Flight Price Prediction

About the Fight Price

When we generally book a flight to go somewhere generally the price of the flight fluctuates but do we know the possible reasons behind these differences? There are many factors which affect these variations. These variations may be because of the following reasons:

- The name of the airline.
- The date of the journey
- The source from which the service begins.
- The destination where the service ends.
- The route taken by the flight to reach the destination.
- The time when the journey starts from the source.
- Time of arrival at the destination.
- Total duration of the flight.
- Total stops between the source and destination.
- Additional information about the flight
- The price of the ticket

Here by help of this project we have tried to understand these variations and factors leading these variations. Later we will predict the flight price depending upon the circumstances.

INTRODUCTION

➢ Business Problem Framing

It is a project related to airlines and its industries. We are basically predicting the price of the flights based on the conditions of the customers.

➢ Review of Literature

In this model we will study different variables and how this independent variables are related with dependent variables and how this will help us to predict whetherthe price of the flights.

Analytical Problem Framing

➢ Mathematical/ Analytical Modeling of the Problem

Let's view some basic statistics about the data like the percentile, mean, maximum, minimum etc.

```
1 df1_train.describe()
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_Hour	Dep_min	Arrival_Hour	Arrival_min	Duration_hours	Duration_mins	1
count	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	10682.000000	
mean	0.824190	9087.214567	13.509081	4.708575	12.491013	24.409287	13.349186	24.690601	10.244898	28.326624	
std	0.675229	4611.548810	8.479363	1.164408	5.748820	18.767801	6.859317	16.506808	8.494916	16.945817	
min	0.000000	1759.000000	1.000000	3.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	5277.000000	6.000000	3.000000	8.000000	5.000000	8.000000	10.000000	2.000000	15.000000	
50%	1.000000	8372.000000	12.000000	5.000000	11.000000	25.000000	14.000000	25.000000	8.000000	30.000000	
75%	1.000000	12373.000000	21.000000	6.000000	18.000000	40.000000	19.000000	35.000000	15.000000	45.000000	
max	4.000000	79512.000000	27.000000	6.000000	23.000000	55.000000	23.000000	55.000000	47.000000	55.000000	

- The average value for flight price is 9087.214.
- The average duration of the flight is 10.244 hours and 28.32 mins.
- The average number of stops is 0.82.

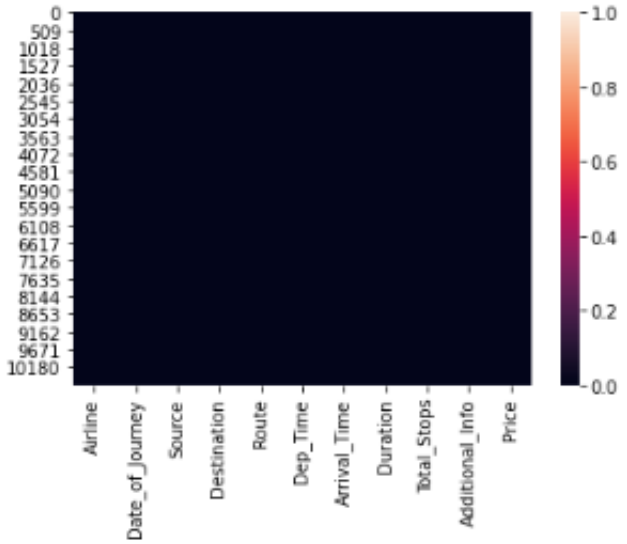
Let’s See co-relation between the Columns

➢ Data Sources and their formats

We got this data in excel format and worked on it accordingly.

➢ Data Preprocessing Done

- Let's check the shape and see count of the number of empty values in each column.

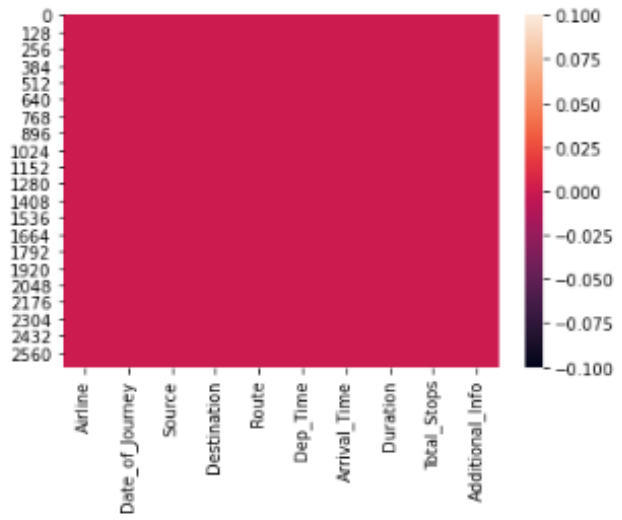



```
In [248]:  1  df_train.isnull().sum()

Out[248]: 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

In [249]:  1  df_train.dropna(inplace=True)

In [250]:  1  df_train.isnull().sum()
```

- As we can see from above train Dataset contains 10683 rows and 11 columns in which label is the dependent target column and rest are independent columns.
- And we can see dataset contains null values. So we dropped the null values.
- Our test dataset doesn't have any null value so far.

```
In [251]:
#checking null values using heat map
sns.heatmap(df_test.isnull())
#we can see that their is no null values present
```

```
Out[251]: <matplotlib.axes._subplots.AxesSubplot at 0x1fafa7820a0>
```




```
In [252]:
df_test.isnull().sum()
```

```
Out[252]: Airline           0
Date_of_Journey   0
Source            0
Destination       0
Route             0
Dep_Time          0
Arrival_Time      0
Duration          0
Total_Stops       0
Additional_Info   0
dtype: int64
```

- Data set contains journey date in format of year, month and date. We will split the date column for further analysis.
- After checking the unique values of each column we can see that year count is only one so we will drop journey date, year and unnamed as it is of no use.

In [259]:
```
#Since we have converted date of journey into integers, Now we can drop it
df_train.drop(["Date_of_Journey"],axis = 1, inplace= True)

df_test.drop(["Date_of_Journey"],axis = 1, inplace= True)
```

In [260]:
```
#Extracting Hours in train data set
df_train["Dep_Hour"] = pd.to_datetime(df_train.Dep_Time).dt.hour
#Extracting Minutes
df_train["Dep_min"] = pd.to_datetime(df_train.Dep_Time).dt.minute
```

In [261]:
```
#Extracting Hours in test  data
df_test["Dep_Hour"] = pd.to_datetime(df_test.Dep_Time).dt.hour
#Extracting Minutes
df_test["Dep_min"] = pd.to_datetime(df_test.Dep_Time).dt.minute
```

In [262]:
```
#Since we have converted Departure time into integers, Now we can drop it
df_train.drop(["Dep_Time"],axis = 1, inplace= True)

df_test.drop(["Dep_Time"],axis = 1, inplace= True)
```

In [263]:
```
#Extracting Hours in train data set
df_train["Arrival_Hour"] = pd.to_datetime(df_train.Arrival_Time).dt.hour
#Extracting Minutes
df_train["Arrival_min"] = pd.to_datetime(df_train.Arrival_Time).dt.minute
```

In [264]:
```
#Extracting Hours in test data set
df_test["Arrival_Hour"] = pd.to_datetime(df_test.Arrival_Time).dt.hour
#Extracting Minutes
df_test["Arrival_min"] = pd.to_datetime(df_test.Arrival_Time).dt.minute
```

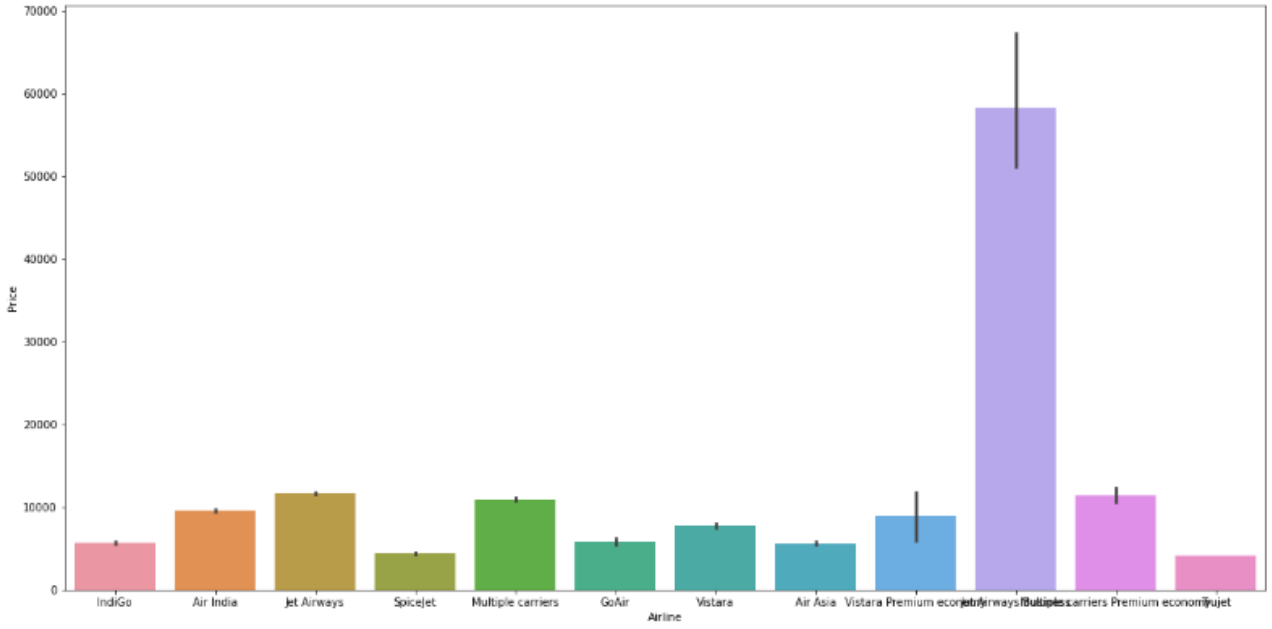
In [265]:
```
#Since we have converted Arrival time into integers, Now we can drop it
df_train.drop(["Arrival_Time"],axis = 1, inplace= True)

df_test.drop(["Arrival_Time"],axis = 1, inplace= True)
```

In [266]:
```
df_train.head()
```

Airline	Source	Destination	Route	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_Hour	Dep_min	Arrival_Hour	Arrival_min
IndiGo	Banglore	New Delhi	BLR → DEL	2h 50m	non-stop	No info	3897	24	3	22	20	1	10
Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	7h 25m	2 stops	No info	7662	1	5	5	50	13	15
Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	19h	2 stops	No info	13882	9	6	9	25	4	25
IndiGo	Kolkata	Banglore	CCU → NAG → BLR	5h 25m	1 stop	No info	6218	12	5	18	5	23	30
IndiGo	Banglore	New Delhi	BLR → NAG → DEL	4h 45m	1 stop	No info	13302	1	3	16	50	21	35

➢ Data Inputs- Logic- Output Relationships

```
plt.figure(figsize=(20,10))
sns.barplot(x="Airline",y="Price",data=df_train)
plt.show()
```



➢ Hardware and Software Requirements and Tools Used

We will use here Jupyter notebook to make your Prediction Model.

Model/s Development and Evaluation

➢ Identification of possible problem-solving approaches (methods)

```
In [301]:
1 x=df1_train.drop('Price',axis=1)
2 y=df1_train['Price']

In [302]:
1 x.head()
```

Out[302]:

	Total_Stops	Journey_day	Journey_month	Dep_Hour	Dep_min	Arrival_Hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	Airline_
0	0	24	3	22	20	1	10	2	50	0	0	
1	2	1	5	5	50	13	15	7	25	1	0	
2	2	9	6	9	25	4	25	19	0	0	0	
3	1	12	5	18	5	23	30	5	25	0	0	
4	1	1	3	16	50	21	35	4	45	0	0	

```
In [303]:
1 df1_test.head()
```

Out[303]:

	Total_Stops	Journey_day	Journey_month	Dep_Hour	Dep_min	Arrival_Hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	Airline_
0	1	6	6	17	30	4	25	10	55	0	0	
1	1	12	5	6	20	10	20	4	0	0	0	
2	1	21	5	19	15	19	0	23	45	0	0	
3	1	21	5	8	0	21	0	13	0	0	0	
4	0	24	6	23	55	2	45	2	50	0	0	

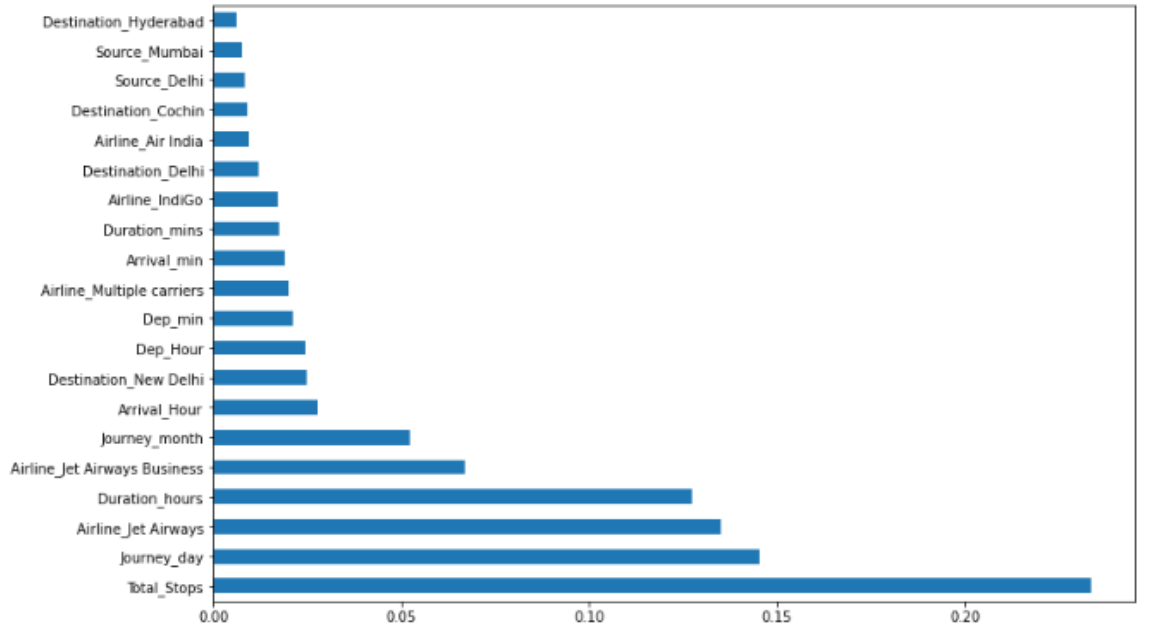
Now we will split the data set into input and output variable. As you can see above x is your input variable and y (label) is your target out variable.

.

Let’s check feature importance of the Data set.

- You can get the feature importance of each feature of your dataset by using the feature importance property of the model.
- Feature importance gives you a score for each feature of your data; the higher the score more important or relevant is the feature towards your output variable.
- Feature importance is an inbuilt class that comes with Tree Based Classifiers, we will be using Extra Tree Classifier for extracting the top 10 features for the dataset

In [310]:
```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=x.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()

```



➢ Testing of Identified Approaches (Algorithms)

Importing Necessary Libraries

In [311]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.svm import SVR
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

1. Linear Regression

- In Logistic Regression, we wish to model a dependent variable(y) in terms of one or more independent variables(x). It is a method for regression. This algorithm isused for the dependent variable that is Continuous. Y is modeled using a function that gives output in linear relationship.

In [312]:
```
def maxr2_score(regr,x,y):
    max_r_score=0
    for i in range(42,100):
        x_train,x_test,y_train,y_test=train_test_split(x,y,random_state= i ,test_size=0.20)
        regr.fit(x_train,y_train)
        y_pred=regr.predict(x_test)
        r2_scr=r2_score(y_test,y_pred)

        if r2_scr>max_r_score:
            max_r_score=r2_scr
            final_i=i
    print("max r2 score corresponding to",final_i,"is",max_r_score)
    return final_i
```

In [215]:
```
lreg=LinearRegression()
i=maxr2_score(lreg,x,y)
```

```
max r2 score corresponding to 57 is 0.679792644911887
```

In [216]:
```
print("Mean r2 score for Linear Regression:",cross_val_score(lreg,x,y,cv=5,scoring="r2").mean())
print("standard deviation in r2 score for Linear Regression",cross_val_score(lreg,x,y,cv=5,scoring="r2").std())

```

```
Mean r2 score for Linear Regression: 0.6199027318916419
standard deviation in r2 score for Linear Regression 0.019257859247068503
```

Further we will use grid search cv to find the best parameter linear regression.

2. Decision Tree Classification

The idea of a decision tree is to divide the data set into smaller data sets based onthe descriptive features until you reach a small enough set that contains data points that fall under one label.
Decision trees are easy to interpret. To build a decision tree requires little datapreparation from the user- there is no need to normalize data.

In [217]:

```
knr=KNeighborsRegressor()
parameters={'n_neighbors':range(1,10)}
gd=GridSearchCV(knr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

lsreg=Lasso()
parameters={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(lsreg,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

dtr=DecisionTreeRegressor()
parameters={'criterion':('mse', 'friedman_mse', 'mae')}
gd=GridSearchCV(dtr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

enr=ElasticNet()
parametors={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(enr,parametors)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

rfr=RandomForestRegressor()
parameters={"n_estimators":[10,50,100,120,150],"max_features": ["auto", "sqrt", "log2"]}
gd=GridSearchCV(rfr,parameters,cv=5)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

svr=SVR()
parameters={"kernel" : ( "Linear" ,"rbf" ) ,"C": [1,10]}
gd=GridSearchCV(svr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")
```

Further we will use grid search cv to find the best parameter for Decision Tree Classifier.

3. Random Forest Classification

Random Forest is a supervised learning algorithm; it creates a forest and makes it somehow random. The "forest" it builds, is an ensemble of Decision Trees. Step-1Pick at random K data points from the training set.
Step-2 Build the Decision tree associated to these K data points
Step-3Choose the Number of trees (n) you want to build and repeat Step1 and Step2
Step-4For a new data points make each one of your 'n' trees predict the category to which the data point belongs and assign the new data point to the category that wins the majority vote.

In [217]:
```
knr=KNeighborsRegressor()
parameters={'n_neighbors':range(1,10)}
gd=GridSearchCV(knr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

lsreg=Lasso()
parameters={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(lsreg,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

dtr=DecisionTreeRegressor()
parameters={'criterion':('mse', 'friedman_mse', 'mae')}
gd=GridSearchCV(dtr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

enr=ElasticNet()
parametors={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(enr,parametors)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

rfr=RandomForestRegressor()
parameters={"n_estimators":[10,50,100,120,150],"max_features": ["auto", "sqrt", "log2"]}
gd=GridSearchCV(rfr,parameters,cv=5)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

svr=SVR()
parameters={"kernel" : ( "Linear" ,"rbf" ) ,"C": [1,10]}
gd=GridSearchCV(svr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")
```

Further we will use grid search cv to find the best parameter for Random Forest.

4. Gradient Boosting-

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

In [217]:

```
knr=KNeighborsRegressor()
parameters={'n_neighbors':range(1,10)}
gd=GridSearchCV(knr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

lsreg=Lasso()
parameters={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(lsreg,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

dtr=DecisionTreeRegressor()
parameters={'criterion':('mse', 'friedman_mse', 'mae')}
gd=GridSearchCV(dtr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

enr=ElasticNet()
parametors={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(enr,parametors)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

rfr=RandomForestRegressor()
parameters={"n_estimators":[10,50,100,120,150],"max_features": ["auto", "sqrt", "log2"]}
gd=GridSearchCV(rfr,parameters,cv=5)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

svr=SVR()
parameters={"kernel" : ( "Linear" ,"rbf" ) ,"C": [1,10]}
gd=GridSearchCV(svr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")
```

Further we will use grid search cv to find the best parameter for Gradient Boosting Classifier.

5. K Neighbors Classifier-

This is a supervised, non-parametric learning algorithm which classifies a given point based on its neighbors. The choice of the ‘k’ becomes very crucial since the data point is assigned to the class of the nearest ‘k’ neighbors. Once we get to know such ‘k’ nearest data points, the test data is assigned a label by taking the majority vote from the class labels of the ‘k’ nearest data points.

In [217]:

```
knr=KNeighborsRegressor()
parameters={'n_neighbors':range(1,10)}
gd=GridSearchCV(knr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

lsreg=Lasso()
parameters={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(lsreg,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

dtr=DecisionTreeRegressor()
parameters={'criterion':('mse', 'friedman_mse', 'mae')}
gd=GridSearchCV(dtr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

enr=ElasticNet()
parametors={"alpha":[0.001,0.01,0.1,1]}
gd=GridSearchCV(enr,parametors)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

rfr=RandomForestRegressor()
parameters={"n_estimators":[10,50,100,120,150],"max_features": ["auto", "sqrt", "log2"]}
gd=GridSearchCV(rfr,parameters,cv=5)
gd.fit(x,y)
print(gd.best_params_)
print("\n")

svr=SVR()
parameters={"kernel" : ( "Linear" ,"rbf" ) ,"C": [1,10]}
gd=GridSearchCV(svr,parameters)
gd.fit(x,y)
print(gd.best_params_)
print("\n")
```

Further we will use grid search cv to find the best parameter for K Neighbors Classifier.

➢ Key Metrics for success in solving problem under consideration

Accuracy Score is the number of correct predictions made as a ratio of all predictions made. It is the most common evaluation metric for classification problems.

Cross-validation is to call the cross_val_score helper function on the estimator and the dataset.

To estimate the accuracy of a linear kernel support vector machine on the dataset by splitting the data, fitting a model and computing the score (n=5 or any number provided by you) consecutive times (with different splits each time):

The **Area under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve. The higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

Receiver Operating Characteristic (ROC) summarizes the model's performance by evaluating the tradeoffs between true positive rate (sensitivity) and false positive rate(1- specificity). For plotting ROC, it is advisable to assume $p > 0.5$ since we are more concerned about success rate.

ROC summarizes the predictive power for all possible values of $p > 0.5$. The area under curve (AUC), referred to as index of accuracy (A) or concordance index, is a perfect performance metric for ROC curve. Higher the area under curve, better the prediction power of the model.

➢ Results for all algorithms

```
Mean r2 score for KNeighbor  Regression: 0.5849170021639158
standard deviation in r2 score for KNeighbor   Regression 0.02281422591052052

_____________________________________________________________________________________________________

max r2 score corresponding to 57 is 0.67959979048739


Mean r2 score for Lasso Regression: 0.6199085514739766
standard deviation in r2 score for Lasso Regression 0.019217149796267416

_____________________________________________________________________________________________________

max r2 score corresponding to 94 is 0.7828300695811019


Mean r2 score for DecisionTreeRegressor : 0.712037227559069
standard deviation in r2 score for DecisionTreeRegressor 0.03670450998469038

_____________________________________________________________________________________________________

max r2 score corresponding to 71 is 0.6373726020449726


Mean r2 score for Elastic net Regression: 0.6073371812718781
standard deviation in r2 score for Elastic net Regression 0.010740226571354997

_____________________________________________________________________________________________________

max r2 score corresponding to 94 is 0.840664100732369


Mean r2 score for RandomForestRegressor : 0.8129005564536532
standard deviation in r2 score for RandomForestRegressor  0.026742578010271356

_____________________________________________________________________________________________________

max r2 score corresponding to 58 is 0.8541004215559405


Mean r2 score for gradient boosting Regression: 0.7973702429421017
standard deviation in r2 score for gradient boosting Regression 0.015489929457405212
```

➢ Interpretation of the Results

We are getting best result from the gradient boost regressor with 85.4033% accuracy.

THANK YOU