FLIP ROBO

# Flight Price Prediction

Submitted by:

YOUR NAME

# ACKNOWLEDGMENT

A unique opportunity like this comes very rarely. It is indeed a pleasure for me to have worked on this project. The satisfaction that accompanies the successful completion of this project is incomplete without the mention of the people whose guidance has made it possible for me to complete this project.I am grateful to my internship company **Flip Robo Technologies** with its ideals and inspiration for providing me with facilities that has made this project a success.

I am grateful to **Vaishali Singh** for providing the opportunity to work as a intern in **Flip Robo Technologies**

I like to acknowledge the effort put in by our SME **Sajid Choudhary** in helping me understand the relevant concepts related to Data preprocessing and providing guidance when necessary.

**ULLAS K C**

# 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

  Airline companies use complex algorithms to calculate flight prices given various conditions present at that particular time. These methods take financial, marketing, and various social factors into account to predict flight prices. Nowadays, the number of people using flights has increased significantly. It is difficult for airlines to maintain prices since prices change dynamically due to different conditions. That's why we will try to use machine learning to solve this problem. This can help airlines by predicting what prices they can maintain. It can also help customers to predict future flight prices and plan their journey accordingly.

- Review of Literature

  Nowadays, airline ticket prices can vary dynamically and significantly for the same flight, even for nearby seats within the same cabin. Customers are seeking to get the lowest price while airlines are trying to keep their overall revenue as high as possible and maximize their profit. Airlines use various kinds of computational techniques to increase their revenue such as demand prediction and price discrimination. From the customer side, two kinds of models are proposed by different researchers to save money for customers: models that predict the optimal time to buy a ticket and models that predict the minimum ticket price. In this paper, we present a review of customer side and airlines side prediction models. Our review analysis shows that models on both sides rely on limited set of features such as historical ticket price data, ticket purchase date and departure date. Features extracted from external factors such as social media data and search engine query are not considered. Therefore, we introduce and discuss the concept of using social media data for ticket/demand prediction.

- ## Motivation for the Problem Undertaken

  Motivated by previous studies, we can think of various additional useful features from social media that can possibly forecast airlines passenger demand and or ticket prices. For example, sentiment analysis of different twitter hash tags could convey the presence of some event at a flight origin/destination city that improves the prediction of ticket price/demand. This kind of feature extraction might involve searching for special keywords or group of terms, determining the number of times they appear, understanding the location and the date, their context etc.

## Analytical Problem Framing

- Mathematical/ Analytical Modeling of the Problem
- Mathematical Summary:

  Dimensions of Dataset: There are 9 columns and 2736 rows in this dataset

  Null Values: There are no null values in this dataset

  Skewness: Skewness is present in only in the Target column.

- Data Sources and their formats

  Standard deviation is very normal in most of the columns.

  There is not much difference between mean and 50th percentile, which means data is skewed

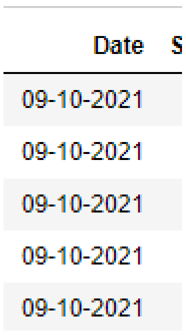
  There is not much difference between 75th percentile and max, which means there are outliers

- Data Sources and their formats

  All the objects were object data type after scraping before pre-processing.

- Data Pre-processing Done
- Feature Extraction:
  At first we had only 9 columns in which Date column consists of Day of the week and Days prior booking. After extracting the required fields using feature extraction we got 11 columns.

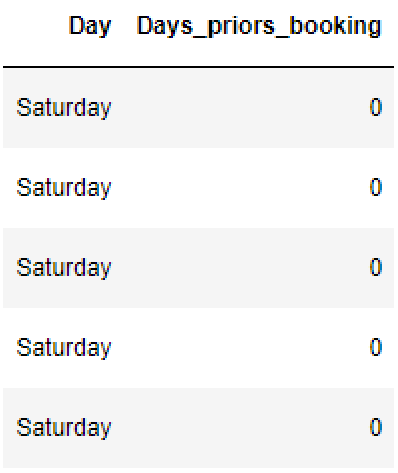
  Date Column (before):



| Date | S |
|---|---|
| 09-10-2021 | |
| 09-10-2021 | |
| 09-10-2021 | |
| 09-10-2021 | |
| 09-10-2021 | |

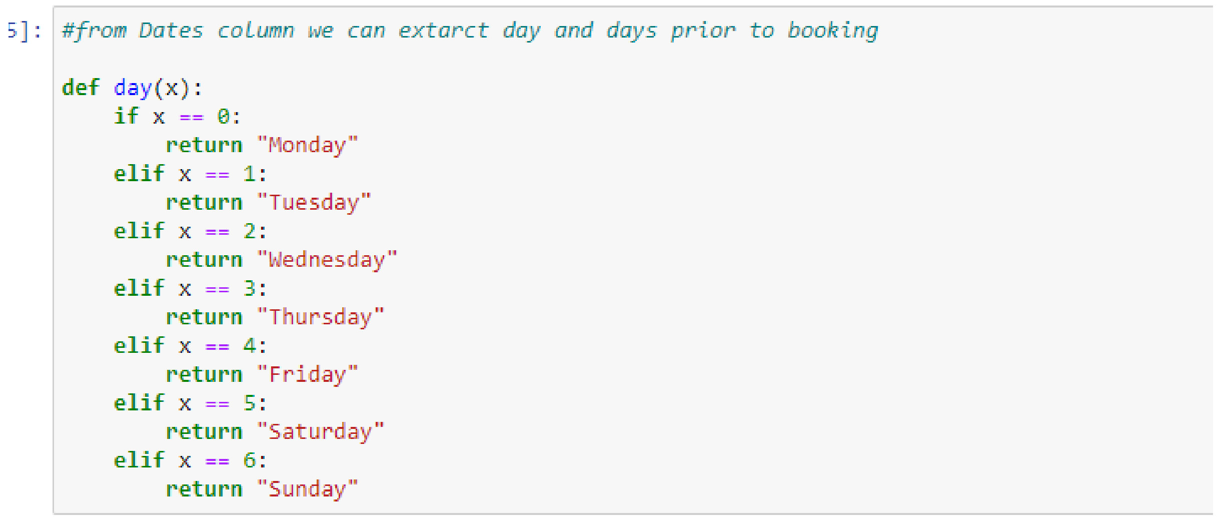
After extracting week day and days prior booking:

Note: I have scraped data of 9th October.

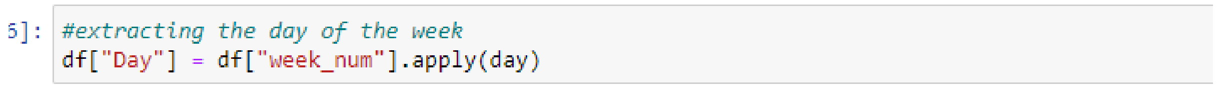


| Day | Days_priors_booking |
|---|---|
| Saturday | 0 |
| Saturday | 0 |
| Saturday | 0 |
| Saturday | 0 |
| Saturday | 0 |

## 1. Extracting Day

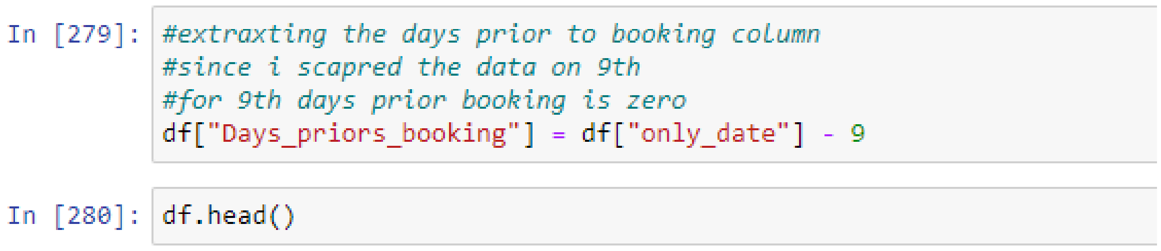


```
5]: #from Dates column we can extarct day and days prior to booking

    def day(x):
        if x == 0:
            return "Monday"
        elif x == 1:
            return "Tuesday"
        elif x == 2:
            return "Wednesday"
        elif x == 3:
            return "Thursday"
        elif x == 4:
            return "Friday"
        elif x == 5:
            return "Saturday"
        elif x == 6:
            return "Sunday"
```



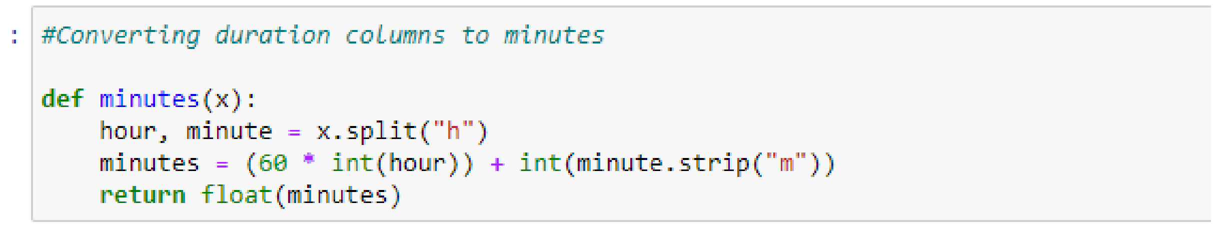
```
6]: #extracting the day of the week
    df["Day"] = df["week_num"].apply(day)
```

## 2. Extracting the days prior booking

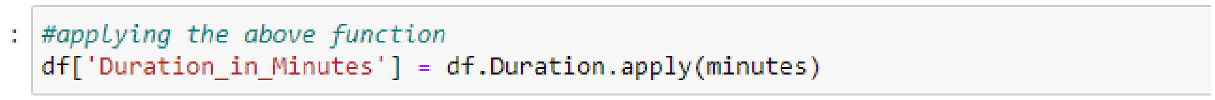


```
In [279]: #extraxting the days prior to booking column
          #since i scapred the data on 9th
          #for 9th days prior booking is zero
          df["Days_priors_booking"] = df["only_date"] - 9

In [280]: df.head()
```

3. Converting Duration columns in minutes

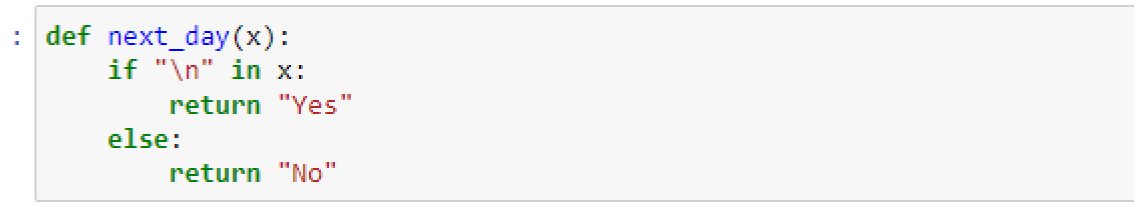


```
: #Converting duration columns to minutes

  def minutes(x):
      hour, minute = x.split("h")
      minutes = (60 * int(hour)) + int(minute.strip("m"))
      return float(minutes)
```

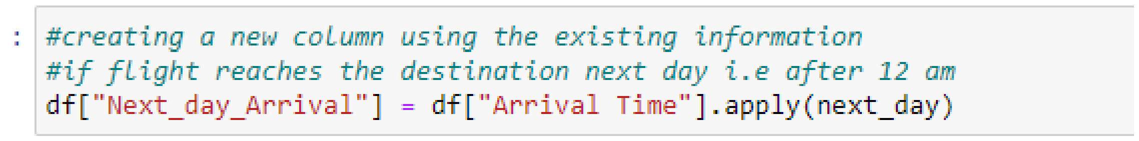


```
: #applying the above function
  df['Duration_in_Minutes'] = df.Duration.apply(minutes)
```

4. Feature extraction of next day arrival column from arrival date



```
: def next_day(x):
      if "\n" in x:
          return "Yes"
      else:
          return "No"
```



```
: #creating a new column using the existing information
  #if flight reaches the destination next day i.e after 12 am
  df["Next_day_Arrival"] = df["Arrival Time"].apply(next_day)
```

5. Extracting date from Arrival date column using Regular expressions

]:
```
import re
def Arrival(x):
    a = re.search(r"\d\d:\d\d",x)
    return a.group()
```

]:
```

#removing the "\n+1 day" from arrival date
df["Arrival_Time"] = df["Arrival Time"].apply(Arrival)
```

]:
```
#dropping the old column
df.drop("Arrival Time",axis = 1,inplace = True)
df.head()
```

6. Feature engineering of Arrival part of the day and departure part of the day

:
```
#lets extract the time of the day from this column Dep_hour and Arrival_hour column
#lets write a function for this
def part_of_the_day(x):
    if (x > 4) and (x <= 8):
        return 'Early Morning'
    elif (x > 8) and (x <= 12 ):
        return 'Morning'
    elif (x > 12) and (x <= 16):
        return'Noon'
    elif (x > 16) and (x <= 20) :
        return 'Eve'
    elif (x > 20) and (x <= 24):
        return'Night'
    elif (x <= 4):
        return'Late Night'
```

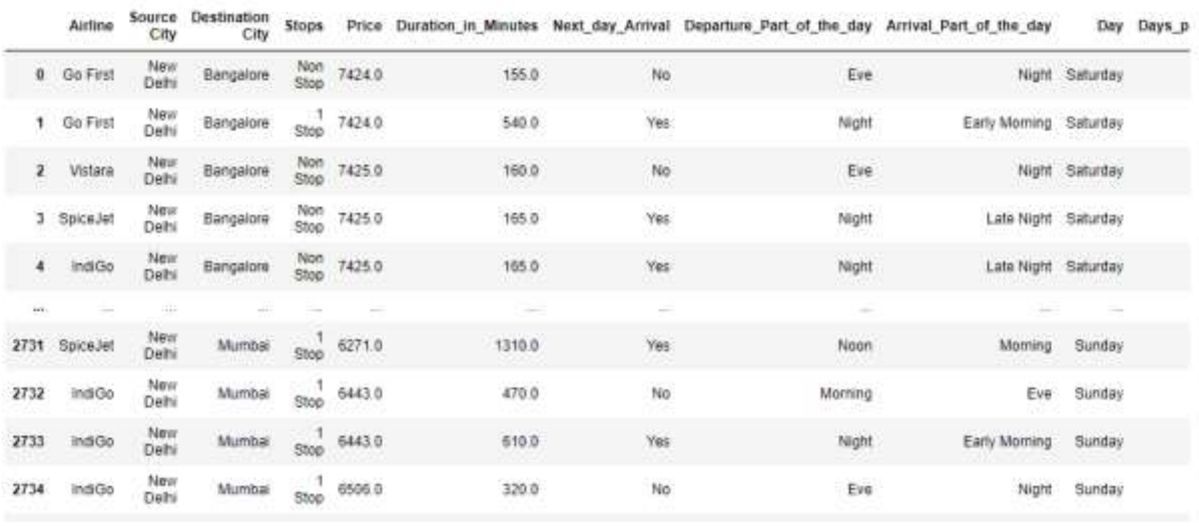
:
```
#applying the above fuction to get part of the day column
df['Departure_Part_of_the_day'] = df.Depature_hour.apply(part_of_the_day)
```

7. Converting week number to week day

```
:
#exatrcting week number
df["week_num"] = pd.DatetimeIndex(df["Dates"]).weekday
```

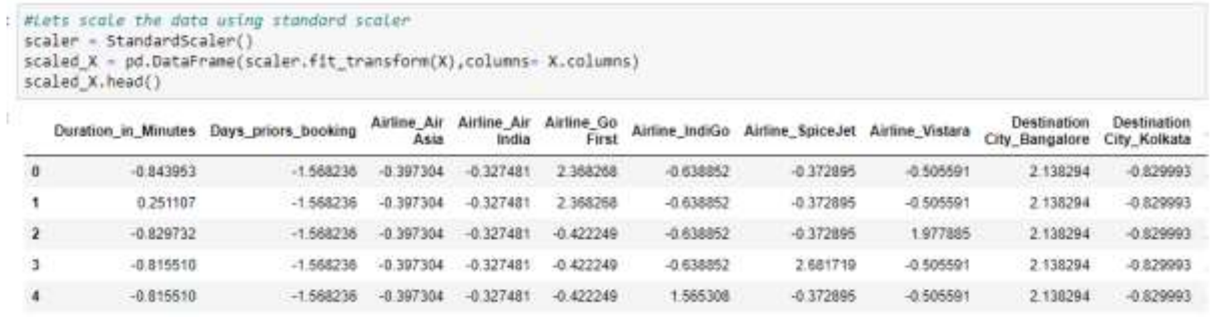
```
: #from Dates column we can extarct day and days prior to booking

def day(x):
    if x == 0:
        return "Monday"
    elif x == 1:
        return "Tuesday"
    elif x == 2:
        return "Wednesday"
    elif x == 3:
        return "Thursday"
    elif x == 4:
        return "Friday"
    elif x == 5:
        return "Saturday"
    elif x == 6:
        return "Sunday"
```

## 8. Final Data Frame after Data Pre-processing



| | Airline | Source City | Destination City | Stops | Price | Duration_in_Minutes | Next_day_Arrival | Departure_Part_of_the_day | Arrival_Part_of_the_day | Day | Days_p |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Go First | New Delhi | Bangalore | Non Stop | 7424.0 | 155.0 | No | Eve | Night | Saturday | |
| 1 | Go First | New Delhi | Bangalore | 1 Stop | 7424.0 | 540.0 | Yes | Night | Early Morning | Saturday | |
| 2 | Vistara | New Delhi | Bangalore | Non Stop | 7425.0 | 160.0 | No | Eve | Night | Saturday | |
| 3 | SpiceJet | New Delhi | Bangalore | Non Stop | 7425.0 | 165.0 | Yes | Night | Late Night | Saturday | |
| 4 | IndiGo | New Delhi | Bangalore | Non Stop | 7425.0 | 165.0 | Yes | Night | Late Night | Saturday | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 2731 | SpiceJet | New Delhi | Mumbai | 1 Stop | 6271.0 | 1310.0 | Yes | Noon | Morning | Sunday | |
| 2732 | IndiGo | New Delhi | Mumbai | 1 Stop | 6443.0 | 470.0 | No | Morning | Eve | Sunday | |
| 2733 | IndiGo | New Delhi | Mumbai | 1 Stop | 6443.0 | 610.0 | Yes | Night | Early Morning | Sunday | |
| 2734 | IndiGo | New Delhi | Mumbai | 1 Stop | 6506.0 | 320.0 | No | Eve | Night | Sunday | |

## 9. Scaling the Input data

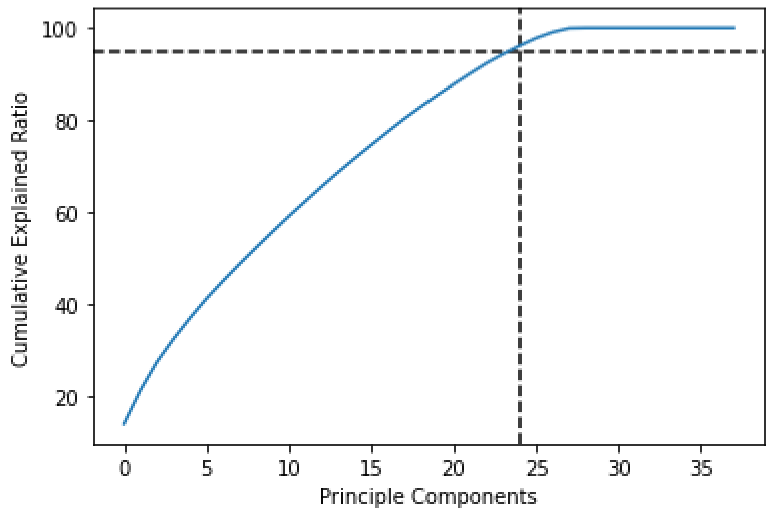


```
: #lets scale the data using standard scaler
scaler = StandardScaler()
scaled_X = pd.DataFrame(scaler.fit_transform(X),columns= X.columns)
scaled_X.head()
```

| | Duration_in_Minutes | Days_priors_booking | Airline_Air Asia | Airline_Air India | Airline_Go First | Airline_IndiGo | Airline_SpiceJet | Airline_Vistara | Destination City_Bangalore | Destination City_Kolkata |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.843953 | -1.568236 | -0.397304 | -0.327481 | 2.368268 | -0.638852 | -0.372895 | -0.505591 | 2.138294 | -0.829993 |
| 1 | 0.251107 | -1.568236 | -0.397304 | -0.327481 | 2.368268 | -0.638852 | -0.372895 | -0.505591 | 2.138294 | -0.829993 |
| 2 | -0.829732 | -1.568236 | -0.397304 | -0.327481 | -0.422249 | -0.638852 | -0.372895 | 1.977885 | 2.138294 | -0.829993 |
| 3 | -0.815510 | -1.568236 | -0.397304 | -0.327481 | -0.422249 | -0.638852 | 2.681719 | -0.505591 | 2.138294 | -0.829993 |
| 4 | -0.815510 | -1.568236 | -0.397304 | -0.327481 | -0.422249 | 1.565308 | -0.372895 | -0.505591 | 2.138294 | -0.829993 |

## 10. PCA (principle component Analysis)

```
#lets plot the graph for graphical understanding
plt.ylabel('Cumulative Explained Ratio')
plt.xlabel('Principle Components')

plt.axvline(x = s, color = 'k', linestyle = '--')
plt.axhline(y = 95, color = 'k', linestyle = '--')

plt.plot(cum_score)
plt.show()
```



- Data Inputs- Logic- Output Relationships
- We see good correlation between day's prior booking and Price as well as weekday and price.
- Airfares change very frequently
- Airfares tend to go down over time

- Hardware and Software Requirements and Tools Used

- Machine: Can use a laptop/desktop.
- Operating system: Windows 8 or 10, Mac OS X 10.9 Mavericks or Higher

- RAM & Processor: 4 GB+ RAM, i3 5th Generation 2.2 Ghz or equivalent/higher
- Tools Used: Jupyter Notebook, Microsoft Excel.
- Libraries Used : Sklearn,Seaborn,Matplotib,Pandas,Numpy ,warnings

# Model/s Development and Evaluation

- Identification of possible problem-solving approaches (methods)
- I started by extracting all the random flights data from Yatra website from five different location which include Delhi, Mumbai, Kolkata and Bangalore.
- After extracting the required information, I had column called Date which consist of three different inputs which days prior booking and day of the week.
- I used Regular Expression and other methods to extract the information from the existing column.
- I then dropped the old columns after extracting the required information.
- Converted Object data type like price to INT after replacing Symbols.

- Testing of Identified Approaches (Algorithms)

  Below are the algorithms used for the training and testing.

  Lr = LinearRegression()

  dtc = DecisionTreeRegressor()

  knn = KNeighborsRegressor(n_neighbors=5)

  rf = RandomForestRegressor()

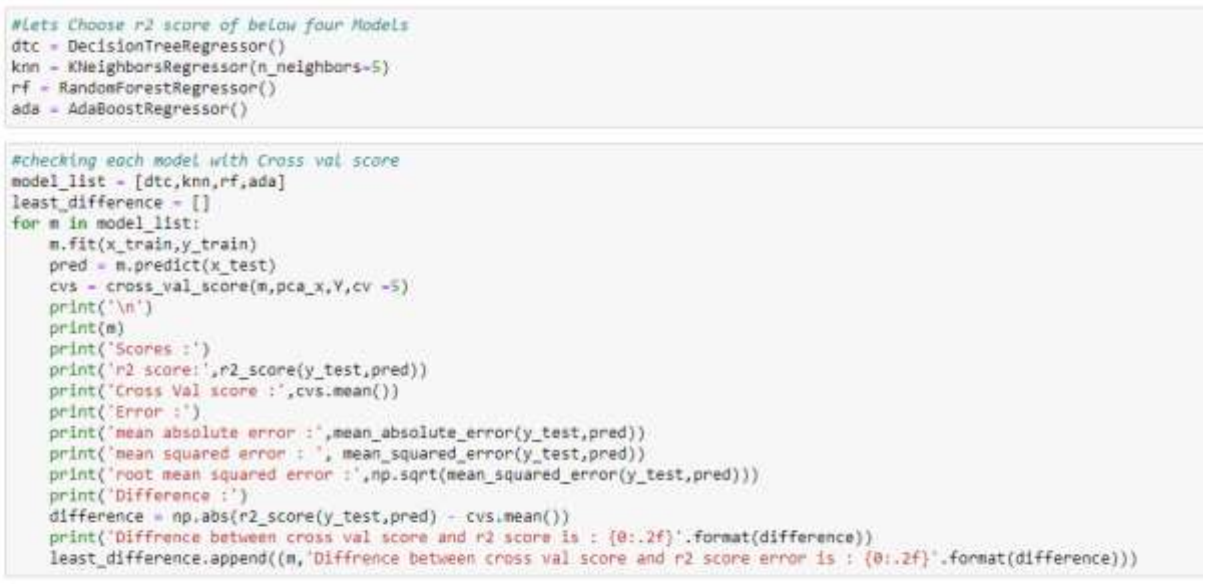
  ada = AdaBoostRegressor()

1. Linear Regression

```
lrg = LinearRegression()
lrg.fit(x_train,y_train)
pr =lrg.predict(x_test)
print("r2_score of linear refression is :", r2_score(y_test,pr))
print('Error :')
print('mean absolute error :',mean_absolute_error(y_test,pred))
print('mean squared error : ', mean_squared_error(y_test,pred))
print('root mean squared error :',np.sqrt(mean_squared_error(y_test,pred)))
```

```
r2_score of linear refression is : 0.7001818627353105
Error :
mean absolute error : 1670.636530491145
mean squared error :  4381786.980240205
root mean squared error : 2093.271836202887
```

Main Code for Running Multiple model at a time



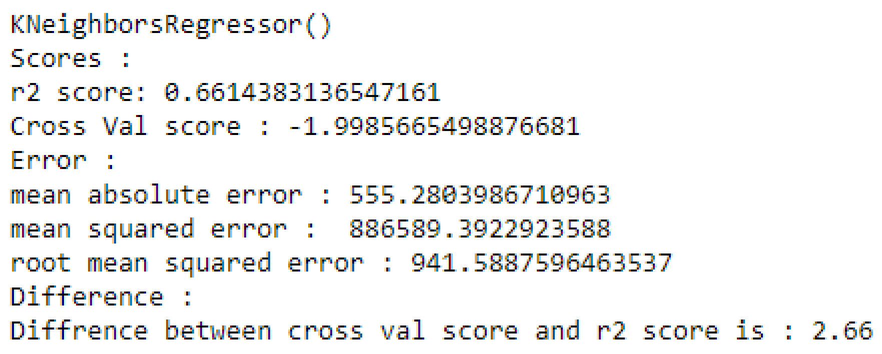
```
#Lets Choose r2 score of below four Models
dtc = DecisionTreeRegressor()
knn = KNeighborsRegressor(n_neighbors=5)
rf = RandomForestRegressor()
ada = AdaBoostRegressor()
```

```
#checking each model with Cross val score
model_list = [dtc,knn,rf,ada]
least_difference = []
for m in model_list:
    m.fit(x_train,y_train)
    pred = m.predict(x_test)
    cvs = cross_val_score(m,pca_x,Y,cv =5)
    print('\n')
    print(m)
    print('Scores :')
    print('r2 score:',r2_score(y_test,pred))
    print('Cross Val score :',cvs.mean())
    print('Error :')
    print('mean absolute error :',mean_absolute_error(y_test,pred))
    print('mean squared error : ', mean_squared_error(y_test,pred))
    print('root mean squared error :',np.sqrt(mean_squared_error(y_test,pred)))
    print('Difference :')
    difference = np.abs(r2_score(y_test,pred) - cvs.mean())
    print('Diffrence between cross val score and r2 score is : {0:.2f}'.format(difference))
    least_difference.append((m,'Diffrence between cross val score and r2 score error is : {0:.2f}'.format(difference)))
```

2. Decision Tree Regressor

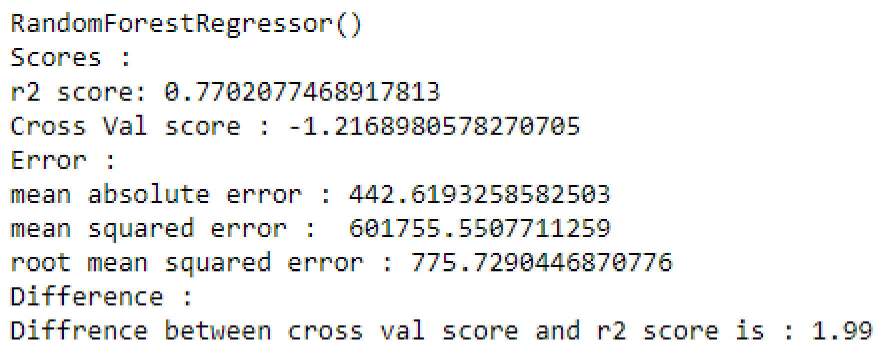
```
DecisionTreeRegressor()
Scores :
r2 score: 0.5334399555510634
Cross Val score : -2.742486354477646
Error :
mean absolute error : 485.2392026578073
mean squared error :  1221777.900332226
root mean squared error : 1105.3406263827571
Difference :
Diffrence between cross val score and r2 score is : 3.28
```

3. KNN Regressor



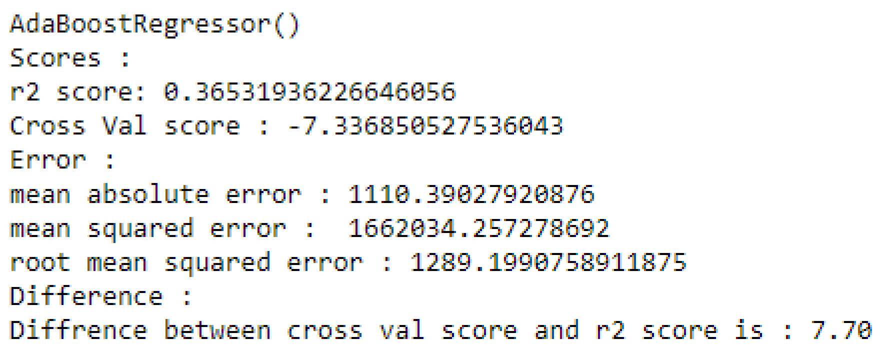
```
KNeighborsRegressor()
Scores :
r2 score: 0.6614383136547161
Cross Val score : -1.9985665498876681
Error :
mean absolute error : 555.2803986710963
mean squared error :  886589.3922923588
root mean squared error : 941.5887596463537
Difference :
Diffrence between cross val score and r2 score is : 2.66
```

4. Random Forest Regressor



```
RandomForestRegressor()
Scores :
r2 score: 0.7702077468917813
Cross Val score : -1.2168980578270705
Error :
mean absolute error : 442.6193258582503
mean squared error :  601755.5507711259
root mean squared error : 775.7290446870776
Difference :
Diffrence between cross val score and r2 score is : 1.99
```

5. AdaBoost Regressor



```
AdaBoostRegressor()
Scores :
r2 score: 0.36531936226646056
Cross Val score : -7.336850527536043
Error :
mean absolute error : 1110.39027920876
mean squared error :  1662034.257278692
root mean squared error : 1289.1990758911875
Difference :
Diffrence between cross val score and r2 score is : 7.70
```

- Run and Evaluate selected models

  These are the algorithms used along with the snapshot of their code.

- Key Metrics for success in solving problem under consideration

- **R2 Score:**

  R-squared is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R2 score of 0.0.

We obtained the r2 score of 70.0018 % which is very good.

- **Root Mean Squared Error (RMSE):**

  Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points. RMSE is a measure of how spread out these residuals are. In other words, it tells you how concentrated the data is around the line of best fit.

  The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. Lower values of RMSE indicate better fit.

  Root Mean Squared of this Random forest model is 2093.3725 which is quite high.

- **Mean Squared Error:**

  Mean squared error is the average of the squared error that is used as the loss function for least squares regression. It is the sum, over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points.

  Mean Squared of this Random forest model is 4381786.98024020 which is quite high.

- **Mean Absolute Error:**

In the context of machine learning, absolute error refers to the magnitude of difference between the prediction of an observation and the true value of that observation. MAE takes the average of absolute errors for a group of predictions and observations as a measurement of the magnitude of errors for the entire group.

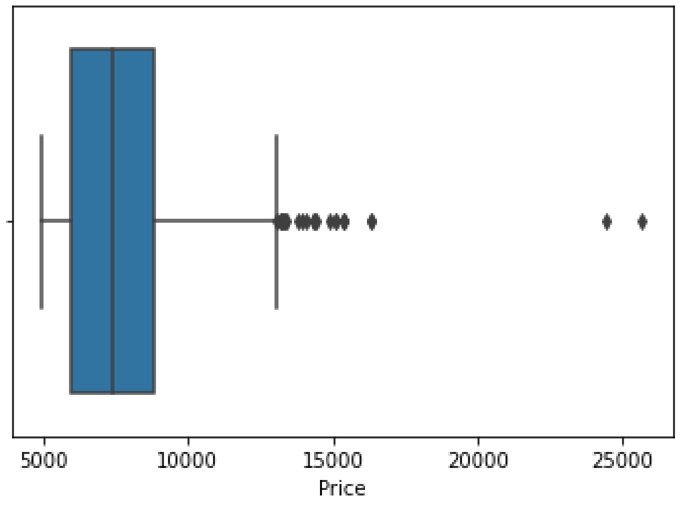
Mean Absolute error of this random forest model is 1670.636530491145.

- Visualizations

The plots made along with their pictures and the inferences and observations obtained from those.

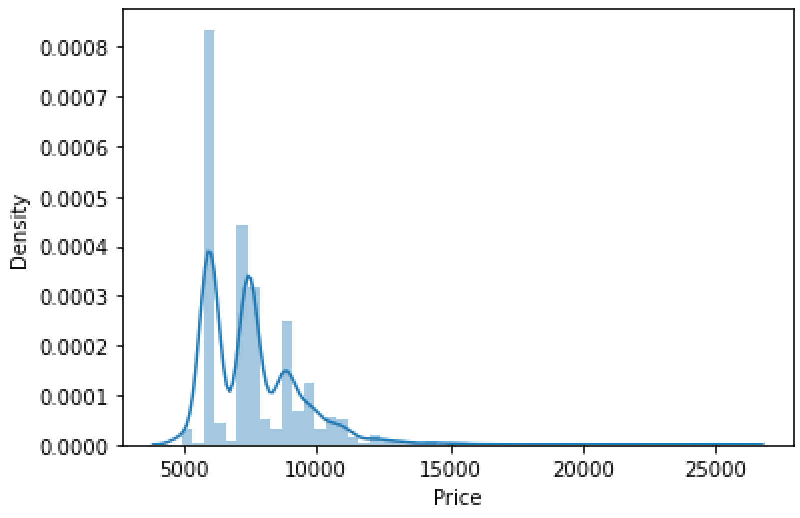
**Outliers in Target column:**

```
#lets check the box plot of our target column, to check if there are outliers
sns.boxplot(df.Price)
plt.show()
```



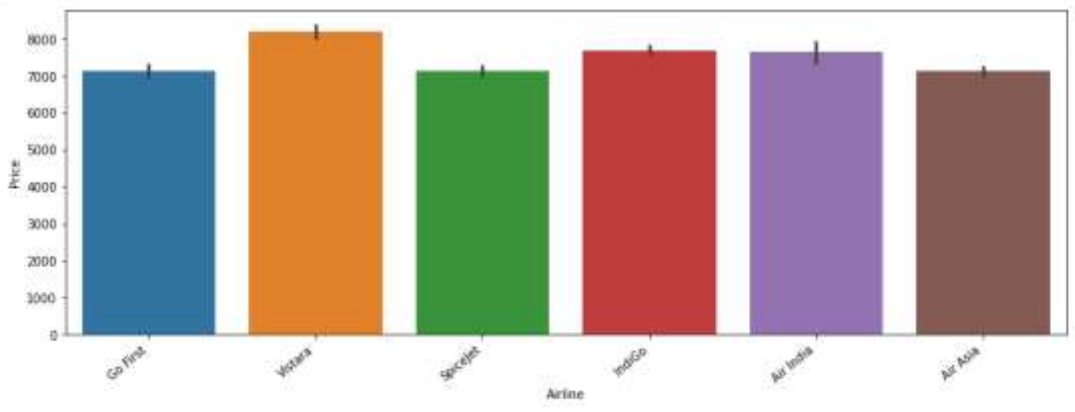
**Distribution of the target column:**

```
#lets check the distribution of the target column
sns.distplot(df.Price)
plt.show()
```



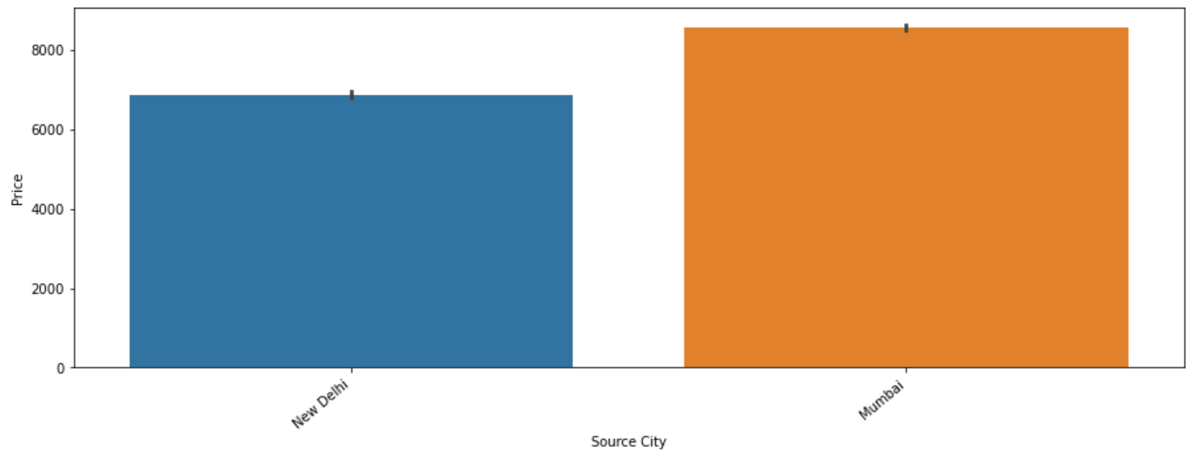
## Bivariate Analysis with Target Variable:

```
#lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Airline'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```

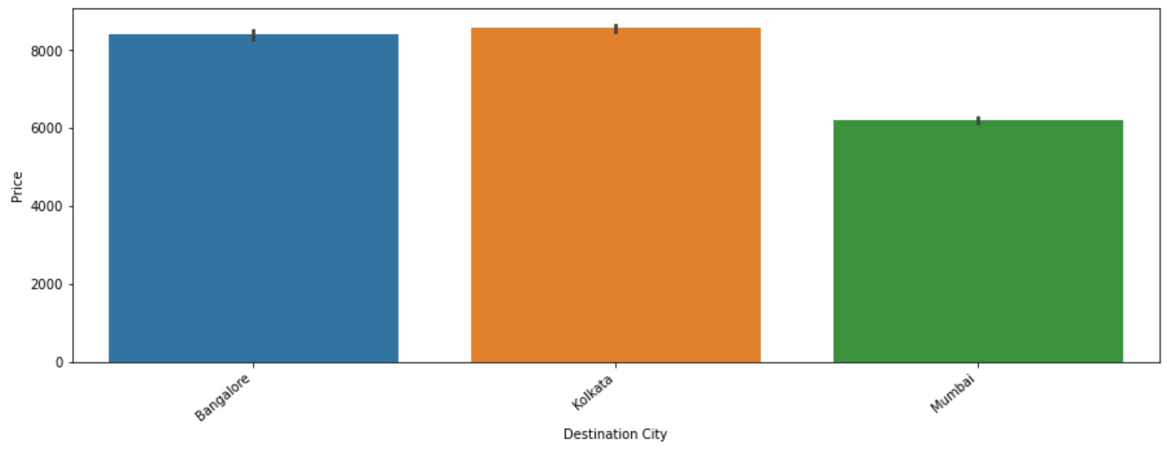


```
#from the above plot we can see that spicejet is very cheap and vistara is more expensive
```

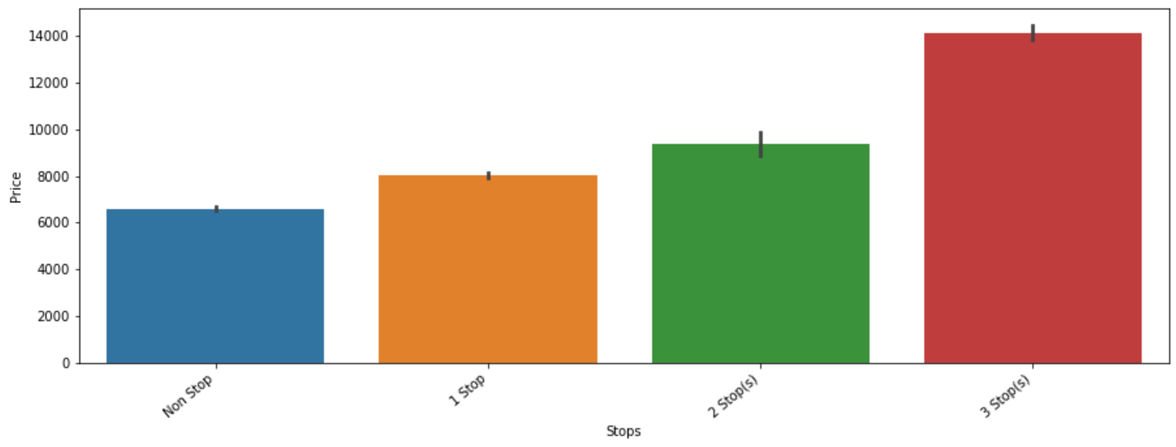
```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Source City'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Destination City'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```

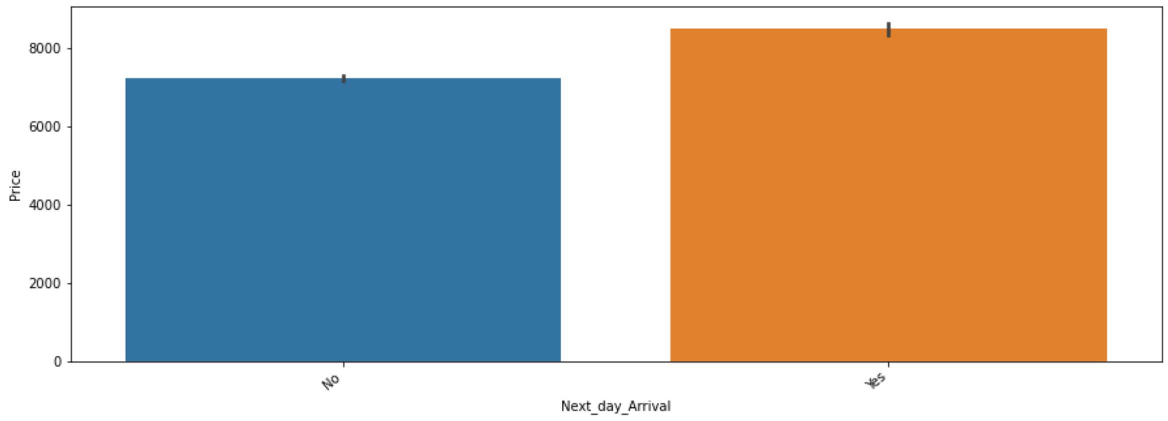


```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Stops'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



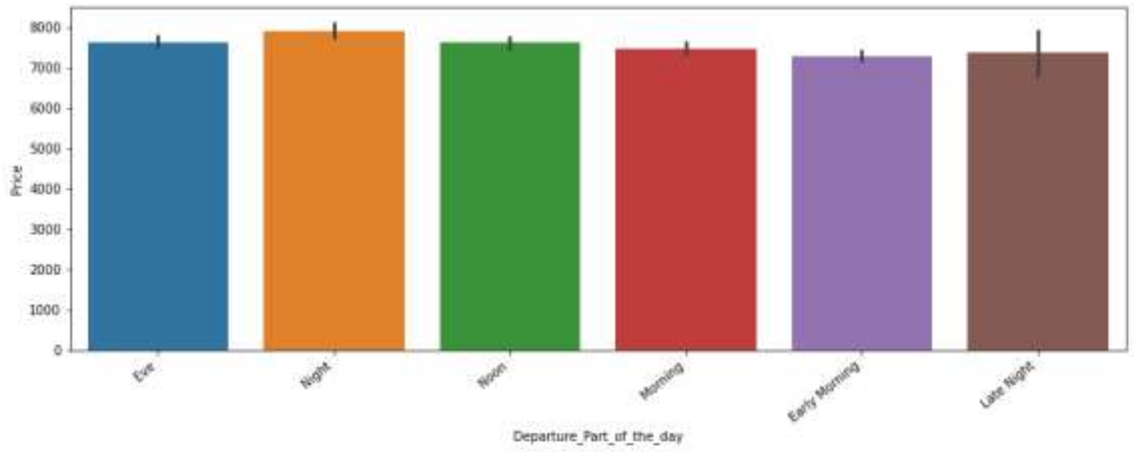
```
#from the above plot we can see that flights having more spots are more expensive
```

```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Next_day_Arrival'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



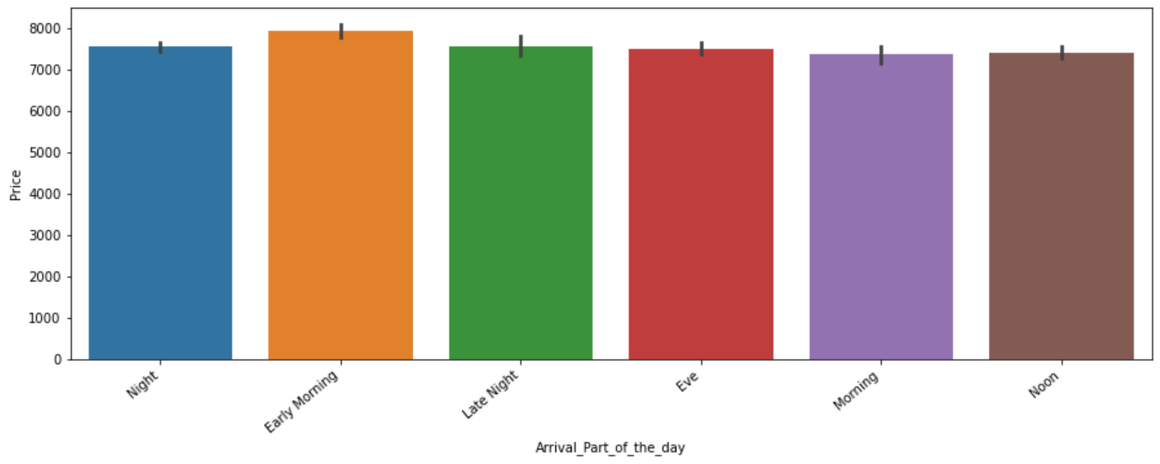
```
#from the above plot we can see that if arrival is not in same day then the flight price is expensive
```

```
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Departure_Part_of_the_day'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```

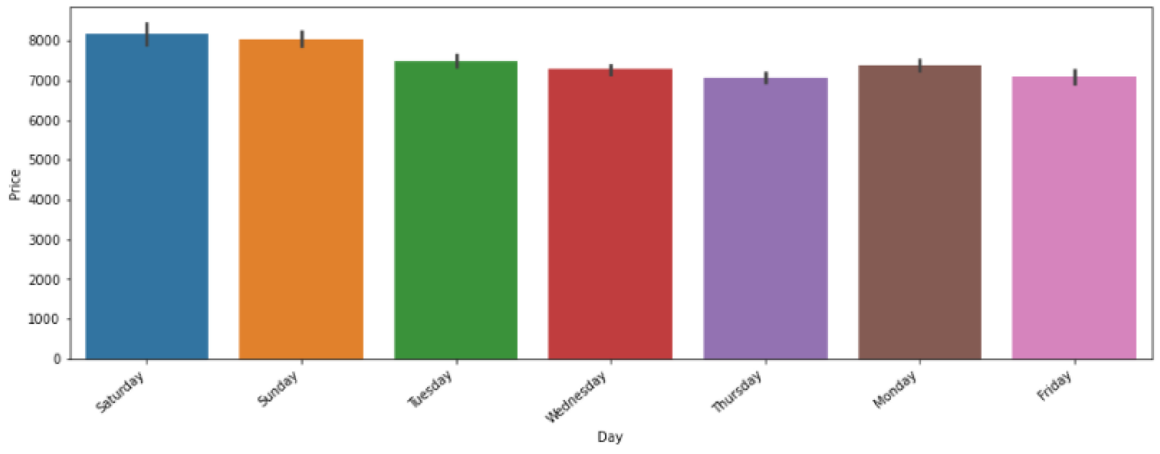


```
#from the above plot we can see that we don't see much diffrence in entries
```

```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Arrival_Part_of_the_day'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```

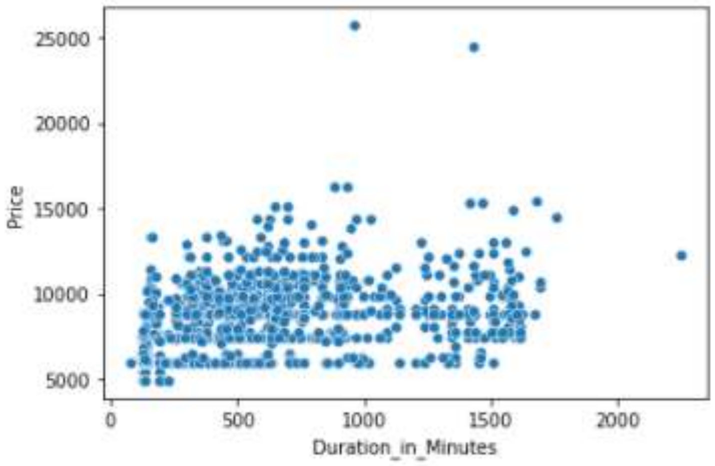


```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Day'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



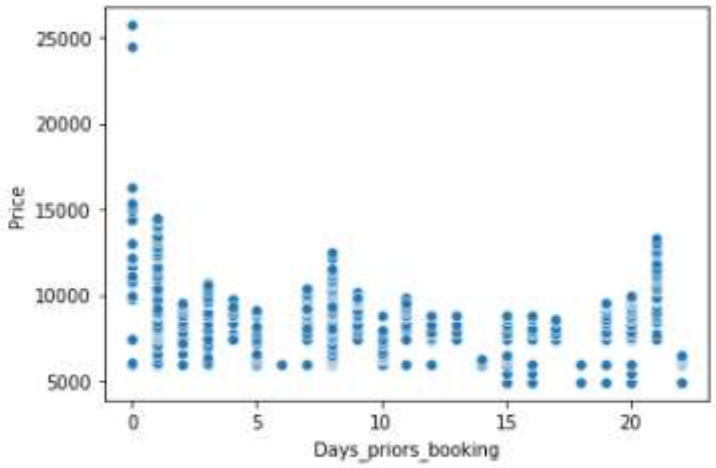
```
#from the above plot we can see that going thurday can save some money
```

```
#Lets check this column againt our target variable
sns.scatterplot(x = 'Duration_in_Minutes', y ='Price' , data  = df)
plt.show()
```



## 1. Do airfares change frequently?

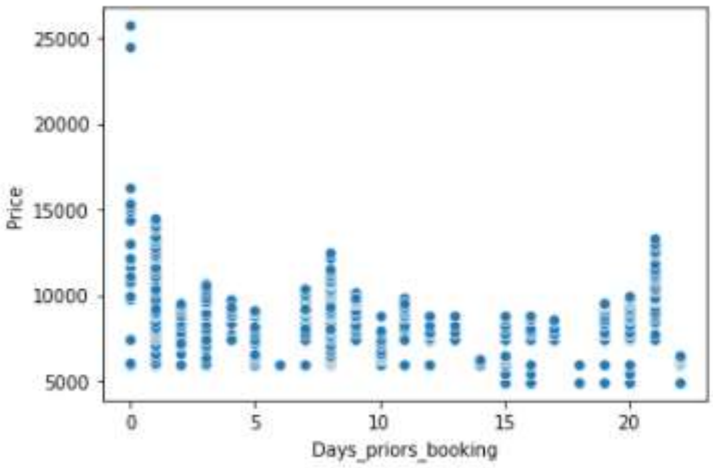
```
#Lets check this column againt our target variable
sns.scatterplot(x = 'Days_priors_booking', y ='Price' , data  = df)
plt.show()
```



From the above plot we can see that airfares change frequently

## 2. Do they move in small increments or in large jumps?

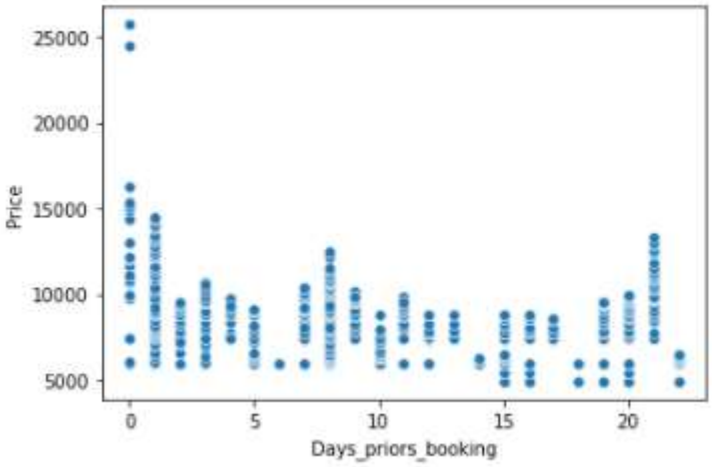
```
#Lets check this column againt our target variable
sns.scatterplot(x = 'Days_priors_booking', y ='Price' , data  = df)
plt.show()
```



Yes,they move in large jumps

## 3. Do they tend to go up or down over time?

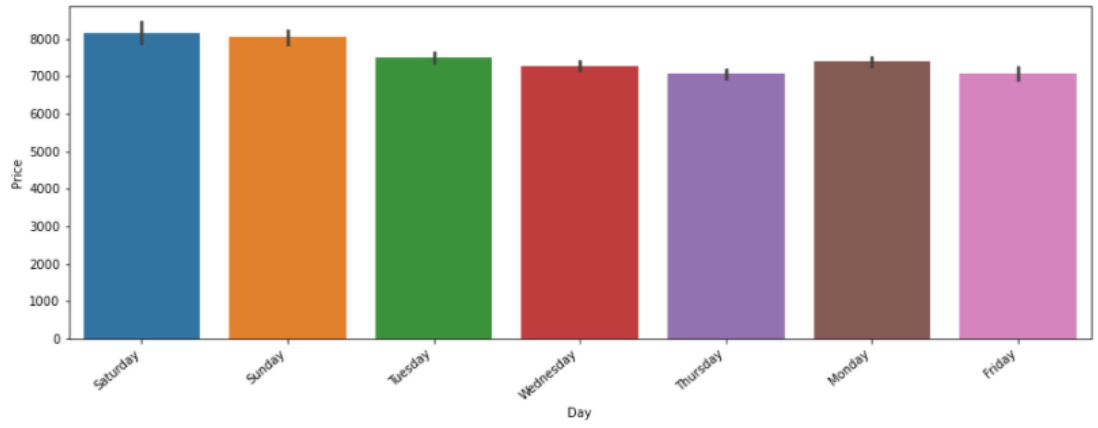
```
#Lets check this column againt our target variable
sns.scatterplot(x = 'Days_priors_booking', y ='Price' , data  = df)
plt.show()
```



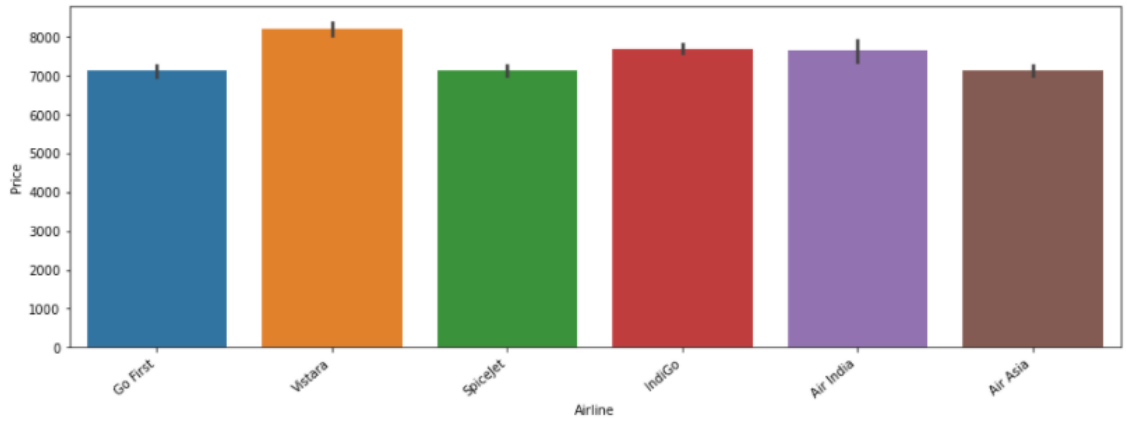
They usally tend go down over time

## 4. What is the best time to buy so that the consumer can save the most by taking the least risk? ¶

```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Day'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



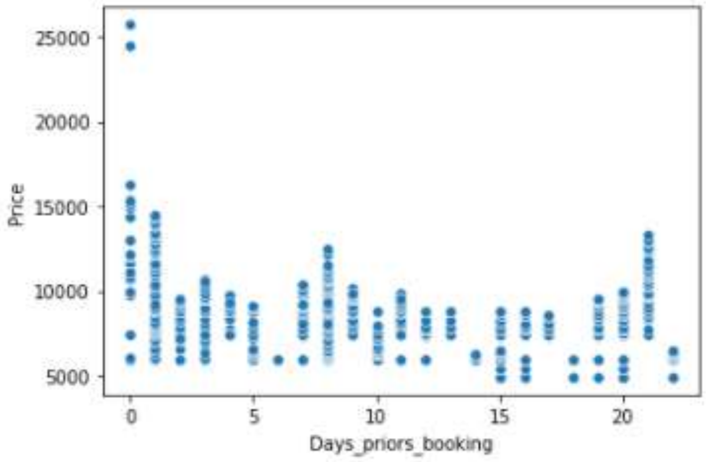
```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Airline'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



Going on Thurday and choosinf spicejet can reduce the money

## 5. Does price increase as we get near to departure date?

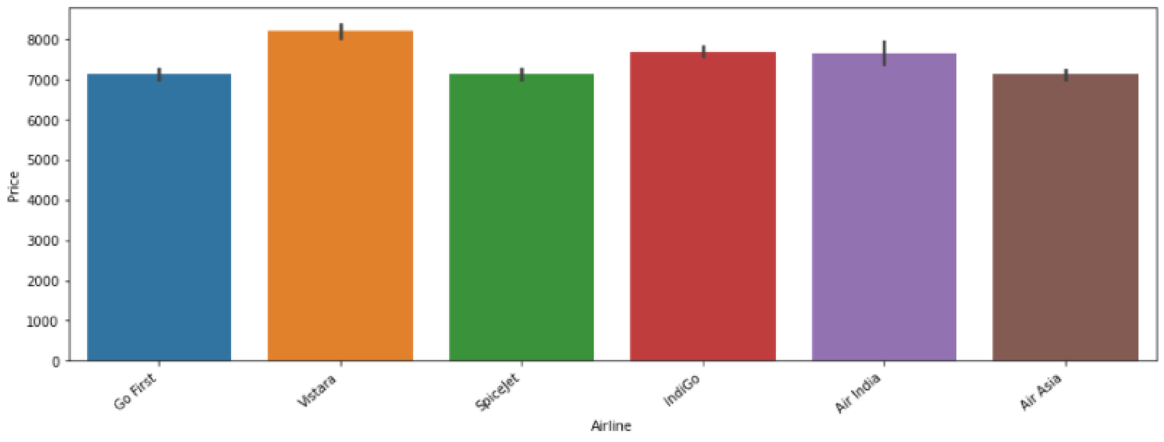
```
#Lets check this column againt our target variable
sns.scatterplot(x = 'Days_priors_booking', y ='Price' , data  = df)
plt.show()
```



Yes, booking prior can save some money

## 6. Is Indigo cheaper than Jet Airways?

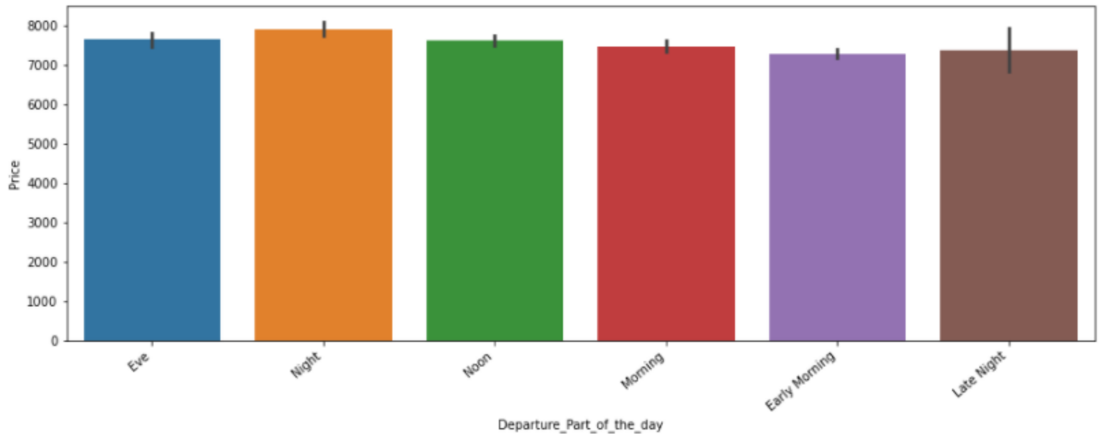
```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Airline'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



No,indifo flights are not cheaper than spicejet

**7. Are morning flights expensive?**

```
#Lets check this column againt our target variable
plt.figure(figsize= (15,5))
col1 = sns.barplot(x = df['Departure_Part_of_the_day'] , y =df['Price'] )
col1.set_xticklabels(col1.get_xticklabels(), rotation=40, ha="right")
plt.show()
```



No, Compared to morning flights night flights are expensive

- Interpretation of the Results

  I choose Linear Regression as the final model since it has the best r2 score among all the model of 70%. This model is very less difference between cross validation cross and r2 score. So, this model is not over fitted or under fitted.

**Hyper parameter tuning of Linear Regression**

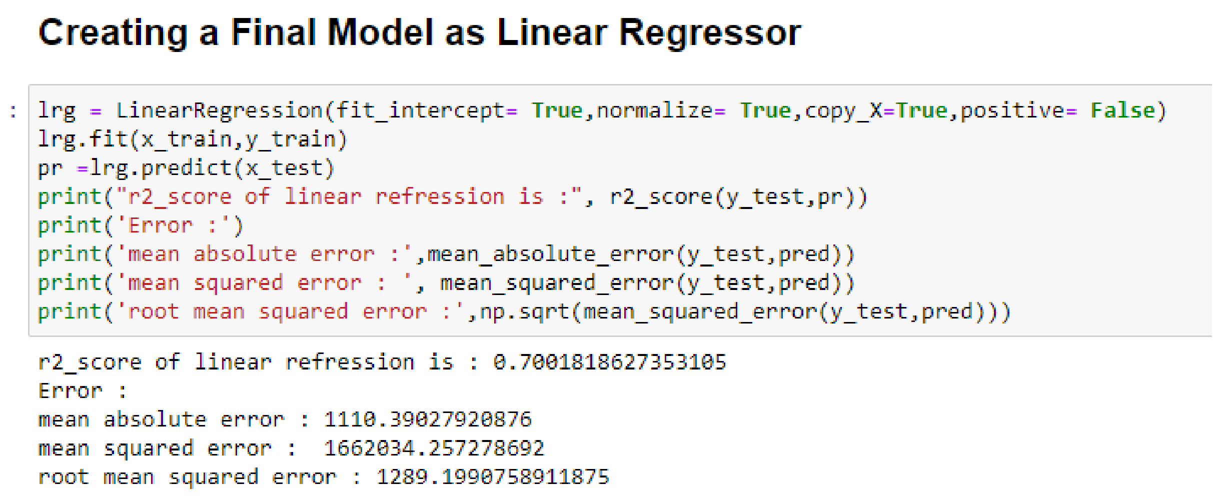
**Linear Regression giving the best Score of 70%**

**Hyperparameter Tuning:**

```
#lets use parameters of linear regression
parameters = {'fit_intercept' : [True,False],'normalize':[True,False],'copy_X':[True,False],'positive':[True,False]}

gsvrf = GridSearchCV(lrg, parameters, cv=5,scoring="r2")
gsvrf.fit(x_train,y_train)
print(gsvrf.best_score_)
print(gsvrf.best_params_)
```

```
0.5506385764768732
{'copy_X': True, 'fit_intercept': True, 'normalize': True, 'positive': False}
```

**Output of Linear Regression with best parameters obtained from hyper parameter tuning:**



### Creating a Final Model as Linear Regressor

```
lrg = LinearRegression(fit_intercept= True,normalize= True,copy_X=True,positive= False)
lrg.fit(x_train,y_train)
pr =lrg.predict(x_test)
print("r2_score of linear refression is :", r2_score(y_test,pr))
print('Error :')
print('mean absolute error :',mean_absolute_error(y_test,pred))
print('mean squared error : ', mean_squared_error(y_test,pred))
print('root mean squared error :',np.sqrt(mean_squared_error(y_test,pred)))
```

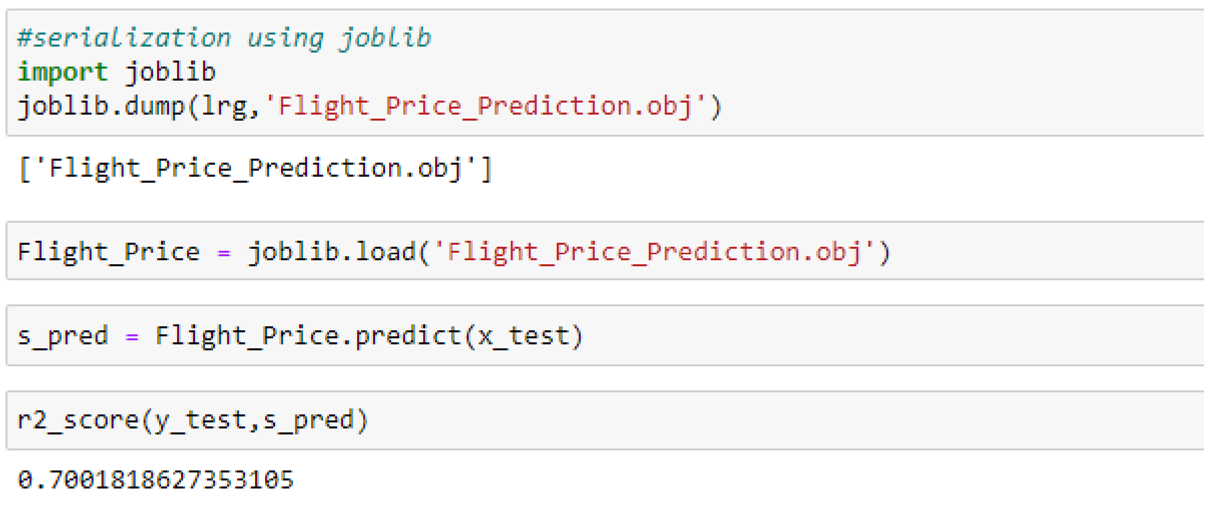
```
r2_score of linear refression is : 0.7001818627353105
Error :
mean absolute error : 1110.39027920876
mean squared error :  1662034.257278692
root mean squared error : 1289.1990758911875
```

## Saving the Final model:

Serialization using joblib.

**Pickled model as a file using joblib:** Joblib is the replacement of pickle as it is more efficient on objects that carry large numpy arrays. These functions also accept file-like object instead of filenames.

**Joblib.dump** to serialize an object hierarchy.



```
#serialization using joblib
import joblib
joblib.dump(lrg,'Flight_Price_Prediction.obj')
```

```
['Flight_Price_Prediction.obj']
```

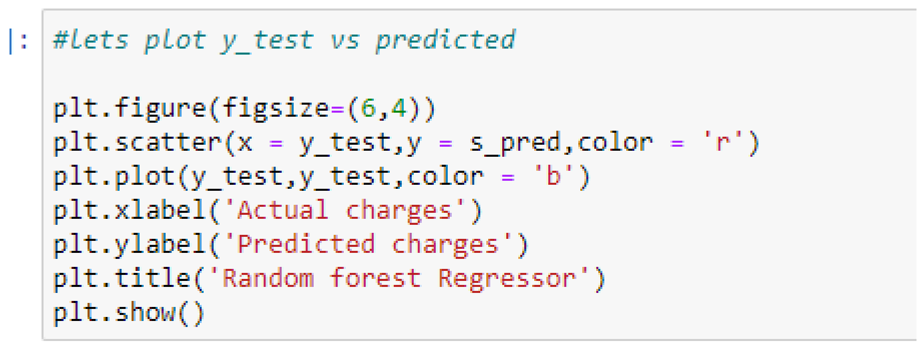
```
Flight_Price = joblib.load('Flight_Price_Prediction.obj')
```

```
s_pred = Flight_Price.predict(x_test)
```

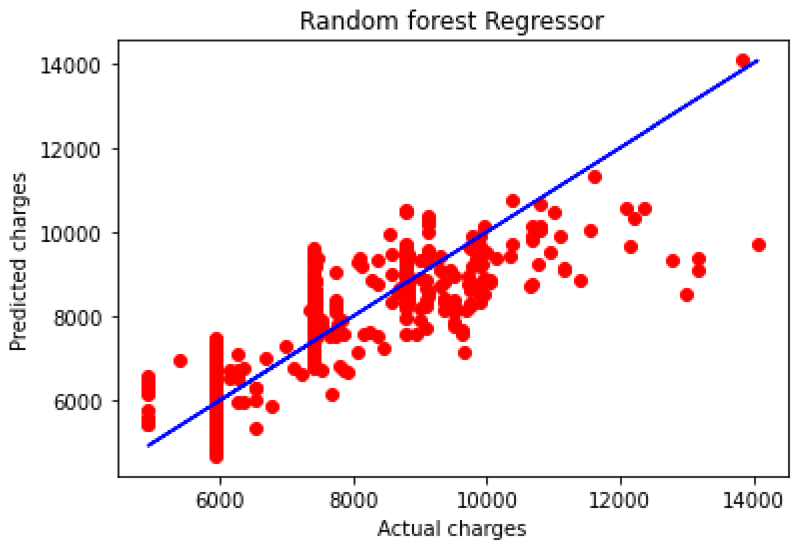
```
r2_score(y_test,s_pred)
```

```
0.7001818627353105
```

## Let’s plot y_test vs predicted:

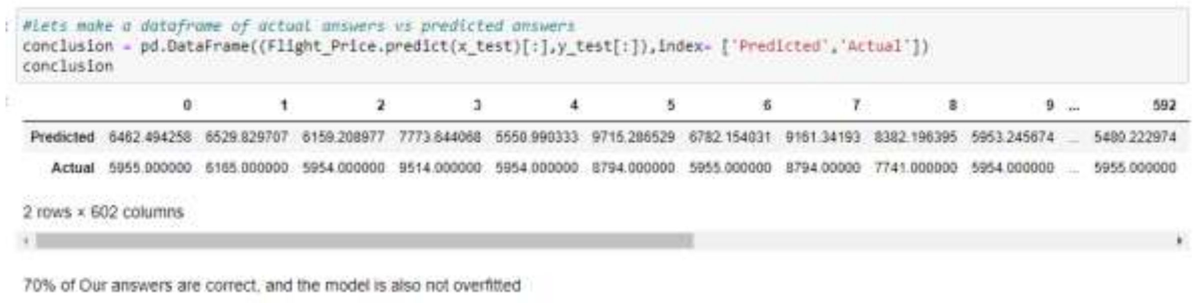


```
|: #lets plot y_test vs predicted

   plt.figure(figsize=(6,4))
   plt.scatter(x = y_test,y = s_pred,color = 'r')
   plt.plot(y_test,y_test,color = 'b')
   plt.xlabel('Actual charges')
   plt.ylabel('Predicted charges')
   plt.title('Random forest Regressor')
   plt.show()
```



we can see that values are very close to the line

## Final Predicted Car Prices:



```
#lets make a dataframe of actual answers vs predicted answers
conclusion = pd.DataFrame((Flight_Price.predict(x_test)[:],y_test[:]),index= ['Predicted','Actual'])
conclusion
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 592 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Predicted** | 6462.494258 | 6529.829707 | 6159.208977 | 7773.644068 | 5550.990333 | 9715.286529 | 6782.154031 | 9161.34193 | 8382.196395 | 5953.245674 | ... | 5480.222974 |
| **Actual** | 5955.000000 | 6165.000000 | 5954.000000 | 9514.000000 | 5954.000000 | 8794.000000 | 5955.000000 | 8794.00000 | 7741.000000 | 5954.000000 | ... | 5955.000000 |

2 rows × 602 columns

70% of Our answers are correct, and the model is also not overfitted

## CONCLUSION

- Key Findings and Conclusions of the Study

  Though this is the simplest model we've built till now, the final predictors still seem to have high correlations. One can go ahead and remove some of these features, though that will affect the adjusted-r2 score significantly (you should try doing that).

  Thus, for now, the final model consists of the 11 variables mentioned above.

- Learning Outcomes of the Study in respect of Data Science

  I choose Linear Regression algorithm as my final model since it was having least difference between its cross validation score and r2 score.

  I learned how to use regular expression to extract the required information from the existing columns

- Limitations of this work and Scope for Future Work

  Yes there is still room for improvement, like doing a more

  Web scraping from different websites, extensive feature engineering, by comparing and plotting the features against each other and identifying and removing the noisy features. The values of R-squared obtained from the algorithm give the accuracy of the model. In the future, if more data could be scraped such as the business class, so predicted results will be more accurate.