 Project Report



**MACHINE LEARNING WITH PYTHON**

**Project Title :** OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

**Team Leader :** GOKULRAJ K

**Team Member :** SINGADURAI K

**Team Member :** JAGADHESHWARAN P

**Team Member :** VIJAYKARTHICK V

* **INTRODUCTION**
* Overview

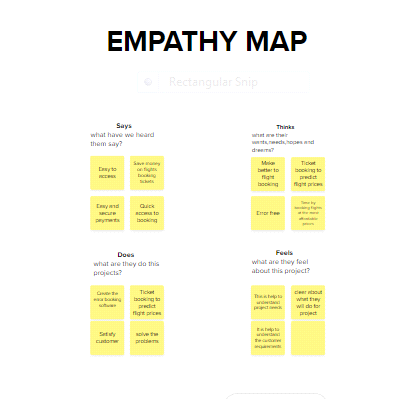
OPTIMIZING FLIGHT BOOKING DECISIONS THROUGH MACHINE LEARNING PRICE PREDICTIONS

* Purpose

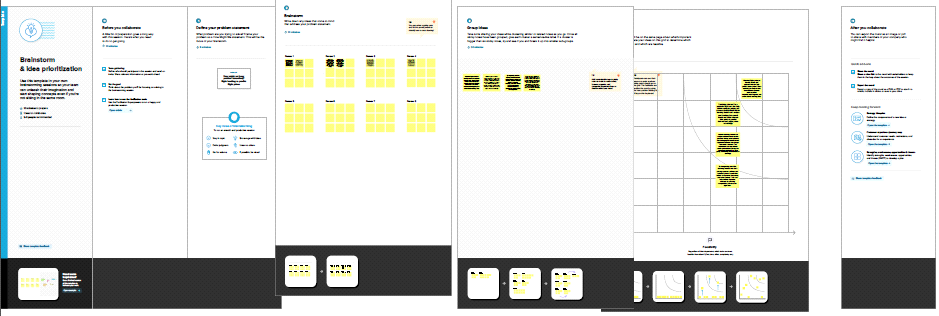
Predict the price flight ticket.

**2 Problem Definition & Design Thinking**

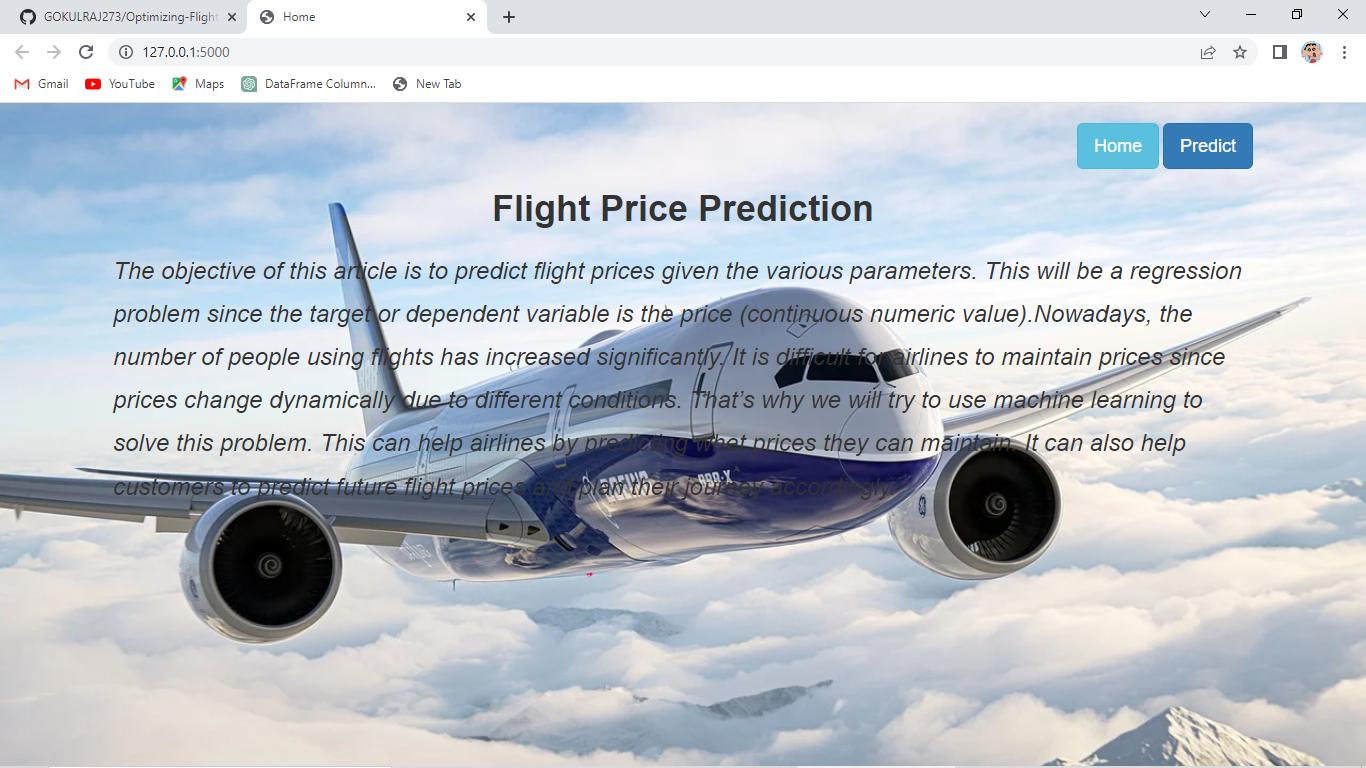
2.1 Empathy Map

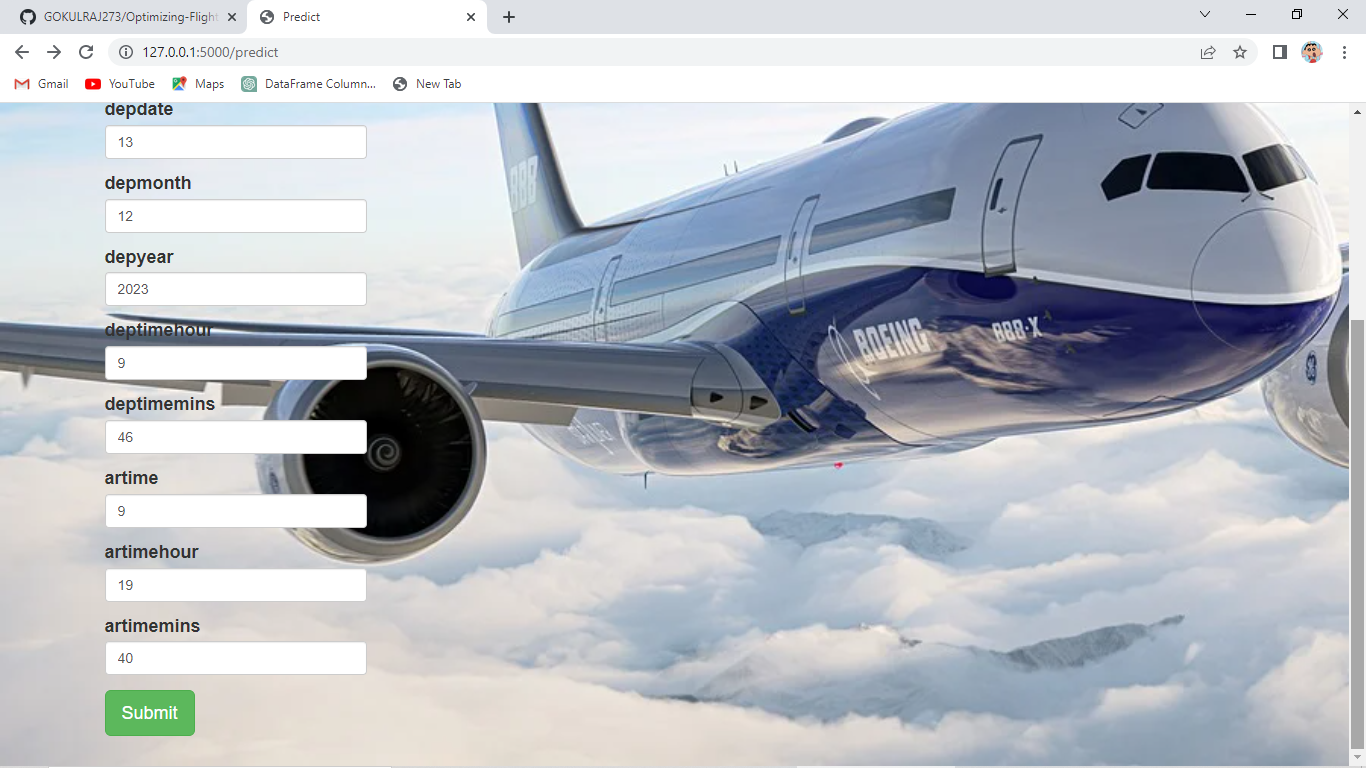


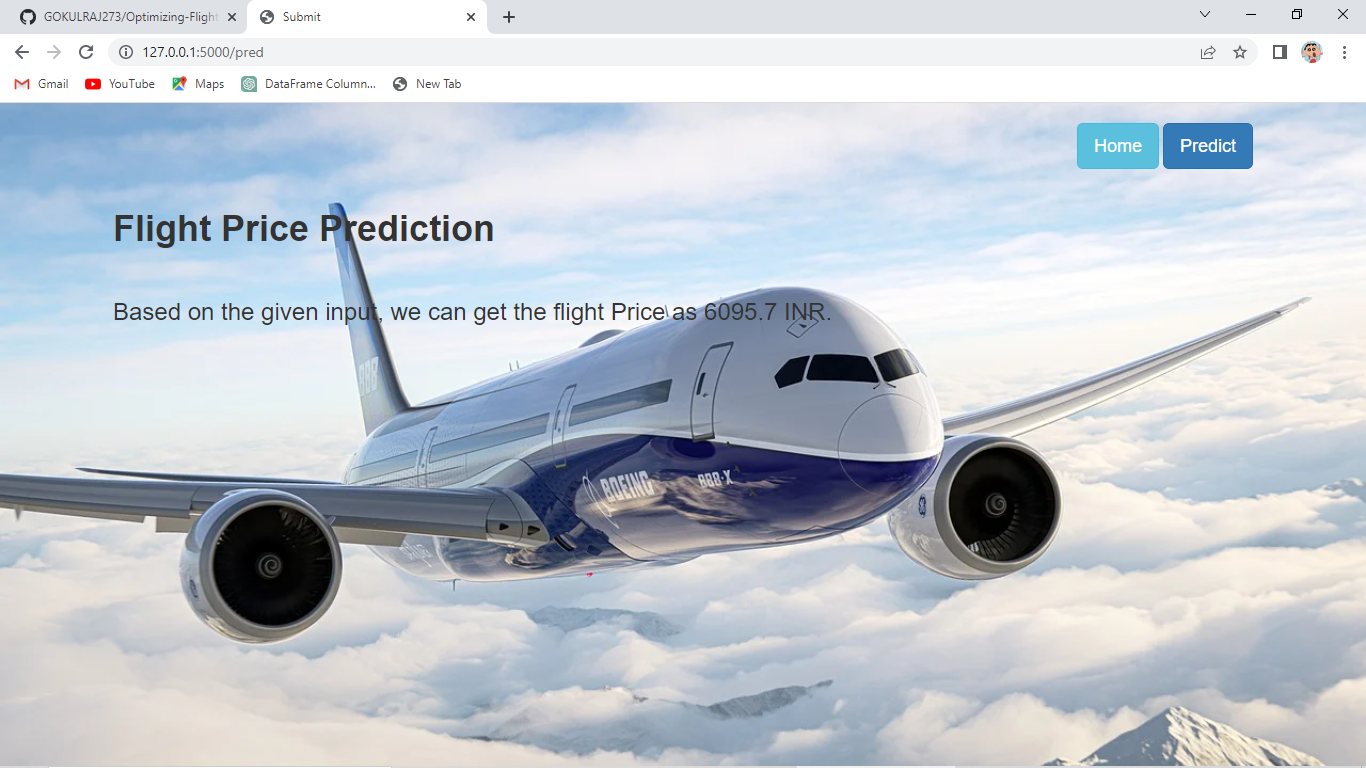
2.2 Ideation & Brainstorming Map



**RESULT:**

****

****

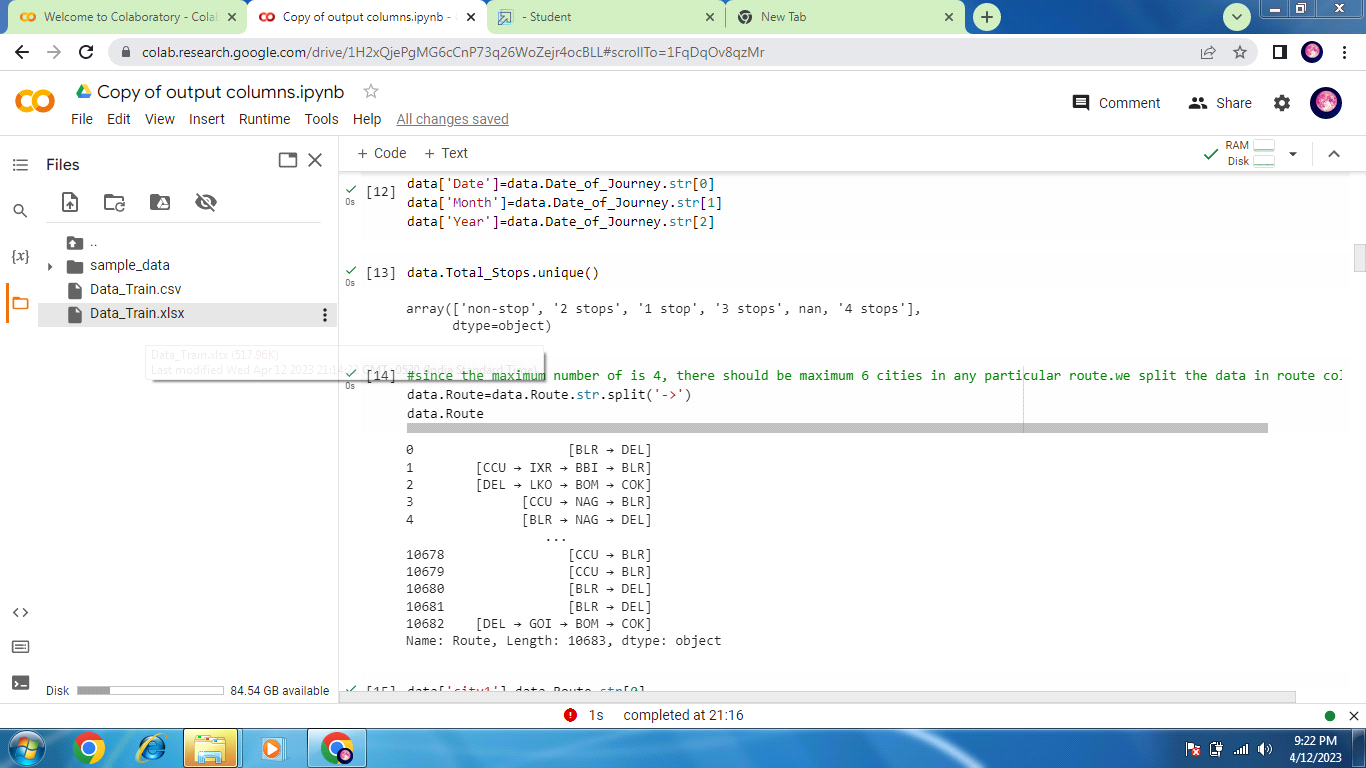
****

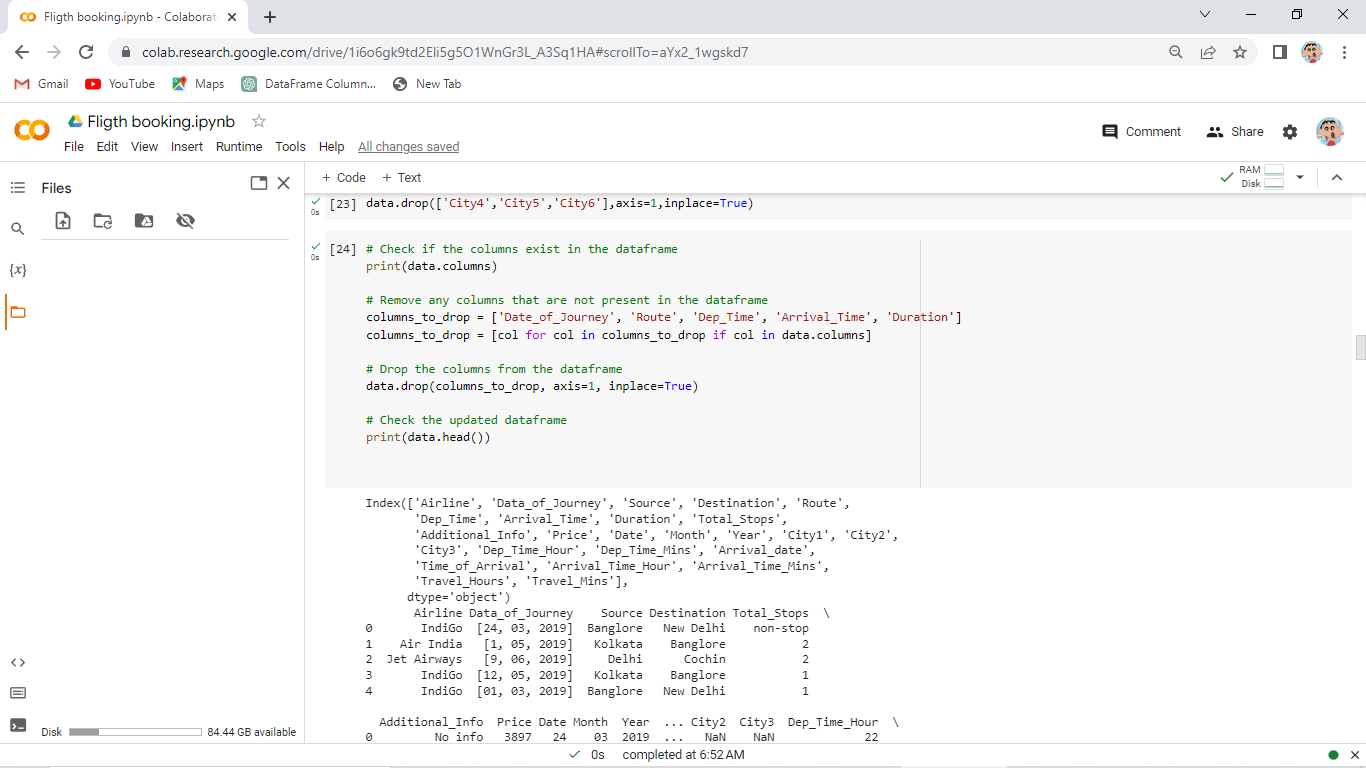
* Data Model:

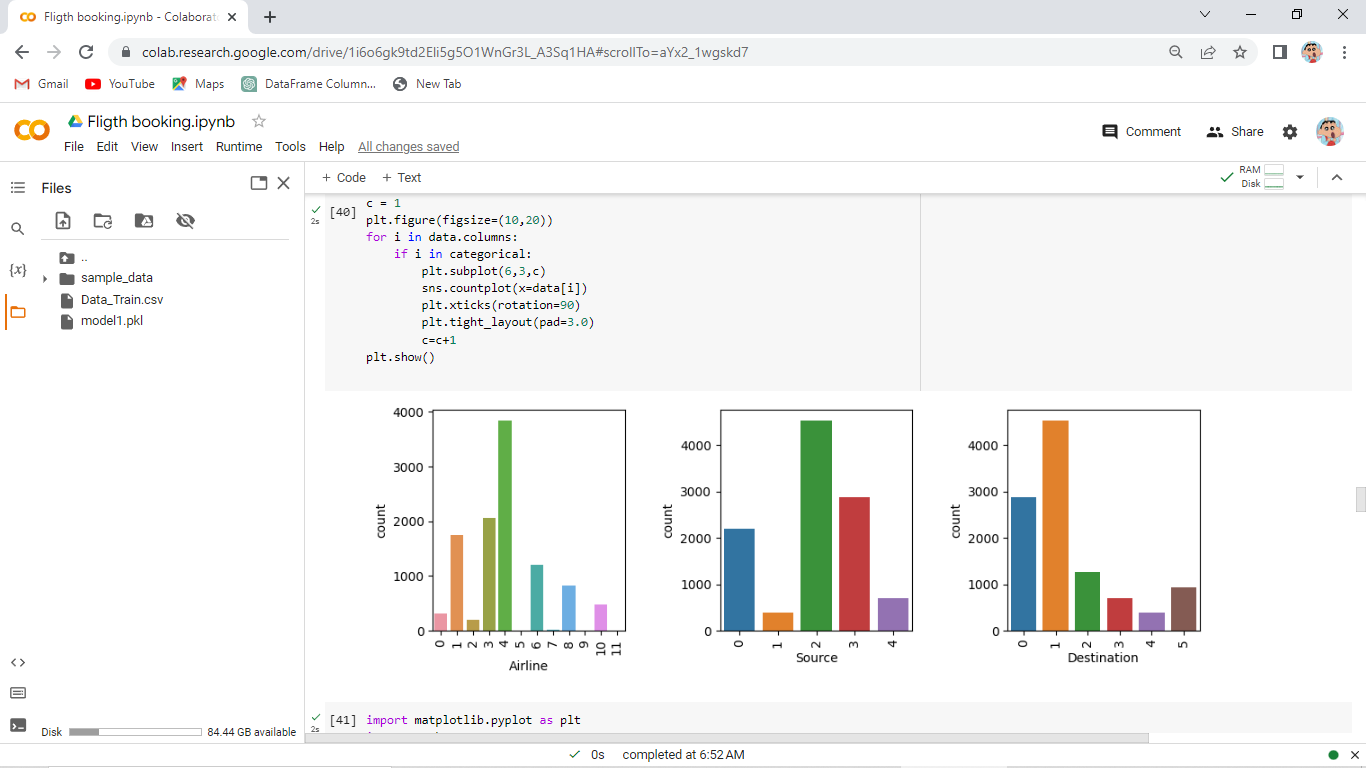
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Object name** | **Fields in the Object** | | | |
| obj1 |  | | | |
|  | Field label | Data type |  |
| Airline | int |
| source | int |
| price | int |
| obj2 |  |  |  | |
| Field label | Data type |  |
| Destinesion | int |
| Dup\_time | int |
| artime int | |

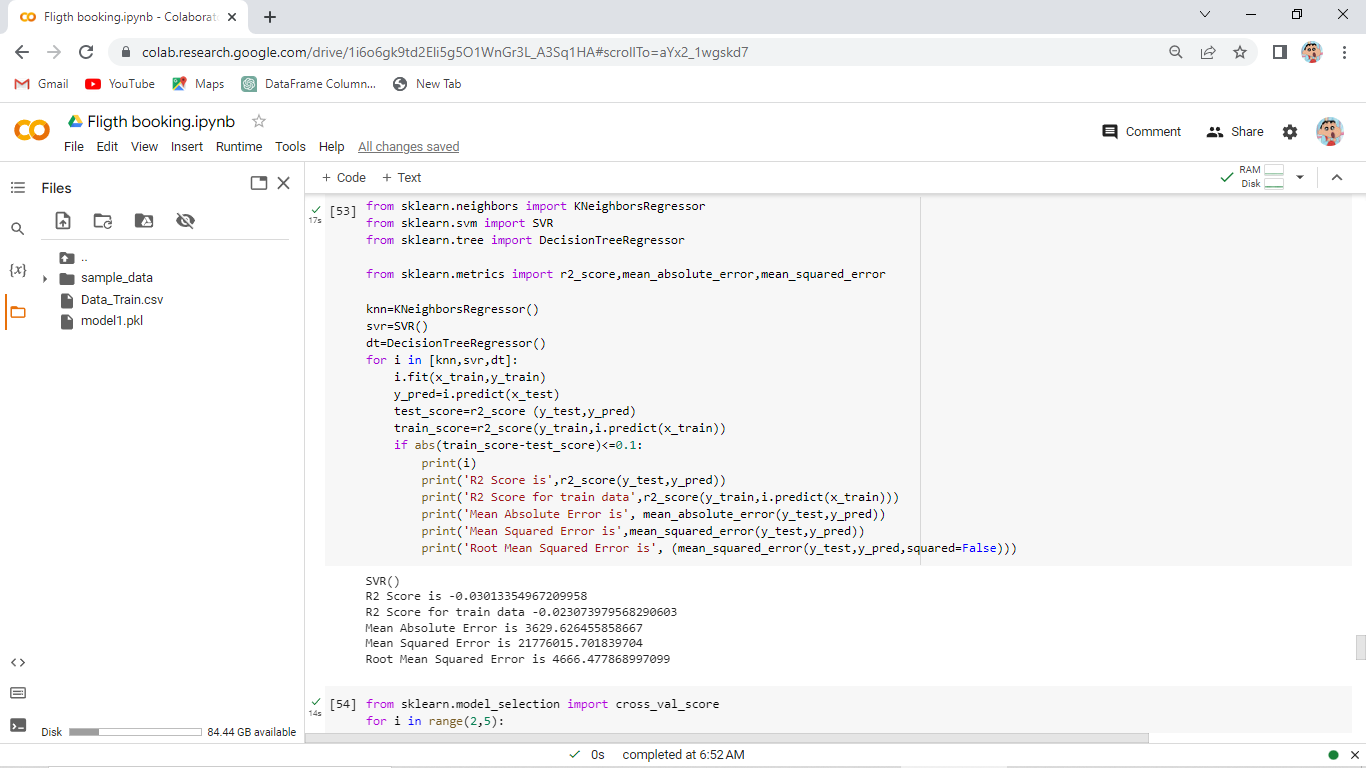
* **Activity & Screenshot**

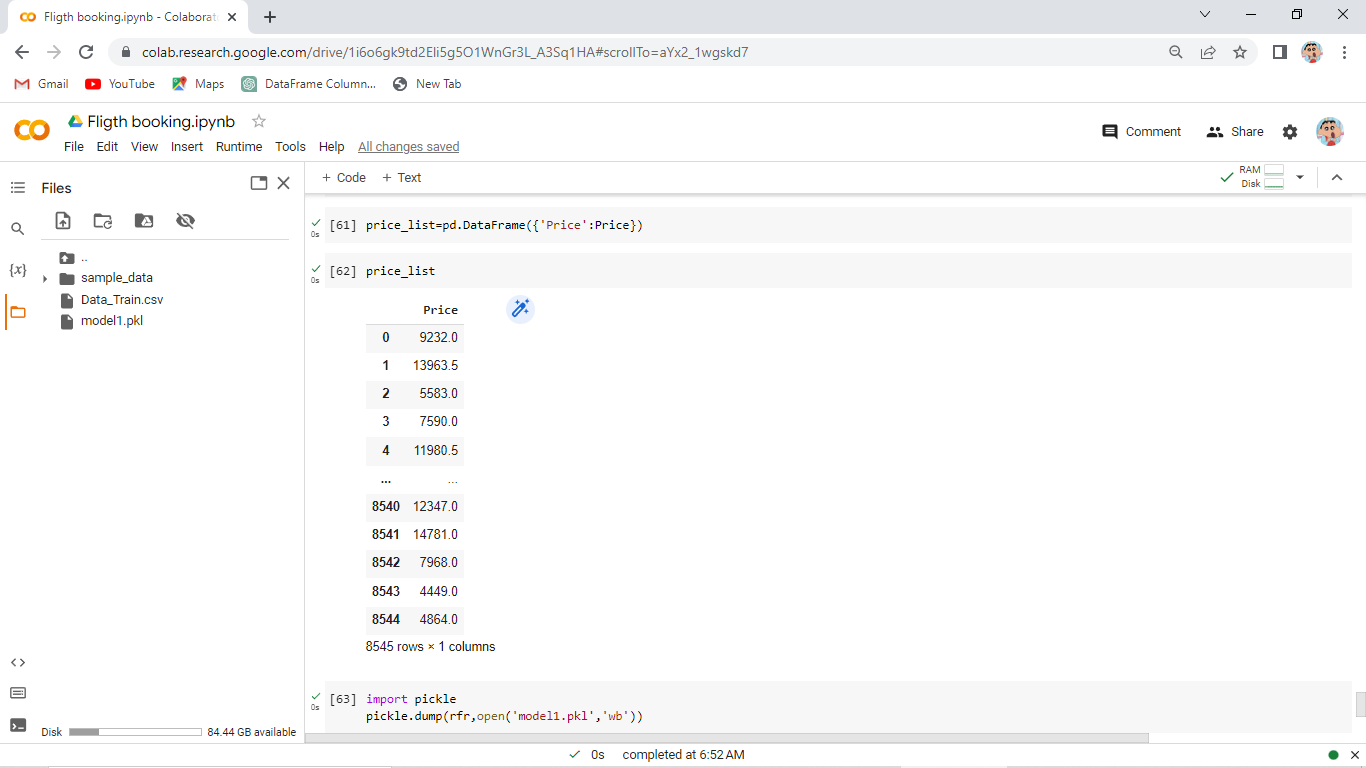
Attach the screenshots of your project activity along with the description.











* **Trailhead Profile Public URL**

**Team Lead:** [**https://trailblazer.me/id/gokuk14**](https://trailblazer.me/id/gokuk14)

**Team Member 1 -** [**https://trailblazer.me/id/sdurai49**](https://trailblazer.me/id/sdurai49)

**Team Member 2 -**

**Team Member 3 –**

 Project Report Template

* **ADVANTAGES & DISADVANTAGE**

List of advantages and disadvantages of the proposed solution

Advantage:

\* Learning video are helpful to understanding concept and help to create program.

\* It is help to predict the correct for flight booking.

\*Web design is well give attractive look to custumer.

Disadvantage:

\*Nothing to say

* **APPLICATIONS**

Google colab ,Anoconda,Pycharm Community.

* **CONCLUSION**

We gave our best.We make good prediction for the flight booking.

* **FUTURE SCOPE**

It will use predict price for fligth booking

**Appendix**

A Source Code :-

#Import Lib

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

from sklearn.metrics import classification\_report, confusion\_matrix

import warnings

import pickle

from scipy import stats

warnings.filterwarnings('ignore')

**#Read the Dataset**

data = pd.read\_csv('/content/Data\_Train.csv')

**#Data Preparation**

for i in data.select\_dtypes(include='object'):

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

data = data.rename(columns={'Date\_of\_Journey': 'Data\_of\_Journey'})

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

data['Date']=data['Data\_of\_Journey'].str[0]

data['Month']=data['Data\_of\_Journey'].str[1]

data['Year']=data['Data\_of\_Journey'].str[2]

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.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]

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]

data.Travel\_Mins=data.Travel\_Mins.str.split('a')

data.Travel\_Mins=data.Travel\_Mins.str[0]

**#Replacing Missing Values**

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.replace('No Info','No info',inplace=True)

data.drop(['City4','City5','City6'],axis=1,inplace=True)

# Check if the columns exist in the dataframe

print(data.columns)

# Remove any columns that are not present in the dataframe

columns\_to\_drop = ['Date\_of\_Journey', 'Route', 'Dep\_Time', 'Arrival\_Time', 'Duration']

columns\_to\_drop = [col for col in columns\_to\_drop if col in data.columns]

# Drop the columns from the dataframe

data.drop(columns\_to\_drop, axis=1, inplace=True)

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

data['Arrival\_date'].fillna('None',inplace=True)

data['Travel\_Mins'].fillna('None',inplace=True)

data['Travel\_Mins'] = data['Travel\_Mins'].str.rstrip('m') # remove 'm' character from the end of each string

data['Travel\_Mins'] = data['Travel\_Mins'].replace('None', '-1') # replace 'None' values with -1

data['Travel\_Mins'] = data['Travel\_Mins'].astype('int64') # convert to int64 datatype

#data['Total\_Stops'] = data['Total\_Stops'].astype('int64')

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

data['Arrival\_date'] = data['Arrival\_date'].astype('int64')

data['Arrival\_Time\_Mins'] = data['Arrival\_Time\_Mins'].fillna(-1).astype('int64')

#data['Travel\_Hours'] = data['Travel\_Hours'].astype('int64')

data['Travel\_Mins'] = data['Travel\_Mins'].astype('int64')

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

data.drop(index=6474,inplace=True,axis=0)

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

**#Label Encoding**

**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.Cityl=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)**

**#Output Columns**

**data = data[['Airline','Source','Destination','Date','Month','Year','Dep\_Time\_Hour','Dep\_Time\_Mins','Dep\_Time\_Mins','Arrival\_date','Arrival\_Time\_Hour','Arrival\_Time\_Mins','Price']]**

**# Exploratory Data Analysis**

data.describe()

**#Visual Analysis**

import seaborn as sns

import matplotlib.pyplot as plt

# Define categorical variables

categorical = ['Airline', 'Source', 'Destination', 'Additional\_Info', 'City1']

# Create a plot for each categorical variable

c = 1

plt.figure(figsize=(10,20))

for i in data.columns:

    if i in categorical:

        plt.subplot(6,3,c)

        sns.countplot(x=data[i])

        plt.xticks(rotation=90)

        plt.tight\_layout(pad=3.0)

        c=c+1

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

plt.figure(figsize=(15,8))

sns.distplot(data['Price'])

**#Checking the Correlation Using HeatMap**

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

**#Outlier Detection for 'Price' Column**

import seaborn as sns

sns.boxplot(data['Price'])

y = data['Price']

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

x\_scaled = ss.fit\_transform(x)

x\_scaled = pd.DataFrame(x\_scaled)

x\_scaled.head()

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)

**#Using Ensemble Techniques**

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", test\_score)

        print("R2 for train data", train\_score)

        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 Absolute Error is", mean\_squared\_error(y\_pred, y\_test, squared=False))

**#Regression Model**

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)))

**# Checking Cross Validation for RandomForestRegressor**

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())

**#Hypertuning the model**

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)

gb=GradientBoostingRegressor()

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

gb\_res.fit(x\_train,y\_train)

**#Accuracy**

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))

**# Evaluating performance of the model and saving the model**

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))

Price=y\_train\_pred

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

price\_list

**#Model Deployment**

import pickle

pickle.dump(rfr,open('model1.pkl','wb'))