PROJECT REPORT

"Flight Price Prediction
by using
Machine Learning
&
Deep Learning Models"

Submitted towards the partial fulfilment of the criteria for award of Post Graduate Program in Data Analytics

Submitted By:

G.VIGNESHWARAN

Course and Batch: PGA-04-APRIL-2021

Abstract

In this project I am going to take Flight Price Prediction as the ticket prices can be something hard to guess, today we might see a price, check out the price of the same flight tomorrow, it will be a different story. We might have often heard travelers saying that flight ticket prices are so unpredictable. We are gonna predict the price of the flight just by entering a few details like Airline, Source, Destination, Departure time, Total no of stops and with Additional info exact flight price is predicted. Here with the prices of flight tickets for various airlines between the months of March and June of 2019 and between various cities data. As the day to day travelers would overcome the problem of how much the price of fare will cost depending on the airline and with other criteria, this project will help to overcome this problem.

Acknowledgements

We are using this opportunity to express our gratitude to everyone who supported us throughout the course of this group project. We are thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the project work. We are sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the project.

Further, we were fortunate to have great teachers who readily shared their immense knowledge in data analytics and guided us in a manner that the outcome resulted in enhancing our data skills.

We wish to thank all the faculties, as this project utilized knowledge gained from every course that formed the PGA program.

We certify that the work done by us for conceptualizing and completing this project is original and authentic.

Date: March, 2021 G.Vigneshwaran

Place: Coimbatore

Certificate of Completion

I hereby certify that the project titled "**Flight Price Prediction by using Machine learning & Deep learning Models"** was undertaken and completed under my supervision by Thulasidass from the batch of PGA 04 COI

Date: March, 2021

Place:

Coimbatore

Table of Contents

CHAPTER 1
INTRODUCTION

1.1Title & Objective of the study

Flight Price Prediction ***Analysis by using Machine Learning & Deep Learning Models.***,

In this project have used various machine learning regression models like, Gradient Boosting Regressor, Extreme Gradient Boosting Regressor, Decision Tree Regressor, Random Forest Regressor & Artificial Neural Network to predict the ticket price values. As the ticket price value prediction is important for many travellers and we might have often heard travelers saying that flight ticket prices are so unpredictable and with these machine learning models will see which model performs well in ticket price prediction

1.2 Need of the Study

In this project, the main purpose is to predict ticket price with details like Airline, Source, Destination, Departure time, Total no of stops and with Additional info exact flight price is predicted and ticket price to happen in future, so that the travellers will overcome the problem of how much ticket price will happen for each Airlines for the current month or upcoming months, In this project, we work on the simpler problem that is to predict the ticket price for the airline and with other criteria using Machine Learning & Deep Learning Models finding out the best.

1.3 Business or Enterprise under study

India had the world's third-largest civil aviation market in 2017, with the number of passengers growing at an average annual rate of 16.3% between 2000 and 2015. Despite this growth, much of the country's aviation potential remains untapped. IndiGo, Air India, SpiceJet and GoAir are the major carriers in order of their market share. These airlines connect more than 80 cities across India, and are joined by several foreign airlines in providing international

routes. It recorded an air traffic of 131 million passengers in 2016., as these passengers are are unable to find the ticket price as the travelers saying that flight ticket prices are so unpredictable., will help the passengers and increase the revenue aviation carriers

1.4 Business Model of Enterprise

Selecting the relevant variables from the dataset and arranging their values in order of importance to creating models to predict the probability of ticket price for the airline by performing different types of algorithms on the data.

1.5 Data Sources

Got the dataset through machinehack.com and will be utilized for analysis,

The data contains:

FEATURES: Airline: The name of the airline.

Date_of_Journey: The date of the journey

Source: The source from which the service begins.

Destination: The destination where the service ends.

Route: The route taken by the flight to reach the destination.

Dep_Time: The time when the journey starts from the source.

Arrival_Time: Time of arrival at the destination.

Duration: Total duration of the flight.

Total_Stops: Total stops between the source and destination.

Additional_Info: Additional information about the flight

Price: The price of the ticket

Data Set Description:

Contains 10683 rows and 11 columns

1.6 Tools & Techniques

Tools: Jupyter Notebook.

Techniques:

- Gradient Boosting Regressor
- Extreme Gradient Boosting Regressor
- Decision Tree Regressor
- Random Forest Regressor
- Artificial Neural Network

CHAPTER 2
DATA PREPARATION AND UNDERSTANDING

One of the first steps we engaged in was to outline the sequence of steps that we will be following for our project. Each of these steps are elaborated below
After importing the required libraries, a sequence of steps was followed to perform data pre-processing.

2.1 Feature Engineering

Feature engineering using **domain knowledge** of the data to create features that make machine learning algorithms work. If feature engineering is done correctly, it increases the predictive power of machine learning algorithms by creating features from raw data that help facilitate the machine learning process. As the Feature Engineering is an art.

2.1.0. Exploratory Data Analysis (EDA)

We will explore our Data set and perform the exploratory data analysis.

2.1.1. Handling missing value

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

: Airline            0
  Date_of_Journey    0
  Source             0
  Destination        0
  Route              1
  Dep_Time           0
  Arrival_Time       0
  Duration           0
  Total_Stops        1
  Additional_Info    0
  Price              0
  dtype: int64

: df = df.dropna()
```

We can see that we have various missing values in the respective columns. There are various ways of treating your missing values in the data set. And which technique to use when is actually dependent on the type of data you are dealing with.

2.1.2. Handling Duplicate records

```
duplicate.sum()
```

```
220
```

```
df.drop_duplicates(inplace=True)
```

```
duplicate = df.duplicated()
duplicate
```

```
0        False
1        False
2        False
3        False
4        False
         ...
10678    False
10679    False
10680    False
10681    False
10682    False
Length: 10462, dtype: bool
```

```
duplicate.sum()
```

```
0
```

Since we have 220 duplicate records in the data, we will remove this from the data set so that we get only distinct records. Post removing the duplicate, we will check whether the duplicates have been removed from the data set or not.

2.1.3. Handling Outlier

```
#Quantity outlier check
Q1=df['Price'].quantile(0.25)
Q3=df['Price'].quantile(0.75)
IQR=Q3-Q1
```

```
Q1
```

```
5224.0
```

```
Q3
```

```
12344.75
```

```
IQR
```

```
7120.75
```

Outliers, being the most extreme observations, may include the sample maximum or sample minimum, or both, depending on whether they are extremely high or low. However, the sample maximum and minimum are not always outliers because they may not be unusually far from other observations.

2.1.4. Correlation

Correlation explains how one or more variables are related to each other. These variables can be input data features which have been used to forecast our target variable

```
df.corr()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
Airline	1.000000	0.014944	-0.013853	0.023310	0.002370	-0.018569	-0.016879	0.027431	0.037995	-0.108317	-0.026710
Date_of_Journey	0.014944	1.000000	0.149661	-0.250873	0.257569	-0.011117	-0.010311	-0.000430	-0.052247	-0.005269	0.019842
Source	-0.013853	0.149661	1.000000	-0.581120	0.384375	0.057423	0.022712	-0.195026	-0.233770	-0.036431	0.073826
Destination	0.023310	-0.250873	-0.581120	1.000000	-0.433271	-0.066174	-0.042090	0.018992	0.354053	0.046003	-0.166970
Route	0.002370	0.257569	0.384375	-0.433271	1.000000	-0.085590	0.018393	-0.054934	-0.448949	0.031258	0.240948
Dep_Time	-0.018569	-0.011117	0.057423	-0.066174	-0.085590	1.000000	-0.030878	0.041898	0.045662	-0.063272	-0.000834
Arrival_Time	-0.016879	-0.010311	0.022712	-0.042090	0.018393	-0.030878	1.000000	0.025699	-0.103350	0.013611	0.049222
Duration	0.027431	-0.000430	-0.195026	0.018992	-0.054934	0.041898	0.025699	1.000000	0.174272	0.075717	-0.165071
Total_Stops	0.037995	-0.052247	-0.233770	0.354053	-0.448949	0.045662	-0.103350	0.174272	1.000000	0.165082	-0.623468
Additional_Info	-0.108317	-0.005269	-0.036431	0.046003	0.031258	-0.063272	0.013611	0.075717	0.165082	1.000000	-0.045193
Price	-0.026710	0.019842	0.073826	-0.166970	0.240948	-0.000834	0.049222	-0.165071	-0.623468	-0.045193	1.000000

2.1.5. Encoding

There are many ways to convert categorical values into numerical values. But we used Label-Encoder are used to convert text or categorical data into numerical data in which the model expects and perform better with.

```
from sklearn.preprocessing import LabelEncoder
```

```
for x in df.columns:
    if  x!= 'Stop'and  x!= 'Price':
        df[x] = LabelEncoder().fit_transform(df[x])
```

2.1.6. Normalizing and Scaling

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since the range of values of data may vary widely, it becomes a necessary step in data pre-processing while using machine learning algorithms.

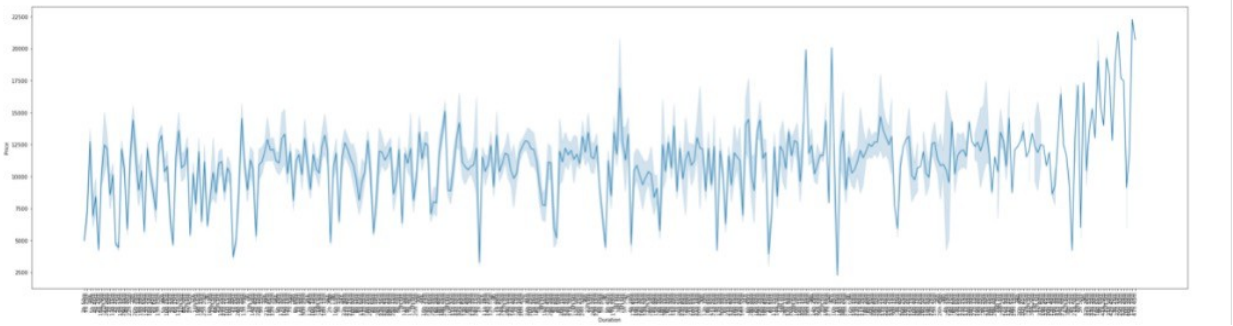
```
from sklearn.preprocessing import scale
```

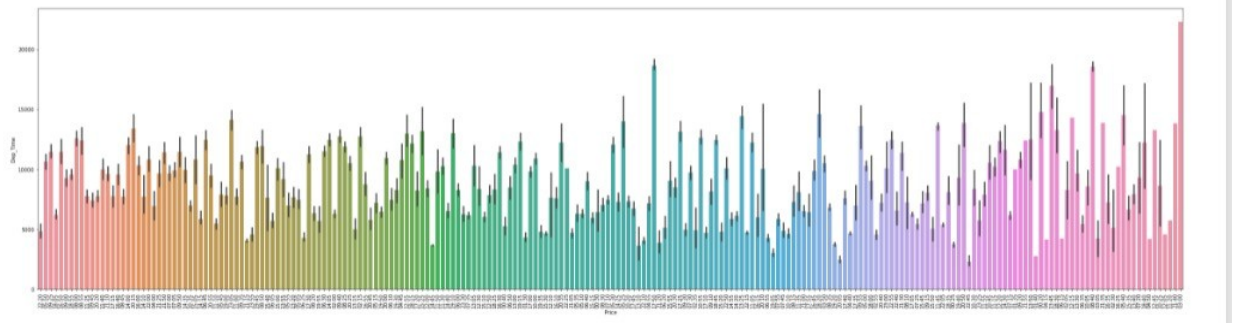
```
x_idep = scale(x_idep)
x_idep
```

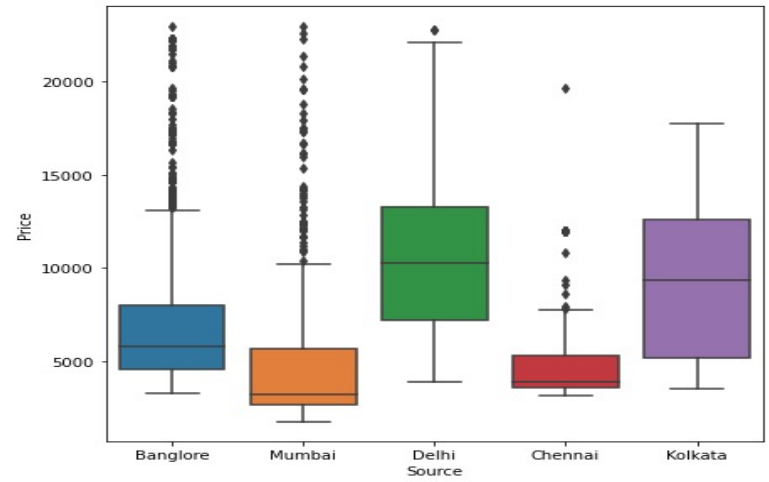
2.2 Data Visualization

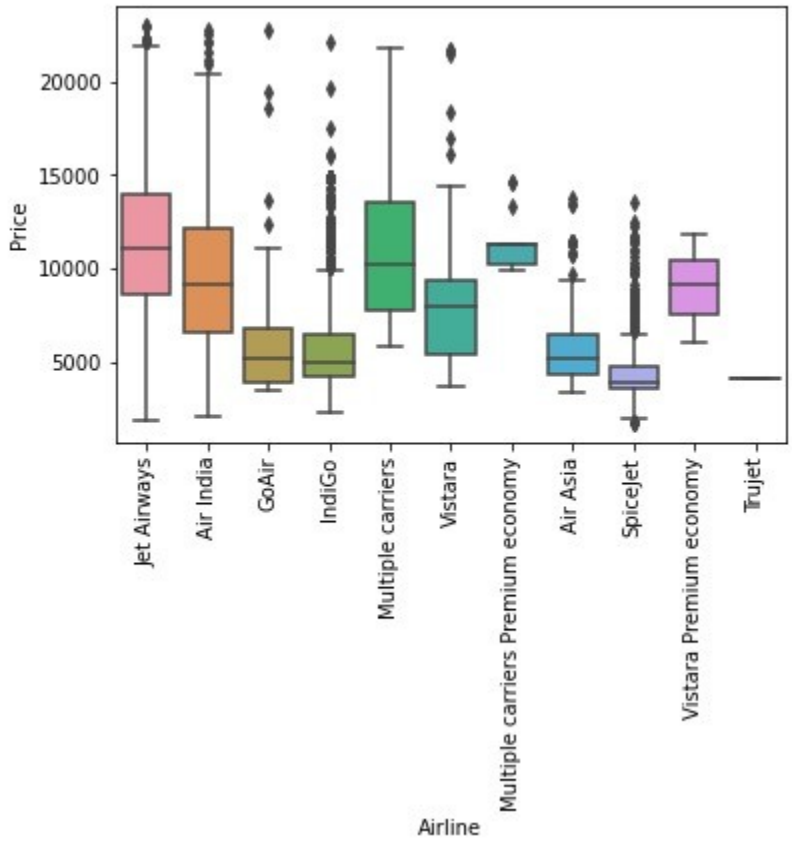
2.2.1. Bivariate Analysis

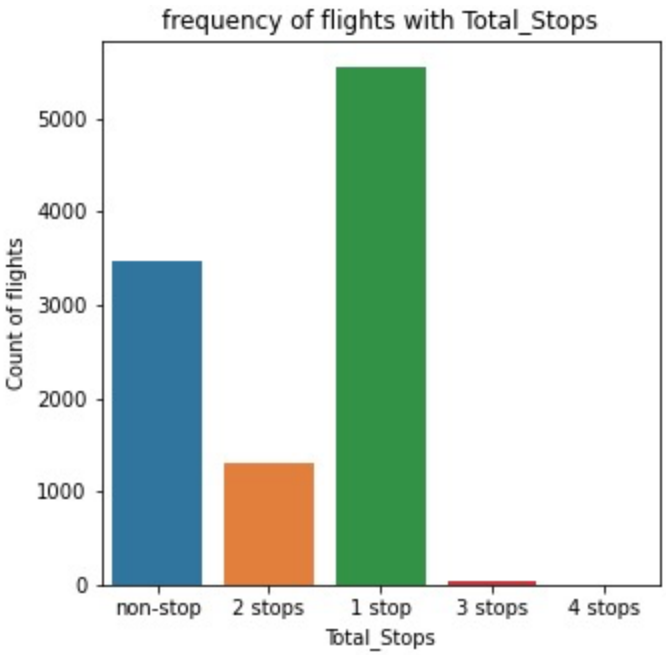
When we talk about bivariate analysis, it means analyzing 2 variables. Since we know there are numerical and categorical variables, there is a way of analyzing these variables as shown below:

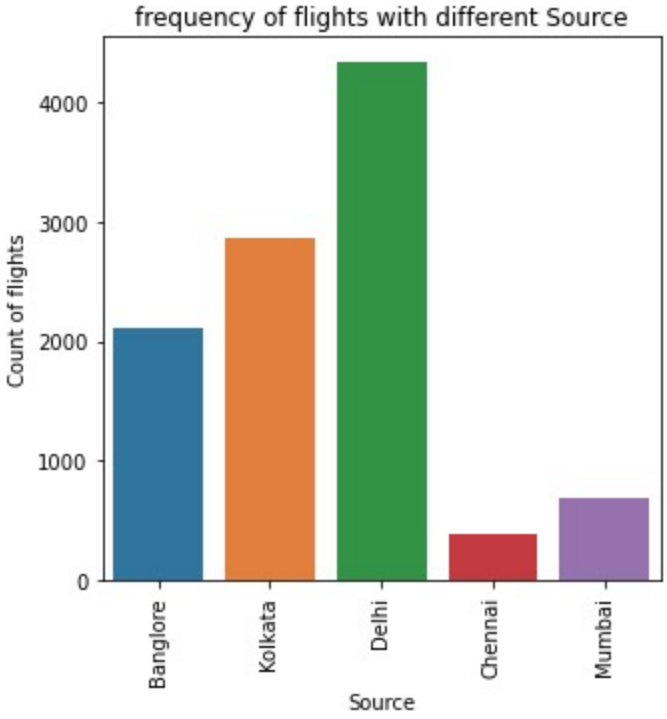


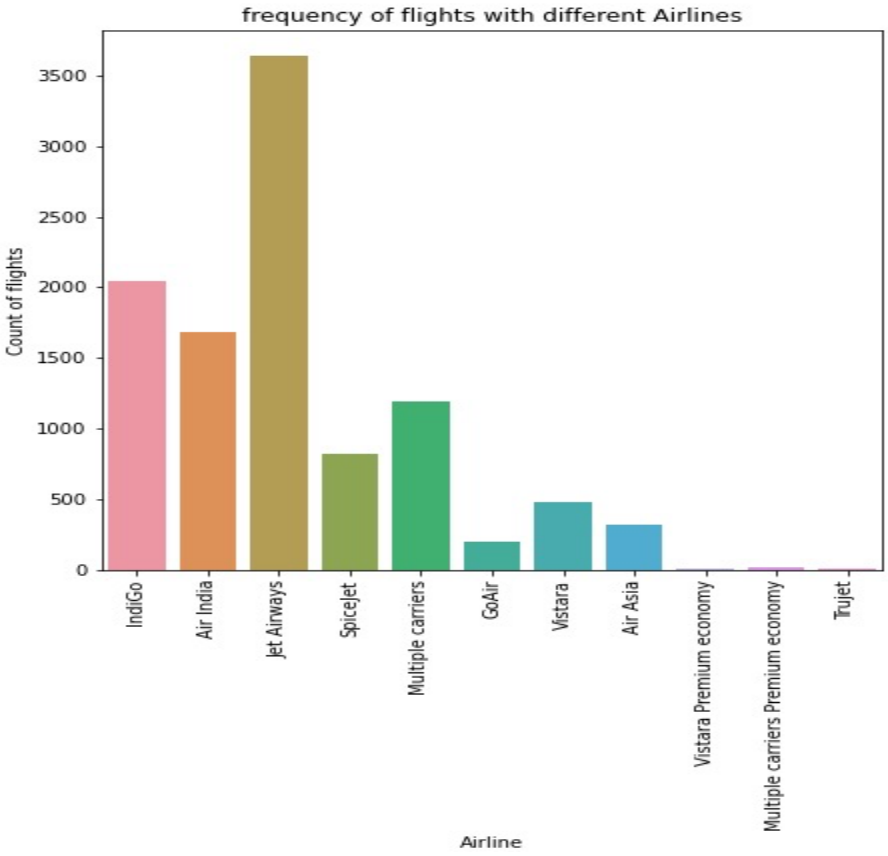



20000
15000
10000
5000
Price
Banglore
Mumbai
Delhi
Chennai
Kolkata
Source


20000
15000
10000
5000
Price
Jet Airways
Air India
GoAir
IndiGo
Multiple carriers
Vistara
Multiple carriers Premium economy
Air Asia
SpiceJet
Vistara Premium economy
Trujet
Airline


frequency of flights with Total_Stops
Count of flights
5000
4000
3000
2000
1000
0
non-stop
2 stops
1 stop
3 stops
4 stops
Total_Stops


frequency of flights with different Source
Count of flights
4000
3000
2000
1000
0
Banglore
Kolkata
Delhi
Chennai
Mumbai
Source


frequency of flights with different Airlines
Count of flights
3500
3000
2500
2000
1500
1000
500
0
IndiGo
Air India
Jet Airways
SpiceJet
Multiple carriers
GoAir
Vistara
Air Asia
Vistara Premium economy
Multiple carriers Premium economy
Trujet
Airline

2.3 Feature Selection

2.3.1 Dropping Constant Features Using Variance Threshold

The variance threshold is a simple baseline approach to feature selection. It removes all features which variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e., features that have the same value in all samples. We assume that features with a higher variance may contain more useful information, but note that we are not taking the relationship between feature variables or feature and target variables into account, which is one of the drawbacks of filter methods.

```
: ### It will zero variance features
  from sklearn.feature_selection import VarianceThreshold
  var_thres=VarianceThreshold(threshold=0)
  var_thres.fit(df)

: VarianceThreshold(threshold=0)

: df.columns[var_thres.get_support()]

: Index(['Airline', 'Date_of_Journey', 'Source', 'Destination', 'Route',
         'Dep_Time', 'Arrival_Time', 'Duration', 'Total_Stops',
         'Additional_Info', 'Price'],
        dtype='object')
```

No features that have the same value in all samples, none of the feature is removed as of now.

2.3.2 Dropping Constant Features Using Pearson Correlation

A Pearson correlation is a number between -1 and 1 that indicates the extent to which variables are correlated; we can predict one from the other. Therefore, if two features are correlated, the model only really needs one of them, as the second one does not add additional information. We will use the Pearson Correlation here.

```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature

def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

```
corr_features = correlation(df, 0.6)
len(set(corr_features))
```

Pearson correlations are suitable only for metrics:

The correlation coefficient has values between -1 to 1

- A value closer to 0 implies weaker correlation (exact 0 implying no correlation)
- A value closer to 1 implies stronger positive correlation

2.3.3 Information gain - mutual information In Regression

Mutual Information

Estimate mutual information for a continuous target variable.

Mutual information (MI) between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency.

The function relies on nonparametric methods based on entropy estimation from k-nearest neighbours' distances

Mutual information is calculated between two variables and measures the reduction in uncertainty for one variable given a known value of the other variable.

In short

A quantity called mutual information measures the amount of information one can obtain from one random variable given another.

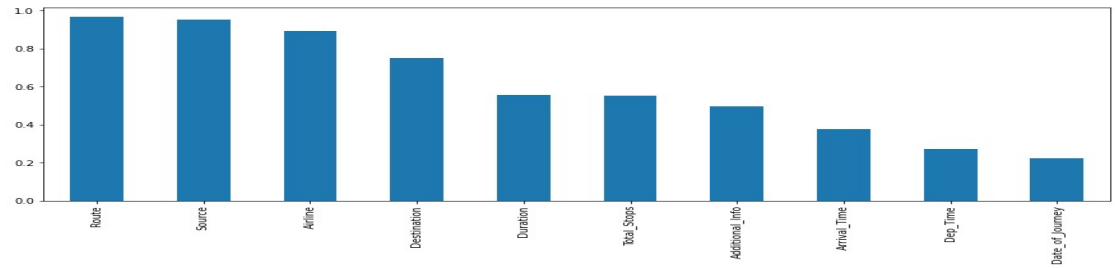
The mutual information between two random variables X and Y

```
mutual_info = pd.Series(mutual_info)
mutual_info.index = X_train.columns
mutual_info.sort_values(ascending=False)
```

```
Route              0.968007
Source             0.953546
Airline            0.891690
Destination        0.749894
Duration           0.556590
Total_Stops        0.551844
Additional_Info    0.495381
Arrival_Time       0.375987
Dep_Time           0.270697
Date_of_Journey    0.222395
dtype: float64
```

```
mutual_info.sort_values(ascending=False).plot.bar(figsize=(15,5))
```

```
<AxesSubplot:>
```




```
df=df.drop(['Route'], axis=1) #we don't need it as we already have total_stops
```

```
# As we found in mutual_info_regression 70 percent of fields are considered for analysis,
#as i feel droping only Duration and Arrival_Time will be helpful for analysis and
# and with deployment point of view passengar does not need to set arrival time
#and for small destination flight passenger cannot choose different longer duration hours, so we drop both
df=df.drop('Duration', axis=1)
df=df.drop('Arrival_Time', axis=1)
```

CHAPTER 3

MODEL BUILDING USING MACHINE LEARNING AND HYPERPARAMETER TUNING.

Process commonly used in Machine Learning Models.

3.0.1 Train-Test Split

In the development of machine learning models, it is desirable that the trained model perform well on new, unseen data. In order to simulate the new, unseen data, the available data is subjected to data splitting whereby it is split to 2 portions (sometimes referred to as the train-test split). Particularly, the first portion is the larger data subset that is used as the training set.

```
## It is always a good practice to split train and test data to avoid
#overfitting
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_indep,y_dep,
    test_size=0.2,
    random_state=0)
```

Once train-test split is done it is used same for all upcoming machine learning models

3.0.2 Cross-Validation

In order to make the most economical use of the available data, an ***N-fold cross-validation (CV)*** is normally used whereby the dataset is partitioned to *N* folds (*i.e.* commonly 5-fold or 10-fold CV are used). In such *N*-fold CV, one of the fold is left out as the testing data while the remaining folds are used as the training data for model building.

3.1 Gradient Boosting Regressor

A Gradient Boosting Machine combines the predictions from multiple decision trees to generate the final predictions. In this model that all the weak learners in a gradient boosting machine are decision trees.

Here – **the nodes in every decision tree take a different subset of features for selecting the best split.** This means that the individual trees aren't all the same and hence they are able to capture different signals from the data.

Additionally, each new tree takes into account the errors or mistakes made by the previous trees. So, every successive decision tree is built on the errors of the previous trees. This is how the trees in a gradient boosting machine algorithm are built sequentially.

```
: from sklearn import preprocessing
```

```
: from sklearn.ensemble import GradientBoostingRegressor
```

```
: from sklearn.model_selection import GridSearchCV, cross_val_score
```

```
: gb = GradientBoostingRegressor()
```

```
model_GBfit = model_GB.fit(X_train, y_train)
```

```
Predict_model1 = model_GBfit.predict(X_test)
Predict_model1
```

```
array([10463.23902885, 12251.62442735,  3715.79092577, ...,
       13340.04049273, 11618.2718098 , 13488.85834225])
```

```
df1=pd.DataFrame({"actual:":y_test,"predicted":Predict_model1}).round()
df1
```

```
gbm_r2 = r2_score(Actual,Predict_model1)
gbm_r2
```

```
0.905521062298469
```

```
gbm_rmse=math.sqrt(mean_squared_error(Actual,Predict_model1))
gbm_rmse
```

```
1245.592451958682
```

We get a Root Mean Square Error in GBM is 1245.59 so we apply some model in this dataset, we see which one has the best accuracy.

3.2 Extreme Gradient Boosting Regressor

XGBoost is another popular boosting algorithm. In fact, XGBoost is simply an improvised version of the GBM algorithm! The working procedure of XGBoost is the same as GBM. The trees in XGBoost are built sequentially, trying to correct the errors of the previous trees.

Additionally, if you are using the XGBM algorithm, you don't have to worry about imputing missing values in your dataset. **The XGBM model can handle the missing values on its own**. During the training process, the model learns whether missing values should be in the right or left node.

```
from xgboost import XGBRegressor
```

```
xgb = XGBRegressor()
```

```
best_xgb = GridSearchCV(xgb,
                        param_grid = {'max_depth': range(7, 15),
                                      'min_samples_split': range(1, 15)},
                        cv=5,
                        n_jobs=-1,
                        scoring='neg_mean_squared_error',refit=True)

best_xgb.fit(X_train, y_train)

print(best_xgb.best_params_)
print(-best_xgb.best_score_)
```

```
# Initiate the best model
best_xgb = XGBRegressor(max_depth=9, min_samples_split=1)
```

```
model_xGBfit=best_xgb.fit(X_train, y_train)
```

```
[16:58:24] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
Predict_model2 = model_xGBfit.predict(X_test)
```

```
xgbm_r2 = r2_score(Actual,Predict_model2)
xgbm_r2
```

```
0.903381926335712
```

```
xgbm_rmse=math.sqrt(mean_squared_error(Actual,Predict_model2))
xgbm_rmse
```

```
1259.614508484763
```

We get an Root Mean Square Error in XGBM **is 1259.61, so we apply some model in this dataset, we see which one the best accuracy.**

3.3 Decision Tree Regressor

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets based on the most significant splitter / differentiator in input variables. Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them. Its graphical representation is very intuitive and users can easily relate their hypothesis.. It requires less data cleaning compared to some other modeling techniques. It is not influenced by outliers and missing values to a fair degree.

```
from sklearn.tree import DecisionTreeRegressor
```

```
# Initiate the model
dt = DecisionTreeRegressor()
```

```
# Grid search
dt_gs = GridSearchCV(dt,param_grid = {'max_depth': range(1, 11),'min_samples_split': range(10, 60, 10)},cv=5,n_jobs=-1,
                     scoring='neg_mean_squared_error',refit=True)
```

```
dt_gs.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_jobs=-1,
             param_grid={'max_depth': range(1, 11),
                         'min_samples_split': range(10, 60, 10)},
             scoring='neg_mean_squared_error')
```

```
print(dt_gs.best_params_)
print(-dt_gs.best_score_)
```

```
{'max_depth': 10, 'min_samples_split': 10}
3027617.533573974
```

```
dt_r2 = r2_score(Actual,Predict_model3)
dt_r2
```

```
0.837813515894131
```

```
xgbm_rmse=math.sqrt(mean_squared_error(Actual,Predict_model3))
xgbm_rmse
```

```
1631.9836244914025
```

We get a Root Mean Square Error in DT is 1631.98, so we apply some model in this dataset, we see which one the best accuracy.

3.4 Random Forest Regressor

Random Forest is also a “Tree”-based algorithm that uses the qualities features of multiple Decision Trees for making decisions.

This data set by implementing the Random Forest Regression in place of the Decision Tree Regression. Additionally, the Random Forest algorithm is also very *fast* and *robust* than other regression models.

To summarize in short, The Random Forest Algorithm merges the output of multiple Decision Trees to generate the final output.

```
from sklearn.ensemble import RandomForestRegressor
```

```
Random_model = RandomForestRegressor()
```

```
# Grid search
rf_gs = GridSearchCV(dt,param_grid = {'max_depth': range(1, 15),'min_samples_split': range(10, 60, 1),
                     "max_features" : ["auto", "log2", "sqrt"]},
                    cv=5,n_jobs=-1,
                    scoring='neg_mean_squared_error',refit=True)
```

```
rf_gs.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_jobs=-1,
             param_grid={'max_depth': range(1, 15),
                         'max_features': ['auto', 'log2', 'sqrt'],
                         'min_samples_split': range(10, 60)},
             scoring='neg_mean_squared_error')
```

```
print(rf_gs.best_params_)
print(-rf_gs.best_score_)
```

```
{'max_depth': 14, 'max_features': 'auto', 'min_samples_split': 16}
2650862.6894814847
```

```
rf_r2 = r2_score(Actual,Predict_model4)
rf_r2
```

```
0.8777944621371194
```

```
rf_rmse=math.sqrt(mean_squared_error(Actual,Predict_model4))
rf_rmse
```

```
1416.6217664966707
```

We get a Root Mean Square Error in Random Forest Regression is1416.62 is the Good one to fit the model in this sales quantity prediction.

CHAPTER 4

MODEL BUILDING USING DEEP LEARNING.

Artificial Neural Network

Artificial Neural Network is capable of learning any nonlinear function. Hence, these networks are popularly known as **Universal Function Approximators**. ANNs have the capacity to learn weights that map any input to the output.

perceptron (or neuron) can be imagined as a Logistic Regression. Artificial Neural Network, or ANN, is a group of multiple perceptron / neurons at each layer. ANN is also known as a **Feed-Forward Neural network** because inputs are processed only in the forward direction:

As you can see here, ANN consists of 3 layers – Input, Hidden and Output. The input layer accepts the inputs, the hidden layer processes the inputs, and the output layer produces the result. Essentially, each layer tries to learn certain weights.

```
: from keras.models import Sequential
  from keras.layers import Dense

  Using TensorFlow backend.

: model1 = Sequential()
```

4.1. Data Preprocessing

Feature scaling (also known as data normalization) is the method used to standardize the range of features of data. Since the range of values of data may vary widely, it becomes a necessary step in data preprocessing while using deep learning algorithms.

```
from sklearn.preprocessing import scale
```

```
x_idep = scale(x_idep)
x_idep
```

```
array([[-0.55202326,  0.31871508,  1.10527883, ..., -0.08476649,
        -0.28287852, -0.12051453],
       [-0.55202326, -0.79282645, -0.23790172, ..., -0.22945214,
        -0.21066638, -0.17594168],
       [-0.55202326, -0.79282645, -0.13458014, ..., -0.24730177,
        -0.23911359, -0.09694657],
       ...,
       [-1.11561216,  0.31871508,  1.10527883, ...,  0.34145566,
        -0.26646667,  0.14529423],
       [-1.11561216,  0.31871508,  1.10527883, ..., -0.14704654,
        -0.28287852, -0.11210906],
       [-1.11561216,  0.31871508,  1.10527883, ...,  0.26082801,
        -0.28287852, -0.11210906]])
```

4.2. Training and Model Fitting

The first thing we need to do before building a model is to create a model object itself, this object will be an instance of the class called Sequential.

```
from sklearn.model_selection import train_test_split
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x_idep,y,test_size=0.2)
```

```
model1 = Sequential()
```

```
input_shape = xtrain[0].shape
```

```
model1.add(Dense(399,input_shape=input_shape))
```

```
model1.add(Dense(499,activation='relu'))
```

```
model1.add(Dense(599,activation='relu'))
```

```
model1.add(Dense(1,activation='relu'))
```

```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 399)               3192
_________________________________________________________________
dense_2 (Dense)              (None, 499)               199600
_________________________________________________________________
dense_3 (Dense)              (None, 599)               299500
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 600
=================================================================
Total params: 502,892
Trainable params: 502,892
Non-trainable params: 0
_________________________________________________________________
```

We use model.fit() method to train the model, we pass three arguments inside the method which are

input → x_train is the input that is fed to the network

output → this contains the correct answers for the x_train i.e. y_train

no.of.epochs → It means the number of times you are going to train the network with the dataset.

4.3. Evaluating the Model

We are going to evaluate the performance of the model by applying it to the test set. Here evaluate method returns two arguments one is the loss incurred in the test set by the prediction and the other one is the accuracy.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import math
```

```
model1.predict(xtest)
```

```
array([[12983.311],
       [11578.917],
       [11183.816],
       ...,
       [14597.774],
       [ 7736.518],
       [ 4132.896]], dtype=float32)
```

```
mean_absolute_error(ytest,model1.predict(xtest))
```

```
1305.957789333715
```

```
ann_r2 = r2_score(ytest,model1.predict(xtest))
ann_r2
```

```
0.7963019338617898
```

```
ann_rmse = math.sqrt(mean_squared_error(ytest,model1.predict(xtest)))
ann_rmse
```

```
1816.210054467495
```

CHAPTER 5

DEPLOYMENT

Deployment is the method by which to integrate a machine learning model into an existing production environment to make practical business decisions based on data. It is one of the last stages in the machine learning life cycle.

5.1. Gradio

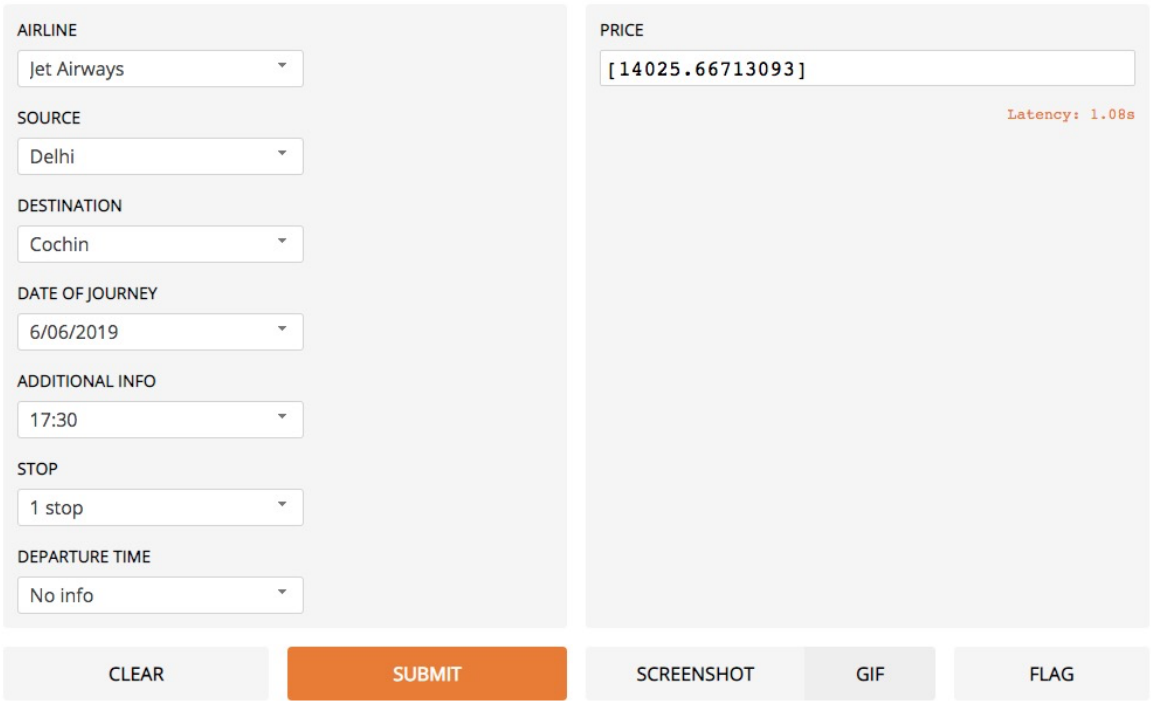
Gradio helps in building an online GUI in a couple of lines of code which is convenient for showing exhibitions of the model presentation. It is quick, simple to set up, and prepared to utilize, and shareable as the public connection which anybody can get to distantly and parallelly run the model in your machine. Gradio **works with a wide range of media-text, pictures, video, and sound.**

```
def mod(Airline,Date_of_Journey, Source, Destination, Dep_Time,Total_Stops,Additional_Info):
    x1 = [Airline,Date_of_Journey, Source, Destination, Dep_Time,Total_Stops,Additional_Info]
    test=pd.read_excel("Flight_Test.xlsx")
    test=test.drop(['Duration'],axis=1)
    test=test.drop(['Arrival_Time'],axis=1)
    test=test.drop(['Route'],axis=1)
    x1 =pd.DataFrame(np.array(x1).reshape(1,-1),columns=test.columns)
    test = test.append(x1,ignore_index = True)
    for x in test.columns:
        test[x] = LabelEncoder().fit_transform(test[x])
    x2 = test.iloc[-1,:]
    pred =  model_GBfit.predict(np.array(x2).reshape(1,-1))
    return pred
```

```
gd.Interface(mod, [Airline, Source, Destination, Date_of_Journey,Dep_Time,Total_Stops,Additional_Info],
             gd.outputs.Textbox(label='Price')).launch()
```

```
Running locally at: http://127.0.0.1:7860/
To create a public link, set `share=True` in `launch()`.
Interface loading below...
```

By entering the values of independent variables (Airline, Source, Destination, Departure time, Total no of stops and with Additional) and output dependent variable (Ticket Price) is gets predicted using machine learning model and displays on the quantity output textbox using Gradio user interface



CHAPTER 6

CONCLUSION

Out of the Machine learning and Deep learning algorithms used, Gradient boosting Regressor model gave us the least RMSE score.

	R2	RMSE
Gradient Boosting Regressor	0.901398	1272.484243
eXtreme Gradient Boosting Regressor	0.903382	1631.983624
Random forest Regressor	0.876799	1422.381811
Decision Tree Regressor	0.837814	1631.983624
Artifical Neural Network	0.796302	1816.210054