Project Name : Optimizing Flight Booking Decisions through

Machine Learning Price Predictions

Team ID :

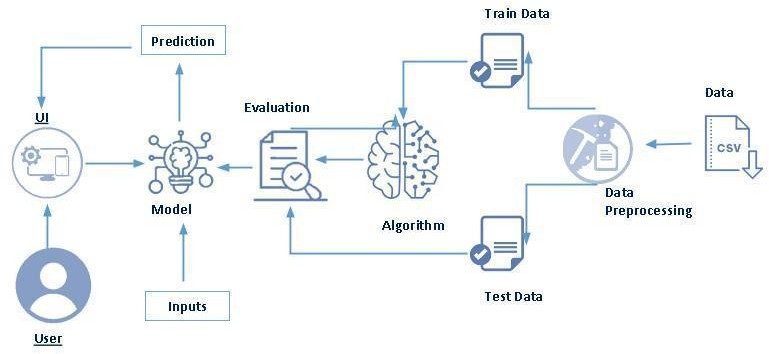
Team Leader :

Team Members :

Optimizing Flight Booking Decisions through Machine Learning Price Predictions

People who work frequently travels through flight will have better knowledge on best discount and right time to buy the ticket. For the business purpose many airline companies change prices according to the seasons or time duration. They will increase the price when people travel more. Estimating the highest prices of the airlines data for the route is collected with features such as Duration, Source, Destination, Arrival and Departure. Features are taken from chosen dataset and in the price wherein the airline price ticket costs vary overtime. we have implemented flight price prediction for users by using KNN, decision tree and random forest algorithms. Random Forest shows the best accuracy of 80% for predicting the flight price. also, we have done correlation tests and metrics for the statistical analysis.

# Technical Architecture:



**Project Flow:**

* User interacts with the UI to enter the input. Entered input is analyzed by the model which is integrated.
* Once model analyses the input the prediction is showcased on the UI To accomplish this, we have to complete all the activities listed below,
* Define Problem / Problem Understanding
* Specify the business problem
* Business requirements
* Literature Survey
* Social or Business Impact.
* Data Collection & Preparation
* Collect the dataset
* Data Preparation
* Exploratory Data Analysis
* Descriptive statistical
* Visual Analysis
* Model Building
* Training the model in multiple algorithms
* Testing the model
* Performance Testing & Hyperparameter Tuning
* Testing model with multiple evaluation metrics
* Comparing model accuracy before & after applying hyperparameter tuning
* Model Deployment
* Save the best model
* Integrate with Web Framework
* Project Demonstration & Documentation
* Record explanation Video for project end to end solution
* Project Documentation-Step by step project development procedure

**2. Problem Definition & Design Thinking**

**2.1 Empathy Map**

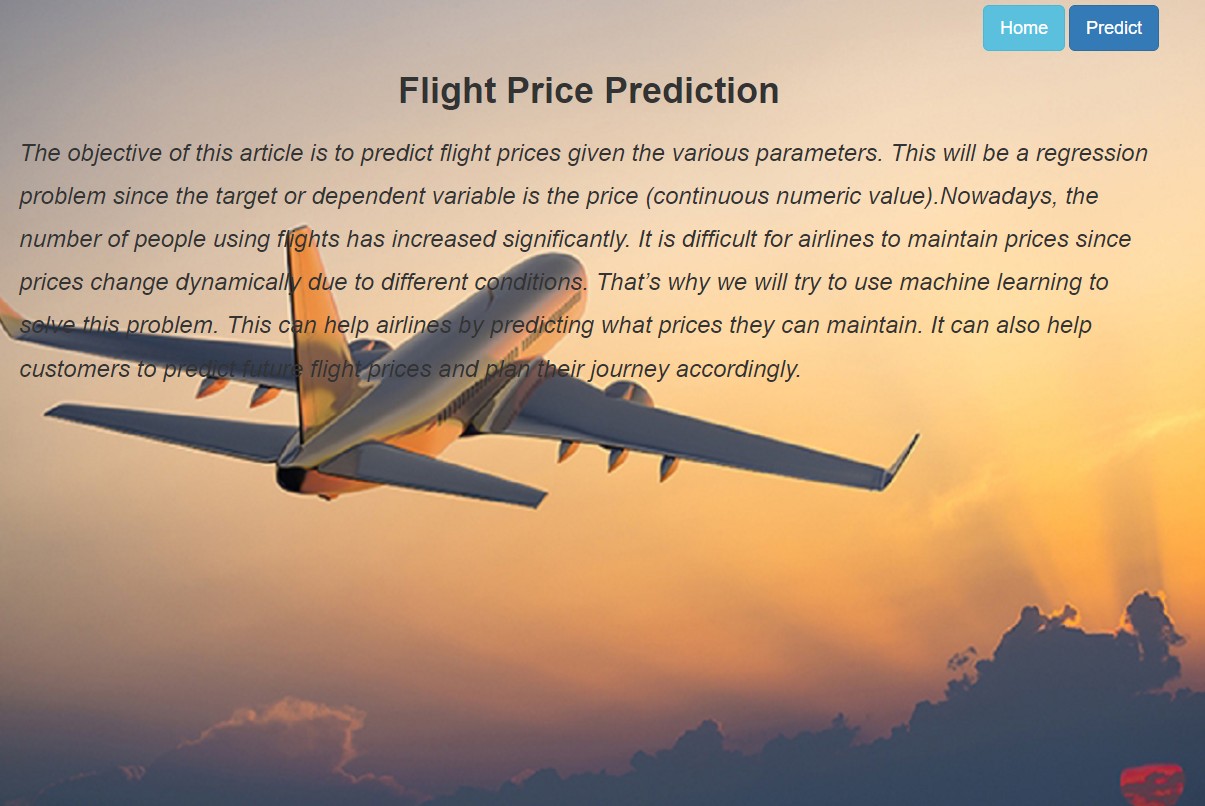
# 

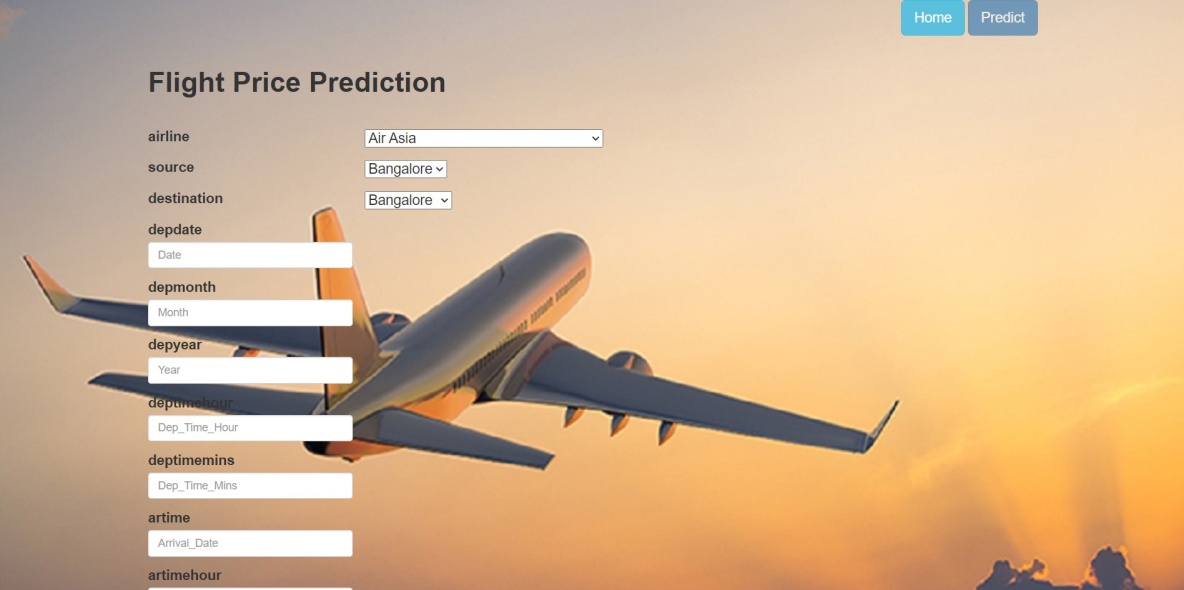
**2.2 Ideation & Brainstorming Map**

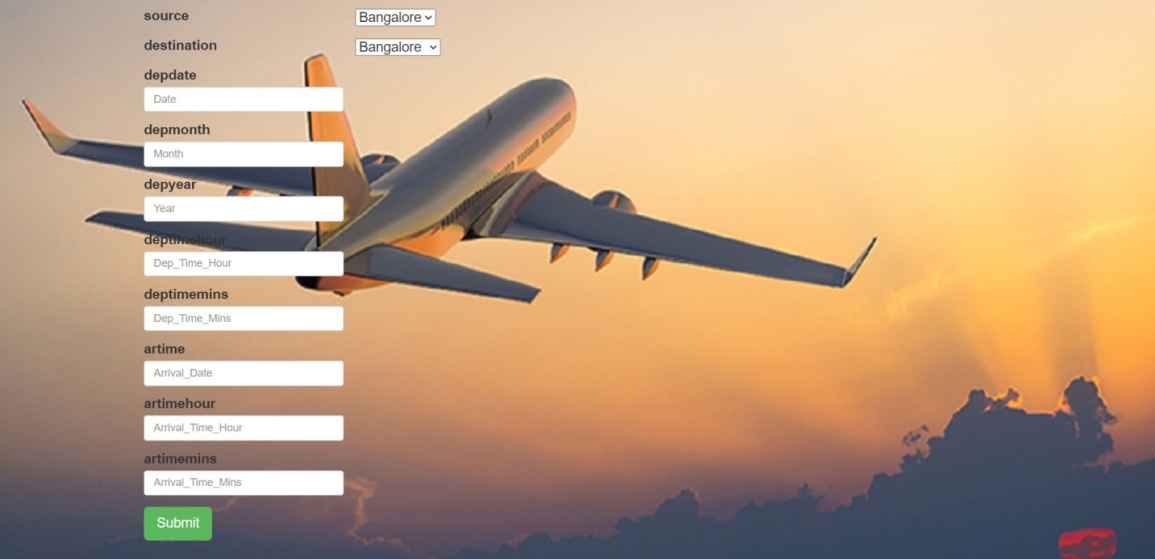
****

**3. RESULT:**

**Home page:**

****

****

****

**4. ADVANTAGES & DISADVANTAGES:**

**ADVANTAGES:**

* Keeps your business running 24/7. ...
* Dramatically reduce your admin workload. ...
* Reduce human error. ...
* Maximize reservations and reduce no-shows. ...
* Up sell easily. ...
* Increase payment speed and security. ...
* Grow your marketing and online presence. ...
* Enjoy insights from data analysis.

**DISADVANTAGES:**

* You need internet access. Reliable internet access is required to check reservations and add bookings that are made over the phone. ...
* You need to be ready for an influx of new customers. ...
* Not all online booking systems are created equal.

**5. APPLICATIONS**

Flight booking applications **help the airline industry automate the booking process**. Users worldwide can book flights on the go using the simple apps, which include features such as quick flight search, download tickets, check and modify booking details, one-tap check-in, and many more.

**QUICK LINKS**

* Flights Discount Coupons,
* Domestic Airlines,
* Indigo Airlines,
* Air Asia,
* SpiceJet,
* GoAir,
* Air India,
* Air India Express,

**IMPORTANT LINKS**

* Cheap Flights,
* Flight Status,

**CORPORATE TRAVEL**

* Corporate Travel,
* Corporate Hotel Booking,
* Corporate Flight Booking,
* Business Travel for SME,
* GST Invoice for International flights,
* Business Travel Solutions,
* GST Invoice for Bus,
* Corporate Bus booking,
* myBiz - Best Business Travel Platform,
* GST Invoice for Flights,
* GST Invoice for Corporate Travel,
* GST Invoice for Hotels,
* myBiz for Small Business,
* Free cancellation on International Flights

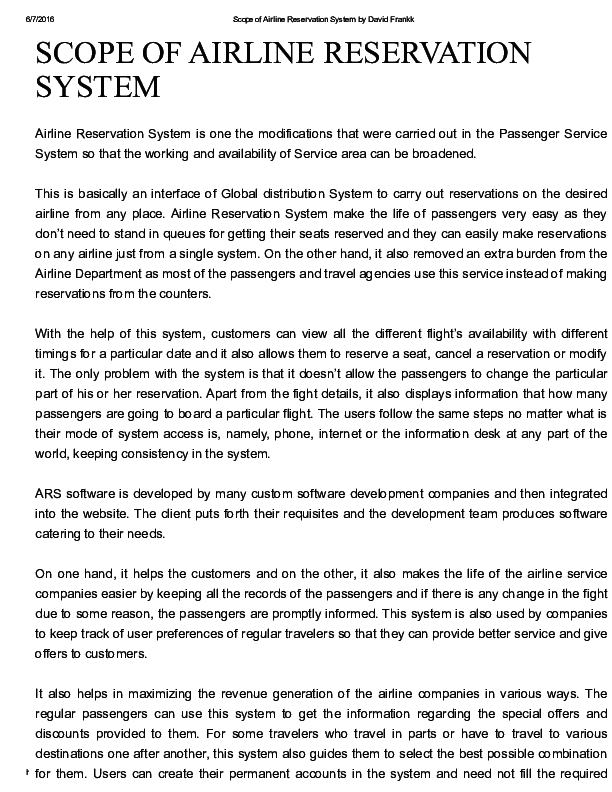
**6. CONCLUSION:**

Online ticket booking system is an application where the customer can book a ticket online and 24\*7 hours a day from anyplace in the world. Customers can also interact with the ticket booking website to know any other details they want. Online ticket booking system has-been developed successfully.

System performance is also found to be satisfactory. This is a user-friendly application. Through this application, the cost can be reduced and efficiency is increased. There are several procedures that can be selected by customers. With the help of this application customers can book tickets, can know the status of a flight, bus or trains, a Source station and destination can be chosen according to their choice, can select seats.

can choose the time, and pay through the portal after reaching the station or airport. Thus online ticket booking system target internal and external audiences. Online ticket booking system is very big to maintain but it always provides excellent facilities to accomplish the goal and help to reduce a complex paperwork process through mobile application.

This can be a benefit using online ticket booking system application rather searching on several websites. With the help of online ticket booking system records are maintained and the database is updated with time to time. Through Online ticket booking system, technologies and features have been introduced.

**7. FUTURE SCOPE**

**8**. **APPENDIX:**

**8.1 Source code:**

import numpy as np import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import f1\_score, classification\_report, confusion\_matrix import warnings

import pickle

from scipy import stats

warnings.filterwarnings('ignore') plt.style.use('fivethirtyeight')

data = pd.read\_csv("/content/sample\_data/Data\_Train.csv") data.head()

data.shape

data.isnull().sum()

data.dropna(inplace=True) data.isnull().sum()

Category = ['Airline', 'Source', 'Destination', 'Additional\_Info']

for i in Category:

print(i, data[i].unique())

category\_cols=data.select\_dtypes (include=['object']) .columns category\_cols

#plotting a barchart for each of the categorical value #for column in category\_cols:

#plt.figure(figsize=(20,4)) #plt.subplot(121)

#data[column].value\_counts().plot(kind='bar') #plt.title(column)

data.Route = data.Route.str.split('->') data['City1']=data.Route.str[0]

data['City2']=data.Route.str[1] data['City3']=data. Route.str[2] data['City4']=data. Route.str[3] data['City5']=data.Route.str[4] data['City6']=data. Route.str[5]

data.Date\_of\_Journey=data.Date\_of\_Journey.str.split('/') data.Date\_of\_Journey

#Treating the data\_column

data['Date']=data.Date\_of\_Journey.str[0] data['Month']=data.Date\_of\_Journey.str[1] data['Year']=data.Date\_of\_Journey.str[2] data.Dep\_Time=data.Dep\_Time.str.split(':') data['Dep\_Time\_Hour']=data.Dep\_Time.str[0] data['Dep\_Time\_Mins']=data.Dep\_Time.str[1]

data.Arrival\_Time=data.Arrival\_Time.str.split(' ') data['Arrival\_date']=data.Arrival\_Time.str[1]

data['Time\_of\_Arrival']=data. Arrival\_Time.str[0]

data['Time\_of\_Arrival']=data.Time\_of\_Arrival.str.split(':') data['Arrival\_Time\_Hour' ]=data.Time\_of\_Arrival.str[0]

data['Arrival\_Time\_Mins']=data.Time\_of\_Arrival.str[1]

#Next, we divide the 'Duration' column to 'Travel\_hours' and Travel\_mins' data.Duration=data.Duration.str.split(' ')

data['Travel\_Hours']=data.Duration.str[0]

data['Travel\_Hours']=data['Travel\_Hours'].str.split('h') data['Travel\_Hours']=data['Travel\_Hours'].str[0]

data.Travel\_Hours = data. Travel\_Hours

data['Travel\_Mins']=data.Duration.str[1]

#Next, we divide the 'Duration' column to 'Travel\_hours' and Travel\_mins' 24 data.Duration=data.Duration.str.split(' ')

Airline ['IndiGo' 'Air India' 'Jet Airways' 'SpiceJet' 'Multiple carriers' 'GoAir' 'Vistara' 'Air Asia' 'Vistara Premium economy' 'Jet Airways Business'

'Multiple carriers Premium economy' 'Trujet']

Source ['Banglore' 'Kolkata' 'Delhi' 'Chennai' 'Mumbai']

Destination ['New Delhi' 'Banglore' 'Cochin' 'Kolkata' 'Delhi' 'Hyderabad']

Additional\_Info ['No info' 'In-flight meal not included' 'No check-in baggage included' '1 Short layover' 'No Info' '1 Long layover' 'Change airports'

'Business class' 'Red-eye flight' '2 Long layover']

data['Duration'] = data['Duration'].astype(str) data['Travel\_Hours']=data.Duration.str[0]

data['Travel\_Hours']=data['Travel\_Hours'].str.split('h') data['Travel\_Hours']=data['Travel\_Hours'].str[0]

data.Travel\_Hours =data.Travel\_Hours

data['Travel\_Mins']=data.Duration.str[1]

data. Travel\_Mins=data.Travel\_Mins.str.split('m') data.Travel\_Mins=data.Travel\_Mins.str[0]

data. Total\_Stops.replace('non\_stop', 0, inplace=True) data. Total\_Stops = data. Total\_Stops.str.split(' ')

data. Total\_Stops=data. Total\_Stops.str[0]

data. Additional\_Info.unique()

array(['No info', 'In-flight meal not included',

'No check-in baggage included', '1 Short layover', 'No Info', '1 Long layover', 'Change airports', 'Business class',

'Red-eye flight', '2 Long layover'], dtype=object)

data. Additional\_Info.replace('No Info', 'No info', inplace=True) data.isnull().sum

if 'City4' in data.columns and 'City5' in data.columns and 'City6' in data.columns: data.drop(['City4', 'City5', 'City6'], axis=1, inplace=True)

print(data.columns)

Index(['Airline', 'Date\_of\_Journey', 'Source', 'Destination', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration', 'Total\_Stops',

'Additional\_Info', 'Price', 'City1', 'City2', 'City3', 'Date', 'Month',

'Year', 'Dep\_Time\_Hour', 'Dep\_Time\_Mins', 'Arrival\_date',

'Time\_of\_Arrival', 'Arrival\_Time\_Hour', 'Arrival\_Time\_Mins', 'Travel\_Hours', 'Travel\_Mins'],

dtype='object')

data.drop(['Date\_of\_Journey','Route', 'Dep\_Time','Arrival\_Time','Duration'], axis=1, inplace=True) data.drop(['Time\_of\_Arrival'], axis=1, inplace=True)

data.isnull().sum() data.info()

data['City3'].fillna ('None', inplace=True) data['City2'].fillna ('None', inplace=True)

data['Arrival\_date'].fillna (data['Date'], inplace=True) data['Travel\_Mins'].fillna(0,inplace=True)

data.info()

data.Date=data.Date.astype('int64')

data.Month=data. Month.astype('int64') data.Year=data.Year.astype('int64')

data.Dep\_Time\_Hour=data.Dep\_Time\_Hour.astype('int64') data.Dep\_Time\_Hour=data.Dep\_Time\_Hour.astype('int64') data.Dep\_Time\_Mins=data.Dep\_Time\_Mins.astype('int64')

data. Arrival\_date=data.Arrival\_date.astype("int64")

data.Arrival\_Time\_Hour=data. Arrival\_Time\_Hour.astype('int64') data. Arrival\_Time\_Mins=data. Arrival\_Time\_Mins.astype('int64')

data.info()

<class 'pandas.core.frame.DataFrame'> Int64Index: 10682 entries, 0 to 10682 Data columns (total 19 columns):

# Column Non-Null Count Dtype

1. Airline 10682 non-null object
2. Source 10682 non-null object
3. Destination 10682 non-null object
4. Total\_Stops 10682 non-null object
5. Additional\_Info 10682 non-null object
6. Price 10682 non-null int64
7. City1 10682 non-null object
8. City2 10682 non-null object
9. City3 10682 non-null object
10. Date 10682 non-null int64
11. Month 10682 non-null int64
12. Year 10682 non-null int64
13. Dep\_Time\_Hour 10682 non-null int64
14. Dep\_Time\_Mins 10682 non-null int64
15. Arrival\_date 10682 non-null int64
16. Arrival\_Time\_Hour 10682 non-null int64
17. Arrival\_Time\_Mins 10682 non-null int64
18. Travel\_Hours 10682 non-null object
19. Travel\_Mins 10682 non-null object dtypes: int64(9), object(10)

memory usage: 1.6+ MB

data[data['Travel\_Hours']=='5m']

data.Travel\_Hours=data.Travel\_Hours.astype('int64')

categorical=['Airline', 'Source', 'Destination', 'Additional Info', 'City1']

l\_Stops', 'Date', 'Month', 'Year', 'Dep\_Time\_Hour', 'Dep\_Time\_Mins', 'Arrival\_date', 'Arrival\_Time\_Hour', 'Arrival\_T

import seaborn as sns c=1

plt.figure(figsize=(20,45)) for i in categorical:

plt.subplot(6,3,c)

sns.scatterplot(x=data[i],y=data.Price) plt.xticks(rotation=90)

c=c+1

plt.show()

KeyError Traceback (most recent call last)

/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py in get\_loc(self, key, method, tolerance) 3628 try:

-> 3629 return self.\_engine.get\_loc(casted\_key) 3630 except KeyError as err:

 4 frames

pandas/\_libs/hashtable\_class\_helper.pxi in pandas.\_libs.hashtable.PyObjectHashTable.get\_item() pandas/\_libs/hashtable\_class\_helper.pxi in pandas.\_libs.hashtable.PyObjectHashTable.get\_item() KeyError: 'Additional Info'

The above exception was the direct cause of the following exception:

KeyError Traceback (most recent call last)

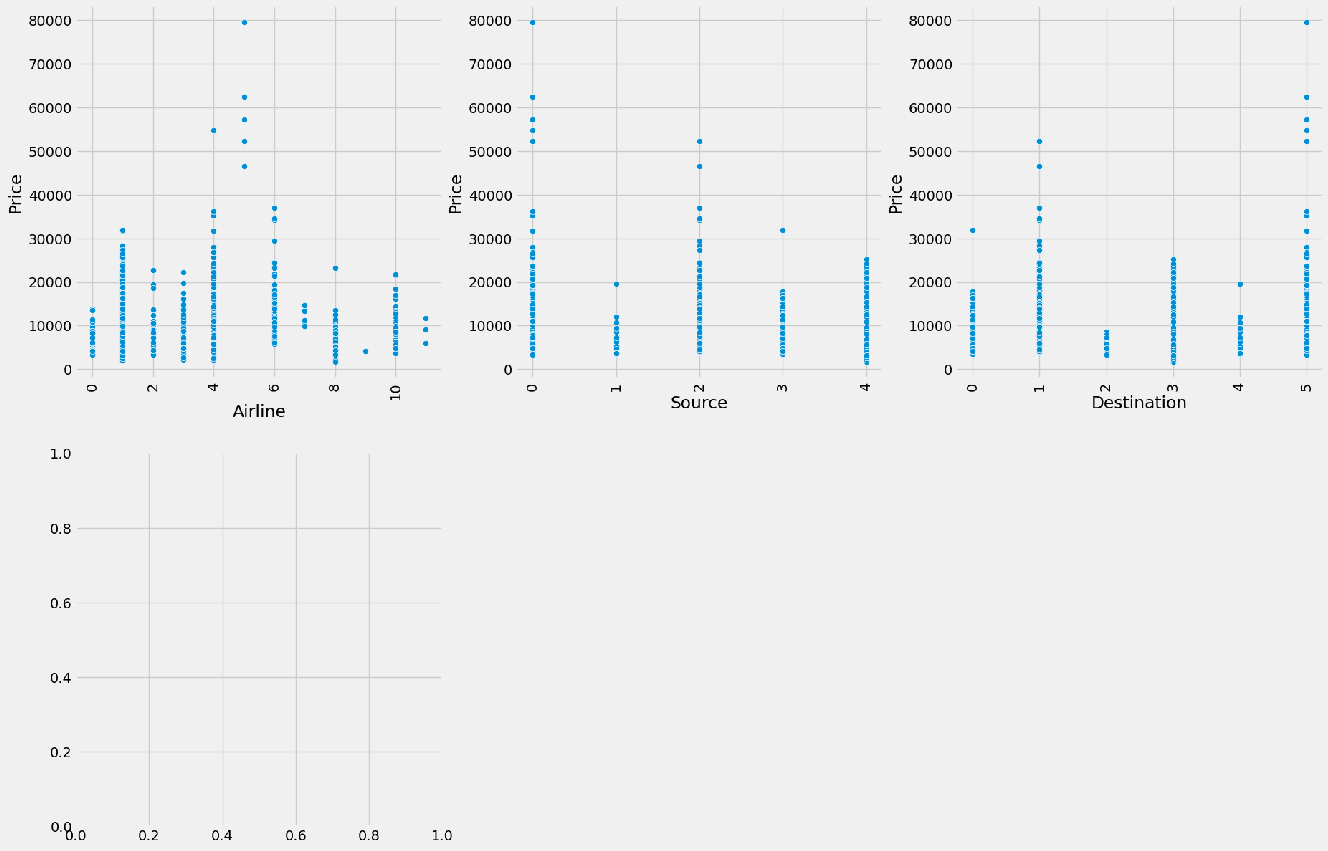
/usr/local/lib/python3.9/dist-packages/pandas/core/indexes/base.py in get\_loc(self, key, method, tolerance) 3629 return self.\_engine.get\_loc(casted\_key)

3630 except KeyError as err:

-> 3631 raise KeyError(key) from err 3632 except TypeError:

3633 # If we have a listlike key, \_check\_indexing\_error will raise

KeyError: 'Additional Info'



SEARCH STACK OVERFLOW

data[data. Price>50000]

data.head()

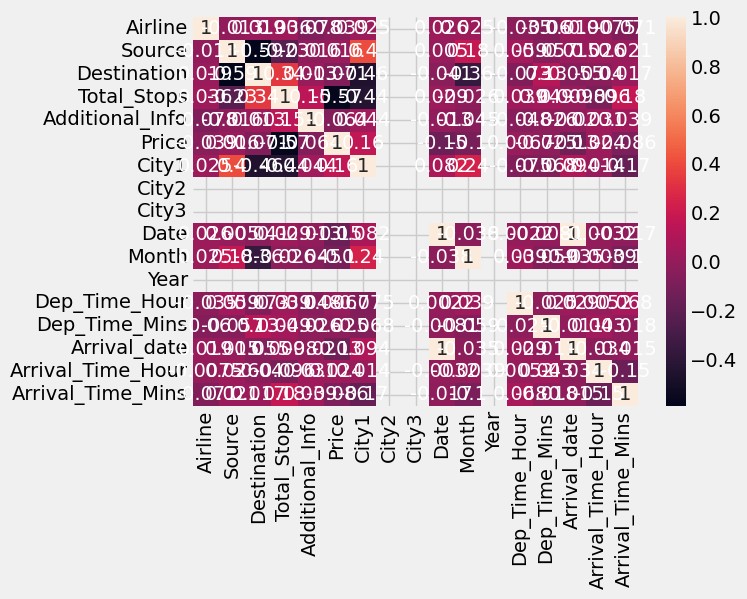
pd.set\_option('display.max\_columns',25) data.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Source** | **Destination** | **Total\_Stops** | **Additional\_Info** | **Price** | **City1** | **City2** | **City3** | **Date** | **Month** | **Year** | **D** |
| **0** 3 | 0 | 5 | 4 | 7 | 3897 | 18 | 0 | 0 | 24 | 3 | 2019 |  |
| **1** 1 | 3 | 0 | 1 | 7 | 7662 | 84 | 0 | 0 | 1 | 5 | 2019 |  |
| **2** 4 | 2 | 1 | 1 | 7 | 13882 | 118 | 0 | 0 | 9 | 6 | 2019 |  |
| **3** 3 | 3 | 0 | 0 | 7 | 6218 | 91 | 0 | 0 | 12 | 5 | 2019 |  |
| **4** 3 | 0 | 5 | 0 | 7 | 13302 | 29 | 0 | 0 | 1 | 3 | 2019 |  |

data['Year'].max()

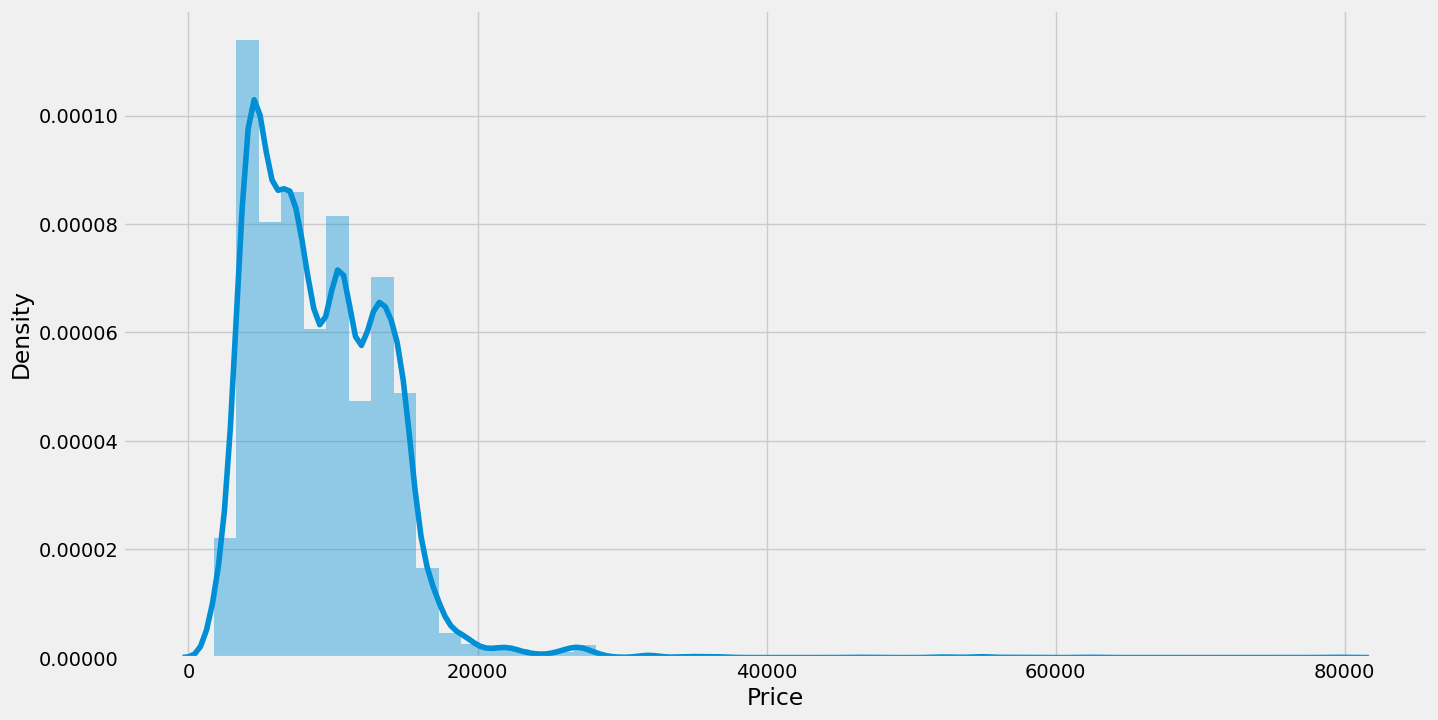
sns.heatmap (data.corr(), annot=True)

<Axes: >



plt.figure(figsize=(15,8)) sns.distplot(data.Price)

<Axes: xlabel='Price', ylabel='Density'>



import seaborn as sns

sns.boxplot(data['Price'])

c=1

for i in numerical:

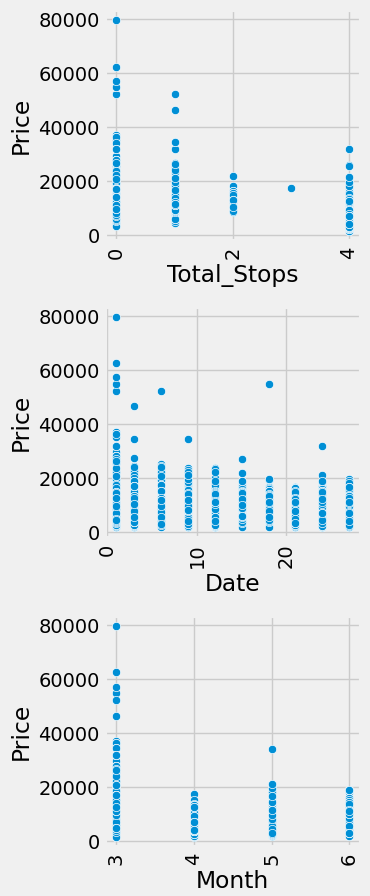
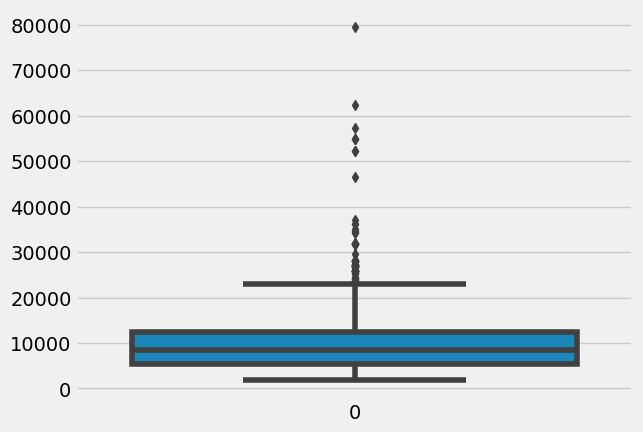
plt.figure(figsize=(10,20)) plt.subplot(6,3,c)

sns.scatterplot(x = data[i], y=data.Price) plt.xticks(rotation=90)

#plt. tight\_Layout (pad=3.0)

C=c+1

plt.show()



from sklearn.preprocessing import LabelEncoder le = LabelEncoder()

data.Airline = le.fit\_transform (data. Airline) data.Source = le.fit\_transform(data.Source)

data.Destination = le.fit\_transform(data. Destination) data.Total\_Stops= le.fit\_transform(data. Total\_Stops) data.City1=le.fit\_transform(data.City1)

data.City2=le.fit\_transform(data.City2) data.City3=le.fit\_transform(data.City3)

data.Additional\_Info = le.fit\_transform(data. Additional\_Info) data.head()

data.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Source** | **Destination** | **Total\_Stops** | **Additional\_Info** | **Price** | **City1** | **City2** | **City3** | **Date** | **Month** | **Year D** |
| **0** 3 | 0 | 5 | 4 | 7 | 3897 | 18 | 0 | 0 | 24 | 3 | 2019 |
| **1** 1 | 3 | 0 | 1 | 7 | 7662 | 84 | 0 | 0 | 1 | 5 | 2019 |
| **2** 4 | 2 | 1 | 1 | 7 | 13882 | 118 | 0 | 0 | 9 | 6 | 2019 |
| **3** 3 | 3 | 0 | 0 | 7 | 6218 | 91 | 0 | 0 | 12 | 5 | 2019 |
| **4** 3 | 0 | 5 | 0 | 7 | 13302 | 29 | 0 | 0 | 1 | 3 | 2019 |

data = data[['Airline', 'Source', 'Destination', 'Date', 'Month', 'Year', 'Dep\_Time\_Hour', 'Dep\_Time\_Mins','Arrival\_ data.head()

# Airline Source Destination Date Month Year Dep\_Time\_Hour Dep\_Time\_Mins Arrival\_date Arrival\_Time\_Ho

**0** 3 0 5 24 3 2019 22 20 22

**1** 1 3 0 1 5 2019 5 50 1

**2** 4 2 1 9 6 2019 9 25 10

**3** 3 3 0 12 5 2019 18 5 12

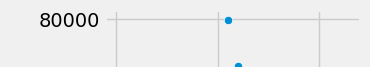
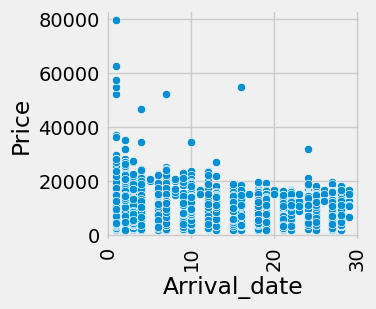
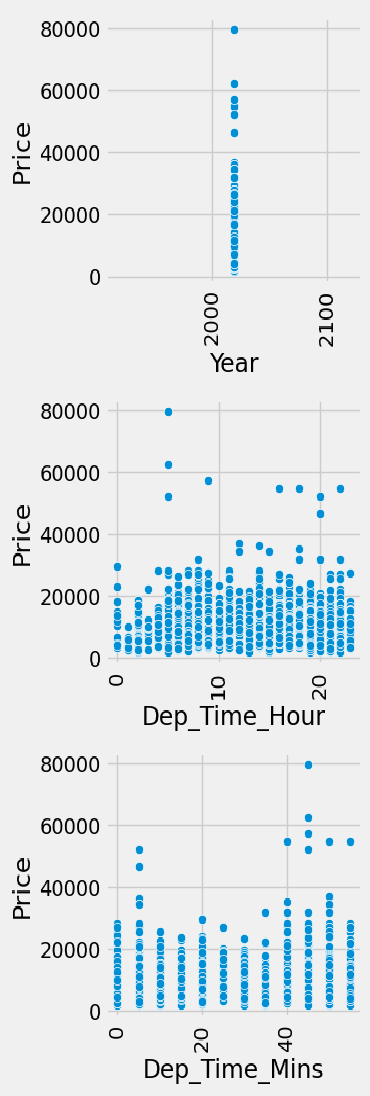
**4** 3 0 5 1 3 2019 16 50 1

### Scaling the Data

from sklearn.preprocessing import StandardScaler ss = StandardScaler()

datal = ss.fit\_transform(data)

datal = pd.DataFrame(datal, columns=data.columns) datal.head()



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Airline** | **Source** | **Destination** | **Date** | **Month** | **Year** | **Dep\_Time\_Hour** | **Dep\_Time\_Mins** | **Arrival\_date Arri** |
| **0** -0.410934 | -1.658354 | 2.416648 | 1.237192 | -1.467619 | 0.0 | 1.654162 | -0.234832 | 0.955658 |
| **1** -1.261305 | 0.890262 | -0.973718 | -1.475375 | 0.250165 | 0.0 | -1.303018 | 1.363790 | -1.524701 |
| **2** 0.014251 | 0.040723 | -0.295645 | -0.531874 | 1.109057 | 0.0 | -0.607211 | 0.031605 | -0.461690 |
| **3** -0.410934 | 0.890262 | -0.973718 | -0.178060 | 0.250165 | 0.0 | 0.958355 | -1.034142 | -0.225465 |
| **4** -0.410934 | -1.658354 | 2.416648 | -1.475375 | -1.467619 | 0.0 | 0.610452 | 1.363790 | -1.524701 |

y = datal['Price']

x = datal.drop(columns = ['Price'], axis=1)

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=42)

x\_train.head()

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Airline** | **Source** | **Destination** | **Date** | **Month** | **Year** | **Dep\_Time\_Hour** | **Dep\_Time\_Mins** | **Arrival\_date** | **A** |
| **10609** | 0.014251 | 1.739800 | 1.060501 | 0.529566 | 0.250165 | 0.0 | -0.955114 | -1.034142 | 0.483209 |  |
| **1034** | 1.714993 | 0.040723 | -0.295645 | 1.237192 | -0.608727 | 0.0 | 0.436500 | 1.097353 | 1.191883 |  |
| **8122** | 0.014251 | 0.040723 | -0.295645 | 1.591005 | 1.109057 | 0.0 | -1.824873 | -0.501269 | 1.546220 |  |
| **4779** | 0.014251 | 0.890262 | -0.973718 | -1.475375 | -0.608727 | 0.0 | -1.129066 | 0.298042 | -1.524701 |  |
| **3207** | -0.410934 | 0.890262 | -0.973718 | 1.237192 | 0.250165 | 0.0 | 0.958355 | -1.034142 | 1.191883 |  |

x\_train.shape

(8544, 11)

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, AdaBoostRegressor rfr = RandomForestRegressor()

gb = GradientBoostingRegressor() ad = AdaBoostRegressor()

from sklearn.metrics import r2\_score,mean\_absolute\_error,mean\_squared\_error for i in [rfr, gb,ad]:

i.fit(x\_train,y\_train)

y\_pred=i.predict(x\_test)

test\_score=r2\_score (y\_test,y\_pred)

train\_score=r2\_score (y\_train, i.predict(x\_train)) if abs (train\_score-test\_score)<=0.2:

print(i)

print("R2 score is", r2\_score (y\_test,y\_pred))

print("R2 for train data", r2\_score (y\_train, i.predict(x\_train))) print("Mean Absolute Error is", mean\_absolute\_error(y\_pred,y\_test)) print("Mean Squared Error is", mean\_squared\_error(y\_pred,y\_test))

print("Root Mean Sqaured Error is", (mean\_squared\_error(y\_pred,y\_test, squared=False)))

AdaBoostRegressor()

R2 score is 0.1920075019031262

R2 for train data 0.2397620967213413

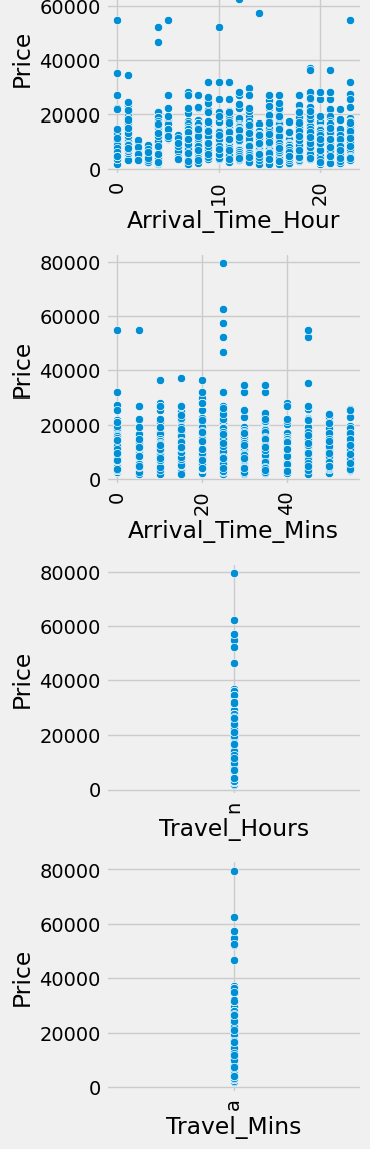
Mean Absolute Error is 0.7474281430734462 Mean Squared Error is 0.8023576762712405

Root Mean Sqaured Error is 0.8957442024770468

from sklearn.neighbors import KNeighborsRegressor from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error knn=KNeighborsRegressor()



svr=SVR()

dt=DecisionTreeRegressor() for i in [knn, svr,dt]:

i.fit(x\_train,y\_train)

y\_pred=i.predict(x\_test)

test\_score=r2\_score (y\_test,y\_pred)

train\_score=r2\_score (y\_train,i.predict(x\_train)) if abs(train\_score-test\_score)<=0.1:

print(i)

print('R2 Score is', r2\_score (y\_test,y\_pred))

print('R2 Score for train data', r2\_score (y\_train, i.predict(x\_train))) print('Mean Absolute Error is', mean\_absolute\_error(y\_test,y\_pred))

print('Mean Squared Error is', mean\_squared\_error(y\_test,y\_pred))

print('Root Mean Squared Error is', (mean\_squared\_error(y\_test, y\_pred, squared=False)))

KNeighborsRegressor()

R2 Score is 0.7357369816529409

R2 Score for train data 0.7900498333828809 Mean Absolute Error is 0.35463454315938664 Mean Squared Error is 0.26242008660326566

Root Mean Squared Error is 0.512269544871902 SVR()

R2 Score is 0.6399736388140904

R2 Score for train data 0.5969176412610055 Mean Absolute Error is 0.40820604052912457 Mean Squared Error is 0.3575155898574727

Root Mean Squared Error is 0.5979260739066935

from sklearn.model\_selection import cross\_val\_score for i in range(2,5):

cv=cross\_val\_score (rfr,x,y,cv=i) print(rfr,cv.mean())

RandomForestRegressor() 0.7913041688665847

RandomForestRegressor() 0.7922244703050666

RandomForestRegressor() 0.8010036945252192

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dense, Activation, Dropout from tensorflow.keras.optimizers import Adam

model = keras.Sequential()

model.add(Dense (7, activation = 'relu', input\_dim=11)) model.add(Dense (7, activation='relu'))

model.add(Dense(1, activation="linear")) model.summary()

Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

|  |  |  |
| --- | --- | --- |
| dense (Dense) | (None, 7) | 84 |
| dense\_1 (Dense) | (None, 7) | 56 |
| dense\_2 (Dense) | (None, 1) | 8 |

=================================================================

Total params: 148

Trainable params: 148

Non-trainable params: 0

model.compile(loss = 'mse', optimizer ='rmsprop', metrics=['mae']) model.fit(x\_train, y\_train, batch\_size = 20, epochs = 10)

Epoch 1/10

428/428 [==============================] - 2s 2ms/step - loss: 1.0014 - mae: 0.7697

Epoch 2/10

428/428 [==============================] - 1s 2ms/step - loss: 0.8445 - mae: 0.6867

Epoch 3/10

428/428 [==============================] - 1s 2ms/step - loss: 0.7852 - mae: 0.6515

Epoch 4/10

428/428 [==============================] - 1s 2ms/step - loss: 0.7519 - mae: 0.6348

Epoch 5/10

428/428 [==============================] - 1s 2ms/step - loss: 0.7240 - mae: 0.6244

Epoch 6/10

428/428 [==============================] - 1s 2ms/step - loss: 0.7012 - mae: 0.6170

Epoch 7/10

428/428 [==============================] - 1s 2ms/step - loss: 0.6856 - mae: 0.6114

Epoch 8/10

428/428 [==============================] - 1s 2ms/step - loss: 0.6749 - mae: 0.6093

Epoch 9/10

428/428 [==============================] - 1s 3ms/step - loss: 0.6686 - mae: 0.6066

Epoch 10/10

428/428 [==============================] - 1s 3ms/step - loss: 0.6601 - mae: 0.6033

<keras.callbacks.History at 0x7fc48177bdf0>

from sklearn.model\_selection import cross\_val\_score for i in range(2,5):

cv=cross\_val\_score (rfr,x,y,cv=i) print (rfr, cv.mean())

from sklearn.model\_selection import RandomizedSearchCV

param\_grid={'n\_estimators': [10, 30, 50, 70, 100], 'max\_depth': [None, 1, 2, 3], 'max\_features': ['auto', 'sqrt']} rfr=RandomForestRegressor()

rf\_res = RandomizedSearchCV(estimator=rfr, param\_distributions=param\_grid, cv=3, verbose=2,n\_jobs=-1) rf\_res.fit(x\_train,y\_train)

RandomForestRegressor() 0.7909852737870107

RandomForestRegressor() 0.7935431966039664

RandomForestRegressor() 0.8004797889513

Fitting 3 folds for each of 10 candidates, totalling 30 fits

**▸**

**RandomizedSearchCV**

**▸ estimator: RandomForestRegressor**

▸ RandomForestRegressor

gb = GradientBoostingRegressor()

# Define parameter grid to search over param\_grid = {

'n\_estimators': [50, 100, 150],

'max\_depth': [3, 5, 7],

'learning\_rate': [0.1, 0.01, 0.001]

}

# Perform randomized search over parameter grid

gb\_res = RandomizedSearchCV(estimator=gb, param\_distributions=param\_grid, cv=3, verbose=2, n\_jobs=-1) gb\_res.fit(x\_train, y\_train)

Fitting 3 folds for each of 10 candidates, totalling 30 fits

**▸**

**RandomizedSearchCV**

**▸ estimator: GradientBoostingRegressor**

▸ GradientBoostingRegressor

rfr=RandomForestRegressor (n\_estimators=10, max\_features='sqrt', max\_depth=None) rfr.fit(x\_train,y\_train)

y\_train\_pred=rfr.predict(x\_train) y\_test\_pred=rfr.predict(x\_test)

print("train accuracy", r2\_score (y\_train\_pred,y\_train)) print("test accuracy", r2\_score (y\_test\_pred,y\_test))

train accuracy 0.9297123116878924

test accuracy 0.7740524823579749

from sklearn.model\_selection import cross\_val\_score for i in range(2,5):

cv=cross\_val\_score (gb, x,y,cv=i) print (rfr, cv.mean())

RandomForestRegressor(max\_features='sqrt', n\_estimators=10) 0.72661809392105

RandomForestRegressor(max\_features='sqrt', n\_estimators=10) 0.7287548229046766

RandomForestRegressor(max\_features='sqrt', n\_estimators=10) 0.728029951483208

gb=GradientBoostingRegressor (n\_estimators=10, max\_features='sqrt', max\_depth=None) gb.fit(x\_train,y\_train)

y\_train\_pred=gb.predict(x\_train) y\_test\_pred=gb.predict(x\_test)

print("train accuracy", r2\_score (y\_train\_pred,y\_train)) print("test accuracy", r2\_score (y\_test\_pred,y\_test))

train accuracy 0.636247261013868

test accuracy 0.24929533156419104

from sklearn.neighbors import KNeighborsRegressor from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error knn=KNeighborsRegressor()

svr=SVR()

dt=DecisionTreeRegressor() for i in [knn, svr,dt]:

i.fit(x\_train,y\_train)

y\_pred=i.predict(x\_test)

test\_score=r2\_score (y\_test,y\_pred)

train\_score=r2\_score (y\_train, i.predict(x\_train)) if abs(train\_score-test\_score)<=0.1:

print(i)

KNeighborsRegressor( SVR()

knn=KNeighborsRegressor (n\_neighbors=2, algorithm= 'auto', metric\_params=None, n\_jobs=-1) knn.fit(x\_train,y\_train)

y\_train\_pred-knn.predict(x\_train) y\_test\_pred-knn.predict(x\_test)

print("train accuracy", r2\_score (y\_train\_pred,y\_train)) print("test accuracy", r2\_score (y\_test\_pred,y\_test))

train accuracy 0.636247261013868

test accuracy 0.24929533156419104

from sklearn.model\_selection import cross\_val\_score for i in range(2,5):

cv=cross\_val\_score (knn, x,y,cv=i) print (knn, cv.mean())

predicted\_values = pd.DataFrame({'Actual' :y\_test, 'Predicted' :y\_pred}) predicted\_values

KNeighborsRegressor(n\_jobs=-1, n\_neighbors=2) 0.6306338018391912

KNeighborsRegressor(n\_jobs=-1, n\_neighbors=2) 0.6447308601134175

KNeighborsRegressor(n\_jobs=-1, n\_neighbors=2) 0.664555765507016

|  |  |  |
| --- | --- | --- |
|  | **Actual** | **Predicted** |
| **4830** | -0.349272 | -0.455760 |
| **3771** | -0.251459 | -0.171648 |
| **1523** | -0.677410 | 0.638830 |
| **3393** | 1.562086 | 0.826648 |
| **4169** | -0.232157 | -0.719485 |
| **...** | ... | ... |
| **9869** | -0.968245 | -0.614949 |
| **10061** | -0.354477 | -0.354477 |

prices=rfr.predict(x\_test)

**6911** -0.348404 -0.348404

**8616** 1.098398 1.138303

**8988** 1.254551 1.254551

2137 rows × 2 columns

price\_list = pd.DataFrame({'Price': prices}) price\_list

|  |  |
| --- | --- |
|  | **Price** |
| **0** | -0.541116 |
| **1** | -0.057338 |
| **2** | 0.496086 |
| **3** | 0.964626 |
| **4** | -0.683179 |
| **...** | ... |
| **2132** | -0.614212 |
| **2133** | -0.549343 |
| **2134** | -0.374603 |
| **2135** | 0.738894 |
| **2136** | 1.056930 |

2137 rows × 1 columns