Term Project

Airlines On-Time Performance, Delays, Cancellations and Diversions

MILESTONE 1 - Data selection and EDA

Introduction: Airline cancellations or delays are one of the major causes of passenger inconvenience. With publicly available dataset, using data science, I am hoping to gain meaningful insights into the best-performing airlines and understand the causes of delays, diversions and cancellations across different airline carriers.

For the final project, I would like to analyze airline data to identify different factors and their effects on a carrier's performance. Using the available performance measures I would like to be able to predict the chances of a flight being on-time/delayed/cancelled.

Data Source: Excel files from BTS. The Excel data has airline performance factors such as cancelled, diverted, delayed and on-time data. The downloaded raw data has up to 34 columns. https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp?20=E (Download Raw Data link for data).

Problem statement addressed:

This study will benefit Customers as it will help predict a flights performance. Customers can lookup the chances of their flight reaching on-time during their booking or even before heading to the airport. Airlines can also benefit by comparing airline performances and predicting possibilities of delay based on aircraft/origin/destination and apply corrective measures to reduce cancellations and delays and improve on-time performance.

Data Transformation

In the data transformation step, I will be modifying the following:

1. Cancellation reason in the flight dataset is represented as A, B, C and D. I will be updating the cancellation code as follows:

   A Carrier

   B Weather

   C National Air System

   D Security

1. I will be adding a new column 'Status' with the status of a flight such as, On-Time, Delayed, Cancelled, Diverted.

2. Diverted column is of binary value which can be modified to a Yes/No

In [35]:
```
#Load necessary libraries
```

```
import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
```

In [36]:

```
#Read flight data from "https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp?20=E"

flight_data_df = pd.read_csv('T_ONTIME_MARKETING.csv')
flight_data_df.head()
```

Out[36]:

	YEAR	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	MKT_UNIQUE_CARRIER	OP_UNIQUE_CAI
0	2022	2	5	1	7	5/1/2022 12:00:00 AM	AA	
1	2022	2	5	1	7	5/1/2022 12:00:00 AM	AA	
2	2022	2	5	1	7	5/1/2022 12:00:00 AM	AA	
3	2022	2	5	1	7	5/1/2022 12:00:00 AM	AA	
4	2022	2	5	1	7	5/1/2022 12:00:00 AM	AA	

5 rows × 38 columns

1. DROP DUPLICATES

Duplicates cause inconsistent results when dealing with statistics. Hence dropping duplicate rows.

In [37]:

```
print('Dataframe before dropping duplicates :', flight_data_df.shape)
flight_data_df = flight_data_df.drop_duplicates() # 1,389 rows dropped
print('Dataframe after dropping duplicates :',flight_data_df.shape)
```

```
Dataframe before dropping duplicates : (602950, 38)
Dataframe after dropping duplicates : (601561, 38)
```

2. Update Null values and Drop null rows, if any

Drop null rows, if any and update null values to 0 for delays

In [38]:

```
#Drop null values
print('Dataframe before dropping null rows :', flight_data_df.shape)
flight_data_df.dropna()
print('Dataframe after dropping null rows :', flight_data_df.shape)
#Update null values to 0
flight_data_df.DISTANCE = flight_data_df.DISTANCE.fillna(0)
flight_data_df.DEP_DELAY = flight_data_df.DEP_DELAY.fillna(0)
flight_data_df.ARR_DELAY = flight_data_df.ARR_DELAY.fillna(0)
flight_data_df.CARRIER_DELAY = flight_data_df.CARRIER_DELAY.fillna(0)
flight_data_df.WEATHER_DELAY = flight_data_df.WEATHER_DELAY.fillna(0)
flight_data_df.NAS_DELAY = flight_data_df.NAS_DELAY.fillna(0)
flight_data_df.SECURITY_DELAY = flight_data_df.SECURITY_DELAY.fillna(0)
flight_data_df.LATE_AIRCRAFT_DELAY = flight_data_df.LATE_AIRCRAFT_DELAY.fillna(0)
```

```
Dataframe before dropping null rows : (601561, 38)
Dataframe after dropping null rows : (601561, 38)
```

In [39]:
```
flight_data_df.loc[pd.isna(flight_data_df.CANCELLATION_CODE), 'CANCELLATION_CODE']='Z'
```

3. Add new features

Cancellation code is represented as A, B, C and D, which is not very informative. The BTS website provided details on this code as follows:

In [40]:
```
flight_data_df['CANCELLATION_REASON'] = ''
flight_data_df.CANCELLATION_REASON = np.where(flight_data_df.CANCELLATION_CODE=='A', 'Ca
                                 np.where(flight_data_df.CANCELLATION_CODE=='B', 'Weathe
                                          np.where(flight_data_df.CANCELLATION_CODE=='C'
                                                   np.where(flight_data_df.CANCELLATION_
                                                             np.where(flight_data_df.CAN

flight_data_df.groupby(['CANCELLATION_REASON'])['CANCELLATION_REASON'].count().sort_inde
```

Out[40]:
```
CANCELLATION_REASON
Carrier                  4902
National Air System      1394
Not Cancelled          590957
Security                    1
Weather                  4307
Name: CANCELLATION_REASON, dtype: int64
```

Adding a new column 'STATUS' that tells the status of a flight

In [41]:
```
flight_data_df['STATUS'] = ''

flight_data_df.STATUS = np.where(flight_data_df.CANCELLED==1, 'Cancelled',
                                 np.where(flight_data_df.DIVERTED==1, 'Diverted',
                                          np.where(flight_data_df.ARR_DELAY<=15, 'On-Tim
                                                   np.where(flight_data_df.ARR_DELAY>15,
flight_data_df.groupby(['STATUS'])['STATUS'].count().sort_index()
```

Out[41]:
```
STATUS
Cancelled     10604
Delayed      119624
Diverted       1581
On-Time      469752
Name: STATUS, dtype: int64
```

Creating a new column 'ARR_DELAYED'. A flag that represents if a flight was delayed. Similar to CANCELLED and DIVERTED As a step to data reduction, I will be considering flights arriving 15 minutes or later as delayed

In [42]:
```
flight_data_df.loc[(flight_data_df['ARR_DELAY']>15), 'ARR_DELAYED'] = True
flight_data_df.loc[(flight_data_df['ARR_DELAY']<=15), 'ARR_DELAYED'] = False

flight_data_df.groupby(['ARR_DELAYED'])['ARR_DELAYED'].count().sort_index()
```

Out[42]:
```
ARR_DELAYED
False    481937
True     119624
Name: ARR_DELAYED, dtype: int64
```

Adding a new column 'DELAY_REASON' that tells the reason for a flight getting delayed

In [43]:
```
flight_data_df['DELAY_REASON'] = np.where(flight_data_df.CARRIER_DELAY != 0, 'Carrier',
                                          np.where(flight_data_df.LATE_AIRCRAFT_DELAY !=
                                                   np.where(flight_data_df.WEATHER_DELAY
```

```
                                                 np.where(flight_data_df.NAS_
                                                          np.where(flight_dat

flight_data_df.groupby(['DELAY_REASON'])['DELAY_REASON'].count().sort_index()
```

Out[43]:
```
DELAY_REASON
                  477611
Carrier            74794
LateAircraft       26097
NAS                18695
Security             142
Weather             4222
Name: DELAY_REASON, dtype: int64
```

Data Visualization:

Implementing arithmetic functions for statistical analysis

Creating a new dataframe with total number of flights per operating carrier to calculate the %

In [44]:
```
flight_totals = flight_data_df.value_counts(subset=['OP_UNIQUE_CARRIER']).reset_index()
flight_totals_df = pd.DataFrame(flight_totals)
flight_totals_df.columns = ['OP_UNIQUE_CARRIER','TOTAL']
flight_totals_df['PERCENTAGE'] = round(flight_totals_df.TOTAL/flight_totals_df.TOTAL.sum

flight_totals_df = flight_totals_df.sort_values('PERCENTAGE',ascending=False)
flight_totals_df.head(5)
```

Out[44]:

	OP_UNIQUE_CARRIER	TOTAL	PERCENTAGE
0	WN	107950	17.94
1	DL	76021	12.64
2	AA	71471	11.88
3	OO	66615	11.07
4	UA	53535	8.90

Calculate percentage by carrier and flight status

In [45]:
```
flight_status = flight_data_df.value_counts(subset=['OP_UNIQUE_CARRIER','STATUS']).reset_
flight_status_df = pd.DataFrame(flight_status) #create a dataframe
flight_status_df.columns = ['OP_UNIQUE_CARRIER','STATUS', 'COUNT'] #Add column names
flight_status_df = flight_status_df.sort_values('OP_UNIQUE_CARRIER') #Sort by operating

flight_status_df['PERCENTAGE'] = ''

for index, row in flight_status_df.iterrows():
    tot = flight_totals.loc[flight_totals.OP_UNIQUE_CARRIER==row.OP_UNIQUE_CARRIER].TOTA
    val = (row.COUNT/tot * 100)
    flight_status_df.at[index,'PERCENTAGE'] = round(val[0].astype(float),2) #Calculate t

flight_status_df.head(10)
```

Out[45]:

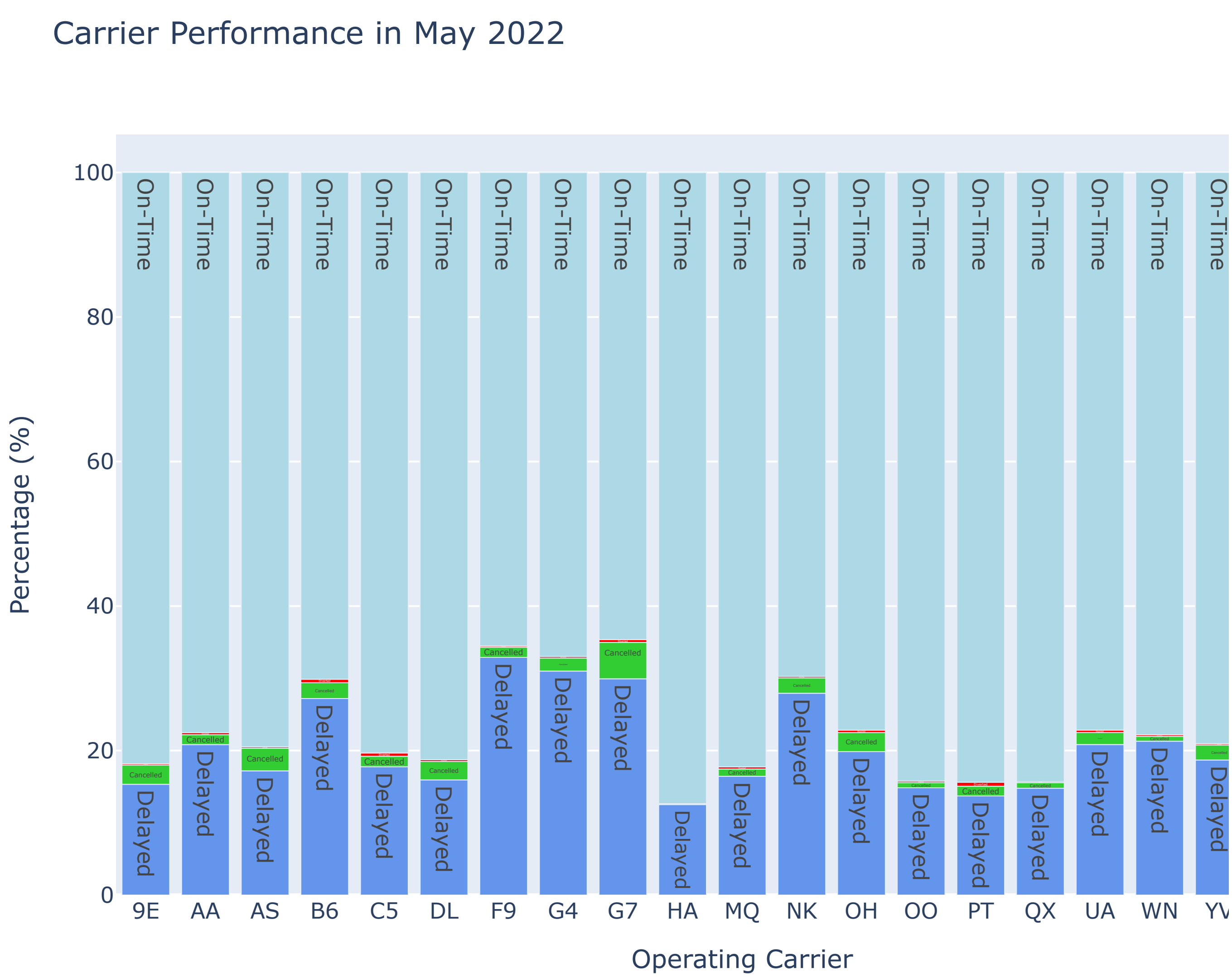
	OP_UNIQUE_CARRIER	STATUS	COUNT	PERCENTAGE
33	9E	Delayed	3113	15.33
48	9E	Cancelled	542	2.67
74	9E	Diverted	35	0.17

8	9E	On-Time	16613	81.83
41	AA	Cancelled	973	1.36
56	AA	Diverted	215	0.3
3	AA	On-Time	55403	77.52
11	AA	Delayed	14880	20.82
47	AS	Cancelled	608	3.12
10	AS	On-Time	15502	79.49

Bar chart for carier performance in May 2022

In [141…

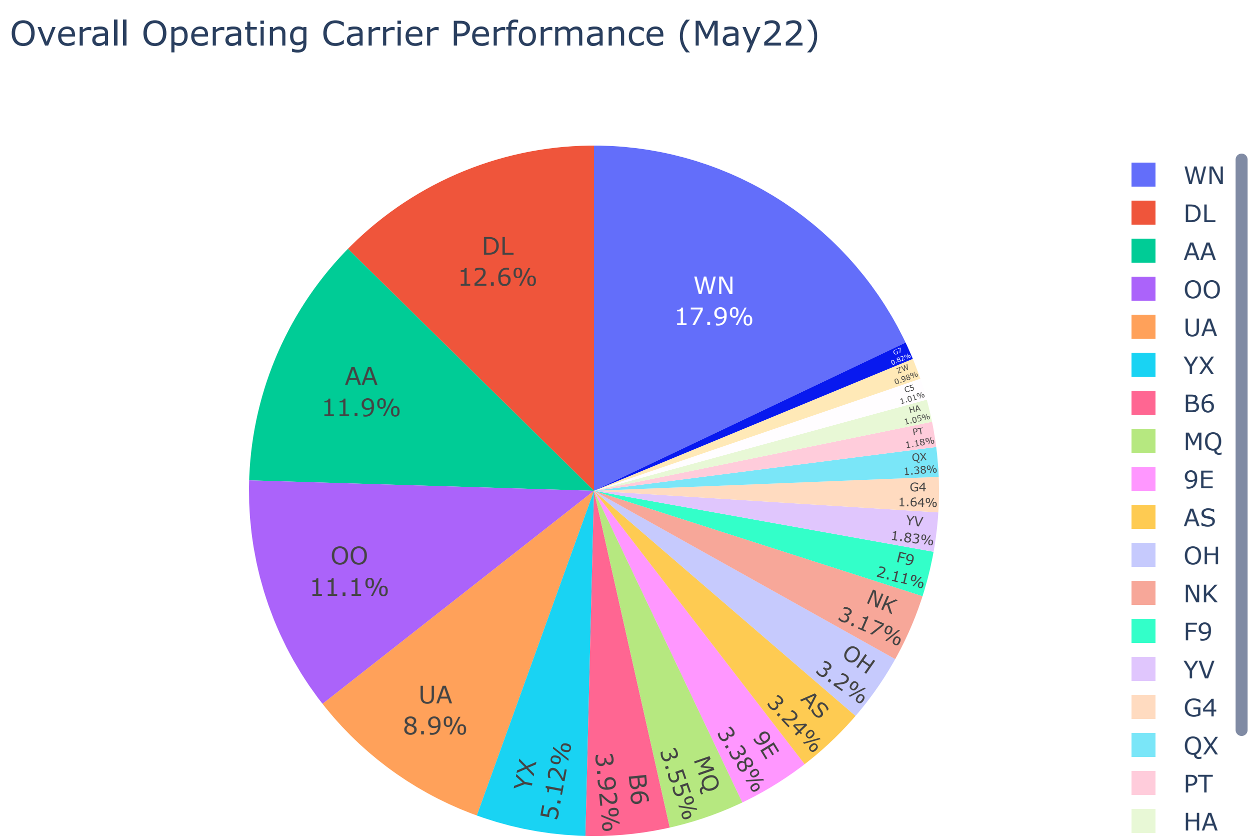
```
colors = [ # matplotlib named colors
          'cornflowerblue','limegreen','red', 'lightblue','lightsteelblue',
         # any color using the color codes
         "#a977e2"]
fig = px.bar(flight_status_df, x="OP_UNIQUE_CARRIER", y="PERCENTAGE", title="Carrier Per
             color="STATUS", text="STATUS",color_discrete_sequence=colors,
             labels={"OP_UNIQUE_CARRIER": "Operating Carrier",
                     "PERCENTAGE": "Percentage (%)"})
fig.update_layout(autosize=False,width=900, height=600)
fig.show('notebook')
```



Hawaian airlines had the best on-time performance in May'22 followed by Air Wisconsin(ZW). Frontier airlines(F9) had the most number of delays at 32.9% GoJet had the most cancellations at 7%

Pie chart for Overall Carrier performance in May'22

In [142…
```
fig = px.pie(flight_totals_df, values='PERCENTAGE', names='OP_UNIQUE_CARRIER',
             title='Overall Operating Carrier Performance (May''22)')
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show('notebook')
```

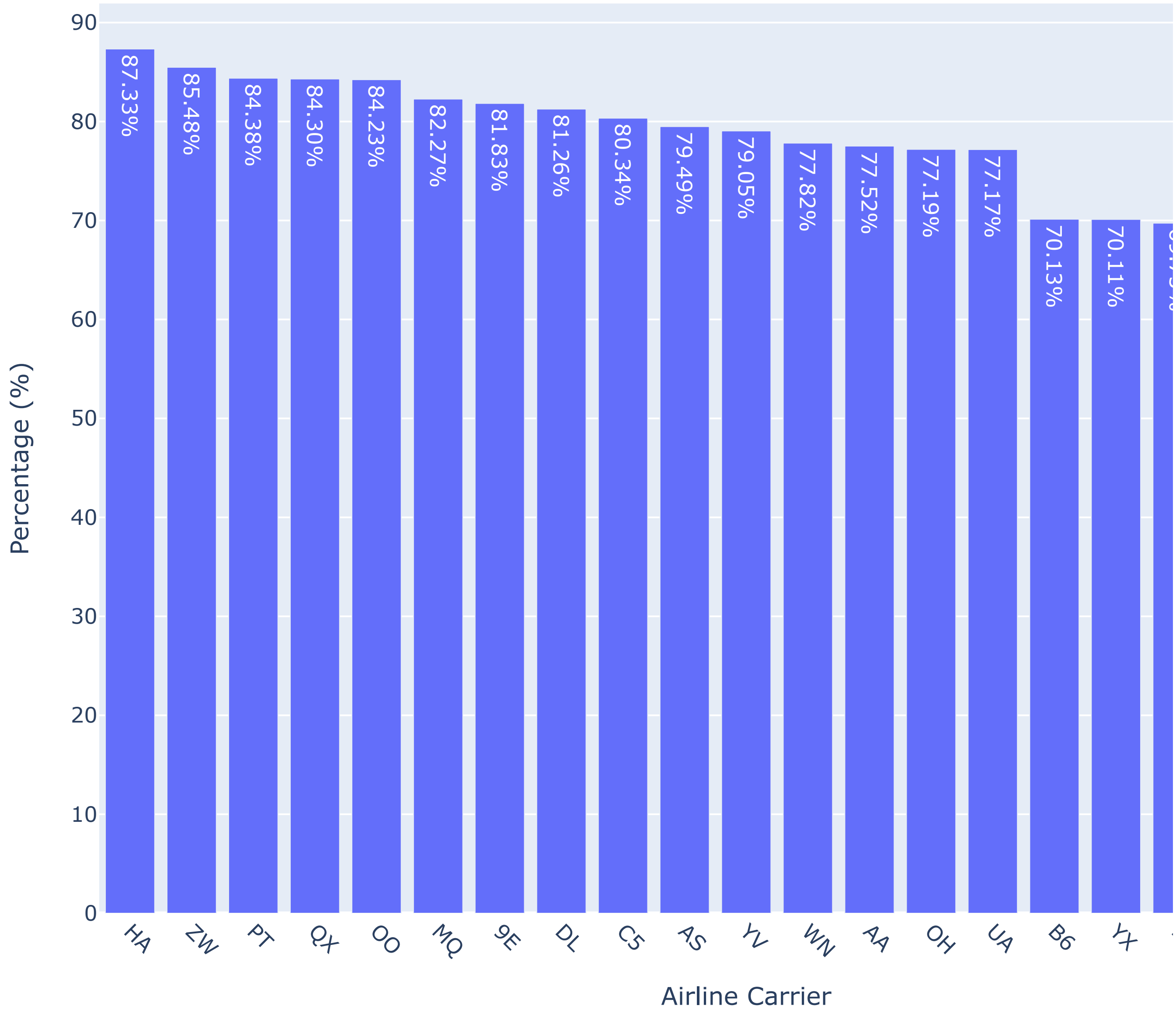


We can see southwest carrier (WN) had the most number of flights in May 2022.

Bar plot for Airline with best on-time performance

In [143…
```
airline_on_time_performance = flight_status_df[flight_status_df.STATUS == 'On-Time'].sor

fig=px.bar(airline_on_time_performance,
           x=airline_on_time_performance.OP_UNIQUE_CARRIER,
           y=airline_on_time_performance.PERCENTAGE, title="Airline On-Time Performance"
           text=airline_on_time_performance.PERCENTAGE.apply(lambda x: '{0:1.2f}%'.forma
           labels=dict(OP_UNIQUE_CARRIER="Airline Carrier", PERCENTAGE="Percentage (%)")
fig.update_xaxes(tickangle=45)
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```

Airline On-Time Performance



Hawaiian airline was the best performing airline in May'22 with 87.33% on time performance and Go-Jet is the least performing airline with 64.6% on-time performance.

In [49]:
```
#Load csv file with airport names for origin and destination
airport_data_df = pd.read_csv('L_AIRPORT.csv')
airport_data_df.head()
```

Out[49]:

	Code	Description
0	01A	Afognak Lake, AK: Afognak Lake Airport
1	03A	Granite Mountain, AK: Bear Creek Mining Strip
2	04A	Lik, AK: Lik Mining Camp
3	05A	Little Squaw, AK: Little Squaw Airport
4	06A	Kizhuyak, AK: Kizhuyak Bay

In [50]:
```
#Create a new dataframe with the percentage by origin airport and status
flight_origin_totals = flight_data_df.value_counts(subset=['ORIGIN']).reset_index()   #ge
flight_origin_totals_df = pd.DataFrame(flight_origin_totals)#create a dataframe
flight_origin_totals_df.columns = ['ORIGIN','TOTAL']#Add column names
```

```
#Calculate the percentage by origin airport
flight_origin_totals_df['PERCENTAGE'] = round(flight_origin_totals_df.TOTAL/flight_origi


origin_airport_delays = flight_data_df.value_counts(subset=['ORIGIN','STATUS']).reset_in
origin_airport_df = pd.DataFrame(origin_airport_delays) #create a dataframe
origin_airport_df.columns = ['ORIGIN','STATUS', 'COUNT'] #add column names
origin_airport_df = origin_airport_df.sort_values('ORIGIN') #sort by origin
origin_airport_df['PERCENTAGE'] = ''

for index, row in origin_airport_df.iterrows():
    tot = flight_origin_totals.loc[flight_origin_totals.ORIGIN==row.ORIGIN].TOTAL.values
    val = (row.COUNT/tot * 100)
    origin_airport_df.at[index,'PERCENTAGE'] = round(val[0].astype(float),2)  #calulate

origin_airport_df.head(10)
origin_airport_df = origin_airport_df.sort_values('PERCENTAGE',ascending=False) #sort by

#Add the airport name from the airport_data_df and add as a new column to the origin_air
origin_airport_df=pd.merge(origin_airport_df, airport_data_df, how='left', left_on='ORIG
origin_airport_df.rename(columns={'Description':'ORIGIN_AIRPORT_NAME'}, inplace=True)
del origin_airport_df['Code']

new = origin_airport_df.ORIGIN_AIRPORT_NAME.str.split(":", n = 1, expand = True)
origin_airport_df["ORIGIN_AIRPORT_NAME"] =  new[1]
origin_airport_df.head()
```

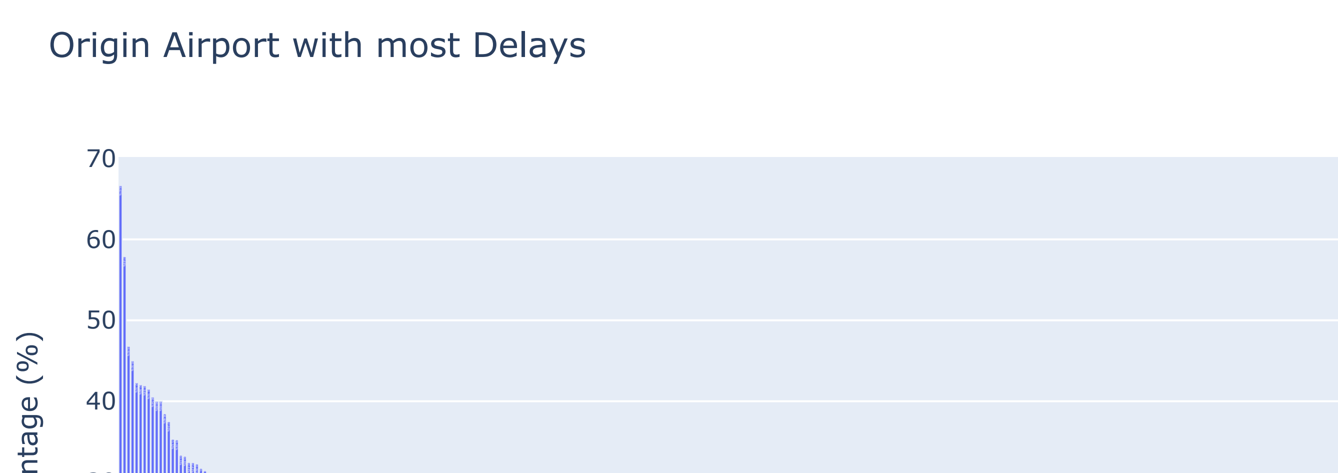
Out[50]:

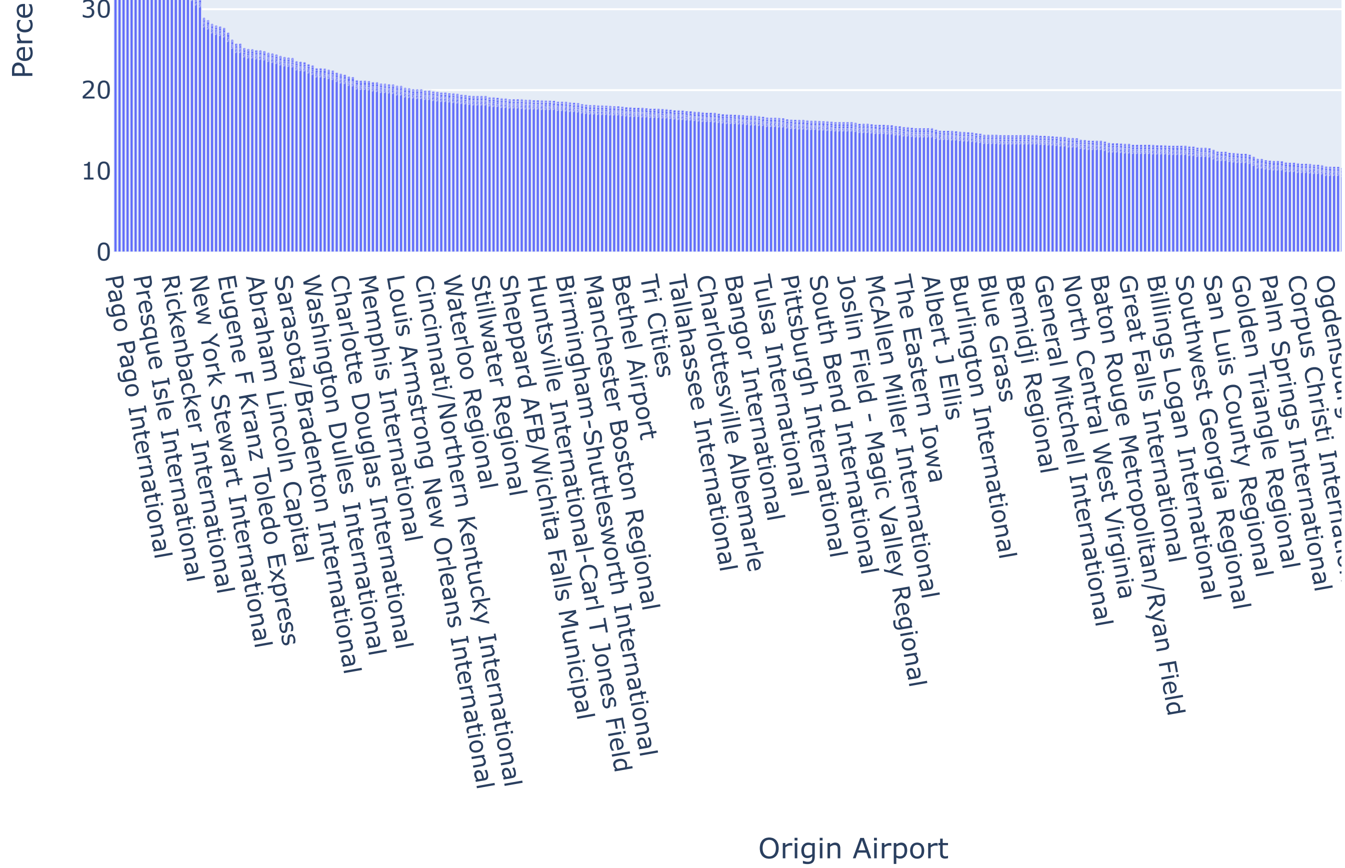
	ORIGIN	STATUS	COUNT	PERCENTAGE	ORIGIN_AIRPORT_NAME
0	GST	On-Time	12	100.0	Gustavus Airport
1	STC	On-Time	1	100.0	St. Cloud Regional
2	LWS	On-Time	95	96.94	Lewiston Nez Perce County
3	BGM	On-Time	30	96.77	Greater Binghamton/Edwin A. Link Field
4	DRT	On-Time	60	96.77	Del Rio International

Bar chart for Origin airport with most delays

In [144…

```
fig = px.bar(origin_airport_df[origin_airport_df.STATUS=="Delayed"], x="ORIGIN_AIRPORT_N
             title="Origin Airport with most Delays",
             text=origin_airport_df[origin_airport_df.STATUS=="Delayed"].PERCENTAGE.appl
             labels=dict(ORIGIN_AIRPORT_NAME="Origin Airport", PERCENTAGE="Percentage (%
fig.update_xaxes(tickangle=80)
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```





It appears Tri Cities has multiple entries for different origin airports. Identify and update the airport name.

In [52]:
```
origin_airport_df[origin_airport_df.ORIGIN_AIRPORT_NAME.str.contains('Tri Cities')]
```

Out[52]:

	ORIGIN	STATUS	COUNT	PERCENTAGE	ORIGIN_AIRPORT_NAME
29	PSC	On-Time	451	90.56	Tri Cities
207	TRI	On-Time	302	81.4	Tri Cities
506	TRI	Delayed	66	17.79	Tri Cities
708	PSC	Delayed	44	8.84	Tri Cities
1018	TRI	Cancelled	3	0.81	Tri Cities
1093	PSC	Diverted	2	0.4	Tri Cities
1178	PSC	Cancelled	1	0.2	Tri Cities

Updating the airport name for PSC

In [53]:
```
origin_airport_df.loc[origin_airport_df["ORIGIN"] == "PSC", "ORIGIN_AIRPORT_NAME"] = 'Tr
```

In [54]:
```
origin_airport_df[origin_airport_df.ORIGIN_AIRPORT_NAME.str.contains('Tri Cities')]
```

Out[54]:

	ORIGIN	STATUS	COUNT	PERCENTAGE	ORIGIN_AIRPORT_NAME
29	PSC	On-Time	451	90.56	Tri Cities(PSC)
207	TRI	On-Time	302	81.4	Tri Cities
506	TRI	Delayed	66	17.79	Tri Cities
708	PSC	Delayed	44	8.84	Tri Cities(PSC)
1018	TRI	Cancelled	3	0.81	Tri Cities

1093	PSC	Diverted	2	0.4	Tri Cities(PSC)
1178	PSC	Cancelled	1	0.2	Tri Cities(PSC)

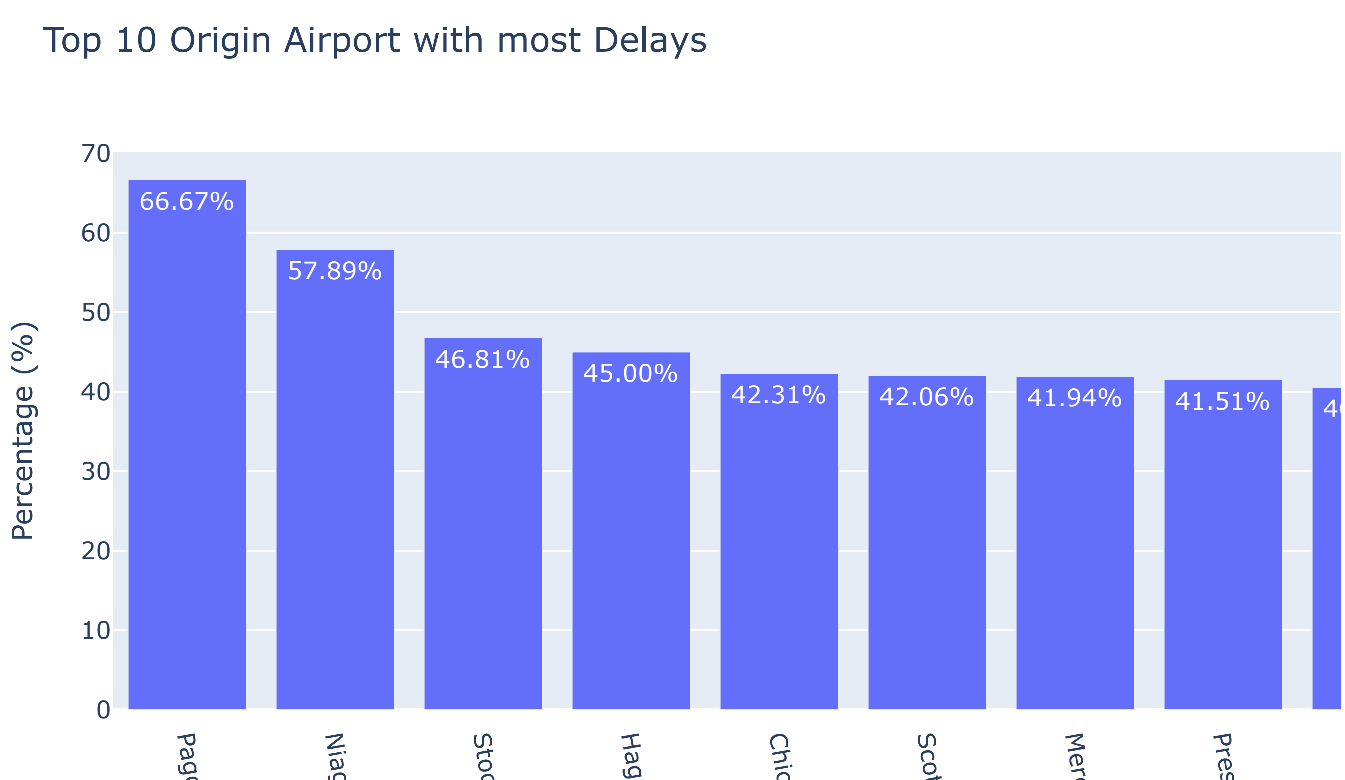
Since the chart has many airports to fit, filtering the list to get the top 10 origin airports with most delays

In [55]:
```
top_10_origin_delay_airports = origin_airport_df[origin_airport_df.STATUS=='Delayed'].he
top_10_origin_delay_airports
```

Out[55]:

	ORIGIN	STATUS	COUNT	PERCENTAGE	ORIGIN_AIRPORT_NAME
344	PPG	Delayed	2	66.67	Pago Pago International
361	IAG	Delayed	22	57.89	Niagara Falls International
368	SCK	Delayed	22	46.81	Stockton Metro
370	HGR	Delayed	9	45.0	Hagerstown Regional-Richard A. Henson Field
371	RFD	Delayed	22	42.31	Chicago/Rockford International
373	BLV	Delayed	45	42.06	Scott AFB MidAmerica St Louis
374	PSE	Delayed	39	41.94	Mercedita
375	PQI	Delayed	22	41.51	Presque Isle International
376	USA	Delayed	30	40.54	Concord Padgett Regional
378	RIW	Delayed	14	40.0	Central Wyoming Regional

In [145…
```
fig = px.bar(top_10_origin_delay_airports[top_10_origin_delay_airports.STATUS=="Delayed"
             title="Top 10 Origin Airport with most Delays",
             text=top_10_origin_delay_airports[top_10_origin_delay_airports.STATUS=="Del
             labels=dict(ORIGIN_AIRPORT_NAME="Origin Airport", PERCENTAGE="Percentage (%
fig.update_xaxes(tickangle=80)
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```



Flights originating from Pago Pago International are delayed 66.67%

DESTINATION

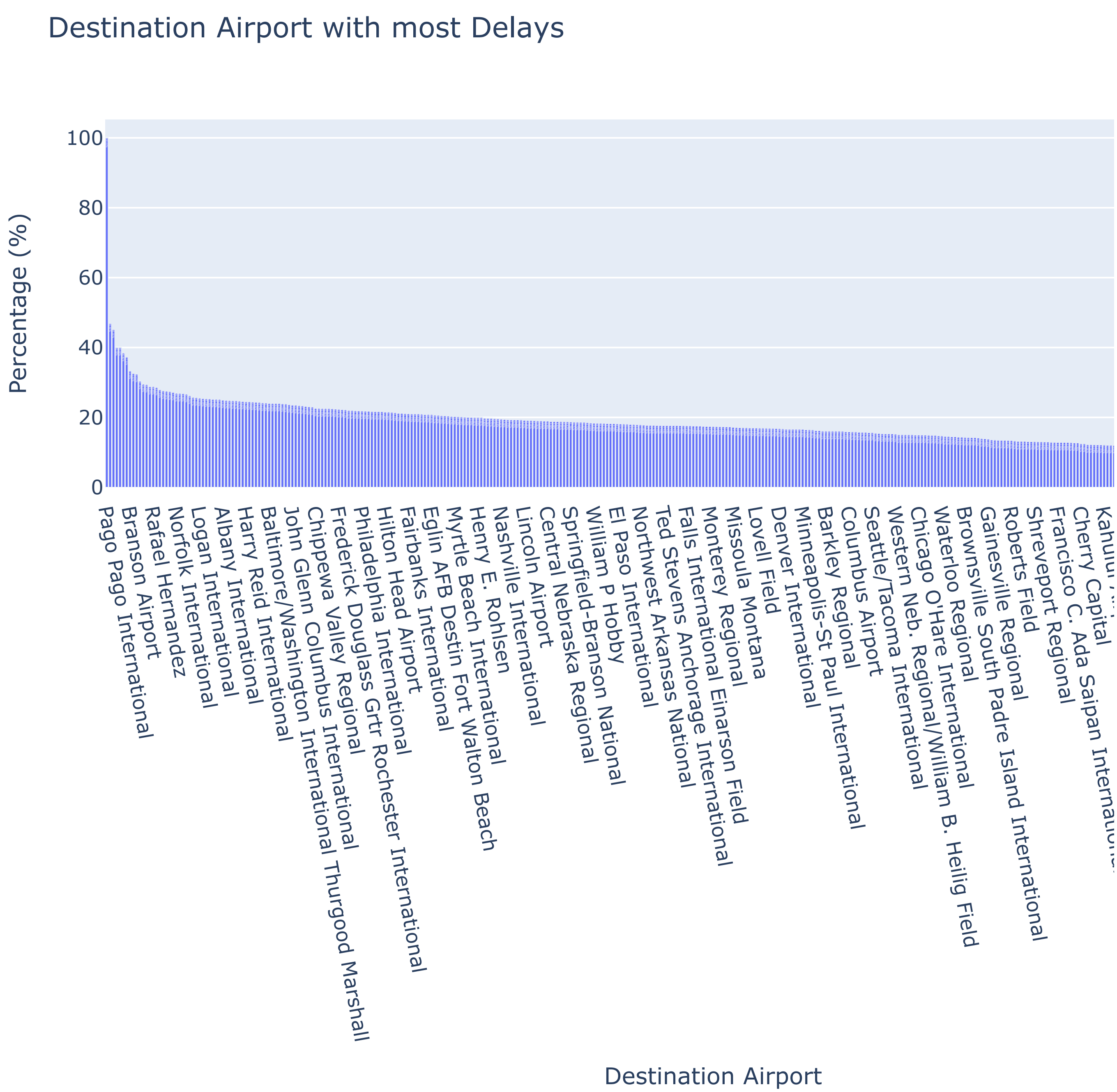
In [57]:
```
#Create a new dataframe with the percentage by origin airport and status
flight_dest_totals = flight_data_df.value_counts(subset=['DEST']).reset_index() #get the
flight_dest_totals_df = pd.DataFrame(flight_dest_totals) #create a dataframe
flight_dest_totals_df.columns = ['DEST','TOTAL'] #Add column names
#Calculate the percentage by destination airport
flight_dest_totals_df['PERCENTAGE'] = round(flight_dest_totals_df.TOTAL/flight_dest_tota


dest_airport_delays = flight_data_df.value_counts(subset=['DEST','STATUS']).reset_index(
dest_airport_df = pd.DataFrame(dest_airport_delays) #create a dataframe
dest_airport_df.columns = ['DEST','STATUS', 'COUNT'] #add column names
dest_airport_df = dest_airport_df.sort_values('DEST') #sort by destination
dest_airport_df['PERCENTAGE'] = ''

for index, row in dest_airport_df.iterrows():
    tot = flight_dest_totals.loc[flight_dest_totals.DEST==row.DEST].TOTAL.values  #get t
    val = (row.COUNT/tot * 100)
    dest_airport_df.at[index,'PERCENTAGE'] = round(val[0].astype(float),2) #calulate the

dest_airport_df.head(10)
dest_airport_df = dest_airport_df.sort_values('PERCENTAGE',ascending=False) #sort by perc

#Add the airport name from the airport_data_df and add as a new column to the dest_airpo
dest_airport_df=pd.merge(dest_airport_df, airport_data_df, how='left', left_on='DEST', r
dest_airport_df.rename(columns={'Description':'DEST_AIRPORT_NAME'}, inplace=True)
del dest_airport_df['Code']

new = dest_airport_df.DEST_AIRPORT_NAME.str.split(":", n = 1, expand = True)
dest_airport_df["DEST_AIRPORT_NAME"] =  new[1]
dest_airport_df.head()
```

Out[57]:

	DEST	STATUS	COUNT	PERCENTAGE	DEST_AIRPORT_NAME
0	GST	On-Time	12	100.0	Gustavus Airport
1	STC	On-Time	1	100.0	St. Cloud Regional
2	PPG	Delayed	3	100.0	Pago Pago International
3	TWF	On-Time	31	96.88	Joslin Field - Magic Valley Regional
4	PIH	On-Time	30	96.77	Pocatello Regional

Bar chart for Destination Airports with most delays

In [146…
```
fig = px.bar(dest_airport_df[dest_airport_df.STATUS=="Delayed"], x="DEST_AIRPORT_NAME",
             title="Destination Airport with most Delays",
             text=dest_airport_df[dest_airport_df.STATUS=="Delayed"].PERCENTAGE.apply(la
             labels=dict(DEST_AIRPORT_NAME="Destination Airport", PERCENTAGE="Percentage
fig.update_xaxes(tickangle=80)
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```



Updating Destination name for PSC

In [59]:
```
dest_airport_df[dest_airport_df.DEST_AIRPORT_NAME.str.contains('Tri Cities')]
dest_airport_df.loc[dest_airport_df["DEST"] == "PSC", "DEST_AIRPORT_NAME"] = 'Tri Cities
dest_airport_df[dest_airport_df.DEST_AIRPORT_NAME.str.contains('Tri Cities')]
```

Out[59]:

	DEST	STATUS	COUNT	PERCENTAGE	DEST_AIRPORT_NAME

40	PSC	On-Time	439	88.15	Tri Cities(PSC)
125	TRI	On-Time	310	83.56	Tri Cities
610	TRI	Delayed	57	15.36	Tri Cities
680	PSC	Delayed	59	11.85	Tri Cities(PSC)
985	TRI	Cancelled	4	1.08	Tri Cities

Since the chart has many airports to fit, filtering the list to get the top 10 destination airports with most delays

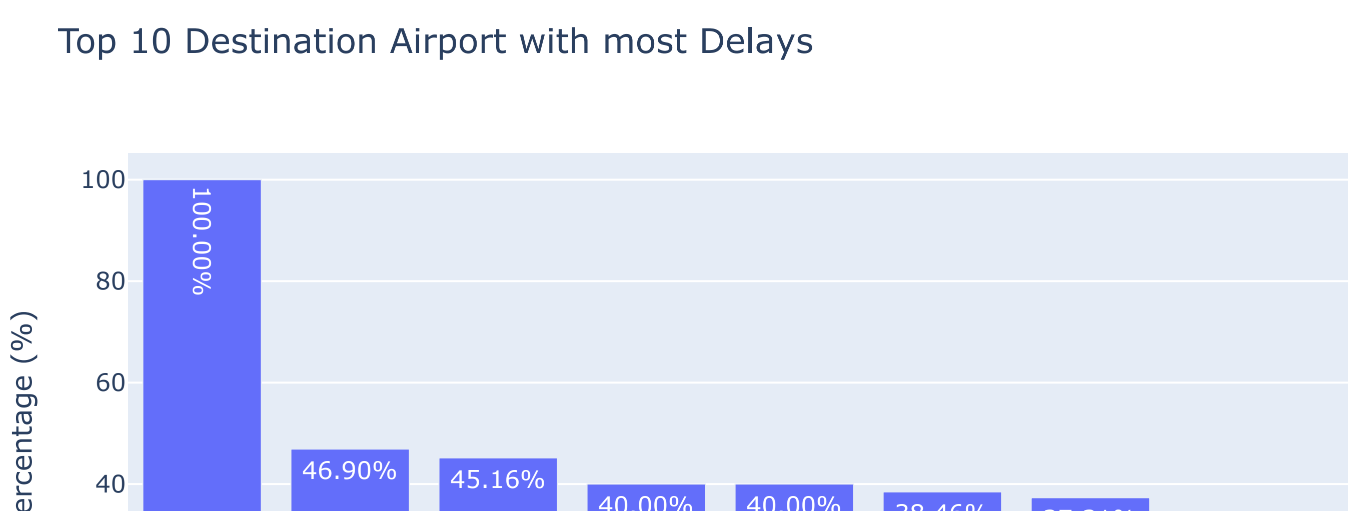
In [60]:
```
top_10_dest_delay_airports = dest_airport_df[dest_airport_df.STATUS=='Delayed'].head(10)
top_10_dest_delay_airports
```

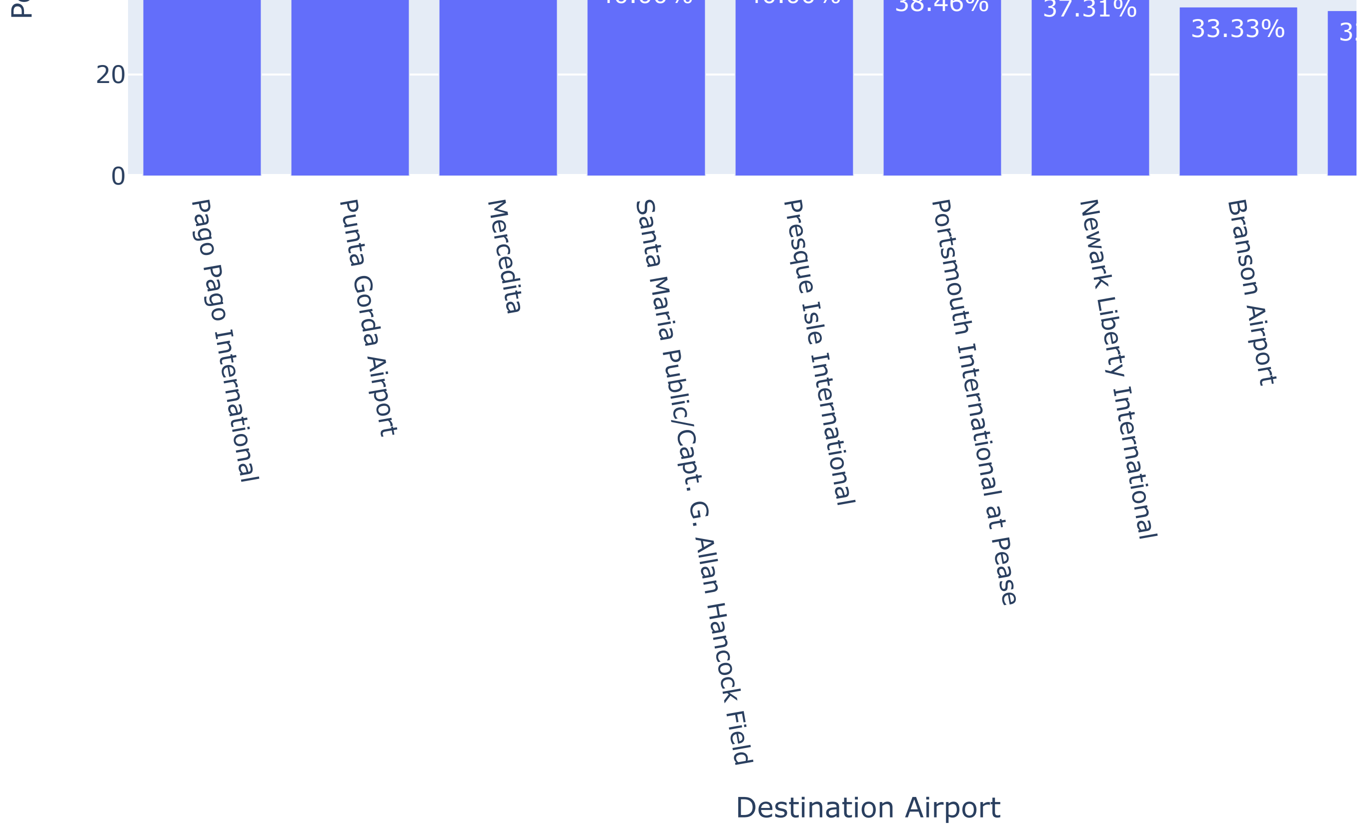
Out[60]:

	DEST	STATUS	COUNT	PERCENTAGE	DEST_AIRPORT_NAME
2	PPG	Delayed	3	100.0	Pago Pago International
368	PGD	Delayed	212	46.9	Punta Gorda Airport
369	PSE	Delayed	42	45.16	Mercedita
372	SMX	Delayed	4	40.0	Santa Maria Public/Capt. G. Allan Hancock Field
373	PQI	Delayed	20	40.0	Presque Isle International
374	PSM	Delayed	10	38.46	Portsmouth International at Pease
375	EWR	Delayed	5097	37.31	Newark Liberty International
376	BKG	Delayed	3	33.33	Branson Airport
377	SCK	Delayed	15	32.61	Stockton Metro
378	USA	Delayed	24	32.43	Concord Padgett Regional

Bar chart for Destination Airprot with most delays

In [147…
```
fig = px.bar(top_10_dest_delay_airports[top_10_dest_delay_airports.STATUS=="Delayed"], x
             title="Top 10 Destination Airport with most Delays",
             text=top_10_dest_delay_airports[top_10_dest_delay_airports.STATUS=="Delayed
             labels=dict(DEST_AIRPORT_NAME="Destination Airport", PERCENTAGE="Percentage
fig.update_xaxes(tickangle=80)
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```



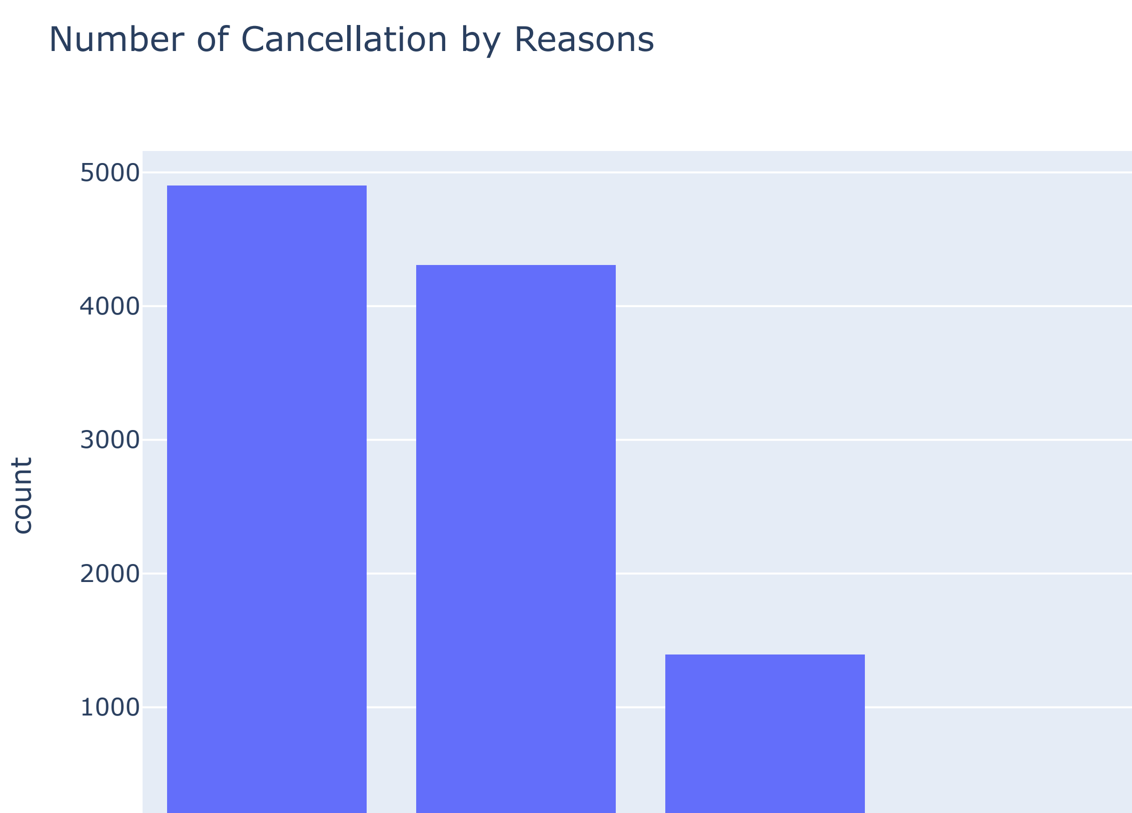


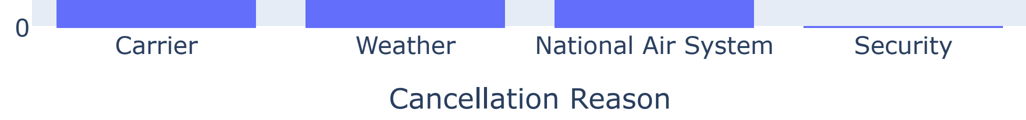
All flights flying into Pago Pago Internation airport are delayed.

Histogram for Overall cancellations by cancellation reason

In [148…

```
#Using CANCELLED==1 to filter the dataframe for only cancelled rows.
fig = px.histogram(flight_data_df[flight_data_df.CANCELLED==1], x="CANCELLATION_REASON",
                   title="Number of Cancellation by Reasons",
                   labels=dict(CANCELLATION_REASON="Cancellation Reason"))
fig.show('notebook')
```





From the chart, we can see that most cancellations in May'22 were due to carriers followed by weather

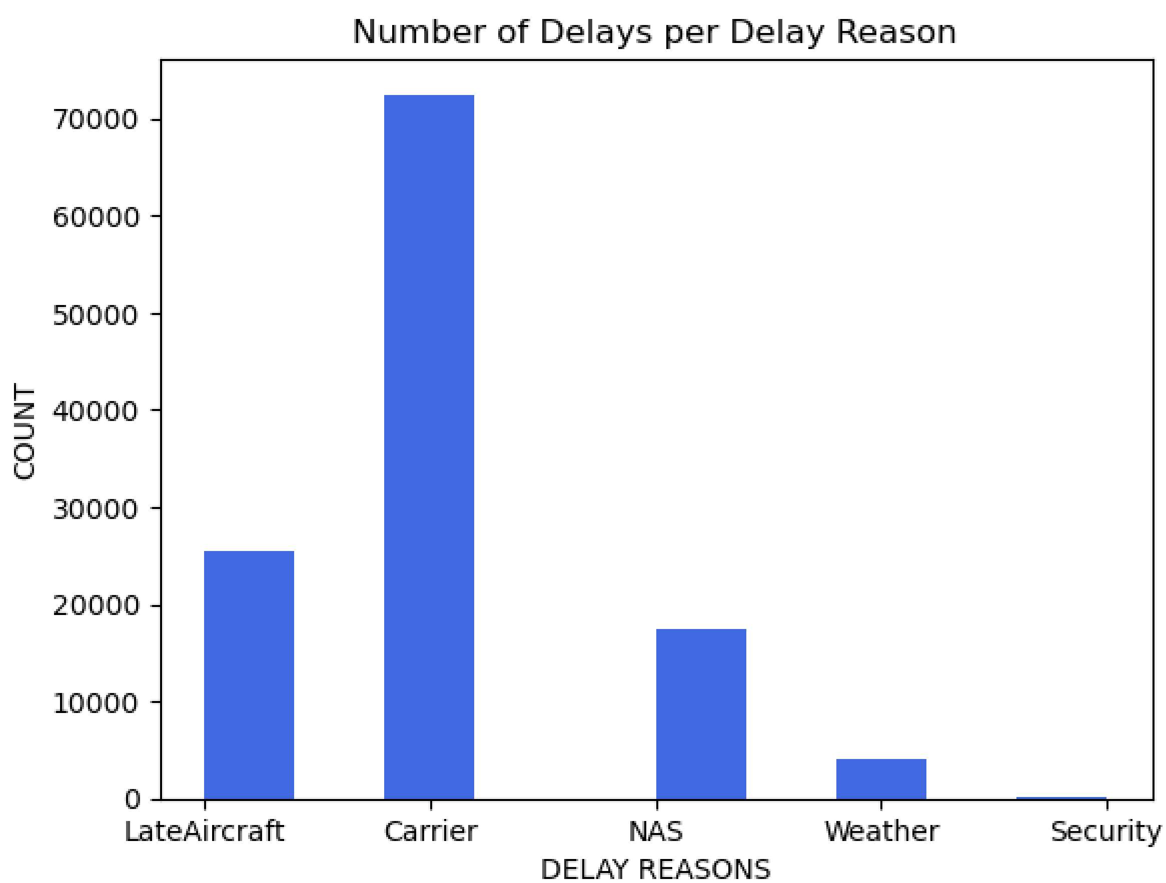
Histogram for Overall delays by delay reason

In [137…

```
plt.hist(x = flight_data_df[flight_data_df.STATUS=='Delayed'].DELAY_REASON, color = "roy
plt.xlabel('DELAY REASONS')
plt.ylabel('COUNT')

# displaying the title
plt.title("Number of Delays per Delay Reason")
```

Out[137]:

```
Text(0.5, 1.0, 'Number of Delays per Delay Reason')
```



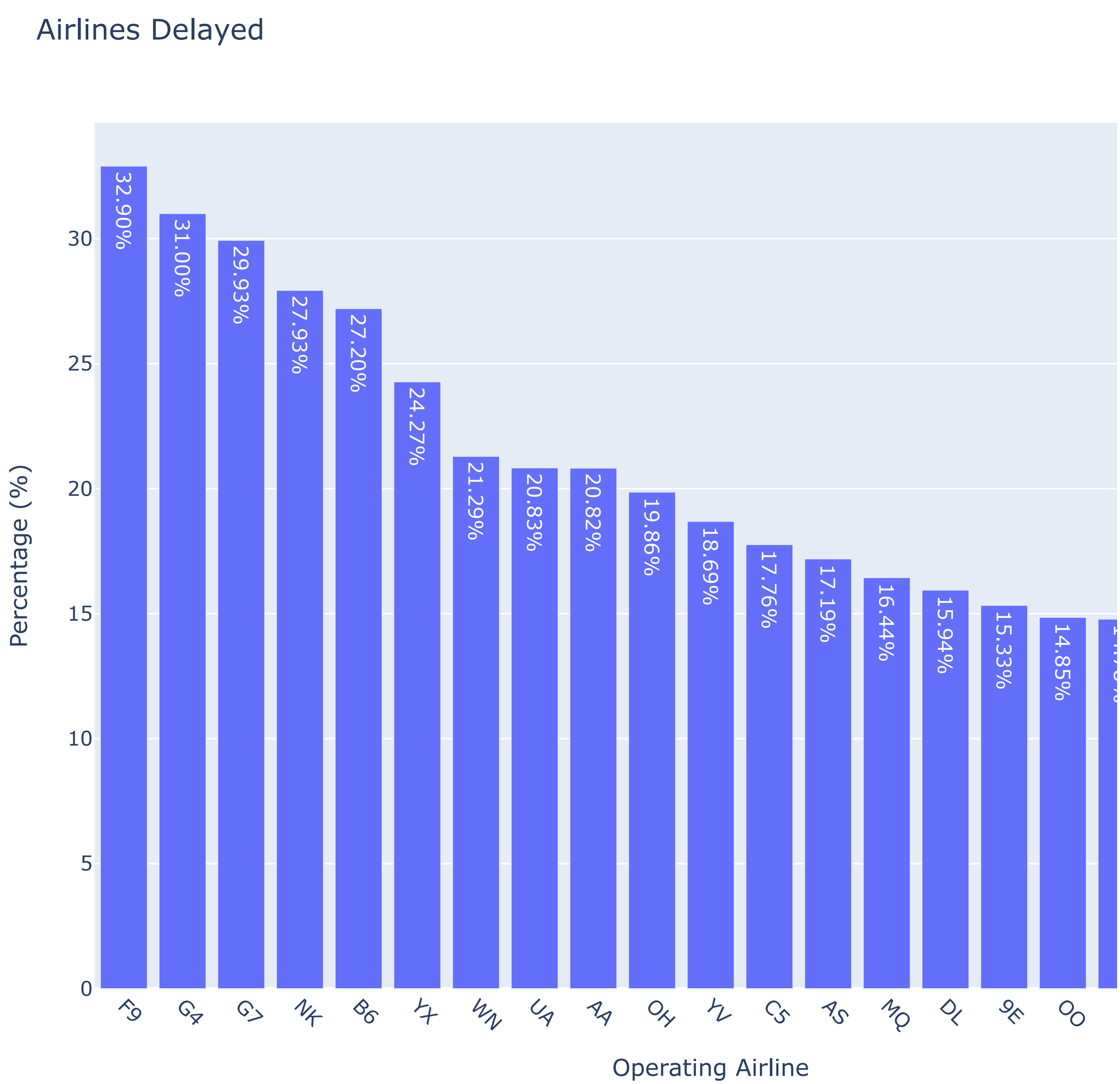
From the above chart, we can see most airline delays in May’22 was due to carrier delays followed by LateAircraft delays.

Bar chart for Delay Percentage per airline

In [149…

```
airline_delayed_performance = flight_status_df[flight_status_df.STATUS == 'Delayed'].sor

fig=px.bar(airline_delayed_performance,
           x=airline_delayed_performance.OP_UNIQUE_CARRIER,
           y=airline_delayed_performance.PERCENTAGE, title="Airlines Delayed",
           text=airline_delayed_performance.PERCENTAGE.apply(lambda x: '{0:1.2f}%'.forma
           labels=dict(OP_UNIQUE_CARRIER="Operating Airline", PERCENTAGE="Percentage (%)
fig.update_xaxes(tickangle=45)
```

```
fig.update_layout(autosize=False,width=900, height=700)
fig.show('notebook')
```



MILESTONE 2 - Data Preparation

Dataframe at this time has most of the required features.

In [65]:
```
flight_data_df.columns
```

Out[65]:
```
Index(['YEAR', 'QUARTER', 'MONTH', 'DAY_OF_MONTH', 'DAY_OF_WEEK', 'FL_DATE',
       'MKT_UNIQUE_CARRIER', 'OP_UNIQUE_CARRIER', 'ORIGIN_AIRPORT_ID',
       'ORIGIN', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_ABR', 'ORIGIN_STATE_NM',
       'ORIGIN_WAC', 'DEST_AIRPORT_ID', 'DEST', 'DEST_CITY_NAME',
       'DEST_STATE_ABR', 'DEST_STATE_NM', 'DEST_WAC', 'DEP_TIME', 'DEP_DELAY',
       'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON', 'TAXI_IN', 'ARR_TIME',
       'ARR_DELAY', 'CANCELLED', 'CANCELLATION_CODE', 'DIVERTED', 'AIR_TIME',
       'DISTANCE', 'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY',
       'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'CANCELLATION_REASON',
```

```
       'STATUS', 'ARR_DELAYED', 'DELAY_REASON'],
      dtype='object')
```

1. Drop any features that are not useful for your model building and explain why they are not useful.

Dropping null rows was performed in Milestone 1 - Data Transformation (Step 1). Some additional features that can be dropped are as follows.

i. Dropping the year, month, day, day of month as these details are in the FL_DATE feature which is in a DateTime format.

In [66]:
```
flight_data_df.drop(['YEAR', 'QUARTER', 'MONTH'], axis=1, inplace=True)
```

In [67]:
```
flight_data_df.head(5)
```

Out[67]:

	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	MKT_UNIQUE_CARRIER	OP_UNIQUE_CARRIER	ORIGIN_AIRPORT_ID
0	1	7	5/1/2022 12:00:00 AM	AA	AA	10140
1	1	7	5/1/2022 12:00:00 AM	AA	AA	10140
2	1	7	5/1/2022 12:00:00 AM	AA	AA	10140
3	1	7	5/1/2022 12:00:00 AM	AA	AA	10140
4	1	7	5/1/2022 12:00:00 AM	AA	AA	10140

5 rows × 39 columns

ii. Dropping Origin Airport ID, Destination Airport ID, Origin_WAC, DEST_WAC as these are not significant for this project.

In [68]:
```
flight_data_df.drop(['ORIGIN_WAC', 'DEST_WAC', 'ORIGIN_AIRPORT_ID', 'DEST_AIRPORT_ID'],
```

In [69]:
```
flight_data_df.columns
```

Out[69]:
```
Index(['DAY_OF_MONTH', 'DAY_OF_WEEK', 'FL_DATE', 'MKT_UNIQUE_CARRIER',
       'OP_UNIQUE_CARRIER', 'ORIGIN', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_ABR',
       'ORIGIN_STATE_NM', 'DEST', 'DEST_CITY_NAME', 'DEST_STATE_ABR',
       'DEST_STATE_NM', 'DEP_TIME', 'DEP_DELAY', 'TAXI_OUT', 'WHEELS_OFF',
       'WHEELS_ON', 'TAXI_IN', 'ARR_TIME', 'ARR_DELAY', 'CANCELLED',
       'CANCELLATION_CODE', 'DIVERTED', 'AIR_TIME', 'DISTANCE',
       'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
       'LATE_AIRCRAFT_DELAY', 'CANCELLATION_REASON', 'STATUS', 'ARR_DELAYED',
       'DELAY_REASON'],
      dtype='object')
```

2. Perform any data extraction/selection steps.

Most of the data extraction and selection steps are already performed in Milestone 1 before plotting. I will update this section should there be any additional extraction/selection steps required.

3. Transform features if necessary.

i. Replacing values in a column

I realised the below step is not required for this project. Hence, commenting it.

Day of week is mentioned as numbers, updating numbers to days of week.

flight_data_df.DAY_OF_WEEK = np.where(flight_data_df.DAY_OF_WEEK==1, 'Sunday', np.where(flight_data_df.DAY_OF_WEEK==2, 'Monday', np.where(flight_data_df.DAY_OF_WEEK==3, 'Tuesday', np.where(flight_data_df.DAY_OF_WEEK==4, 'Wednesday', np.where(flight_data_df.DAY_OF_WEEK==5, 'Thursday', np.where(flight_data_df.DAY_OF_WEEK==6, 'Friday', np.where(flight_data_df.DAY_OF_WEEK==7, 'Wednesday','')))))))
flight_data_df.groupby(['DAY_OF_WEEK'])['DAY_OF_WEEK'].count().sort_index()

Arrival and departure delays in the '*_New' column for flights departing/arriving earlier than schedule are updated to 0. For this project we are considering flights arriving 15 minutes or later as delayed. Updating arr_delay to 0 for 15 minutes or less.

In [70]:
```
flight_data_df.loc[flight_data_df.ARR_DELAY<=15, 'ARR_DELAY'] = 0
flight_data_df.ARR_DELAY.unique()
```

Out[70]:
```
array([   0.,    17.,    20., ..., 1830.,   658.,   651.])
```

4. Engineer new useful features.

Features such as DELAY_REASON,ARR_DELAYED, CANCELLATION_REASON and STATUS were engineered and created in Milestone 1, data transformation section. These features were required for plotting.

5. Deal with missing data (do not just drop rows or columns without justifying this).

In [71]:
```
flight_data_df_null = flight_data_df.isnull().sum() / len(flight_data_df) #Calculate % o
flight_data_df_null
```

Out[71]:
```
DAY_OF_MONTH           0.000000
DAY_OF_WEEK            0.000000
FL_DATE                0.000000
MKT_UNIQUE_CARRIER     0.000000
OP_UNIQUE_CARRIER      0.000000
ORIGIN                 0.000000
ORIGIN_CITY_NAME       0.000000
ORIGIN_STATE_ABR       0.000000
ORIGIN_STATE_NM        0.000000
DEST                   0.000000
DEST_CITY_NAME         0.000000
DEST_STATE_ABR         0.000000
DEST_STATE_NM          0.000000
DEP_TIME               0.016954
DEP_DELAY              0.000000
TAXI_OUT               0.017551
WHEELS_OFF             0.017551
WHEELS_ON              0.017902
TAXI_IN                0.017902
ARR_TIME               0.017902
ARR_DELAY              0.000000
CANCELLED              0.000000
CANCELLATION_CODE      0.000000
DIVERTED               0.000000
```

```
AIR_TIME               0.020256
DISTANCE               0.000000
CARRIER_DELAY          0.000000
WEATHER_DELAY          0.000000
NAS_DELAY              0.000000
SECURITY_DELAY         0.000000
LATE_AIRCRAFT_DELAY    0.000000
CANCELLATION_REASON    0.000000
STATUS                 0.000000
ARR_DELAYED            0.000000
DELAY_REASON           0.000000
dtype: float64
```

In [72]:
```
#Look for any features with over 40% missing data
missing_features = flight_data_df_null[flight_data_df_null > 0.40].index #Identify colum
missing_features
```

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

There are no features with missing data over 40%.

6. Create dummy variables if necessary.

At this point, I don't belive I need dummy variables. Will revisit this section based on outcomes from Model creations. For milestone 3, I needed to create dummy variables (IN [104])

Conclusion

It appears we have enough information at this time to apply different models on the dataframe to be able to predict a flights performance. From the different charts, we can also see the best performing carriers, airports with most delays for arrivals and departures and reasons for cancellations.

On futher analysis, it appears we do not need any external data at this time for the model prediction. I target on predicting which airline and/or airport are more likely to be delayed based on the arrival delay feature.

MILESTONE 3 - Model building and Evaluation

In [73]:
```
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score,confusion_matrix, c
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTETomek
from sklearn.model_selection import train_test_split, cross_val_score,KFold

from IPython.display import Image

import warnings
from pandas.errors import SettingWithCopyWarning
```

In [74]:
```
warnings.simplefilter(action="ignore", category=SettingWithCopyWarning)
```

In [75]:
```
flight_data_df.head(5)
```

Out[75]:

	DAY_OF_MONTH	DAY_OF_WEEK	FL_DATE	MKT_UNIQUE_CARRIER	OP_UNIQUE_CARRIER	ORIGIN	ORIGIN_CITY
0	1	7	5/1/2022 12:00:00 AM	AA	AA	ABQ	Albuquer
1	1	7	5/1/2022 12:00:00 AM	AA	AA	ABQ	Albuquer
2	1	7	5/1/2022 12:00:00 AM	AA	AA	ABQ	Albuquer
3	1	7	5/1/2022 12:00:00 AM	AA	AA	ABQ	Albuquer
4	1	7	5/1/2022 12:00:00 AM	AA	AA	ABQ	Albuquer

5 rows × 35 columns

In [76]:

```
flight_data_df.columns
```

Out[76]:

```
Index(['DAY_OF_MONTH', 'DAY_OF_WEEK', 'FL_DATE', 'MKT_UNIQUE_CARRIER',
       'OP_UNIQUE_CARRIER', 'ORIGIN', 'ORIGIN_CITY_NAME', 'ORIGIN_STATE_ABR',
       'ORIGIN_STATE_NM', 'DEST', 'DEST_CITY_NAME', 'DEST_STATE_ABR',
       'DEST_STATE_NM', 'DEP_TIME', 'DEP_DELAY', 'TAXI_OUT', 'WHEELS_OFF',
       'WHEELS_ON', 'TAXI_IN', 'ARR_TIME', 'ARR_DELAY', 'CANCELLED',
       'CANCELLATION_CODE', 'DIVERTED', 'AIR_TIME', 'DISTANCE',
       'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY',
       'LATE_AIRCRAFT_DELAY', 'CANCELLATION_REASON', 'STATUS', 'ARR_DELAYED',
       'DELAY_REASON'],
      dtype='object')
```

In [77]:

```
#Select columns for modeling
model_df = flight_data_df[(flight_data_df.STATUS=='On-Time')|(flight_data_df.STATUS=='De

#Intial Status was created for all types, cancelled, diverted, delayed and on-time. For
#to on-time and delayed, I'm dropping this column and recreating the same with 0's and 1
flight_model_df.drop(columns = ['STATUS'], axis = 1, inplace = True)
model_df.loc[model_df.ARR_DELAY <= 15, 'STATUS'] = 0
model_df.loc[model_df.ARR_DELAY > 15, 'STATUS'] = 1

flight_model_df = model_df[['DAY_OF_MONTH', 'DAY_OF_WEEK','DEP_DELAY', 'TAXI_OUT','TAXI_
                            'AIR_TIME', 'DISTANCE','FL_DATE', 'MKT_UNIQUE_CARRIER','OP_U
                            'ORIGIN','DEST','DEP_TIME','WHEELS_OFF','WHEELS_ON','STATUS'

flight_model_df.head(5)
```

Out[77]:

	DAY_OF_MONTH	DAY_OF_WEEK	DEP_DELAY	TAXI_OUT	TAXI_IN	ARR_TIME	ARR_DELAY	AIR_TIME	DISTANC
0	1	7	-5.0	12.0	25.0	932.0	0.0	73.0	569.
1	1	7	-9.0	11.0	22.0	1122.0	0.0	71.0	569.
2	1	7	0.0	10.0	11.0	1321.0	0.0	72.0	569.

3	1	7	-8.0	13.0	12.0	1539.0	0.0	72.0	569.
4	1	7	-5.0	9.0	16.0	1718.0	0.0	73.0	569.

In [78]:

```
#Check the data types for features to split into categorical and numerical fields
#for standaridizing and creating dummy variables
flight_model_df.dtypes
```

Out[78]:

```
DAY_OF_MONTH            int64
DAY_OF_WEEK             int64
DEP_DELAY             float64
TAXI_OUT              float64
TAXI_IN               float64
ARR_TIME              float64
ARR_DELAY             float64
AIR_TIME              float64
DISTANCE              float64
FL_DATE                object
MKT_UNIQUE_CARRIER     object
OP_UNIQUE_CARRIER      object
ORIGIN                 object
DEST                   object
DEP_TIME              float64
WHEELS_OFF            float64
WHEELS_ON             float64
STATUS                 object
dtype: object
```

In [133…

```
#Converting data types to avoid memory issues while executing the model fit.
cols=['TAXI_OUT', 'TAXI_IN', 'ARR_TIME','AIR_TIME', 'DISTANCE', 'DEP_TIME', 'WHEELS_OFF'
flight_model_df[cols] = flight_model_df[cols].astype('float16') #Converting float64 to f

flight_model_df['DAY_OF_MONTH'] = flight_model_df['DAY_OF_MONTH'].astype(np.uint8)  #Con
flight_model_df['DAY_OF_WEEK'] = flight_model_df['DAY_OF_WEEK'].astype(np.uint8)  #Conve

obj_cols = ['FL_DATE','MKT_UNIQUE_CARRIER','OP_UNIQUE_CARRIER','ORIGIN','DEST','STATUS']
flight_model_df[obj_cols] = flight_model_df[obj_cols].astype('category') # #Converting o
```

In [134…

```
#Filtering categorical and numeric fields
X_cat = flight_model_df[['FL_DATE', 'MKT_UNIQUE_CARRIER', 'OP_UNIQUE_CARRIER', 'ORIGIN',
X_num = flight_model_df.drop(['FL_DATE','MKT_UNIQUE_CARRIER','OP_UNIQUE_CARRIER','ORIGIN
```

In [102…

```
X_cat.columns, X_num.columns
```

Out[102]:

```
(Index(['FL_DATE_5/10/2022 12:00:00 AM', 'FL_DATE_5/11/2022 12:00:00 AM',
        'FL_DATE_5/12/2022 12:00:00 AM', 'FL_DATE_5/13/2022 12:00:00 AM',
        'FL_DATE_5/14/2022 12:00:00 AM', 'FL_DATE_5/15/2022 12:00:00 AM',
        'FL_DATE_5/16/2022 12:00:00 AM', 'FL_DATE_5/17/2022 12:00:00 AM',
        'FL_DATE_5/18/2022 12:00:00 AM', 'FL_DATE_5/19/2022 12:00:00 AM',
        ...
        'DEST_VEL', 'DEST_VLD', 'DEST_VPS', 'DEST_WRG', 'DEST_WYS', 'DEST_XNA',
        'DEST_XWA', 'DEST_YAK', 'DEST_YKM', 'DEST_YUM'],
       dtype='object', length=797),
 Index(['DAY_OF_MONTH', 'DAY_OF_WEEK', 'TAXI_OUT', 'TAXI_IN', 'ARR_TIME',
        'AIR_TIME', 'DISTANCE', 'DEP_TIME', 'WHEELS_OFF', 'WHEELS_ON'],
       dtype='object'))
```

In [104…

```
#Creating dummy variables for categorical columns
X_cat = pd.get_dummies(X_cat, drop_first=True)
```

In [105…
```
# Create standardizer
scaler = StandardScaler()

# Standardize features
scaler.fit(X_num)
X_scaled = scaler.transform(X_num)
X_scaled = pd.DataFrame(X_scaled, index=X_num.index, columns=X_num.columns)
```

In [106…
```
#Concatenate the categorical and scaled numeric fields and set the features
X = pd.concat([X_scaled, X_cat], axis=1)
X.isna().sum()
```

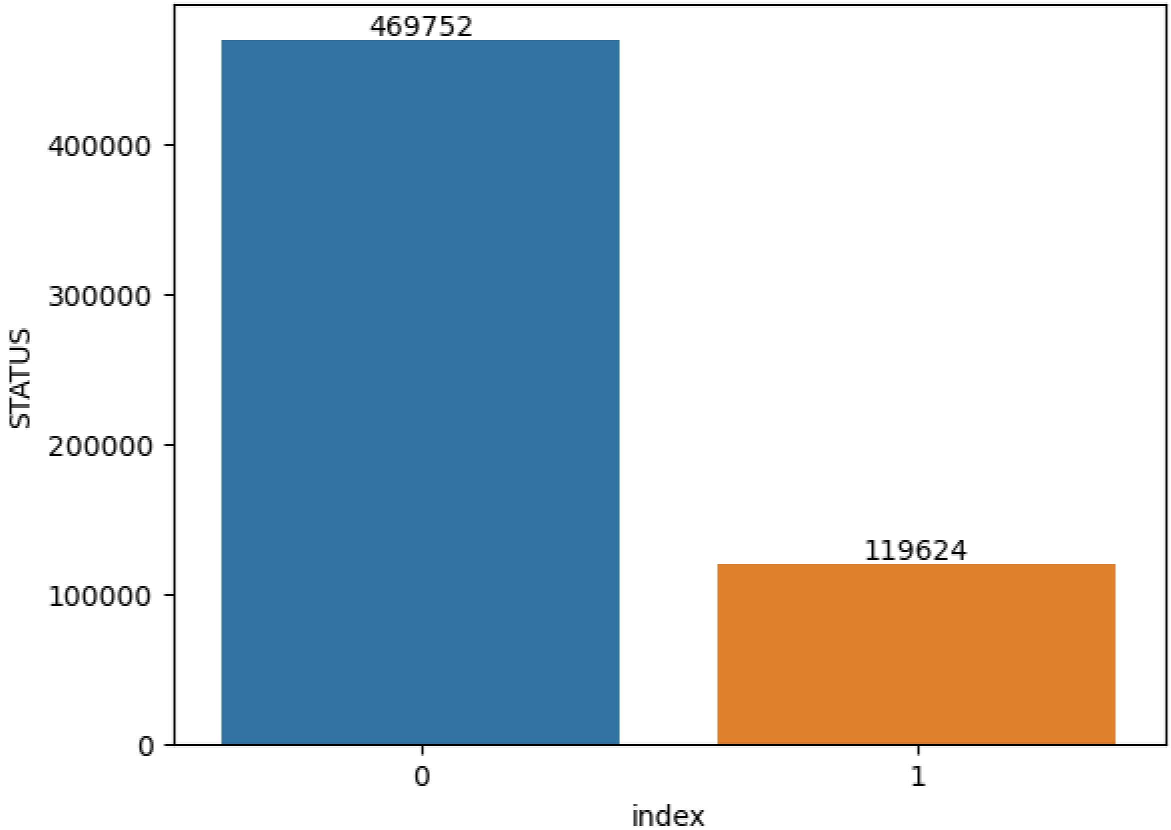
Out[106]:
```
DAY_OF_MONTH    0
DAY_OF_WEEK     0
TAXI_OUT        0
TAXI_IN         0
ARR_TIME        0
               ..
DEST_XNA        0
DEST_XWA        0
DEST_YAK        0
DEST_YKM        0
DEST_YUM        0
Length: 807, dtype: int64
```

In [107…
```
#Set the training data in Y
Y = flight_model_df['STATUS'].astype('int')
```

In [108…
```
#Adding this step to clear memory, to avoid memory issues suring execution
import gc

gc.collect()
```

Out[108]:
```
1465
```

In [109…
```
#Check the balance of test and train data
xx = flight_model_df['STATUS'].value_counts().reset_index()
ax = sns.barplot(x="index",y="STATUS",data=xx)
for i in ax.containers:
    ax.bar_label(i,)
```



We can clearly see that the data is not balanced. The number of delayed flights in the dataset are very low in comparison to the flights on-time. Building models with this data could give inaccurate results.

Testing a model with unbalanced data with RandomForestClassifier

In [110…
```
# Split dataset into random train and test subsets:
x_train_ub, x_test_ub, y_train_ub, y_test_ub = train_test_split(X,Y,test_size = 0.2)
```

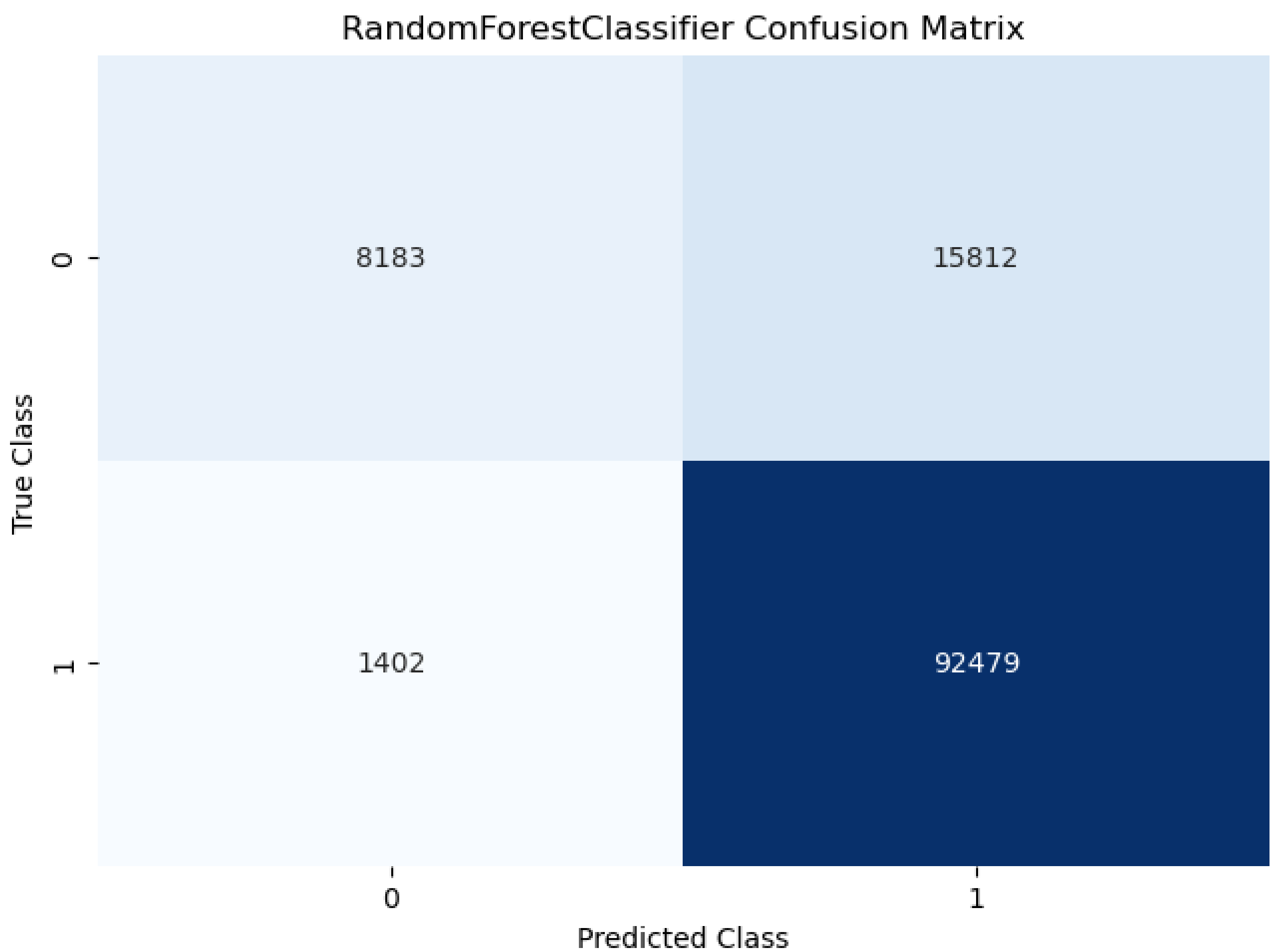
In [135…
```
#Use RandomForestClassifier to fit the unbalanced data
rfc = RandomForestClassifier()
rfc_model = rfc.fit(x_train_ub,y_train_ub)
#Predict y data with classifier:
y_pred_ub = rfc_model.predict(x_test_ub)

#Print model results
print(classification_report(y_test_ub, y_pred_ub))
print(confusion_matrix(y_test_ub, y_pred_ub))
print(f'ROC-AUC score : {roc_auc_score(y_test_ub, y_pred_ub)}')
print(f'Accuracy score : {accuracy_score(y_test_ub, y_pred_ub)}')
```

```
              precision    recall  f1-score   support

           0       0.85      0.99      0.91     93881
           1       0.85      0.34      0.49     23995

    accuracy                           0.85    117876
   macro avg       0.85      0.66      0.70    117876
weighted avg       0.85      0.85      0.83    117876

[[92479  1402]
 [15812  8183]]
ROC-AUC score : 0.6630477909748879
Accuracy score : 0.8539651837524178
```

In [136…
```
#Build the confusion matrix
matrix = confusion_matrix(y_test_ub, y_pred_ub, labels=[1,0])
```

```
print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[ 8183 15812]
 [ 1402 92479]]
```



Data looks unbalanced for delayed and on-time flights, as expected. Using SMOTE to balance data and re-build the model.

Smote oversampling for imbalanced classification

In [111…

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 589376 entries, 0 to 602949
Columns: 807 entries, DAY_OF_MONTH to DEST_YUM
dtypes: float64(10), uint8(797)
memory usage: 497.4 MB
```

In [112…

```
smote = SMOTE()
x, y = smote.fit_resample(X, Y)
```

In [113…

```
smote_df = pd.concat([pd.DataFrame(x), pd.DataFrame(y)], axis=1)
smote_df.head(5)
```

Out[113]:

	DAY_OF_MONTH	DAY_OF_WEEK	TAXI_OUT	TAXI_IN	ARR_TIME	AIR_TIME	DISTANCE	DEP_TIME	WHEELS_O
0	-1.667176	1.493434	-0.521921	2.579922	-0.961536	-0.533474	-0.377751	-1.363487	-1.3770
1	-1.667176	1.493434	-0.634723	2.127350	-0.611151	-0.562260	-0.377751	-0.975586	-0.9928
2	-1.667176	1.493434	-0.747524	0.467917	-0.244170	-0.547867	-0.377751	-0.559977	-0.5810
3	-1.667176	1.493434	-0.409119	0.618774	0.157850	-0.547867	-0.377751	-0.057289	-0.0747
4	-1.667176	1.493434	-0.860326	1.222204	0.487949	-0.533474	-0.377751	0.215825	0.1892

5 rows × 808 columns

In [114…

```
#Split the smote (balanced) data into random train and test subsets:
x_train_sm, x_test_sm, y_train_sm, y_test_sm = train_test_split(x,y,test_size = 0.2)
```

In [115…

```
print(x.shape)
print(y.shape)
print(x_train_sm.shape)
print(y_train_sm.shape)
print(x_test_sm.shape)
print(y_test_sm.shape )
```

```
(939504, 807)
(939504,)
(751603, 807)
(751603,)
(187901, 807)
(187901,)
```

RandomForestClassifier

In [116…

```
# Use the RandomForestClassifier to fit balanced data
rfc = RandomForestClassifier()
rfc_model = rfc.fit(x_train_sm,y_train_sm)
```

In [117…

```
#Predict y data with classifier:
y_pred_rfc = rfc_model.predict(x_test_sm)

print(classification_report(y_test_sm, y_pred_rfc))
print(confusion_matrix(y_test_sm, y_pred_rfc))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_rfc)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_rfc)}')
```

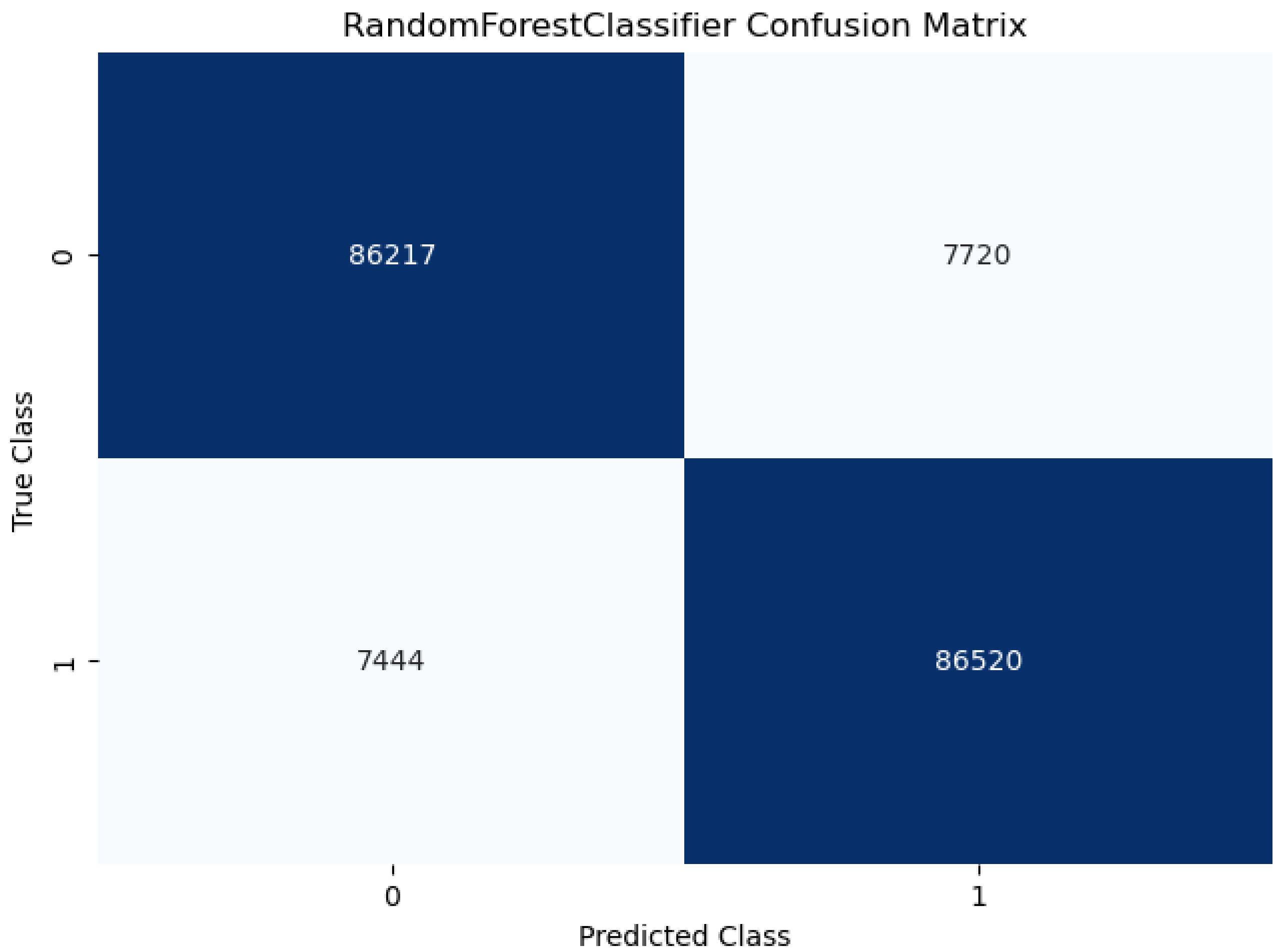
```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92     93964
           1       0.92      0.92      0.92     93937

    accuracy                           0.92    187901
   macro avg       0.92      0.92      0.92    187901
weighted avg       0.92      0.92      0.92    187901

[[86520  7444]
 [ 7720 86217]]
ROC-AUC score : 0.9192977154337492
Accuracy score : 0.9192979281642993
```

In [118…

```
#Build the confusion matrix
matrix = confusion_matrix(y_test_sm, y_pred_rfc, labels=[1,0])

print(matrix)
```

```
# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[86217  7720]
 [ 7444 86520]]
```



DecisionTreeClassifier

In [119…

```
#save the feature name and target variables
feature_names = x.columns
labels = y.unique()

#split the dataset into train and test
x_train_dt, x_test_dt, y_train_dt, y_test_dt = train_test_split(x,y,
                                                                  test_size = 0.4,
                                                                  random_state = 42)
# Use the DecisionTreeClassifier to fit data
clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
clf.fit(x_train_dt, y_train_dt)
```

Out[119]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=3, random_state=42)
```

In [120…

```
#Predict y data with classifier:
y_pred_dtc = clf.predict(x_test_sm)
```

```python
#Print results
print(classification_report(y_test_sm, y_pred_dtc))
print(confusion_matrix(y_test_sm, y_pred_dtc))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_dtc)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_dtc)}')
```

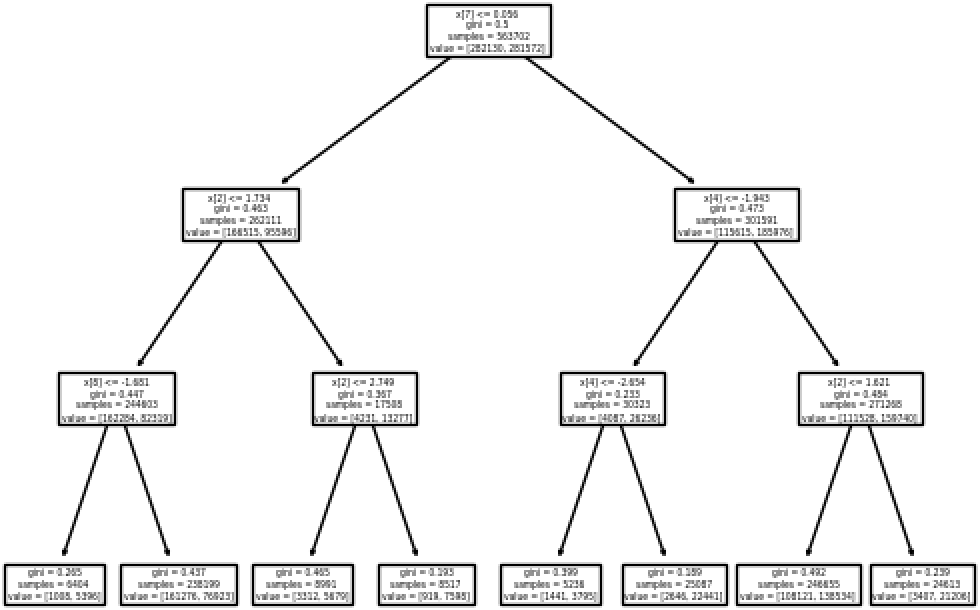
```
              precision    recall  f1-score   support

           0       0.68      0.57      0.62     93964
           1       0.63      0.73      0.67     93937

    accuracy                           0.65    187901
   macro avg       0.65      0.65      0.65    187901
weighted avg       0.65      0.65      0.65    187901

[[53700 40264]
 [25802 68135]]
ROC-AUC score : 0.6484110074965592
Accuracy score : 0.6483999552956078
```

In [121…

```python
# Draw graph
tree.plot_tree(clf)
```

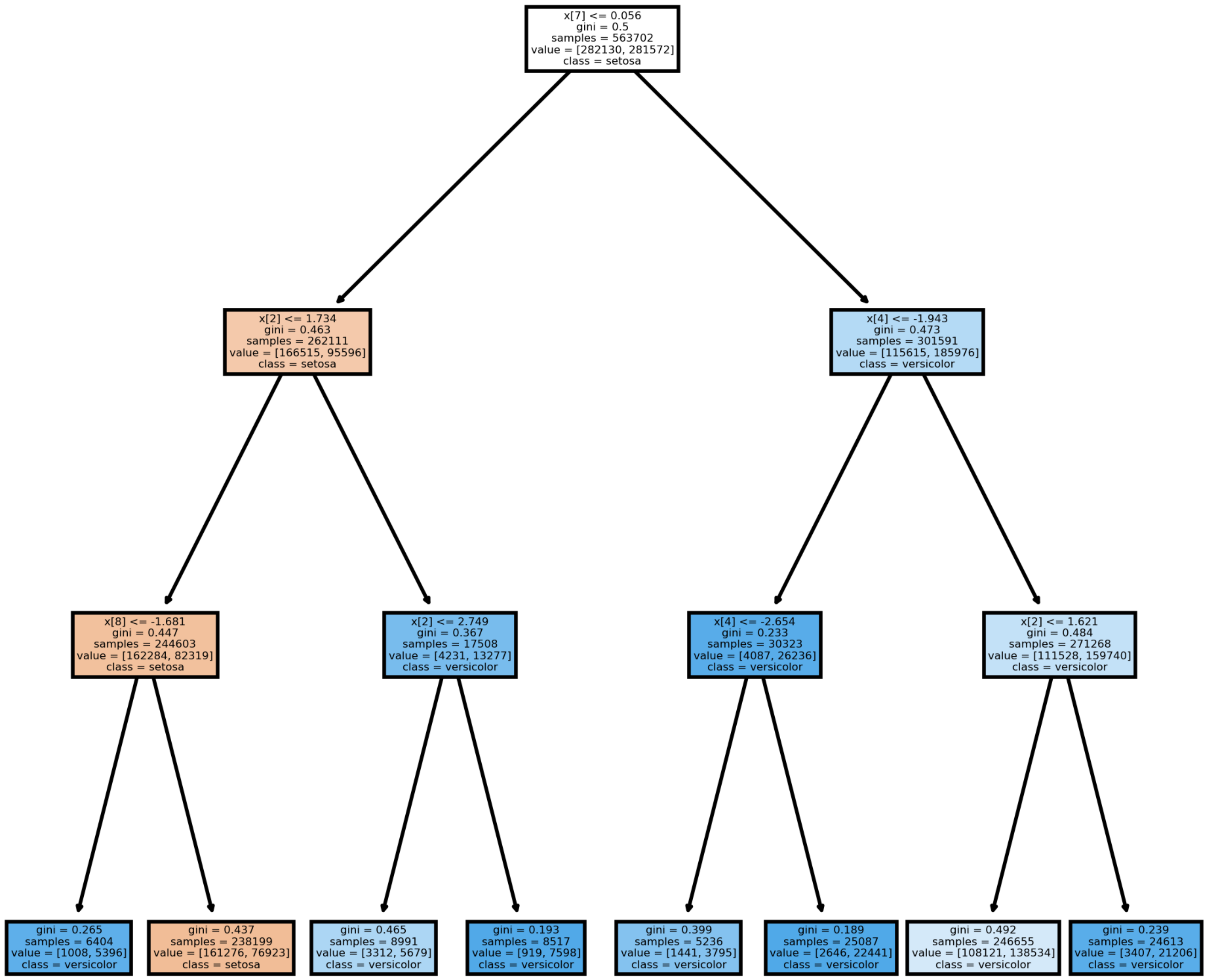
Out[121]:

```
[Text(0.5, 0.875, 'x[7] <= 0.056\ngini = 0.5\nsamples = 563702\nvalue = [282130, 28157
2]'),
 Text(0.25, 0.625, 'x[2] <= 1.734\ngini = 0.463\nsamples = 262111\nvalue = [166515, 9559
6]'),
 Text(0.125, 0.375, 'x[8] <= -1.681\ngini = 0.447\nsamples = 244603\nvalue = [162284, 82
319]'),
 Text(0.0625, 0.125, 'gini = 0.265\nsamples = 6404\nvalue = [1008, 5396]'),
 Text(0.1875, 0.125, 'gini = 0.437\nsamples = 238199\nvalue = [161276, 76923]'),
 Text(0.375, 0.375, 'x[2] <= 2.749\ngini = 0.367\nsamples = 17508\nvalue = [4231, 1327
7]'),
 Text(0.3125, 0.125, 'gini = 0.465\nsamples = 8991\nvalue = [3312, 5679]'),
 Text(0.4375, 0.125, 'gini = 0.193\nsamples = 8517\nvalue = [919, 7598]'),
 Text(0.75, 0.625, 'x[4] <= -1.943\ngini = 0.473\nsamples = 301591\nvalue = [115615, 185
976]'),
 Text(0.625, 0.375, 'x[4] <= -2.654\ngini = 0.233\nsamples = 30323\nvalue = [4087, 2623
6]'),
 Text(0.5625, 0.125, 'gini = 0.399\nsamples = 5236\nvalue = [1441, 3795]'),
 Text(0.6875, 0.125, 'gini = 0.189\nsamples = 25087\nvalue = [2646, 22441]'),
 Text(0.875, 0.375, 'x[2] <= 1.621\ngini = 0.484\nsamples = 271268\nvalue = [111528, 159
740]'),
 Text(0.8125, 0.125, 'gini = 0.492\nsamples = 246655\nvalue = [108121, 138534]'),
 Text(0.9375, 0.125, 'gini = 0.239\nsamples = 24613\nvalue = [3407, 21206]')]
```



In [122…

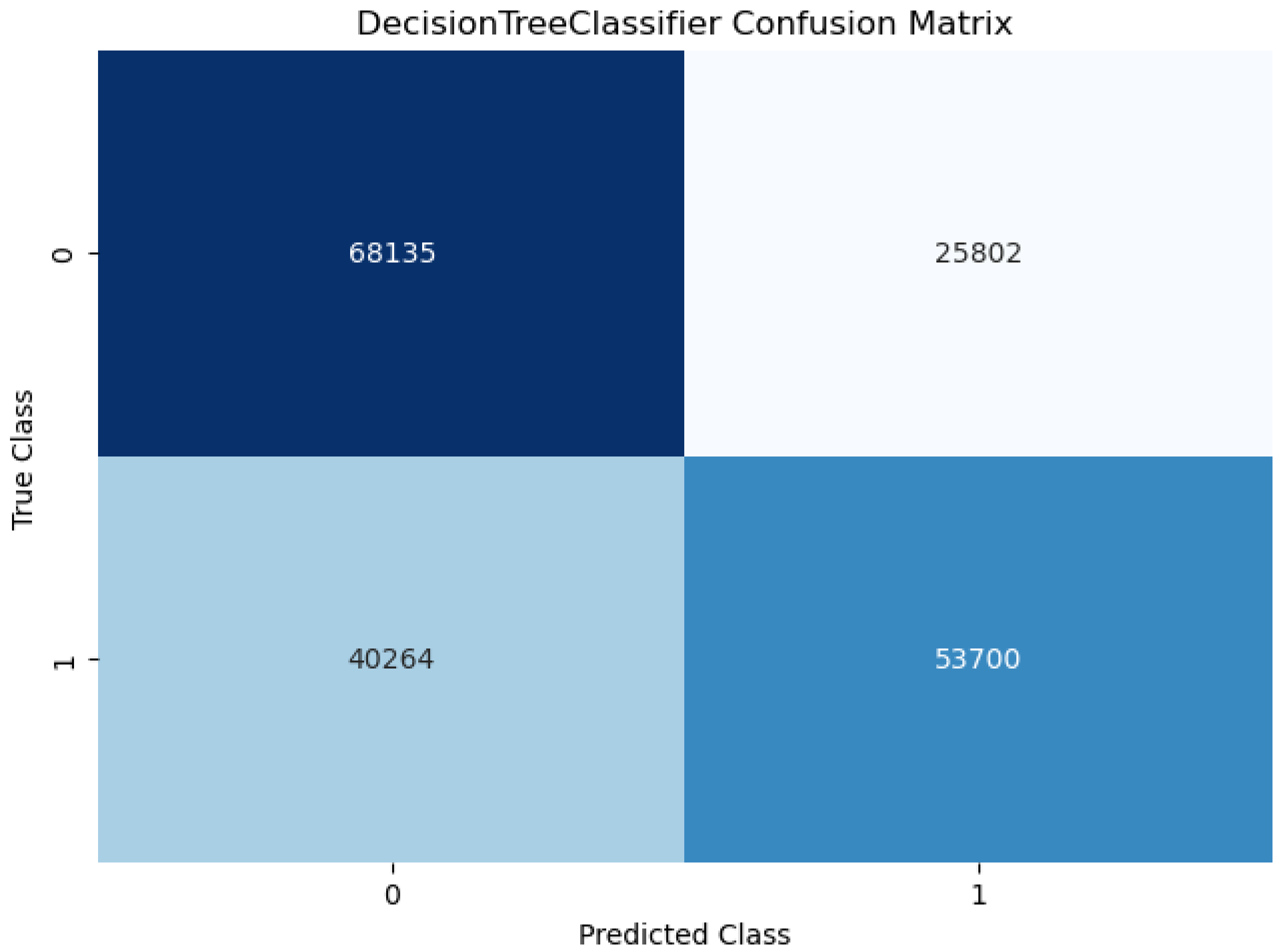
```
# Draw graph

cn=['setosa', 'versicolor', 'virginica']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (6,6), dpi=400)
tree.plot_tree(clf,
               class_names=cn,
               filled = True);
fig.savefig('dtc.png')
```



In [123…

```
#Build the confusion matrix
matrix = confusion_matrix(y_test_sm, y_pred_dtc, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("DecisionTreeClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[68135 25802]
 [40264 53700]]
```



LogisticRegression

In [124…

```
# Use the LogisticRegression to fit data:
lr_model = LogisticRegression(solver='liblinear')
model = lr_model.fit(x_train_sm, y_train_sm)
```

In [125…

```
#Predict y data with classifier:
y_pred_lr = model.predict(x_test_sm)

#Pritn results
print(classification_report(y_test_sm, y_pred_lr))
print(confusion_matrix(y_test_sm, y_pred_lr))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_lr)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_lr)}')
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89     93964
           1       0.95      0.80      0.87     93937

    accuracy                           0.88    187901
   macro avg       0.89      0.88      0.88    187901
weighted avg       0.89      0.88      0.88    187901

[[90337  3627]
 [18857 75080]]
ROC-AUC score : 0.8803295942866582
Accuracy score : 0.8803412435271766
```

In [126…

```
# Cross-validate model using accuracy
cross_val_score(lr_model, x, y, scoring="accuracy")
```

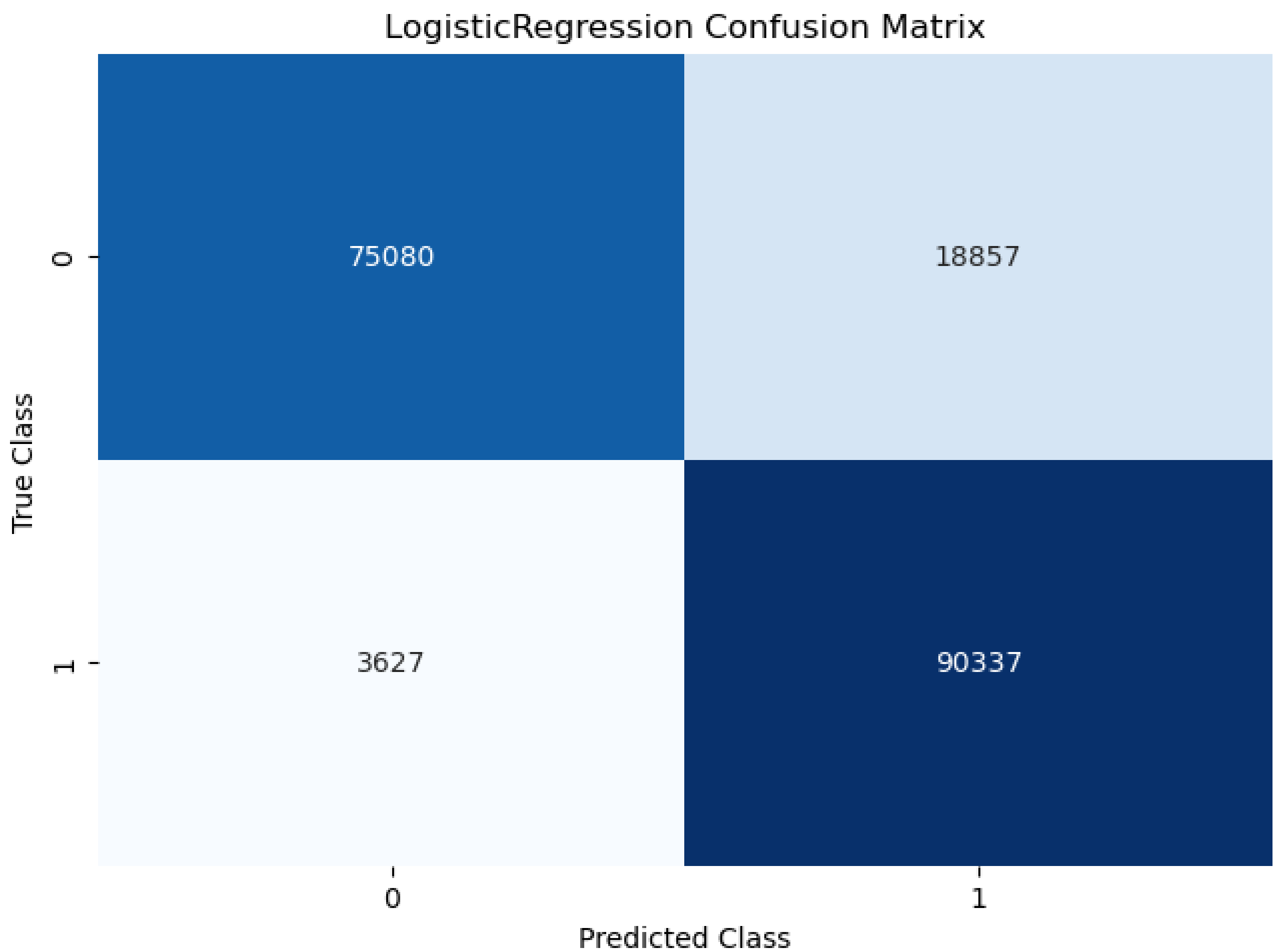
Out[126]:

```
array([0.14782252, 0.36521892, 0.50728841, 0.55988526, 0.49660458])
```

In [127…

```python
#Build the confusion matrix
matrix = confusion_matrix(y_test_sm, y_pred_lr, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("LogisticRegression Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[75080 18857]
 [ 3627 90337]]
```



from yellowbrick.classifier import DiscriminationThreshold lr_visualizer = DiscriminationThreshold(lr_model) lr_visualizer.fit(x,y) # Fit the data to the visualizer lr_visualizer.show() # Finalize and render the figure

Using class_weight='balaced' to check if the imbalance get's any better.

In [128…

```python
logit = LogisticRegression( solver='liblinear', class_weight='balanced')
model_logit = logit.fit(x_train_sm, y_train_sm)

y_pred_logit = model_logit.predict(x_test_sm)

print(classification_report(y_test_sm, y_pred_logit))
print(confusion_matrix(y_test_sm, y_pred_logit))
print(f'ROC-AUC score : {roc_auc_score(y_test_sm, y_pred_logit)}')
print(f'Accuracy score : {accuracy_score(y_test_sm, y_pred_logit)}')

# Cross-validate model using accuracy
cross_val_score(logit, x, y, scoring="accuracy")
```

```
              precision    recall  f1-score   support

           0       0.83      0.96      0.89     93964
           1       0.95      0.80      0.87     93937

    accuracy                           0.88    187901
   macro avg       0.89      0.88      0.88    187901
weighted avg       0.89      0.88      0.88    187901

[[90335  3629]
 [18857 75080]]
ROC-AUC score : 0.8803189519129831
Accuracy score : 0.8803305996242702
```

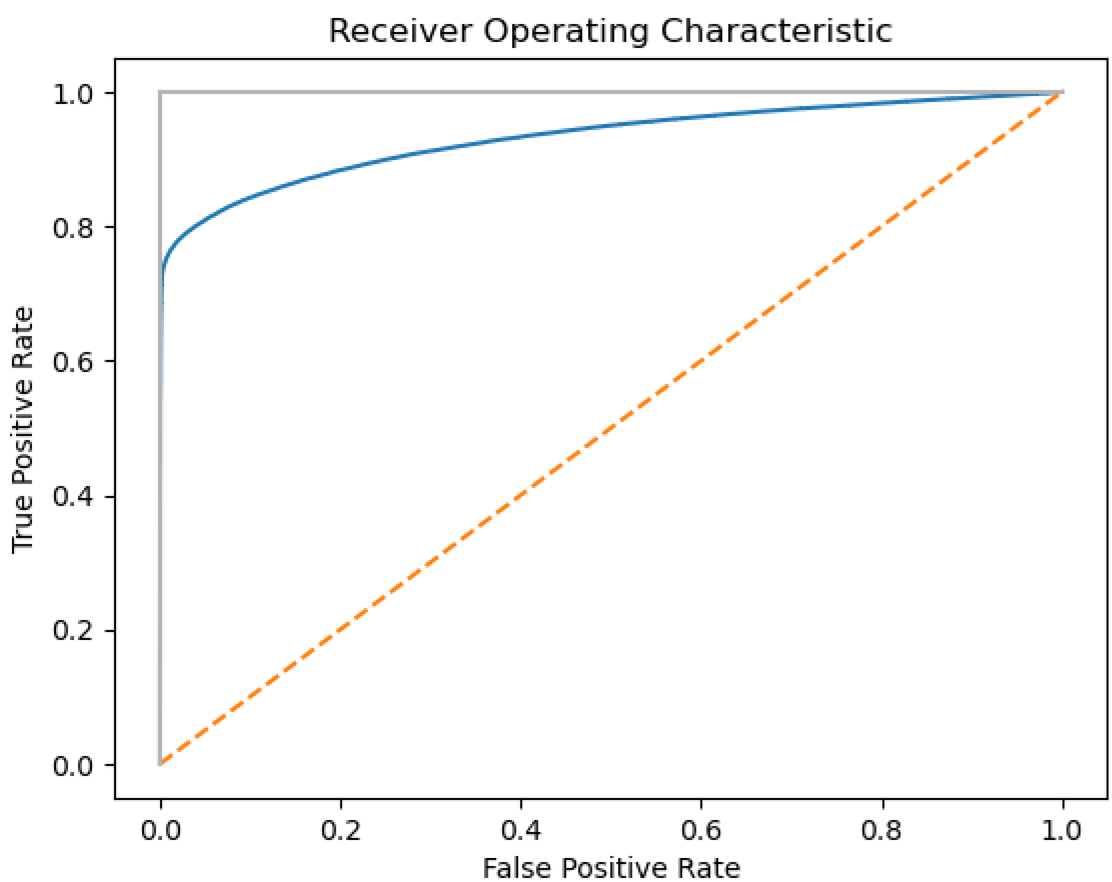
Out[128]:

```
array([0.1478172 , 0.3652136 , 0.50728841, 0.55989058, 0.49660458])
```

Adding the class_weight = balanced has no impact on the model outcome.

In [129…

```
logit.fit(x_train_sm, y_train_sm)

# Get predicted probabilities
target_probabilities = logit.predict_proba(x_test_sm)[:,1]
# Create true and false positive rates
false_positive_rate, true_positive_rate, threshold = roc_curve(y_test_sm,target_probabil

# Plot ROC curve
plt.title("Receiver Operating Characteristic")
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.show()
```



In [140…

```
print("Threshold:", threshold[116])
print("True Positive Rate:", true_positive_rate[116])
```

```
print("False Positive Rate:", false_positive_rate[116])
```

```
Threshold: 0.9696407718809488
True Positive Rate: 0.5930676943057581
False Positive Rate: 0.0006066152994763952
```

Conclusion

05/21/2023 - Week10

The following 4 models were built and evaluated with theBTS airline performance data

- LogisticRegression
- RandomForestClassifier
- DecisionTreeClassifier

Due to imbalance in the data I am getting a perfect accuracy score for all models. I tried using different datasets from the BTS website (May, June, entire 2022 data from Jan through Nov, and the current execution with Dec'22 data), and I am getting the same accuracy score, precision, recall and f1-score for all datasets from the website.

06/01/2023 - Week 12 Final Project

After removing the delay time parameters, ARR_DELAY,DEP_DELAY,CARRIER_DELAY,WEATHER_DELAY,NAS_DELAY,SECURITY_DELAY and LATE_AIRCRAFT_DELAY there has been a significant change in the models.

Following are the observations:

Summary:

With Random Forest Classifier, after evaluating the training model we got an accuracy score of 91.92%.

With Decision Tree Classifier, we got an accuracy of 64.84%.

With Logistic Regression, the accuracy score of the model is 88.03%.

Clearly, the RandomForest Classifier is the best model.