Machine Learning and Data Mining

Project Report

Flight Fare Prediction

Team Members:

1. Sidhant Moza (23303026)
2. Ehtesham Ashraf (23003021)
3. Abdul Qadir (23303017)
4. Himani Agrawal (23303025)

Table Of Contents

Abstract

In the contemporary era, the dynamic nature of airline ticket prices presents a challenge for travelers seeking cost-effective and convenient flight options. The "Flight Fare Prediction Model" addresses this challenge through advanced data science techniques, leveraging historical flight data, weather conditions, and other parameters. This initiative aims to provide accurate fare predictions, empowering travelers to plan economically and potentially reducing the carbon footprint of air travel. The report reviews existing literature on flight fare prediction, outlining various methodologies and their outcomes. The comprehensive methodology involves a Kaggle dataset, Python programming, and machine learning algorithms, evaluating multiple regression models for precise price prediction. Rigorous testing, hyperparameter tuning, and deployment as a user-friendly web application are part of the development journey. The report highlights key components, methodologies, and insights, offering a foundation for future research in dynamic pricing and demand prediction in the airline industry.

Introduction

In an age where travel has become an integral part of our lives, finding the most cost-effective and convenient flight options is of paramount importance. The airline industry is characterized by its dynamic nature, with flight ticket prices fluctuating constantly due to a multitude of factors, such as demand, fuel costs, seasonal variations, and competition among airlines. This unpredictability often leaves travelers uncertain about when and where to book their flights to get the best deals.

The "Flight Fare Prediction Model" is a machine learning project aimed at addressing this challenge by leveraging advanced data science techniques to predict and optimize flight ticket prices. By utilizing historical flight data, weather conditions, booking trends, and other relevant parameters, our model will provide accurate fare predictions, allowing travelers to plan their journeys more efficiently and economically. Additionally, it can potentially contribute to reducing the carbon footprint of air travel by enabling travelers to choose the most environmentally friendly options through efficient flight routing.

The journey to develop this Flight Fare Prediction Model encompasses various stages of machine learning, data preprocessing, feature engineering, model selection, and evaluation. It requires harnessing the power of modern technology and data analytics to create a user-friendly and robust platform that can cater to a diverse range of travelers and their specific needs. In this synopsis, we will outline the key components and methodologies that will be employed throughout the project, including the data sources, machine learning algorithms, and evaluation metrics.

Problem Statement

1. **Budget Management :** Many potential travelers experience significant difficulty in managing their travel budgets due to the high variability in airline ticket prices. Other sections of their budget may get affected due to this.

2. **Optimal booking time :** Travelers often face the inability to determine the most cost-effective time to book flights, due to the highly dynamic nature of flight fares, and end up buying a fairly expensive flight.

3. **Lack of Price Transparency :** Airlines use complex pricing algorithms that can lead to immense fluctuation in air fares with no seeming logic to consumers. This lack of price transparency can lead to confusion and a sentiment of being exploited.

4. **Fixed Date Booking :** Individuals with inflexible vacation or travel dates often bear the brunt of peak airfares.

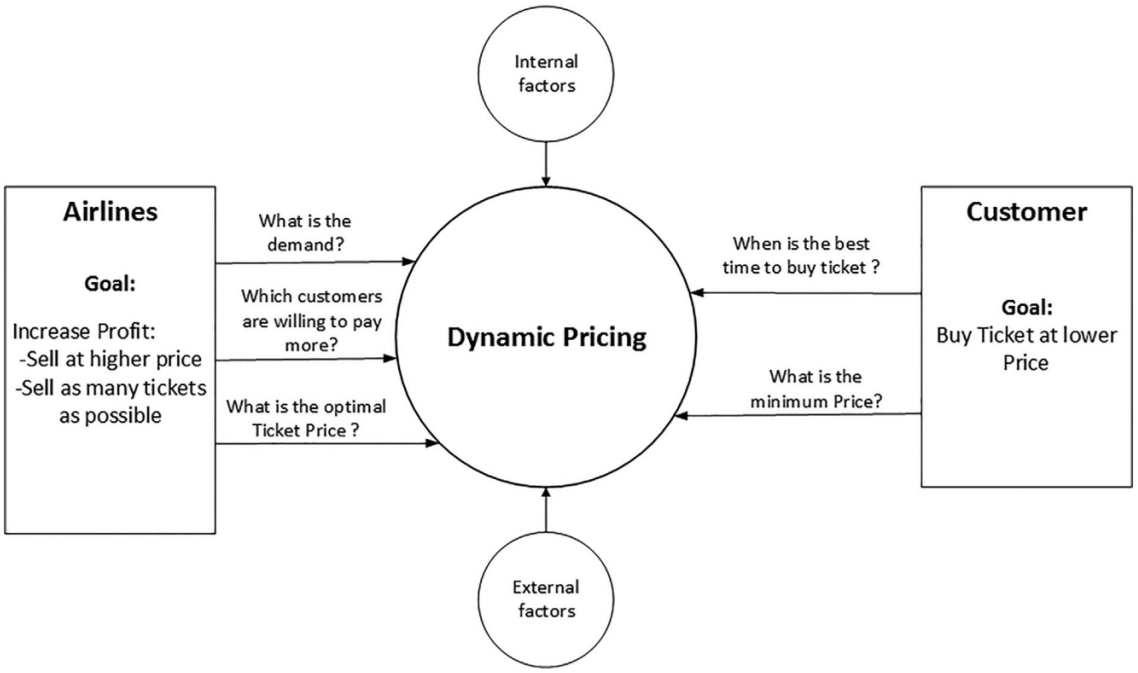
Related Literature

Summary

Ref.	Addressed problem	Method	Performance result
Tziridis et. al. [1]	Predict flight prices considering various input features	Using multiple regression models with 10-fold CV using GridSearch	Bagging Regression Tree gave best accuracy results among all models
William et. al. [2]	Predict the minimum ticket price of all available flights	Addition of lagged features and PLS regression to create 2 proposed models	Performance of proposed models are optimum in comparison to existing models.
Wohlfarth et. al. [3]	Predict the best time to buy tickets	CART and RF	CART and RF are recommended for pre-registered purchase periods to provide initial, general guidance to the customer.
Wang et. al. [4]	Market segment level ticket price prediction	SVM, XGBoost, and RF	XGBoost performed better than other methods
Madaan et. al. [5]	Predict flight prices for different flights	RF along with other algorithms	RF performed best and model was deployed as web app
Kimbhahune et. al. [7]	Predict flight prices for different flights	Linear Regression, Decision Tree, Random Forest	Final Model is deployed as Front-End Application
Zhichao et. al. [8]	Predict flight prices for different flights	Multi-attribute dual-stage attention (MADA)	MADA performed much better than other deep-learning methods such as CNN and LSTM
Pavithra et. al. [9]	Predict flight prices for different no. of days from departure.	RF and Decison Tree	Decision tree shows the best accuracy of 80% for predicting the flight price.
Prithviraj et. al. [11]	Predict cost of flight ticket	XGBoost, RF and Decision Tree	Decision Tree had the highest training accuracy, while RF had the highest testing accuracy.
Supriya et. al. [12]	Predict flight price from source to destination	RF, MLP, Gradient Boosting, Decision Tree, KNN, SVM, AdaBoost	RF & MLP had better results than other algorithms

Objectives

- From the customer's point of view, determining the minimum price or the best time to buy a ticket is the key issue.
- A significant number of research studies have proposed prediction models for dynamic pricing in airlines, which can be classified into two groups: demand prediction and price discrimination [6]



[6]

Methodology

Our project works on a comprehensive dataset of Flight Fare information, which we have sourced from Kaggle. This dataset contains crucial attributes, which include- the airline responsible for the flight, the origin city of the journey, the destination city, departure time, arrival time, flight duration, class of travel, and other pertinent details. The project code is developed in Python programming language using Jupyter Notebook and Google Colab.

Tools/Libraries Used:

- Python for programming and data analysis using Jupyter Notebook and Google Colab.
- Pandas and NumPy for data manipulation.
- Matplotlib and Seaborn for data visualization.
- Decision Trees Regressor and K Neighbor Regressor as learning algorithms

Our approach to this project involves a rigorous and systematic methodology. We will begin by meticulously preparing and preprocessing the dataset to ensure its quality and consistency. This crucial step will involve handling missing data, encoding categorical variables, and standardizing numerical features to make them suitable for machine learning algorithms. We would also perform exploratory data analysis on the dataset to identify the hidden patterns that affect the flight fare and the relationships between different attributes in the dataset.

Techniques Used:

Subsequently, we will start on the model development phase, where we will explore and evaluate multiple machine learning algorithms to ascertain their efficacy in predicting flight fares accurately. The algorithms under consideration encompass a spectrum of techniques, each with its own unique strengths and capabilities. These include the **k-Nearest Neighbors (kNN) algorithm and the Decision Tree Regressor.** Collecting from most of the research papers, the best regression model is identified as Bagging Regression Tree. So the Decision tree and the KNN regressor are selected as the learning algorithms for the model.

Our project will not be limited to merely applying these algorithms; instead, we will conduct extensive training and testing iterations to fine-tune their hyperparameters and optimize their performance. Following that, we will employ robust evaluation metrics to objectively compare the performance of these models, such as **Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R2_score and Mean Absolute Percentage Error (MAPE)**. The model with the best testing results, i.e., the model with the highest overall accuracy, can be selected for deployment as a web application to create a user-friendly platform that enables travelers to access data-driven flight fare predictions. The deployment of web applications is a followup and addition exercise outside of this scope.

Experimental Results

a) Results of EDA with observations

The dataset contains 45,000 records and 19 feature attributes with below details about each attribute -

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45000 entries, 0 to 44999
Data columns (total 19 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Flight_ID           45000 non-null  object
 1   Airline             41427 non-null  object
 2   Departure_City      44660 non-null  object
 3   Arrival_City        44814 non-null  object
 4   Distance            44909 non-null  float64
 5   Departure_Time      45000 non-null  object
 6   Arrival_Time        45000 non-null  object
 7   Duration            45000 non-null  float64
 8   Aircraft_Type       44957 non-null  object
 9   Number_of_Stops     45000 non-null  int64
 10  Day_of_Week         44775 non-null  object
 11  Month_of_Travel     44733 non-null  object
 12  Holiday_Season      45000 non-null  object
 13  Demand              44683 non-null  object
 14  Weather_Conditions  44698 non-null  object
 15  Passenger_Count     45000 non-null  int64
 16  Promotion_Type      44597 non-null  object
 17  Fuel_Price          44910 non-null  float64
 18  Flight_Price        45000 non-null  float64
dtypes: float64(4), int64(2), object(13)
memory usage: 6.5+ MB
```

As we can see, there are very few records with null values, so we fill them wherever feasible in the next few steps.

```
df.isnull().sum()
```

```
Airline                3573
Departure_City          340
Arrival_City            186
Distance                 91
Departure_Time            0
Arrival_Time              0
Duration                  0
Aircraft_Type            43
Number_of_Stops           0
Day_of_Week             225
Month_of_Travel         267
Demand                  317
Weather_Conditions      302
Passenger_Count           0
Promotion_Type          403
Fuel_Price               90
Flight_Price              0
dtype: int64
```

"Airline" and "Promotion_Type" are balanced, while "Demand" is fairly biased towards low values.

```
df['Airline'].value_counts()
```

```
Airline A    13863
Airline C    13809
Airline B    13755
Name: Airline, dtype: int64
```

```
df['Demand'].value_counts()
```

```
Low       28946
Medium     8954
High       6783
Name: Demand, dtype: int64
```

```
df['Promotion_Type'].value_counts()
```

```
Special Offer    14896
Discount         14889
None             14812
Name: Promotion_Type, dtype: int64
```

Similarly, a little bias is in features “Arrival_Time” and "Departure_Time.”

```
df['Arrival_Time'].value_counts()
```

```
Morning       11725
Late Night     9956
Evening        6792
Afternoon      6695
Night          4836
Name: Arrival_Time, dtype: int64
```

```
df['Departure_Time'].value_counts()
```

```
Morning       11692
Late Night    10048
Afternoon      6728
Evening        6542
Night          4994
Name: Departure_Time, dtype: int64
```

b) Results of Pre-Processing with observations

To correct the NaN values, we treat “Demand” with low values and “Promotion_Type” as having no offer. The rest of the NaN records are dropped.

```
df['Demand'] = df['Demand'].fillna('Low')
df['Promotion_Type'] = df['Promotion_Type'].fillna('No Offer')
df['Promotion_Type'] = df['Promotion_Type'].replace("None","No Offer")
```

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

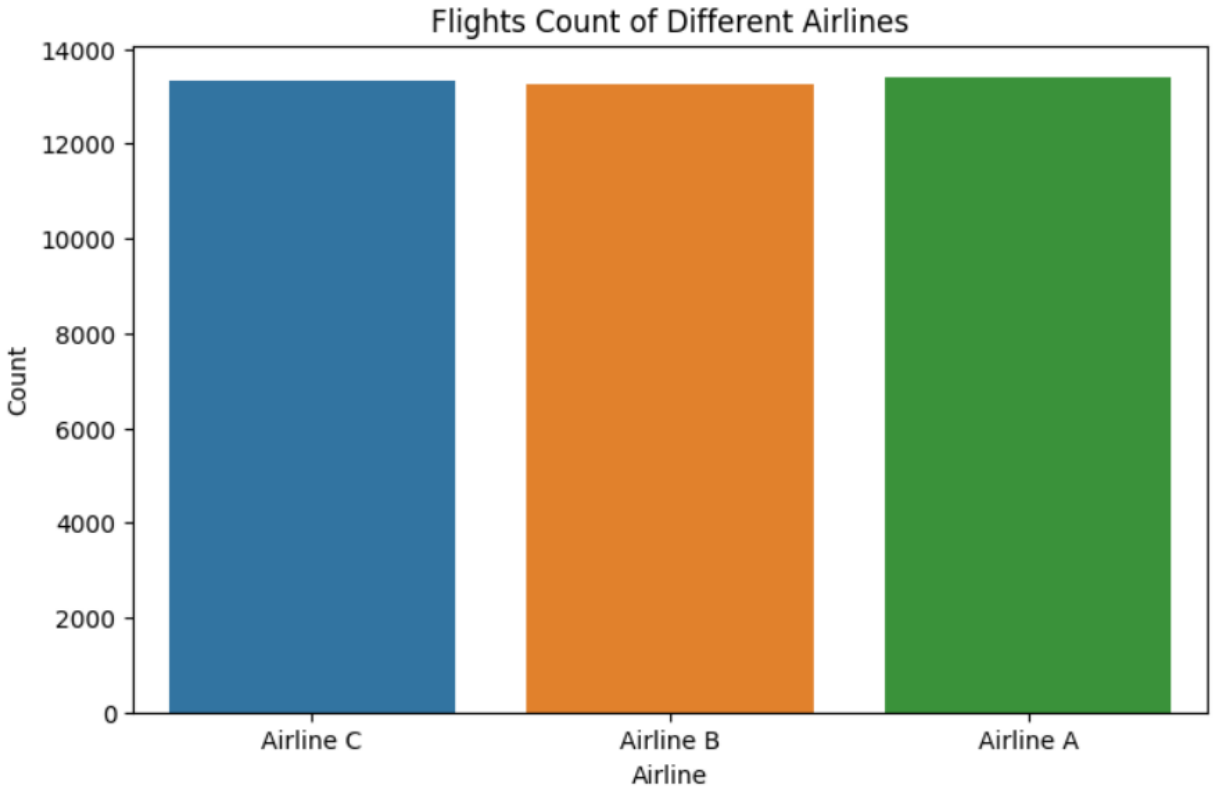
To understand the flight timing patterns, we bucket the timing as if it were morning time, night time, or afternoon time with below-

```
def encodeDepArr(x):
    t = pd.to_datetime(x, format='%H:%M')
    hr = t.hour
    if 5 <= hr < 12:
        return "Morning"
    elif 12 <= hr < 16:
        return "Afternoon"
    elif 16 <= hr < 20:
        return "Evening"
    elif 20 <= hr < 23:
        return "Night"
    else:
        return "Late Night"
df[['Departure_Time', 'Arrival_Time']] = df[['Departure_Time', 'Arrival_Time']].applymap(encodeDepArr)
df
```

	Airline	Departure_City	Arrival_City	Distance	Departure_Time	Arrival_Time	Duration	Aircraft_Type	Number_of_Stops
1	Airline C	Leonardland	New Stephen	2942.0	Night	Late Night	5.29	Airbus A320	0
2	Airline B	South Dylanville	Port Ambermouth	2468.0	Morning	Afternoon	4.41	Boeing 787	1
4	Airline B	Michaelport	Onealborough	5558.0	Night	Morning	8.09	Boeing 737	1
6	Airline B	West Samanthaland	Port Brentport	3274.0	Evening	Night	5.60	Boeing 787	0
8	Airline C	Bryanland	Jessebury	1116.0	Morning	Afternoon	2.46	Airbus A320	0

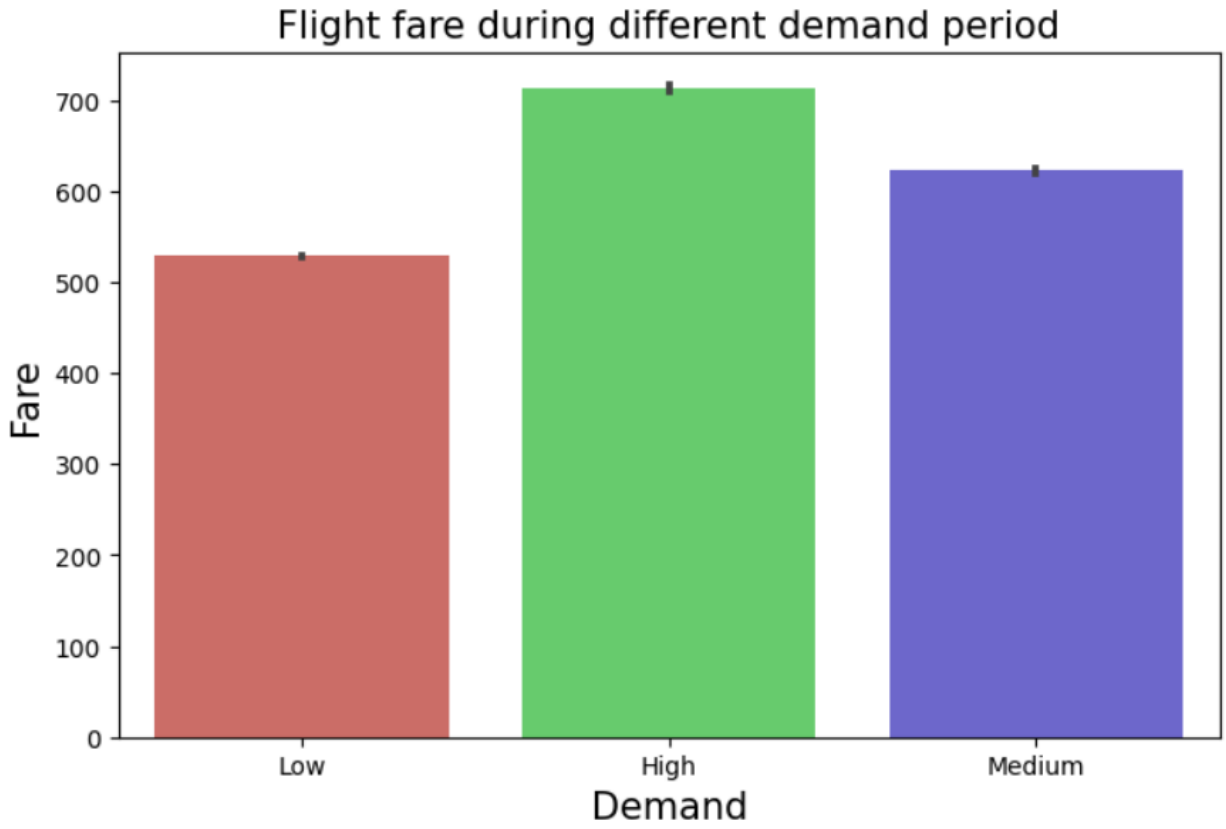
Furthermore, the data is analyzed, and its characteristics are noted below:

```
plt.figure(figsize=(8,5))
sns.countplot(x=df['Airline'],data=df)
plt.title('Flights Count of Different Airlines')
plt.xlabel('Airline')
plt.ylabel('Count')
plt.show()
```



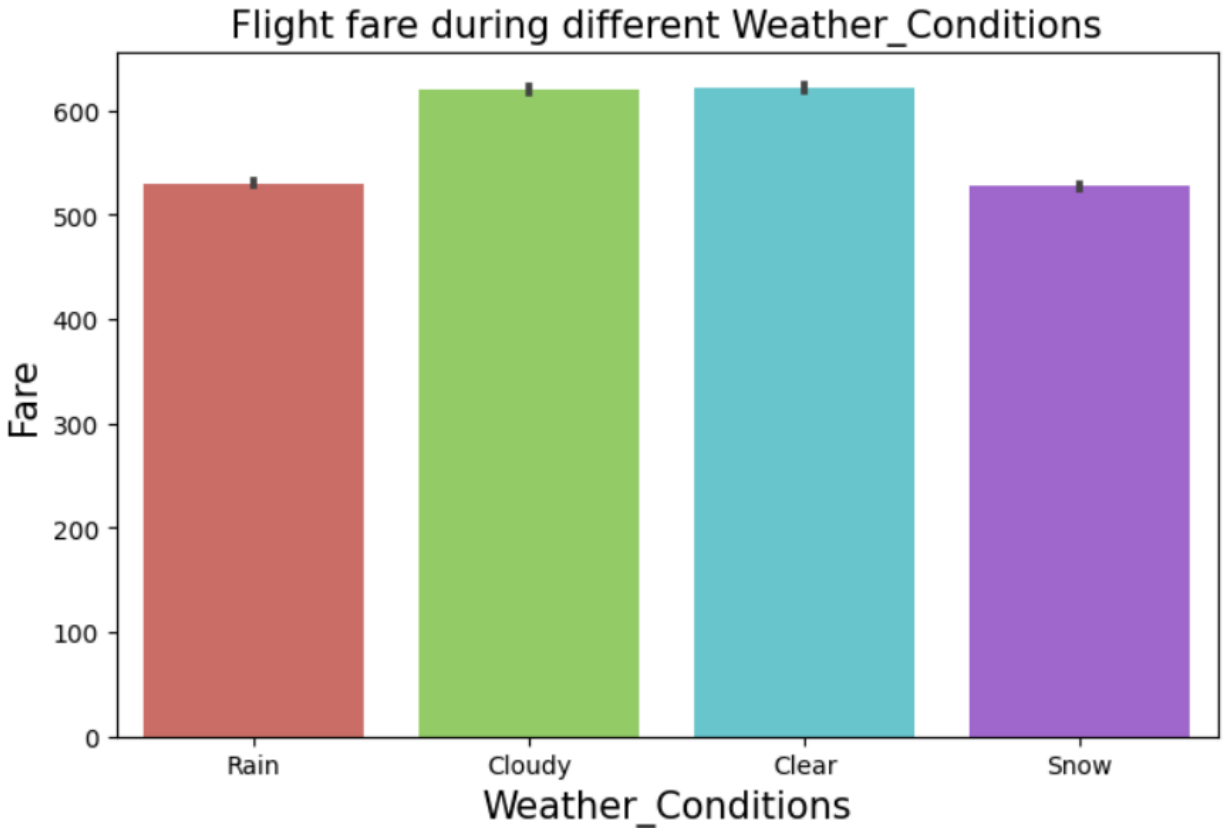
Flights are evenly distributed across dataset; there is no bias

```
plt.figure(figsize=(8,5))
sns.barplot(x=df['Demand'],y=df['Flight_Price'],palette='hls')
plt.title('Flight fare during different demand period',fontsize=15)
plt.xlabel('Demand',fontsize=15)
plt.ylabel('Fare',fontsize=15)
plt.show()
```



A clear trend is seen when the demand is high, the flights also have a high fare.

```
plt.figure(figsize=(8,5))
sns.barplot(x=df['Weather_Conditions'],y=df['Flight_Price'],palette='hls')
plt.title('Flight fare during different Weather_Conditions',fontsize=15)
plt.xlabel('Weather_Conditions',fontsize=15)
plt.ylabel('Fare',fontsize=15)
plt.show()
```



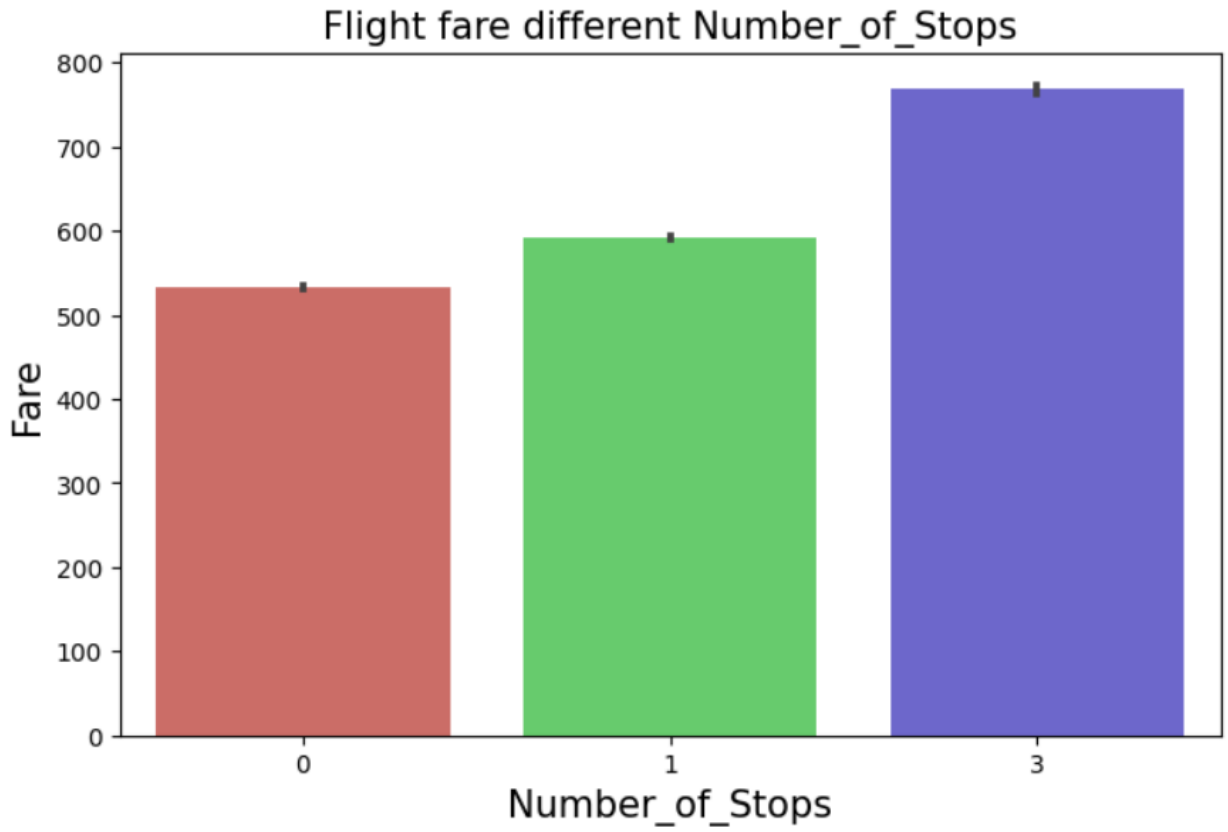
The weather setting suggests rainy and snowy weather tend to have lower fare prices.

```
lst=df["Aircraft_Type"].unique()
for i in range(0,len(lst)):
    l=df[df["Aircraft_Type"]==lst[i]]["Passenger_Count"].mean()
    print(lst[i])
    print(int(l))
```

```
Airbus A320
175
Boeing 787
174
Boeing 737
174
Airbus A380
174
Boeing 777
175
```

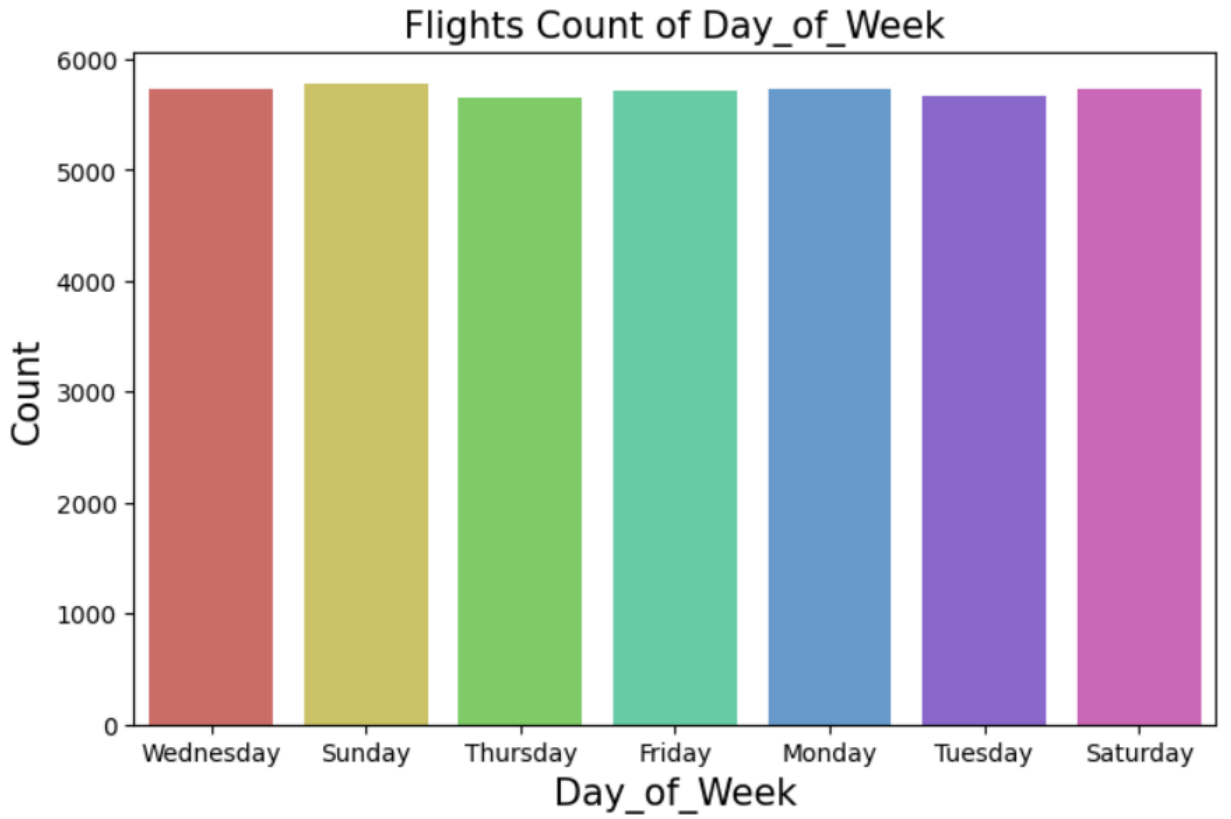
From the Aircraft type we can check the seating capacity of all the planes in record is almost the same.

```
plt.figure(figsize=(8,5))
sns.barplot(x=df['Number_of_Stops'],y=df['Flight_Price'],palette='hls')
plt.title('Flight fare different Number_of_Stops',fontsize=15)
plt.xlabel('Number_of_Stops',fontsize=15)
plt.ylabel('Fare',fontsize=15)
plt.show()
```

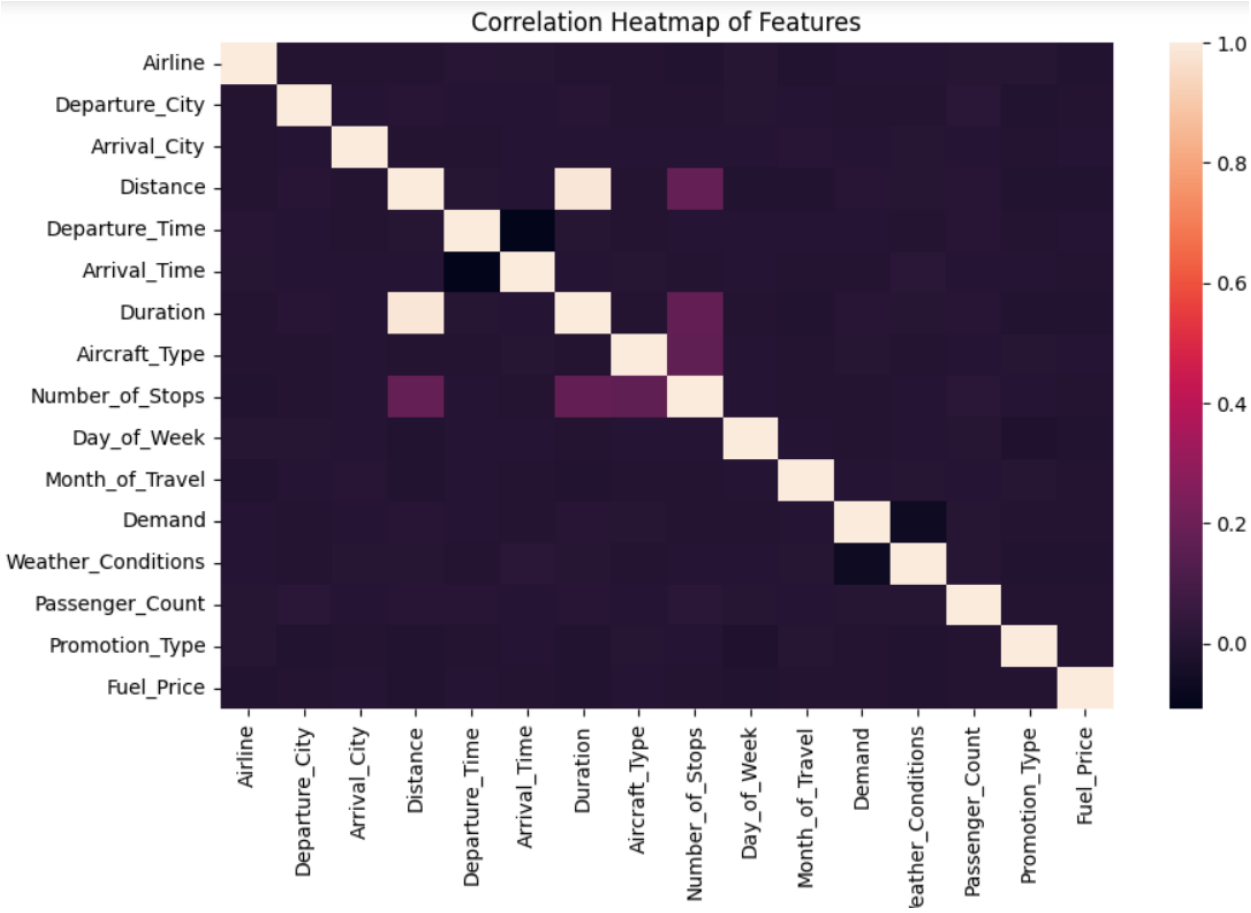


The graph clearly suggests that the greater the number of stops, the higher the flight fares.

```
plt.figure(figsize=(8,5))
sns.countplot(x=df['Day_of_Week'],palette='hls')
plt.title('Flights Count of Day_of_Week',fontsize=15)
plt.xlabel('Day_of_Week',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.show()
```



The data suggests a balanced and consistent number of flights, be it any day of the week.



The data correlation heatmap suggests 4 features having strong correlation interdependence. Hence, we consider applying the PCA technique to select the most effective top features.

Encode the categorical data into numerics.

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for col in df.columns:
    if df[col].dtype=="object":
        df[col]=le.fit_transform(df[col])
```

Train test split with 80/20.

Scale the features set.

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.20,random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
stsc = StandardScaler()
x_train = stsc.fit_transform(x_train)
x_test = stsc.transform(x_test)
```

TRAINING MODELS ON DECISION TREES & KNN REGRESSOR

We get the MAPE and R squared values as below-

```
dtree=DecisionTreeRegressor(random_state=42)
dtree.fit(x_train,y_train)
ypred1=dtree.predict(x_test)
p1=round(dtree.score(x_test,y_test),5)
err1 = mean_absolute_percentage_error(y_test,ypred1)*100
print(f"MAPE for the Decision tree regressor = {err1:.3f}%.")
print(f"R-Squared for the Decision tree regressor = {p1}.")
```

```
MAPE for the Decision tree regressor = 4.466%.
R-Squared for the Decision tree regressor = 0.95652.
```

```
knn=KNeighborsRegressor(n_neighbors=5)
knn.fit(x_train,y_train)
ypred2=knn.predict(x_test)
p2=round(knn.score(x_test,y_test),5)
err2 = mean_absolute_percentage_error(y_test,ypred2)*100
print(f"MAPE for the KNN = {err2:.3f}%.")
print(f"R-Squared for the KNN = {p2}.")
```

```
MAPE for the KNN = 7.351%.
R-Squared for the KNN = 0.90321.
```

We get fairly good performance from these models.
As noticed, there are four features which have strong correlations. Therefore, we apply PCA technique to reduce dimensionality and to check if the performance improves.

```
stsc2 = StandardScaler()
X = stsc2.fit_transform(X)
```

```
principal=PCA(n_components=13)
principal.fit(X)
xpc=principal.transform(X)
```

PCA applied to reduce features down to 13, since there are 4 correlated features similarity in correlation heatmap.

```
x_train2,x_test2,y_train2,y_test2=train_test_split(xpc,y,test_size=0.20,random_state=42)
```

```
dtreepca=DecisionTreeRegressor(random_state=42)
dtreepca.fit(x_train2,y_train2)
yprddt = dtreepca.predict(x_test2)
p1pca=round(dtreepca.score(x_test2,y_test2),3)
mapeer = mean_absolute_percentage_error(y_test2,yprddt)*100
print(f"MAPE for the Decision tree regressor = {mapeer:.3f}%.")
print(f"R-Squared for the Decision tree regressor = {p1pca}.")
```

```
MAPE for the Decision tree regressor = 8.817%.
R-Squared for the Decision tree regressor = 0.854.
```

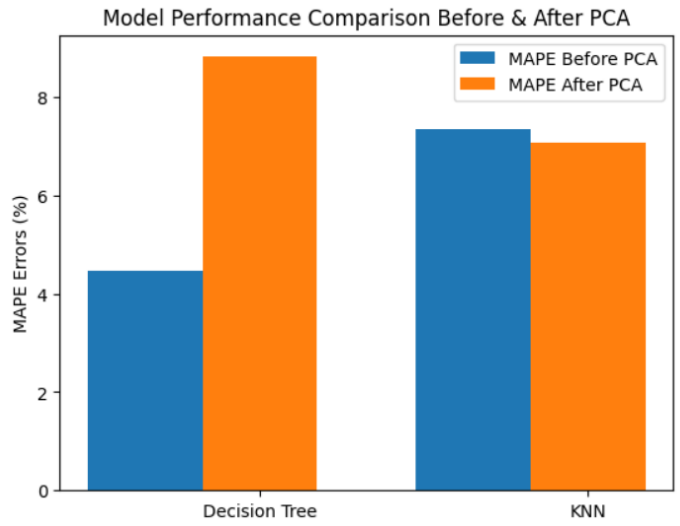
```
knnpca=KNeighborsRegressor(n_neighbors=5)
knnpca.fit(x_train2,y_train2)
yprdknn = knnpca.predict(x_test2)
mapeer2 = mean_absolute_percentage_error(y_test2,yprdknn)*100
p2pca=round(knnpca.score(x_test2,y_test2),5)
print(f"MAPE for the KNN = {mapeer2:.3f}%.")
print(f"R-Squared for the KNN = {p2pca}.")
```

```
MAPE for the KNN = 7.080%.
R-Squared for the KNN = 0.91246.
```

We observed that on applying PCA, the decision tree's R-Squared value reduced by 10%, while the KNN R Squared improved by 1%.

The results obtained before and after applying PCA to both models are visualized in terms of MAPE error, in the bar plot below:

```
indices = np.arange(2)
fig, ax = plt.subplots()
ax.bar(indices, [err1, err2], 0.35, label='MAPE Before PCA')
ax.bar(indices + 0.35, [mapeer, mapeer2], 0.35, label='MAPE After PCA')
ax.set_ylabel('MAPE Errors (%)')
ax.set_title('Model Performance Comparison Before & After PCA')
ax.set_xticks(indices + 0.35)
ax.set_xticklabels(['Decision Tree', 'KNN'])
ax.legend()
plt.show()
```

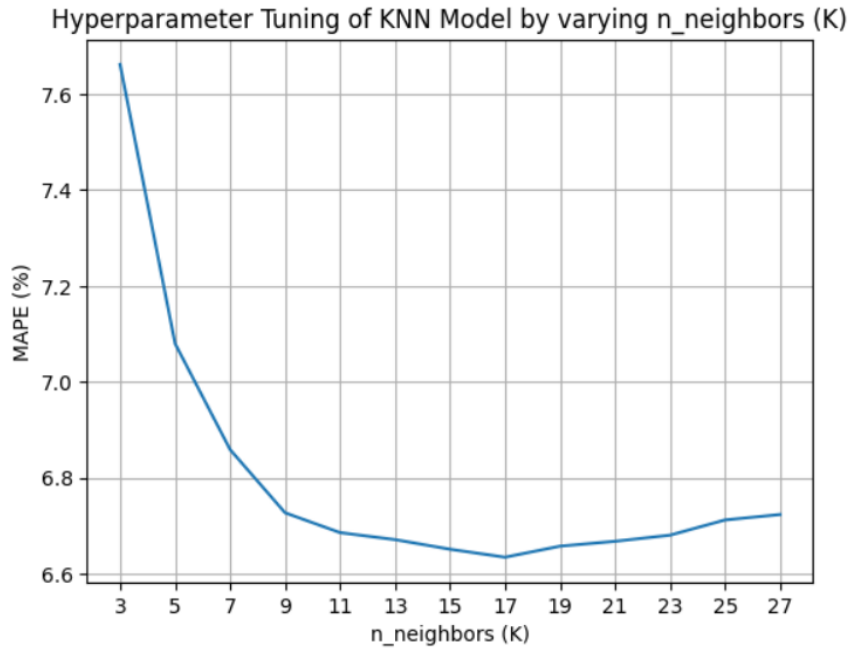


c) Hyperparameter Tuning of Models

[i] Hyperparameter Tuning of KNN Model

To get the optimal K value in KNN, we plotted the MAPE for all values of K from 3 to 29, considering only odd numbers, and we observed that, at K = 17, the algorithm had minimum MAPE error.

```
mape = {}
for k in range(3,29,2):
  knn2=KNeighborsRegressor(n_neighbors=k)
  knn2.fit(x_train2,y_train2)
  ypred3=knn2.predict(x_test2)
  err3 = mean_absolute_percentage_error(y_test2,ypred3)*100
  mape[k] = err3
plt.plot(mape.keys(),mape.values())
plt.title("Hyperparameter Tuning of KNN Model by varying n_neighbors (K)")
plt.xlabel("n_neighbors (K)")
plt.ylabel("MAPE (%)")
plt.xticks(range(3,29,2))
plt.grid()
plt.show()
```



[ii] Hyperparameter Tuning of Decision Tree Model

Since the PCA application on decision trees reduces the model performance, we DO NOT apply PCA to the decision tree regressor and tune the hyperparameters of the original model (without PCA).

To tune the hyperparameters of Decision tree, we use Randomized SearchCV library to obtain the hyperparameters that give overall optimal results of Decision tree regressor-

```
param_grid = {
    'max_depth': [i for i in range(5,35)]+[None],
    'max_features': [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16],
    'min_samples_leaf': [1,2,3,4,5],
    'min_samples_split' : [2,3,4,5,6,7],
    'ccp_alpha' : list(np.arange(0,0.05,0.005))
}
rand_search = RandomizedSearchCV(DecisionTreeRegressor(random_state=42),
                                 param_grid,n_iter=100,n_jobs=-1,verbose=5)
rand_search.fit(x_train,y_train)
print("Best Parameters for DTree :")
print(rand_search.best_estimator_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters for DTree :
DecisionTreeRegressor(ccp_alpha=0.03, max_depth=29, max_features=16,
                      min_samples_leaf=5, min_samples_split=7, random_state=42)
```

Hence, the optimal hyperparameters for decision tree are as follows:-

- ccp_alpha = 0.03,
- max_depth = 29,
- max_features = 16
- min_samples_leaf = 5 ,
- min_samples _split = 7

d) Final results with observations

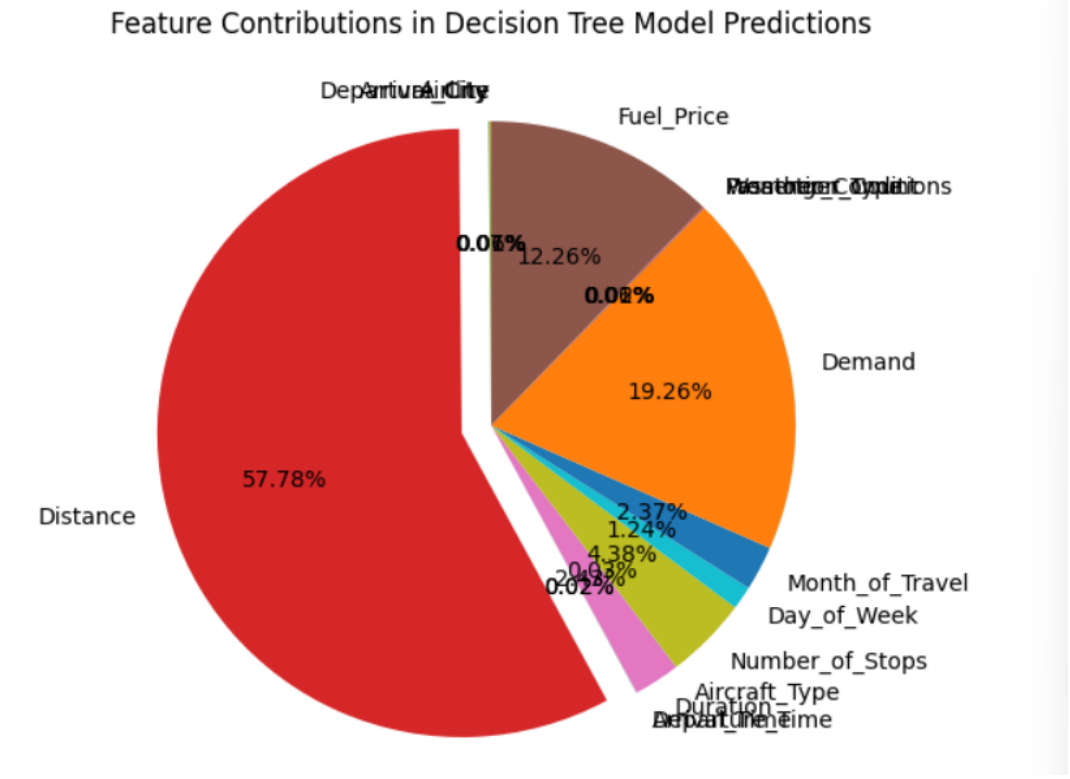
After tuning its hyperparameters, the final model got a MAPE error of 4.12% and R-Squared value of 96.4%

```
fdtree = DecisionTreeRegressor(ccp_alpha=0.03, max_depth=29, max_features=16,
                               min_samples_leaf=5, min_samples_split=7, random_state=42)
fdtree.fit(x_train,y_train)
ypred4=fdtree.predict(x_test)
p4=round(fdtree.score(x_test,y_test),5)
err4 = mean_absolute_percentage_error(y_test,ypred4)*100
print(f"MAPE for the Decision tree regressor = {err4:.3f}%.")
print(f"R-Squared for the Decision tree regressor = {p4}.")
```

```
MAPE for the Decision tree regressor = 4.119%.
R-Squared for the Decision tree regressor = 0.96431.
```

To visualize how each feature contributed towards the results, we plotted the feature importances of the decision tree model as a pie chart, as shown below:

```
fimp = fdtree.feature_importances_
explode = [0.1 if f == max(fimp) else 0 for f in fimp]
plt.figure(figsize=(6,6))
plt.pie(fimp,labels=df.columns[:-1],autopct='%.2f%%',startangle=90,explode=explode)
plt.title("Feature Contributions in Decision Tree Model Predictions")
plt.tight_layout()
plt.show()
```



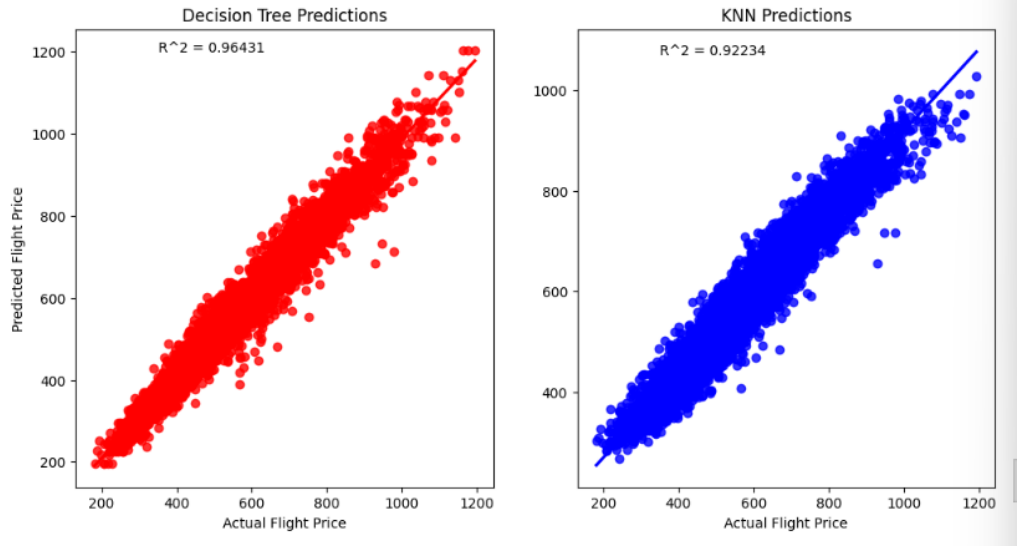
In regards to the KNN model, after applying PCA and hyperparameter tuning on it, the optimal model with K=17 and with PCA applied gives best value as below-

```
fknn=KNeighborsRegressor(n_neighbors=17)
fknn.fit(x_train2,y_train2)
ypred5 = fknn.predict(x_test2)
err5 = mean_absolute_percentage_error(y_test2,ypred5)*100
sc=round(fknn.score(x_test2,y_test2),5)
print(f"MAPE for the Decision tree regressor = {err5:.3f}%.")
print(f"R-Squared for the KNN = {sc}.")
```

```
MAPE for the Decision tree regressor = 6.634%.
R-Squared for the KNN = 0.92234.
```

A regression graph was also plotted for both models to visualize their final performances in predicting the flight prices.

```
plt.figure(figsize=(12,6))
plt.subplot(121)
plt.title("Decision Tree Predictions")
plt.ylabel("Predicted Flight Price")
sns.regplot(x=y_test,y=ypred4,color='red')
plt.text(350,1200,f"R^2 = {p4}")
plt.xlabel("Actual Flight Price")
plt.subplot(122)
sns.regplot(x=y_test2,y=ypred5,color='blue')
plt.text(350,1070,f"R^2 = {sc}")
plt.title("KNN Predictions")
plt.xlabel("Actual Flight Price")
plt.show()
```



Hence, decision tree predictions are better than KNN

We observed that it is not necessary that PCA always improves model performance. For KNN, PCA gives better model performance, while Decision tree's performance degrades with PCA application.

The regression plots of both models show a fairly good performance. The narrower distribution of points and higher R-Squared value in case of decision tree indicate that it performed better than KNN.

Finally, the performance of both models is summarized using accuracy and error metrics, as shown below. The unit of flight price in the case of MAE, MSE, and RMSE is in dollars. Since the values of all the evaluation metrics are better in case of Decision Tree Regressor, it is found to be performing better than KNN, in this problem.

```
print("Performance of the DecisionTree Regressor:")
print('Accuracy (R^2):',round(p4*100,3),'%')
print('Mean Absolute Percentage Error (MAPE):', round(mean_absolute_percentage_error(y_test, ypred4)*100,3),"%")
print('Mean Absolute Error (MAE): $', round(mean_absolute_error(y_test, ypred4),2))
print('Mean Squared Error (MSE): $', round(mean_squared_error(y_test, ypred4),2))
print('Root Mean Squared Error (RMSE): $', round(np.sqrt(mean_squared_error(y_test, ypred4)),2))

print()
print("Performance of the KNN Regressor:")
print('Accuracy (R^2):',round(sc*100,3),'%')
print('Mean Absolute Percentage Error (MAPE):', round(mean_absolute_percentage_error(y_test2, ypred5)*100,3),"%")
print('Mean Absolute Error (MAE): $', round(mean_absolute_error(y_test2, ypred5),2))
print('Mean Squared Error (MSE): $', round(mean_squared_error(y_test2, ypred5),2))
print('Root Mean Squared Error (RMSE): $', round(np.sqrt(mean_squared_error(y_test2, ypred5)),2))
```

```
Performance of the DecisionTree Regressor:
Accuracy (R^2): 96.431 %
Mean Absolute Percentage Error (MAPE): 4.119 %
Mean Absolute Error (MAE): $ 22.81
Mean Squared Error (MSE): $ 936.45
Root Mean Squared Error (RMSE): $ 30.6

Performance of the KNN Regressor:
Accuracy (R^2): 92.234 %
Mean Absolute Percentage Error (MAPE): 6.634 %
Mean Absolute Error (MAE): $ 34.93
Mean Squared Error (MSE): $ 2037.55
Root Mean Squared Error (RMSE): $ 45.14
```

Conclusion

This extensive investigation into flight fare prediction represents a meticulous exploration from exploratory data analysis (EDA) to the deployment of predictive models. The preprocessing stage effectively addressed missing values and introduced temporal categorizations, revealing significant insights into demand distribution, weather impact, and route characteristics. The employment of decision tree and KNN regressors, coupled with the strategic application of Principal Component Analysis (PCA), revealed a discernible trade-off in model performance. Subsequent hyperparameter tuning refined the models, leading to the identification of an optimal configuration for the decision tree model.

The outcomes demonstrated a nuanced performance differential between the models, with the decision tree model exhibiting superior Rsquared performance at 96% when compared to KNN. Noteworthy is the varying impact of PCA on model efficacy, emphasizing the context-specific nature of its application. As we navigate the intricacies of model selection and meticulous tuning, this project contributes valuable insights to the domain of flight fare prediction, underscoring the dynamic landscape inherent in machine learning methodologies.

References

[1] K. Tziridis, Th. Kalampokas, G.A. Papakostas, K.I. Diamantaras , Airfare Prices Prediction Using Machine Learning Techniques, EUSIPCO, 2017.

[2] William Groves and Maria Gini, A regression model for predicting optimal purchase timing for airline tickets, 2011.

[3] Till Wohlfarth, Ste´phan Cle´mencon, Francois Roueff, Xavier Casellato, A Data-Mining Approach to Travel Price Forecasting, 2011.

[4] Tianyi Wang, Samira Pouyanfar, Haiman Tian, Yudong Tao, Miguel Alonso Jr., Steven Luis and Shu-Ching Chen, A Framework for Airfare Price Prediction: A Machine Learning Approach, 2019.

[5] Aryan Madaan, Dr. S. Gnanavel, Implementation of Flight Fare Prediction Web App Project with Deployment, 2022.

[6] Juhar Ahmed Abdella , NM Zaki , Khaled Shuaib , Fahad Khan Airline ticket price and demand prediction: A survey, 2021.

[7] Dr. V. V. Kimbhaune, Harshil Donga, Ashutosh Trivedi, Sonam Mahajan, Viraj Mahajan, Flight Fare Prediction System, 2021.

[8] Zhichao Zhao, Jinguo You, Guoyu Gan, Xiaowu Li, Jiaman Ding, Civil airline fare prediction with a multi-attribute dual-stage attention mechanism, 2021.

[9] Pavithra Maria K , Anitha K L, Flight Price Prediction for Users by Machine Learning Techniques, 2021.

[10] Deepak Ram R1 , Jerome J1 , Mohammed Taha Meeran R1 , Mrs. Uma Devi G, Flight Fare Prediction System, 2023.

[11] Prithviraj Biswas, Rohan Chakraborty, Tathagata Mallik, Rohan Chakraborty, Sk Imran Uddin, Shreya Saha, Pallabi Das, Sourish Mitra, Flight Price Prediction: A Case Study, 2022.

[12] Supriya Rajankar, Neha Sakharkar, A Survey on Flight Pricing Prediction using Machine Learning, 2019.

Appendix-I

(Code)