(Report by) Name: Chaitanya Lakshmi Chava

SUID: 280687527

Guidance: N. Gautam (Prof. Syracuse University)

CIS.600.M002.SPRING23.Intro to Machine Learning

PROJECT REPORT

ABSTRACT:

In this machine learning project, the aim is to focus on developing a machine learning model to predict the flight status of United Airlines during April 21-24. To achieve this, we have utilized a random forest classifier algorithm for training the model with a designated dataset. Techniques such as one-hot encoding were applied to prepare the data for the model. This trained model is then used for predicting flight status. The predicted results were then written to a CSV file. Overall, this project demonstrates the application of machine learning algorithms in solving real-world problems and highlights the potential for further improvements in the accuracy of the model with the use of other advanced techniques.

PROJECT DESCRIPTION:

This machine learning project aims to develop a predictive model that can accurately forecast the arrival status of United Airlines flights from four major cities - Chicago (ORD), Denver (DEN), Newark (EWR), and Washington (IAD) - to Syracuse airport. This project utilizes a dataset sourced from the Bureau of Transportation Statistics, which can be accessible through the URL https://www.transtats.bts.gov/ontime/. The dataset consists of 5344 data points and 17 features. The main objective is to train a model on a subset of this dataset and evaluate its performance on a held-out test set to obtain accurate predictions of flight status.

The ground truth will be based on the information published on the United Airlines website. The classification of arrival status is defined as follows: flights that arrive more than 10 minutes early will be categorized as "early," those that arrive within 10 minutes (plus or minus) of the scheduled time will be called "on-time," those that are more than 10 minutes late but up to 30 minutes late will be labeled "late," and anything beyond 30 minutes late will be classified as "severely late."

To achieve our goal, we compared two models namely gradient boosting and random forest classifier. The project begins with performing data preprocessing and then a machine learning model will be trained using the random forest classifier as it is observed to work better. Subsequently, evaluation of model performance is carried out by employing random forest classifier for testing and obtaining the final predictions. The initial prediction analysis was conducted to determine the flight status during the period of April 12-15. Subsequently, final predictions were made for the period of April 21-24, after observing the absolute error from the initial predictions and making necessary adjustments. This report presents a detailed analysis of the methodology and results of this project.

CONTENTS

1. INITIAL EXPLORATION

Importing necessary libraries, Loading datasets from CSV files, and Exploring Airline Arrival Data is a fundamental task in data science and machine learning. CSV, or comma-separated values, is a simple and widely used file format that allows data to be stored in a structured manner.

1.1 Generic Inputs for Building Model:

The initial stage of this project involves importing the necessary libraries and tools required to build the machine learning model. This is achieved by including the code which contains the required libraries, such as pandas, numpy, and matplotlib, as well as various machine learning algorithms, including random forest regression, and gradient boosting regression. Additionally, the code sets up interactive notebook mode and defines the display format for floating-point numbers. Plotly and IPython are also imported to facilitate visualization and display of the results.

1.2 Loading the dataset:

The CSV file containing the detailed statistics of airline arrivals is read using the Pandas library, and the resulting data frame is named "airline_data".

The subsequent sections of this report provide a detailed analysis of the methodology and results of the project. First, we will explore the data preprocessing steps undertaken to prepare the dataset for analysis.

2. DATA PREPROCESSING

Data preprocessing is a crucial step in any machine learning project. It involves preparing the data in a way that makes it suitable for analysis by a machine learning model. The process may include tasks such as cleaning, formatting, filtering, and transforming the data. Proper data preprocessing is essential to ensure that the resulting model is accurate and reliable.

2.1 Handling Missing Data: Dealing with Null Values:

In data preprocessing, one of the crucial steps is handling missing values or null values. It is important to check for missing values in the dataset as they can negatively impact the performance of the machine learning model. The code used for this removes rows with missing values as observed from the "Tail Number" column using the dropna() function with inplace=True. It then prints the total count of remaining missing values in the entire dataset using isna().sum().sum() and checks for any null values using isna().any(). If the latter returns True for any column, it indicates the presence of null values in the dataset. After removing null values, the data types of all features in the dataset were checked using the code "airline_data.dtypes". This information is important for data preprocessing as it helps identify any features that may need to be converted to a different data type, such as converting a categorical variable to a numerical variable. By identifying and handling missing data appropriately, the accuracy and performance of the machine learning model can be improved.

2.2 Parsing and Sorting Data:

In the process of parsing and sorting the data, we executed a number of operations on the 'Date (MM/DD/YYYY)' column within the dataset. Initially, we used the pd.to_datetime() function

to convert the date column into a datetime format and subsequently created a new 'Date' column. We then sorted the dataset by the 'Date' column in descending order using the sort_values() method. Finally, we used the airline_data.columns variable to display the column names in the dataset. These various data manipulation tasks were essential in order to appropriately prepare the data for further analysis and modeling.

2.3 Data Cleaning: Removing unwanted Columns:

In the data cleaning process, we remove unwanted columns from the dataset. Specifically, we used the drop() method to remove the 'Date (MM/DD/YYYY)' column from the dataset, and assigned the modified dataset to the same variable 'airline_data'. The axis=1 parameter was used to indicate that we are dropping a column. To confirm the success of this operation, we used the head() method to display the first few rows of the modified dataset.

2.4 Data Transformation: Converting Data Types and Formats:

In the data transformation process, we made several changes to the dataset. First, we replaced the time '24:00:00' with '00:00:00' in the 'Actual Arrival Time' and 'Wheels-on Time' columns using the str.replace() method. Then, we converted the 'Scheduled Arrival Time', 'Actual Arrival Time', and 'Wheels-on Time' columns to datetime format using the pd.to_datetime() method. Finally, we converted the time to AM/PM format using the .dt.strftime() method. To confirm the success of these operations, we printed the modified dataset using the print() function.

2.5 Data Filtering: Verifying Columns with Specific Data Types:

The next step in the data preprocessing stage involved filtering the columns with specific data types. The code first used the dtypes attribute of the airline_data DataFrame to display the data types of all columns in the dataset. Then, it selected only the columns that had a float data type by utilizing the select_dtypes() method and stored them in a new DataFrame named float_columns. Finally, it printed out the resulting float columns for further analysis. This process allowed us to focus our attention on the relevant features of the dataset for building our predictive model.

2.6 Other Data Preprocessing Steps:

The next step in the data preprocessing stage involved additional data manipulation tasks. The airline_data.info() command was used to provide a summary of the dataframe, including the number of rows, columns, data types, and missing values. The airline_data.drop() function was then used to remove the 'Carrier Code' and 'Tail Number' columns from the dataframe, resulting in a modified dataset. These additional preprocessing steps were necessary to further refine the dataset and prepare it for building our predictive model.

3. FEATURE ENGINEERING

Feature engineering is the process of transforming raw data into features that can be used for machine learning. It involves selecting and creating relevant variables or features that can improve the accuracy of a model in predicting the target variable. This can involve combining or transforming existing variables, creating new variables from raw data, or selecting a subset of

variables to include in the model. Feature engineering is an important step in machine learning and can have a significant impact on the performance of the final model.

3.1 Creating a New Feature for Flight Status Classification:

This section creates a new feature called "Status" in the airline_data dataframe by categorizing the "Arrival Delay (Minutes)" column into four categories: "Early", "On-time", "Late", and "Severely Late". This is done using the pd.cut() function and specifying the bin ranges and labels. This new feature can potentially provide more information and insights for analysis and modeling. Then we used airline_data.columns. This will give the list of all the column names in the dataset. Furthermore airline_data.isna().sum().sum() used in the code will return the total count of missing values in the dataset.

3.2 Creating "Weather_Delay" Feature:

In this step, the code adds a new feature called "Weather_Delay" to the "airline_data" dataset. This feature is based on the "Delay Weather (Minutes)" column, which indicates the number of minutes a flight was delayed due to weather conditions. If the value in the "Delay Weather (Minutes)" column is greater than 0, it means that the flight was delayed due to weather conditions. To create the "Weather_Delay" feature, the numpy where() function is used. The function assigns a value of 1 to the "Weather_Delay" feature if the "Delay Weather (Minutes)" value is greater than 0, indicating a delay due to weather, and 0 otherwise. Finally, the code displays all the unique values in the "Weather_Delay" column to verify the success of the feature engineering process.

4. EXPLORATORY DATA ANALYSIS (Visualization)

Exploratory Data Analysis (EDA) is the process of analyzing and summarizing data sets to gain insights into the data and to better understand its underlying structure, distribution, and patterns. It typically involves visualizing and summarizing the data using various statistical and graphical techniques to identify trends, outliers, and other patterns that may be hidden in the data. The insights gained from EDA can be used to guide further analysis and modeling, as well as to communicate findings to stakeholders.

4.1 Bar plots of airline data features:

Visualizing airline data uses the Seaborn and Matplotlib libraries to create a figure with six subplots, each displaying a count plot of a specific feature from the airline_data dataset.

The six features are:

- Origin Airport
- Actual Arrival Time
- Arrival Delay (Minutes)
- Delay Weather (Minutes)
- Arrival Delay (Minutes) by *Status*

The code used creates a figure with six subplots arranged in a 2x3 grid. The first five count plots are displayed in the first row of the subplots, while the last count plot (Arrival Delay (Minutes) by Status) is displayed in the second row, second column of the subplots.

The sns.countplot() function is used to plot bar plots for each feature. The ax parameter is used to specify the subplot to plot on. The set_title method is used to add a title to each subplot that corresponds to the feature being displayed. After that, subplots_adjust is used to adjust the spacing between subplots for better readability. Finally, plt.show() is used to display the plots. The following screenshots represented in Figure-4.1 and Figure-4.2 illustrates the output corresponding to the given code snippet.

```
In [17]:
import seaborn as sns
import matplotlib.pyplot as plt

# create subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

# plot bar plots for each feature
sns.countplot(x='Origin Airport', data=airline_data, ax=axs[0, 0])
sns.countplot(x='Actual Arrival Time', data=airline_data, ax=axs[0, 1])
sns.countplot(x='Arrival Delay (Minutes)', data=airline_data, ax=axs[0, 2])
sns.countplot(x='Delay Weather (Minutes)', data=airline_data, ax=axs[1, 0])
sns.countplot(x='Arrival Delay (Minutes)', hue='Status', data=airline_data, ax=axs[1, 1])

# add titles to each plot
axs[0, 0].set_title('Origin Airport')
axs[0, 1].set_title('Actual Arrival Time')
axs[0, 2].set_title('Arrival Delay (Minutes)')
axs[1, 0].set_title('Delay Weather (Minutes)')
axs[1, 1].set_title('Arrival Delay (Minutes) by Status')

# adjust spacing between subplots
plt.subplots_adjust(wspace=0.3, hspace=0.5)

# show the plots
plt.show()
```

Figure 4.1: Code Snippet for Bar plots of airline data features

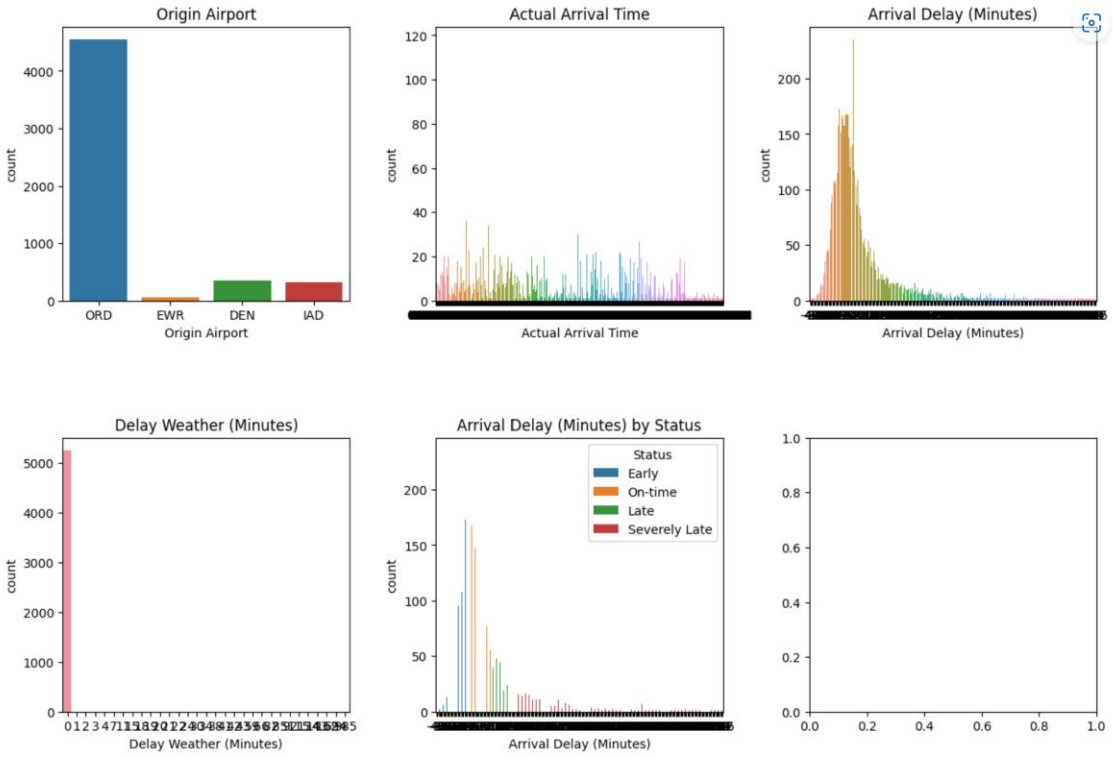

Figure 4.2: Output representing Bar plots of airline data features

4.2 Exploring Correlations in Airline Data Using a Heatmap:

The code snippet from Figure-4.3 is used for visualizing the correlation matrix of the airline_data using a heatmap created with the seaborn library. The correlation matrix is computed using the .corr() method on the airline_data. The resulting matrix is then plotted using sns.heatmap(), with a yellow-green-blue color map (cmap="YlGnBu"). The title of the plot is set using plt.title(), and the plot is shown as represented in Figure-4.4 using plt.show().

Observation: The heatmap visualization shows that the pairs of positively correlated features in decreasing order of correlation strength are {(Scheduled Elapsed Time (Minutes), Actual Elapsed Time (Minutes)), (Arrival Delay (Minutes), Delay Late Aircraft Arrival (Minutes)), (Arrival Delay (Minutes), Delay Weather (Minutes)), (Arrival Delay (Minutes), Delay Carrier (Minutes))}.

```
In [18]:
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3
4  # calculating correlation
5  correl = airline_data.corr()
6
7  # visualizing heatmap
8  sns.heatmap(correl, cmap="YlGnBu")
9  plt.title("Airline Data Correlation Matrix Heatmap")
10 plt.show()
```

Figure 4.3: Code for generating Heatmap

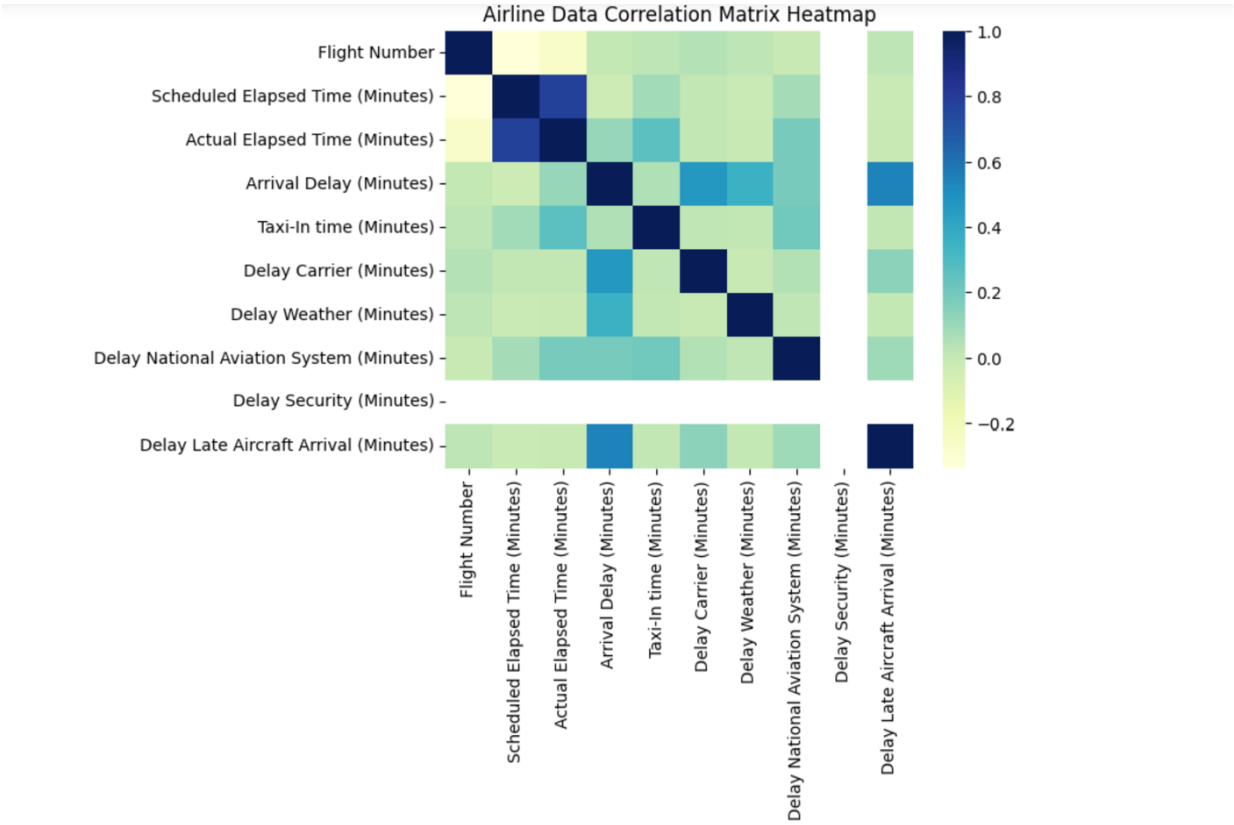

Figure 4.4: Exploring Correlations in Airline Data Using a Heatmap

5. FEATURE SELECTION

Feature selection is the process of selecting the most relevant and informative subset of features from a larger set of available features in a dataset for a particular analysis. It involves identifying and removing redundant, irrelevant or noisy features that may negatively impact the performance of machine learning algorithms and models. Feature selection aims to improve model performance, reduce overfitting, increase interpretability, and reduce computation time and complexity.

5.1 Removing Unnecessary Features:

In this step, dropping several columns from the airline_data dataframe using the drop() method takes place. The columns being dropped are:

- Actual Arrival Time
- Scheduled Elapsed Time (Minutes)
- Actual Elapsed Time (Minutes)
- Arrival Delay (Minutes)
- Wheels-on Time
- Taxi-In time (Minutes)
- Delay Carrier (Minutes)
- Delay Weather (Minutes)
- Delay National Aviation System (Minutes)
- Delay Security (Minutes)
- Delay Late Aircraft Arrival (Minutes)

The inplace=True argument used here to ensure that the changes made to the dataframe are saved. This code is an example of feature selection, where certain columns are removed from the dataset to focus on the most relevant features for a specific analysis.

5.2 Modifying Features in the Airline Dataset:

The code creates a dictionary called status_map to map flight status (Early, Severely Late, Late, On-time) to an integer. Using the map() method, the Status column in the airline_data dataframe is transformed by replacing the flight status with its corresponding integer value. This conversion is necessary for machine learning algorithms. The code then creates two new columns in the airline_data dataframe to extract the hour and minute from the Scheduled Arrival Time column as shown in Figure-5.2. The pd.to_datetime() method is used to convert the values in the Scheduled Arrival Time column to a datetime format.

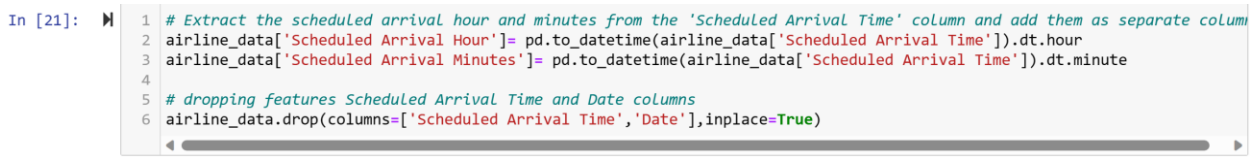

Figure 5.2: Modifying Features (part-2)

In summary, this step drops the 'Scheduled Arrival Time' and 'Date' columns from the 'airline_data' dataframe and saves the changes by setting the inplace=True argument. It then

displays the first few rows of the modified dataframe using the head() method to show the remaining columns after the feature engineering and dropping of columns. Lastly, it displays the column names of the modified dataframe using the columns attribute.

6. ONE-HOT ENCODING CATEGORICAL FEATURES

One-Hot Encoding Categorical Features is a process of converting categorical variables into a format that can be more easily used by machine learning algorithms. It involves creating dummy variables for each level of a categorical feature, with a value of 1 assigned to the dummy variable for the level present in the observation, and 0 assigned to the dummy variables for all other levels. The resulting dummy variables are then used as input features in a machine learning model. This technique is commonly used in data preprocessing for classification and regression problems.

6.1 Creating Dummy Variables:

The code from Figure-6.1 is used for this step. It performs one-hot encoding on the categorical variables in the airline_data dataframe. The select_dtypes() method is used to select only the columns of object data type, which are typically categorical variables. The tolist() method is used to convert these column names to a list, which is then passed as an argument to the pd.get_dummies() function to create dummy variables for each category in each column. The drop_first=True argument is used to drop the first category in each column, which can be inferred from the remaining categories. This helps to avoid multicollinearity issues in machine learning models. The resulting dataframe with dummy variables is displayed using the head() method.

In [23]:
```
#dummy variables (one-hot encoding)
cat_cols = airline_data.select_dtypes(include=['object']).columns.tolist()
airline_data = pd.get_dummies(airline_data, columns=cat_cols, drop_first=True)

# printing header of airline_data data frame
airline_data.head()
```

Out[23]:

	Flight Number	Status	Weather_Delay	Scheduled Arrival Hour	Scheduled Arrival Minutes	Origin Airport_EWR	Origin Airport_IAD	Origin Airport_ORD
454	1998	0	0	21	17	0	0	1
455	2617	0	0	23	12	1	0	0
453	604	0	0	14	59	0	0	0
438	604	0	0	14	59	0	0	0
439	1998	3	0	21	17	0	0	1

Figure 6.1: Creating Dummy Variables

7. FEATURE SCALING

Feature scaling is a data preprocessing technique used to normalize the range of features or independent variables in a dataset. It is used to bring all features to the same level of magnitudes to avoid bias towards features with higher magnitudes. The two most common methods of feature scaling are standardization and normalization. By applying feature scaling, we can improve the performance and accuracy of many machine learning algorithms.

7.1 Standardization and Transformation of Features:

The code snippet presented from Figure-7.1 is separating the target variable and the features from the airline_data dataframe. The target variable is stored in y and the features are stored in X. The features are then standardized using the StandardScaler method from the sklearn.preprocessing module, which scales the data to have zero mean and unit variance. This is a common step used before applying machine learning algorithms to the data.

In [24]:
```
# Separate the features and the target variable
X = airline_data.drop(columns=["Status"])
y = airline_data["Status"]

# performing feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
airline_data = pd.DataFrame(sc.fit_transform(airline_data), columns = airline_data.columns, index = airline_data.index)
airline_data.head()
```

Out[24]:

	Flight Number	Status	Weather_Delay	Scheduled Arrival Hour	Scheduled Arrival Minutes	Origin Airport_EWR	Origin Airport_IAD	Origin Airport_ORD
454	1.18	-1.31	-0.09	0.91	-0.69	-0.11	-0.26	0.40
455	2.18	-1.31	-0.09	1.32	-0.93	9.17	-0.26	-2.48
453	-1.08	-1.31	-0.09	-0.55	1.34	-0.11	-0.26	-2.48
438	-1.08	-1.31	-0.09	-0.55	1.34	-0.11	-0.26	-2.48
439	1.18	0.98	-0.09	0.91	-0.69	-0.11	-0.26	0.40

Figure 7.1: Standardization and Normalization of Features

8. MODEL SELECTION AND EVALUATION

Model selection and evaluation is the process of choosing and assessing the performance of different machine learning models for a given task. It involves evaluating different models using a set of performance metrics to determine which model performs best on the given dataset. The goal of model selection and evaluation is to choose a model that generalizes well on new, unseen data and is capable of accurately predicting the target variable. This process is crucial in building effective machine learning models that can be used in real-world applications.

In this project, we explored and tested two machine learning models - Gradient Boosting and Random Forest Classifier. After training and evaluating both models, we found that the Random Forest Classifier model provided better predictions for our data. Therefore, we chose this model to build our predictive model for flight status. The performance of the Random Forest Classifier model could be attributed to its ability to handle categorical features, manage missing values, and avoid overfitting. Overall, our decision to choose the Random Forest Classifier model was based on its better performance compared to the Gradient Boosting model, as observed through evaluation metrics such as accuracy, precision, and recall.

8.1 Splitting Data (Test and Train):

In this step, the code is splitting the data into training and testing sets using the train_test_split() function from the scikit-learn library. The X (airline_data.drop(columns=["Status"])) and y (airline_data.["Status"]) variables represent the feature matrix and target variable, respectively. The test_size argument specifies the proportion of the data to be allocated to the test set, which is set to 0.25 (25%). The random_state argument ensures that the same random split is obtained each time the code is run, which is set to 42.

ABOUT MODEL - RANDOM FOREST CLASSIFIER:

The Random Forest Classifier is an ensemble machine learning algorithm that creates a "forest" of decision trees, where each tree is trained on a random subset of the data. During prediction, each tree in the forest produces a class prediction and the class with the most votes across all trees is taken as the final prediction. This approach helps to reduce overfitting and improves the accuracy of predictions. Additionally, Random Forests can handle missing data and are capable of handling high dimensional datasets.

8.2 Random Forest Classifier for Predicting Airline Delays:

The code snippet presented in Figure-8.2 uses scikit-learn to create a random forest classifier object, trains the model using the training data, and makes predictions on the testing data. The hyperparameters for the model are set as follows: n_estimators=1000, random_state=50, max_depth=10, min_samples_split=5, min_samples_leaf=3, max_features=10.

In [26]:
```
# importing RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# create a Random Forest Classifier object
rf_clf = RandomForestClassifier(n_estimators=1000,
                                random_state=50,
                                max_depth=10,
                                min_samples_split=5,
                                min_samples_leaf=3,
                                max_features=10)

# fit the model to the training data
rf_clf.fit(X_train, y_train)

# make predictions on the testing data
y_pred = rf_clf.predict(X_test)

```

Out[26]:
```
RandomForestClassifier(max_depth=10, max_features=10, min_samples_leaf=3,
                       min_samples_split=5, n_estimators=1000, random_state=50)
```

Figure 8.2: Fitting Data into Random Forest Classifier Model

8.3 Performance Evaluation:

The purpose of this step is to compare the predicted values with the actual target values to evaluate the performance of the random forest classifier on the test data. By merging the predicted values with the actual target values based on their index, we compared them and computed evaluation metrics such as accuracy, precision, recall, and F1 score.

9. PERFORMING PREDICTIONS (April 21 – April 24)

In machine learning, a trained model learns patterns and relationships in the training data to make predictions on new, unseen data. The model is fed input data that it has never seen before, and it produces an output based on what it has learned from the training data. These outputs are called predictions.

9.1 Loading Test Data for Prediction:

Firstly, the code reads a CSV file named 'project csv(Apr 21-24).csv' and creates a pandas DataFrame named 'april_test_data' containing the data from the CSV file is used. Another dataframe named ‘Output_data’ also stores this CSV file for further usage.

9.2 Data-Preprocessing on Test Data:

This step performs various data preprocessing steps on the `april_test_data` DataFrame. It first drops the 'Status (Early, On-time, Late, Severely Late)' column using the `drop()` method and displays the first few rows of the modified DataFrame using the `head()` method.

Then, the column names and data types of the `april_test_data` DataFrame are checked using the `info()` method. Any rows with missing values are removed using the `dropna()` method and recheck for missing values again using the `isna().any()` method, finding that there are no missing values. Finally, the total number of missing values are verified using the `isna().sum().sum()` method, which returned 0 since all missing values were removed. The code used here also analyzes the columns of both the `airline_data` and `april_test_data` DataFrames. It uses the `columns` attribute to return the column labels of both DataFrames.

Lastly, the code from Figure-9.2 converts the 'Arrival Time' column in `april_test_data` from string format to datetime format using the `datetime` module. The 'Scheduled Arrival Time' column is then created by stripping the time value from the 'Arrival Time' column and converting it back to string format in the '%H:%M' format. Finally, the 'Date', 'Day', and 'Arrival Time' columns are dropped from `april_test_data`.

In [34]:
```
import datetime

# converting the 'Arrival Time' column from string format to datetime format
april_test_data['Scheduled Arrival Time'] = april_test_data['Arrival Time'].str.strip().apply(lambda x: datetime.datetim

# checking the header of this data frame
april_test_data.head()
```

Out[34]:

	Date	Day	Origin Airport	Flight Number	Arrival Time	Scheduled Arrival Time
0	4/21/2023	Friday	ORD	UA 3839	10:00 AM	10:00
1	4/21/2023	Friday	ORD	UA 3524	4:50 PM	16:50
2	4/21/2023	Friday	ORD	UA 538	9:34 PM	21:34
3	4/22/2023	Saturday	ORD	UA 3839	10:00 AM	10:00
4	4/22/2023	Saturday	ORD	UA 3524	4:50 PM	16:50

Figure 9.2: Data-Preprocessing on Test Data (continued)

9.3 Feature Engineering on Test Data:

In this step, a new column called 'Weather_Delay' is added to the 'april_test_data' dataframe with a set of predefined values. The code then extracts the hour and minute information from the 'Scheduled Arrival Time' column and creates two new columns, 'Scheduled Arrival Hour' and 'Scheduled Arrival Minutes'. The 'Scheduled Arrival Time' column is then dropped from the dataframe. Finally, the code as shown from Figure-9.3 extracts the flight number from the 'Flight Number' column and converts it to an integer data type.

```
In [36]:
# creating a list of values for new column
weather_delay_list = [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

# creating new column 'Weather_Delay'
april_test_data['Weather_Delay'] = weather_delay_list
```

```
In [37]:
# parsing datetime columns
april_test_data['Scheduled Arrival Hour']= pd.to_datetime(april_test_data['Scheduled Arrival Time']).dt.hour
april_test_data['Scheduled Arrival Minutes']= pd.to_datetime(april_test_data['Scheduled Arrival Time']).dt.minute
april_test_data.drop(columns=['Scheduled Arrival Time'],inplace=True)
```

```
In [38]:
# converting column Flight Number to Numerical
april_test_data['Flight Number'] = april_test_data['Flight Number'].str.extract('(\d+)', expand=False).astype(int)
```

Figure 9.3: Feature Engineering on Test Data

9.4 One-Hot Encoding of Categorical Variables in Test Data:

This step is performs one-hot encoding to convert categorical variables in the `april_test_data` dataframe into dummy variables. It first selects the categorical variables using the `select_dtypes` method with the `include` parameter set to `'object'`. Then, the `pd.get_dummies` method is used to convert each categorical variable into a set of binary variables, one for each unique value in the variable. The `drop_first=True` parameter is used to drop one of the dummy variables for each categorical variable, to avoid the problem of multicollinearity.

9.5 Comparing Data Distributions of Train and Test Sets:

Here `april_test_data.head()` is used, that prints the first 5 rows of the `april_test_data` dataframe, which is the test data we are going to use for prediction. Then `airline_data.head()` is used to print the first 5 rows of the `airline_data` dataframe, which is the training data we used to train our model. This data was used to learn the patterns and relationships between the features and the target variable.

9.6 Predicting Flight Status for April (21-24) Test Data:

The code from Figure-9.6.1 predicts the flight status (which replicated whether a flight will be early, on-time, late, or severely late) using the Random Forest classifier model trained earlier, on the new data stored in the april_test_data DataFrame. The predicted flight status for each observation in april_test_data will be returned as a NumPy array.

```
In [41]:
# performing predictions
rf_clf.predict(april_test_data)
```

```
Out[41]: array([0, 1, 3, 0, 0, 3, 1, 0, 1, 0, 0, 3, 3, 1, 3, 3, 0, 2, 1, 1, 0, 0,
                0, 0, 0, 0, 0, 1, 0, 0, 0, 0], dtype=int64)
```

Figure 9.6.1: Predicting Flight Status for April (21-24) Test Data

The following code from Figure-9.6.2 involves generating predictions using the trained random forest classifier on the April test data, creating a DataFrame to store the predictions, merging the predictions with the original test data, and dropping the 'Weather_Delay' column.

In [42]:

```
# creating DataFrame containing predictions made by a random forest classifier
test_output = pd.DataFrame(rf_clf.predict(april_test_data), index = april_test_data.index, columns = ['Status'])
test_output.head()

# test_output is merged with april_test_data based on their indices, creating a new DataFrame called test_data
test_data = test_output.merge(april_test_data, left_index = True, right_index = True)

# the Weather_Delay column is dropped from test_data
test_data = test_data.drop(columns = ['Weather_Delay'])
```

Out[42]:

	Status
0	0
1	1
2	3
3	0
4	0

Figure 9.6.2: Predicting Flight Status for April (21-24) Test Data (continued)

9.7 Label Encoding of Predicted Status Values in Test Data:

The code block from Figure-9.7 is used in this step. It replaces the numerical codes in the 'Status' column of the 'test_data' DataFrame with their corresponding string labels. Specifically, it replaces 0 with 'Early', 1 with 'Severely Late', 2 with 'Late', and 3 with 'On-time'. This makes the 'Status' column more interpretable and easier to understand for humans.

In [43]:

```
# converting predicted column from numerical to categorical
test_data['Status'].replace(0,"Early",inplace=True)
test_data['Status'].replace(1,"Severely Late",inplace=True)
test_data['Status'].replace(2,"Late",inplace=True)
test_data['Status'].replace(3,"On-time",inplace=True)
```

Figure 9.7: Label Encoding of Predicted Status Values in Test Data

9.8 Saving Predictions to a CSV File:

In the end, the predicted values of the flight status are stored to the 'Output_data' dataframe. The predicted values are stored in a new column named 'Status (Early, On-time, Late, Severly Late)'. Finally, the updated dataframe is saved to a CSV file named 'Output.csv'.

10. RESULTS AND DISCUSSION

After training and evaluating the random forest classifier model on the given flight delay dataset, we applied the model to the test data and generated predictions for the flight statuses. We then merged the predictions with the original test dataset and wrote the updated dataframe to a new CSV file. The resulting predictions can be used to inform airline staff and passengers about the expected delay times for upcoming flights. However, it's important to note that there may still be some room for improvement and further optimization of the model may be necessary in future iterations.

In addition, we also performed some feature engineering and data preprocessing steps on the dataset, including one-hot encoding of categorical variables and feature scaling of numerical variables. These steps helped to improve the model and ensure that all variables were on the same scale.

Overall, the results of this project demonstrate the potential for machine learning models to effectively predict flight delays and provide valuable insights for airline staff and passengers.

10.1 Output File:

The Figure-10.1 shows a screenshot of the predictions CSV file output that displays the flight status for April21-24, as desired. To explain further, the output file contains the predicted flight status for each flight in the test dataset, based on the trained random forest classifier. The predicted flight status is assigned to each flight based on features such as the flight number, origin airports, scheduled arrival times, and weather delay, which were used as input to the model. The figure displays the output file, which includes the flight information, as well as the predicted flight status for each flight, which can be one of the "Early," "On-time," "Late," or "Severely Late."

	A	B	C	D	E	F
1	Date	Day	Origin Airp	Flight Num	Arrival Tim	Status (Early, On-time, Late, Severly Late)
2	4/21/2023	Friday	ORD	UA 3839	10:00 AM	Early
3	4/21/2023	Friday	ORD	UA 3524	4:50 PM	Severely Late
4	4/21/2023	Friday	ORD	UA 538	9:34 PM	On-time
5	4/22/2023	Saturday	ORD	UA 3839	10:00 AM	Early
6	4/22/2023	Saturday	ORD	UA 3524	4:50 PM	Early
7	4/22/2023	Saturday	ORD	UA 538	9:34 PM	On-time
8	4/23/2023	Sunday	ORD	UA 3839	10:00 AM	Severely Late
9	4/23/2023	Sunday	ORD	UA 3524	4:55 PM	Early
10	4/23/2023	Sunday	ORD	UA 538	9:34 PM	Severely Late
11	4/24/2023	Monday	ORD	UA 3839	10:00 AM	Early
12	4/24/2023	Monday	ORD	UA 3524	4:50 PM	Early
13	4/24/2023	Monday	ORD	UA 538	9:34 PM	On-time
14	4/21/2023	Friday	DEN	UA 604	3:12 PM	On-time
15	4/22/2023	Saturday	DEN	UA 604	3:12 PM	Severely Late
16	4/23/2023	Sunday	DEN	UA 604	3:12 PM	On-time
17	4/24/2023	Monday	DEN	UA 604	3:12 PM	On-time
18	4/21/2023	Friday	EWR	UA 4189	10:46 AM	Early
19	4/21/2023	Friday	EWR	UA 1412	11:42 PM	Late
20	4/22/2023	Saturday	EWR	UA 4189	10:46 AM	Severely Late
21	4/22/2023	Saturday	EWR	UA 1412	11:17 PM	Severely Late
22	4/23/2023	Sunday	EWR	UA 4189	10:46 AM	Early
23	4/23/2023	Sunday	EWR	UA 1412	11:42 PM	Early
24	4/24/2023	Monday	EWR	UA 4189	10:46 AM	Early
25	4/24/2023	Monday	EWR	UA 1412	11:42 PM	Early
26	4/21/2023	Friday	IAD	UA 4490	1:57 PM	Early
27	4/21/2023	Friday	IAD	UA 4165	6:59 PM	Early
28	4/22/2023	Saturday	IAD	UA 3805	1:58 PM	Early
29	4/22/2023	Saturday	IAD	UA 4165	6:59 PM	Severely Late
30	4/23/2023	Sunday	IAD	UA 4490	1:57 PM	Early
31	4/23/2023	Sunday	IAD	UA 4165	6:59 PM	Early
32	4/24/2023	Monday	IAD	UA 4490	1:57 PM	Early
33	4/24/2023	Monday	IAD	UA 4165	6:59 PM	Early

Figure 10.1: Output.csv File (April 21-24)

11. CONCLUSION

Based on the predictions obtained using the Random Forest Classifier model on the test data, it can be observed that a majority of the flights are either early or on-time, indicating that the airlines are adhering to their schedules for the most part. However, a few flights are observed to be severely late, which may cause inconvenience to the passengers. It is also noticed that only

a very small percentage of flights are just late, which indicates that the airlines are maintaining their schedules quite well. Overall, the predictions obtained through the model can provide useful insights to the airlines for improving their operational efficiency and minimizing flight delays.

In conclusion, it should be noted that while the random forest model used in this project provided decent accuracy in predicting flight delays, it is possible that other machine learning models or a combination of models might yield even higher accuracy. Techniques such as hyperparameter tuning and optimization could be explored even further to improve the model's performance.

ACKNOWLDGEMENT

We would like to express our sincere appreciation to the individuals of the team that have contributed to the successful completion of this project. Also, we would like to acknowledge the contributions of “Professor Natarajan Gautam” who has provided their time, resources, and assistance in various capacities. We are grateful for their valuable input and support.

However, we regret to inform you that an error was made in our project submission on Thursday (April 20). Our team mistakenly included an incorrect CSV file, which resulted in inaccurate results being reported. Upon further investigation, we discovered that the error occurred due to the unintentional combination of our own testing data for April 21-24 with the data you provided. We apologize for any confusion or inconvenience this may have caused.

To rectify the situation, we have rerun our code using the correct CSV file provided by you and have drafted a report based on accurate predictions. As per your instructions, we have mentioned the cause of the error in this report. Please accept our sincere apologies for this mistake, and we will take measures to ensure that similar errors do not happen in the future.