onpremises

November 3, 2024

1 Problem: Predicting Airplane Delays

The goals of this notebook are:

- Process and create a dataset from downloaded ZIP files
- Exploratory data analysis (EDA)
- Establish a baseline model and improve it

1.1 Introduction to business scenario

You work for a travel booking website that is working to improve the customer experience for flights that were delayed. The company wants to create a feature to let customers know if the flight will be delayed due to weather when the customers are booking the flight to or from the busiest airports for domestic travel in the US.

You are tasked with solving part of this problem by leveraging machine learning to identify whether the flight will be delayed due to weather. You have been given access to the a dataset of on-time performance of domestic flights operated by large air carriers. You can use this data to train a machine learning model to predict if the flight is going to be delayed for the busiest airports.

1.1.1 Dataset

The provided dataset contains scheduled and actual departure and arrival times reported by certified US air carriers that account for at least 1 percent of domestic scheduled passenger revenues. The data was collected by the Office of Airline Information, Bureau of Transportation Statistics (BTS). The dataset contains date, time, origin, destination, airline, distance, and delay status of flights for flights between 2014 and 2018. The data are in 60 compressed files, where each file contains a CSV for the flight details in a month for the five years (from 2014 - 2018). The data can be downloaded from this link: [https://ucstaff-my.sharepoint.com/:f:/g/personal/ibrahim_radwan_canberra_edu_au/Er0nVreXmihEmtMz5qC5kVIB81-ugSusExPYdcyQTglfLg?e=bNO312]. Please download the data files and place them on a relative path. Dataset(s) used in this assignment were compiled by the Office of Airline Information, Bureau of Transportation Statistics (BTS), Airline On-Time Performance Data, available with the following link: [https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ].

2 Step 1: Problem formulation and data collection

Start this project off by writing a few sentences below that summarize the business problem and the business goal you're trying to achieve in this scenario. Include a business metric you would like your team to aspire toward. With that information defined, clearly write out the machine learning problem statement. Finally, add a comment or two about the type of machine learning this represents.

2.0.1 1. Determine if and why ML is an appropriate solution to deploy.

Machine Learning solution is appropriate for this problem, due to the availability of large datasets, which could effectively make use of existing Machine Learning algorithms that performs exceptionally well on such large datasets, and also availability of cheap compute power for training, testing and validating the model.

2.0.2 2. Formulate the business problem, success metrics, and desired ML output.

The business problem we are trying to solve is to determine whether the flight will be delayed or not, so that the customers can make their booking choices accordingly. This could potentially lead to improved customer satisfaction rates. The accuracy of the model predictions would be the primary metric used for measuring the success. The desired ML output is to determine whether a flight will be delayed or not.

2.0.3 3. Identify the type of ML problem you're dealing with.

The Machine learning problem we are dealing with is a binary classification problem, which comes under the Supervised Machine Learning.

2.0.4 Setup

Now that we have decided where to focus our energy, let's set things up so you can start working on solving the problem.

[1]:
```
import os
# from pathlib2 import Path
from zipfile import ZipFile
import time

import pandas as pd
import numpy as np
import subprocess

import matplotlib.pyplot as plt
import seaborn as sns

sns.set()

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline

# <please add any other library or function you are aiming to import here>
```

3 Step 2: Data preprocessing and visualization

In this data preprocessing phase, you should take the opportunity to explore and visualize your data to better understand it. First, import the necessary libraries and read the data into a Pandas dataframe. After that, explore your data. Look for the shape of the dataset and explore your columns and the types of columns you're working with (numerical, categorical). Consider performing basic statistics on the features to get a sense of feature means and ranges. Take a close look at your target column and determine its distribution.

3.0.1 Specific questions to consider

1. What can you deduce from the basic statistics you ran on the features?
2. What can you deduce from the distributions of the target classes?
3. Is there anything else you deduced from exploring the data?

Start by bringing in the dataset from an Amazon S3 public bucket to this notebook environment.

[14]:
```
# download the files

# <note: make them all relative, absolute path is not accepted>
dropbox_link = 'https://www.dropbox.com/scl/fi/tjqxgop82n4nnm2w78j3d/
↪data_compressed.zip?rlkey=c205jkvrk8rputgxz1zl5ottq&st=lkat0xua&dl=0'
output_path = 'data/compressed.zip'
zip_path = './data/compressed/'
base_path = './'
csv_base_path = './data/csv/'

!mkdir -p {csv_base_path}
```

```
The syntax of the command is incorrect.
```

[16]:
```
# Count the number of zip files in the directory
zip_files = [f for f in os.listdir(zip_path) if f.endswith(".zip")]
num_zip_files = len(zip_files)


print(f"Number of zip files: {num_zip_files}")
```

```
Number of zip files: 60
```

Extract CSV files from ZIP files

[17]:
```
def zip2csv(zipFile_name, file_path):
    """
    Extract csv from zip files
    zipFile_name: name of the zip file
    file_path : name of the folder to store csv
    """
```

```python
    try:
        with ZipFile(zipFile_name, "r") as z:
            print(f"Extracting {zipFile_name} to {file_path}")
            z.extractall(path=file_path)
    except Exception as e:
        print(f"zip2csv failed for {zipFile_name}: error: {e}")


for file in zip_files:
    zip2csv(zip_path + file, csv_base_path)

print("Files Extracted")
```

```
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_1.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_10.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_11.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_12.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_2.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_3.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_4.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_5.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_6.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_7.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_8.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2014_9.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_1.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_10.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_11.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_12.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_2.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
```

```
present_2015_3.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_4.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_5.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_6.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_7.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_8.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2015_9.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_1.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_10.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_11.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_12.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_2.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_3.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_4.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_5.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_6.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_7.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_8.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2016_9.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_1.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_10.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_11.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_12.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_2.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
```

```
present_2017_3.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_4.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_5.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_6.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_7.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_8.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2017_9.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_1.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_10.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_11.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_12.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_2.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_3.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_4.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_5.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_6.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_7.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_8.zip to ./data/csv/
Extracting ./data/compressed/On_Time_Reporting_Carrier_On_Time_Performance_1987_
present_2018_9.zip to ./data/csv/
Files Extracted
```

[18]:

```python
# Count the number of CSV files in the directory
csv_files = [f for f in os.listdir(csv_base_path) if f.endswith(".csv")]
num_csv_files = len(csv_files)

print(f"Number of CSV files extracted: {num_csv_files}")
```

```
Number of CSV files extracted: 60
```

Before loading the CSV file, read the HTML file from the extracted folder. This HTML file includes the background and more information on the features included in the dataset.

[19]:
```python
from IPython.display import IFrame

IFrame(src=os.path.relpath(f"{csv_base_path}readme.html"), width=1000,␣
 ↪height=600)
```

[19]:
```
<IPython.lib.display.IFrame at 0x2e2cfdc5810>
```

Load sample CSV Before combining all the CSV files, get a sense of the data from a single CSV file. Using Pandas, read the `On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.csv` file first. You can use the Python built-in `read_csv` function (documentation).

[20]:
```python
df_temp = pd.read_csv(
    csv_base_path
    + "On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.csv"
)

df_temp.head()
```

[20]:
```
   Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate Reporting_Airline  \
0  2018        3      9           3          1  2018-09-03                9E
1  2018        3      9           9          7  2018-09-09                9E
2  2018        3      9          10          1  2018-09-10                9E
3  2018        3      9          13          4  2018-09-13                9E
4  2018        3      9          14          5  2018-09-14                9E

   DOT_ID_Reporting_Airline IATA_CODE_Reporting_Airline Tail_Number  …  \
0                     20363                          9E      N908XJ  …
1                     20363                          9E      N315PQ  …
2                     20363                          9E      N582CA  …
3                     20363                          9E      N292PQ  …
4                     20363                          9E      N600LR  …

   Div4TailNum  Div5Airport  Div5AirportID  Div5AirportSeqID Div5WheelsOn  \
0          NaN          NaN            NaN               NaN          NaN
1          NaN          NaN            NaN               NaN          NaN
2          NaN          NaN            NaN               NaN          NaN
3          NaN          NaN            NaN               NaN          NaN
4          NaN          NaN            NaN               NaN          NaN

  Div5TotalGTime Div5LongestGTime  Div5WheelsOff Div5TailNum  Unnamed: 109
0            NaN              NaN            NaN         NaN           NaN
1            NaN              NaN            NaN         NaN           NaN
2            NaN              NaN            NaN         NaN           NaN
3            NaN              NaN            NaN         NaN           NaN
4            NaN              NaN            NaN         NaN           NaN
```

```
[5 rows x 110 columns]
```

Question: Print the row and column length in the dataset, and print the column names.

[21]:
```python
df_shape = df_temp.shape
print(f"Rows and columns in one csv file is {df_shape}")
```

```
Rows and columns in one csv file is (585749, 110)
```

Question: Print the first 10 rows of the dataset.

[22]:
```python
# Enter your code here
df_temp.head(10)
```

[22]:
```
   Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate Reporting_Airline  \
0  2018        3      9           3          1  2018-09-03                9E
1  2018        3      9           9          7  2018-09-09                9E
2  2018        3      9          10          1  2018-09-10                9E
3  2018        3      9          13          4  2018-09-13                9E
4  2018        3      9          14          5  2018-09-14                9E
5  2018        3      9          16          7  2018-09-16                9E
6  2018        3      9          17          1  2018-09-17                9E
7  2018        3      9          20          4  2018-09-20                9E
8  2018        3      9          21          5  2018-09-21                9E
9  2018        3      9          23          7  2018-09-23                9E

   DOT_ID_Reporting_Airline IATA_CODE_Reporting_Airline Tail_Number  …  \
0                     20363                          9E      N908XJ  …
1                     20363                          9E      N315PQ  …
2                     20363                          9E      N582CA  …
3                     20363                          9E      N292PQ  …
4                     20363                          9E      N600LR  …
5                     20363                          9E      N316PQ  …
6                     20363                          9E      N916XJ  …
7                     20363                          9E      N371CA  …
8                     20363                          9E      N601LR  …
9                     20363                          9E      N906XJ  …

   Div4TailNum  Div5Airport  Div5AirportID  Div5AirportSeqID Div5WheelsOn  \
0          NaN          NaN            NaN               NaN          NaN
1          NaN          NaN            NaN               NaN          NaN
2          NaN          NaN            NaN               NaN          NaN
3          NaN          NaN            NaN               NaN          NaN
4          NaN          NaN            NaN               NaN          NaN
5          NaN          NaN            NaN               NaN          NaN
6          NaN          NaN            NaN               NaN          NaN
7          NaN          NaN            NaN               NaN          NaN
8          NaN          NaN            NaN               NaN          NaN
9          NaN          NaN            NaN               NaN          NaN
```

```
  Div5TotalGTime Div5LongestGTime  Div5WheelsOff Div5TailNum  Unnamed: 109
0            NaN              NaN            NaN         NaN           NaN
1            NaN              NaN            NaN         NaN           NaN
2            NaN              NaN            NaN         NaN           NaN
3            NaN              NaN            NaN         NaN           NaN
4            NaN              NaN            NaN         NaN           NaN
5            NaN              NaN            NaN         NaN           NaN
6            NaN              NaN            NaN         NaN           NaN
7            NaN              NaN            NaN         NaN           NaN
8            NaN              NaN            NaN         NaN           NaN
9            NaN              NaN            NaN         NaN           NaN

[10 rows x 110 columns]
```

Question: Print all the columns in the dataset. Use `<dataframe>.columns` to view the column names.

[23]:
```python
print(f"The column names are :")
print("#########")
for col in df_temp.columns:
    print(col)
```

```
The column names are :
#########
Year
Quarter
Month
DayofMonth
DayOfWeek
FlightDate
Reporting_Airline
DOT_ID_Reporting_Airline
IATA_CODE_Reporting_Airline
Tail_Number
Flight_Number_Reporting_Airline
OriginAirportID
OriginAirportSeqID
OriginCityMarketID
Origin
OriginCityName
OriginState
OriginStateFips
OriginStateName
OriginWac
DestAirportID
DestAirportSeqID
DestCityMarketID
```

Dest
DestCityName
DestState
DestStateFips
DestStateName
DestWac
CRSDepTime
DepTime
DepDelay
DepDelayMinutes
DepDel15
DepartureDelayGroups
DepTimeBlk
TaxiOut
WheelsOff
WheelsOn
TaxiIn
CRSArrTime
ArrTime
ArrDelay
ArrDelayMinutes
ArrDel15
ArrivalDelayGroups
ArrTimeBlk
Cancelled
CancellationCode
Diverted
CRSElapsedTime
ActualElapsedTime
AirTime
Flights
Distance
DistanceGroup
CarrierDelay
WeatherDelay
NASDelay
SecurityDelay
LateAircraftDelay
FirstDepTime
TotalAddGTime
LongestAddGTime
DivAirportLandings
DivReachedDest
DivActualElapsedTime
DivArrDelay
DivDistance
Div1Airport
Div1AirportID

```
Div1AirportSeqID
Div1WheelsOn
Div1TotalGTime
Div1LongestGTime
Div1WheelsOff
Div1TailNum
Div2Airport
Div2AirportID
Div2AirportSeqID
Div2WheelsOn
Div2TotalGTime
Div2LongestGTime
Div2WheelsOff
Div2TailNum
Div3Airport
Div3AirportID
Div3AirportSeqID
Div3WheelsOn
Div3TotalGTime
Div3LongestGTime
Div3WheelsOff
Div3TailNum
Div4Airport
Div4AirportID
Div4AirportSeqID
Div4WheelsOn
Div4TotalGTime
Div4LongestGTime
Div4WheelsOff
Div4TailNum
Div5Airport
Div5AirportID
Div5AirportSeqID
Div5WheelsOn
Div5TotalGTime
Div5LongestGTime
Div5WheelsOff
Div5TailNum
Unnamed: 109
```

Question: Print all the columns in the dataset that contain the word 'Del'. This will help you see how many columns have delay data in them.

Hint: You can use a Python list comprehension to include values that pass certain `if` statement criteria.

For example: `[x for x in [1,2,3,4,5] if x > 2]`

Hint: You can use the `in` keyword (documentation) to check if the value is in a list or not.

For example: `5 in [1,2,3,4,5]`

[24]:
```
# Enter your code here
del_columns = [col for col in df_temp.columns if "Del" in col]

print(f"Columns with delay data in it : \n{del_columns}")
```

```
Columns with delay data in it :
['DepDelay', 'DepDelayMinutes', 'DepDel15', 'DepartureDelayGroups', 'ArrDelay',
'ArrDelayMinutes', 'ArrDel15', 'ArrivalDelayGroups', 'CarrierDelay',
'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircraftDelay', 'DivArrDelay']
```

Here are some more questions to help you find out more about your dataset.

Questions

1. How many rows and columns does the dataset have?
2. How many years are included in the dataset?
3. What is the date range for the dataset?
4. Which airlines are included in the dataset?
5. Which origin and destination airports are covered?

[25]:
```
# to answer above questions, complete the following code
print("The #rows and #columns are ", df_shape[0], " and ", df_shape[1])
print("The years in this dataset are: ", list(df_temp["Year"].unique()))
print("The months covered in this dataset are: ", list(df_temp["Month"].
 ↪unique()))
print(
    "The date range for data is :",
    min(df_temp["FlightDate"]),
    " to ",
    max(df_temp["FlightDate"]),
)
print(
    "The airlines covered in this dataset are: ",
    list(df_temp["Reporting_Airline"].unique()),
)
print("The Origin airports covered are: ", list(df_temp["Origin"].unique()))
print("The Destination airports covered are: ", list(df_temp["Dest"].unique()))
```

```
The #rows and #columns are  585749  and  110
The years in this dataset are:  [2018]
The months covered in this dataset are:  [9]
The date range for data is : 2018-09-01  to  2018-09-30
The airlines covered in this dataset are:  ['9E', 'B6', 'WN', 'YV', 'YX', 'EV',
'AA', 'AS', 'DL', 'HA', 'UA', 'F9', 'G4', 'MQ', 'NK', 'OH', 'OO']
The Origin airports covered are:  ['DFW', 'LGA', 'MSN', 'MSP', 'ATL', 'BDL',
'VLD', 'JFK', 'RDU', 'CHS', 'DTW', 'GRB', 'PVD', 'SHV', 'FNT', 'PIT', 'RIC',
'RST', 'RSW', 'CVG', 'LIT', 'ORD', 'JAX', 'TRI', 'BOS', 'CWA', 'DCA', 'CHO',
'AVP', 'IND', 'GRR', 'BTR', 'MEM', 'TUL', 'CLE', 'STL', 'BTV', 'OMA', 'MGM',
```

```
'TVC', 'SAV', 'GSP', 'EWR', 'OAJ', 'BNA', 'MCI', 'TLH', 'ROC', 'LEX', 'PWM',
'BUF', 'AGS', 'CLT', 'GSO', 'BWI', 'SAT', 'PHL', 'TYS', 'ACK', 'DSM', 'GNV',
'AVL', 'BGR', 'MHT', 'ILM', 'MOT', 'IAH', 'SBN', 'SYR', 'ORF', 'MKE', 'XNA',
'MSY', 'PBI', 'ABE', 'HPN', 'EVV', 'ALB', 'LNK', 'AUS', 'PHF', 'CHA', 'GTR',
'BMI', 'BQK', 'CID', 'CAK', 'ATW', 'ABY', 'CAE', 'SRQ', 'MLI', 'BHM', 'IAD',
'CSG', 'CMH', 'MCO', 'MBS', 'FLL', 'SDF', 'TPA', 'MVY', 'LAS', 'LGB', 'SFO',
'SAN', 'LAX', 'RNO', 'PDX', 'ANC', 'ABQ', 'SLC', 'DEN', 'PHX', 'OAK', 'SMF',
'SJU', 'SEA', 'HOU', 'STX', 'BUR', 'SWF', 'SJC', 'DAB', 'BQN', 'PSE', 'ORH',
'HYA', 'STT', 'ONT', 'HRL', 'ICT', 'ISP', 'LBB', 'MAF', 'MDW', 'OKC', 'PNS',
'SNA', 'TUS', 'AMA', 'BOI', 'CRP', 'DAL', 'ECP', 'ELP', 'GEG', 'LFT', 'MFE',
'MDT', 'JAN', 'COS', 'MOB', 'VPS', 'MTJ', 'DRO', 'GPT', 'BFL', 'MRY', 'SBA',
'PSP', 'FSD', 'BRO', 'RAP', 'COU', 'STS', 'PIA', 'FAT', 'SBP', 'FSM', 'HSV',
'BIS', 'DAY', 'BZN', 'MIA', 'EYW', 'MYR', 'HHH', 'GJT', 'FAR', 'SGF', 'HOB',
'CLL', 'LRD', 'AEX', 'ERI', 'MLU', 'LCH', 'ROA', 'LAW', 'MHK', 'GRK', 'SAF',
'GRI', 'JLN', 'ROW', 'FWA', 'CRW', 'LAN', 'OGG', 'HNL', 'KOA', 'EGE', 'LIH',
'MLB', 'JAC', 'FAI', 'RDM', 'ADQ', 'BET', 'BRW', 'SCC', 'KTN', 'YAK', 'CDV',
'JNU', 'SIT', 'PSG', 'WRG', 'OME', 'OTZ', 'ADK', 'FCA', 'FAY', 'PSC', 'BIL',
'MSO', 'ITO', 'PPG', 'MFR', 'EUG', 'GUM', 'SPN', 'DLH', 'TTN', 'BKG', 'SFB',
'PIE', 'PGD', 'AZA', 'SMX', 'RFD', 'SCK', 'OWB', 'HTS', 'BLV', 'IAG', 'USA',
'GFK', 'BLI', 'ELM', 'PBG', 'LCK', 'GTF', 'OGD', 'IDA', 'PVU', 'TOL', 'PSM',
'CKB', 'HGR', 'SPI', 'STC', 'ACT', 'TYR', 'ABI', 'AZO', 'CMI', 'BPT', 'GCK',
'MQT', 'ALO', 'TXK', 'SPS', 'SWO', 'DBQ', 'SUX', 'SJT', 'GGG', 'LSE', 'LBE',
'ACY', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'IMT', 'WYS', 'CPR', 'SCE',
'HLN', 'SUN', 'ISN', 'CMX', 'EAU', 'LWB', 'SHD', 'LBF', 'HYS', 'SLN', 'EAR',
'VEL', 'CNY', 'GCC', 'RKS', 'PUB', 'LBL', 'MKG', 'PAH', 'CGI', 'UIN', 'BFF',
'DVL', 'JMS', 'LAR', 'SGU', 'PRC', 'ASE', 'RDD', 'ACV', 'OTH', 'COD', 'LWS',
'ABR', 'APN', 'ESC', 'PLN', 'BJI', 'BRD', 'BTM', 'CDC', 'CIU', 'EKO', 'TWF',
'HIB', 'BGM', 'RHI', 'ITH', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
The Destination airports covered are:  ['CVG', 'PWM', 'RDU', 'MSP', 'MSN',
'SHV', 'CLT', 'PIT', 'RIC', 'IAH', 'ATL', 'JFK', 'DCA', 'DTW', 'LGA', 'TYS',
'PVD', 'FNT', 'LIT', 'BUF', 'ORD', 'TRI', 'IND', 'BGR', 'AVP', 'BWI', 'LEX',
'BDL', 'GRR', 'CWA', 'TUL', 'MEM', 'AGS', 'EWR', 'MGM', 'PHL', 'SYR', 'OMA',
'STL', 'TVC', 'ORF', 'CLE', 'ABY', 'BOS', 'OAJ', 'TLH', 'BTR', 'SAT', 'JAX',
'BNA', 'CHO', 'VLD', 'ROC', 'DFW', 'GNV', 'ACK', 'PBI', 'CHS', 'GRB', 'MOT',
'MKE', 'DSM', 'ILM', 'GSO', 'MCI', 'SBN', 'BTV', 'MVY', 'XNA', 'RST', 'EVV',
'HPN', 'RSW', 'MDT', 'ROA', 'GSP', 'MCO', 'CSG', 'SAV', 'PHF', 'ALB', 'CHA',
'ABE', 'BMI', 'MSY', 'IAD', 'GTR', 'CID', 'CAK', 'ATW', 'AUS', 'BQK', 'MLI',
'CAE', 'CMH', 'AVL', 'MBS', 'FLL', 'SDF', 'TPA', 'LNK', 'SRQ', 'MHT', 'BHM',
'LAS', 'SFO', 'SAN', 'RNO', 'LGB', 'ANC', 'PDX', 'SJU', 'ABQ', 'SLC', 'DEN',
'LAX', 'PHX', 'OAK', 'SMF', 'SEA', 'STX', 'BUR', 'DAB', 'SJC', 'SWF', 'HOU',
'BQN', 'PSE', 'ORH', 'HYA', 'STT', 'ONT', 'DAL', 'ECP', 'ELP', 'HRL', 'MAF',
'MDW', 'OKC', 'PNS', 'SNA', 'AMA', 'BOI', 'GEG', 'ICT', 'LBB', 'TUS', 'ISP',
'CRP', 'MFE', 'LFT', 'VPS', 'JAN', 'COS', 'MOB', 'DRO', 'GPT', 'BFL', 'COU',
'SBP', 'MTJ', 'SBA', 'PSP', 'FSD', 'FSM', 'BRO', 'PIA', 'STS', 'FAT', 'RAP',
'MRY', 'HSV', 'BIS', 'DAY', 'BZN', 'MIA', 'EYW', 'MYR', 'HHH', 'GJT', 'FAR',
'MLU', 'LRD', 'CLL', 'LCH', 'FWA', 'GRK', 'SGF', 'HOB', 'LAW', 'MHK', 'SAF',
'JLN', 'ROW', 'GRI', 'AEX', 'CRW', 'LAN', 'ERI', 'HNL', 'KOA', 'OGG', 'EGE',
'LIH', 'JAC', 'MLB', 'RDM', 'BET', 'ADQ', 'BRW', 'SCC', 'FAI', 'JNU', 'CDV',
```

```
'YAK', 'SIT', 'KTN', 'WRG', 'PSG', 'OME', 'OTZ', 'ADK', 'FCA', 'BIL', 'PSC',
'FAY', 'MSO', 'ITO', 'PPG', 'MFR', 'DLH', 'EUG', 'GUM', 'SPN', 'TTN', 'BKG',
'AZA', 'SFB', 'LCK', 'BLI', 'SCK', 'PIE', 'RFD', 'PVU', 'PBG', 'BLV', 'PGD',
'SPI', 'USA', 'TOL', 'IDA', 'ELM', 'HTS', 'HGR', 'SMX', 'OGD', 'GFK', 'STC',
'GTF', 'IAG', 'CKB', 'OWB', 'PSM', 'ABI', 'TYR', 'ALO', 'SUX', 'AZO', 'ACT',
'CMI', 'BPT', 'TXK', 'SWO', 'SPS', 'DBQ', 'SJT', 'GGG', 'LSE', 'MQT', 'GCK',
'LBE', 'ACY', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'WYS', 'SCE', 'IMT',
'HLN', 'ASE', 'SUN', 'ISN', 'EAR', 'SGU', 'VEL', 'SHD', 'LWB', 'MKG', 'SLN',
'HYS', 'BFF', 'PUB', 'LBL', 'CMX', 'EAU', 'PAH', 'UIN', 'RKS', 'CGI', 'CNY',
'JMS', 'DVL', 'LAR', 'GCC', 'LBF', 'PRC', 'RDD', 'ACV', 'OTH', 'COD', 'LWS',
'ABR', 'APN', 'PLN', 'BJI', 'CPR', 'BRD', 'BTM', 'CDC', 'CIU', 'ESC', 'EKO',
'ITH', 'HIB', 'BGM', 'TWF', 'RHI', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
```

Question: What is the count of all the origin and destination airports?

Hint: You can use the Pandas `values_count` function (documentation) to find out the values for each airport using the columns `Origin` and `Dest`.

[26]:
```python
counts = pd.DataFrame(
    {
        "Origin": df_temp["Origin"].value_counts(),
        "Destination": df_temp["Dest"].value_counts(),
    }
)


counts
```

[26]:
```
     Origin  Destination
ABE     303          303
ABI     169          169
ABQ    2077         2076
ABR      60           60
ABY      79           79
..      …            …
WRG      60           60
WYS      52           52
XNA    1004         1004
YAK      60           60
YUM      96           96

[346 rows x 2 columns]
```

Question: Print the top 15 origin and destination airports based on number of flights in the dataset.

Hint: You can use the Pandas `sort_values` function (documentation).

[27]:
```python
counts.sort_values(by=["Origin", "Destination"], ascending=False).head(15)
```

[27]:

	Origin	Destination
ATL	31525	31521
ORD	28257	28250
DFW	22802	22795
DEN	19807	19807
CLT	19655	19654
LAX	17875	17873
SFO	14332	14348
IAH	14210	14203
LGA	13850	13850
MSP	13349	13347
LAS	13318	13322
PHX	13126	13128
DTW	12725	12724
BOS	12223	12227
SEA	11872	11877

Question: Given all the information about a flight trip, can you predict if it would be delayed?

Answer:

Yes, with the required information provided regarding a particular flight, we can predict the delay of the flight.The past flight details would assist us with accurately predicting the flight delay.

Now, assume you are traveling from San Francisco to Los Angeles on a work trip. You want to have an ideas if your flight will be delayed, given a set of features, so that you can manage your reservations in Los Angeles better. How many features from this dataset would you know before your flight?

Columns such as `DepDelay`, `ArrDelay`, `CarrierDelay`, `WeatherDelay`, `NASDelay`, `SecurityDelay`, `LateAircraftDelay`, and `DivArrDelay` contain information about a delay. But this delay could have occured at the origin or destination. If there were a sudden weather delay 10 minutes before landing, this data would not be helpful in managing your Los Angeles reservations.

So to simplify the problem statement, consider the following columns to predict an arrival delay:

`Year`, `Quarter`, `Month`, `DayofMonth`, `DayOfWeek`, `FlightDate`, `Reporting_Airline`, `Origin`, `OriginState`, `Dest`, `DestState`, `CRSDepTime`, `DepDelayMinutes`, `DepartureDelayGroups`, `Cancelled`, `Diverted`, `Distance`, `DistanceGroup`, `ArrDelay`, `ArrDelayMinutes`, `ArrDel15`, `AirTime`

You will also filter the source and destination airports to be:

- Top airports: ATL, ORD, DFW, DEN, CLT, LAX, IAH, PHX, SFO
- Top 5 airlines: UA, OO, WN, AA, DL

This should help in reducing the size of data across the CSV files to be combined.

Combine all CSV files **Hint**:
First, create an empy dataframe that you will use to copy your individual dataframes from each file. Then, for each file in the `csv_files` list:

1. Read the CSV file into a dataframe

2. Filter the columns based on the `filter_cols` variable

   ```
   columns = ['col1', 'col2']
   df_filter = df[columns]
   ```

3. Keep only the subset_vals in each of the subset_cols. Use the `isin` Pandas function (documentation) to check if the `val` is in the dataframe column and then choose the rows that include it.

   ```
   df_eg[df_eg['col1'].isin('5')]
   ```

4. Concatenate the dataframe with the empty dataframe

[28]:
```python
def combine_csv(csv_files, filter_cols, subset_cols, subset_vals, file_name):
    """
    Combine csv files into one Data Frame
    csv_files: list of csv file paths
    filter_cols: list of columns to filter
    subset_cols: list of columns to subset rows
    subset_vals: list of list of values to subset rows
    """
    # Create an empty dataframe
    df = pd.DataFrame()

    # loop through the csv files
    for file in csv_files:

        # csv file path
        file_path = csv_base_path + file

        # reading the csv file
        print(f"Reading {file}")

        # read the csv file
        temp_df = pd.read_csv(file_path)

        # filter the columns
        temp_df = temp_df[filter_cols]

        # Subset rows based on specified columns and values
        for col, vals in zip(subset_cols, subset_vals):
            temp_df = temp_df[temp_df[col].isin(vals)]

        # append the dataframe
        df = pd.concat([df, temp_df], ignore_index=True)

    # save the dataframe
    df.to_csv(file_name, index=False)
```

```
    # return the dataframe
    return df
```

[29]:
```
# cols is the list of columns to predict Arrival Delay
cols = [
    "Year",
    "Quarter",
    "Month",
    "DayofMonth",
    "DayOfWeek",
    "FlightDate",
    "Reporting_Airline",
    "Origin",
    "OriginState",
    "Dest",
    "DestState",
    "CRSDepTime",
    "Cancelled",
    "Diverted",
    "Distance",
    "DistanceGroup",
    "ArrDelay",
    "ArrDelayMinutes",
    "ArrDel15",
    "AirTime",
]

subset_cols = ["Origin", "Dest", "Reporting_Airline"]

# subset_vals is a list collection of the top origin and destination airports␣
 ↪and top 5 airlines
subset_vals = [
    ["ATL", "ORD", "DFW", "DEN", "CLT", "LAX", "IAH", "PHX", "SFO"],
    ["ATL", "ORD", "DFW", "DEN", "CLT", "LAX", "IAH", "PHX", "SFO"],
    ["UA", "OO", "WN", "AA", "DL"],
]
```

Use the function above to merge all the different files into a single file that you can read easily.

Note: This will take 5-7 minutes to complete.

[30]:
```
start = time.time()

combined_csv_filename = f"{base_path}combined_files.csv"

# < write code to call the combined_csv function>
combined_csv = combine_csv(
    csv_files, cols, subset_cols, subset_vals, combined_csv_filename
```

```python
)


print(f"csv's merged in {round((time.time() - start)/60,2)} minutes")
```

```
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_1.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_10.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_11.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_12.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_2.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_3.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_4.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_5.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_6.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_7.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_8.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2014_9.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_1.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_10.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_11.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_12.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_2.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_3.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_4.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_5.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_6.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_7.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_8.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2015_9.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_1.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_10.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_11.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_12.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_2.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_3.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_4.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_5.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_6.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_7.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_8.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2016_9.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_1.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_10.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_11.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_12.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_2.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_3.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_4.csv
```

```
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_5.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_6.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_7.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_8.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2017_9.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_1.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_10.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_11.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_12.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_2.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_3.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_4.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_5.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_6.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_7.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_8.csv
Reading On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.csv
csv's merged in 2.95 minutes
```

Load dataset Load the combined dataset.

[31]:
```
data = pd.read_csv(
    combined_csv_filename
)  # Enter your code here to read the combined csv file.
```

[32]:
```
data.shape
```

[32]:
```
(1658130, 20)
```

Print the first 5 records.

[33]:
```
# Enter your code here

data.head(5)
```

[33]:
```
   Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate Reporting_Airline  \
0  2014        1      1          26          7  2014-01-26                DL
1  2014        1      1          26          7  2014-01-26                DL
2  2014        1      1          26          7  2014-01-26                DL
3  2014        1      1          26          7  2014-01-26                DL
4  2014        1      1          26          7  2014-01-26                DL

  Origin OriginState Dest DestState  CRSDepTime  Cancelled  Diverted  \
0    ATL          GA  IAH        TX        2145        0.0       0.0
1    DFW          TX  ATL        GA         945        0.0       0.0
2    ATL          GA  DEN        CO        1855        0.0       0.0
3    ATL          GA  PHX        AZ        1634        0.0       0.0
4    PHX          AZ  ATL        GA         700        0.0       0.0
```

```
   Distance  DistanceGroup  ArrDelay  ArrDelayMinutes  ArrDel15  AirTime
0     689.0              3     -20.0              0.0       0.0     99.0
1     731.0              3      -3.0              0.0       0.0     98.0
2    1199.0              5      -7.0              0.0       0.0    174.0
3    1587.0              7      -4.0              0.0       0.0    233.0
4    1587.0              7     -13.0              0.0       0.0    179.0
```

Here are some more questions to help you find out more about your dataset.

Questions

1. How many rows and columns does the dataset have?
2. How many years are included in the dataset?
3. What is the date range for the dataset?
4. Which airlines are included in the dataset?
5. Which origin and destination airports are covered?

[34]:
```
# to answer above questions, complete the following code
print("The #rows and #columns are ", data.shape[0], " and ", data.shape[1])
print("The years in this dataset are: ", list(data["Year"].unique()))
print("The months covered in this dataset are: ", sorted(list(data["Month"].
 ↪unique())))
print(
    "The date range for data is :",
    min(data["FlightDate"]),
    " to ",
    max(data["FlightDate"]),
)
print(
    "The airlines covered in this dataset are: ",
    list(data["Reporting_Airline"].unique()),
)
print("The Origin airports covered are: ", list(data["Origin"].unique()))
print("The Destination airports covered are: ", list(data["Dest"].unique()))
```

```
The #rows and #columns are  1658130  and  20
The years in this dataset are:  [2014, 2015, 2016, 2017, 2018]
The months covered in this dataset are:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
The date range for data is : 2014-01-01  to  2018-12-31
The airlines covered in this dataset are:  ['DL', 'OO', 'WN', 'UA', 'AA']
The Origin airports covered are:  ['ATL', 'DFW', 'PHX', 'DEN', 'IAH', 'CLT',
'SFO', 'LAX', 'ORD']
The Destination airports covered are:  ['IAH', 'ATL', 'DEN', 'PHX', 'CLT',
'LAX', 'DFW', 'SFO', 'ORD']
```

Let's define our **target column : is_delay** (1 - if arrival time delayed more than 15 minutes, 0 - otherwise). Use the `rename` method to rename the column from `ArrDel15` to `is_delay`.

Hint: You can use the Pandas `rename` function (documentation).

For example:

`df.rename(columns={'col1':'column1'}, inplace=True)`

[35]:
```
data.rename(columns={"ArrDel15": "is_delay"}, inplace=True)  # Enter your code␣
↪here
```

[36]:
```
data["is_delay"].value_counts()
```

[36]:
```
is_delay
0.0    1292258
1.0     343332
Name: count, dtype: int64
```

Look for nulls across columns. You can use the `isnull()` function (documentation).

Hint: `isnull()` detects whether the particular value is null or not and gives you a boolean (True or False) in its place. Use the `sum(axis=0)` function to sum up the number of columns.

[37]:
```
# Enter your code here
data.isnull().sum()
```

[37]:
```
Year                     0
Quarter                  0
Month                    0
DayofMonth               0
DayOfWeek                0
FlightDate               0
Reporting_Airline        0
Origin                   0
OriginState              0
Dest                     0
DestState                0
CRSDepTime               0
Cancelled                0
Diverted                 0
Distance                 0
DistanceGroup            0
ArrDelay             22540
ArrDelayMinutes      22540
is_delay             22540
AirTime              22540
dtype: int64
```

The arrival delay details and airtime are missing for 22540 out of 1658130 rows, which is 1.3%. You can either remove or impute these rows. The documentation does not mention anything about missing rows.

Hint: Use the ~ operator to choose the values that aren't null from the `isnull()` output.

For example:

```
null_eg = df_eg[~df_eg['column_name'].isnull()]
```

[38]:
```
### Remove null columns
data = data[~data["ArrDelay"].isnull()]
```

[39]:
```
data.head(5)
```

[39]:
```
   Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate Reporting_Airline  \
0  2014        1      1          26          7  2014-01-26                DL
1  2014        1      1          26          7  2014-01-26                DL
2  2014        1      1          26          7  2014-01-26                DL
3  2014        1      1          26          7  2014-01-26                DL
4  2014        1      1          26          7  2014-01-26                DL

  Origin OriginState Dest DestState  CRSDepTime  Cancelled  Diverted  \
0    ATL          GA  IAH        TX        2145        0.0       0.0
1    DFW          TX  ATL        GA         945        0.0       0.0
2    ATL          GA  DEN        CO        1855        0.0       0.0
3    ATL          GA  PHX        AZ        1634        0.0       0.0
4    PHX          AZ  ATL        GA         700        0.0       0.0

   Distance  DistanceGroup  ArrDelay  ArrDelayMinutes  is_delay  AirTime
0     689.0              3     -20.0              0.0       0.0     99.0
1     731.0              3      -3.0              0.0       0.0     98.0
2    1199.0              5      -7.0              0.0       0.0    174.0
3    1587.0              7      -4.0              0.0       0.0    233.0
4    1587.0              7     -13.0              0.0       0.0    179.0
```

Get the hour of the day in 24-hour time format from CRSDepTime.

[40]:
```
data["DepHourofDay"] = data["CRSDepTime"] // 100
data.head()
```

[40]:
```
   Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate Reporting_Airline  \
0  2014        1      1          26          7  2014-01-26                DL
1  2014        1      1          26          7  2014-01-26                DL
2  2014        1      1          26          7  2014-01-26                DL
3  2014        1      1          26          7  2014-01-26                DL
4  2014        1      1          26          7  2014-01-26                DL

  Origin OriginState Dest  … CRSDepTime  Cancelled  Diverted  Distance  \
0    ATL          GA  IAH  …       2145        0.0       0.0     689.0
1    DFW          TX  ATL  …        945        0.0       0.0     731.0
2    ATL          GA  DEN  …       1855        0.0       0.0    1199.0
3    ATL          GA  PHX  …       1634        0.0       0.0    1587.0
4    PHX          AZ  ATL  …        700        0.0       0.0    1587.0

   DistanceGroup  ArrDelay  ArrDelayMinutes  is_delay  AirTime  DepHourofDay
```

0	3	-20.0	0.0	0.0	99.0	21
1	3	-3.0	0.0	0.0	98.0	9
2	5	-7.0	0.0	0.0	174.0	18
3	7	-4.0	0.0	0.0	233.0	16
4	7	-13.0	0.0	0.0	179.0	7

```
[5 rows x 21 columns]
```

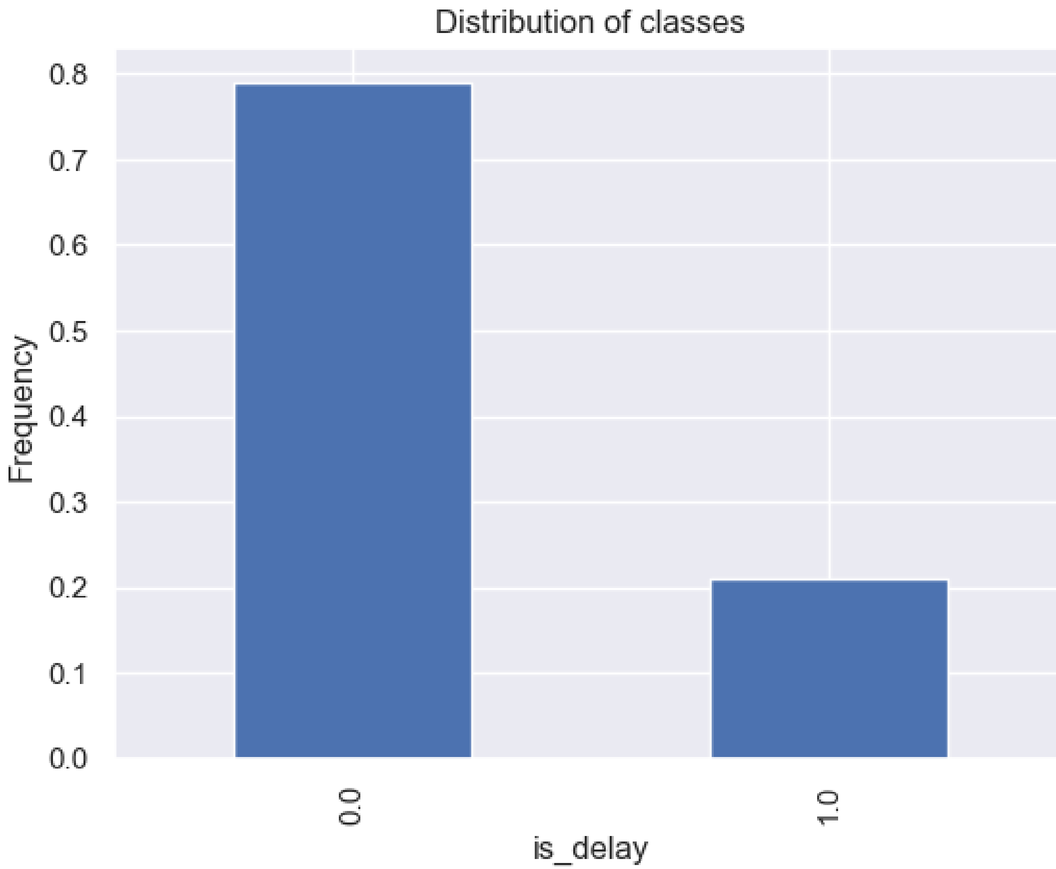
3.1 The ML problem statement

- Given a set of features, can you predict if a flight is going to be delayed more than 15 minutes?
- Because the target variable takes only 0/1 value, you could use a classification algorithm.

3.1.1 Data exploration

Check class delay vs. no delay **Hint**: Use a `groupby` plot (documentation) with a `bar` plot (documentation) to plot the frequency vs. distribution of the class.

[41]:
```
(data.groupby("is_delay").size() / len(data)).plot(kind="bar")  # Enter your␣
 ↪code here
plt.ylabel("Frequency")
plt.title("Distribution of classes")
plt.show()
```



Question: What can you deduce from the bar plot about the ratio of delay vs. no delay?

Answer:

From the bar plot, it can be deduced that nearly 80% of the flights are on-time and only 20% of the flights are delayed for the selected subset of filtered data, that we have chosen for the analysis and modelling.

This indicates a significant imbalance in the dataset, which could potentially lead to a bias to the majority class. This means that the model may show higher accuracy, however would perform poorly in detecting the delayed flights. This is a serious concern as our goal is to accurately identify delayed flights.

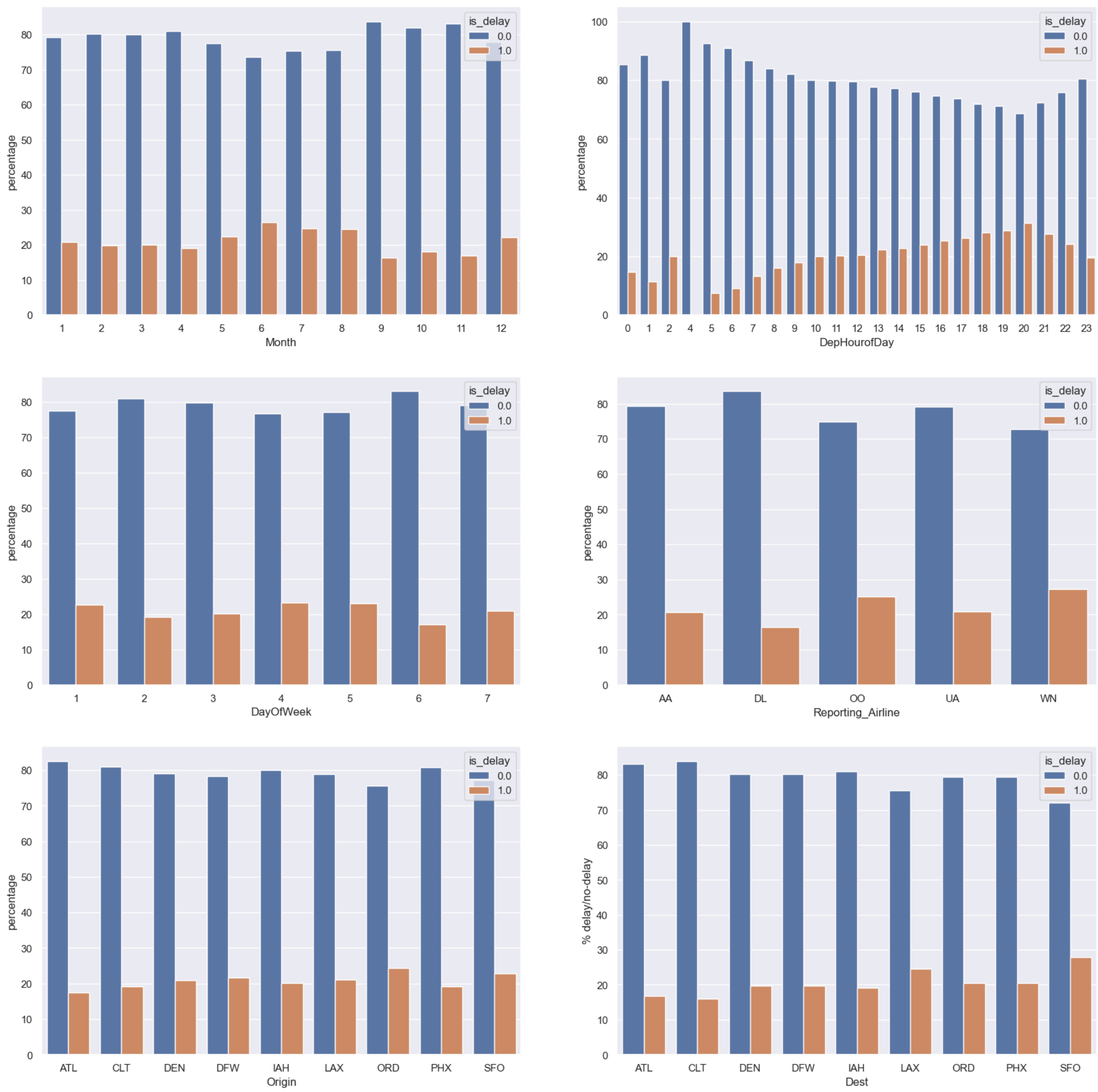
Use of Undersampling for on-time flights or using appropriate evaluation metrics like precision, recall, F1 score and ROC-AUC curve instead of accuracy need to be considered.

Questions:

- Which months have the most delays?
- What time of the day has the most delays?
- What day of the week has the most delays?
- Which airline has the most delays?
- Which origin and destination airports have the most delays?

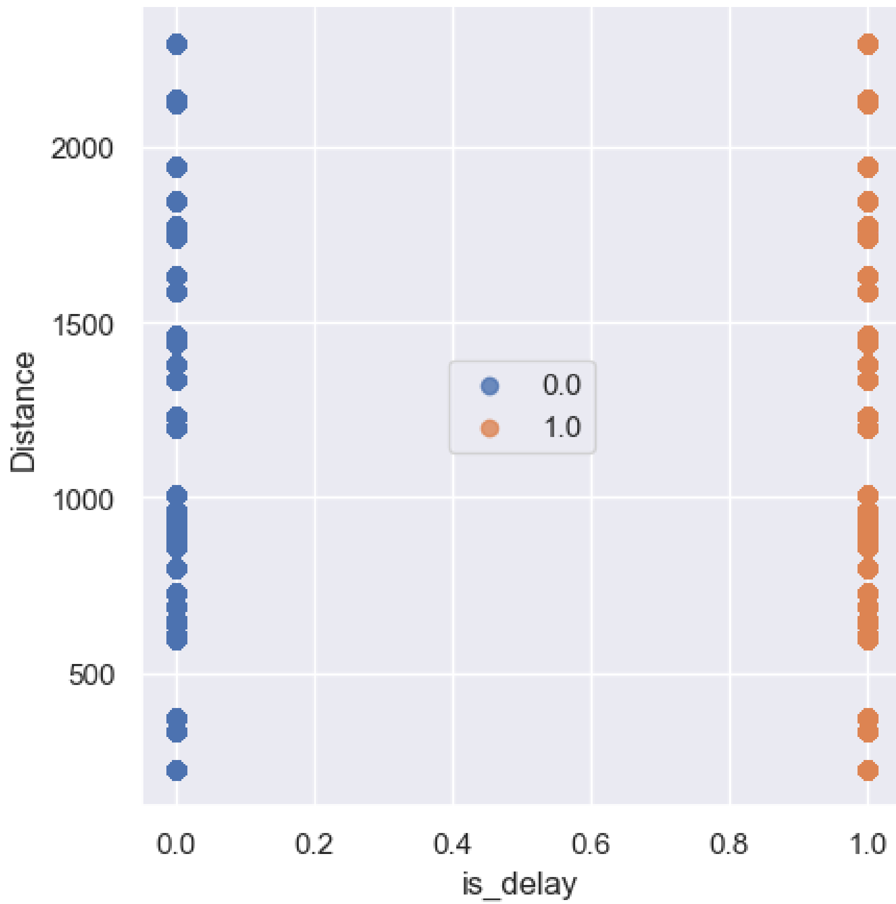
- Is flight distance a factor in the delays?

[42]:
```
viz_columns = [
    "Month",
    "DepHourofDay",
    "DayOfWeek",
    "Reporting_Airline",
    "Origin",
    "Dest",
]


fig, axes = plt.subplots(3, 2, figsize=(20, 20), squeeze=False)


# fig.autofmt_xdate(rotation=90)


for idx, column in enumerate(viz_columns):

    ax = axes[idx // 2, idx % 2]

    temp = (
        data.groupby(column)["is_delay"]
        .value_counts(normalize=True)
        .rename("percentage")
        .mul(100)
        .reset_index()
        .sort_values(column)
    )
    sns.barplot(x=column, y="percentage", hue="is_delay", data=temp, ax=ax)

    plt.ylabel("% delay/no-delay")


plt.show()
```



[43]:
```python
sns.lmplot(
    x="is_delay", y="Distance", data=data, fit_reg=False, hue="is_delay",␣
 ↪legend=False
)


plt.legend(loc="center")


plt.xlabel("is_delay")


plt.ylabel("Distance")
```

```
plt.show()
```



Answer:

1. The percentage of delays is higher in the month of `June`, `July` and `August`.
2. Departure hour of the day `20` has most delays
3. Day of the week `1` and `4` has most delays
4. Airline `WN` has most delays.
5. `ORD` among Origin Airports and `SFO` among Destination Airports has most number of delayed flights.
6. Flight distance doen't seems to be a significant factor contributing to the delays.

3.1.2 Features

Look at all the columns and what their specific types are.

[44]:
```
data.columns
```

[44]:
```
Index(['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'FlightDate',
       'Reporting_Airline', 'Origin', 'OriginState', 'Dest', 'DestState',
       'CRSDepTime', 'Cancelled', 'Diverted', 'Distance', 'DistanceGroup',
       'ArrDelay', 'ArrDelayMinutes', 'is_delay', 'AirTime', 'DepHourofDay'],
      dtype='object')
```

[45]:
```
data.dtypes
```

[45]:
```
Year                    int64
Quarter                 int64
Month                   int64
DayofMonth              int64
DayOfWeek               int64
FlightDate             object
Reporting_Airline      object
Origin                 object
OriginState            object
Dest                   object
DestState              object
CRSDepTime              int64
Cancelled             float64
Diverted              float64
Distance              float64
DistanceGroup           int64
ArrDelay              float64
ArrDelayMinutes       float64
is_delay              float64
AirTime               float64
DepHourofDay            int64
dtype: object
```

Filtering the required columns:

- Date is redundant, because you have Year, Quarter, Month, DayofMonth, and DayOfWeek to describe the date.
- Use Origin and Dest codes instead of OriginState and DestState.
- Because you are just classifying whether the flight is delayed or not, you don't need TotalDelayMinutes, DepDelayMinutes, and ArrDelayMinutes.

Treat DepHourofDay as a categorical variable because it doesn't have any quantitative relation with the target.

- If you had to do a one-hot encoding of it, it would result in 23 more columns.
- Other alternatives to handling categorical variables include hash encoding, regularized mean encoding, and bucketizing the values, among others.
- Just split into buckets here.

Hint: To change a column type to category, use the `astype` function (documentation).

[46]:
```python
data_orig = data.copy()
data = data[
    [
        "is_delay",
        "Quarter",
        "Month",
        "DayofMonth",
        "DayOfWeek",
        "Reporting_Airline",
        "Origin",
        "Dest",
        "Distance",
        "DepHourofDay",
    ]
]
categorical_columns = [
    "Quarter",
    "Month",
    "DayofMonth",
    "DayOfWeek",
    "Reporting_Airline",
    "Origin",
    "Dest",
    "DepHourofDay",
]
for c in categorical_columns:
    data[c] = data[c].astype("category")  # Enter your code here

# Bucketize DepHourofDay into different time intervals
bins = [0, 6, 12, 18, 24]
labels = ["Night", "Morning", "Afternoon", "Evening"]
data["DepHourofDay"] = pd.cut(
    data["DepHourofDay"], bins=bins, labels=labels, right=False
)
```

[53]:
```python
data.head()
```

[53]:
```
   is_delay Quarter Month DayofMonth DayOfWeek Reporting_Airline Origin Dest  \
0       0.0       1     1         26         7                DL    ATL  IAH
1       0.0       1     1         26         7                DL    DFW  ATL
2       0.0       1     1         26         7                DL    ATL  DEN
3       0.0       1     1         26         7                DL    ATL  PHX
4       0.0       1     1         26         7                DL    PHX  ATL

   Distance DepHourofDay
0     689.0      Evening
1     731.0      Morning
```

```
2     1199.0      Evening
3     1587.0    Afternoon
4     1587.0      Morning
```

To use one-hot encoding, use the Pandas `get_dummies` function for the categorical columns that you selected above. Then, you can concatenate those generated features to your original dataset using the Pandas `concat` function. For encoding categorical variables, you can also use *dummy encoding* by using a keyword `drop_first=True`. For more information on dummy encoding, see https://en.wikiversity.org/wiki/Dummy_variable_(statistics).

For example:

`pd.get_dummies(df[['column1','columns2']], drop_first=True)`

[47]:
```python
# Perform one-hot encoding on the categorical columns
data_dummies = pd.get_dummies(
    data[categorical_columns], drop_first=True
)  # Enter your code here

# Concatenate the original data with the one-hot encoded columns
data = pd.concat([data, data_dummies], axis=1)

# Drop the original categorical columns
data.drop(categorical_columns, axis=1, inplace=True)
```

Check the length of the dataset and the new columnms.

[48]:
```python
# Enter your code here
print(f"The dataset has {data.shape[0]} rows and {data.shape[1]} columns")
```

```
The dataset has 1635590 rows and 75 columns
```

[49]:
```python
# Enter your code here
data.columns
```

[49]:
```
Index(['is_delay', 'Distance', 'Quarter_2', 'Quarter_3', 'Quarter_4',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12',
       'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'DayofMonth_5',
       'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'DayofMonth_9',
       'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'DayofMonth_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMonth_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMonth_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMonth_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMonth_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Airline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Origin_SFO',
```

```
'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX', 'Dest_ORD',
'Dest_PHX', 'Dest_SFO', 'DepHourofDay_Morning',
'DepHourofDay_Afternoon', 'DepHourofDay_Evening'],
dtype='object')
```

Sample Answer:

```
Index(['Distance', 'is_delay', 'Quarter_2', 'Quarter_3', 'Quarter_4',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6', 'Month_7',
       'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12',
       'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'DayofMonth_5',
       'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'DayofMonth_9',
       'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'DayofMonth_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMonth_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMonth_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMonth_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMonth_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Airline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Origin_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX', 'Dest_ORD',
       'Dest_PHX', 'Dest_SFO'],
      dtype='object')
```

Now you are ready to do model training. Before splitting the data, rename the column `is_delay` to `target`.

Hint: You can use the Pandas `rename` function (documentation).

[51]:
```python
data.rename(columns={"is_delay": "target"}, inplace=True)  # Enter your code here

data.head()
```

[51]:
```
   target  Distance  Quarter_2  Quarter_3  Quarter_4  Month_2  Month_3  \
0     0.0     689.0      False      False      False    False    False
1     0.0     731.0      False      False      False    False    False
2     0.0    1199.0      False      False      False    False    False
3     0.0    1587.0      False      False      False    False    False
4     0.0    1587.0      False      False      False    False    False

   Month_4  Month_5  Month_6  …  Dest_DEN  Dest_DFW  Dest_IAH  Dest_LAX  \
0    False    False    False  …     False     False      True     False
1    False    False    False  …     False     False     False     False
2    False    False    False  …      True     False     False     False
3    False    False    False  …     False     False     False     False
4    False    False    False  …     False     False     False     False
```

```
   Dest_ORD  Dest_PHX  Dest_SFO  DepHourofDay_Morning  DepHourofDay_Afternoon  \
0     False     False     False                 False                   False
1     False     False     False                  True                   False
2     False     False     False                 False                   False
3     False      True     False                 False                    True
4     False     False     False                  True                   False

   DepHourofDay_Evening
0                  True
1                 False
2                  True
3                 False
4                 False

[5 rows x 75 columns]
```

[52]:
```
data.shape
```

[52]:
```
(1635590, 75)
```

[53]:
```
# write code to Save the combined csv file (combined_csv_v1.csv) to your local␣
↪computer
# note this combined file will be used in part B
data.to_csv(f"{base_path}combined_csv_v1.csv", index=False)
```

4 Step 3: Model training and evaluation

1. Split the data into `train_data`, and `test_data` using `sklearn.model_selection.train_test_split`.
2. Build a logistic regression model for the data, where training data is 80%, and test data is 20%.

Use the following cells to complete these steps. Insert and delete cells where needed.

4.0.1 Train test split

[54]:
```
# check for null values across columns
data.isnull().sum().sum()
```

[54]:
```
0
```

[55]:
```
# write Code here to split data into train, validate and test
from sklearn.model_selection import train_test_split

X = data.drop("target", axis=1)
y = data["target"]
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

4.0.2 Baseline classification model

[56]:
```
# <write code here>

# logistc regression
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

# fit the model
logreg.fit(X_train, y_train)

# predict the target on train and test data
y_train_pred = logreg.predict(X_train)
y_test_pred = logreg.predict(X_test)

# calculate the accuracy
train_accuracy = np.mean(y_train == y_train_pred)

test_accuracy = np.mean(y_test == y_test_pred)

print(f"Train accuracy: {train_accuracy}")
print(f"Test accuracy: {test_accuracy}")
```

```
Train accuracy: 0.7901751050079787
Test accuracy: 0.7900298974681919
```

4.1 Model evaluation

In this section, you'll evaluate your trained model on test data and report on the following metrics:

- Confusion Matrix plot
- Plot the ROC
- Report statistics such as Accuracy, Percision, Recall, Sensitivity and Specificity

To view a plot of the confusion matrix, and various scoring metrics, create a couple of functions:

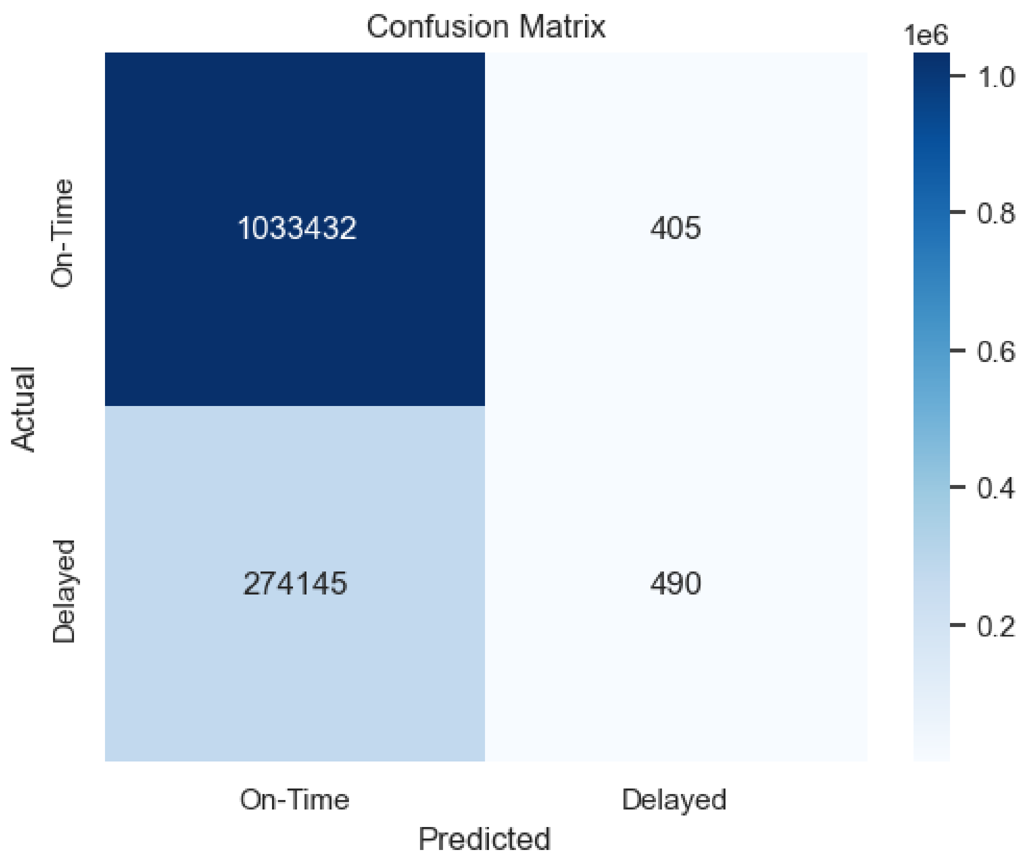
[57]:
```
from sklearn.metrics import confusion_matrix


def plot_confusion_matrix(test_labels, target_predicted):
    # complete the code here
    cm = confusion_matrix(test_labels, target_predicted)
    # Create a heatmap
```

```
    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        xticklabels=["On-Time", "Delayed"],
        yticklabels=["On-Time", "Delayed"],
    )
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")
    plt.show()
```

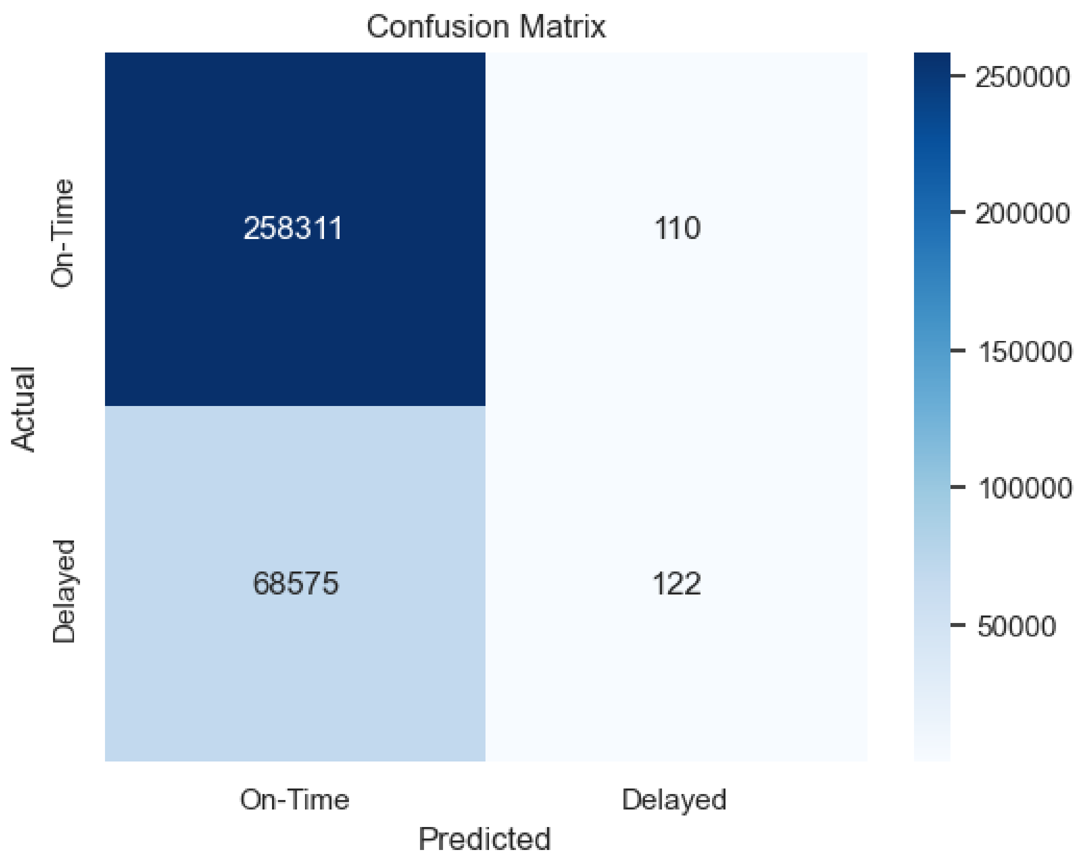
[58]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


# Function to plot ROC curve
def plot_roc_curve(test_labels, target_predicted_prob):
    fpr, tpr, _ = roc_curve(test_labels, target_predicted_prob)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color="blue", lw=2, label=f"ROC curve (area = {roc_auc:.
 ↪2f})")
    plt.plot([0, 1], [0, 1], color="gray", lw=2, linestyle="--")
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("Receiver Operating Characteristic (ROC) Curve")
    plt.legend(loc="lower right")
    plt.show()
```

To plot the confusion matrix, call the `plot_confusion_matrix` function on the `test_labels` and `target_predicted` data from your batch job:

[59]:
```
# confusion matrix on train data
plot_confusion_matrix(y_train, y_train_pred)
```

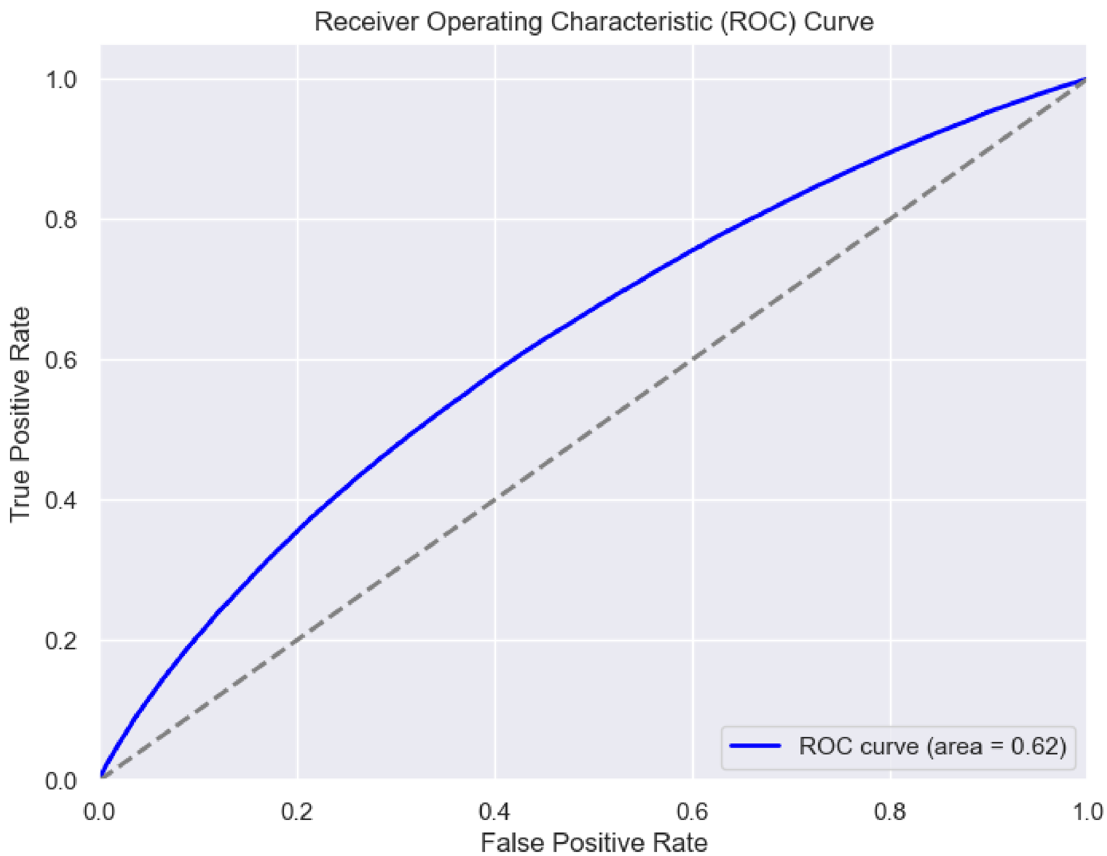


[60]:
```
# confusion matrix on test data
plot_confusion_matrix(y_test, y_test_pred)
```



To print statistics and plot an ROC curve, call the `plot_roc` function on the `test_labels` and `target_predicted` data from your batch job:

[61]:
```
# predict the probability of target on test data
y_test_pred_prob = logreg.predict_proba(X_test)[:, 1]

# plot ROC curve
plot_roc_curve(y_test, y_test_pred_prob)
```



[62]:

```
# classification report
from sklearn.metrics import classification_report


# classification report for train data
print("Classification Report on Train Data")
print(classification_report(y_train, y_train_pred))

# classification report for test data
print("Classification Report on Test Data")
print(classification_report(y_test, y_test_pred))
```

```
Classification Report on Train Data
              precision    recall  f1-score   support

         0.0       0.79      1.00      0.88   1033837
         1.0       0.55      0.00      0.00    274635

    accuracy                           0.79   1308472
   macro avg       0.67      0.50      0.44   1308472
```

```
weighted avg       0.74      0.79      0.70   1308472

Classification Report on Test Data
              precision    recall  f1-score   support

         0.0       0.79      1.00      0.88    258421
         1.0       0.53      0.00      0.00     68697

    accuracy                           0.79    327118
   macro avg       0.66      0.50      0.44    327118
weighted avg       0.73      0.79      0.70    327118
```

4.1.1 Key questions to consider:

1. How does your model's performance on the test set compare to the training set? What can you deduce from this comparison?
2. Are there obvious differences between the outcomes of metrics like accuracy, precision, and recall? If so, why might you be seeing those differences?
3. Is the outcome for the metric(s) you consider most important sufficient for what you need from a business standpoint? If not, what are some things you might change in your next iteration (in the feature engineering section, which is coming up next)?

Use the cells below to answer these and other questions. Insert and delete cells where needed.

Question: What can you summarize from the confusion matrix?

Answer:

1. There isn't any significant diffrence in model performance across train and test dataset.
2. This classification report reveals that the model has a high accuracy (79%) but struggles with identifying the minority class `is_delay`(1.0), achieving a recall of 0.00 for this class. This discrepancy is due to class imbalance, where the majority class `on-time`(0.0) dominates, leading the model to favor it heavily. As a result, the model performs well on the majority class `on-time` but fails to detect the minority class accurately.
3. Given that accurately predicting delays (1.0) is essential from a business perspective, the current model's low recall for this class is inadequate. Missing delays could lead to operational inefficiencies and poor customer experience. To improve, the next iteration should focus on feature engineering techniques like balancing the dataset with oversampling, adding relevant interaction features, and transforming features to better capture delay patterns. These adjustments can help the model improve its recall for delayed cases, aligning its predictions more closely with business goals.

5 Step 4: Deployment

1. In this step you are required to push your source code and requirements file to a GitLab repository without the data files. Please use the Git commands to complete this task

2. Create a “readme.md” markdown file that describes the code of this repository and how to run it and what the user would expect if got the code running.

In the cell below provide the link of the pushed repository on your GitLab account.

5.0.1 Provide a link for your Gitlab repository here

Gitlab

6 Iteration II

7 Step 5: Feature engineering

You’ve now gone through one iteration of training and evaluating your model. Given that the outcome you reached for your model the first time probably wasn’t sufficient for solving your business problem, what are some things you could change about your data to possibly improve model performance?

7.0.1 Key questions to consider:

1. How might the balance of your two main classes (delay and no delay) impact model performance?
2. Do you have any features that are correlated?
3. Are there feature reduction techniques you could perform at this stage that might have a positive impact on model performance?
4. Can you think of adding some more data/datasets?
5. After performing some feature engineering, how does your model performance compare to the first iteration?

Use the cells below to perform specific feature engineering techniques (per the questions above) that you think could improve your model performance. Insert and delete cells where needed.

Before you start, think about why the precision and recall are around 80% while the accuracy is 99%.

Add more features

1. Holidays
2. Weather

Because the list of holidays from 2014 to 2018 is known, you can create an indicator variable **is_holiday** to mark these. The hypothesis is that airplane delays could be higher during holidays compared to the rest of the days. Add a boolean variable `is_holiday` that includes the holidays for the years 2014-2018.

[63]:
```
# Source: http://www.calendarpedia.com/holidays/federal-holidays-2014.html

holidays_14 = [
    "2014-01-01",
    "2014-01-20",
```

```
    "2014-02-17",
    "2014-05-26",
    "2014-07-04",
    "2014-09-01",
    "2014-10-13",
    "2014-11-11",
    "2014-11-27",
    "2014-12-25",
]
holidays_15 = [
    "2015-01-01",
    "2015-01-19",
    "2015-02-16",
    "2015-05-25",
    "2015-06-03",
    "2015-07-04",
    "2015-09-07",
    "2015-10-12",
    "2015-11-11",
    "2015-11-26",
    "2015-12-25",
]
holidays_16 = [
    "2016-01-01",
    "2016-01-18",
    "2016-02-15",
    "2016-05-30",
    "2016-07-04",
    "2016-09-05",
    "2016-10-10",
    "2016-11-11",
    "2016-11-24",
    "2016-12-25",
    "2016-12-26",
]
holidays_17 = [
    "2017-01-02",
    "2017-01-16",
    "2017-02-20",
    "2017-05-29",
    "2017-07-04",
    "2017-09-04",
    "2017-10-09",
    "2017-11-10",
    "2017-11-23",
    "2017-12-25",
]
```

```
holidays_18 = [
    "2018-01-01",
    "2018-01-15",
    "2018-02-19",
    "2018-05-28",
    "2018-07-04",
    "2018-09-03",
    "2018-10-08",
    "2018-11-12",
    "2018-11-22",
    "2018-12-25",
]
holidays = holidays_14 + holidays_15 + holidays_16 + holidays_17 + holidays_18

### Add indicator variable for holidays
data_orig["is_holiday"] = data_orig["FlightDate"].isin(holidays)
```

Weather data was fetched from https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summaries&stations=USW00023174,USW00012960,USW00003017,USW00094846,USW00013874,USW00023234,U
01-01&endDate=2018-12-31.

This dataset has information on wind speed, precipitation, snow, and temperature for cities by their airport codes.

Question: Could bad weather due to rains, heavy winds, or snow lead to airplane delay? Let's check!

[15]:
```
# download data from the link above and place it into the data folder
```

Import weather data prepared for the airport codes in our dataset. Use the stations and airports below for the analysis, and create a new column called `airport` that maps the weather station to the airport name.

[64]:
```
weather = pd.read_csv(
    "./daily-summaries.csv"
)  # Enter your code here to read 'daily-summaries.csv' file
weather.head()


station = [
    "USW00023174",
    "USW00012960",
    "USW00003017",
    "USW00094846",
    "USW00013874",
    "USW00023234",
    "USW00003927",
    "USW00023183",
    "USW00013881",
```

```
]


airports = ["LAX", "IAH", "DEN", "ORD", "ATL", "SFO", "DFW", "PHX", "CLT"]


# ### Map weather stations to airport code


station_map = dict(zip(station, airports))


weather["airport"] = weather["STATION"].map(station_map)
```

[65]:
```
# head of the weather data
weather.head()
```

[65]:
```
       STATION        DATE  AWND  PRCP  SNOW  SNWD   TAVG   TMAX   TMIN  \
0  USW00023174  2014-01-01    16     0   NaN   NaN  131.0  178.0   78.0
1  USW00023174  2014-01-02    22     0   NaN   NaN  159.0  256.0  100.0
2  USW00023174  2014-01-03    17     0   NaN   NaN  140.0  178.0   83.0
3  USW00023174  2014-01-04    18     0   NaN   NaN  136.0  183.0  100.0
4  USW00023174  2014-01-05    18     0   NaN   NaN  151.0  244.0   83.0

  airport
0     LAX
1     LAX
2     LAX
3     LAX
4     LAX
```

Create another column called `MONTH` from the `DATE` column.

[66]:
```
weather["MONTH"] = weather["DATE"].apply(
    lambda x: x.split("-")[1]
)  # Enter your code here


weather.head()
```

[66]:
```
       STATION        DATE  AWND  PRCP  SNOW  SNWD   TAVG   TMAX   TMIN  \
0  USW00023174  2014-01-01    16     0   NaN   NaN  131.0  178.0   78.0
1  USW00023174  2014-01-02    22     0   NaN   NaN  159.0  256.0  100.0
2  USW00023174  2014-01-03    17     0   NaN   NaN  140.0  178.0   83.0
3  USW00023174  2014-01-04    18     0   NaN   NaN  136.0  183.0  100.0
4  USW00023174  2014-01-05    18     0   NaN   NaN  151.0  244.0   83.0

  airport MONTH
```

```
0      LAX     01
1      LAX     01
2      LAX     01
3      LAX     01
4      LAX     01
```

7.0.2 Sample output

	STATION	DATE	AWND	PRCP	SNOW	SNWD	TAVG	TMAX	TMIN	airport	MONTH
0	USW00023174	2014-01-01	16	0	NaN	NaN	131.0	178.0	78.0	LAX	01
1	USW00023174	2014-01-02	22	0	NaN	NaN	159.0	256.0	100.0	LAX	01
2	USW00023174	2014-01-03	17	0	NaN	NaN	140.0	178.0	83.0	LAX	01
3	USW00023174	2014-01-04	18	0	NaN	NaN	136.0	183.0	100.0	LAX	01
4	USW00023174	2014-01-05	18	0	NaN	NaN	151.0	244.0	83.0	LAX	01

Analyze and handle the `SNOW` and `SNWD` columns for missing values using `fillna()`. Use the `isna()` function to check the missing values for all the columns.

[67]:
```
weather.SNOW.fillna(0, inplace=True)  # Enter your code here
weather.SNWD.fillna(0, inplace=True)  # Enter your code here
weather.isna().sum()
```

[67]:
```
STATION     0
DATE        0
AWND        0
PRCP        0
SNOW        0
SNWD        0
TAVG       62
TMAX       20
TMIN       20
airport     0
MONTH       0
dtype: int64
```

Question: Print the index of the rows that have missing values for TAVG, TMAX, TMIN.

Hint: Use the `isna()` function to find the rows that are missing, and then use the list on the idx variable to get the index.

[68]:
```
idx = np.array([i for i in range(len(weather))])
TAVG_idx = idx[weather["TAVG"].isna()]
TMAX_idx = idx[weather["TMAX"].isna()]
TMIN_idx = idx[weather["TMIN"].isna()]

print(TAVG_idx)
```

```
[ 3956 3957 3958 3959 3960 3961 3962 3963 3964 3965 3966 3967
  3968 3969 3970 3971 3972 3973 3974 3975 3976 3977 3978 3979
  3980 3981 3982 3983 3984 3985 4017 4018 4019 4020 4021 4022
```

```
 4023  4024  4025  4026  4027  4028  4029  4030  4031  4032  4033  4034
 4035  4036  4037  4038  4039  4040  4041  4042  4043  4044  4045  4046
 4047 13420]
```

7.0.3 Sample output

```
array([ 3956,  3957,  3958,  3959,  3960,  3961,  3962,  3963,  3964,
        3965,  3966,  3967,  3968,  3969,  3970,  3971,  3972,  3973,
        3974,  3975,  3976,  3977,  3978,  3979,  3980,  3981,  3982,
        3983,  3984,  3985,  4017,  4018,  4019,  4020,  4021,  4022,
        4023,  4024,  4025,  4026,  4027,  4028,  4029,  4030,  4031,
        4032,  4033,  4034,  4035,  4036,  4037,  4038,  4039,  4040,
        4041,  4042,  4043,  4044,  4045,  4046,  4047, 13420])
```

You can replace the missing TAVG, TMAX, and TMIN with the average value for a particular station/airport. Because the consecutive rows of TAVG_idx are missing, replacing with a previous value would not be possible. Instead, replace it with the mean. Use the `groupby` function to aggregate the variables with a mean value.

[69]:
```
weather_impute = (
    weather.groupby(["STATION", "MONTH"])
    .agg({"TAVG": "mean", "TMAX": "mean", "TMIN": "mean"})
    .reset_index()
)  # Enter your code here


weather_impute.head(2)
```

[69]:
```
       STATION MONTH       TAVG       TMAX       TMIN
0  USW00003017    01  -2.741935  74.000000 -69.858065
1  USW00003017    02  11.219858  88.553191 -65.035461
```

Merge the mean data with the weather data.

[70]:
```
### get the yesterday's data
weather = pd.merge(
    weather,
    weather_impute,
    how="left",
    left_on=["MONTH", "STATION"],
    right_on=["MONTH", "STATION"],
).rename(
    columns={
        "TAVG_y": "TAVG_AVG",
        "TMAX_y": "TMAX_AVG",
        "TMIN_y": "TMIN_AVG",
        "TAVG_x": "TAVG",
        "TMAX_x": "TMAX",
        "TMIN_x": "TMIN",
```

```
    }
)
```

Check for missing values again.

[71]:
```
weather.TAVG[TAVG_idx] = weather.TAVG_AVG[TAVG_idx]
weather.TMAX[TMAX_idx] = weather.TMAX_AVG[TMAX_idx]
weather.TMIN[TMIN_idx] = weather.TMIN_AVG[TMIN_idx]
weather.isna().sum()
```

[71]:
```
STATION     0
DATE        0
AWND        0
PRCP        0
SNOW        0
SNWD        0
TAVG        0
TMAX        0
TMIN        0
airport     0
MONTH       0
TAVG_AVG    0
TMAX_AVG    0
TMIN_AVG    0
dtype: int64
```

Drop `STATION,MONTH,TAVG_AVG,TMAX_AVG,TMIN_AVG,TMAX,TMIN,SNWD` from the dataset

[72]:
```
weather.drop(
    columns=[
        "STATION",
        "MONTH",
        "TAVG_AVG",
        "TMAX_AVG",
        "TMIN_AVG",
        "TMAX",
        "TMIN",
        "SNWD",
    ],
    inplace=True,
)
```

Add the origin and destination weather conditions to the dataset.

[73]:
```
### Add origin weather conditions
data_orig = (
    pd.merge(
        data_orig,
        weather,
```

```
        how="left",
        left_on=["FlightDate", "Origin"],
        right_on=["DATE", "airport"],
    )
    .rename(
        columns={"AWND": "AWND_O", "PRCP": "PRCP_O", "TAVG": "TAVG_O", "SNOW":␣
↪"SNOW_O"}
    )
    .drop(columns=["DATE", "airport"])
)

### Add destination weather conditions
data_orig = (
    pd.merge(
        data_orig,
        weather,
        how="left",
        left_on=["FlightDate", "Dest"],
        right_on=["DATE", "airport"],
    )
    .rename(
        columns={"AWND": "AWND_D", "PRCP": "PRCP_D", "TAVG": "TAVG_D", "SNOW":␣
↪"SNOW_D"}
    )
    .drop(columns=["DATE", "airport"])
)
```

Note: It is always a good practice to check nulls/NAs after joins.

[74]:
```
sum(data.isna().any())
```

[74]: 0

[75]:
```
data_orig.columns
```

[75]:
```
Index(['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'FlightDate',
       'Reporting_Airline', 'Origin', 'OriginState', 'Dest', 'DestState',
       'CRSDepTime', 'Cancelled', 'Diverted', 'Distance', 'DistanceGroup',
       'ArrDelay', 'ArrDelayMinutes', 'is_delay', 'AirTime', 'DepHourofDay',
       'is_holiday', 'AWND_O', 'PRCP_O', 'SNOW_O', 'TAVG_O', 'AWND_D',
       'PRCP_D', 'SNOW_D', 'TAVG_D'],
      dtype='object')
```

Convert the categorical data into numerical data using one-hot encoding.

[76]:
```
data = data_orig.copy()
data = data[
    [
```

```
        "is_delay",
        "Year",
        "Quarter",
        "Month",
        "DayofMonth",
        "DayOfWeek",
        "Reporting_Airline",
        "Origin",
        "Dest",
        "Distance",
        "DepHourofDay",
        "is_holiday",
        "AWND_O",
        "PRCP_O",
        "TAVG_O",
        "AWND_D",
        "PRCP_D",
        "TAVG_D",
        "SNOW_O",
        "SNOW_D",
    ]
]


categorical_columns = [
    "Year",
    "Quarter",
    "Month",
    "DayofMonth",
    "DayOfWeek",
    "Reporting_Airline",
    "Origin",
    "Dest",
    "is_holiday",
]
for c in categorical_columns:
    data[c] = data[c].astype("category")
```

[81]:
```
# data for visualaisation in tableau

data.to_csv(f"{base_path}data_tableau.csv", index=False)
```

[77]:
```
data_dummies = pd.get_dummies(data[categorical_columns], drop_first=True)

data = pd.concat([data, data_dummies], axis=1)

data.drop(categorical_columns, axis=1, inplace=True)
```

7.0.4 Sample code

```
data_dummies = pd.get_dummies(data[['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'Rep
data = pd.concat([data, data_dummies], axis = 1)
categorical_columns.remove('is_delay')
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the new columns.

[78]:
```
data.columns
```

[78]:
```
Index(['is_delay', 'Distance', 'DepHourofDay', 'AWND_O', 'PRCP_O', 'TAVG_O',
        'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D', 'Year_2015',
        'Year_2016', 'Year_2017', 'Year_2018', 'Quarter_2', 'Quarter_3',
        'Quarter_4', 'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6',
        'Month_7', 'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12',
        'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'DayofMonth_5',
        'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'DayofMonth_9',
        'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'DayofMonth_13',
        'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMonth_17',
        'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMonth_21',
        'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMonth_25',
        'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMonth_29',
        'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_3',
        'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
        'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Airline_UA',
        'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_DFW',
        'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Origin_SFO',
        'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX', 'Dest_ORD',
        'Dest_PHX', 'Dest_SFO', 'is_holiday_True'],
       dtype='object')
```

7.0.5 Sample output

```
Index(['Distance', 'DepHourofDay', 'is_delay', 'AWND_O', 'PRCP_O', 'TAVG_O',
        'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D', 'Year_2015',
        'Year_2016', 'Year_2017', 'Year_2018', 'Quarter_2', 'Quarter_3',
        'Quarter_4', 'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6',
        'Month_7', 'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_12',
        'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'DayofMonth_5',
        'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'DayofMonth_9',
        'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'DayofMonth_13',
        'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMonth_17',
        'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMonth_21',
        'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMonth_25',
        'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMonth_29',
        'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_3',
        'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
        'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Airline_UA',
```

```
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Origin_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX', 'Dest_ORD',
       'Dest_PHX', 'Dest_SFO', 'is_holiday_1'],
      dtype='object')
```

Rename the `is_delay` column to `target` again. Use the same code as before.

[79]:
```python
data.rename(columns={"is_delay": "target"}, inplace=True)
```

[80]:
```python
# write code to Save the new combined csv file (combined_csv_v2.csv) to your␣
↪local computer
# note this combined file will be also used in part B

data.to_csv(f"{base_path}combined_csv_v2.csv", index=False)
```

Create the training and testing sets again.

[88]:
```python
# Enter your code here
X = data.drop("target", axis=1)
y = data["target"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

[89]:
```python
X_train.head()
```

[89]:
```
         Distance  DepHourofDay  AWND_O  PRCP_O  TAVG_O  AWND_D  PRCP_D  \
1350011     967.0            14      35       0    64.0      28       0
877477      862.0            19      52       0   242.0      38       0
1113872    1846.0            23      30      71    83.0      94       0
1382185    2139.0            11      47       0   120.0      26     881
1221680     370.0            22      31       0   209.0      49       0

         TAVG_D  SNOW_O  SNOW_D  …  Origin_SFO  Dest_CLT  Dest_DEN  \
1350011   166.0     0.0     0.0  …           0         0         0
877477    228.0     0.0     0.0  …           0         0         0
1113872   127.0     0.0     0.0  …           1         0         0
1382185   131.0     0.0     0.0  …           1         0         0
1221680   341.0     0.0     0.0  …           0         0         0

         Dest_DFW  Dest_IAH  Dest_LAX  Dest_ORD  Dest_PHX  Dest_SFO  \
1350011         0         0         0         0         0         1
877477          0         0         1         0         0         0
1113872         0         0         0         1         0         0
1382185         0         0         0         0         0         0
1221680         0         0         0         0         1         0
```

```
         is_holiday_True
1350011                0
877477                 0
1113872                0
1382185                0
1221680                0

[5 rows x 85 columns]
```

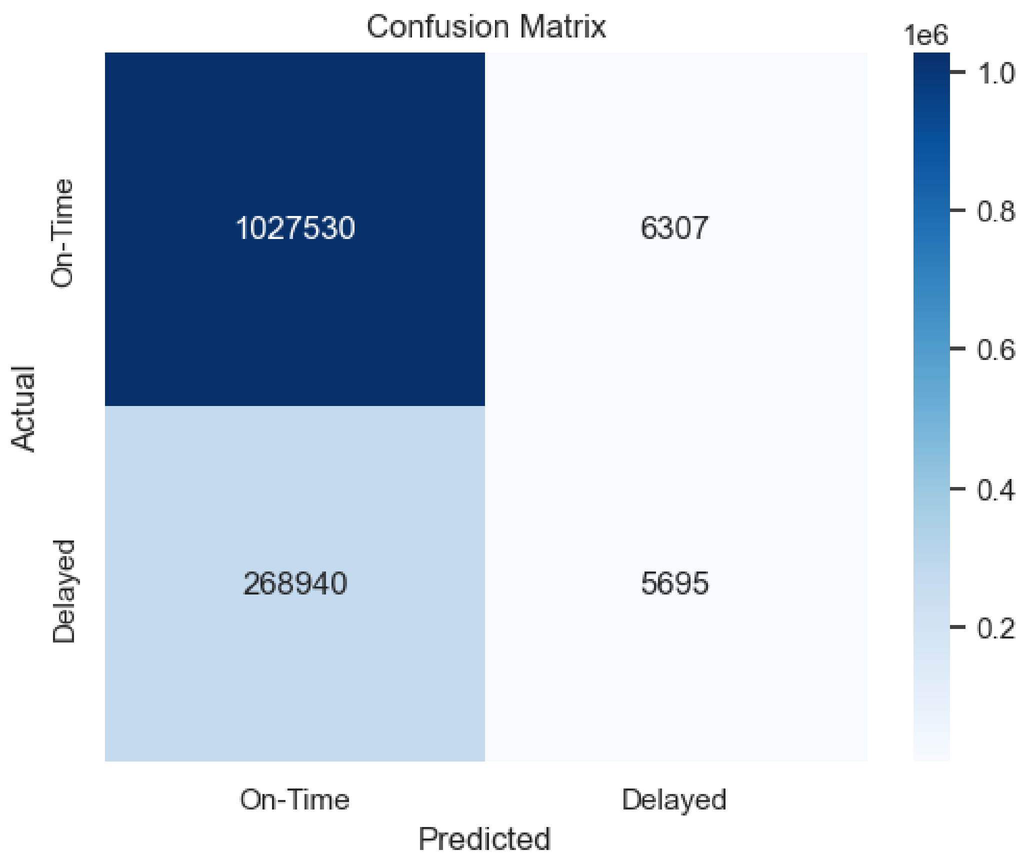
7.0.6 New baseline classifier

Now, see if these new features add any predictive power to the model.

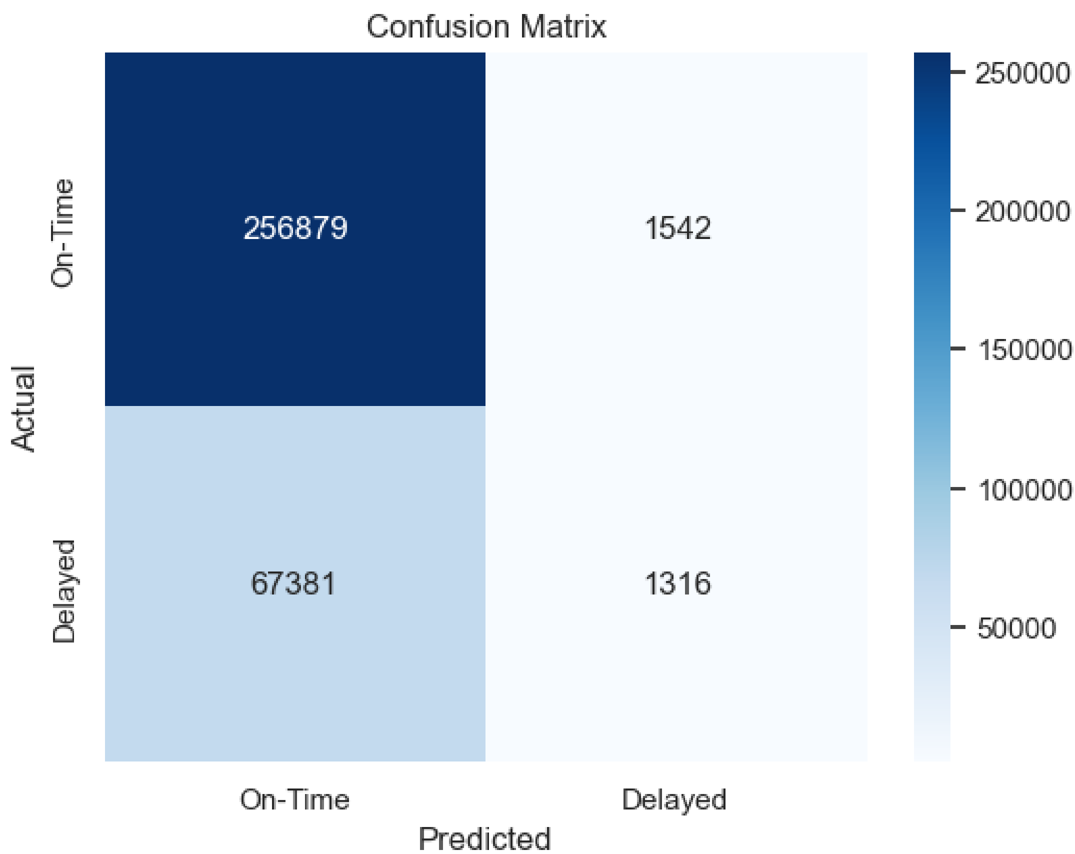
[90]:
```
# Instantiate another logistic regression model
classifier2 = LogisticRegression()

# Fit the model
classifier2.fit(X_train, y_train)

# Predict the target on train and test data
y_train_pred2 = classifier2.predict(X_train)
y_test_pred2 = classifier2.predict(X_test)

# Calculate the accuracy
train_accuracy2 = np.mean(y_train == y_train_pred2)
test_accuracy2 = np.mean(y_test == y_test_pred2)

print(f"Train accuracy: {train_accuracy2}")
print(f"Test accuracy: {test_accuracy2}")
```

```
Train accuracy: 0.7896424226120238
Test accuracy: 0.7893023312688388
```

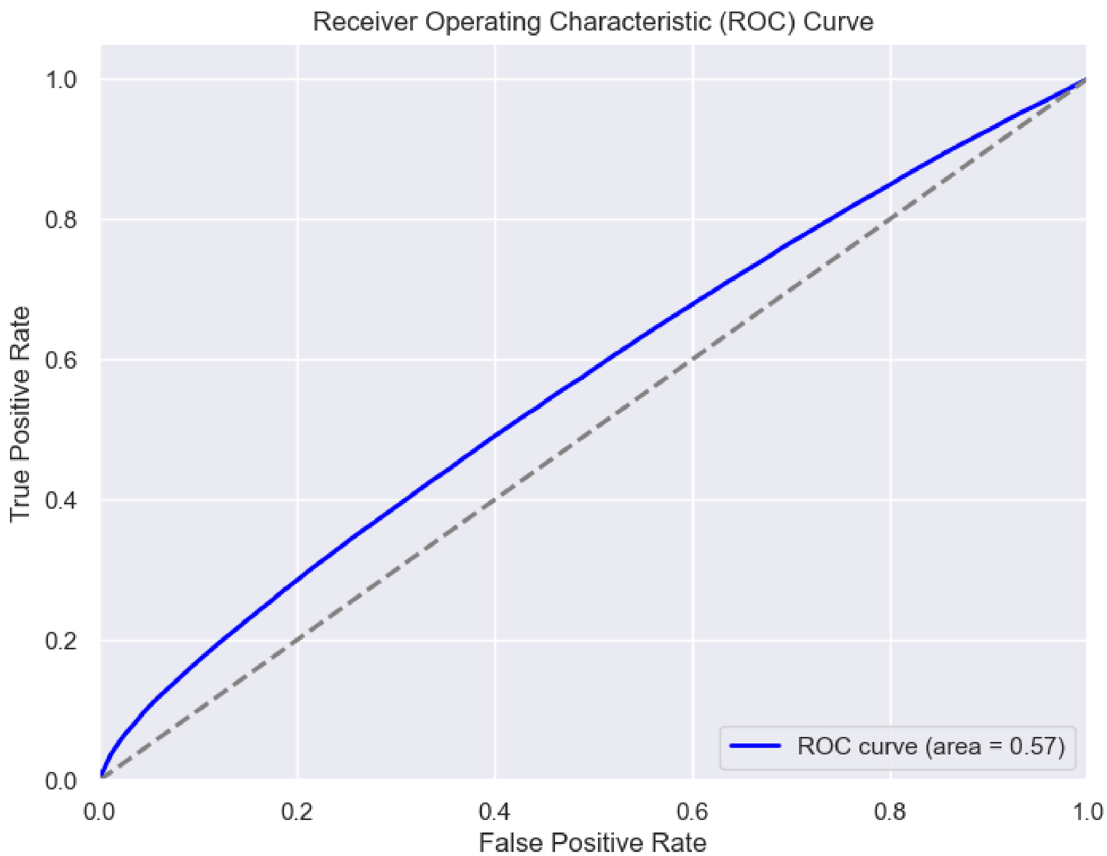
[91]:
```
# confusion matrix on train data
plot_confusion_matrix(y_train, y_train_pred2)
```



[92]:
```
# confusion matrix on test data
plot_confusion_matrix(y_test, y_test_pred2)
```



[93]:
```
# Get predicted probabilities for the positive class (class 1)
y_test_pred_prob = classifier2.predict_proba(X_test)[:, 1]

# Plot ROC curve
plot_roc_curve(y_test, y_test_pred_prob)
```



[94]:
```
# Classification report for train data
print("Classification Report on Train Data")
print(classification_report(y_train, y_train_pred2))

# Classification report for test data
print("Classification Report on Test Data")
print(classification_report(y_test, y_test_pred2))
```

```
Classification Report on Train Data
              precision    recall  f1-score   support

         0.0       0.79      0.99      0.88   1033837
         1.0       0.47      0.02      0.04    274635

    accuracy                           0.79   1308472
   macro avg       0.63      0.51      0.46   1308472
weighted avg       0.73      0.79      0.71   1308472

Classification Report on Test Data
              precision    recall  f1-score   support
```

```
         0.0       0.79      0.99      0.88    258421
         1.0       0.46      0.02      0.04     68697

    accuracy                           0.79    327118
   macro avg       0.63      0.51      0.46    327118
weighted avg       0.72      0.79      0.70    327118
```

Perform the evaluaion as you have done with the previous model and plot/show the same metrics

Question: did you notice a difference by adding the extra data on the results?

Yes, adding extra data improved the recall for the delayed flights. But still the model is not good enough to predict the delayed flights. The model can be further improved by adding more features and using more complex models like Random Forest, Gradient Boosting, etc.

8 Step 6: Using Tableau

Use Tableau to load the combined_csv_v2.csv file and build a dashboard that show your understanding of the data and business problem.

8.0.1 what to do:

1. Load the data into Tableau and build the dashboard
2. Share the dashboard on your Tableau public account
3. Copy the link of the shared dashboard below

Note: The dashboard needs to be self explainable to others, so make it simple and add only the features that you feel heighlight the main question(s) of the prblem statement.

Tableau Public - Flights Delay

8.1 Conclusion

You've now gone through at least a couple iterations of training and evaluating your model. It's time to wrap up this project and reflect on what you've learned and what types of steps you might take moving forward (assuming you had more time). Use the cell below to answer some of these and other relevant questions:

1. Does your model performance meet your business goal? If not, what are some things you'd like to do differently if you had more time for tuning?
2. To what extent did your model improve as you made changes to your dataset? What types of techniques did you employ throughout this project that you felt yielded the greatest improvements in your model?
3. What were some of the biggest challenges you encountered throughout this project?
4. What were the three most important things you learned about machine learning while completing this project?

8.1.1 Answer:

1. **Model Performance and Business Goal:** The model achieves high precision for predicting non-delayed flights (0.79) but struggles significantly with identifying delayed flights, as evidenced by a low recall (0.03) and f1-score (0.06) for the delayed class. This suggests it does not meet the business objective of reliably predicting delayed flights. To improve, I would focus on tuning to better balance recall for delayed flights, perhaps by adjusting class weights or exploring more complex models.

2. **Model Improvement:** The model's accuracy did see incremental improvements with changes in feature engineering, class balancing, and possibly by introducing new variables to enhance feature representation. Techniques such as resampling or adding synthetic data for delayed flights might have been used to address class imbalance.

3. **Challenges Encountered:** One of the biggest challenges was likely the severe class imbalance between delayed and non-delayed flights, which hindered the model's ability to learn meaningful patterns for delays. Additionally, limited feature diversity have made it hard to extract predictive insights for delays.

4. **Key Learnings:**
   - Handling class imbalance is crucial, especially in domains where one class is much rarer but of high business importance.
   - Model evaluation metrics (like recall for the delayed class) should align closely with business objectives.
   - The importance of feature engineering and data preprocessing, as these steps often have a larger impact on model performance than model complexity.