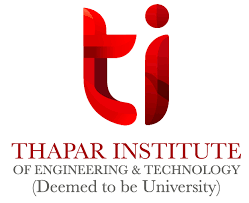
**Flight Price Prediction**



Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

(Deemed to be University), Patiala – 147004

**Machine Learning Project**

**Submitted By**:

Rishi Sharma

102103270

Samridhi Sahu

102283028

Submitted To:

Ms. Kudratdeep Aulakh

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Content used** | **Page No.** |
| 1. | Introduction | 03 |
| 2 | Libraries used | 05 |
| 3. | Algorithm used | 06 |
| 4. | Code and Screenshots | 07 |

1. **Introduction**
   1. **Name of the dataset**
2. Dataset for different flights

<https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction?select=Clean_Dataset.csv>

1. Dataset for business class seats

<https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction?select=business.csv>

1. Dataset for economy class seats

<https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction?select=economy.csv>

* 1. **Description**

A flight price prediction model based on the Random Forest algorithm involves using machine learning techniques to analyse historical flight data and predict future prices. Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve the overall accuracy and robustness of the model.

1. Data Collection:

Gather historical flight data, including features such as departure and arrival locations, dates, times, airline, and any other relevant information.

Include target variable: the actual flight prices.

1. Data Preprocessing:

Clean and preprocess the data by handling missing values, encoding categorical variables, and scaling numerical features.

Split the data into training and testing sets to evaluate the model's performance.

1. Feature Engineering:

Extract relevant features from the data that might influence flight prices, such as day of the week, time of day, and seasonal trends.

1. Model Training:

Use the Random Forest algorithm to train the model on the historical data.

Random Forest builds multiple decision trees during training, where each tree is trained on a random subset of the features and a random subset of the data.

1. Hyperparameter Tuning:

Optimize the performance of the Random Forest model by fine-tuning hyperparameters. This can involve adjusting parameters such as the number of trees in the forest, the maximum depth of each tree, and the minimum number of samples required to split a node.

1. Model Evaluation:

Evaluate the model's performance on the testing set using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or Root Mean Squared Error (RMSE).

1. Prediction:

Once the model is trained and evaluated, it can be used to predict future flight prices based on new input data.

1. Deployment:

Integrate the trained model into a web application, mobile app, or any other platform where users can input their flight details and receive predicted prices.

1. Continuous Improvement:

Regularly update the model with new data to improve its accuracy and adapt to changing trends in flight prices.

1. **Libraries Used**

In Python, libraries are collections of modules or packages containing pre-written code that provides functionalities for a specific purpose. These libraries can save developers time and effort by offering ready-made solutions for common tasks. Here are the Python libraries used in our Project:

1. NumPy:

->Purpose: Numerical computing library.

->Key Features: Provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.

1. Pandas:

->Purpose: Data manipulation and analysis library.

->Key Features: Offers data structures like Data Frame for efficient data manipulation and analysis. Ideal for working with structured data.

1. Matplotlib:

->Purpose: 2D plotting library.

->Key Features: Creates static, animated, and interactive visualizations in Python. Commonly used for creating graphs, charts, and other visual representations of data.

1. Seaborn:

->Purpose: Data visualization library based on Matplotlib.

->Key Features: Simplifies the process of creating attractive and informative statistical graphics. It works well with Pandas data structures.

1. Pickle

->Purpose: Persistence, Data Exchange and Object Serialisation

->Key Features: Useful for saving and loading complex data structures, such as dictionaries, lists, and custom objects, to and from files or transmitting them over the network.

1. **Algorithm Used**

Algorithm used in our project is Random Forest

The Random Forest algorithm is a powerful and versatile machine learning algorithm that belongs to the ensemble learning family. It is widely used for both classification and regression tasks. Here are key points about the Random Forest algorithm:

1. Ensemble Learning:

Random Forest is an ensemble learning method that builds a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

1. Bagging Technique:

It employs a bagging (Bootstrap Aggregating) technique. Each tree in the Random Forest is trained on a random subset of the training data (with replacement), and predictions are combined through averaging (regression) or voting (classification).

1. Random Feature Selection:

Random Forest introduces additional randomness by selecting a random subset of features at each split in the decision tree. This helps decorrelate the trees and improves the overall performance of the model.

1. Reduced Overfitting:

The ensemble nature of Random Forest helps mitigate overfitting, a common issue with individual decision trees. The randomness in feature selection and the combination of multiple trees contribute to a more robust and generalizable model.

1. Handles Missing Values:

Random Forest can handle missing values in the dataset. When making predictions for a sample with a missing value, the algorithm considers all available features for that particular tree.

1. **Code**

#!/usr/bin/env python

# coding: utf-8

# In[ ]:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import pickle as pkl

sns.set()

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

from sklearn.linear\_model import LinearRegression, Ridge, Lasso, ElasticNet

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.pipeline import make\_pipeline

from sklearn.model\_selection import cross\_val\_score, train\_test\_split, KFold, GridSearchCV

from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor

from sklearn import metrics

# In[ ]:

df\_1 = pd.read\_csv("data/PAR\_NYC.csv")

df\_2 = pd.read\_csv("data/PAR\_SVO.csv")

df\_3 = pd.read\_csv("data/SVO\_NYC.csv")

df\_4 = pd.read\_csv("data/SVO\_RUH.csv")

df\_5 = pd.read\_csv("data/NYC\_PAR.csv")

df\_6 = pd.read\_csv("data/NYC\_SVO.csv")

df\_7 = pd.read\_csv("data/RUH\_NYC.csv")

df\_8 = pd.read\_csv("data/RUH\_PAR.csv")

df\_9 = pd.read\_csv("data/RUH\_SVO.csv")

df\_10 = pd.read\_csv("data/SVO\_PAR.csv")

df\_11 = pd.read\_csv("data/PAR\_RUH.csv")

df\_12 = pd.read\_csv("data/NYC\_RUH.csv")

# In[ ]:

print(f"{df\_1['Source'][0]} => {df\_1['Destination'][0]} route has {df\_1.shape[0]} trips")

print(f"{df\_2['Source'][0]} => {df\_2['Destination'][0]} route has {df\_2.shape[0]} trips")

print(f"{df\_3['Source'][0]} => {df\_3['Destination'][0]} route has {df\_3.shape[0]} trips")

print(f"{df\_4['Source'][0]} => {df\_4['Destination'][0]} route has {df\_4.shape[0]} trips")

print(f"{df\_5['Source'][0]} => {df\_5['Destination'][0]} route has {df\_5.shape[0]} trips")

print(f"{df\_6['Source'][0]} => {df\_6['Destination'][0]} route has {df\_6.shape[0]} trips")

print(f"{df\_7['Source'][0]} => {df\_7['Destination'][0]} route has {df\_7.shape[0]} trips")

print(f"{df\_8['Source'][0]} => {df\_8['Destination'][0]} route has {df\_8.shape[0]} trips")

print(f"{df\_9['Source'][0]} => {df\_9['Destination'][0]} route has {df\_9.shape[0]} trips")

print(f"{df\_10['Source'][0]} => {df\_10['Destination'][0]} route has {df\_10.shape[0]} trips")

print(f"{df\_11['Source'][0]} => {df\_11['Destination'][0]} route has {df\_11.shape[0]} trips")

print(f"{df\_12['Source'][0]} => {df\_12['Destination'][0]} route has {df\_12.shape[0]} trips")

# In[ ]:

# convert duration to numerical format in minutes

def clean\_duration(duration):

duration = list(duration)

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

d = []

for i in range(len(duration)):

d.append(duration\_hours[i]\*60+duration\_mins[i])

return d

# convert price to numerical format in USD

def clean\_price(price):

price = price.str.replace(',','',regex=True)

price = price.str.replace('SAR','',regex=True)

price = price.str.strip()

price = round(pd.to\_numeric(price)/3.75,2)

return price

# convert date to datetime format

def clean\_date(date):

date = pd.to\_datetime(date)

return date

# get price quantile to deal with outliers

def get\_price\_quantile(price):

Q1 = price.quantile(0.25)

Q3 = price.quantile(0.75)

IQR = Q3 - Q1

lower\_lim = Q1 - 1.5 \* IQR

upper\_lim = Q3 + 1.5 \* IQR

return (lower\_lim,upper\_lim)

# get average of each airline

def get\_avg\_per\_airline(x):

# average for trips with multiple airlines

multiple\_airlines = x[x["Airline"].str.contains(",")]

b = list(multiple\_airlines["Airline"].str.split(","))

d = [] # Airline 1

e = [] # Airline 2

for i in range(len(b)):

d.append(b[i][0])

e.append(b[i][1])

for i in range(len(e)):

e[i] = e[i].strip()

m\_airlines = list(set(d)) + list(set(e))

column\_names = ["Airline", "Average Price"]

t\_ = pd.DataFrame(columns = column\_names)

for airline in m\_airlines:

t = pd.DataFrame(x[x["Airline"].str.contains(airline)]["Airline"])

t["Average Price"] = x[x["Airline"].str.contains(airline)]["Price"].mean()

t\_ = t\_.append(t)

t\_\_ = t\_.groupby("Airline",as\_index = False)["Average Price"].mean()

k = multiple\_airlines.copy()

k = k.merge(t\_\_, on = "Airline", how = "left")

# average for trips with single airlines

single\_airlines = x[~x["Airline"].str.contains(",")]

avg\_per\_airline = single\_airlines.groupby("Airline", as\_index = False)["Price"].mean()

avg\_per\_airline = avg\_per\_airline.rename(columns={"Price" : "Average Price"})

temp = single\_airlines.copy()

temp = temp.merge(avg\_per\_airline, on='Airline', how ="left")

temp\_1 = temp.groupby("Airline", as\_index = False)["Average Price"].mean()

k\_1 = k.groupby("Airline", as\_index = False)["Average Price"].mean()

k\_temp = pd.concat([k\_1,temp\_1])

y = x.merge(k\_temp, on = "Airline")

return y

# In[ ]:

dfs\_raw = [df\_1,df\_2,df\_3,df\_4,df\_5,df\_6,df\_7,df\_8,df\_9,df\_10,df\_11,df\_12]

# In[ ]:

# cleaning the data

dfs = []

for df in dfs\_raw:

df.drop\_duplicates() # drop duplicate rows

df["Duration"] = clean\_duration(df["Duration"]) # convert duration to numerical minutes format

df["Price"] = clean\_price(df["Price"]) # convert price to numerical format in USD

df["Date"] = clean\_date(df["Date"]) # convert date to datetime format

dfs.append(get\_avg\_per\_airline(df)) # get average per airline

# In[ ]:

# boxplots for each route

k=0

figure, axis = plt.subplots(4, 3, figsize=(15,15))

for i in range(4):

for j in range(3):

axis[i,j].boxplot(dfs[k]['Price'])

axis[i,j].set\_title(f"{dfs[k]['Source'][0]} TO {dfs[k]['Destination'][0]}")

k += 1

# In[ ]:

# get quantile to deal with outliers

lower = []

upper = []

for df in dfs:

x = get\_price\_quantile(df['Price'])

lower.append(x[0])

upper.append(x[1])

# In[ ]:

# drop outliers

k = 0

for df in dfs:

low = df['Price'] < lower[k]

up = df['Price'] > upper[k]

df['Price'] = df['Price'][~(low|up)]

df.dropna(inplace=True)

df.reset\_index(drop = True,inplace=True)

k+=1

# In[ ]:

# boxplot for each route after dealing with outliers

k=0

figure, axis = plt.subplots(4, 3, figsize=(15,15))

for i in range(4):

for j in range(3):

axis[i,j].boxplot(dfs[k]['Price'])

axis[i,j].set\_title(f"{dfs[k]['Source'][0]} TO {dfs[k]['Destination'][0]}")

k += 1

# In[ ]:

# concat all routes into one dataframe

df = pd.concat(dfs)

df

# In[ ]:

# check for null values

df.isnull().sum()

# In[ ]:

# the Airline column will be replaced by the average price per airline.

df.drop("Airline", axis = 1, inplace = True)

# In[ ]:

# source

df["Source"].value\_counts()

# In[ ]:

# source vs price

sns.catplot(y = "Price", x= "Source", data = df.sort\_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)

# In[ ]:

# performing OneHotEncoding on Source since it's nominal categorical data

source =df[["Source"]]

source =pd.get\_dummies(source, drop\_first=True)

source.head()

# In[ ]:

# destination

df["Destination"].value\_counts()

# In[ ]:

# destination vs price

sns.catplot(y = "Price", x= "Destination", data = df.sort\_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)

# In[ ]:

# performing OneHotEncoding on Destination since it's nominal categorical data

destination = df[["Destination"]]

destination = pd.get\_dummies(destination, drop\_first=True)

destination.head()

# In[ ]:

# total stops

print(df["Total stops"].value\_counts())

df["Total stops"].unique()

# In[ ]:

# performing LabelEncoder on Total stops since it's ordinal categorical data

df.replace({"nonstop ":0, "1 stop ": 1, "2 stops ": 2, "3 stops ":3}, inplace=True)

# In[ ]:

final\_df = pd.concat([df,source,destination], axis=1).reset\_index(drop = True)

# In[ ]:

final\_df

# In[ ]:

# drop date since it'll not be used as a feature

final\_df.drop(["Source","Destination","Date"], axis=1, inplace=True)

# In[ ]:

final\_df

# In[ ]:

final\_df.shape

# In[ ]:

final\_df.isnull().sum()

# In[ ]:

final\_df.columns

# In[ ]:

X = final\_df[['Duration', 'Total stops', 'Average Price', 'Source\_PAR',

'Source\_RUH', 'Source\_SVO', 'Destination\_PAR', 'Destination\_RUH',

'Destination\_SVO']]

y = final\_df["Price"]

# In[ ]:

plt.figure(figsize = (18,18))

sns.heatmap(final\_df.corr(),annot= True, cmap = "coolwarm")

plt.show()

# In[ ]:

# getting feature importance to the target variable "Price".

selection =ExtraTreesRegressor()

selection.fit(X,y)

selection.feature\_importances\_

# In[ ]:

# plotting graph of important features

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

feat\_importances = pd.Series(selection.feature\_importances\_,index = X.columns)

feat\_importances.nlargest(20).plot(kind="barh")

plt.show()

# In[ ]:

# 60% Train - 20% Val - 20% Test

X\_train\_or, X\_test, y\_train\_or, y\_test = train\_test\_split(X, y, test\_size=0.2)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train\_or, y\_train\_or, test\_size=0.25)

# In[ ]:

def get\_metrics(model):

print(f'Train score {model.score(X\_train, y\_train)}')

print(f'Val score {model.score(X\_val, y\_val)}')

print("MAE:" , metrics.mean\_absolute\_error(y\_val,model.predict(X\_val)))

print("MSE:" , metrics.mean\_squared\_error(y\_val,model.predict(X\_val)))

print("RMSE:" , np.sqrt(metrics.mean\_squared\_error(y\_val,model.predict(X\_val))))

# In[ ]:

lr = LinearRegression()

lr.fit(X\_train, y\_train)

score = lr.score(X\_val, y\_val)

get\_metrics(lr)

# In[ ]:

for degree in [1,2,3,4,5]:

poly = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

poly.fit(X\_train, y\_train)

print("-"\*20)

print("Degree", degree)

get\_metrics(poly)

# In[ ]:

lasso\_model = Lasso()

lasso\_model.fit(X\_train, y\_train)

get\_metrics(lasso\_model)

# In[ ]:

ridge\_model = Ridge()

ridge\_model.fit(X\_train, y\_train)

get\_metrics(ridge\_model)

# In[ ]:

EN\_model = ElasticNet(alpha=1)

EN\_model.fit(X\_train, y\_train)

EN\_model.score(X\_val, y\_val)

get\_metrics(EN\_model)

# In[ ]:

rf = RandomForestRegressor()

rf.fit(X\_train,y\_train)

get\_metrics(rf)

# In[ ]:

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train.values)

X\_val\_scaled = scaler.transform(X\_val.values)

X\_test\_scaled = scaler.transform(X\_test.values)

# In[ ]:

# function to get metrics for scaled features

def scaled\_metrics(model):

print(f'Train score {model.score(X\_train\_scaled, y\_train)}')

print(f'Val score {model.score(X\_val\_scaled, y\_val)}')

print("MAE:" , metrics.mean\_absolute\_error(y\_val,model.predict(X\_val\_scaled)))

print("MSE:" , metrics.mean\_squared\_error(y\_val,model.predict(X\_val\_scaled)))

print("RMSE:" , np.sqrt(metrics.mean\_squared\_error(y\_val,model.predict(X\_val\_scaled))))

## Baseline: Linear Regression

lr = LinearRegression()

lr.fit(X\_train\_scaled, y\_train)

score = lr.score(X\_val\_scaled, y\_val)

print("LR")

scaled\_metrics(lr)

print("-"\*50)

## Polynomial

for degree in [1,2,3,4,5]:

poly = make\_pipeline(PolynomialFeatures(degree), LinearRegression())

poly.fit(X\_train, y\_train)

print("Polynomial - Degree", degree)

scaled\_metrics(poly)

print("-"\*50)

## Lasso

lasso\_model = Lasso()

lasso\_model.fit(X\_train\_scaled, y\_train)

print("Lasso")

scaled\_metrics(lasso\_model)

print("-"\*50)

## Ridge

ridge\_model = Ridge()

ridge\_model.fit(X\_train\_scaled, y\_train)

print("Ridge")

scaled\_metrics(ridge\_model)

print("-"\*50)

## ElasticNet

EN\_model = ElasticNet(alpha=1)

EN\_model.fit(X\_train\_scaled, y\_train)

EN\_model.score(X\_val\_scaled, y\_val)

print("ElasticNet")

scaled\_metrics(EN\_model)

print("-"\*50)

## Random Forest

rf = RandomForestRegressor()

rf.fit(X\_train,y\_train)

print("Random Forest")

scaled\_metrics(rf)

# In[ ]:

# retraining the random forest model on train + val, and scoring on test

X\_train\_val = pd.concat([X\_train,X\_val])

y\_train\_val = pd.concat([y\_train,y\_val])

rf = RandomForestRegressor()

rf.fit(X\_train\_val,y\_train\_val)

print(f'Train score {rf.score(X\_train\_val, y\_train\_val)}')

print(f'Test score {rf.score(X\_test, y\_test)}')

print("MAE:" , metrics.mean\_absolute\_error(y\_test,rf.predict(X\_test)))

print("MSE:" , metrics.mean\_squared\_error(y\_test,rf.predict(X\_test)))

print("RMSE:" , np.sqrt(metrics.mean\_squared\_error(y\_test,rf.predict(X\_test))))

# In[ ]:

y\_train\_val\_pred = rf.predict(X\_train\_val)

y\_test\_pred = rf.predict(X\_test)

# In[ ]:

plt.scatter(y\_test,y\_test\_pred,alpha =0.2,color="DarkBlue")

plt.title('Actual vs. Predicted Airline Prices')

plt.xlabel('Predicted Airline Prices')

plt.ylabel('Actual Airline Prices')

# In[ ]:

# randomized search CV

n\_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]

max\_features = ['auto', 'sqrt']

max\_depth = [int(x) for x in np.linspace(5, 30, num = 6)]

min\_samples\_split = [2, 5, 10, 15, 100]

min\_samples\_leaf = [1, 2, 5, 10]

# In[ ]:

# create the random grid

random\_grid = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf}

# In[ ]:

rf\_random = RandomizedSearchCV(estimator = rf, param\_distributions = random\_grid,scoring='neg\_mean\_squared\_error', n\_iter = 10, cv = 5, verbose=2, n\_jobs = 1)

# In[ ]:

rf\_random.fit(X\_train\_val,y\_train\_val)

# In[ ]:

rf\_random.best\_params\_

# In[ ]:

prediction = rf\_random.predict(X\_test)

# In[ ]:

plt.scatter(y\_test,prediction,alpha =0.2,color="DarkBlue")

plt.title('Actual vs. Predicted Airline Prices')

plt.xlabel('Predicted Airline Prices')

plt.ylabel('Actual Airline Prices')

# In[ ]:

print("MAE:" , metrics.mean\_absolute\_error(y\_test,prediction))

print("MSE:" , metrics.mean\_squared\_error(y\_test,prediction))

print("RMSE:" , np.sqrt(metrics.mean\_squared\_error(y\_test,prediction)))

# In[ ]:

test\_df = pd.DataFrame({

"Predicted Price" : rf.predict(X\_test),

"Actual Price" : y\_test,

}).reset\_index(drop = True)

test\_df

# In[ ]:

# save the model

file = open('rf\_flight\_prediction.pkl', 'wb')

pkl.dump(rf, file)

# In[ ]:

# open the model

model = open('rf\_flight\_prediction.pkl','rb')

rf\_flight\_prediction = pkl.load(model)

# In[ ]:

print(f'R2 score {metrics.r2\_score(y\_test,rf\_flight\_prediction.predict(X\_test))}')

print("MAE:" , metrics.mean\_absolute\_error(y\_test,rf\_flight\_prediction.predict(X\_test)))

print("MSE:" , metrics.mean\_squared\_error(y\_test,rf\_flight\_prediction.predict(X\_test)))

print("RMSE:" , np.sqrt(metrics.mean\_squared\_error(y\_test,rf\_flight\_prediction.predict(X\_test))))