IE 7275 Data Mining

FLIGHT DATA

GROUP 2

SHUBHAM GAUR

MALHAR GHOGARE

INTRODUCTION

Overview:
Flight delays pose serious problems for the aviation sector, since they may cause inconvenience for travelers and financial losses for carriers. Our goal is to implement predictive models that can be used for regression (predicting delay time) and classification (identifying the kind of airline) tasks by utilizing data mining techniques.

Objective:
The goal of the project is to improve knowledge and offer practical insights into the variables that affect flight delays and the features of various airlines. This is to be followed by implementation of various machine learning models and ultimately compare their performance

DATA COLLECTION

Dataset link:
https://figshare.com/articles/dataset/flights_csv/9820139
Our dataset covers a diverse range of airlines, airports, and flight routes, providing a holistic perspective on the aviation landscape.

Data Description:

The dataset contains over 105,000 rows of data after cleaning null entries. It contains a total of 30 features of various datatypes (int, float, object)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5819079 entries, 0 to 5819078
Data columns (total 31 columns):
 #   Column               Dtype
---  ------               -----
 0   YEAR                 int64
 1   MONTH                int64
 2   DAY                  int64
 3   DAY_OF_WEEK          int64
 4   AIRLINE              object
 5   FLIGHT_NUMBER        int64
 6   TAIL_NUMBER          object
 7   ORIGIN_AIRPORT       object
 8   DESTINATION_AIRPORT  object
 9   SCHEDULED_DEPARTURE  int64
 10  DEPARTURE_TIME       float64
 11  DEPARTURE_DELAY      float64
 12  TAXI_OUT             float64
 13  WHEELS_OFF           float64
 14  SCHEDULED_TIME       float64
 15  ELAPSED_TIME         float64
 16  AIR_TIME             float64
 17  DISTANCE             int64
 18  WHEELS_ON            float64
 19  TAXI_IN              float64
 20  SCHEDULED_ARRIVAL    int64
 21  ARRIVAL_TIME         float64
 22  ARRIVAL_DELAY        float64
 23  DIVERTED             int64
 24  CANCELLED            int64
 25  CANCELLATION_REASON  object
 26  AIR_SYSTEM_DELAY     float64
 27  SECURITY_DELAY       float64
 28  AIRLINE_DELAY        float64
 29  LATE_AIRCRAFT_DELAY  float64
 30  WEATHER_DELAY        float64
dtypes: float64(16), int64(10), object(5)
memory usage: 1.3+ GB
```

Data Dictionary:

Variable Name	Description
YEAR	The year of the flight
MONTH	The month of the flight.
DAY	The day of the month of the flight.
DAY_OF_WEEK	The day of the week of the flight
AIRLINE	The code representing the airline operating the flight
FLIGHT_NUMBER	The unique identification number assigned to the flight
TAIL_NUMBER	The registration number of the aircraft
ORIGIN_AIRPORT	The code representing the origin airport of the flight
DESTINATION_AIRPORT	The code representing the destination airport of the flight
SCHEDULED_DEPARTURE	The scheduled departure time of the flight (in local time)

DATA CLEANING AND PROCESSING

In [ ]:
```
df.isnull().sum()
```

Calculating the sum of null values in the dataframe

In [ ]:
```
# columns_to_drop = ['CANCELLATION_REASON', 'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY',
columns_to_drop = ['DIVERTED', 'CANCELLED']
df.drop(columns=columns_to_drop, inplace=True)

print(df)
```

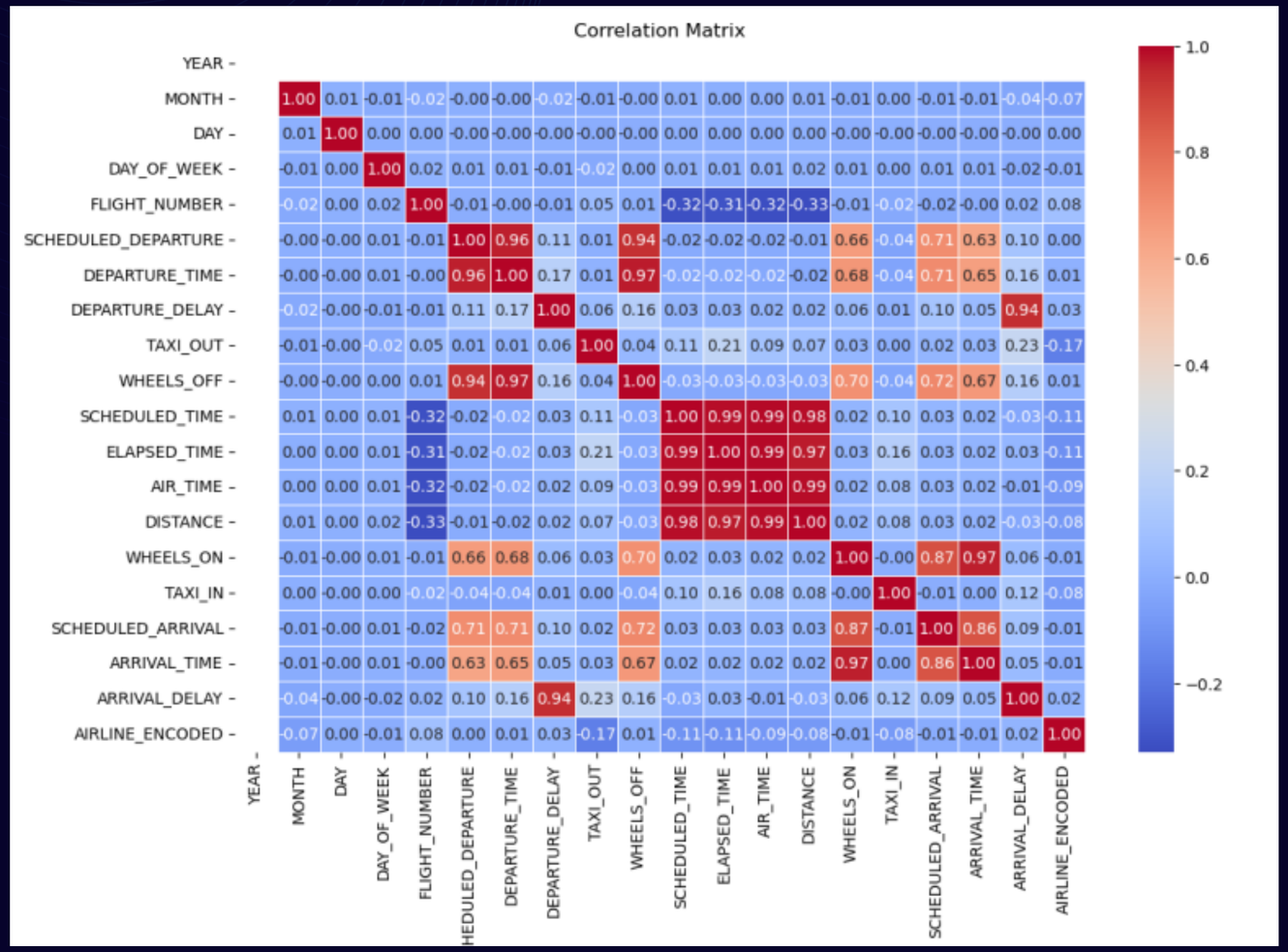
Dropped the columns because of there was no data present in them making them irrelevant to our project

In [ ]:
```
df.dropna(inplace=True)
```

Dropping rows with null values as there are maximum 105,000 rows with null values. Therefore it makes sense to drop the size of 5,000,000 rows even if we exclude the rows with null values

We identified and removed columns with missing data irrelevant to our study and opted to delete rows with null values—totaling 105,000 out of 5 million—maintaining data quality and integrity for our analysis and model training.

We then dropped columns like ‘DIVERTED’ and ‘CANCELLED’ as they contained a majority of the null entries



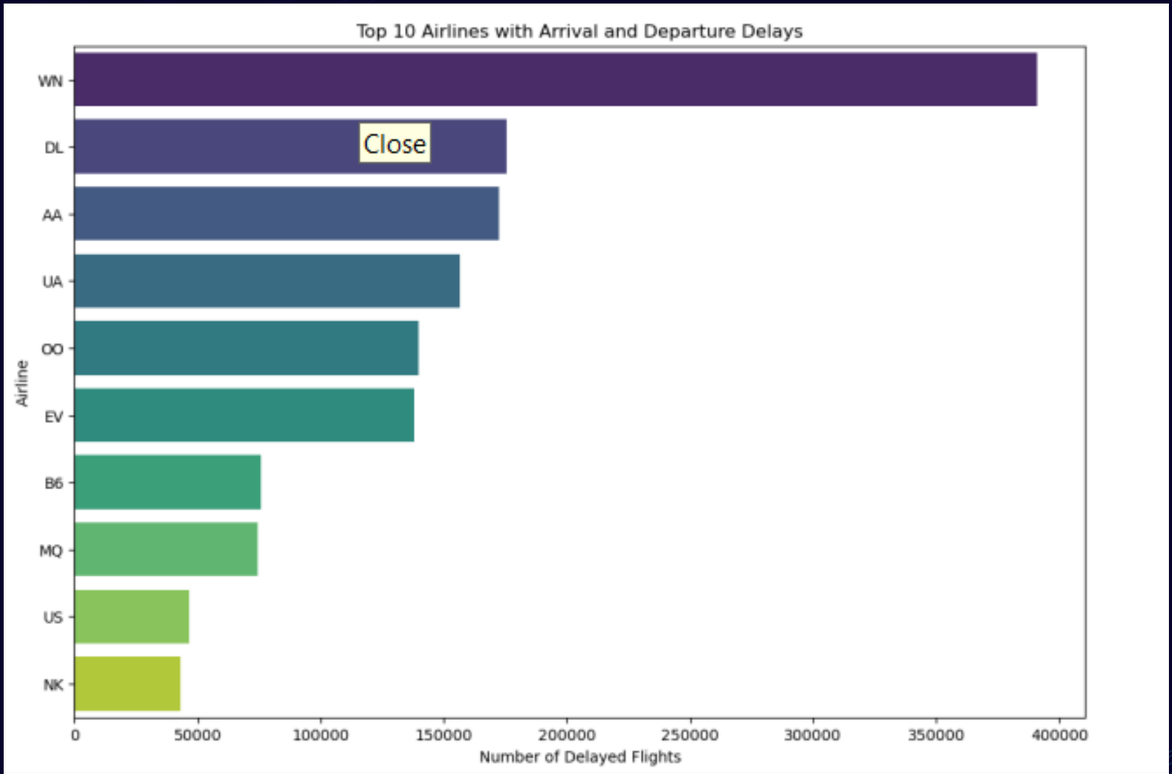
EXPLORATORY DATA ANALYSIS (EDA)

01 Correlation Matrix

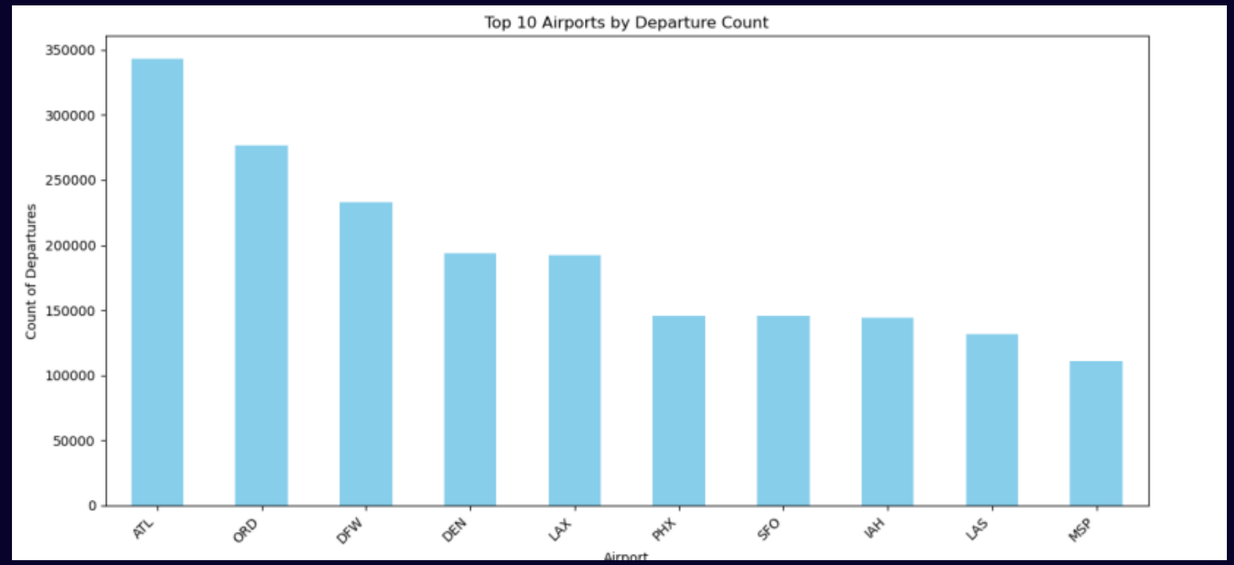
Correlation matrix to observe the relationships between the variables and develop an understanding of the interdependencies. It reveals that "SCHEDULED_DEPARTURE" has a significant correlation with several other time-related variables such as "WHEELS_OFF," "DEPARTURE_TIME," "WHEELS_ON," "SCHEDULED_ARRIVAL," and "ARRIVAL_TIME.". In contrast, "DEPARTURE_DELAY" shows a notable correlation only with "ARRIVAL_DELAY," indicating that delays in departure tend to correspond with delays in arrival.

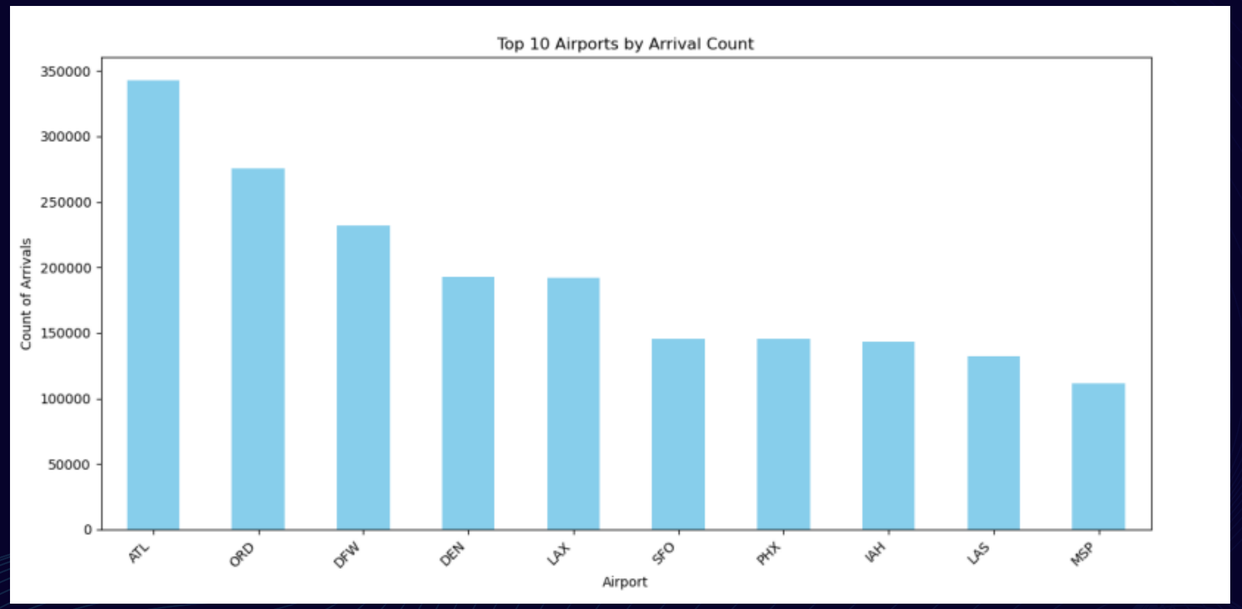
EXPLORATORY DATA ANALYSIS (EDA)

02 Top 10 airports by arrival and departure delays



03 Top 10 airports by count of flight arrival and departure





MODEL BUILDING

Feature Engineering :

Our feature engineering process begins by encoding the airline information seen in the 'AIRLINE' column. We also encoded the following columns along with airline columns as they contained alphanumeric data: 'ORIGIN_AIRPORT', "DESTINATION_AIRPORT', 'TAIL_NUMBER'

In [11]:
```
flight_df['ORIGIN_AIRPORT'] = flight_df['ORIGIN_AIRPORT'].astype(str)
flight_df['DESTINATION_AIRPORT'] = flight_df['DESTINATION_AIRPORT'].astype(str)
flight_df['TAIL_NUMBER'] = flight_df['TAIL_NUMBER'].astype(str)
label_encoder = LabelEncoder()
flight_df['ORIGIN_AIRPORT'] = label_encoder.fit_transform(flight_df['ORIGIN_AIRPORT'])
flight_df['DESTINATION_AIRPORT'] = label_encoder.fit_transform(flight_df['DESTINATION_AIRPORT'])
flight_df['TAIL_NUMBER'] = label_encoder.fit_transform(flight_df['TAIL_NUMBER'])
```

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Fit LabelEncoder and transform 'AIRLINE' column
df['AIRLINE_ENCODED'] = label_encoder.fit_transform(df['AIRLINE'])

# Get unique encoded values
unique_encoded_airlines = df['AIRLINE_ENCODED'].unique()

# Map encoded values back to original airline names
airline_mapping = dict(zip(df['AIRLINE_ENCODED'], df['AIRLINE']))

# Display unique encoded values and their corresponding airline names
print("Unique Encoded Airline Values:")
print(unique_encoded_airlines)

print("\nMapping of Encoded Values to Airline Names:")
print(airline_mapping)
```

Standardization:

We standardized our dataset using the StandardScaler() function. This conversion guarantees that every feature has a mean of 0 and a standard deviation of 1, which prepares our dataset for the best possible feature selection procedures later on in our machine learning project.

:
```
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df), columns=df.columns[0:])
display(df_scaled)
```

MODEL BUILDING

Feature Selection:

For selecting the features, we utilized the SelectKBest function with the ANOVA F-test as the score function. The ANOVA F-test is a statistical method that compares the means of the features to determine their correlation with the target variable.
We specified the parameter k=10 in the SelectKBest function to select the top 10 features that have the highest F-scores.
The SelectKBest function was then fitted to the feature matrix X and the target variable y, which resulted in the transformation of X into X_new, a subset containing only the selected features.

In [ ]:
```
from sklearn.feature_selection import f_regression, SelectKBest

# Assuming X is your features dataframe and y is the target variable
X = df.drop('ARRIVAL_DELAY', axis=1)
y = df['ARRIVAL_DELAY']

# Apply SelectKBest with ANOVA F-test
# You can adjust 'k' to select the number of top features
selector = SelectKBest(score_func=f_regression, k=10)  # Or use k=10 for top 10 features
X_new = selector.fit_transform(X, y)

# Get the selected feature names
selected_features = X.columns[selector.get_support()]

print("Selected features:", selected_features)
```

```
Selected features: Index(['MONTH', 'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY', 'TAXI_OUT', 'WHEELS_OFF',
'WHEELS_ON', 'TAXI_IN', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME'], dtype='object')
```

MODEL BUILDING

1. ***Linear Regression*:**

Purpose: To predict the duration of flight delays based on other numeric features in the dataset.

Insights: Understand the linear relationship between delay time and other features like departure time, day of the week, and distance. This can help identify which factors contribute most significantly to delays.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
predictions = linear_reg.predict(X_test)

# Calculate MAE
mae_linear = mean_absolute_error(y_test, predictions)

# Calculate MSE
mse_linear = mean_squared_error(y_test, predictions)

# Calculate R-squared
r2_linear = r2_score(y_test, predictions)

# Calculate RMSE
rmse_linear = np.sqrt(mse_linear)

print(f"Mean Squared Error: {mean_squared_error(y_test, predictions)}")
print(f"Linear Regression MAE: {mae_linear}")
print(f"Linear Regression MSE: {mse_linear}")
print(f"Linear Regression R-squared: {r2_linear}")
print(f"Linear Regression RMSE: {rmse_linear}")
```

```
Mean Squared Error: 1.2998509230205864
Linear Regression MAE: 0.6334286637183362
Linear Regression MSE: 1.2998509230205864
Linear Regression R-squared: 0.11788267256944718
Linear Regression RMSE: 1.1401100486446851
```

```
# Evaluate Model
# Calculate train and test scores
train_score_linreg = linear_reg.score(X_train, y_train)
test_score_linreg = linear_reg.score(X_test, y_test)

print(f'Train Score: {train_score_linreg:.2%}')
print(f'Test Score: {test_score_linreg:.2%}')
```

```
Train Score: 11.94%
Test Score: 11.79%
```

MODEL BUILDING

2. Random Forest

Purpose: To predict departure delays using a Random Forest regressor to capture complex, non-linear relationships between features.

Insights: Evaluate feature importance and predict delay patterns, aiding in more effective delay mitigation strategies.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Initialize Random Forest regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
```

```python
# Calculate MAE
mae_rf = mean_absolute_error(y_test, rf_predictions)

# Calculate MSE
mse_rf = mean_squared_error(y_test, rf_predictions)

# Calculate R-squared
r2_rf = r2_score(y_test, rf_predictions)

print(f"Random Forest Regression Mean Absolute Error (MAE): {mae_rf}")
print(f"Random Forest Regression Mean Squared Error (MSE): {mse_rf}")
print(f"Random Forest Regression R-squared: {r2_rf}")
```

```
Random Forest Regression Mean Absolute Error (MAE): 0.07007212213787493
Random Forest Regression Mean Squared Error (MSE): 0.02793124366880649
Random Forest Regression R-squared: 0.9810450309488691
```

```python
# Calculate train and test scores
train_score_rf = rf_model.score(X_train, y_train)
test_score_rf = rf_model.score(X_test, y_test)

print(f"Random Forest Regression Train Score: {train_score_rf:.2f}")
print(f"Random Forest Regression Test Score: {test_score_rf:.2f}")

# Store the test score in a dictionary
models["rf"] = test_score_rf
```

```
Random Forest Regression Train Score: 0.99
Random Forest Regression Test Score: 0.98
```

MODEL BUILDING

3. Gradient Boosting Machines (XGBoost)

Purpose: These algorithms can be used for both regression and classification tasks related to flight delays.

Insights: They can model complex relationships between features and delays, often providing superior predictive accuracy and highlighting nonlinear relationships.

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Initialize and train the XGBoost regressor
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', colsample_bytree = 0.3, learning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
xgb_model.fit(X_train, y_train)

# Predicting
xgb_predictions = xgb_model.predict(X_test)
```

```
# Evaluating the model
xgb_mse = mean_squared_error(y_test, xgb_predictions)
print(f"Mean Squared Error for XGBoost: {xgb_mse}")

# Calculate MAE
mae_xgb = mean_absolute_error(y_test, xgb_predictions)

# Calculate MSE
mse_xgb = mean_squared_error(y_test, xgb_predictions)

# Calculate R-squared
r2_xgb = r2_score(y_test, xgb_predictions)

print(f"XGBoost Regression Mean Absolute Error (MAE): {mae_xgb}")
print(f"XGBoost Regression Mean Squared Error (MSE): {mse_xgb}")
print(f"XGBoost Regression R-squared: {r2_xgb}")
```

```
Mean Squared Error for XGBoost: 1.1375265164485966
XGBoost Regression Mean Absolute Error (MAE): 0.6408607292171448
XGBoost Regression Mean Squared Error (MSE): 1.1375265164485966
XGBoost Regression R-squared: 0.22804082160494732
```

```
# Calculate train and test scores
train_score_xgb = xgb_model.score(X_train, y_train)
test_score_xgb = xgb_model.score(X_test, y_test)

print(f"XGBoost Regression Train Score: {train_score_xgb:.2f}")
print(f"XGBoost Regression Test Score: {test_score_xgb:.2f}")

# Store the test score in a dictionary
models["gbr"] = test_score_xgb
```

```
XGBoost Regression Train Score: 0.23
XGBoost Regression Test Score: 0.23
```

MODEL BUILDING

4. Neural Networks and Deep Learning

Purpose: To capture complex nonlinear relationships between various flight attributes and delays.

Insights: Can potentially reveal intricate patterns and interactions between variables that affect delays, although interpretability might be challenging.

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Defining the model
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))  # Input layer and the first hidden layer
model.add(Dense(64, activation='relu'))  # Second hidden layer
model.add(Dense(32, activation='relu'))  # Third hidden layer
model.add(Dense(1, activation='linear'))  # Output layer

# Re-compiling the model
model.compile(loss='mean_squared_error', optimizer='adam')

# Training the model with validation data
history = model.fit(X_train, y_train, epochs=10, batch_size=256, verbose=1, validation_split=0.2)

# Predicting with the model
nn_predictions = model.predict(X_test).flatten()
```

```
Epoch 1/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 1s 874us/step - loss: 1.0415 - val_loss: 0.2343
Epoch 2/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 772us/step - loss: 0.2732 - val_loss: 0.1582
Epoch 3/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 720us/step - loss: 0.2048 - val_loss: 0.1421
Epoch 4/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 704us/step - loss: 0.1668 - val_loss: 0.1318
Epoch 5/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 715us/step - loss: 0.1768 - val_loss: 0.1267
Epoch 6/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 715us/step - loss: 0.2099 - val_loss: 0.1366
Epoch 7/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 711us/step - loss: 0.1257 - val_loss: 0.0964
Epoch 8/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 717us/step - loss: 0.1011 - val_loss: 0.0829
Epoch 9/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 740us/step - loss: 0.1390 - val_loss: 0.0835
Epoch 10/10
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 721us/step - loss: 0.1027 - val_loss: 0.0644
625/625 ━━━━━━━━━━━━━━━━━━━━ 0s 222us/step
```

MODEL BUILDING

4. Neural Networks and Deep Learning

Purpose: To capture complex nonlinear relationships between various flight attributes and delays.

Insights: Can potentially reveal intricate patterns and interactions between variables that affect delays, although interpretability might be challenging.

```python
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import numpy as np

# Preparing the data
X = df_scaled.drop(['DEPARTURE_DELAY', 'ARRIVAL_DELAY'], axis=1)
y = df_scaled['DEPARTURE_DELAY']

# Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Defining the model
model = MLPRegressor()
# Training the model
model.fit(X_train, y_train)

# Predicting with the model
nn_predictions = model.predict(X_test)
```

```python
# Calculating additional metrics
mae = mean_absolute_error(y_test, nn_predictions)
mse = mean_squared_error(y_test, nn_predictions)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, nn_predictions)

print(f"Neural Networks Mean Absolute Error (MAE): {mae:.2f}")
print(f"Neural Networks Mean Squared Error (MSE): {mse:.2f}")
print(f"Neural Networks Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Neural Networks R-squared: {r2:.2f}")
```

```
Neural Networks Mean Absolute Error (MAE): 0.11
Neural Networks Mean Squared Error (MSE): 0.02
Neural Networks Root Mean Squared Error (RMSE): 0.15
Neural Networks R-squared: 0.98
```

```python
# Calculate train and test scores
train_score_nn = model.score(X_train, y_train)
test_score_nn = model.score(X_test, y_test)

print(f'Train Score: {train_score_nn}\nTest Score: {test_score_nn}')

# Store the test score in a dictionary
models["nn"] = test_score_nn
```

```
Train Score: 0.9787707836330136
Test Score: 0.984642785140068
```

MODEL BUILDING

5. Decision Tree Regression

Purpose: Decision tree regression uncovers complex nonlinear relationships between flight attributes and delays.

Insights: It reveals potential patterns in delay-causing factors, though interpreting results can be challenging due to complex models.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Training the Decision Tree model
decision_tree = DecisionTreeRegressor(random_state=42)
decision_tree.fit(X_train, y_train)

# Making predictions on the testing set
predictions = decision_tree.predict(X_test)
```

```
# Calculating evaluation metrics
mae = mean_absolute_error(y_test, predictions)
mse = mean_squared_error(y_test, predictions)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, predictions)

# Printing evaluation metrics
print(f"Decision Tree Mean Absolute Error (MAE): {mae:.2f}")
print(f"Decision Tree Mean Squared Error (MSE): {mse:.2f}")
print(f"Decision Tree Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Decision Tree R-squared: {r2:.2f}")
```

```
Decision Tree Mean Absolute Error (MAE): 0.14
Decision Tree Mean Squared Error (MSE): 0.10
Decision Tree Root Mean Squared Error (RMSE): 0.31
Decision Tree R-squared: 0.94
```

```
# Calculate train and test scores
train_score_dt = decision_tree.score(X_train, y_train)
test_score_dt = decision_tree.score(X_test, y_test)

# Print train and test scores
print(f"Train Score: {train_score_dt:.2f}")
print(f"Test Score: {test_score_dt:.2f}")

# Store the test score in a dictionary
models["dt"] = test_score_dt
```

```
Train Score: 1.00
Test Score: 0.94
```

HYPERPARAMETER TUNING AND CROSS VALIDATION

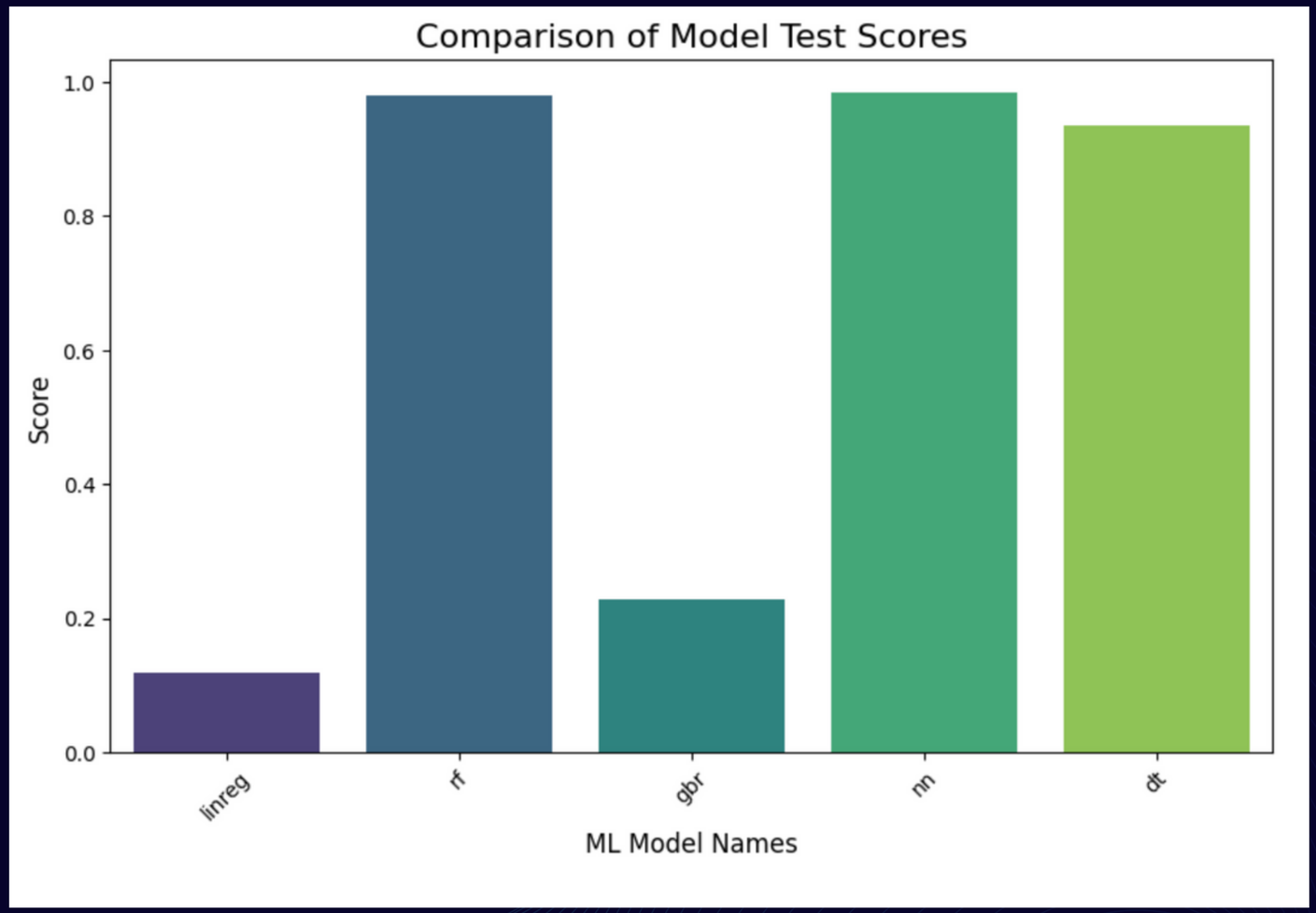
```python
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from keras.models import Sequential
from keras.layers import Dense

# Define the neural network model
def neural_network_model():
    model = Sequential()
    model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

# Define the hyperparameter grids
dt_param_grid = {
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10]
}

gbr_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1]
}

# Define the scoring metrics
scoring_metrics = {
    'r2': make_scorer(r2_score),
    'neg_mae': make_scorer(mean_absolute_error, greater_is_better=False),
    'neg_mse': make_scorer(mean_squared_error, greater_is_better=False)
}

# Create the models
models_ML = {
    'linreg': LinearRegression(),
    'rf': RandomForestRegressor(),
    'gbr': GradientBoostingRegressor(),
    'dt': DecisionTreeRegressor()
}
```

```python
# Perform cross-validation and hyperparameter tuning
for model_name, model in models_ML.items():
    print(f"Cross-validation and Hyperparameter Tuning for {model_name}")

    if model_name == 'linreg':
        cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')
        print(f"Cross-Validation R-squared Scores: {cv_scores}")
        print(f"Mean Cross-Validation R-squared Score: {cv_scores.mean():.2f}")
    else:
        param_grid = globals()[f"{model_name}_param_grid"]  # Access parameter grid using globals()
        grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring=scoring_metrics, cv=5, refit='r2'
        grid_search.fit(X_train, y_train)

        print("Best Hyperparameters: ", grid_search.best_params_)
        print("Best R-squared Score: ", grid_search.best_score_)
        print("Best MAE Score: ", -grid_search.cv_results_['mean_test_neg_mae'][grid_search.best_index_])
        print("Best MSE Score: ", -grid_search.cv_results_['mean_test_neg_mse'][grid_search.best_index_])

    print("\n")
```

```
Cross-validation and Hyperparameter Tuning for linreg
Cross-Validation R-squared Scores: [0.13150251 0.0937311  0.11298277 0.11345436 0.14052559]
Mean Cross-Validation R-squared Score: 0.12


Cross-validation and Hyperparameter Tuning for rf
Best Hyperparameters:  {'max_depth': None, 'min_samples_split': 2, 'n_estimators': 100}
Best R-squared Score:  0.9627477372226693
Best MAE Score:  0.07563080819623916
Best MSE Score:  0.05874295109469785


Cross-validation and Hyperparameter Tuning for gbr
Best Hyperparameters:  {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 300}
Best R-squared Score:  0.9749459651943294
Best MAE Score:  0.07021696501533906
Best MSE Score:  0.03991455964108491


Cross-validation and Hyperparameter Tuning for dt
Best Hyperparameters:  {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best R-squared Score:  0.9284022968499948
Best MAE Score:  0.1301399125792671
Best MSE Score:  0.11195176509650398
```

CONCLUSION

- Neural network model (labeled "nn") achieves highest score, nearly 1.0
- Indicates superior performance compared to other models
- Suggests effective capture of data patterns by neural network
- Consideration of overfitting crucial; assess training score vs. testing score
- Review model complexity alongside performance for informed decision-making



THANK YOU