Optimizing Flight Booking Decisions through Machine Learning Price Predictions

1.INTRODUCTION

1.1 Overview

People who work frequently travel 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

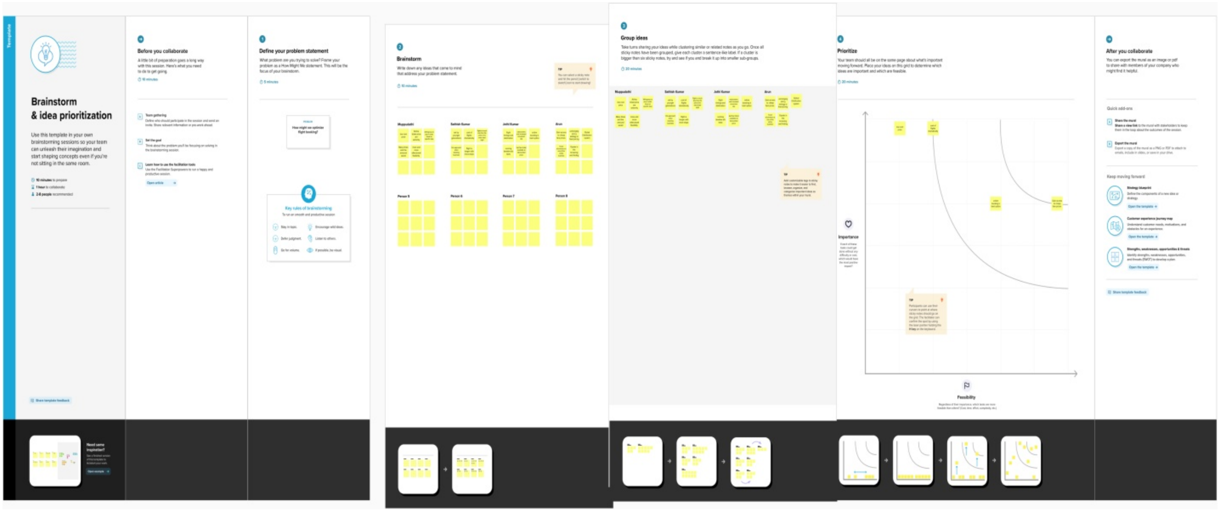
1.2 Purpose

● User interacts with the UI to enter the input. ● Entered input is analysed 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 & DocumentationRecord explanation Video for project end to end solution ○ Project Documentation-Step by step project development procedure

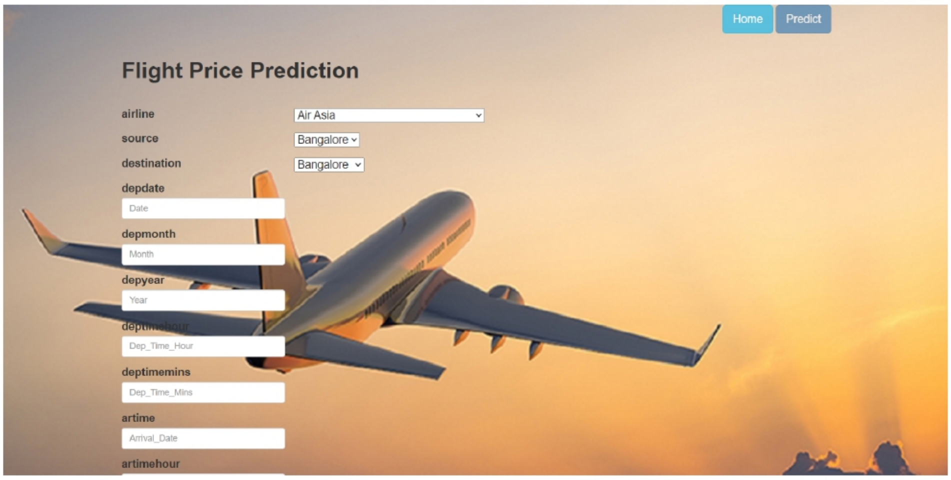
Problem Definition & Design Thinking

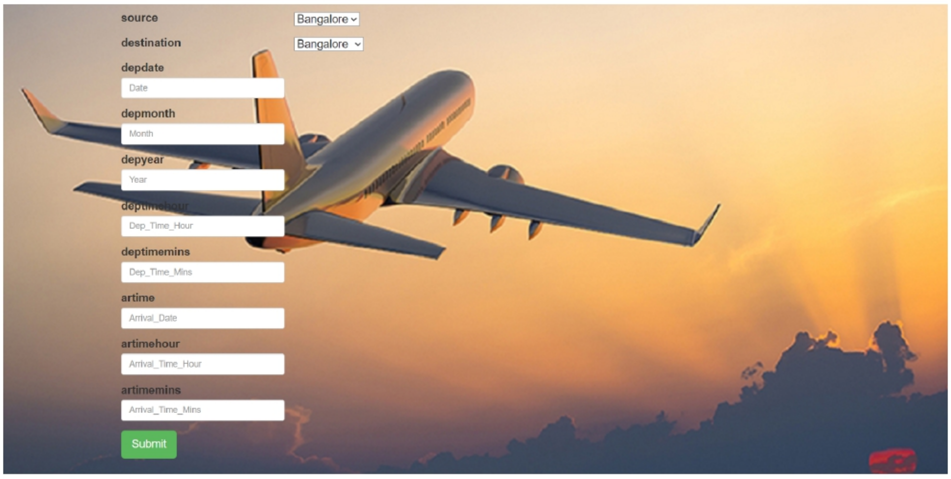
2.1 Empathy Map

2.2 Ideation & Brainstroming Map



3.RESULT





Input 1- Now, the user will give inputs to get the predicted result after clicking onto the submit button.

ADVANTAGES & DISADVANTAGES

It seems that online booking systems are the way of the future.

Does that make the transition to cloud-based tools an inevitable necessity if you want your business to succeed in the future?

Outside of helping you stand out from your competition, what are the benefits of offering online booking options for your players? Do the benefits outweigh any potential hurdles?

Before making your decision, consider the advantages and disadvantages of online

In other words, your customers can make a reservation whenever it fits into their schedule (without you or your employees having to be there). Over $450 billion was spent by consumers online in 2017, and that number will only go up. Additionally, studies have shown that immediate availability when shopping for products or services dramatically increases the number of purchases or appointments.

APPLICATION

Open anaconda prompt from the start menu ● Navigate to the folder where your python script is. ● Now type “python app.py” command ● Navigate to the localhost where you can view your web page. ● Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

CONCLUSION

Machine Learning algorithms are applied on the dataset to predict the dynamic fare of flights.

This gives the predicted values of flight fare to get a flight ticket at minimum cost.

Data is collected from the websites which sell the flight tickets so only limited information can be accessed.

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 have created the entire process of predicting an airline ticket and given a proof of our predictions based on the previous trends with our prediction.

FUTURE SCOPE

To evaluate the conventional algorithm, a dataset is built for route BOMBAY to DELHI and studied a trend of price variation for the period of limited days.

Machine Learning algorithms are

applied on the dataset to predict the dynamic fare of flights.

This gives the predicted values of flight fare to get a flight

ticket at minimum cost.

Data is collected from the websites

which sell the flight tickets so only limited information can be

accessed.

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

APPENDIX

```
import numby 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,
GrandientBoosting
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbours import KNeighboursClassifier
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')
plt.style.use('fivethirrtyright')

data=pd.red_csv)("Data_Train.csv")
data.head()

for i in category:
    print(i, data[i].unique())
    data.Date_of_journey=data.Date_of_journey.str.split('/')
    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.Total_Stops.unique()

data.Route=data.Route.str.split('->')
data.Route

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_date']=data.Arrival_Time.str[1]
data['Time_of_Arrival']=data.Arrival_Time.str[2]
```

```
data['Time_of_Arrival']=data.Time_of_Arrival.str.splite(':')

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.st.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.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()

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

data.isnull().sum()

data.drop(['City4','City5','city6'],axis=1,inplace=True)

data.drop(['Date_of_journey','Route','Dep_Time','Arriva_Time','Duratio
n'],axis=1, inplace=True)
data.drop(['Time_of_Arrival'],axis=1,inplace=True)

data.isnull().sum

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

```
data['Arrival_date'].fillna(data['Date'],inplace=True)

data['Travel_Mins'].fillna(0,inplace=True)

data.info()
data.Date=date.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.Travel_Mins=data.Travel_Mins.astype('int64')

data[data['Travel_Hours']=='5m']
```

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

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

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=data[('Airline','source','Destrination','Date','Month','Year','Dep_Time_Hour','Dep_Time_Min','Arrival_date','Arrival_Time_
```

```
data.head()

#plotting Countplots for Categeorical Data

import seanorn as ns

c=1

plt.figure(figsize=(20,45))


for i in categorical:

    plt.subplot(6,3,c)

    sns.countplot(data[i])

    plt.xticks(rotation=90)

    plt.tight_layout(pad=3.0)

    c=c+1


plt.show()

#Ditribution of 'PRICE' Column

plt.figure(figsize=(15,8)

sns.distplot(data.Price)


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

import seaborn as ns
```

```
sns.boxplot(data['Price'])

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

        from sklearn.preprocessing import StandardsScaler
        ss=StandardScaler()

x_scaled = ss.fit_transform(x)

x_scaled = ss.fit_transform(x)
x_scaled.head()

from sklearn.model_selection import train_test_split
x_train,x
_test,y_train=train_test_split(x,y,test_size=0.2,random_state=42)

x_train.head()
```

```
    from sklearn.ensemble import RandomForestRegressor,
GrandientBoostingRegressor, AdaBoostRegressor

    rft=RandomForestRegressor()

    gb=GrandientBoostingRegressor()

    ad=AdaBoostRegressor()


    from sklearn.metrics import
r2_score,mean_absolte_error,mean_squard_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 Squared Error
is",(mean_squared_error(y_pred,y_test,squared=False)))


from sklearn.neighbours import KNeighboursRegressor

from sklearn.svm import SVR

from sklearn.tree import DecisionTreeRegressor


from sklearn.metrics import
r2_score,mean_absolute_error,mean_squared_error


knn=KNeighboursRegressor()

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.predct(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.prectict(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)))


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 RandomizedSerachCV


param_grid={'n_estimators':[10,30,50,70,100],'max_depth':[None,1,2,3],
```

```
                'max+_features':['auto','sqrt']}
            rfr=RandomForestRegressor()

rf_res=RandomizedSeaerchCV(estimator=rfr,param_distribution=param_grid,cv=3,verbose=2,n_jobs=-1)

            rf_res.fit(x_train,y_train)
           gb=GradientBootingRegresor()

gb_res=RandomizedSearchCV(estimator=gb,param_distribution=param_grid,cv=3,verbose=2,n_jobs=-1)
          gb_res.fit(x_train,y_train)

rfr=RandomForestRegressor(n_estimators=10,max_feature='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_)prend,y_train))
            print("test accuracy",r2_score(y_test_pred,y_test))
```

```
knn=KNeighboursRegressor(n_neighbours=2,algrithm='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,y_test))


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))


        price_list=pd.DataFrame({'Price':prices})
        price_list
```

```
import pickle

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


import pickle

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


from flask import Flask, render_template,request

import numby as np

import pickle


model = pickle.load(open(r"model1.pkl",'rb'))


@app.route("/home")

def home():

    return render_template('home.html')


   @app.route("/predict")

   def home1():

       return render_template('predict.html')
```

```
@app.route("/pred", methods=['POST','GET'])
def predict():
    x=[[int(x)for x in request.form.values()]]
    print(x)

    x = np,array(x)
    print(x.shape)

    print(x)
    pred = model.predict(x)
    print(pred)
    return render_template('submit.html',
prediction_text=pred)

if _ _ name_ _ == "" _ _main_ _":
 app.run(debug=False)
```