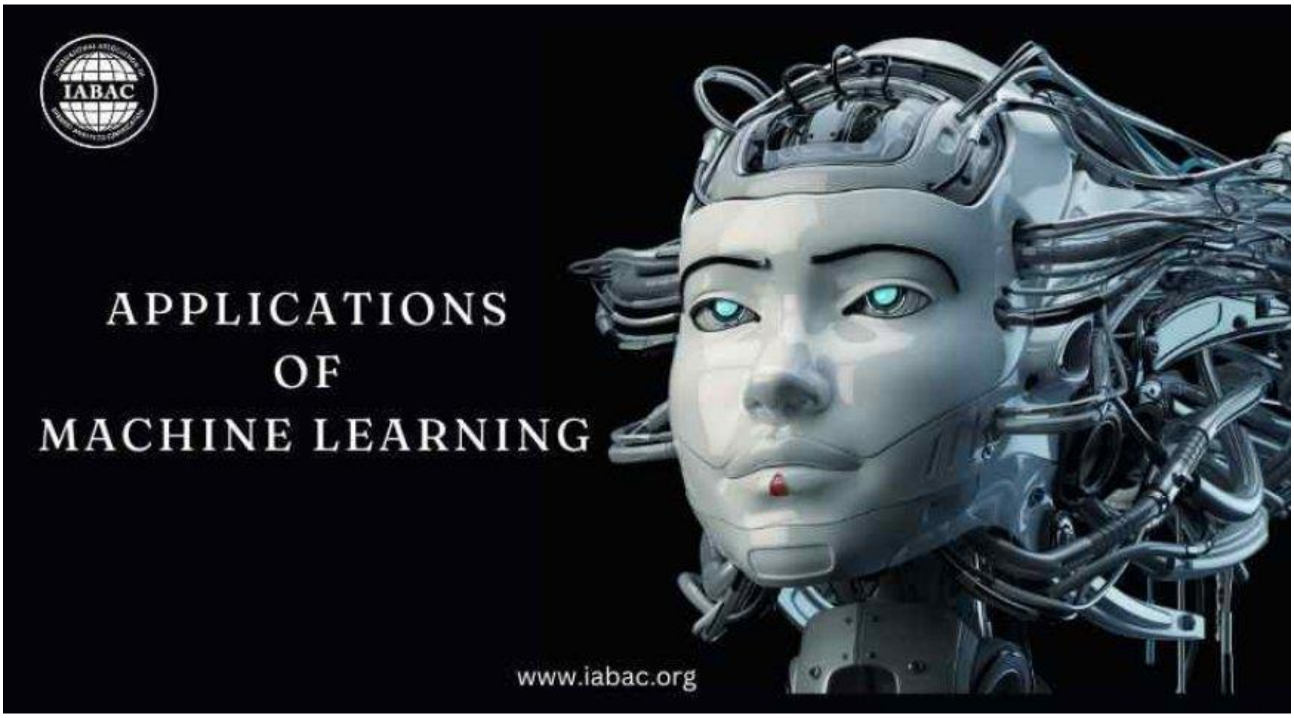
MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

ASSESSMENT 2



Nayan Maharjan

Contents

List of Figures

Introduction

Problem Statement

The main aim of this project is to develop a machine learning model that can predict the number of flights (All_Flights) for international routes which are operated by various Australian airlines from Australian cities. Accurately predicting the flight numbers can help the airlines and airports to optimize their operations, improve scheduling, and enhance overall efficiency.

Data Description

The dataset contains 16 columns and 110055 numbers of rows.

- _id ---> unique number
- Month ---> Date in month and day
- In_Out ---> Status of flight [Incoming/Outgoing]
- Australian_City ---> Australian city name
- International_City ---> International city name
- Airline ---> Airline owning the flight
- Route ---> Route taken by the flight
- Port_Country ---> Port country
- Port_Region ---> Port region
- Service_Country ---> Service country
- Service_Region ---> Service region
- Stops ---> Number of stops taken by the flight
- All_Flights ---> Total number of flights
- Max_Seats ---> Total capacity of seats in flight
- Year ---> Date in year
- Month_num ---> Date in month number

Desired Algorithm

The algorithm for predicting the number of flights that have been used for this project are

Linear Regression

Linear regression is the straightforward approach model which deals with one dependent variable and one or more independent variable by fitting a linear equation (George A. F. Seber, 2012). This model was used because of the simplicity and interpretability that can provide the benchmark against the other complex model for the comparison.

Random Forest Regressor

It is the model that creates the numbers of decision tree during the training and merges their outputs for the better accuracy and preventing overfitting (Rigatti, 2017). It is useful for handling all the non-linear relationships and interaction between the features.

Decision Tree Regressor

This model splits the data into subsets based on feature values, creating a tree-like model of decisions (Xu, 2005). It was used because of its ability to model complex relationships without requiring data scaling, making it more easier to interpret.

XGBoost Regressor

Extreme Gradient Boosting is an advance boosting algorithm that builds the trees sequentially and each new tree correcting errors made by the previous ones (Dong, 2022). It is very efficient and capable of handling missing data, and often outperforms other algorithms in terms of predictive accuracy.

Description of Steps

Data Cleaning and Pre-Processing

In this step simply data was read using pandas and checked for the missing values as well as converted the “Month” column to datetime format.

```python
# Data Cleaning

import pandas as pd

# Load the dataset and read the data set
flights_data = pd.read_csv("Flights.csv")

# Display the first few rows of the dataset to understand its structure
print("Initial Data Preview:")
print(flights_data.head())

# Check the basic information about the dataset (data types, missing values, etc.)
print("\nDataset Information:")
print(flights_data.info())

# Check for missing values in the dataset
missing_values = flights_data.isnull().sum()
print("\nMissing values in each column:")
print(missing_values)

# Convert the 'Month' column to datetime format
# Assuming 'Month' is in the format like 'Sep-03' which means September 2003
flights_data['Month'] = pd.to_datetime(flights_data['Month'], format='%b-%y')

# Check the changes after converting 'Month' to datetime
print("\nData Preview After Converting 'Month' to Datetime:")
print(flights_data.head())

# Ensure that the 'Month' conversion was successful
print("\nDataset Information After Date Conversion:")
print(flights_data.info())
```

Figure 1: Code for cleaning and Pre-Processing

```
...  Initial Data Preview:
        _id   Month In_Out Australian_City International_City  \
     0    1  Sep-03      I        Adelaide          Denpasar
     1    2  Sep-03      I        Adelaide         Hong Kong
     2    3  Sep-03      I        Adelaide      Kuala Lumpur
     3    4  Sep-03      I        Adelaide         Singapore
     4    5  Sep-03      I        Adelaide         Singapore

                      Airline            Route     Port_Country Port_Region  \
     0       Garuda Indonesia      DPS-ADL-MEL        Indonesia     SE Asia
     1 Cathay Pacific Airways      HKG-ADL-MEL  Hong Kong (SAR)     NE Asia
     2      Malaysia Airlines          KUL-ADL         Malaysia     SE Asia
     3         Qantas Airways  SIN-DRW-ADL-MEL        Singapore     SE Asia
     4         Qantas Airways  SIN-DRW-ADL-SYD        Singapore     SE Asia

        Service_Country Service_Region  Stops  All_Flights  Max_Seats  Year  \
     0       Indonesia        SE Asia      0           13       3809  2003
     1 Hong Kong (SAR)        NE Asia      0            8       2008  2003
     2        Malaysia        SE Asia      0           17       4726  2003
     3       Singapore        SE Asia      1            4        908  2003
     4       Singapore        SE Asia      1            9       2038  2003

        Month_num
     0          9
     1          9
     ...
      15  Month_num          110055 non-null  int64
     dtypes: datetime64[ns](1), int64(6), object(9)
     memory usage: 13.4+ MB
     None
     Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 2: Result of the cleaning and Pre-Processing codes

Exploratory Data Analysis (EDA)

In this step dataset was analysed for their characteristics and relationships within.

Summary Statistics of Numerical Features

It contains summary statistics of numerical variables such as All_Flights and Max_Seats to gain insights into their distributions and identify potential outliers.

```python
# Exploratory Data Analysis (EDA) - Adjusted for Correlation Matrix

import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for seaborn plots
sns.set(style="whitegrid")

# 1. Overview of Numerical Features
# Summary statistics of numerical features
print("\nSummary Statistics of Numerical Features:")
print(flights_data.describe())
```

Figure 3: Code for overview of numerical features

```
...
Summary Statistics of Numerical Features:
                 _id                          Month          Stops  \
count  110055.000000                         110055  110055.000000
mean    55028.000000  2013-03-08 19:15:00.564263168       0.162464
min         1.000000            2003-09-01 00:00:00       0.000000
25%     27514.500000            2009-03-01 00:00:00       0.000000
50%     55028.000000            2013-03-01 00:00:00       0.000000
75%     82541.500000            2017-04-01 00:00:00       0.000000
max    110055.000000            2022-09-01 00:00:00       3.000000
std     31770.286275                            NaN       0.388295

         All_Flights      Max_Seats           Year      Month_num
count  110055.000000  110055.000000  110055.000000  110055.000000
mean       24.775367    6610.760910    2012.726573       6.511762
min         0.000000       0.000000    2003.000000       1.000000
25%        12.000000    2461.000000    2009.000000       3.000000
50%        21.000000    4928.000000    2013.000000       6.000000
75%        31.000000    9018.000000    2017.000000       9.000000
max       178.000000   52596.000000    2022.000000      12.000000
std        21.450937    6197.412623       4.817944       3.472462
```

Figure 4: Result of overview of numerical features

Distribution Plots

Histogram of All_Flights displayed the distribution of the All_Flights variable to understand its spread and skewness. Histogram of Max_Seats displays the distibutio of maximum number of seats across the flights.

```python
# 2. Univariate Analysis: Distribution of 'All_Flights'
plt.figure(figsize=(10, 5))
sns.histplot(flights_data['All_Flights'], bins=50, kde=True, color='blue')
plt.title('Distribution of All Flights')
plt.xlabel('Number of Flights')
plt.ylabel('Frequency')
plt.show()

# 3. Univariate Analysis: Distribution of 'Max_Seats'
plt.figure(figsize=(10, 5))
sns.histplot(flights_data['Max_Seats'], bins=50, kde=True, color='green')
plt.title('Distribution of Max Seats')
plt.xlabel('Number of Seats')
plt.ylabel('Frequency')
plt.show()
```

Figure 5: Code for histograms

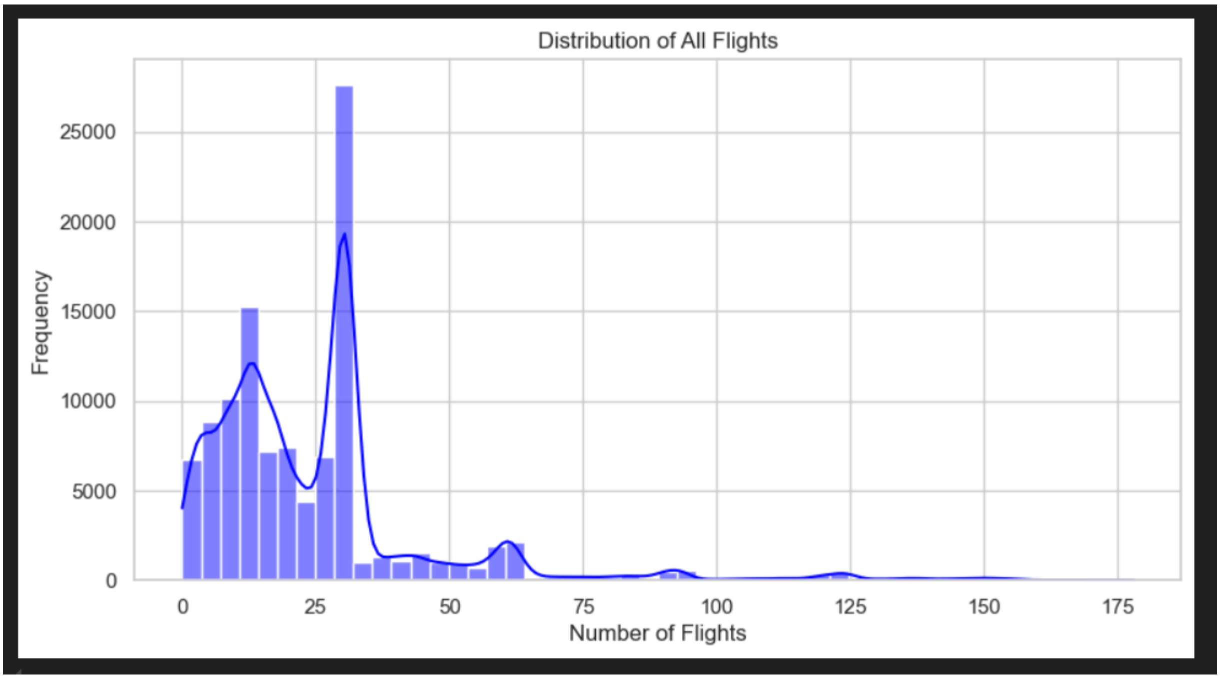


Figure 6: Distribution of flights

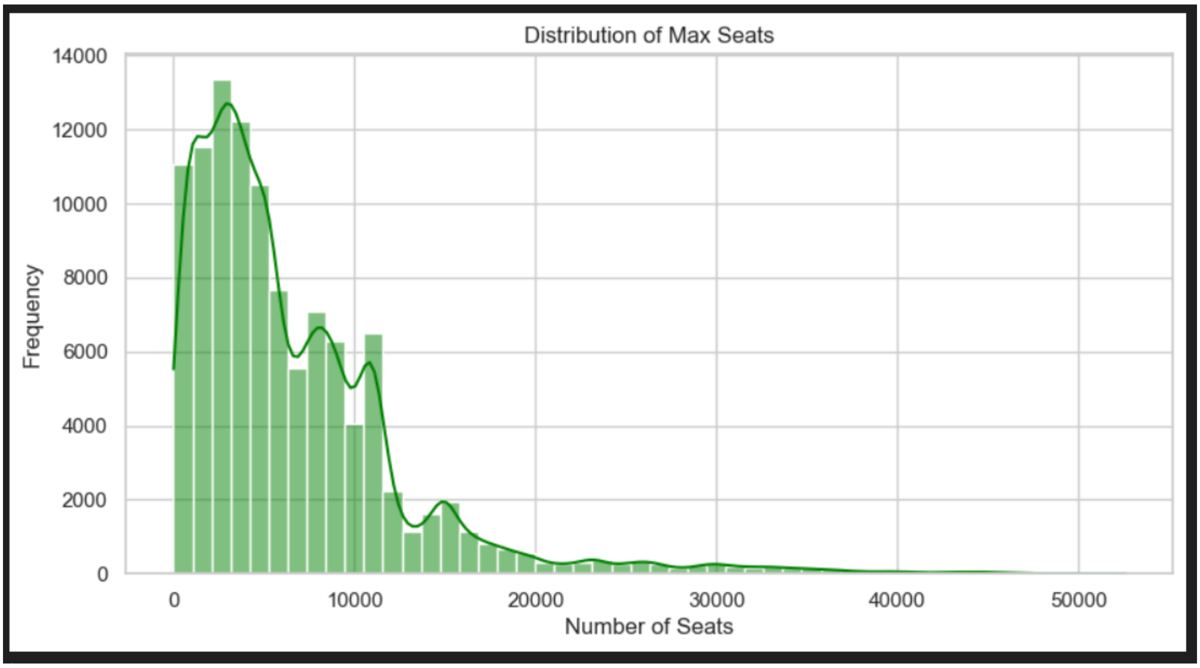


Figure 7: Distribution of Max Seats

Correlation Heatmap

It shows the correlation ship between the All_Flights and Max_Seats.

```
# 4. Bivariate Analysis: Correlation Heatmap
# Select only numeric columns for the correlation matrix
numeric_cols = flights_data.select_dtypes(include=['float64', 'int64'])  # Filter numeric columns

# Compute the correlation matrix
correlation_matrix = numeric_cols.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

Figure 8: Code for Correlation Heatmap

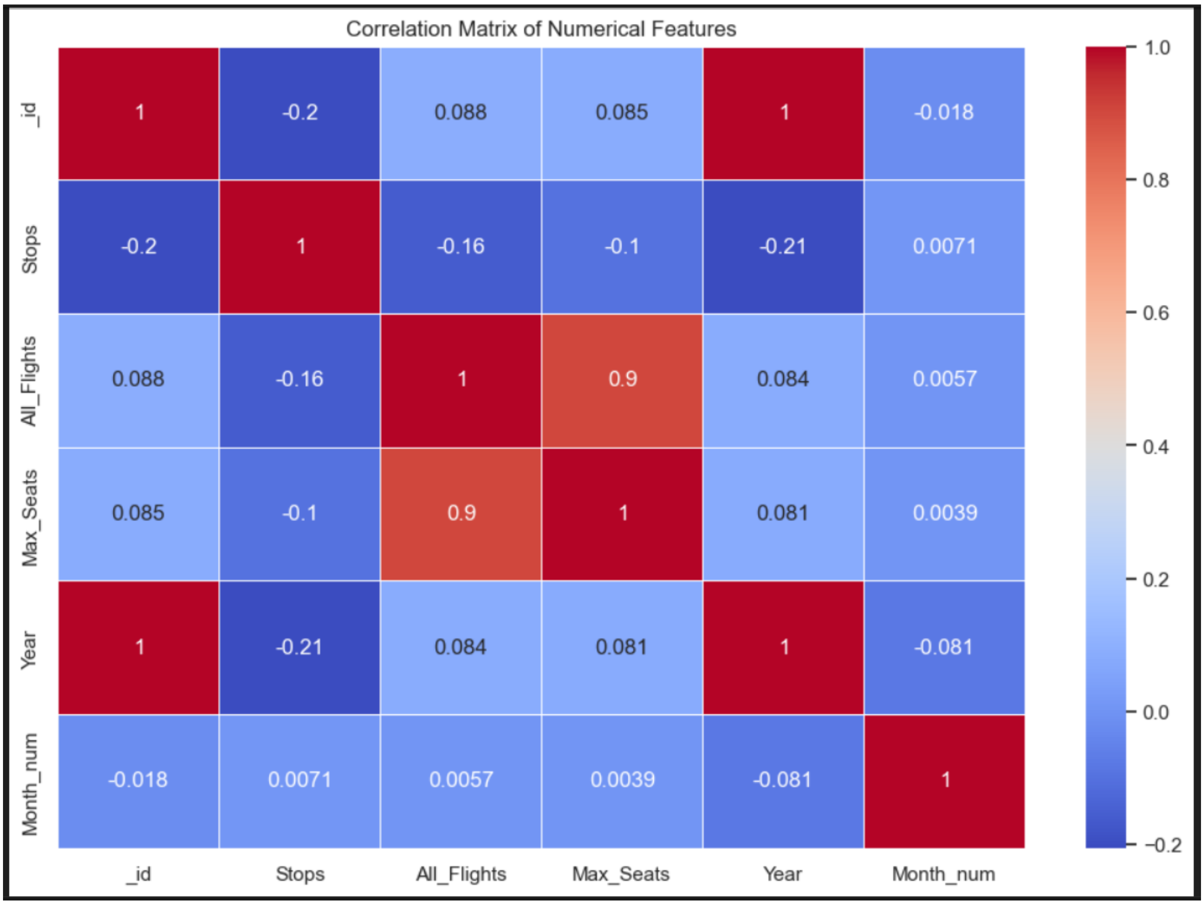


Figure 9: Result of Correlation Heatmap

Scatter Plot – All_Flights vs. Max_Seats

Scatter plot was used to examine the relationship between the number of flights and the seating capacity which provided the visual representation of their correlation.

```
# 5. Bivariate Analysis: Scatter Plot - 'All_Flights' vs 'Max_Seats'
plt.figure(figsize=(10, 6))
sns.scatterplot(x='All_Flights', y='Max_Seats', data=flights_data)
plt.title('All Flights vs. Max Seats')
plt.xlabel('Number of Flights')
plt.ylabel('Number of Seats')
plt.show()
```

Figure 10: Codes for Scatter Plot

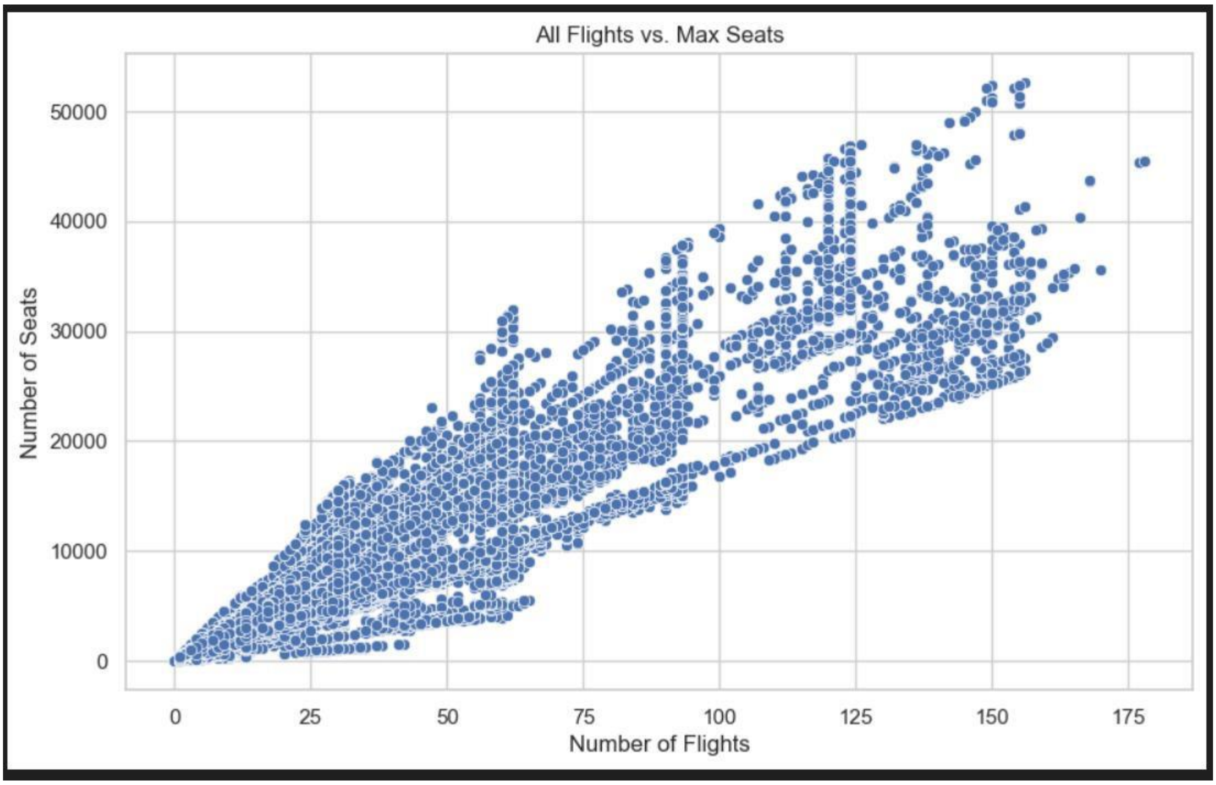


Figure 11: Result of Scatter Plot

Top 10 Airlines by Number of Flights

It demonstrates the top 10 airlines based on the flights.

```
# 6. Analyzing Categorical Variables: Top 10 Airlines by Number of Flights
top_airlines = flights_data['Airline'].value_counts().head(10)
plt.figure(figsize=(12, 6))
sns.barplot(x=top_airlines.values, y=top_airlines.index, palette='viridis')
plt.title('Top 10 Airlines by Number of Flights')
plt.xlabel('Number of Flights')
plt.ylabel('Airline')
plt.show()
```

Figure 12: Codes for the top 10 airlines

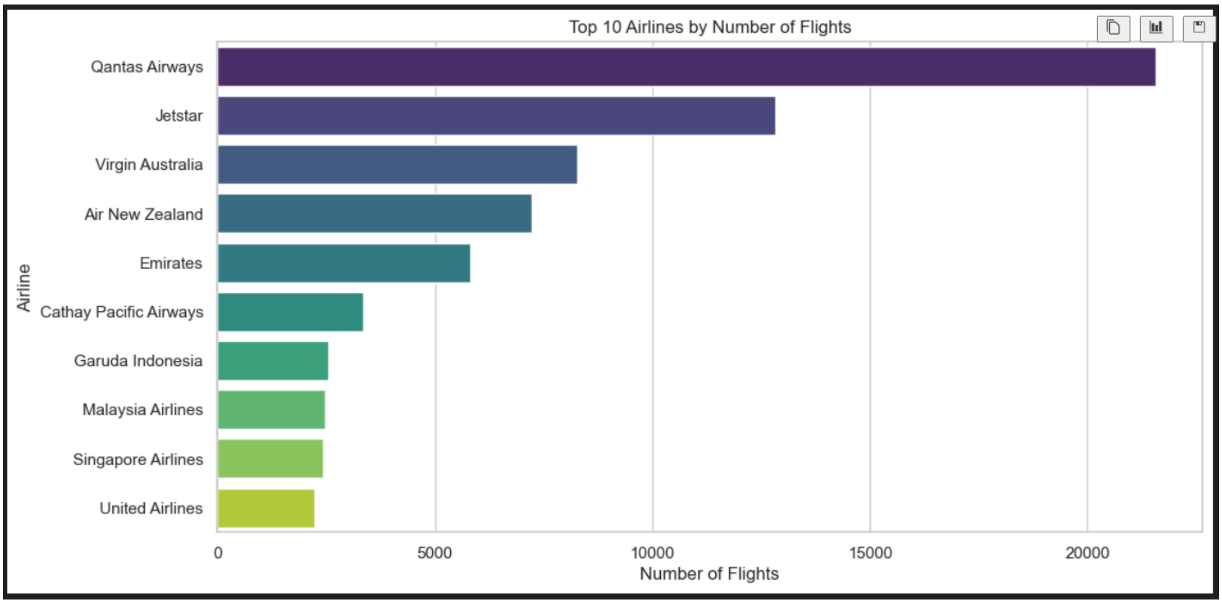


Figure 13: Result of top 10 airlines.

Trend Analysis Over Time

The trend of All_Flights over time was displayed using a line plot, which allowed us to see any patterns or seasonal fluctuations in the number of flights.

```python
# 7. Time Series Analysis: Trend of All Flights Over Time
# Assuming 'Month' has been converted to datetime during data cleaning
plt.figure(figsize=(14, 6))
sns.lineplot(x='Month', y='All_Flights', data=flights_data)
plt.title('Trend of All Flights Over Time')
plt.xlabel('Month')
plt.ylabel('Number of Flights')
plt.xticks(rotation=45)
plt.show()
```

Figure 14: Codes for the Time Series Analysis

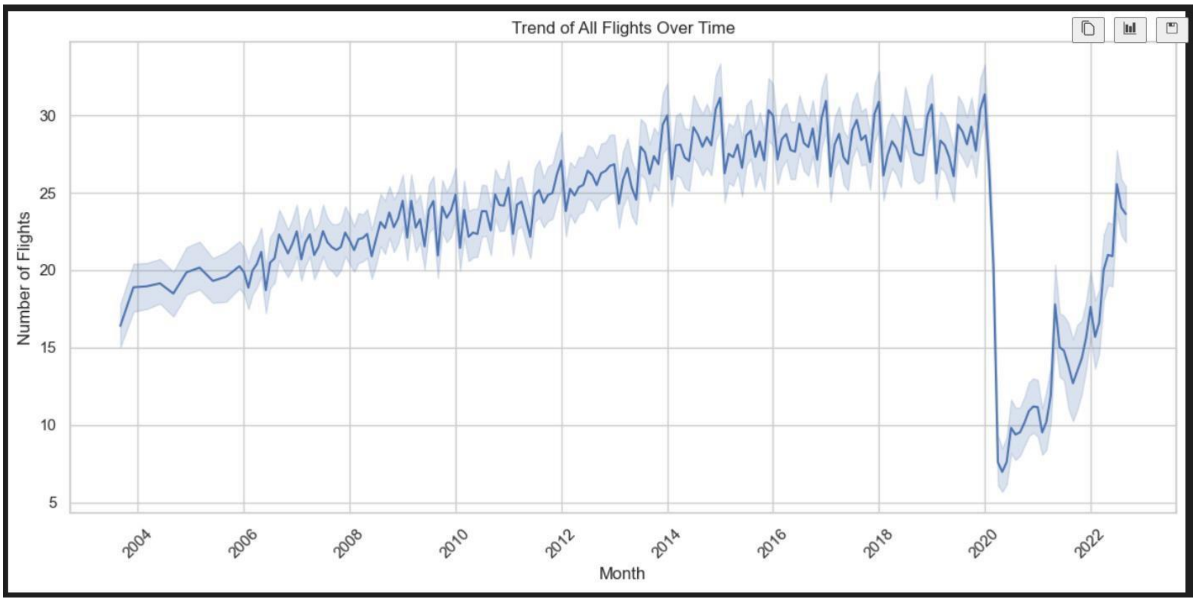


Figure 15: Result of Trend of all flights over Time

Feature Engineering

New features such as Season, Capacity_Utilization_Rate, Airline_Route_Combo was introduced to enrich the data for better results in prediction.

```
# Feature Engineering

#  Route Popularity: Count of total flights for each unique route
flights_data['Route_Popularity'] = flights_data.groupby('Route')['Route'].transform('count')

# Seasonal Indicator: Map months to seasons (e.g., Summer, Autumn, Winter, Spring)
flights_data['Season'] = flights_data['Month'].dt.month.map({
    12: 'Summer', 1: 'Summer', 2: 'Summer',
    3: 'Autumn', 4: 'Autumn', 5: 'Autumn',
    6: 'Winter', 7: 'Winter', 8: 'Winter',
    9: 'Spring', 10: 'Spring', 11: 'Spring'
})

# Capacity Utilization Rate: Ratio of flights to the maximum seat capacity
flights_data['Capacity_Utilization_Rate'] = flights_data['All_Flights'] / flights_data['Max_Seats']

# Airline-Route Interaction: Combine 'Airline' and 'Route' to create a combined feature
flights_data['Airline_Route_Combo'] = flights_data['Airline'] + '_' + flights_data['Route']

# Check the updated dataset with new features
print("Updated Data Preview with New Features:")
print(flights_data.head())
```

Figure 16: Codes for Feature Engineering

Encoding Categorical Data

Majorly two types of encoding were used which are Binary encoding for High-cardinality features and One-Hot Encoding for Low-cardinality features.

```python
# Drop all columns except the ones needed for encoding and normalization
columns_to_keep = ['Airline', 'Route', 'Airline_Route_Combo', 'In_Out', 'Season',
                   'Stops', 'Max_Seats', 'Capacity_Utilization_Rate', 'Month_Num', 'Month_Year', 'All_Flights']
flights_data = flights_data[columns_to_keep]

# Apply Encoding to Categorical Variables
# High-cardinality features for Binary Encoding
high_cardinality_vars = ['Airline', 'Route', 'Airline_Route_Combo']

# Apply Binary Encoding to high-cardinality categorical variables
binary_encoder = BinaryEncoder(cols=high_cardinality_vars, drop_invariant=True)
flights_data_encoded = binary_encoder.fit_transform(flights_data)

# Low-cardinality features for One-Hot Encoding
low_cardinality_vars = ['In_Out', 'Season']

# Apply One-Hot Encoding to low-cardinality categorical variables
flights_data_encoded = pd.get_dummies(flights_data_encoded, columns=low_cardinality_vars, drop_first=True)

# Print the columns to see the encoded column names
print("Columns after encoding:")
print(flights_data_encoded.columns)
```

Figure 17: Codes for Encoding for Categorical Data

Standardization and Feature Selection

All the numerical features were standardized so that each numerical feature has a mean of 0 and a standard deviation of 1. The features were selected manually for this project.

```python
# Standardize Numerical Features
numerical_features = ['Stops', 'Max_Seats', 'Capacity_Utilization_Rate', 'Month_Num', 'Month_Year']

# Replace infinite values with NaN and fill them if necessary
flights_data_encoded.replace([np.inf, -np.inf], np.nan, inplace=True)
flights_data_encoded[numerical_features] = flights_data_encoded[numerical_features].fillna(flights_data_encoded[numerical_features].mean())

# Initialize the StandardScaler
scaler = StandardScaler()

# Apply standardization to numerical features
flights_data_encoded[numerical_features] = scaler.fit_transform(flights_data_encoded[numerical_features])

# Define Features and Target Variable
# Assuming 'All_Flights' is the target variable
X = flights_data_encoded.drop(columns=['All_Flights'])
y = flights_data_encoded['All_Flights']

# Manually select features to be used based on actual column names after encoding
# Get the list of columns after encoding
encoded_columns = list(X.columns)

# Define the selected features manually
# Include numerical features directly
selected_features = numerical_features.copy()

# Include encoded feature columns
selected_features += [col for col in encoded_columns if col.startswith('Airline') or
                                                        col.startswith('Route') or
                                                        col.startswith('Airline_Route_Combo') or
                                                        col.startswith('In_Out') or
                                                        col.startswith('Season')]
```

Figure 18: Code for standardization and feature selection

Split of Data

The refined data after encoding and normalization are separated in 70% train and 30% test for the model training.

```python
# Ensure that the selected features are present in the dataset
X = X[selected_features]

# Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Final check of data shapes
print(f"Training data shape: {X_train.shape}, Testing data shape: {X_test.shape}")
```

Figure 19: Code for Splitting the Dataset

Result Output

Linear Regression

The result of linear regression states the value of R^2 is 0.8744 which means

Accuracy = 87.44%

```python
# 1. Linear Regression with Ridge Regularization
lr = Ridge(alpha=1.0)  # L2 regularization parameter
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

mae_lr = mean_absolute_error(y_test, y_pred_lr)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("\nLinear Regression (Ridge) Evaluation Metrics:")
print(f"MAE: {mae_lr:.4f}, MSE: {mse_lr:.4f}, R²: {r2_lr:.4f}")
```

Figure 20: Code for Linear Regression

```
Linear Regression (Ridge) Evaluation Metrics:
MAE: 4.5179, MSE: 56.4038, R²: 0.8744
```

Figure 21: Result of Linear Regression

Random Forest Regressor

The output of this model is $R^2 = 0.9988$ which means

Accuracy = 99.88%

```python
# 2. Random Forest Regressor with Depth Limitation
rf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print("\nRandom Forest Regressor Evaluation Metrics:")
print(f"MAE: {mae_rf:.4f}, MSE: {mse_rf:.4f}, R²: {r2_rf:.4f}")
```

Figure 22: Codes for Random Forest Regressor

```
Random Forest Regressor Evaluation Metrics:
MAE: 0.4437, MSE: 0.5480, R²: 0.9988
```

Figure 23: Result of Random Forest Regressor

Decision Tree Regressor

The result of this model is $R^2 = 0.9570$ which means that

Accuracy = 95.70%

```python
# 3. Decision Tree Regressor with Pruning
dt = DecisionTreeRegressor(max_depth=5, min_samples_split=10, random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

mae_dt = mean_absolute_error(y_test, y_pred_dt)
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print("\nDecision Tree Regressor Evaluation Metrics:")
print(f"MAE: {mae_dt:.4f}, MSE: {mse_dt:.4f}, R²: {r2_dt:.4f}")
```

Figure 24: Codes For Decision Tree Regressor

```
Decision Tree Regressor Evaluation Metrics:
MAE: 2.5831, MSE: 19.2992, R²: 0.9570
```

Figure 25: Result of Decision Tree Regressor

XGBoost Regressor

The result of the XGBoost Regressor is $R^2 = 0.9997$ which means

Accuracy = 99.97%

```python
# 4. XGBoost with Hyperparameter Tuning and Cross-Validation
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.6, 0.8, 1.0]
}

# Initialize GridSearchCV with XGBoost
grid_search_xgb = GridSearchCV(
    XGBRegressor(random_state=42, eval_metric='rmse'),
    param_grid_xgb,
    cv=5,
    scoring='neg_mean_squared_error'
)

# Fit the GridSearchCV to find the best parameters
grid_search_xgb.fit(X_train, y_train)

# Extract the best estimator from the grid search
best_xgb = grid_search_xgb.best_estimator_

# Print the best parameters found by GridSearchCV
print(f"\nBest parameters for XGBoost: {grid_search_xgb.best_params_}")

# Evaluate the best model on the test set
y_pred_best_xgb = best_xgb.predict(X_test)

# Calculate evaluation metrics for the best XGBoost model
mae_xgb = mean_absolute_error(y_test, y_pred_best_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_best_xgb)
r2_xgb = r2_score(y_test, y_pred_best_xgb)

print("\nBest XGBoost Regressor Evaluation Metrics:")
print(f"MAE: {mae_xgb:.4f}, MSE: {mse_xgb:.4f}, R²: {r2_xgb:.4f}")
```

Figure 26: codes for XGBosst

```
Best XGBoost Regressor Evaluation Metrics:
MAE: 0.1097, MSE: 0.1544, R²: 0.9997
```

Figure 27: Result of XGBoost

Conclusion

XGBoost and Random Forest effectively predicted the flight numbers, while simpler models performed reasonably well but less accurately.

References

Dong, J., 2022. A neural network boosting regression model based on XGBoost. *Applied Soft Computing,* 125(1), p. np.

George A. F. Seber, A. J. L., 2012. *Linear Regression Analysis.* Auckland: John Wiley & Sons.

Rigatti, S. J., 2017. Random Forest. *Journal of Insurance Medicine,* 47(1), p. 31–39.

Xu, M., 2005. Decision tree regression for soft classification of remote sensing data. *Remote Sensing of Environment,* 97(3), pp. 322-336.