E0123002-A.M.MUKILAN

Step 1: Problem formulation and data collection

Start this project by writing a few sentences that summarize the business problem and the business goal that you want to achieve in this scenario. You can write down your ideas in the following sections. Include a business metric that you would like your team to aspire toward. After you define that information, write the ML problem statement. Finally, add a comment or two about the type of ML this activity represents.

Project presentation: Include a summary of these details in your project presentation.

1. Determine if and why ML is an appropriate solution to deploy for this scenario.

In [ ]:
```
# Write your answer here
```

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

In [ ]:
```
# Write your answer here
```

3. Identify the type of ML problem that you're working with.

In [ ]:
```
# Write your answer here
```

4. Analyze the appropriateness of the data that you're working with.

In [ ]:
```
# Write your answer here
```

Setup

Now that you have decided where you want to focus your attention, you will set up this lab so that you can start solving the problem.

Note: This notebook was created and tested on an `ml.m4.xlarge` notebook instance with 25 GB storage.

In [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()
instance_type='ml.m4.xlarge'

import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
```

```
/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/pandas/core/
computation/expressions.py:21: UserWarning: Pandas requires version '2.8.4' or
newer of 'numexpr' (version '2.7.3' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
Matplotlib is building the font cache; this may take a moment.
```

Step 2: Data preprocessing and visualization

In this data preprocessing phase, you explore and visualize your data to better understand it. First, import the necessary libraries and read the data into a pandas DataFrame. After you import the data, explore the dataset. Look for the shape of the dataset and explore your columns and the types of columns that you will work with (numerical, categorical). Consider performing basic statistics on the features to get a sense of feature means and ranges. Examine your target column closely, and determine its distribution.

Specific questions to consider

Throughout this section of the lab, consider the following questions:

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

Project presentation: Include a summary of your answers to these questions (and other similar questions) in your project presentation.

Start by bringing in the dataset from a public Amazon Simple Storage Service (Amazon S3) bucket to this notebook environment.

In [2]:
```
# download the files

zip_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
base_path = '/home/ec2-user/SageMaker/project/data/FlightDelays/'
csv_base_path = '/home/ec2-user/SageMaker/project/data/csvFlightDelays/'

!mkdir -p {zip_path}
```

```
!mkdir -p {csv_base_path}
!aws s3 cp s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_1.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_11.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2014_11.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_12.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2014_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_2.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_10.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2014_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_3.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_3.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_4.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_7.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_7.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_5.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_6.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_8.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_1.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_12.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2015_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_2.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_11.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2015_11.zip
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_10.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2015_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_3.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_3.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2014_9.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2014_9.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_4.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_5.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_6.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_9.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_9.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_8.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_1.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2015_7.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2015_7.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_10.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2016_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_11.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2016_11.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_12.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2016_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_2.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_3.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_3.zip
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_4.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_5.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_6.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_7.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_7.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_1.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_8.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2016_9.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2016_9.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_10.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2017_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_11.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2017_11.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_4.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_12.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2017_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_2.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_3.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_3.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_1.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_1.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_9.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_9.zip
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_11.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2018_11.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_12.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2018_12.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_10.zip to ../
project/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pr
esent_2018_10.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_3.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_3.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_2.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_2.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_4.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_4.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_6.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_5.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_5.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_5.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_8.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_9.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_9.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_8.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_8.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_7.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_7.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2017_6.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2017_6.zip
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a/On_Time_Reporting_Carrier_On_Time_Performance_1987_present_2018_7.zip to ../p
roject/data/FlightDelays/On_Time_Reporting_Carrier_On_Time_Performance_1987_pre
sent_2018_7.zip
```

In [3]:
```
zip_files = [str(file) for file in list(Path(base_path).iterdir()) if '.zip' in
len(zip_files)
```

Out[3]: 60

Extract comma-separated values (CSV) files from the .zip files.

In [4]:
```
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
    """

    try:
        with ZipFile(zipFile_name, 'r') as z:
            print(f'Extracting {zipFile_name} ')
            z.extractall(path=file_path)
    except:
        print(f'zip2csv failed for {zipFile_name}')

for file in zip_files:
    zip2csv(file, csv_base_path)

print("Files Extracted")
```

```
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_1.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_1.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_1.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_8.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_7.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_2.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_1.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_2.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_7.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_2.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_8.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_2.zip
```

```
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_7.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_7.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_7.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_11.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_3.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_1.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_2.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_8.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2014_8.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_5.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_9.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_4.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2016_8.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2015_12.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_6.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2018_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
_Carrier_On_Time_Performance_1987_present_2017_10.zip
Extracting /home/ec2-user/SageMaker/project/data/FlightDelays/On_Time_Reporting
```

```
_Carrier_On_Time_Performance_1987_present_2016_12.zip
Files Extracted
```

In [5]:
```
csv_files = [str(file) for file in list(Path(csv_base_path).iterdir()) if '.csv'
len(csv_files)
```

Out[5]:
```
60
```

Before you load the CSV file, read the HTML file from the extracted folder. This HTML file includes the background and more information about the features that are included in the dataset.

In [6]:
```
from IPython.display import IFrame

IFrame(src=os.path.relpath(f"{csv_base_path}readme.html"), width=1000, height=60
```

Out[6]:

Load sample CSV file

Before you combine all the CSV files, examine the data from a single CSV file. By using pandas, read the

`On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.csv` file first. You can use the built-in `read_csv` function in Python (pandas.read_csv documentation).

In [7]:
```
df_temp = pd.read_csv(f"{csv_base_path}On_Time_Reporting_Carrier_On_Time_Perform
```

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

Hint: To view the rows and columns of a DataFrame, use the `<DataFrame>.shape` function. To view the column names, use the `<DataFrame>.columns` function.

In [8]:
```
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.

Hint: To print `x` number of rows, use the built-in `head(x)` function in pandas.

In [9]:
```
df_temp.head(10)
```

Out[9]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	DOT_ID_Re
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	

10 rows × 110 columns

Question: Print all the columns in the dataset. To view the column names, use `<DataFrame>.columns` .

In [10]:
```
print(f'The column names are :')
# List comprehension to filter columns containing "Del"
for col in [c for c in df_temp.columns if "Del" in c]:
    print(col)
```

```
The column names are :
DepDelay
DepDelayMinutes
DepDel15
DepartureDelayGroups
ArrDelay
ArrDelayMinutes
ArrDel15
ArrivalDelayGroups
CarrierDelay
WeatherDelay
NASDelay
SecurityDelay
LateAircraftDelay
DivArrDelay
```

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: To include values that pass certain `if` statement criteria, you can use a Python list comprehension.

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

Hint: To check if the value is in a list, you can use the `in` keyword (Python in Keyword documentation).

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

In [12]:
```
# Print all columns that contain "Del"
[col for col in df_temp.columns if "Del" in col]
```

Out[12]:
```
['DepDelay',
 'DepDelayMinutes',
 'DepDel15',
 'DepartureDelayGroups',
 'ArrDelay',
 'ArrDelayMinutes',
 'ArrDel15',
 'ArrivalDelayGroups',
 'CarrierDelay',
 'WeatherDelay',
 'NASDelay',
 'SecurityDelay',
 'LateAircraftDelay',
 'DivArrDelay']
```

Here are some more questions to help you learn 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?

Hints

- To show the dimensions of the DataFrame, use `df_temp.shape`.
- To refer to a specific column, use `df_temp.columnName` (for example, `df_temp.CarrierDelay`).
- To get unique values for a column, use `df_temp.column.unique()` (for, example `df_temp.Year.unique()`).

In [13]:
```
# Ensure FlightDate is datetime
df_temp['FlightDate'] = pd.to_datetime(df_temp['FlightDate'])

print("The #rows and #columns are ", df_temp.shape[0], " and ", df_temp.shape[1]
print("The years in this dataset are: ", sorted(df_temp['Year'].unique()))
print("The months covered in this dataset are: ", sorted(df_temp['Month'].unique
print("The date range for data is :", df_temp['FlightDate'].min(), " to ", df_te
print("The airlines covered in this dataset are: ", list(df_temp['Reporting_Airl
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 00:00:00  to  2018-09-30 00:00:00
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', 'V
LD', 'JFK', 'RDU', 'CHS', 'DTW', 'GRB', 'PVD', 'SHV', 'FNT', 'PIT', 'RIC', 'RS
T', 'RSW', 'CVG', 'LIT', 'ORD', 'JAX', 'TRI', 'BOS', 'CWA', 'DCA', 'CHO', 'AV
P', 'IND', 'GRR', 'BTR', 'MEM', 'TUL', 'CLE', 'STL', 'BTV', 'OMA', 'MGM', 'TV
C', 'SAV', 'GSP', 'EWR', 'OAJ', 'BNA', 'MCI', 'TLH', 'ROC', 'LEX', 'PWM', 'BU
F', 'AGS', 'CLT', 'GSO', 'BWI', 'SAT', 'PHL', 'TYS', 'ACK', 'DSM', 'GNV', 'AV
L', 'BGR', 'MHT', 'ILM', 'MOT', 'IAH', 'SBN', 'SYR', 'ORF', 'MKE', 'XNA', 'MS
Y', 'PBI', 'ABE', 'HPN', 'EVV', 'ALB', 'LNK', 'AUS', 'PHF', 'CHA', 'GTR', 'BM
I', 'BQK', 'CID', 'CAK', 'ATW', 'ABY', 'CAE', 'SRQ', 'MLI', 'BHM', 'IAD', 'CS
G', 'CMH', 'MCO', 'MBS', 'FLL', 'SDF', 'TPA', 'MVY', 'LAS', 'LGB', 'SFO', 'SA
N', 'LAX', 'RNO', 'PDX', 'ANC', 'ABQ', 'SLC', 'DEN', 'PHX', 'OAK', 'SMF', 'SJ
U', 'SEA', 'HOU', 'STX', 'BUR', 'SWF', 'SJC', 'DAB', 'BQN', 'PSE', 'ORH', 'HY
A', 'STT', 'ONT', 'HRL', 'ICT', 'ISP', 'LBB', 'MAF', 'MDW', 'OKC', 'PNS', 'SN
A', 'TUS', 'AMA', 'BOI', 'CRP', 'DAL', 'ECP', 'ELP', 'GEG', 'LFT', 'MFE', 'MD
T', 'JAN', 'COS', 'MOB', 'VPS', 'MTJ', 'DRO', 'GPT', 'BFL', 'MRY', 'SBA', 'PS
P', 'FSD', 'BRO', 'RAP', 'COU', 'STS', 'PIA', 'FAT', 'SBP', 'FSM', 'HSV', 'BI
S', 'DAY', 'BZN', 'MIA', 'EYW', 'MYR', 'HHH', 'GJT', 'FAR', 'SGF', 'HOB', 'CL
L', 'LRD', 'AEX', 'ERI', 'MLU', 'LCH', 'ROA', 'LAW', 'MHK', 'GRK', 'SAF', 'GR
I', 'JLN', 'ROW', 'FWA', 'CRW', 'LAN', 'OGG', 'HNL', 'KOA', 'EGE', 'LIH', 'ML
B', 'JAC', 'FAI', 'RDM', 'ADQ', 'BET', 'BRW', 'SCC', 'KTN', 'YAK', 'CDV', 'JN
U', 'SIT', 'PSG', 'WRG', 'OME', 'OTZ', 'ADK', 'FCA', 'FAY', 'PSC', 'BIL', 'MS
O', 'ITO', 'PPG', 'MFR', 'EUG', 'GUM', 'SPN', 'DLH', 'TTN', 'BKG', 'SFB', 'PI
E', 'PGD', 'AZA', 'SMX', 'RFD', 'SCK', 'OWB', 'HTS', 'BLV', 'IAG', 'USA', 'GF
K', 'BLI', 'ELM', 'PBG', 'LCK', 'GTF', 'OGD', 'IDA', 'PVU', 'TOL', 'PSM', 'CK
B', 'HGR', 'SPI', 'STC', 'ACT', 'TYR', 'ABI', 'AZO', 'CMI', 'BPT', 'GCK', 'MQ
T', 'ALO', 'TXK', 'SPS', 'SWO', 'DBQ', 'SUX', 'SJT', 'GGG', 'LSE', 'LBE', 'AC
Y', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'IMT', 'WYS', 'CPR', 'SCE', 'HL
N', 'SUN', 'ISN', 'CMX', 'EAU', 'LWB', 'SHD', 'LBF', 'HYS', 'SLN', 'EAR', 'VE
L', 'CNY', 'GCC', 'RKS', 'PUB', 'LBL', 'MKG', 'PAH', 'CGI', 'UIN', 'BFF', 'DV
L', 'JMS', 'LAR', 'SGU', 'PRC', 'ASE', 'RDD', 'ACV', 'OTH', 'COD', 'LWS', 'AB
R', 'APN', 'ESC', 'PLN', 'BJI', 'BRD', 'BTM', 'CDC', 'CIU', 'EKO', 'TWF', 'HI
B', 'BGM', 'RHI', 'ITH', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
The Destination airports covered are:  ['CVG', 'PWM', 'RDU', 'MSP', 'MSN', 'SH
V', 'CLT', 'PIT', 'RIC', 'IAH', 'ATL', 'JFK', 'DCA', 'DTW', 'LGA', 'TYS', 'PV
D', 'FNT', 'LIT', 'BUF', 'ORD', 'TRI', 'IND', 'BGR', 'AVP', 'BWI', 'LEX', 'BD
L', 'GRR', 'CWA', 'TUL', 'MEM', 'AGS', 'EWR', 'MGM', 'PHL', 'SYR', 'OMA', 'ST
L', 'TVC', 'ORF', 'CLE', 'ABY', 'BOS', 'OAJ', 'TLH', 'BTR', 'SAT', 'JAX', 'BN
A', 'CHO', 'VLD', 'ROC', 'DFW', 'GNV', 'ACK', 'PBI', 'CHS', 'GRB', 'MOT', 'MK
E', 'DSM', 'ILM', 'GSO', 'MCI', 'SBN', 'BTV', 'MVY', 'XNA', 'RST', 'EVV', 'HP
N', 'RSW', 'MDT', 'ROA', 'GSP', 'MCO', 'CSG', 'SAV', 'PHF', 'ALB', 'CHA', 'AB
E', 'BMI', 'MSY', 'IAD', 'GTR', 'CID', 'CAK', 'ATW', 'AUS', 'BQK', 'MLI', 'CA
E', 'CMH', 'AVL', 'MBS', 'FLL', 'SDF', 'TPA', 'LNK', 'SRQ', 'MHT', 'BHM', 'LA
S', 'SFO', 'SAN', 'RNO', 'LGB', 'ANC', 'PDX', 'SJU', 'ABQ', 'SLC', 'DEN', 'LA
X', 'PHX', 'OAK', 'SMF', 'SEA', 'STX', 'BUR', 'DAB', 'SJC', 'SWF', 'HOU', 'BQ
N', 'PSE', 'ORH', 'HYA', 'STT', 'ONT', 'DAL', 'ECP', 'ELP', 'HRL', 'MAF', 'MD
W', 'OKC', 'PNS', 'SNA', 'AMA', 'BOI', 'GEG', 'ICT', 'LBB', 'TUS', 'ISP', 'CR
P', 'MFE', 'LFT', 'VPS', 'JAN', 'COS', 'MOB', 'DRO', 'GPT', 'BFL', 'COU', 'SB
P', 'MTJ', 'SBA', 'PSP', 'FSD', 'FSM', 'BRO', 'PIA', 'STS', 'FAT', 'RAP', 'MR
Y', 'HSV', 'BIS', 'DAY', 'BZN', 'MIA', 'EYW', 'MYR', 'HHH', 'GJT', 'FAR', 'ML
U', 'LRD', 'CLL', 'LCH', 'FWA', 'GRK', 'SGF', 'HOB', 'LAW', 'MHK', 'SAF', 'JL
N', 'ROW', 'GRI', 'AEX', 'CRW', 'LAN', 'ERI', 'HNL', 'KOA', 'OGG', 'EGE', 'LI
H', 'JAC', 'MLB', 'RDM', 'BET', 'ADQ', 'BRW', 'SCC', 'FAI', 'JNU', 'CDV', 'YA
K', 'SIT', 'KTN', 'WRG', 'PSG', 'OME', 'OTZ', 'ADK', 'FCA', 'BIL', 'PSC', 'FA
Y', 'MSO', 'ITO', 'PPG', 'MFR', 'DLH', 'EUG', 'GUM', 'SPN', 'TTN', 'BKG', 'AZ
```

```
A', 'SFB', 'LCK', 'BLI', 'SCK', 'PIE', 'RFD', 'PVU', 'PBG', 'BLV', 'PGD', 'SP
I', 'USA', 'TOL', 'IDA', 'ELM', 'HTS', 'HGR', 'SMX', 'OGD', 'GFK', 'STC', 'GT
F', 'IAG', 'CKB', 'OWB', 'PSM', 'ABI', 'TYR', 'ALO', 'SUX', 'AZO', 'ACT', 'CM
I', 'BPT', 'TXK', 'SWO', 'SPS', 'DBQ', 'SJT', 'GGG', 'LSE', 'MQT', 'GCK', 'LB
E', 'ACY', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'WYS', 'SCE', 'IMT', 'HL
N', 'ASE', 'SUN', 'ISN', 'EAR', 'SGU', 'VEL', 'SHD', 'LWB', 'MKG', 'SLN', 'HY
S', 'BFF', 'PUB', 'LBL', 'CMX', 'EAU', 'PAH', 'UIN', 'RKS', 'CGI', 'CNY', 'JM
S', 'DVL', 'LAR', 'GCC', 'LBF', 'PRC', 'RDD', 'ACV', 'OTH', 'COD', 'LWS', 'AB
R', 'APN', 'PLN', 'BJI', 'CPR', 'BRD', 'BTM', 'CDC', 'CIU', 'ESC', 'EKO', 'IT
H', 'HIB', 'BGM', 'TWF', 'RHI', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
```

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

Hint: To find the values for each airport by using the **Origin** and **Dest** columns, you can use the `values_count` function in pandas (pandas.Series.value_counts documentation).

In [14]:
```python
print("Origin airport counts:")
print(df_temp['Origin'].value_counts())

print("\nDestination airport counts:")
print(df_temp['Dest'].value_counts())
```

```
Origin airport counts:
Origin
ATL    31525
ORD    28257
DFW    22802
DEN    19807
CLT    19655
       ...
PPG        8
OGD        8
HGR        8
STC        5
HYA        4
Name: count, Length: 346, dtype: int64

Destination airport counts:
Dest
ATL    31521
ORD    28250
DFW    22795
DEN    19807
CLT    19654
       ...
OGD        8
OWB        8
PPG        8
STC        5
HYA        4
Name: count, Length: 346, dtype: int64
```

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

Hint: You can use the `sort_values` function in pandas (pandas.DataFrame.sort_values documentation).

In [15]:
```
print("Top 15 Origin Airports:")
print(df_temp['Origin'].value_counts().sort_values(ascending=False).head(15))

print("\nTop 15 Destination Airports:")
print(df_temp['Dest'].value_counts().sort_values(ascending=False).head(15))
```

```
Top 15 Origin Airports:
Origin
ATL    31525
ORD    28257
DFW    22802
DEN    19807
CLT    19655
LAX    17875
SFO    14332
IAH    14210
LGA    13850
MSP    13349
LAS    13318
PHX    13126
DTW    12725
BOS    12223
SEA    11872
Name: count, dtype: int64

Top 15 Destination Airports:
Dest
ATL    31521
ORD    28250
DFW    22795
DEN    19807
CLT    19654
LAX    17873
SFO    14348
IAH    14203
LGA    13850
MSP    13347
LAS    13322
PHX    13128
DTW    12724
BOS    12227
SEA    11877
Name: count, dtype: int64
```

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

The **ArrDel15** column is an indicator variable that takes the value *1* when the delay is more than 15 minutes. Otherwise, it takes a value of *0*.

You could use this as a target column for the classification problem.

Now, assume that you are traveling from San Francisco to Los Angeles on a work trip. You want to better manage your reservations in Los Angeles. Thus, want to have an idea of whether your flight will be delayed, given a set of features. How many features from this dataset would you need to 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 the destination. If there were a sudden weather delay 10 minutes before landing, this data wouldn't be helpful to 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 five airlines: UA, OO, WN, AA, DL

This information should help reduce the size of data across the CSV files that will be combined.

Combine all CSV files

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`. To check if the `val` is in the DataFrame column, use the `isin` function in pandas (pandas.DataFram.isin documentation). Then, choose the rows that include it.

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

4. Concatenate the DataFrame with the empty DataFrame

In [16]:
```
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
    """
```

```
    df = pd.DataFrame()

    for file in csv_files:
        df_temp = pd.read_csv(file)
        df_temp = df_temp[filter_cols]
        for col, val in zip(subset_cols,subset_vals):
            df_temp = df_temp[df_temp[col].isin(val)]

        df = pd.concat([df, df_temp], axis=0)

    df.to_csv(file_name, index=False)
    print(f'Combined csv stored at {file_name}')
```

In [17]:
```
#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 an
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 previous function to merge all the different files into a single file that you can read easily.

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

In [18]:
```
start = time.time()
combined_csv_filename = f"{base_path}combined_files.csv"
combine_csv(csv_files, cols, subset_cols, subset_vals, combined_csv_filename)
print(f'CSVs merged in {round((time.time() - start)/60,2)} minutes')
```

```
Combined csv stored at /home/ec2-user/SageMaker/project/data/FlightDelays/combi
ned_files.csv
CSVs merged in 4.5 minutes
```

Load the dataset

Load the combined dataset.

In [19]:
```
data = pd.read_csv(combined_csv_filename)
```

Print the first five records.

In [20]:
```
data.head()
```

Out[20]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	Origin	Ori
0	2016	4	11	1	2	2016-11-01	AA	SFO	
1	2016	4	11	2	3	2016-11-02	AA	SFO	
2	2016	4	11	3	4	2016-11-03	AA	SFO	
3	2016	4	11	4	5	2016-11-04	AA	SFO	
4	2016	4	11	5	6	2016-11-05	AA	SFO	

Here are some more questions to help you learn 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?

In [21]:

```
# Ensure FlightDate is datetime
data['FlightDate'] = pd.to_datetime(data['FlightDate'])

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'].uniq
print("The date range for data is :" , data['FlightDate'].min(), " to ", data['F
print("The airlines covered in this dataset are: ", list(data['Reporting_Airline
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:  [2016, 2018, 2017, 2014, 2015]
The months covered in this dataset are:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1
2]
The date range for data is : 2014-01-01 00:00:00  to  2018-12-31 00:00:00
The airlines covered in this dataset are:  ['AA', 'DL', 'WN', 'UA', 'OO']
The Origin airports covered are:  ['SFO', 'DFW', 'ORD', 'LAX', 'IAH', 'DEN', 'A
TL', 'PHX', 'CLT']
The Destination airports covered are:  ['DFW', 'SFO', 'ORD', 'LAX', 'CLT', 'PH
X', 'IAH', 'DEN', 'ATL']
```

Define your target column: **is_delay** (*1* means that the arrival time delayed more than 15 minutes, and *0* means all other cases). To rename the column from **ArrDel15** to *is_delay*, use the `rename` method .

Hint: You can use the `rename` function in pandas (pandas.DataFrame.rename documentation).

For example:

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

In [22]:
```
data.rename(columns={'ArrDel15': 'is_delay'}, inplace=True)
```

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

Hint: `isnull()` detects whether the particular value is null or not. It returns a boolean (*True* or *False*) in its place. To sum the number of columns, use the `sum(axis=0)` function (for example, `df.isnull().sum(axis=0)`).

In [23]:
```
data.isnull()
```

Out[23]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	Orig
0	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	
1658125	False	False	False	False	False	False	False	Fa
1658126	False	False	False	False	False	False	False	Fa
1658127	False	False	False	False	False	False	False	Fa
1658128	False	False	False	False	False	False	False	Fa
1658129	False	False	False	False	False	False	False	Fa

1658130 rows × 20 columns

The arrival delay details and airtime are missing for 22,540 out of 1,658,130 rows, which is 1.3 percent. You can either remove or impute these rows. The documentation doesn't mention any information about missing rows.

In [24]:
```
### Remove null columns
data = data[~data.is_delay.isnull()]
data.isnull().sum(axis = 0)
```

Out[24]:
```
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             0
ArrDelayMinutes      0
is_delay             0
AirTime              0
dtype: int64
```

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

In [25]:
```
data['DepHourofDay'] = (data['CRSDepTime']//100)
```

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 a value of *0* or *1*, you could use a classification algorithm.

Before you start modeling, it's a good practice to look at feature distribution, correlations, and others.

- This will give you an idea of any non-linearity or patterns in the data
  - Linear models: Add power, exponential, or interaction features
  - Try a non-linear model
- Data imbalance
  - Choose metrics that won't give biased model performance (accuracy versus the area under the curve, or AUC)
  - Use weighted or custom loss functions
- Missing data
  - Do imputation based on simple statistics -- mean, median, mode (numerical variables), frequent class (categorical variables)
  - Clustering-based imputation (k-nearest neighbors, or KNNs, to predict column value)
  - Drop column

Data exploration

Check the classes *delay* versus *no delay*.

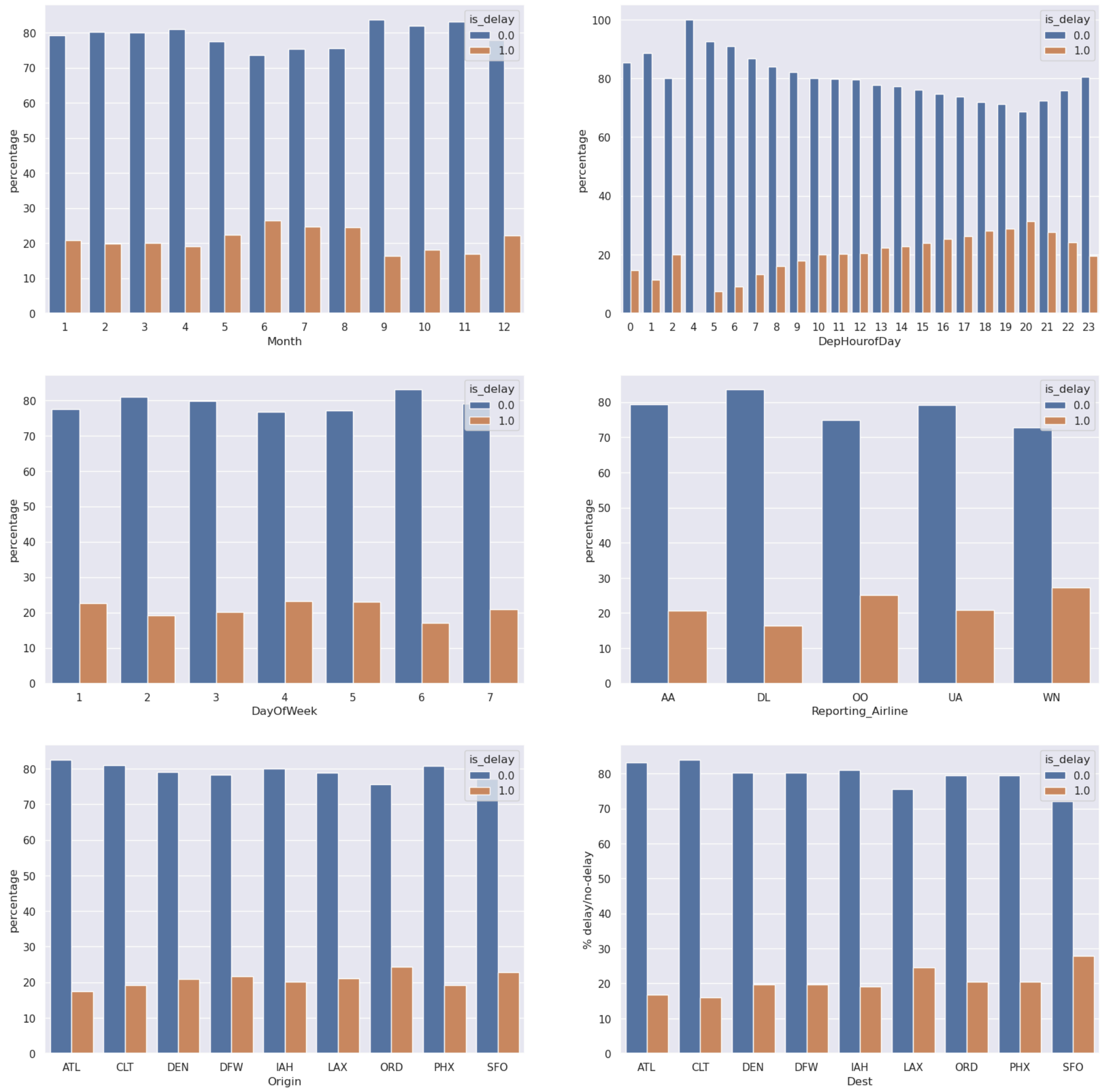
In [26]:
```
(data.groupby('is_delay').size()/len(data) ).plot(kind='bar')# Enter your code h
plt.ylabel('Frequency')
plt.title('Distribution of classes')
plt.show()
```

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

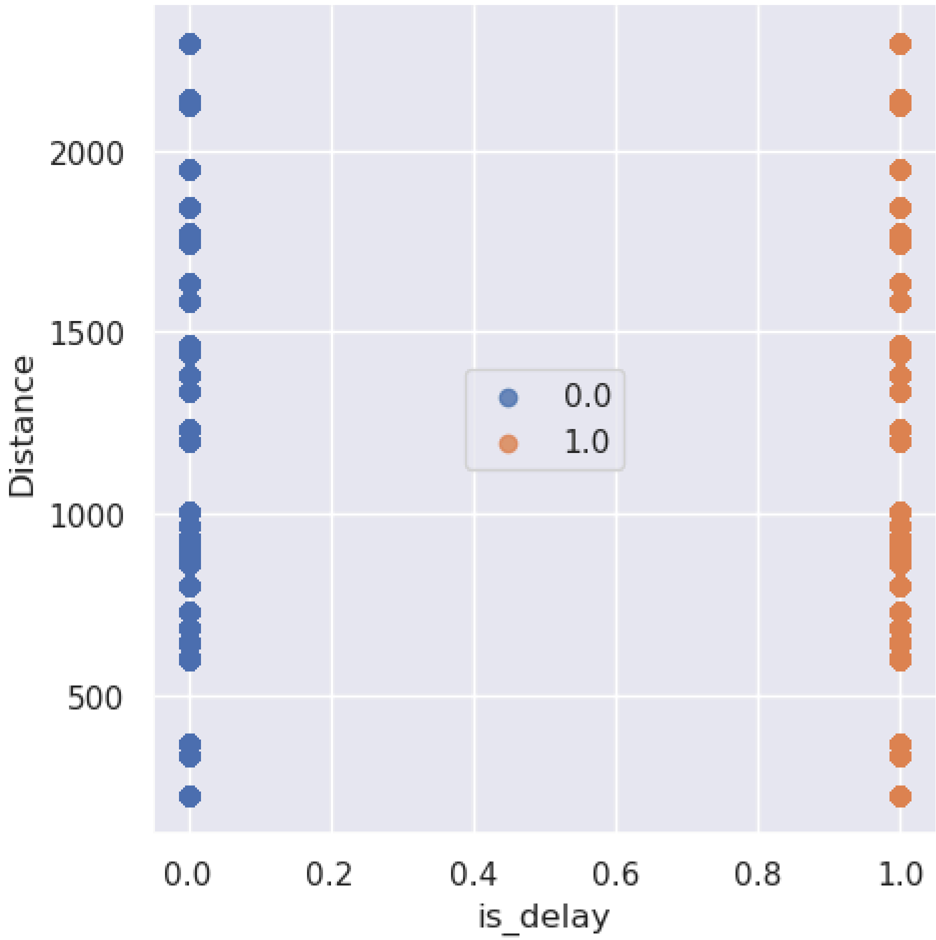
In [ ]:
```
# Enter your answer here
```

Run the following two cells and answer the questions.

In [27]:
```
viz_columns = ['Month', 'DepHourofDay', 'DayOfWeek', 'Reporting_Airline', 'Origi
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(
    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()
```



In [28]:
```
sns.lmplot( x="is_delay", y="Distance", data=data, fit_reg=False, hue='is_delay'
plt.legend(loc='center')
plt.xlabel('is_delay')
plt.ylabel('Distance')
plt.show()
```



Questions

Using the data from the previous charts, answer these 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?

In [ ]:
```
# Enter your answers here
```

Features

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

In [29]:
```
data.columns
```

Out[29]:
```
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')
```

In [30]:
```
data.dtypes
```

Out[30]:
```
Year                          int64
Quarter                       int64
Month                         int64
DayofMonth                    int64
DayOfWeek                     int64
FlightDate           datetime64[ns]
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 only 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 needed to do a one-hot encoding of this variable, 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.
- In this case, you only need to split into buckets.

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

In [31]:
```
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')
```

To use one-hot encoding, use the `get_dummies` function in pandas for the categorical columns that you selected. Then, you can concatenate those generated features to your original dataset by using the `concat` function in pandas. For encoding categorical variables, you can also use *dummy encoding* by using a keyword `drop_first=True`. For more information about dummy encoding, see [Dummy variable (statistics)](#).

For example:

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

In [33]:
```
# Create dummy variables for categorical columns
data_dummies = pd.get_dummies(data[categorical_columns], drop_first=True)
data_dummies = data_dummies.replace({True: 1, False: 0})

# Concatenate the dummy variables with the original data
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 columns.

Hint: Use the `shape` and `columns` properties.

In [34]:
```
data.shape
```

Out[34]:
```
(1635590, 94)
```

In [35]:
```
data.columns
```

Out[35]:
```
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_1', 'DepHourofDay_2',
       'DepHourofDay_4', 'DepHourofDay_5', 'DepHourofDay_6', 'DepHourofDay_7',
       'DepHourofDay_8', 'DepHourofDay_9', 'DepHourofDay_10',
       'DepHourofDay_11', 'DepHourofDay_12', 'DepHourofDay_13',
       'DepHourofDay_14', 'DepHourofDay_15', 'DepHourofDay_16',
       'DepHourofDay_17', 'DepHourofDay_18', 'DepHourofDay_19',
       'DepHourofDay_20', 'DepHourofDay_21', 'DepHourofDay_22',
       'DepHourofDay_23'],
      dtype='object')
```

You are now ready to train the model. Before you split the data, rename the **is_delay** column to *target*.

Hint: You can use the `rename` function in pandas (pandas.DataFrame.rename documentation).

In [36]:
```
data.rename(columns={'is_delay': 'target'}, inplace=True)
```

End of Step 2

Save the project file to your local computer. Follow these steps:

1. In the file explorer on the left, right-click the notebook that you're working on.
2. Choose **Download**, and save the file locally.

This action downloads the current notebook to the default download folder on your computer.

Step 3: Model training and evaluation

You must include some preliminary steps when you convert the dataset from a DataFrame to a format that a machine learning algorithm can use. For Amazon SageMaker, you must perform these steps:

1. Split the data into `train_data`, `validation_data`, and `test_data` by using `sklearn.model_selection.train_test_split`.
2. Convert the dataset to an appropriate file format that the Amazon SageMaker training job can use. This can be either a CSV file or record protobuf. For more information, see Common Data Formats for Training.
3. Upload the data to your S3 bucket. If you haven't created one before, see Create a Bucket.

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

Project presentation: In your project presentation, write down the key decisions that you made in this phase.

Train-test split

In [37]:
```
from sklearn.model_selection import train_test_split
def split_data(data):
    train, test_and_validate = train_test_split(data, test_size=0.2, random_stat
    test, validate = train_test_split(test_and_validate, test_size=0.5, random_s
    return train, validate, test
```

In [38]:
```
train, validate, test = split_data(data)
print(train['target'].value_counts())
print(test['target'].value_counts())
print(validate['target'].value_counts())
```

```
target
0.0    1033806
1.0     274666
Name: count, dtype: int64
target
0.0    129226
1.0     34333
Name: count, dtype: int64
target
0.0    129226
1.0     34333
Name: count, dtype: int64
```

Sample answer

```
0.0    1033570
1.0     274902
Name: target, dtype: int64
0.0    129076
1.0     34483
Name: target, dtype: int64
0.0    129612
1.0     33947
Name: target, dtype: int64
```

Baseline classification model

In [40]:

```
import sagemaker
from sagemaker.serializers import CSVSerializer
from sagemaker.amazon.amazon_estimator import RecordSet
import boto3

# Instantiate the LinearLearner estimator object with 1 ml.m4.xlarge
# Instantiate the LinearLearner estimator object with 1 ml.m4.xlarge
classifier_estimator = sagemaker.LinearLearner(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.m4.xlarge',
    predictor_type='binary_classifier',
    binary_classifier_model_selection_criteria='cross_entropy_loss'
)
```

Sample code

```
num_classes = len(pd.unique(train_labels))
classifier_estimator = 
sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                              instance_count=1,

instance_type='ml.m4.xlarge',

predictor_type='binary_classifier',

binary_classifier_model_selection_criteria = 
'cross_entropy_loss')
```

Linear learner accepts training data in protobuf or CSV content types. It also accepts inference requests in protobuf, CSV, or JavaScript Object Notation (JSON) content types. Training data has features and ground-truth labels, but the data in an inference request has only features.

In a production pipeline, AWS recommends converting the data to the Amazon SageMaker protobuf format and storing it in Amazon S3. To get up and running quickly, AWS provides the `record_set` operation for converting and uploading the dataset when it's small enough to fit in local memory. It accepts NumPy arrays like the ones you already have, so you will use it for this step. The `RecordSet` object will track the temporary Amazon S3 location of your data. Create train, validation, and test records by using the `estimator.record_set` function. Then, start your training job by using the `estimator.fit` function.

In [41]:
```
### Create train, validate, and test records
train_records = classifier_estimator.record_set(train.values[:, 1:].astype(np.fl
val_records = classifier_estimator.record_set(validate.values[:, 1:].astype(np.f
test_records = classifier_estimator.record_set(test.values[:, 1:].astype(np.floa
```

Now, train your model on the dataset that you just uploaded.

Sample code

```
linear.fit([train_records,val_records,test_records])
```

In [43]:
```
# Train the model using train, validation, and test datasets
classifier_estimator.fit([train_records, val_records, test_records])
```

```
INFO:sagemaker.image_uris:Same images used for training and inference. Defaulti
ng to image scope: inference.
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: None.
INFO:sagemaker:Creating training-job with name: linear-learner-2025-08-17-01-39
-35-087
2025-08-17 01:39:36 Starting - Starting the training job...
2025-08-17 01:40:02 Starting - Preparing the instances for training...
2025-08-17 01:40:27 Downloading - Downloading input data...
2025-08-17 01:41:07 Downloading - Downloading the training image............
2025-08-17 01:42:43 Training - Training image download completed. Training in p
rogress.........................
2025-08-17 01:47:04 Uploading - Uploading generated training model...
2025-08-17 01:47:22 Completed - Training job completed
..Training seconds: 416
Billable seconds: 416
```

Model evaluation

In this section, you will evaluate your trained model.

First, examine the metrics for the training job:

In [44]:
```
sagemaker.analytics.TrainingJobAnalytics(classifier_estimator._current_job_name,
                                         metric_names = ['test:objective_loss',
```

```
                                                          'test:binary_f_beta',
                                                          'test:precision',
                                                          'test:recall']
                                         ).dataframe()
```

```
WARNING:sagemaker.analytics:Warning: No metrics called test:objective_loss foun
d
WARNING:sagemaker.analytics:Warning: No metrics called test:binary_f_beta found
WARNING:sagemaker.analytics:Warning: No metrics called test:precision found
WARNING:sagemaker.analytics:Warning: No metrics called test:recall found
```

Out[44]: —

Next, set up some functions that will help load the test data into Amazon S3 and perform a prediction by using the batch prediction function. Using batch prediction will help reduce costs because the instances will only run when predictions are performed on the supplied test data.

Note: Replace `<LabBucketName>` with the name of the lab bucket that was created during the lab setup.

In [45]:
```
from sagemaker.analytics import TrainingJobAnalytics

# Get the last training job name
job_name = classifier_estimator.latest_training_job.name
print(f"Using job name: {job_name}")

# Retrieve evaluation metrics
metrics_df = TrainingJobAnalytics(
    job_name,
    metric_names=[
        'test:objective_loss',
        'test:binary_f_beta',
        'test:precision',
        'test:recall'
    ]
).dataframe()

display(metrics_df)
```

```
WARNING:sagemaker.analytics:Warning: No metrics called test:objective_loss foun
d
WARNING:sagemaker.analytics:Warning: No metrics called test:binary_f_beta found
WARNING:sagemaker.analytics:Warning: No metrics called test:precision found
WARNING:sagemaker.analytics:Warning: No metrics called test:recall found
Using job name: linear-learner-2025-08-17-01-39-35-087
```

—

In [46]:
```
import sagemaker
bucket = sagemaker.Session().default_bucket()
print(bucket)
```

```
sagemaker-us-east-1-834320805887
```

In [47]:
```
import io
bucket='sagemaker-us-east-1-834320805887'
prefix='flight-linear'
train_file='flight_train.csv'
test_file='flight_test.csv'
```

```
validate_file='flight_validate.csv'
whole_file='flight.csv'
s3_resource = boto3.Session().resource('s3')

def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).pu
```

```
INFO:botocore.credentials:Found credentials from IAM Role: BaseNotebookInstance
Ec2InstanceRole
```

In [48]:

```
def batch_linear_predict(test_data, estimator):
    batch_X = test_data.iloc[:,1:];
    batch_X_file='batch-in.csv'
    upload_s3_csv(batch_X_file, 'batch-in', batch_X)

    batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
    batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

    classifier_transformer = estimator.transformer(instance_count=1,
                                        instance_type='ml.m4.xlarge',
                                        strategy='MultiRecord',
                                        assemble_with='Line',
                                        output_path=batch_output)

    classifier_transformer.transform(data=batch_input,
                                 data_type='S3Prefix',
                                 content_type='text/csv',
                                 split_type='Line')

    classifier_transformer.wait()

    s3 = boto3.client('s3')
    obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batc
    target_predicted_df = pd.read_json(io.BytesIO(obj['Body'].read()),orient="re
    return test_data.iloc[:,0], target_predicted_df.iloc[:,0]
```

To run the predictions on the test dataset, run the `batch_linear_predict` function (which was defined previously) on your test dataset.

In [49]:

```
test_labels, target_predicted = batch_linear_predict(test, classifier_estimator)
```

```
INFO:sagemaker.image_uris:Same images used for training and inference. Defaulti
ng to image scope: inference.
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: None.
INFO:sagemaker:Creating model with name: linear-learner-2025-08-17-01-50-49-603
INFO:sagemaker:Creating transform job with name: linear-learner-2025-08-17-01-5
0-50-246
..........................................
...
```

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

In [50]:

```
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(test_labels, target_predicted):
    matrix = confusion_matrix(test_labels, target_predicted)
```

```
    df_confusion = pd.DataFrame(matrix)
    colormap = sns.color_palette("BrBG", 10)
    sns.heatmap(df_confusion, annot=True, fmt='.2f', cbar=None, cmap=colormap)
    plt.title("Confusion Matrix")
    plt.tight_layout()
    plt.ylabel("True Class")
    plt.xlabel("Predicted Class")
    plt.show()
```

In [51]:
```
from sklearn import metrics

def plot_roc(test_labels, target_predicted):
    TN, FP, FN, TP = confusion_matrix(test_labels, target_predicted).ravel()
    # Sensitivity, hit rate, recall, or true positive rate
    Sensitivity  = float(TP)/(TP+FN)*100
    # Specificity or true negative rate
    Specificity  = float(TN)/(TN+FP)*100
    # Precision or positive predictive value
    Precision = float(TP)/(TP+FP)*100
    # Negative predictive value
    NPV = float(TN)/(TN+FN)*100
    # Fall out or false positive rate
    FPR = float(FP)/(FP+TN)*100
    # False negative rate
    FNR = float(FN)/(TP+FN)*100
    # False discovery rate
    FDR = float(FP)/(TP+FP)*100
    # Overall accuracy
    ACC = float(TP+TN)/(TP+FP+FN+TN)*100

    print("Sensitivity or TPR: ", Sensitivity, "%")
    print( "Specificity or TNR: ",Specificity, "%")
    print("Precision: ",Precision, "%")
    print("Negative Predictive Value: ",NPV, "%")
    print( "False Positive Rate: ",FPR,"%")
    print("False Negative Rate: ",FNR, "%")
    print("False Discovery Rate: ",FDR, "%" )
    print("Accuracy: ",ACC, "%")

    test_labels = test.iloc[:,0];
    print("Validation AUC", metrics.roc_auc_score(test_labels, target_predicted)

    fpr, tpr, thresholds = metrics.roc_curve(test_labels, target_predicted)
    roc_auc = metrics.auc(fpr, tpr)

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % (roc_auc))
    plt.plot([0, 1], [0, 1], 'k--')
    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')
    plt.legend(loc="lower right")

    # create the axis of thresholds (scores)
    ax2 = plt.gca().twinx()
    ax2.plot(fpr, thresholds, markeredgecolor='r',linestyle='dashed', color='r')
    ax2.set_ylabel('Threshold',color='r')
```

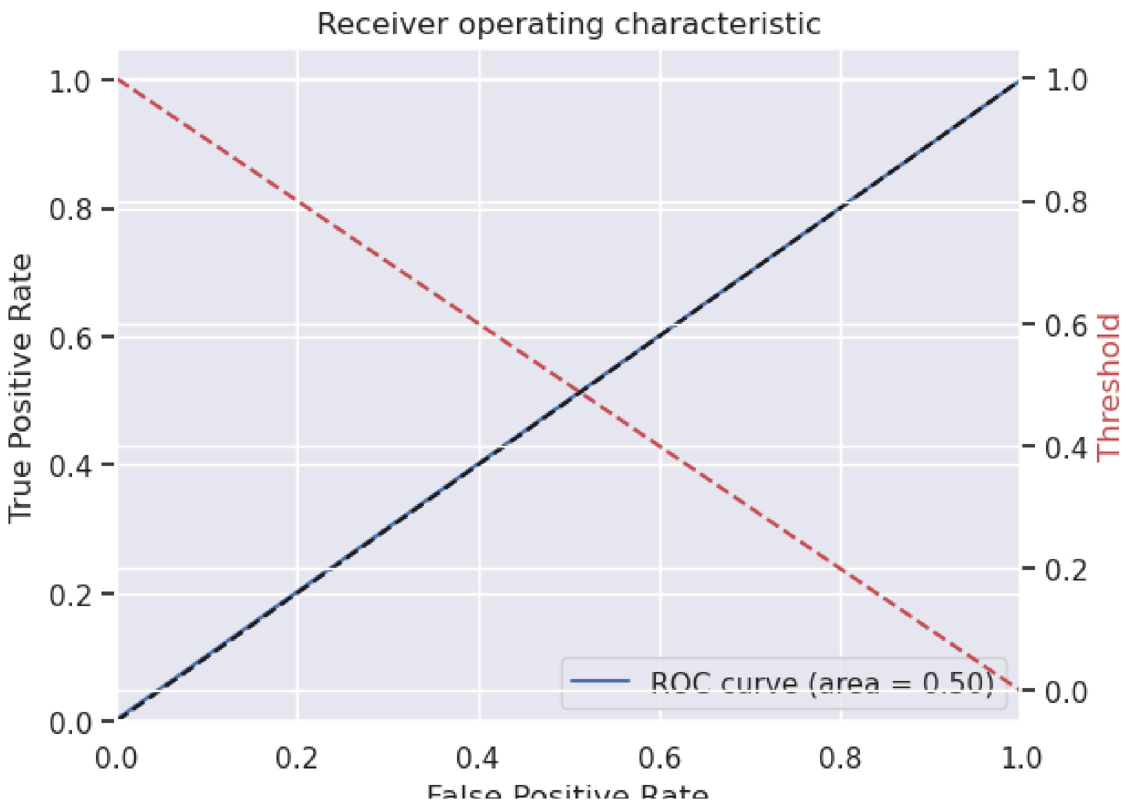
```
    ax2.set_ylim([thresholds[-1],thresholds[0]])
    ax2.set_xlim([fpr[0],fpr[-1]])

    print(plt.figure())
```

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

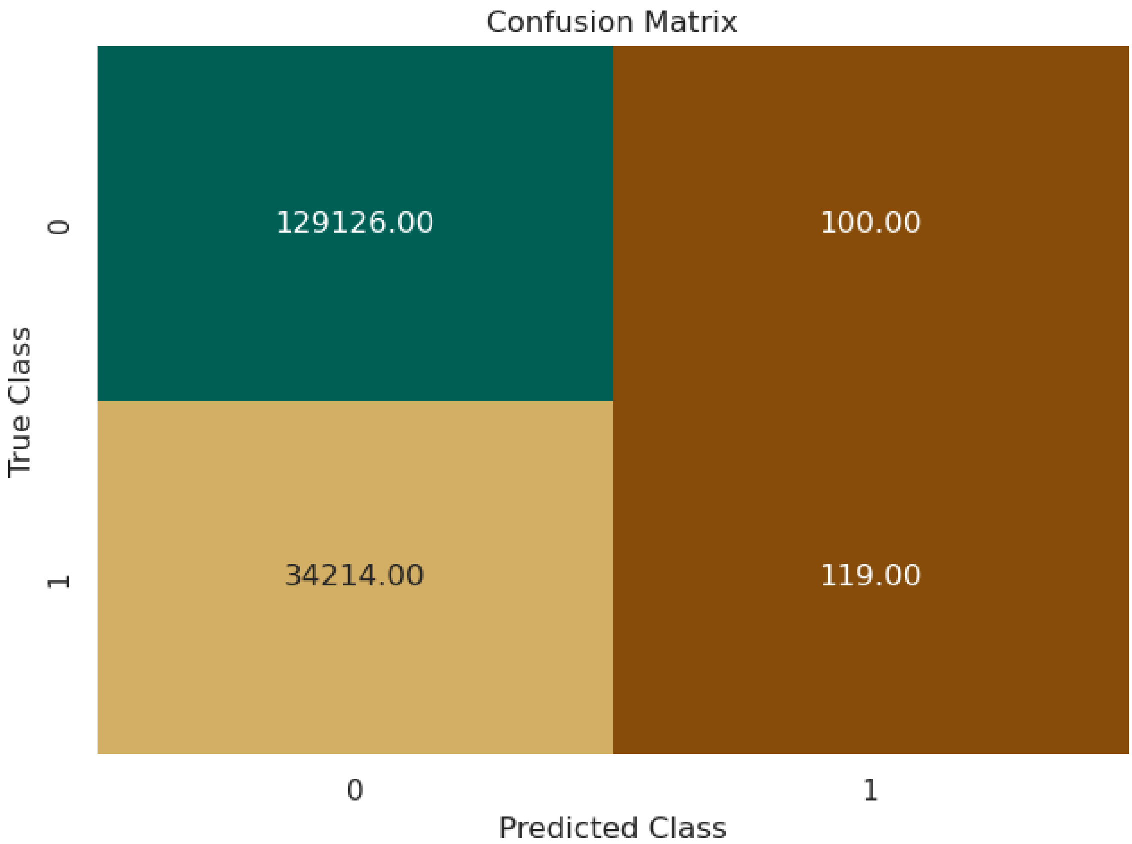
In [52]:
```
plot_roc(test_labels, target_predicted)
```

```
Sensitivity or TPR:  0.3466053068476393 %
Specificity or TNR:  99.92261619178802 %
Precision:  54.337899543378995 %
Negative Predictive Value:  79.05350802008081 %
False Positive Rate:  0.07738380821196973 %
False Negative Rate:  99.65339469315236 %
False Discovery Rate:  45.662100456621005 %
Accuracy:  79.02041465159361 %
Validation AUC 0.5013461074931783
```

```
─────────────────── Traceback (most recent call last) ───────────
in <module>:1

❱ 1 plot_roc(test_labels, target_predicted)
  2

in plot_roc:51

  48 │   ax2 = plt.gca().twinx()
  49 │   ax2.plot(fpr, thresholds, markeredgecolor='r',linestyle='dashed'
  50 │   ax2.set_ylabel('Threshold',color='r')
❱ 51 │   ax2.set_ylim([thresholds[-1],thresholds[0]])
  52 │   ax2.set_xlim([fpr[0],fpr[-1]])
  53 │
  54 │   print(plt.figure())

/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/matplo
in set_ylim

  4049 │   │   │   if top is not None:
  4050 │   │   │   │   raise TypeError("Cannot pass both 'top' and 'ymax'
  4051 │   │   │   top = ymax
❱ 4052 │   │   return self.yaxis._set_lim(bottom, top, emit=emit, auto=au
  4053 │
  4054 │   get_yscale = _axis_method_wrapper("yaxis", "get_scale")
  4055 │   set_yscale = _axis_method_wrapper("yaxis", "_set_axes_scale")

/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/matplo
_set_lim

  1214 │   │
  1215 │   │   self.axes._process_unit_info([(name, (v0, v1))], convert=F
  1216 │   │   v0 = self.axes._validate_converted_limits(v0, self.convert
❱ 1217 │   │   v1 = self.axes._validate_converted_limits(v1, self.convert_
  1218 │   │
  1219 │   │   if v0 is None or v1 is None:
  1220 │   │   │   # Axes init calls set_xlim(0, 1) before get_xlim() can

/home/ec2-user/anaconda3/envs/python3/lib/python3.10/site-packages/matplo
in _validate_converted_limits

  3736 │   │   │   │   converted_limit = converted_limit.squeeze()
  3737 │   │   │   if (isinstance(converted_limit, Real)
  3738 │   │   │   │   │   and not np.isfinite(converted_limit)):
❱ 3739 │   │   │   │   raise ValueError("Axis limits cannot be NaN or Inf
  3740 │   │   │   return converted_limit
  3741 │
  3742 │   def set_xlim(self, left=None, right=None, *, emit=True, auto=F
──────────────────────────────────────────────────────────────────
ValueError: Axis limits cannot be NaN or Inf
```



In [53]:
```
plot_confusion_matrix(test_labels, target_predicted)
```



Key questions to consider:

1. How does your model's performance on the test set compare to its performance on 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. Given your business situation and goals, which metric (or metrics) is the most important for you to consider? Why?
4. From a business standpoint, is the outcome for the metric (or metrics) that you consider to be the most important sufficient for what you need? If not, what are some things you might change in your next iteration? (This will happen in the feature engineering section, which is next.)

Use the following cells to answer these (and other) questions. Insert and delete cells where needed.

Project presentation: In your project presentation, write down your answers to these questions -- and other similar questions that you might answer -- in this section. Record the key details and decisions that you made.

Question: What can you summarize from the confusion matrix?

In [ ]:
```
# Enter your answer here
```

End of Step 3

Save the project file to your local computer. Follow these steps:

1. In the file explorer on the left, right-click the notebook that you're working on.

2. Select **Download**, and save the file locally.

This action downloads the current notebook to the default download folder on your computer.

Iteration II

Step 4: Feature engineering

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

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. At this stage, could you perform any feature-reduction techniques that might have a positive impact on model performance?
4. Can you think of adding some more data or datasets?
5. After performing some feature engineering, how does the performance of your model compare to the first iteration?

Use the following cells to perform specific feature-engineering techniques that you think could improve your model performance (use the previous questions as a guide). Insert and delete cells where needed.

Project presentation: In your project presentation, record your key decisions and the methods that you use in this section. Also include any new performance metrics that you obtain after you evaluate your model again.

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

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 them.

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.

In [54]:
```
# 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
holidays_15 = ['2015-01-01',  '2015-01-19', '2015-02-16', '2015-05-25', '2015-06
holidays_16 = ['2016-01-01',  '2016-01-18', '2016-02-15', '2016-05-30', '2016-07
holidays_17 = ['2017-01-02', '2017-01-16', '2017-02-20', '2017-05-29' , '2017-07
holidays_18 = ['2018-01-01', '2018-01-15', '2018-02-19', '2018-05-28' , '2018-07
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).astype(int)
```

Weather data was fetched from https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summaries&stations=USW00023174,USW00012960,USW00003017,USW00094846,USW0001
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 because of rain, heavy winds, or snow lead to airplane delays? You will now check.

In [55]:
```
!aws s3 cp s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
#!wget 'https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summarie
```

```
download: s3://aws-tc-largeobjects/CUR-TF-200-ACMLFO-1/flight_delay_project/dat
a2/daily-summaries.csv to ../project/data/daily-summaries.csv
```

Import the weather data that was prepared for the airport codes in the dataset. Use the following stations and airports for the analysis. Create a new column called *airport* that maps the weather station to the airport name.

In [56]:
```
weather = pd.read_csv('/home/ec2-user/SageMaker/project/data/daily-summaries.csv
station = ['USW00023174','USW00012960','USW00003017','USW00094846','USW00013874'
airports = ['LAX', 'IAH', 'DEN', 'ORD', 'ATL', 'SFO', 'DFW', 'PHX', 'CLT']

### Map weather stations to airport code
station_map = {s:a for s,a in zip(station, airports)}
weather['airport'] = weather['STATION'].map(station_map)
```

From the **DATE** column, create another column called *MONTH*.

In [57]:
```
weather['MONTH'] = weather['DATE'].apply(lambda x: x.split('-')[1])
weather.head()
```

Out[57]:

	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

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 by using `fillna()`. To check the missing values for all the columns, use the `isna()` function.

In [58]:
```python
weather.SNOW.fillna(0, inplace=True)
weather.SNWD.fillna(0, inplace=True)
weather.isna().sum()
```

Out[58]:
```
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: To find the rows that are missing, use the `isna()` function. Then, to get the index, use the list on the *idx* variable.

In [59]:
```python
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()]

TAVG_idx
```

Out[59]:
```
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])
```

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* values with the average value for a particular station or airport. Because consecutive rows of *TAVG_idx* are missing, replacing them with a previous value would not be possible. Instead, replace them with the mean. Use the `groupby` function to aggregate the variables with a mean value.

Hint: Group by `MONTH` and `STATION`.

In [61]:
```python
# Replace missing TAVG, TMAX, and TMIN with the mean for each MONTH and STATION
weather_impute = weather.groupby(['MONTH', 'STATION']).agg({
    'TAVG': 'mean',
    'TMAX': 'mean',
    'TMIN': 'mean'
}).reset_index()

weather_impute.head(2)
```

Out[61]:

	MONTH	STATION	TAVG	TMAX	TMIN
0	01	USW00003017	-2.741935	74.000000	-69.858065
1	01	USW00003927	79.529032	143.767742	20.696774

Merge the mean data with the weather data.

In [62]:
```python
weather = pd.merge(weather, weather_impute,  how='left', left_on=['MONTH','STATI
.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.

In [63]:
```python
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()
```

Out[63]:
```
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.

In [64]:
```
weather.drop(columns=['STATION','MONTH','TAVG_AVG', 'TMAX_AVG', 'TMIN_AVG', 'TMA
```

Add the origin and destination weather conditions to the dataset.

In [66]:
```
# Ensure both FlightDate and DATE are datetime
data_orig['FlightDate'] = pd.to_datetime(data_orig['FlightDate'])
weather['DATE'] = pd.to_datetime(weather['DATE'])

# 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_C
 .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's always a good practice to check for nulls or NAs after joins.

In [67]:
```
sum(data.isna().any())
```

Out[67]: 0

In [68]:
```
data_orig.columns
```

Out[68]:
```
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 by using one-hot encoding.

In [69]:
```
data = data_orig.copy()
data = data[['is_delay', 'Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
       'Reporting_Airline', 'Origin', 'Dest','Distance','DepHourofDay','is_holid
       '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')
```

In [70]:
```
data_dummies = pd.get_dummies(data[['Year', 'Quarter', 'Month', 'DayofMonth', 'D
data_dummies = data_dummies.replace({True: 1, False: 0})
data = pd.concat([data, data_dummies], axis = 1)
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the new columns.

In [71]:
```
data.shape
```

Out[71]:
```
(1635590, 86)
```

In [72]:
```
data.columns
```

Out[72]:
```
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_1'],
      dtype='object')
```

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 that you used previously.

In [73]:

```
data.rename(columns={'is_delay': 'target'}, inplace=True)
```

Create the training sets again.

Hint: Use the `split_data` function that you defined (and used) earlier.

In [75]:

```
# Create the training, validation, and test sets again

def split_data(data):
    y = data['target'].values.astype('float32')
    X = data.drop('target', axis=1).values.astype('float32')

    # split into train, validation, test
    from sklearn.model_selection import train_test_split
    train_X, temp_X, train_y, temp_y = train_test_split(X, y, test_size=0.3, ran
    val_X, test_X, val_y, test_y = train_test_split(temp_X, temp_y, test_size=0.

    return train_X, val_X, test_X, train_y, val_y, test_y

train_features, val_features, test_features, train_labels, val_labels, test_labe
```

New baseline classifier

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

In [76]:
```
# Number of unique classes (binary classification → 2)
num_classes = len(pd.unique(train_labels))

# Instantiate the LinearLearner estimator object
classifier_estimator2 = sagemaker.LinearLearner(
    role=sagemaker.get_execution_role(),
    instance_count=1,
    instance_type='ml.m4.xlarge',
    predictor_type='binary_classifier',
    binary_classifier_model_selection_criteria='cross_entropy_loss',
    num_classes=num_classes
)
```

Sample code

```
num_classes = len(pd.unique(train_labels))
classifier_estimator2 =
sagemaker.LinearLearner(role=sagemaker.get_execution_role(),
                                               instance_count=1,

instance_type='ml.m4.xlarge',

predictor_type='binary_classifier',

binary_classifier_model_selection_criteria =
'cross_entropy_loss')
```

In [77]:
```
train_records = classifier_estimator2.record_set(train.values[:, 1:].astype(np.f
val_records = classifier_estimator2.record_set(validate.values[:, 1:].astype(np.
test_records = classifier_estimator2.record_set(test.values[:, 1:].astype(np.flo
```

Train your model by using the three datasets that you just created.

In [85]:
```
# Train the model using all three RecordSets
classifier_estimator2.fit([train_records, val_records, test_records])
```

```
INFO:sagemaker.image_uris:Same images used for training and inference. Defaulti
ng to image scope: inference.
INFO:sagemaker.image_uris:Ignoring unnecessary instance type: None.
INFO:sagemaker:Creating training-job with name: linear-learner-2025-08-17-03-08
-42-159
2025-08-17 03:08:43 Starting - Starting the training job...
2025-08-17 03:09:17 Downloading - Downloading input data......
2025-08-17 03:09:53 Downloading - Downloading the training image......
2025-08-17 03:11:09 Training - Training image download completed. Training in p
rogress......................
2025-08-17 03:14:42 Uploading - Uploading generated training model
2025-08-17 03:14:42 Completed - Training job completed
..Training seconds: 325
Billable seconds: 325
```

Plot a confusion matrix.

The linear model shows only a little improvement in performance. Try a tree-based ensemble model, which is called *XGBoost*, with Amazon SageMaker.

Try the XGBoost model

Perform these steps:

1. Use the training set variables and save them as CSV files: train.csv, validation.csv and test.csv.
2. Store the bucket name in the variable. The Amazon S3 bucket name is provided to the left of the lab instructions.

a. `bucket = <LabBucketName>`
b. `prefix = 'flight-xgb'`
3. Use the AWS SDK for Python (Boto3) to upload the model to the bucket.

In [86]:
```
bucket='c169682a4380827l11217960t1w834320805887-labbucket-dj5twjo77qsz'
prefix='flight-xgb'
train_file='flight_train.csv'
test_file='flight_test.csv'
validate_file='flight_validate.csv'
whole_file='flight.csv'
s3_resource = boto3.Session().resource('s3')

def upload_s3_csv(filename, folder, dataframe):
    csv_buffer = io.StringIO()
    dataframe.to_csv(csv_buffer, header=False, index=False )
    s3_resource.Bucket(bucket).Object(os.path.join(prefix, folder, filename)).pu

upload_s3_csv(train_file, 'train', train)
upload_s3_csv(test_file, 'test', test)
upload_s3_csv(validate_file, 'validate', validate)
```

```
INFO:botocore.credentials:Found credentials from IAM Role: BaseNotebookInstance
Ec2InstanceRole
```

Use the `sagemaker.inputs.TrainingInput` function to create a `record_set` for the training and validation datasets.

In [87]:
```
train_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/train/".format(bucket,prefix,train_file),
    content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
    "s3://{}/{}/validate/".format(bucket,prefix,validate_file),
    content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}
```

In [88]:
```
from sagemaker.image_uris import retrieve
container = retrieve('xgboost',boto3.Session().region_name,'1.0-1')
```

```
INFO:sagemaker.image_uris:Defaulting to only available Python version: py3
INFO:sagemaker.image_uris:Defaulting to only supported image scope: cpu.
```

In [89]:
```
sess = sagemaker.Session()
s3_output_location="s3://{}/{}/output/".format(bucket,prefix)

xgb = sagemaker.estimator.Estimator(container,
                                    role = sagemaker.get_execution_role(),
                                    instance_count=1,
                                    instance_type=instance_type,
                                    output_path=s3_output_location,
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        eval_metric = "auc",
                        num_round=100)

xgb.fit(inputs=data_channels)
```

```
INFO:sagemaker.telemetry.telemetry_logging:SageMaker Python SDK will collect te
lemetry to help us better understand our user's needs, diagnose issues, and del
iver additional features.
To opt out of telemetry, please disable via TelemetryOptOut parameter in SDK de
faults config. For more information, refer to https://sagemaker.readthedocs.io/
en/stable/overview.html#configuring-and-using-defaults-with-the-sagemaker-pytho
n-sdk.
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-08-17-03
-22-28-236
2025-08-17 03:22:29 Starting - Starting the training job...
2025-08-17 03:22:44 Starting - Preparing the instances for training...
2025-08-17 03:23:06 Downloading - Downloading input data...
2025-08-17 03:23:36 Downloading - Downloading the training image...
2025-08-17 03:24:27 Training - Training image download completed. Training in p
rogress........................
2025-08-17 03:28:28 Uploading - Uploading generated training model...
2025-08-17 03:28:41 Completed - Training job completed
..Training seconds: 336
Billable seconds: 336
```

Use the batch transformer for your new model, and evaluate the model on the test dataset.

In [90]:
```
batch_X = test.iloc[:,1:];
batch_X_file='batch-in.csv'
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

In [91]:
```
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)

xgb_transformer = xgb.transformer(instance_count=1,
                                       instance_type=instance_type,
                                       strategy='MultiRecord',
                                       assemble_with='Line',
                                       output_path=batch_output)

xgb_transformer.transform(data=batch_input,
```

```
                                 data_type='S3Prefix',
                                 content_type='text/csv',
                                 split_type='Line')
xgb_transformer.wait()
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-17-03-29-49-
312
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-08-17-0
3-29-49-850
....................................
...
```

Get the predicted target and test labels.

In [92]:
```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['ta
test_labels = test.iloc[:,0]
```

Calculate the predicted values based on the defined threshold.

Note: The predicted target will be a score, which must be converted to a binary class.

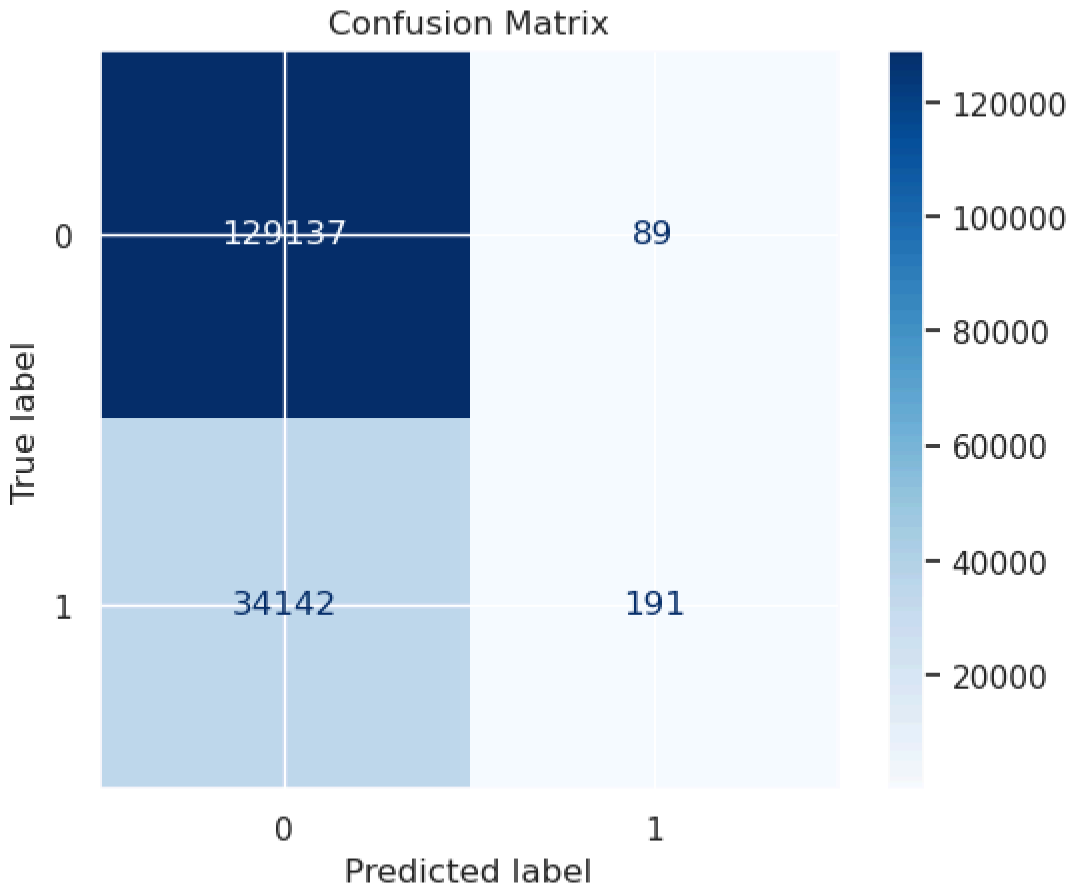
In [93]:
```
print(target_predicted.head())

def binary_convert(x):
    threshold = 0.55
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['target'] = target_predicted['target'].apply(binary_convert)

test_labels = test.iloc[:,0]

print(target_predicted.head())
```

```
     target
0  0.194350
1  0.311986
2  0.249462
3  0.165834
4  0.198785
   target
0       0
1       0
2       0
3       0
4       0
```

Plot a confusion matrix for your `target_predicted` and `test_labels`.

In [94]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Generate the confusion matrix
cm = confusion_matrix(test_labels, target_predicted)
```

```
# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues", values_format="d")

# Add title
plt.title("Confusion Matrix")
plt.show()
```



Try different thresholds

Question: Based on how well the model handled the test set, what can you conclude?

In [ ]:
```
#Enter your answer here
```

Hyperparameter optimization (HPO)

In [95]:
```
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousPa

### You can spin up multiple instances to do hyperparameter optimization in para

xgb = sagemaker.estimator.Estimator(container,
                                    role=sagemaker.get_execution_role(),
                                    instance_count= 1, # make sure you have a li
                                    instance_type=instance_type,
                                    output_path='s3://{}/{}/output'.format(bucke
                                    sagemaker_session=sess)

xgb.set_hyperparameters(eval_metric='auc',
                        objective='binary:logistic',
                        num_round=100,
```

```
                        rate_drop=0.3,
                        tweedie_variance_power=1.4)

hyperparameter_ranges = {'alpha': ContinuousParameter(0, 1000, scaling_type='Lin
                         'eta': ContinuousParameter(0.1, 0.5, scaling_type='Line
                         'min_child_weight': ContinuousParameter(3, 10, scaling_
                         'subsample': ContinuousParameter(0.5, 1),
                         'num_round': IntegerParameter(10,150)}

objective_metric_name = 'validation:auc'

tuner = HyperparameterTuner(xgb,
                            objective_metric_name,
                            hyperparameter_ranges,
                            max_jobs=10, # Set this to 10 or above depending upo
                            max_parallel_jobs=1)
```

In [ ]:
```
tuner.fit(inputs=data_channels)
tuner.wait()
```

```
WARNING:sagemaker.estimator:No finished training job found associated with this
estimator. Please make sure this estimator is only used for building workflow c
onfig
WARNING:sagemaker.estimator:No finished training job found associated with this
estimator. Please make sure this estimator is only used for building workflow c
onfig
INFO:sagemaker:Creating hyperparameter tuning job with name: sagemaker-xgboost-
250817-0343
```

```
.............................................................................
!
```

Wait until the training job is finished. It might take 25-30 minutes.

To monitor hyperparameter optimization jobs:

1. In the AWS Management Console, on the **Services** menu, choose **Amazon SageMaker**.
2. Choose **Training > Hyperparameter tuning jobs**.
3. You can check the status of each hyperparameter tuning job, its objective metric value, and its logs.

Check that the job completed successfully.

In [97]:
```
boto3.client('sagemaker').describe_hyper_parameter_tuning_job(
    HyperParameterTuningJobName=tuner.latest_tuning_job.job_name)['HyperParamete
```

Out[97]: 'Completed'

The hyperparameter tuning job will have a model that worked the best. You can get the information about that model from the tuning job.

In [98]:
```
sage_client = boto3.Session().client('sagemaker')
tuning_job_name = tuner.latest_tuning_job.job_name
print(f'tuning job name:{tuning_job_name}')
tuning_job_result = sage_client.describe_hyper_parameter_tuning_job(HyperParamet
best_training_job = tuning_job_result['BestTrainingJob']
```

```
best_training_job_name = best_training_job['TrainingJobName']
print(f"best training job: {best_training_job_name}")

best_estimator = tuner.best_estimator()

tuner_df = sagemaker.HyperparameterTuningJobAnalytics(tuning_job_name).dataframe
tuner_df.head()
```

```
INFO:botocore.credentials:Found credentials from IAM Role: BaseNotebookInstance
Ec2InstanceRole
tuning job name:sagemaker-xgboost-250817-0343
best training job: sagemaker-xgboost-250817-0343-008-b35996d4

2025-08-17 04:24:48 Starting - Found matching resource for reuse
2025-08-17 04:24:48 Downloading - Downloading the training image
2025-08-17 04:24:48 Training - Training image download completed. Training in p
rogress.
2025-08-17 04:24:48 Uploading - Uploading generated training model
2025-08-17 04:24:48 Completed - Resource reused by training job: sagemaker-xgbo
ost-250817-0343-009-a22a21b0
```

Out[98]:

	alpha	eta	min_child_weight	num_round	subsample	TrainingJobName	Training.
0	0.000000	0.100000	10.000000	18.0	0.929605	sagemaker-xgboost-250817-0343-010-08fea50e	C
1	0.538085	0.427278	9.990059	40.0	0.836188	sagemaker-xgboost-250817-0343-009-a22a21b0	C
2	0.000000	0.263009	10.000000	97.0	0.648683	sagemaker-xgboost-250817-0343-008-b35996d4	C
3	121.659241	0.308247	10.000000	115.0	0.946151	sagemaker-xgboost-250817-0343-007-e4ce614b	C
4	0.000000	0.238870	8.062406	105.0	0.826393	sagemaker-xgboost-250817-0343-006-98650f01	C

Use the estimator `best_estimator` and train it by using the data.

Tip: See the previous XGBoost estimator fit function.

In [ ]:

```
# Enter your code here'
```

Use the batch transformer for your new model, and evaluate the model on the test dataset.

In [99]:

```
batch_output = "s3://{}/{}/batch-out/".format(bucket,prefix)
batch_input = "s3://{}/{}/batch-in/{}".format(bucket,prefix,batch_X_file)
```

```
xgb_transformer = best_estimator.transformer(instance_count=1,
                                             instance_type=instance_type,
                                             strategy='MultiRecord',
                                             assemble_with='Line',
                                             output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                          data_type='S3Prefix',
                          content_type='text/csv',
                          split_type='Line')
xgb_transformer.wait()
```

```
INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-08-17-04-35-56-
058
INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-08-17-0
4-35-56-648
....................................
...
```

In [100…

```
s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="{}/batch-out/{}".format(prefix,'batch-in
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()),sep=',',names=['ta
test_labels = test.iloc[:,0]
```

Get the predicted target and test labels.

In [101…

```
print(target_predicted.head())

def binary_convert(x):
    threshold = 0.55
    if x > threshold:
        return 1
    else:
        return 0

target_predicted['target'] = target_predicted['target'].apply(binary_convert)

test_labels = test.iloc[:,0]

print(target_predicted.head())
```

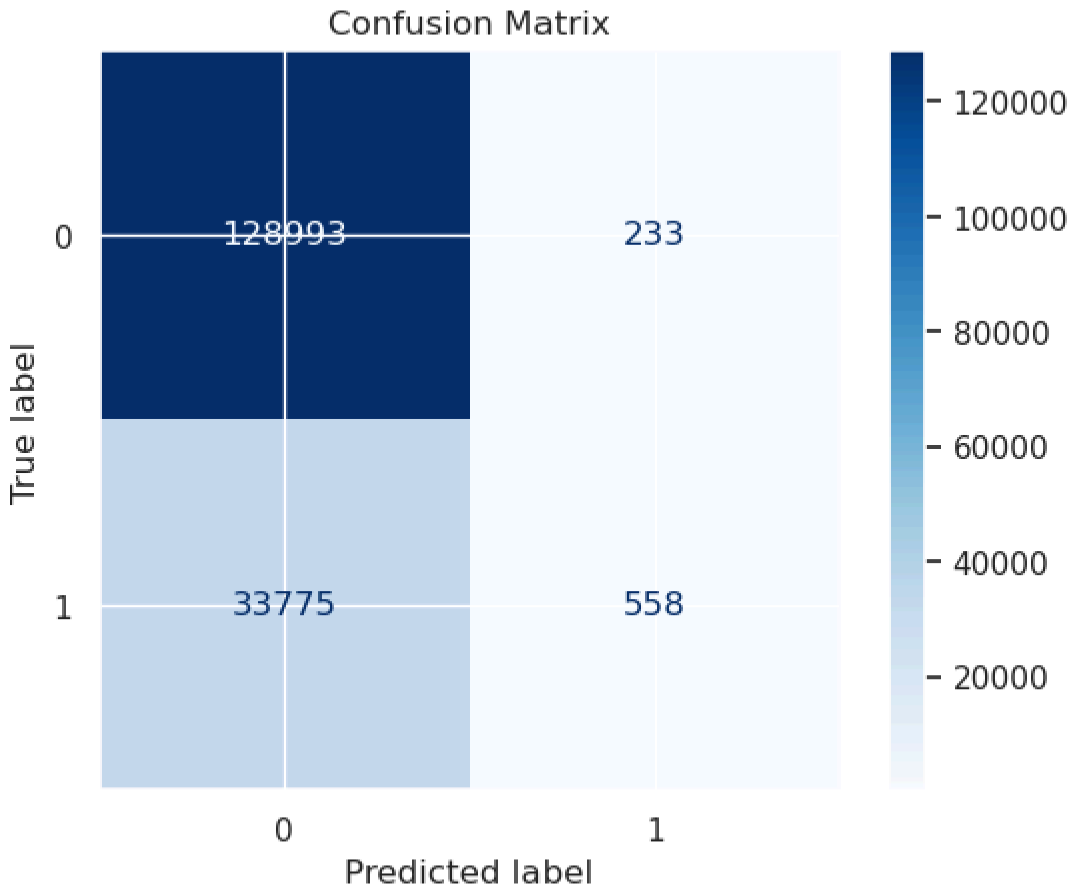
```
     target
0  0.163211
1  0.349700
2  0.280943
3  0.174373
4  0.197460
   target
0       0
1       0
2       0
3       0
4       0
```

Plot a confusion matrix for your `target_predicted` and `test_labels`.

In [102…

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
```

```
cm = confusion_matrix(test_labels, target_predicted)

# Plot confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap="Blues", values_format="d")

plt.title("Confusion Matrix")
plt.show()
```



Question: Try different hyperparameters and hyperparameter ranges. Do these changes improve the model?

Conclusion

You have now iterated through training and evaluating your model at least a couple of times. It's time to wrap up this project and reflect on:

- What you learned
- What types of steps you might take moving forward (assuming that you had more time)

Use the following cell to answer some of these questions 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. How much did your model improve as you made changes to your dataset, features, and hyperparameters? What types of techniques did you employ throughout this

project, and which yielded the greatest improvements in your model?
3. What were some of the biggest challenges that you encountered throughout this project?
4. Do you have any unanswered questions about aspects of the pipeline that didn't make sense to you?
5. What were the three most important things that you learned about machine learning while working on this project?

Project presentation: Make sure that you also summarize your answers to these questions in your project presentation. Combine all your notes for your project presentation and prepare to present your findings to the class.

In [ ]:
```
# Write your answers here
```