Configuration Manual

MSc Research Project
Data Analytics

Enwere Chibuike Kenneth

StudentID:X18178090

School of Computing

National College of Ireland

Supervisor: Dr. Cristina Muntean

National College of Ireland
Project Submission Sheet School of Computing

Student Name:	Enwere Chibuike Kenneth
Student ID:	x18178090
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Dr. Cristina Muntean
Submission Due Date:	12/12/2019
Project Title:	Configuration Manual
Word Count:	2201
Page Count:	24

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	

Penalty Applied (if applicable):	

Configuration Manual

Enwere Chibuike Kenneth x18178090

1 Introduction

In this configuration manual a detailed procedure used in achieving "Predicting Flight Arrival delay Reduction For Delta Airlines " is explained. This includes comprehensive instructions on the requirements (hardware and software), source of the data, environment specification and modelling techniques used

2 System Specification

This research has been carried out a windows environment using DELL Inspiron 14 5000 with the system specification is shown below.

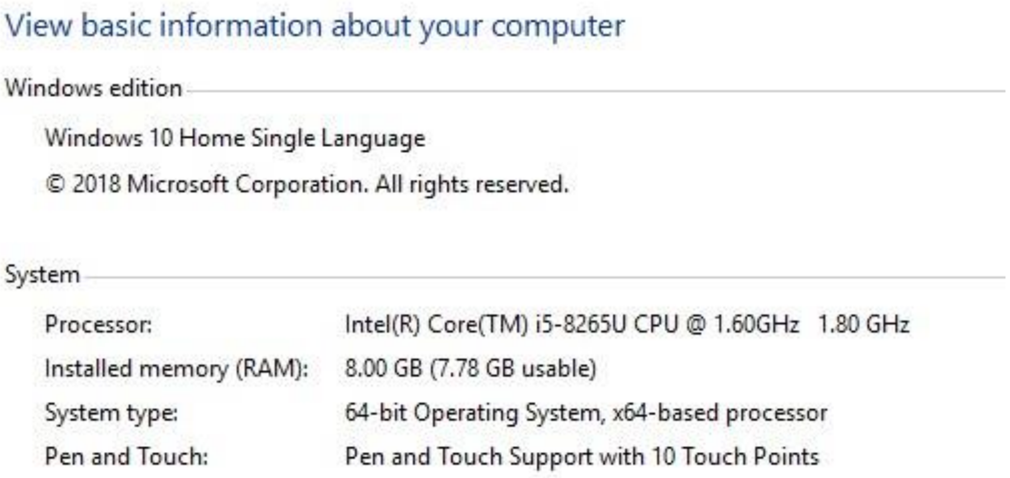

Figure 1: System specification

3 Data Collection

Data was collected from a primary source, this was the Bureau of transport statistics (BTS) on the airline on-time statistics for Flight arrival [1]

[1] https:// https://transtats.bts.gov/ONTIME/Arrivals.aspx

4. Data Storage and Preparation

- The dataset that was sourced from BTS for flight arrival delay was a structured data in a CSV format and was directly downloaded and stored on the system.

- This data was saved in C:\Users\kenne\OneDrive\Desktop\data

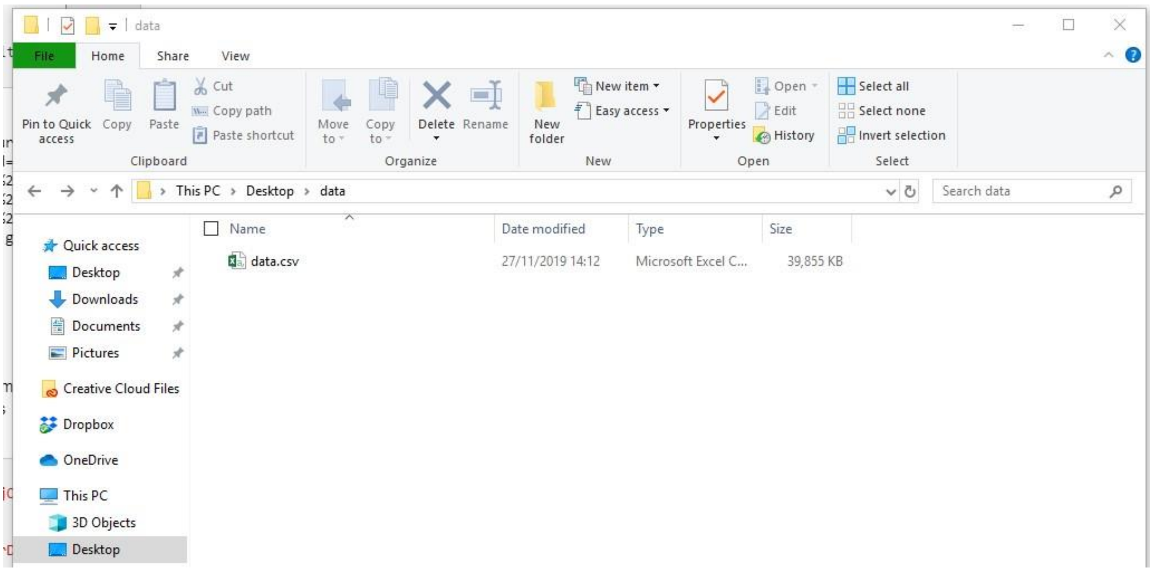

Figure 1: Location where dataset is stored

5. Download and Installation of Anaconda

Anaconda is a package manager, an environment manager, and Python distribution that contains a collection of many open source packages. With packages such as (numpy,scikit-learn , scipy and pandas). Below are the steps to download anaconda

- Go to https://www.anaconda.com/distribution/

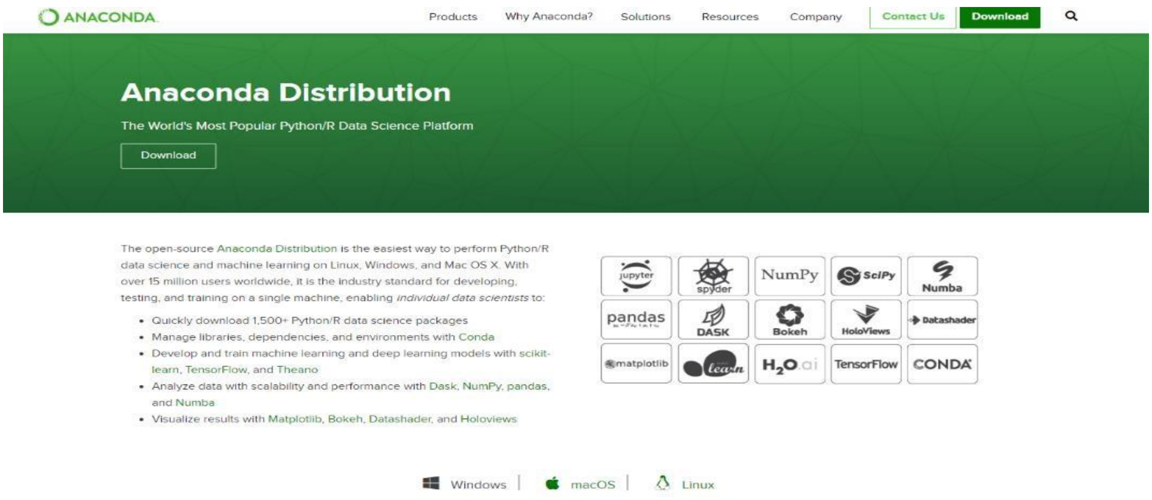

Figure 2: How to download anaconda

- Locate where to download it and

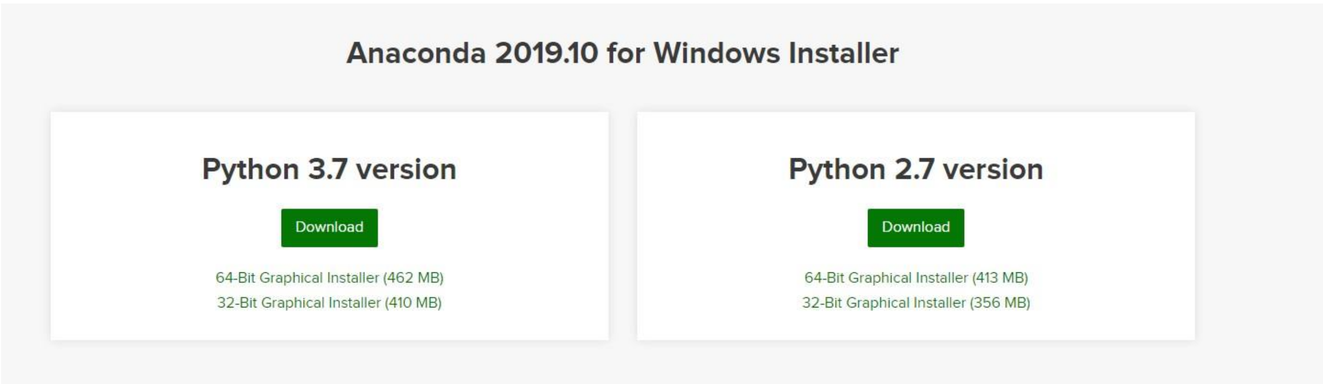

Figure 3: Version of Anaconda

- Download Python version 3.7 below

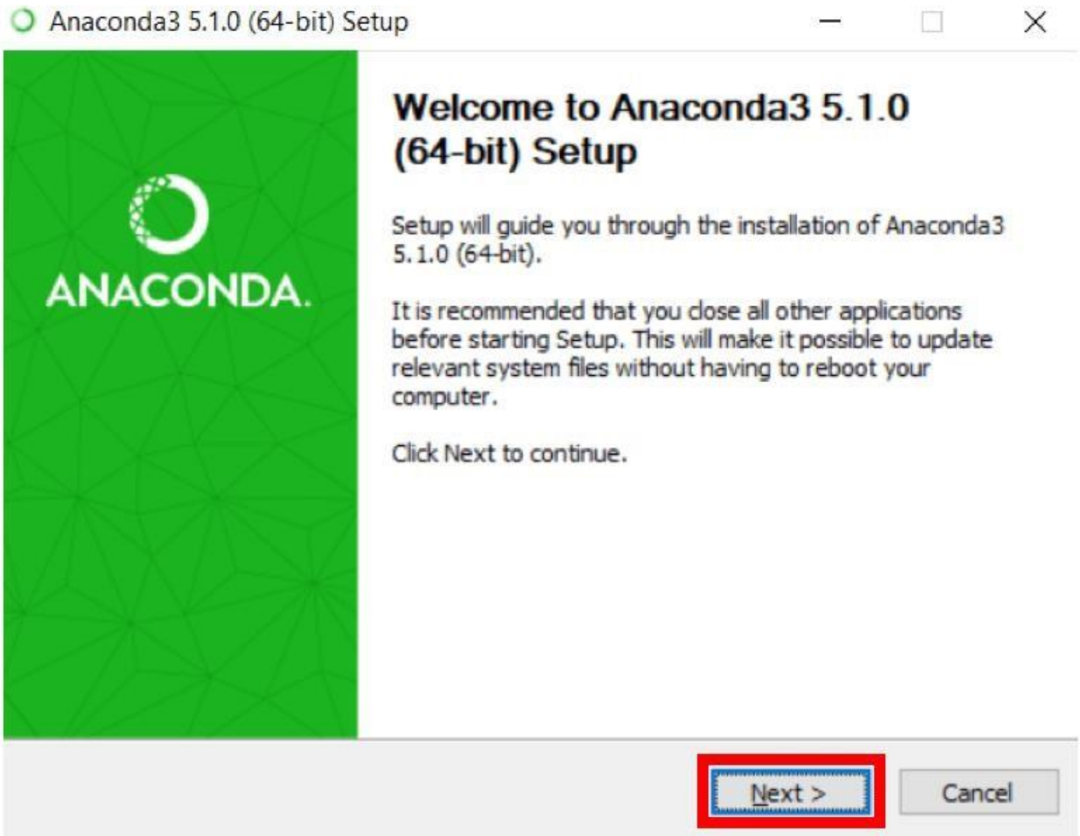

Figure 4: Anaconda setup process

- You can use either of the two approach I went with the recommended approach .The recommended approach enables youto use Anaconda Navigator or the Anaconda Command Prompt (located in the Start Menu under "Anaconda")

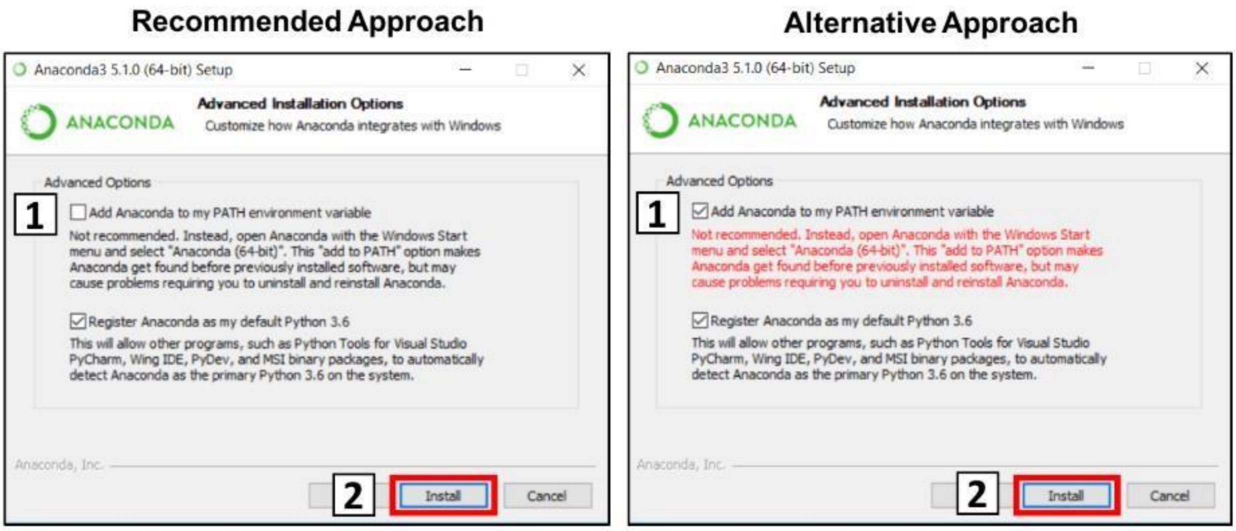

Figure 4: Recommended Approach

- Anaconda installation is completed

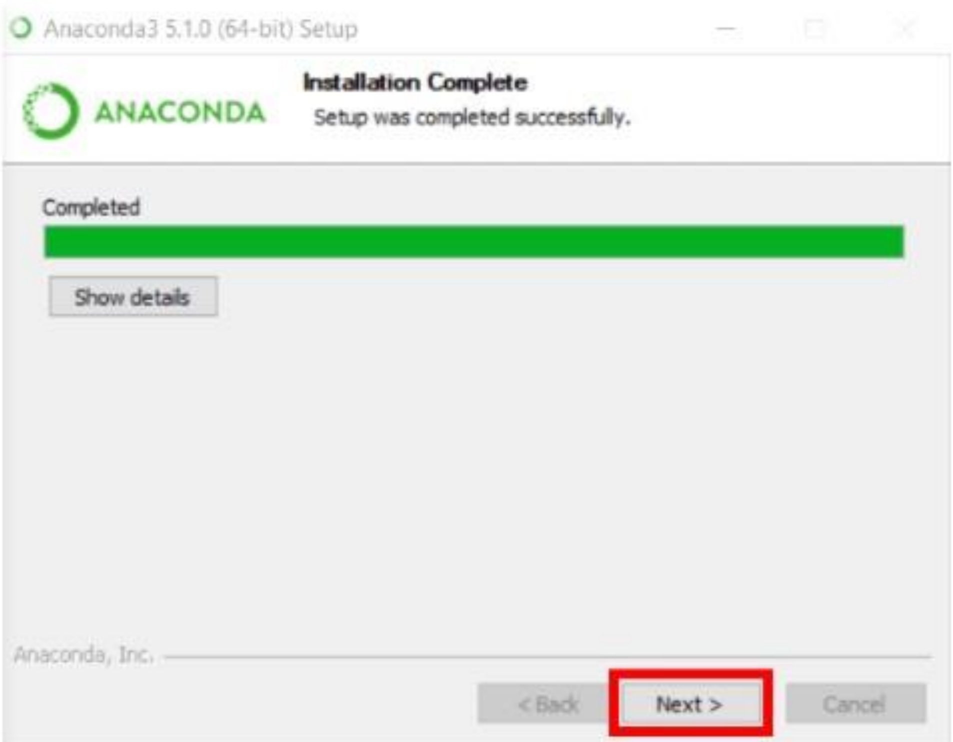

Figure 6: Anaconda setup completion

6. Preparing Data For Analysis

Here we load the entire dataset, performing cleaning and create data visualizations. The aim of the project is to predict arrival delays for the months of summer.

Step 1: Import necessary libraries to be used in the analysis

```
In [1]:  # Importing required libraries.
         import pandas as pd
         import numpy as np
         import seaborn as sns
         # visualisation libraries
         import matplotlib.pyplot as plt
         %matplotlib inline
         sns.set(color_codes=True)
```

Figure 7: Code to import libraries

Step 2: Load Dataset

```
In [2]:  # Load Dataset
         # file_loc = 'data/data.csv'
         file_loc = 'C:\\Users\\kenne\\OneDrive\\Desktop\\data\\data.csv'
         df = pd.read_csv(file_loc)  # This will load our dataset on pandas dataframe
```

Figure 8: Code to load dataset

7 Data Cleaning and Feature Extraction

Cleaning: We need to remove columns which will not be of any use in our analysis. All the irrelevant columns needs to be removed as it might reduce our models accuracy. Also it is important to extract required features in order to predict arrival delay more accurately.

Feature Extraction: It leads to accuracy improvements and speedup in training. It is very important to get the right set of features to get good prediction of our model. It aims to reduce no of features from dataset by extracting relevant information and discarding irrelevant ones.

Steps for data cleaning:

- Removing irrelevant columns: Irrelevant data are those that are not actually needed, and don't fit under the context of the problem we're trying to solve.
- Removing duplicate data: Duplicates are data points that are repeated in your dataset.
- Fixing null values by either discarding or relevant substitution
- Type conversion: Changing columns to valid datatypes. Make sure numbers are stored as numerical data types. Categorical values can be converted into and from numbers if needed.
- Syntax errors: Removing whitespaces or fixing typos.

In [4]:

```
######################### REMOVING IRRELEVANT COLUMNS #########################

# We need to remove irrelevant columns. It is important to remove data which is not providing any

# YEAR - We are using 2017 data so there is no point considering it.
# OP_UNIQUE_CARRIER, OP_CARRIER_AIRLINE_ID, OP_CARRIER_FL_NUM - We are considered Delta Airlines so these fields can be remo
# ORIGIN_AIRPORT_ID, DEST_AIRPORT_ID - We considered ORIGIN and DEST insteadd of these fields, so they are duplicates and ca
# DEP_DELAY, DEP_DELAY_NEW, DEP_TIME, CARRIER_DELAY, WEATHER_DELAY, NAS_DELAY, SECURITY_DELAY, LATE_AIRCRAFT_DELAY - Removed
# Other irrelevant coulmns are also ignored.

df = df.drop(['YEAR', 'OP_UNIQUE_CARRIER', \
              'OP_CARRIER_AIRLINE_ID', 'OP_CARRIER_FL_NUM', \
              'ORIGIN_AIRPORT_ID', 'DEST_AIRPORT_ID','DEP_DELAY',\
              'DEP_DELAY_NEW', 'WHEELS_ON', 'DEP_TIME', \
              'TAXI_IN', 'CRS_ARR_TIME', 'CRS_ELAPSED_TIME', 'ACTUAL_ELAPSED_TIME', \
              'ARR_TIME_BLK', 'ARR_DELAY_GROUP', 'DIVERTED', \
              'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY'], axis=1)
df.head(5)  # displays top 5 rows from dataframe
```

Out[4]:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	DEP_DEL15	ARR_TIME	ARR_DELAY	ARR_DELAY_NEW	ARR_DEL15	CA
0	9	4	1	LGA	MCO	1130	0.0	1354.0	-44.0	0.0	0.0	
1	9	4	1	ATL	SAN	1930	0.0	2044.0	-16.0	0.0	0.0	
2	9	4	1	BWI	DTW	1945	1.0	2252.0	95.0	95.0	1.0	
3	9	4	1	DTW	BWI	1730	1.0	1928.0	26.0	26.0	1.0	
4	9	4	1	MKE	MSP	910	0.0	1019.0	-8.0	0.0	0.0	

Figure 9: Code showing how data cleaning is done

In-order to fix the null values we check how columns and rows are presented in the data. We have the total number of rows to be 322199 and columns as 15. This can be seen in the code below

```
In [8]:
######################### FIXING NULL VALUES #########################

# Let us now check how many columns/features and rows/examples are present in our data

# In order to clean the data it is important to either fill null values with some data or remove them from the dataset.
# In this case we will drop the missing values. Hence, we are not considering null values for ARR_DEL15 as it is our output/
df = df.dropna(subset=['ARR_DEL15'])
# df = df.fillna({'ARR_DEL15': 1}

df.count()

rows, cols = df.shape  # returns dimensions of the dataframe
print("Number of rows: ", rows)
print("Number of columns: ", cols)

# Converting both columns as int as they have either 0 or 1 reoresenting ontime or delayed
df.astype({'DEP_DEL15': 'int32'})  # converting datatype of DEP_DEL15 column to int
df.astype({'ARR_DEL15': 'int32'})  # converting datatype of ARR_DEL15 column to int
```

```
Number of rows:  322199
Number of columns:  15
```

Out[8]:

EEK	ORIGIN	DEST	CRS_DEP_TIME	DEP_DEL15	ARR_TIME	ARR_DELAY	ARR_DELAY_NEW	ARR_DEL15	CANCELLED	AIR_TIME	FLIGHTS	DISTANCE
1	LGA	MCO	1130	0.0	1354.0	-44.0	0.0	0	0	127.0	1	950
1	ATL	SAN	1930	0.0	2044.0	-16.0	0.0	0	0	226.0	1	1892
1	BWI	DTW	1945	1.0	2252.0	95.0	95.0	1	0	100.0	1	409
1	DTW	BWI	1730	1.0	1928.0	26.0	26.0	1	0	66.0	1	409
1	MKE	MSP	910	0.0	1019.0	-8.0	0.0	0	0	51.0	1	297
...	...	...	...	...	...	...	...	...	...	...	...	...
2	DTW	LAS	840	0.0	938.0	-25.0	0.0	0	0	218.0	1	1749
2	ATL	MSP	1645	0.0	1811.0	-19.0	0.0	0	0	125.0	1	907
2	MSP	ATL	2015	0.0	2328.0	-27.0	0.0	0	0	111.0	1	907
2	ATL	BWI	2229	0.0	2.0	-23.0	0.0	0	0	74.0	1	577
2	ATL	BHM	1537	0.0	1522.0	-8.0	0.0	0	0	32.0	1	134

Figure 10: Code showing how to fix null values

To find out if the dataset is free from null values we run the code print (df.isnull().sum) as seen below

```
In [9]:
# Let's check if we have any null values remaining in the dataset. If yes, we will handle according
print(df.isnull().sum())  # Displays count of null values for each column
```

```
MONTH            0
DAY_OF_MONTH     0
DAY_OF_WEEK      0
ORIGIN           0
DEST             0
CRS_DEP_TIME     0
DEP_DEL15        0
ARR_TIME         0
ARR_DELAY        0
ARR_DELAY_NEW    0
ARR_DEL15        0
CANCELLED        0
AIR_TIME         0
FLIGHTS          0
DISTANCE         0
dtype: int64
```

Figure 11: Code showing rechecking for null values

Steps for feature extraction

- Extracting critical information from complex features

- Creating new columns with information from combination of two or more features

The codes describes the dataframe and shows feature that will provide insight to the analysis.

In [5]:
```
# Few features can provide important information for prediction of arrival delay and hence are considered.
# MONTH - Can be an important factor as people might travel more in few months which can lead to more aircrafts and hence mor
# DAY_OF_MONTH - Might be a useful feature to consider
# DAY_OF_WEEK - Very important information to consider as people have tendency to travel more during weekends
# CRS_DEP_TIME - Time is also very important as airports are often crowded for some hours in a day
# DISTANCE - Can play an important role as flights with shorter distance can get delayed more often
# FLIGHTS - More flights can lead to more delay.
df.describe()  # Lets see the description of our dataframe
```

Out[5]:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	CRS_DEP_TIME	DEP_DEL15	ARR_TIME	ARR_DELAY	ARR_DELAY_NEW	ARR_DEL15
count	324947.000000	324947.000000	324947.000000	324947.000000	322951.000000	322900.000000	322199.000000	322199.000000	322199.000000
mean	7.476330	15.926585	3.935208	1334.886341	0.147796	1478.598969	0.044277	9.845847	0.138135
std	1.101323	8.732798	1.972664	492.610812	0.354899	533.099945	41.690151	38.146251	0.345042
min	6.000000	1.000000	1.000000	1.000000	0.000000	1.000000	-238.000000	0.000000	0.000000
25%	7.000000	8.000000	2.000000	910.000000	0.000000	1051.000000	-16.000000	0.000000	0.000000
50%	7.000000	16.000000	4.000000	1322.000000	0.000000	1511.000000	-8.000000	0.000000	0.000000
75%	8.000000	23.000000	6.000000	1735.000000	0.000000	1924.000000	2.000000	2.000000	0.000000
max	9.000000	31.000000	7.000000	2359.000000	1.000000	2400.000000	1169.000000	1169.000000	1.000000

Figure 12: Code describing data frames

Next after getting information of the data-frame. We check the data-frame to know the different columns and datatype format

In [7]:
```
df.info()  # Getting information of our dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 324947 entries, 0 to 324946
Data columns (total 15 columns):
MONTH             324947 non-null int64
DAY_OF_MONTH      324947 non-null int64
DAY_OF_WEEK       324947 non-null int64
ORIGIN            324947 non-null object
DEST              324947 non-null object
CRS_DEP_TIME      324947 non-null int64
DEP_DEL15         322951 non-null float64
ARR_TIME          322900 non-null float64
ARR_DELAY         322199 non-null float64
ARR_DELAY_NEW     322199 non-null float64
ARR_DEL15         322199 non-null float64
CANCELLED         324947 non-null int64
AIR_TIME          322199 non-null float64
FLIGHTS           324947 non-null int64
DISTANCE          324947 non-null int64
dtypes: float64(6), int64(7), object(2)
memory usage: 37.2+ MB
```

Figure 13: Code showing dataframe data types

The next step is to remove irrelevant features by using the df.drop function

```
In [10]: # We are removing flights which got cancelled as we are considering only those flights which are not cancelled
         df = df.loc[~(df['CANCELLED'] == 1)]
         df = df.drop(['CANCELLED'], axis=1)
```

Figure 14: Code to remove cancelled flights

8 Data Exploration and Visualization

Here we find unique airports used by Delta airlines for both origin and destination flights. We had 150 airport origin and 150 destination

```
In [11]: # In this section of code we will find how many unique airports are present in the data from where
         unique_origin = df['ORIGIN'].unique()  # Finds unique values for ORIGIN column
         unique_origin_count = len(unique_origin)  # Finds total unique origins
         print("The total number of origins are: " , unique_origin_count)
         print('\n')
         print(unique_origin)

The total number of origins are:  150


['LGA' 'ATL' 'BWI' 'DTW' 'MKE' 'SAN' 'SAT' 'DEN' 'CMH' 'MDW' 'LAX' 'MYR'
 'SFO' 'SLC' 'TPA' 'PBI' 'MCO' 'PHL' 'GSP' 'PHX' 'CHS' 'DCA' 'BUF' 'RIC'
 'LAS' 'BNA' 'MSP' 'SJC' 'ORF' 'STL' 'BDL' 'IAD' 'BOS' 'FLL' 'MSO' 'DFW'
 'GEG' 'CLT' 'OMA' 'EWR' 'SRQ' 'CVG' 'AUS' 'ECP' 'CID' 'PDX' 'CLE' 'SMF'
 'MIA' 'SEA' 'SAV' 'ABQ' 'SNA' 'IND' 'GRR' 'FNT' 'OGG' 'MEM' 'RDU' 'ORD'
 'GSO' 'JAX' 'IAH' 'ATW' 'HOU' 'PNS' 'MHT' 'MSN' 'VPS' 'LIT' 'HNL' 'CHA'
 'JAN' 'LFT' 'TRI' 'PIT' 'MSY' 'DSM' 'BHM' 'SYR' 'ROA' 'OAK' 'ALB' 'JFK'
 'ANC' 'MCI' 'BTV' 'SDF' 'PVD' 'COS' 'LEX' 'KOA' 'ELP' 'ROC' 'GPT' 'FCA'
 'DAY' 'GRB' 'CAE' 'OKC' 'RSW' 'BIL' 'DAB' 'BOI' 'DAL' 'ICT' 'TLH' 'MDT'
 'CHO' 'TYS' 'EYW' 'BZN' 'AGS' 'MLB' 'AVP' 'SJU' 'RNO' 'TUL' 'STT' 'PHF'
 'BIS' 'XNA' 'CRW' 'HSV' 'ABE' 'ONT' 'FAR' 'JAC' 'PWM' 'FAY' 'CAK' 'FSD'
 'GNV' 'TUS' 'LIH' 'RAP' 'EVV' 'PSC' 'ILM' 'TVC' 'AVL' 'MOB' 'BGR' 'SBN'
 'FAI' 'JNU' 'SGF' 'BTR' 'STX' 'GTF']
```

Figure 15: Code for Destination (Arrival airport)

Next, we drilled down to find the top 20 unique airport by their flight volume. Note since we considering arrival delay we will only look at destination airports. Hartsfield-Jackson Atlanta International airport (ATL) had the highest flight volume of 82519 this was for the year 2017 during the summer months

```
In [16]:  hi_volume_dest = df['DEST'].value_counts()[:20]
           print(hi_volume_dest)

ATL    82519
MSP    25351
DTW    19613
SLC    15530
LAX    11421
JFK     9977
SEA     8763
LGA     7403
MCO     5771
BOS     5280
SFO     4809
LAS     4438
TPA     3670
FLL     3653
DEN     3629
MIA     3226
DCA     3054
PDX     2903
ORD     2876
CVG     2605
Name: DEST, dtype: int64

In [

'MIA' 'CLE' 'GRR' 'ORD' 'JAX' 'SAT' 'IAH' 'BOS' 'HOU' 'PNS' 'AUS' 'MSN'
'MCI' 'SMF' 'LIT' 'ALB' 'DAL' 'IND' 'ANC' 'TLH' 'PIT' 'MSY' 'DSM' 'RIC'
'TVC' 'SYR' 'IAD' 'ROA' 'OAK' 'CLT' 'BIL' 'PVD' 'COS' 'ILM' 'ELP' 'ROC'
'GPT' 'FCA' 'SDF' 'BZN' 'PWM' 'MHT' 'TRI' 'GRB' 'DAY' 'PSC' 'ATW' 'AGS'
'DAB' 'BOI' 'RAP' 'ICT' 'SJU' 'MDT' 'OKC' 'TYS' 'EYW' 'MLB' 'PHF' 'HSV'
'BIS' 'FSD' 'LFT' 'XNA' 'RNO' 'JAN' 'STT' 'TUL' 'CRW' 'AVP' 'JAC' 'ABE'
'CAK' 'ONT' 'OGG' 'KOA' 'BTV' 'FAR' 'MOB' 'MSO' 'FAY' 'BGR' 'CHO' 'FAI'
'SBN' 'BTR' 'SGF' 'JNU' 'STX' 'GTF']
```

Figure 16: Code showing top 20 unique airport used by Delta airlines by their flight volume

8.1 Arrival Delay Visualization

Here a histogram was plotted to show frequency of flights at destination airports

In [17]:

```
# Plotting a Histogram
# df['DEST'].value_counts().nlargest(20).plot(kind='bar', figsize=(10,5))
# plt.title('Histogram')
# plt.ylabel('Frequency of flights')
# plt.xlabel('Destination Airport');

plt.figure(figsize=(10, 10))
plt.title('Histogram for Number of flights arrived at airport vs Destination Airports')
axis = sns.countplot(x=df['DEST'], data = df,
              order=df['DEST'].value_counts().iloc[:20].index)
axis.set_xticklabels(axis.get_xticklabels(), rotation=90, ha="right")
plt.tight_layout()
plt.show()
```

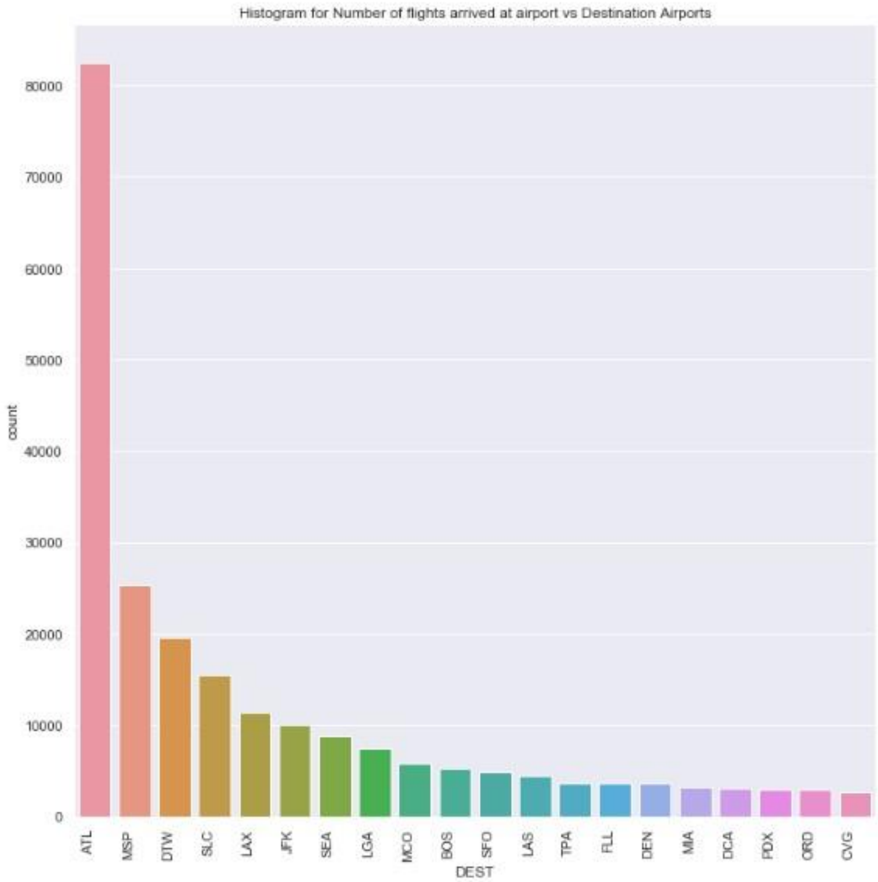

Figure 17: Hartsfield-Jackson Atlanta International airport (ATL) has the highest flight frequency as compared to other airports used by Delta airlines

Next, we find the role of distance in arrival delay. From the analysis, flight with shorter distance have arrival delays

```
In [24]:  ########################## ROLE OF DISTANCE IN ARRIVAL DELAY ##########################

          # Checking if distance affects delays.
          import statsmodels.api as sm

          # create a data frame that stores 2 variables: distance and arrival delay
          dist_delay = df.groupby('DISTANCE', as_index = False)['ARR_DELAY_NEW'].mean()  # Grouping by distar

          # plot a scatter plot that takes distance as predictor, and arrival delay as response. This will sh
          x = dist_delay['DISTANCE']
          y = dist_delay['ARR_DELAY_NEW']
          plt.xlabel("Distance (in km)")
          plt.ylabel("Average Arrival Delay (in min)")
          plt.scatter(x, y, alpha=0.4)
          # Graph below crearly indicates that shorter distance flights are more prone to arrival delays.
```

Figure 18: Code check if distance affects delay

```
Out[24]: <matplotlib.collections.PathCollection at 0x20355b6a6c8>
```

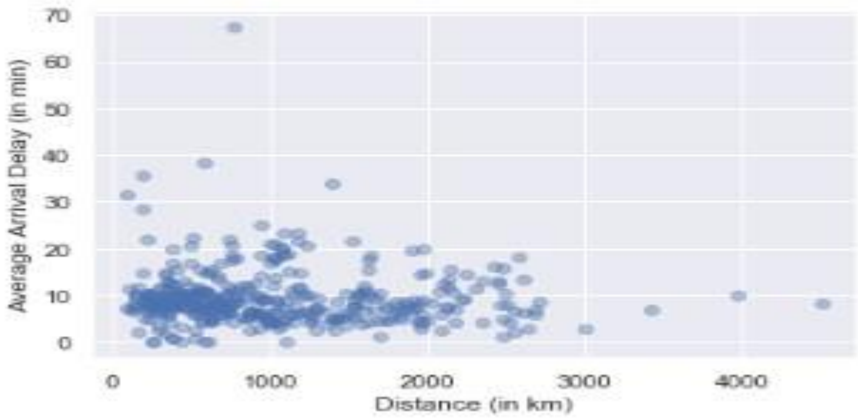

Figure 19: Visualization showing flight with shorter distance have arrival delays

Next, we check how the role of month of travel affects arrival delay

```
In [26]:
######################### ROLE OF MONTH OF TRAVEL IN ARRIVAL DELAY #########################

# Finding probablity of arrival delay for according to months
# Some months can see more delays than the others. Below histogram clearly indicates more delays in
# Then we can see drop in delays for next two months
month_delay = df.groupby('MONTH', as_index = False)['ARR_DELAY_NEW'].mean()

# print out average arrival delay time by month:
print(month_delay)

# plot a scatter plot that takes distance as predictor, and arrival delay as response
x = month_delay['MONTH']
y = month_delay['ARR_DELAY_NEW']
plt.xlabel("Month")
plt.ylabel("Average Arrival Delay (in min)")
plt.bar(x, y, alpha=0.4)
```

	MONTH	ARR_DELAY_NEW
0	6	12.204245
1	7	12.258788
2	8	8.028336
3	9	6.615392

Figure 20: Code showing output of arrival delay caused by travel months

```
Out[26]: <BarContainer object of 4 artists>
```

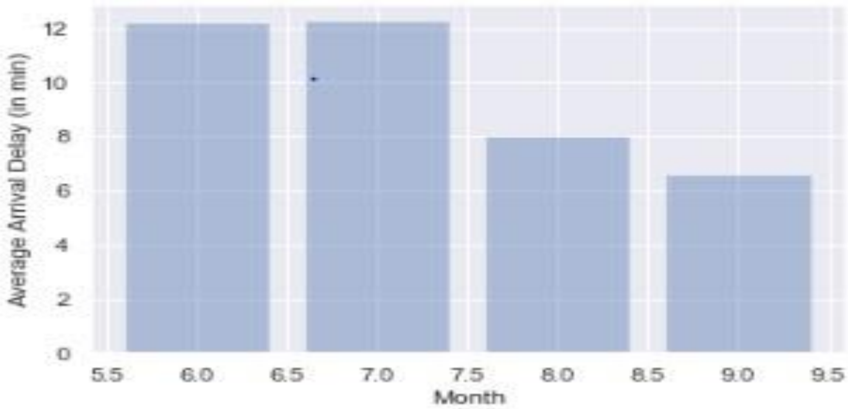

Figure 21 : Visualization showing months with the highest arrival delay for Delta airlines

We notice that June which is the 6th month and July the 7th month have the highest arrival delay during the summer while there is a slight difference of 0.5 to make July the highest arrival delay month in the year 2017.

Here, we analyse how depature time affects arrival delay

```
In [31]:  ######################### ROLE OF DEPARTURE TIME IN ARRIVAL DELAY #########################

# in the same way above, create a data frame that aggregate ARR_DELAY_NEW by hour in a day
hour_delay = df.groupby('CRS_DEP_TIME', as_index = False)['ARR_DELAY_NEW'].mean()

# print out average arrival delay time by hour:
print(hour_delay)

# plot a line graph:
# hour_delay.plot(x = 'CRS_DEP_TIME', y = 'ARR_DELAY_NEW')
ax = sns.lineplot(x="CRS_DEP_TIME", y="ARR_DELAY_NEW", data=hour_delay, label='Average delay (in mi
```

```
    CRS_DEP_TIME  ARR_DELAY_NEW
0              0       4.849298
1              1       6.530120
2              5       4.418650
3              6       5.269229
4              7       5.680161
5              8       5.572165
6              9       6.449561
7             10       6.288707
8             11       6.958391
9             12       7.859506
10            13       8.955663
11            14       9.408976
12            15      11.388974
13            16      13.419919
14            17      14.510493
15            18      15.631798
16            19      18.896218
17            20      13.727559
18            21      13.776196
19            22      12.443405
20            23      13.524565
21            24      10.862595
```

Figure 23 : Code to show how depture time affect arrival delay

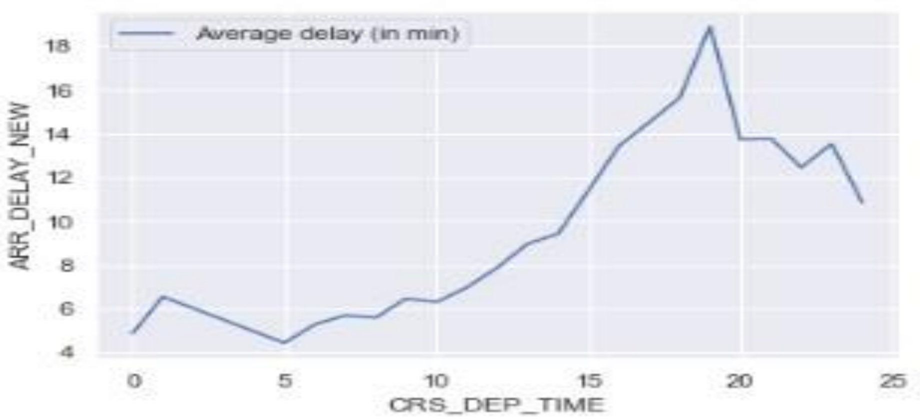

Figure 24: Visualization showing lateness in departure time causes flight arrival delay

9 Data Modelling

The first thing done here was to use the label encoder to convert strings to numbers. The values of origin and destination had the data type float and was converted to int using the python function as seen in the codes

```
In [32]:  ########################## DATA CLEANING: CATEGORICAL VALUES ##########################

          # Converting origin and dest to numeric values for analysis. In order to train our ML models, we ne
          # We are using a label encoder to convert strings to numbers. Each number will act as a label for c
          from sklearn.preprocessing import LabelEncoder
          le = LabelEncoder()
          df["ORIGIN"] = le.fit_transform(df["ORIGIN"])
          df["DEST"] = le.fit_transform(df["DEST"])
```

Figure 25: Code to convert values of origin and destination to int

Next was to carry out another feature extraction to remove columns not relevant in training our model for arrival delay

```
In [33]:  ########################## DATA CLEANING: REMOVING IRRELEVANT COLUMNS ##########################

          # Removing these values as we are not considering departure and arrival delay in input parameters.
          df = df.drop(['DEP_DEL15', 'ARR_TIME', 'ARR_DELAY', 'ARR_DELAY_NEW'], axis=1)
```

```
In [34]:  df.head()
```

Out[34]:

	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	ORIGIN	DEST	CRS_DEP_TIME	ARR_DEL15	AIR_TIME	FLIGHTS	DISTAN
0	9	4	1	81	85	11	0.0	127.0	1	9
1	9	4	1	5	123	19	0.0	226.0	1	18
2	9	4	1	21	42	19	1.0	100.0	1	4
3	9	4	1	42	21	17	1.0	66.0	1	4
4	9	4	1	91	96	9	0.0	51.0	1	2

```
In [35]:  df.info()

          <class 'pandas.core.frame.DataFrame'>
          Int64Index: 322199 entries, 0 to 324946
          Data columns (total 10 columns):
          MONTH           322199 non-null int64
          DAY_OF_MONTH    322199 non-null int64
          DAY_OF_WEEK     322199 non-null int64
          ORIGIN          322199 non-null int32
          DEST            322199 non-null int32
          CRS_DEP_TIME    322199 non-null int32
          ARR_DEL15       322199 non-null float64
          AIR_TIME        322199 non-null float64
          FLIGHTS         322199 non-null int64
          DISTANCE        322199 non-null int64
          dtypes: float64(2), int32(3), int64(5)
          memory usage: 23.4 MB
```

Figure 26: Code to remove irrelevant columns

After this the data is saved

```
In [36]: # Saving cleaned data in csv
df.to_csv('data_cleaned.csv')
```

Figure 27: Code showing data been saved

10. Preparing Training set

We will prepare a balanced dataset for training so that we get better accuracy on test data. Below are the codes used to prepare the dataset by dividing to training and testing dataset. Here are the steps we followed to make our training data balanced with same positive and negative value. i.e. positive =1 negative= 0

10.1 Splitting the data set

These are the steps in splitting the data

- **Step 1:** Get validation data and test data

```
In [38]: # Getting 30% validation data for test
df_validation_test = df.sample(frac=0.3, random_state=42)
```

Figure 28: code showing Validation of dat

This will remove 30% data from our dataframe to create both validation and test data.

- **Step 2:** Divide validation and test data further into 15% each the remaining 50% will be used for cross validation

```
In [39]: # Dividing validation data to test and validation data
df_test = df_validation_test.sample(frac=0.5, random_state=42)  # 50% of this validation + test dat
df_valid = df_validation_test.drop(df_test.index)  # remaining 50% will be used for cross validatic
```

Figure 29: Code showing validation and test data further splitted

- **Step 3:** Getting the train data

```
In [40]: # Complete training data after removing validation and test data
df_train_all = df.drop(df_validation_test.index)
```

Figure 30: Code showing how to get training data

- **Step 4:** Calculating prevalence to show the total number of cases where arrival delay happened in the test and validation data

```
In [41]:  calculate_prevalence(df_test['ARR_DEL15'])  # Displays ratio of test data having arrival delays.
Out[41]:  0.1398510242085661

In [42]:  calculate_prevalence(df_valid['ARR_DEL15'])  # Displays ratio of validation data having arrival del
Out[42]:  0.13858886819780675

In [43]:  calculate_prevalence(df_train_all['ARR_DEL15'])  # Displays ratio of training data having arrival d
Out[43]:  0.13767020337945987
```

Figure 31: Code calculating prevalence

- **Step 5:** Separating the training dataset into input and output. Note all columns are input except that of ARR_DEL15 which is the output. The figure below the block 45 in the code shows the size of the matrix in rows and columns.

```
In [45]:  # Preparing training and test data
          X_train = df_train.drop(['ARR_DEL15'], axis=1).values  # returns numpy array with training values.
          X_train_all = df_train_all.drop(['ARR_DEL15'], axis=1).values
          X_valid = df_valid.drop(['ARR_DEL15'], axis=1).values  # returns numpy array with validation values

          y_train = df_train['ARR_DEL15'].values
          y_valid = df_valid['ARR_DEL15'].values

          print(X_train_all.shape)
          print(X_train.shape, y_train.shape)
          print(X_valid.shape, y_valid.shape)

          (225539, 9)
          (62100, 9) (62100,)
          (48330, 9) (48330,)
```

Figure 32 : Code showing the separation of training data

10.2 Steps To Prepare the Dataset

The Listed steps are as follows:

1. Taking out all positive values from training data so that it gives all the data of po
2. Taking out all the negative values by removing positive data from the complete training dataset
3. Calculating how many positive values we have
4. Taking out the same no of negative data randomly from the negative data

5. Concatenating both positive data and negative data which we got from step 4
6. Randomly shuffling the data from step 5 in order to mix positive and negative examples

CODE FOR STEPS IN BLOCK 44

```
In [44]:  # In order to successfully train a model it is important that training dataset has balanced example
          # This is because negative examples are more so our model can be biased towards negatives examples
          # It is important not only to get the best accuracy but also metrics like precision, recall, f1 sco
          # Balancing positive and negative examples for a balanced dataset
          rows_pos = df_train_all['ARR_DEL15'] == 1  # indices for positive examples
          df_train_pos = df_train_all.loc[rows_pos]  # dataframe with positive examples. This means ARR_DEL15
          df_train_neg = df_train_all.loc[~rows_pos]  # dataframe with negative examples.

          df_train = pd.concat([df_train_pos, df_train_neg.sample(n = len(df_train_pos), random_state=42)], a
          df_train = df_train.sample(n = len(df_train), random_state=42).reset_index(drop = True)
```

Figure 33: Code to prepare dataset for modelling

For step 1 and 3 the code line 1 and 2 rows_pos = df_train_all['ARR_DEL15'] ==1 and df_train_pos = df_train_all.loc[rows_pos] explains how the positive values from the training data is taken out to give all positive dataset

For step 2, code line 3 df_train_neg = df_train_all.loc[~rows_pos] takes out all the negative values by removing positive data from the complete training dataset

For step 3, 4 and 5 code line 4 df_train = pd.concat([df_train_pos, df_train_neg.sample(n = len(df_train_pos), random_state=42)], axis = 0) pd.concat does randomization and takes the same number of negative values as positive and removing remaining ones. In this way we will have a balanced training data with equal positive and negative values

For step 6 , shuffling was done by randomly mixing the values to achieve values that are representatives of the entire data distribution which will produce good performance of the algorithm and avoid bias

```
In [37]:  # Shuffling data for better results and avoid biased results
          df = df.sample(n = len(df), random_state=42)
          df = df.reset_index(drop = True)  # This will shuffle our data randomly
```

Figure 34: Code showing Shuffling

11 Training Machine Learning Models

We will train different Machine learning Algorithms and see how they perform on our dataset. It is important to note that each machine learning algorithm performs differently and we need to choose the right algorithm for our problem. We will also calculate metrics associated with each algorithm for a comparative analysis. Models used are Logistics regression, Support vector Classifier, Random forest, Naïve bayes, Gradient Boosting, SDG classifier

STEP 1: Import the necessary libraries to carry out analysis and define the necessary metrics to be used

```
In [46]:
from sklearn.metrics import roc_auc_score, accuracy_score, precision_score, recall_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV

def calculate_specificity(y, y_pred, th=0.5):
    return sum((y_pred < th) & (y == 0)) / sum(y == 0)

def print_report(y, y_pred, th=0.5):
    auc = roc_auc_score(y, y_pred)
    accuracy = accuracy_score(y, (y_pred > th))
    precision = precision_score(y, (y_pred > th))
    recall = recall_score(y, (y_pred > th))
    specificity = calculate_specificity(y, y_pred, th)
    prevalence = calculate_prevalence(y)
    print('auc: ', auc)
    print('accuracy: ', accuracy)
    print('precision: ', precision)
    print('recall: ', recall)
    print('specificity: ', specificity)
    print('prevalence: ', prevalence)
```

Figure 35: Code showing libraries imported for modelling and

Training Proper

Logistics Regression

Logistic regression was implemented using the sklearn.linear_model in LogisticRegression. This implementation can fit binary, One-vs-Rest, or multinomial logistic regression with optional , or Elastic-Net regularization

```
In [47]: # Training a logistic regression model and testing its accuracy
         from sklearn.linear_model import LogisticRegression
         lr = LogisticRegression()
         lr.fit(X_train, y_train)

C:\Users\kenne\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: De
fault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

Out[47]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=1, l1_ratio=None, max_iter=100,
                            multi_class='warn', n_jobs=None, penalty='l2',
                            random_state=None, solver='warn', tol=0.0001, verbose=0,
                            warm_start=False)

In [48]: lr.score(X_valid, y_valid)

Out[48]: 0.6405131388371612

In [49]: y_pred = lr.predict(X_valid)
         print_report(y_valid, y_pred)

auc:  0.6407538608700521
accuracy:  0.6405131388371612
precision:  0.22290282392026578
recall:  0.6410868916094357
specificity:  0.6404208301306688
prevalence:  0.13858886819780675

In [50]: confusion_matrix(y_valid, y_pred)

Out[50]: array([[26662, 14970],
                [ 2404,  4294]], dtype=int64)
```

Figure 36: Logistics regression code for modelling

In block code 47 line 1 and 2 the logistic regression is initialised, in line 3 the model is trained to fit function using different parameters . Block code 48 shows the output from the cross validation. Looking at the confusion matrix the number of true positives (TP) are 26662 False positives are 14970 False negative are 2404 and True negative are 4292. So, since we considering accuracy, TP+TN/total will give the accuracy of 64%

Naïve Bayes

The sklearn.naive_bayes function was used while importing the Gaussian Naïve Bayes. Several parameters were defined to ensure effective modelling of the trained data

```
In [55]:  # Training a naive bayes classifier and testing its accuracy
          from sklearn.naive_bayes import GaussianNB
          nb = GaussianNB()
          nb.fit(X_train, y_train)

Out[55]: GaussianNB(priors=None, var_smoothing=1e-09)

In [56]:  nb.score(X_valid, y_valid)

Out[56]: 0.6027312228429547

In [57]:  y_pred = nb.predict(X_valid)
          print_report(y_valid, y_pred)

          auc:  0.6208907070760508
          accuracy:  0.6027312228429547
          precision:  0.2045282662128947
          recall:  0.6460137354434159
          specificity:  0.5957676787086856
          prevalence:  0.13858886819780675

In [58]:  confusion_matrix(y_valid, y_pred)

Out[58]: array([[24803, 16829],
                [ 2371,  4327]], dtype=int64)
```

Figure 35: Naïve Bayes code for modelling

After using the confusion matrix the accuracy was 60%

Decision trees

performs multi-class classification on a dataset. They are supervised machine learning methods that performs classification tasks. However, the aim here was to develop a model that predicts the value of a target variable by learning simple decision rules inferred from the data features

```
In [59]:  # Training a Decision Tree Classifier and testing its accuracy
           from sklearn.tree import DecisionTreeClassifier
           tree = DecisionTreeClassifier(max_depth = 10)
           tree.fit(X_train, y_train)

Out[59]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=10,
                                max_features=None, max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, presort=False,
                                random_state=None, splitter='best')

In [60]:  tree.score(X_valid, y_valid)

Out[60]: 0.7068901303538175

In [61]:  y_pred = tree.predict(X_valid)
           print_report(y_valid, y_pred)

          auc:  0.6646869927042363
          accuracy:  0.7068901303538175
          precision:  0.26048749198203974
          recall:  0.6063003881755747
          specificity:  0.7230735972328978
          prevalence:  0.13858886819780675

In [62]:  confusion_matrix(y_valid, y_pred)

Out[62]: array([[30103, 11529],
                [ 2637,  4061]], dtype=int64)
```

Figure 36: Decision tree code for modelling

The sklearn.tree function was employed to use in the decision tree classifier and the different paraments where used to find the appropriate performance. Confusion matrix showed how the data performance calculation. Accuracy here was 66%

Random forest

```
Out[63]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=6, max_features='auto', max_leaf_nodes=None,
                                min_impurity_decrease=0.0, min_impurity_split=None,
                                min_samples_leaf=1, min_samples_split=2,
                                min_weight_fraction_leaf=0.0, n_estimators=50,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)

n [64]:  rf.score(X_valid, y_valid)

Out[64]: 0.6666873577488103

n [65]:  y_pred = rf.predict(X_valid)
         print_report(y_valid, y_pred)

         auc:  0.6542552439162377
         accuracy:  0.6666873577488103
         precision:  0.23778211200891614
         recall:  0.6370558375634517
         specificity:  0.6714546502690238
         prevalence:  0.13858886819780675

n [66]:  confusion_matrix(y_valid, y_pred)

Out[66]: array([[27954, 13678],
                [ 2431,  4267]], dtype=int64)
```

The performance metrics considered here was accuracy and it was 66%

Gradient Boosting Classifier

Using the sklearn.ensemble function the gradient boosting classifier was imported

```
In [67]:  # Training a Gradient Boosting Classifier and testing its accuracy
          from sklearn.ensemble import GradientBoostingClassifier
          gbc = GradientBoostingClassifier(n_estimators=100, max_depth=3, learning_rate=1.0)
          gbc.fit(X_train, y_train)

Out[67]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
                                    learning_rate=1.0, loss='deviance', max_depth=3,
                                    max_features=None, max_leaf_nodes=None,
                                    min_impurity_decrease=0.0, min_impurity_split=None,
                                    min_samples_leaf=1, min_samples_split=2,
                                    min_weight_fraction_leaf=0.0, n_estimators=100,
                                    n_iter_no_change=None, presort='auto',
                                    random_state=None, subsample=1.0, tol=0.0001,
                                    validation_fraction=0.1, verbose=0,
                                    warm_start=False)

In [68]:  gbc.score(X_valid, y_valid)

Out[68]: 0.7097661907717774

In [69]:  y_pred = gbc.predict(X_valid)
          print_report(y_valid, y_pred)

          auc:  0.6922263533471852
          accuracy:  0.7097661907717774
          precision:  0.2748663758677889
          recall:  0.6679605852493281
          specificity:  0.7164921214450423
          prevalence:  0.13858886819780675

In [70]:  confusion_matrix(y_valid, y_pred)

Out[70]: array([[29829, 11803],
                [ 2224,  4474]], dtype=int64)
```

The gradient boosting classifier was the best classifier as it outperformed all other models predicting flight arrival delay for delta airlines with a 70% accuracy.