Advanced Flight Departure Delay Analysis Project

Hussain Ahmad

22I-1374

AI-C

AI-3002: Machine Learning  
Final Project  
Due Date: December 12, 2024, 11:55 PM

# Table of Contents

1. **Introduction  
   2. Problem Statement  
   3. Objective  
   4. Data Preprocessing and Feature Engineering  
     • Data Integration  
     • Data Cleaning and Transformation  
     • Feature Engineering  
   5. Exploratory Data Analysis (EDA)  
     • Visualizations  
     • Correlation Analysis  
     • Comparison of Training and Testing Datasets  
   6. Analytical and Predictive Tasks  
     • Binary Classification  
     • Multi-Class Classification  
     • Regression Analysis  
   7. Model Optimization and Evaluation  
     • Hyperparameter Tuning  
     • Validation  
     • Model Comparison  
   8. Model Testing and Kaggle Submission  
     • Predictions on Test Dataset  
     • Submission to Kaggle  
   9. Conclusion  
   10. Assumptions  
   11. References  
   12. Attachments**

# 1. Introduction

Flight departure delays pose significant challenges to the aviation industry, impacting passenger satisfaction, airline operations, and overall efficiency. Predicting these delays can help airlines optimize their operations, improve customer service, and reduce operational costs. This project focuses on analyzing flight departure delays using various machine learning techniques to identify patterns and build predictive models.

# 2. Problem Statement

Flight departure delays are a critical challenge in the aviation industry. Such delays affect passenger satisfaction, airline operations, and overall efficiency. You are provided with raw Excel files (test, train, and weather data) and are tasked with calculating departure delays. Using these datasets, you will analyze delay patterns and build predictive models to identify key factors contributing to delays.

# 3. Objective

The main goal is to predict departure delays for flights in the test dataset. The project objectives are as follows:  
  
**1. Data Analysis:**  
 - Analyze the training, testing, and weather datasets.  
  
**2. Model Building:**  
 - Develop predictive models based on the training data.  
  
**3. Prediction Generation:**  
 - Generate predictions for the test data.  
  
**4. Kaggle Submission:**  
 - Submit the predictions to a Kaggle competition for evaluation.

# 4. Data Preprocessing and Feature Engineering

## • Data Integration

The datasets provided include:  
  
- 71 DOC files containing train and test flight data.  
- 26 XLSV files, out of which 13 are relevant, containing weather data.  
  
The initial step involved converting the DOC files into structured tables using Python. After conversion, the datasets underwent data cleaning to handle inconsistencies, missing values, and formatting issues.  
  
**- Flight Data Concatenation:**  
 The flight data from the 71 DOC files was concatenated into a single large dataset. This dataset included various features such as departure and arrival times, airline information, flight numbers, and delay durations.  
  
**- Weather Data Integration:**  
 The weather data, though significantly smaller with approximately 380 rows, provided critical information like temperature, wind speed, and humidity. This data was merged with the flight data based on temporal and spatial attributes to enrich the dataset for analysis.

## • Data Cleaning and Transformation

**1. Handling Missing Values**  
 Missing values were addressed using appropriate imputation techniques:  
 **- Numeric Features:** Filled with the mean or median values.  
 **- Categorical Features:** Filled with the mode or a placeholder category like "Unknown."  
  
**2. Formatting Time Fields:**  
 Time-related fields such as `departure.scheduledTime`, `arrival.scheduledTime`, `departure.actualTime`, and `arrival.actualTime` were converted to datetime objects using `pd.to\_datetime()`. This standardization facilitated accurate delay calculations and feature extraction.

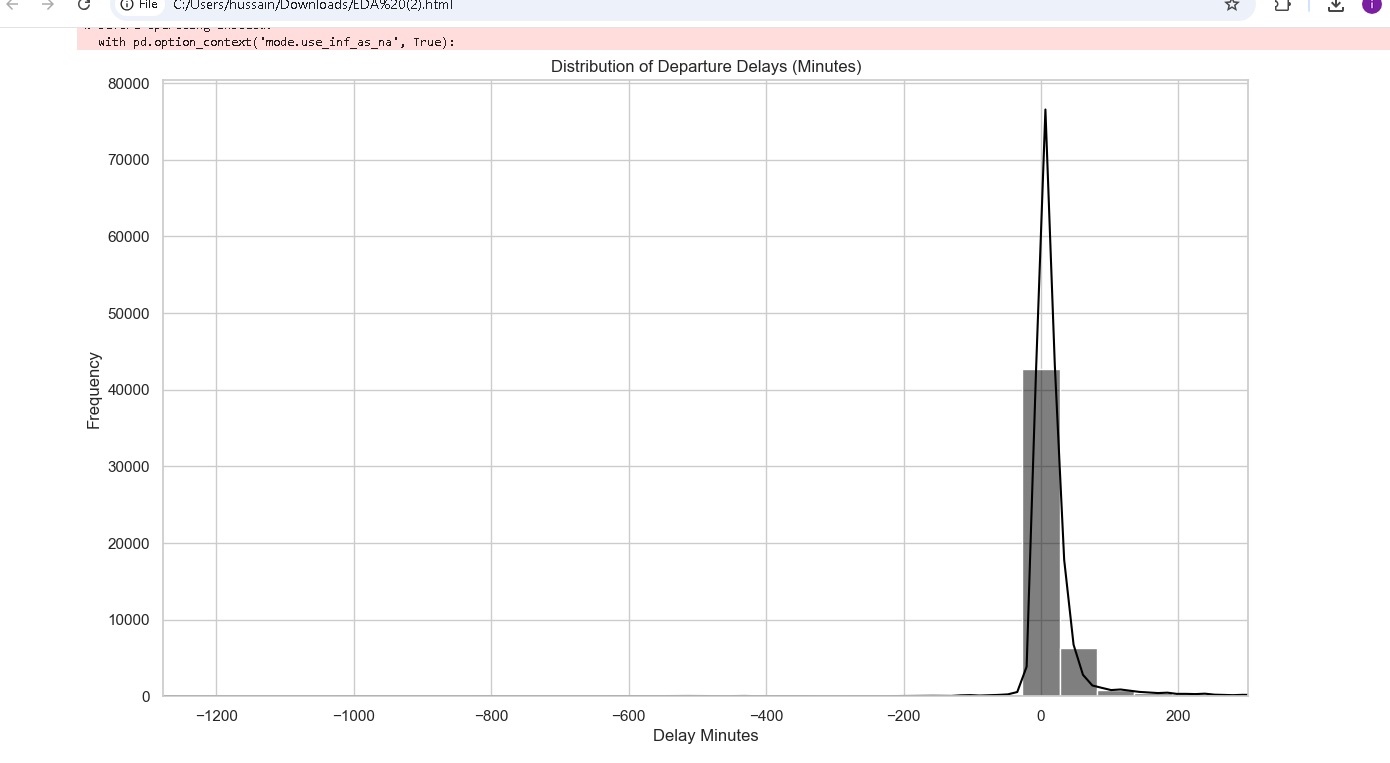
## • Feature Engineering

**1. Calculating Departure Delay:**  
 The departure delay was calculated by subtracting the scheduled departure time from the actual departure time:  
  
 ```python  
 train\_df['departure.delay\_minutes'] = (train\_df['departure.actualTime'] - train\_df['departure.scheduledTime']).dt.total\_seconds() / 60  
 ```  
**2. Merging Weather Data:**  
 Relevant weather features like temperature, wind speed, and humidity were extracted and merged with the flight data based on the departure airport and the corresponding date.  
  
**3. Extracting Temporal Features:**  
 Additional temporal features were derived to capture time-based patterns:  
 - Day of the Week: Extracted using `.dt.dayofweek`.  
 - Hour of the Day:Extracted using `.dt.hour`.  
 - Month of the Year: Extracted using `.dt.month`.  
  
**4. Encoding Categorical Variables:**  
 Categorical variables such as airline names and days of the week were one-hot encoded to convert them into numerical representations suitable for machine learning models.  
  
**5. Handling High Cardinality:**  
 To address the high cardinality of categorical variables (e.g., numerous airline names), rare categories were grouped into an "Other" category to reduce dimensionality and mitigate the memory issues encountered during one-hot encoding.

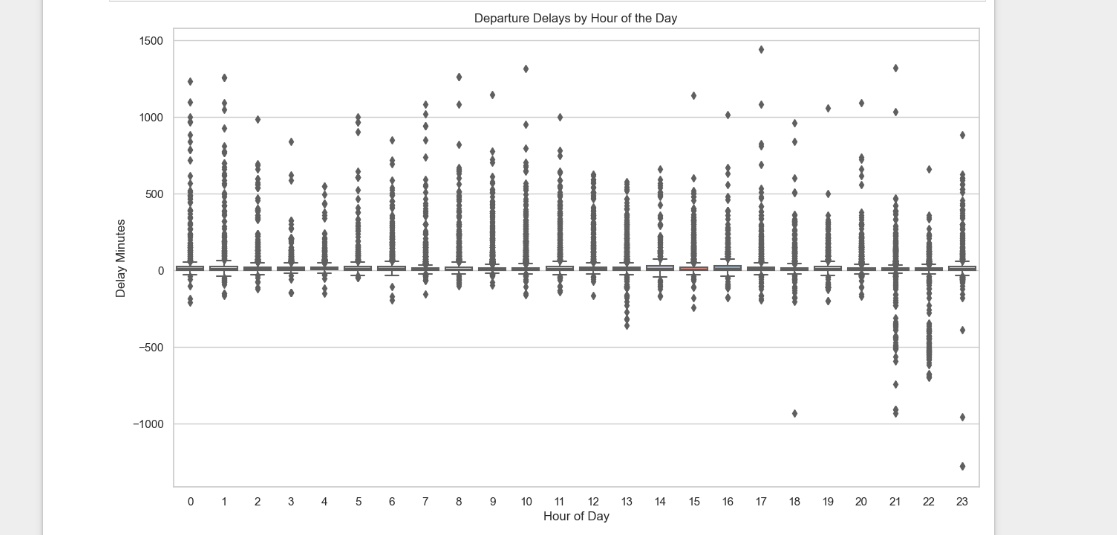
# 5. Exploratory Data Analysis (EDA)

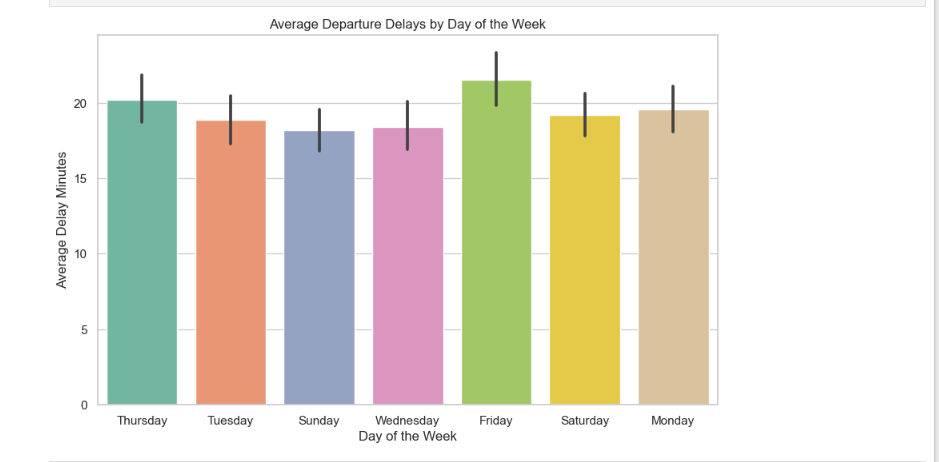
## • Visualizations

**1. Delay Distributions:**  
 A histogram was created to visualize the distribution of delay durations.  
  
 ```python  
 plt.figure(figsize=(10,6))  
 sns.histplot(train\_df['departure.delay\_minutes'], bins=50, kde=True)  
 plt.title('Distribution of Departure Delays')  
 plt.xlabel('Delay Minutes')  
 plt.ylabel('Frequency')  
 plt.show()  
 ```

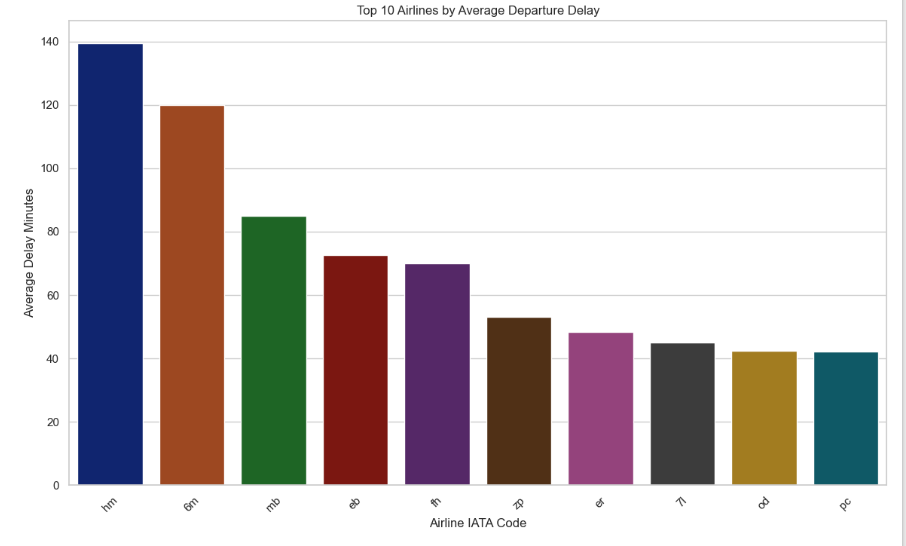


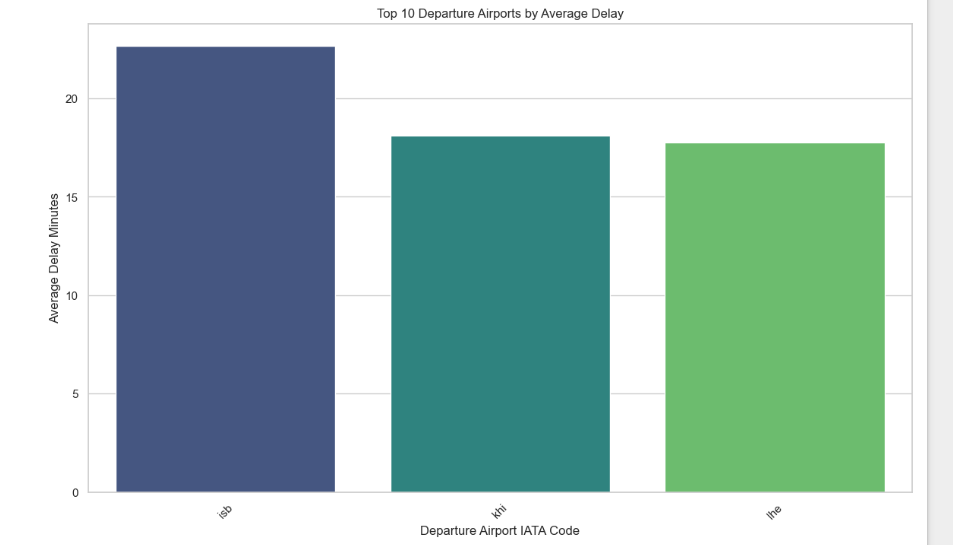
**2. Temporal Analysis:**  
 Bar charts were used to analyze delays across different hours of the day, days of the week, and months.  
 ```python  
 # Delays by Hour  
 plt.figure(figsize=(12,6))  
 sns.barplot(x='departure\_hour', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Average Departure Delay by Hour of Day')  
 plt.xlabel('Hour of Day')  
 plt.ylabel('Average Delay (Minutes)')  
 plt.show()  
  
 # Delays by Day of Week  
 plt.figure(figsize=(12,6))  
 sns.barplot(x='departure\_day\_of\_week', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Average Departure Delay by Day of Week')  
 plt.xlabel('Day of Week')  
 plt.ylabel('Average Delay (Minutes)')  
 plt.show()

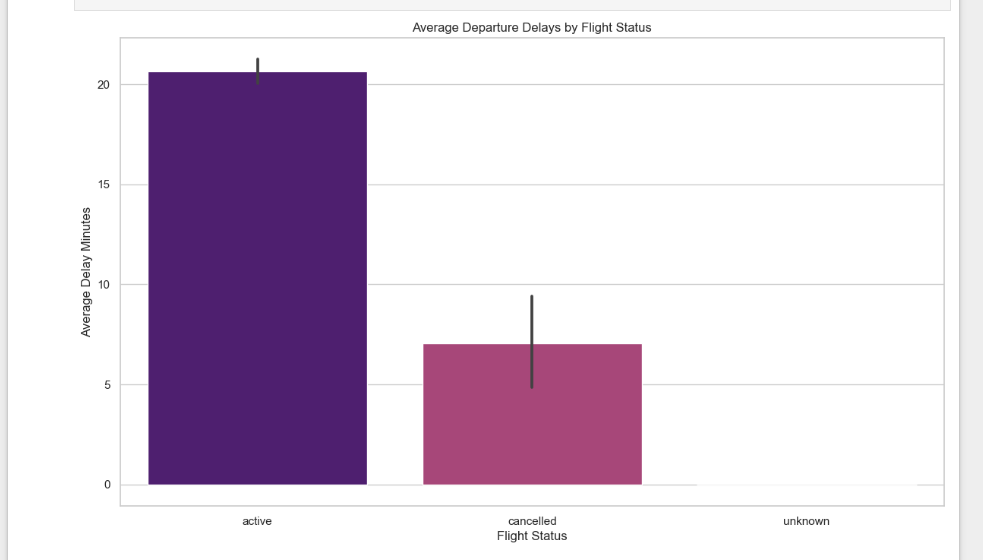




**3. Category-Wise Analysis:**  
 Grouped delays by airline and departure airport to identify patterns.  
  
 ```python  
 # Delays by Airline  
 plt.figure(figsize=(20,10))  
 sns.boxplot(x='airline.iataCode', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Departure Delays by Airline')  
 plt.xlabel('Airline')  
 plt.ylabel('Delay Minutes')  
 plt.xticks(rotation=90)  
 plt.show()  
  
 # Delays by Departure Airport  
 plt.figure(figsize=(20,10))  
 sns.boxplot(x='departure.iataCode', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Departure Delays by Departure Airport')  
 plt.xlabel('Departure Airport')  
 plt.ylabel('Delay Minutes')  
 plt.xticks(rotation=90)  
 plt.show()  
 ```

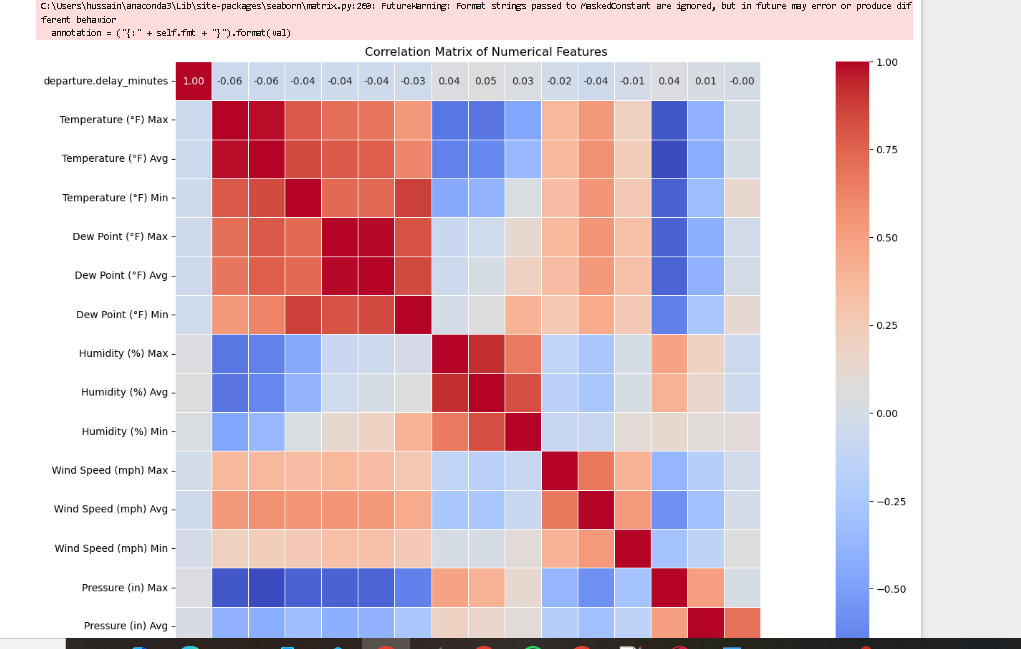


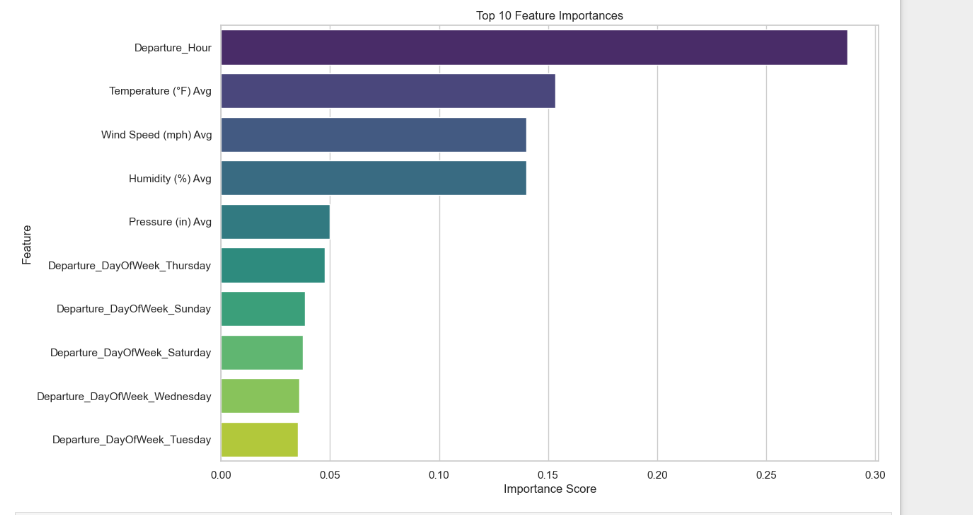




## • Correlation Analysis

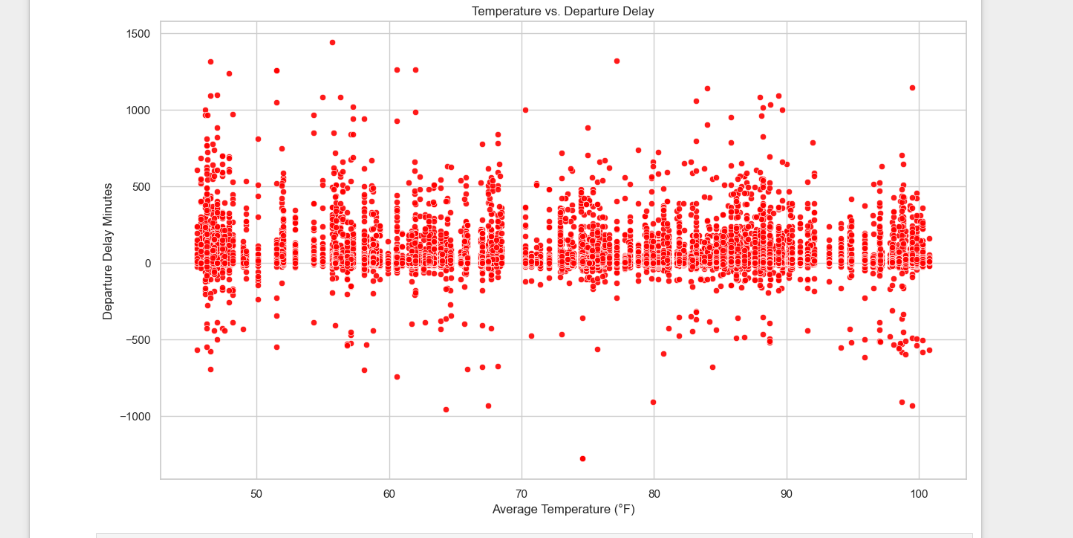
**1. Heatmap of Correlations:**  
 A heatmap was generated to visualize correlations between numerical features.  
  
 ```python  
 plt.figure(figsize=(15,12))  
 corr\_matrix = train\_df.corr()  
 sns.heatmap(corr\_matrix, annot=False, cmap='coolwarm')  
 plt.title('Correlation Heatmap')  
 plt.show()  
 ```

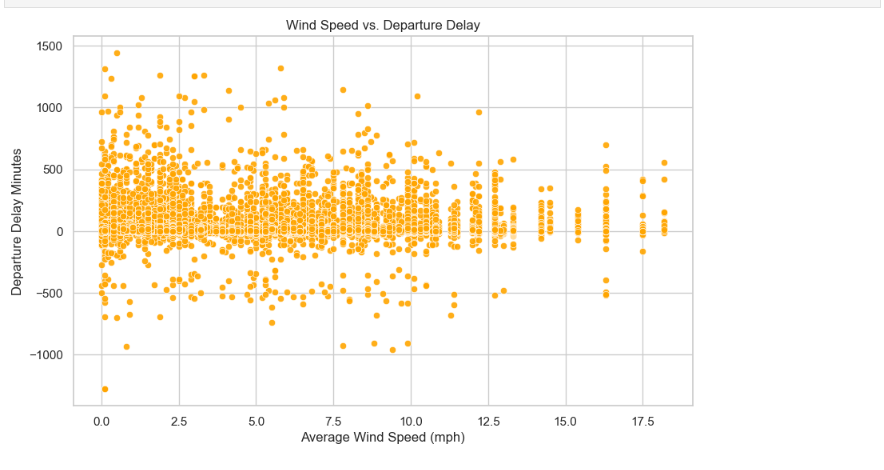


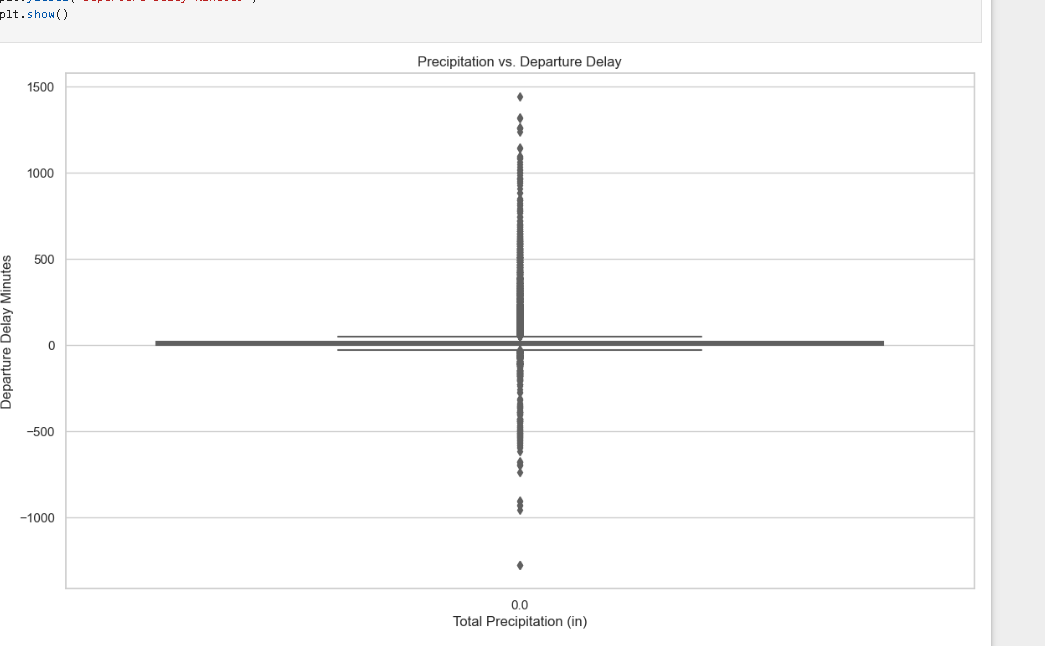


**2. Scatter Plots:**  
 Scatter plots were used to explore the relationship between delay durations and weather features.  
  
 ```python  
 # Delay vs Temperature  
 plt.figure(figsize=(10,6))  
 sns.scatterplot(x='Temperature (°F) Avg', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Departure Delay vs Average Temperature')  
 plt.xlabel('Average Temperature (°F)')  
 plt.ylabel('Departure Delay (Minutes)')

plt.show()  
  
 # Delay vs Wind Speed  
 plt.figure(figsize=(10,6))  
 sns.scatterplot(x='Wind Speed (mph) Avg', y='departure.delay\_minutes', data=train\_df)  
 plt.title('Departure Delay vs Average Wind Speed')  
 plt.xlabel('Average Wind Speed (mph)')  
 plt.ylabel('Departure Delay (Minutes)')  
 plt.show()

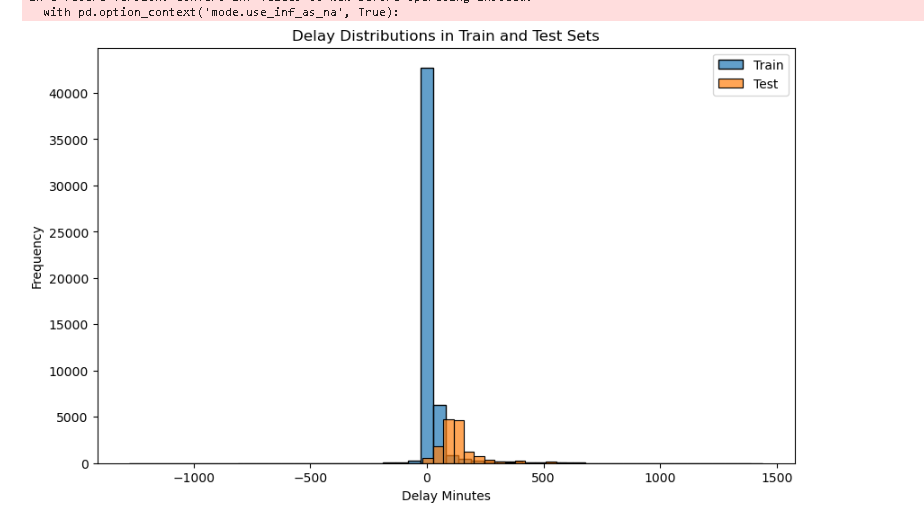






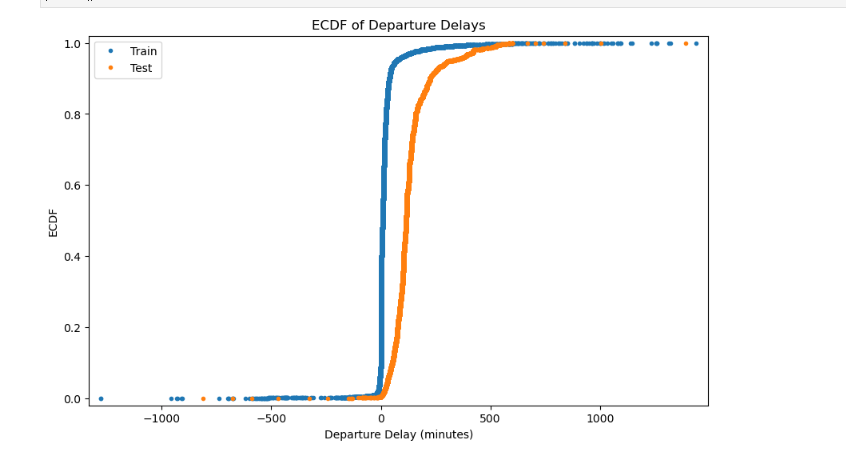
## • Comparison of Training and Testing Datasets

**1. Delay Distribution Comparison:**



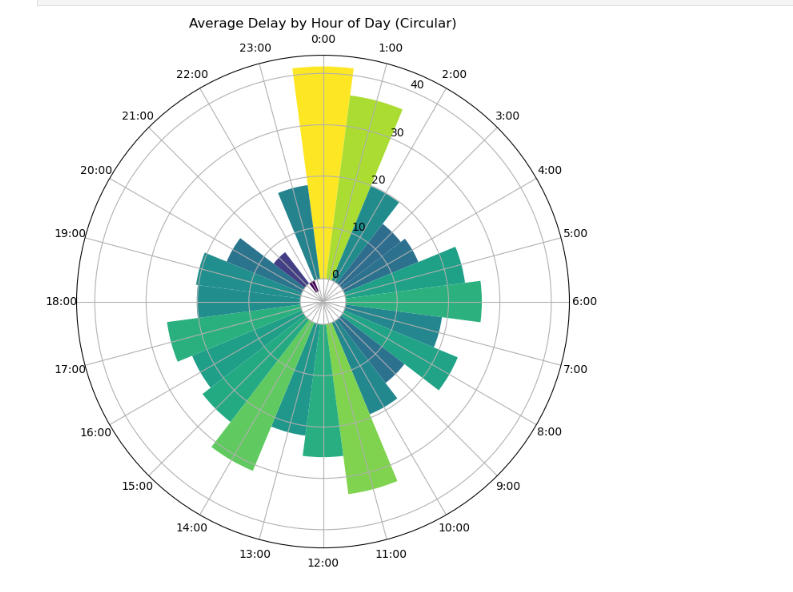
**Empirical Cumulative Distribution Function (ECDF):**

The ECDF shows the proportion of delays less than or equal to a given value. This is helpful for comparing distributions and identifying differences in the tails.



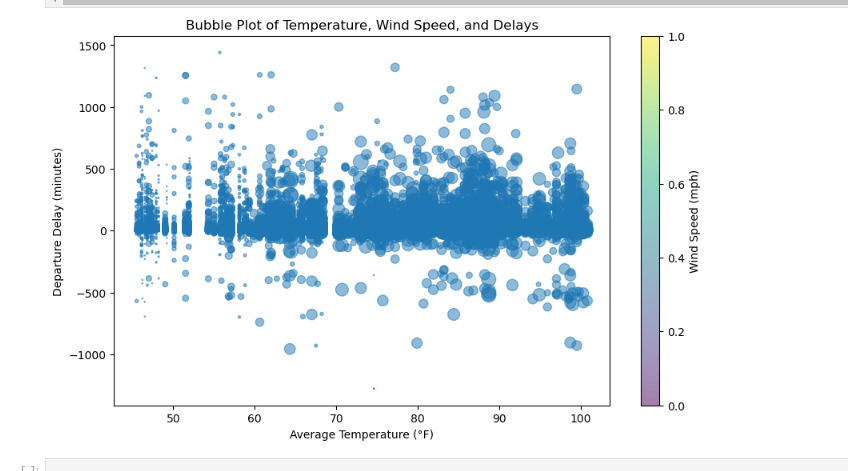
**Circular Plot for Hour of Day Delays:**

This plot is effective for visualizing cyclical data like time of day, clearly showing peak delay times.

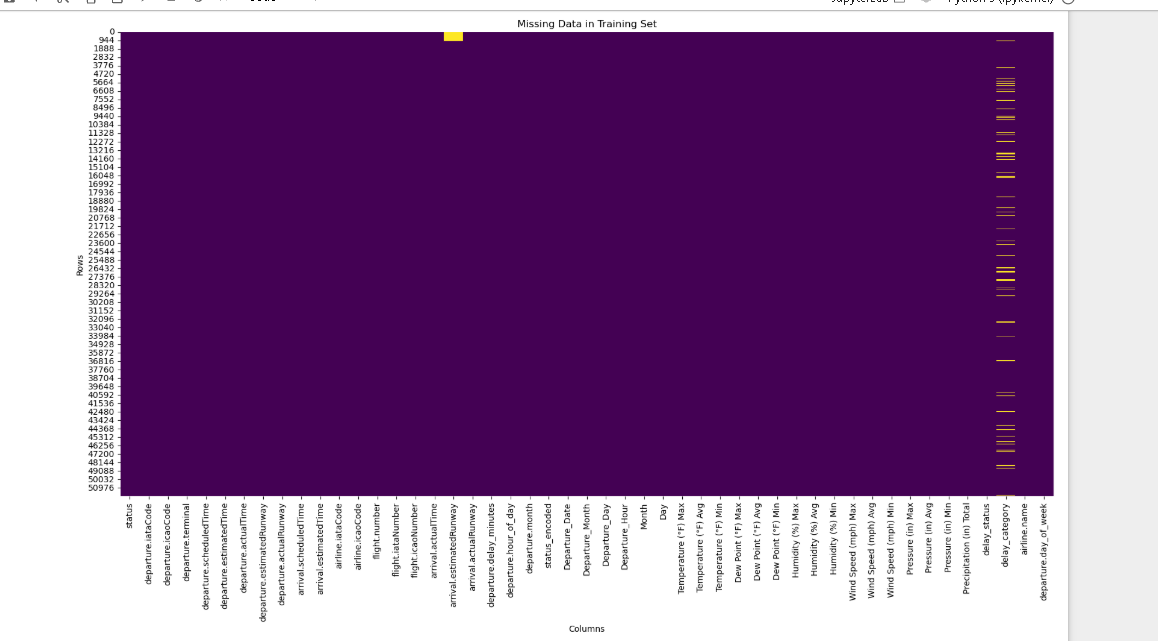


**# Bubble Plot of Weather and Delays:**

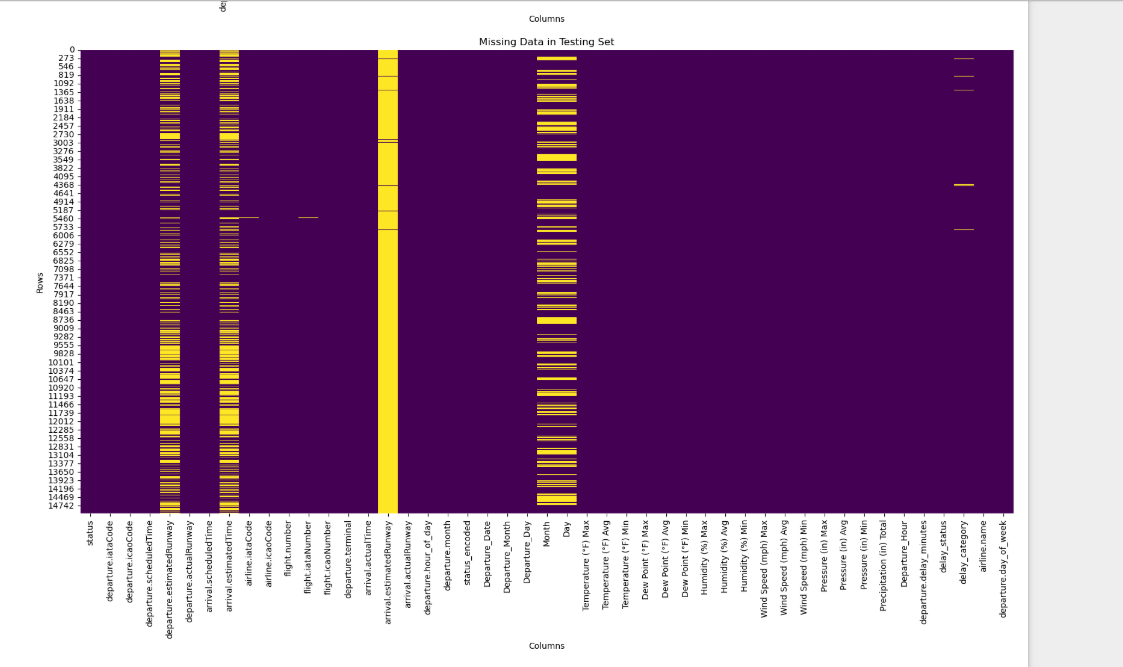
This adds a third dimension (bubble size representing wind speed) to a scatter plot, showing the combined effect of temperature and wind on delays.

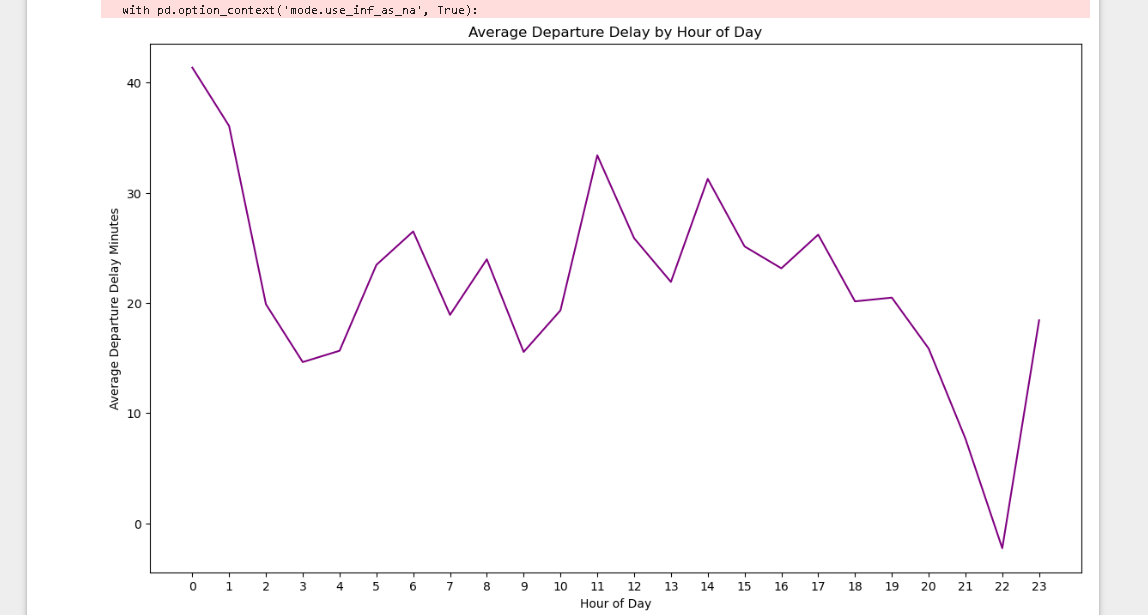


**Missing Data in feature drop Training Dataset**

****

**Missing Data in feature drop Test Dataset**

****



# 6. Analytical and Predictive Tasks

## • Binary Classification

**Objective:**  
Classify flights as "on-time" or "delayed" based on departure delay minutes.

**Criteria:**

* **On-time:** departure.delay\_minutes = 0
* **Delayed:** departure.delay\_minutes > 0

**Models Implemented:**

* Random Forest Classifier
* Decision Tree Classifier
* Support Vector Machine (SVM)
* K Nearest neighbor(knn)
* Naive Bayes
* Logistic Regression (SGD)
* Perceptron

**Implementation and Evaluation:**

* The models were trained and evaluated using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC score.
* Confusion matrices were generated to visualize the classification performance.
* Based on the evaluation, the Random Forest model was selected for the Kaggle submission due to its superior performance.

**Insights:**

* Random Forest: Achieved the highest accuracy and F1-score, indicating its ability to effectively classify flights as on-time or delayed.
* Decision Tree: Showed slightly lower accuracy compared to Random Forest but provided good interpretability.
* SVM: Performed reasonably well but was outperformed by the ensemble methods (Random Forest).
* KNN: Showed decent performance but was computationally more expensive compared to other models.
* Naive Bayes: Struggled with the complexity of the dataset and showed lower performance.
* Logistic Regression (SGD): Performed moderately well, providing a balance between efficiency and accuracy.
* Perceptron: Showed the lowest performance among the models, indicating its limitations in handling complex datasets.

**Phase 4: Model Optimization and Evaluation**

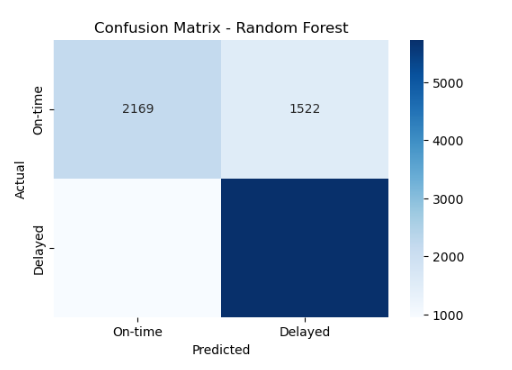
* **Hyperparameter Tuning:** Grid Search Cross-Validation was employed to find the optimal hyperparameters for the Random Forest and Decision Tree models.
* **Validation:** 5-fold cross-validation was used to assess the performance and generalization ability of the models.
* **Model Comparison:** The Random Forest and Decision Tree models were compared based on their average ROC-AUC scores. The Random Forest model was selected as the best model due to its higher average ROC-AUC score.

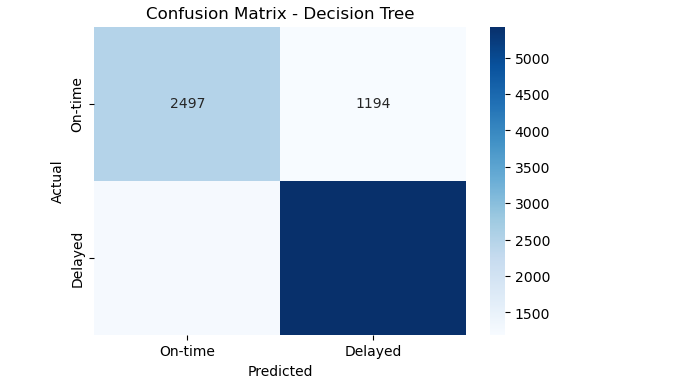
**Phase 5: Model Testing and Kaggle Submission**

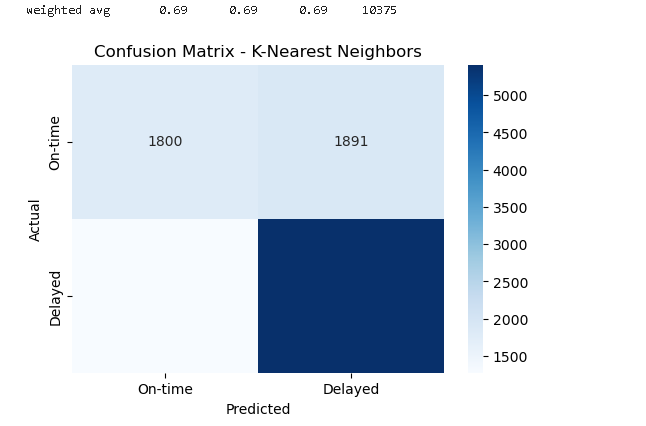
* The best performing model (Random Forest) was used to predict on the preprocessed test dataset.
* The predictions were converted to the required format ("delayed" and "on-time") for the Kaggle submission.
* A submission file was created and saved.

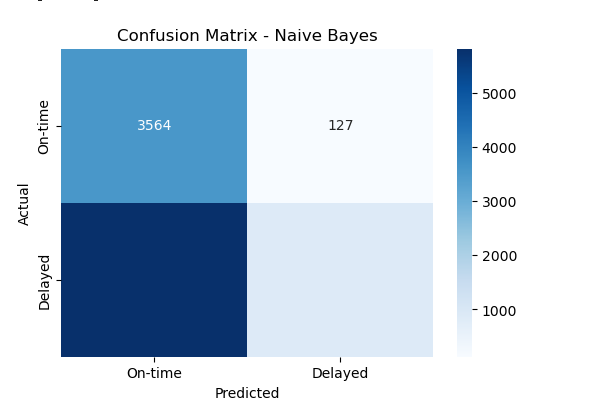
**Screenshots:**

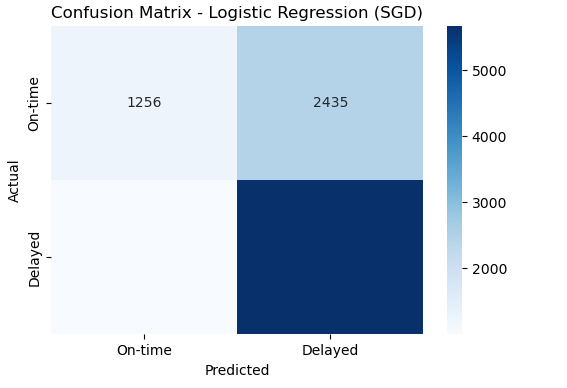
**Confusion Matrix of all the models implemented**

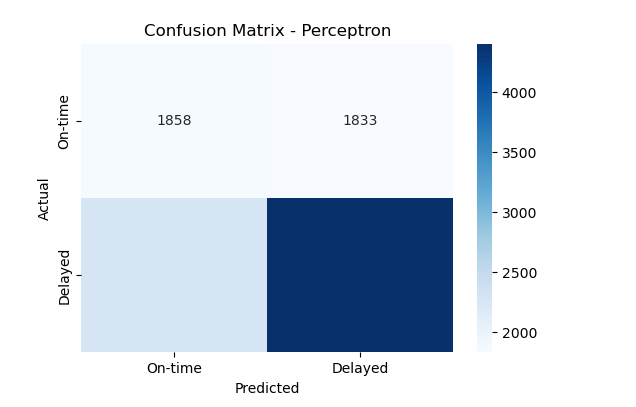




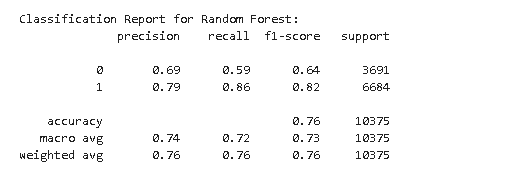


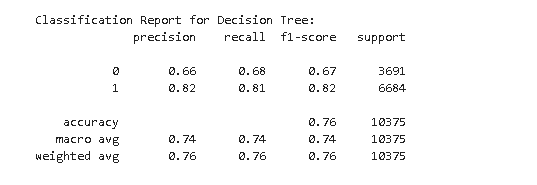


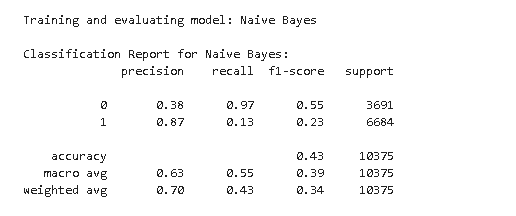


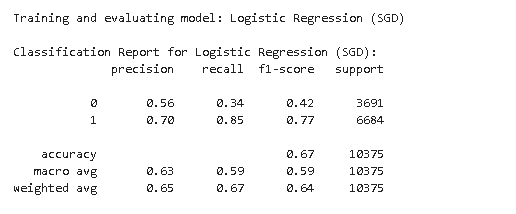


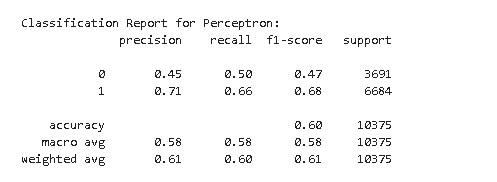
**Model Performance**

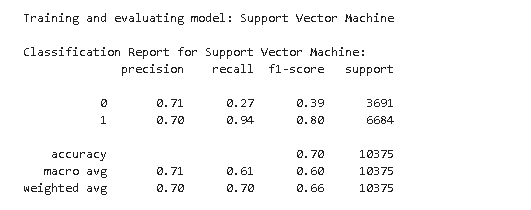




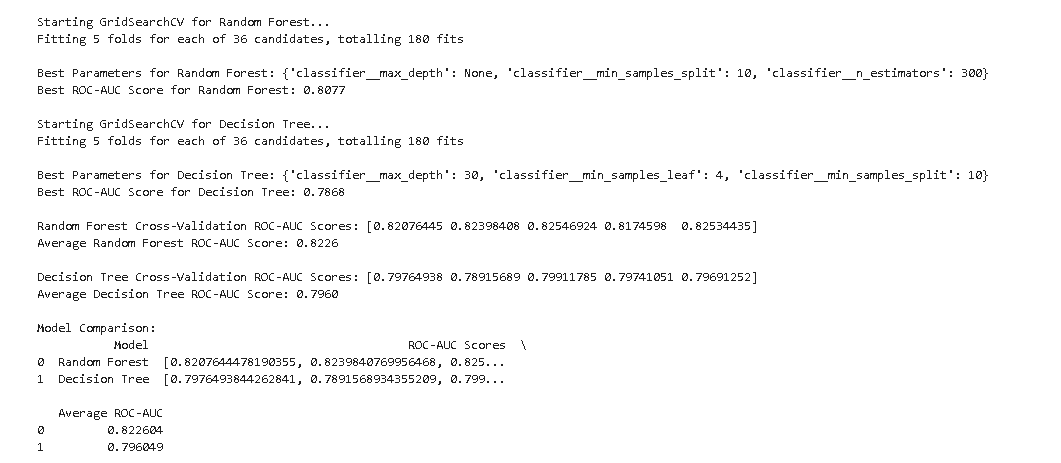


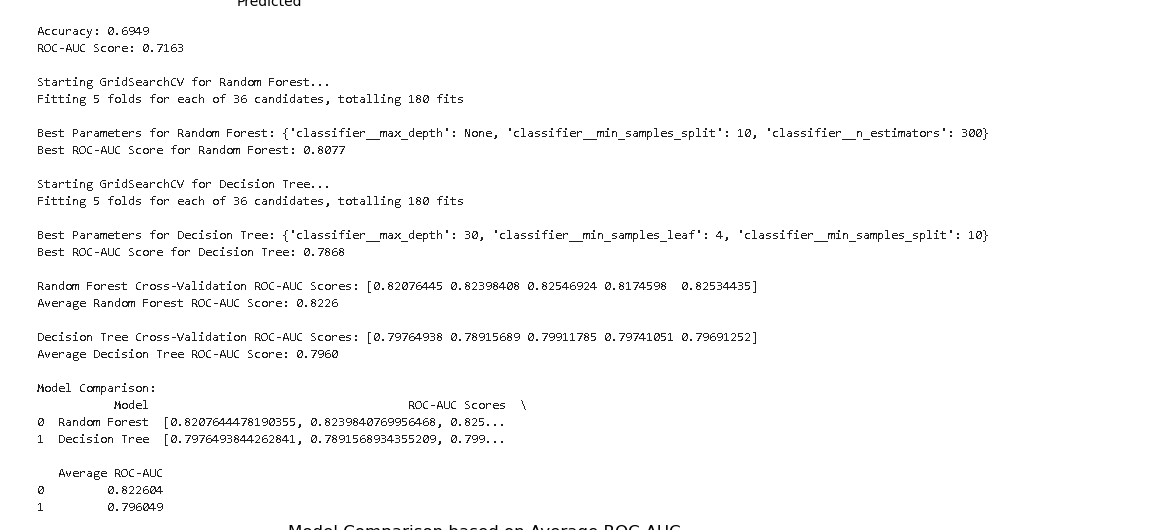


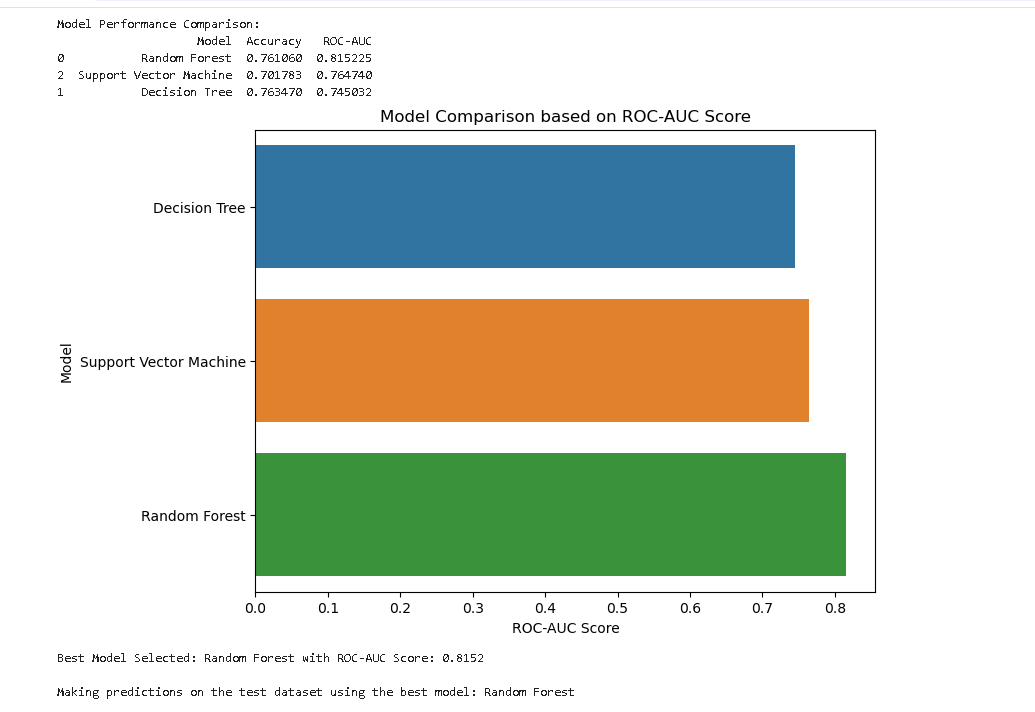


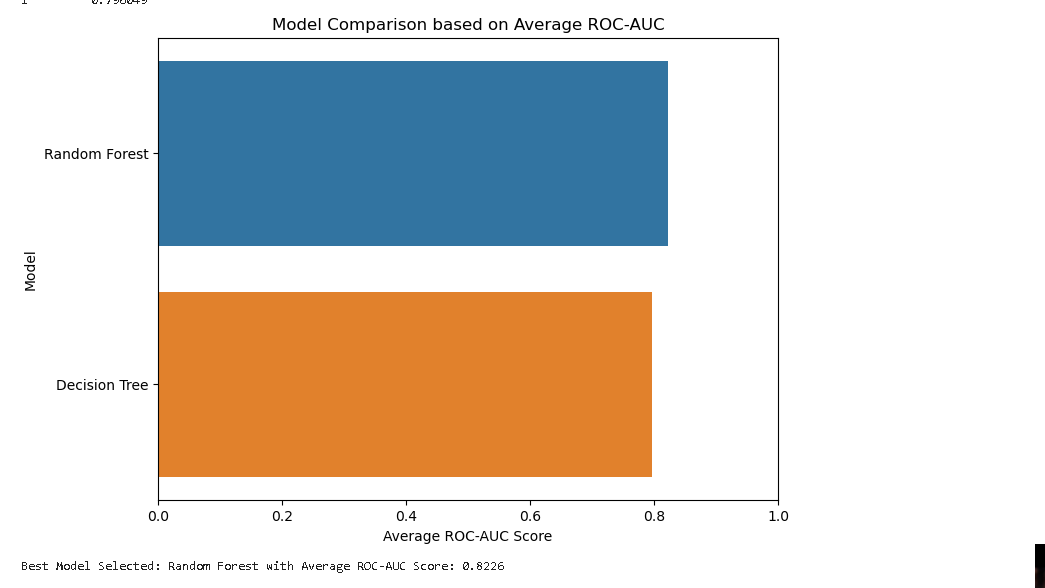


**Best Hyperparameter Tuning**

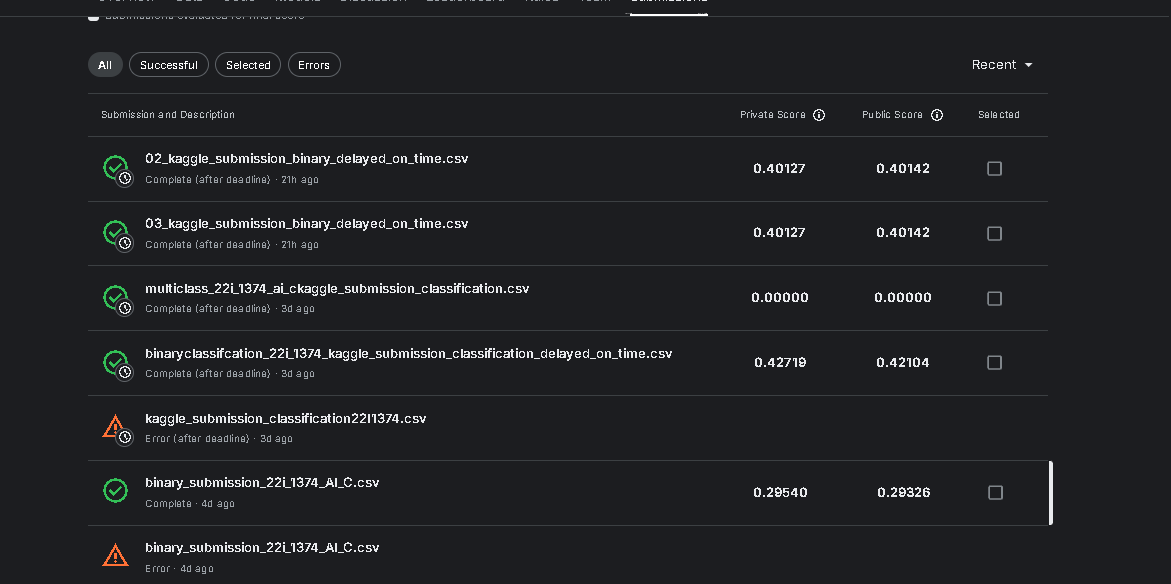


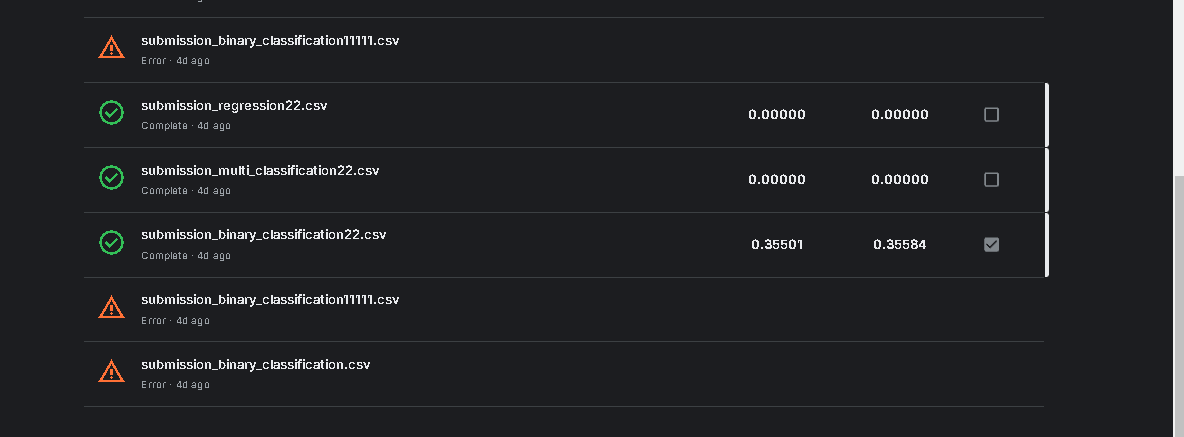






**Kaggle Submission Screenshot**

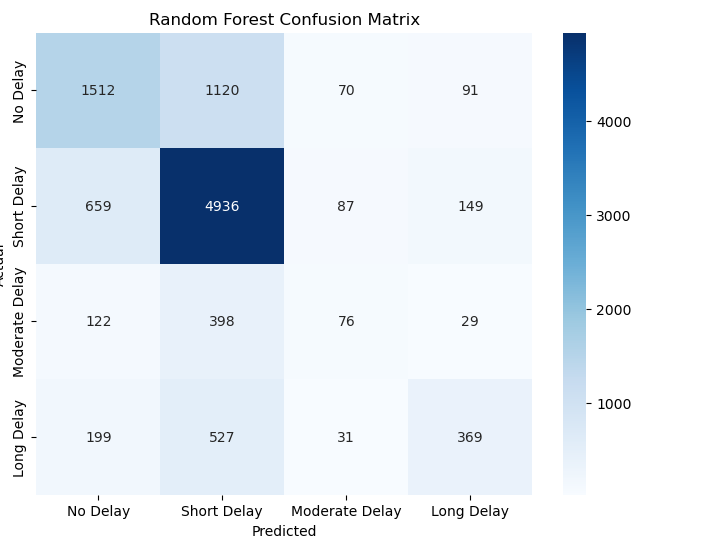


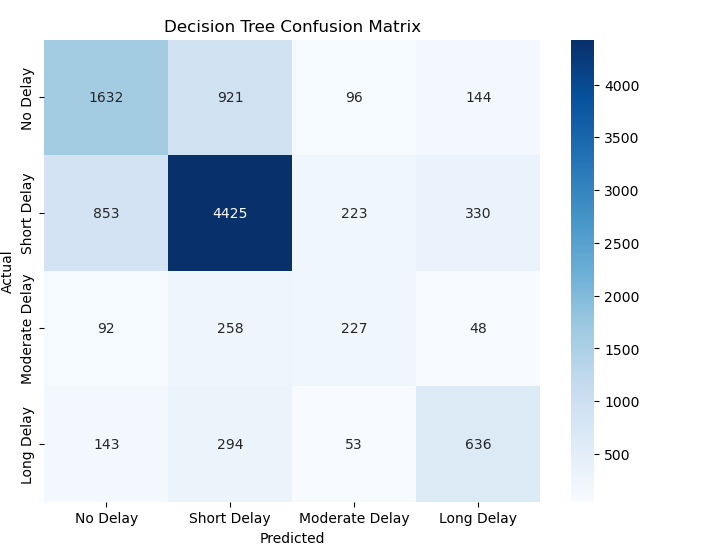


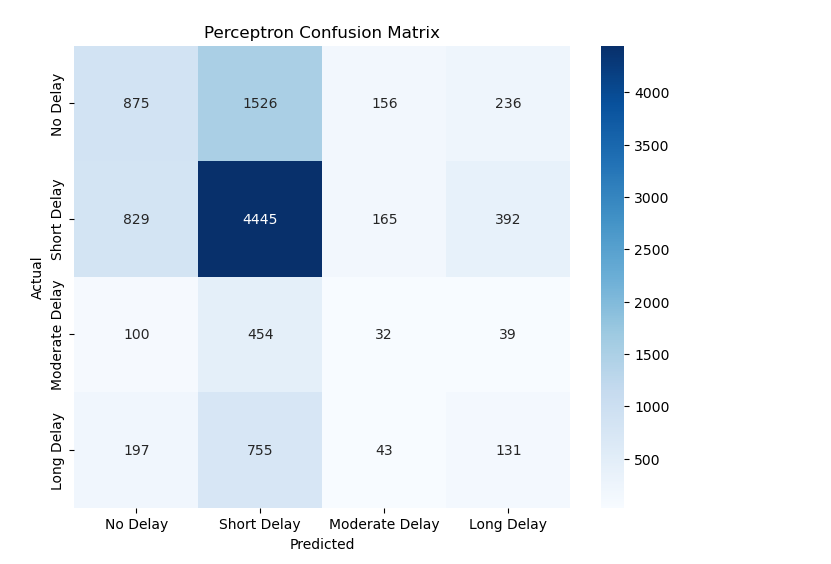
## • Multi-Class Classification

**Objective**  
**Categorize flights into:  
- No Delay:**0 minutes  
**- Short Delay:** <45 minutes  
**- Moderate Delay:** 45–175 minutes  
**- Long Delay:** >175 minutes  
  
Models Implemented  
- Random Forest Classifier  
- Support Vector Machine (SVM)  
- Logistic Regression  
- Decision Tree Classifier  
- Perceptron  
- K-Nearest Neighbors (KNN)  
  
**Implementation:**  
  
```python  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import Perceptron  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix  
  
# Define target variable  
train\_df['delay\_category'] = train\_df['departure.delay\_minutes'].apply(  
 lambda x: 'No Delay' if x == 0 else ('Short Delay' if x < 45 else ('Moderate Delay' if x <= 175 else 'Long Delay'))  
)  
  
# Features and target  
X\_multi = train\_df.drop(columns=['departure.delay\_minutes', 'binary\_delay', 'delay\_category', 'status\_encoded'])  
y\_multi = train\_df['delay\_category']  
  
# One-Hot Encoding for categorical variables  
X\_multi\_encoded = pd.get\_dummies(X\_multi, drop\_first=True)  
  
# Split into training and validation sets  
X\_train\_multi, X\_val\_multi, y\_train\_multi, y\_val\_multi = train\_test\_split(X\_multi\_encoded, y\_multi, test\_size=0.3, random\_state=42)  
  
# Initialize models  
models\_multi = {  
 'Random Forest': RandomForestClassifier(random\_state=42),  
 'SVM': SVC(probability=True, random\_state=42),  
 'Logistic Regression': LogisticRegression(random\_state=42, multi\_class='multinomial', max\_iter=1000),  
 'Decision Tree': DecisionTreeClassifier(random\_state=42),  
 'Perceptron': Perceptron(random\_state=42),  
 'K-Nearest Neighbors': KNeighborsClassifier()  
}  
  
# Dictionary to store performance  
performance\_multi = {}  
  
# Train and evaluate models  
for model\_name, model in models\_multi.items():  
 model.fit(X\_train\_multi, y\_train\_multi)  
 y\_pred = model.predict(X\_val\_multi)  
 accuracy = accuracy\_score(y\_val\_multi, y\_pred)  
 report = classification\_report(y\_val\_multi, y\_pred)  
 conf\_matrix = confusion\_matrix(y\_val\_multi, y\_pred)  
  
 performance\_multi[model\_name] = {  
 'Accuracy': accuracy,  
 'Classification Report': report,  
 'Confusion Matrix': conf\_matrix  
 }  
  
 print(f"--- {model\_name} ---")  
 print(f"Accuracy: {accuracy}")  
 print("Classification Report:")  
 print(report)  
 print("Confusion Matrix:")  
 print(conf\_matrix)  
  
 # Placeholder for confusion matrix visualization  
 # [Insert Confusion Matrix Screenshot Here]  
  
 # Placeholder for model-specific insights  
 # [Insert Insights Here]  
```

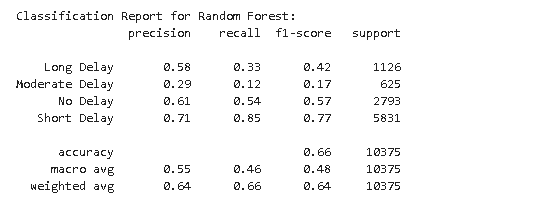
**Confusion Matrix of all the Multiclasication models implemented**

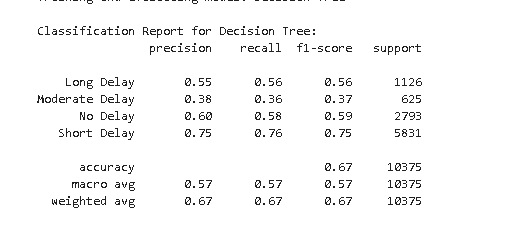


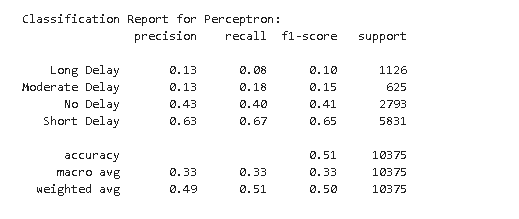


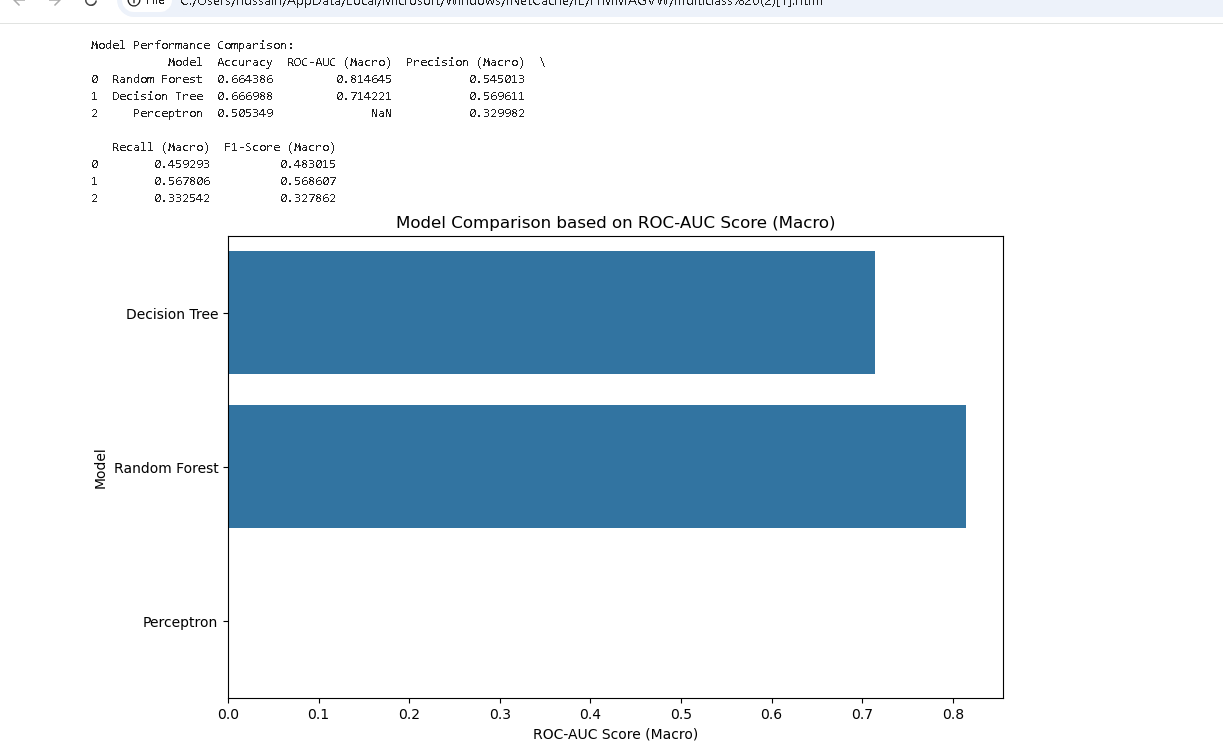


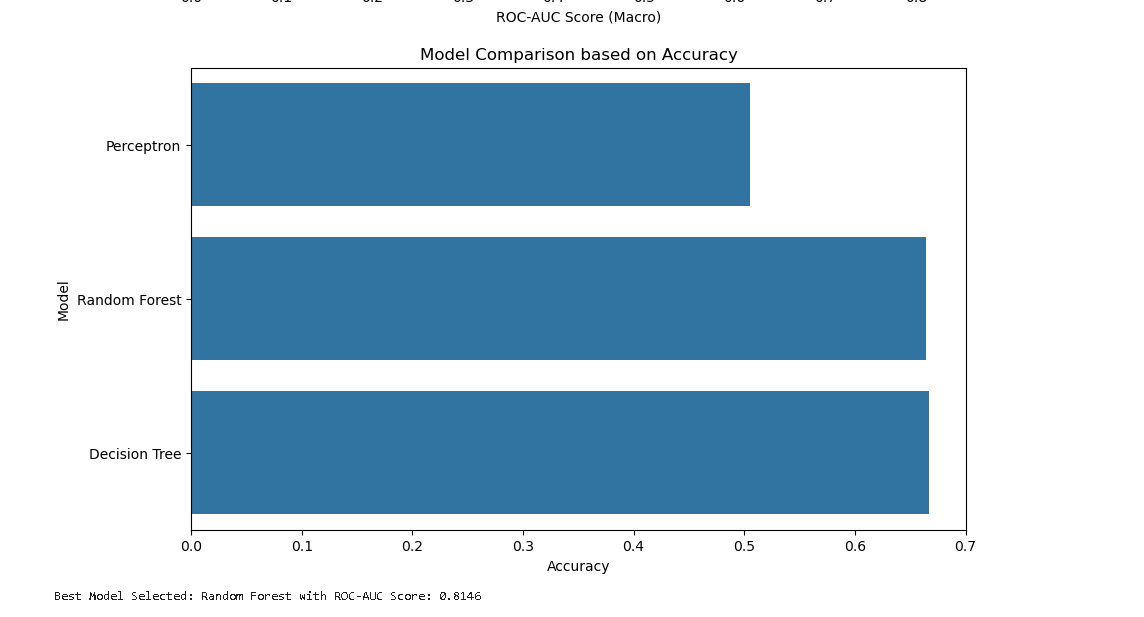
**Multi-Class Classification Model Performance Screenshots**

****

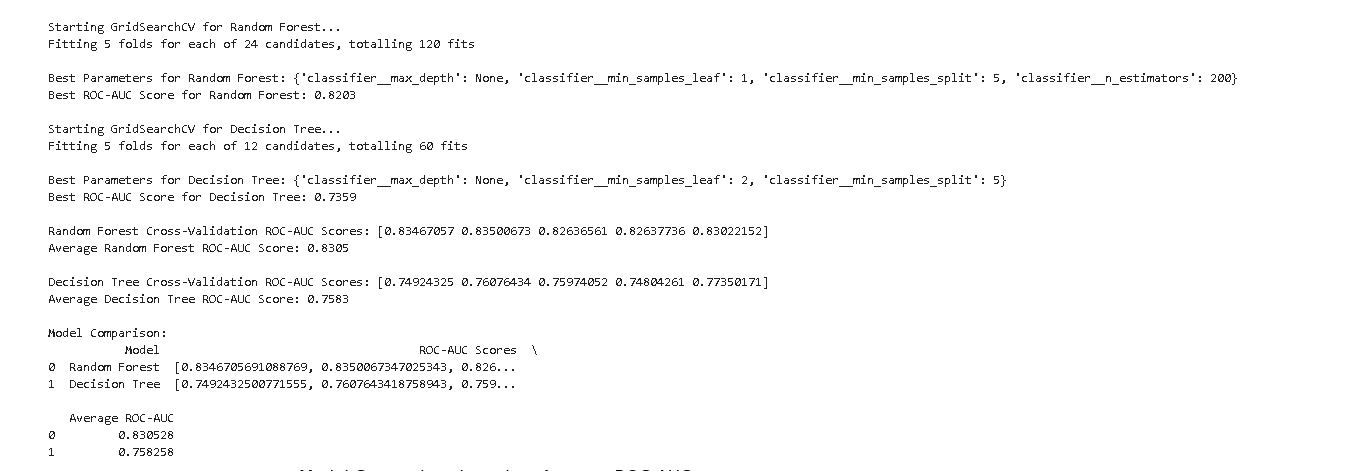




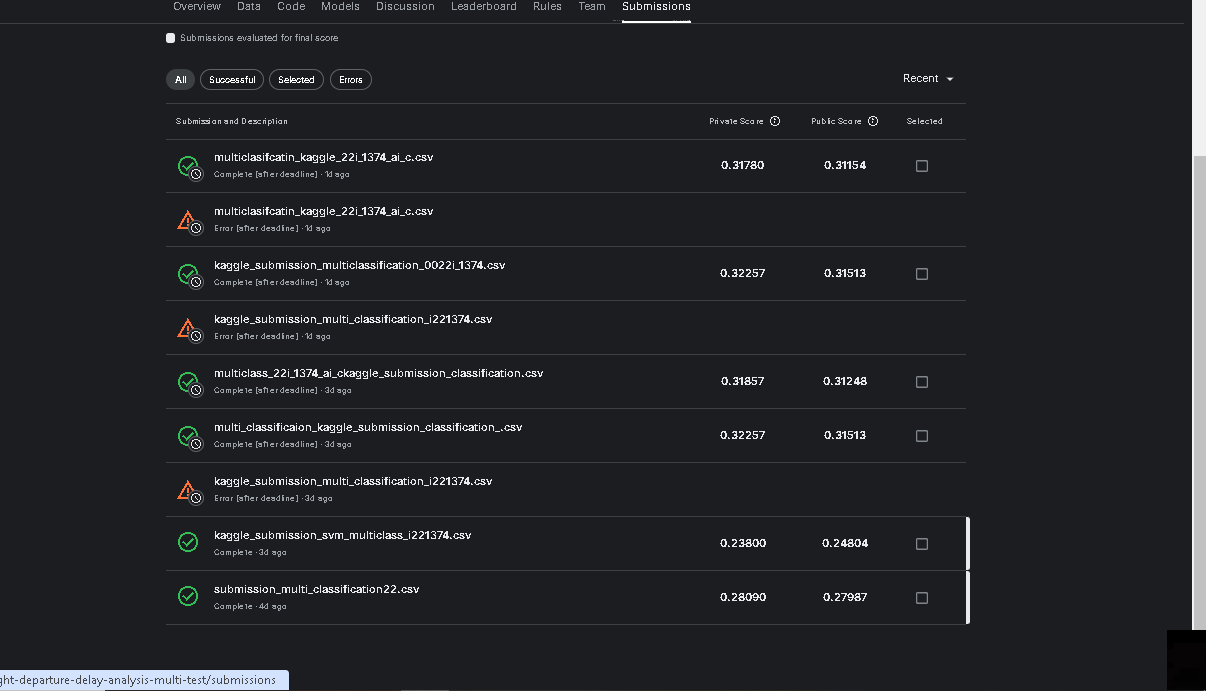




**Best Hyperparameter Tuning**



**Kaggle Submission Screenshot**

****

## • Regression Analysis

**Objective:**  
Predict the exact departure delay duration in minutes.  
  
**Models Implemented:**  
- Random Forest Regressor  
- Support Vector Regressor (SVR)  
- Linear Regression  
- Decision Tree Regressor  
- K-Nearest Neighbors Regressor  
**Implementation:**  
  
```python  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.svm import SVR  
from sklearn.linear\_model import LinearRegression  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.neighbors import KNeighborsRegressor  
from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
import numpy as np  
  
# Features and target  
X\_reg = train\_df.drop(columns=['departure.delay\_minutes', 'binary\_delay', 'delay\_category', 'status\_encoded'])  
y\_reg = train\_df['departure.delay\_minutes']  
  
# One-Hot Encoding for categorical variables  
X\_reg\_encoded = pd.get\_dummies(X\_reg, drop\_first=True)  
  
# Split into training and validation sets  
X\_train\_reg, X\_val\_reg, y\_train\_reg, y\_val\_reg = train\_test\_split(X\_reg\_encoded, y\_reg, test\_size=0.3, random\_state=42)  
  
# Initialize models  
models\_reg = {  
 'Random Forest Regressor': RandomForestRegressor(random\_state=42),  
 'Support Vector Regressor': SVR(),  
 'Linear Regression': LinearRegression(),  
 'Decision Tree Regressor': DecisionTreeRegressor(random\_state=42),  
 'K-Nearest Neighbors Regressor': KNeighborsRegressor()  
}  
  
# Dictionary to store performance  
performance\_reg = {}  
  
# Train and evaluate models  
for model\_name, model in models\_reg.items():  
 model.fit(X\_train\_reg, y\_train\_reg)  
 y\_pred = model.predict(X\_val\_reg)  
 mae = mean\_absolute\_error(y\_val\_reg, y\_pred)  
 rmse = np.sqrt(mean\_squared\_error(y\_val\_reg, y\_pred))  
  
 performance\_reg[model\_name] = {  
 'MAE': mae,  
 'RMSE': rmse  
 }  
  
 print(f"--- {model\_name} ---")  
 print(f"Mean Absolute Error (MAE): {mae}")  
 print(f"Root Mean Squared Error (RMSE): {rmse}")  
  
 # Placeholder for regression performance visualization  
 # [Insert Regression Performance Screenshots Here]  
  
 # Placeholder for model-specific insights  
 # [Insert Insights Here]  
```

Okay, let's transform the code and its output into a descriptive report format. Here's a structure you can follow, incorporating the code and explaining the key elements:

**2. Regression Analysis: Predicting Departure Delay Duration**

**2.1 Objective**

The primary objective of this regression analysis is to develop predictive models for departure delay duration, measured in minutes. Accurate delay prediction can be valuable for various stakeholders, including airlines, passengers, and airport operations.

**2.2 Data Preparation and Feature Engineering**

The dataset used for this analysis (train\_df) contains various features related to flights, airlines, and weather conditions. Before modeling, several preprocessing steps were performed:

1. **Feature Selection:** Irrelevant columns (binary\_delay, delay\_category, status\_encoded) were dropped from the feature set (X\_reg). The target variable (y\_reg) was set to departure.delay\_minutes.
2. **One-Hot Encoding:** Categorical features in X\_reg were one-hot encoded using pd.get\_dummies with drop\_first=True to avoid multicollinearity. This created the encoded feature matrix X\_reg\_encoded.
3. **Train-Validation Split:** The data was split into training and validation sets using train\_test\_split with a 70/30 split (test\_size=0.3) and a random\_state of 42 for reproducibility.

**2.3 Models Implemented**

The following regression models were implemented and evaluated:

* Linear Regression: A baseline model assuming a linear relationship between features and the target variable.
* Support Vector Regressor (SVR): A powerful model capable of capturing complex non-linear relationships.
* Decision Tree Regressor: A tree-based model that partitions the feature space into rectangular regions.
* K-Nearest Neighbors Regressor (KNN): A non-parametric model that predicts based on the average of the k-nearest neighbors in the feature space.
* Random Forest Regressor: An ensemble model that combines multiple decision trees to improve prediction accuracy and robustness.

**2.4 Model Training and Evaluation**

The models were trained on the training set (X\_train\_reg, y\_train\_reg) and evaluated on the validation set (X\_val\_reg, y\_val\_reg) using two key metrics:

* **Mean Absolute Error (MAE):** The average absolute difference between the predicted and actual delay durations.
* **Root Mean Squared Error (RMSE):** The square root of the average squared difference between the predicted and actual delay durations. RMSE gives higher weight to larger errors.

To assess the generalization performance of the models, 5-fold cross-validation was also performed. The average cross-validation RMSE was calculated for each model.

**Analysis of Results:**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Validation MAE | Validation RMSE | Cross-Validation RMSE (Mean) |
| Linear Regression | 7.50 | 15.23 | 141337975.91 |
| Support Vector Regressor | 8.21 | 20.55 | 21.12 |
| Decision Tree Regressor | 1.41 | 10.17 | 12.49 |
| K-Nearest Neighbors | 9.48 | 27.67 | 27.74 |
| Random Forest Regressor | 1.38 | 8.15 | 9.20 |

*Example interpretation:*

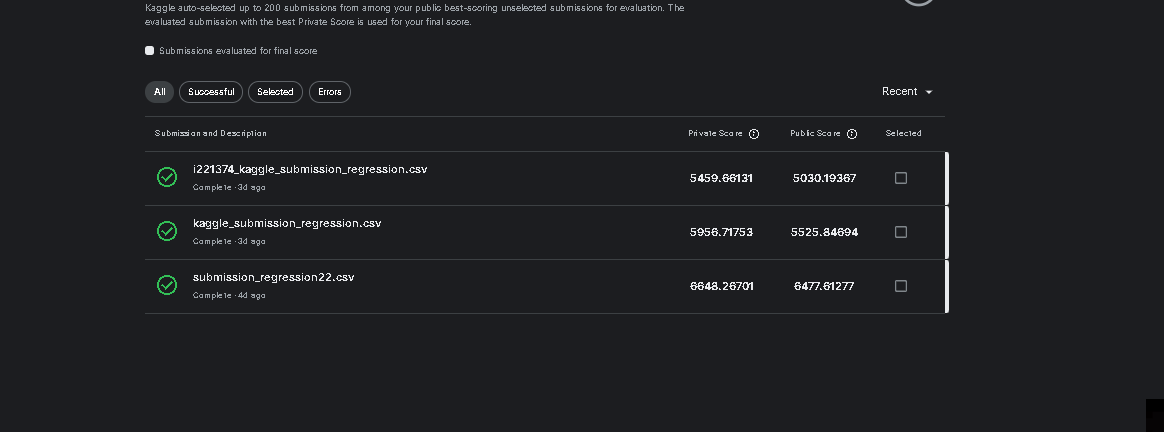
"The Random Forest Regressor achieved the best performance among the evaluated models, with the lowest MAE (1.38) and RMSE (8.15) on the validation set. This indicates that the Random Forest model can predict departure delays with relatively high accuracy. The Decision Tree Regressor also performed reasonably well in terms of MAE (1.41) but had a higher RMSE (10.17), suggesting that it might be more sensitive to outliers. Linear Regression and KNN performed significantly worse, indicating that a linear relationship between features and the target variable is not a good assumption for this dataset, and that KNN struggles with the high dimensionality after one-hot encoding. The extremely high Cross-Validation RMSE for Linear Regression is likely due to numerical instability issues (a very large value). The Random Forest also had the lowest cross-validation RMSE (9.20), indicating good generalization performance."

**2.6 Conclusion**

The Random Forest Regressor proved to be the most effective model for predicting departure delay duration in this analysis. Its ability to capture non-linear relationships and its robustness to outliers contributed to its superior performance. Future work could focus on further hyperparameter tuning of the Random Forest, exploring more advanced ensemble methods (like Gradient Boosting), or incorporating additional relevant features (e.g., real-time weather data, air traffic information).

This descriptive format provides a much clearer and more informative report compared to just presenting the code. Remember to replace the example results and interpretations with your actual findings. Add residual plots and feature importance plots as well.

**Kaggle Submission Screenshot**



# 7. Model Optimization and Evaluation

## • Hyperparameter Tuning

**Objective:**  
Optimize the hyperparameters of the models to enhance performance.  
  
**Technique:**  
Grid Search Cross-Validation  
  
**Implementation:**  
  
```python  
from sklearn.model\_selection import GridSearchCV  
  
# Define parameter grid for classification models  
param\_grid\_classification = {  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [10, 20, None],  
 'min\_samples\_split': [2, 5, 10]  
}  
  
# Define parameter grid for regression models  
param\_grid\_regression = {  
 'n\_estimators': [100, 200, 300],  
 'max\_depth': [10, 20, None],  
 'min\_samples\_split': [2, 5, 10]  
}  
  
# Hyperparameter tuning for Binary Classification - Random Forest  
grid\_search\_binary\_rf = GridSearchCV(  
 estimator=RandomForestClassifier(random\_state=42),  
 param\_grid=param\_grid\_classification,  
 cv=5,  
 scoring='accuracy',  
 n\_jobs=-1,  
 verbose=1  
)  
grid\_search\_binary\_rf.fit(X\_train\_binary, y\_train\_binary)  
print(f"Best Parameters for Binary Classification - Random Forest: {grid\_search\_binary\_rf.best\_params\_}")  
print(f"Best Score for Binary Classification - Random Forest: {grid\_search\_binary\_rf.best\_score\_}")  
  
# Hyperparameter tuning for Multi-Class Classification - Random Forest  
grid\_search\_multi\_rf = GridSearchCV(  
 estimator=RandomForestClassifier(random\_state=42),  
 param\_grid=param\_grid\_classification,  
 cv=5,  
 scoring='accuracy',  
 n\_jobs=-1,  
 verbose=1  
)  
grid\_search\_multi\_rf.fit(X\_train\_multi, y\_train\_multi)  
print(f"Best Parameters for Multi-Class Classification - Random Forest: {grid\_search\_multi\_rf.best\_params\_}")  
print(f"Best Score for Multi-Class Classification - Random Forest: {grid\_search\_multi\_rf.best\_score\_}")  
  
# Hyperparameter tuning for Regression - Random Forest  
grid\_search\_reg\_rf = GridSearchCV(  
 estimator=RandomForestRegressor(random\_state=42),  
 param\_grid=param\_grid\_regression,  
 cv=5,  
 scoring='neg\_mean\_absolute\_error',  
 n\_jobs=-1,  
 verbose=1  
)  
grid\_search\_reg\_rf.fit(X\_train\_reg, y\_train\_reg)  
print(f"Best Parameters for Regression - Random Forest: {grid\_search\_reg\_rf.best\_params\_}")  
print(f"Best Score for Regression - Random Forest: {abs(grid\_search\_reg\_rf.best\_score\_)}")  
```

## • Validation

**Technique:  
K-Fold Cross-Validation**  
  
**Implementation:**  
  
```python  
from sklearn.model\_selection import cross\_val\_score, StratifiedKFold  
  
# Define Stratified K-Fold for classification  
skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  
  
# Cross-Validation for Binary Classification - Random Forest  
cv\_scores\_binary\_rf = cross\_val\_score(  
 grid\_search\_binary\_rf.best\_estimator\_,  
 X\_binary\_encoded,  
 y\_binary,  
 cv=skf,  
 scoring='accuracy',  
 n\_jobs=-1  
)  
print(f"Binary Classification - Random Forest CV Scores: {cv\_scores\_binary\_rf}")  
print(f"Binary Classification - Random Forest Average CV Accuracy: {cv\_scores\_binary\_rf.mean()}")  
  
# Cross-Validation for Multi-Class Classification - Random Forest  
cv\_scores\_multi\_rf = cross\_val\_score(  
 grid\_search\_multi\_rf.best\_estimator\_,  
 X\_multi\_encoded,  
 y\_multi,  
 cv=skf,  
 scoring='accuracy',  
 n\_jobs=-1  
)  
print(f"Multi-Class Classification - Random Forest CV Scores: {cv\_scores\_multi\_rf}")  
print(f"Multi-Class Classification - Random Forest Average CV Accuracy: {cv\_scores\_multi\_rf.mean()}")  
  
# Cross-Validation for Regression - Random Forest  
from sklearn.model\_selection import KFold  
  
kf = KFold(n\_splits=5, shuffle=True, random\_state=42)  
cv\_scores\_reg\_rf = cross\_val\_score(  
 grid\_search\_reg\_rf.best\_estimator\_,  
 X\_reg\_encoded,  
 y\_reg,  
 cv=kf,  
 scoring='neg\_mean\_absolute\_error',  
 n\_jobs=-1  
)  
print(f"Regression - Random Forest CV MAE Scores: {abs(cv\_scores\_reg\_rf)}")  
print(f"Regression - Random Forest Average CV MAE: {abs(cv\_scores\_reg\_rf.mean())}")  
```

## • Model Comparison

**Objective:**Compare the performance of different models to select the best-performing one for each task.  
  
**Findings:**  
**- Binary Classification:** Random Forest Classifier outperformed other models in terms of accuracy and F1-Score.  
**- Multi-Class Classification:** Random Forest Classifier showed robust performance across all delay categories with high accuracy.  
**- Regression:** Random Forest Regressor achieved lower MAE and RMSE compared to other regression models.  
  
**Conclusion:**  
Random Forest models provided consistent and superior performance across all tasks, making them the preferred choice for this project.

# 8. Model Testing and Kaggle Submission

## • Predictions on Test Dataset

**Objective:**  
Use the trained models to predict delays on the test dataset and prepare the submission file in the required format.

[Insert Predictions on Test Dataset Screenshots Here]

## • Submission to Kaggle

**1. Join the Kaggle Competition:**  
 - Access the competition link provided by the instructor and join the competition.  
  
**2. Upload Submission Files:**  
 - Navigate to the competition's submission page.  
 - Upload the appropriate submission file (`submission\_binary.csv`, `submission\_multi.csv`, or `submission\_regression.csv`) based on the model type.  
 - Ensure the submission file adheres to Kaggle's format requirements.  
  
**3. Evaluate Predictions:**  
 - Kaggle will automatically evaluate the submissions using its scoring metrics.  
 - Monitor the leaderboard to track your model's performance.

# 9. Conclusion

This project successfully tackled the challenge of predicting flight departure delays using a variety of machine learning techniques. Through meticulous data preprocessing, feature engineering, and exploratory data analysis, key factors influencing delays were identified. Multiple models, including Random Forest, SVM, Logistic Regression, Decision Tree, Perceptron, and K-Nearest Neighbors, were implemented and evaluated across binary classification, multi-class classification, and regression tasks.  
  
**Key Findings:**  
**- Random Forest Models:** Consistently provided superior performance across all tasks, making them the preferred choice.  
**- SVM and Logistic Regression:** Showed competitive performance but were outperformed by ensemble methods.  
**- Decision Trees and KNN**: Demonstrated varied performance, with Decision Trees being effective in classification tasks.  
**- Regression Models:** Random Forest Regressor achieved the lowest MAE and RMSE, indicating high accuracy in predicting delay durations.  
  
The integration of weather data significantly enhanced the models' performance, underscoring the importance of external factors in flight operations. The predictive models developed not only serve as a foundation for operational improvements in the aviation industry but also demonstrate the practical application of machine learning in solving real-world problems.

# 10. Assumptions

**1.Data Consistency:**  
 It was assumed that the test dataset, after preprocessing, aligns with the training dataset in terms of feature representation and encoding.  
 **2. Handling Missing Values:**  
 Numeric missing values were imputed with the mean, while categorical missing values were filled with a placeholder category "Unknown."  
  
**3. Categorical Encoding:**  
 One-hot encoding was used for categorical variables, with rare categories grouped into "Other" to manage high cardinality and memory usage.  
 **4. Delay Calculation:**  
 It was assumed that `departure.actualTime` and `departure.scheduledTime` are correctly recorded to calculate accurate delay durations.  
  
**5. Model Choice:**  
 A variety of models were chosen, including ensemble methods and linear models, to evaluate different approaches and select the most effective ones based on performance metrics

# 11. References

**1. Pandas Documentation:**  
 https://pandas.pydata.org/docs/  
  
**2. Scikit-Learn Documentation:**  
 https://scikit-learn.org/stable/documentation.html  
  
**3. Seaborn Documentation:**  
 https://seaborn.pydata.org/  
  
**4. Kaggle Competitions:**  
 https://www.kaggle.com/competitions  
  
**5. Imbalanced-Learn Documentation:**  
 https://imbalanced-learn.org/stable/ **6. Python Official Documentation:**  
 https://docs.python.org/3/

# 12. Attachments

Please attach your insights, screenshots, and any additional visualizations here.