INDUSTRIAL TRAINING PROJECT REPORT

ON

IMPLEMENTATION OF NEURAL NETWORKS TO PREDICT PROBABILITY OF SCHEDULE DELAY BY USING DATA MINING AND REGRESSION

Submitted by:

JAGBEER SINGH
1/17/FET/BCS/020

Under the Guidance of

BORIS PASKHAVER.
UDEMY INSTRUCTOR

DR. INDU KASHYAP
PROFESSOR, FET, CSE, MRIIRS

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN
Computer Science & Engineering

Faculty of Engineering & Technology

MANAV RACHNA INTERNATIONAL INSTITUTE OF RESEARCH AND STUDIES, Faridabad
NAAC ACCREDITED 'A' GRADE

Acknowledgement

The successful realization of project is an outgrowth of a consolidated effort of people from desperate fronts. I am thankful to **DR. INDU KASHYAP , Professor, FET(CSE), MRIIRS** for her variable advice and support extended to me without which i could not be able to complete this project for a success.

I express my deep gratitude to **DR. SUPRIYA PANDA** Head of department (CSE) for her endless support and affection towards us. her constant encouragement has helped to widen the horizon of our knowledge and inculcate the spirit of dedication to the purpose.

I would like to express my sincere gratitude to **DR GEETA NIJHAWAN**, ASSOCIATE DEAN FET for providing me the facilities in the institute for completion of my work.

Jagbeer Singh
1/17/FET/BCS/020

Declaration

I hereby declare that this project report entitled “**IMPLEMENTATION OF NEURAL NETWORKS TO PREDICT PROBABILITY OF SCHEDULE DELAY BY USING DATA MINING AND REGRESSION**” *by* **JAGBEER SINGH (1/17/FET/BCS/020),** being submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science and Engineering** under Faculty of Engineering & Technology of Manav Rachna International Institute of Research and Studies, Faridabad, during the academic year DECEMBER ,2020, is a bonafide record of my original work carried out under guidance and supervision of **DR. INDU KASHYAP , Professor, FET(CSE), MRIIRS** and has not been presented elsewhere.

Jagbeer Singh
1/17/FET/BCS/020

Manav Rachna International Institute of Research and Studies, Faridabad

Faculty of Engineering & Technology

Department of Computer Science & Engineering

December, 2020

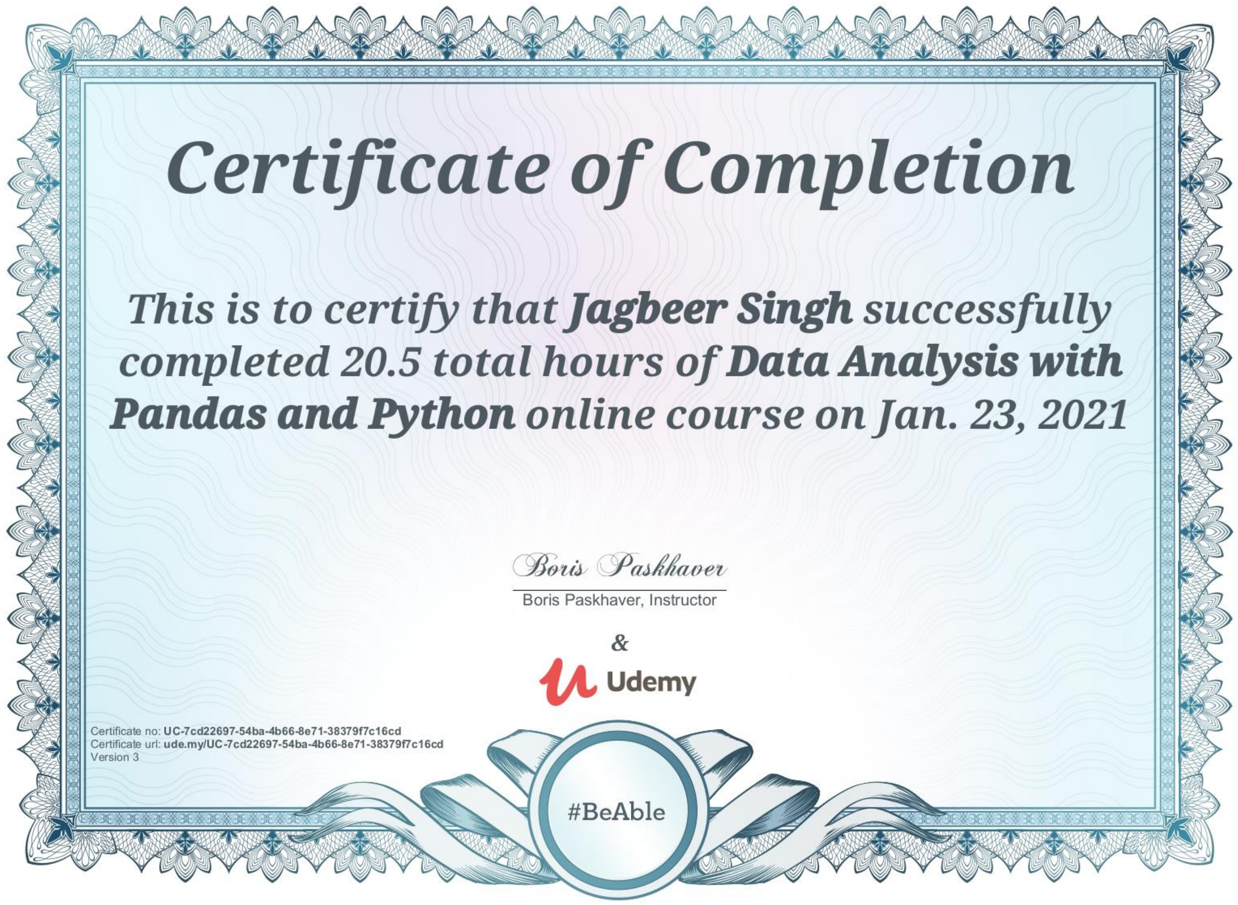
Certificate

This is to certify that this project report entitled "**IMPLEMENTATION OF NEURAL NETWORKS TO PREDICT PROBABILITY OF SCHEDULE DELAY BY USING DATA MINING AND REGRESSION**" *by* **JAGBEER SINGH (1/17/FET/BCS/020),** submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science and Engineering** under Faculty of Engineering & Technology of Manav Rachna International Institute of Research and Studies Faridabad, during the academic year December 2020, is a bonafide record of work carried out under my guidance and supervision.

DR. INDU KASHYAP
Department. of Computer Science & Engineering(IBM)
Faculty of Engineering & Technology
Manav Rachna International Institute of Research and Studies, Faridabad

Dr. Supriya Panda
Professor & Head of Department
Department. of Computer Science & Engineering
Faculty of Engineering & Technology
Manav Rachna International Institute of Research and Studies, Faridabad

<u>Certificate of Completion:</u>



Certificate of Completion

This is to certify that **Jagbeer Singh** successfully completed 20.5 total hours of **Data Analysis with Pandas and Python** online course on Jan. 23, 2021

Boris Paskhaver

Boris Paskhaver, Instructor

&

Udemy

Certificate no: UC-7cd22697-54ba-4b66-8e71-38379f7c16cd
Certificate url: ude.my/UC-7cd22697-54ba-4b66-8e71-38379f7c16cd
Version 3

#BeAble

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

Customer satisfaction is an important activity of the airline. Due to any reason, the late arrival of the plane at the take-off point, flights are delayed and lead to customer dissatisfaction. Flight delay is a really big problem for airlines, airports and passengers hence leaving a negative impact on them. There are two reasons why a flight may be delayed or canceled. One is the factor of the business situation. Another is the factor of the inevitable. Under the circumstances of the company, the airline is responsible for problems with machine parts and system failures. The natural disaster, which involves heavy snowfall or bad weather, is inevitable. Consequently, punctual flight is targeted and punctual flight predictions are developed because flight performance is not affected by the circumstances and the company's inevitability. Conventionally, when the departure time or arrival time of a flight is more than 15 minutes than expected departure or arrival times, it is assumed that there is a departure or arrival delay compared to at the corresponding airports.

Flight prediction project is written in Python. The project file contains a python script and a database file. This is a simple console based system which is very easy to understand and use. The aim of this project is to propose a methodology that can be used to predict flight delays at airports with the help of machine learning. First, we collect realistic raw flight data for airports. The expected departure time at the airport is then set as the prediction target, and individual flight data is processed based on the airport-related aggregate characteristics for prediction modeling. Finally, some machine learning methods have been explored to improve the predictability and accuracy of the proposed model.

1.1 Proposed System

The proposed system allows us to observe a preexisting flight data to generate a machine learning model which has capability to predict the schedules for upcoming flights.

This system has the ability to predict the delay in a scheduled flight with a precision of 99%.
Our system is solely a team work to understand various aspects of the data prediction where we have used regressions for our purpose.

1.2 Objectives

The objective of our project is to import as preexisting data and generate a ML model over it to predict the delay in scheduled flight

1.1 Literature Review

In the past, researchers have attempted to predict flight delays using machine learning, deep learning, and big data.

• Review of machine learning techniques:

- Chakrabarty et al. proposed a machine learning model that uses the Gradient Booster Classifier to predict American Airlines arrival delays at the 5 busiest airports in the United States. USA

- Manna et al. Reviewed and analyzed flight data and developed a regression model using the gradient escalator to predict flight departure and arrival delays

-Ding et al. proposed a multiple linear regression approach to predict flight delay and also compared model performance with the Naive Bayes and C4.5 approaches.

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

It does not need any additional hardware or software to operate the program but the following requirements should be strongly maintained:

512MB of RAM or Higher
CD ROM
20 MB of Hard Disk Space
800MHz processor or above

Processor: Intel Core

The following requirements should be strongly maintained:

Operating System WINDOWS 7 or Higher
Program PYTHON 2 or Higher needs to be installed.

TECHNOLOGY USED

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

CHAPTER 2. *Artificial Neural Network (ANN)*

An Artificial Neural Network has hundreds, or thousands of artificial neurons called processing units, which are interconnected by nodes. These processing units are made up of input and output units. The input units receive various forms and structures of information based on an internal weighting system, and the neural network attempts to learn about the information presented to produce one output report. Just like humans need rules and guidelines to come up with a result or output, ANNs also use a set of learning rules called backpropagation, an abbreviation for backward propagation of error, to perfect their output results.

An ANN initially goes through a training phase where it learns to recognize patterns in data, whether visually, aurally, or textually. During this supervised phase, the network compares its actual output produced with what it was meant to produce—the desired output. The difference between both outcomes is adjusted using backpropagation. This means that the network works backward, going from the output unit to the input units to adjust the weight of its connections between the units until the difference between the actual and desired outcome produces the lowest possible error.

So, basically artificial neural network (ANN) is the piece of a computing system designed to simulate the way the human brain analyzes and processes information. It is the foundation of artificial intelligence (AI) and solves problems that would prove impossible or difficult by human or statistical standards. ANNs have self-learning capabilities that enable them to produce better results as more data becomes available.

Artificial neural networks are paving the way for life-changing applications to be developed for use in all sectors of the economy. Artificial intelligence platforms that are built on ANNs are disrupting the traditional ways of doing things. From translating web pages into other languages to having a virtual assistant order grocery online to conversing with chatbots to solve problems, AI platforms are simplifying transactions and making services accessible to all at negligible costs.

Artificial neural networks have been applied in all areas of operations. Email service providers use ANNs to detect and delete spam from a user's inbox; asset managers use it to forecast the direction of a company's stock; credit rating firms use it to improve their credit scoring methods; e-commerce platforms use it to personalize recommendations to their audience; chatbots are developed with ANNs for natural language processing; deep learning algorithms use ANN to predict the likelihood of an event; and the list of ANN incorporation goes on across multiple sectors, industries, and countries.

CHAPTER 3. *Activation function*

Activation function defines the output of input or set of inputs or in other terms defines node of the output of node that is given in inputs. They basically decide to deactivate neurons or activate them to get the desired output. It also performs a nonlinear transformation on the input to get better results on a complex **neural network**.

Activation function also helps to normalize the output of any input in the range between **1 to -1**. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points.

Activation function also helps to normalize the output of any input in the range between **1 to -1**. Activation function must be efficient, and it should reduce the computation time because the neural network sometimes trained on millions of data points.

Activation function basically decides in any neural network that given input or receiving information is relevant or it is irrelevant.
The neuron is basically is a weighted average of input, then this sum is passed through an activation function to get an output.

So, we have to bound our output to get the desired prediction or generalized results. We pass that neuron to activation function to bound output values.

Without activation function, weight and bias would only have a linear transformation, or neural network is just a linear regression model, a linear equation is polynomial of one degree only which is simple to solve but limited in terms of ability to solve complex problems or higher degree polynomials.

But opposite to that, the addition of activation function to neural network executes the non-linear transformation to input and make it capable to solve complex problems such as language translations and image classifications.

In addition to that, Activation functions are differentiable due to which they can easily implement back propagations, optimized strategy while performing backpropagations to measure gradient loss functions in the neural networks.

CHAPTER 4. ***Neural Network working and learning***

How does artificial neural networks work?

Artificial Neural Networks can be best viewed as weighted directed graphs, where the nodes are formed by the artificial neurons and the connection between the neuron outputs and neuron inputs can be represented by the directed edges with weights. The Artificial Neural Network receives the input signal from the external world in the form of a pattern and image in the form of a vector. These inputs are then mathematically designated by the notations x(n) for every n number of inputs.

Each of the input is then multiplied by its corresponding weights (these weights are the details used by the artificial neural networks to solve a certain problem). In general terms, these weights typically represent the strength of the interconnection amongst neurons inside the artificial neural network. All the weighted inputs are summed up inside the computing unit (yet another artificial neuron).

If the weighted sum equates to zero, a bias is added to make the output non-zero or else to scale up to the system’s response. Bias has the weight and the input to it is always equal to 1. Here the sum of weighted inputs can be in the range of 0 to positive infinity. To keep the response in the limits of the desired value, a certain threshold value is benchmarked. And then the sum of weighted inputs is passed through the activation function.

The activation function, in general, is the set of transfer functions used to get the desired output of it. There are various flavors of the activation function, but mainly either linear or non-linear sets of functions. Some of the most commonly used set of activation functions are the Binary, Sigmoidal (linear) and Tan hyperbolic sigmoidal (non-linear) activation functions.

How do Neural Networks Learn?

If we observe, we can see that systems that are able to learn are highly adaptable. In their quest to acquire knowledge, these systems use input from the outside world and modify information that they've already collected or modify their internal structure. That is exactly what ANNs do. They adapt and modify their architecture in order to learn. To be more precise, the ANNs change weights of connections based on input and desired output.

If we look closer into the structure of the ANNs, there are a few components we could change inside of the ANN if we want to modify their architecture. For example, we could create new connections among neurons, or delete them, or add and delete neurons. We could even modify input function or activation function. As it turns out, changing weights is the most practical approach. Plus, most of the other cases could be covered by changing weights. Deleting a connection, for example, can be done by setting the weight to 0. And a neuron can be deleted if we set weights on all its connections to zero. Learning is a necessary process for every ANN, and it is a process in which the ANN gets familiar with the problem it needs to solve. In practice, we usually have some collected data based on which we need to create our predictions, or classification, or any other processing. This data is called training set. In fact, based on behavior during the training and the nature of training set, we have a few classes of learning:

Unsupervised learning – Training set contains only inputs. The network attempts to identify similar inputs and to put them into categories. This type of learning is biologically motivated, but it is not suitable for all the problems.

Reinforcement learning – Training set contains inputs, but the network is also provided with additional information during the training. What happens is that once the network calculates the output for one of the inputs, we provide information that indicates whether the result was right or wrong and possibly, the nature of the mistake that the network made.

Supervised learning – Training set contains inputs and desired outputs. This way the network can check its calculated output the same as desired output and take appropriate actions based on that. Once the network calculates the output for one of the inputs, cost function calculates the error vector. This error indicates how close our guess is to the desired output.

Chapter 5. ***Gradient Descent***

The most used algorithm to train neural networks is gradient descent. The gradient is a numeric calculation allowing us to know how to adjust the parameters of a network in such a way that its output deviation is minimized.

The algorithm has several versions depending on the number of samples that we introduce to the network for each iteration:

Batch gradient descent: all available data is injected at once. This version implies a high risk of getting stuck, since the gradient will be calculated using all the samples, and the variations will be minimal sooner or later. As a general rule: for a neural network it's always positive to have an input with some randomness.

Stochastic gradient descent: a single random sample is introduced on each iteration. The gradient will be calculated for that specific sample only, implying the introduction of the desired randomness, and making more difficult the possibility of getting stuck. The main problem with this version is its slowness, since it needs many more iterations. Additionally, it doesn't take advantage of the available computing resources.

Mini-Batch gradient descent: instead of feeding the network with single samples, N random items are introduced on each iteration. This preserves the advantages of the second version and also getting a faster training due to the parallelization of operations. We choose a value for N that provides a good balance between randomness and training time being not too large for the available GPU memory.

These are the iterations of the Gradient Descent algorithm using its third version:

1) We introduce a mini-batch input with N random samples from our previously labeled training dataset (we have the actual output).

2) After the appropriate calculations on each layer of the network, we obtain the predictions at the output. This step is known as forward propagation (of the input).

3)We evaluate the loss function for the mini batch. It’s only a previously chosen mathematical function, based on our specific problem type, and used to evaluate the difference between the predictions of our network and the actual labels, in a proper way. The value of this loss function is the number the algorithm is trying to minimize, and the following steps are oriented towards this.

We calculate the gradient as the multi-variable derivative of the loss function with respect to all the network parameters. Graphically it would be the slope of the tangent line to the loss function at the current point evaluating the current parameter values. Mathematically it’s a vector that gives us the direction in which the loss function increases faster, so we should move in the opposite direction if we try to minimize it.

Chapter 6. *Backpropagation*

Back-propagation is the essence of neural net training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous iteration. Proper tuning of the weights allows you to reduce error rates and to make the model reliable by increasing its generalization.

Backpropagation is a short form for "backward propagation of errors." It is a standard method of training artificial neural networks. This method helps to calculate the gradient of a loss function with respects to all the weights in the network.

Static Backpropagation:

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Back Propagation:

Recurrent backpropagation is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is that the mapping is rapid in static backpropagation while it is non static in recurrent backpropagation.

- Simplifies the network structure by elements weighted links that have the least effect on the trained network

- A group of input and activation values to develop the relationship between the input and hidden unit layers.
- It helps to assess the impact that a given input variable has on a network output. The knowledge gained from this analysis should be represented in rules.
- Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition.
- Backpropagation takes advantage of the chain and power rules allows backpropagation to function with any number of outputs.
- Backpropagation is fast, simple and easy to program
- It has no parameters to tune apart from the numbers of input
- It is a flexible method as it does not require prior knowledge about the network
- It is a standard method that generally works well
- It does not need any special mention of the features of the function to be learned.

Chapter 7. <u>***what is Theano?***</u>

Theano

About

Theano is a well-known Python library. Developed to optimize compilers to manipulate and evaluate mathematical expressions like matrix-valued ones. In Theano, computations are expressed using a NumPy-esque syntax and are compiled to run adeptly on either CPU or GPU architectures.

Theano is an Open source roject primarily developed by a Montreak Institute for Learning Algorithm (MILA) at the University of Montreal.

The name of the software references is taken from the ancient philosopher Theano, Longley associated with development of the Golden mean.

On the day of 28 September 2017, Pascal Lamblin posted a message from Yoshua Benjio, Head of MILA: Major development would cease after the 1.0 release due to competing offerings by strong industrial players. 15 November 2017 was the day when Theano 1.0.0 was finally released.

On 17 May 2018, Chris Fonnesbeck posted on behalf of the PyMC development team that the PyMC developers will officially assume control of Theano maintenance once they step down.

Optimisation

Theano’s compiler applies many optimizations of varying complexity to these symbolic expressions. These optimizations include, but are not limited to:

- For computation GPU is used
- constant folding
- to avoid redundant calculation, merging of similar subgraphs is done
- arithmetic simplification (e.g. x*y/x -> y, --x -> x)
- inserting efficient BLASoperations (e.g. GEMM) in a variety of contexts
- To avoid calculations memory aliasing is done.
- In place operations are used wherever it does not interfere with aliasing
- loop fusion for elementwise sub-expressions
- improvements to numerical stability (e.g. $\log(1+\exp(x))$and $\log(\sum_i \exp(x[i]))$)

Main Features

Theano is a well-known Python library. Developed to optimize compilers to manipulate and evaluate mathematical expressions like matrix-valued ones. Manipulation of matrices is basically done using the numpy package, so why is Theano different and used in place of Python and numpy?

- *speed optimizations during execution*: Theano uses *g++* and *nvcc* to compile parts in your expression graph in GPU instructions, which runs much faster than simple Python.
- *symbolic differentiation*: Theano automatically builds symbolic graphs in use for computing gradients.
- *stability optimizations*: Theano is well known for recognizing some numerically unstable expressions and computing them with a little more stable and sophisticated algorithm.

The closest Python package to Theano is sympy. While Theano focuses more on tensor expressions than Sympy and has a little more machinery for compilation. Whereas Sympy has more sophisticated algebra rules and can also handle a wider variety of mathematical operations like series, integers, and limits.

If we go comparing numpy to MATLAB and sympy to Mathematica, Theano will be hybrid of the two which tries to combine best of both worlds.

API of Theano

This is well suited to find the Types and Ops that are used to build and compile expression graphs.

- Compile- Expression graphs to function transformation.
- Config- configuring Theano
- D3viz- Theano compute graph Interactive visualization
- Gof- Theano internals
- Gpuarray- It is the new backend of GPU
- Gradient- Symbolic Differentiation
- Misc.pkl.utils- Serialization tools.
- Printing- Symbolic print and graph printing statement
- Sandbox- Experimental code
- Scalar- Symbolic scalar types, Ops [doc TODO]
- Scan- Looping in Theano
- Sparse- Symbolic Sparse Matrices
- Sparse- Sparse Op
- Sparse.sandbox- Sandbox Sparse Op
- Tensor- Types and Ops for symbolic numpy
- Typed.list- Typed list
- Tests- Tests

Chapter 8. ***what is Keras?***

About

Keras was developed as a neural network API. It is a library written specifically for Python (I am sure it will be used with R and Javascript eventually.). It works with other libraries and packages such as TensorFlow which makes deep learning easier. Keras was developed to allow fast prototyping and a quick experimentation.

From what I understand, Keras promotes allows for modules to be constructed and can be integrated with previous code or used as a base for a new project. Keras runs on both GPU and CPU of any computer or laptop device. It supports recurrent and convolutional neural networks.

If you have a package manager such as Chocolatey (Windows), pip install (Linux and Mac), or Home-brew (Mac), it should be easy to install Keras. Otherwise you can clone the repository (Go open source!) and install it using setup.py. Making sure that Keras is installed is as easy as writing a Python program with the line “import keras” or “from keras.model import Sequential.” From the documentation, Sequential is supposed to create a stack of layers. I guess it helps with parallelization if neural networks need that. If the program runs, then the module is available for use.

Why this name keras?

Keras (κέρας) is Greek meaning “*horn*”. It is derived from ancient Greek and Latin literature, firstly looked in the *Odyssey*, where was a dream spirits (*Oneiroi*, singular *Oneiros*) which are divided among those with deceive dreamers with a false sight, who reach Earth through a gate of ivory, and those who predict a future that will come to pass, who arrive via a gate of horn. It is simply a play of words κέρας (horn) / κραίνω (fulfill), and ἐλέφας (ivory) / ἐλεφαίρομαι (deceive).

ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) was the project on which Keras was developed.

Keras Principles

Keras was found to be user friendly, modular and to work on Python. The API was "made for humans, not machines," and "follows best practices in reducing load."

Neural layers, cost function, optimizer, initialization schemes, activation functions, and regularization schemes all are stand alone modules that can be combined to make new models. New modules are easy to add, as new classes and functions. Models are made in Python code, not on separate model configuration files.

Keras Backend

Keras proper don't do its own lower-level operations, like tensor products and convolutions; it depends itself on back-end engine for that. Keras works with a lot of back-end engines but its primary back-end is TensorFlow and is mainly supported by Google. Keras API comes packaged in TensorFlow as tf.keras, which like mentioned earlier will become the primary TensorFlow API as of TensorFlow 2.0.

In case if you want to edit back-ends, just change your $HOME/.keras/keras.json file and mention a different back-end name, say as theano or CNTK. Other than this you can also override the configured back-end file by defining environment variable KERAS_BACKEND, either in your python code or in your shell using a property named os.environ["KERAS_BACKEND"].

Keras Layers

Core layers include Dense (dot product plus bias), Activation (transfer function or neuron shape), Dropout (randomly set a fraction of input units to 0 at each training update to avoid overfitting), Lambda (wrap an arbitrary expression as a Layer object), and several others. Convolution layers (the use of a filter to create a feature map) run from 1D to 3D and including most common variants, such as transposed and cropping convolution layers for every dimensionality. 2D convolution, which came by the functionality of the visual cortex, is mainly used for image recognition.

Pooling (downscaling) layers runs from 1D to 3D including the most common variants, such as average and mac pooling. Locally connected layers act like convolution layers, except that the weights are unshared. Recurrent layers include simple (fully connected recurrence), gated, LSTM, and others; which are useful for language processing different other applications. Noise layers avoids overfitting.

Keras datasets

There are 7 main deep learning sample datasets which are supplied my Keras via keras.datasets class. Which includes cifar100 and cifar10 small colour images, Reuters newswire topics, MNIST hand written digits IMDB reviews, Boston housing prices and MNIST fashion images.

Keras Applications and Examples

10 well known models are supplied by Keras, which are known as Keras Applications, pretrained against ImageNet: Xception, VGG16, VGG19, ResNet50, InceptionV3, InceptionResNetV2, MobileNet, DenseNet, NASNet, MobileNetV2TK. These are used to predict the classification of images, fine tuning of models and extracting features from them, on different set of classes.

Fine-tuning existing models is a really good way to speed up training. For say, you can add as many layers as you wish, freezing the base layers to train a new layer, then unfreezing some of the base layers to fine-tune the training. Layer freezing is done by setting layer.trainable = False.

WHAT IS TENSOR FLOW

About :

TensorFlow is a free software library focused on machine learning created by Google. Initially released as part of the Apache 2.0 open-source license, TensorFlow was originally developed by engineers and researchers of the Google Brain Team, mainly for internal use.

Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning (aka neural networking) models and algorithms and makes them useful by way of a common metaphor .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.

History : 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.

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.

Hardware and Software requirements:

TensorFlow 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.

Components:

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.

Popularity:

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 Tensorboad. 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.

Chapter 9. Regression and its types

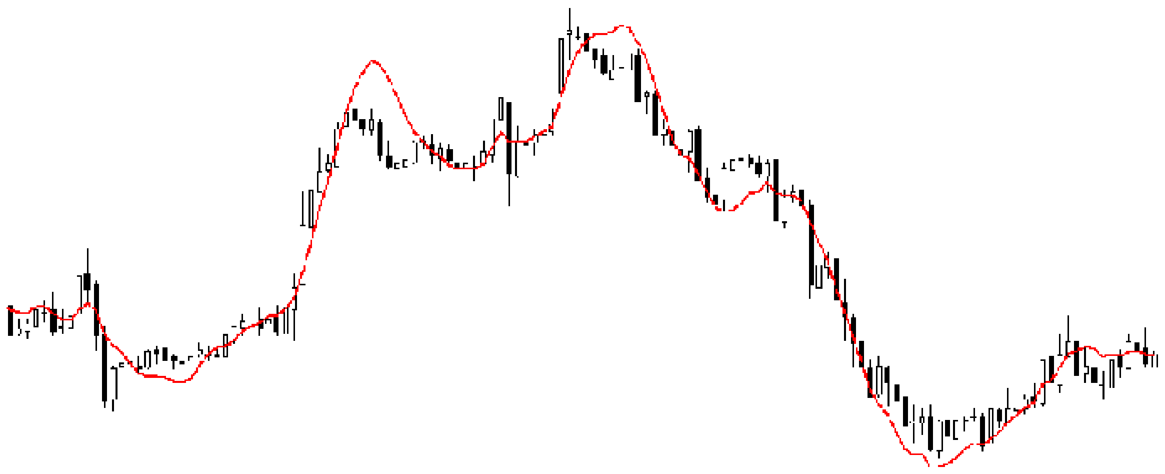
About

Logistic and Linear regressions are basically the first two algorithms learnt by people in data science. Because of their huge popularity, many of the analysts might end up thinking that those are the only forms of regressions. The ones which are a little more involved than others, start thinking that they are the most important among all other forms of regression analysis.

But the truth is that there are innumerable forms of regressions, which are being used these days. Each one has its own importance and a specific condition where they are best suited to apply.

Regression analysis

It is a form of predictive modelling technique (PMT) which is used in investigating the relation between a **dependent** and **independent variable (s)**. The technique is mainly used for time series modelling, forecasting, and finding a casual effect relationship between the various variables. For say, relationship between rash drivers and several road accidents by a driver is best studied through regression.



Regression analysis is a very important tool for modelling and analysing data. In Here, we fit a curve or a line to the data points, in such a way that the differences among the distances of data points from the line or a curve is minimized.

Types of regression techniques

There are several regression techniques which are used in making predictions. These are mostly driven by 3 metrics (type of dependent variable, number of independent variables, and the very shape of regression line).

The most used regression Techniques are:

1. Linear Regression

Linear regression is the most used basic type of regression in machine learning. The LR model has a dependent variable and a predictor variable related linearly to each other. If the data involves more than one independent variable, then Linear regression becomes multiple linear regression model.

y=mx+c+e

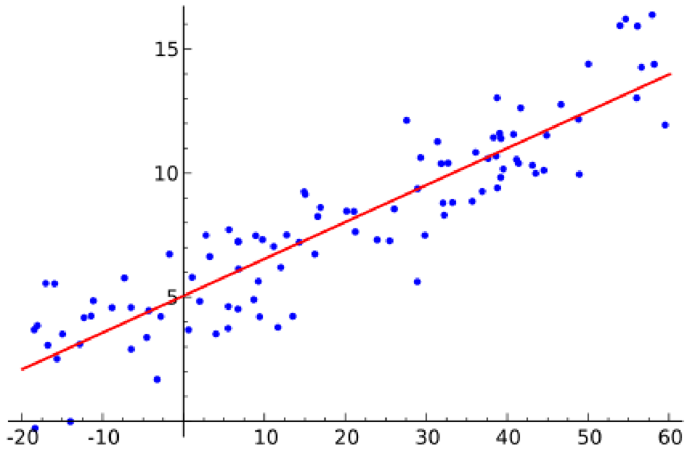
above is the equation used to denote linear regression model.

Where

m is slope of the line

c is an intercept,

and e represents the error in the model.



The best fit line is determined by varying the values of m and c. The predictor error is the difference between the observed values and the predicted value. The values of m and c gets selected in such a way that it generates the minimum predictor error.

2. Logical regression

Logistic regression is another type of regression analysis technique, which is used when the dependent variable is discrete. Example: 0 or 1, true or false, etc.

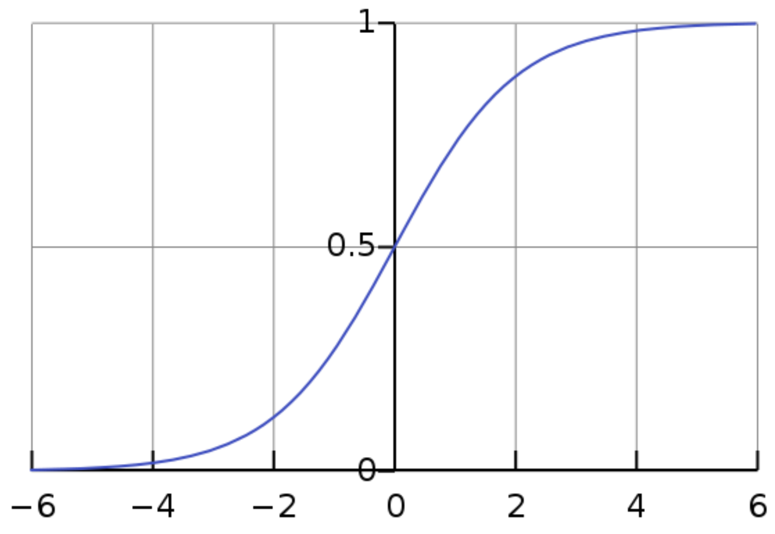
It means that the target variable should only have two values, and a sigmoid curve denoting the relation between the target variable and the independent variable.

Logic function is used in Logistic Regression to check the relationship between the independent variables and target variable.

Below is the equation that denotes the logistic regression.

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3….+bkXk

where p is the probability of occurrence of the feature.



For selecting logistic regression, as the regression analyst technique, it should be noted, the size of data is large with the almost equal occurrence of values to come in target variables. Also, there should be no multicollinearity, which means that there should be no correlation between independent variables in the dataset.

3. Stepwise regression

This form of regression is used when we are dealing with multiple independent variables. Automatic process is used for the selection of independent variables, which involves *no* human intervention.

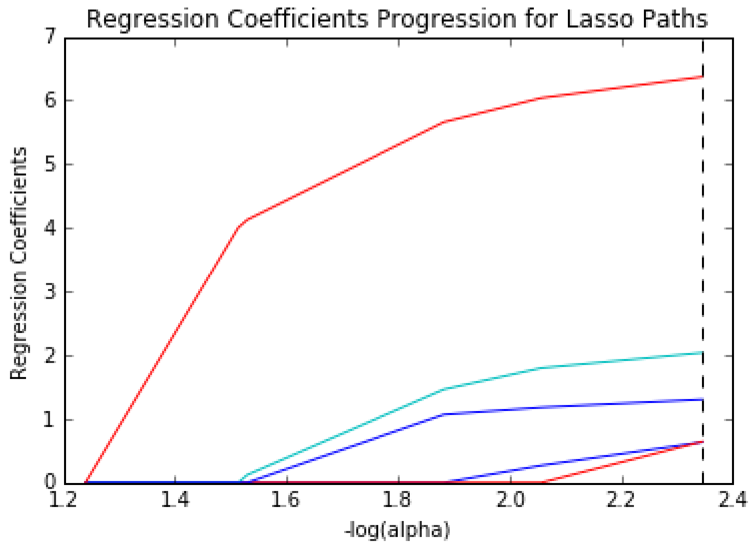
This feat is achieved by observing statistical values like R-square, t-stats and AIC metric to discern significant variables. This type of regression basically fits the regression model by adding/dropping co-variates one at a time based on a specified criterion. Some of the most used Stepwise regression methods are listed below:

- Standard stepwise regression- It does two things, Adding and removing predictors as needed for each step.
- Forward selection- It starts with most significant predictor in which the model adds variable for each step.
- Backward elimination- It starts with all predictors in the model and removes the least significant variable by each step.

4. Lasso Regression

It is that type of regression technique used in ML which performs regularization along with the feature selection. It prohibits the absolute size of the regression coefficient. Resulting, the coefficient value to get nearer to zero, which not happens in the case of Ridge Regression.

Because of this, feature selection gets used in Lasso Regression, it allows selecting a set of features from the dataset in building of a model. In the case of Lasso Regression, only the required features are used, and the other ones are made zero. Which helps in avoiding the overfitting of the model. In case the independent variables are highly collinear, then Lasso regression picks only one variable and makes other variables to shrink to zero.



N^{-1}Σ^{N}_{i=1}f(x_{i}, y_{I}, α, β)

Above is the equation representing Lasso Regression method.

How to choose right regression model?

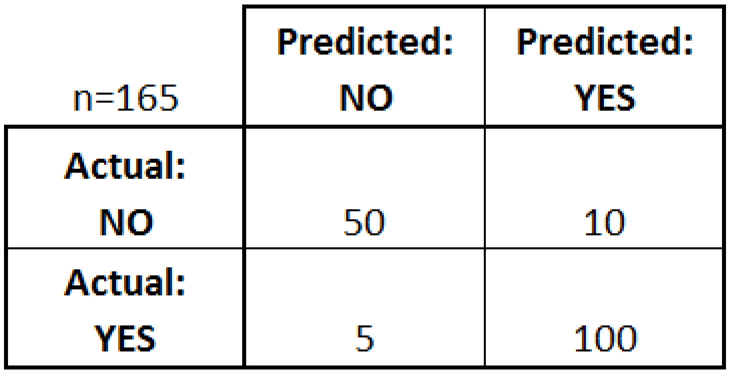
Life is usually simple when we acknowledge only one or two techniques. Among the training institutes I know of telling their students – if outcome is continuous – go with linear regression. If outcome is binary – try logistic regression! Although, the higher number of options today which are available at our disposal, more difficulty happens in choosing the right one. Same happens with regression models. Among multiple types of regression models, it is most important to choose the right one based on the type of independent and dependent variables, dimensionality in the data and other essential characteristics of the data.

Chapter 11. Confusion Matrix

A confusion matrix is a table mainly used to describe the performance of a classifier on a set of testing data for the known true values. Confusion matrix is easily understandable, but the related terminology can be a little confusing.

Let us start with some **example,**

Confusion matrix for a binary classifier (though it can easily be extended to the case of more than two classes):



n=165	**Predicted: NO**	**Predicted: YES**
Actual: NO	50	10
Actual: YES	5	100

What to learn from this matrix?

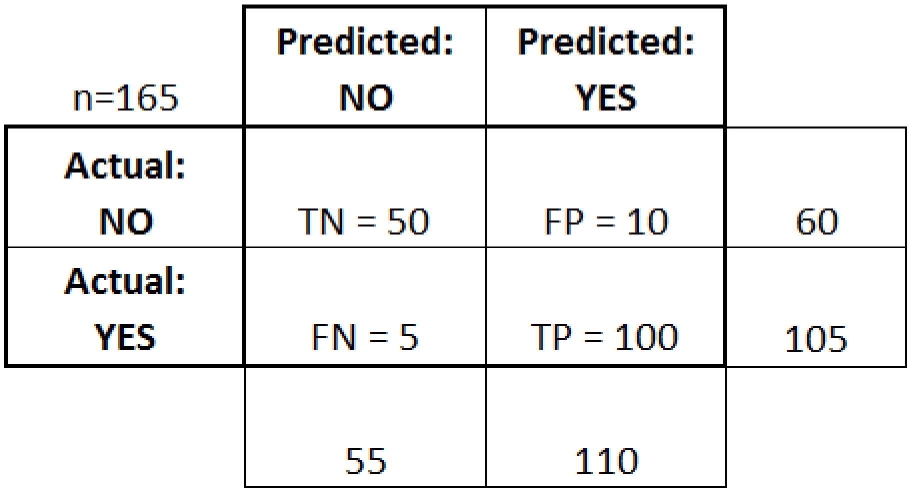
- Two possible predicted classes exist: "yes" and "no". If we were to predict the presence of a disease, for say, "yes" would mean that they have a disease, and "no" would mean that they don't have a disease.
- The classification model made a total of 165 predictions (e.g., 165 patients were being tested for the presence of that disease).

- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- 105 patients in the sample have the disease, and 60 patients do not.

Let us now define the most basic terms, which are whole numbers (not rates):

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they do not have the disease.
- **false positives (FP):** We predicted yes, but they do not actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they do have the disease. (Also known as a "Type II error.")

I have added these terms to the confusion matrix, and added the row and column totals:



n=165	**Predicted: NO**	**Predicted: YES**	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

This is a list of rates that are often computed from a confusion matrix for a binary classifier:

- **Accuracy:** Overall, how often is the classifier correct?
    - (TP+TN)/total = (100+50)/165 = 0.91
- **Misclassification Rate:** Overall, how often is it wrong?
    - (FP+FN)/total = (10+5)/165 = 0.09
    - equivalent to 1 minus Accuracy
    - also known as "Error Rate"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
    - TP/actual yes = 100/105 = 0.95
    - also known as "Sensitivity" or "Recall"
- **False Positive Rate:** When it's actually no, how often does it predict yes?
    - FP/actual no = 10/60 = 0.17
- **True Negative Rate:** When it's actually no, how often does it predict no?
    - TN/actual no = 50/60 = 0.83
    - equivalent to 1 minus False Positive Rate
    - also known as "Specificity"
- **Precision:** When it predicts yes, how often is it correct?
    - TP/predicted yes = 100/110 = 0.91
- **Prevalence:** How often does the yes condition actually occur in our sample?
    - actual yes/total = 105/165 = 0.64

Implementation and Result

CODE

```
!curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv
```

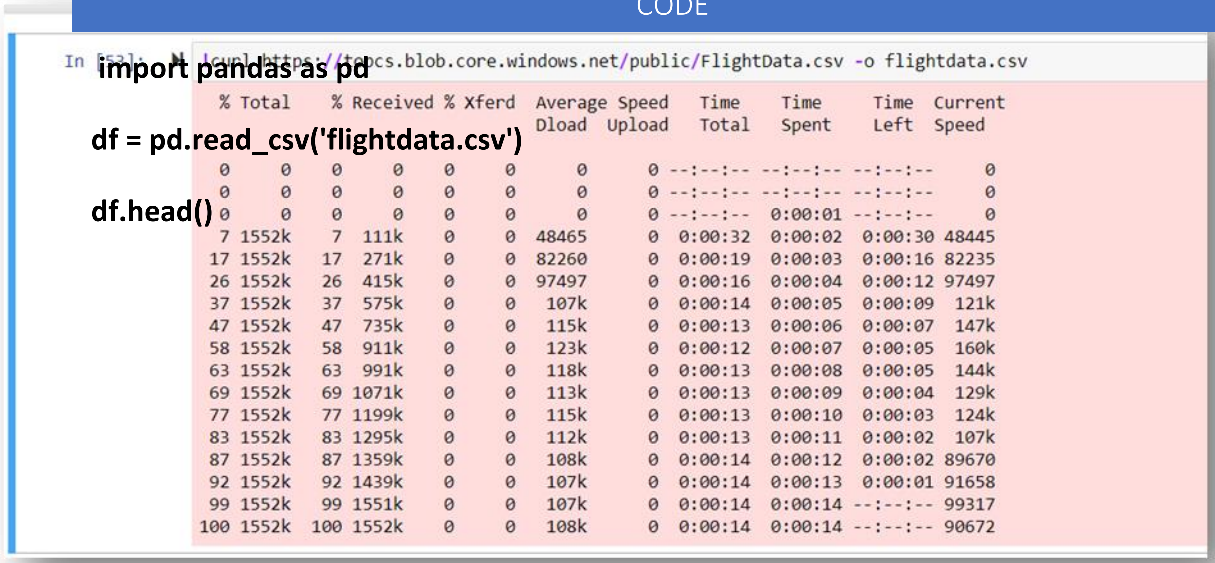
In this module, we have:

- Created a Jupyter notebook , imported data, and viewed data loaded into the notebook.
- Used Pandas to clean and prepare data to be used for the machine-learning model.
- Used scikit-learn to create the machine learning model.
- Used Matplotlib to visualize the model's performance.

curl is a Bash command. we can execute Bash commands in a Jupyter notebook by prefixing them with an exclamation mark. This command downloads a CSV file from Azure blob storage and saves it using the name flightdata.csv.

CODE

```
import pandas as pd

df = pd.read_csv('flightdata.csv')

df.head()
```



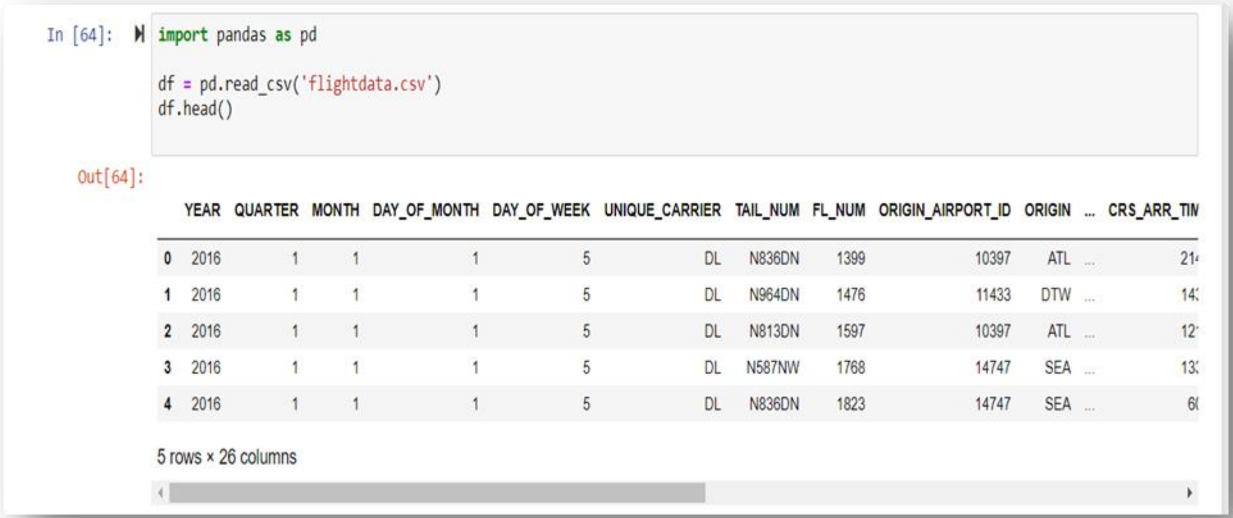
```
In [53]:  !curl https://topcs.blob.core.windows.net/public/FlightData.csv -o flightdata.csv

   % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                  Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
  7 1552k    7  111k    0     0  48465      0  0:00:32  0:00:02  0:00:30 48445
 17 1552k   17  271k    0     0  82260      0  0:00:19  0:00:03  0:00:16 82235
 26 1552k   26  415k    0     0  97497      0  0:00:16  0:00:04  0:00:12 97497
 37 1552k   37  575k    0     0   107k      0  0:00:14  0:00:05  0:00:09  121k
 47 1552k   47  735k    0     0   115k      0  0:00:13  0:00:06  0:00:07  147k
 58 1552k   58  911k    0     0   123k      0  0:00:12  0:00:07  0:00:05  160k
 63 1552k   63  991k    0     0   118k      0  0:00:13  0:00:08  0:00:05  144k
 69 1552k   69 1071k    0     0   113k      0  0:00:13  0:00:09  0:00:04  129k
 77 1552k   77 1199k    0     0   115k      0  0:00:13  0:00:10  0:00:03  124k
 83 1552k   83 1295k    0     0   112k      0  0:00:13  0:00:11  0:00:02  107k
 87 1552k   87 1359k    0     0   108k      0  0:00:14  0:00:12  0:00:02 89670
 92 1552k   92 1439k    0     0   107k      0  0:00:14  0:00:13  0:00:01 91658
 99 1552k   99 1551k    0     0   107k      0  0:00:14  0:00:14 --:--:-- 99317
100 1552k  100 1552k    0     0   108k      0  0:00:14  0:00:14 --:--:-- 90672
```

Importing a dataset

In the notebook's second cell, we entered the following Python code to load flightdata.csv, and create a Pandas DataFrame from it, and displayed the first five rows.

CODE

df.shape



```
In [64]:  import pandas as pd

          df = pd.read_csv('flightdata.csv')
          df.head()
```

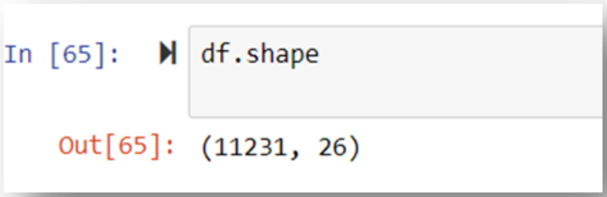
```
Out[64]:
   YEAR  QUARTER  MONTH  DAY_OF_MONTH  DAY_OF_WEEK  UNIQUE_CARRIER  TAIL_NUM  FL_NUM  ORIGIN_AIRPORT_ID  ORIGIN  ...  CRS_ARR_TIM
0  2016        1      1             1            5              DL    N836DN    1399              10397     ATL  ...           21
1  2016        1      1             1            5              DL    N964DN    1476              11433     DTW  ...           14
2  2016        1      1             1            5              DL    N813DN    1597              10397     ATL  ...           12
3  2016        1      1             1            5              DL    N587NW    1768              14747     SEA  ...           13
4  2016        1      1             1            5              DL    N836DN    1823              14747     SEA  ...            6

5 rows × 26 columns
```

Loading the dataset

The Data Frame which we have created contains information regarding many US airlines., after visualising it we get to know that it has more than 26 columns and 11,000 rows.

Every single row represents a single flight and provides information regarding the flight such as its origin, its destination, and its scheduled times.

The previously added codes helped in creating a data frame from the imported file. To know about how many rows and columns in this data frame we can use the below code:



```
In [65]:  df.shape

Out[65]:  (11231, 26)
```

VISUALIZING THE DATA

Now here we can visualize the Data as it is shown below., which can help us to know about the information which is provided by each column

TABLE 1

Column	Description
YEAR	Year that the flight took place
QUARTER	Quarter that the flight took place (1-4)
MONTH	Month that the flight took place (1-12)
DAY_OF_MONTH	Day of the month that the flight took place (1-31)
DAY_OF_WEEK	Day of the week that the flight took place (1=Monday, 2=Tuesday, etc.)
UNIQUE_CARRIER	Airline carrier code (e.g., DL)
TAIL_NUM	Aircraft tail number
FL_NUM	Flight number
ORIGIN_AIRPORT_ID	ID of the airport of origin
ORIGIN	Origin airport code (ATL, DFW, SEA, etc.)
DEST_AIRPORT_ID	ID of the destination airport
DEST	Destination airport code (ATL, DFW, SEA, etc.)

CRS_DEP_TIME	Scheduled departure time
DEP_TIME	Actual departure time
DEP_DELAY	Number of minutes departure was delayed
DEP_DEL15	0=Departure delayed less than 15 minutes, 1=Departure delayed 15 minutes or more
CRS_ARR_TIME	Scheduled arrival time
ARR_TIME	Actual arrival time
ARR_DELAY	Number of minutes flight arrived late
ARR_DEL15	0=Arrived less than 15 minutes late, 1=Arrived 15 minutes or more late
CANCELLED	0=Flight was not cancelled, 1=Flight was cancelled
DIVERTED	0=Flight was not diverted, 1=Flight was diverted
CRS_ELAPSED_TIME	Scheduled flight time (in min)
ACTUAL_ELAPSED_TIME	Actual flight time (in min)
DISTANCE	Distance travelled in miles
DAY_OF_MONTH	Day of the month that the flight took place (1-31)

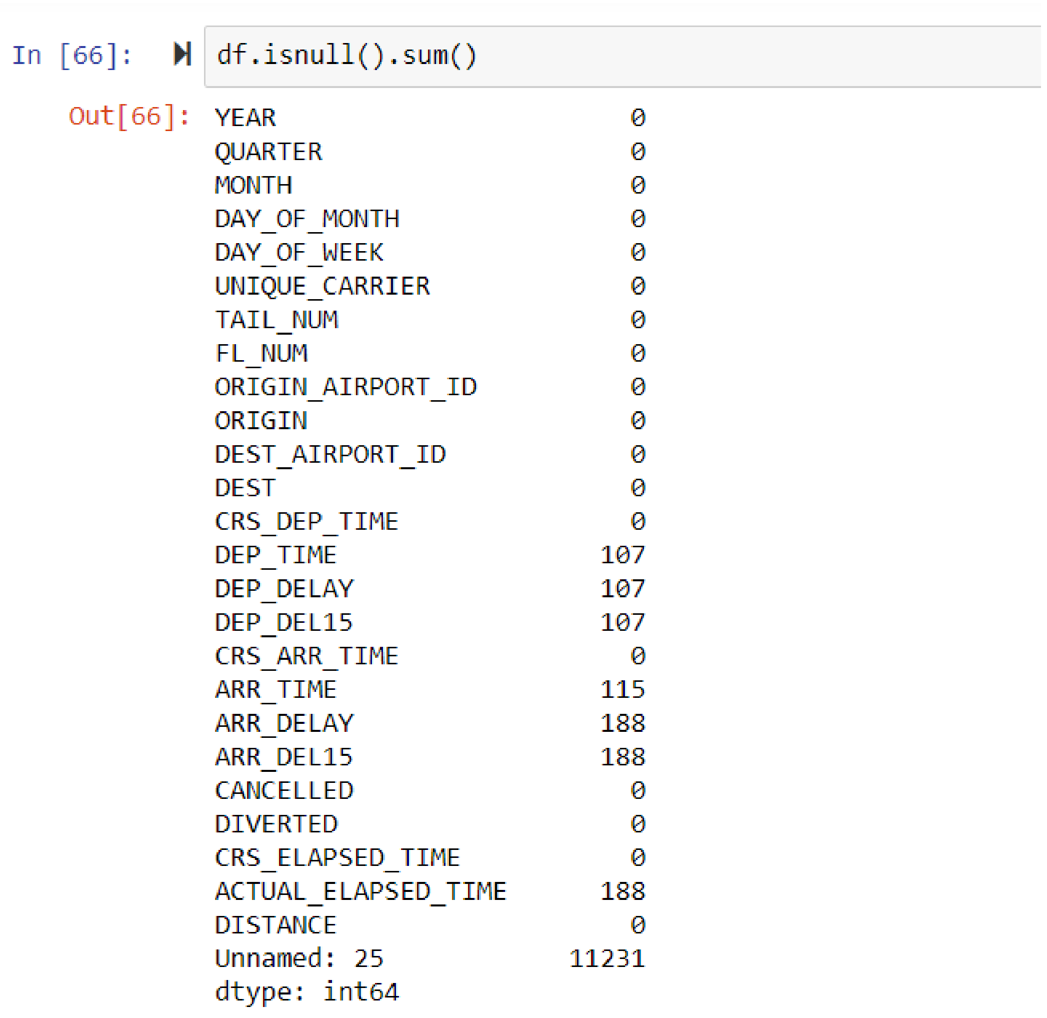
Herein, the dataset includes the date and time for every single flight. Here the applier logic is that any flight in us is more likely to be delayed in winter season due to winter storms.

CODE

```
df.isnull().sum()
```

But firstly, we need to clean the Data set prior to make it usable., in machine learning it becomes really important to select the useful features of the dataset so that a better training model can be generated from it.

Another important task is to identify the missing places and the next step is to either fill them or to delete the specific row.

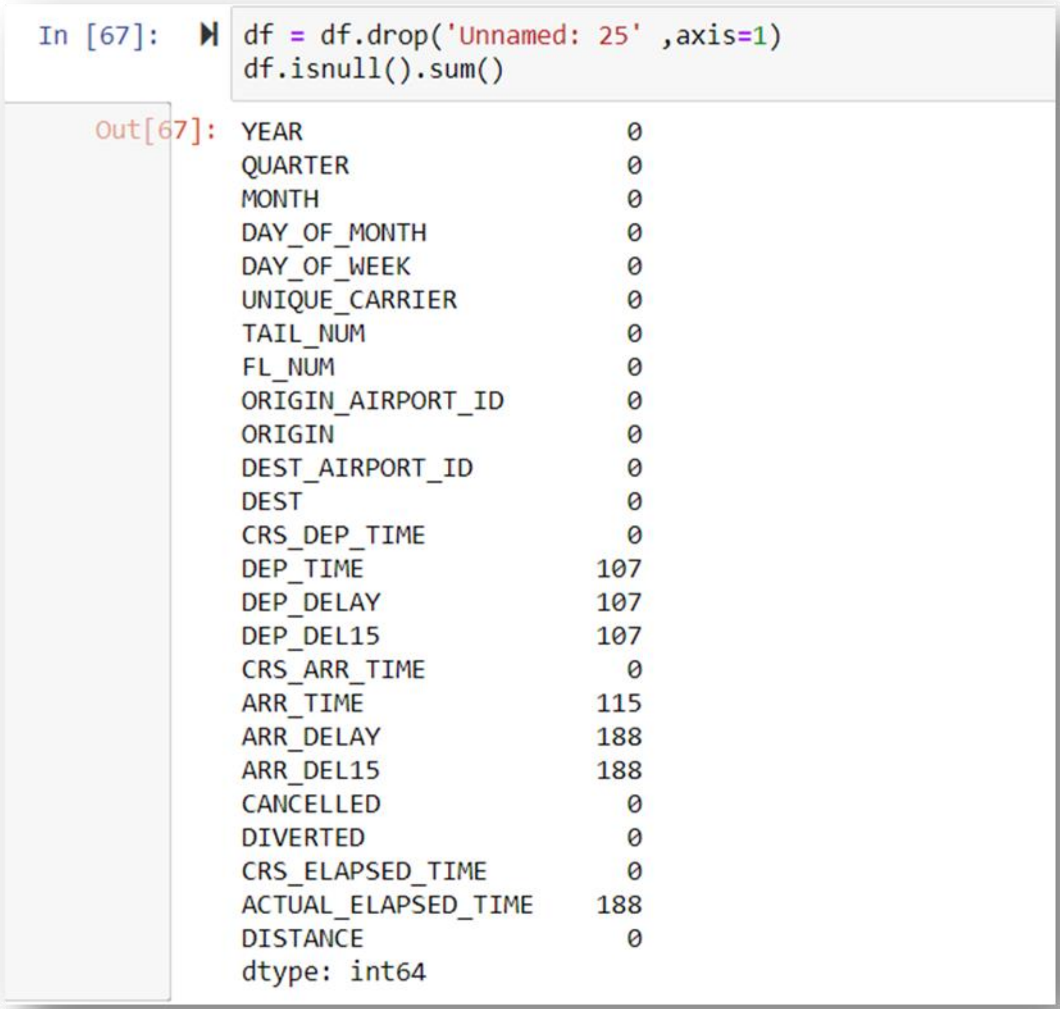

```
In [66]:  df.isnull().sum()

Out[66]:  YEAR                       0
          QUARTER                    0
          MONTH                      0
          DAY_OF_MONTH               0
          DAY_OF_WEEK                0
          UNIQUE_CARRIER             0
          TAIL_NUM                   0
          FL_NUM                     0
          ORIGIN_AIRPORT_ID          0
          ORIGIN                     0
          DEST_AIRPORT_ID            0
          DEST                       0
          CRS_DEP_TIME               0
          DEP_TIME                 107
          DEP_DELAY                107
          DEP_DEL15                107
          CRS_ARR_TIME               0
          ARR_TIME                 115
          ARR_DELAY                188
          ARR_DEL15                188
          CANCELLED                  0
          DIVERTED                   0
          CRS_ELAPSED_TIME           0
          ACTUAL_ELAPSED_TIME      188
          DISTANCE                   0
          Unnamed: 25            11231
          dtype: int64
```

Here, we can see that there is a 26th column which is Unnamed and contains 11,231 missing values. It is to be presumed that it has been mistakenly created and needs to be removed before using the data set.

To eliminate that column, we add the following code to the notebook and execute it:

CODE

```
df = df.drop('Unnamed: 25', axis=1)

df.isnull().sum()
```



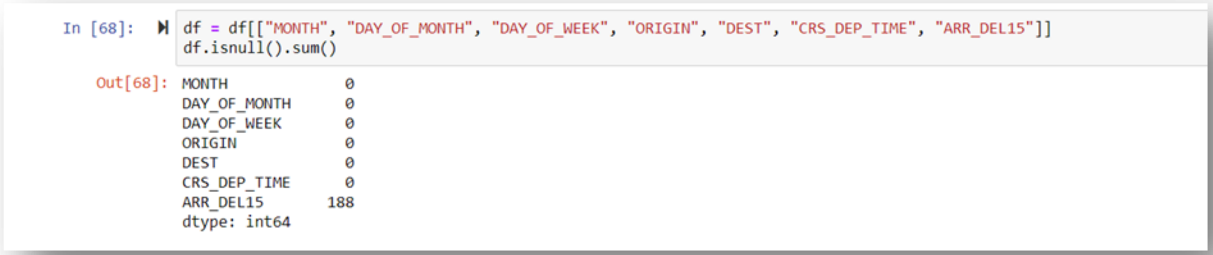
```
In [67]:  df = df.drop('Unnamed: 25' ,axis=1)
          df.isnull().sum()
```

```
Out[67]: YEAR                     0
         QUARTER                  0
         MONTH                    0
         DAY_OF_MONTH             0
         DAY_OF_WEEK              0
         UNIQUE_CARRIER           0
         TAIL_NUM                 0
         FL_NUM                   0
         ORIGIN_AIRPORT_ID        0
         ORIGIN                   0
         DEST_AIRPORT_ID          0
         DEST                     0
         CRS_DEP_TIME             0
         DEP_TIME               107
         DEP_DELAY              107
         DEP_DEL15              107
         CRS_ARR_TIME             0
         ARR_TIME               115
         ARR_DELAY              188
         ARR_DEL15              188
         CANCELLED                0
         DIVERTED                 0
         CRS_ELAPSED_TIME         0
         ACTUAL_ELAPSED_TIME    188
         DISTANCE                 0
         dtype: int64
```

But the noticeable point is that our Dataset still contains a lot of missing values and needs to be cleaned up before actually using it. We know that some of the columns are not useful for training our ML model .

Therefore, the next most important step is to remove the irrelevant columns from our dataset.

Pandas provides us an easy way to filter out columns which we don't want.



```
In [68]:  df = df[["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_DEP_TIME", "ARR_DEL15"]]
           df.isnull().sum()

Out[68]:  MONTH            0
          DAY_OF_MONTH     0
          DAY_OF_WEEK      0
          ORIGIN           0
          DEST             0
          CRS_DEP_TIME     0
          ARR_DEL15      188
          dtype: int64
```

Doing all this has still made us left with a column of missing values i.e., ARR_DEL15 column.

The logic it uses is :

0= the flight arrived on time

1=the flight arrived late

Here we will filter out those rows which have no value with the code shown below:

CODE

df=df[["MONTH", "DAY_OF_MONTH", "DAY_OF_WEEK", "ORIGIN", "DEST", "CRS_DEP_TIME", "ARR_DEL15"]]

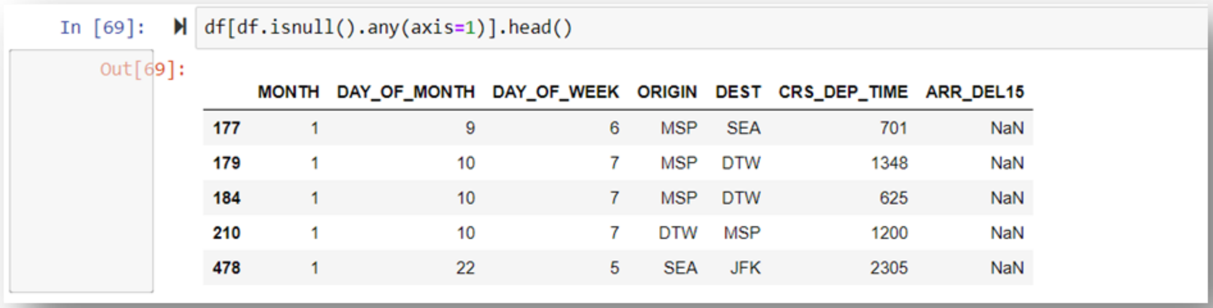
df.isnull().sum()

Pandas represents missing values with NaN, which stands for Not a Number.

The output shows that these rows are indeed missing values in the ARR_DEL15 column:

CODE

df= df[df.isnull().values.any(axis=1)].head()



```
In [69]:  df[df.isnull().any(axis=1)].head()
```

Out[69]:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	NaN
179	1	10	7	MSP	DTW	1348	NaN
184	1	10	7	MSP	DTW	625	NaN
210	1	10	7	DTW	MSP	1200	NaN
478	1	22	5	SEA	JFK	2305	NaN

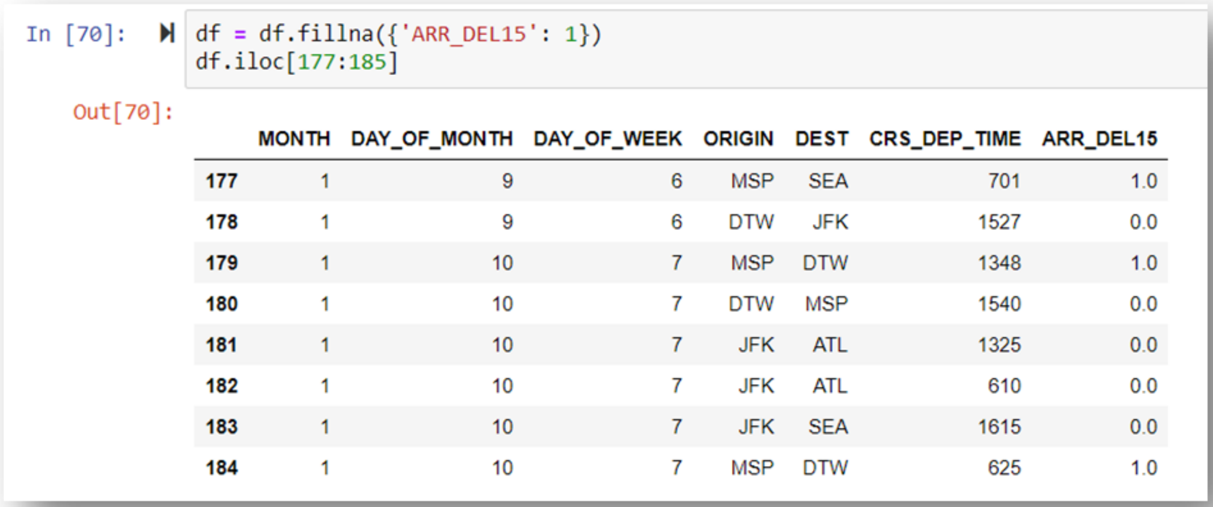
The reason these rows are missing ARR_DEL15 values is that they all correspond to flights that were canceled or diverted. We can call dropna on the DataFrame to remove these rows. But since a flight that is canceled or diverted to another airport could be considered "late," let's use the fillna method to replace the missing values with 1s.

CODE

df = df.fillna({'ARR_DEL15': 1})

df.iloc[177:185]

Use the following code to replace missing values in the ARR_DEL15 column with 1s and display rows 177 through 184:



```
In [70]:  df = df.fillna({'ARR_DEL15': 1})
          df.iloc[177:185]
```

Out[70]:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
177	1	9	6	MSP	SEA	701	1.0
178	1	9	6	DTW	JFK	1527	0.0
179	1	10	7	MSP	DTW	1348	1.0
180	1	10	7	DTW	MSP	1540	0.0
181	1	10	7	JFK	ATL	1325	0.0
182	1	10	7	JFK	ATL	610	0.0
183	1	10	7	JFK	SEA	1615	0.0
184	1	10	7	MSP	DTW	625	1.0

The missing values have been replaced and the irrelevant columns have been removed too. Now, we can call our data as a “cleaned data”

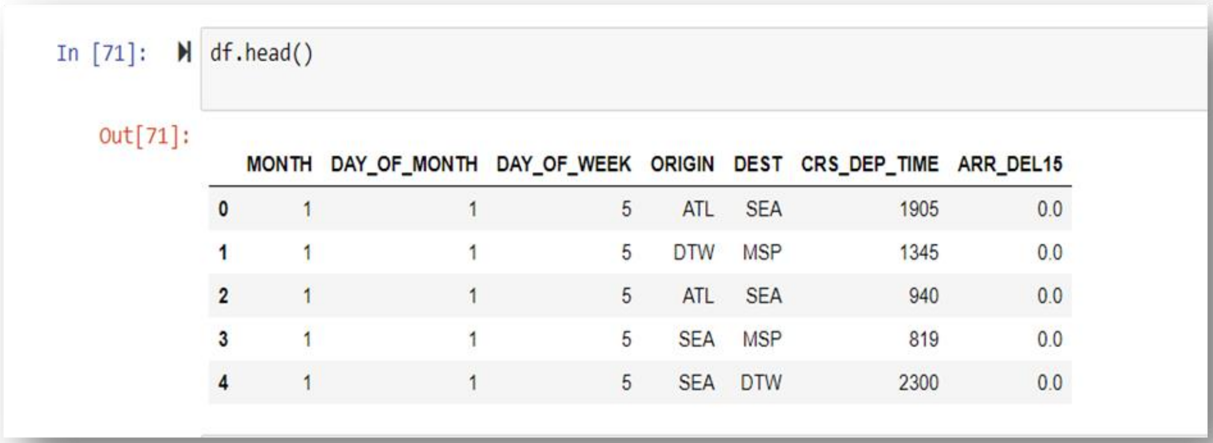
The CRS_DEP_TIME column of the dataset we are using represents scheduled departure times. The granularity of the numbers in this column — it contains more than 500 unique values — could have a negative impact on accuracy in a machine-learning model. This can be resolved using a technique called binning or quantization. What if we divided each number in this column by 100 and rounded down to the nearest integer? 1030 would become 10, 1925 would become 19, and so on, and we would be left with a maximum of 24 discrete values in this column. Intuitively, it makes sense, because it probably doesn't matter much whether a flight leaves at 10:30 a.m. or 10:40 a.m. It matters a great deal whether it leaves at 10:30 a.m. or 5:30 p.m.

In addition, the dataset's ORIGIN and DEST columns contain airport codes that represent categorical machine-learning values. These columns need to be converted into discrete columns containing indicator variables, sometimes known as "dummy" variables. In other words, the ORIGIN column, which contains five airport codes, needs to be converted into five columns, one per airport, with each column containing 1s and 0s indicating whether a flight originated at the airport that the column represents. The DEST column needs to be handled in a similar manner.

Here, we will "bin" the departure times in the CRS_DEP_TIME column and use Pandas' get_dummies method to create indicator columns from the ORIGIN and DEST columns.

CODE

```
df.head()

import math

for index, row in df.iterrows():

   df.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME'] / 100)

df.head()
```



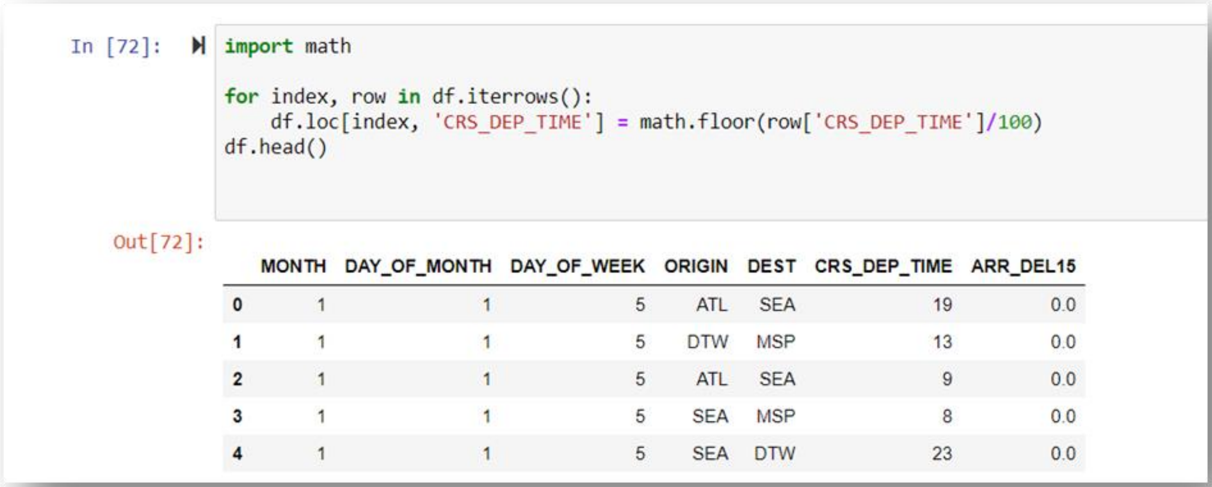
```
In [71]:  df.head()

Out[71]:
```

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	1905	0.0
1	1	1	5	DTW	MSP	1345	0.0
2	1	1	5	ATL	SEA	940	0.0
3	1	1	5	SEA	MSP	819	0.0
4	1	1	5	SEA	DTW	2300	0.0

CODE

df = pd.get_dummies(df, columns=['ORIGIN', 'DEST'])

df.head()

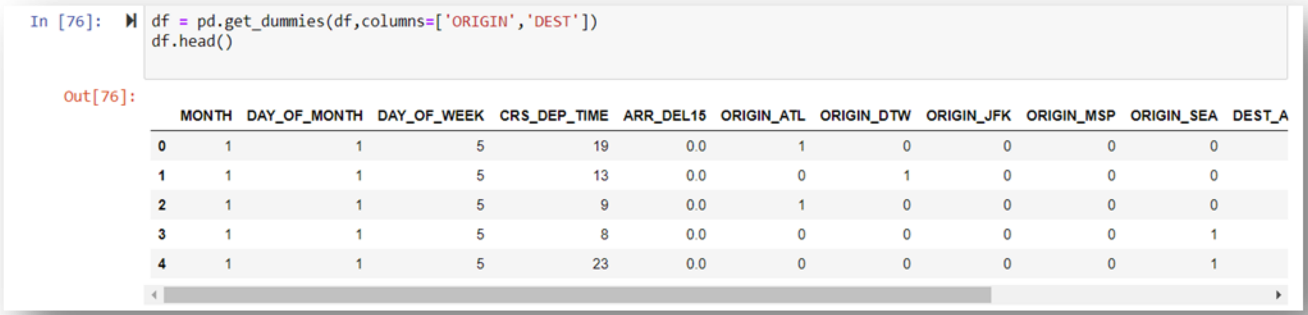


```
In [72]:  import math

          for index, row in df.iterrows():
              df.loc[index, 'CRS_DEP_TIME'] = math.floor(row['CRS_DEP_TIME']/100)
          df.head()

Out[72]:
```

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15
0	1	1	5	ATL	SEA	19	0.0
1	1	1	5	DTW	MSP	13	0.0
2	1	1	5	ATL	SEA	9	0.0
3	1	1	5	SEA	MSP	8	0.0
4	1	1	5	SEA	DTW	23	0.0

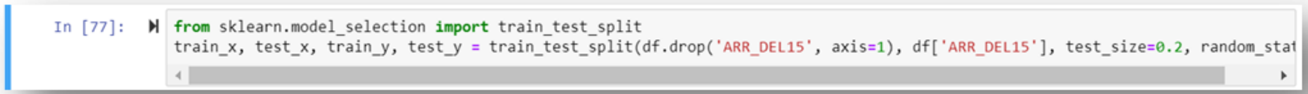
Here, we will use the below code to generate the indicator columns inclusively with the ORIGIN and DEST columns, while dropping the ORIGIN and DEST columns themselves:

Here, we can see that our ORIGIN and DEST columns a=have been replaced with new altered DUMMY columns corresponding to the airport codes present in the original columns. The new columns have 1s and 0s indicating whether a given flight originated at or was destined for the corresponding airport.



```
In [76]:  df = pd.get_dummies(df,columns=['ORIGIN','DEST'])
          df.head()

Out[76]:
     MONTH  DAY_OF_MONTH  DAY_OF_WEEK  CRS_DEP_TIME  ARR_DEL15  ORIGIN_ATL  ORIGIN_DTW  ORIGIN_JFK  ORIGIN_MSP  ORIGIN_SEA  DEST_A
  0      1             1            5            19        0.0           1           0           0           0           0
  1      1             1            5            13        0.0           0           1           0           0           0
  2      1             1            5             9        0.0           1           0           0           0           0
  3      1             1            5             8        0.0           0           0           0           0           1
  4      1             1            5            23        0.0           0           0           0           0           1
```

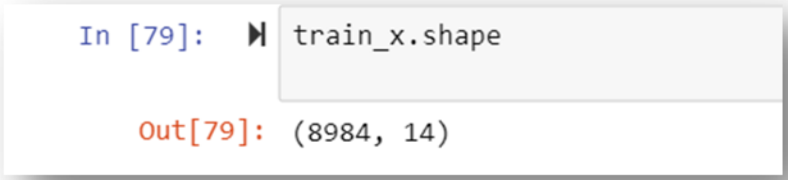
To create a machine learning model, we need two datasets: one for training and one for testing. In practice, we often have only one dataset, so we split it into two. We have performed an 80:20 split on the DataFrame we prepared previously so we can use it to train a machine learning model. We will also separate the DataFrame into feature columns and label columns. The former contains the columns used as input to the model (for example, the flight's origin and destination and the scheduled departure time), while the latter contains the column that the model will attempt to predict — in this case, the ARR_DEL15 column, which indicates whether a flight will arrive on time.



```
In [77]:  from sklearn.model_selection import train_test_split
          train_x, test_x, train_y, test_y = train_test_split(df.drop('ARR_DEL15', axis=1), df['ARR_DEL15'], test_size=0.2, random_stat
```

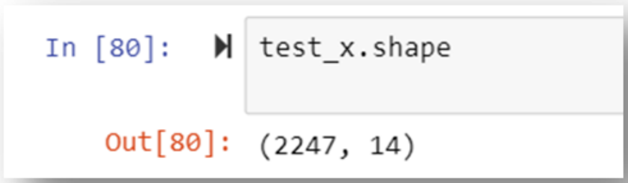
The first statement imports scikit-learn's train_test_split helper function. The second line uses the function to split the DataFrame into a training set containing 80% of the original data, and a test set containing the remaining 20%. The random_state parameter seeds the random-number generator used to do the splitting, while the first and second parameters are DataFrames containing the feature columns and the label column.

train_test_split returns four DataFrames. Use the following command to display the number of rows and columns in the DataFrame containing the feature columns used for training:



```
In [79]:  train_x.shape

Out[79]:  (8984, 14)
```

Now use this command to display the number of rows and columns in the DataFrame containing the feature columns used for testing:



```
In [80]:  test_x.shape

Out[80]:  (2247, 14)
```

How do the two outputs differ, and why?

There are many types of machine learning models. One of the most common is the regression model, which uses one of a number of regression algorithms to produce a numeric value — for example, a person's age or the probability that a credit-card transaction is fraudulent. We'll train a classification model, which seeks to resolve

CODE

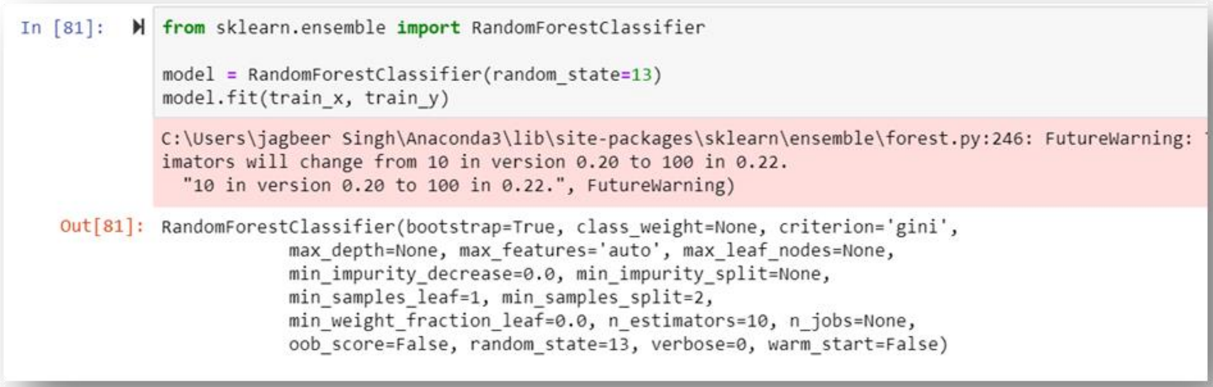
```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=13)

model.fit(train_x, train_y)
```

a set of inputs into one of a set of known outputs. A classic example of a classification model is one that examines e-mails and classifies them as "spam" or "not spam." our model will be a binary classification model that predicts whether a flight will arrive on-time or late ("binary" because there are only two possible outputs).

One of the benefits of using scikit-learn is that we don't have to build these models — or implement the algorithms that they use — by hand. Scikit-learn includes a variety of classes for implementing common machine learning models. One of them is RandomForestClassifier, which fits multiple decision trees to the data and uses averaging to boost the overall accuracy and limit overfitting.

Execute the following code in a new cell to create a RandomForestClassifier object and train it by calling the fit method.

The output shows the parameters used in the classifier, including n_estimators, which specifies the number of trees in each decision-tree forest, and max_depth, which specifies the maximum depth of the decision trees. The values shown are the defaults, but we can override any of them when creating the RandomForestClassifier object.

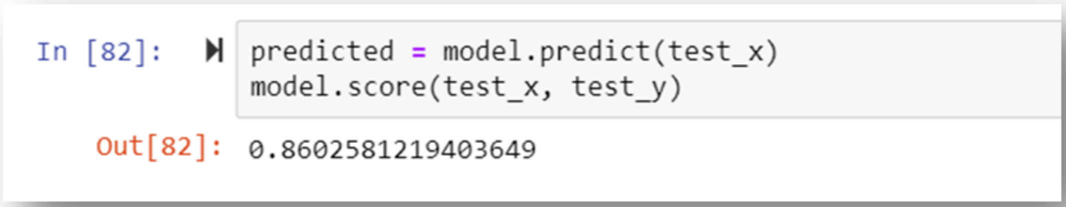


```
In [81]:  from sklearn.ensemble import RandomForestClassifier

          model = RandomForestClassifier(random_state=13)
          model.fit(train_x, train_y)

          C:\Users\jagbeer Singh\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning:
          imators will change from 10 in version 0.20 to 100 in 0.22.
            "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Out[81]:  RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                      oob_score=False, random_state=13, verbose=0, warm_start=False)
```

Now calling the predict method to test the model using the values in test_x, followed by the score method to determine the mean accuracy of the model:

CODE

```
predicted = model.predict(test_x)

model.score(test_x, test_y)
```



```
In [82]:  predicted = model.predict(test_x)
          model.score(test_x, test_y)

Out[82]:  0.8602581219403649
```

The mean accuracy is 86%, which seems good on the surface. However, mean accuracy isn't always a reliable indicator of the accuracy of a classification model. Let's dig a little deeper and determine how accurate the model really is — that is, how adept it is at determining whether a flight will arrive on time.

There are several ways to measure the accuracy of a classification model. One of the best overall measures for a binary classification model is Area Under Receiver Operating Characteristic Curve (sometimes referred to as "ROC AUC"), which essentially quantifies how often the model will make a correct prediction regardless of the outcome. In this unit, we'll compute an ROC AUC score for the model we built previously and learn about some of the reasons why that score is loour than the mean accuracy output by the score method. We'll also learn about other ways to gauge the accuracy of the model.

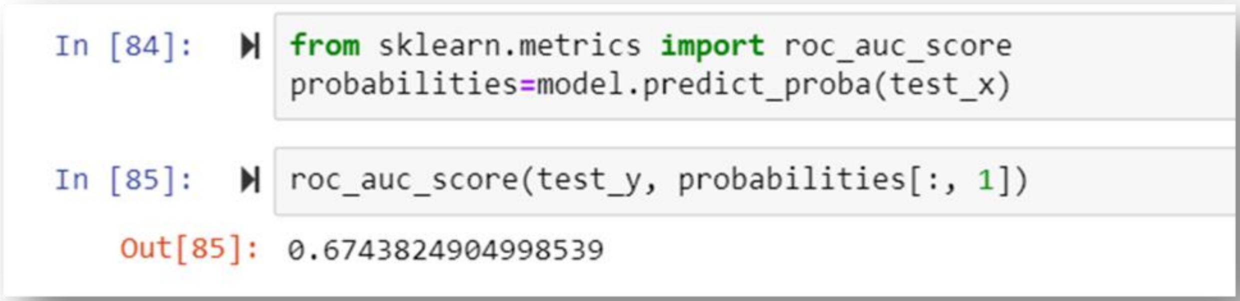
Before we compute the ROC AUC, we must generate prediction probabilities for the test set. These probabilities are estimates for each of the classes, or answers, the model can predict. For example, [0.88199435, 0.11800565] means that there's an 89% chance that a flight will arrive on time (ARR_DEL15 = 0) and a 12% chance that it won't (ARR_DEL15 = 1). The sum of the two probabilities adds up to 100%.

Run the following code to generate a set of prediction probabilities from the test data:

Now we use the following statement to generate an ROC AUC score from the probabilities using scikit-learn's roc_auc_score method:

CODE

```
from sklearn.metrics import roc_auc_score

probabilities = model.predict_proba(test_x)

roc_auc_score(test_y, probabilities[:, 1])
```



```
In [84]:   from sklearn.metrics import roc_auc_score
           probabilities=model.predict_proba(test_x)

In [85]:   roc_auc_score(test_y, probabilities[:, 1])

Out[85]:   0.6743824904998539
```

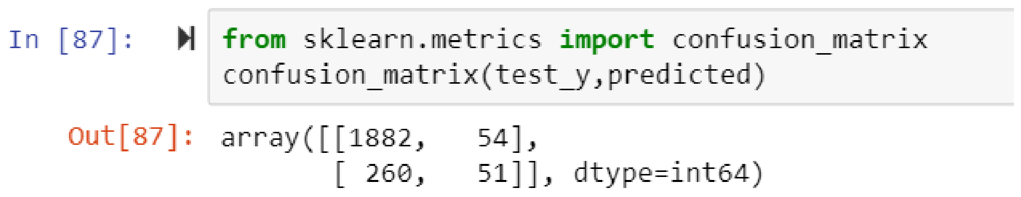
Why is the AUC score lower than the mean accuracy computed in the previously?

The output from the score method reflects how many of the items in the test set the model predicted correctly. This score is skewed by the fact that the dataset the model was trained and tested with contains many more rows representing on-time arrivals than rows representing late arrivals. Because of this imbalance in the data, we're more likely to be correct if we predict that a flight will be on time than if we predict that a flight will be late.

ROC AUC takes this into account and provides a more accurate indication of how likely it is that a prediction of on-time or late will be correct.

We can learn more about the model's behavior by generating a confusion matrix, also known as an error matrix. The confusion matrix quantifies the number of times each answer was classified correctly or incorrectly. Specifically, it quantifies the number of false positives, false negatives, true positives, and true negatives. This is important, because if a binary classification model trained to recognize cats and dogs is tested with a dataset that is 95% dogs, it could score 95% simply by guessing "dog" every time. But if it failed to identify cats at all, it would be of little value.

We Use the following code to produce a confusion matrix for our model:



```
In [87]:    from sklearn.metrics import confusion_matrix
            confusion_matrix(test_y,predicted)

Out[87]: array([[1882,   54],
                [ 260,   51]], dtype=int64)
```

The first row in the output represents flights that were on time. The first column in that row shows how many flights were correctly predicted to be on time, while the second column reveals how many flights were predicted as delayed but weren't. From this, the model appears to be adept at predicting that a flight will be on time.

CODE

```
from sklearn.metrics import confusion_matrix

confusion_matrix(test_y, predicted)
```

But look at the second row, which represents flights that were delayed. The first column shows how many delayed flights were incorrectly predicted to be on time. The second column shows how many flights were correctly predicted to be delayed. Clearly, the model isn't nearly as adept at predicting that a flight will be delayed as it is at predicting that a flight will arrive on time. What we want in a confusion matrix is large numbers in the upper-left and lower-right corners, and small numbers (preferably zeros) in the upper-right and lower-left corners.

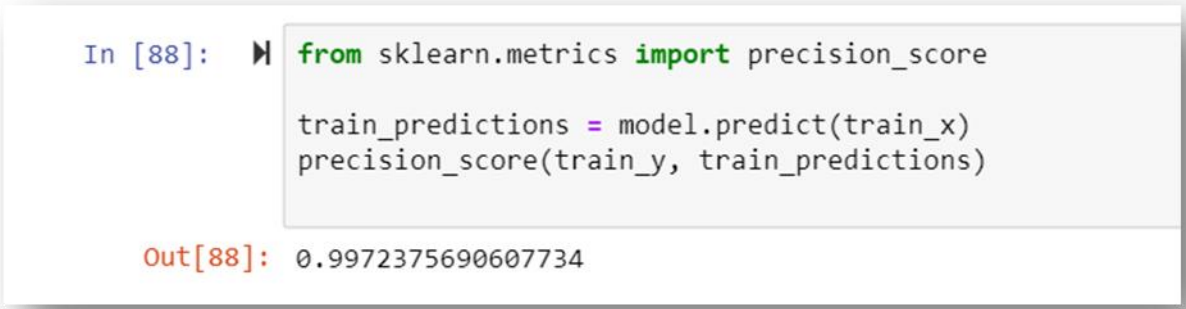
Other measures of accuracy for a classification model include precision and recall. Suppose the model was presented with three on-time arrivals and three delayed arrivals, and that it correctly predicted two of the on-time arrivals, but incorrectly predicted that two of the delayed arrivals would be on time. In this case, the precision would be 50% (two of the four flights it classified as being on time actually were on time), while its recall would be 67% (it correctly identified two of the three on-time arrivals). We can learn more about precision and recall from https://en.wikipedia.org/wiki/Precision_and_recall

Scikit-learn contains a handy method named precision_score for computing precision. To quantify the precision of our model, execute the following

CODE

```
from sklearn.metrics import precision_score

train_predictions = model.predict(train_x)

precision_score(train_y, train_predictions)
```

statements:



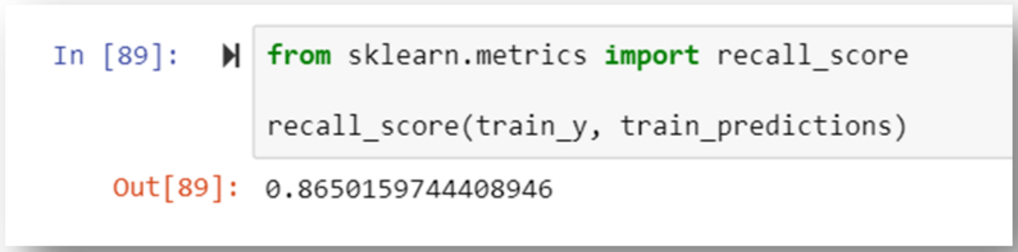
```
In [88]:  from sklearn.metrics import precision_score

          train_predictions = model.predict(train_x)
          precision_score(train_y, train_predictions)

Out[88]:  0.9972375690607734
```

CODE

from sklearn.metrics import recall_score

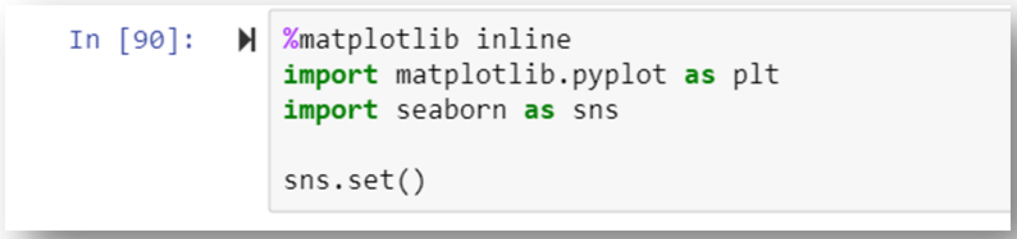
recall_score(train_y, train_predictions)

Scikit-learn also contains a method named recall_score for computing recall. To measure model's recall, we execute the following statements:



```
In [89]:  from sklearn.metrics import recall_score

           recall_score(train_y, train_predictions)

Out[89]:   0.8650159744408946
```

Now we Execute the following statements in a new cell at the end of the notebook while Ignoring any warning messages that are displayed related to font caching:


```
In [90]:  %matplotlib inline
          import matplotlib.pyplot as plt
          import seaborn as sns

          sns.set()
```

The first statement is one of several magic commands supported by the Python kernel that we selected when we created the notebook. It enables Jupyter to render Matplotlib output in a notebook without making repeated calls to show. And it must appear before any references to Matplotlib itself. The final statement configures Seaborn to enhance the output from Matplotlib.

CODE

```
%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()
```

To see Matplotlib at work, we execute the following code in a new cell to plot the ROC curve for the machine-learning model we built in the previous lab:

CODE

from sklearn.metrics import roc_curve

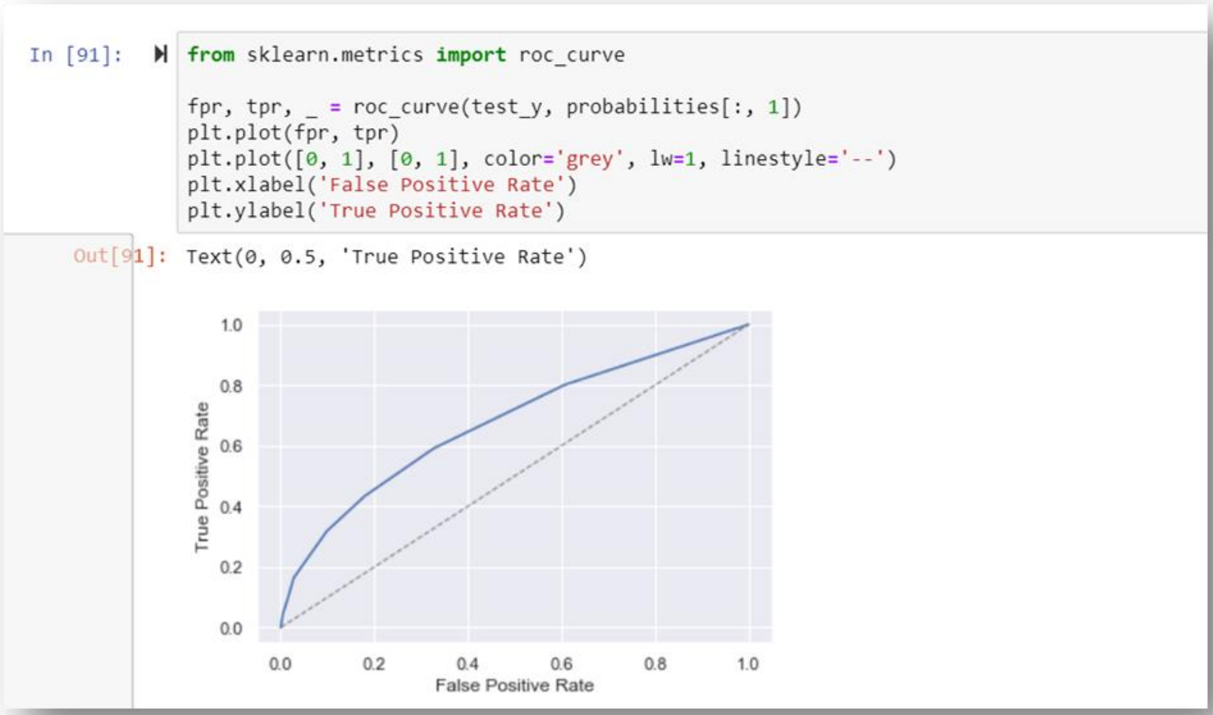
fpr, tpr, _ = roc_curve(test_y, probabilities[:, 1])

plt.plot(fpr, tpr)

plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')



```
In [91]:  from sklearn.metrics import roc_curve

          fpr, tpr, _ = roc_curve(test_y, probabilities[:, 1])
          plt.plot(fpr, tpr)
          plt.plot([0, 1], [0, 1], color='grey', lw=1, linestyle='--')
          plt.xlabel('False Positive Rate')
          plt.ylabel('True Positive Rate')

Out[91]:  Text(0, 0.5, 'True Positive Rate')
```

ROC curve generated with Matplotlib

The dotted line in the middle of the graph represents a 50-50 chance of obtaining a correct answer. The blue curve represents the accuracy of our model. More importantly, the fact that this chart appears at all demonstrates that we can use Matplotlib in a Jupyter notebook.

The reason we built a machine-learning model is to predict whether a flight will arrive on time or late. In this exercise, we'll write a Python function that calls the machine-learning model we built in the previous lab to compute the likelihood that a flight will be on time. Then we'll use the function to analyse several flights.

Now we Enter the following function definition in a new cell, and then run the cell.

CODE

```
def predict_delay(departure_date_time, origin, destination):
    from datetime import datetime


    try:
        departure_date_time_parsed = datetime.strptime(departure_date_time, '%d/%m/%Y
%H:%M:%S')
    except ValueError as e:
        return 'Error parsing date/time - {}'.format(e)
    month = departure_date_time_parsed.month
    day = departure_date_time_parsed.day
    day_of_week = departure_date_time_parsed.isoweekday()
    hour = departure_date_time_parsed.hour


    origin = origin.upper()
    destination = destination.upper()
    input = [{'MONTH': month,
             'DAY': day,
             'DAY_OF_WEEK': day_of_week,
             'CRS_DEP_TIME': hour,
             'ORIGIN_ATL': 1 if origin == 'ATL' else 0,
             'ORIGIN_DTW': 1 if origin == 'DTW' else 0,
```

CODE

'ORIGIN_SEA': 1 if origin == 'SEA' else 0,

'DEST_ATL': 1 if destination == 'ATL' else 0,

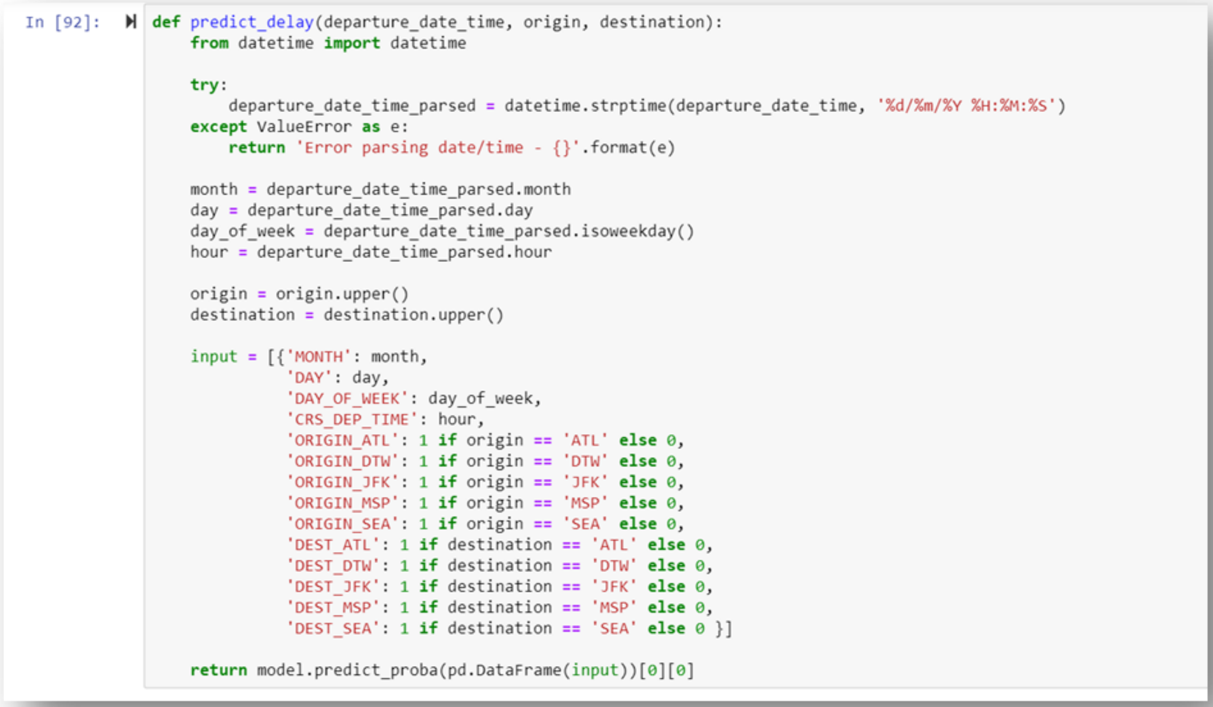
'DEST_DTW': 1 if destination == 'DTW' else 0,

'DEST_JFK': 1 if destination == 'JFK' else 0,

'DEST_MSP': 1 if destination == 'MSP' else 0,

'DEST_SEA': 1 if destination == 'SEA' else 0 }]

return model.predict_proba(pd.DataFrame(input))[0][0]



```
In [92]:  def predict_delay(departure_date_time, origin, destination):
              from datetime import datetime

              try:
                  departure_date_time_parsed = datetime.strptime(departure_date_time, '%d/%m/%Y %H:%M:%S')
              except ValueError as e:
                  return 'Error parsing date/time - {}'.format(e)

              month = departure_date_time_parsed.month
              day = departure_date_time_parsed.day
              day_of_week = departure_date_time_parsed.isoweekday()
              hour = departure_date_time_parsed.hour

              origin = origin.upper()
              destination = destination.upper()

              input = [{'MONTH': month,
                        'DAY': day,
                        'DAY_OF_WEEK': day_of_week,
                        'CRS_DEP_TIME': hour,
                        'ORIGIN_ATL': 1 if origin == 'ATL' else 0,
                        'ORIGIN_DTW': 1 if origin == 'DTW' else 0,
                        'ORIGIN_JFK': 1 if origin == 'JFK' else 0,
                        'ORIGIN_MSP': 1 if origin == 'MSP' else 0,
                        'ORIGIN_SEA': 1 if origin == 'SEA' else 0,
                        'DEST_ATL': 1 if destination == 'ATL' else 0,
                        'DEST_DTW': 1 if destination == 'DTW' else 0,
                        'DEST_JFK': 1 if destination == 'JFK' else 0,
                        'DEST_MSP': 1 if destination == 'MSP' else 0,
                        'DEST_SEA': 1 if destination == 'SEA' else 0 }]

              return model.predict_proba(pd.DataFrame(input))[0][0]
```

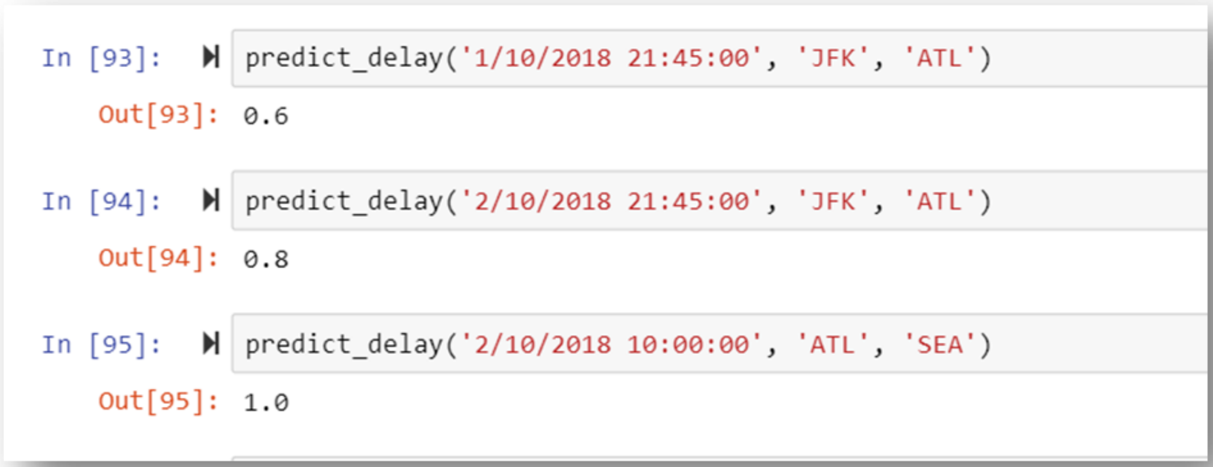
This function takes as input a date and time, an origin airport code, and a destination airport code, and returns a value between 0.0 and 1.0 indicating the probability that the flight will arrive at its destination on time. It uses the machine-learning model we built in the previous lab to compute the probability. And to call the model, it passes a DataFrame containing the input values to predict_proba. The structure of the DataFrame exactly matches the structure of the DataFrame we used earlier.

Date input to the predict_delay function use the international date format dd/mm/year.

We Use the code below to compute the probability that a flight from New York to Atlanta on the evening of October 1 will arrive on time. The year we enter is irrelevant because it isn't used by the model.

CODE

predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')



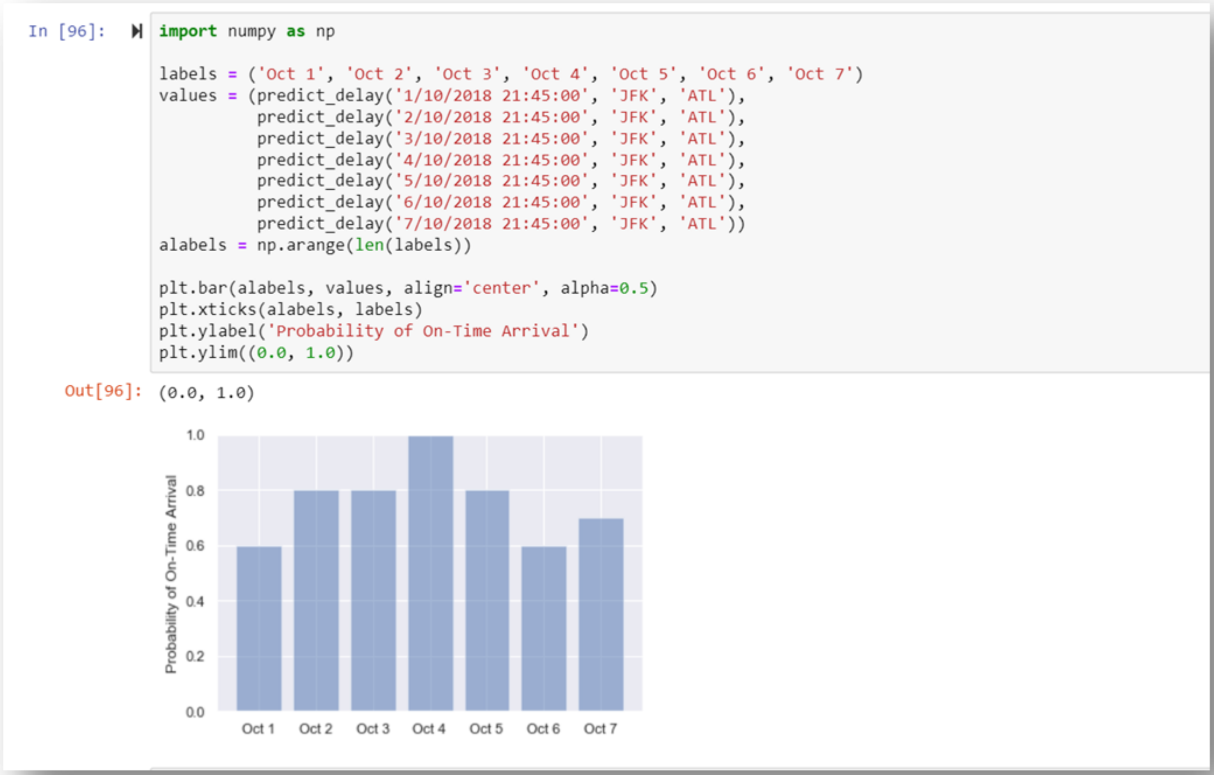
```
In [93]:  predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL')
   Out[93]: 0.6

In [94]:  predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL')
   Out[94]: 0.8

In [95]:  predict_delay('2/10/2018 10:00:00', 'ATL', 'SEA')
   Out[95]: 1.0
```

We now have an easy way to predict, with a single line of code, whether a flight is likely to be on time or late. Feel free to experiment with other dates, times, origins, and destinations. But keep in mind that the results are only meaningful for the airport codes ATL, DTW, JFK, MSP, and SEA because those are the only airport codes the model was trained with.

Execute the following Code to plot the probability of on-time arrivals for an evening flight from JFK to ATL over a range of days:r

CODE

```
import numpy as np


labels = ('Oct 1', 'Oct 2', 'Oct 3', 'Oct 4', 'Oct 5', 'Oct 6', 'Oct 7')

values = (predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('3/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('4/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('5/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('6/10/2018 21:45:00', 'JFK', 'ATL'),

        predict_delay('7/10/2018 21:45:00', 'JFK', 'ATL'))

alabels = np.arange(len(labels))


plt.bar(alabels, values, align='center', alpha=0.5)

plt.xticks(alabels, labels)

plt.ylabel('Probability of On-Time Arrival')
```



```
In [96]:  import numpy as np

          labels = ('Oct 1', 'Oct 2', 'Oct 3', 'Oct 4', 'Oct 5', 'Oct 6', 'Oct 7')
          values = (predict_delay('1/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('2/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('3/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('4/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('5/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('6/10/2018 21:45:00', 'JFK', 'ATL'),
                    predict_delay('7/10/2018 21:45:00', 'JFK', 'ATL'))
          alabels = np.arange(len(labels))

          plt.bar(alabels, values, align='center', alpha=0.5)
          plt.xticks(alabels, labels)
          plt.ylabel('Probability of On-Time Arrival')
          plt.ylim((0.0, 1.0))

Out[96]:  (0.0, 1.0)
```

Results and conclusion:

From all the observations done over data and after the implementation of machine learning model I conclude this project with the result as being shown as a graph in which the probability of flight being on time at a very particular timestamp can be identified.

References:

[1] XiaoboZhou,Kuang-Yu Liua, b,Stephen T.C. Wonga, "Biomedical Machine Learning Cancer classification and prediction using logistic regression with Bayesian gene selection,"Volume 37, Issue 4, August 2004.

[2] Abidatul Izzah et. All, Highly Accurate and Real-Time Determinationof Resonant Characteristics: Complex LinearRegression of the Transmission Coefficient, IEEE TRANSACTIONS ON MICROWAVE THEORY AND TECHNIQUES, VOL. 52, NO. 9, SEPTEMBER 2004

[3]FENG ZHAO,QING HUA LI, " A plane regression-based sequence forecast algorithm for stream data", **Published in:** Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on18-21 Aug. 2005 IEEE.

[4] Duck Jin Chai EunHee Kim “Prediction of Frequent Items to One Dimensional Stream Data Fifth International Conference on Computational Science and Applications” 0-7695-2945-3/07 © 2007 IEEE.

[5]Andreas Hecker, Thomas K¨urner,” Application of Classification and Regression” Trees for Paging Traffic Prediction in LAC Planning 1550-2252/$25.00 ©2007 IEEE.

[6] Qingshan Ni1, Zhengzhi Wang1, Qingjuan Han2, Gangguo Li1, Xiaomin Wang1, GuangyunWang,“Using logistic regression method to predict protein function from protein-protein interaction” data 978-1-4244-2902-8/09/$25.00 ©2009 IEEE.

[7] Alberto Landi, Paolo Piaggi Artificial Neural Networks for Nonlinear Regression and Classification 978-1-4244-8136-1/10/$26.00_c 2010 IEEE.

[8] SonaliTiwari” Sequence Forecast Algorithm Based on Nonlinear Regression Technique for Stream Data” Vol. 1, No. 2, July-December 2010.

[9]KrisztianBuza, AlexandrosNanopoulos, Lars Schmidt-Time-Series Classification based on Individualised Error Prediction Thieme 2010 13th IEEE International Conference on Coputational Science and Engineering 978-0-7695-4323-9/10 $26.00 © 2010 IEEE.

[10] Umesh Kumar Pandey, Saurabh Pal Data Mining : A prediction of performer or underperformer using classification (IJCSIT) International Journal of Computer Science and Information Technology, Vol. 2(2), 2011.

[11] NidhiBhatlaKiranJyoti An Analysis of Heart Disease Prediction using Different DataMining Techniques International Journal of Engineering Research & Technology (IJERT) vol. 1 Issue 8, October – 2012.

[12] Rita P. RibeiroTowards Utility Maximization in Regression 2012 IEEE 12th InternationalConference on Data Mining Workshops 978-0-7695-4925-5/12 $26.00 © 2012 IEEE.

[13] Mythili T., DevMukherji, Nikita Padalia, and Abhiram Naidu A Heart Disease Prediction Model using SVM-Decision Trees-Logistic Regression (SDL) International Journal of Computer Applications (0975 – 8887) Volume 68– No.16, April 2013

[14] Nirmala, M. and V. Palanisamy” AN EFFICIENT PREDICTION OF MISSING ITEMSET IN SHOPPING CART” Journal of Computer Science, 9 (1): 55-62, 2013 ISSN 1549-3636 2013