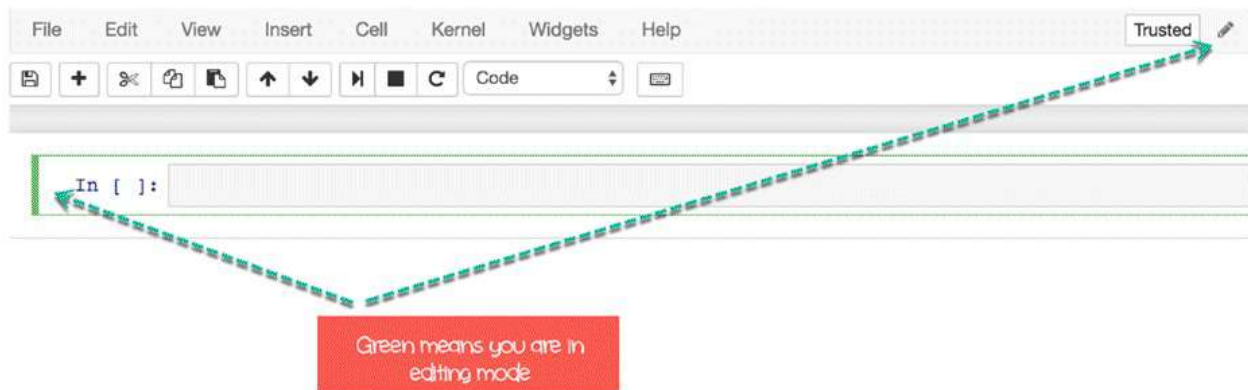
```
In [6]: # Run the code
%matplotlib inline
import seaborn as sns
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "distributions")))
x = np.random.normal(size=100)
sns.distplot(x)
plt
```

It will produce this output.

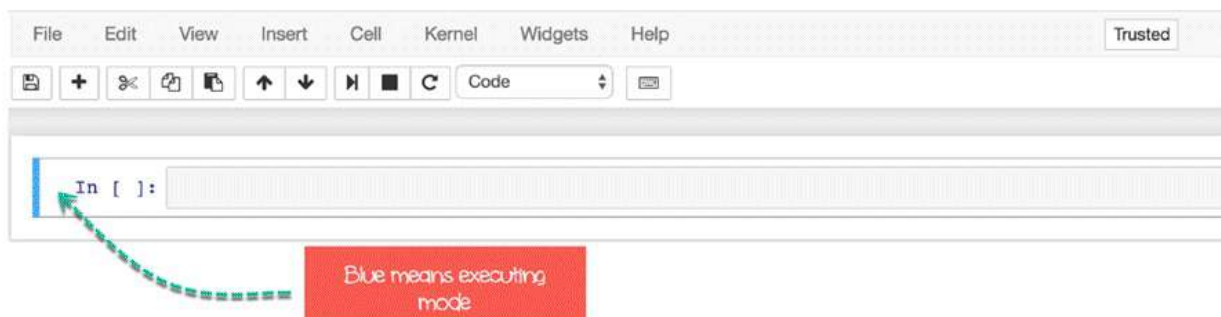
Out[6]: <module 'matplotlib.pyplot' from '/Users/Thomas/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py'>



Step 5) You are ready to write your first line of code. You can notice the cell have two colors. The green color mean you are in the **editing mode**.



The blue color, however, indicates you are in **executing mode**.



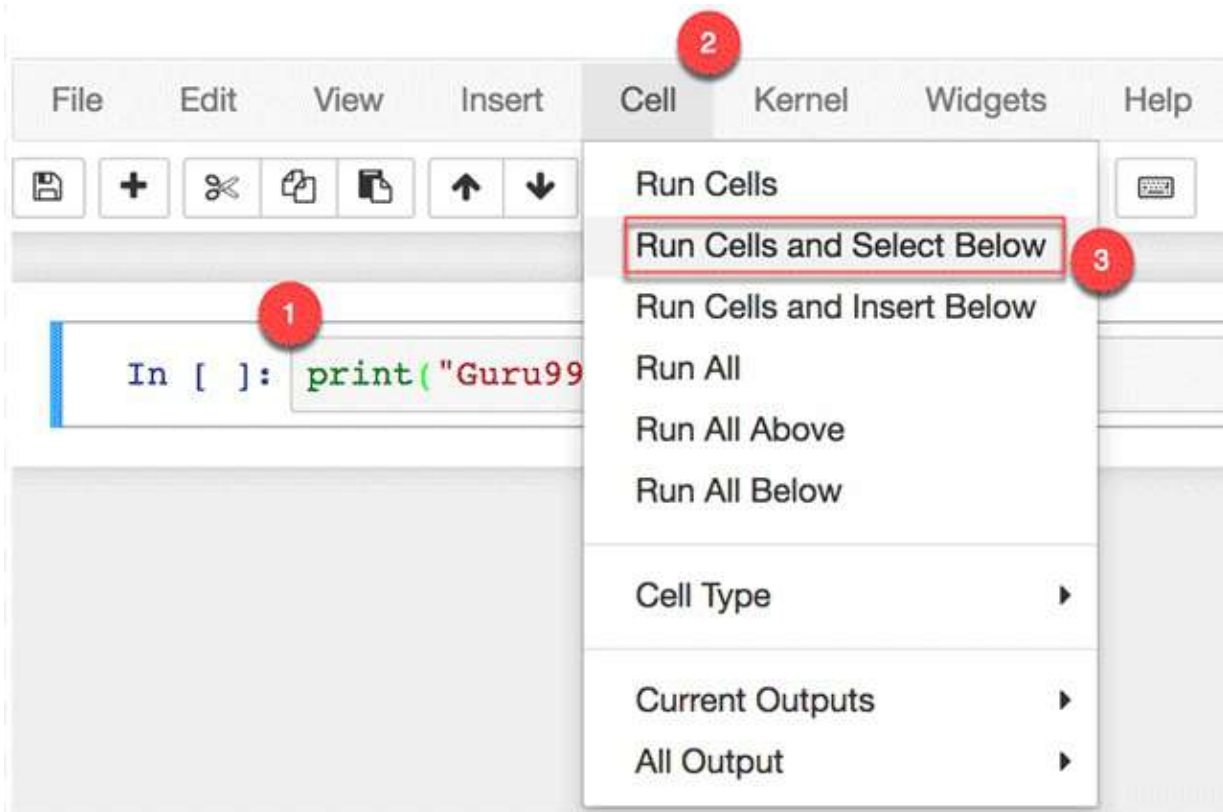
You first line of code will be to print Guru99!. Inside the cell, you can write

```
print("Guru99!")
```

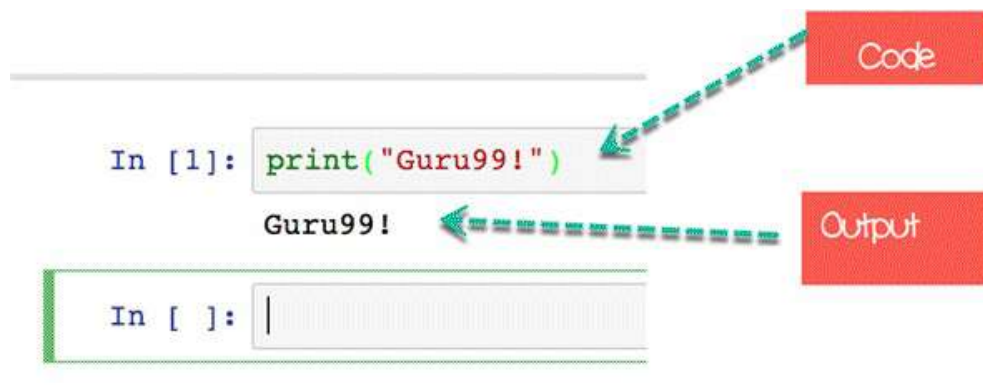
There are two ways to run a code in Jupyter:

- **Click and Run**
- **Keyboard Shortcuts**

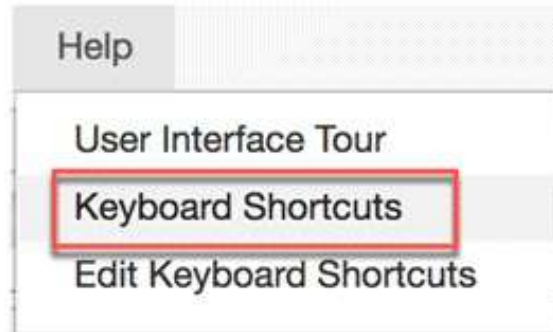
To run the code, you can click on **Cell** and then **Run Cells and Select Below**



You can see the code is printed below the cell and a new cell has appeared right after the output.



A faster way to run a code is to use the **Keyboard Shortcuts**. To access the Keyboard Shortcuts, go to **Help** and **Keyboard Shortcuts**



Below the list of shortcuts for a MacOS keyboard. You can edit the shortcuts in the editor.

Keyboard shortcuts ✕

Command Mode (press Esc to enable) Edit Shortcuts

F : find and replace	⇧↓ : extend selected cells below
↵ : enter edit mode	⇧J : extend selected cells below
⌘⇧F : open the command palette	A : insert cell above
⌘⇧P : open the command palette	B : insert cell below
P : open the command palette	X : cut selected cells
⇧↵ : run cell, select below	C : copy selected cells
⌘↵ : run selected cells	⇧V : paste cells above
⌘↵ : run cell, insert below	V : paste cells below

Following are shortcuts for Windows

Command Mode (press `Esc` to enable)

`F`: find and replace
`Ctrl-Shift-F`: open the command palette
`Ctrl-Shift-P`: open the command palette
`Enter`: enter edit mode
`P`: open the command palette
`Shift-Enter`: run cell, select below
`Ctrl-Enter`: run selected cells
`Alt-Enter`: run cell and insert below
`Y`: change cell to code

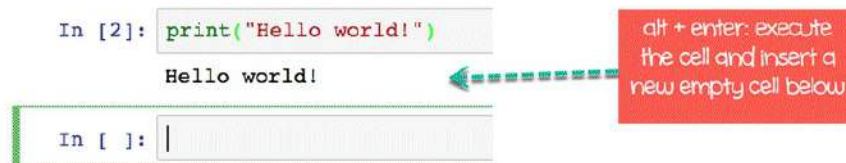
Edit Shortcuts

`Shift-Down`: extend selected cells below
`Shift-J`: extend selected cells below
`A`: insert cell above
`B`: insert cell below
`X`: cut selected cells
`C`: copy selected cells
`Shift-V`: paste cells above
`V`: paste cells below
`Z`: undo cell deletion

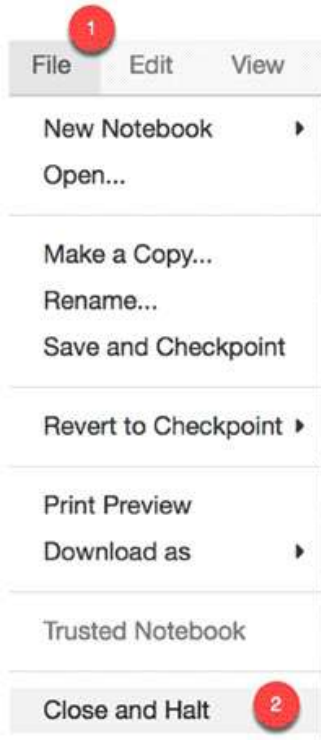
Write this line

```
print("Hello world!")
```

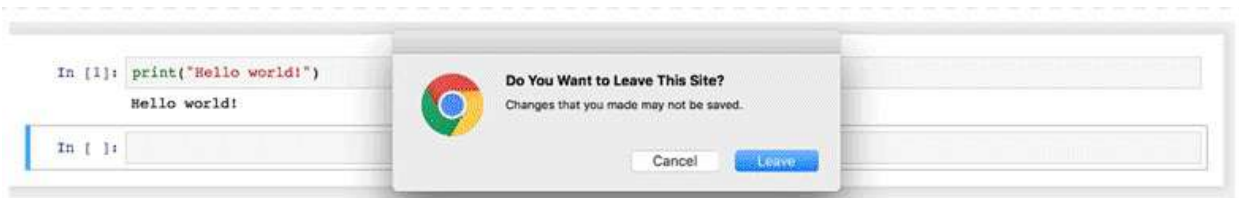
and try to use the Keyboard Shortcuts to run the code. Use `alt+enter`. it will execute the cell and insert a new empty cell below, like you did before.



Step 6) It is time to close the Notebook. Go to **File** and click on **Close and Halt**



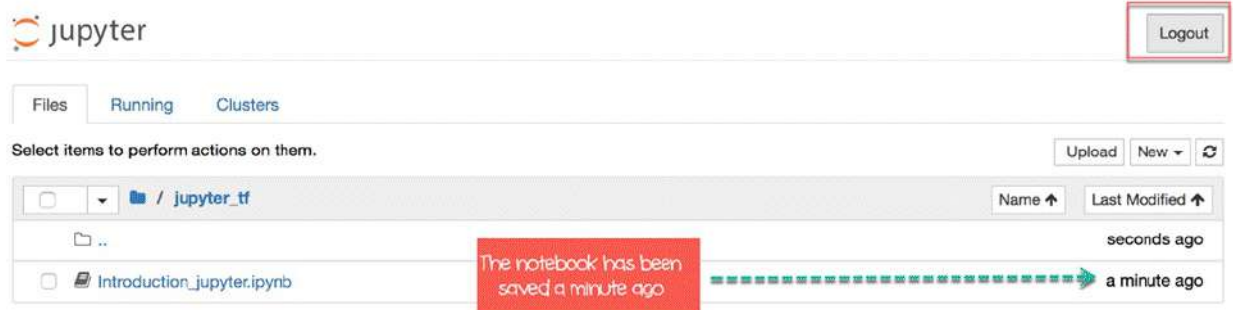
Note: Jupyter automatically saves the notebook with checkpoint. If you have the following message:



It means Jupyter didn't save the file since the last checkpoint. You can manually save the notebook



You will be redirected to the main panel. You can see your notebook has been saved a minute ago. You can safely logout.



Summary

- Jupyter notebook is a web application where you can run your Python and R codes. It is easy to share and deliver rich data analysis with Jupyter.
- To launch jupyter, write in the terminal: `jupyter notebook`
- You can save you notebook wherever you want
- A cell contains your Python code. The kernel will read the code one by one.
- You can use the shortcut to run a cell. By default: `Ctrl+Enter`

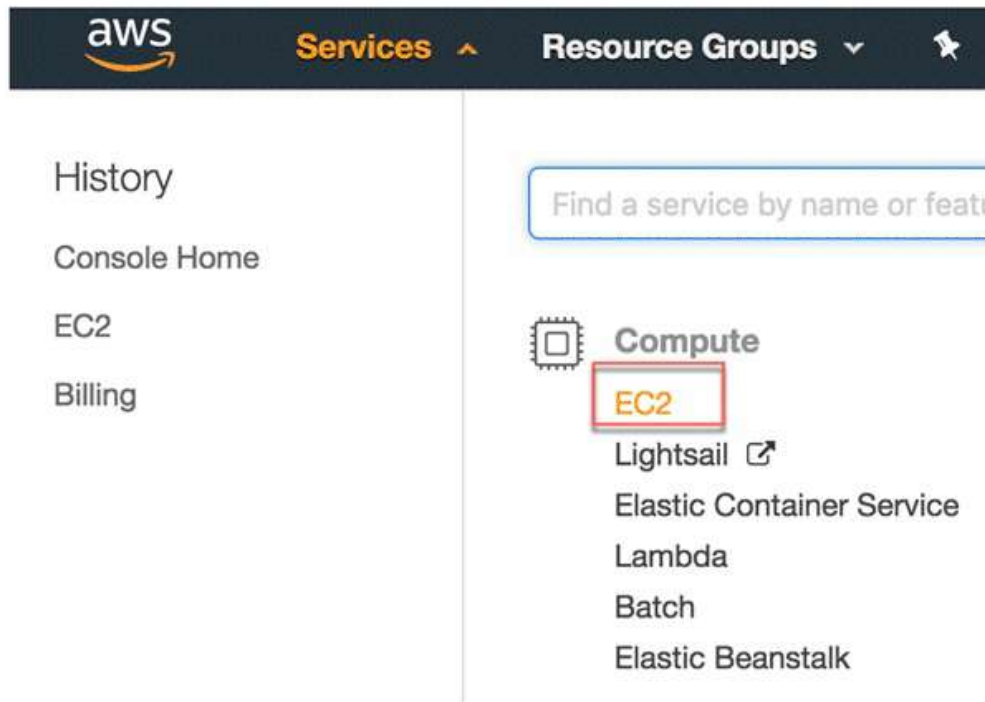
Chapter 7: Tensorflow on AWS

This is a step by step tutorial, to used Jupyter Notebook on AWS

If you do not have an account at AWS, create a free account [here](#).

PART 1: Set up a key pair

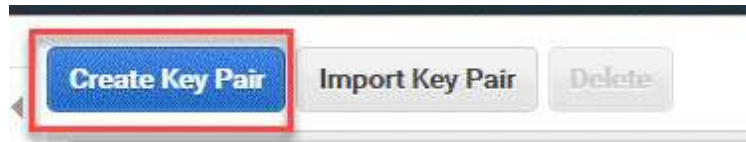
Step 1) Go to **Services** and find **EC2**



Step 2) In the panel and click on **Key Pairs**



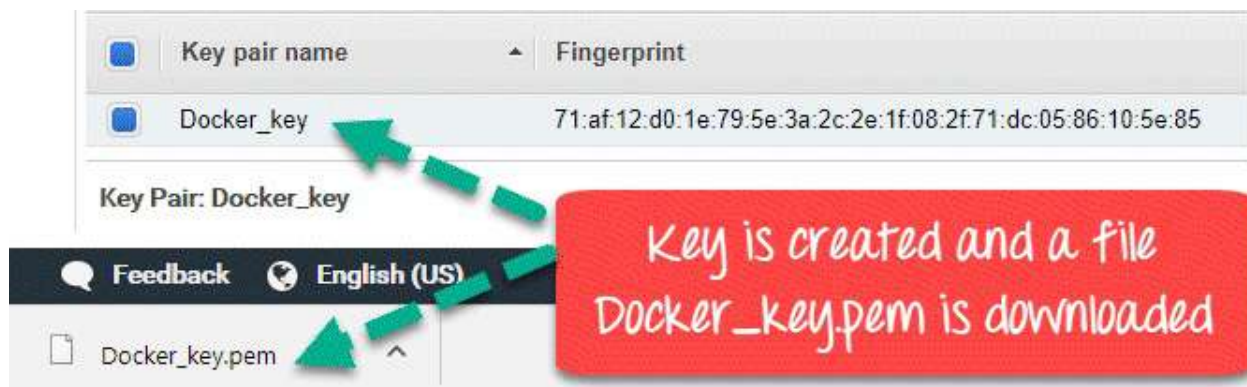
Step 3) Click Create Key Pair



1. You can call it Docker key
2. Click Create



A file name Docker_key.pem downloads.



Step 4) Copy and paste it into the folder key. We will need it soon.

For Mac OS user only

This step concerns only Mac OS user. For Windows or Linux users, please proceed to PART 2

You need to set a working directory that will contain the file key

First of all, create a folder named key. For us, it is located inside the main folder Docker. Then, you set this path as your working directory

```
mkdir Docker/key  
cd Docker/key
```

```
Thomass-MacBook-Pro:~ Thomas$ mkdir Docker/key  
Thomass-MacBook-Pro:~ Thomas$ cd Docker/key  
Thomass-MacBook-Pro:key Thomas$ █
```

Create new folder

Make users/Docker/
key the new working
directory

PART 2: Set up a security group

Step 1) You need to configure a security group. You can access it with the panel



Step 2) Click on Create Security Group



Step 3) In the next Screen

1. Enter Security group name "jupyter_docker" and Description Security Group for Docker
2. You need to add 4 rules on top of
 - ssh: port range 22, source Anywhere
 - http: port range 80, source Anywhere
 - https: port range 443, source Anywhere
 - Custom TCP: port range 8888, source Anywhere
3. Click Create

Create Security Group

Security group name:

Description:

VPC:

Security group rules:

Inbound | Outbound

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom	CIDR, IP or Security Group
HTTP	TCP	80	Custom	0.0.0.0/0, ::/0
HTTPS	TCP	443	Custom	0.0.0.0/0, ::/0
Custom TCP F	TCP	8888	Custom	CIDR, IP or Security Group

Add Rule

Cancel **Create**

For Jupyter

Step 4) The newly created Security Group will be listed

Create Security Group | Actions

Filter by tags and attributes or search by keyword

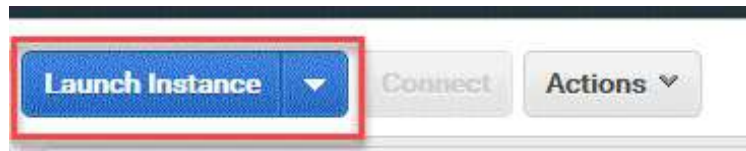
Name	Group ID	Group Name	VPC ID	Description
	sg-c3424a89	jupyter_docker	vpc-620d0207	Security Group for Docker

Part 3: Launch instance

You are finally ready to create the instance

- INSTANCES
- Instances**
- Launch Templates
- Spot Requests
- Reserved Instances
- Dedicated Hosts

Step 1) Click on Launch Instance



The default server is enough for your need. You can choose Amazon Linux AMI. The current instance is 2018.03.0.

AMI stands for Amazon Machine Image. It contains the information required to successfully start an instance that runs on a virtual server stored in the cloud.



Note that AWS has a server dedicated to deep learning such as:

- **Deep Learning AMI (Ubuntu)**
- **Deep Learning AMI**
- **Deep Learning Base AMI (Ubuntu)**

All of them comes with latest binaries of deep learning frameworks pre-installed in separate virtual environments:

- TensorFlow,
- Caffe
- PyTorch,
- Keras,
- Theano
- CNTK.

Fully-configured with NVIDIA CUDA, cuDNN and NCCL as well as Intel MKL-DNN

Step 2) Choose t2.micro. It is a free tier server. AWS offers for free this virtual machine equipped with 1 vCPU and 1 GB of memory. This server provides a good tradeoff between computation, memory and network performance. It fits for small and medium database

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 2: Choose an Instance Type

Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn more about instance types and how they can meet your computing needs.

Filter by: All instance types Current generation Show/Hide Columns

Currently selected: t2.micro (Variable ECU's, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes

Cancel Previous Review and Launch Next: Configure Instance Details

Step 3) Keep settings default in next screen and click Next: Add Storage

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 3: Configure Instance Details

Network: vpc-620d0207 (default) Create new VPC

Subnet: No preference (default subnet in any Availability Zone) Create new subnet

Auto-assign Public IP: Use subnet setting (Enable)

Placement group: Add instance to placement group.

IAM role: None Create new IAM role

Shutdown behavior: Stop

Enable termination protection: Protect against accidental termination

Monitoring: Enable CloudWatch detailed monitoring. Additional charges apply.

Tenancy: Shared - Run a shared hardware instance. Additional charges will apply for dedicated tenancy.

T2 Unlimited: Enable. Additional charges may apply.

Cancel Previous Review and Launch Next: Add Storage

Step 4) Increase storage to 10GB and click Next

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MiB/s)	Delete on Termination	Encrypted
Root	/dev/xvda	snap-07ad5635957a48b3e	10	General Purpose SSD (GP2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

[Add New Volume](#)

Free tier eligible customers can get up to 30 GiB of EBS General Purpose (SSD) or Magnetic storage. [Learn more about free usage tier eligibility and usage restrictions.](#)

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Tags](#)

Step 5) Keep settings default and click Next: Configure Security Group

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. Learn more about tagging your Amazon EC2 resources.

Key (127 characters maximum) **Value** (255 characters maximum) **Instances** **Volumes**

This resource currently has no tags.

Choose the **Add tag** button or **click** to add a Name tag. Make sure your IAM policy includes permissions to create tags.

[Add Tag](#) (Up to 50 tags maximum)

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Configure Security Group](#)

Step 6) Choose the security group you created before, which is **jupyter_docker**

Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: Create a new security group Select an existing security group

Security Group ID	Name	Description	Actions
sg-4ee20c29	default	default VPC security group	Copy to new
<input checked="" type="checkbox"/> sg-c3424a89	jupyter_docker	Security Group for Docker	Copy to new
sg-055d3ae27	launch-wizard-1	launch-wizard-1 created 2017-06-15T10:09:34.308+00:30	Copy to new

Inbound rules for sg-c3424a89 (Selected security groups: sg-c3424a89)

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	:::0	
Custom TCP Rule	TCP	8888	0.0.0.0/0	

[Cancel](#) [Previous](#) [Review and Launch](#)

Step 7) Review your settings and Click the launch button

Step 7: Review Instance Launch

Please review your instance launch details. You can go back to edit changes for each section. Click **Launch** to assign a key pair to your instance and complete the launch process.

⚠ Improve your instances' security. Your security group, jupyter_docker, is open to the world.
Your instances may be accessible from any IP address. We recommend that you update your security group rules to allow access from known IP addresses only. You can also open additional ports in your security group to facilitate access to the application or service you're running, e.g., HTTP (80) for web servers. [Edit security groups](#)

▼ AMI Details Edit AMI

Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-0fe4b2b0

Free tier Available Roll Back to Previous Version Simulate the new item

The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.

Security Group ID	Name	Description
-------------------	------	-------------

Cancel Previous Launch

Step 8) The last step is to link the key pair to the instance.

Select an existing key pair or create a new key pair ✕

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. [Learn more about removing existing key pairs from a public AMI.](#)

Choose an existing key pair ▼

Select a key pair 1

Docker_key ▼

I acknowledge that I have access to the selected private key file (Docker_key.pem), and that 2 without this file, I won't be able to log into my instance.

Cancel Launch Instances 3

Step 8) Instance will launch

Launch Status

✔ Your instances are now launching
The following instance launches have been initiated: i-090447e9c051efdce [View launch log](#)

ℹ Get notified of estimated charges
Create billing alerts to get an email notification when estimated charges on your AWS bill exceed an amount you define (for ex...

How to connect to your instances

Step 9) Below a summary of the instances currently in use. Note the public IP

The screenshot shows the AWS Management Console interface for an EC2 instance. The instance name is i-090447e9c051efdce, type is t2.micro, and it is in the us-east-1b availability zone. The instance state is 'running'. The public DNS is ec2-52-23-241-75.compute-1.amazonaws.com. The public IP address is 52.23.241.75, which is highlighted with a red box. A green dashed arrow points from the public IP in the table above to the highlighted IP in the details section.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)	IPv4 Public IP
	i-090447e9c051efdce	t2.micro	us-east-1b	running	Initializing	None	ec2-52-23-241-75.com...	52.23.241.75

Instance: i-090447e9c051efdce Public DNS: ec2-52-23-241-75.compute-1.amazonaws.com

Description	Status Checks	Monitoring	Tags
Instance ID: i-090447e9c051efdce	Instance state: running	Instance type: t2.micro	Elastic IPs:

Public DNS (IPv4): ec2-52-23-241-75.compute-1.amazonaws.com
IPv4 Public IP: 52.23.241.75
IPv6 IPs:
Private DNS: ip-172-31-57-55.ec2.internal

Step 9) Click on Connect

The screenshot shows the AWS Management Console interface for an EC2 instance. The 'Connect' button is highlighted with a red box. Below the button is a search bar and a table of instance details.

Launch Instance **Connect** Actions

Filter by tags and attributes or search by keyword

Name	Instance ID	Instance Type	Availability Zone
	i-090447e9c051efdce	t2.micro	us-east-1b

You will find the connection details

Connect To Your Instance



I would like to connect with A standalone SSH client
 A Java SSH Client directly from my browser (Java required)

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (Docker_key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 Docker_key.pem
```

4. Connect to your instance using its Public DNS:

```
ec2-18-188-151-171.us-east-2.compute.amazonaws.com
```

Example:

```
ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-171.us-east-2.compute.amazonaws.com
```

Please note that in most cases the username above will be correct, however please ensure that you read your AMI usage instructions to ensure that the AMI owner has not changed the default AMI username.

If you need any assistance connecting to your instance, please see our [connection documentation](#).

Launch your instance (Mac OS users)

At first make sure that inside the terminal, your working directory points to the folder with the key pair file docker

run the code

```
chmod 400 docker.pem
```

Open the connection with this code.

There are two codes. in some case, the first code avoids Jupyter to open the notebook.

In this case, use the second one to force the connection.

```
# If able to launch Jupyter  
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-
```

2.compute.amazonaws.com

```
# If not able to launch Jupyter
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

The first time, you are prompted to accept the connection

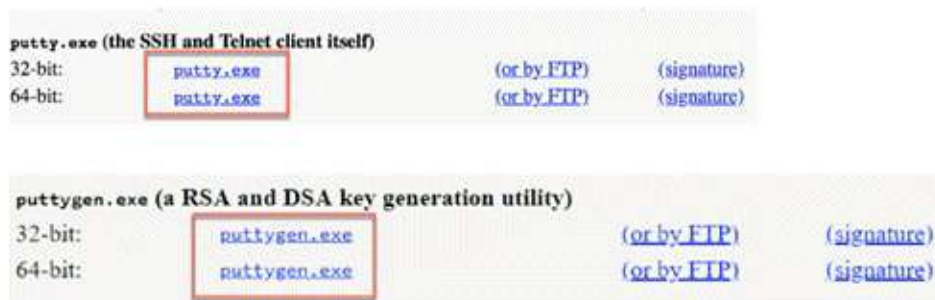
```
Thomass-MacBook-Pro:~$ ssh -i "Docker_key.pem" ec2-user@ec2-18-188-151-171.us-east-2.compute.amazonaws.com
-L 8888:127.0.0.1:8888
The authenticity of host 'ec2-18-188-151-171.us-east-2.compute.amazonaws.com (18.188.151.171)' can't be established.
ECDSA key fingerprint is SHA256:UuNljpxnup20pilz0T1LL60Z1o3TdyE86kB6Pmujf0.
Are you sure you want to continue connecting (yes/no)?
```

Launch your instance (Windows users)

Step 1) Go to this website to download PuTTY and PuTTYgen PuTTY

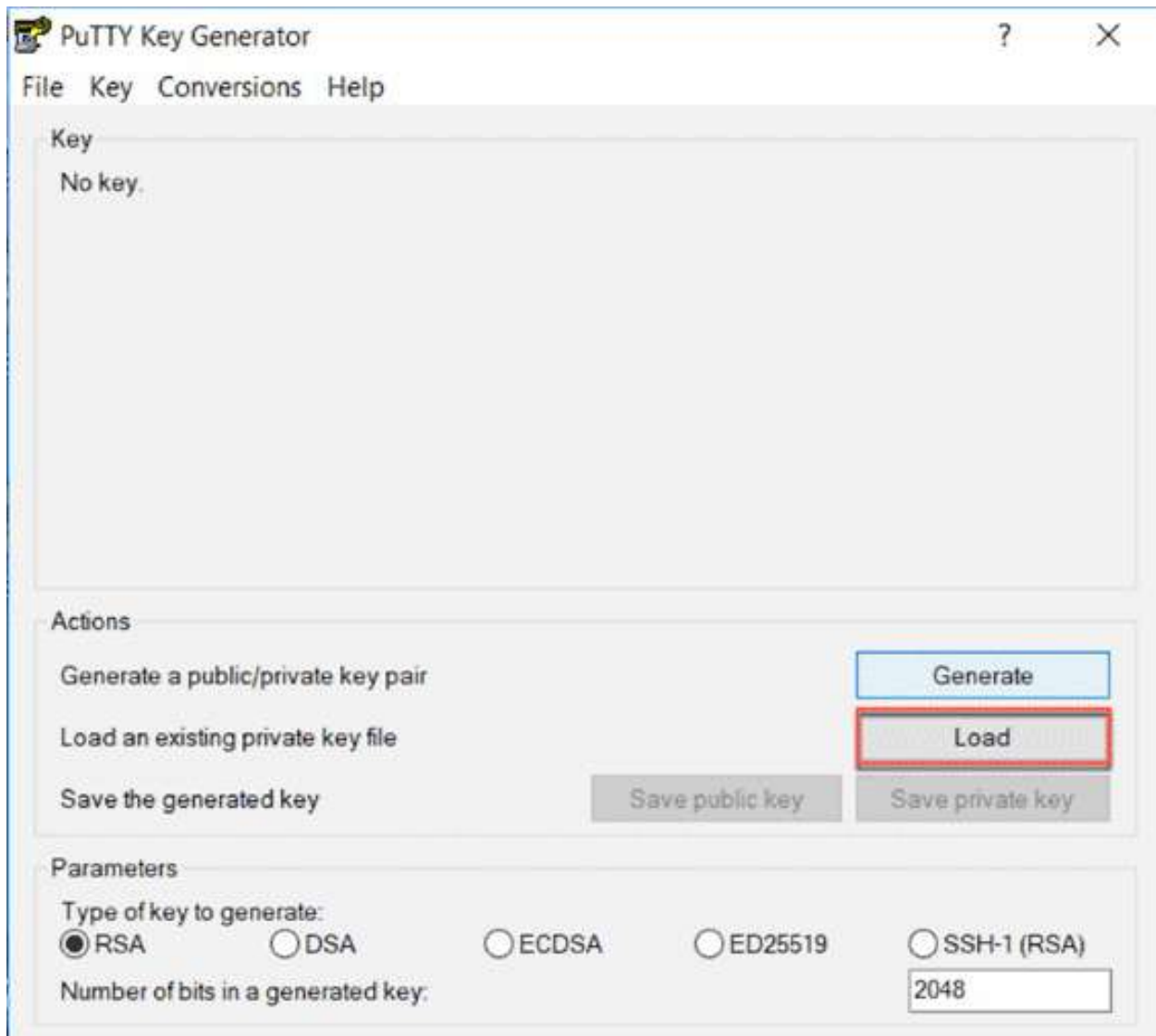
You need to download

- PuTTY: launch the instance
- PuTTYgen: convert the pem file to ppk

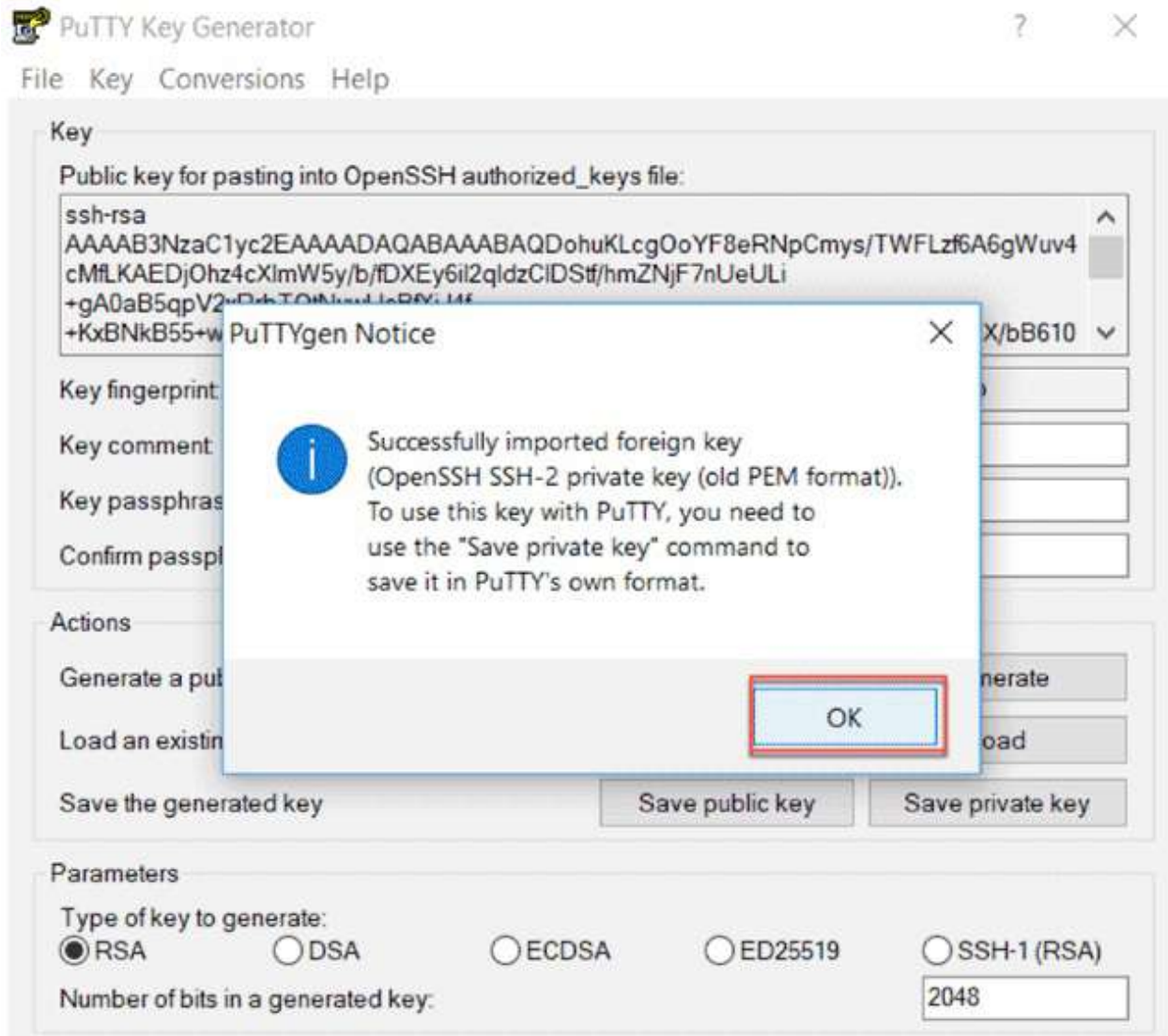


Now that both software are installed, you need to convert the .pem file to .ppk. PuTTY can only read .ppk. The pem file contains the unique key created by AWS.

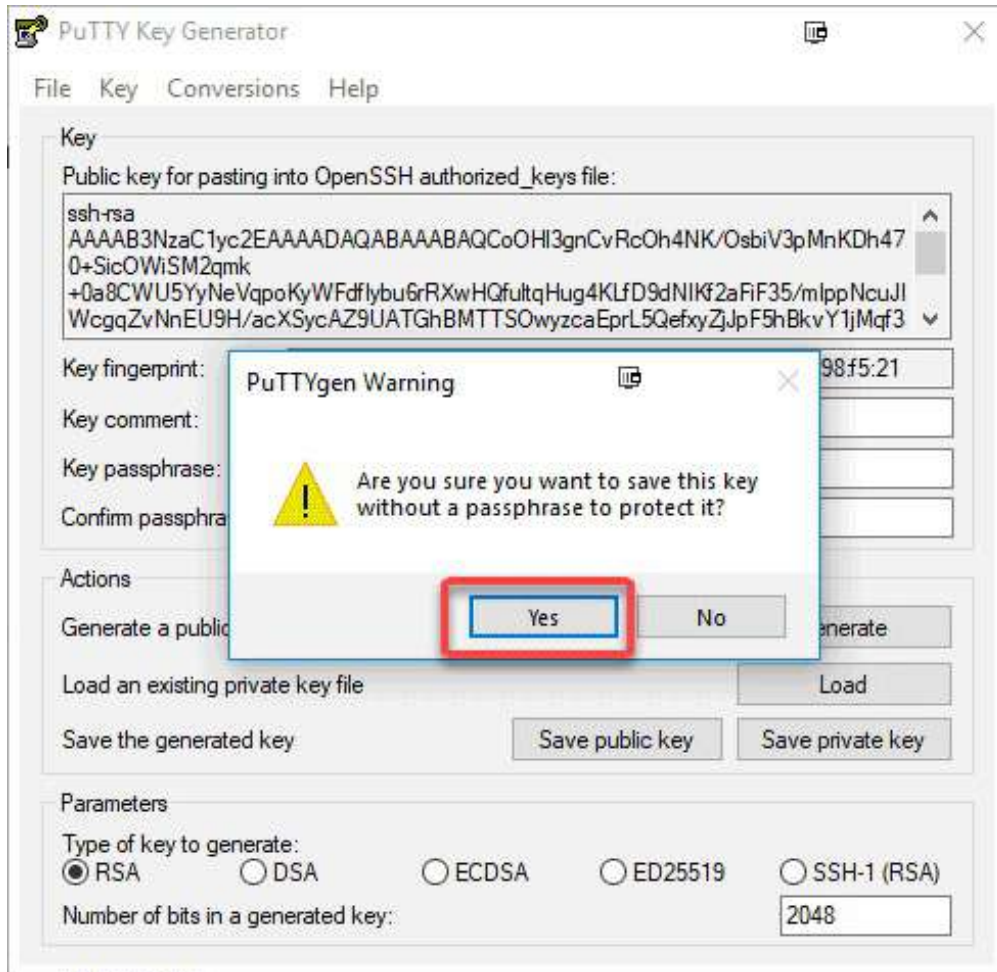
Step 2) Open PuTTYgen and click on Load. Browse the folder where the .pem file is located.



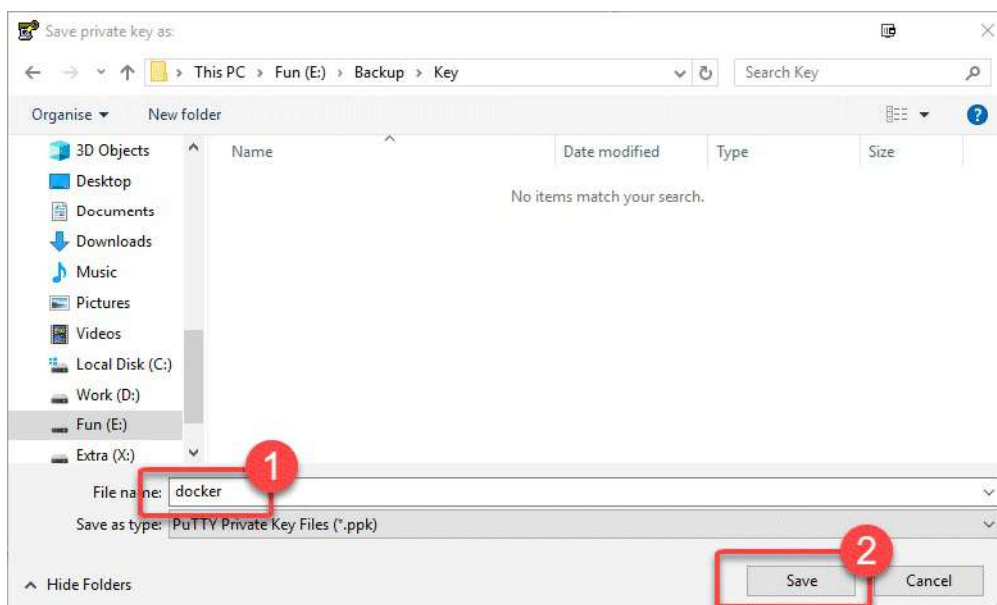
Step 3) After you loaded the file, you should get a notice informing you that the key has been successfully imported. Click on OK



Step 4) Then click on Save private key. You are asked if you want to save this key without a passphrase. Click on yes.



Step 5) Save the Key

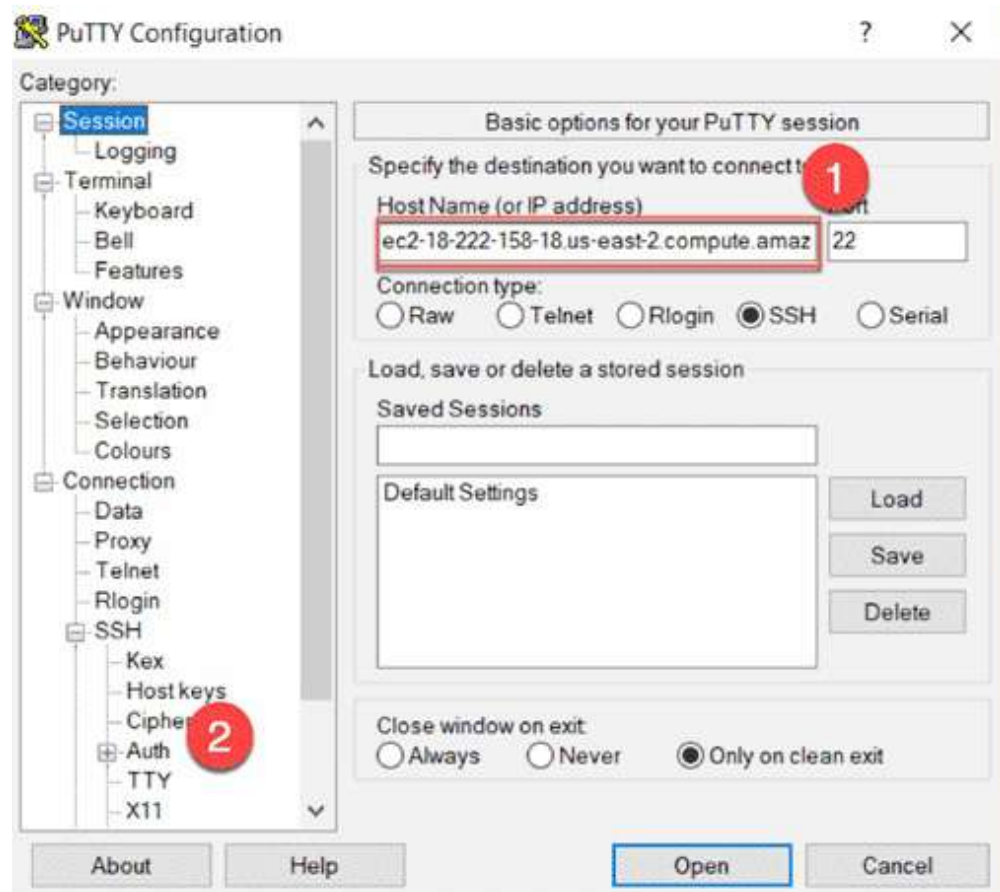


Step 6) Go to AWS and copy the public DNS

4. Connect to your instance using its Public DNS:

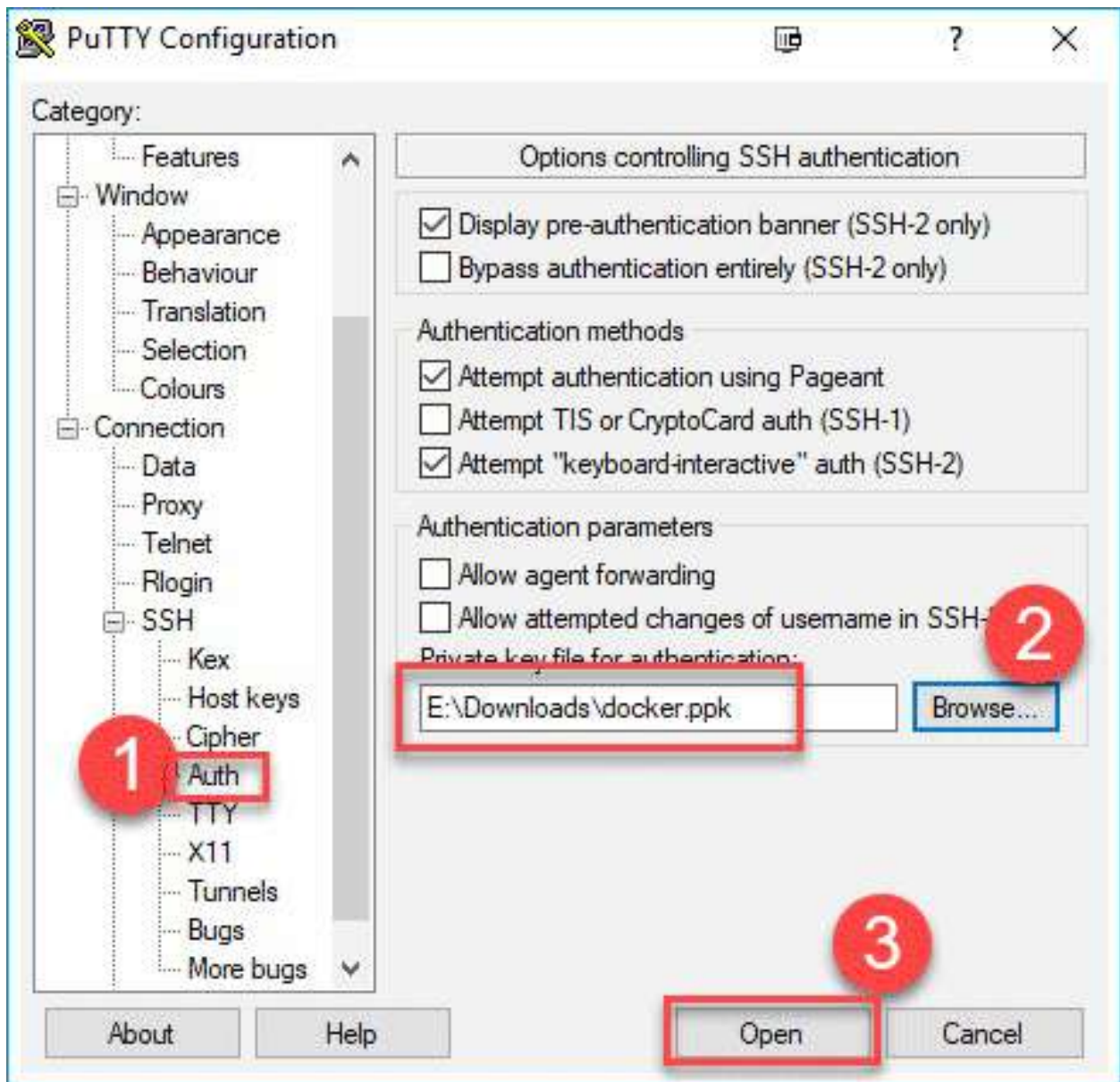
ec2-13-59-162-131.us-east-2.compute.amazonaws.com

Open PuTTY and paste the Public DNS in the Host Name



Step 7)

1. On the left panel, unfold SSH and open Auth
2. Browse the Private Key. You should select the .ppk
3. Click on Open.



Step 8)

When this step is done, a new window will be opened. Click Yes if you see this pop-up



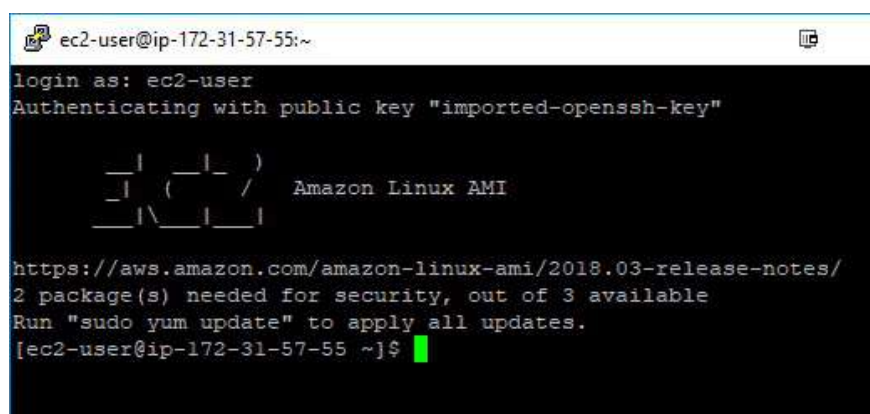
Step 9)

You need to login as: ec2-user



Step 10)

You are connected to the Amazon Linux AMI.



Part 4: Install Docker

While you are connected with the server via Putty/Terminal, you can install **Docker** container.

Execute the following codes

```
sudo yum update -y
sudo yum install -y docker
sudo service docker start
sudo user-mod -a -G docker ec2-user
exit
```

Launch again the connection

```
ssh -i "docker.pem" ec2-user@ec2-18-219-192-34.us-east-2.compute.amazonaws.com -L 8888:127.0.0.1:8888
```

Windows users use SSH as mentioned above

Part 5: Install Jupyter

Step 1) Create Jupyter with a pre-built image

```
## Tensorflow
docker run -v ~/work:/home/jovyan/work -d -p 8888:8888
jupyter/tensorflow-notebook
## Sparkdocker
run -v ~/work:/home/jovyan/work -d -p 8888:8888 jupyter/pyspark-
notebook
```

Code Explanation

- docker run: Run the image
- v: attach a volume
- ~/work:/home/jovyan/work: Volume
- 8888:8888: port
- jupyter/datascience-notebook: Image

For other pre-build images, go here

Allow preserving Jupyter notebook

```
sudo chown 1000 ~/work
```

Step 2) Install tree to see our working directory next

```
sudo yum install -y tree
```

```
[ec2-user@ip-172-31-16-239 ~]$ tree
├── work
1 directory, 0 files
```

Step 3)

1. Check the container and its name (changes with every installation) Use command

```
docker ps
```

2. Get the name and use the log to open Jupyter. In the tutorial, the container's name is vigilant_easley. Use command

```
docker logs vigilant_easley
```

3. Get the URL

```
cc2-user@ip-172-31-57-55~  
[cc2-user@ip-172-31-57-55 ~]$ docker ps  
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES  
90a3c09282d6       jupyter/tensorflow-notebook  "tini -g -- start-no..."  3 minutes ago      Up 3 minutes       0.0.0.0:8888->8888/tcp  vigilant_easley  
[cc2-user@ip-172-31-57-55 ~]$ docker logs vigilant_easley  
/usr/local/bin/start-notebook.sh: ignoring /usr/local/bin/...-notebook.d/*  
  
Container must be run with group "root" to update passwd file  
Executing the command: jupyter notebook  
[I 08:06:47.206 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.local/share/jupyter/runtime/notebook_cookie_secret  
[W 08:06:47.671 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.  
[I 08:06:47.724 NotebookApp] JupyterLab extension loaded from /opt/conda/lib/python3.6/site-packages/jupyterlab  
[I 08:06:47.735 NotebookApp] JupyterLab application directory is /opt/conda/share/jupyter/lab  
[I 08:06:47.735 NotebookApp] Serving notebooks from local directory: /home/jovyan  
[I 08:06:47.735 NotebookApp] The Jupyter Notebook is running at:  
[I 08:06:47.735 NotebookApp] http://[90a3c09282d6 or 127.0.0.1]:8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed  
[I 08:06:47.735 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).  
[C 08:06:47.736 NotebookApp]  
  
Copy/paste this URL into your browser when you connect for the first time,  
to login with a token:  
http://[90a3c09282d6 or 127.0.0.1]:8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed  
[cc2-user@ip-172-31-57-55 ~]$
```

Step 4)

In the URL

[http://\(90a3c09282d6 or 127.0.0.1\):8888/?](http://(90a3c09282d6 or 127.0.0.1):8888/?)

[token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed](http://(90a3c09282d6 or 127.0.0.1):8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed)

Replace (90a3c09282d6 or 127.0.0.1) with Public DNS of your instance

Connect To Your Instance ✕

I would like to connect with

- A standalone SSH client ⓘ
- A Java SSH Client directly from my browser (Java required) ⓘ

To access your instance:

1. Open an SSH client. (find out how to [connect using PuTTY](#))
2. Locate your private key file (Docker_key.pem). The wizard automatically detects the key you used to launch the instance.
3. Your key must not be publicly viewable for SSH to work. Use this command if needed:

```
chmod 400 Docker_key.pem
```
4. Connect to your instance using its Public DNS:

ec2-174-129-135-16.compute-1.amazonaws.com

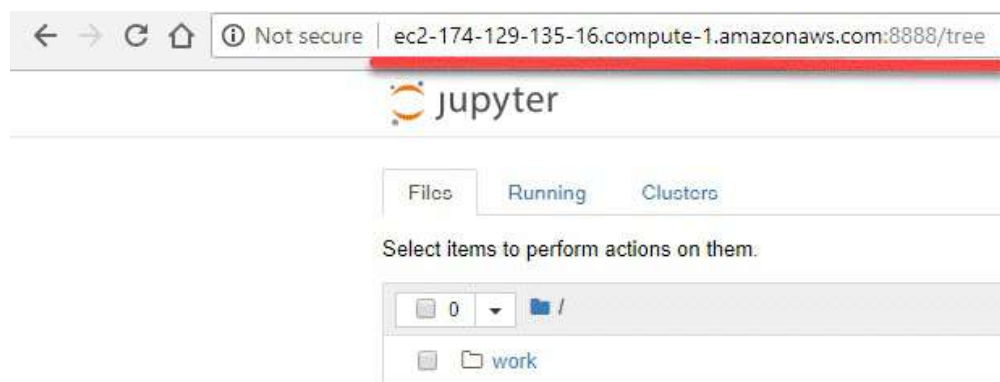
Example:

Step 5)

The new URL becomes

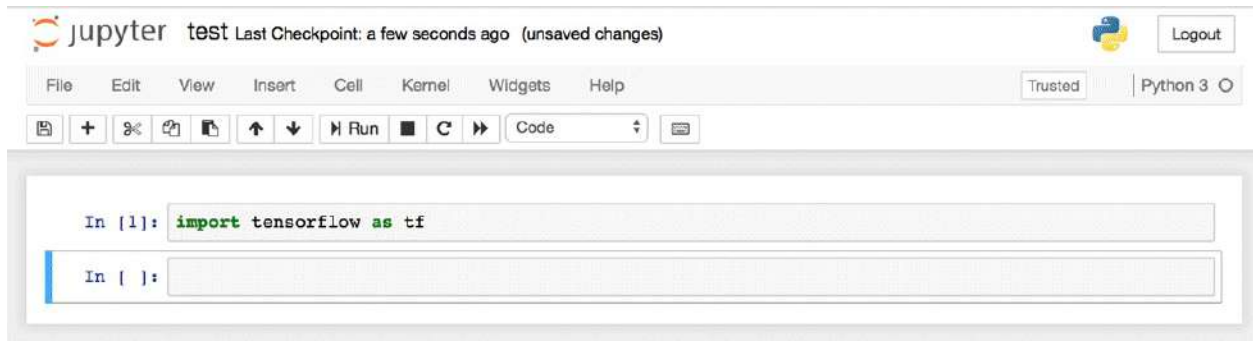
<http://ec2-174-129-135-16.compute-1.amazonaws.com:8888/?token=f460f1e79ab74c382b19f90fe3fd55f9f99c5222365eceed>

Step 6) Copy and paste the URL into your browser. Jupyter Opens



Step 7)

You can write a new Notebook in the work folder

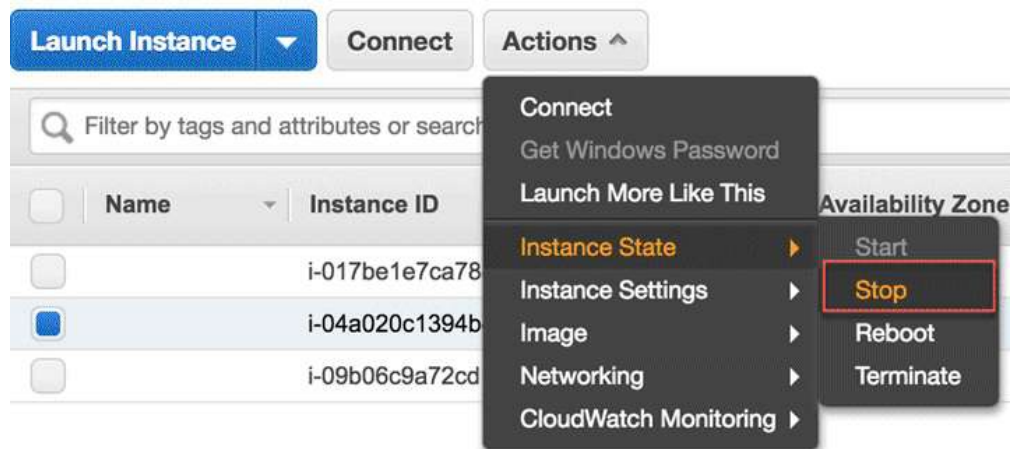


Part 6: Close connection

Close the connection in the terminal

```
exit
```

Go back to AWS and stop the server.



Troubleshooting

If ever docker doesnot work, try to rebuilt image using

```
docker run -v ~/work:/home/jovyan/work -d -p 8888:8888  
jupyter/tensorflow-notebook
```

Chapter 8: TensorFlow Basics: Tensor, Shape, Type, Graph, Sessions & Operators

What is a Tensor?

Tensorflow's name is directly derived from its core framework: **Tensor**. In Tensorflow, all the computations involve tensors. A tensor is a **vector** or **matrix** of n-dimensions that represents all types of data. All values in a tensor hold identical data type with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.

A tensor can be originated from the input data or the result of a computation. In TensorFlow, all the operations are conducted inside a **graph**. The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.

The graph outlines the ops and connections between the nodes. However, it does not display the values. The edge of the nodes is the tensor, i.e., a way to populate the operation with data.

In Machine Learning, models are feed with a list of objects called **feature vectors**. A feature vector can be of any data type. The feature vector will usually be the primary input to populate a tensor. These values will flow into an op node through the tensor and the result of this operation/computation will create a new tensor which in turn will be used in a new operation. All these operations can be viewed in the graph.

Representation of a Tensor

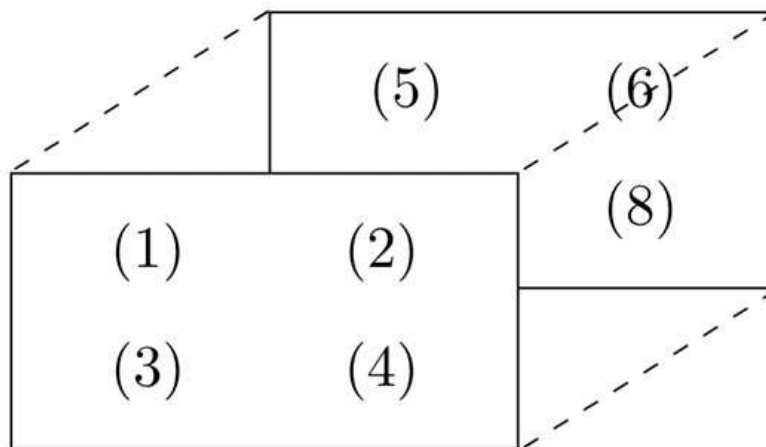
In TensorFlow, a tensor is a collection of feature vectors (i.e., array) of n-dimensions. For instance, if we have a 2x3 matrix with values from 1 to 6, we write:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

TensorFlow represents this matrix as:

```
[[1, 2, 3],  
 [4, 5, 6]]
```

If we create a three-dimensional matrix with values from 1 to 8, we have:



TensorFlow represents this matrix as:

```
[ [[1, 2],  
   [[3, 4],  
   [[5, 6],  
   [[7, 8] ]
```

Note: A tensor can be represented with a scalar or can have a shape of more than three dimensions. It is just more complicated to visualize higher dimension level.

Types of Tensor

In TensorFlow, all the computations pass through one or more tensors. A tensor is an object with three properties:

- A unique label (name)
- A dimension (shape)
- A data type (dtype)

Each operation you will do with TensorFlow involves the manipulation of a tensor. There are four main tensors you can create:

- `tf.Variable`
- `tf.constant`
- `tf.placeholder`
- `tf.SparseTensor`

In this tutorial, you will learn how to create a `tf.constant` and a `tf.Variable`.

Before we go through the tutorial, make sure you activate the conda environment with TensorFlow. We named this environment `hello-tf`.

For MacOS user:

```
source activate hello-tf
```

For Windows user:

```
activate hello-tf
```

After you have done that, you are ready to import tensorflow

```
# Import tf
import tensorflow as tf
```

Create a tensor of n-dimension

You begin with the creation of a tensor with one dimension, namely a scalar.

To create a tensor, you can use `tf.constant()`

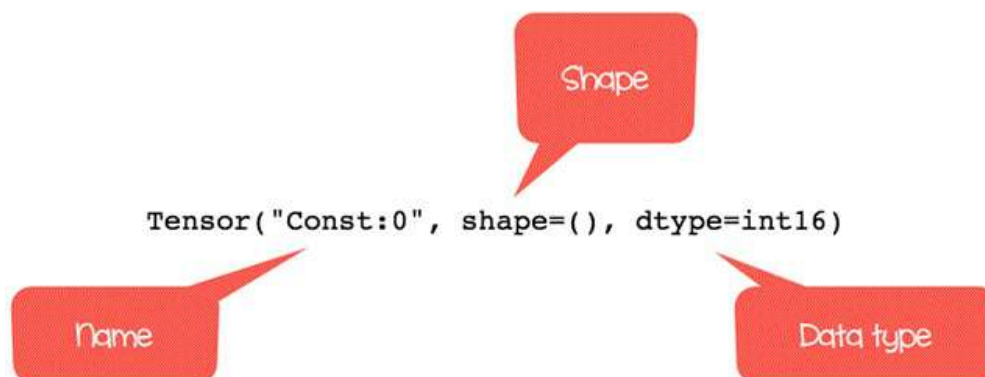
```
tf.constant(value, dtype, name = "")
arguments
- `value`: Value of n dimension to define the tensor. Optional
- `dtype`: Define the type of data:
  - `tf.string`: String variable
  - `tf.float32`: Flot variable
  - `tf.int16`: Integer variable
- "name": Name of the tensor. Optional. By default, `Const_1:0`
```

To create a tensor of dimension 0, run the following code

```
## rank 0
# Default name
r1 = tf.constant(1, tf.int16)
print(r1)
```

Output

```
Tensor("Const:0", shape=(), dtype=int16)
```



```
# Named my_scalar
r2 = tf.constant(1, tf.int16, name = "my_scalar")
```

```
print(r2)
```

Output

```
Tensor("my_scalar:0", shape=(), dtype=int16)
```

Each tensor is displayed by the tensor name. Each tensor object is defined with a unique label (name), a dimension (shape) and a data type (dtype).

You can define a tensor with decimal values or with a string by changing the type of data.

```
# Decimal
r1_decimal = tf.constant(1.12345, tf.float32)
print(r1_decimal)
# String
r1_string = tf.constant("Guru99", tf.string)
print(r1_string)
```

Output

```
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("Const_2:0", shape=(), dtype=string)
```

A tensor of dimension 1 can be created as follow:

```
## Rank 1
r1_vector = tf.constant([1,3,5], tf.int16)
print(r1_vector)
r2_boolean = tf.constant([True, True, False], tf.bool)
print(r2_boolean)
```

Output

```
Tensor("Const_3:0", shape=(3,), dtype=int16)
Tensor("Const_4:0", shape=(3,), dtype=bool)
```

You can notice the shape is only composed of 1 column.

To create an array of 2 dimensions, you need to close the brackets after each row. Check the examples below

```
## Rank 2
r2_matrix = tf.constant([ [1, 2],
                          [3, 4] ], tf.int16)
print(r2_matrix)
```

Output

```
Tensor("Const_5:0", shape=(2, 2), dtype=int16)
```

The matrix has 2 rows and 2 columns filled with values 1, 2, 3, 4.

A matrix with 3 dimensions is constructed by adding another level with the brackets.

```
## Rank 3
r3_matrix = tf.constant([ [[1, 2],
                           [3, 4],
                           [5, 6]] ], tf.int16)
print(r3_matrix)
```

Output

```
Tensor("Const_6:0", shape=(1, 3, 2), dtype=int16)
```

The matrix looks like the picture two.

Shape of tensor

When you print the tensor, TensorFlow guesses the shape. However, you can get the shape of the tensor with the shape property.

Below, you construct a matrix filled with a number from 10 to 15 and you check the shape of m_shape

```
# Shape of tensor
m_shape = tf.constant([ [10, 11],
                        [12, 13],
                        [14, 15] ])
m_shape.shape
```

Output

```
TensorShape([Dimension(3), Dimension(2)])
```

The matrix has 3 rows and 2 columns.

TensorFlow has useful commands to create a vector or a matrix filled with 0 or 1. For instance, if you want to create a 1-D tensor with a specific shape of 10, filled with 0, you can run the code below:

```
# Create a vector of 0
print(tf.zeros(10))
```

Output

```
Tensor("zeros:0", shape=(10,), dtype=float32)
```

The property works for matrix as well. Here, you create a 10x10 matrix filled with 1

```
# Create a vector of 1
print(tf.ones([10, 10]))
```

Output

```
Tensor("ones:0", shape=(10, 10), dtype=float32)
```

You can use the shape of a given matrix to make a vector of ones. The matrix `m_shape` is a 3x2 dimensions. You can create a tensor with 3 rows filled by one's with the following code:

```
# Create a vector of ones with the same number of rows as m_shape
print(tf.ones(m_shape.shape[0]))
```

Output

```
Tensor("ones_1:0", shape=(3, ), dtype=float32)
```

If you pass the value 1 into the bracket, you can construct a vector of ones equals to the number of columns in the matrix `m_shape`.

```
# Create a vector of ones with the same number of column as m_shape
print(tf.ones(m_shape.shape[1]))
```

Output

```
Tensor("ones_2:0", shape=(2, ), dtype=float32)
```

Finally, you can create a matrix 3x2 with only one's

```
print(tf.ones(m_shape.shape))
```

Output

```
Tensor("ones_3:0", shape=(3, 2), dtype=float32)
```


Type of data

The second property of a tensor is the type of data. A tensor can only have one type of data at a time. A tensor can only have one type of data. You can return the type with the property `dtype`.

```
print(m_shape.dtype)
```

Output

```
<dtype: 'int32'>
```

In some occasions, you want to change the type of data. In TensorFlow, it is possible with `tf.cast` method.

Example

Below, a float tensor is converted to integer using you use the method `cast`.

```
# Change type of data
type_float = tf.constant(3.123456789, tf.float32)
type_int = tf.cast(type_float, dtype=tf.int32)
print(type_float.dtype)
print(type_int.dtype)
```

Output

```
<dtype: 'float32'>
<dtype: 'int32'>
```

TensorFlow chooses the type of data automatically when the argument is not specified during the creation of the tensor. TensorFlow will guess what is the most likely types of data. For instance, if you pass a text, it will guess it is a string and convert it to string.

Creating operator

Some Useful TensorFlow operators

You know how to create a tensor with TensorFlow. It is time to learn how to perform mathematical operations.

TensorFlow contains all the basic operations. You can begin with a simple one. You will use TensorFlow method to compute the square of a number. This operation is straightforward because only one argument is required to construct the tensor.

The square of a number is constructed with `tf.sqrt(x)` with `x` as a floating number.

```
x = tf.constant([2.0], dtype = tf.float32)
print(tf.sqrt(x))
```

Output

```
Tensor("Sqrt:0", shape=(1, ), dtype=float32)
```

Note: The output returned a tensor object and not the result of the square of 2. In the example, you print the definition of the tensor and not the actual evaluation of the operation. In the next section, you will learn how TensorFlow works to execute the operations.

Following is a list of commonly used operations. The idea is the same. Each operation requires one or more arguments.

- `tf.add(a, b)`
- `tf.substract(a, b)`
- `tf.multiply(a, b)`
- `tf.div(a, b)`

- `tf.pow(a, b)`
- `tf.exp(a)`
- `tf.sqrt(a)`

Example

```
# Add
tensor_a = tf.constant([[1,2]], dtype = tf.int32)
tensor_b = tf.constant([[3, 4]], dtype = tf.int32)

tensor_add = tf.add(tensor_a, tensor_b)print(tensor_add)
```

Output

```
Tensor("Add:0", shape=(1, 2), dtype=int32)
```

Code Explanation

Create two tensors:

- one tensor with 1 and 2
- one tensor with 3 and 4

You add up both tensors.

Notice: that both tensors need to have the same shape. You can execute a multiplication over the two tensors.

```
# Multiply
tensor_multiply = tf.multiply(tensor_a, tensor_b)
print(tensor_multiply)
```

Output

```
Tensor("Mul:0", shape=(1, 2), dtype=int32)
```

Variables

So far, you have only created constant tensors. It is not of great use. Data always arrive with different values, to capture this, you can use the Variable class. It will represent a node where the values always change.

To create a variable, you can use `tf.get_variable()` method

```
tf.get_variable(name = "", values, dtype, initializer)
argument
- `name = ""`: Name of the variable
- `values`: Dimension of the tensor
- `dtype`: Type of data. Optional
- `initializer`: How to initialize the tensor. Optional
If initializer is specified, there is no need to include the
`values` as the shape of `initializer` is used.
```

For instance, the code below creates a two-dimensional variable with two random values. By default, TensorFlow returns a random value. You name the variable `var`

```
# Create a Variable
## Create 2 Randomized values
var = tf.get_variable("var", [1, 2])
print(var.shape)
```

Output

```
(1, 2)
```

In the second example, you create a variable with one row and two columns. You need to use `[1,2]` to create the dimension of the variable

The initials values of this tensor are zero. For instance, when you train a model, you need to have initial values to compute the weight of the features. Below, you set these initial value to zero.

```
var_init_1 = tf.get_variable("var_init_1", [1, 2], dtype=tf.int32,
initializer=tf.zeros_initializer)
print(var_init_1.shape)
```

Output

```
(1, 2)
```

You can pass the values of a constant tensor in a variable. You create a constant tensor with the method `tf.constant()`. You use this tensor to initialize the variable.

The first values of the variable are 10, 20, 30 and 40. The new tensor will have a shape of 2x2.

```
# Create a 2x2 matrix
tensor_const = tf.constant([[10, 20],
[30, 40]])
# Initialize the first value of the tensor equals to tensor_const
var_init_2 = tf.get_variable("var_init_2", dtype=tf.int32,
initializer=tensor_const)
print(var_init_2.shape)
```

Output

```
(2, 2)
```

Placeholder

A placeholder has the purpose of feeding the tensor. Placeholder is used to initialize the data to flow inside the tensors. To supply a placeholder, you need to use the method `feed_dict`. The placeholder will be fed only within a session.

In the next example, you will see how to create a placeholder with the method `tf.placeholder`. In the next session, you will learn to feed a placeholder with actual value.

The syntax is:

```
tf.placeholder(dtype, shape=None, name=None )
arguments:
- `dtype`: Type of data
- `shape`: dimension of the placeholder. Optional. By default,
shape of the data
- `name`: Name of the placeholder. Optional
data_placeholder_a = tf.placeholder(tf.float32, name =
"data_placeholder_a")
print(data_placeholder_a)
```

Output

```
Tensor("data_placeholder_a:0", dtype=float32)
```

Session

TensorFlow works around 3 main components:

- Graph
- Tensor
- Session

Components	Description
Graph	The graph is fundamental in TensorFlow. All of the mathematical operations (ops) are performed inside a graph. You can imagine a graph as a project where every operations are done. The nodes represent these ops, they can absorb or create new tensors.
Tensor	A tensor represents the data that progress between operations. You saw previously how to initialize a tensor. The difference between a constant and variable is the initial values of a variable will change over time.
Session	A session will execute the operation from the graph. To feed the graph with the values of a tensor, you need to open a session. Inside a session, you must run an operator to create an output.

Graphs and sessions are independent. You can run a session and get the values to use later for further computations.

In the example below, you will:

- Create two tensors
- Create an operation
- Open a session

- Print the result

Step 1) You create two tensors x and y

```
## Create, run and evaluate a session
x = tf.constant([2])
y = tf.constant([4])
```

Step 2) You create the operator by multiplying x and y

```
## Create operator
multiply = tf.multiply(x, y)
```

Step 3) You open a session. All the computations will happen within the session. When you are done, you need to close the session.

```
## Create a session to run the code
sess = tf.Session()
result_1 = sess.run(multiply)
print(result_1)
sess.close()
```

Output

```
[8]
```

Code explanation

- `tf.Session()`: Open a session. All the operations will flow within the sessions
- `run(multiply)`: execute the operation created in step 2.
- `print(result_1)`: Finally, you can print the result
- `close()`: Close the session

The result shows 8, which is the multiplication of x and y.

Another way to create a session is inside a block. The advantage is it automatically closes the session.

```
with tf.Session() as sess:
```



```
result_2 = multiply.eval()
print(result_2)
```

Output

```
[8]
```

In a context of the session, you can use the `eval()` method to execute the operation. It is equivalent to `run()`. It makes the code more readable.

You can create a session and see the values inside the tensors you created so far.

```
## Check the tensors created before
sess = tf.Session()
print(sess.run(r1))
print(sess.run(r2_matrix))
print(sess.run(r3_matrix))
```

Output

```
1
[[1 2]
 [3 4]]
[[[1 2]
 [3 4]
 [5 6]]]
```

Variables are empty by default, even after you create a tensor. You need to initialize the variable if you want to use the variable. The object `tf.global_variables_initializer()` needs to be called to initialize the values of a variable. This object will explicitly initialize all the variables. This is helpful before you train a model.

You can check the values of the variables you created before. Note that you need to use `run` to evaluate the tensor

```
sess.run(tf.global_variables_initializer())
print(sess.run(var))
print(sess.run(var_init_1))
```

```
print(sess.run(var_init_2))
```

Output

```
[[ -0.05356491  0.75867283]]  
[[0 0]]  
[[10 20]]  
[[30 40]]
```

You can use the placeholder you created before and feed it with actual value. You need to pass the data into the method `feed_dict`.

For example, you will take the power of 2 of the placeholder `data_placeholder_a`.

```
import numpy as np  
power_a = tf.pow(data_placeholder_a, 2)  
with tf.Session() as sess:  
    data = np.random.rand(1, 10)  
    print(sess.run(power_a, feed_dict={data_placeholder_a: data})) #  
    Will succeed.
```

Code Explanation

- `import numpy as np`: Import numpy library to create the data
- `tf.pow(data_placeholder_a, 2)`: Create the ops
- `np.random.rand(1, 10)`: Create a random array of data
- `feed_dict={data_placeholder_a: data}`: Feed the placeholder with data

Output

```
[[0.05478134 0.27213147 0.8803037 0.0398424 0.21172127 0.01444725  
0.02584014 0.3763949 0.66022706 0.7565559 ]]
```

Graph

TensorFlow depends on a genius approach to render the operation. All the computations are represented with a dataflow scheme. The dataflow graph has been developed to see to data dependencies between individual operation. Mathematical formula or algorithm are made of a number of successive operations. A graph is a convenient way to visualize how the computations are coordinated.

The graph shows a **node** and an **edge**. The node is the representation of a operation, i.e. the unit of computation. The edge is the tensor, it can produce a new tensor or consume the input data. It depends on the dependencies between individual operation.

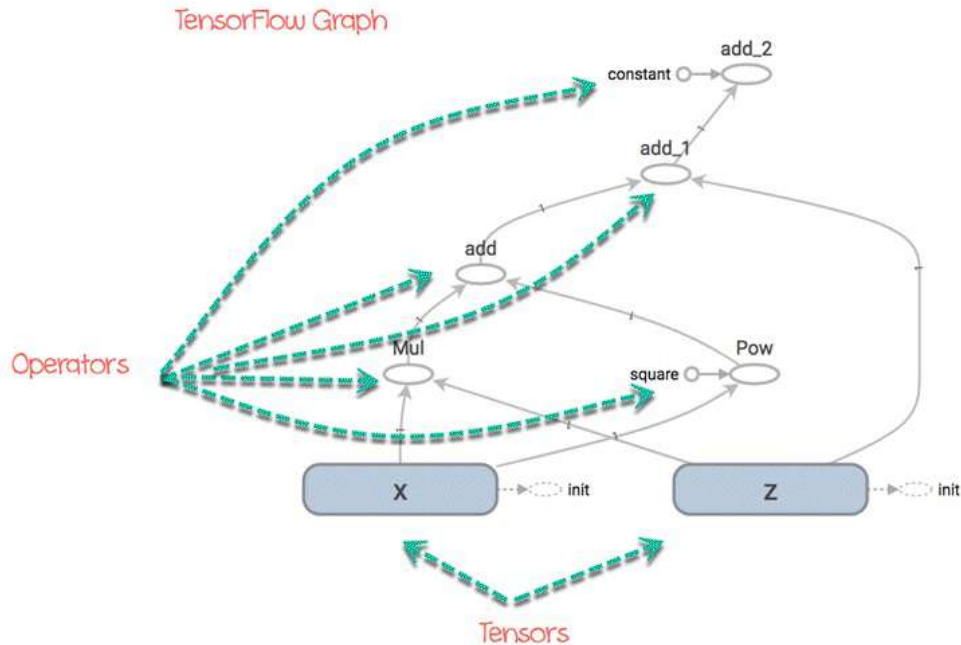
The structure of the graph connects together the operations (i.e. the nodes) and how those are operation are feed. Note that the graph does not display the output of the operations, it only helps to visualize the connection between individual operations.

Let's see an example.

Imagine you want to evaluate the following function:

$$f(x, z) = xz + x^2 + z + 5$$

TensorFlow will create a graph to execute the function. The graph looks like this:



You can easily see the path that the tensors will take to reach the final destination.

For instance, you can see the operation add cannot be done before and . The graph explains that it will:

1. compute and :
2. add 1) together
3. add to 2)
4. add 3) to

```
x = tf.get_variable("x", dtype=tf.int32,
initializer=tf.constant([5]))
z = tf.get_variable("z", dtype=tf.int32,
initializer=tf.constant([6]))
c = tf.constant([5], name = "constant")square =
tf.constant([2], name = "square")
f = tf.multiply(x, z) + tf.pow(x, square) + z + c
```

Code Explanation

- x: Initialize a variable called x with a constant value of 5

- z: Initialize a variable called z with a constant value of 6
- c: Initialize a constant tensor called c with a constant value of 5
- square: Initialize a constant tensor called square with a constant value of 2
- f: Construct the operator

In this example, we choose to keep the values of the variables fixed. We also created a constant tensor called c which is the constant parameter in the function f. It takes a fixed value of 5. In the graph, you can see this parameter in the tensor called constant.

We also constructed a constant tensor for the power in the operator `tf.pow()`. It is not necessary. We did it so that you can see the name of the tensor in the graph. It is the circle called square.

From the graph, you can understand what will happen of the tensors and how it can return an output of 66.

The code below evaluate the function in a session.

```
init = tf.global_variables_initializer() # prepare to initialize
all variables
with tf.Session() as sess:
    init.run() # Initialize x and y
    function_result = f.eval()
print(function_result)
```

Output

```
[66]
```

Summary

TensorFlow works around:

- Graph: Computational environment containing the operations and tensors
- Tensors: Represents the data (or value) that will flow in the graph. It is the edge in the graph
- Sessions: Allow the execution of the operations

Create a constant tensor

constant	object
D0	<code>tf.constant(1, tf.int16)</code>
D1	<code>tf.constant([1,3,5], tf.int16)</code>
D2	<code>tf.constant([[1, 2], [3, 4]],tf.int16)</code>
D3	<code>tf.constant([[[1, 2],[3, 4], [5, 6]]], tf.int16)</code>

Create an operator

Create an operator	Object
<code>a+b</code>	<code>tf.add(a, b)</code>
<code>a*b</code>	<code>tf.multiply(a, b)</code>

Create a variable tensor

Create a variable	object
randomized value	<code>tf.get_variable("var", [1, 2])</code>
initialized first value	<code>tf.get_variable("var_init_2", dtype=tf.int32, initializer=[[1, 2], [3, 4]])</code>

Open a session

Session	object
Create a session	<code>tf.Session()</code>
Run a session	<code>tf.Session.run()</code>
Evaluate a tensor	<code>variable_name.eval()</code>
Close a session	<code>sess.close()</code>
Session by block	<code>with tf.Session() as sess:</code>

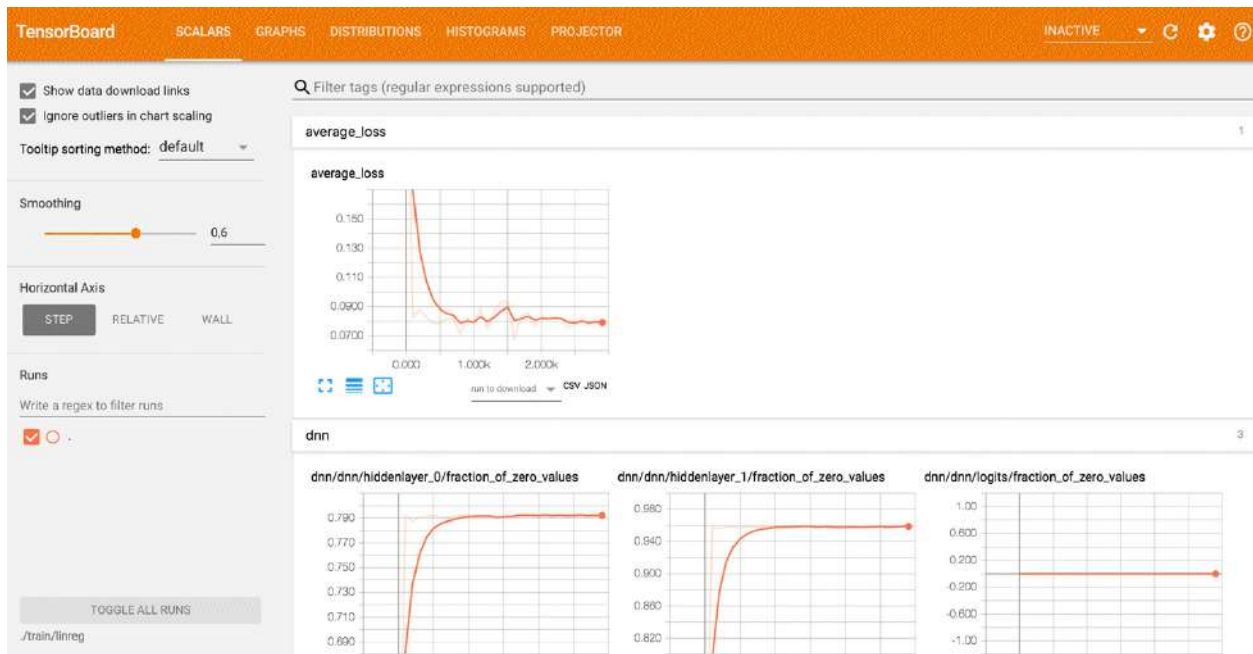
Chapter 9: Tensorboard: Graph Visualization with Example

What is TensorBoard

Tensorboard is the interface used to visualize the graph and other tools to understand, debug, and optimize the model.

Example

The image below comes from the graph you will generate in this tutorial. It is the main panel:



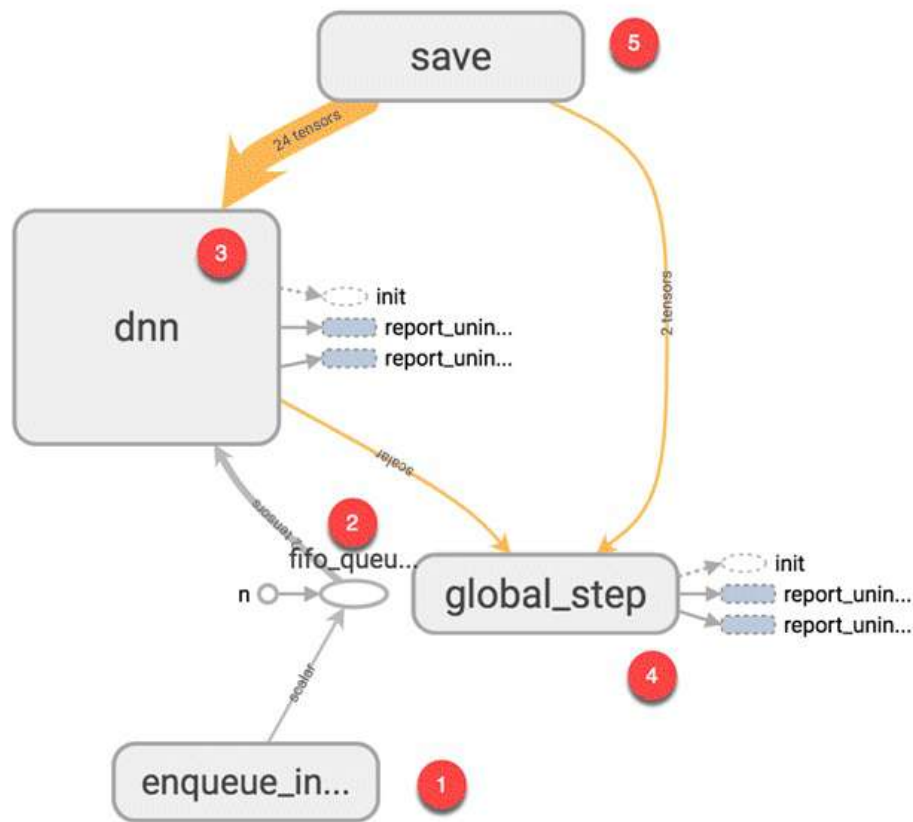
From the picture below, you can see the panel of Tensorboard. The panel contains different tabs, which are linked to the level of information you add when you run the model.

- Scalars: Show different useful information during the model training
- Graphs: Show the model
- Histogram: Display weights with a histogram
- Distribution: Display the distribution of the weight
- Projector: Show Principal component analysis and T-SNE algorithm. The technique uses for dimensionality reduction

During this tutorial, you will train a simple deep learning model. You will learn how it works in a future tutorial.

If you look at the graph, you can understand how the model work.

1. Enqueue the data to the model: Push an amount of data equal to the batch size to the model, i.e., Number of data feed after each iteration
2. Feed the data to the Tensors
3. Train the model
4. Display the number of batches during the training. Save the model on the disk.



The basic idea behind tensorboard is that neural network can be something known as a black box and we need a tool to inspect what's inside this box. You can imagine tensorboard as a flashlight to start dive into the neural network.

It helps to understand the dependencies between operations, how the weights are computed, displays the loss function and much other useful information. When you bring all these pieces of information together, you have a great tool to debug and find how to improve the model.

To give you an idea of how useful the graph can be, look at the picture below:



A neural network decides how to connect the different "neurons" and how many layers before the model can predict an outcome. Once you have defined the architecture, you not only need to train the model but also a metrics to compute the accuracy of the prediction. This metric is referred to as a **loss function**. The objective is to minimize the loss function. In different words, it means the model is making fewer errors. All machine learning algorithms will repeat many times the computations until the loss reach a flatter line. To minimize this loss function, you need to define a **learning rate**. It is the speed you want the model to learn. If you set a learning rate too high, the model does not have time to learn anything. This is the case in the left picture. The line is moving up and down, meaning the model predicts with pure guess the outcome. The picture on the right shows that the loss is decreasing over iteration until the curve got flatten, meaning the model found a solution.

TensorBoard is a great tool to visualize such metrics and highlight potential issues. The neural network can take hours to weeks before they find a solution. TensorBoard updates the metrics very often. In this case, you don't need to wait until the end to see if the model trains correctly. You can open TensorBoard check how the training is going and make the appropriate change if necessary.

In this tutorial, you will learn how to open TensorBoard from the terminal for MacOS and the Command line for Windows.

The code will be explained in a future tutorial, the focus here is on TensorBoard.

First, you need to import the libraries you will use during the training

```
## Import the library
import tensorflow as tf
import numpy as np
```

You create the data. It is an array of 10000 rows and 5 columns

```
X_train = (np.random.sample((10000,5)))
y_train = (np.random.sample((10000,1)))
X_train.shape
```

Output

```
(10000, 5)
```

The codes below transform the data and create the model.

Note that the learning rate is equal to 0.1. If you change this rate to a higher value, the model will not find a solution. This is what happened on the left side of the above picture.

During most of the TensorFlow tutorials, you will use TensorFlow estimator. This is TensorFlow API that contains all the mathematical computations.

To create the log files, you need to specify the path. This is done with the argument `model_dir`.

In the example below, you store the model inside the working directory, i.e., where you store the notebook or python file. Inside this path, TensorFlow will create a folder called `train` with a child folder name `linreg`.

```

feature_columns = [
    tf.feature_column.numeric_column('x',
shape=X_train.shape[1:])]
DNN_reg =
tf.estimator.DNNRegressor(feature_columns=feature_columns,
# Indicate where to store the log file
    model_dir='train/linreg',
    hidden_units=[500, 300],
    optimizer=tf.train.ProximalAdagradOptimizer(
        learning_rate=0.1,
        l1_regularization_strength=0.001
    )
)

```

Output

```

INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train/linreg',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1818e63828>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}

```

The last step consists to train the model. During the training, TensorFlow writes information in the model directory.

```

# Train the estimator
train_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train, shuffle=False,num_epochs=None)
DNN_reg.train(train_input,steps=3000)

```

Output

```

INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.

```

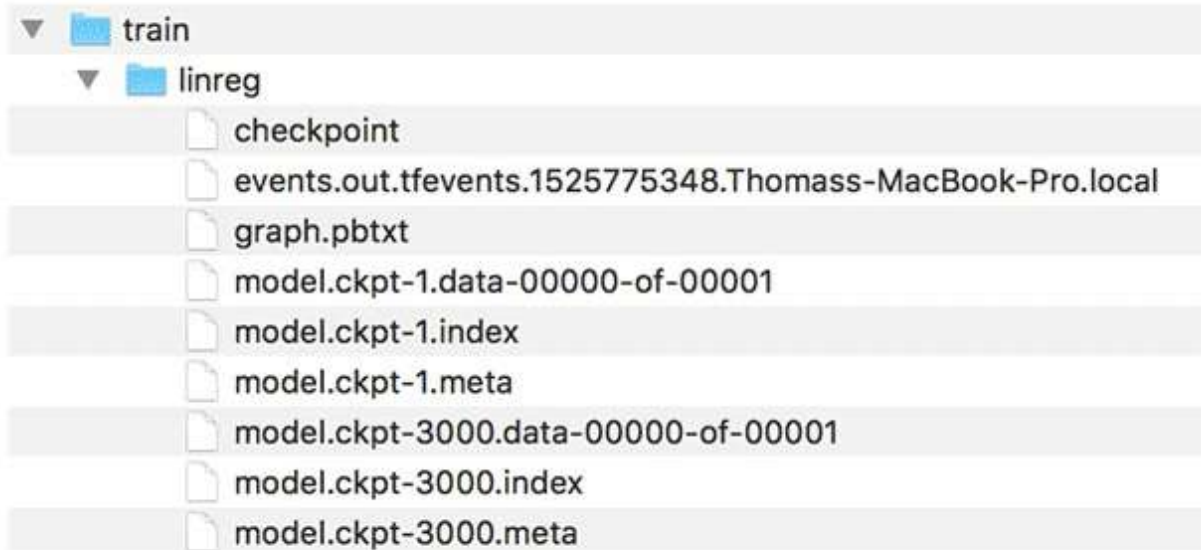
```
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into
train/linreg/model.ckpt.
INFO:tensorflow:loss = 40.060104, step = 1
INFO:tensorflow:global_step/sec: 197.061
INFO:tensorflow:loss = 10.62989, step = 101 (0.508 sec)
INFO:tensorflow:global_step/sec: 172.487
INFO:tensorflow:loss = 11.255318, step = 201 (0.584 sec)
INFO:tensorflow:global_step/sec: 193.295
INFO:tensorflow:loss = 10.604872, step = 301 (0.513 sec)
INFO:tensorflow:global_step/sec: 175.378
INFO:tensorflow:loss = 10.090343, step = 401 (0.572 sec)
INFO:tensorflow:global_step/sec: 209.737
INFO:tensorflow:loss = 10.057928, step = 501 (0.476 sec)
INFO:tensorflow:global_step/sec: 171.646
INFO:tensorflow:loss = 10.460144, step = 601 (0.583 sec)
INFO:tensorflow:global_step/sec: 192.269
INFO:tensorflow:loss = 10.529617, step = 701 (0.519 sec)
INFO:tensorflow:global_step/sec: 198.264
INFO:tensorflow:loss = 9.100082, step = 801 (0.504 sec)
INFO:tensorflow:global_step/sec: 226.842
INFO:tensorflow:loss = 10.485607, step = 901 (0.441 sec)
INFO:tensorflow:global_step/sec: 152.929
INFO:tensorflow:loss = 10.052481, step = 1001 (0.655 sec)
INFO:tensorflow:global_step/sec: 166.745
INFO:tensorflow:loss = 11.320213, step = 1101 (0.600 sec)
INFO:tensorflow:global_step/sec: 161.854
INFO:tensorflow:loss = 9.603306, step = 1201 (0.619 sec)
INFO:tensorflow:global_step/sec: 179.074
INFO:tensorflow:loss = 11.110269, step = 1301 (0.556 sec)
INFO:tensorflow:global_step/sec: 202.776
INFO:tensorflow:loss = 11.929443, step = 1401 (0.494 sec)
INFO:tensorflow:global_step/sec: 144.161
INFO:tensorflow:loss = 11.951693, step = 1501 (0.694 sec)
INFO:tensorflow:global_step/sec: 154.144
INFO:tensorflow:loss = 8.620987, step = 1601 (0.649 sec)
INFO:tensorflow:global_step/sec: 151.094
INFO:tensorflow:loss = 10.666125, step = 1701 (0.663 sec)
INFO:tensorflow:global_step/sec: 193.644
INFO:tensorflow:loss = 11.0349865, step = 1801 (0.516 sec)
INFO:tensorflow:global_step/sec: 189.707
INFO:tensorflow:loss = 9.860596, step = 1901 (0.526 sec)
INFO:tensorflow:global_step/sec: 176.423
```

```
INFO:tensorflow:loss = 10.695, step = 2001 (0.567 sec)
INFO:tensorflow:global_step/sec: 213.066
INFO:tensorflow:loss = 10.426752, step = 2101 (0.471 sec)
INFO:tensorflow:global_step/sec: 220.975
INFO:tensorflow:loss = 10.594796, step = 2201 (0.452 sec)
INFO:tensorflow:global_step/sec: 219.289
INFO:tensorflow:loss = 10.4212265, step = 2301 (0.456 sec)
INFO:tensorflow:global_step/sec: 215.123
INFO:tensorflow:loss = 9.668612, step = 2401 (0.465 sec)
INFO:tensorflow:global_step/sec: 175.65
INFO:tensorflow:loss = 10.009649, step = 2501 (0.569 sec)
INFO:tensorflow:global_step/sec: 206.962
INFO:tensorflow:loss = 10.477722, step = 2601 (0.483 sec)
INFO:tensorflow:global_step/sec: 229.627
INFO:tensorflow:loss = 9.877638, step = 2701 (0.435 sec)
INFO:tensorflow:global_step/sec: 195.792
INFO:tensorflow:loss = 10.274586, step = 2801 (0.512 sec)
INFO:tensorflow:global_step/sec: 176.803
INFO:tensorflow:loss = 10.061047, step = 2901 (0.566 sec)
INFO:tensorflow:Saving checkpoints for 3000 into
train/linreg/model.ckpt.
INFO:tensorflow:Loss for final step: 10.73032.

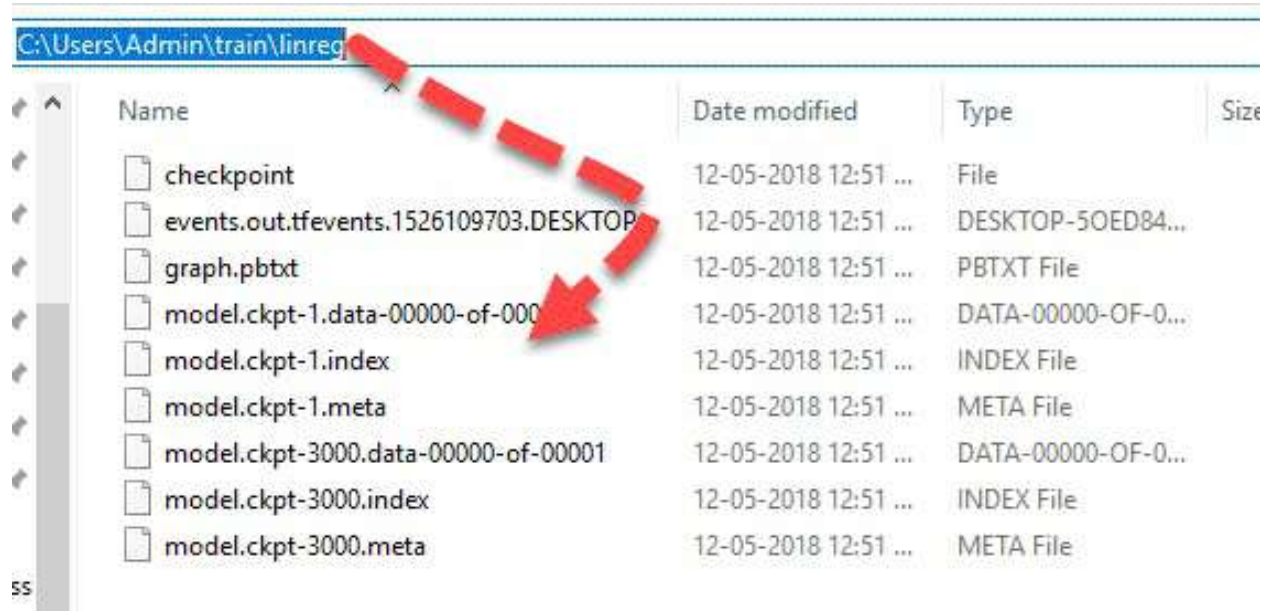
<tensorflow.python.estimator.canned.dnn.DNNRegressor at
0x1818e63630>
```

For MacOS user

New folder inside the working directory



For Windows user



You can see this information in the TensorBoard.

Now that you have the log events written, you can open Tensorboard. Tensorboard runs on port 6006 (Jupyter runs on port 8888). You can use the Terminal for MacOs user or Anaconda prompt for Windows user.

For MacOS user

```
# Different for you
cd /Users/Guru99/tuto_TF
source activate hello-tf!
```

The notebook is stored in the path /Users/Guru99/tuto_TF

For Windows users

```
cd C:\Users\Admin\Anaconda3
activate hello-tf
```

The notebook is stored in the path C:\Users\Admin\Anaconda3

To launch Tensorboard, you can use this code

For MacOS user

```
tensorboard --logdir=./train/linreg
```

For Windows users

```
tensorboard --logdir=.\train\linreg
```

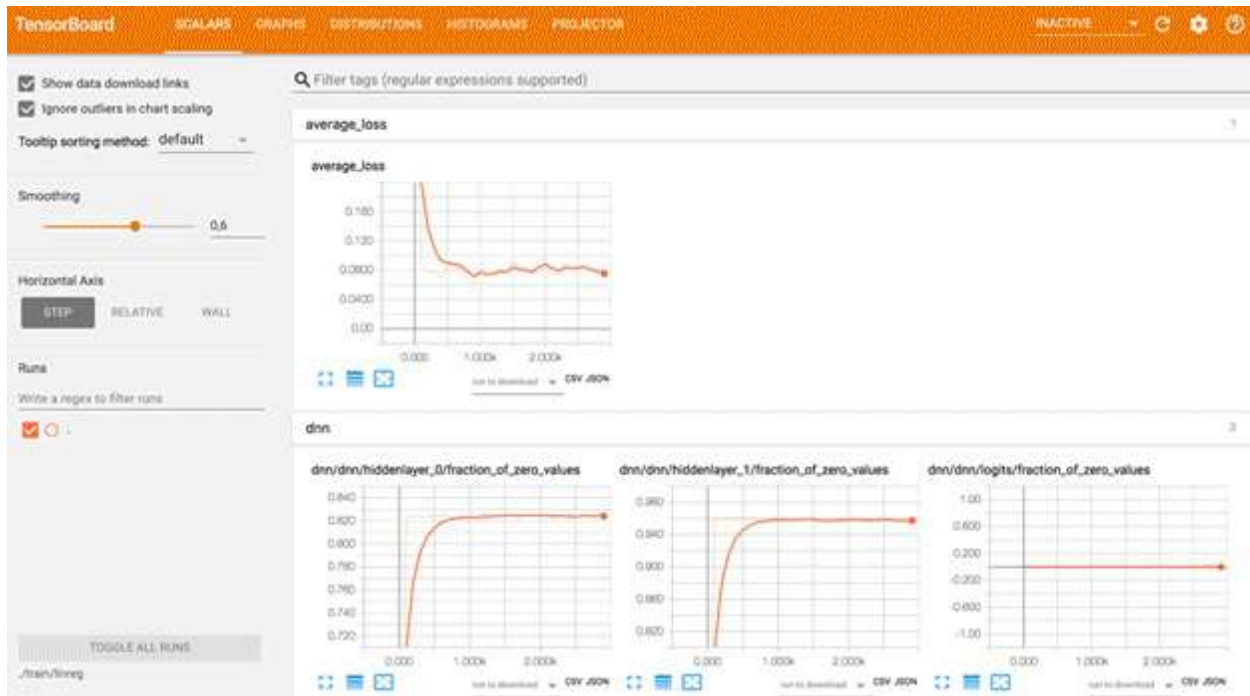
Tensorboard is located in this URL: <http://localhost:6006>

It could also be located at the following location.

```
(hello-tf) C:\Users\Admin>tensorboard --logdir=.\train\linreg
2018-05-13 19:08:34.355370: I T:\src\github\tensorflow\tensorflow\core\platform\cpu_feature_guard.cc:140] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
W0513 19:08:34.371735 Reloader tf_logging.py:121] Found more than one graph event per run, or there was a metagraph containing a graph_def, as well as one or more graph events. Overwriting the graph with the newest event.
W0513 19:08:34.373239 Reloader tf_logging.py:121] Found more than one metagraph event per run. Overwriting the metagraph with the newest event.
TensorBoard 1.8.0 at http://DESKTOP-50ED84V:6006 (Press CTRL+C to quit)
```

Copy and paste the URL into your favorite browser. You should see this:

Note that, we will learn how to read the graph in the tutorial dedicated to the deep learning.



If you see something like this:

No graph definition files were found.

To store a graph, create a `tf.summary.FileWriter` and pass the graph either via the constructor, or by calling its `add_graph()` method. You may want to check out the [graph visualizer tutorial](#).

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the [README](#) and perhaps the [TensorBoard tutorial](#).

If you think TensorBoard is configured properly, please see [the section of the README devoted to missing data problems](#) and consider filing an issue on GitHub.

It means Tensorboard cannot find the log file. Make sure you point the cd to the right path or double check if the log event has been creating. If not, re-run the code.

If you want to close TensorBoard Press CTRL+C

Hat Tip: Check your anaconda prompt for the current working directory,



```
Anaconda Prompt
(hello-tf) C:\Users\Admin>
```

The log file should be created at C:\Users\Admin

Summary:

TensorBoard is a great tool to visualize your model. Besides, many metrics are displayed during the training, such as the loss, accuracy or weights.

To activate Tensorboard, you need to set the path of your file:

```
cd /Users/Guru99/tuto_TF
```

Activate Tensorflow's environment

```
activate hello-tf
```

Launch Tensorboard

```
tensorboard --logdir=.+ PATH
```

Chapter 10: NumPy

What is NumPy?

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication

For any scientific project, NumPy is the tool to know. It has been built to work with the N-dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated to C/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top of the arrays and matrices, NumPy supports a large number of mathematical operations. In this part, we will review the essential functions that you need to know for the tutorial on 'TensorFlow.'

Why use NumPy?

NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.

How to install NumPy?

To install Pandas library, please refer our tutorial [How to install TensorFlow](#). NumPy is installed by default. In remote case, NumPy not installed-

You can install NumPy using:

- Anaconda: `conda install -c anaconda numpy`
- In Jupyter Notebook :

```
import sys
!conda install --yes --prefix {sys.prefix} numpy
```

Import NumPy and Check Version

The command to import numpy is

```
import numpy as np
```

Above code renames the Numpy namespace to np. This permits us to prefix Numpy function, methods, and attributes with " np " instead of typing " numpy." It is the standard shortcut you will find in the numpy literature

To check your installed version of Numpy use the command

```
print (np.__version__)
```

Output

```
1.14.0
```

Create a NumPy Array

Simplest way to create an array in Numpy is to use Python List

```
myPythonList = [1,9,8,3]
```

To convert python list to a numpy array by using the object np.array.

```
numpy_array_from_list = np.array(myPythonList)
```

To display the contents of the list

```
numpy_array_from_list
```

Output

```
array([1, 9, 8, 3])
```

In practice, there is no need to declare a Python List. The operation can be combined.

```
a = np.array([1,9,8,3])
```

NOTE: Numpy documentation states use of np.ndarray to create an array. However, this is the recommended method

You can also create a numpy array from a Tuple

Mathematical Operations on an Array

You could perform mathematical operations like additions, subtraction, division and multiplication on an array. The syntax is the array name followed by the operation (+,-,*,/) followed by the operand

Example:

```
numpy_array_from_list + 10
```

Output:

```
array([11, 19, 18, 13])
```

This operation adds 10 to each element of the numpy array.

Shape of Array

You can check the shape of the array with the object shape preceded by the name of the array. In the same way, you can check the type with dtypes.

```
import numpy as np
a = np.array([1,2,3])
print(a.shape)
print(a.dtype)

(3,)
int64
```

An integer is a value without decimal. If you create an array with decimal, then the type will change to float.

```
#### Different type
b = np.array([1.1,2.0,3.2])
print(b.dtype)

float64
```

2 Dimension Array

You can add a dimension with a ","coma

Note that it has to be within the bracket []

```
### 2 dimension
c = np.array([(1,2,3),
              (4,5,6)])
print(c.shape)
(2, 3)
```

3 Dimension Array

Higher dimension can be constructed as follow:

```
### 3 dimension
d = np.array([
    [1, 2, 3],
    [4, 5, 6]],
    [[7, 8, 9],
    [10, 11, 12]]
])
print(d.shape)
(2, 2, 3)
```

np.zeros and np.ones

You can create matrix full of zeroes or ones. It can be used when you initialized the weights during the first iteration in TensorFlow.

The syntax is

```
numpy.zeros(shape, dtype=float, order='C')
```

```
numpy.ones(shape, dtype=float, order='C')
```

Here,

Shape: is the shape of the array

Dtype: is the datatype. It is optional. The default value is float64

Order: Default is C which is an essential row style.

Example:

```
np.zeros((2,2))
```

Output:

```
array([[0., 0.],  
       [0., 0.]])
```

```
np.zeros((2,2), dtype=np.int16)
```

Output:

```
array([[0, 0],  
       [0, 0]], dtype=int16)
```

```
## Create 1  
np.ones((1,2,3), dtype=np.int16)  
array([[[1, 1, 1],
```

```
[1, 1, 1]], dtype=int16)
```

Reshape and Flatten Data

In some occasion, you need to reshape the data from wide to long.

```
e = np.array([(1,2,3), (4,5,6)])  
print(e)  
e.reshape(3,2)
```

Output:

```
[[1 2 3]  
 [4 5 6]]
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

When you deal with some neural network like convnet, you need to flatten the array. You can use `flatten()`

```
e.flatten()
```

```
array([1, 2, 3, 4, 5, 6])
```

hstack and vstack

Numpy library has also two convenient function to horizontally or vertically append the data. Lets study them with an example:

```
## Stack
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))

Horizontal Append: [1 2 3 4 5 6]
Vertical Append: [[1 2 3]
 [4 5 6]]
```

Generate Random Numbers

To generate random numbers for Gaussian distribution use

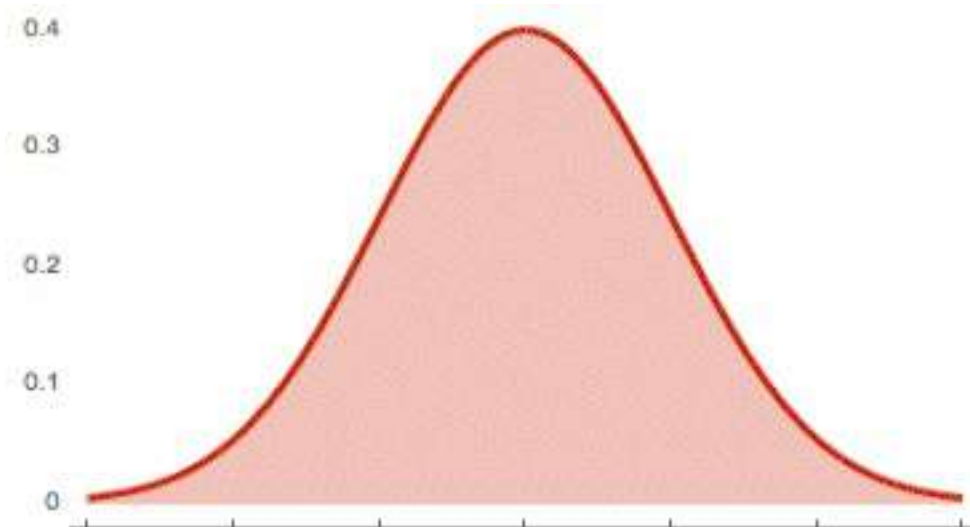
`numpy.random.normal(loc, scale, size)`

Here

- Loc: the mean. The center of distribution
- scale: standard deviation.
- Size: number of returns

```
## Generate random number from normal distribution
normal_array = np.random.normal(5, 0.5, 10)
print(normal_array)
[5.56171852 4.84233558 4.65392767 4.946659    4.85165567 5.61211317
 4.46704244 5.22675736 4.49888936 4.68731125]
```

If plotted the distribution will be similar to following plot



Asarray

Consider the following 2-D matrix with four rows and four columns filled by 1

```
A = np.matrix(np.ones((4,4)))
```

If you want to change the value of the matrix, you cannot. The reason is, it is not possible to change a copy.

```
np.array(A)[2]=2
print(A)
[[1.  1.  1.  1.]
 [1.  1.  1.  1.]
 [1.  1.  1.  1.]
 [1.  1.  1.  1.]]
```

Matrix is immutable. You can use asarray if you want to add modification in the original array. let's see if any change occurs when you want to change the value of the third rows with the value 2

```
np.asarray(A)[2]=2
print(A)
```

Code Explanation:

`np.asarray(A)`: converts the matrix A to an array

`[2]`: select the third rows

```
[[1.  1.  1.  1.]
 [1.  1.  1.  1.]
 [2.  2.  2.  2.] # new value
 [1.  1.  1.  1.]]
```

Arrange

In some occasion, you want to create value evenly spaced within a given interval. For instance, you want to create values from 1 to 10; you can use `arrange`

Syntax:

```
numpy.arange(start, stop, step)
```

- Start: Start of interval
- Stop: End of interval
- Step: Spacing between values. Default step is 1

Example:

```
np.arange(1, 11)
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

If you want to change the step, you can add a third number in the parenthesis. It will change the step.

```
np.arange(1, 14, 4)
```

```
array([ 1,  5,  9, 13])
```

Linspace

Linspace gives evenly spaced samples.

Syntax:

```
numpy.linspace(start, stop, num, endpoint)
```

Here,

- Start: Starting value of the sequence
- Stop: End value of the sequence
- Num: Number of samples to generate. Default is 50
- Endpoint: If True (default), stop is the last value. If False, stop value is not included.

For instance, it can be used to create 10 values from 1 to 5 evenly spaced.

```
np.linspace(1.0, 5.0, num=10)
array([1.          , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.          ])
```

If you do not want to include the last digit in the interval, you can set endpoint to false

```
np.linspace(1.0, 5.0, num=5, endpoint=False)
```

```
array([1. , 1.8, 2.6, 3.4, 4.2])
```

LogSpace

LogSpace returns even spaced numbers on a log scale. Logspace has the same parameters as np.linspace.

```
np.logspace(3.0, 4.0, num=4)
array([ 1000. ,  2154.43469003,  4641.58883361, 10000.      ])
```

Finally, if you want to check the size of an array, you can use `itemsize`

```
x = np.array([1,2,3], dtype=np.complex128)
x.itemsize
```

16

The `x` element has 16 bytes.

Indexing and slicing

Slicing data is trivial with numpy. We will slice the matrix `e`. Note that, in Python, you need to use the brackets to return the rows or columns

```
## Slice
e = np.array([(1,2,3), (4,5,6)])
print(e)
[[1 2 3]
 [4 5 6]]
```

Remember with numpy the first array/column starts at 0.

```
## First column
print('First row:', e[0])

## Second col
print('Second row:', e[1])
First row: [1 2 3]
Second row: [4 5 6]
```

In Python, like many other languages,

- The values before the comma stand for the rows
- The value on the right stands for the columns.
- If you want to select a column, you need to add `:` before the column index.

- : means you want all the rows from the selected column.

```
print('Second column:', e[:,1])
```

```
Second column: [2 5]
```

To return the first two values of the second row. You use : to select all columns up to the second

```
##
print(e[1, :2])
[4 5]
```

Statistical function

Numpy is equipped with the robust statistical function as listed below

Function	Numpy
Min	np.min()
Max	np.max()
Mean	np.mean()
Median	np.median()
Standard deviation	np.std()

```
## Statistical function
### Min
print(np.min(normal_array))

### Max
print(np.max(normal_array))
### Mean
```

```
print(np.mean(normal_array))
### Median
print(np.median(normal_array))
### Sd
print(np.std(normal_array))
```

```
4.467042435266913
5.612113171990201
4.934841002270593
4.846995625786663
0.3875019367395316
```

Dot Product

Numpy is powerful library for matrices computation. For instance, you can compute the dot product with `np.dot`

```
## Linear algebra
### Dot product: product of two arrays
f = np.array([1,2])
g = np.array([4,5])
### 1*4+2*5
np.dot(f, g)
```

14

Matrix Multiplication

In the same way, you can compute matrices multiplication with `np.matmul`

```
### Matmul: matruc product of two arrays
h = [[1,2],[3,4]]
i = [[5,6],[7,8]]
### 1*5+2*7 = 19
np.matmul(h, i)
```

```
array([[19, 22],
       [43, 50]])
```

Determinant

Last but not least, if you need to compute the determinant, you can use `np.linalg.det()`. Note that numpy takes care of the dimension.

```
## Determinant 2*2 matrix  
### 5*8-7*6np.linalg.det(i)
```

```
-2.0000000000000005
```

Summary

Below, a summary of the essential functions used with NumPy

Objective	Code
Create array	<code>array([1,2,3])</code>
print the shape	<code>array([.]).shape</code>
reshape	<code>reshape</code>
flat an array	<code>flatten</code>
append vertically	<code>vstack</code>
append horizontally	<code>hstack</code>
create a matrix	<code>matrix</code>
create space	<code>arange</code>
Create a linear space	<code>linspace</code>
Create a log space	<code>logspace</code>

Below is a summary of basic statistical and arithmetical function

Objective	Code

min	min()
max	max()
mean	mean()
median	median()
standard deviation	std()

Here is the complete code:

```
import numpy as np

##Create array
### list
myPythonList = [1,9,8,3]
numpy_array_from_list = np.array(myPythonList)
### Directly in numpy
np.array([1,9,8,3])

### Shape
a = np.array([1,2,3])
print(a.shape)

### Type
print(a.dtype)

### 2D array
c = np.array([(1,2,3),
(4,5,6)])
print("2d Array",c)

### 3D array
d = np.array([
[[1, 2,3],
[4, 5, 6]],
[[7, 8,9],
[10, 11, 12]]
])
```

```
print("3d Array",d)
### Reshape
e = np.array([(1,2,3), (4,5,6)])
print(e)
e.reshape(3,2)
print("After Reshape",e)
### Flatten
e.flatten()

print("After Flatten",e)

### hstack & vstack
f = np.array([1,2,3])
g = np.array([4,5,6])

print('Horizontal Append:', np.hstack((f, g)))
print('Vertical Append:', np.vstack((f, g)))

### random number
normal_array = np.random.normal(5, 0.5, 10)
print("Random Number",normal_array)

### asarray
A = np.matrix(np.ones((4,4)))
np.asarray(A)
print("Asrray",A)
### Arrange

print("Arrange",np.arange(1, 11))

### linspace

lin = np.linspace(1.0, 5.0, num=10)
print("Linspace",lin)

### logspace
log1 = np.logspace(3.0, 4.0, num=4)
print("Logspace",log1)

### Slicing
#### rows
e = np.array([(1,2,3), (4,5,6)])
```

```
print(e[0])

#### columns
print(e[:,1])

#### rows and columns
print(e[1, :2])
```

Chapter 11: Pandas

What is Pandas?

Pandas is an open-source library that allows you to perform data manipulation in Python. Pandas library is built on top of Numpy, meaning Pandas needs Numpy to operate. Pandas provide an easy way to create, manipulate and wrangle the data. Pandas is also an elegant solution for time series data.

Why use Pandas?

Data scientists use Pandas for its following advantages:

- Easily handles missing data
- It uses **Series for one-dimensional data structure** and **DataFrame for multi-dimensional data structure**
- It provides an efficient way to slice the data
- It provides a flexible way to merge, concatenate or reshape the data
- It includes a powerful time series tool to work with

In a nutshell, Pandas is a useful library in data analysis. It can be used to perform data manipulation and analysis. Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

How to install Pandas?

To install Pandas library, please refer our tutorial [How to install TensorFlow](#). Pandas is installed by default. In remote case, pandas not installed-

You can install Pandas using:

- Anaconda: `conda install -c anaconda pandas`
- In Jupyter Notebook :

```
import sys
!conda install --yes --prefix {sys.prefix} pandas
```

What is a data frame?

A data frame is a two-dimensional array, with labeled axes (rows and columns).

A data frame is a standard way to store data.

Data frame is well-known by statistician and other data practitioners. A data frame is a tabular data, with rows to store the information and columns to name the information. For instance, the price can be the name of a column and 2,3,4 the price values.

Below a picture of a Pandas data frame:

	Item	Price
0	A	2
1	B	3

What is a Series?

A series is a one-dimensional data structure. It can have any data structure like integer, float, and string. It is useful when you want to perform computation or return a one-dimensional array. A series, by definition, cannot have multiple columns. For the latter case, please use the data frame structure.

Series has one parameters:

- Data: can be a list, dictionary or scalar value

```
pd.Series([1., 2., 3.])
```

```
0    1.0
1    2.0
2    3.0
dtype: float64
```

You can add the index with `index`. It helps to name the rows. The length should be equal to the size of the column

```
pd.Series([1., 2., 3.], index=['a', 'b', 'c'])
```

Below, you create a Pandas series with a missing value for the third rows. Note, missing values in Python are noted "NaN." You can use numpy to create missing value: `np.nan` artificially

```
pd.Series([1, 2, np.nan])
```

Output

```
0    1.0
1    2.0
2    NaN
dtype: float64
```


Create Data frame

You can convert a numpy array to a pandas data frame with `pd.DataFrame()`. The opposite is also possible. To convert a pandas Data Frame to an array, you can use `np.array()`

```
## Numpy to pandas
import numpy as np
h = [[1,2],[3,4]]
df_h = pd.DataFrame(h)
print('Data Frame:', df_h)

## Pandas to numpy
df_h_n = np.array(df_h)
print('Numpy array:', df_h_n)
Data Frame:    0  1
0  1  2
1  3  4
Numpy array: [[1 2]
 [3 4]]
```

You can also use a dictionary to create a Pandas dataframe.

```
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}
pd.DataFrame(data=dic)
```

	Age	Name
0	30	John
1	40	Smith

Range Data

Pandas have a convenient API to create a range of date

`pd.date_range(date,period,frequency):`

- The first parameter is the starting date

- The second parameter is the number of periods (optional if the end date is specified)
- The last parameter is the frequency: day: 'D,' month: 'M' and year: 'Y.'

```
## Create date
# Days
dates_d = pd.date_range('20300101', periods=6, freq='D')
print('Day:', dates_d)
```

Output

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03',
'2030-01-04', '2030-01-05', '2030-01-06'], dtype='datetime64[ns]',
freq='D')
```

```
# Months
dates_m = pd.date_range('20300101', periods=6, freq='M')
print('Month:', dates_m)
```

Output

```
Month: DatetimeIndex(['2030-01-31', '2030-02-28', '2030-03-31',
'2030-04-30', '2030-05-31', '2030-06-30'], dtype='datetime64[ns]',
freq='M')
```

Inspecting data

You can check the head or tail of the dataset with `head()`, or `tail()` preceded by the name of the panda's data frame

Step 1) Create a random sequence with numpy. The sequence has 4 columns and 6 rows

```
random = np.random.randn(6, 4)
```

Step 2) Then you create a data frame using pandas.

Use `dates_m` as an index for the data frame. It means each row will be given a

"name" or an index, corresponding to a date.

Finally, you give a name to the 4 columns with the argument columns

```
# Create data with date
df = pd.DataFrame(random,
                  index=dates_m,
                  columns=list('ABCD'))
```

Step 3) Using head function

```
df.head(3)
```

	A	B	C	D
2030-01-31	1.139433	1.318510	-0.181334	1.615822
2030-02-28	-0.081995	-0.063582	0.857751	-0.527374
2030-03-31	-0.519179	0.080984	-1.454334	1.314947

Step 4) Using tail function

```
df.tail(3)
```

	A	B	C	D
2030-04-30	-0.685448	-0.011736	0.622172	0.104993
2030-05-31	-0.935888	-0.731787	-0.558729	0.768774
2030-06-30	1.096981	0.949180	-0.196901	-0.471556

Step 5) An excellent practice to get a clue about the data is to use describe(). It provides the counts, mean, std, min, max and percentile of the dataset.

```
df.describe()
```

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	0.002317	0.256928	-0.151896	0.467601
std	0.908145	0.746939	0.834664	0.908910
min	-0.935888	-0.731787	-1.454334	-0.527374
25%	-0.643880	-0.050621	-0.468272	-0.327419
50%	-0.300587	0.034624	-0.189118	0.436883

75%	0.802237	0.732131	0.421296	1.178404
max	1.139433	1.318510	0.857751	1.615822

Slice data

The last point of this tutorial is about how to slice a pandas data frame.

You can use the column name to extract data in a particular column.

```
## Slice
### Using name
df['A']

2030-01-31    -0.168655
2030-02-28     0.689585
2030-03-31     0.767534
2030-04-30     0.557299
2030-05-31    -1.547836
2030-06-30     0.511551
Freq: M, Name: A, dtype: float64
```

To select multiple columns, you need to use two times the bracket, [[...]]

The first pair of bracket means you want to select columns, the second pairs of bracket tells what columns you want to return.

```
df[['A', 'B']].
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

You can slice the rows with :

The code below returns the first three rows

```
### using a slice for row  
df[0:3]
```

	A	B	C	D
2030-01-31	-0.168655	0.587590	0.572301	-0.031827
2030-02-28	0.689585	0.998266	1.164690	0.475975
2030-03-31	0.767534	-0.940617	0.227255	-0.341532

The loc function is used to select columns by names. As usual, the values before the coma stand for the rows and after refer to the column. You need to use the brackets to select more than one column.

```
## Multi col  
df.loc[:, ['A', 'B']]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

There is another method to select multiple rows and columns in Pandas. You can use iloc[]. This method uses the index instead of the columns name. The code below returns the same data frame as above

```
df.iloc[:, :2]
```

	A	B
2030-01-31	-0.168655	0.587590
2030-02-28	0.689585	0.998266
2030-03-31	0.767534	-0.940617
2030-04-30	0.557299	0.507350
2030-05-31	-1.547836	1.276558
2030-06-30	0.511551	1.572085

Drop a column

You can drop columns using `pd.drop()`

```
df.drop(columns=['A', 'C'])
```

	B	D
2030-01-31	0.587590	-0.031827
2030-02-28	0.998266	0.475975
2030-03-31	-0.940617	-0.341532
2030-04-30	0.507350	-0.296035
2030-05-31	1.276558	0.523017
2030-06-30	1.572085	-0.594772

Concatenation

You can concatenate two DataFrame in Pandas. You can use `pd.concat()`

First of all, you need to create two DataFrames. So far so good, you are already familiar with dataframe creation

```
import numpy as np
df1 = pd.DataFrame({'name': ['John', 'Smith', 'Paul'],
                    'Age': ['25', '30', '50']},
                   index=[0, 1, 2])
df2 = pd.DataFrame({'name': ['Adam', 'Smith'],
                    'Age': ['26', '11']},
                   index=[3, 4])
```

Finally, you concatenate the two DataFrame

```
df_concat = pd.concat([df1, df2])
df_concat
```

	Age	name
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Drop_duplicates

If a dataset can contain duplicates information use, `drop_duplicates` is an easy to exclude duplicate rows. You can see that `df_concat` has a duplicate observation, `Smith` appears twice in the column `name`.

```
df_concat.drop_duplicates('name')
```

	Age	name
--	-----	------

0	25	John
1	30	Smith
2	50	Paul
3	26	Adam

Sort values

You can sort value with `sort_values`

```
df_concat.sort_values('Age')
```

	Age	name
4	11	Smith
0	25	John
3	26	Adam
1	30	Smith
2	50	Paul

Rename: change of index

You can use `rename` to rename a column in Pandas. The first value is the current column name and the second value is the new column name.

```
df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
```

	Age_ppl	Surname
0	25	John
1	30	Smith
2	50	Paul
3	26	Adam
4	11	Smith

Import CSV

During the TensorFlow tutorial, you will use the adult dataset. It is often used

with classification task. It is available in this URL

<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data> The data is stored in a CSV format. This dataset includes eight categorical variables:

This dataset includes eight categorical variables:

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss

hours_week

To import a CSV dataset, you can use the object `pd.read_csv()`. The basic argument inside is:

Syntax:

```
pandas.read_csv(filepath_or_buffer, sep=',',  
' , `names=None`, `index_col=None`, `skipinitialspace=False` )
```

- `filepath_or_buffer`: Path or URL with the data
- `sep=', '`: Define the delimiter to use

- ``names=None``: Name the columns. If the dataset has ten columns, you need to pass ten names
- ``index_col=None``: If yes, the first column is used as a row index
- ``skipinitialspace=False``: Skip spaces after delimiter.

For more information about `readcsv()`, please check the official documentation

https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html.

Consider the following Example

```
## Import csv
import pandas as pd
## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
            'education_num', 'marital',
            'occupation', 'relationship', 'race', 'sex',
            'capital_gain', 'capital_loss',
            'hours_week', 'native_country', 'label']
PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
df_train = pd.read_csv(PATH,
                       skipinitialspace=True,
                       names = COLUMNS,
                       index_col=False)
df_train.shape
```

Output:

```
(32561, 15)
```

Groupby

An easy way to see the data is to use the `groupby` method. This method can help you to summarize the data by group. Below is a list of methods available with `groupby`:

- `count`: count

- min: min
- max: max
- mean: mean
- median: median
- standard deviation: sdt
- etc

Inside `groupby()`, you can use the column you want to apply the method.

Let's have a look at a single grouping with the adult dataset. You will get the mean of all the continuous variables by type of revenue, i.e., above 50k or below 50k

```
df_train.groupby(['label']).mean()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_week
label						
<=50K	36.783738	190340.86517	9.595065	148.752468	53.142921	38.840210
>50K	44.249841	188005.00000	11.611657	4006.142456	195.001530	45.473026

You can get the minimum of age by type of household

```
df_train.groupby(['label'])['age'].min()
```

```
label
<=50K    17
>50K     19
Name: age, dtype: int64
```

You can also group by multiple columns. For instance, you can get the maximum capital gain according to the household type and marital status.

```
df_train.groupby(['label', 'marital'])['capital_gain'].max()
label marital
<=50K Divorced          34095
      Married-AF-spouse    2653
      Married-civ-spouse  41310
      Married-spouse-absent 6849
```

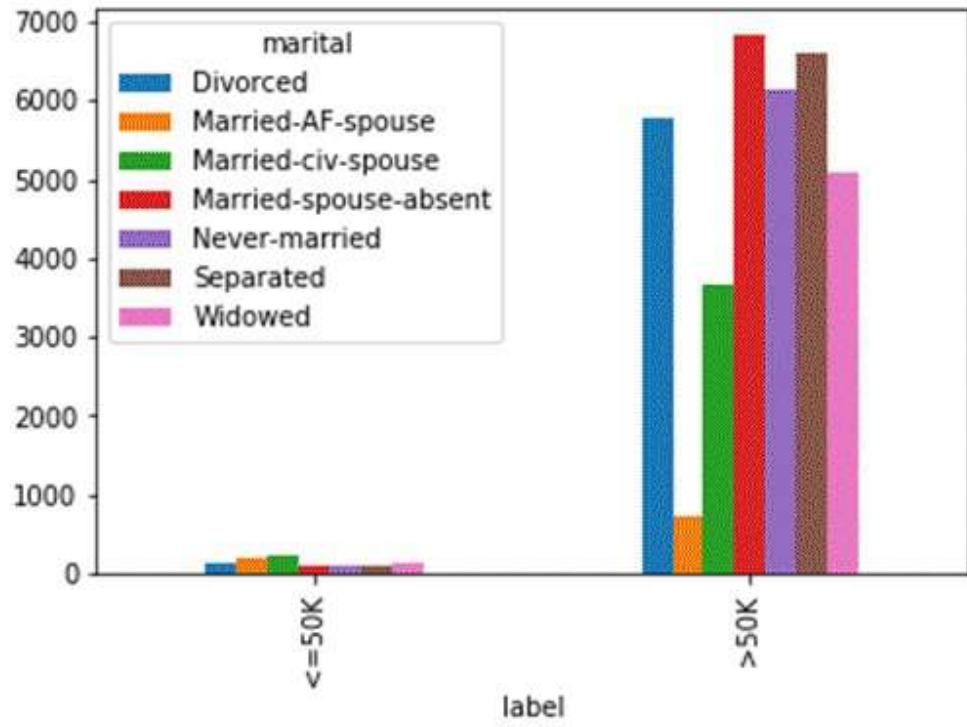
```
Never-married      34095
Separated          7443
Widowed            6849
>50K Divorced      99999
Married-AF-spouse  7298
Married-civ-spouse 99999
Married-spouse-absent 99999
Never-married      99999
Separated          99999
Widowed            99999
Name: capital_gain, dtype: int64
```

You can create a plot following groupby. One way to do it is to use a plot after the grouping.

To create a more excellent plot, you will use unstack() after mean() so that you have the same multilevel index, or you join the values by revenue lower than 50k and above 50k. In this case, the plot will have two groups instead of 14 (2*7).

If you use Jupyter Notebook, make sure to add % matplotlib inline, otherwise, no plot will be displayed

```
% matplotlib inline
df_plot = df_train.groupby(['label', 'marital'])
['capital_gain'].mean().unstack()
df_plot
```



Summary

Below is a summary of the most useful method for data science with Pandas

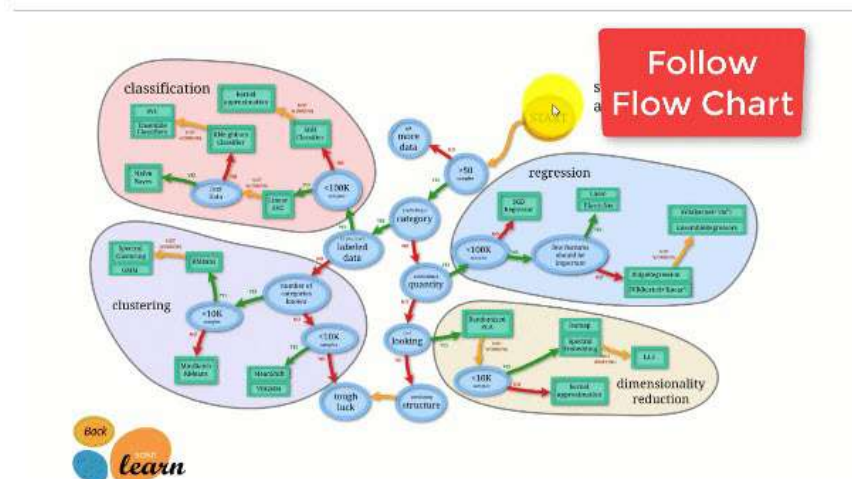
import data	read_csv
create series	Series
Create Dataframe	DataFrame
Create date range	date_range
return head	head
return tail	tail
Describe	describe
slice using name	dataname['columnname']
Slice using rows	data_name[0:5]

Chapter 12: Scikit-Learn

What is Scikit-learn?

Scikit-learn is an open source Python library for machine learning. The library supports state-of-the-art algorithms such as KNN, XGBoost, random forest, SVM among others. It is built on top of Numpy. Scikit-learn is widely used in kaggle competition as well as prominent tech companies. Scikit-learn helps in preprocessing, dimensionality reduction(parameter selection), classification, regression, clustering, and model selection.

Scikit-learn has the best documentation of all opensource libraries. It provides you an interactive chart at http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.



Scikit-learn is not very difficult to use and provides excellent results. However, scikit learn does not support parallel computations. It is possible to run a deep learning algorithm with it but is not an optimal solution, especially if you know how to use TensorFlow.

Download and Install scikit-learn

Option 1: AWS

scikit-learn can be used over AWS. Please refer The docker image has scikit-learn preinstalled.

To use developer version use the command in Jupyter

```
import sys
!{sys.executable} -m pip install git+git://github.com/scikit-learn/scikit-learn.git
```

Option 2: Mac or Windows using Anaconda

To learn about Anaconda installation refer <https://www.guru99.com/download-install-tensorflow.html>

Recently, the developers of scikit have released a development version that tackles common problem faced with the current version. We found it more convenient to use the developer version instead of the current version.

If you installed scikit-learn with the conda environment, please follow the step to update to version 0.20

Step 1) Activate tensorflow environment

```
source activate hello-tf
```

Step 2) Remove scikit learn using the conda command

```
conda remove scikit-learn
```

Step 3) Install scikit learn developer version along with necessary libraries.


```
conda install -c anaconda git
pip install Cython
pip install h5py
pip install git+git://github.com/scikit-learn/scikit-learn.git
```

NOTE: Windows users will need to install Microsoft Visual C++ 14. You can get it from [here](#)

Machine learning with scikit-learn

This tutorial is divided into two parts:

1. Machine learning with scikit-learn
2. How to trust your model with LIME

The first part details how to build a pipeline, create a model and tune the hyperparameters while the second part provides state-of-the-art in term of model selection.

Step 1) Import the data

During this tutorial, you will be using the adult dataset. For a background in this dataset refer [If you are interested to know more about the descriptive statistics](#), please use Dive and Overview tools. Refer [this tutorial](#) learn more about Dive and Overview

You import the dataset with Pandas. Note that you need to convert the type of the continuous variables in float format.

This dataset includes eight categorical variables:

The categorical variables are listed in CATE_FEATURES

- workclass
- education
- marital
- occupation
- relationship
- race
- sex
- native_country

moreover, six continuous variables:

The continuous variables are listed in CONTI_FEATURES

- age
- fnlwgt
- education_num
- capital_gain
- capital_loss

- hours_week

Note that we fill the list by hand so that you have a better idea of what columns we are using. A faster way to construct a list of categorical or continuous is to use:

```
## List Categorical
CATE_FEATURES =
df_train.iloc[:, :-1].select_dtypes('object').columns
print(CATE_FEATURES)

## List continuous
CONTI_FEATURES = df_train._get_numeric_data()
print(CONTI_FEATURES)
```

Here is the code to import the data:

```
# Import dataset
import pandas as pd

## Define path data
COLUMNS = ['age', 'workclass', 'fnlwgt', 'education',
            'education_num', 'marital',
            'occupation', 'relationship', 'race', 'sex',
            'capital_gain', 'capital_loss',
            'hours_week', 'native_country', 'label']
### Define continuous list
CONTI_FEATURES = ['age', 'fnlwgt', 'capital_gain', 'education_num',
                 'capital_loss', 'hours_week']
### Define categorical list
CATE_FEATURES = ['workclass', 'education', 'marital', 'occupation',
                 'relationship', 'race', 'sex', 'native_country']

## Prepare the data
features = ['age', 'workclass', 'fnlwgt', 'education',
            'education_num', 'marital',
            'occupation', 'relationship', 'race', 'sex',
            'capital_gain', 'capital_loss',
            'hours_week', 'native_country']

PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
```

```
df_train = pd.read_csv(PATH, skipinitialspace=True, names =
COLUMNS, index_col=False)
df_train[CONTI_FEATURES]
=df_train[CONTI_FEATURES].astype('float64')
df_train.describe()
```

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

You can check the count of unique values of the native_country features. You can see that only one household comes from Holand-Netherlands. This household won't bring us any information, but will through an error during the training.

```
df_train.native_country.value_counts()
```

```
United-States          29170
Mexico                  643
?                       583
Philippines            198
Germany                137
Canada                 121
Puerto-Rico           114
El-Salvador           106
India                  100
Cuba                   95
England                90
Jamaica                81
South                  80
China                  75
Italy                  73
Dominican-Republic    70
Vietnam                67
```

```
Guatemala 64
Japan 62
Poland 60
Columbia 59
Taiwan 51
Haiti 44
Iran 43
Portugal 37
Nicaragua 34
Peru 31
France 29
Greece 29
Ecuador 28
Ireland 24
Hong 20
Cambodia 19
Trinidad&Tobago 19
Thailand 18
Laos 18
Yugoslavia 16
Outlying-US(Guam-USVI-etc) 14
Honduras 13
Hungary 13
Scotland 12
Holand-Netherlands 1
Name: native_country, dtype: int64
```

You can exclude this uninformative row from the dataset

```
## Drop Netherland, because only one row
df_train = df_train[df_train.native_country != "Holand-
Netherlands"]
```

Next, you store the position of the continuous features in a list. You will need it in the next step to build the pipeline.

The code below will loop over all columns names in CONTI_FEATURES and get its location (i.e., its number) and then append it to a list called conti_features

```
## Get the column index of the categorical features
conti_features = []
for i in CONTI_FEATURES:
```

```
position = df_train.columns.get_loc(i)
conti_features.append(position)
print(conti_features)
```

```
[0, 2, 10, 4, 11, 12]
```

The code below does the same job as above but for the categorical variable. The code below repeats what you have done previously, except with the categorical features.

```
## Get the column index of the categorical features
categorical_features = []
for i in CATE_FEATURES:
    position = df_train.columns.get_loc(i)
    categorical_features.append(position)
print(categorical_features)
```

```
[1, 3, 5, 6, 7, 8, 9, 13]
```

You can have a look at the dataset. Note that, each categorical feature is a string. You cannot feed a model with a string value. You need to transform the dataset using a dummy variable.

```
df_train.head(5)
```

In fact, you need to create one column for each group in the feature. First, you can run the code below to compute the total amount of columns needed.

```
print(df_train[CATE_FEATURES].nunique(),
      'There are', sum(df_train[CATE_FEATURES].nunique()), 'groups
in the whole dataset')
```

```
workclass      9
education      16
marital         7
occupation     15
relationship    6
race            5
sex             2
native_country  41
dtype: int64 There are 101 groups in the whole dataset
```

The whole dataset contains 101 groups as shown above. For instance, the features of workclass have nine groups. You can visualize the name of the groups with the following codes

`unique()` returns the unique values of the categorical features.

```
for i in CATE_FEATURES:
    print(df_train[i].unique())

['State-gov' 'Self-emp-not-inc' 'Private' 'Federal-gov' 'Local-gov'
'?'
 'Self-emp-inc' 'Without-pay' 'Never-worked']
['Bachelors' 'HS-grad' '11th' 'Masters' '9th' 'Some-college'
'Assoc-acdm'
 'Assoc-voc' '7th-8th' 'Doctorate' 'Prof-school' '5th-6th' '10th'
 '1st-4th' 'Preschool' '12th']
['Never-married' 'Married-civ-spouse' 'Divorced' 'Married-spouse-
absent'
 'Separated' 'Married-AF-spouse' 'Widowed']
['Adm-clerical' 'Exec-managerial' 'Handlers-cleaners' 'Prof-
specialty'
 'Other-service' 'Sales' 'Craft-repair' 'Transport-moving'
 'Farming-fishing' 'Machine-op-inspct' 'Tech-support' '?'
 'Protective-serv' 'Armed-Forces' 'Priv-house-serv']
['Not-in-family' 'Husband' 'Wife' 'Own-child' 'Unmarried' 'Other-
relative']
['White' 'Black' 'Asian-Pac-Islander' 'Amer-Indian-Eskimo' 'Other']
['Male' 'Female']
['United-States' 'Cuba' 'Jamaica' 'India' '?' 'Mexico' 'South'
 'Puerto-Rico' 'Honduras' 'England' 'Canada' 'Germany' 'Iran'
 'Philippines' 'Italy' 'Poland' 'Columbia' 'Cambodia' 'Thailand'
'Ecuador'
 'Laos' 'Taiwan' 'Haiti' 'Portugal' 'Dominican-Republic' 'El-
Salvador'
 'France' 'Guatemala' 'China' 'Japan' 'Yugoslavia' 'Peru'
 'Outlying-US(Guam-USVI-etc)' 'Scotland' 'Trinidad&Tobago' 'Greece'
 'Nicaragua' 'Vietnam' 'Hong' 'Ireland' 'Hungary']
```

Therefore, the training dataset will contain 101 + 7 columns. The last seven columns are the continuous features.

Scikit-learn can take care of the conversion. It is done in two steps:

- First, you need to convert the string to ID. For instance, State-gov will have the ID 1, Self-emp-not-inc ID 2 and so on. The function LabelEncoder does this for you
- Transpose each ID into a new column. As mentioned before, the dataset has 101 group's ID. Therefore there will be 101 columns capturing all categorical features' groups. Scikit-learn has a function called OneHotEncoder that performs this operation

Step 2) Create the train/test set

Now that the dataset is ready, we can split it 80/20. 80 percent for the training set and 20 percent for the test set.

You can use `train_test_split`. The first argument is the dataframe is the features and the second argument is the label dataframe. You can specify the size of the test set with `test_size`.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(df_train[features],
                 df_train.label,
                 test_size =
0.2,
                 random_state=0)
X_train.head(5)
print(X_train.shape, X_test.shape)
```

```
(26048, 14) (6512, 14)
```

Step 3) Build the pipeline

The pipeline makes it easier to feed the model with consistent data. The idea behind is to put the raw data into a 'pipeline' to perform operations. For instance, with the current dataset, you need to standardize the continuous variables and convert the categorical data. Note that you can perform any operation inside the pipeline. For instance, if you have 'NA's' in the dataset, you can replace them by the mean or median. You can also create new variables.

You have the choice; hard code the two processes or create a pipeline. The first choice can lead to data leakage and create inconsistencies over time. A better option is to use the pipeline.

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder,
LabelEncoder
from sklearn.compose import ColumnTransformer,
make_column_transformer
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
```

The pipeline will perform two operations before feeding the logistic classifier:

1. Standardize the variable: `StandardScaler()``
2. Convert the categorical features: `OneHotEncoder(sparse=False)`

You can perform the two steps using the `make_column_transformer`. This function is not available in the current version of scikit-learn (0.19). It is not possible with the current version to perform the label encoder and one hot encoder in the pipeline. It is one reason we decided to use the developer version.

`make_column_transformer` is easy to use. You need to define which columns to apply the transformation and what transformation to operate. For instance, to standardize the continuous feature, you can do:

- `conti_features`, `StandardScaler()` inside `make_column_transformer`.
 - `conti_features`: list with the continuous variable
 - `StandardScaler`: standardize the variable

The object `OneHotEncoder` inside `make_column_transformer` automatically encodes the label.

```
preprocess = make_column_transformer(  
    (conti_features, StandardScaler()),  
    ### Need to be numeric not string to specify columns name  
    (categorical_features, OneHotEncoder(sparse=False))  
)
```

You can test if the pipeline works with `fit_transform`. The dataset should have the following shape: `(26048, 107)`

```
preprocess.fit_transform(X_train).shape
```

```
(26048, 107)
```

The data transformer is ready to use. You can create the pipeline with `make_pipeline`. Once the data are transformed, you can feed the logistic regression.

```
model = make_pipeline(  
    preprocess,  
    LogisticRegression())
```

Training a model with scikit-learn is trivial. You need to use the object `fit` preceded by the pipeline, i.e., `model`. You can print the accuracy with the `score` object from the scikit-learn library

```
model.fit(X_train, y_train)  
print("logistic regression score: %f" % model.score(X_test,  
y_test))
```

```
logistic regression score: 0.850891
```

Finally, you can predict the classes with `predict_proba`. It returns the probability

for each class. Note that it sums to one.

```
model.predict_proba(X_test)
```

```
array([[0.83576663, 0.16423337],  
       [0.94582765, 0.05417235],  
       [0.64760587, 0.35239413],  
       ...,  
       [0.99639252, 0.00360748],  
       [0.02072181, 0.97927819],  
       [0.56781353, 0.43218647]])
```

Step 4) Using our pipeline in a grid search

Tune the hyperparameter (variables that determine network structure like hidden units) can be tedious and exhausting. One way to evaluate the model could be to change the size of the training set and evaluate the performances. You can repeat this method ten times to see the score metrics. However, it is too much work.

Instead, scikit-learn provides a function to carry out parameter tuning and cross-validation.

Cross-validation

Cross-Validation means during the training, the training set is split n number of times in folds and then evaluates the model n time. For instance, if cv is set to 10, the training set is trained and evaluates ten times. At each round, the classifier chooses randomly nine fold to train the model, and the 10th fold is meant for evaluation.

Grid search Each classifier has hyperparameters to tune. You can try different values, or you can set a parameter grid. If you go to the scikit-learn official website, you can see the logistic classifier has different parameters to tune. To make the training faster, you choose to tune the C parameter. It controls for the regularization parameter. It should be positive. A small value gives more weight to the regularizer.

You can use the object GridSearchCV. You need to create a dictionary containing the hyperparameters to tune.

You list the hyperparameters followed by the values you want to try. For instance, to tune the C parameter, you use:

- 'logisticregression__C': [0.1, 1.0, 1.0]: The parameter is preceded by the

name, in lower case, of the classifier and two underscores.

The model will try four different values: 0.001, 0.01, 0.1 and 1.

You train the model using 10 folds: cv=10

```
from sklearn.model_selection import GridSearchCV
# Construct the parameter grid
param_grid = {
    'logisticregression__C': [0.001, 0.01, 0.1, 1.0],
}
```

You can train the model using GridSearchCV with the parameter grid and cv.

```
# Train the model
grid_clf = GridSearchCV(model,
                        param_grid,
                        cv=10,
                        iid=False)
grid_clf.fit(X_train, y_train)
```

OUTPUT

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                               steps=[('columntransformer', ColumnTransformer(n_jobs=1,
                                     remainder='drop', transformer_weights=None,
                                     transformers=[('standardscaler', StandardScaler(copy=True,
                                               with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12])),
                                             ('onehotencoder', OneHotEncoder(categorical_features=None,
                                               categories=None, ...ty='l2', random_state=None, solver='liblinear',
                                               tol=0.0001,
                                               verbose=0, warm_start=False))])),
             fit_params=None, iid=False, n_jobs=1,
             param_grid={'logisticregression__C': [0.001, 0.01, 0.1,
1.0]},
             pre_dispatch='2*n_jobs', refit=True,
             return_train_score='warn',
             scoring=None, verbose=0)
```

To access the best parameters, you use best_params_

```
grid_clf.best_params_
```

OUTPUT

```
{'logisticregression__C': 1.0}
```

After trained the model with four different regularization values, the optimal parameter is

```
print("best logistic regression from grid search: %f" %  
grid_clf.best_estimator_.score(X_test, y_test))
```

best logistic regression from grid search: 0.850891

To access the predicted probabilities:

```
grid_clf.best_estimator_.predict_proba(X_test)
```

```
array([[0.83576677, 0.16423323],  
       [0.9458291 , 0.0541709 ],  
       [0.64760416, 0.35239584],  
       ...,  
       [0.99639224, 0.00360776],  
       [0.02072033, 0.97927967],  
       [0.56782222, 0.43217778]])
```


XGBoost Model with scikit-learn

Let's try to train one of the best classifiers on the market. XGBoost is an improvement over the random forest. The theoretical background of the classifier is out of the scope of this tutorial. Keep in mind that, XGBoost has won lots of kaggle competitions. With an average dataset size, it can perform as good as a deep learning algorithm or even better.

The classifier is challenging to train because it has a high number of parameters to tune. You can, of course, use GridSearchCV to choose the parameter for you.

Instead, let's see how to use a better way to find the optimal parameters. GridSearchCV can be tedious and very long to train if you pass many values. The search space grows along with the number of parameters. A preferable solution is to use RandomizedSearchCV. This method consists of choosing the values of each hyperparameter after each iteration randomly. For instance, if the classifier is trained over 1000 iterations, then 1000 combinations are evaluated. It works more or less like. GridSearchCV

You need to import xgboost. If the library is not installed, please use pip3 install xgboost or

```
use import sys
!{sys.executable} -m pip install xgboost
```

In Jupyter environment

Next,

```
import xgboost
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import StratifiedKFold
```

The next step includes specifying the parameters to tune. You can refer to the

official documentation to see all the parameters to tune. For the sake of the tutorial, you only choose two hyperparameters with two values each. XGBoost takes lots of time to train, the more hyperparameters in the grid, the longer time you need to wait.

```
params = {
    'xgbclassifier__gamma': [0.5, 1],
    'xgbclassifier__max_depth': [3, 4]
}
```

You construct a new pipeline with XGBoost classifier. You choose to define 600 estimators. Note that `n_estimators` are a parameter that you can tune. A high value can lead to overfitting. You can try by yourself different values but be aware it can takes hours. You use the default value for the other parameters

```
model_xgb = make_pipeline(
    preprocess,
    xgboost.XGBClassifier(
        n_estimators=600,
        objective='binary:logistic',
        silent=True,
        nthread=1)
)
```

You can improve the cross-validation with the Stratified K-Folds cross-validator. You construct only three folds here to faster the computation but lowering the quality. Increase this value to 5 or 10 at home to improve the results.

You choose to train the model over four iterations.

```
skf = StratifiedKFold(n_splits=3,
                      shuffle = True,
                      random_state = 1001)

random_search = RandomizedSearchCV(model_xgb,
                                   param_distributions=params,
                                   n_iter=4,
                                   scoring='accuracy',
                                   n_jobs=4,
```

```
cv=skf.split(X_train, y_train),
verbose=3,
random_state=1001)
```

The randomized search is ready to use, you can train the model

```
#grid_xgb = GridSearchCV(model_xgb, params, cv=10, iid=False)
random_search.fit(X_train, y_train)
```

Fitting 3 folds for each of 4 candidates, totalling 12 fits

```
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8759645283888057, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8729701715996775, total= 1.0min
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=0.5,
score=0.8706519235199263, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8735460094437406, total= 1.3min
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8722791661868018, total= 57.7s
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8753886905447426, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
```

```
score=0.8697304768486523, total= 1.3min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=0.5,
score=0.8740066797189912, total= 1.4min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1
.....
[CV] xgbclassifier__max_depth=3, xgbclassifier__gamma=1,
score=0.8707671043538355, total= 1.0min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8729701715996775, total= 1.2min
[Parallel(n_jobs=4)]: Done 10 out of 12 | elapsed: 3.6min
remaining: 43.5s
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8736611770125533, total= 1.2min
[CV] xgbclassifier__max_depth=4, xgbclassifier__gamma=1,
score=0.8692697535130154, total= 1.2min
```

```
[Parallel(n_jobs=4)]: Done 12 out of 12 | elapsed: 3.6min
finished
/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-
packages/sklearn/model_selection/_search.py:737:
DeprecationWarning: The default of the `iid` parameter will change
from True to False in version 0.22 and will be removed in 0.24.
This will change numeric results when test-set sizes are unequal.
DeprecationWarning)
```

```
RandomizedSearchCV(cv=<generator object _BaseKFold.split at
0x1101eb830>,
                  error_score='raise-deprecating',
                  estimator=Pipeline(memory=None,
                  steps=[('columntransformer', ColumnTransformer(n_jobs=1,
remainder='drop', transformer_weights=None,
transformers=[('standardscaler', StandardScaler(copy=True,
with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12])),
('onehotencoder', OneHotEncoder(categorical_features=None,
categories=None,...
reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
silent=True, subsample=1))]),
                  fit_params=None, iid='warn', n_iter=4, n_jobs=4,
                  param_distributions={'xgbclassifier__gamma': [0.5, 1],
'xgbclassifier__max_depth': [3, 4]},
                  pre_dispatch='2*n_jobs', random_state=1001, refit=True,
                  return_train_score='warn', scoring='accuracy', verbose=3)
```

As you can see, XGBoost has a better score than the previous logistic regression.

```
print("Best parameter", random_search.best_params_)  
print("best logistic regression from grid search: %f" %  
random_search.best_estimator_.score(X_test, y_test))
```

```
Best parameter {'xgbclassifier__max_depth': 3,  
'xgbclassifier__gamma': 0.5}  
best logistic regression from grid search: 0.873157
```

```
random_search.best_estimator_.predict(X_test)
```

```
array(['<=50K', '<=50K', '<=50K', ..., '<=50K', '>50K', '<=50K'],  
      dtype=object)
```

Create DNN with MLPClassifier in scikit-learn

Finally, you can train a deep learning algorithm with scikit-learn. The method is the same as the other classifier. The classifier is available at MLPClassifier.

```
from sklearn.neural_network import MLPClassifier
```

You define the following deep learning algorithm:

- Adam solver
- Relu activation function
- Alpha = 0.0001
- batch size of 150
- Two hidden layers with 100 and 50 neurons respectively

```
model_dnn = make_pipeline(  
    preprocess,  
    MLPClassifier(solver='adam',  
                  alpha=0.0001,  
                  activation='relu',  
                  batch_size=150,  
                  hidden_layer_sizes=(200, 100),  
                  random_state=1))
```

You can change the number of layers to improve the model

```
model_dnn.fit(X_train, y_train)  
print("DNN regression score: %f" % model_dnn.score(X_test,  
y_test))
```

DNN regression score: 0.821253

LIME: Trust your Model

Now that you have a good model, you need a tool to trust it. Machine learning algorithm, especially random forest and neural network, are known to be black-box algorithm. Say differently, it works but no one knows why.

Three researchers have come up with a great tool to see how the computer makes a prediction. The paper is called Why Should I Trust You?

They developed an algorithm named **Local Interpretable Model-Agnostic Explanations (LIME)**.

Take an example:

sometimes you do not know if you can trust a machine-learning prediction:

A doctor, for example, cannot trust a diagnosis just because a computer said so. You also need to know if you can trust the model before putting it into production.

Imagine we can understand why any classifier is making a prediction even incredibly complicated models such as neural networks, random forests or svms with any kernel

will become more accessible to trust a prediction if we can understand the reasons behind it. From the example with the doctor, if the model told him which symptoms are essential you would trust it, it is also easier to figure out if you should not trust the model.

Lime can tell you what features affect the decisions of the classifier

Data Preparation

They are a couple of things you need to change to run LIME with python. First of all, you need to install lime in the terminal. You can use pip install lime

Lime makes use of LimeTabularExplainer object to approximate the model locally. This object requires:

- a dataset in numpy format
- The name of the features: feature_names
- The name of the classes: class_names
- The index of the column of the categorical features: categorical_features
- The name of the group for each categorical features: categorical_names

Create numpy train set

You can copy and convert df_train from pandas to numpy very easily

```
df_train.head(5)
# Create numpy data
df_lime = df_train
df_lime.head(3)
```

Get the class name The label is accessible with the object unique(). You should see:

- '<=50K'
- '>50K'

```
# Get the class name
class_names = df_lime.label.unique()
class_names
```

```
array(['<=50K', '>50K'], dtype=object)
```

index of the column of the categorical features

You can use the method you lean before to get the name of the group. You

encode the label with LabelEncoder. You repeat the operation on all the categorical features.

```
##
import sklearn.preprocessing as preprocessing
categorical_names = {}
for feature in CATE_FEATURES:
    le = preprocessing.LabelEncoder()
    le.fit(df_lime[feature])
    df_lime[feature] = le.transform(df_lime[feature])
    categorical_names[feature] = le.classes_
print(categorical_names)
```

```
{'workclass': array(['?', 'Federal-gov', 'Local-gov', 'Never-
worked', 'Private',
    'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-
pay'],
    dtype=object), 'education': array(['10th', '11th', '12th',
'1st-4th', '5th-6th', '7th-8th', '9th',
    'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-
grad',
    'Masters', 'Preschool', 'Prof-school', 'Some-college'],
    dtype=object), 'marital': array(['Divorced', 'Married-AF-
spouse', 'Married-civ-spouse',
    'Married-spouse-absent', 'Never-married', 'Separated',
'Widowed'],
    dtype=object), 'occupation': array(['?', 'Adm-clerical',
'Armed-Forces', 'Craft-repair',
    'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',
'Machine-op-inspct', 'Other-service', 'Priv-house-serv',
    'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-
support',
    'Transport-moving'], dtype=object), 'relationship':
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',
    'Unmarried', 'Wife'], dtype=object), 'race': array(['Amer-
Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',
    'White'], dtype=object), 'sex': array(['Female', 'Male'],
dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',
'China', 'Columbia', 'Cuba',
    'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',
'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',
'Honduras',
    'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',
'Jamaica',
```

```

    'Japan', 'Laos', 'Mexico', 'Nicaragua',
    'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',
    'Poland',
    'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',
    'Thailand', 'Trinidad&Tobago', 'United-States', 'Vietnam',
    'Yugoslavia'], dtype=object)}

df_lime.dtypes

```

```

age                float64
workclass          int64
fnlwgt            float64
education          int64
education_num     float64
marital           int64
occupation        int64
relationship      int64
race              int64
sex              int64
capital_gain     float64
capital_loss     float64
hours_week       float64
native_country   int64
label            object
dtype: object

```

Now that the dataset is ready, you can construct the different dataset. You actually transform the data outside of the pipeline in order to avoid errors with LIME. The training set in the LimeTabularExplainer should be a numpy array without string. With the method above, you have a training dataset already converted.

```

from sklearn.model_selection import train_test_split
X_train_lime, X_test_lime, y_train_lime, y_test_lime =
train_test_split(df_lime[features],
                df_lime.label,
                test_size =
0.2,
                random_state=0)
X_train_lime.head(5)

```

You can make the pipeline with the optimal parameters from XGBoost

```

model_xgb = make_pipeline(
    preprocess,
    xgboost.XGBClassifier(max_depth = 3,
                          gamma = 0.5,
                          n_estimators=600,
                          objective='binary:logistic',
                          silent=True,
                          nthread=1))

model_xgb.fit(X_train_lime, y_train_lime)

```

```

/Users/Thomas/anaconda3/envs/hello-tf/lib/python3.6/site-
packages/sklearn/preprocessing/_encoders.py:351: FutureWarning: The
handling of integer data will change in version 0.22. Currently,
the categories are determined based on the range [0, max(values)],
while in the future they will be determined based on the unique
values.

```

If you want the future behavior and silence this warning, you can specify "categories='auto'." In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```

Pipeline(memory=None,
         steps=[('columntransformer', ColumnTransformer(n_jobs=1,
                                                         remainder='drop', transformer_weights=None,
                                                         transformers=[('standardscaler', StandardScaler(copy=True,
                                                                 with_mean=True, with_std=True), [0, 2, 10, 4, 11, 12])),
                                                         ('onehotencoder', OneHotEncoder(categorical_features=None,
                                                                 categories=None,...
                                                                 reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                                                                 silent=True, subsample=1)))]))

```

You get a warning. The warning explains that you do not need to create a label encoder before the pipeline. If you do not want to use LIME, you are fine to use the method from the first part of the tutorial. Otherwise, you can keep with this method, first create an encoded dataset, set get the hot one encoder within the pipeline.

```

print("best logistic regression from grid search: %f" %
      model_xgb.score(X_test_lime, y_test_lime))

```

```
best logistic regression from grid search: 0.873157
```

```
model_xgb.predict_proba(X_test_lime)
```

```
array([[7.9646105e-01, 2.0353897e-01],  
       [9.5173013e-01, 4.8269872e-02],  
       [7.9344827e-01, 2.0655173e-01],  
       ...,  
       [9.9031430e-01, 9.6856682e-03],  
       [6.4581633e-04, 9.9935418e-01],  
       [9.7104281e-01, 2.8957171e-02]], dtype=float32)
```

Before to use LIME in action, let's create a numpy array with the features of the wrong classification. You can use that list later to get an idea about what misled the classifier.

```
temp = pd.concat([X_test_lime, y_test_lime], axis= 1)  
temp['predicted'] = model_xgb.predict(X_test_lime)  
temp['wrong']= temp['label'] != temp['predicted']  
temp = temp.query('wrong==True').drop('wrong', axis=1)  
temp= temp.sort_values(by=['label'])  
temp.shape
```

(826, 16)

You create a lambda function to retrieve the prediction from the model with the new data. You will need it soon.

```
predict_fn = lambda x: model_xgb.predict_proba(x).astype(float)  
X_test_lime.dtypes
```

```
age                float64  
workclass          int64  
fnlwgt            float64  
education          int64  
education_num     float64  
marital           int64  
occupation        int64  
relationship      int64  
race              int64  
sex               int64  
capital_gain      float64  
capital_loss      float64  
hours_week        float64
```

```
native_country      int64
dtype: object
```

```
predict_fn(X_test_lime)
```

```
array([[7.96461046e-01, 2.03538969e-01],
       [9.51730132e-01, 4.82698716e-02],
       [7.93448269e-01, 2.06551731e-01],
       ...,
       [9.90314305e-01, 9.68566816e-03],
       [6.45816326e-04, 9.99354184e-01],
       [9.71042812e-01, 2.89571714e-02]])
```

You convert the pandas dataframe to numpy array

```
X_train_lime = X_train_lime.values
X_test_lime = X_test_lime.values
X_test_lime
```

```
array([[4.00000e+01, 5.00000e+00, 1.93524e+05, ..., 0.00000e+00,
        4.00000e+01, 3.80000e+01],
       [2.70000e+01, 4.00000e+00, 2.16481e+05, ..., 0.00000e+00,
        4.00000e+01, 3.80000e+01],
       [2.50000e+01, 4.00000e+00, 2.56263e+05, ..., 0.00000e+00,
        4.00000e+01, 3.80000e+01],
       ...,
       [2.80000e+01, 6.00000e+00, 2.11032e+05, ..., 0.00000e+00,
        4.00000e+01, 2.50000e+01],
       [4.40000e+01, 4.00000e+00, 1.67005e+05, ..., 0.00000e+00,
        6.00000e+01, 3.80000e+01],
       [5.30000e+01, 4.00000e+00, 2.57940e+05, ..., 0.00000e+00,
        4.00000e+01, 3.80000e+01]])
```

```
model_xgb.predict_proba(X_test_lime)
```

```
array([[7.9646105e-01, 2.0353897e-01],
       [9.5173013e-01, 4.8269872e-02],
       [7.9344827e-01, 2.0655173e-01],
       ...,
       [9.9031430e-01, 9.6856682e-03],
       [6.4581633e-04, 9.9935418e-01],
       [9.7104281e-01, 2.8957171e-02]], dtype=float32)
```

```
print(features,
      class_names,
```

```
categorical_features,  
categorical_names)
```

```
['age', 'workclass', 'fnlwgt', 'education', 'education_num',  
'marital', 'occupation', 'relationship', 'race', 'sex',  
'capital_gain', 'capital_loss', 'hours_week', 'native_country']  
['<=50K' '>50K'] [1, 3, 5, 6, 7, 8, 9, 13] {'workclass':  
array(['?', 'Federal-gov', 'Local-gov', 'Never-worked', 'Private',  
      'Self-emp-inc', 'Self-emp-not-inc', 'State-gov', 'Without-  
pay'],  
      dtype=object), 'education': array(['10th', '11th', '12th',  
'1st-4th', '5th-6th', '7th-8th', '9th',  
      'Assoc-acdm', 'Assoc-voc', 'Bachelors', 'Doctorate', 'HS-  
grad',  
      'Masters', 'Preschool', 'Prof-school', 'Some-college'],  
      dtype=object), 'marital': array(['Divorced', 'Married-AF-  
spouse', 'Married-civ-spouse',  
      'Married-spouse-absent', 'Never-married', 'Separated',  
'Widowed'],  
      dtype=object), 'occupation': array(['?', 'Adm-clerical',  
'Armed-Forces', 'Craft-repair',  
      'Exec-managerial', 'Farming-fishing', 'Handlers-cleaners',  
      'Machine-op-inspct', 'Other-service', 'Priv-house-serv',  
      'Prof-specialty', 'Protective-serv', 'Sales', 'Tech-  
support',  
      'Transport-moving'], dtype=object), 'relationship':  
array(['Husband', 'Not-in-family', 'Other-relative', 'Own-child',  
      'Unmarried', 'Wife'], dtype=object), 'race': array(['Amer-  
Indian-Eskimo', 'Asian-Pac-Islander', 'Black', 'Other',  
      'White'], dtype=object), 'sex': array(['Female', 'Male'],  
dtype=object), 'native_country': array(['?', 'Cambodia', 'Canada',  
'China', 'Columbia', 'Cuba',  
      'Dominican-Republic', 'Ecuador', 'El-Salvador', 'England',  
      'France', 'Germany', 'Greece', 'Guatemala', 'Haiti',  
'Honduras',  
      'Hong', 'Hungary', 'India', 'Iran', 'Ireland', 'Italy',  
'Jamaica',  
      'Japan', 'Laos', 'Mexico', 'Nicaragua',  
      'Outlying-US(Guam-USVI-etc)', 'Peru', 'Philippines',  
'Poland',  
      'Portugal', 'Puerto-Rico', 'Scotland', 'South', 'Taiwan',  
      'Thailand', 'Trinidad&Tobago', 'United-States', 'Vietnam',  
      'Yugoslavia'], dtype=object)}
```

```
import lime
```

```

import lime.lime_tabular
### Train should be label encoded not one hot encoded
explainer = lime.lime_tabular.LimeTabularExplainer(X_train_lime ,
                                                    feature_names =
features,
                                                    class_names=class_names,
                                                    categorical_features=categorical_features,
                                                    categorical_names=categorical_names,
                                                    kernel_width=3)

```

Lets choose a random household from the test set and see the model prediction and how the computer made his choice.

```

import numpy as np
np.random.seed(1)
i = 100
print(y_test_lime.iloc[i])
>50K

```

```
X_test_lime[i]
```

```

array([4.20000e+01, 4.00000e+00, 1.76286e+05, 7.00000e+00,
1.20000e+01,
        2.00000e+00, 4.00000e+00, 0.00000e+00, 4.00000e+00,
1.00000e+00,
        0.00000e+00, 0.00000e+00, 4.00000e+01, 3.80000e+01])

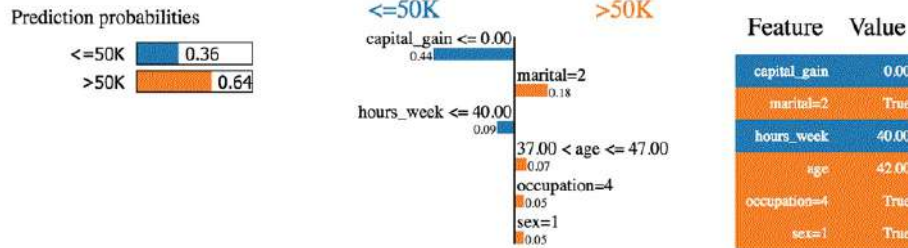
```

You can use the explainer with `explain_instance` to check the explanation behind the model

```

exp = explainer.explain_instance(X_test_lime[i], predict_fn,
num_features=6)
exp.show_in_notebook(show_all=False)

```



We can see that the classifier predicted the household correctly. The income is, indeed, above 50k.

The first thing we can say is the classifier is not that sure about the predicted probabilities. The machine predicts the household has an income over 50k with a probability of 64%. This 64% is made up of Capital gain and marital. The blue color contributes negatively to the positive class and the orange line, positively.

The classifier is confused because the capital gain of this household is null, while the capital gain is usually a good predictor of wealth. Besides, the household works less than 40 hours per week. Age, occupation, and sex contribute positively to the classifier.

If the marital status were single, the classifier would have predicted an income below 50k ($0.64 - 0.18 = 0.46$)

We can try with another household which has been wrongly classified

```
temp.head(3)
temp.iloc[1, :-2]
```

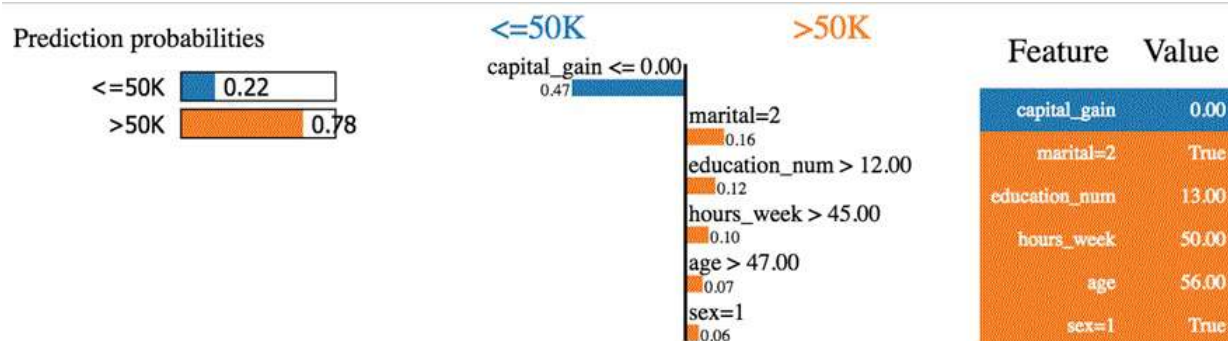
age	58
workclass	4
fnlwgt	68624
education	11
education_num	9
marital	2
occupation	4
relationship	0
race	4
sex	1


```
capital_gain      0
capital_loss      0
hours_week        45
native_country    38
Name: 20931, dtype: object
```

```
i = 1
print('This observation is', temp.iloc[i,-2:])
```

```
This observation is label      <=50K
predicted      >50K
Name: 20931, dtype: object
```

```
exp = explainer.explain_instance(temp.iloc[1,-2], predict_fn,
num_features=6)
exp.show_in_notebook(show_all=False)
```



The classifier predicted an income below 50k while it is untrue. This household seems odd. It does not have a capital gain, nor capital loss. He is divorced and is 60 years old, and it is an educated people, i.e., education_num > 12. According to the overall pattern, this household should, like explain by the classifier, get an income below 50k.

You try to play around with LIME. You will notice gross mistakes from the classifier.

You can check the GitHub of the owner of the library. They provide extra documentation for image and text classification.

Summary

Below is a list of some useful command with scikit learn version ≥ 0.20

create train/test dataset	trainees split
Build a pipeline	
select the column and apply the transformation	makecolumntransformer
type of transformation	
standardize	StandardScaler
min max	MinMaxScaler
Normalize	Normalizer
Impute missing value	Imputer
Convert categorical	OneHotEncoder
Fit and transform the data	fit_transform
Make the pipeline	make_pipeline
Basic model	
logistic regression	LogisticRegression
XGBoost	XGBClassifier

Neural net	MLPClassifier
Grid search	GridSearchCV
Randomized search	RandomizedSearchCV

Chapter 13: Linear Regression

Linear regression

In this tutorial, you will learn basic principles of linear regression and machine learning in general.

TensorFlow provides tools to have full control of the computations. This is done with the low-level API. On top of that, TensorFlow is equipped with a vast array of APIs to perform many machine learning algorithms. This is the high-level API. TensorFlow calls them estimators

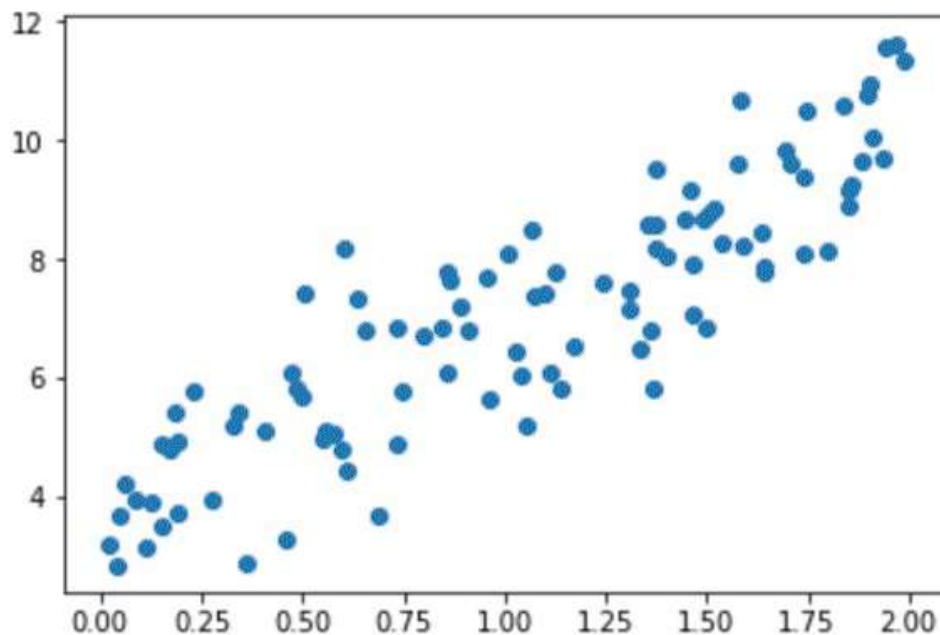
- Low-level API: Build the architecture, optimization of the model from scratch. It is complicated for a beginner
- High-level API: Define the algorithm. It is easier-friendly. TensorFlow provides a toolbox calls **estimator** to construct, train, evaluate and make a prediction.

In this tutorial, you will use the **estimators only**. The computations are faster and are easier to implement. The first part of the tutorial explains how to use the gradient descent optimizer to train a linear regression. In a second part, you will use the Boston dataset to predict the price of a house using TensorFlow estimator.

How to train a linear regression model

Before we begin to train the model, let's have look at what is a linear regression.

Imagine you have two variables, x and y and your task is to predict the value of y knowing the value of x . If you plot the data, you can see a positive relationship between your independent variable, x and your dependent variable y .



You may observe, if $x=1$, y will roughly be equal to 6 and if $x=2$, y will be around 8.5.

This is not a very accurate method and prone to error, especially with a dataset with hundreds of thousands of points.

A linear regression is evaluated with an equation. The variable y is explained by one or many covariates. In your example, there is only one dependent variable. If you have to write this equation, it will be:

$$y = \beta + \alpha X + \epsilon$$

With:

- β is the bias. i.e. if $x=0$, $y=\beta$
- α is the weight associated to x
- ϵ is the residual or the error of the model. It includes what the model cannot learn from the data

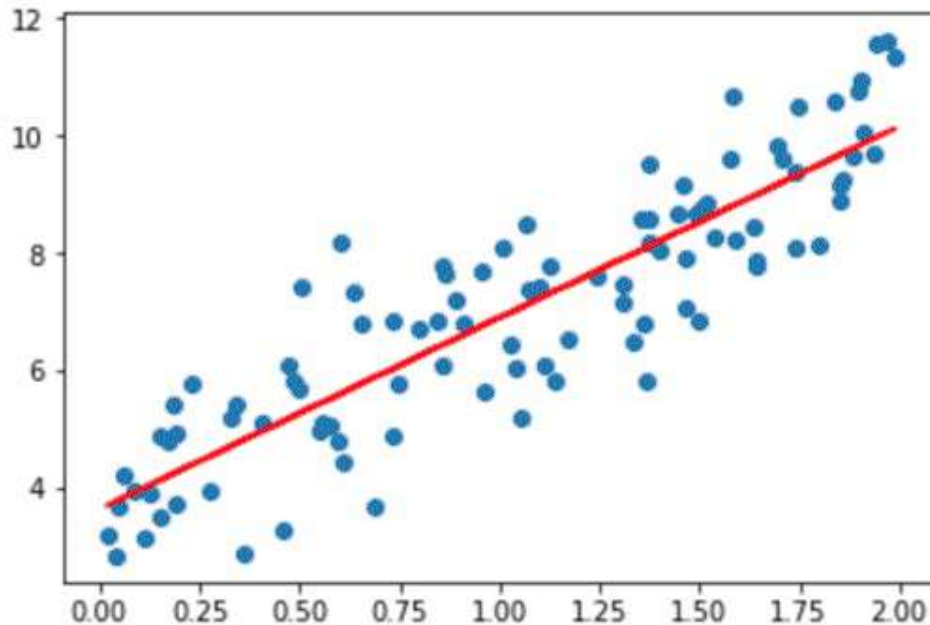
Imagine you fit the model and you find the following solution for:

- $\beta = 3.8$
- $\alpha = 2.78$

You can substitute those numbers in the equation and it becomes:

$$y = 3.8 + 2.78x$$

You have now a better way to find the values for y . That is, you can replace x with any value you want to predict y . In the image below, we have replace x in the equation with all the values in the dataset and plot the result.



The red line represents the fitted value, that is the values of y for each value of x . You don't need to see the value of x to predict y , for each x there is any which belongs to the red line. You can also predict for values of x higher than 2!

If you want to extend the linear regression to more covariates, you can by adding more variables to the model. The difference between traditional analysis and linear regression is the linear regression looks at how y will react for each variable x taken independently.

Let's see an example. Imagine you want to predict the sales of an ice cream shop. The dataset contains different information such as the weather (i.e rainy, sunny, cloudy), customer informations (i.e salary, gender, marital status).

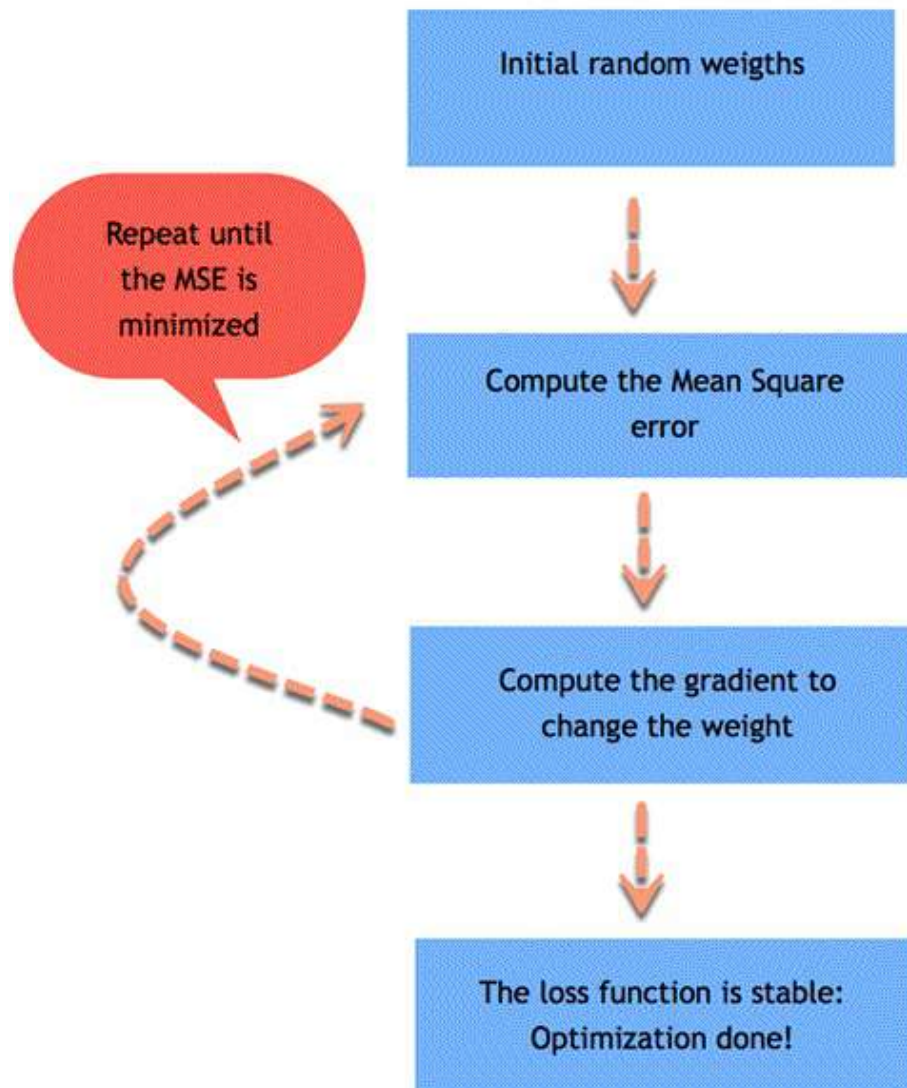
Traditional analysis will try to predict the sale by let's say computing the average for each variable and try to estimate the sale for different scenarios. It will lead to poor predictions and restrict the analysis to the chosen scenario.

If you use linear regression, you can write this equation:

$$Sales = \beta + \alpha_1 weather + \alpha_2 salary + \alpha_3 gender + \alpha_4 marital + \epsilon$$

The algorithm will find the best solution for the weights; it means it will try to minimize the cost (the difference between the fitted line and the data points).

How the algorithm works



The algorithm will choose a random number for each β and α and replace the value of x to get the predicted value of y . If the dataset has 100 observations, the algorithm computes 100 predicted values.

We can compute the error, noted ϵ of the model, which is the difference between the predicted value and the real value. A positive error means the model underestimates the prediction of y , and a negative error means the model overestimates the prediction of y .

$$\epsilon = y - y_{pred}$$

Your goal is to minimize the square of the error. The algorithm computes the mean of the square error. This step is called minimization of the error. For linear regression is the **Mean Square Error**, also called MSE. Mathematically, it is:

$$MSE(X) = \frac{1}{m} \sum_{i=1}^m (\theta^T x^i - y^i)^2$$

Where:

- θ^T is the weights so $\theta^T x^i$ refers to the predicted value
- y is the real values
- m is the number of observations

Note that θ^T means it uses the transpose of the matrices. The $\frac{1}{m} \sum$ is the mathematical notation of the mean.

The goal is to find the best θ that minimize the MSE

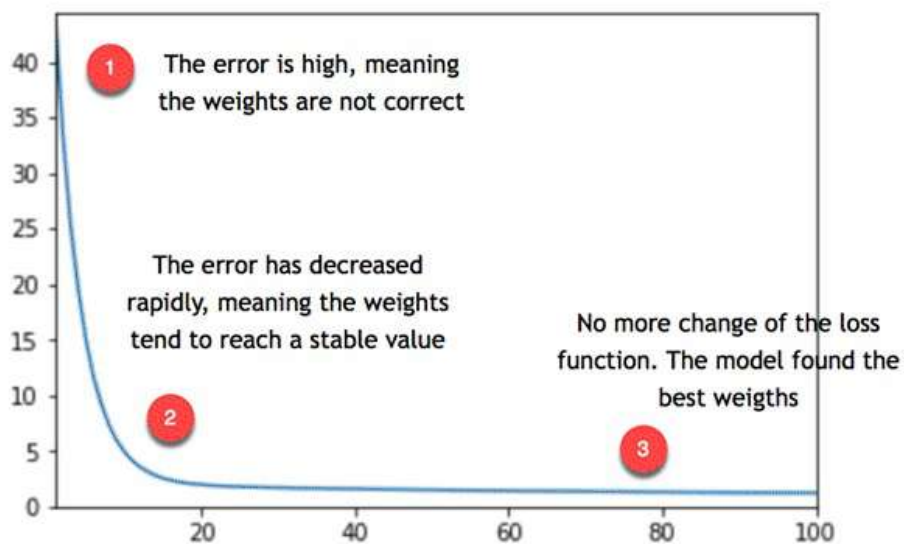
If the average error is large, it means the model performs poorly and the weights are not chosen properly. To correct the weights, you need to use an optimizer.

The traditional optimizer is called **Gradient Descent**.

The gradient descent takes the derivative and decreases or increases the weight. If the derivative is positive, the weight is decreased. If the derivative is negative,

the weight increases. The model will update the weights and recompute the error. This process is repeated until the error does not change anymore. Each process is called an **iteration**. Besides, the gradients are multiplied by a learning rate. It indicates the speed of the learning.

If the learning rate is too small, it will take very long time for the algorithm to converge (i.e requires lots of iterations). If the learning rate is too high, the algorithm might never converge.



You can see from the picture above, the model repeats the process about 20 times before to find a stable value for the weights, therefore reaching the lowest error.

Note that, the error is not equal to zero but stabilizes around 5. It means, the model makes a typical error of 5. If you want to reduce the error, you need to add more information to the model such as more variables or use different estimators.

You remember the first equation

$$y = \beta + \alpha X + \epsilon$$

The final weights are 3.8 and 2.78. The video below shows you how the gradient descent optimize the loss function to find this weights

How to train a Linear Regression with TensorFlow

Now that you have a better understanding of what is happening behind the hood, you are ready to use the estimator API provided by TensorFlow to train your first linear regression.

You will use the Boston Dataset, which includes the following variables

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft.
indus	proportion of non-retail business acres per town.
nox	nitric oxides concentration
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built before 1940
dis	weighted distances to five Boston employment centers
tax	full-value property-tax rate per dollars 10,000
ptratio	pupil-teacher ratio by town
medv	Median value of owner-occupied homes in thousand dollars

You will create three different datasets:

dataset	objective	shape
Training	Train the model and obtain the weights	400, 10
Evaluation	Evaluate the performance of the model on unseen data	100, 10
Predict	Use the model to predict house value on new data	6, 10

The objectives is to use the features of the dataset to predict the value of the house.

During the second part of the tutorial, you will learn how to use TensorFlow with three different way to import the data:

- With Pandas

- With Numpy
- Only TF

Note that, all options **provide the same results.**

You will learn how to use the high-level API to build, train and evaluate a linear regression model. If you were using the low-level API, you had to define by hand the:

- Loss function
- Optimize: Gradient descent
- Matrices multiplication
- Graph and tensor

This is tedious and more complicated for beginner.

Pandas

You need to import the necessary libraries to train the model.

```
import pandas as pd
from sklearn import datasets
import tensorflow as tf
import itertools
```

Step 1) Import the data with panda.

You define the column names and store it in COLUMNS. You can use `pd.read_csv()` to import the data.

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",
            "dis", "tax", "ptratio", "medv"]
```

```
training_set = pd.read_csv("E:/boston_train.csv",
skipinitialspace=True,skiprows=1, names=COLUMNS)
```

```
test_set = pd.read_csv("E:/boston_test.csv", skipinitialspace=True,skiprows=1,
names=COLUMNS)
```

```
prediction_set = pd.read_csv("E:/boston_predict.csv",
skipinitialspace=True,skiprows=1, names=COLUMNS)
```

You can print the shape of the data.

```
print(training_set.shape, test_set.shape, prediction_set.shape)
```

Output

```
(400, 10) (100, 10) (6, 10)
```

Note that the label, i.e. your `y`, is included in the dataset. So you need to define two other lists. One containing only the features and one with the name of the

label only. These two lists will tell your estimator what are the features in the dataset and what column name is the label

It is done with the code below.

```
FEATURES = ["crim", "zn", "indus", "nox", "rm",  
            "age", "dis", "tax", "ptratio"]  
LABEL = "medv"
```

Step 2) Convert the data

You need to convert the numeric variables in the proper format. Tensorflow provides a method to convert continuous variable:
`tf.feature_column.numeric_column()`.

In the previous step, you define a list a feature you want to include in the model. Now you can use this list to convert them into numeric data. If you want to exclude features in your model, feel free to drop one or more variables in the list `FEATURES` before you construct the `feature_cols`

Note that you will use Python list comprehension with the list `FEATURES` to create a new list named `feature_cols`. It helps you avoid writing nine times `tf.feature_column.numeric_column()`. A list comprehension is a faster and cleaner way to create new lists

```
feature_cols = [tf.feature_column.numeric_column(k) for k in  
FEATURES]
```

Step 3) Define the estimator

In this step, you need to define the estimator. Tensorflow currently provides 6 pre-built estimators, including 3 for classification task and 3 for regression task:

- Regressor
 - `DNNRegressor`
 - `LinearRegressor`

- DNNLineaCombinedRegressor
- Classifier
 - DNNClassifier
 - LinearClassifier
 - DNNLineaCombinedClassifier

In this tutorial, you will use the Linear Regressor. To access this function, you need to use `tf.estimator`.

The function needs two arguments:

- `feature_columns`: Contains the variables to include in the model
- `model_dir`: path to store the graph, save the model parameters, etc

Tensorflow will automatically create a file named `train` in your working directory. You need to use this path to access the Tensorboard.

```
estimator = tf.estimator.LinearRegressor(  
    feature_columns=feature_cols,  
    model_dir="train")
```

Output

```
INFO:tensorflow:Using default config.  
INFO:tensorflow:Using config: {'_model_dir': 'train',  
'_tf_random_seed': None, '_save_summary_steps': 100,  
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,  
'_session_config': None, '_keep_checkpoint_max': 5,  
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':  
100, '_train_distribute': None, '_service': None, '_cluster_spec':  
<tensorflow.python.training.server_lib.ClusterSpec object at  
0x1a215dc550>, '_task_type': 'worker', '_task_id': 0,  
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':  
'', '_is_chief': True, '_num_ps_replicas': 0,  
'_num_worker_replicas': 1}
```

The tricky part with TensorFlow is the way to feed the model. Tensorflow is designed to work with parallel computing and very large dataset. Due to the

limitation of the machine resources, it is impossible to feed the model with all the data at once. For that, you need to feed a batch of data each time. Note that, we are talking about huge dataset with millions or more records. If you don't add batch, you will end up with a memory error.

For instance, if your data contains 100 observations and you define a batch size of 10, it means the model will see 10 observations for each iteration (10*10).

When the model has seen all the data, it finishes one **epoch**. An epoch defines how many times you want the model to see the data. It is better to set this step to none and let the model performs iteration number of time.

A second information to add is if you want to shuffle the data before each iteration. During the training, it is important to shuffle the data so that the model does not learn specific pattern of the dataset. If the model learns the details of the underlying pattern of the data, it will have difficulties to generalize the prediction for unseen data. This is called **overfitting**. The model performs well on the training data but cannot predict correctly for unseen data.

TensorFlow makes this two steps easy to do. When the data goes to the pipeline, it knows how many observations it needs (batch) and if it has to shuffle the data.

To instruct Tensorflow how to feed the model, you can use `pandas_input_fn`. This object needs 5 parameters:

- `x`: feature data
- `y`: label data
- `batch_size`: batch. By default 128
- `num_epoch`: Number of epoch, by default 1
- `shuffle`: Shuffle or not the data. By default, None

You need to feed the model many times so you define a function to repeat this process. all this function `get_input_fn`.

```
def get_input_fn(data_set, num_epochs=None, n_batch = 128,
shuffle=True):
    return tf.estimator.inputs.pandas_input_fn(
        x=pd.DataFrame({k: data_set[k].values for k in FEATURES}),
        y = pd.Series(data_set[LABEL].values),
        batch_size=n_batch,
        num_epochs=num_epochs,
        shuffle=shuffle)
```

The usual method to evaluate the performance of a model is to:

- Train the model
- Evaluate the model in a different dataset
- Make prediction

Tensorflow estimator provides three different functions to carry out this three steps easily.

Step 4): Train the model

You can use the estimator train to evaluate the model. The train estimator needs an input_fn and a number of steps. You can use the function you created above to feed the model. Then, you instruct the model to iterate 1000 times. Note that, you don't specify the number of epochs, you let the model iterates 1000 times. If you set the number of epoch to 1, then the model will iterate 4 times: There are 400 records in the training set, and the batch size is 128

1. 128 rows
2. 128 rows
3. 128 rows
4. 16 rows

Therefore, it is easier to set the number of epoch to none and define the number of iteration.

```
estimator.train(input_fn=get_input_fn(training_set,
                                     num_epochs=None,
```

```
n_batch = 128,  
shuffle=False),  
steps=1000)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 238.616  
INFO:tensorflow:loss = 13909.657, step = 101 (0.420 sec)  
INFO:tensorflow:global_step/sec: 314.293  
INFO:tensorflow:loss = 12881.449, step = 201 (0.320 sec)  
INFO:tensorflow:global_step/sec: 303.863  
INFO:tensorflow:loss = 12391.541, step = 301 (0.327 sec)  
INFO:tensorflow:global_step/sec: 308.782  
INFO:tensorflow:loss = 12050.5625, step = 401 (0.326 sec)  
INFO:tensorflow:global_step/sec: 244.969  
INFO:tensorflow:loss = 11766.134, step = 501 (0.407 sec)  
INFO:tensorflow:global_step/sec: 155.966  
INFO:tensorflow:loss = 11509.922, step = 601 (0.641 sec)  
INFO:tensorflow:global_step/sec: 263.256  
INFO:tensorflow:loss = 11272.889, step = 701 (0.379 sec)  
INFO:tensorflow:global_step/sec: 254.112  
INFO:tensorflow:loss = 11051.9795, step = 801 (0.396 sec)  
INFO:tensorflow:global_step/sec: 292.405  
INFO:tensorflow:loss = 10845.855, step = 901 (0.341 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into train/model.ckpt.  
INFO:tensorflow:Loss for final step: 5925.9873.
```

You can check the Tensorboard will the following command:

```
activate hello-tf  
# For MacOS  
tensorboard --logdir=./train  
# For Windows  
tensorboard --logdir=train
```

Step 5) Evaluate your model

You can evaluate the fit of your model on the test set with the code below:

```
ev = estimator.evaluate(  
    input_fn=get_input_fn(test_set,  
    num_epochs=1,  
    n_batch = 128,  
    shuffle=False))
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Starting evaluation at 2018-05-13-01:43:13  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Finished evaluation at 2018-05-13-01:43:13  
INFO:tensorflow:Saving dict for global step 1000: average_loss =  
32.15896, global_step = 1000, loss = 3215.896
```

You can print the loss with the code below:

```
loss_score = ev["loss"]  
print("Loss: {0:f}".format(loss_score))
```

Output

```
Loss: 3215.895996
```

The model has a loss of 3215. You can check the summary statistic to get an idea of how big the error is.

```
training_set['medv'].describe()
```

Output

```
count    400.000000  
mean      22.625500
```

```
std          9.572593
min          5.000000
25%         16.600000
50%         21.400000
75%         25.025000
max         50.000000
Name: medv, dtype: float64
```

From the summary statistic above, you know that the average price for a house is 22 thousand, with a minimum price of 9 thousands and maximum of 50 thousand. The model makes a typical error of 3k dollars.

Step 6) Make the prediction

Finally, you can use the estimator predict to estimate the value of 6 Boston houses.

```
y = estimator.predict(
    input_fn=get_input_fn(prediction_set,
        num_epochs=1,
        n_batch = 128,
        shuffle=False))
```

To print the estimated values of , you can use this code:

```
predictions = list(p["predictions"] for p in itertools.islice(y,
6))print("Predictions: {}".format(str(predictions)))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Predictions: [array([32.297546], dtype=float32), array([18.96125],
dtype=float32), array([27.270979], dtype=float32),
array([29.299236], dtype=float32), array([16.436684],
dtype=float32), array([21.460876], dtype=float32)]
```

The model forecast the following values:

House	Prediction
1	32.29
2	18.96
3	27.27
4	29.29
5	16.43
7	21.46

Note that we don't know the true value of y . In the tutorial of deep learning, you will try to beat the linear model

Numpy Solution

This section explains how to train the model using a numpy estimator to feed the data. The method is the same except that you will use `numpy_input_fn` estimator.

```
training_set_n = pd.read_csv("E:/boston_train.csv").values
```

```
test_set_n = pd.read_csv("E:/boston_test.csv").values
```

```
prediction_set_n = pd.read_csv("E:/boston_predict.csv").values
```

Step 1) Import the data

First of all, you need to differentiate the feature variables from the label. You need to do this for the training data and evaluation. It is faster to define a function to split the data.

```
def prepare_data(df):  
    X_train = df[:, :-3]  
    y_train = df[:, -3]  
    return X_train, y_train
```

You can use the function to split the label from the features of the train/evaluate dataset

```
X_train, y_train = prepare_data(training_set_n)  
X_test, y_test = prepare_data(test_set_n)
```

You need to exclude the last column of the prediction dataset because it contains only NaN

```
x_predict = prediction_set_n[:, :-2]
```

Confirm the shape of the array. Note that, the label should not have a dimension, it means (400,).

```
print(X_train.shape, y_train.shape, x_predict.shape)
```

Output

```
(400, 9) (400,) (6, 9)
```

You can construct the feature columns as follow:

```
feature_columns = [      tf.feature_column.numeric_column('x',
shape=X_train.shape[1:])] ]
```

The estimator is defined as before, you instruct the feature columns and where to save the graph.

```
estimator = tf.estimator.LinearRegressor(
    feature_columns=feature_columns,
    model_dir="train1")
```

Output

```
INFO:tensorflow:Using default config.
INFO:tensorflow:Using config: {'_model_dir': 'train1',
'_tf_random_seed': None, '_save_summary_steps': 100,
'_save_checkpoints_steps': None, '_save_checkpoints_secs': 600,
'_session_config': None, '_keep_checkpoint_max': 5,
'_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps':
100, '_train_distribute': None, '_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1a218d8f28>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

You can use the numpy estimator to feed the data to the model and then train the model. Note that, we define the `input_fn` function before to ease the readability.

```
# Train the estimator
train_input =
tf.estimator.inputs.numpy_input_fn(
    x={"x": X_train},
    y=y_train,
    batch_size=128,
```



```
shuffle=False,  
num_epochs=None)  
estimator.train(input_fn = train_input, steps=5000)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Create CheckpointSaverHook.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
INFO:tensorflow:Saving checkpoints for 1 into train1/model.ckpt.  
INFO:tensorflow:loss = 83729.64, step = 1  
INFO:tensorflow:global_step/sec: 490.057  
INFO:tensorflow:loss = 13909.656, step = 101 (0.206 sec)  
INFO:tensorflow:global_step/sec: 788.986  
INFO:tensorflow:loss = 12881.45, step = 201 (0.126 sec)  
INFO:tensorflow:global_step/sec: 736.339  
INFO:tensorflow:loss = 12391.541, step = 301 (0.136 sec)  
INFO:tensorflow:global_step/sec: 383.305  
INFO:tensorflow:loss = 12050.561, step = 401 (0.260 sec)  
INFO:tensorflow:global_step/sec: 859.832  
INFO:tensorflow:loss = 11766.133, step = 501 (0.117 sec)  
INFO:tensorflow:global_step/sec: 804.394  
INFO:tensorflow:loss = 11509.918, step = 601 (0.125 sec)  
INFO:tensorflow:global_step/sec: 753.059  
INFO:tensorflow:loss = 11272.891, step = 701 (0.134 sec)  
INFO:tensorflow:global_step/sec: 402.165  
INFO:tensorflow:loss = 11051.979, step = 801 (0.248 sec)  
INFO:tensorflow:global_step/sec: 344.022  
INFO:tensorflow:loss = 10845.854, step = 901 (0.288 sec)  
INFO:tensorflow:Saving checkpoints for 1000 into train1/model.ckpt.  
INFO:tensorflow:Loss for final step: 5925.985.  
Out[23]:  
<tensorflow.python.estimator.canned.linear.LinearRegressor at  
0x1a1b6ea860>
```

You replicate the same step with a different estimator to evaluate your model

```
eval_input = tf.estimator.inputs.numpy_input_fn(  
    x={"x": X_test},  
    y=y_test,  
    shuffle=False,
```

```
    batch_size=128,
    num_epochs=1)
estimator.evaluate(eval_input, steps=None)
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-13-01:44:00
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-13-01:44:00
INFO:tensorflow:Saving dict for global step 1000: average_loss =
32.158947, global_step = 1000, loss = 3215.8945
Out[24]:
{'average_loss': 32.158947, 'global_step': 1000, 'loss': 3215.8945}
```

Finally, you can compute the prediction. It should be the similar as pandas.

```
test_input = tf.estimator.inputs.numpy_input_fn(
    x={"x": x_predict},
    batch_size=128,
    num_epochs=1,
    shuffle=False)
y = estimator.predict(test_input)
predictions = list(p["predictions"] for p in itertools.islice(y,
6))
print("Predictions: {}".format(str(predictions)))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train1/model.ckpt-1000
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
Predictions: [array([32.297546], dtype=float32), array([18.961248],
dtype=float32), array([27.270979], dtype=float32),
array([29.299242], dtype=float32), array([16.43668],
dtype=float32), array([21.460878], dtype=float32)]
```

Tensorflow solution

The last section is dedicated to a TensorFlow solution. This method is slightly more complicated than the other one.

Note that if you use Jupyter notebook, you need to Restart and clean the kernel to run this session.

TensorFlow has built a great tool to pass the data into the pipeline. In this section, you will build the `input_fn` function by yourself.

Step 1) Define the path and the format of the data

First of all, you declare two variables with the path of the csv file. Note that, you have two files, one for the training set and one for the testing set.

```
import tensorflow as tf
```

```
df_train = "E:/boston_train.csv"
```

```
df_eval = "E:/boston_test.csv"
```

Then, you need to define the columns you want to use from the csv file. We will use all. After that, you need to declare the type of variable it is.

Floats variable are defined by `[0.]`

```
COLUMNS = ["crim", "zn", "indus", "nox", "rm", "age",  
            "dis", "tax", "ptratio", "medv"]  
RECORDS_ALL =  
[[0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], [0.0]]
```

Step 2) Define the `input_fn` function

The function can be broken into three part:

1. Import the data

2. Create the iterator
3. Consume the data

Below is the overall code to define the function. The code will be explained after

```
def input_fn(data_file, batch_size, num_epoch = None):
    # Step 1
    def parse_csv(value):
        columns = tf.decode_csv(value, record_defaults=
RECORDS_ALL)
        features = dict(zip(COLUMNS, columns))
        #labels = features.pop('median_house_value')
        labels = features.pop('medv')
        return features, labels

    # Extract lines from input files using the
    Dataset API.    dataset =
(tf.data.TextLineDataset(data_file) # Read text file
    .skip(1) # Skip header row
    .map(parse_csv))

    dataset = dataset.repeat(num_epoch)
    dataset = dataset.batch(batch_size)
    # Step 3
    iterator = dataset.make_one_shot_iterator()
    features, labels = iterator.get_next()
    return features, labels
```

**** Import the data****

For a csv file, the dataset method reads one line at a time. To build the dataset, you need to use the object `TextLineDataset`. Your dataset has a header so you need to use `skip(1)` to skip the first line. At this point, you only read the data and exclude the header in the pipeline. To feed the model, you need to separate the features from the label. The method used to apply any transformation to the data is `map`.

This method calls a function that you will create in order to instruct how to transform the data. In a nutshell, you need to pass the data in the `TextLineDataset`

object, exclude the header and apply a transformation which is instructed by a function. Code explanation

- `tf.data.TextLineDataset(data_file)`: This line read the csv file
- `.skip(1)` : skip the header
- `.map(parse_csv)`: parse the records into the tensors You need to define a function to instruct the map object. You can call this function `parse_csv`.

This function parses the csv file with the method `tf.decode_csv` and declares the features and the label. The features can be declared as a dictionary or a tuple. You use the dictionary method because it is more convenient. Code explanation

- `tf.decode_csv(value, record_defaults= RECORDS_ALL)`: the method `decode_csv` uses the output of the `TextLineDataset` to read the csv file. `record_defaults` instructs TensorFlow about the columns type.
- `dict(zip(_CSV_COLUMNS, columns))`: Populate the dictionary with all the columns extracted during this data processing
- `features.pop('median_house_value')`: Exclude the target variable from the feature variable and create a label variable

The Dataset needs further elements to iteratively feeds the Tensors. Indeed, you need to add the method `repeat` to allow the dataset to continue indefinitely to feed the model. If you don't add the method, the model will iterate only one time and then throw an error because no more data are fed in the pipeline.

After that, you can control the batch size with the `batch` method. It means you tell the dataset how many data you want to pass in the pipeline for each iteration. If you set a big batch size, the model will be slow.

Step 3) Create the iterator

Now you are ready for the second step: create an iterator to return the elements in the dataset.

The simplest way of creating an operator is with the method `make_one_shot_iterator`.

After that, you can create the features and labels from the iterator.

Step 4) Consume the data

You can check what happens with `input_fn` function. You need to call the function in a session to consume the data. You try with a batch size equals to 1.

Note that, it prints the features in a dictionary and the label as an array.

It will show the first line of the csv file. You can try to run this code many times with different batch size.

```
next_batch = input_fn(df_train, batch_size = 1, num_epoch = None)
with tf.Session() as sess:
    first_batch = sess.run(next_batch)
    print(first_batch)
```

Output

```
({'crim': array([2.3004], dtype=float32), 'zn': array([0.],
dtype=float32), 'indus': array([19.58], dtype=float32), 'nox':
array([0.605], dtype=float32), 'rm': array([6.319], dtype=float32),
'age': array([96.1], dtype=float32), 'dis': array([2.1],
dtype=float32), 'tax': array([403.], dtype=float32), 'ptratio':
array([14.7], dtype=float32)}, array([23.8], dtype=float32))
```

Step 4) Define the feature column

You need to define the numeric columns as follow:

```
X1= tf.feature_column.numeric_column('crim')
X2= tf.feature_column.numeric_column('zn')
X3= tf.feature_column.numeric_column('indus')
X4= tf.feature_column.numeric_column('nox')
X5= tf.feature_column.numeric_column('rm')
X6= tf.feature_column.numeric_column('age')
```

```
X7= tf.feature_column.numeric_column('dis')
X8= tf.feature_column.numeric_column('tax')
X9= tf.feature_column.numeric_column('ptratio')
```

Note that you need to combined all the variables in a bucket

```
base_columns = [X1, X2, X3,X4, X5, X6,X7, X8, X9]
```

Step 5) Build the model

You can train the model with the estimator LinearRegressor.

```
model = tf.estimator.LinearRegressor(feature_columns=base_columns,
model_dir='train3')
```

Output

```
INFO:tensorflow:Using default config. INFO:tensorflow:Using config:
{'_model_dir': 'train3', '_tf_random_seed': None,
'_save_summary_steps': 100, '_save_checkpoints_steps': None,
'_save_checkpoints_secs': 600, '_session_config': None,
'_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None,
'_service': None, '_cluster_spec':
<tensorflow.python.training.server_lib.ClusterSpec object at
0x1820a010f0>, '_task_type': 'worker', '_task_id': 0,
'_global_id_in_cluster': 0, '_master': '', '_evaluation_master':
'', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
```

You need to use a lambda function to allow to write the argument in the function `input_fn`. If you don't use a lambda function, you cannot train the model.

```
# Train the estimator model.train(steps =1000,
input_fn= lambda : input_fn(df_train,batch_size=128,
num_epoch = None))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
```

```
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 1 into train3/model.ckpt.
INFO:tensorflow:loss = 83729.64, step = 1
INFO:tensorflow:global_step/sec: 72.5646
INFO:tensorflow:loss = 13909.657, step = 101 (1.380 sec)
INFO:tensorflow:global_step/sec: 101.355
INFO:tensorflow:loss = 12881.449, step = 201 (0.986 sec)
INFO:tensorflow:global_step/sec: 109.293
INFO:tensorflow:loss = 12391.541, step = 301 (0.915 sec)
INFO:tensorflow:global_step/sec: 102.235
INFO:tensorflow:loss = 12050.5625, step = 401 (0.978 sec)
INFO:tensorflow:global_step/sec: 104.656
INFO:tensorflow:loss = 11766.134, step = 501 (0.956 sec)
INFO:tensorflow:global_step/sec: 106.697
INFO:tensorflow:loss = 11509.922, step = 601 (0.938 sec)
INFO:tensorflow:global_step/sec: 118.454
INFO:tensorflow:loss = 11272.889, step = 701 (0.844 sec)
INFO:tensorflow:global_step/sec: 114.947
INFO:tensorflow:loss = 11051.9795, step = 801 (0.870 sec)
INFO:tensorflow:global_step/sec: 111.484
INFO:tensorflow:loss = 10845.855, step = 901 (0.897 sec)
INFO:tensorflow:Saving checkpoints for 1000 into train3/model.ckpt.
INFO:tensorflow:Loss for final step: 5925.9873.
Out[8]:
<tensorflow.python.estimator.canned.linear.LinearRegressor at
0x18225eb8d0>
```

You can evaluate the fit of your model on the test set with the code below:

```
results = model.evaluate(steps=None, input_fn=lambda:
input_fn(df_eval, batch_size=128, num_epoch=1))
for key in results:
print("{} {}, was: {}".format(key, results[key]))
```

Output

```
INFO:tensorflow:Calling model_fn.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Starting evaluation at 2018-05-13-02:06:02
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000
```



```
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Finished evaluation at 2018-05-13-02:06:02
INFO:tensorflow:Saving dict for global step 1000: average_loss =
32.15896, global_step = 1000, loss = 3215.896
  average_loss, was: 32.158958435058594
  loss, was: 3215.89599609375
  global_step, was: 1000
```

The last step is predicting the value of based on the value of , the matrices of the features. You can write a dictionary with the values you want to predict. Your model has 9 features so you need to provide a value for each. The model will provide a prediction for each of them.

In the code below, you wrote the values of each features that is contained in the `df_predict` csv file.

You need to write a new `input_fn` function because there is no label in the dataset. You can use the API `from_tensor` from the Dataset.

```
prediction_input = {
    'crim':
[0.03359, 5.09017, 0.12650, 0.05515, 8.15174, 0.24522],
    'zn': [75.0, 0.0, 25.0, 33.0, 0.0, 0.0],
    'indus': [2.95, 18.10, 5.13, 2.18, 18.10, 9.90],
    'nox': [0.428, 0.713, 0.453, 0.472, 0.700, 0.544],
    'rm': [7.024, 6.297, 6.762, 7.236, 5.390, 5.782],
    'age': [15.8, 91.8, 43.4, 41.1, 98.9, 71.7],
    'dis': [5.4011, 2.3682, 7.9809, 4.0220, 1.7281, 4.0317],
    'tax': [252, 666, 284, 222, 666, 304],
    'ptratio': [18.3, 20.2, 19.7, 18.4, 20.2, 18.4]
}
def test_input_fn():
    dataset = tf.data.Dataset.from_tensors(prediction_input)
    return dataset

# Predict all our prediction_inputpred_results =
model.predict(input_fn=test_input_fn)
```

Finally, you print the predictions.

```
for pred in enumerate(pred_results):  
    print(pred)
```

Output

```
INFO:tensorflow:Calling model_fn.  
INFO:tensorflow:Done calling model_fn.  
INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train3/model.ckpt-1000  
INFO:tensorflow:Running local_init_op.  
INFO:tensorflow:Done running local_init_op.  
(0, {'predictions': array([32.297546], dtype=float32)})  
(1, {'predictions': array([18.96125], dtype=float32)})  
(2, {'predictions': array([27.270979], dtype=float32)})  
(3, {'predictions': array([29.299236], dtype=float32)})  
(4, {'predictions': array([16.436684], dtype=float32)})  
(5, {'predictions': array([21.460876], dtype=float32)})  
  
INFO:tensorflow:Calling model_fn. INFO:tensorflow:Done calling  
model_fn. INFO:tensorflow:Graph was finalized.  
INFO:tensorflow:Restoring parameters from train3/model.ckpt-5000  
INFO:tensorflow:Running local_init_op. INFO:tensorflow:Done running  
local_init_op. (0, {'predictions': array([35.60663],  
dtype=float32)}) (1, {'predictions': array([22.298521],  
dtype=float32)}) (2, {'predictions': array([25.74533],  
dtype=float32)}) (3, {'predictions': array([35.126694],  
dtype=float32)}) (4, {'predictions': array([17.94416],  
dtype=float32)}) (5, {'predictions': array([22.606628],  
dtype=float32)})
```

Summary

To train a model, you need to:

- Define the features: Independent variables: X
- Define the label: Dependent variable: y
- Construct a train/test set
- Define the initial weight
- Define the loss function: MSE
- Optimize the model: Gradient descent

- Define:
 - Learning rate
 - Number of epoch
 - Batch size

In this tutorial, you learned how to use the high level API for a linear regression estimator. You need to define:

1. Feature columns. If continuous: `tf.feature_column.numeric_column()`. You can populate a list with python list comprehension
2. The estimator: `tf.estimator.LinearRegressor(feature_columns, model_dir)`
3. A function to import the data, the batch size and epoch: `input_fn()`

After that, you are ready to train, evaluate and make prediction with `train()`, `evaluate()` and `predict()`

Chapter 14: Linear Regression Case Study

In this tutorial, you will learn how to check the data and prepare it to create a linear regression task.

This tutorial is divided into two parts:

- Look for interaction
- Test the model

In the previous tutorial, you used the Boston dataset to estimate the median price of a house. Boston dataset has a small size, with only 506 observations. This dataset is considered as a benchmark to try new linear regression algorithms.

The dataset is composed of:

Variable	Description
zn	The proportion of residential land zoned for lots over 25,000 sq.ft.
indus	The proportion of non-retail business acres per town.
nox	nitric oxides concentration
rm	average number of rooms per dwelling
age	the proportion of owner-occupied units built before 1940
dis	weighted distances to five Boston employment centers
tax	full-value property-tax rate per dollars 10,000
ptratio	the pupil-teacher ratio by a town
medv	The median value of owner-occupied homes in thousand dollars
crim	per capita crime rate by town
chas	Charles River dummy variable (1 if bounds river; 0 otherwise)
B	the proportion of blacks by the town

In this tutorial, we will estimate the median price using a linear regressor, but the focus is on one particular process of machine learning: "data preparation."

A model generalizes the pattern in the data. To capture such a pattern, you need to find it first. A good practice is to perform a data analysis before running any machine learning algorithm.

Choosing the right features makes all the difference in the success of your model. Imagine you try to estimate the wage of a people, if you do not include the gender as a covariate, you end up with a poor estimate.

Another way to improve the model is to look at the correlation between the independent variable. Back to the example, you can think of education as an excellent candidate to predict the wage but also the occupation. It is fair to say, the occupation depends on the level of education, namely higher education often leads to a better occupation. If we generalize this idea, we can say the correlation between the dependent variable and an explanatory variable can be magnified of yet another explanatory variable.

To capture the limited effect of education on occupation, we can use an interaction term.

INTERACTION TERM

Interaction terms allow us model relationships when the effects of a feature on the target is influenced by another feature.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + e$$

The interaction of features x_1 and x_2 .

ChrisAlbon

If you look at the wage equation, it becomes:

$$wage = \alpha + \beta_1 occupation + \beta_2 education + \beta_3 occupation * education + \epsilon$$

If β_3 is positive, then it implies that an additional level of education yields a higher increase in the median value of a house for a high occupation level. In other words, there is an interaction effect between education and occupation.

In this tutorial, we will try to see which variables can be a good candidate for interaction terms. We will test if adding this kind of information leads to better price prediction.

Summary statistics

There are a few steps you can follow before proceeding to the model. As mentioned earlier, the model is a generalization of the data. The best practice is to understand the data and then make a prediction. If you do not know your data, you have slim chances to improve your model.

As a first step, load the data as a pandas dataframe and create a training set and testing set.

Tips: For this tutorial, you need to have matplotlib and seaborn installed in Python. You can install Python packages on the fly with Jupyter. You **Should not** do this

```
!conda install -- yes matplotlib
```

but

```
import sys
!{sys.executable} -m pip install matplotlib # Already installed
!{sys.executable} -m pip install seaborn
```

Note that this step is not necessary if you have matplotlib and seaborn installed.

Matplotlib is the library to create a graph in Python. Seaborn is a statistical visualization library built on top of matplotlib. It provides attractive and beautiful plots.

The code below imports the necessary libraries.

```
import pandas as pd
from sklearn import datasets
import tensorflow as tf
from sklearn.datasets import load_boston
import numpy as np
```

The library sklearn includes the Boston dataset. You can call its API to import the data.

```
boston = load_boston()
df = pd.DataFrame(boston.data)
```

The feature's name are stored in the object feature_names in an array.

```
boston.feature_names
```

Output

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
       'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

You can rename the columns.

```
df.columns = boston.feature_names
df['PRICE'] = boston.target
df.head(2)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.9	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6

You convert the variable CHAS as a string variable and label it with yes if CHAS = 1 and no if CHAS = 0

```
df['CHAS'] = df['CHAS'].map({1:'yes', 0:'no'})
df['CHAS'].head(5)
0    no
1    no
2    no
3    no
4    no
Name: CHAS, dtype: object
```

With pandas, it is straightforward to split the dataset. You randomly divide the dataset with 80 percent training set and 20 percent testing set. Pandas have a

built-in function to split a data frame sample.

The first parameter `frac` is a value from 0 to 1. You set it to 0.8 to select randomly 80 percent of the data frame.

`Random_state` allows to have the same dataframe returned for everyone.

```
### Create train/test set
df_train=df.sample(frac=0.8,random_state=200)
df_test=df.drop(df_train.index)
```

You can get the shape of the data. It should be:

- Train set: $506 * 0.8 = 405$
- Test set: $506 * 0.2 = 101$

```
print(df_train.shape, df_test.shape)
```

Output

```
(405, 14) (101, 14)
```

```
df_test.head(5)
```

Output

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PR
0	0.00632	18.0	2.31	no	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	no	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.0
3	0.03237	0.0	2.18	no	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.0
6	0.08829	12.5	7.87	no	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.0
7	0.14455	12.5	7.87	no	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.0

Data is messy; it's often misbalanced and sprinkled with outlier values that throw off the analysis and machine learning training.

The first step to getting the dataset cleaned up is understanding where it needs cleaning. Cleaning up a dataset can be tricky to do, especially in any

generalizable manner

Google Research team has developed a tool for this job called **Facets** that help to visualize the data and slice it in all sorts of manners. This is a good starting point to comprehend how the dataset is laid out.

Facets allow you to find where the data does not quite look the way you are thinking.

Except for their web app, Google makes it easy to embed the toolkit into a Jupyter notebook.

There are two parts to Facets:

- Facets Overview
- Facets Deep Dive

Facets Overview

Facets Overview gives an overview of the dataset. Facets Overview splits the columns of the data into rows of salient information showing

1. the percentage of missing observation
2. min and max values
3. statistics like the mean, median, and standard deviation.
4. It also adds a column that shows the percentage of values that are zeroes, which is helpful when most of the values are zeroes.
5. It is possible to see these distributions on the test dataset as well as the training set for each feature. It means you can double-check that the test has a similar distribution to the training dataset.

This is at least the minimum to do before any machine learning task. With this tool, you do not miss this crucial step, and it highlights some abnormalities.

Facets Deep Dive

Facets Deep Dive is a cool tool. It allows to have some clarity on your dataset and zoom all the way in to see an individual piece of data. It means you can facet the data by row and column across any of the features of the dataset.

We will use these two tools with the Boston dataset.

Note: You cannot use Facets Overview and Facets Deep Dive at the same time. You need to clear the notebook first to change the tool.

Install Facet

You can use the Facet web app for most of the analysis. In this tutorial, you will see how to use it within a Jupyter Notebook.

First of all, you need to install nbextensions. It is done with this code. You copy and paste the following code in the terminal of your machine.

```
pip install jupyter_contrib_nbextensions
```

Right after that, you need to clone the repositories in your computer. You have two choices:

Option 1) Copy and paste this code in the terminal (**Recommended**)

If you do not have Git installed on your machine, please go to this URL <https://git-scm.com/download/win> and follow the instruction. Once you are done, you can use the git command in the terminal for Mac User or Anaconda prompt for Windows user

```
git clone https://github.com/PAIR-code/facets
```

Option 2) Go to <https://github.com/PAIR-code/facets> and download the repositories.

105 commits 16 branches 2 releases 17 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

Clone with HTTPS Use SSH
Use Git or checkout with SVN using the web URL.
<https://github.com/PAIR-code/facets.git>

Open in Desktop Download ZIP

Click here to download Facets

File	Description	Time
facets-dist	added rank histogram to custom stats	11 months ago
facets	fix facets jupyter build and distribute new facets dist compi	11 months ago
facets_dive	fix bug with selected indices	11 months ago
facets_overview	expose search string publically	11 months ago
img	adding images for readme	11 months ago
.gitignore	Initial commit	11 months ago
AUTHORS	Initial commit	11 months ago
CONTRIBUTING.md	Initial commit	11 months ago
CONTRIBUTORS	Initial commit	11 months ago
LICENSE	Initial commit	11 months ago
README.md	Add blank lines before lists	9 months ago
WORKSPACE	Add compile=True to HTML binary rules	2 months ago

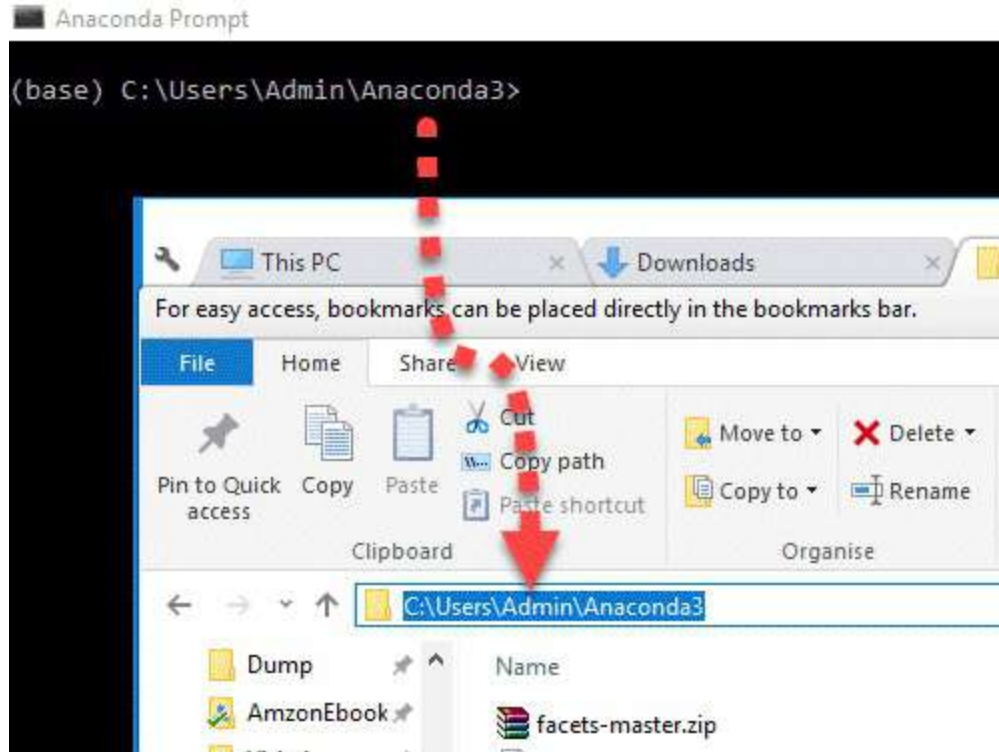
If you choose the first option, the file ends up in your download file. You can either let the file in download or drag it to another path.

You can check where Facets is stored with this command line:

```
echo `pwd`/`ls facets`
```

Now that you have located Facets, you need to install it in Jupyter Notebook. You need to set the working directory to the path where facets is located.

Your present working directory and location of Facets zip should be same.



You need to point the working directory to Facet:

```
cd facets
```

To install Facets in Jupyter, you have two options. If you installed Jupyter with Conda for all the users, copy this code:

can use `jupyter nbextension install facets-dist/`

```
jupyter nbextension install facets-dist/
```

Otherwise, use:

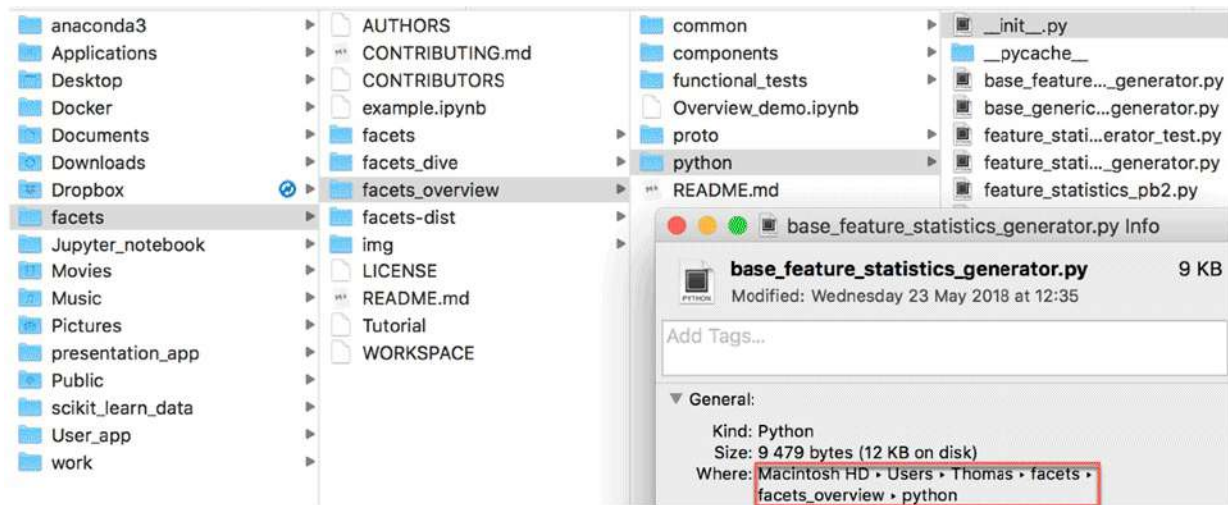
```
jupyter nbextension install facets-dist/ --user
```

All right, you are all set. Let's open Facet Overview.

Overview

Overview uses a Python script to compute the statistics. You need to import the script called `generic_feature_statistics_generator` to Jupyter. Don't worry; the script is located in the facets files.

You need to locate its path. It is easily done. You open facets, open the file `facets_overview` and then `python`. Copy the path



After that, go back to Jupyter, and write the following code. Change the path `'/Users/Thomas/facets/facets_overview/python'` to your path.

```
# Add the facets overview python code to the python path# Add t
import sys
sys.path.append('/Users/Thomas/facets/facets_overview/python')
```

You can import the script with the code below.

```
from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

In windows, the same code becomes


```
import sys
sys.path.append(r"C:\Users\Admin\Anaconda3\facets-
master\facets_overview\python")

from generic_feature_statistics_generator import
GenericFeatureStatisticsGenerator
```

To calculate the feature statistics, you need to use the function `GenericFeatureStatisticsGenerator()`, and you use the object `ProtoFromDataFrames`. You can pass the data frame in a dictionary. For instance, if we want to create a summary statistic for the train set, we can store the information in a dictionary and use it in the object `ProtoFromDataFrames`

- `'name': 'train', 'table': df_train`

Name is the name of the table displays, and you use the name of the table you want to compute the summary. In your example, the table containing the data is `df_train`

```
# Calculate the feature statistics proto from the datasets and
stringify it for use in facets overview
import base64

gfsg = GenericFeatureStatisticsGenerator()

proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train},
                                {'name': 'test', 'table':
df_test}])

#proto = gfsg.ProtoFromDataFrames([{'name': 'train', 'table':
df_train}])
protostr = base64.b64encode(proto.SerializeToString()).decode("utf-
8")
```

Lastly, you just copy and paste the code below. The code comes directly from GitHub. You should be able to see this:



```
# Display the facets overview visualization for this data# Displ
from IPython.core.display import display, HTML

HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-
dist/facets-jupyter.html" >
    <facets-overview id="elem"></facets-overview>
    <script>
        document.querySelector("#elem").protoInput = "
{protostr}";
    </script>"""
html = HTML_TEMPLATE.format(protostr=protostr)
display(HTML(html))
```

Graph

After you check the data and their distribution, you can plot a correlation matrix. The correlation matrix computes the Pearson coefficient. This coefficient is bonded between -1 and 1, with a positive value indicates a positive correlation and negative value a negative correlation.

You are interested to see which variables can be a good candidate for interaction terms.

```
## Choose important feature and further check with Dive
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="ticks")
# Compute the correlation matrix
corr = df.corr('pearson')
# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

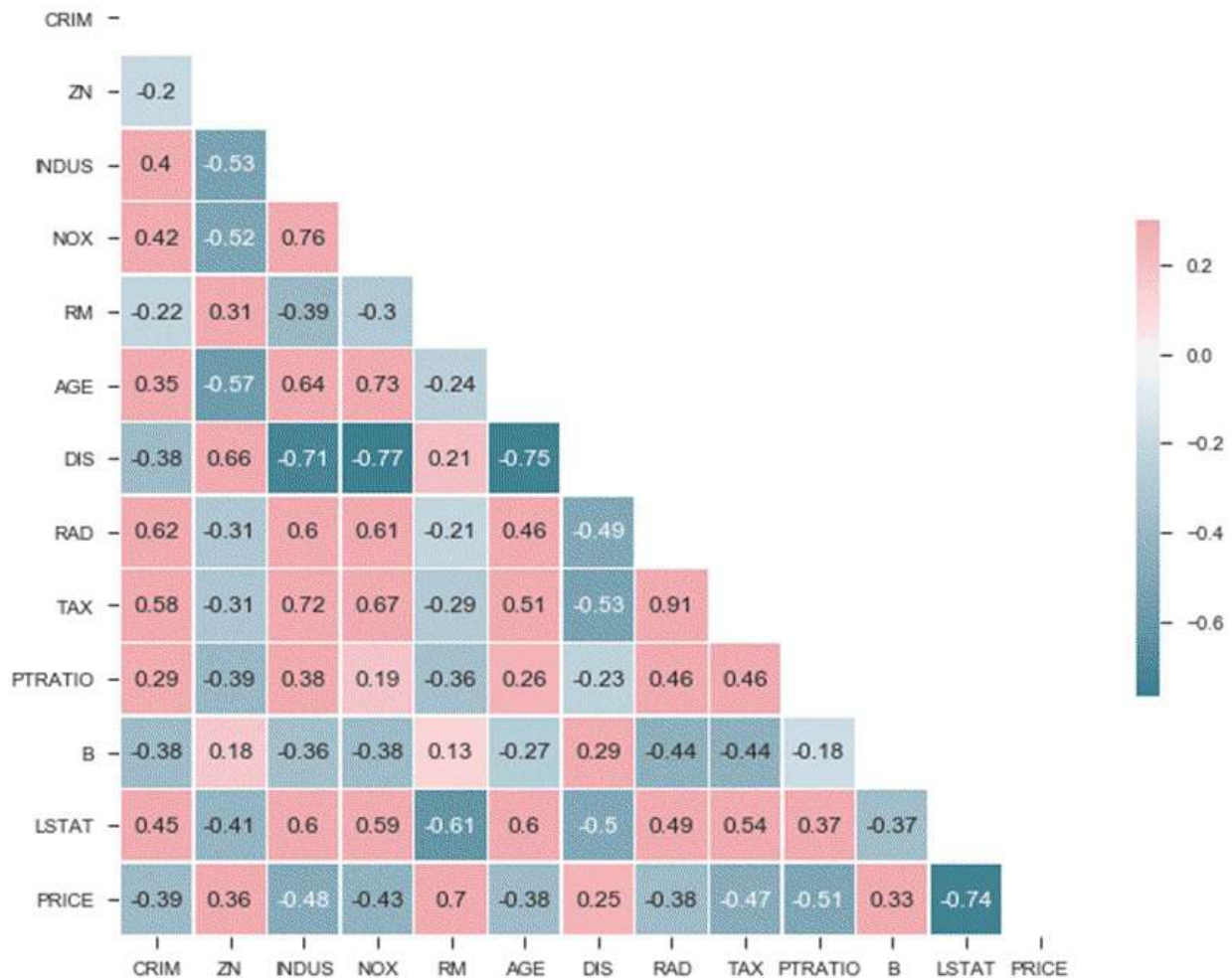
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,
            center=0, annot=True,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

Output

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a184d6518>
```

png



From the matrix, you can see:

- LSTAT
- RM

Are strongly correlated with PRICE. Another exciting feature is the strong positive correlation between NOX and INDUS, which means those two variables move in the same direction. Besides, there are also correlated with the PRICE. DIS is also highly correlated with IND and NOX.

You have some first hint that IND and NOX can be good candidates for the interaction term and DIS might also be interesting to focus on.

You can go a little bit deeper by plotting a pair grid. It will illustrate more in detail the correlation map you plotted before.

The pair grid we are composed as follow:

- Upper part: Scatter plot with fitted line
- Diagonal: Kernel density plot
- Lower part: Multivariate kernel density plot

You choose the focus on four independent variables. The choice corresponds to the variables with strong correlation with PRICE

- INDUS
- NOX
- RM
- LSTAT

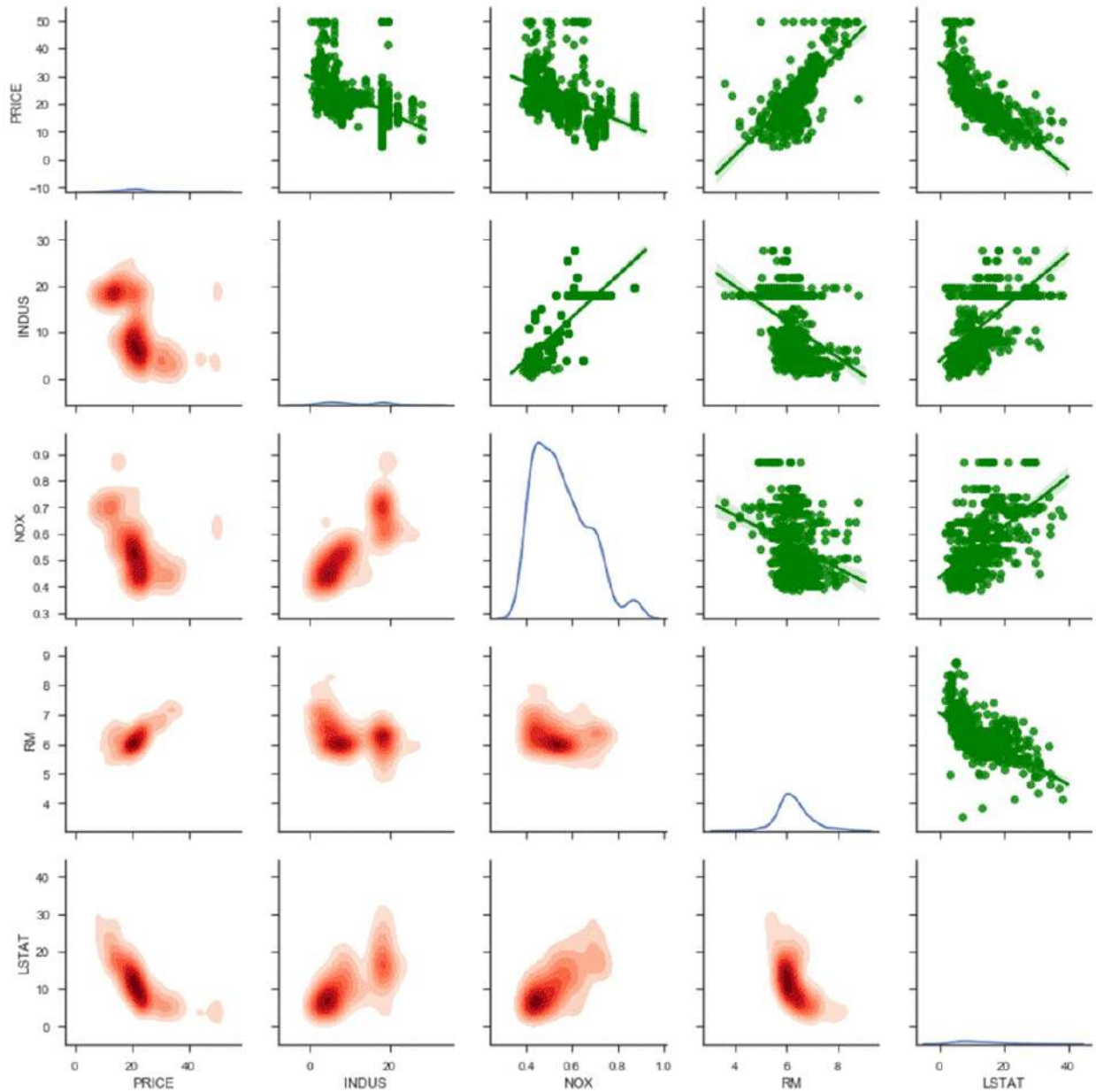
moreover, the PRICE.

Note that the standard error is added by default to the scatter plot.

```
attributes = ["PRICE", "INDUS", "NOX", "RM", "LSTAT"]

g = sns.PairGrid(df[attributes])
g = g.map_upper(sns.regplot, color="g")
g = g.map_lower(sns.kdeplot, cmap="Reds", shade=True,
shade_lowest=False)
g = g.map_diag(sns.kdeplot)
```

Output



Let's begin with the upper part:

- Price is negatively correlated with INDUS, NOX, and LSTAT; positively correlated with RM.
- There is a slightly non-linearity with LSTAT and PRICE
- There is like a straight line when the price is equal to 50. From the description of the dataset, PRICE has been truncated at the value of 50

Diagonal

- NOX seems to have two clusters, one around 0.5 and one around 0.85.

To check more about it, you can look at the lower part. The Multivariate Kernel Density is interesting in a sense it colors where most of the points are. The difference with the scatter plot draws a probability density, even though there is no point in the dataset for a given coordinate. When the color is stronger, it indicates a high concentration of point around this area.

If you check the multivariate density for INDUS and NOX, you can see the positive correlation and the two clusters. When the share of the industry is above 18, the nitric oxides concentration is above 0.6.

You can think about adding an interaction between INDUS and NOX in the linear regression.

Finally, you can use the second tools created by Google, Facets Deep Dive. The interface is divided up into four main sections. The central area in the center is a zoomable display of the data. On the top of the panel, there is the drop-down menu where you can change the arrangement of the data to controls faceting, positioning, and color. On the right, there is a detailed view of a specific row of data. It means you can click on any dot of data in the center visualization to see the detail about that particular data point.

During the data visualization step, you are interested in looking for the pairwise correlation between the independent variable on the price of the house. However, it involves at least three variables, and 3D plots are complicated to work with.

One way to tackle this problem is to create a categorical variable. That is, we can create a 2D plot a color the dot. You can split the variable PRICE into four categories, with each category is a quartile (i.e., 0.25, 0.5, 0.75). You call this new variable Q_PRICE.

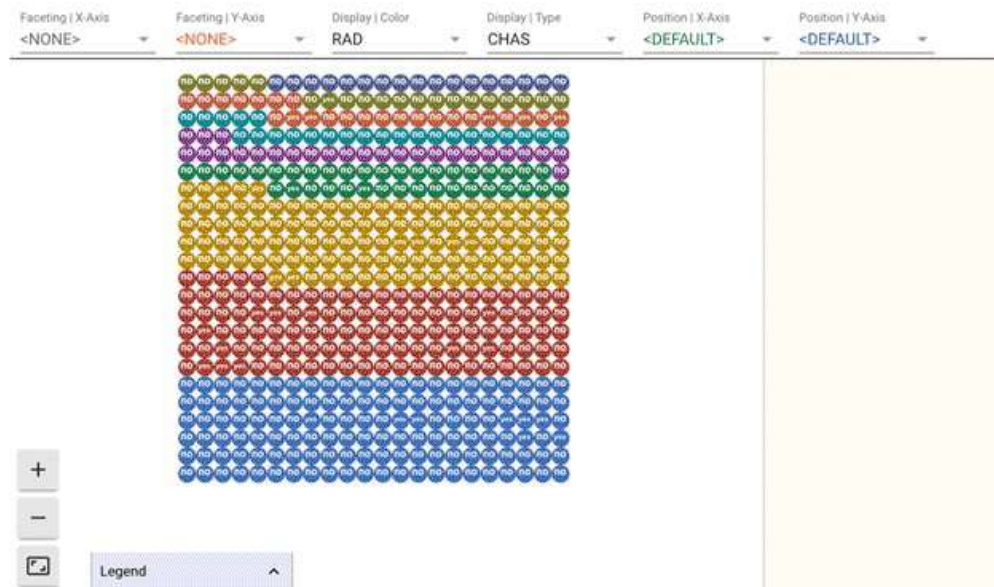
Facets Deep Dive

To open Deep Dive, you need to transform the data into a json format. Pandas as an object for that. You can use `to_json` after the Pandas dataset.

The first line of code handle the size of the dataset.

```
df['Q_PRICE'] = pd.qcut(df['PRICE'], 4, labels=["Lowest", "Low", "Upper", "upper_plus"])
sprite_size = 32 if len(df.index)>50000 else 64
jsonstr = df.to_json(orient='records')
```

The code below comes from Google GitHub. After you run the code, you should be able to see this:



```
# Display thde Dive visualization for this data
from IPython.core.display import display, HTML

# Create Facets template
HTML_TEMPLATE = """<link rel="import" href="/nbextensions/facets-
dist/facets-jupyter.html">
    <facets-dive sprite-image-width="{sprite_size}" sprite-
image-height="{sprite_size}" id="elem" height="600"></facets-dive>
```

```
<script>
  document.querySelector("#elem").data = {jsonstr};
</script>""

# Load the json dataset and the sprite_size into the template
html = HTML_TEMPLATE.format(jsonstr=jsonstr,
  sprite_size=sprite_size)

# Display the template
display(HTML(html))
```

You are interested to see if there is a connection between the industry rate, oxide concentration, distance to the job center and the price of the house.

For that, you first split the data by industry range and color with the price quartile:

- Select faceting X and choose INDUS.
- Select Display and choose DIS. It will color the dots with the quartile of the house price

here, darker colors mean the distance to the first job center is far.

So far, it shows again what you know, lower industry rate, higher price. Now you can look at the breakdown by INDUX, by NOX.

- Select faceting Y and choose NOX.

Now you can see the house far from the first job center have the lowest industry share and therefore the lowest oxide concentration. If you choose to display the type with Q_PRICE and zoom the lower-left corner, you can see what type of price it is.

You have another hint that the interaction between IND, NOX, and DIS can be good candidates to improve the model.

TensorFlow

In this section, you will estimate the linear classifier with TensorFlow estimators API. You will proceed as follow:

- Prepare the data
- Estimate a benchmark model: No interaction
- Estimate a model with interaction

Remember, the goal of machine learning is the minimize the error. In this case, the model with the lowest mean square error will win. The TensorFlow estimator automatically computes this metric.

Preparation data

In most of the case, you need to transform your data. That is why Facets Overview is fascinating. From the summary statistic, you saw there are outliers. Those values affect the estimates because they do not look like the population you are analyzing. Outliers usually biased the results. For instance, a positive outlier tends to overestimate the coefficient.

A good solution to tackle this problem is to standardize the variable. Standardization means a standard deviation of one and means of zero. The process of standardization involves two steps. First of all, it subtracts the mean value of the variable. Secondly, it divides by the variance so that the distribution has a unit variance

The library sklearn is helpful to standardize variables. You can use the module preprocessing with the object scale for this purpose.

You can use the function below to scale a dataset. Note that you don't scale the label column and categorical variables.