FLIGHT BOOKING

Introduction

The flight ticket buying system is to purchase a ticket many days prior to flight take-off so as to stay away from the effect to the most extreme charge. Mostly, aviation routes don't agree this procedure. Plane organizations may diminish the cost at the time, they need to build the market and at the time when the tickets are less accessible. They may maximize the costs. So, the cost may rely upon different factors. To foresee the costs this venture uses AI to exhibit the ways of flight tickets after some time. All organization have the privilege and opportunity to change booking.

OVER VIEW

The Flight ticket prices increase or decrease every now and then depending on various factors like timing of the flights, destination, duration of flights. In the proposed system a predictive model will be created by applying machine learning algorithms to the collected historical data of flights. Optimal timing for airline ticket purchasing from the consumer's perspective is challenging principally because buyers have insufficient information for reasoning about future price movements. In this project we majorly targeted to uncover underlying tends of flight prices in India using historical data and also to suggest the best time to buy a flight ticket.

PURPOSES

- Motivation is to help people who tends to pay more for the flight fare ticket and for those who are naive to this booking tickets process. This will also help us to get more exposure to the machine learning techniques that will help us to excel and improve in the existing skills.

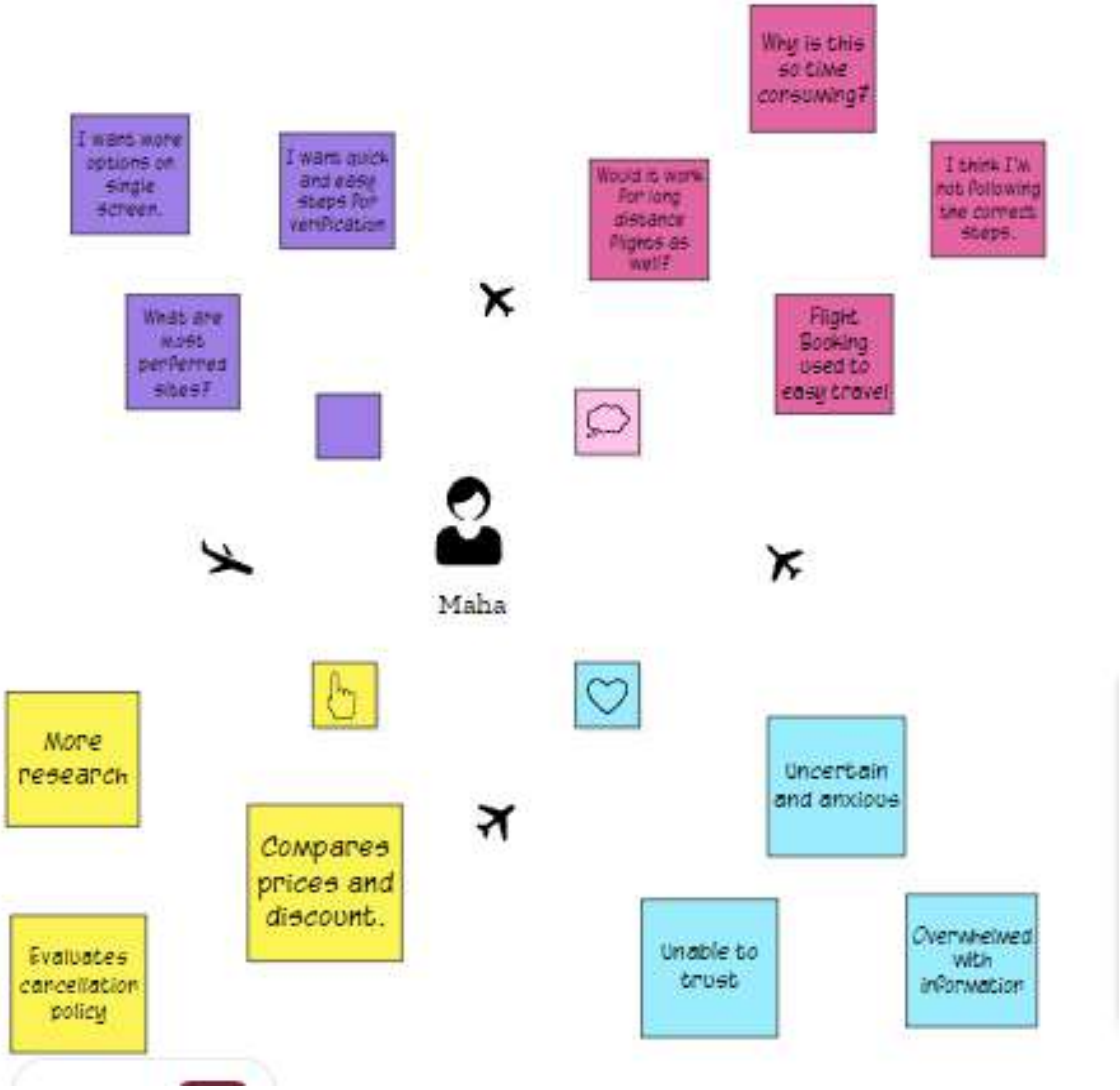
- To get effective price for the customers.

- Make UI user friendly.

- Use of various ML methods to know more about dataset and get accurate results.

PROBLEM DEFINITION & DESIGN THINKING

EMPATHY MAP:



BRAINSTROM:


K.MAHALAKSHMI
Intention to use Flight ticket booking app on Mobile Devices
A Booking is made for a single person
Perceived Trust
Insert Passenger Data
P.ESAKKIAMMAL@INDHU
The Process skip the whole POS part
Each Flight is identified by the flight number
The Duration of the flight number must to kept in the database
A booking course for not only for rich people
G.KALAISELVI
Track the passengers
The Arrival time is listed in the arrival airport time zone
Its use for common people also
Holidays in Flight Booking in so many members booking
N.BRINDHA
Create a new reservation through the regular booking procedure
Flight Booking in correct time because passengers to save the booking
Lowest cost to booking
Easy to Transport products in Airlines
MK

ADVANTAGES

- Traveler get the fare prediction handy using which it's easy to decide the airlines.
- Saves time in searching /deciding for airlines.

DISADVANTAGES

- Improper data will result in incorrect fare predictions.

CONCLUSION

Machine Learning algorithms are applied on the dataset to predicted the dynamic fare of flights. This gives the predicted values of flight fare to get a flight ticket at minimum cost. The values of R-squared obtained from the algorithm give the accuracy of the model. In the future, If more data could be accessed such as the current availability of seats, the predicted results will be more accurate. Finally, we conclude that this methodology is not preferred for performing this project. We can add more methods, more data for more accurate results.

APPENDIX

SOURCE CODE

IMPORTING PACKAGES

```
import matplotlib .pyplot as plt
import seaborn as sns
import pandas as pd
```

IMPORTING DATASET

```
train_data = pd.read_excel("/content/Data_Train (1).xlsx")
pd.set_option('display.max_columns', None)
train_data.head()
```

HEAD

```
train_data.head()
```

TAIL

```
train_data.head()
```

DATA PREPROCESSING

```
train_data.info()

train_data.isnull().sum()
```

ANALYISING THE CORRELTION FLIGHT ATTRIBUTES WITH OTHER ATTRIBUTES

```
plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")
```

```
plt.show()
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
))
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show()

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
))
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show()

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()from sklearn.model_selection import train_test_split
```

IMPORTING RANDOMFORESTREGRESSOR

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train, y_train)
```

TRAINING AND TESTING THE RECORDS OF DATASET FOR PREDICTION

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
 random_state = 42

y_pred = reg_rf.predict(X_test)
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
```

```
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

```
plt.show()tances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show() *ERROR OF PREDICTIOIN*

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
))
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show() *VISUALIZING THE ACCURACY OF PREDICTED RESULT*

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

IMPORTING DATASET

```
test_data = pd.read_excel("/content/Data_Train (1).xlsx")
```

HEAD

data_train *ERROR OF PREDICTIOIN*

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
```

```
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show()

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show().head()
```

TAIL

```
data_train.shape
```

DATA PREPROCESSING

```
train_data.info()

train_data.isnull().sum()
```

ANALYSING THE CORRELATION OF FLIGHT ATTRIBUTESWITH OTHER ATTRIBUTES

y = data_t

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()rain.iloc[:, 1]
y.head()
```

SPLITTING RECORDS FOR TRAING AND TESTING

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

TRAINING AND TESTING THE RECORDSS OF DATASET FOR PREDICTION

```
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)
```

IMPORT DECISIONTREEREGRESSOR

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state = 0)
regressor.fit(X, y)
```

TRAINING AND TESTING THE RECORDS OF DATASET FOR PREDICTION

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
 random_state = 42

y_pred = reg_rf.predict(X_test)
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
```

```
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show()

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
```

plt.show()

IMPORTING DATASET

```
test_data = pd.read_excel("/content/Data_Train (1).xlsx")
```

HEAD

```
data_train.head()
```

TAIL

```
data_train.shape
```

DATA PREPROCESSING

```
train_data.info()

train_data.isnull().sum()
```

ANALYSING THE CORRELATION OF FLIGHT ATTRIBUTESWITH OTHER ATTRIBUTES

```
y = data_train.iloc[:, 1]
y.head()
```

SPLITTING RECORDS FOR TRAING AND TESTING

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

TRAINING AND TESTING THE RECORDSS OF DATASET FOR PREDICTION

```
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)
```

ERROR OF PREDICTIOIN

```
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)
)
```

CLASSIFICATION REPORT

```
from sklearn import metrics
```

PRINTING THE FINAL ACCURACY SCORE OF PREDICTION

```
y_pred = regressor.predict(X_test)
```

VISUALIZING THE ACCURACY OF PREDICTED RESULT

```
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.colum
ns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```

RESULT

```
import matplotlib .pyplot as plt
import seaborn as sns
import pandas as pd
```

```
[ ] train_data = pd.read_excel("/content/Data_Train (1).xlsx")
    pd.set_option('display.max_columns', None)
    train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

```
[ ] train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10683 entries, 0 to 10682
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   Airline          10683 non-null  object
 1   Date_of_Journey  10683 non-null  object
 2   Source           10683 non-null  object
 3   Destination      10683 non-null  object
 4   Route            10682 non-null  object
 5   Dep_Time         10683 non-null  object
 6   Arrival_Time     10683 non-null  object
 7   Duration         10683 non-null  object
 8   Total_Stops      10682 non-null  object
 9   Additional_Info  10683 non-null  object
 10  Price            10683 non-null  int64 
dtypes: int64(1), object(10)
memory usage: 918.2+ KB
```

```
train_data["Duration"].value_counts()
```

```
2h 50m     550
1h 30m     386
2h 45m     337
2h 55m     337
2h 35m     329
          ...
31h 30m      1
30h 25m      1
42h 5m       1
4h 10m       1
47h 40m      1
Name: Duration, Length: 368, dtype: int64
```

```
train_data.dropna(inplace = True)
```

```
train_data.isnull().sum()
```

```
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
Price              0
dtype: int64
```

```
[ ] train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day
```

```
[ ] train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
```

```
[ ] train_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662	1	5
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882	9	6
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218	12	5
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302	1	3

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

```
[ ] train_data.head()
```

	Airline	Source	Destination	Route	Arrival_Time	Duration	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min
0	IndiGo	Banglore	New Delhi	BLR → DEL	01:10 22 Mar	2h 50m	non-stop	No info	3897	24	3	22	20
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	13:15	7h 25m	2 stops	No info	7662	1	5	5	50
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	04:25 10 Jun	19h	2 stops	No info	13882	9	6	9	25
3	IndiGo	Kolkata	Banglore	CCU → NAG → BLR	23:30	5h 25m	1 stop	No info	6218	12	5	18	5
4	IndiGo	Banglore	New Delhi	BLR → NAG → DEL	21:35	4h 45m	1 stop	No info	13302	1	3	16	50

```
# Arrival time is when the plane pulls up to the gate.
# Similar to Date_of_Journey we can extract values from Arrival_Time

# Extracting Hours
train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour

# Extracting Minutes
train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute
```

```
[ ]  # Time taken by plane to reach destination is called Duration
     # It is the differnce betwwen Departure Time and Arrival time


     # Assigning and converting Duration column into list
     duration = list(train_data["Duration"])

     for i in range(len(duration)):
         if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
             if "h" in duration[i]:
                 duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
             else:
                 duration[i] = "0h " + duration[i]           # Adds 0 hour

     duration_hours = []
     duration_mins = []
     for i in range(len(duration)):
         duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
         duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
     # Adding duration_hours and duration_mins list to train_data dataframe
     train_data["Duration_hours"] = duration_hours
     train_data["Duration_mins"] = duration_mins
```

```
[ ]  # Time taken by plane to reach destination is called Duration
     # It is the differnce betwwen Departure Time and Arrival time


     # Assigning and converting Duration column into list
     duration = list(train_data["Duration"])

     for i in range(len(duration)):
         if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
             if "h" in duration[i]:
                 duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
             else:
                 duration[i] = "0h " + duration[i]           # Adds 0 hour

     duration_hours = []
     duration_mins = []
     for i in range(len(duration)):
         duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
         duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
     # Adding duration_hours and duration_mins list to train_data dataframe
     train_data["Duration_hours"] = duration_hours
     train_data["Duration_mins"] = duration_mins
```

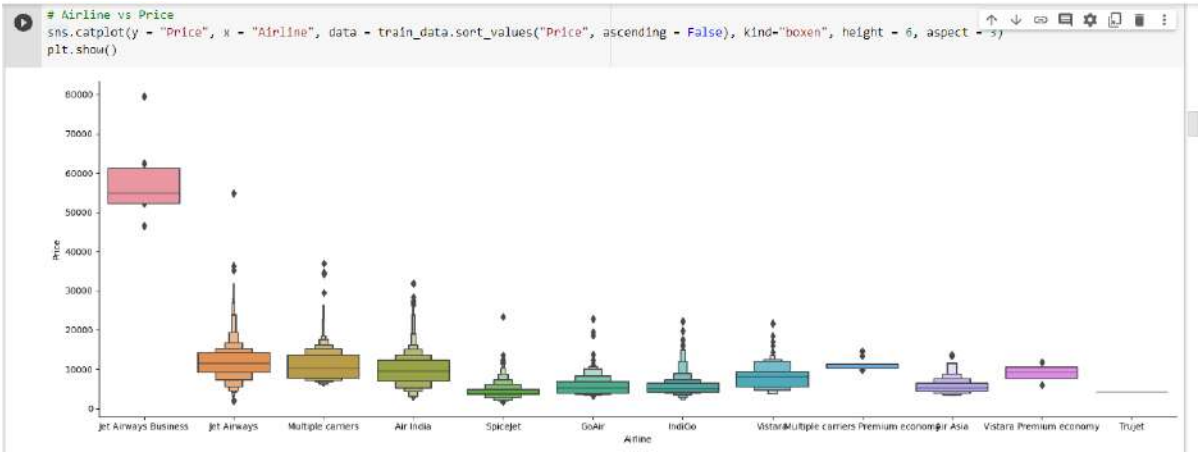
```
train_data.head()
```

	Airline	Source	Destination	Route	Total_Stops	Additional_Info	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_ho
0	IndiGo	Banglore	New Delhi	BLR → DEL	non-stop	No info	3897	24	3	22	20	1	10	
1	Air India	Kolkata	Banglore	CCU → IXR → BBI → BLR	2 stops	No info	7662	1	5	5	50	13	15	
2	Jet Airways	Delhi	Cochin	DEL → LKO → BOM → COK	2 stops	No info	13882	9	6	9	25	4	25	
3	IndiGo	Kolkata	Banglore	CCU → NAG →	1 stop	No info	6218	12	5	18	5	23	30	

```
train_data["Airline"].value_counts()
```

```
Jet Airways                          3849
IndiGo                               2053
Air India                            1751
Multiple carriers                    1196
SpiceJet                              818
Vistara                               479
Air Asia                              319
GoAir                                 194
Multiple carriers Premium economy      13
Jet Airways Business                    6
Vistara Premium economy                 3
Trujet                                  1
Name: Airline, dtype: int64
```

```
# From graph we can see that Jet Airways Business have the highest Price.
# Apart from the first Airline almost all are having similar median

# Airline vs Price
sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)
plt.show()
```

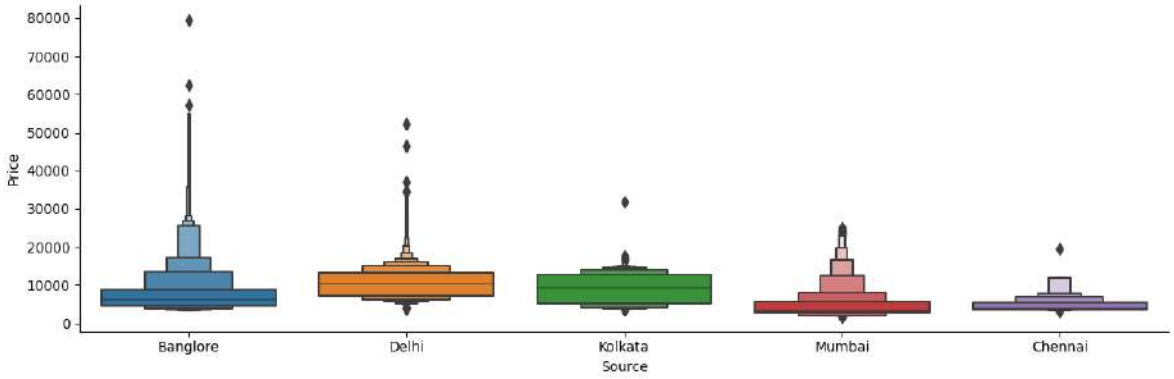


```
# As Airline is Nominal Categorical data we will perform OneHotEncoding

Airline = train_data[["Airline"]]

Airline = pd.get_dummies(Airline, drop_first= True)

Airline.head()
```

	Airline_Air India	Airline_GoAir	Airline_IndiGo	Airline_Jet Airways	Airline_Jet Airways Business	Airline_Multiple carriers	Airline_Multiple carriers Premium economy	Airline_SpiceJet	Airline_Trujet	Airline_Vistara	Air Pre
0	0	0	1	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	
2	0	0	0	1	0	0	0	0	0	0	
3	0	0	1	0	0	0	0	0	0	0	
4	0	0	1	0	0	0	0	0	0	0	

```
[ ] train_data["Source"].value_counts()
```

```
# Source vs Price

sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3)
plt.show()
```




```
# As Source is Nominal Categorical data we will perform OneHotEncoding

Source = train_data[["Source"]]

Source = pd.get_dummies(Source, drop_first= True)

Source.head()
```

	Source_Chennai	Source_Delhi	Source_Kolkata	Source_Mumbai
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	0

```
[ ] train_data["Destination"].value_counts()
```

```
Cochin       4536
Banglore     2871
Delhi        1265
New Delhi     932
```

```
train_data["Route"]
```

```
0                  BLR → DEL
1        CCU → IXR → BBI → BLR
2        DEL → LKO → BOM → COK
3              CCU → NAG → BLR
4              BLR → NAG → DEL
                 ...
10678              CCU → BLR
10679              CCU → BLR
10680              BLR → DEL
10681              BLR → DEL
10682    DEL → GOI → BOM → COK
Name: Route, Length: 10682, dtype: object
```

```
# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other

train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
train_data["Total_Stops"].value_counts()
```

```
1 stop      5625
non-stop    3491
2 stops     1520
```

```
# Concatenate dataframe --> train_data + Airline + Source + Destination

data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)
```

```
data_train.head()
```

	Airline	Source	Destination	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airl
0	IndiGo	Banglore	New Delhi	0	3897	24	3	22	20	1	10	2	50	
1	Air India	Kolkata	Banglore	2	7662	1	5	5	50	13	15	7	25	
2	Jet Airways	Delhi	Cochin	2	13882	9	6	9	25	4	25	19	0	
3	IndiGo	Kolkata	Banglore	1	6218	12	5	18	5	23	30	5	25	
4	IndiGo	Banglore	New Delhi	1	13302	1	3	16	50	21	35	4	45	

```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
```


```
data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)
```


```
data_train.head()
```

	Total_Stops	Price	Journey_day	Journey_month	Dep_hour	Dep_min	Arrival_hour	Arrival_min	Duration_hours	Duration_mins	Airline_Air India	Airline_GoAir	Airline
0	0	3897	24	3	22	20	1	10	2	50	0	0	
1	2	7662	1	5	5	50	13	15	7	25	1	0	
2	2	13882	9	6	9	25	4	25	19	0	0	0	
3	1	6218	12	5	18	5	23	30	5	25	0	0	
4	1	13302	1	3	16	50	21	35	4	45	0	0	

```
data_train.shape
```

```
(10682, 30)
```

```
test_data.head()
```

	Airline	Date_of_Journey	Source	Destination	Route	Dep_Time	Arrival_Time	Duration	Total_Stops	Additional_Info	Price
0	IndiGo	24/03/2019	Banglore	New Delhi	BLR → DEL	22:20	01:10 22 Mar	2h 50m	non-stop	No info	3897
1	Air India	1/05/2019	Kolkata	Banglore	CCU → IXR → BBI → BLR	05:50	13:15	7h 25m	2 stops	No info	7662
2	Jet Airways	9/06/2019	Delhi	Cochin	DEL → LKO → BOM → COK	09:25	04:25 10 Jun	19h	2 stops	No info	13882
3	IndiGo	12/05/2019	Kolkata	Banglore	CCU → NAG → BLR	18:05	23:30	5h 25m	1 stop	No info	6218
4	IndiGo	01/03/2019	Banglore	New Delhi	BLR → NAG → DEL	16:50	21:35	4h 45m	1 stop	No info	13302

```
[ ] # Preprocessing same as training data that we have done

    print("Test data Info")
    print("-"*75)
    print(test_data.info())

    print()
    print()

    print("Null values :")
    print("-"*75)
    test_data.dropna(inplace = True)
```

```
# Preprocessing same as training data that we have done

print("Test data Info")
print("-"*75)
print(test_data.info())

print()
print()

print("Null values :")
print("-"*75)
test_data.dropna(inplace = True)
print(test_data.isnull().sum())

# EDA

# Date_of_Journey
test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/%m/%Y").dt.day
test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)

# Dep_Time
test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
test_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

```
X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
       'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
       'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
       'Airline_Jet Airways', 'Airline_Jet Airways Business',
       'Airline_Multiple carriers',
       'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
       'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
       'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
       'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
       'Destination_Kolkata', 'Destination_New Delhi']]
X.head()
```

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

```
y = data_train.iloc[:, 1]
y.head()
```

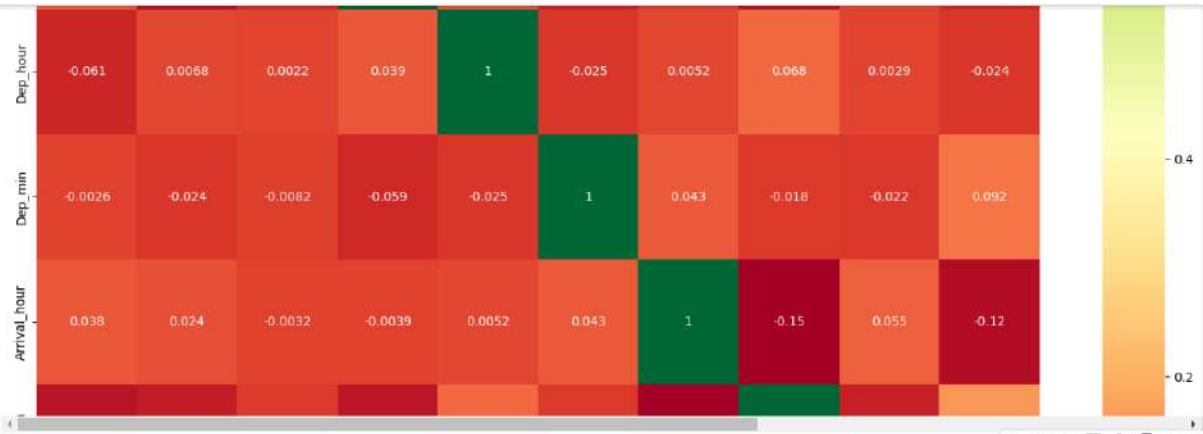
```
0     3897
1     7662
2    13882
3     6218
4    13302
Name: Price, dtype: int64
```

```
# Finds correlation between Independent and dependent attributes

plt.figure(figsize = (18,18))
sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

plt.show()
```

```
<ipython-input-186-fa209aad5cf8>:4: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to F
  sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")
```


```
plt.show()
```

```
<ipython-input-186-fa209aad5cf8>:4: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to F
  sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")
```




```
import sklearn
```

```
[ ] import sklearn
```

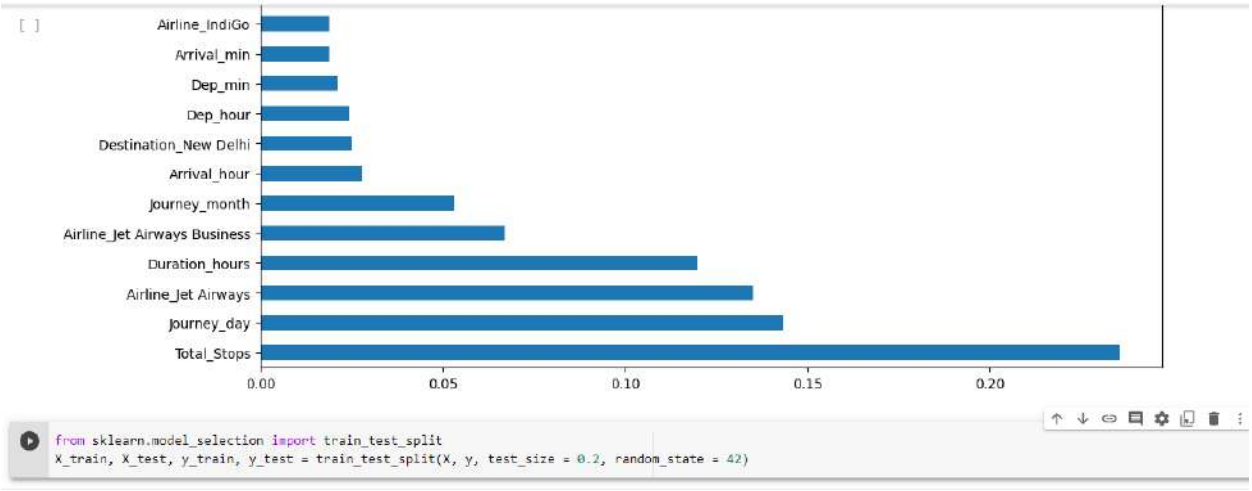
```
[ ] # Important feature using ExtraTreesRegressor

    from sklearn.ensemble import ExtraTreesRegressor
    selection = ExtraTreesRegressor()
    selection.fit(X, y)
```

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
print(selection.feature_importances_)
```

```
[2.35540961e-01 1.43368153e-01 5.31879845e-02 2.44551381e-02
 2.13237241e-02 2.79588161e-02 1.89602001e-02 1.20066050e-01
 1.75391141e-02 9.69056307e-03 1.90027462e-03 1.87675376e-02
 1.35026426e-01 6.70235062e-02 1.86162353e-02 8.34294329e-04
 3.44501001e-03 1.23555295e-04 5.06605600e-03 8.06786231e-05
 6.06487586e-04 1.35377054e-02 3.26163577e-03 6.48679485e-03
 7.94912307e-03 1.26467405e-02 7.19926903e-03 4.44067452e-04
 2.48938981e-02]
```

```
#plot graph of feature importances for better visualization
plt.figure(figsize = (12,8))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh')
plt.show()
```




```
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)
```

```
[ ] 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)
```

```
[ ] from sklearn.ensemble import RandomForestRegressor
    reg_rf = RandomForestRegressor()
    reg_rf.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor()
```

```
[ ] y_pred = reg_rf.predict(X_test)
```

```
[ ] reg_rf.score(X_train, y_train)
```

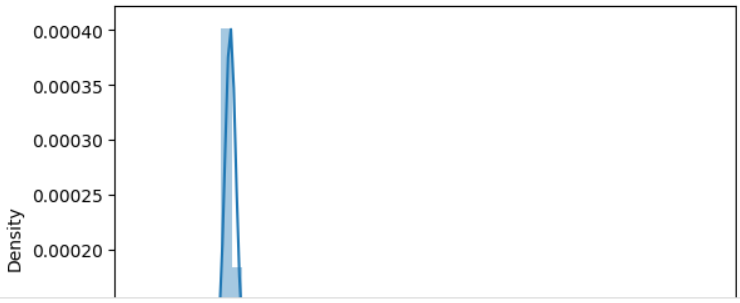
```
0.9536854837936594
```

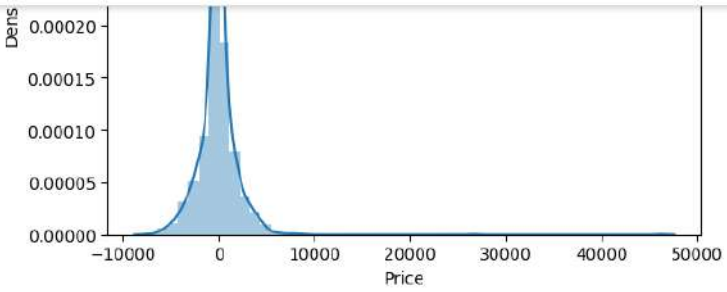
```
reg_rf.score(X_test, y_test)
```

```
0.7985747471067733
```

```
sns.distplot(y_test-y_pred)
plt.show()
```

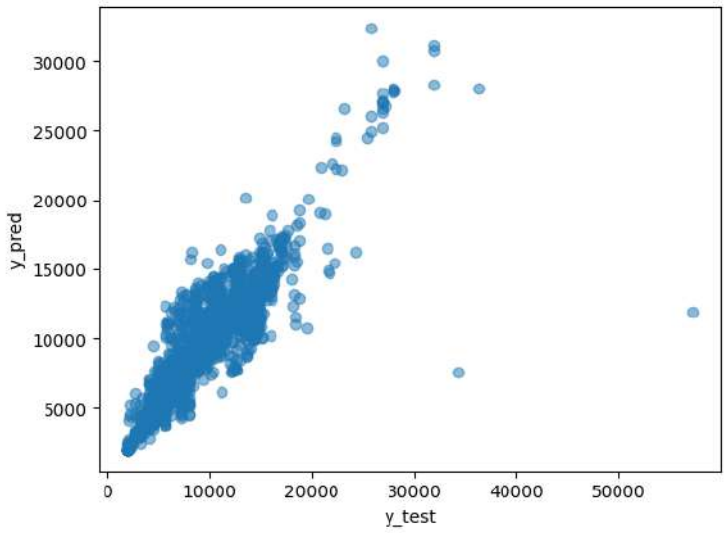
```
<ipython-input-196-75adb1dd5983>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(y_test-y_pred)
```






```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```

```
plt.ylabel("y_pred")
plt.show()
```




```
[ ] from sklearn import metrics
```

```
[ ] import numpy as np
    print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
    print('MSE:', metrics.mean_squared_error(y_test, y_pred))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 1169.4015682542383
MSE: 4343142.647992284
RMSE: 2084.0207887620227
```

```
[ ] # RMSE/(max(DV)-min(DV))
    2090.5509/(max(y)-min(y))
```

```
0.026887077025966846
```

```
[ ] metrics.r2_score(y_test, y_pred)
```

```
0.7985747471067733
```

```
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)
```

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, y)
```

```
DecisionTreeRegressor(random_state=0)
```

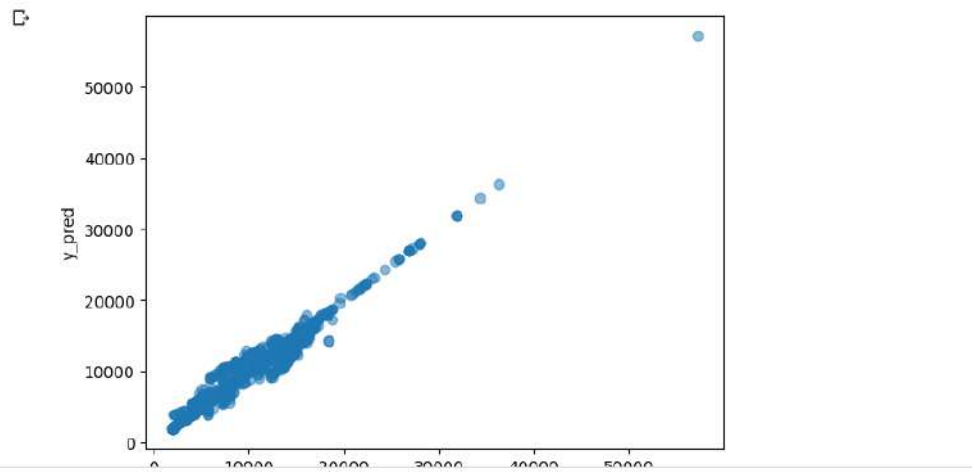
```
[ ] y_pred = regressor.predict(X_test)
```

```
[ ] from sklearn import metrics
```

```
[ ] import numpy as np
    print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
    print('MSE:', metrics.mean_squared_error(y_test, y_pred))
    print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 380.2993827573144
```

```
plt.scatter(y_test, y_pred, alpha = 0.5)
plt.xlabel("y_test")
plt.ylabel("y_pred")
plt.show()
```



```
[ ]  # RMSE/(max(DV)-min(DV))
     2090.5509/(max(y)-min(y))
```

```
0.026887077025966846
```

```
[ ]  metrics.r2_score(y_test, y_pred)
```

```
0.9658262236657966
```