Problem: Predicting Airplane Delays

The goals of this notebook are:

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

Introduction to business scenario

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

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

Dataset

The provided dataset contains scheduled and actual departure and arrival times reported by certified US air carriers that account for at least 1 percent of domestic scheduled passenger revenues. The data was collected by the Office of Airline Information, Bureau of Transportation Statistics (BTS). The dataset contains date, time, origin, destination, airline, distance, and delay status of flights for flights between 2014 and 2018. The data are in 60 compressed files, where each file contains a CSV for the flight details in a month for the five years (from 2014 - 2018). The data can be downloaded from this link (https://ucstaff-my.sharepoint.com/:f:/g/personal/ibrahim_radwan_canberra_edu_au/EhWeqeQsh-9Mr1fneZc9_0sBOBzEdXngvxFJtAIla-eAgA?e=8ukWwa). Please download the data files and place them on a relative path. Dataset(s) used in this assignment were compiled by the Office of Airline Information, Bureau of Transportation Statistics (BTS), Airline On-Time Performance Data, available with the following link (https://www.transtats.bts.gov/Fields.asp?gnoyr_VQ=FGJ).

Step 1: Problem formulation and data collection

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

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

Machine learning is suitable for this problem because of a few reasons:

- Complexity of Patterns: flight delays due to weather depend on many interconnected factors like season, airport, airline, or time of day. Patterns that are too complex for traditional rule-based systems.
- Large Dataset: dataset consists of 60 compressed files with detailed information about delays, routes, airlines, and timing. ML algorithms are good at discovering patterns in large historical datasets that humans might miss.

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

- **Business Problem**: Flight delays, especially those caused by weather, create inconvenience for customers and operational challenges for airlines. The company wants to alert customers during booking if their flight is likely to be delayed due to weather.
- **Business Goal + Success metrics**: Build a machine learning model that predicts whether a scheduled flight will be delayed due to weather conditions.
- Accuracy: ≥ 80%
- Recall: ≥ 70%
- Precision: ≥ 75%
- **Desired ML output**: A binary prediction:

1 means Flight delayed due to weather while **0** means Flight not delayed due to weather

This output can be used to inform travelers during the booking process, helping them make better decisions and improving trust in the platform.

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

This is a supervised learning problem, specifically it is Classification.

This is because the dataset includes labeled examples (each flight record shows whether it was delayed or not.

Setup

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

```python
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()

        import warnings
        warnings.filterwarnings('ignore')

        %matplotlib inline

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

```python
In [2]: # pip install pathlib2
```

Step 2: Data preprocessing and visualization

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

Specific questions to consider

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

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

In [3]:
```python
# download the files

# <note: make them all relative, absolute path is not accepted>
zip_path = 'data_compressed'
base_path = '.'
csv_base_path = 'datasets'

!mkdir -p {csv_base_path}
```

In [4]:
```python
# How many zip files do we have? write a code to answer it.

import os

count_files = 0
zip_files=[]

all_files = os.listdir(zip_path)
# print(all_files[:3])

for file in all_files:
    if file.lower().endswith(".zip"):
        count_files = count_files + 1
        zip_files.append(file)
#         print(count_files)

print("Number of zip files:", count_files)
```

Number of zip files: 60

Extract CSV files from ZIP files

```python
In [5]: 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:
            zip_file_path = os.path.join(zip_path, file)
            zip2csv(zip_file_path, csv_base_path)

        print("Files Extracted")
```

```
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2014_1.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2016_3.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2016_2.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2014_2.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2016_1.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2014_3.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2014_7.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2016_5.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
ance_1987_present_2016_4.zip
Extracting data_compressed/On_Time_Reporting_Carrier_On_Time_Perform
```

```python
In [6]: # How many csv files have we extracted? write a code to answer it.

        count_csv = 0
        csv_files=[]

        all_csv_files = os.listdir(csv_base_path)
        # print(all_csv_files[:3])


        for file in all_csv_files:
            if file.lower().endswith(".csv"):
                count_csv = count_csv + 1
                csv_files.append(file)
        #         print(count_csv)

        print("Number of csv files:", count_csv)
```

```
Number of csv files: 60
```

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

In [7]:
```python
from IPython.display import IFrame

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

Out[7]:

BACKGROUND

The data contained in the compressed file has been extracted from the Reporting Carrier C Time" database from the TranStats data library. The time period is indicated in the name o contains data of the first month of the year 2001.

RECORD LAYOUT

Below are fields in the order that they appear on the records:

Year	Year
Quarter	Quarter (1-4)
Month	Month
DayofMonth	Day of Month
DayOfWeek	Day of Week
FlightDate	Flight Date (yyyymmdd)
Reporting_Airline	Unique Carrier Code. When the same code has been users, for example, PA, PA(1), PA(2). Use this field
DOT_ID_Reporting_Airline	An identification number assigned by US DOT to ic defined as one holding and reporting under the same company/corporation.
IATA_CODE_Reporting_Airline	Code assigned by IATA and commonly used to iden different carriers over time, the code is not always u
Tail_Number	Tail Number
Flight_Number_Reporting_Airline	Flight Number
OriginAirportID	Origin Airport, Airport ID. An identification numbe field for airport analysis across a range of years bec can be reused.
OriginAirportSeqID	Origin Airport, Airport Sequence ID. An identificati a given point of time. Airport attributes, such as air

Load sample CSV

Before combining all the CSV files, get a sense of the data from a single CSV file. Using Pandas, read the `On_Time_Reporting_Carrier_On_Time_Performance_(1987_present)_2018_9.c` file first. You can use the Python built-in `read_csv` function ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)).

In [8]:
```python
df_temp = pd.read_csv(f'{csv_base_path}/On_Time_Reporting_Carrier_On_1
```

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

In [9]:
```python
df_shape = df_temp.shape
print(f'Rows and columns in one csv file is {df_shape}')
```

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

Question: Print the first 10 rows of the dataset.

```
In [10]: first_ten_rows = df_temp[:10]

         print(first_ten_rows)
         first_ten_rows
```

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

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

     Div4TailNum  Div5Airport  Div5AirportID  Div5AirportSeqID Div5Whe
elsOn  \
0            NaN          NaN            NaN               NaN
NaN
1            NaN          NaN            NaN               NaN
NaN
2            NaN          NaN            NaN               NaN
NaN
3            NaN          NaN            NaN               NaN
NaN
4            NaN          NaN            NaN               NaN
NaN
5            NaN          NaN            NaN               NaN
NaN
6            NaN          NaN            NaN               NaN
```

```
     NaN
7        NaN           NaN           NaN           NaN
     NaN
8        NaN           NaN           NaN           NaN
     NaN
9        NaN           NaN           NaN           NaN
     NaN

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

[10 rows x 110 columns]
```

Out[10]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	DOT_ID_Repo
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. Use `<dataframe>.columns` to view the column names.

In [11]:
```python
print(f'The column names are :')
print('')
for col in df_temp.columns:
    print(col)
```

The column names are :

Year
Quarter
Month
DayofMonth
DayOfWeek
FlightDate
Reporting_Airline
DOT_ID_Reporting_Airline
IATA_CODE_Reporting_Airline
Tail_Number
Flight_Number_Reporting_Airline
OriginAirportID
OriginAirportSeqID
OriginCityMarketID
Origin
OriginCityName
OriginState
OriginStateFips

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

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

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

Hint: You can use the `in` keyword ([documentation (https://www.w3schools.com/python/ref_keyword_in.asp)](https://www.w3schools.com/python/ref_keyword_in.asp)) to check if the value is in a list or not.

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

In [12]:
```python
del_columns = [x for x in df_temp.columns if 'Del' in x]
print(del_columns)
```

```
['DepDelay', 'DepDelayMinutes', 'DepDel15', 'DepartureDelayGroups',
'ArrDelay', 'ArrDelayMinutes', 'ArrDel15', 'ArrivalDelayGroups', 'Ca
rrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay', 'LateAircr
aftDelay', 'DivArrDelay']
```

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

Questions

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

```
In [13]:   # to answer above questions, complete the following code
           print("The #rows and #columns are ", df_temp.shape[0] , " and ",  df_t
           print("The years in this dataset are: ", df_temp["Year"].unique())
           print("The months covered in this dataset are: ", df_temp["Month"].un:
           print("The date range for data is :" , min(df_temp['FlightDate']), " 1
           print("The airlines covered in this dataset are: ", list(df_temp['IAT/
           print("The Origin airports covered are: ", list(df_temp['Origin'].unic
           print("The Destination airports covered are: ", list(df_temp['Dest'].u
```

```
The #rows and #columns are  585749  and   110
The years in this dataset are:  [2018]
The months covered in this dataset are:   [9]
The date range for data is : 2018-09-01  to  2018-09-30
The airlines covered in this dataset are: ['9E', 'B6', 'WN', 'YV',
'YX', 'EV', 'AA', 'AS', 'DL', 'HA', 'UA', 'F9', 'G4', 'MQ', 'NK', 'O
H', 'OO']
The Origin airports covered are:  ['DFW', 'LGA', 'MSN', 'MSP', 'AT
L', 'BDL', 'VLD', 'JFK', 'RDU', 'CHS', 'DTW', 'GRB', 'PVD', 'SHV',
'FNT', 'PIT', 'RIC', 'RST', 'RSW', 'CVG', 'LIT', 'ORD', 'JAX', 'TR
I', 'BOS', 'CWA', 'DCA', 'CHO', 'AVP', 'IND', 'GRR', 'BTR', 'MEM',
'TUL', 'CLE', 'STL', 'BTV', 'OMA', 'MGM', 'TVC', 'SAV', 'GSP', 'EW
R', 'OAJ', 'BNA', 'MCI', 'TLH', 'ROC', 'LEX', 'PWM', 'BUF', 'AGS',
'CLT', 'GSO', 'BWI', 'SAT', 'PHL', 'TYS', 'ACK', 'DSM', 'GNV', 'AV
L', 'BGR', 'MHT', 'ILM', 'MOT', 'IAH', 'SBN', 'SYR', 'ORF', 'MKE',
'XNA', 'MSY', 'PBI', 'ABE', 'HPN', 'EVV', 'ALB', 'LNK', 'AUS', 'PH
F', 'CHA', 'GTR', 'BMI', 'BQK', 'CID', 'CAK', 'ATW', 'ABY', 'CAE',
'SRQ', 'MLI', 'BHM', 'IAD', 'CSG', 'CMH', 'MCO', 'MBS', 'FLL', 'SD
F', 'TPA', 'MVY', 'LAS', 'LGB', 'SFO', 'SAN', 'LAX', 'RNO', 'PDX',
'ANC', 'ABQ', 'SLC', 'DEN', 'PHX', 'OAK', 'SMF', 'SJU', 'SEA', 'HO
U', 'STX', 'BUR', 'SWF', 'SJC', 'DAB', 'BQN', 'PSE', 'ORH', 'HYA',
'STT', 'ONT', 'HRL', 'ICT', 'ISP', 'LBB', 'MAF', 'MDW', 'OKC', 'PN
S', 'SNA', 'TUS', 'AMA', 'BOI', 'CRP', 'DAL', 'ECP', 'ELP', 'GEG',
'LFT', 'MFE', 'MDT', 'JAN', 'COS', 'MOB', 'VPS', 'MTJ', 'DRO', 'GP
T', 'BFL', 'MRY', 'SBA', 'PSP', 'FSD', 'BRO', 'RAP', 'COU', 'STS',
'PIA', 'FAT', 'SBP', 'FSM', 'HSV', 'BIS', 'DAY', 'BZN', 'MIA', 'EY
W', 'MYR', 'HHH', 'GJT', 'FAR', 'SGF', 'HOB', 'CLL', 'LRD', 'AEX',
'ERI', 'MLU', 'LCH', 'ROA', 'LAW', 'MHK', 'GRK', 'SAF', 'GRI', 'JL
N', 'ROW', 'FWA', 'CRW', 'LAN', 'OGG', 'HNL', 'KOA', 'EGE', 'LIH',
'MLB', 'JAC', 'FAI', 'RDM', 'ADQ', 'BET', 'BRW', 'SCC', 'KTN', 'YA
K', 'CDV', 'JNU', 'SIT', 'PSG', 'WRG', 'OME', 'OTZ', 'ADK', 'FCA',
'FAY', 'PSC', 'BIL', 'MSO', 'ITO', 'PPG', 'MFR', 'EUG', 'GUM', 'SP
N', 'DLH', 'TTN', 'BKG', 'SFB', 'PIE', 'PGD', 'AZA', 'SMX', 'RFD',
'SCK', 'OWB', 'HTS', 'BLV', 'IAG', 'USA', 'GFK', 'BLI', 'ELM', 'PB
G', 'LCK', 'GTF', 'OGD', 'IDA', 'PVU', 'TOL', 'PSM', 'CKB', 'HGR',
'SPI', 'STC', 'ACT', 'TYR', 'ABI', 'AZO', 'CMI', 'BPT', 'GCK', 'MQ
T', 'ALO', 'TXK', 'SPS', 'SWO', 'DBQ', 'SUX', 'SJT', 'GGG', 'LSE',
'LBE', 'ACY', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'IMT', 'WY
S', 'CPR', 'SCE', 'HLN', 'SUN', 'ISN', 'CMX', 'EAU', 'LWB', 'SHD',
'LBF', 'HYS', 'SLN', 'EAR', 'VEL', 'CNY', 'GCC', 'RKS', 'PUB', 'LB
L', 'MKG', 'PAH', 'CGI', 'UIN', 'BFF', 'DVL', 'JMS', 'LAR', 'SGU',
'PRC', 'ASE', 'RDD', 'ACV', 'OTH', 'COD', 'LWS', 'ABR', 'APN', 'ES
C', 'PLN', 'BJI', 'BRD', 'BTM', 'CDC', 'CIU', 'EKO', 'TWF', 'HIB',
'BGM', 'RHI', 'ITH', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
The Destination airports covered are:  ['CVG', 'PWM', 'RDU', 'MSP',
'MSN', 'SHV', 'CLT', 'PIT', 'RIC', 'IAH', 'ATL', 'JFK', 'DCA', 'DT
W', 'LGA', 'TYS', 'PVD', 'FNT', 'LIT', 'BUF', 'ORD', 'TRI', 'IND',
'BGR', 'AVP', 'BWI', 'LEX', 'BDL', 'GRR', 'CWA', 'TUL', 'MEM', 'AG
S', 'EWR', 'MGM', 'PHL', 'SYR', 'OMA', 'STL', 'TVC', 'ORF', 'CLE',
'ABY', 'BOS', 'OAJ', 'TLH', 'BTR', 'SAT', 'JAX', 'BNA', 'CHO', 'VL
D', 'ROC', 'DFW', 'GNV', 'ACK', 'PBI', 'CHS', 'GRB', 'MOT', 'MKE',
'DSM', 'ILM', 'GSO', 'MCI', 'SBN', 'BTV', 'MVY', 'XNA', 'RST', 'EV
V', 'HPN', 'RSW', 'MDT', 'ROA', 'GSP', 'MCO', 'CSG', 'SAV', 'PHF',
'ALB', 'CHA', 'ABE', 'BMI', 'MSY', 'IAD', 'GTR', 'CID', 'CAK', 'AT
W', 'AUS', 'BQK', 'MLI', 'CAE', 'CMH', 'AVL', 'MBS', 'FLL', 'SDF',
'TPA', 'LNK', 'SRQ', 'MHT', 'BHM', 'LAS', 'SFO', 'SAN', 'RNO', 'LG
B', 'ANC', 'PDX', 'SJU', 'ABQ', 'SLC', 'DEN', 'LAX', 'PHX', 'OAK',
'SMF', 'SEA', 'STX', 'BUR', 'DAB', 'SJC', 'SWF', 'HOU', 'BQN', 'PS
E', 'ORH', 'HYA', 'STT', 'ONT', 'DAL', 'ECP', 'ELP', 'HRL', 'MAF',
'MDW', 'OKC', 'PNS', 'SNA', 'AMA', 'BOI', 'GEG', 'ICT', 'LBB', 'TU
S', 'ISP', 'CRP', 'MFE', 'LFT', 'VPS', 'JAN', 'COS', 'MOB', 'DRO',
```

```
'GPT', 'BFL', 'COU', 'SBP', 'MTJ', 'SBA', 'PSP', 'FSD', 'FSM', 'BR
O', 'PIA', 'STS', 'FAT', 'RAP', 'MRY', 'HSV', 'BIS', 'DAY', 'BZN',
'MIA', 'EYW', 'MYR', 'HHH', 'GJT', 'FAR', 'MLU', 'LRD', 'CLL', 'LC
H', 'FWA', 'GRK', 'SGF', 'HOB', 'LAW', 'MHK', 'SAF', 'JLN', 'ROW',
'GRI', 'AEX', 'CRW', 'LAN', 'ERI', 'HNL', 'KOA', 'OGG', 'EGE', 'LI
H', 'JAC', 'MLB', 'RDM', 'BET', 'ADQ', 'BRW', 'SCC', 'FAI', 'JNU',
'CDV', 'YAK', 'SIT', 'KTN', 'WRG', 'PSG', 'OME', 'OTZ', 'ADK', 'FC
A', 'BIL', 'PSC', 'FAY', 'MSO', 'ITO', 'PPG', 'MFR', 'DLH', 'EUG',
'GUM', 'SPN', 'TTN', 'BKG', 'AZA', 'SFB', 'LCK', 'BLI', 'SCK', 'PI
E', 'RFD', 'PVU', 'PBG', 'BLV', 'PGD', 'SPI', 'USA', 'TOL', 'IDA',
'ELM', 'HTS', 'HGR', 'SMX', 'OGD', 'GFK', 'STC', 'GTF', 'IAG', 'CK
B', 'OWB', 'PSM', 'ABI', 'TYR', 'ALO', 'SUX', 'AZO', 'ACT', 'CMI',
'BPT', 'TXK', 'SWO', 'SPS', 'DBQ', 'SJT', 'GGG', 'LSE', 'MQT', 'GC
K', 'LBE', 'ACY', 'LYH', 'PGV', 'HVN', 'EWN', 'DHN', 'PIH', 'WYS',
'SCE', 'IMT', 'HLN', 'ASE', 'SUN', 'ISN', 'EAR', 'SGU', 'VEL', 'SH
D', 'LWB', 'MKG', 'SLN', 'HYS', 'BFF', 'PUB', 'LBL', 'CMX', 'EAU',
'PAH', 'UIN', 'RKS', 'CGI', 'CNY', 'JMS', 'DVL', 'LAR', 'GCC', 'LB
F', 'PRC', 'RDD', 'ACV', 'OTH', 'COD', 'LWS', 'ABR', 'APN', 'PLN',
'BJI', 'CPR', 'BRD', 'BTM', 'CDC', 'CIU', 'ESC', 'EKO', 'ITH', 'HI
B', 'BGM', 'TWF', 'RHI', 'INL', 'FLG', 'YUM', 'MEI', 'PIB', 'HDN']
```

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

Hint: You can use the Pandas `values_count` function (documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.value_counts.html)) to find out the values for each airport using the columns `Origin` and `Dest`.

```
In [14]:  counts = pd.DataFrame({
              'Origin': df_temp['Origin'].value_counts(),
              'Destination':df_temp['Dest'].value_counts()})
          counts
```

Out[14]:

	Origin	Destination
ABE	303	303
ABI	169	169
ABQ	2077	2076
ABR	60	60
ABY	79	79
...	...	...
WRG	60	60
WYS	52	52
XNA	1004	1004
YAK	60	60
YUM	96	96

346 rows × 2 columns

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

Hint: You can use the Pandas `sort_values` function ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html)).

In [15]:
```
counts.sort_values(by='Origin', ascending=False).head(15)
```

Out[15]:

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

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

Yes, I think flight delays can be predicted using the information in the dataset. Features like the airline, origin/destination airports, day of week, month, and scheduled departure time could capture patterns like certain airlines having more delays, specific airports being prone to congestion, seasonal weather impacts (winter storms, summer thunderstorms), or even peak travel times.

Variable like ArrDel15 could potentially be used as the target variable.

In [ ]:

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

Columns such as `DepDelay`, `ArrDelay`, `CarrierDelay`, `WeatherDelay`, `NASDelay`, `SecurityDelay`, `LateAircraftDelay`, and `DivArrDelay` contain information about a delay. But this delay could have occured at the origin or destination. If

there were a sudden weather delay 10 minutes before landing, this data would not be helpful in managing your Los Angeles reservations.

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

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

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

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

Combine all CSV files

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

1. Read the CSV file into a dataframe
2. Filter the columns based on the `filter_cols` variable

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

3. Keep only the subset_vals in each of the subset_cols. Use the `isin` Pandas function ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isin.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isin.html)) to check if the `val` is in the dataframe column and then choose the rows that include it.

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

4. Concatenate the dataframe with the empty dataframe

```python
In [16]: def combine_csv(csv_files, filter_cols, subset_cols, subset_vals, file
             """
             Combine csv files into one Data Frame
             csv_files: list of csv file paths
             filter_cols: list of columns to filter
             subset_cols: list of columns to subset rows
             subset_vals: list of list of values to subset rows
             """
             # Create an empty dataframe
             df = pd.DataFrame()

             for file in csv_files:

                 file_path = f"datasets/{file}"

                 df_temp = pd.read_csv(file_path)

                 # Filter the columns (keep only the columns we want)
                 df_temp = df_temp[filter_cols]

                 # Filter rows - keep only rows where Origin is in first list
                 df_temp = df_temp[df_temp['Origin'].isin(subset_vals[0])]

                 # Filter rows - keep only rows where Dest is in second list
                 df_temp = df_temp[df_temp['Dest'].isin(subset_vals[1])]

                 # Filter rows - keep only rows where Reporting_Airline is in t
                 df_temp = df_temp[df_temp['Reporting_Airline'].isin(subset_val

                 # Add this filtered data to our main dataframe
                 df = pd.concat([df, df_temp])

             # Save the combined data to a CSV file
             df.to_csv(file_name, index=False)


             #<complete the code of this function>
```

```python
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 a
         subset_vals = [['ATL', 'ORD', 'DFW', 'DEN', 'CLT', 'LAX', 'IAH', 'PHX
                         ['ATL', 'ORD', 'DFW', 'DEN', 'CLT', 'LAX', 'IAH', 'PHX
                         ['UA', 'OO', 'WN', 'AA', 'DL']]
```

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

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

In [18]:
```python
start = time.time()

combined_csv_filename = f"{base_path}/combined_files.csv"

df_combined = combine_csv(csv_files, cols, subset_cols, subset_vals, 

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

```
csv's merged in 2.07 minutes
```

Load dataset

Load the combined dataset.

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

Print the first 5 records.

In [20]:
```
print(data)

data
```

```
         Year  Quarter  Month  DayofMonth  DayOfWeek  FlightDate  \
0        2014        4     10           1          3  2014-10-01
1        2014        4     10           2          4  2014-10-02
2        2014        4     10           3          5  2014-10-03
3        2014        4     10           4          6  2014-10-04
4        2014        4     10           5          7  2014-10-05
...       ...      ...    ...         ...        ...         ...
1658125  2017        3      9           5          2  2017-09-05
1658126  2017        3      9           5          2  2017-09-05
1658127  2017        3      9           5          2  2017-09-05
1658128  2017        3      9           5          2  2017-09-05
1658129  2017        3      9           5          2  2017-09-05
```

```
         Reporting_Airline Origin OriginState Dest DestState  CRSDepT
ime  \
0                       AA    DFW          TX  SFO        CA
755
1                       AA    DFW          TX  SFO        CA
755
2                       AA    DFW          TX  SFO        CA
755
3                       AA    DFW          TX  SFO        CA
755
4                       AA    DFW          TX  SFO        CA
755
...                    ...    ...         ...  ...       ...
...
1658125                 UA    ORD          IL  SFO        CA
950
1658126                 UA    IAH          TX  PHX        AZ
909
1658127                 UA    IAH          TX  LAX        CA
910
1658128                 UA    LAX          CA  ORD        IL        1
347
1658129                 UA    ATL          GA  SFO        CA
716
```

```
         Cancelled  Diverted  Distance  DistanceGroup  ArrDelay  \
0              0.0       0.0    1464.0              6      -9.0
1              0.0       0.0    1464.0              6      40.0
2              0.0       0.0    1464.0              6       9.0
3              0.0       0.0    1464.0              6     -16.0
4              0.0       0.0    1464.0              6      -8.0
...            ...       ...       ...            ...       ...
1658125        0.0       0.0    1846.0              8     -36.0
1658126        1.0       0.0    1009.0              5       NaN
1658127        1.0       0.0    1379.0              6       NaN
1658128        0.0       0.0    1744.0              7      -9.0
1658129        0.0       0.0    2139.0              9     -22.0
```

```
         ArrDelayMinutes  ArrDel15  AirTime
0                    0.0       0.0    195.0
1                   40.0       1.0    199.0
2                    9.0       0.0    196.0
3                    0.0       0.0    195.0
4                    0.0       0.0    192.0
...                  ...       ...      ...
1658125              0.0       0.0    228.0
1658126              NaN       NaN      NaN
1658127              NaN       NaN      NaN
```

```
1658128               0.0        0.0      221.0
1658129               0.0        0.0      263.0

[1658130 rows x 20 columns]
```

Out[20]:

	Year	Quarter	Month	DayofMonth	DayOfWeek	FlightDate	Reporting_Airline	Origin
0	2014	4	10	1	3	2014-10-01	AA	DFW
1	2014	4	10	2	4	2014-10-02	AA	DFW
2	2014	4	10	3	5	2014-10-03	AA	DFW
3	2014	4	10	4	6	2014-10-04	AA	DFW
4	2014	4	10	5	7	2014-10-05	AA	DFW
...	...	...	...	...	...	...	...	...
1658125	2017	3	9	5	2	2017-09-05	UA	ORD
1658126	2017	3	9	5	2	2017-09-05	UA	IAH
1658127	2017	3	9	5	2	2017-09-05	UA	IAH
1658128	2017	3	9	5	2	2017-09-05	UA	LAX
1658129	2017	3	9	5	2	2017-09-05	UA	ATL

1658130 rows × 20 columns

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

Questions

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

In [21]:
```python
# to answer above questions, complete the following code
print("The #rows and #columns are ", data.shape[0] , " and ", data.sha
print("The years in this dataset are: ", list(data["Year"].unique()))
print("The months covered in this dataset are: ", sorted(list(data["Mo
print("The date range for data is :" , min(data['FlightDate']), " to '
print("The airlines covered in this dataset are: ", list(df_temp['IATA
print("The Origin airports covered are: ", list(data['Origin'].unique(
print("The Destination airports covered are: ", list(data['Dest'].uni
```

```
The #rows and #columns are  1658130  and  20
The years in this dataset are:  [2014, 2017, 2018, 2015, 2016]
The months covered in this dataset are:  [1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12]
The date range for data is : 2014-01-01  to  2018-12-31
The airlines covered in this dataset are:  ['9E', 'B6', 'WN', 'YV',
'YX', 'EV', 'AA', 'AS', 'DL', 'HA', 'UA', 'F9', 'G4', 'MQ', 'NK', 'O
H', 'OO']
The Origin airports covered are:  ['DFW', 'SFO', 'DEN', 'PHX', 'IA
H', 'ATL', 'ORD', 'LAX', 'CLT']
The Destination airports covered are:  ['SFO', 'DFW', 'DEN', 'PHX',
'ORD', 'IAH', 'ATL', 'LAX', 'CLT']
```

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

Hint: You can use the Pandas `rename` function ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html)).

For example:

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

In [22]:
```python
# data.columns

data.rename(columns={'ArrDel15': 'is_delay'}, inplace=True)

data.columns
```

Out[22]:
```
Index(['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'Fligh
tDate',
       'Reporting_Airline', 'Origin', 'OriginState', 'Dest', 'DestSt
ate',
       'CRSDepTime', 'Cancelled', 'Diverted', 'Distance', 'DistanceG
roup',
       'ArrDelay', 'ArrDelayMinutes', 'is_delay', 'AirTime'],
      dtype='object')
```

Look for nulls across columns. You can use the `isnull()` function ([documentation (https://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.isnull.html)](https://pandas.pydata.org/pandas-docs/version/0.17.0/generated/pandas.isnull.html)).

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

In [23]: `data.isnull().sum(axis=0)`

Out[23]:
```
Year                     0
Quarter                  0
Month                    0
DayofMonth               0
DayOfWeek                0
FlightDate               0
Reporting_Airline        0
Origin                   0
OriginState              0
Dest                     0
DestState                0
CRSDepTime               0
Cancelled                0
Diverted                 0
Distance                 0
DistanceGroup            0
ArrDelay             22540
ArrDelayMinutes      22540
is_delay             22540
AirTime              22540
dtype: int64
```

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

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

For example:

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

In [24]:
```python
### Remove null columns
data = data[~data['is_delay'].isnull()] # I just remove is_delay becau

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]:
```python
# data['DepHourofDay'] = # Enter your code here

data['DepHourofDay'] = (data['CRSDepTime'] // 100).astype(int)
```

The ML problem statement

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

Data exploration

Check class delay vs. no delay

Hint: Use a `groupby` plot ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.groupby.html)) with a `bar` plot ([documentation (https://matplotlib.org/tutorials/introductory/pyplot.html)](https://matplotlib.org/tutorials/introductory/pyplot.html)) to plot the frequency vs. distribution of the class.

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

Distribution of classes

[Bar chart titled "Distribution of classes" with x-axis "is_delay" showing two bars: class 0.0 at approximately 0.79 frequency and class 1.0 at approximately 0.21 frequency. Y-axis labeled "Frequency" ranging from 0.0 to 0.8.]
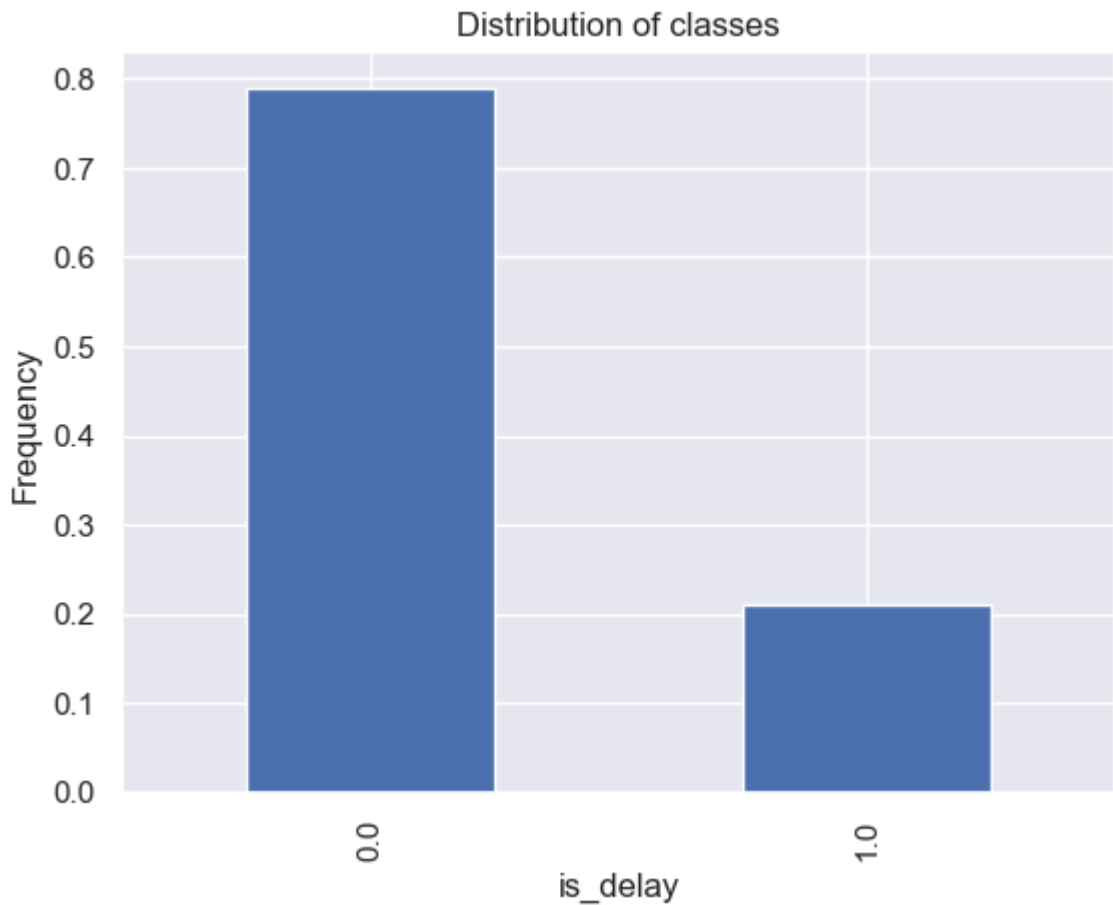
Question: What can you deduce from the bar plot about the ratio of delay vs. no delay?

From that bar plot, I can deduce that:

The dataset shows a significant class imbalance between delay and no delay cases. No delay cases have around 80% of the observations while delay cases have only approximately 20%.

So, the majority class (no delay) dominates the dataset, and the minority class (delay) is underrepresented. This imbalance may cause machine learning models to be biased toward predicting no delays.

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 [27]:
```python
viz_columns = ['Month', 'DepHourofDay', 'DayOfWeek', 'Reporting_Airlin
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=Tru
    mul(100).reset_index().sort_values(column)
    sns.barplot(x=column, y="percentage", hue="is_delay", data=temp, a
    plt.ylabel('% delay/no-delay')


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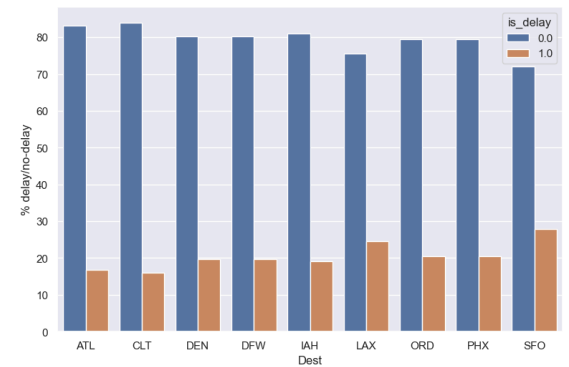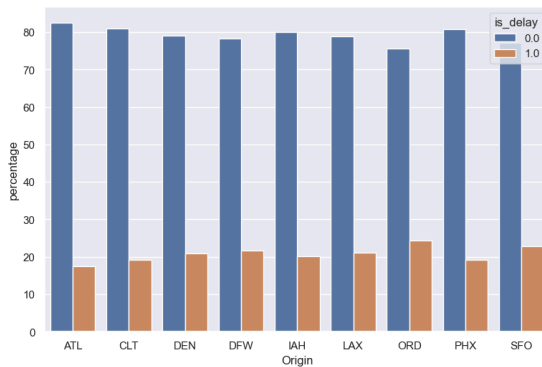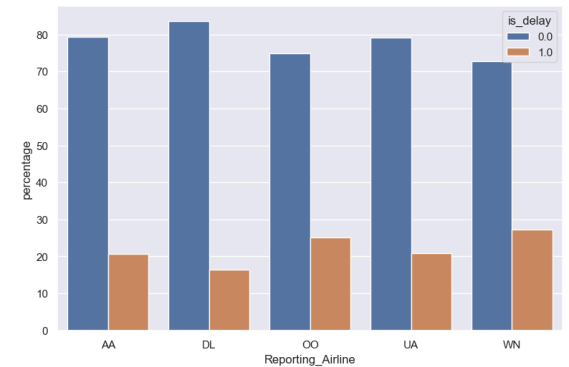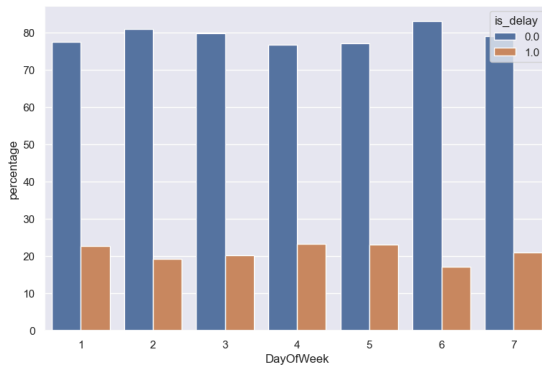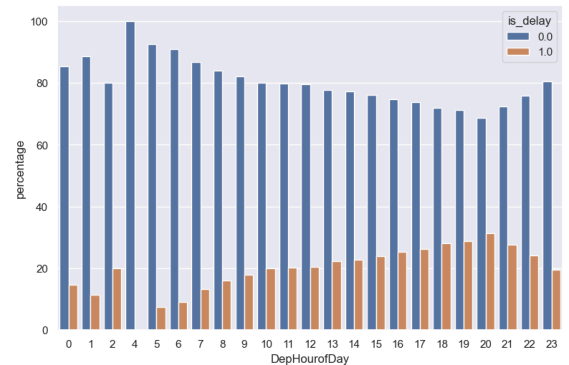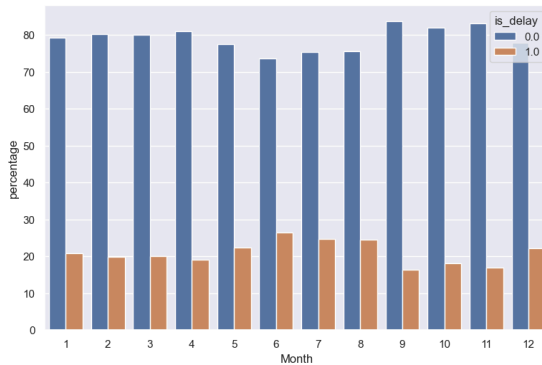
In [28]:
```python
sns.lmplot( x="is_delay", y="Distance", data=data, fit_reg=False, hue=
plt.legend(loc='center')
plt.xlabel('is_delay')
plt.ylabel('Distance')
plt.show()
```
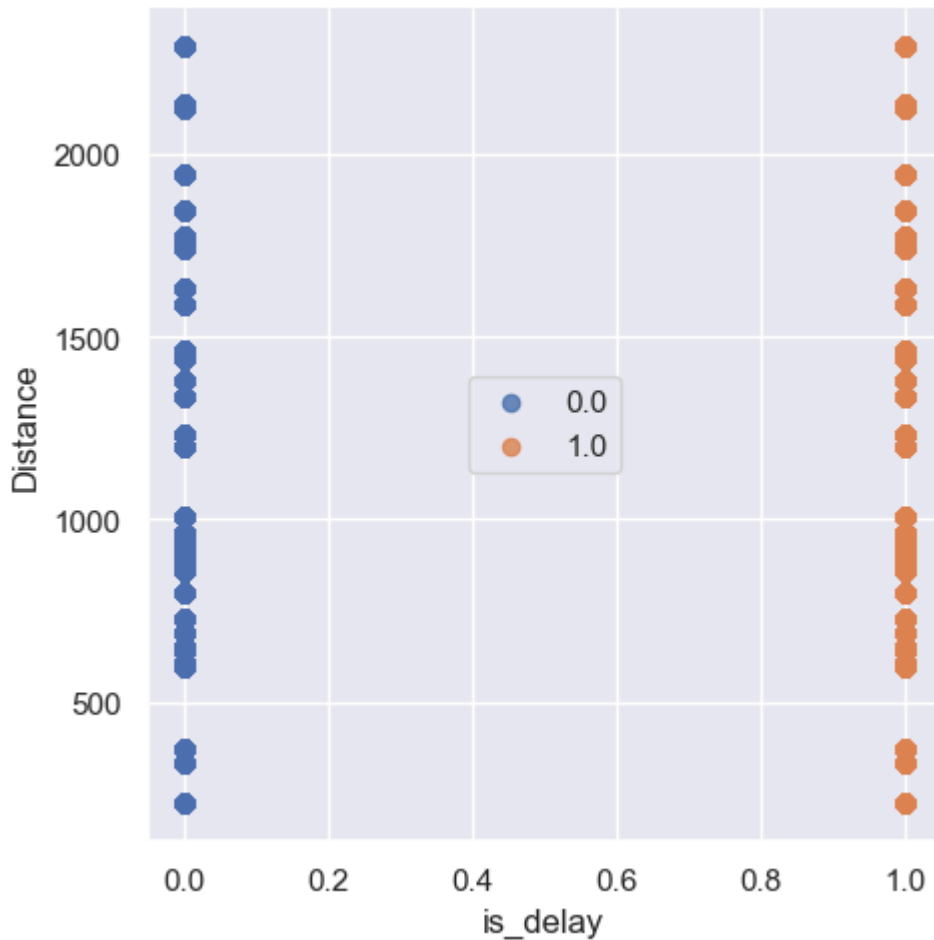

Which months have the most delays?

- June has the highest delays, after that, July, August and December.

What time of the day has the most delays?

- 8pm has the most delays, but delays starts to increase from 6pm-9pm.

What day of the week has the most delays?

- It seems like Monday(1), Thursday(4), and Friday(5) have the most delays.

Which airline has the most delays?

- WN and then OO have the most delays.

Which origin and destination airports have the most delays?

- origin: ORN and destination: SFO

Is flight distance a factor in the delays

- Flight distance is NOT a determining factor in whether a flight will be delayed.

- Both delayed flights and non-delayed flights show nearly identical distance

Features

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

In [29]: `data.columns`

Out[29]:
```
Index(['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'Fligh
tDate',
       'Reporting_Airline', 'Origin', 'OriginState', 'Dest', 'DestSt
ate',
       'CRSDepTime', 'Cancelled', 'Diverted', 'Distance', 'DistanceG
roup',
       'ArrDelay', 'ArrDelayMinutes', 'is_delay', 'AirTime', 'DepHou
rofDay'],
      dtype='object')
```

In [30]: `data.dtypes`

Out[30]:
```
Year                 int64
Quarter              int64
Month                int64
DayofMonth           int64
DayOfWeek            int64
FlightDate          object
Reporting_Airline   object
Origin              object
OriginState         object
Dest                object
DestState           object
CRSDepTime           int64
Cancelled          float64
Diverted           float64
Distance           float64
DistanceGroup        int64
ArrDelay           float64
ArrDelayMinutes    float64
is_delay           float64
AirTime            float64
DepHourofDay         int64
dtype: object
```

Filtering the required columns:

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

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

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

- Just split into buckets here.

Hint: To change a column type to category, use the `astype` function ([documentation](https://pandas.pydata.org/pandas-)
(https://pandas.pydata.org/pandas-

- Just split into buckets here.

Hint: To change a column type to category, use the `astype` function ([documentation](https://pandas.pydata.org/pandas-)
(https://pandas.pydata.org/pandas-

```python
In [31]: data_orig = data.copy()
         data = data[[ 'is_delay', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek
                      'Reporting_Airline', 'Origin', 'Dest','Distance','DepHourofDay

         # I did the BUCKETIZE DepHourofDay before treating as categorical
         def bucket_hour(hour):
             if 0 <= hour < 6:
                 return 'Night'
             elif 6 <= hour < 12:
                 return 'Morning'
             elif 12 <= hour < 18:
                 return 'Afternoon'
             else:  # 18 <= hour < 24
                 return 'Evening'

         # Create the bucketed column and drop the original
         data['DepHourBucket'] = data['DepHourofDay'].apply(bucket_hour)
         data.drop('DepHourofDay', axis=1, inplace=True)

         # categorical_columns  = ['Quarter', 'Month', 'DayofMonth', 'DayOfWeek
         #          'Reporting_Airline', 'Origin', 'Dest', 'DepHourofDay']

         # Update categorical_columns to use DepHourBucket instead of DepHouro
         categorical_columns = ['Quarter', 'Month', 'DayofMonth', 'DayOfWeek',
                                'Reporting_Airline', 'Origin', 'Dest', 'DepHour


         for c in categorical_columns:
             data[c] = data[c].astype('category')


         # Verify the changes
         print(data.dtypes)
         print(f"\nDataset shape: {data.shape}")
         print(f"\nFirst few rows:\n{data.head()}")
```

```
is_delay              float64
Quarter              category
Month                category
DayofMonth           category
DayOfWeek            category
Reporting_Airline    category
Origin               category
Dest                 category
Distance              float64
DepHourBucket        category
dtype: object

Dataset shape: (1635590, 10)

First few rows:
    is_delay Quarter Month DayofMonth DayOfWeek Reporting_Airline Ori
gin Dest  \
0        0.0       4    10          1         3                AA
DFW  SFO
1        1.0       4    10          2         4                AA
DFW  SFO
2        0.0       4    10          3         5                AA
DFW  SFO
3        0.0       4    10          4         6                AA
DFW  SFO
4        0.0       4    10          5         7                AA
DFW  SFO


    Distance DepHourBucket
0     1464.0       Morning
1     1464.0       Morning
2     1464.0       Morning
3     1464.0       Morning
4     1464.0       Morning
```

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

For example:

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

In [32]:
```python
data_dummies = pd.get_dummies(data[categorical_columns], drop_first=T
data = pd.concat([data, data_dummies], axis = 1)
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the length of the dataset and the new columnms.

In [33]:
```python
# Verify the changes
print("New dataset shape:", data.shape)
```

```
New dataset shape: (1635590, 75)
```

In [34]:
```python
print("\nData types:")
print(data.dtypes)

print("\nColumn names:")
print(data.columns)
```

```
Data types:
is_delay                   float64
Distance                   float64
Quarter_2                     bool
Quarter_3                     bool
Quarter_4                     bool
                           ...
Dest_PHX                      bool
Dest_SFO                      bool
DepHourBucket_Evening         bool
DepHourBucket_Morning         bool
DepHourBucket_Night           bool
Length: 75, dtype: object

Column names:
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', 'DayofMont
h_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMont
h_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMont
h_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMont
h_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMont
h_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_
3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Ai
rline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_D
FW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Orig
in_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_LAX',
'Dest_ORD',
       'Dest_PHX', 'Dest_SFO', 'DepHourBucket_Evening',
       'DepHourBucket_Morning', 'DepHourBucket_Night'],
      dtype='object')
```

Sample Answer:

```
Index(['Distance', 'is_delay', 'Quarter_2', 'Quarter_3', 'Qua
rter_4',
       'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Month_6',
'Month_7',
       'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Month_1
2',
       'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'Dayof
Month_5',
       'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'Dayof
Month_9',
       'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'Da
yofMonth_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'Da
yofMonth_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'Da
yofMonth_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'Da
yofMonth_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'Da
yofMonth_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayO
fWeek_3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWee
k_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Repor
ting_Airline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'O
rigin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PH
X', 'Origin_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_
LAX', 'Dest_ORD',
       'Dest_PHX', 'Dest_SFO'],
      dtype='object')
```

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

Hint: You can use the Pandas `rename` function ([documentation (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html)).

```
In [35]: data.rename(columns={'is_delay': 'target'}, inplace=True)
```

In [36]:
```python
# write code to Save the combined csv file (combined_csv_v1.csv) to yo
# note this combined file will be used in part B

# Save the processed data to CSV
data.to_csv('combined_csv_v1.csv', index=False)

print("File saved successfully as 'combined_csv_v1.csv'")
print(f"Shape of saved data: {data.shape}")
print(f"Columns in saved file: {len(data.columns)}")
```

```
File saved successfully as 'combined_csv_v1.csv'
Shape of saved data: (1635590, 75)
Columns in saved file: 75
```

Step 3: Model training and evaluation

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

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

Train test split

In [37]:
```python
# write Code here to split data into train, validate and test

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, con

# Separate features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Separate test set (20%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y # maintain class balance
)

# Then I split again for: train and validation
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp,
    test_size=0.2,
    random_state=42,
    stratify=y_temp
)
```

Baseline classification model

```python
In [38]:  # base model logicstic regression

log_reg = LogisticRegression(
    max_iter=1000,
    random_state=42,
    class_weight='balanced',
    solver='lbfgs'
)

log_reg.fit(X_train, y_train)
```

```
Out[38]:
                          LogisticRegression                         ⓘ ?
                                                                     (ht
                                                                     lea

LogisticRegression(class_weight='balanced', max_iter=1000, random_s
tate=42)
```

Model evaluation

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

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

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

```python
In [39]:  from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(test_labels, target_predicted):

    cm = confusion_matrix(test_labels, target_predicted)

    # Plot
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()
```

In [40]:
```python
from sklearn.metrics import roc_curve, auc, accuracy_score, precision_

def plot_roc(test_labels, target_predicted):
    # Get predicted probabilities for positive class
    target_predicted_proba = log_reg.predict_proba(X_test)[:, 1]

    # Calculate ROC curve
    fpr, tpr, _ = roc_curve(test_labels, target_predicted_proba)
    roc_auc = auc(fpr, tpr)

    # Plot ROC
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.4f})')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()

    # Calculate and print statistics
    cm = confusion_matrix(test_labels, target_predicted)
    tn, fp, fn, tp = cm.ravel()

    accuracy = accuracy_score(test_labels, target_predicted)
    precision = precision_score(test_labels, target_predicted)
    recall = recall_score(test_labels, target_predicted)
    sensitivity = recall
    specificity = tn / (tn + fp)


    print("STATISTICS")
    print("\n")
    print(f"Accuracy:    {accuracy:.4f}")
    print(f"Precision:   {precision:.4f}")
    print(f"Recall:      {recall:.4f}")
    print(f"Sensitivity: {sensitivity:.4f}")
    print(f"Specificity: {specificity:.4f}")
    print(f"ROC-AUC:     {roc_auc:.4f}")
```

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

In [41]:
```python
y_test_pred = log_reg.predict(X_test)

plot_confusion_matrix(y_test, y_test_pred)
```


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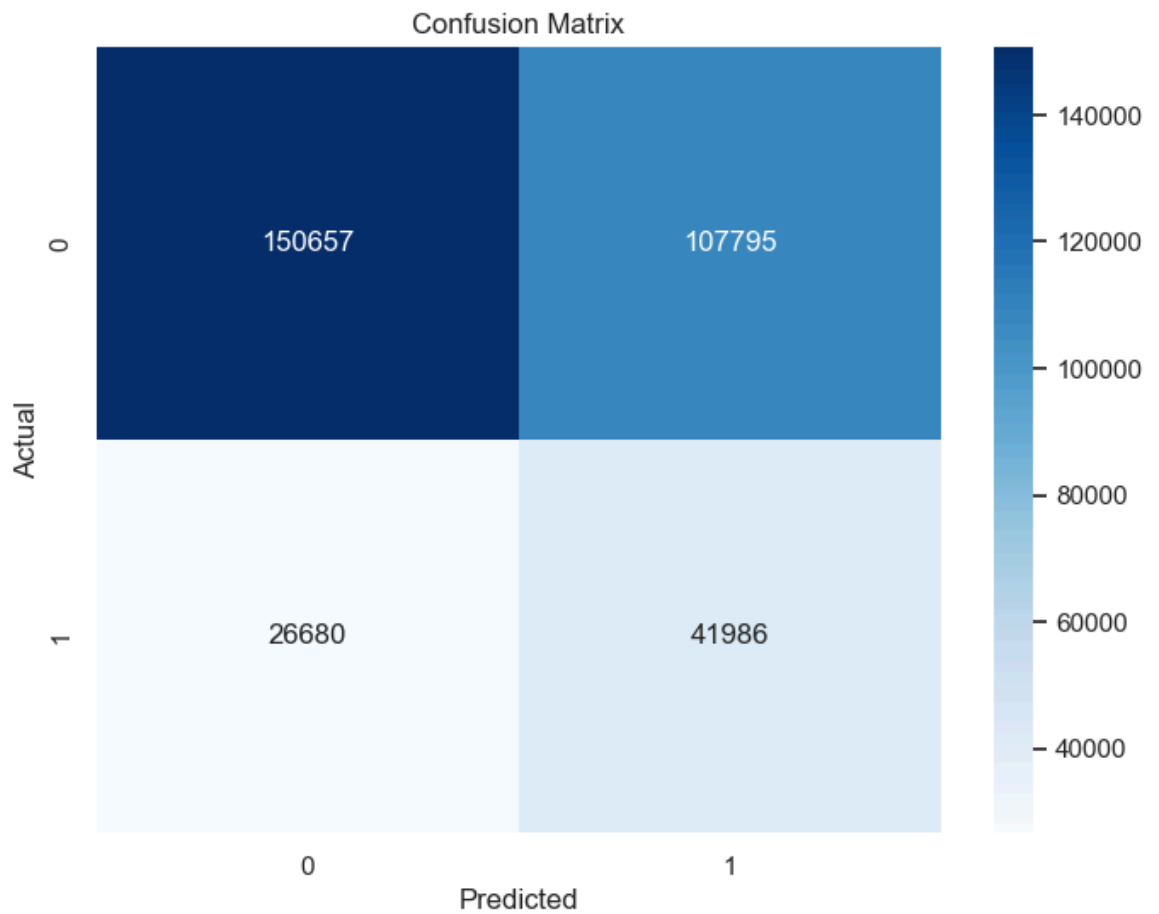
To print statistics and plot an ROC curve, call the `plot_roc` function on the `test_labels` and `target_predicted` data from your batch job:

In [42]: `plot_roc(y_test, y_test_pred)`

ROC Curve
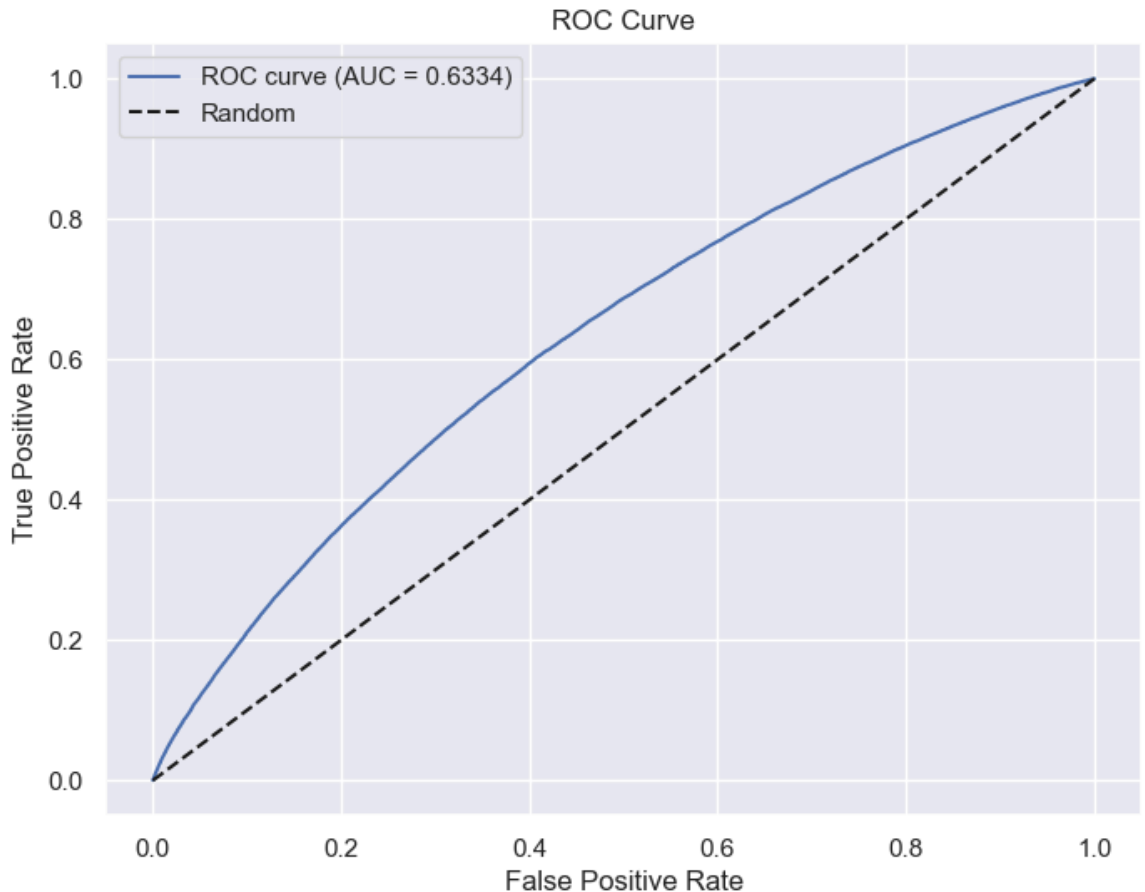
```
STATISTICS


Accuracy:    0.5889
Precision:   0.2803
Recall:      0.6115
Sensitivity: 0.6115
Specificity: 0.5829
ROC-AUC:     0.6334
```

Key questions to consider:

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

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

In [43]:
```python
# Make predictions on all sets
y_train_pred = log_reg.predict(X_train)
y_val_pred = log_reg.predict(X_val)
y_test_pred = log_reg.predict(X_test)

# Get probabilities for ROC-AUC
y_train_proba = log_reg.predict_proba(X_train)[:, 1]
y_val_proba = log_reg.predict_proba(X_val)[:, 1]
y_test_proba = log_reg.predict_proba(X_test)[:, 1]
```

In [44]:
```python
from sklearn.metrics import f1_score

train_acc = accuracy_score(y_train, y_train_pred)
train_prec = precision_score(y_train, y_train_pred)
train_rec = recall_score(y_train, y_train_pred)
train_f1 = f1_score(y_train, y_train_pred)
train_auc = roc_auc_score(y_train, y_train_proba)

val_acc = accuracy_score(y_val, y_val_pred)
val_prec = precision_score(y_val, y_val_pred)
val_rec = recall_score(y_val, y_val_pred)
val_f1 = f1_score(y_val, y_val_pred)
val_auc = roc_auc_score(y_val, y_val_proba)

test_acc = accuracy_score(y_test, y_test_pred)
test_prec = precision_score(y_test, y_test_pred)
test_rec = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_auc = roc_auc_score(y_test, y_test_proba)

# Display comparison
print("\nMetric           Training      Validation      Test")
print("-" * 55)
print(f"Accuracy         {train_acc:.4f}        {val_acc:.4f}          {tes
print(f"Precision        {train_prec:.4f}         {val_prec:.4f}         {t
print(f"Recall           {train_rec:.4f}         {val_rec:.4f}          {tes
print(f"F1-Score         {train_f1:.4f}         {val_f1:.4f}          {test_
print(f"ROC-AUC          {train_auc:.4f}         {val_auc:.4f}          {tes
```

```
Metric          Training    Validation    Test
--------------------------------------------------------
Accuracy        0.5903      0.5888        0.5889
Precision       0.2809      0.2792        0.2803
Recall          0.6100      0.6064        0.6115
F1-Score        0.3846      0.3824        0.3844
ROC-AUC         0.6341      0.6321        0.6334
```

Q1: How does your model's performance on the test set compare to the training set? What can you deduce from this comparison?

The model demonstrates identical performance across training (59.03%), validation (58.89%), and test (58.89%) sets, indicating no overfitting.

However, overall performance remains moderate to weak. Its precision is low (about 28%) and recall is moderate (around 61%), meaning it often predicts delays incorrectly and still misses about 39% of real ones. These results show the model struggles with class imbalance.

While this gives a useful baseline, it's not good enough. Better features and more advanced models are needed to improve performance.

Q2: Are there obvious differences between the outcomes of metrics like accuracy, precision, and recall? If so, why might you be seeing those differences?

Yes, there are clear differences between the outcomes of accuracy, precision, and recall.

While the accuracy is around 59%, which might seem acceptable, it doesn't reflect how well the model handles the minority class (delayed flights). The precision is much lower (about 28%), showing that many of the flights predicted as delayed are actually on time — the model produces many false positives. In contrast, the recall is higher (around 61%), meaning the model successfully identifies more than half of the real delays but still misses about 39% of them.

These differences occur because the dataset is likely imbalanced, with far more on-time flights than delayed ones. Accuracy remains moderately high because the model performs well on the majority class, but precision drops due to incorrect delay predictions. Recall is higher because the model leans toward predicting more delays to catch true ones, at the cost of precision.

In summary, the gap between these metrics highlights that accuracy alone is misleading, and class imbalance causes the model to struggle in correctly identifying true delay cases.

Q3: Is the outcome for the metric(s) you consider most important sufficient for what you need from a business standpoint? If not, what are some things you might change in your next iteration (in the feature engineering section, which is coming up next)?

From a business standpoint, the current results are not sufficient. In a real-world setting like flight delay prediction, precision and recall are the most important metrics because false alarms (low precision) can waste resources, while missed delays (lower recall) can reduce customer satisfaction and operational efficiency.

To improve performance in the next iteration, I would:

- Address class imbalance using techniques like class weighting.
- Engineer more predictive features, such as holidays.
- Try advanced models like Random Forest which can handle nonlinearity and imbalance better than logistic regression.
- Tune hyperparameters and experiment with threshold adjustment to balance precision and recall.

Question: What can you summarize from the confusion matrix?

For no delays, the model correctly predicts around 150,657 flights, but it incorrectly labels about 26,680 delayed flights as on-time.

For delays, it misclassifies 107,795 on-time flights as delayed, while correctly identifying 41,986 true delays.

Overall, the confusion matrix shows that the model favors predicting the majority class (on-time flights), resulting in many false positives when predicting delays. This aligns with the earlier metrics, low precision (around 28%) and moderate recall (about 61%), indicating the

model struggles with class imbalance and has limited reliability for real-world deployment

Step 4: Deployment

1. In this step you are required to push your source code and requirements file to a GitHub repository without the data files. Please use the Git commands to complete this task
2. Create a "readme.md" markdown file that describes the code of this repository and how to run it and what the user would expect if got the code running.

In the cell below provide the link of the pushed repository on your GitHub account, and ensure it is public.

https://github.com/manilesrun/final_project_u3273786 (https://github.com/manilesrun/final_project_u3273786)

Iteration II

Step 5: Feature engineering

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

Key questions to consider:

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

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

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

Add more features

1. Holidays
2. Weather

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

```
In [45]: # data_orig
         # data_orig.dtypes
```

```
In [46]: # Source: http://www.calendarpedia.com/holidays/federal-holidays-2014.

         holidays_14 = ['2014-01-01',  '2014-01-20', '2014-02-17', '2014-05-26
         holidays_15 = ['2015-01-01',  '2015-01-19', '2015-02-16', '2015-05-25
         holidays_16 = ['2016-01-01',  '2016-01-18', '2016-02-15', '2016-05-30
         holidays_17 = ['2017-01-02', '2017-01-16', '2017-02-20', '2017-05-29'
         holidays_18 = ['2018-01-01', '2018-01-15', '2018-02-19', '2018-05-28'
         holidays = holidays_14+ holidays_15+ holidays_16 + holidays_17+ holida

         # Convert FlightDate column to datetime if not already
         data_orig['FlightDate'] = pd.to_datetime(data_orig['FlightDate'])

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


         # data_orig

         print(f"Number of holiday flights: {data_orig['is_holiday'].sum():,}"}
```

```
Number of holiday flights: 43,912
```

Weather data was fetched from [this link](https://www.ncei.noaa.gov/access/services/data/v1?dataset=daily-summaries&stations=USW00023174,USW00012960,USW00003017,USW00094846,USW000701-01&endDate=2018-12-31).

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

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

```
In [47]: # download data from the link above and place it into the data folder

         weather_data = pd.read_csv("daily-summaries-2025-11-02T01-54-31.csv")

         # Display basic info about the dataset
         print(f"Weather data shape: {weather_data.shape}")
         print(f"\nFirst few rows:")
         print(weather_data.head(10))

         print(f"\nColumn names:")
         print(weather_data.columns.tolist())

         print("\n\nData types:")
         print(weather_data.dtypes)

         print("\n\nBasic statistics:")
         print(weather_data.describe())

         # Check for missing values
         print("\n\nMissing values:")
         print(weather_data.isnull().sum())
```

```
                 AWND           PRCP           SNOW           SNWD         TAVG  \
count   16434.000000   16434.000000   10956.000000   10956.000000   1637
2.000000
mean       37.656505      22.898625       1.122855       3.487222     17
3.172429
std        16.193890      91.440626      10.612414      23.419737      9
1.088532
min         4.000000       0.000000       0.000000       0.000000    -22
2.000000
25%        26.000000       0.000000       0.000000       0.000000     12
3.000000
50%        35.000000       0.000000       0.000000       0.000000     17
8.000000
75%        47.000000       0.000000       0.000000       0.000000     23
8.000000
max       123.000000    4082.000000     411.000000     460.000000     40
7.000000

                 TMAX           TMIN
count   16414.000000   16414.000000
```

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

In [48]:
```python
weather_data = pd.read_csv("daily-summaries-2025-11-02T01-54-31.csv")
station = ['USW00023174','USW00012960','USW00003017','USW00094846',
           'USW00013874','USW00023234','USW00003927','USW00023183','US
airports = ['LAX', 'IAH', 'DEN', 'ORD', 'ATL', 'SFO', 'DFW', 'PHX', 'C

# map weather stations to airport code
station_map = {}
for i in range(len(station)):
    station_map[station[i]] = airports[i]

# station_map
# create a new column airport using the map
weather_data['airport'] = weather_data['STATION'].map(station_map)

weather_data
```

Out[48]:

	STATION	DATE	AWND	PRCP	SNOW	SNWD	TAVG	TMAX	TMIN	airport
0	USW00023174	2014-01-01	16	0	NaN	NaN	131.0	178.0	78.0	LAX
1	USW00023174	2014-01-02	22	0	NaN	NaN	159.0	256.0	100.0	LAX
2	USW00023174	2014-01-03	17	0	NaN	NaN	140.0	178.0	83.0	LAX
3	USW00023174	2014-01-04	18	0	NaN	NaN	136.0	183.0	100.0	LAX
4	USW00023174	2014-01-05	18	0	NaN	NaN	151.0	244.0	83.0	LAX
...	...	...	...	...	...	...	...	...	...	...
16429	USW00013881	2018-12-27	31	41	0.0	0.0	68.0	89.0	39.0	CLT
16430	USW00013881	2018-12-28	27	196	0.0	0.0	86.0	144.0	61.0	CLT
16431	USW00013881	2018-12-29	14	0	0.0	0.0	146.0	189.0	94.0	CLT
16432	USW00013881	2018-12-30	16	23	0.0	0.0	117.0	139.0	89.0	CLT
16433	USW00013881	2018-12-31	29	41	0.0	0.0	135.0	194.0	117.0	CLT

16434 rows × 10 columns

Create another column called MONTH from the DATE column.

In [49]:
```
weather_data['MONTH'] = weather_data['DATE'].apply(lambda x: x.split(
weather_data.head()
```

Out[49]:

	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 a
irport MONTH
0 USW00023174 2014-01-01 16   0   NaN  NaN 131.0 178.0 78.0
LAX    01
1 USW00023174 2014-01-02 22   0   NaN  NaN 159.0 256.0 100.0
LAX    01
2 USW00023174 2014-01-03 17   0   NaN  NaN 140.0 178.0 83.0
LAX    01
3 USW00023174 2014-01-04 18   0   NaN  NaN 136.0 183.0 100.0
LAX    01
4 USW00023174 2014-01-05 18   0   NaN  NaN 151.0 244.0 83.0
LAX    01
```

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

```
In [50]: print(weather_data.isna().sum())

         weather_data.SNOW.fillna(0, inplace=True)
         weather_data.SNWD.fillna(0, inplace=True)
         weather_data.isna().sum()

         print("\nMissing values AFTER filling:")
         print(weather_data.isna().sum())
```

```
STATION        0
DATE           0
AWND           0
PRCP           0
SNOW        5478
SNWD        5478
TAVG          62
TMAX          20
TMIN          20
airport        0
MONTH          0
dtype: int64

Missing values AFTER filling:
STATION        0
DATE           0
AWND           0
PRCP           0
SNOW           0
SNWD           0
TAVG          62
TMAX          20
TMIN          20
airport        0
MONTH          0
dtype: int64
```

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

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

In [51]:
```python
idx = np.array([i for i in range(len(weather_data))])

# Find indices where TAVG is missing
TAVG_idx = idx[weather_data['TAVG'].isna()]

# Find indices where TMAX is missing
TMAX_idx = idx[weather_data['TMAX'].isna()]

# Find indices where TMIN is missing
TMIN_idx = idx[weather_data['TMIN'].isna()]

# Display the results
print("missing TAVG indices:")
print(TAVG_idx)

print("\n missing TMAX indices:")
print(TMAX_idx)

print("\n missing TMIN indices:")
print(TMIN_idx)
```

```
missing TAVG indices:
[ 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]

 missing TMAX indices:
[10763 10764 10765 10766 10767 10768 10769 10770 10771 10772 10773 1
0774
 10775 10776 10777 10778 10779 10780 10781 10782]

 missing TMIN indices:
[10763 10764 10765 10766 10767 10768 10769 10770 10771 10772 10773 1
0774
 10775 10776 10777 10778 10779 10780 10781 10782]
```

Sample output

```
array([ 3956,  3957,  3958,  3959,  3960,  3961,  3962,  396
3,  3964,
        3965,  3966,  3967,  3968,  3969,  3970,  3971,  397
2,  3973,
        3974,  3975,  3976,  3977,  3978,  3979,  3980,  398
1,  3982,
        3983,  3984,  3985,  4017,  4018,  4019,  4020,  402
1,  4022,
        4023,  4024,  4025,  4026,  4027,  4028,  4029,  403
0,  4031,
        4032,  4033,  4034,  4035,  4036,  4037,  4038,  403
9,  4040,
        4041,  4042,  4043,  4044,  4045,  4046,  4047,  1342
0])
```

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

```
In [52]: weather_impute = weather_data.groupby(['STATION']).agg({'TAVG':'mean',
         weather_impute.head(2)
```

Out[52]:

	STATION	TAVG	TMAX	TMIN
0	USW00003017	112.931445	190.020263	32.978642
1	USW00003927	198.463308	256.026287	143.678532

Merge the mean data with the weather data.

```
In [53]: ### get the yesterday's data
         weather = pd.merge(weather_data, weather_impute,  how='left', left_on=
         .rename(columns = {'TAVG_y':'TAVG_AVG',
                            'TMAX_y':'TMAX_AVG',
                            'TMIN_y':'TMIN_AVG',
                            'TAVG_x':'TAVG',
                            'TMAX_x':'TMAX',
                            'TMIN_x':'TMIN'})


         # weather
```

Check for missing values again.

In [54]:
```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[54]:
```
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 [55]:
```python
weather.drop(columns=['STATION','MONTH','TAVG_AVG', 'TMAX_AVG', 'TMIN_
```

Add the origin and destination weather conditions to the dataset.

In [56]:
```python
# First, ensure both date columns are datetime type
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=['Flight
.rename(columns = {'AWND':'AWND_O','PRCP':'PRCP_O', 'TAVG':'TAVG_O',
.drop(columns=['DATE','airport'])

### Add destination weather conditions
data_orig = pd.merge(data_orig, weather,  how='left', left_on=['Flight
.rename(columns = {'AWND':'AWND_D','PRCP':'PRCP_D', 'TAVG':'TAVG_D',
.drop(columns=['DATE','airport'])
```

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

In [57]:
```python
sum(data.isna().any())
```

Out[57]: 0

```
In [58]: data_orig.columns
```

```
Out[58]: Index(['Year', 'Quarter', 'Month', 'DayofMonth', 'DayOfWeek', 'Fligh
         tDate',
                'Reporting_Airline', 'Origin', 'OriginState', 'Dest', 'DestSt
         ate',
                'CRSDepTime', 'Cancelled', 'Diverted', 'Distance', 'DistanceG
         roup',
                'ArrDelay', 'ArrDelayMinutes', 'is_delay', 'AirTime', 'DepHou
         rofDay',
                'is_holiday', 'AWND_O', 'PRCP_O', 'SNOW_O', 'TAVG_O', 'AWND_
         D',
                'PRCP_D', 'SNOW_D', 'TAVG_D'],
               dtype='object')
```

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

```
In [59]: data = data_orig.copy()
         data = data[['is_delay', 'Year', 'Quarter', 'Month', 'DayofMonth', 'Da
                'Reporting_Airline', 'Origin', 'Dest','Distance','DepHourofDay
                'TAVG_O', 'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D']]

         categorical_columns = ['Year', 'Quarter', 'Month', 'DayofMonth', 'Day
                'Reporting_Airline', 'Origin', 'Dest', 'is_holiday']

         for c in categorical_columns:
             data[c] = data[c].astype('category')
```

```
In [60]: data_dummies = pd.get_dummies(data[categorical_columns], drop_first=Tr
         data = pd.concat([data, data_dummies], axis=1)
         data.drop(categorical_columns,axis=1, inplace=True)
```

Sample code

```
data_dummies = pd.get_dummies(data[['Year', 'Quarter', 'Mont
h', 'DayofMonth', 'DayOfWeek', 'Reporting_Airline', 'Origin',
'Dest', 'is_holiday']], drop_first=True)
data = pd.concat([data, data_dummies], axis = 1)
categorical_columns.remove('is_delay')
data.drop(categorical_columns,axis=1, inplace=True)
```

Check the new columns.

In [61]: `data.columns`

Out[61]: Index(['is_delay', 'Distance', 'DepHourofDay', 'AWND_O', 'PRCP_O',
       'TAVG_O',
            'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D', 'Year_201
       5',
            'Year_2016', 'Year_2017', 'Year_2018', 'Quarter_2', 'Quarter_
       3',
            'Quarter_4', 'Month_2', 'Month_3', 'Month_4', 'Month_5', 'Mon
       th_6',
            'Month_7', 'Month_8', 'Month_9', 'Month_10', 'Month_11', 'Mon
       th_12',
            'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'DayofMonth_
       5',
            'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'DayofMonth_
       9',
            'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'DayofMont
       h_13',
            'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'DayofMont
       h_17',
            'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'DayofMont
       h_21',
            'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'DayofMont
       h_25',
            'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'DayofMont
       h_29',
            'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayOfWeek_
       3',
            'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWeek_7',
            'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Reporting_Ai
       rline_UA',
            'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'Origin_D
       FW',
            'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PHX', 'Orig
       in_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', 'PRC
P_O', 'TAVG_O',
       'AWND_D', 'PRCP_D', 'TAVG_D', 'SNOW_O', 'SNOW_D', 'Yea
r_2015',
       'Year_2016', 'Year_2017', 'Year_2018', 'Quarter_2', 'Q
uarter_3',
       'Quarter_4', 'Month_2', 'Month_3', 'Month_4', 'Month_
5', 'Month_6',
       'Month_7', 'Month_8', 'Month_9', 'Month_10', 'Month_1
1', 'Month_12',
       'DayofMonth_2', 'DayofMonth_3', 'DayofMonth_4', 'Dayof
Month_5',
       'DayofMonth_6', 'DayofMonth_7', 'DayofMonth_8', 'Dayof
Month_9',
       'DayofMonth_10', 'DayofMonth_11', 'DayofMonth_12', 'Da
yofMonth_13',
       'DayofMonth_14', 'DayofMonth_15', 'DayofMonth_16', 'Da
yofMonth_17',
       'DayofMonth_18', 'DayofMonth_19', 'DayofMonth_20', 'Da
yofMonth_21',
       'DayofMonth_22', 'DayofMonth_23', 'DayofMonth_24', 'Da
yofMonth_25',
       'DayofMonth_26', 'DayofMonth_27', 'DayofMonth_28', 'Da
yofMonth_29',
       'DayofMonth_30', 'DayofMonth_31', 'DayOfWeek_2', 'DayO
fWeek_3',
       'DayOfWeek_4', 'DayOfWeek_5', 'DayOfWeek_6', 'DayOfWee
k_7',
       'Reporting_Airline_DL', 'Reporting_Airline_OO', 'Repor
ting_Airline_UA',
       'Reporting_Airline_WN', 'Origin_CLT', 'Origin_DEN', 'O
rigin_DFW',
       'Origin_IAH', 'Origin_LAX', 'Origin_ORD', 'Origin_PH
X', 'Origin_SFO',
       'Dest_CLT', 'Dest_DEN', 'Dest_DFW', 'Dest_IAH', 'Dest_
LAX', 'Dest_ORD',
       'Dest_PHX', 'Dest_SFO', 'is_holiday_1'],
      dtype='object')
```

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

In [62]: `data.rename(columns = {'is_delay':'target'}, inplace=True )# Enter you`

In [63]:
```python
# write code to Save the new combined csv file (combined_csv_v2.csv) ʈ
# note this combined file will be also used in part B

data.to_csv('combined_csv_v2.csv', index=False)
```

Create the training and testing sets again.

In [64]:
```python
# Separate features (X) and target (y)
X = data.drop('target', axis=1)
y = data['target']

# Separate test set (20%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y # maintain class balance
)

# Then I split again for: train and validation
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp,
    test_size=0.2,
    random_state=42,
    stratify=y_temp
)
```

New baseline classifier

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

In [65]:
```python
# Instantiate another logistic regression model
classifier2 = LogisticRegression(
    max_iter=1000,
    random_state=42,
    class_weight='balanced',
    solver='lbfgs'
)

classifier2.fit(X_train, y_train)
```

Out[65]:
```
                          LogisticRegression                    ⓘ ?
                                                                (ht
                                                                lea
LogisticRegression(class_weight='balanced', max_iter=1000, random_s
tate=42)
```

In [66]:
```python
def plot_confusion_matrix(test_labels, target_predicted):

    cm = confusion_matrix(test_labels, target_predicted)

    # Plot
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.title('Confusion Matrix')
    plt.show()
```

In [67]:
```python
def plot_roc(test_labels, target_predicted):
    # Get predicted probabilities for positive class
    target_predicted_proba = classifier2.predict_proba(X_test)[:, 1]

    # Calculate ROC curve
    fpr, tpr, _ = roc_curve(test_labels, target_predicted_proba)
    roc_auc = auc(fpr, tpr)

    # Plot ROC
    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.4f})')
    plt.plot([0, 1], [0, 1], 'k--', label='Random')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()

    # Calculate and print statistics
    cm = confusion_matrix(test_labels, target_predicted)
    tn, fp, fn, tp = cm.ravel()

    accuracy = accuracy_score(test_labels, target_predicted)
    precision = precision_score(test_labels, target_predicted)
    recall = recall_score(test_labels, target_predicted)
    sensitivity = recall
    specificity = tn / (tn + fp)


    print("STATISTICS")
    print("\n")
    print(f"Accuracy:    {accuracy:.4f}")
    print(f"Precision:   {precision:.4f}")
    print(f"Recall:      {recall:.4f}")
    print(f"Sensitivity: {sensitivity:.4f}")
    print(f"Specificity: {specificity:.4f}")
    print(f"ROC-AUC:     {roc_auc:.4f}")
```
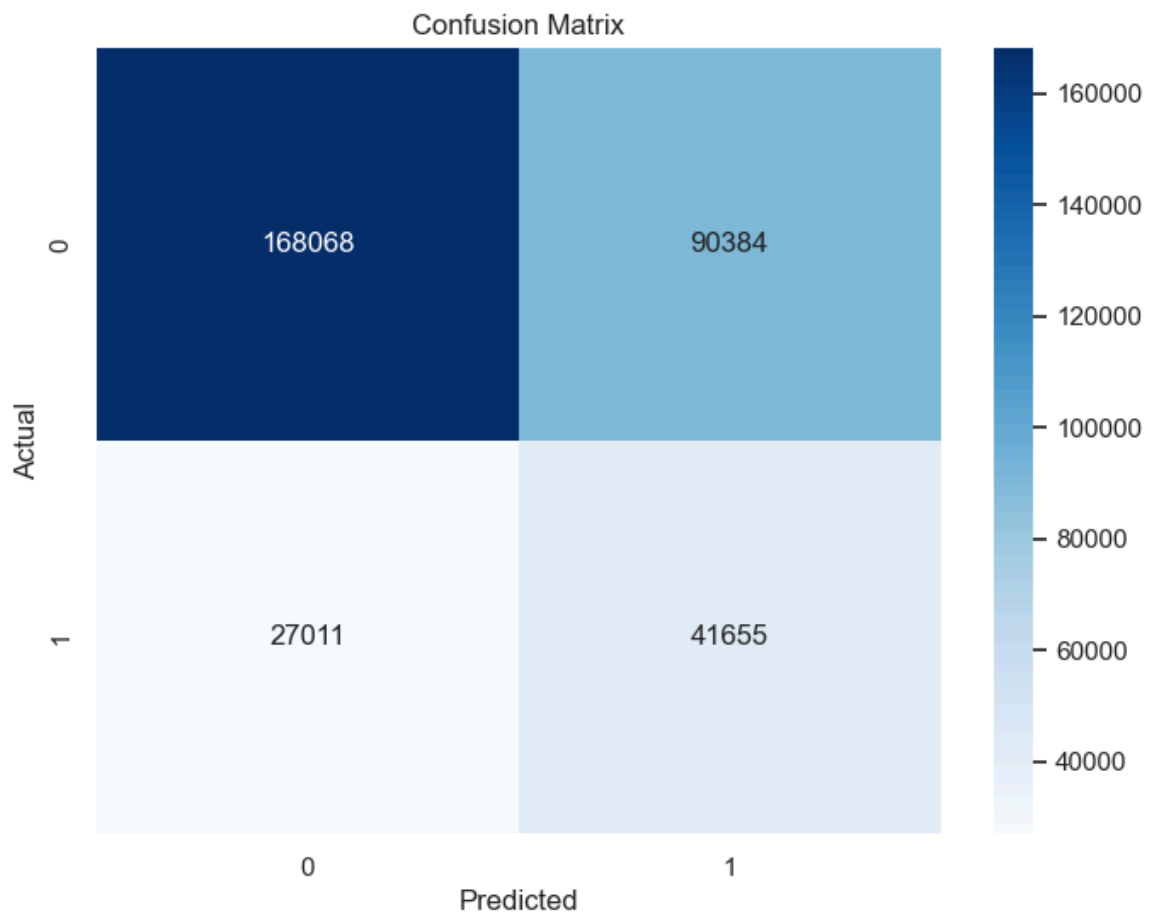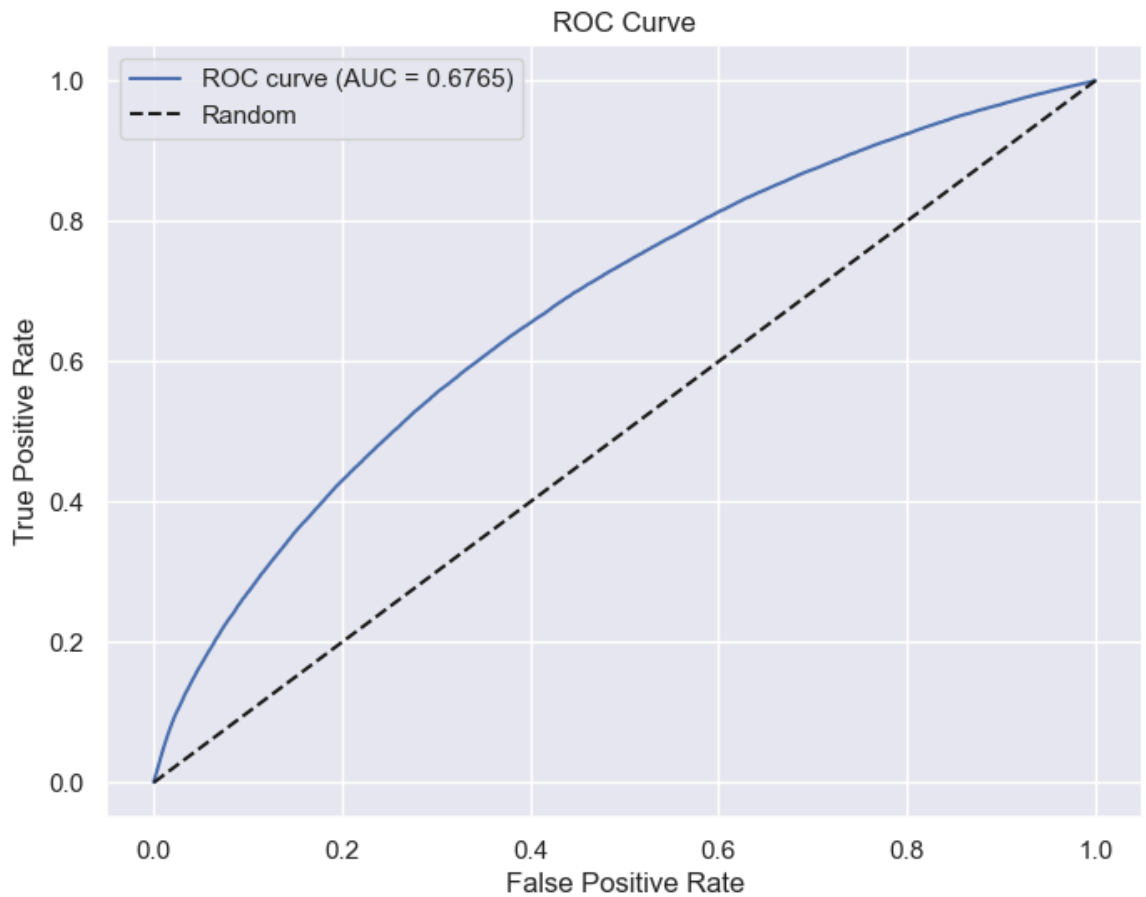
In [68]:
```python
y_test_pred = classifier2.predict(X_test)
```

In [69]: `plot_confusion_matrix(y_test, y_test_pred)`

Confusion Matrix

	Predicted 0	Predicted 1
Actual 0	168068	90384
Actual 1	27011	41655

In [70]: `plot_roc(y_test, y_test_pred)`

ROC Curve

STATISTICS

Accuracy: 0.6411
Precision: 0.3155
Recall: 0.6066
Sensitivity: 0.6066
Specificity: 0.6503
ROC-AUC: 0.6765

In [71]:
```python
# Make predictions on all sets
y_train_pred = classifier2.predict(X_train)
y_val_pred = classifier2.predict(X_val)
y_test_pred = classifier2.predict(X_test)

# Get probabilities for ROC-AUC
y_train_proba = classifier2.predict_proba(X_train)[:, 1]
y_val_proba = classifier2.predict_proba(X_val)[:, 1]
y_test_proba = classifier2.predict_proba(X_test)[:, 1]
```

```
In [72]: train_acc = accuracy_score(y_train, y_train_pred)
         train_prec = precision_score(y_train, y_train_pred)
         train_rec = recall_score(y_train, y_train_pred)
         train_f1 = f1_score(y_train, y_train_pred)
         train_auc = roc_auc_score(y_train, y_train_proba)

         val_acc = accuracy_score(y_val, y_val_pred)
         val_prec = precision_score(y_val, y_val_pred)
         val_rec = recall_score(y_val, y_val_pred)
         val_f1 = f1_score(y_val, y_val_pred)
         val_auc = roc_auc_score(y_val, y_val_proba)

         test_acc = accuracy_score(y_test, y_test_pred)
         test_prec = precision_score(y_test, y_test_pred)
         test_rec = recall_score(y_test, y_test_pred)
         test_f1 = f1_score(y_test, y_test_pred)
         test_auc = roc_auc_score(y_test, y_test_proba)

         # Display comparison
         print("\nMetric          Training     Validation     Test")
         print("-" * 55)
         print(f"Accuracy        {train_acc:.4f}        {val_acc:.4f}         {tes
         print(f"Precision       {train_prec:.4f}         {val_prec:.4f}          {t
         print(f"Recall          {train_rec:.4f}        {val_rec:.4f}         {tes
         print(f"F1-Score        {train_f1:.4f}         {val_f1:.4f}        {test_
         print(f"ROC-AUC         {train_auc:.4f}        {val_auc:.4f}          {tes
```

Metric	Training	Validation	Test
Accuracy	0.6418	0.6415	0.6411
Precision	0.3160	0.3151	0.3155
Recall	0.6063	0.6033	0.6066
F1-Score	0.4154	0.4140	0.4151
ROC-AUC	0.6762	0.6754	0.6765

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

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

Yes, there is a noticeable improvement in the model's performance after adding the holiday and weather data. The updated model achieved higher scores across all key metrics: accuracy increased from 0.5889 to 0.6411, precision improved from 0.2803 to 0.3155, and ROC-AUC rose from 0.6334 to 0.6765. While there was a slight decrease in recall of 0.80% (from 61.15% to 60.66%), meaning the model identified marginally fewer actual delay cases, this minor drop is offset by a substantial reduction in false positives (from 107,795 to 90,384). This indicates that after incorporating holiday and weather features, the model became more precise. It is now less likely to incorrectly predict a delay when there isn't one. The ROC curve also shows a greater separation from the random baseline, indicating stronger discriminative power.

In terms of the confusion matrix, the improved model correctly identified 168,068 true negatives and 41,655 true positives, compared to 150,657 true negatives and 41,986 true positives in the base model. While the number of true positives remained relatively stable,

the addition of holiday and weather features significantly reduced the number of false positives, demonstrating that the model became better at distinguishing between the two classes, especially for correctly predicting non-delay cases.

Overall, the inclusion of external contextual variables such as holidays and weather data led to more accurate and reliable predictions, improving both classification balance and the overall robustness of the model.

Step 6: Using Tableau

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

what to do:

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

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

```
In [73]:  # copy the link here
```

Conclusion

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

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

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?

Unfortunately, the model performance does not fully meet the initial business goals. While Iteration 2 showed significant improvements (accuracy: 64.11%, precision: 31.55%, recall: 60.66%), these metrics fall short of the targets (accuracy ≥80%, precision ≥75%, recall ≥70%).

If I had more time, I would explore several additional approaches to improve performance.

- First, I would experiment with more advanced algorithms such as Random Forest, Gradient Boosting (XGBoost), or ensemble methods that can better capture non-linear relationships in the data.
- Second, I would address the severe class imbalance more aggressively using techniques like adjusting class weights more strategically.

2. To what extent did your model improve as you made changes to your dataset? What types of techniques did you employ throughout this project that you felt yielded the greatest improvements in your model?

The model improved substantially through iterative feature engineering. Starting from the baseline logistic regression model (Iteration 1) with basic flight attributes, the addition of holiday and weather features in Iteration 2 resulted in an 8.86% increase in accuracy and a remarkable 12.56% improvement in precision.

The most impactful techniques were: (1) Adding domain-specific features (holidays and weather conditions) that have logical connections to flight delays, which validated the hypothesis that external factors significantly influence delays; (2) Using one-hot encoding for categorical variables to properly represent airlines, airports, and temporal features; (3) Feature selection by carefully choosing which columns to include based on their availability at prediction time. The weather and holiday features proved particularly valuable, reducing false positives by approximately 17,411 cases while maintaining similar true positive rates, demonstrating that incorporating contextual real-world data can dramatically enhance predictive performance.

3. What were some of the biggest challenges you encountered throughout this project?

Several significant challenges emerged throughout this project.

- Firstly, dealing with severe class imbalance was problematic, as approximately 80% of flights were not delayed, causing the model to be biased toward predicting the majority class. This resulted in high accuracy but poor precision and recall for the minority class (delays).
- Secondly, merging multiple datasets and formats was complex, particularly aligning weather data by date and airport code.
- Thirdly, the large dataset size required careful memory management and processing time considerations.

4. What were the three most important things you learned about machine learning while completing this project?

- First, I learned that feature engineering is often more impactful than algorithm selection. The addition of domain knowledge through holiday and weather features improved model performance more significantly than simply adjusting hyperparameters or trying different algorithms.
- Second, I learned that model evaluation requires looking beyond accuracy, especially with imbalanced datasets. A model with 99% accuracy that simply predicts "no delay" for everything is useless in practice. Understanding the trade-offs between precision, recall, and other metrics, and interpreting confusion matrices in business context, is essential for developing actually useful models.

- Third, I learned that iterative improvement and systematic experimentation are key to the machine learning workflow. Starting with a baseline model, adding features incrementally, evaluating their impact, and documenting changes allows for

In [ ]: