Flight Price Prediction

Module Code	:	CSMAI21
Module Name	:	Artificial Intelligence and Machine Learning
Student Name	:	Mridul Soni
Student Number	:	30832715
Date	:	10-March-2023

Abstract

The study on utilizing machine learning approaches to estimate flight fares is presented in the report. To create a model that accurately forecasts flight prices, the study makes use of a dataset that includes a variety of features, including airline, source city, departure time, number of stops, and remaining days of the journey, among others. The dataset's exploratory data analysis, which included visualizing the correlation between different attributes and flight costs, is described in the report. A few machine learning algorithms are assessed, and measures like mean absolute error and mean squared error is used to compare the effectiveness of the models. The machine learning model includes Linear Regression, Decision Tree, Random Forest, and Artificial Neural Network (ANN). The outcomes demonstrate that the ANN algorithm delivers the greatest results followed by Random Forest Algorithm. The consequences of the study and prospective directions for further research are covered in the report's conclusion.

Background and problem to be addressed:

Flight price prediction is a difficult subject that involves forecasting the demand and supply of airline tickets, as well as the price elasticity of different flights. Price elasticity quantifies how responsive a flight is to price changes, or how much demand will change when prices change. A study by Alshamari et al. found that current demand prediction algorithms often attempt to forecast passenger demand for a single flight/route and the market share of a particular airline. These models, however, do not account for the interactions between various airlines and routes, which could have an impact on demand and pricing generally. Some of the factors that influence flight prices are:

- Seasonality: Flight demand varies according to the time of year, month, week, or day. Holidays, weekends, and peak seasons, for example, have higher demand and prices than off-peak periods.
- Competition: The number and type of airlines that operate on a given route may influence each airline's pricing strategy and market share. Low-cost carriers, for example, may offer lower prices than full-service carriers to attract more customers.
- Fuel costs: One of the major expenses for airlines, fuel costs can affect profitability and pricing decisions. For example, when fuel prices rise, airlines may raise fares or reduce capacity to save money.
- Booking time: The amount of time between booking and departure can also influence the cost of a flight ticket. In general, booking earlier than the departure date may result in lower prices.

Research Question:

Our study aims to answer the below research questions:

1. What is the relationship between airline and flight prices?
2. How departure time affects the flight prices?
3. How number of stops affect the flight prices?
4. How destination city influence the flight prices?
5. What is the correlation between flight duration and price?
6. How number of days left until the flight, impact the flight prices?
7. How price got affected when tickets were bought 1 or 2 days before?
8. How price varies between economy and business class?
9. How price varies by the day of the week?

Task 1: Exploratory Data Analysis

The dataset consists of information about flight booking options from the Easemytrip website for flight travel between India's top 6 metro cities. There are 300,159 data points and 13 features in the cleaned dataset.

Features:

1. Airline: The airline column contains the name of the airline provider. There are six different airlines, making it a category trait.
2. Source City: The location where the flight departs. It is a classification feature with 6 distinctive cities.
3. Departure Time: By organising periods into bins, we build this derived categorical feature. It has six different time labels and stores information about the departure time.
4. Number of Stops: The number of stops between the source and destination cities is stored in a categorical feature with three separate values.
5. Destination City: The location of the aircraft's landing. It is a classification feature with 6 cities.
6. Arrival Time: By dividing periods into bins, this derived categorical feature was produced. It maintains information regarding the arrival time and has six different time labels.
7. Trip Duration: A running feature that shows the total number of hours needed to travel between two cities.
8. Date of Journey: Journey date of the flight.
9. Days Left: To calculate this derived characteristic, divide the trip date by the booking date.
10. Day of Week: It is derived from the Date of the Journey.
11. Flight Code: Information about the flight of the aircraft is kept in the flight code.
12. Journey Class: This is a categorised characteristic with two unique values: Business and Economy, and it contains information on seat class.
13. Ticket Price: This column keeps track of the ticket price information.

Data Acquisition and Preprocessing

In this step, we will have obtained two unprocessed files, economy.csv and business.csv file containing the flight data based on the Journey Class.

I have created a user-defined function called concat flights( ), which takes df as input. This function reads the data from the two files business and economy and concatenates them into a single file, this function also assigns the 'Business' and 'Economy' class labels to their respective rows.

From the below Fig 1., we can see that the columns are not arranged, have unwanted values and have some missing features to continue with the analysis.

```
concat_flights(df)
```

	date	airline	ch_code	num_code	dep_time	from	time_taken	stop	arr_time	to	price	class
0	11-02-2022	Air India	AI	868	18:00	Delhi	02h 00m	non-stop	20:00	Mumbai	25,612	Business
1	11-02-2022	Air India	AI	624	19:00	Delhi	02h 15m	non-stop	21:15	Mumbai	25,612	Business
2	11-02-2022	Air India	AI	531	20:00	Delhi	24h 45m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:45	Mumbai	42,220	Business
3	11-02-2022	Air India	AI	839	21:25	Delhi	26h 30m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	44,450	Business
4	11-02-2022	Air India	AI	544	17:15	Delhi	06h 40m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	23:55	Mumbai	46,690	Business
...	...	...	...	...	...	...	...	...	...	...	...	...
206769	31-03-2022	Vistara	UK	832	07:05	Chennai	13h 50m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:55	Hyderabad	7,697	Economy
206770	31-03-2022	Vistara	UK	832	07:05	Chennai	13h 50m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:55	Hyderabad	7,709	Economy
206771	31-03-2022	Vistara	UK	826	12:30	Chennai	20h 35m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	09:05	Hyderabad	8,640	Economy
206772	31-03-2022	Vistara	UK	822	09:45	Chennai	23h 20m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	09:05	Hyderabad	8,640	Economy
206773	31-03-2022	Vistara	UK	824	20:30	Chennai	24h 25m	1-stop\n\t\t\t\t\t\t\t\t\t\t\t\t\n\t\t\t\t\t\t...	20:55	Hyderabad	8,640	Economy

300261 rows × 12 columns

Fig 1. Dataframe after concatenation of business and economy files

To Process the data, I have created a series of transform functions with the pipelines, which when called give the output to the original data frame. By executing the below code snippet, we will obtain the processed dataset.

Code Snippet:

```
def preprocess_pipeline(df):

    #Preprocess function to call all the functions one after another
    df = concat_flights(df)
    df = clean_flights(df)
    df = transform_flights(df)
    df = drop_airlines(df)

    return df
```

```
%%time
preprocessed_df = preprocess_pipeline(df)
preprocessed_df.head()
```

```
CPU times: user 3.2 s, sys: 112 ms, total: 3.32 s
Wall time: 3.32 s
```

	Airline	Source City	Departure Time	Number of Stops	Destination City	Arrival Time	Trip Duration	Date of Journey	Days Left	Day of Week	Flight Code	Journey Class	Ticket Price
165631	GO FIRST	Hyderabad	Afternoon	0	Bangalore	Afternoon	1.25	2022-02-11	1	Friday	G8-806	Economy	4452
70310	Indigo	Mumbai	Evening	0	Hyderabad	Evening	1.42	2022-02-11	1	Friday	6E-5208	Economy	4841
70311	Indigo	Mumbai	Evening	0	Hyderabad	Evening	1.33	2022-02-11	1	Friday	6E-5217	Economy	4946
127555	AirAsia	Kolkata	Evening	2	Mumbai	Early Morning	11.50	2022-02-11	1	Friday	I5-1563	Economy	5070
127556	AirAsia	Kolkata	Evening	2	Mumbai	Morning	15.58	2022-02-11	1	Friday	I5-1563	Economy	5070

Fig 2. Dataframe after preprocessing

The below fig 3. Shows us the pipeline flow, in which sequence the functions are called and executed.

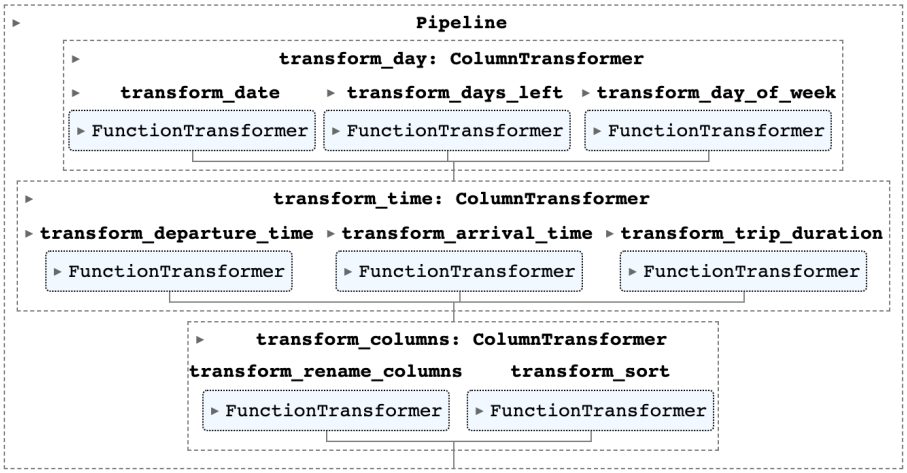


Fig 3. Preprocessing Pipeline

With the preprocessing done, we can see that in the dataset, there are no missing values, and columns are arranged as required for analysis. We can use df.info () function to see the datatypes, null values, and entries in the columns. With the below code snippets, we can easily see the results.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 300159 entries, 165631 to 80957
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Airline           300159 non-null  object
 1   Source City       300159 non-null  object
 2   Departure Time    300159 non-null  object
 3   Number of Stops   300159 non-null  int64
 4   Destination City  300159 non-null  object
 5   Arrival Time      300159 non-null  object
 6   Trip Duration     300159 non-null  float64
 7   Date of Journey   300159 non-null  datetime64[ns]
 8   Days Left         300159 non-null  int64
 9   Day of Week       300159 non-null  object
 10  Flight Code       300159 non-null  object
 11  Journey Class     300159 non-null  object
 12  Ticket Price      300159 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(3), object(8)
memory usage: 32.1+ MB
```

Fig. 4. Dataframe Info

```
df.describe()
```

	Number of Stops	Trip Duration	Days Left	Ticket Price
count	300159.000000	300159.000000	300159.000000	300159.000000
mean	0.924320	12.220923	26.004558	20889.368728
std	0.398111	7.192079	13.560982	22697.637487
min	0.000000	0.830000	1.000000	1105.000000
25%	1.000000	6.830000	15.000000	4783.000000
50%	1.000000	11.250000	26.000000	7425.000000
75%	1.000000	16.170000	38.000000	42521.000000
max	2.000000	49.830000	49.000000	123071.000000

Fig. 5. Dataframe Describe

I have created a function to save the plots in .png format for the plots made during the analysis.

Code Snippet:

```
def save_plot_as_png(plot, title):
    # Replace spaces in the title with underscores and add the file extension
    filename = title.replace(' ', '_') + '.png'

    # Save the plot as a PNG file
    plot.savefig(filename, dpi=600, bbox_inches='tight')

    print(f"Plot saved as {filename}")
```

Categorical Representation of Data:

Starting with the first graphical analysis as seen in Fig. 6, I have plotted the categorical columns with their respective counts. Below is the code Snippet used for the visualization.

Code Snippet:

```
# Plotting the Categorical values and there counts

# Plotting all the plots in the same line
fig, axes = plt.subplots(1, len(list1), figsize=(20, 5))

# Adjusting the space in between
fig.subplots_adjust(wspace=0.4)

# Running loop to iterate for each column, list1 from config file
for i, l in enumerate(list1):
    counts = df[[l]].value_counts().reset_index(name='count')
    counts.plot(x=l, y='count', kind='bar', ax=axes[i], color=list1_colors[i % len(list1_colors)])
    axes[i].set_title(l)
    axes[i].set_xlabel(l)
    axes[i].set_ylabel('Count')

# Showing the Plots
plt.show()

# Saving the plots to .png in the working directory
save_plot_as_png(fig, 'Bar Plots of Categorical Columns')
```

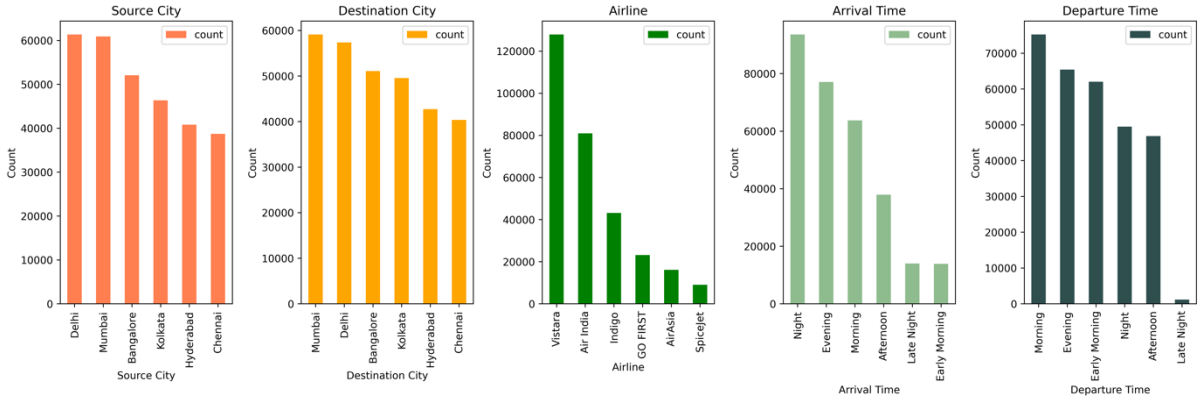


Fig. 6. Bar Plots of Categorical Columns

Interpretation:

Plot 1 & 2: From this bar plot we can see that Delhi and Mumbai are the most popular source/ destination cities and Chennai is the least popular.
Plot 3: From this plot, we can see that Vistara operates most flights followed by Air India in second place, and SpiceJet is a very less operated flight.
Plot 4: This Plot shows that most flights operate in such a way that their arrival time is either Night or Evening.
Plot 5: This Plot shows that most flights operate in such a way that their departure time is either Morning or Evening, and Late Night Departures are very less compared to others.

The below code visualises the Ticket prices range with the counts

Code Snippet:

```
# Plotting the Ticket Prices with their respective counts

fig = plt.figure(figsize=(10,4))

# Histogram plot using seaborn with KDE
sns.histplot(data=df, x='Ticket Price', color="green", label="Airline", kde=True)

# Saving the plots to .png in the working directory
save_plot_as_png(fig, 'Ticket Price vs Flight Counts')
```

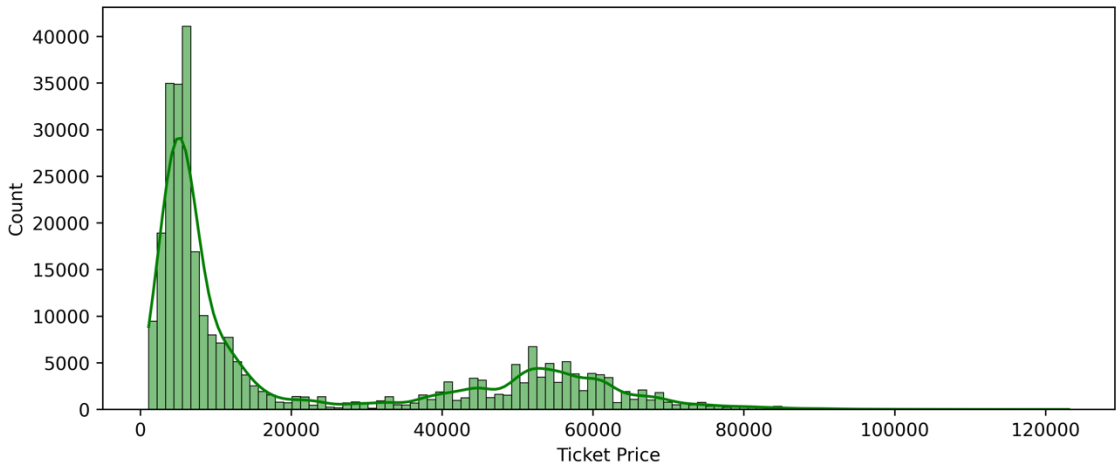


Fig.7. Ticket Price vs Flight Count

Interpretation:
The x-axis represents the range of ticket prices, while the y-axis indicates the frequency or count of flights falling within that price range. The chart illustrates the distribution of flight ticket prices. A KDE (Kernel Density Estimation) plot is used to show a line on the graph that represents the distribution's density. Overall, the graphic offers a quick and simple way to see how airline ticket prices are distributed as well as the relative frequency of various price categories.

I have created a function to plot the line graph for various columns just by changing a few parameters in the function.

Code Snippet:

```
def plot_flight_prices(data, x_col, y_col, hue_col, palette, title, layout):

    # Sorting the Business flights dataframe by the order of departure times
    business_flights_sorted = business_flights.sort_values(x_col)

    # Sorting the Economy flights dataframe by the order of departure times
    economy_flights_sorted = economy_flights.sort_values(x_col)

    # Set up the figure with two subplots
    if (layout =='H'):
        fig, axs = plt.subplots(1, 2, figsize=(20, 7))
    else:
        fig, axs = plt.subplots(2, 1, figsize=(15, 15))

    # Plot the distribution of flight prices across different departure times for Business flights
    sns.lineplot(data=business_flights_sorted, x=x_col, y=y_col, hue=hue_col, ax=axs[0], palette=palette)
    axs[0].set_title('Business Flights', fontsize =14)
    axs[0].get_legend().remove()
    axs[0].grid(color='lightgrey', linestyle='--', linewidth=0.9)

    # Plot the distribution of flight prices across different departure times for Economy flights
    sns.lineplot(data=economy_flights_sorted, x=x_col, y=y_col, hue=hue_col, ax=axs[1], palette=palette)
    axs[1].set_title('Economy Flights', fontsize =14)
    axs[1].get_legend().remove()
    axs[1].grid(color='lightgrey', linestyle='--', linewidth=0.9)

    # Plotting the common legend for both plots on right center
    handles, labels = axs[1].get_legend_handles_labels()
    unique_labels = list(set(labels))
    unique_handles = [handles[labels.index(label)] for label in unique_labels]
    fig.legend(unique_handles, unique_labels,loc='center right', bbox_to_anchor=(1.0, 0.5), fancybox= True)

    # Set the title of the figure
    fig.suptitle(title, fontsize =18)
    plt.grid(color = 'lightgrey', linestyle = '--', linewidth = 0.9)
    plt.show()
    save_plot_as_png(fig, title)
```

To enhance the plot readability, I am dividing the data frame in the Business and Economy class.

```
# Select rows where Journey class is Business or Economy
business_flights = df[df['Journey Class'] == 'Business']
economy_flights = df[df['Journey Class'] == 'Economy']
```

Effect on Flight Prices based on the Departure Time

Code Snippet:

```
# Plotting Flight Prices based on the Departure Time with the function

plot_flight_prices(df, 'Departure Time', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Different Departure Time', 'H')
```

This code is creating a visual representation of how ticket prices vary based on departure time and airline, allowing for easy comparison between the different airlines and identifying any patterns or trends in ticket pricing over time.

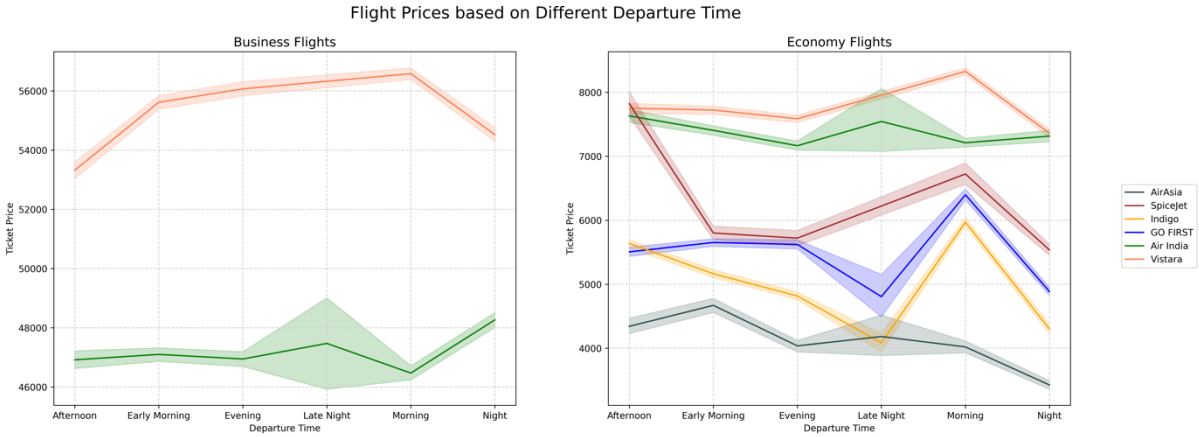


Fig. 8. Flight Prices based on Different Departure Times

From Fig. 8. we can see that in Business Class and Economy Class, Vistara have the highest price during the morning. All the other airlines have their prices high in the morning, while lowest at the night.

Effect on Flight Prices based on the Source City

Code Snippet:

```
# Plotting Flight Prices based on the Source City with the function

plot_flight_prices(df, 'Source City', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Different Source City', 'H')
```

This code is creating a visual representation of how ticket prices vary based on the source city and airline, allowing for easy comparison between the different airlines and identifying any patterns or trends in ticket pricing over the source city.

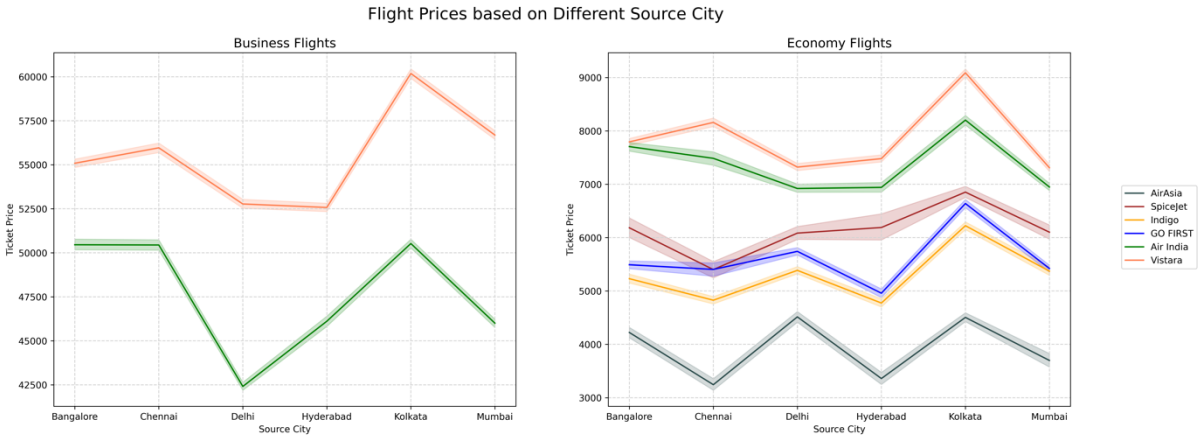


Fig. 9. Flight Prices based on Different Source City

By analyzing Fig.9. we can see that Kolkata is the most expensive city to get boarded on all airlines in both business and economy class.

Effect on Flight Prices based on the Destination City

Code Snippet:

```
# Plotting Flight Prices based on the Destination City with the function

plot_flight_prices(df, 'Destination City', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Different Destination City', 'H')
```

This code is creating a visual representation of how ticket prices vary based on destination city and airline, allowing for easy comparison between the different airlines and identifying any patterns or trends in ticket pricing over destination city.

Fig. 10. Flight Prices based on Different Destination City

By analyzing Fig.10. we can see that Kolkata still is the most expensive city to get off-boarded on all airlines in both business and economy class. While Chennai has the least expensive destination.

Effect on Flight Prices based on the Number of Stops

Code Snippet:

```
# Plotting Flight Prices based on the Number of Stops with the defined function

plot_flight_prices(df, 'Number of Stops', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Number of Stops', 'H')
```

This code is creating a visual representation of how ticket prices vary based on the Number of Stops and airline, allowing for easy comparison between the different airlines and identifying any patterns or trends in ticket pricing over the Number of Stops.

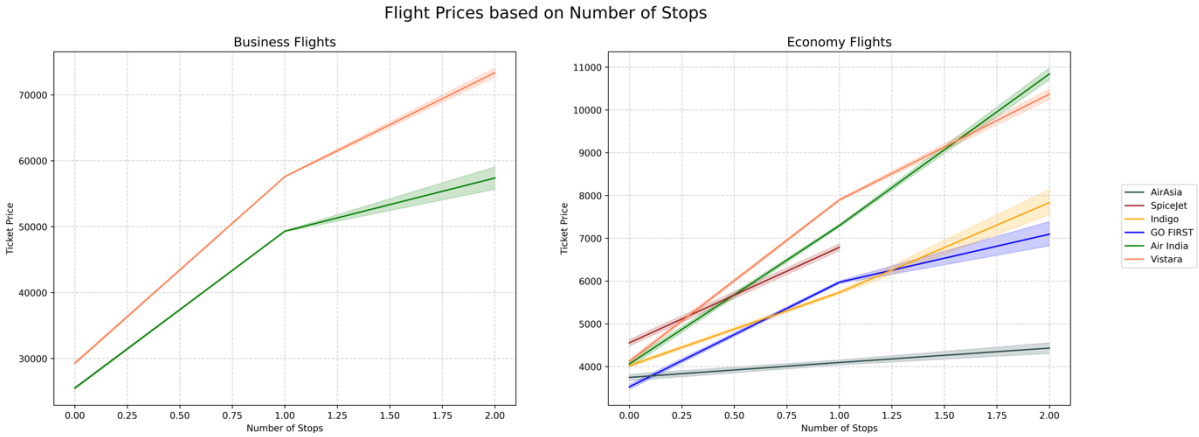


Fig. 11. Flight Prices based on the Number of Stops

By analyzing Fig.11. we can see that as the number of stops increases, the prices of airlines also increase for both economy and business class.

Effect on Flight Prices based on the Trip Duration

```
# Plotting Flight Prices based on the Trip Duration with the defined function

plot_flight_prices(df, 'Trip Duration', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Trip Duration', 'V')
```

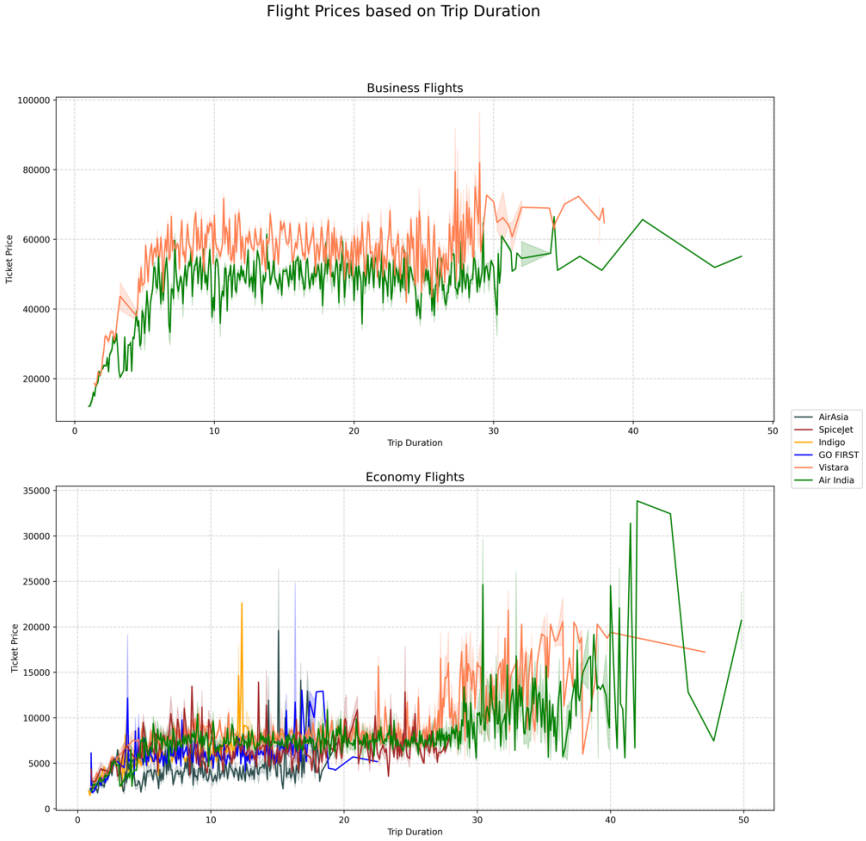


Fig. 12. Flight Prices based on Trip Duration of All Flights

Simplified Version:

```
# Plotting Flight Prices based on the Simplified Trip Duration

fig = plt.figure(figsize=(20,8))
sns.lineplot(data=df, x='Trip Duration', y='Ticket Price', hue='Journey Class')
plt.xticks(range(0, 52, 5))
plt.show()

# Saving the plots to .png in the working directory
save_plot_as_png(fig, 'Trip Duration vs Ticket Price')
```

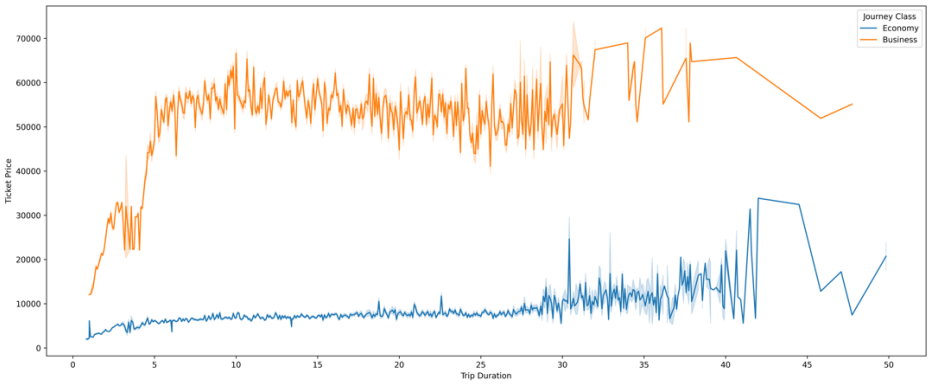


Fig. 13. Flight Prices based on Trip Duration of All Flights (Simplified)

From Fig. 12 and 13, we can see that as the trip duration increased, the prices of flights also increase.

Effect on Flight Prices based on the Day of Week

Code Snippet:

```
# Plotting Flight Prices based on the Day of Week with the defined function

plot_flight_prices(df, 'Day of Week', 'Ticket Price', 'Airline',
                   airline_colors, 'Flight Prices based on Day of Week', 'H')
```

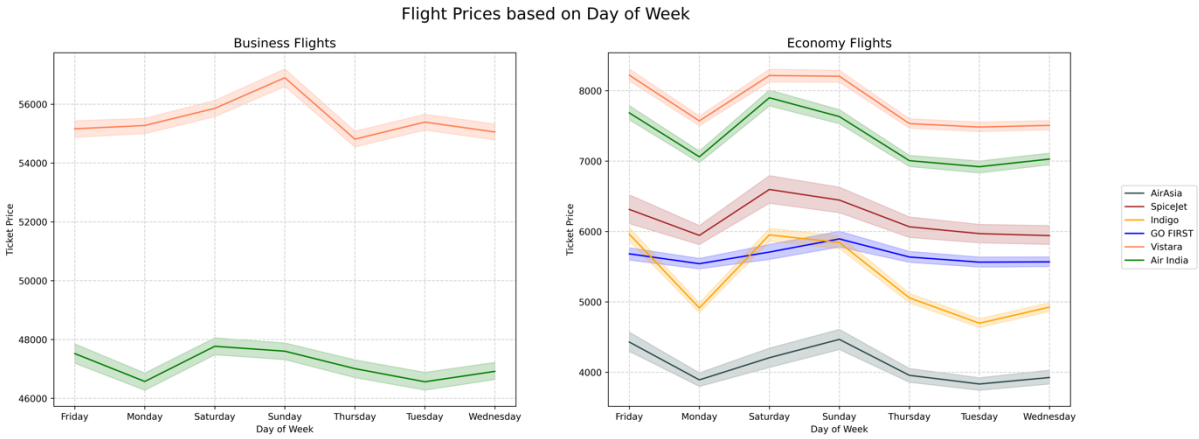


Fig. 14. Flight Prices based on the Day of Week

From the Fig.14. we can see that on weekends, flight prices are high especially on Sundays for both Business and Economy classes. While the lowest prices are on Monday, Tuesday, and Wednesday for respective airlines.

Effect on Flight Prices based on the Days Left

```
# Plotting Flight Prices based on the Days left

fig = plt.figure(figsize=(10,4))
sns.lineplot(data=df, x='Days Left', y='Ticket Price', hue='Journey Class')
plt.xticks(range(0, 52, 5))

# Shaded area for Economy Flight Price Change
plt.axhspan(5000, 10000, 0.305, 0.37, alpha=0.2, color='dodgerblue')

# Adding a label to the shaded area with an arrow pointing
plt.annotate('Economy Class Price Change', xy=(18, 10000), xytext=(27, 25000),
             arrowprops=dict(facecolor='black', arrowstyle='->'))
plt.annotate('Business Class Price Change', xy=(11, 55000), xytext=(27, 60000),
             arrowprops=dict(facecolor='black', arrowstyle='->'))

# Shaded area for Business Flight Price Change
plt.axhspan(50000, 55000, 0.175, 0.24, alpha=0.2, color='orange')
plt.show()

# Saving the plot witht the function
save_plot_as_png(fig, 'Price change window based on Journey Class')
```

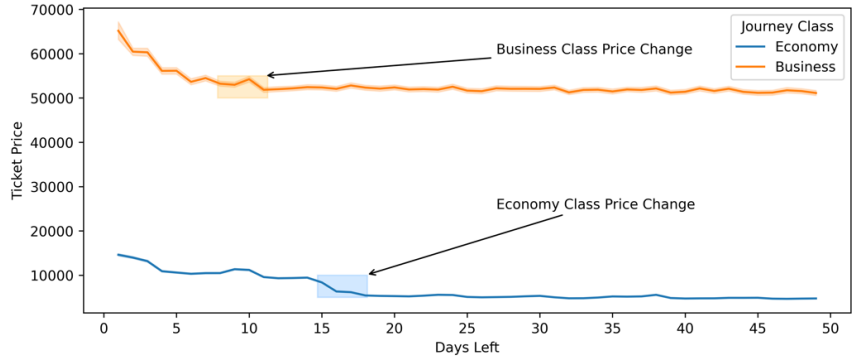


Fig. 15. Flight Prices based on Days Left

From figure 15 we can see that flight prices remain constant for a while but soon starts to increase as the days left become fewer. From the plot, prices for economy class change between 15-18 days and for business class change between 8-11 days.

Effect on Flight Prices based on the Number of Stops with variations using Violin Plot

```
# Plotting the Violin Plots for Flight Price Variation based on Number of Stops

fig, axs = plt.subplots(1, 2, figsize=(20, 7))

# Plot for Business Flights
sns.violinplot(x='Number of Stops', y='Ticket Price', ax=axs[0], data=business_flights)
axs[0].set_title('Business Flights', fontsize =14)

# Plot for Economy Flights
sns.violinplot(x='Number of Stops', y='Ticket Price', ax=axs[1], data=economy_flights)
axs[1].set_title('Economy Flights', fontsize =14)

fig.suptitle('Flight Price Variation based on Stops', fontsize =18)
plt.show()

# Saving the plot with the function
save_plot_as_png(fig, 'Flight Price Variation based on Stops')
```

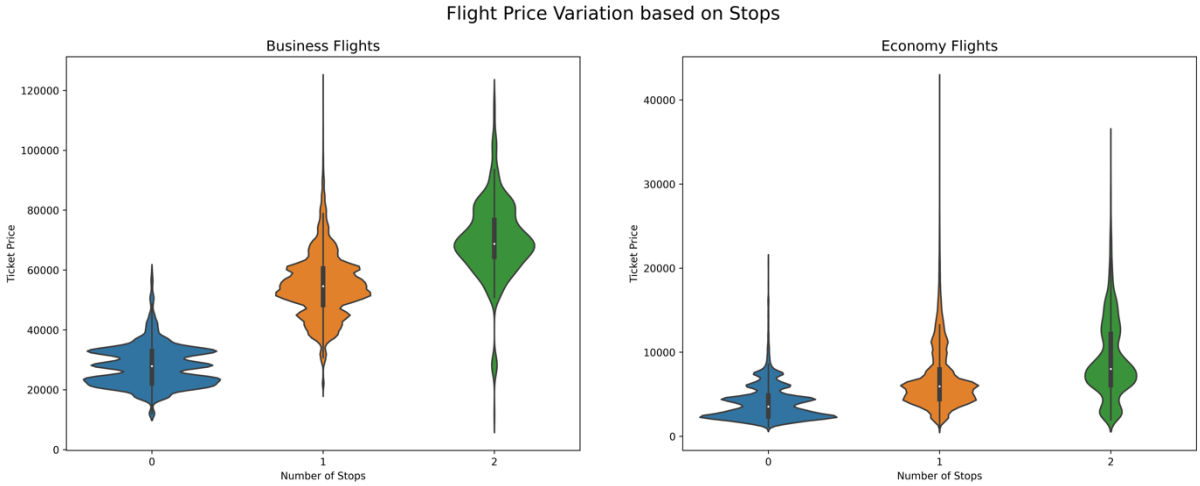


Fig. 16. Flight Prices based on the Number of Stops (Violin Plot)

From Fig.16, we can see that prices for business class vary drastically as the number of stops increases, while for the economy class, it increases with fewer variations.

I have defined a function to plot the pie charts by just changing the columns names and titles of the pie chart.

Code Snippet:

```
def pie_plots(column, title):
    counts = df[column].value_counts()

    # Create the pie chart
    fig, ax = plt.subplots()
    pie = ax.pie(counts,labels= counts.index, autopct='%1.1f%%',
                 wedgeprops={ 'linewidth' : 2, 'edgecolor' : 'white' },
                 startangle=150,labeldistance=1.1)
    # Add a circle at the center of the chart to create the donut shape
    center_circle = plt.Circle((0, 0), 0.8, fc='white')
    fig.gca().add_artist(center_circle)

    plt.text(0, 0, column, ha='center', va='center', fontsize=10)
    plt.title(title, fontsize=18)
    # Show the plot
    plt.show()

    # Save the plot with the function
    save_plot_as_png(fig, title)
```

Pie Charts

```
pie_plots('Airline', 'Airline Market Share')
pie_plots('Source City', 'Flights from Source City')
pie_plots('Destination City', 'Flights to Destination City')
pie_plots('Arrival Time', 'Arrival Time')
```

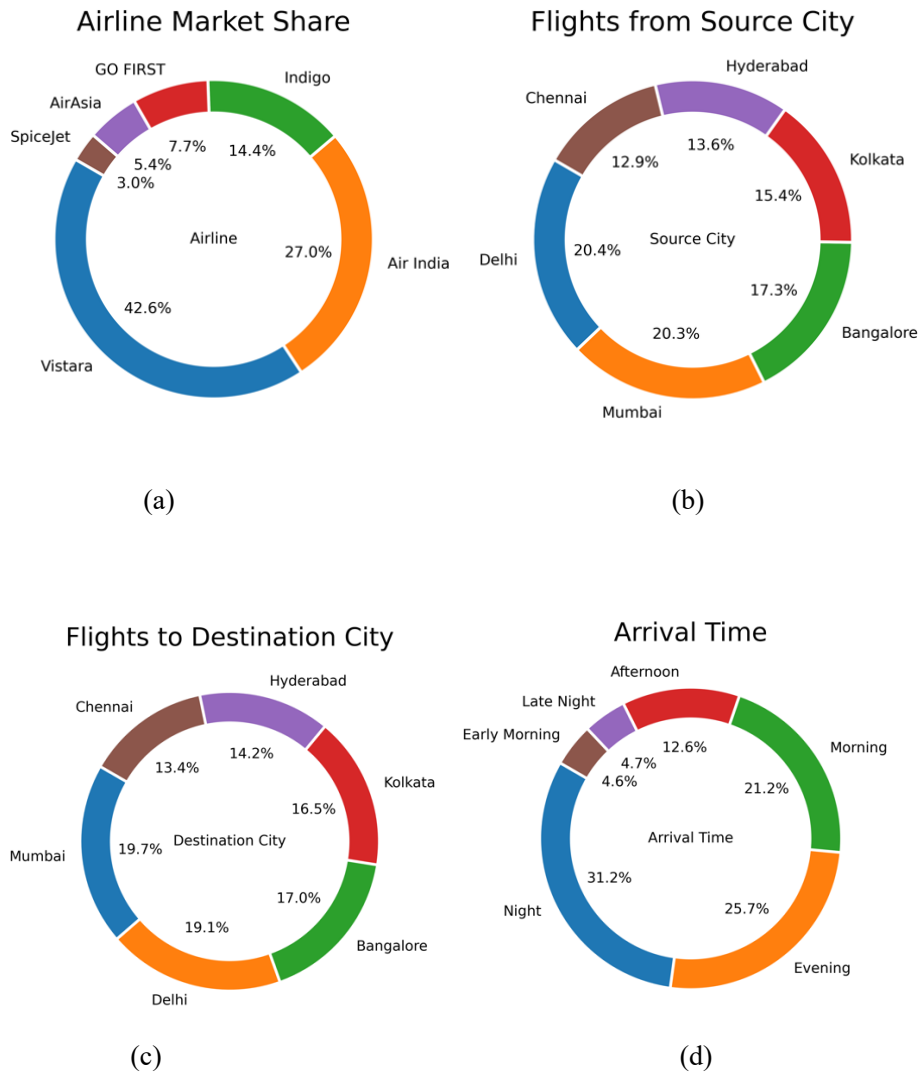


Fig. 18. Pie-Charts

The above pie charts suggest the percentage share of the categorical columns.
Fig 18 (a) shows Vistara has the highest market share and Spicejet have the lowest.
Fig 18 (b) shows that Delhi and Mumbai are the popular source city for the airline.
Fig 18 (c) shows that Delhi and Mumbai are the popular destination city for the airline.
Fig 18 (d) shows that Night and Evening are the popular arrival time.

Top Popular Flights and Least Popular Flights

Code Snippet:

```
top_flights = df['Flight Code'].value_counts().head(5)
least_flights = df['Flight Code'].value_counts().tail(5)

print('Top Popular Flights: \n')
print(top_flights)
print('\n')
print('Least Popular Flights: \n')
print(least_flights)
print('\n')
```

```
Top Popular Flights:

UK-706    3235
UK-772    2741
UK-720    2650
UK-836    2542
UK-822    2468
Name: Flight Code, dtype: int64


Least Popular Flights:

AI-9991    1
6E-6613    1
6E-3211    1
6E-6474    1
G8-705     1
Name: Flight Code, dtype: int64
```

Correlation Plot of Numerical Columns

```
corr = df.corr()

# Visualize the correlation matrix using a heatmap
fig = sns.heatmap(corr, annot=True, cmap='coolwarm').get_figure()
plt.title('Correlation matrix of Flight Data')
plt.show()
save_plot_as_png(fig,'Correlation matrix of Flight Data')
```

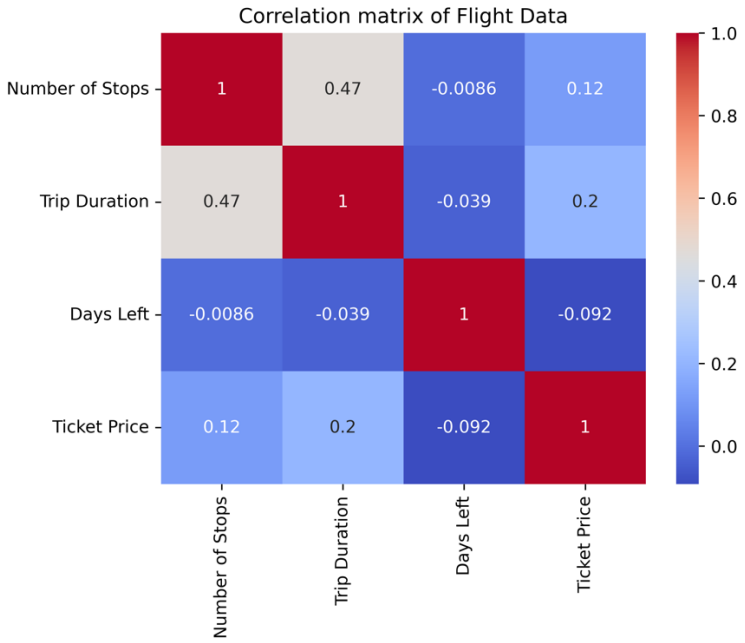


Fig. 19. Correlation Matrix of Flight Data

Task 2: Machine Learning Models

Starting with accessing the data file.

```
df = pd.read_csv('cleaned_flight_dataset.csv')
```

Now we have to drop a few columns that are not required for our machine-learning analysis

```
df = df.drop(['Date of Journey', 'Flight Code'], axis=1)
```

Separating the data frame from the target variable ticket prices

```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

We have used one hot encoder instead of a label encoder because while assigning labels to the categorical value the machine will misinterpret them as a priority to the variables having higher weight. To avoid this we are using one hot encoder in our machine-learning models. Also, One hot encoding is applied to the whole dataset not only the training set, to ensure consistency and accuracy in the model's predictions.

```
ct = ColumnTransformer(transformers=[
    ('ohe',OneHotEncoder(drop='first',sparse_output=False,handle_unknown='ignore'),
    ['Airline','Source City','Destination City','Journey Class','Departure Time','Arrival Time', 'Day of Week'])
],remainder='passthrough')

# Fit and transform the model with the encoded values
X= ct.fit_transform(X)
X
```

```
array([[ 0.  ,  1.  ,  0.  , ...,  0.  ,  1.25,  1.  ],
       [ 0.  ,  0.  ,  1.  , ...,  0.  ,  1.42,  1.  ],
       [ 0.  ,  0.  ,  1.  , ...,  0.  ,  1.33,  1.  ],
       ...,
       [ 0.  ,  0.  ,  0.  , ...,  1.  ,  9.42, 49.  ],
       [ 0.  ,  0.  ,  0.  , ...,  1.  , 12.5 , 49.  ],
       [ 0.  ,  0.  ,  0.  , ...,  1.  ,  9.67, 49.  ]])
```

To start with the modules we have to split datasets into training and testing, To do this we are using the train test split function.

```
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.3, random_state=36)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(210111, 35)
(90048, 35)
(210111,)
(90048,)
```

In this step, we are converting X to a data frame which was an array, for assigning the column headers that are lost during one hot encoding. To do this we are using the get_feature_names_out function to retain the headers.

```
#Converting X into Dataframe to assign the column header
X = pd.DataFrame(X)

# Getting column name that are lost after one hot encoding
X.columns=ct.get_feature_names_out()
X
```

Below is the data frame we get after applying the get features name out to the X data frame

	ohe__Airline_AirAsia	ohe__Airline_GO FIRST	ohe__Airline_Indigo	ohe__Airline_SpiceJet	ohe__Airline_Vistara	ohe__Source City_Chennai	ohe__Source City_Delhi	ohe__Source City_Hyderabad
0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
300154	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
300155	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
300156	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
300157	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
300158	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0

300159 rows × 35 columns

Calculating VIF

I have defined a function called calculate_vif. This function will return the VIF score of all the columns. Vif score is used to see whether the columns have the problem of multicollinearity or not. Multicollinearity arises after one hot encoding is applied to the data frame.

```
# Function to calculate VIF
def calculate_vif(X):
    vif_df = pd.DataFrame(columns = ['Var', 'Vif'])
    x_var_names = X.columns
    for i in range(0, x_var_names.shape[0]):
        y = X[x_var_names[i]]
        x = X[x_var_names.drop([x_var_names[i]])]
        r_squared = sm.OLS(y,x).fit().rsquared
        vif = round(1/(1-r_squared),2)
        vif_df.loc[i] = [x_var_names[i], vif]
    return vif_df.sort_values(by = 'Vif', axis = 0, ascending=False, inplace=False)

calculate_vif(X)
```

We applied this function and as a result, we found that most of the columns have a score of less than 5 which means that they are having less multicollinearity. we are ignoring reminder columns as they are not one hot encoded so they are not having multicollinearity. As a result, we are not dropping any columns.

We are using another feature selection using an extra tree regressor. This function returns a set of selected features of the columns, which is helpful for column selection.

```
# Important feature using ExtraTreesRegressor
selection = ExtraTreesRegressor()
selection.fit(X, y)

print(selection.feature_importances_)
```

```
[5.11107158e-04 2.17687247e-04 4.23774137e-04 1.97032984e-04
 1.01181084e-02 1.06467981e-03 4.46224212e-03 1.47957783e-03
 2.10882318e-03 2.38878791e-03 1.07182094e-03 4.12042699e-03
 1.92963349e-03 2.00258503e-03 2.03483750e-03 8.80364530e-01
 1.13438472e-03 1.56577651e-03 5.32865847e-05 1.35017569e-03
 1.05118839e-03 9.20506817e-04 1.48241234e-03 2.18561289e-04
 1.02707482e-03 1.73314948e-03 5.42700271e-04 6.26403906e-04
 8.24662957e-04 4.24920250e-04 5.20014293e-04 4.93073199e-04
 3.26267840e-02 2.40567768e-02 1.48524930e-02]
```

Now we have we are plotting the feature's importance.
We can see that the top 5 columns that are the most important in our data frame.

```
#plot graph of feature importances for better visualization
plt.figure(figsize = (8,4))
feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.xlabel('Importance ---->')
plt.ylabel('Column Names ---->')
plt.show()
```

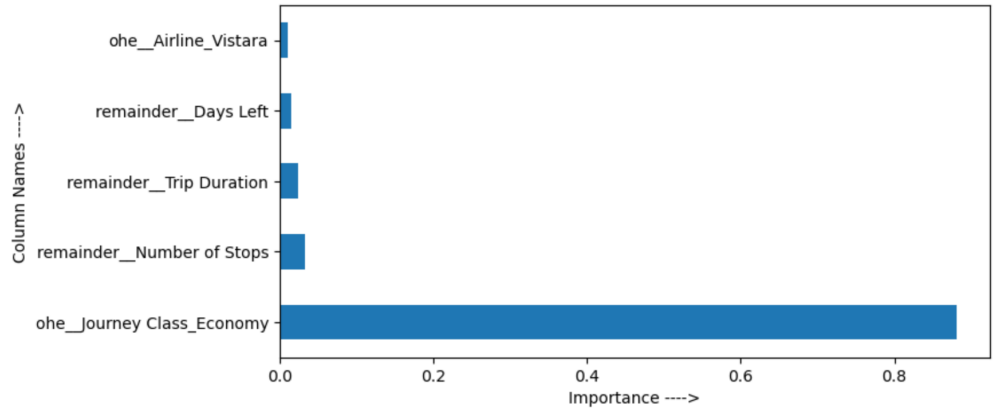
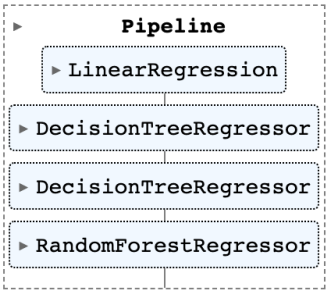


Fig. 20. Correlation Matrix of Flight Data

I have created a pipeline for machine learning models which includes linear regression, decision trees, and random forest algorithms.

```
# Creating Pipeline
# dt1 and dt2 represents model with different hyperparameters
pipeline = Pipeline([
    ('lr', LinearRegression()),
    ('dt1', DecisionTreeRegressor()),
    ('dt2', DecisionTreeRegressor()),
    ('rf', RandomForestRegressor())
])
pipeline
```



Reason for model Selections:

Decision Tree, Random Forest, and Artificial Neural Network (ANN) models are all viable options for flight price prediction due to their ability to understand and interpret data, handle categorical and numerical data, and identify important features affecting the price of flights. Random Forest combines multiple Decision Trees to improve accuracy and robustness, while ANN models can learn to identify subtle patterns and relationships between factors that influence flight prices and achieve high levels of accuracy. The choice of model depends on the specific requirements of the task and the available data and computing resources.

Linear Regression

Talking about linear regression, we will fit the model using x_train and y_train.

```
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
```

```
y_pred_lr
```

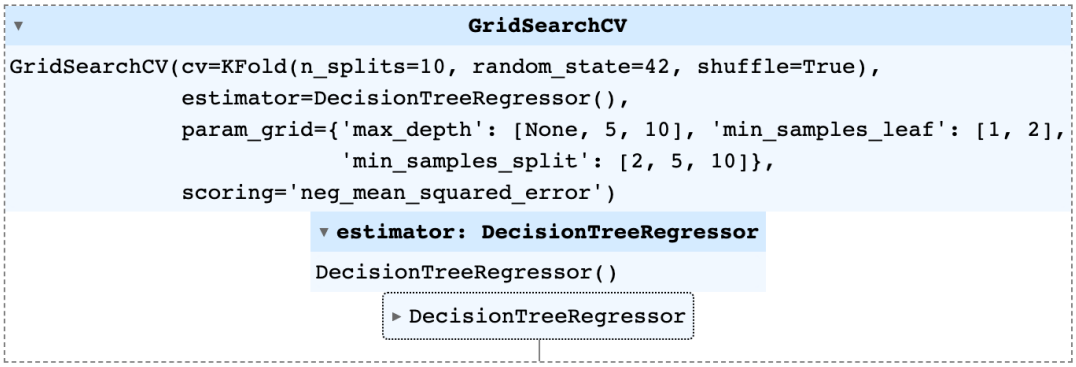
```
array([ 5251.17844606, 52464.98651004,  4142.69889904, ...,
        6574.64477449,  8460.99846399, 53564.9439631 ])
```

Decision Tree

For the decision tree regressor, we are using grid search and cross-validation for finding out the best parameters for the model from the given set of parameters. As the dataset is large we have fitted the model to training data.

```
cv = KFold(n_splits=10, shuffle=True, random_state=42)
```

```
# Using Grid Search and fitting the model
dt1 = GridSearchCV(dt1, param_grid=param_grid, cv=cv, scoring='neg_mean_squared_error')
dt1.fit(X_train,y_train)
```



Displaying the Best Parameters

```
print(f"Best hyperparameters: {dt1.best_params_}")
print(f"Best cross-validation score: {-dt1.best_score_:.2f}")
```

```
Best hyperparameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10}
Best cross-validation score: 9495321.69
```

Comparing 2 different Decision tree models to see the effect of hyperparameter tuning.

Fitting model 1 with Best Parameters

```
#Fitting the model
dt1 = DecisionTreeRegressor(max_depth=None, min_samples_split=10, min_samples_leaf= 2)
dt1.fit(X_train, y_train)

y_pred_dt = dt1.predict(X_test)
y_pred_dt
```

```
array([ 3194.75, 51457.  ,  2074.  , ...,  4819.75,  4964.5 , 43885.  ])
```

Fitting model 2 with not tuned hyperparameters.

```
dt2 = DecisionTreeRegressor(max_depth=5, min_samples_split=10)
dt2.fit(X_train, y_train)

y_pred = dt2.predict(X_test)
y_pred
```

```
array([ 4095.55221771, 57065.88782623,  4095.55221771, ...,
        5962.61735245,  4450.54305913, 57065.88782623])
```

Random Forest Regressor

```
#Random forest model with tuned hyperparameters
rf = RandomForestRegressor(n_estimators=100, max_depth=20, random_state=75)
rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)
y_pred_rf
```

```
array([ 3240.45685386, 55241.65083398,  2347.31224453, ...,
        6250.85059378,  5827.69074569, 49005.12352525])
```

I have created a function to print out the scores of the models.

```
def evaluate_pipeline(pipeline, y_pred_all, model_name):
    score = round((pipeline.score(X_test, y_test)*100), 2)
    mse = round(mean_squared_error(y_test, y_pred_all), 3)
    rmse = round(np.sqrt(mean_squared_error(y_test, y_pred_all)), 3)
    r2 = round(r2_score(y_test, y_pred_all), 3)
    mae = mean_absolute_error(y_test, y_pred_all)

    metrics = pd.DataFrame({
        'Model Name': model_name,
        'Accuracy(%)': [score],
        'Mean Squared Error (MSE)': [mse],
        'Root Mean Squared Error (RMSE)': [rmse],
        'R-squared (R2-Score)': [r2],
        'Mean Absolute Error (MAE)': [mae]
    })
    return metrics
```

Calling the functions

```
r1 = evaluate_pipeline(lr, y_pred_lr, 'Linear Regression')
r2 = evaluate_pipeline(dt1,y_pred_dt, 'Decision Tree - Best Tuned')
r3 = evaluate_pipeline(dt2,y_pred, 'Decision Tree - Basic Tuned')
r4 = evaluate_pipeline(rf, y_pred_rf, 'Random Forest')


all_results = pd.concat([r1, r2, r3, r4], ignore_index=True)
all_results
```

Results

	Model Name	Accuracy(%)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)	R-squared (R2-Score)	Mean Absolute Error (MAE)
0	Linear Regression	90.95	4.657037e+07	6824.249	0.909	4537.826712
1	Decision Tree - Best Tuned	98.21	9.205503e+06	3034.057	0.982	1179.342641
2	Decision Tree - Basic Tuned	94.23	2.970222e+07	5449.974	0.942	3201.268597
3	Random Forest	98.46	7.902766e+06	2811.186	0.985	1330.760183

We can see that Regression Tree Performed well among the others and the Decision Tree Model which is tuned with the best parameters. The decision with untuned parameters has less accuracy.
Linear regression is the worst-performing model in the pipeline, we can discard the Linear Regression model and focus on Decision Tree and Random Forest Regression.

Plotting the Comparison Scatter Plot for Different Regression Models

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 6))

# Plot the first scatter plot in the left subplot
ax1.scatter(y_test, y_pred_lr, s=2)
ax1.set_title('Linear Regression Results')
ax1.set_xlabel('True Values')
ax1.set_ylabel('Predicted Values')
ax1.set_ylim(0,125000)
ax1.set_xlim(0,125000)

# Plot the second scatter plot in the middle subplot
ax2.scatter(y_test, y_pred_dt, s=2)
ax2.set_title('Decision Tree Regression Results')
ax2.set_xlabel('True Values')
ax2.set_ylabel('Predicted Values')
ax2.set_ylim(0,125000)
ax2.set_xlim(0,125000)

# Plot the third scatter plot in the right subplot
ax3.scatter(y_test, y_pred_rf, s=2)
ax3.set_title('Random Forest Regression Results')
ax3.set_xlabel('True Values')
ax3.set_ylabel('Predicted Values')
ax3.set_ylim(0,125000)
ax3.set_xlim(0,125000)

for ax in (ax1, ax2, ax3):
    x = np.linspace(*plt.xlim())
    ax.plot(x, x, transform=ax.transAxes, ls='--', c='red')

# Adjust the spacing between the subplots
fig.subplots_adjust(wspace=0.4)

plt.suptitle("Comparison Scatter Plots of Different Regression Results", fontsize=20)

plt.show()
save_plot_as_png(fig, 'Comparison Scatter Plots of Different Regression Results')
```

Fig. 21. Comparison of Scatter Plots for Different Regressions

Cross- Validation Box plots for Different Regression Models

Code Snippet:

```
cv_results_lr = cross_val_score(lr, X, y, cv=5)
cv_results_dt = cross_val_score(dt1, X, y, cv=5)
cv_results_rf = cross_val_score(rf, X, y, cv=5)

# Plot the cross-validation results using boxplots
fig, ax = plt.subplots(1,3, figsize=(15, 6))
fig.subplots_adjust(wspace=0.4)

# Plot the first box plot in the left subplot
ax[0].boxplot(cv_results_lr, notch=True)
ax[0].set_title('Cross-validation -> Linear Regression')
ax[0].set_ylabel('Accuracy ---->')

# Plot the second box plot in the middle subplot
ax[1].boxplot(cv_results_dt, notch=True)
ax[1].set_title('Cross-validation -> Decision Tree')
ax[1].set_ylabel('Accuracy ---->')

# Plot the third box plot in the right subplot
ax[2].boxplot(cv_results_rf, notch=True)
ax[2].set_title('Cross-validation -> Random Forest')
ax[2].set_ylabel('Accuracy ---->')

plt.suptitle('Cross-validation Results', fontsize=18)

plt.show()
save_plot_as_png(fig, 'Cross-validation Results Box Plots')
```

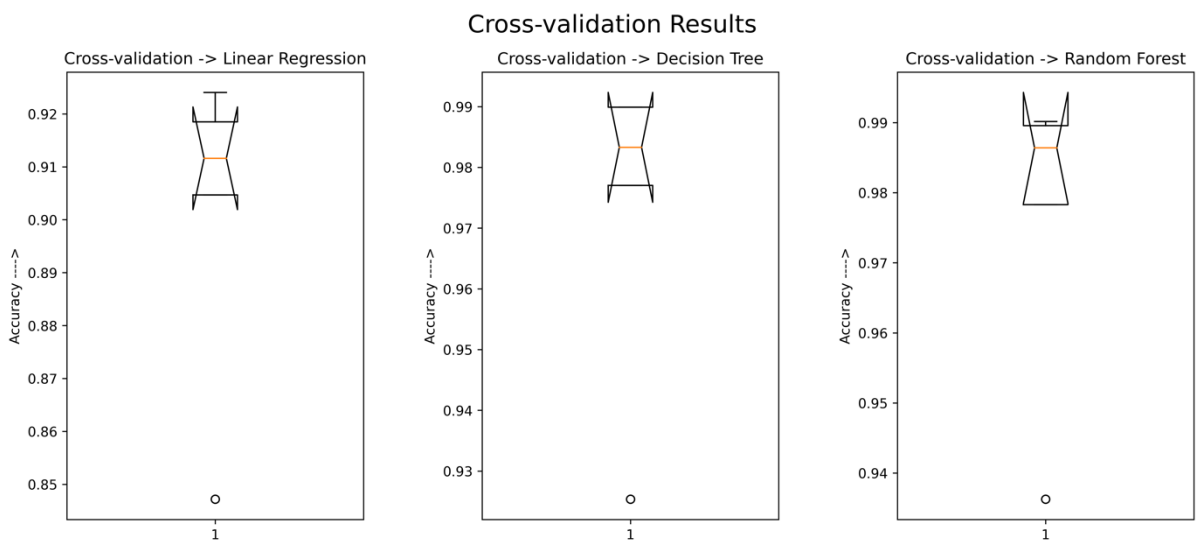


Fig. 21. Cross- Validation Box plots for Different Regression Models

The boxplot allows us to compare the distribution of accuracy scores for each model and to see whether there are any outliers or significant differences between the folds. We can also compare the medians and the spread of the data between the different models to get an idea of which model is performing better overall. In our case Decision Tree and Random Forest are performing much better.

Deep Learning Model – Artificial Neural Network

Splitting the data again to get the refreshed testing and training data.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.3, random_state=0)
```

Standard Scalar Model

First, we are using the Standard Scalar technique to scale the test and train data.
Scaling requires the adjustment of the y data frame, so we are reshaping the data frame

```
# Changing the dimension of y for scalar transformation
y = np.ravel(y).astype('float32').reshape(-1,1)
y = y.reshape(-1,1)
```

```
#scaling the data before running the model
scaler_x = StandardScaler()
X_train = scaler_x.fit_transform(X_train)
X_test = scaler_x.transform(X_test)

scaler_y = StandardScaler()
y_train = scaler_y.fit_transform(y_train)
y_test = scaler_y.transform(y_test)
```

Defining the Artificial Neural Network (ANN) model

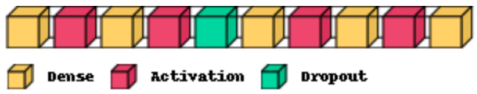
Code Snippet:

```
ANN_model = keras.Sequential()
ANN_model.add(Dense(100, input_dim = 35))
ANN_model.add(Activation('relu'))
ANN_model.add(Dense(150))
ANN_model.add(Activation('relu'))
ANN_model.add(Dropout(0.25))
ANN_model.add(Dense(150))
ANN_model.add(Activation('relu'))
ANN_model.add(Dense(150))
ANN_model.add(Activation('linear'))
ANN_model.add(Dense(1))
ANN_model.compile(loss = 'mse', optimizer = 'adam')
ANN_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               3600

 activation (Activation)     (None, 100)               0

 dense_1 (Dense)             (None, 150)               15150

 activation_1 (Activation)   (None, 150)               0

 dropout (Dropout)           (None, 150)               0

 dense_2 (Dense)             (None, 150)               22650

 activation_2 (Activation)   (None, 150)               0

 dense_3 (Dense)             (None, 150)               22650

 activation_3 (Activation)   (None, 150)               0

 dense_4 (Dense)             (None, 1)                 151

=================================================================
Total params: 64,201
Trainable params: 64,201
Non-trainable params: 0
_________________________________________________________________
```

Visualizing the model

```
import visualkeras
visualkeras.layered_view(ANN_model, legend=True)
```



Compiling the model

```
ANN_model.compile(optimizer = 'Adam', loss = 'mean_squared_error')
```

```
epochs_hist = ANN_model.fit(X_train, y_train, epochs = 20, batch_size = 20, validation_split = 0.2)
```

```
Epoch 1/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0608 - val_loss: 0.0395
Epoch 2/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0401 - val_loss: 0.0359
Epoch 3/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0368 - val_loss: 0.0335
Epoch 4/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0347 - val_loss: 0.0374
Epoch 5/20
8405/8405 [==============================] - 10s 1ms/step - loss: 0.0338 - val_loss: 0.0320
```

Testing the Model and Results

```
result = ANN_model.evaluate(X_test, y_test)
accuracy_ANN = 1 - result
```

```
2814/2814 [==============================] - 2s 644us/step - loss: 0.0304
```

```
print(f"Accuracy: {round((accuracy_ANN)*100, 3)}%")
```

```
Accuracy: 96.96%
```

MinMax Scalar Model

This time we will scale the model using the MinMax Scalar technique to scale the training and testing data.

```
#scaling the data before running the model

minmax_x = MinMaxScaler()
X_train_m = minmax_x.fit_transform(X_train)
X_test_m = minmax_x.transform(X_test)

minmax_y = MinMaxScaler()
y_train_m = minmax_y.fit_transform(y_train)
y_test_m = minmax_y.transform(y_test)
```

Compiling the same model with MinMax Scalar

```
epochs_hist_m = ANN_model.fit(X_train_m, y_train_m, epochs = 20, batch_size = 20, validation_split = 0.2)
```

```
Epoch 1/20
8405/8405 [==============================] - 10s 1ms/step - loss: 0.0023 - val_loss: 0.0027
Epoch 2/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0015 - val_loss: 0.0030
Epoch 3/20
8405/8405 [==============================] - 8s 1ms/step - loss: 0.0013 - val_loss: 0.0025
Epoch 4/20
8405/8405 [==============================] - 8s 981us/step - loss: 0.0013 - val_loss: 0.0019
Epoch 5/20
8405/8405 [==============================] - 9s 1ms/step - loss: 0.0012 - val_loss: 0.0024
```

Testing the Model and Results

```
result_m = ANN_model.evaluate(X_test_m, y_test_m)
accuracy_ANN_m = 1 - result_m
```

```
2814/2814 [==============================] - 2s 599us/step - loss: 0.0027
```

```
print(f"Accuracy: {round((accuracy_ANN_m)*100, 3)}%")
```

```
Accuracy: 99.735%
```

Comparing the two models

```
print(f"Accuracy (StandardScaler) : {round((accuracy_ANN)*100, 3)}%")
print(f"Accuracy (MinMaxScaler)   : {round((accuracy_ANN_m)*100, 3)}%")
```

```
Accuracy (StandardScaler) : 96.96%
Accuracy (MinMaxScaler)   : 99.735%
```

Loss Progress During Different Training

```
# create figure and axes objects for subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

# plot data on the first subplot
ax1.plot(epochs_hist.history['loss'])
ax1.plot(epochs_hist.history['val_loss'])
ax1.set_title('Standard Scalar Training', fontsize=14)
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Training & Valid Loss')
ax1.set_xlim(0,20)
ax1.set_ylim(0,0.06)
ax1.legend(['Training Loss','Valid Loss'])

# plot data on the second subplot
ax2.plot(epochs_hist_m.history['loss'])
ax2.plot(epochs_hist_m.history['val_loss'])
ax2.set_title('MinMax Scalar Training', fontsize=14)
ax2.set_xlabel('Epoch')
ax2.set_ylabel('Training & Valid Loss')
ax2.set_xlim(0,20)
ax2.set_ylim(0,0.06)
ax2.legend(['Training Loss','Valid Loss'])

# display the plot
plt.suptitle('Loss Progress During Different Training', fontsize=18)
plt.legend(['Training Loss','Valid Loss'])
plt.show()

save_plot_as_png(fig,'Loss Progress During Different Training')
```

Fig. 22. Loss Progress During Different Training

Loss progress is a crucial component of deep learning since it enables us to gauge how effectively the model is assimilating the data. The objective of deep learning model training is to minimize the loss function, which measures how well the model can predict the output from a given input. We can improve the model's performance and identify problems by adjusting the model's architecture, hyperparameters, or training data. Loss progress can also be used to assist us to decide when to stop training the model because it can show when the model is performing at its best and when more training could result in overfitting.

Predicting the Model

```
# Predicting the model
y_predict_m = ANN_model.predict(X_test_m)

#Plotting the model
plt.plot(figsize=(15, 8))
plt.plot(y_test_m, y_predict_m, "o", color = 'xkcd:sky blue', alpha = 0.1)
x = np.linspace(*plt.xlim())
plt.plot(x, x, color='red', linestyle='--')
plt.xlabel('Model Predictions')
plt.ylabel('True Values')
plt.title('True Value v/s Predicted Value (Scaled)')
```

```
2814/2814 [==============================] - 2s 518us/step
```

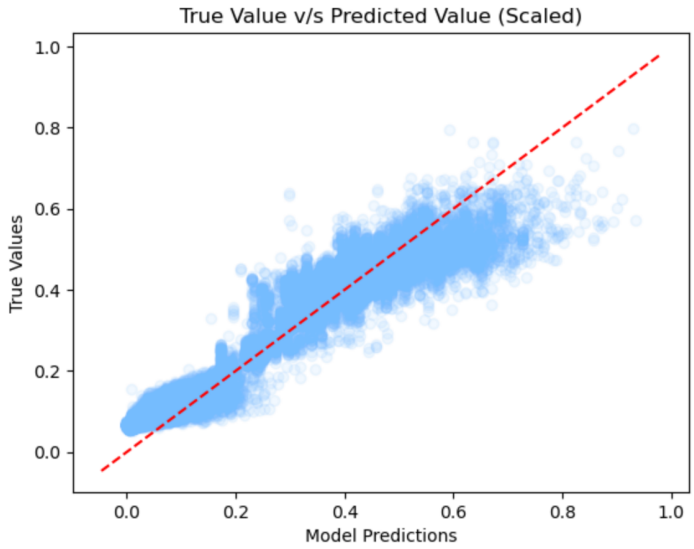


Fig. 22. True Value v/s Predicted Value (Scaled)

Inversing the results for the prediction and getting the final results

```
# Inversing the scaled data
y_predict_orig_m = minmax_y.inverse_transform(y_predict_m)
y_test_orig_m = minmax_y.inverse_transform(y_test_m)
```

```
# Results of Deep Learning Model
score = round((accuracy_ANN_m)*100, 3)
mse = round(mean_squared_error(y_test_orig_m, y_predict_orig_m), 3)
rmse = round(np.sqrt(mean_squared_error(y_test_orig_m, y_predict_orig_m)), 3)
r2 = round(r2_score(y_test_orig_m, y_predict_orig_m), 3)
mae = mean_absolute_error(y_test_orig_m, y_predict_orig_m)

#Getting results into Dataframe
result = pd.DataFrame({
        'Model Name': 'Artificial Neural Network',
        'Accuracy(%)': [score],
        'Mean Squared Error (MSE)': [mse],
        'Root Mean Squared Error (RMSE)': [rmse],
        'R-squared (R2-Score)': [r2],
        'Mean Absolute Error (MAE)': [mae]
    })
```

Results of All Models

	Model Name	Accuracy(%)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)	R-squared (R2-Score)	Mean Absolute Error (MAE)
0	Linear Regression	90.950	4.657037e+07	6824.249	0.909	4537.826712
1	Decision Tree - Best Tuned	98.210	9.205503e+06	3034.057	0.982	1179.342641
2	Decision Tree - Basic Tuned	94.230	2.970222e+07	5449.974	0.942	3201.268597
3	Random Forest	98.460	7.902766e+06	2811.186	0.985	1330.760183
4	Artificial Neural Network	99.735	7.700000e-02	0.277	0.924	0.242688

Conclusion

The Random Forest and Artificial Neural Network models have demonstrated very high accuracy scores, with the Random Forest model achieving an accuracy of 98.4% and the Artificial Neural Network model achieving an accuracy of 99.7%, according to the evaluation of three models for the prediction of flight prices. With a 98.2% accuracy rate, the Decision Tree model also performed well. The Artificial Neural Network model, which had the greatest accuracy score out of the three, performed the best overall. It's crucial to remember that the Random Forest model also scored highly for accuracy and can be a wise choice if model interpretability is a concern. In contrast, the Decision Tree model, which performed nearly as well as the Random Forest model but is simpler to comprehend and interpret, would be a useful option if a simpler model is desired.

It is significant to note that there are other aspects to consider when selecting a model for predicting travel prices in addition to accuracy scores. While choosing the appropriate model for a particular task, additional considerations including model complexity, computation time, and interpretability should also be taken into consideration.

Future Scope

Future research should concentrate on feature engineering, ensemble approaches, time series analysis, deep learning techniques, user-specific models, and external data sources to increase the precision and usefulness of flight price-prediction models. Incorporating more pertinent factors, such as weather information, fuel prices, or holidays, into the process of predicting flight pricing is known as feature engineering. The predictions of various models are combined using ensemble methods. Analyzing time series entails capturing the temporal dynamics of events. Deep learning approaches entail identifying more intricate data patterns.

User-specific models could increase prediction accuracy and offer more specialized recommendations. User evaluations and social media data are examples of external data sources that can offer additional insights into the variables that affect flight prices. We can create more precise and individualized models that better serve the interests of travellers and the travel industry by investigating these areas.

References

K. Tziridis, T. Kalampokas, G. A. Papakostas and K. I. Diamantaras, "Airfare prices prediction using machine learning techniques," 2017 25th European Signal Processing Conference (EUSIPCO), Kos, Greece, 2017, pp. 1036-1039, doi: 10.23919/EUSIPCO.2017.8081365

J. Abdella, NM Zaki, K. Shuaib, F. Khan, “Airline ticket price and demand prediction: A survey,”
Journal of King Saud University - Computer and Information Sciences, Volume 33, Issue 4,2021, Pages 375-391, ISSN 1319-1578, doi: 10.1016/j.jksuci.2019.02.001

Wang, Tianyi & Pouyanfar, Samira & Tian, Haiman & Tao, Yudong & Alonso, Miguel & Luis, Steven & Chen, Shu-Ching. (2019). A Framework for Airfare Price Prediction: A Machine Learning Approach. 200-207. 10.1109/IRI.2019.00041

Supriya Rajankar, Neha Sakharkar, 2019, A Survey on Flight Pricing Prediction using Machine Learning, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 08, Issue 06 (June 2019)

Dataset: https://www.kaggle.com/datasets/shubhambathwal/flight-price-prediction