FLIP ROBO

FLIGHT PRICE PREDICTION

Submitted by:

Purnima Pati

ACKNOWLEDGMENT

As the airline industry is flourishing various practices are being used by the airline industry which they call revenue management or yield management. Data visualization and machine learning algorithms are basis of the project airline price prediction. The use of logistic regression suggests the customer whether is it the right time to buy the airline ticket or should he wait. The process involves data extracting, analyzing, and interpreting which consist of data mining and decision making.

To complete this project I have gone through various case studies and project reports , how flight price prediction is useful for the public. I have gone through some articles available in google.
I wish to express our sincere thanks to the above people, without whom I would not have been able to complete this project.

INTRODUCTION

- Business Problem Framing

  Anyone who has booked a flight ticket knows how unexpectedly the prices vary. The cheapest available ticket on a given flight gets more and less expensive over time. This usually happens as an attempt to maximize revenue based on - 1. Time of purchase patterns (making sure last-minute purchases are expensive) 2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases) So, you have to work on a project where you collect data of flight fares with other features and work to make a model to predict fares of flights.

- Conceptual Background of the Domain Problem

  Anyone who has booked a flight ticket knows how unexpectedly the prices vary. Airlines use using sophisticated quasi-academic tactics known as "revenue management" or "yield management". The cheapest available ticket for a given date gets more or less expensive over time. This usually happens as an attempt to maximize revenue based on

  1. Time of purchase patterns (making sure last-minute purchases are expensive)
  2. Keeping the flight as full as they want it (raising prices on a flight which is filling up in order to reduce sales and hold back inventory for those expensive last-minute expensive purchases)

  So, if we could inform the travellers with the optimal time to buy their flight tickets based on the historic data and also show them various trends in the airline industry we could help them save money on their travels. This would be a practical implementation of a data analysis, statistics and machine learning techniques to solve a daily problem faced by travellers.

- ## Review of Literature

  It is very difficult for the customer to purchase a flight ticket at the minimum price. For this several techniques are used to obtain the day at which the price of air ticket will be minimum. Most of these techniques are using sophisticated artificial intelligence(AI) research is known as Machine Learning.

  Utilizing AI models, connected PLSR(Partial Least Square Regression) model to acquire the greatest presentation to get the least cost of aircraft ticket buying, having 75.3% precision. Janssen presented a direct quantile blended relapse model to anticipate air ticket costs for cheap tickets numerous prior days takeoff. Ren, Yuan, and Yang , contemplated the exhibition of Linear Regression (77.06% precision), Naive Bayes (73.06% exactness, Softmax Regression (76.84% precision) and SVM (80.6% exactness) models in anticipating air ticket costs. Papadakis anticipated that the cost of the ticket drop later on, by accepting the issue as a grouping issue with the assistance of Ripple Down Rule Learner (74.5 % exactness.), Logistic Regression with 69.9% precision and Linear SVM with the (69.4% exactness) Machine Learning models.

  Gini and Groves took the Partial Least Square Regression(PLSR) for developing a model of predicting the best purchase time for flight tickets. The data was collected from major travel journey booking websites from 22 February 2011 to 23 June 2011. Additional data were also collected and are used to check the comparisons of the performances of the final model.

  Janssen built up an expectation model utilizing the Linear Quantile Blended Regression strategy for SanFrancisco to NewYork course with existing every day airfares given by www.infare.com. The model utilized two highlights including the number of days left until the takeoff date and whether the flight date is at the end of the week or weekday. The model predicts airfare well for the days that are a long way from the takeoff date, anyway for a considerable length of time close the takeoff date, the expectation isn’t compelling.

Wohlfarth proposed a ticket buying time enhancement model dependent on an extraordinary pre-preparing step known as macked point processors and information mining systems (arrangement and bunching) and measurable investigation strategy. This system is proposed to change over heterogeneous value arrangement information into added value arrangement direction that can be bolstered to unsupervised grouping calculation. The value direction is bunched into gathering dependent on comparative estimating conduct. Advancement model gauge the value change designs. A treebased order calculation used to choose the best coordinating group and afterward comparing the advancement mode
A study by Dominguez-Menchero recommends the ideal buying time dependent on nonparametric isotonic relapse method for a particular course, carriers, and timeframe. The model gives the most extreme number of days before buying a flight ticket. two sorts of the variable are considered for the expectation. One is the passage and date of procurement.

- ## Motivation for the Problem Undertaken

This project aims to develop an application which will predict the flight prices for various flights using machine learning model. The user will get the predicted values and with its reference the user can decide to book their tickets accordingly. In the current day scenario flight companies try to manipulate the flight ticket prices to maximize their profits. There are many people who travel regularly through flights and so they have an idea about the best time to book cheap tickets. But there are also many people who are inexperienced in booking tickets and end up falling in discount traps made by the companies where actually they end up spending more than they should have.
 The proposed system can help save millions of rupees of customers by proving them the information to book tickets at the right time. The proposed problem statement is “Flight Fare prediction system”.

Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem

  Data preparation is followed by analysing the data, uncovering hidden trends and then applying various predictive & classification models on the training set. These included Random Forest, Logistic Regression, Gradient Boosting and combination of these models to increase the accuracy. Further statistical models and trend analyzer model have been built to increase the accuracy of the ML algorithms for this task.

- Data Sources and their formats

  The collection of data is the most important aspect of this project. There are various sources of the data on different websites which are used to train the models. Websites give information about the multiple routes, times, airlines and fare. Various sources from API's to consumer travel websites are available for data scraping. In this section details of the various sources and parameters that are collected are discussed. To implement this data is collected from a website "Makemytrip.com" and python is used for the implementation of the models and collection of the data
  Collection of data The script extracts the information from the website and creates a CSV file as output. This file contains the information with features and its details[13]. Now an important aspect is to select the features that might be needed for the flight prediction algorithm.

  Output collected from the website contains numerous variable for each flight but not all are required, so only the following feature is considered.

  - ✓ Origin
  - ✓ Destination
  - ✓ Departure Date
  - ✓ Departure Time
  - ✓ Arrival Time
  - ✓ Total Fare
  - ✓ Airways

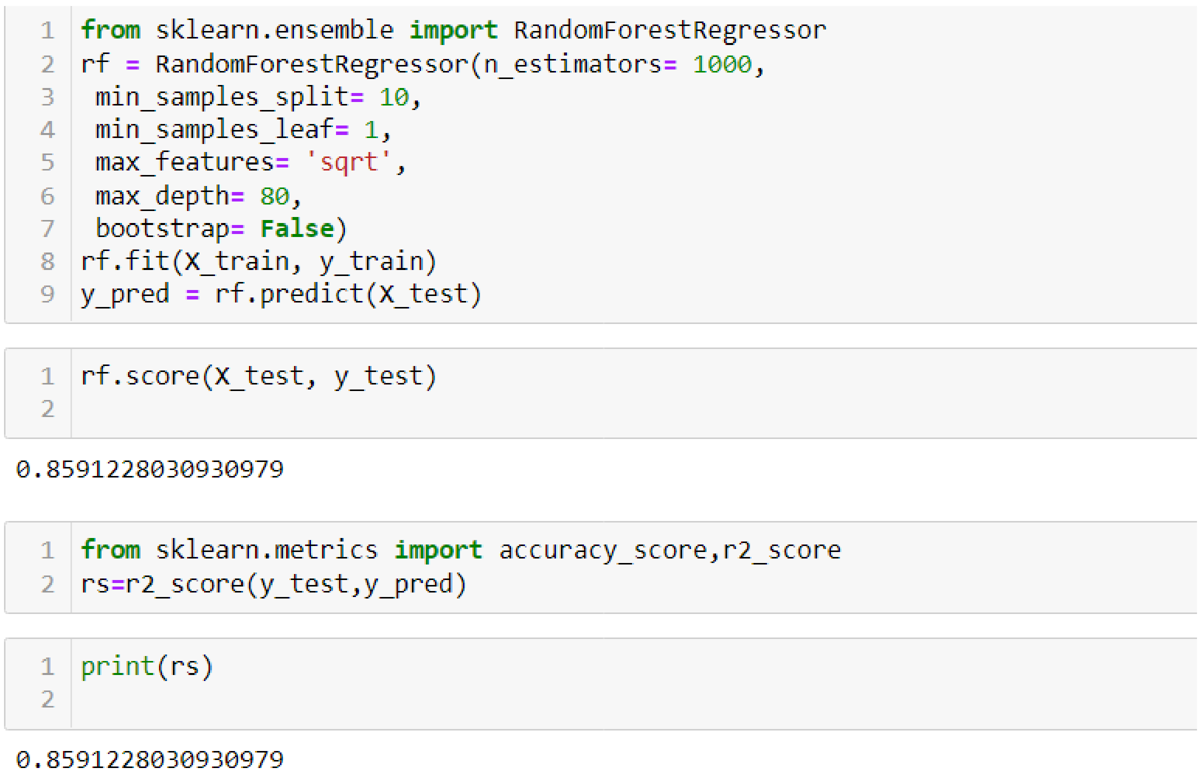
- Data Preprocessing Done

  All the collected data needed a lot of work so after the collection of data, it is needed to be clean and prepare according to the model requirements. All the unnecessary data is removed like duplicates and null values. In all machine learning this technology, this is the most important and time consuming step. Various statistical techniques and logic built-in python are used to clean and prepare the data. For example, the price was character type, not an integer.

- Data Inputs- Logic- Output Relationships

  Data preparation is followed by analyzing the data, uncovering the hidden trends and then applying various machine learning models. Also, some features can be calculated from the existing feature. Days to departure can be obtained by calculating the difference between the departure date and the date on which data is taken. This parameter is considered to be within 45 days. Also, the day of departure plays an important role in whether it is holiday or weekday. Intuitively the flights scheduled during weekends have a more price compared to the flights on Wednesday or Thursday. Similarly, time also seems to play an important factor. So the time is been divided into four categories: Morning, afternoon, evening, night.

  MACHINE LEARNING ALGORITHM To develop the model for the flight price prediction, many conventional machine learning algorithms are evaluated. They are as follows: Linear regression, Decision tree, Random Forest Algorithm, K-Nearest neighbors, Multilayer Perceptron, Support Vector Machine (SVM) and Gradient Boosting. All these models are implemented in the scikit learn. To evaluate the performance of this model, certain parameters are considered. They are as follows: R-squared value, Mean Absolute Error (MAE) and Mean Squared Error (MSE).



```python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000,
 min_samples_split= 10,
 min_samples_leaf= 1,
 max_features= 'sqrt',
 max_depth= 80,
 bootstrap= False)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```python
rf.score(X_test, y_test)
```

```
0.8591228030930979
```

```python
from sklearn.metrics import accuracy_score,r2_score
rs=r2_score(y_test,y_pred)
```

```python
print(rs)
```

```
0.8591228030930979
```

- State the set of assumptions (if any) related to the problem under consideration

  Though there are different algorithm to build model , I have used Random Forest

- Hardware and Software Requirements and Tools Used

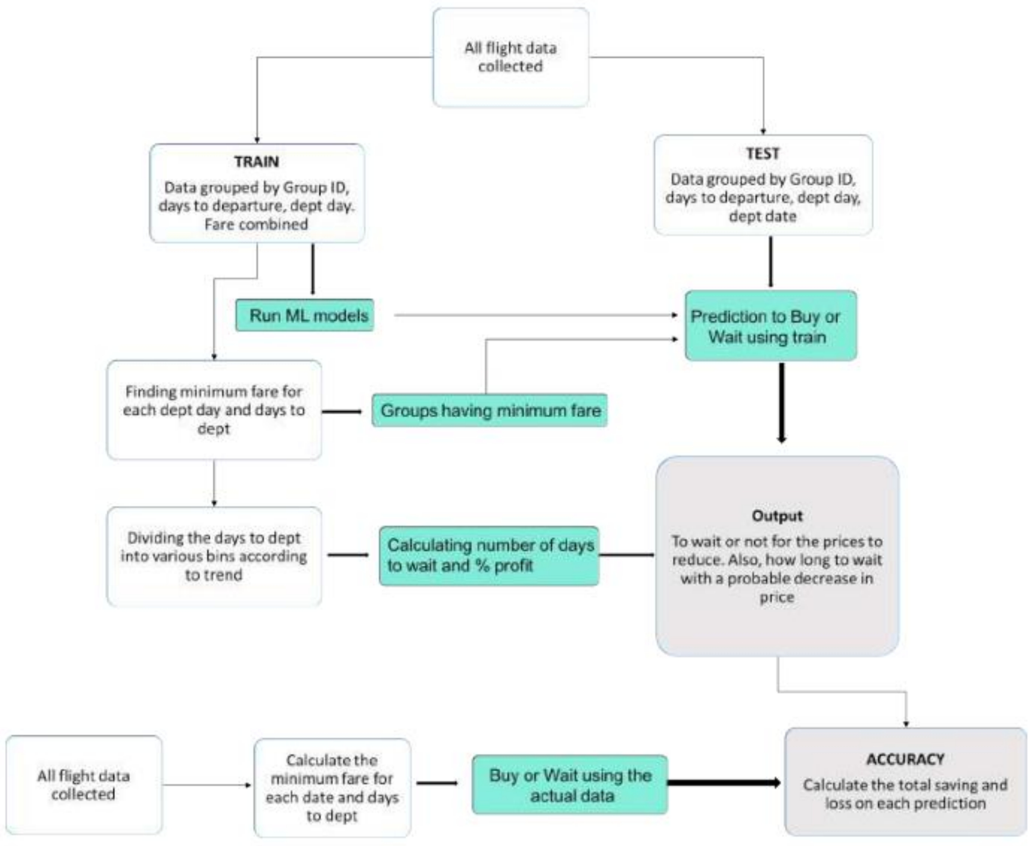
  Hardware requirements
  Operating system- Windows 10
  Processor- dual core 2.4 GHz (i5 series Intel processor or equivalent AMD)
  RAM-4GB

  Software Requirements
  Python
  PIP 2.7
  Jupyter Notebook
  Chrome

Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)

  Having built various models, we have to test the models on our testing set and calculate the savings or loss done on each query put by the user. A statistic of the over Savings, Loss and the mean saving per transaction are the measures used to calculate the Accuracy of the model implemented.



All flight data collected
TRAIN
Data grouped by Group ID, days to departure, dept day. Fare combined
TEST
Data grouped by Group ID, days to departure, dept day, dept date
Run ML models
Prediction to Buy or Wait using train
Finding minimum fare for each dept day and days to dept
Groups having minimum fare
Output
To wait or not for the prices to reduce. Also, how long to wait with a probable decrease in price
Dividing the days to dept into various bins according to trend
Calculating number of days to wait and % profit
All flight data collected
Calculate the minimum fare for each date and days to dept
Buy or Wait using the actual data
ACCURACY
Calculate the total saving and loss on each prediction

- Testing of Identified Approaches (Algorithms)

  Machine learning (ML) is the study of computer algorithms, which improve with experience and use of data. Machine learning algorithms build a model based on sample data (training data), and make predictions or decisions using this model without being programmed to do so.

Machine learning algorithms have a wide variety of applications, like fraud detections, email filtering etc. One such application of machine learning lies in the ‘Aviation industry’, to predict the prices of flights. There are various factors/features which impact the prices of flights — distance, flight time, number of stops etc. These factors help create a pattern to decide the price of a flight, and the machine learning models get trained on this pattern to make the predictions in future, automating the process and making the process quicker.

To develop the model for the flight price prediction, many conventional machine learning algorithms are evaluated. They are as follows: Linear regression, Decision tree, Random Forest Algorithm, K-Nearest neighbors, Multilayer Perceptron, Support Vector Machine (SVM) and Gradient Boosting. All these models are implemented in the scikit learn. To evaluate the performance of this model, certain parameters are considered. They are as follows: R-squared value, Mean Absolute Error (MAE) and Mean Squared Error (MSE).

Random Forest It is a supervised learning algorithm. The benefit of the random forest is, it very well may be utilized for both characterization and relapse issue which structure most of current machine learning framework. Random forest forms numerous decision trees, what’s more, adds them together to get an increasingly exact and stable expectation. Random Forest has nearly the equivalent parameters as a decision tree or a stowing classifier model. It is very simple to discover the significance of each element on the expectation when contrasted with others in this calculation. The regular component in these techniques is, for the kth tree, a random vector theta k is produced, autonomous of the past random vectors theta 1, ... , theta k-1 however with the equivalent distribution,while a tree is developed utilizing the preparation set and bringing about a classifier. x is an information vector. For a period, in stowing the random vector is created as the includes in N boxes where N is the number of models in the preparation set of information. In random split, choice includes various autonomous random whole numbers between 1 to K. The dimensionality and nature of theata rely upon its utilization in the development of a tree. After countless trees are created, they select the most famous class. These methodology are called as random forests

- ## Run and Evaluate selected models

  Random Forest It is a supervised learning algorithm. The benefit of the random forest is, it very well may be utilized for both characterization and relapse issue which structure most of current machine learning framework. Random forest forms numerous decision trees, what’s more, adds them together to get an increasingly exact and stable expectation. Random Forest has nearly the equivalent parameters as a decision tree or a stowing classifier model. It is very simple to discover the significance of each element on the expectation when contrasted with others in this calculation.

Model

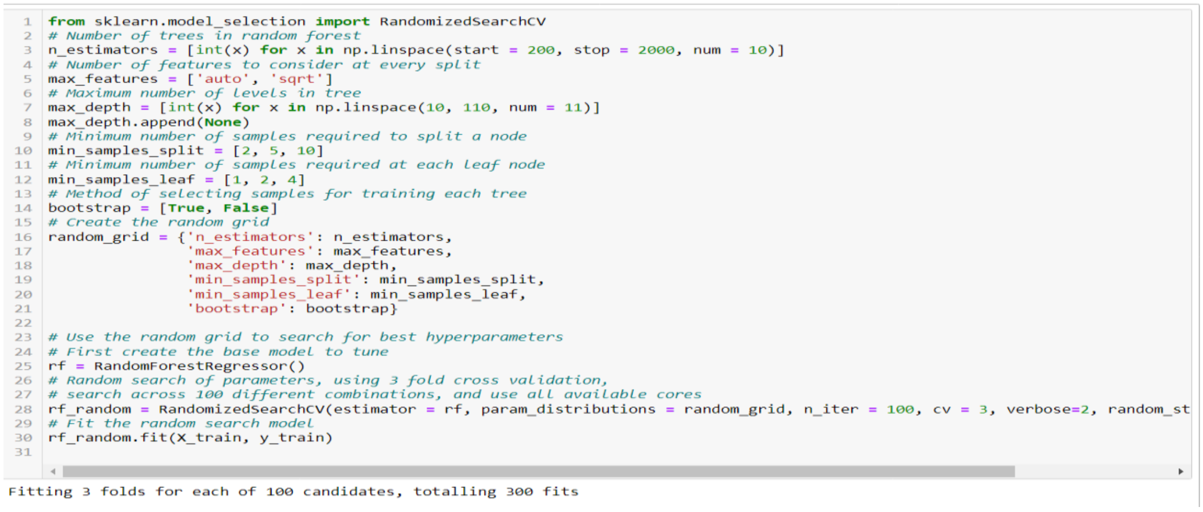
```
rf = RandomForestRegressor()
```

```
#Randomized Search CV
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]
```

```
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'max_dept
h': [5, 10, 15, 20, 25, 30], 'min_samples_split': [2, 5, 10, 15, 100], 'min_samples_leaf': [1, 2, 5, 10]}
```



```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
# Fit the random search model
rf_random.fit(X_train, y_train)

```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```python
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
# Fit the random search model
rf_random.fit(X_train, y_train)

```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

]:
```python
# Random search of parameters, using 3 fold cross validation,
# search across 50 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter =

```

]:
```python
rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.5s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.2s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.4s
[CV] END max_depth=5, max_features=sqrt, min_samples_leaf=10, min_samples_split=100, n_estimators=400; total time=   1.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   3.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   2.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   2.3s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   2.5s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_split=5, n_estimators=200; total time=   3.5s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=200; total time=   0.6s
[CV] END max_depth=25, max_features=sqrt, min_samples_leaf=10, min_samples_split=15, n_estimators=200; total time=   0.6s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=600; total time=   7.7s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=600; total time=   6.4s
[CV] END max_depth=20, max_features=auto, min_samples_leaf=5, min_samples_split=15, n_estimators=600; total time=   7.0s
```

]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

```python
rf_random.best_params_

```

```
{'n_estimators': 1400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 100,
 'bootstrap': True}
```

```python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000,
 min_samples_split= 10,
 min_samples_leaf= 1,
 max_features= 'sqrt',
 max_depth= 80,
 bootstrap= False)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```python
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
# Fit the random search model
rf_random.fit(X_train, y_train)

```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

```python
rf_random.best_params_
```

```
{'n_estimators': 1400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 100,
 'bootstrap': True}
```

```python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000,
 min_samples_split= 10,
 min_samples_leaf= 1,
 max_features= 'sqrt',
 max_depth= 80,
 bootstrap= False)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```python
rf.score(X_test, y_test)
```

```
0.8591228030930979
```

```python
from sklearn.metrics import accuracy_score,r2_score
rs=r2_score(y_test,y_pred)
```

```python
print(rs)
```

```
0.8591228030930979
```

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
print(rmse)
```

```
1727.107658308885
```

```python
print(1 - np.sqrt(np.square(np.log10(y_pred +1) - np.log10(y_test +1)).mean()))
```

```
0.9371839596336282
```

- Key Metrics for success in solving problem under consideration

- The Route column contains a list of cities which we will need to separate, since we would have multiple combinations in our dataset.

- The Arrival time column has dates attached along with, which we will need to separate. These are the cases when the flight takes off from the source on a date and reaches its destination on the next day.

- The Duration is in a string format, which we will need to convert to integer type.

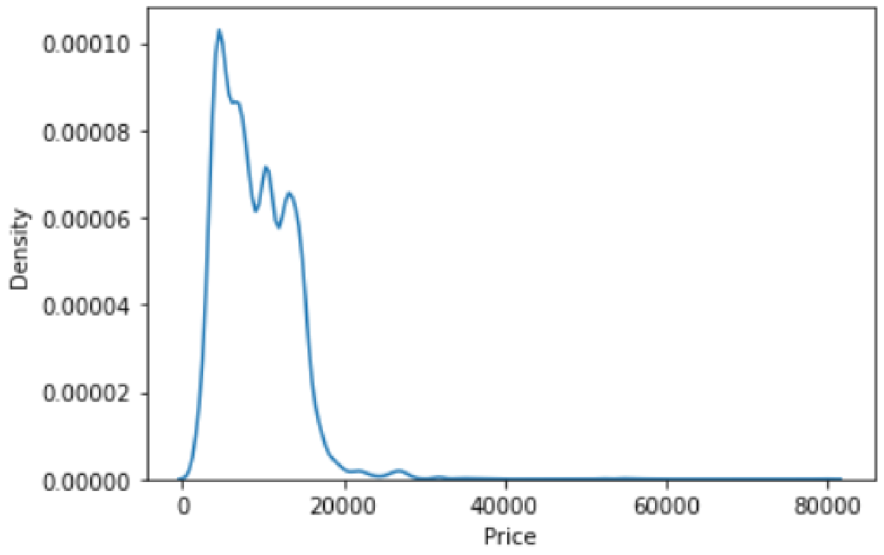
- The total stops also has text 'stops' added along with the number of stops, and certain columns as 'non-stop', which we will need to convert to integer types.

  We further proceed to explore the dataset.

- Visualizations
  ```
  big_df.dtypes
  big_df.isnull().sum()
  big_df[big_df['Total_Stops'].isnull()]
  big_df['Route'].count()
  big_df.isnull().sum()
  ```

```
sns.kdeplot(data = x)
```

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



```
df_train = big_df[0:10683]
df_test = big_df[10683:]
```

```
X = df_train.drop(columns='Price')
y = df_train['Price']
```

```
from sklearn.model_selection import train_test_split
X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.3 , random_state = 0)
```

```
model = SelectFromModel(Lasso(alpha = 0.005 , random_state = 0))
```

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

```
SelectFromModel(estimator=Lasso(alpha=0.005, random_state=0))
```

```
model.get_support()
```

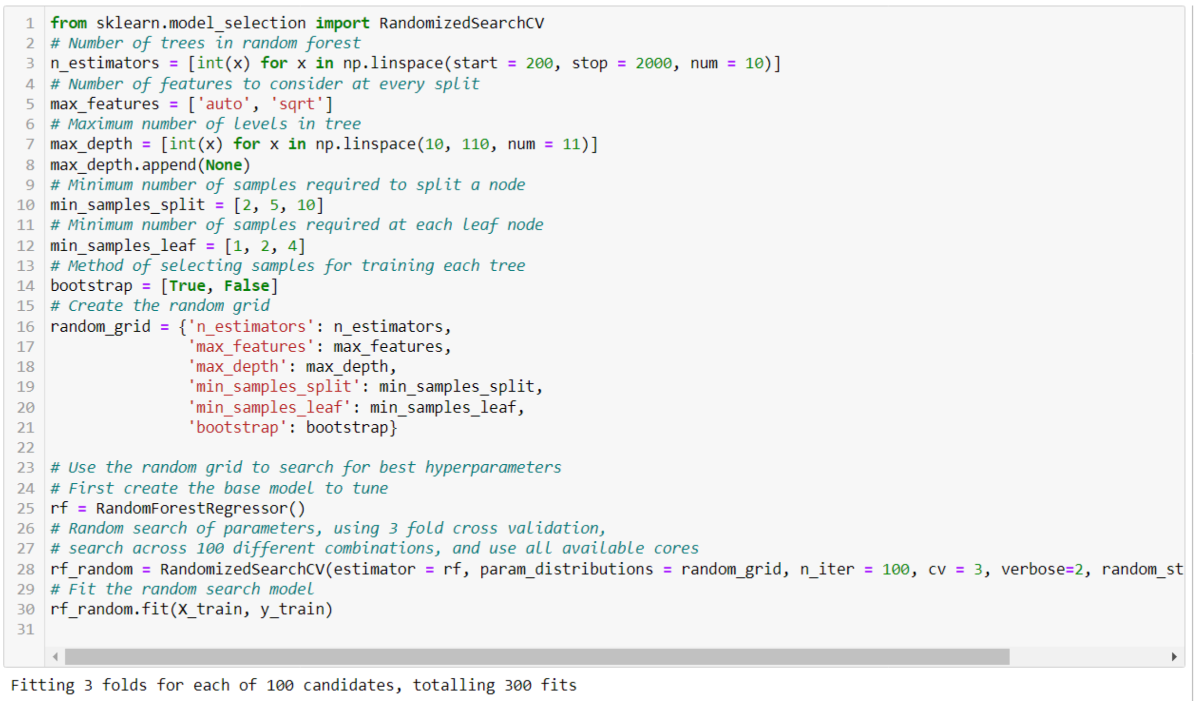
```
array([ True,  True,  True,  True,  True,  True, False,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False])
```

```
X_train.columns[(model.get_support())]
```

```
Index(['Airline', 'Source', 'Destination', 'Additional_Info', 'Date', 'Month',
       'Arrival_hour', 'Arrival_minute', 'Dep_hour', 'Dep_minute', 'stop',
       'Route1', 'Route2', 'Route3', 'Route4', 'Route5'],
      dtype='object')
```

```
# drop column which are not important
X_train.drop(columns=['year' , 'Route6'] , inplace =True)
X_test.drop(columns=['year' , 'Route6'] , inplace =True)
```

- Interpretation of the Results



```python
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_st
# Fit the random search model
rf_random.fit(X_train, y_train)

```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

```python
rf_random.best_params_

```

```
{'n_estimators': 1400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 100,
 'bootstrap': True}
```

```python
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators= 1000,
 min_samples_split= 10,
 min_samples_leaf= 1,
 max_features= 'sqrt',
 max_depth= 80,
 bootstrap= False)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

```python
rf.score(X_test, y_test)

```

```
0.8591228030930979
```

```python
from sklearn.metrics import accuracy_score,r2_score
rs=r2_score(y_test,y_pred)
```

```python
print(rs)

```

```
0.8591228030930979
```

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(y_test, y_pred))
print(rmse)

```

```
1727.107658308885
```

```python
print(1 - np.sqrt(np.square(np.log10(y_pred +1) - np.log10(y_test +1)).mean()))

```

```
0.9371839596336282
```

CONCLUSION

- ## Key Findings and Conclusions of the Study

  In the proposed paper the overall survey for the dynamic price changes in the flight tickets is presented. this gives the information about the highs and lows in the airfares according to the days, weekend and time of the day that is morning, evening and night , also the machine learning models in the computational intelligence field that are evaluated before on different datasets are studied. their accuracy and performances are evaluated and compared in order to get better result. For the prediction of the ticket prices perfectly different prediction models are tested for the better prediction accuracy. As the pricing models of the company are developed in order to maximize the revenue management. So to get result with maximum accuracy regression analysis is used. From the studies , the feature that influences the prices of the ticket are to be considered. In future the details about number of available seats can improve the performance of the model

- Learning Outcomes of the Study in respect of Data Science
    - ➢ The airfare varies depending on the time of departure, making timeslot used in analysis is an important parameter.

    - ➢ The airfare increases during a holiday season. In our time period, during Diwali the fare remained high for all the values of days to departure. We have considered holiday season as a parameter which helped in increasing the accuracy.

    - ➢ Airfare varies according to the day of the week of travel. It is higher for weekends and Monday and slightly lower for the other days.
    - ➢ There are a few times when an offer is run by an airline because of which the prices drop suddenly. These are difficult to incorporate in our mathematical models, and hence lead to error.

  ● Along the business routes, we find that the price of flights increases or remains constant as the days to departure decreases. This is because of the high frequency of the flights, high demand and also could be due to heavy competition.
- Limitations of this work and Scope for Future Work

✓ More routes can be added and the same analysis can be expanded to major airports and travel routes in India.

✓ The analysis can be done by increasing the data points and increasing the historical data used. That will train the model better giving better accuracies and more savings.

✓ More rules can be added in the Rule based learning based on our understanding of the industry, also incorporating the offer periods given by the airlines.

✓ Developing a more user friendly interface for various routes giving more flexibility to the users.