Airline Arrivals Report

Abstract

This project uses Airline Arrivals dataset from RITA to predict how late fights will be by using some classification algorithms. Moreover, sveral technique for features selection are also applied, and then finding the best parameters for models. Finally, the results of each model is compared to the others to evaluate which are bad and good, then discusing on them to improve performance of algorithms in this dataset.

Table of Contents

INTRODUCTION

Airline Arrivals dataset is from stat-computing.org which gives information related to Airline in several airports.

This data is utilized in the project to make a prediction which flight will be late (a flight considered as late if it is more than 30 minutes late) on various classification algorithms. The procedure below shows how to address this issue:

- Explore the data, find anomalies in datset and interactions among features.
- Process data to make it eaiser and suitable for computation before going to train it.
- Apply models in Naïve Bayes, Logistic Regression, Decision Tree, Random Forest, Gradient Boosting for Prediction.
- Apply PCA for feature selections in all models above to evalutate performances.
- GridsearchCV is utilized to gain the best parameters for model giving best result

- GridsearchCV is utilized to gain the best parameters for model giving best result.
- Compare the prerformance of each algorithm on this dataset and disscus on them to find the way to improve them.

The next session is [Data Exploration](#)

DATA EXPLORATION

Import the necessary libraries

In [1]:

```
import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt

import scipy as sci # use for ttest
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression

import time
```

Data Overview

Load the dataset and show the first five samples.

In [2]:

```
data = pd.read_csv('2008.csv')
data.head()
```

Out[2]:

	Year	Month	DayofMonth	DayOfWeek	DepTime	CRSDepTime	ArrTime	CRSArrTime	UniqueCarrier	FlightNum	...	TaxiIn	TaxiOu
0	2008	1	3	4	2003.0	1955	2211.0	2225	WN	335	...	4.0	8.
1	2008	1	3	4	754.0	735	1002.0	1000	WN	3231	...	5.0	10.
2	2008	1	3	4	628.0	620	804.0	750	WN	448	...	3.0	17.
3	2008	1	3	4	926.0	930	1054.0	1100	WN	1746	...	3.0	7.
4	2008	1	3	4	1829.0	1755	1959.0	1925	WN	3920	...	3.0	10.

5 rows × 29 columns

Some Key Details:

1 Year 2008

2 Month 1-12

3 DayofMonth 1-31

4 DayOfWeek 1 (Monday) - 7 (Sunday)

5 DepTime actual departure time (local, hhmm)

6 CRSDepTime scheduled departure time (local, hhmm)

7 ArrTime actual arrival time (local, hhmm)

8 CRSArrTime scheduled arrival time (local, hhmm)

9 UniqueCarrier unique carrier code

10 FlightNum flight number

11 TailNum plane tail number

12 ActualElapsedTime in minutes

13 CRSElapsedTime in minutes

14 AirTime in minutes

15 ArrDelay arrival delay, in minutes

16 DepDelay departure delay, in minutes

17 Origin origin IATA airport code

18 Dest destination IATA airport code

19 Distance in miles

20 TaxiIn taxi in time, in minutes

21 TaxiOut taxi out time in minutes

22 Cancelled was the flight cancelled?

23 CancellationCode reason for cancellation (A = carrier, B = weather, C = NAS, D = security)

24 Diverted 1 = yes, 0 = no

25 CarrierDelay in minutes

26 WeatherDelay in minutes

27 NASDelay in minutes

28 SecurityDelay in minutes

29 LateAircraftDelay in minutes

Because 'Year' is only 1 value 2008 => it should be dropped.

In [3]:

```
data.drop(axis=1,columns='Year',inplace=True)
```

ArrDelay = CRSArrTime - ArrTime

DepDelay= CRSDepTime - DepTime

Therfore, 'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime' should be removed.

In [ ]:

```

```

In [4]:

```
data.drop(axis=1, columns=['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime'], inplace=True)
```

Check:

- How many samples and features included in the dataset
- Types of features
- Whether or not missing values existed in this dataset

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 7009728 entries, 0 to 7009727
Data columns (total 24 columns):
Month                int64
DayofMonth           int64
DayOfWeek            int64
UniqueCarrier        object
FlightNum            int64
TailNum              object
ActualElapsedTime    float64
CRSElapsedTime       float64
AirTime              float64
ArrDelay             float64
DepDelay             float64
Origin               object
Dest                 object
Distance             int64
TaxiIn               float64
TaxiOut              float64
Cancelled            int64
CancellationCode     object
Diverted             int64
CarrierDelay         float64
WeatherDelay         float64
NASDelay             float64
SecurityDelay        float64
LateAircraftDelay    float64
dtypes: float64(12), int64(7), object(5)
memory usage: 1.3+ GB
```

Actually **=> 'Year', 'Month','DayofMonth', 'DayOfWeek', 'FlightNum', 'Cancelled', 'Diverted'** are category, so they are converted to Category types.

In [6]:

```
data[[ 'Month','DayofMonth', 'DayOfWeek', 'FlightNum', 'Cancelled',
'Diverted']]=data[['Month','DayofMonth', 'DayOfWeek', 'FlightNum', 'Cancelled',
'Diverted']].astype('category')
```

Convert int64 type to float64 type for minmaxscaler in the scaling features sesstion.

In [7]:

```
data[data.select_dtypes(include='int64').columns[:]] = data[data.select_dtypes(include='int64').co
lumns[:]].astype('float64')
```

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7009728 entries, 0 to 7009727
Data columns (total 24 columns):
Month                category
DayofMonth           category
DayOfWeek            category
UniqueCarrier        object
FlightNum            category
TailNum              object
ActualElapsedTime    float64
CRSElapsedTime       float64
AirTime              float64
ArrDelay             float64
DepDelay             float64
Origin               object
Dest                 object
Distance             float64
TaxiIn               float64
TaxiOut              float64
Cancelled            category
CancellationCode     object
Diverted             category
CarrierDelay         float64
```

```
WeatherDelay          float64
NASDelay              float64
SecurityDelay         float64
LateAircraftDelay     float64
dtypes: category(6), float64(13), object(5)
memory usage: 1009.8+ MB
```

=> There are 7009728 samples and 29 features, so the size of dataset is 7009728x29. => The dataset includes:

- float64: 17.
- object and category: 12.

The list of 29 features:

In [9]:

```
print(*data.columns + ',')
```

```
Month, DayofMonth, DayOfWeek, UniqueCarrier, FlightNum, TailNum, ActualElapsedTime,
CRSElapsedTime, AirTime, ArrDelay, DepDelay, Origin, Dest, Distance, TaxiIn, TaxiOut, Cancelled, C
ancellationCode, Diverted, CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay,
```

Outliers Detection and Removal

Data is divided into four groups via the 25th, 50th and 75th values.

The IQR defines the middle 50% of the data, or the body of the data.

The IQR can be used to identify outliers by defining limits on the sample values that are a factor k of the IQR below the 25th percentile or above the 75th percentile. The common value for the factor k is the value 1.5. A factor k of 3 or more can be used to identify values that are extreme outliers or “far outs” when described in the context of box and whisker plots.

In [10]:

```
data.describe()
```

Out[10]:

	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	Ca
count	6.855029e+06	7.008884e+06	6.855029e+06	6.855029e+06	6.873482e+06	7.009728e+06	6.858079e+06	6.872670e+06	1.5
mean	1.273224e+02	1.288668e+02	1.040186e+02	8.168452e+00	9.972570e+00	7.263870e+02	6.860852e+00	1.645305e+01	1.5
std	7.018731e+01	6.940974e+01	6.743980e+01	3.850194e+01	3.531127e+01	5.621018e+02	4.933649e+00	1.133280e+01	4.0
min	1.200000e+01	-1.410000e+02	0.000000e+00	-5.190000e+02	-5.340000e+02	1.100000e+01	0.000000e+00	0.000000e+00	0.0
25%	7.700000e+01	8.000000e+01	5.500000e+01	-1.000000e+01	-4.000000e+00	3.250000e+02	4.000000e+00	1.000000e+01	0.0
50%	1.100000e+02	1.100000e+02	8.600000e+01	-2.000000e+00	-1.000000e+00	5.810000e+02	6.000000e+00	1.400000e+01	0.0
75%	1.570000e+02	1.590000e+02	1.320000e+02	1.200000e+01	8.000000e+00	9.540000e+02	8.000000e+00	1.900000e+01	1.6
max	1.379000e+03	1.435000e+03	1.350000e+03	2.461000e+03	2.467000e+03	4.962000e+03	3.080000e+02	4.290000e+02	2.4

The results show 17 numbers for each column in the dataset:

- **count** shows how many rows have non-missing values.
- **mean** is the average.
- **std** is the standard deviation which measures how numerically spread out the values are.
- **min, 25%, 50%, 75% and max** values, in each column from lowest to highest value. The first (smallest) value is the min. If you go a quarter way through the list, you'll find a number that is bigger than 25% of the values and smaller than 75% of the values. That is the 25% value (pronounced "25th percentile"). The 50th and 75th percentiles are defined analogously, and the max is the largest number.

In [11]:

```
data.describe().min() < 0
```

Out[11]:

```
ActualElapsedTime    False
CRSElapsedTime        True
AirTime              False
ArrDelay              True
DepDelay              True
Distance             False
TaxiIn               False
TaxiOut              False
CarrierDelay         False
WeatherDelay         False
NASDelay             False
SecurityDelay        False
LateAircraftDelay    False
dtype: bool
```

ArrDelay: Difference in minutes between scheduled and actual arrival time. Early arrivals show negative numbers. DepDelay: Difference in minutes between scheduled and actual departure time. Early departures show negative numbers.

In [12]:

```
data['CRSElapsedTime'][data['CRSElapsedTime'] < 0]
```

Out[12]:

```
768964      -9.0
1358057    -21.0
1992427    -25.0
1992500    -12.0
3171278    -10.0
3171282    -18.0
3995853   -140.0
3995871   -140.0
3995876   -140.0
3995877   -140.0
3995883   -140.0
3997639   -141.0
3997645   -141.0
3997646   -141.0
3997652   -141.0
Name: CRSElapsedTime, dtype: float64
```

In [13]:

```
data.iloc[768964,:]
```

Out[13]:

```
Month                      2
DayofMonth                 3
DayOfWeek                  7
UniqueCarrier             OO
FlightNum               6004
TailNum               N934SW
ActualElapsedTime        NaN
CRSElapsedTime            -9
AirTime                  NaN
ArrDelay                 NaN
DepDelay                  69
Origin                   ORD
Dest                     IND
Distance                 177
TaxiIn                   NaN
TaxiOut                   29
Cancelled                  0
CancellationCode         NaN
Diverted                   1
CarrierDelay             NaN
WeatherDelay             NaN
```

```
WeatherDelay              NaN
NASDelay                  NaN
SecurityDelay             NaN
LateAircraftDelay         NaN
Name: 768964, dtype: object
```

'CRSElapsedTime', 'ArDelay', 'DepDelay' contain values < 0 they should be considered in Outlier section.**

Because features as 'DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime' are in hour hour min min type, so they will be converted to min min min min type to be suitable for other features in min.

In [14]:

```
"""df_HHMM = data[['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime']]
def Convert_HHMM_MMMM(Data):
    MMin = Data % 100
    HHtoMin = 60*(Data - MMin)/100
    Data = MMin + HHtoMin
    return Data
data[['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime']]  = Convert_HHMM_MMMM(df_HHMM)"""
```

Out[14]:

```
"df_HHMM = data[['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime']]\ndef
Convert_HHMM_MMMM(Data):\n    MMin = Data % 100\n    HHtoMin = 60*(Data - MMin)/100\n    Data =
MMin + HHtoMin\n    return Data\ndata[['DepTime', 'CRSDepTime', 'ArrTime', 'CRSArrTime']]  =
Convert_HHMM_MMMM(df_HHMM)"
```

Now the remained Numeric features are used IQR to check the outliers.

In [15]:

```
dataNumeric = data.select_dtypes(include=np.number)
# get quartile 1st and 3rd
Q1, Q3 = dataNumeric.quantile(0.25), dataNumeric.quantile(0.75)
IQR = Q3 - Q1
# Boundary
cutOff = 3*IQR
# Setup Boudnary
lower, upper = Q1 - cutOff, Q3 + cutOff
# identify Outliers
for i in dataNumeric:
    print(dataNumeric[i][(dataNumeric[i] < lower[i]) | (dataNumeric[i] > upper[i])].value_counts().
sort_index())
    print('Total Outliers for ' + i + ' is ' , dataNumeric[i][(dataNumeric[i] < lower[i]) |
(dataNumeric[i] > upper[i])].value_counts().sum())
    print('---------------------------------------------------------------\n')
```

```
398.0     591
399.0     600
400.0     579
401.0     548
402.0     525
         ...
905.0       1
1003.0      1
1114.0      1
1182.0      1
1379.0      1
Name: ActualElapsedTime, Length: 304, dtype: int64
Total Outliers for ActualElapsedTime is  22632
---------------------------------------------------------------

397.0      652
398.0      664
399.0      315
400.0     1336
401.0      396
          ...
604.0       22
605.0        1
635.0      210
```

```
660.0      152
1435.0       1
Name: CRSElapsedTime, Length: 125, dtype: int64
Total Outliers for CRSElapsedTime is  18923
---------------------------------------------------------------

364.0     585
365.0     545
366.0     542
367.0     522
368.0     465
         ...
886.0       1
981.0       1
1091.0      1
1154.0      1
1350.0      1
Name: AirTime, Length: 299, dtype: int64
Total Outliers for AirTime is  19258
---------------------------------------------------------------

-519.0     1
-129.0     1
-109.0     1
-92.0      1
-91.0      1
          ..
 1655.0    1
 1707.0    1
 1951.0    1
 2453.0    1
 2461.0    1
Name: ArrDelay, Length: 998, dtype: int64
Total Outliers for ArrDelay is  320366
---------------------------------------------------------------

-534.0     1
-92.0      1
-79.0      1
-71.0      1
-70.0      3
          ..
 1597.0    1
 1710.0    1
 1952.0    1
 2457.0    1
 2467.0    1
Name: DepDelay, Length: 1049, dtype: int64
Total Outliers for DepDelay is  585932
---------------------------------------------------------------

2845.0    1172
2846.0    1122
2860.0     660
2917.0    1828
2936.0     522
2979.0     560
2986.0     192
2994.0     732
3043.0     306
3110.0     180
3266.0     413
3303.0     326
3329.0      90
3365.0     733
3386.0     288
3414.0      90
3417.0     230
3711.0     732
3784.0    1464
3904.0    1404
3972.0     732
4184.0     221
4213.0     221
4243.0    1282
4502.0    1274
4962.0     724
```

```
Name: Distance, dtype: int64
Total Outliers for Distance is  17498
--------------------------------------------------------------

21.0     17843
22.0     15213
23.0     13039
24.0     10996
25.0     10148
         ...
213.0        1
225.0        1
233.0        1
240.0        1
308.0        1
Name: TaxiIn, Length: 169, dtype: int64
Total Outliers for TaxiIn is  139580
--------------------------------------------------------------

47.0     9112
48.0     8545
49.0     7716
50.0     7309
51.0     6740
         ...
383.0       1
386.0       1
393.0       2
422.0       1
429.0       1
Name: TaxiOut, Length: 295, dtype: int64
Total Outliers for TaxiOut is  151838
--------------------------------------------------------------

65.0      2152
66.0      1812
67.0      1848
68.0      1799
69.0      1794
          ...
1542.0       1
1552.0       1
1707.0       1
1951.0       1
2436.0       1
Name: CarrierDelay, Length: 919, dtype: int64
Total Outliers for CarrierDelay is  98241
--------------------------------------------------------------

1.0       1869
2.0       2002
3.0       2054
4.0       2077
5.0       2377
          ...
1148.0       1
1153.0       1
1225.0       1
1297.0       1
1352.0       1
Name: WeatherDelay, Length: 598, dtype: int64
Total Outliers for WeatherDelay is  99985
--------------------------------------------------------------

85.0      1162
86.0      1121
87.0      1074
88.0      1031
89.0      1041
          ...
1195.0       1
1207.0       1
1289.0       1
1337.0       1
1357.0       1
Name: NASDelay, Length: 489, dtype: int64
Total Outliers for NASDelay is  58456
```

```
----------------------------------------------------------------

1.0       204
2.0       215
3.0       232
4.0       265
5.0       241
         ...
254.0       1
280.0       1
284.0       1
357.0       1
392.0       1
Name: SecurityDelay, Length: 155, dtype: int64
Total Outliers for SecurityDelay is  6202
----------------------------------------------------------------

105.0     1386
106.0     1275
107.0     1163
108.0     1168
109.0     1164
          ...
1184.0       1
1236.0       1
1254.0       1
1303.0       1
1316.0       1
Name: LateAircraftDelay, Length: 459, dtype: int64
Total Outliers for LateAircraftDelay is  67818
----------------------------------------------------------------
```

From the list above:

- DepTime, CRSDepTime, ArrTime, and CRSArrTime do not contain outliers.
- Features are considered as unsual which are 'ActualElapsedTime', 'CRSElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'TaxiIn', 'TaxiOut', 'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay', and 'LateAircraftDelay'

CRSElapsedTime

In [16]:

```
data[data['CRSElapsedTime'] == 1435.0]
```

Out[16]:

	Month	DayofMonth	DayOfWeek	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay
400846	1	6	7	MQ	3956	NaN	NaN	1435.0	NaN	NaN

1 rows × 24 columns

It should be removed because lack of many information

In [17]:

```
data.drop(axis=0,index=data[data['CRSElapsedTime'] == 1435.0].index,inplace=True)
```

ActualElapsedTime

'ActualElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay', 'TaxiIn', 'TaxiOut', 'CarrierDelay', 'WeatherDelay', 'NASDelay', 'SecurityDelay', and 'LateAircraftDelay' will exam in 50th least frequency values and the first detecting value > 5 which is going to be droped with all value higher than it. Because they can be unsual features or natural, if natural features it also should be droped (very small quantities in dataset).

In [18]:

```
def Remove_Outliers(data,listOutlier):
    dataTmp = data.copy()
    indexRemove = []
    for i in listOutlier:
        leastFiftyRows = dataTmp[i].value_counts().sort_index().iloc[-50:]
        tmp = leastFiftyRows[leastFiftyRows > 5]
        if tmp.tolist() == []:
            startPoint = leastFiftyRows.index.min()
        else:
            startPoint = tmp.index.max()
        indexRemove = indexRemove + dataTmp[i][dataTmp[i] > startPoint].index.tolist()
        print(str(i) +" "+ str( startPoint))
    dataTmp.drop(axis=0, index=indexRemove,inplace=True)
    return dataTmp
OutliersList = ['ActualElapsedTime', 'AirTime', 'ArrDelay', 'DepDelay',
                'TaxiIn', 'TaxiOut', 'CarrierDelay', 'WeatherDelay',
                'NASDelay', 'SecurityDelay','LateAircraftDelay']
dataOutliersRemoved = Remove_Outliers(data,OutliersList)
```

```
ActualElapsedTime 663.0
AirTime 629.0
ArrDelay 1190.0
DepDelay 1192.0
TaxiIn 140.0
TaxiOut 292.0
CarrierDelay 1126.0
WeatherDelay 720.0
NASDelay 589.0
SecurityDelay 115.0
LateAircraftDelay 570.0
```

In [19]:

```
print('Size of data Outliers removed' + str(dataOutliersRemoved.shape))
dataOutliersRemoved.describe(percentiles=[.25, .5, .75, .90, .99])
```

```
Size of data Outliers removed(7009234, 24)
```

Out[19]:

	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut	Ca
count	6.854538e+06	7.008390e+06	6.854538e+06	6.854538e+06	6.872989e+06	7.009234e+06	6.857586e+06	6.872177e+06	1.5
mean	1.273087e+02	1.288597e+02	1.040109e+02	8.128997e+00	9.939952e+00	7.263319e+02	6.858706e+00	1.644907e+01	1.5
std	7.014701e+01	6.938975e+01	6.741380e+01	3.803495e+01	3.484668e+01	5.619534e+02	4.898904e+00	1.128311e+01	3.9
min	1.200000e+01	-1.410000e+02	0.000000e+00	-5.190000e+02	-5.340000e+02	1.100000e+01	0.000000e+00	0.000000e+00	0.0
25%	7.700000e+01	8.000000e+01	5.500000e+01	-1.000000e+01	-4.000000e+00	3.250000e+02	4.000000e+00	1.000000e+01	0.0
50%	1.100000e+02	1.100000e+02	8.600000e+01	-2.000000e+00	-1.000000e+00	5.810000e+02	6.000000e+00	1.400000e+01	0.0
75%	1.570000e+02	1.590000e+02	1.320000e+02	1.200000e+01	8.000000e+00	9.540000e+02	8.000000e+00	1.900000e+01	1.6
90%	2.230000e+02	2.250000e+02	1.970000e+02	4.100000e+01	3.700000e+01	1.518000e+03	1.200000e+01	2.700000e+01	4.5
99%	3.580000e+02	3.600000e+02	3.260000e+02	1.700000e+02	1.630000e+02	2.562000e+03	2.600000e+01	6.100000e+01	1.7
max	6.630000e+02	6.600000e+02	6.290000e+02	1.170000e+03	1.189000e+03	4.962000e+03	1.400000e+02	2.920000e+02	1.1

=> Now data set is more reasonably and its size is 7009234x28.

Check and Fill Missing Values

Check the number of missing values in each column.

In [20]:

```
dataOutliersRemoved.isnull().sum()
```

Out[20]:

```
Month                      0
DayofMonth                 0
DayOfWeek                  0
UniqueCarrier              0
FlightNum                  0
TailNum                83364
ActualElapsedTime     154696
CRSElapsedTime           844
AirTime               154696
ArrDelay              154696
DepDelay              136245
Origin                     0
Dest                       0
Distance                   0
TaxiIn                151648
TaxiOut               137057
Cancelled                  0
CancellationCode     6871801
Diverted                   0
CarrierDelay         5484977
WeatherDelay         5484977
NASDelay             5484977
SecurityDelay        5484977
LateAircraftDelay    5484977
dtype: int64
```

In [21]:

```
def MissValuePercentage(data):
    # Number of missing values in each column
    missingValueColumns = (data.isnull().sum())
    # Find missing column in data
    missingValueColumnsFrame = missingValueColumns[missingValueColumns > 0].to_frame()
    # Rename to 0 to Count
    missingValueColumnsFrame=missingValueColumnsFrame.rename(columns={0:'Count'})
    # add percentage column
    missingValueColumnsFrame['Percentage'] = missingValueColumnsFrame/data.shape[0] * 100
    return missingValueColumnsFrame
missingValueColumnsFrame = MissValuePercentage(dataOutliersRemoved)
missingValueColumnsFrame
```

Out[21]:

	Count	Percentage
TailNum	83364	1.189345
ActualElapsedTime	154696	2.207031
CRSElapsedTime	844	0.012041
AirTime	154696	2.207031
ArrDelay	154696	2.207031
DepDelay	136245	1.943793
TaxiIn	151648	2.163546
TaxiOut	137057	1.955378
CancellationCode	6871801	98.039258
CarrierDelay	5484977	78.253587
WeatherDelay	5484977	78.253587
NASDelay	5484977	78.253587
SecurityDelay	5484977	78.253587
LateAircraftDelay	5484977	78.253587

=> There are 154696 null values in 'AirTime' after remove Outliers

The purpose is to predict delay flights, so the target of prediction is 'ArrDelay'. Therefore, samples with null value of 'ArrDelay' should be removed.

Moreover, percentage of missing values of Features have samples >= 70%, so these features should be remmoved.

In [ ]:

```

```

In [22]:

```
dataFillMiss = dataOutliersRemoved.dropna(axis=0,subset=['ArrDelay'])
FeaturesRemoveList = missingValueColumnsFrame[missingValueColumnsFrame['Percentage'] > 70 ].index
dataFillMiss = dataFillMiss.drop(axis=1, columns=FeaturesRemoveList)
```

In [23]:

```
ttt = dataFillMiss.copy()
def GreaterThirty(num):
    if num > 30:
        return 1
    elif num <= 30:
        return 0
    else:
        return -1
ttt['ArrDelay']=ttt['ArrDelay'].apply(GreaterThirty)
#plt.pie(fracs, labels=labels, autopct='%1.1f%%', shadow=True)
```

In [24]:

```
#plt.pie(['3','2'], autopct='%1.1f%%', shadow=True)
```

In [25]:

```
MissValuePercentage(dataFillMiss)
```

Out[25]:

	Count	Percentage
TailNum	5	0.000073

In [26]:

```
dataFillMiss.dropna(axis=0,subset=['TailNum'],inplace=True)
```

In [27]:

```
dataFillMiss.isnull().sum()
```

Out[27]:

```
Month                0
DayofMonth           0
DayOfWeek            0
UniqueCarrier        0
FlightNum            0
TailNum              0
ActualElapsedTime    0
CRSElapsedTime       0
AirTime              0
ArrDelay             0
DepDelay             0
Origin               0
Dest                 0
Distance             0
TaxiIn               0
TaxiOut              0
Cancelled            0
Diverted             0
```

```
dtype: int64
```

=> Now, There are no any null values in the data set.

In [28]:

```
dataFillMiss.reset_index(inplace = True, drop=True)
```

In [29]:

```
dataFillMiss.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6854533 entries, 0 to 6854532
Data columns (total 18 columns):
Month                    category
DayofMonth               category
DayOfWeek                category
UniqueCarrier            object
FlightNum                category
TailNum                  object
ActualElapsedTime        float64
CRSElapsedTime           float64
AirTime                  float64
ArrDelay                 float64
DepDelay                 float64
Origin                   object
Dest                     object
Distance                 float64
TaxiIn                   float64
TaxiOut                  float64
Cancelled                category
Diverted                 category
dtypes: category(6), float64(8), object(4)
memory usage: 673.7+ MB
```

=>After dealing with missing values, the dataset's size is 6854533x18

Processing Data

Reduce Dataset to 15% by choosing random samples.

In [30]:

```
dataFillMiss = dataFillMiss.sample(frac=0.15, random_state=1)
```

In [31]:

```
dataFillMiss.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1028180 entries, 1193770 to 4478550
Data columns (total 18 columns):
Month                    1028180 non-null category
DayofMonth               1028180 non-null category
DayOfWeek                1028180 non-null category
UniqueCarrier            1028180 non-null object
FlightNum                1028180 non-null category
TailNum                  1028180 non-null object
ActualElapsedTime        1028180 non-null float64
CRSElapsedTime           1028180 non-null float64
AirTime                  1028180 non-null float64
ArrDelay                 1028180 non-null float64
DepDelay                 1028180 non-null float64
Origin                   1028180 non-null object
Dest                     1028180 non-null object
Distance                 1028180 non-null float64
TaxiIn                   1028180 non-null float64
```

```
TaxiOut              1028180 non-null float64
Cancelled            1028180 non-null category
Diverted             1028180 non-null category
dtypes: category(6), float64(8), object(4)
memory usage: 109.2+ MB
```

In [32]:

```
dataFillMiss.reset_index(drop=True,inplace=True)
dataFillMiss
```

Out[32]:

	Month	DayofMonth	DayOfWeek	UniqueCarrier	FlightNum	TailNum	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay
0	3	21	5	WN	134	N655WN	72.0	75.0	61.0	-5.
1	6	1	7	AA	419	N592AA	277.0	270.0	245.0	5.
2	10	21	2	UA	398	N421UA	114.0	126.0	97.0	-22.
3	10	21	2	US	1025	N423US	106.0	110.0	80.0	-11.
4	10	15	3	FL	92	N972AT	101.0	103.0	86.0	-5.
...	...	...	...	...	...	...	...	...	...	..
1028175	4	19	6	XE	3124	N25134	126.0	160.0	112.0	-40.
1028176	11	2	7	EV	5009	N686BR	137.0	145.0	112.0	-5.
1028177	11	10	1	OO	6016	N906SW	125.0	127.0	103.0	-6.
1028178	11	23	7	WN	3855	N311SW	82.0	85.0	71.0	-5.
1028179	8	24	7	F9	629	N920FR	88.0	88.0	62.0	-4.

1028180 rows × 18 columns

=> After reducing, the size of datases is 1028180x18

Selecting the Prediction Target

A flight only counts as late if it is more than 30 minutes late; therefore, values > 30 are assigned to 1 and the others assigned to 0.

In [33]:

```
def TargetConver(num):
    if num > 30:
        return 1
    elif num <= 30:
        return 0
    else:
        return -1
dataFillMiss['ArrDelay'] = dataFillMiss['ArrDelay'].apply(TargetConver)
y = dataFillMiss['ArrDelay']
```

In [34]:

```
print(y.value_counts())
print('percentage of non-delay flight: ' ,round(y.value_counts()[0]/len(y) * 100,2) ,'%')
print('percentage of delay flight     :  ',round(y.value_counts()[1]/len(y) * 100,2) ,'%')
```

```
0    892784
1    135396
Name: ArrDelay, dtype: int64
percentage of non-delay flight:  86.83 %
percentage of delay flight     :   13.17 %
```

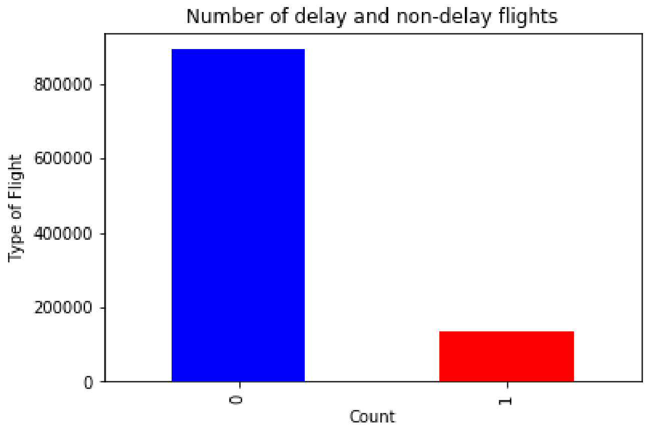
In [35]:

```
countTarget = pd.value_counts(y, sort = True).sort_index()
colors = ["#0101DF", "#DF0101"]
countTarget.plot(kind = 'bar', color=['blue', 'red'])
```

```
countTarget.plot(kind = 'bar',color=['blue', 'red'])
plt.title("Number of delay and non-delay flights")
plt.xlabel("Count")
plt.ylabel("Type of Flight")
```

Out[35]:

```
Text(0, 0.5, 'Type of Flight')
```



Note: The dataset is imbalanced. Most of the flights are non-delay. If we use this dataframe as the base for our predictive models and analysis we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most flights are non-delay.

=> To deal with imbalance robust alogorithim solving this issuse such as RandomForest... will be use, and for logistic regresion tuning model to choose the best result.

Feature Selections

Split the dataset into Numerical and Categorical features, in order to choose features can affect the delay flights.

In [36]:

```
#Seperate categorical from numerical data
numerical = dataFillMiss.select_dtypes(include = [np.number])
categorical = dataFillMiss.select_dtypes(exclude = [np.number])
listSelectedFeatures = []
```

NUMERICAL FEATURES

In [37]:

```
numerical.describe()
```

Out[37]:

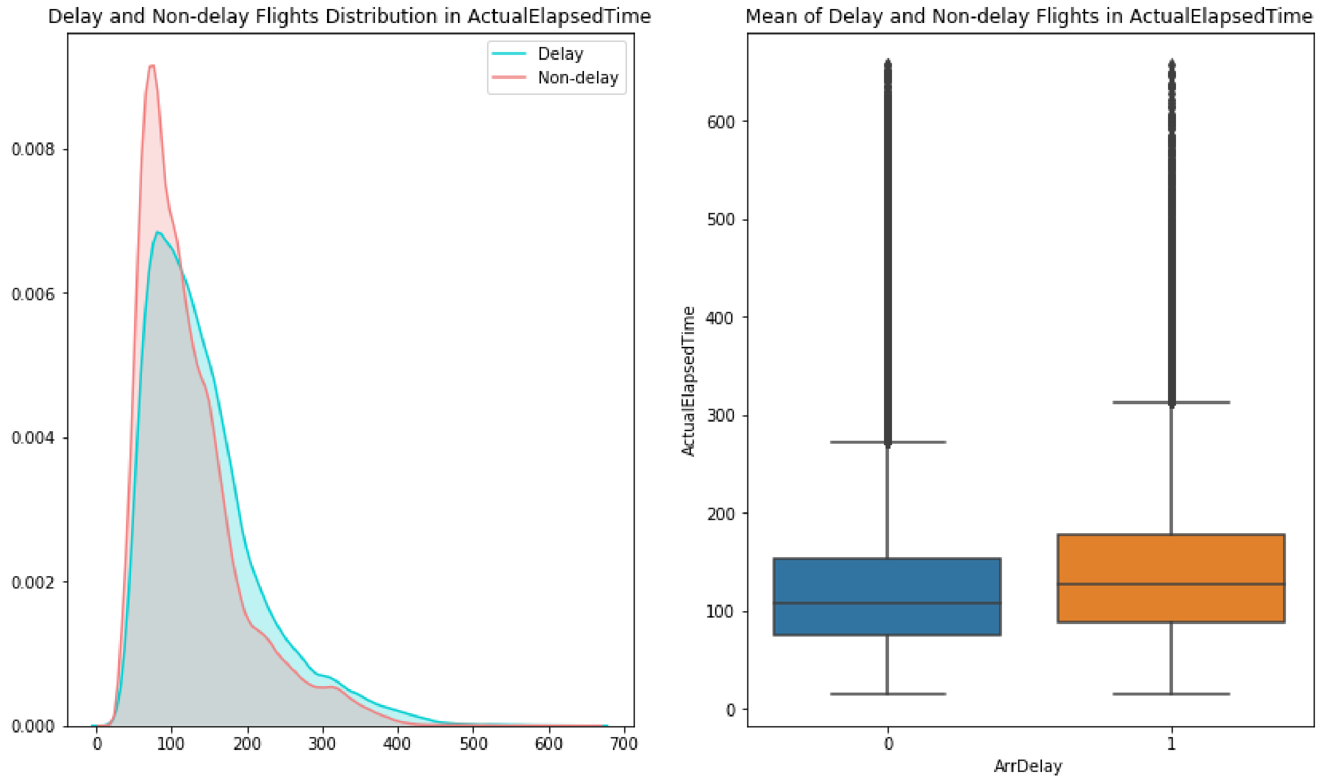
	ActualElapsedTime	CRSElapsedTime	AirTime	ArrDelay	DepDelay	Distance	TaxiIn	TaxiOut
count	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06
mean	1.273279e+02	1.290865e+02	1.040128e+02	1.316851e-01	9.895836e+00	7.286988e+02	6.860146e+00	1.645487e+01
std	7.015911e+01	6.953945e+01	6.743251e+01	3.381483e-01	3.481117e+01	5.632204e+02	4.903184e+00	1.128949e+01
min	1.500000e+01	-1.410000e+02	0.000000e+00	0.000000e+00	-5.340000e+02	2.400000e+01	0.000000e+00	0.000000e+00
25%	7.700000e+01	8.000000e+01	5.500000e+01	0.000000e+00	-4.000000e+00	3.260000e+02	4.000000e+00	1.000000e+01
50%	1.100000e+02	1.110000e+02	8.600000e+01	0.000000e+00	-1.000000e+00	5.810000e+02	6.000000e+00	1.400000e+01
75%	1.570000e+02	1.590000e+02	1.320000e+02	0.000000e+00	8.000000e+00	9.540000e+02	8.000000e+00	1.900000e+01
max	6.580000e+02	6.600000e+02	6.290000e+02	1.000000e+00	1.172000e+03	4.962000e+03	1.370000e+02	2.890000e+02

In [38]:

```
def NumericalPlot(target,feature,data,label=None,legend=None,title=None):

    '''target: Name of target (str)
     feature: Name of feature (str)
     data: dataFrame
     lable: list of lables for distribution(x,y) and boxplot(x,y)
     legend: for distribution
     title: list of title distribution and boxplot'''
    plt.figure(figsize=(14,8))

    plt.subplot(1,2,1)
    sns.kdeplot(data[feature][data[target] == 1], color="darkturquoise", shade=True)
    sns.kdeplot(data[feature][data[target] == 0], color="lightcoral", shade=True)
    #plt.xlabel('Age')
    plt.legend(legend)
    plt.title(title[0] + ' in ' + feature)
    #plt.xlabel('sdd')
    #plt.ylabel('sdd')
    #plt.xlim(-10,85)

    plt.subplot(1,2,2)
    plt.title(title[1] + ' in ' + feature)
    sns.boxplot(x=target, y=feature, data=data)
    plt.show()
```

In [39]:

```
NumericalPlot(target='ArrDelay',feature='ActualElapsedTime',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```



From these graph above, There are no large differences between Delay and Non-delay on ActualElapsedTime. 0 < ActualElapsedTime < 150 Non-Delay is higher delay and vicersa for remain. The mean of ActualElapsedTime in Delay is slightly higher than Non-delay

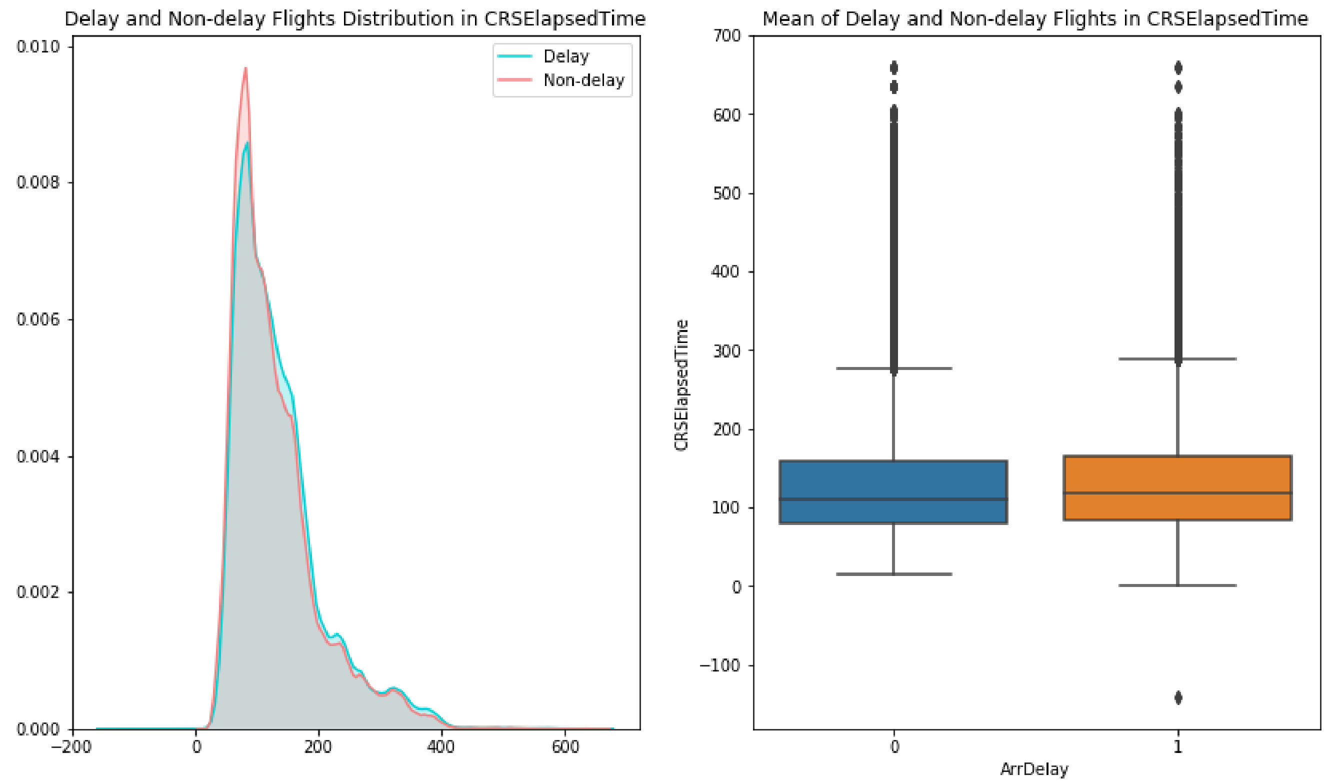
In [40]:

```
listSelectedFeatures.append('ActualElapsedTime')
```

In [41]:

```
NumericalPlot(target='ArrDelay',feature='CRSElapsedTime',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
```

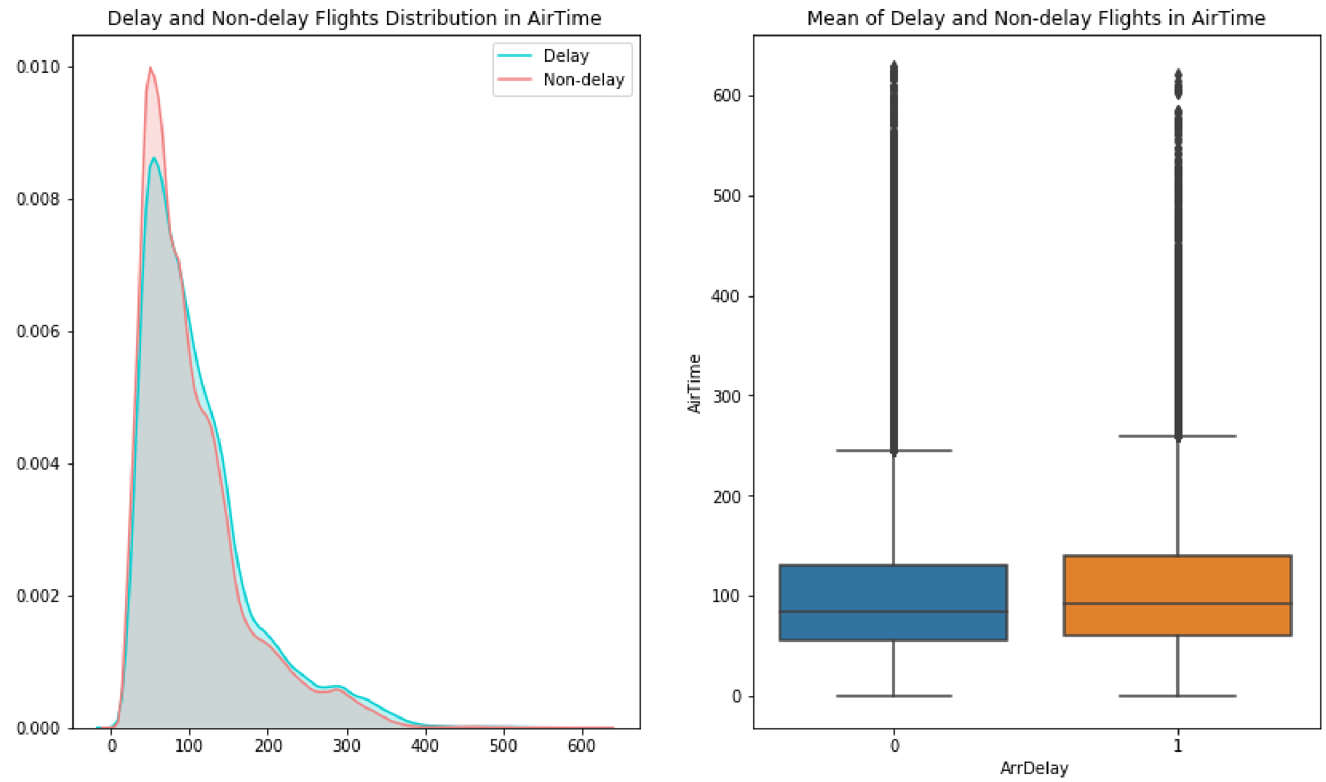
```
            legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```



In CRSElapsedTime, Delay and Non-delay are very similar in both distribution and mean.

In [42]:

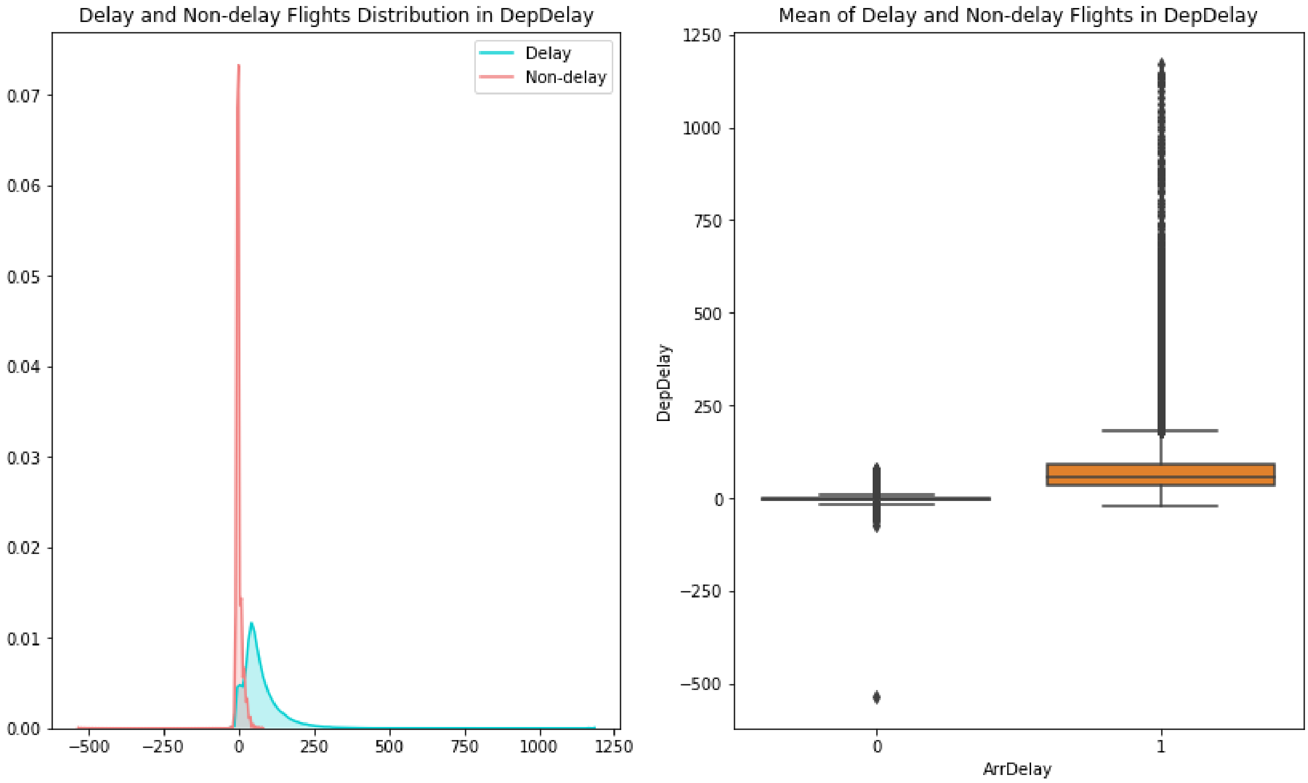
```
NumericalPlot(target='ArrDelay',feature='AirTime',data=numerical,label = None,
            legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```



In AirTime, Delay and Non-delay are very similar in both distribution and mean.

In [43]:

```
NumericalPlot(target='ArrDelay',feature='DepDelay',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```



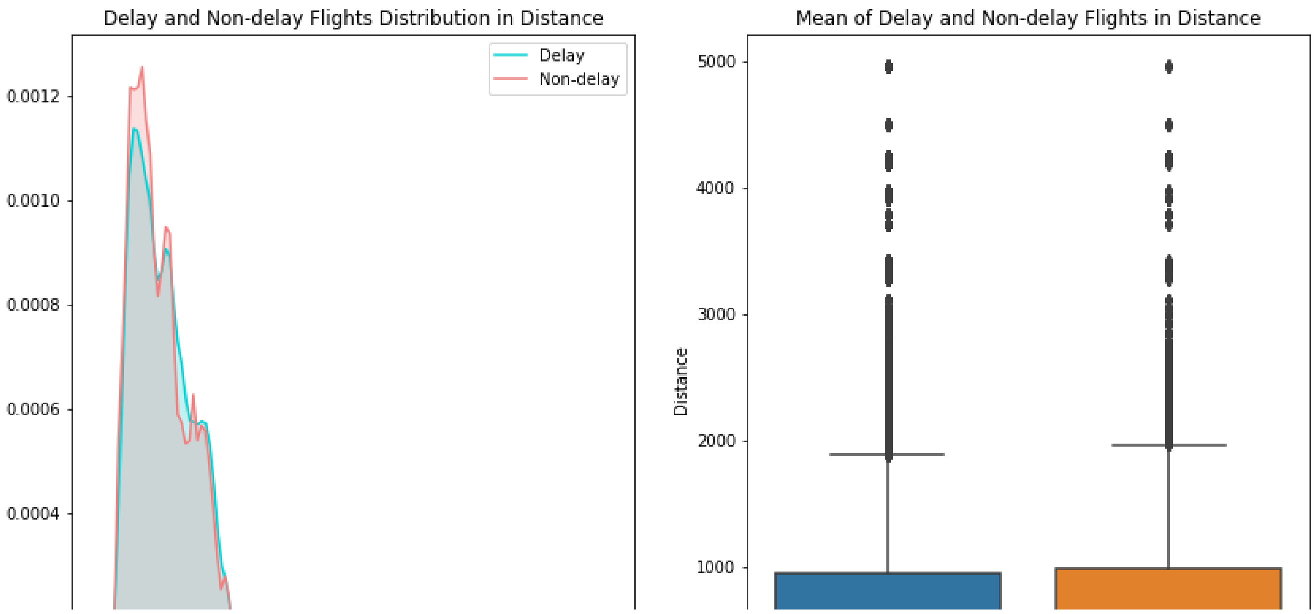
There are significant differences of Delay and Non-delay in DepDelay. In delay, most of DeDelay is 0 to 100 min, while in non-delay, it is around 0. Moreover, mean of DepDelay in Non-delay is nearly zero, whereas another is approximately 50 min. It can conclude that DepDelay has relation with ArrDelay (No Depdelay nearly no Arrdelay).

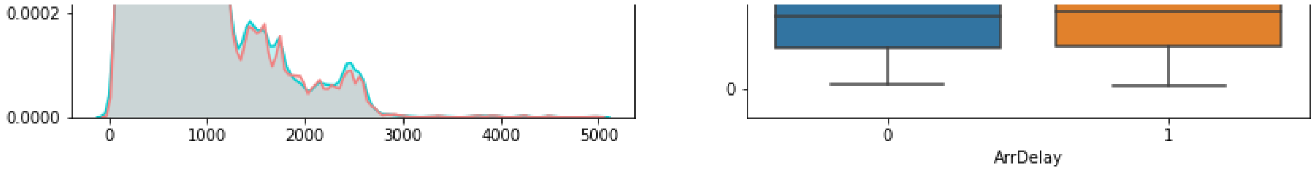
In [44]:

```
listSelectedFeatures.append('DepDelay')
```

In [45]:

```
NumericalPlot(target='ArrDelay',feature='Distance',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```





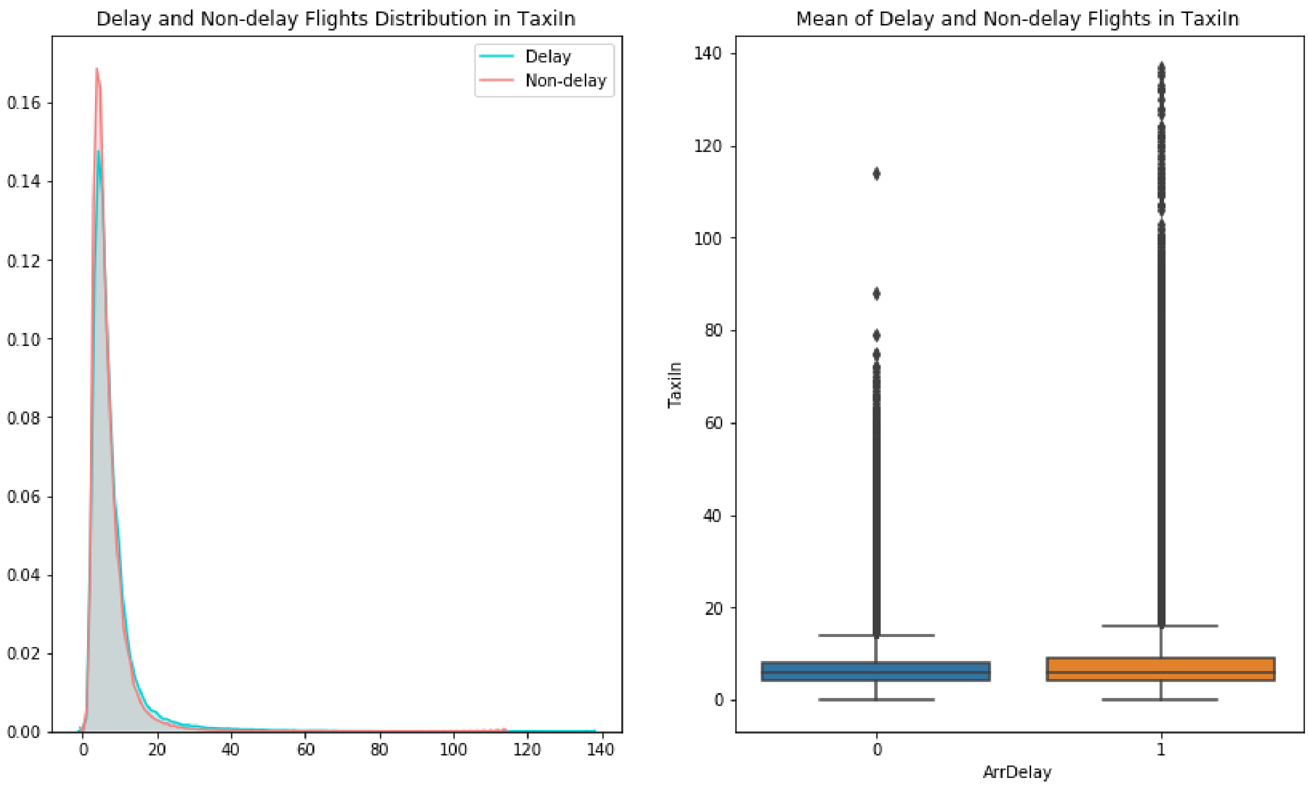
In Distance, Delay and Non-delay are small differences in both distribution and mean.

In [46]:

```
listSelectedFeatures.append('Distance')
```

In [47]:

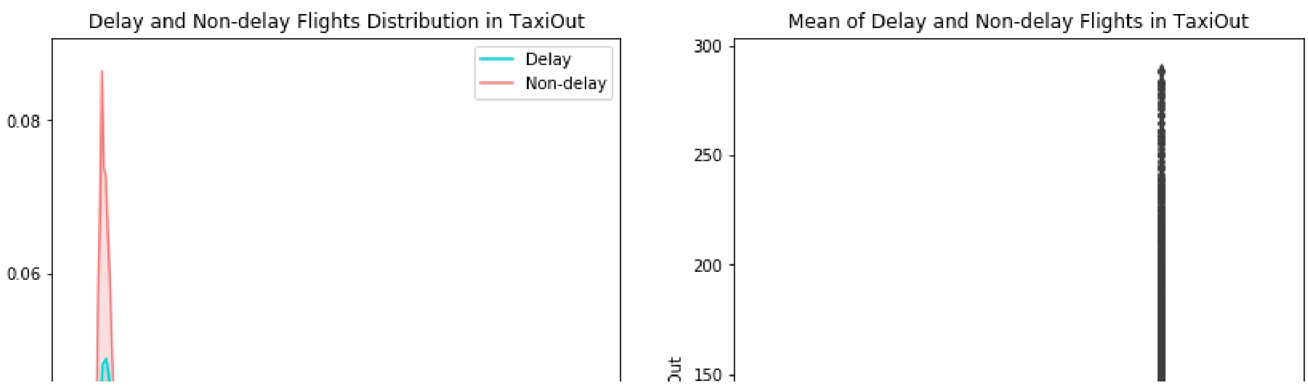
```
NumericalPlot(target='ArrDelay',feature='TaxiIn',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```

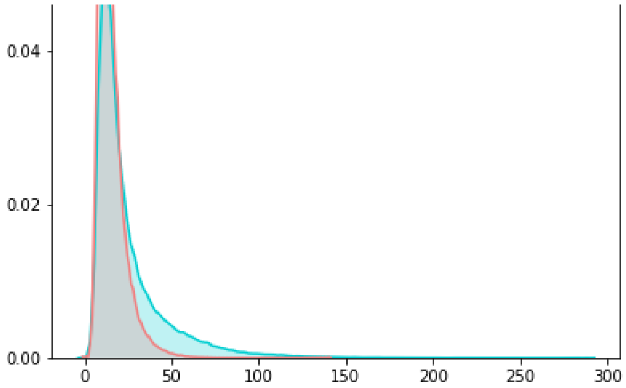


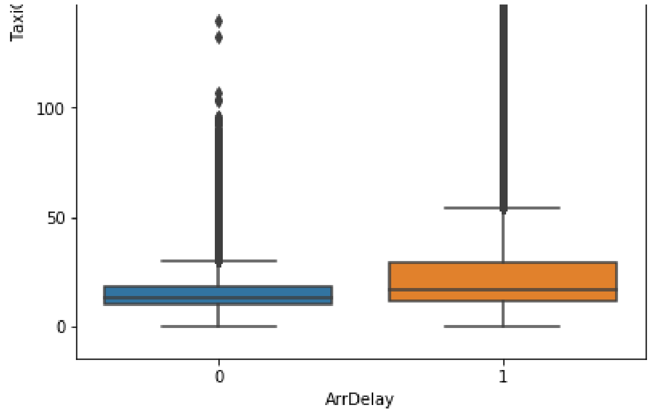
In TaxiIn, Delay and Non-delay are very similar in both distribution and mean.

In [48]:

```
NumericalPlot(target='ArrDelay',feature='TaxiOut',data=numerical,label = None,
              legend=['Delay','Non-delay'],title=['Delay and Non-delay Flights Distribution', 'Mean
of Delay and Non-delay Flights'])
```







There are differences of Delay and Non-delay in DepDelay. Distribution of Delay seem to wider than Non-delay (over 40 to 100 min of TaxiOut is seem to delay)

In [49]:

```
listSelectedFeatures.append('TaxiOut')
```

In [50]:

```
listSelectedFeatures
```

Out[50]:

```
['ActualElapsedTime', 'DepDelay', 'Distance', 'TaxiOut']
```

=> There are 4 Numerical Features selected

CATEGORICAL FEATURES

In [51]:

```
categorical.describe()
```

Out[51]:

	Month	DayofMonth	DayOfWeek	UniqueCarrier	FlightNum	TailNum	Origin	Dest	Cancelled	Diverted
count	1028180	1028180	1028180	1028180	1028180	1028180	1028180	1028180	1028180	1028180
unique	12	31	7	20	7479	5331	302	302	1	1
top	7	3	3	WN	152	N477HA	ATL	ATL	0	0
freq	92303	34622	152585	177984	741	697	61220	61263	1028180	1028180

In [52]:

```
categorical['ArrDelay'] = numerical['ArrDelay'].copy()
```

```
C:\Users\Annie Nguyen\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

'FlightNum' and 'TailNum' are removed because they cannot affect the delay

'Cancelled' and 'Diverted' are dropped beacause they have only 1 value

In [53]:

```
categorical.drop(axis=1,columns=['FlightNum','TailNum','Diverted','Cancelled'],inplace=True)
categorical.head()
```

```
C:\Users\Annie Nguyen\Anaconda3\lib\site-packages\pandas\core\frame.py:4102:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
```

Out[53]:

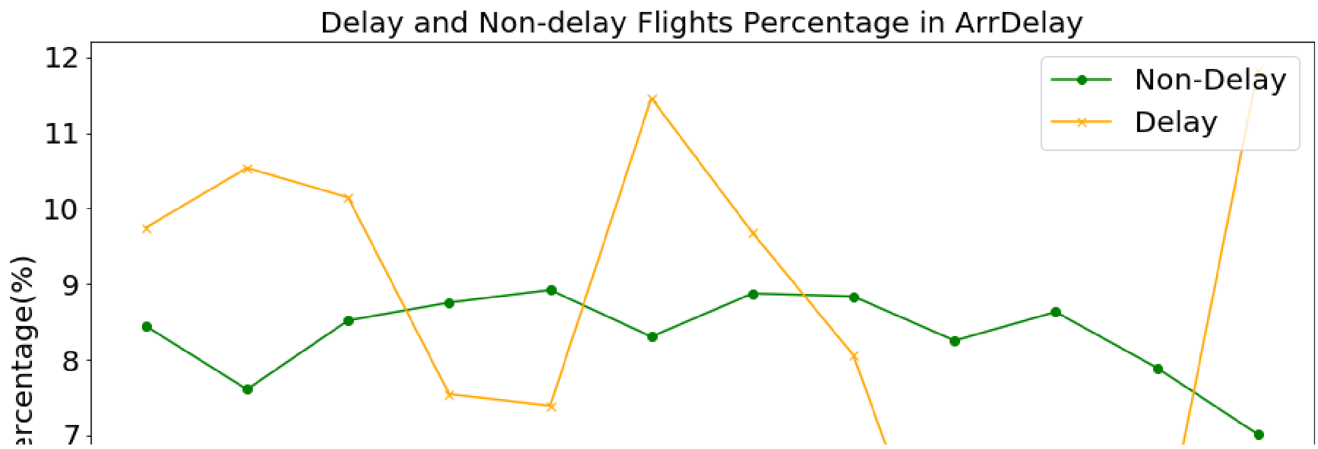
	Month	DayofMonth	DayOfWeek	UniqueCarrier	Origin	Dest	ArrDelay
0	3	21	5	WN	DAL	MSY	0
1	6	1	7	AA	ORD	SEA	0
2	10	21	2	UA	ORD	BDL	0
3	10	21	2	US	EWR	CLT	0
4	10	15	3	FL	ATL	FLL	0

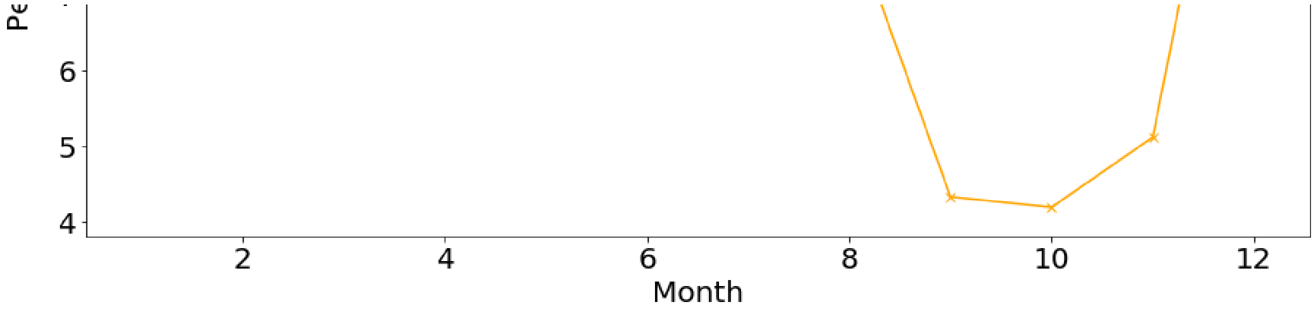
In [54]:

```
def CategoricalPlot(target,feature,data,label=None,legend=None,title=None,percentagePrint=False):
    '''target: Name of target (str)
     feature: Name of feature (str)
     data: dataFrame
     lable:  lables for (x,y)
     legend: list for two line
     title: title of line'''
    FeatureCount      = data.groupby(target)[feature].value_counts().sort_index()
    FeatureCountNo    = FeatureCount[0]/data[target].value_counts()[0]*100
    FeatureCountYes   = FeatureCount[1]/data[target].value_counts()[1]*100
    if(percentagePrint):
        print('Non-delay: ',FeatureCountNo , '%' )
        print('Delay: ',FeatureCountYes , '%' )

    plt.figure(figsize=(15,8)) # figure size
    plt.plot(FeatureCountNo, 'o-', color='g',label=legend[0])
    plt.plot(FeatureCountYes, 'x-', color='orange',label=legend[1])
    plt.xlabel(label[0],fontsize=20)
    plt.xticks(fontsize=20) ## Major tick lable size
    plt.ylabel('Percentage(%)',fontsize=20)
    plt.yticks(fontsize=20) ## Major tick lable size
    plt.title(title + ' in '+ target,fontsize=20)
    plt.legend(loc='upper right',fontsize=20)
    plt.show()
```

In [55]:

```
CategoricalPlot(target='ArrDelay',feature='Month',data=categorical,label = ['Month'],
              legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```





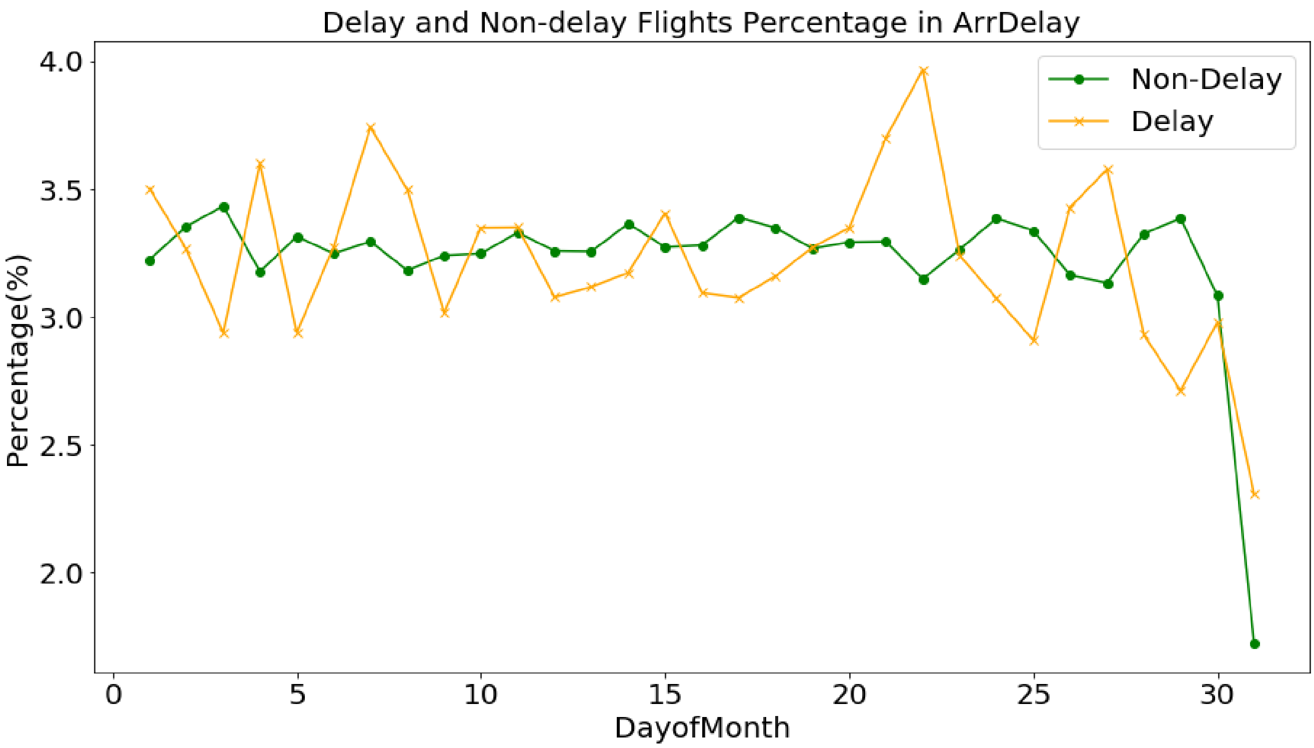
The line percentage of Non-Delay and delay are not neary equal so Month can affect Delay of flights. For example from 6 to 10 the % of delay decrase rapidly (may be this period is the end of summer so people seem to don't travle leading less flights -> less delay) from 10 to 12 this period rocket to peak of all months (this period is ready for Chirstmas and New year)

In [56]:

```
listSelectedFeatures.append('Month')
```

In [57]:

```
CategoricalPlot(target='ArrDelay',feature='DayofMonth',data=categorical,label = ['DayofMonth'],
             legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```



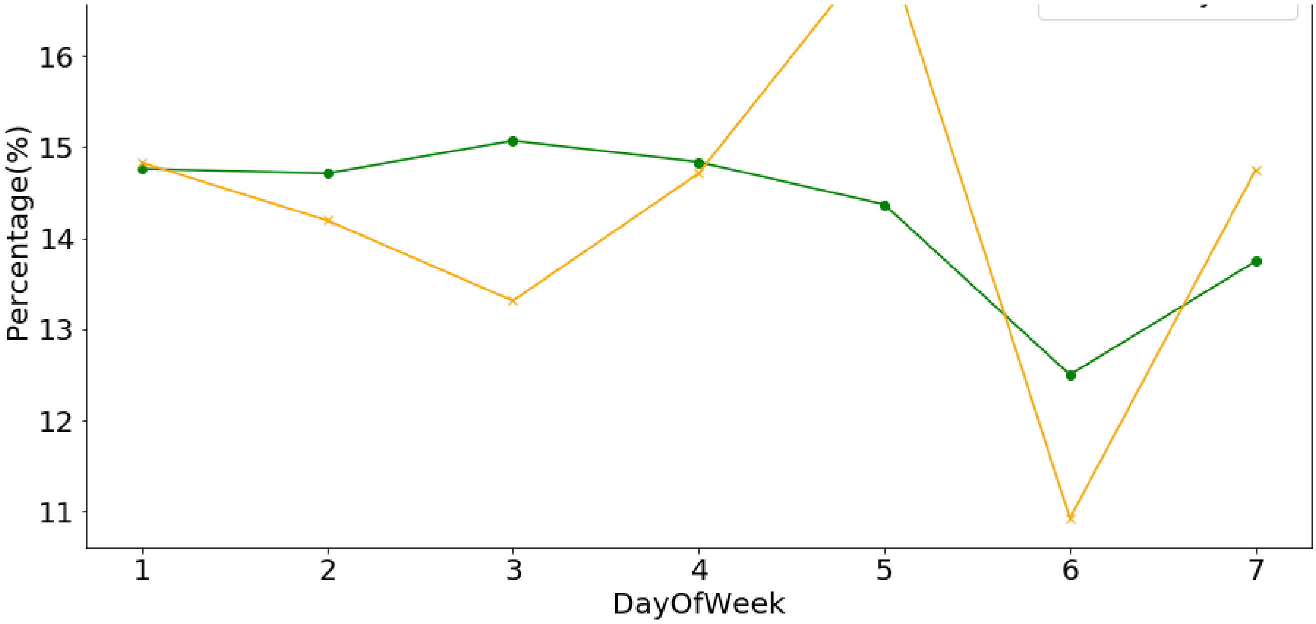
**The line percentage of Non-Delay and delay are not neary equal so DayofMonth can affect Delay of flights

In [58]:

```
listSelectedFeatures.append('DayofMonth')
```

In [59]:

```
CategoricalPlot(target='ArrDelay',feature='DayOfWeek',data=categorical,label = ['DayOfWeek'],
             legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```



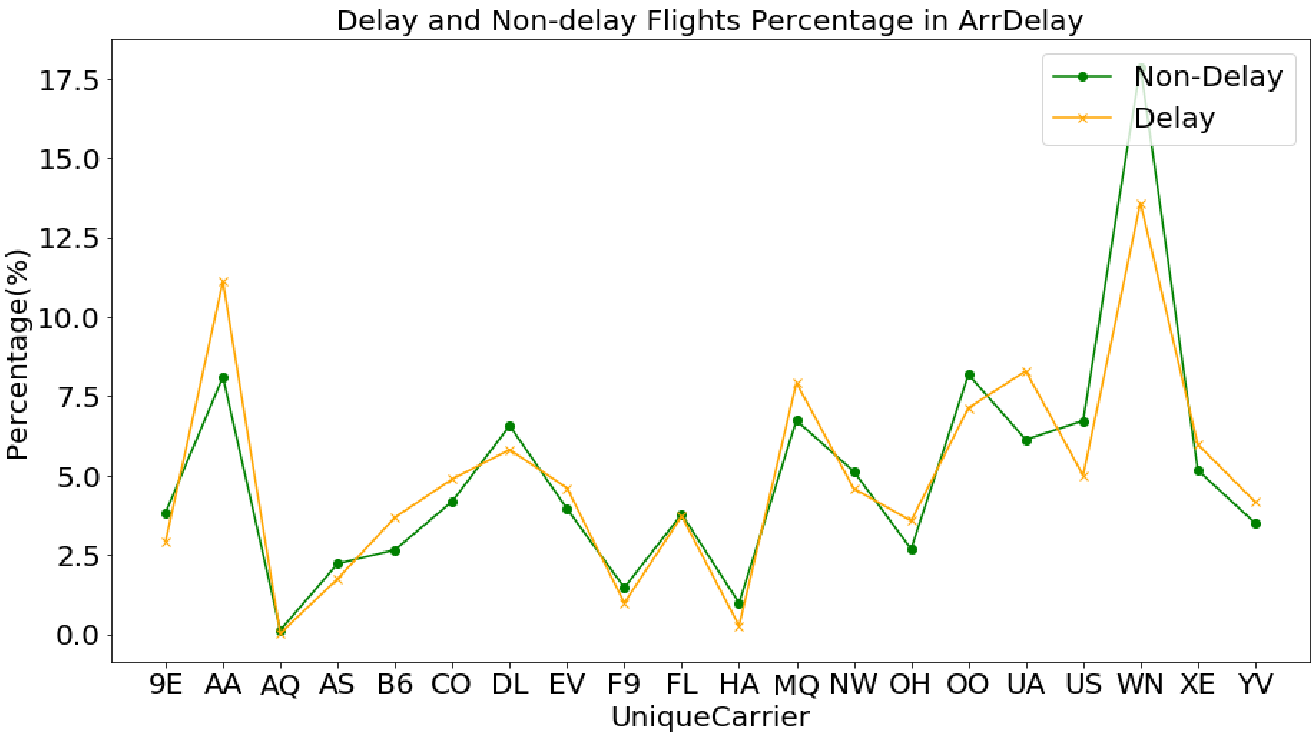
The line percentage of Non-Delay and delay are not neary equal so DayOfWeek can affect Delay of flights.

In [60]:

```
listSelectedFeatures.append('DayOfWeek')
```

In [61]:

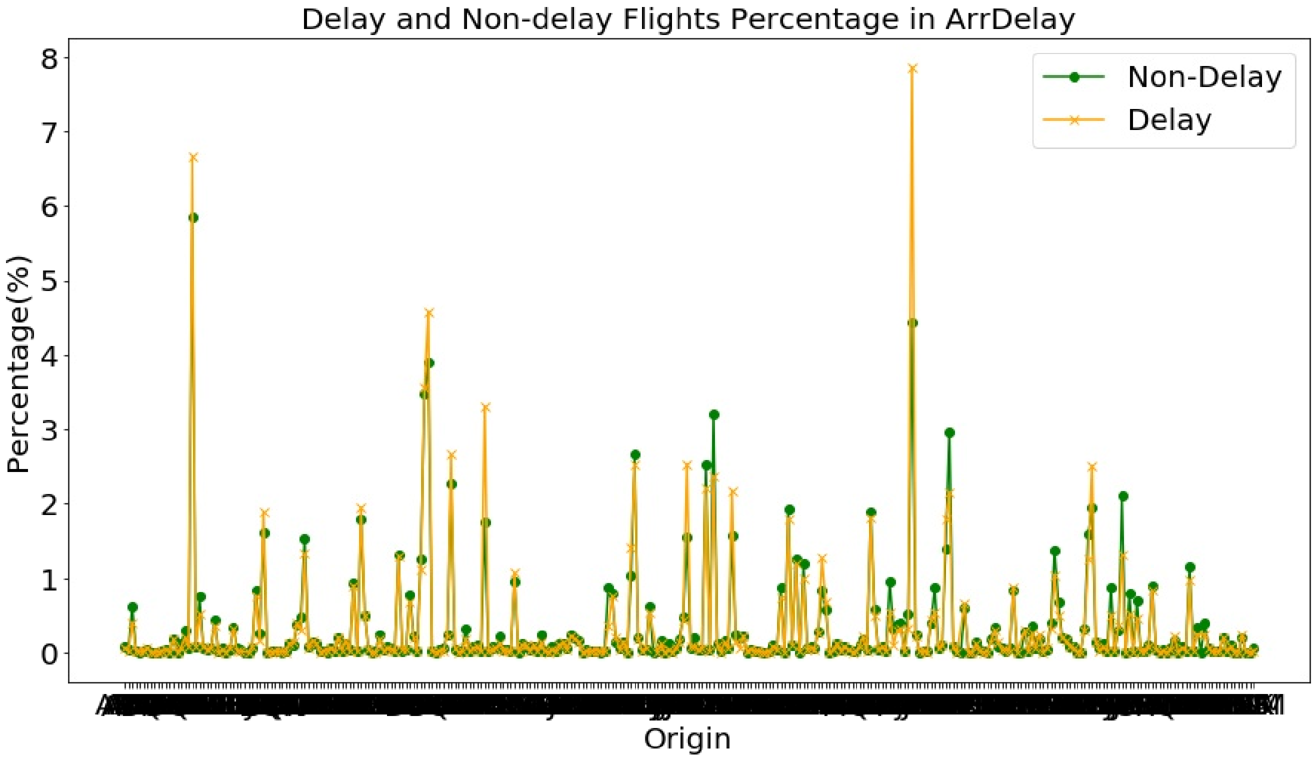
```
CategoricalPlot(target='ArrDelay',feature='UniqueCarrier',data=categorical,label =
['UniqueCarrier'],
              legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```



The line percentage of Non-Delay and delay are neary equal so UniqueCarrier cannot affect Delay of flights.

In [62]:

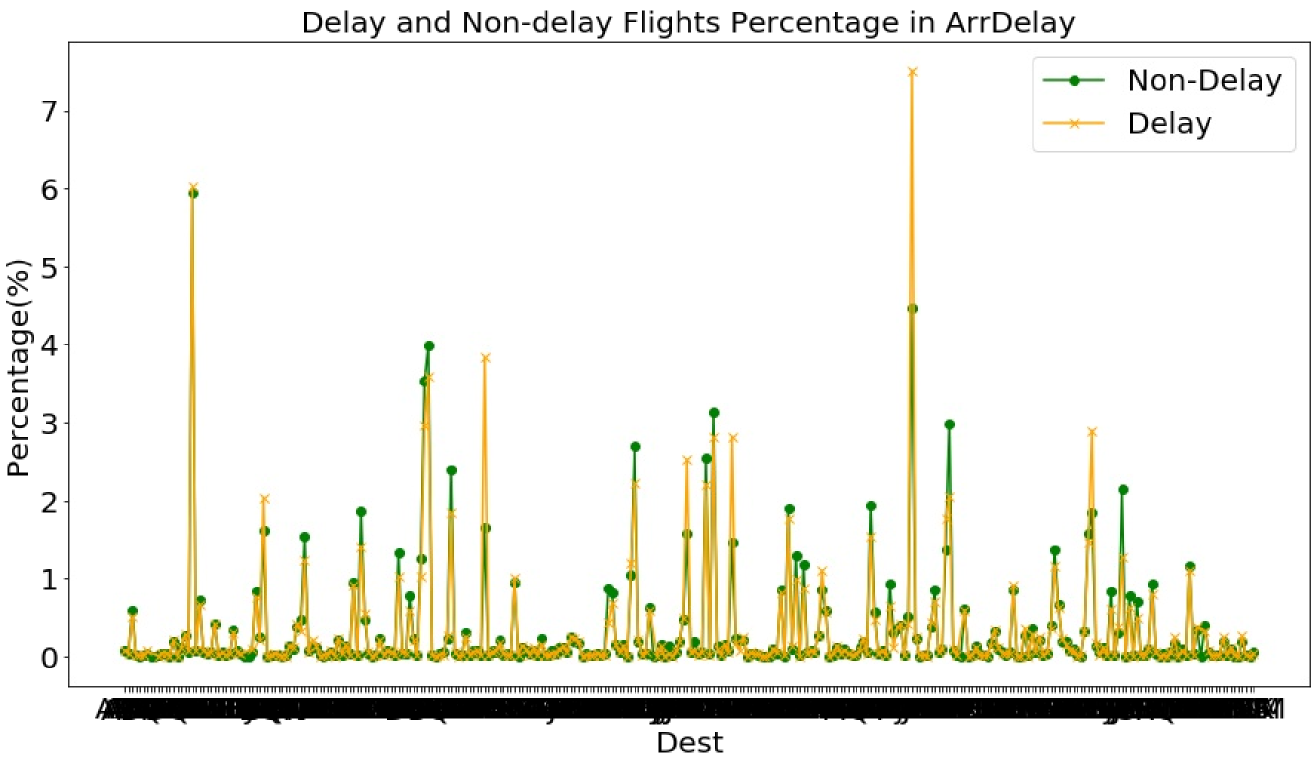
```
CategoricalPlot(target='ArrDelay',feature='Origin',data=categorical,label = ['Origin'],
              legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```



The line percentage of Non-Delay and delay are neary equal so Origin cannot affect Delay of flights.

In [63]:

```
CategoricalPlot(target='ArrDelay',feature='Dest',data=categorical,label = ['Dest'],
                legend=['Non-Delay','Delay'],title='Delay and Non-delay Flights Percentage')
```



The line percentage of Non-Delay and delay are neary equal so Dest cannot affect Delay of flights.

In [64]:

```
listSelectedFeatures
```

Out[64]:

```
['ActualElapsedTime',
 'DepDelay',
 'Distance',
 'TaxiOut',
 'Month',
 'DayofMonth',
 'DayOfWeek']
```

Selected Features DataFrame.

In [65]:

```
dataSelected = pd.DataFrame()
dataSelected = numerical[listSelectedFeatures[:4]].copy()
dataSelected = pd.concat([dataSelected,categorical[listSelectedFeatures[4:]]],axis=1)
dataSelected
```

Out[65]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month	DayofMonth	DayOfWeek
0	72.0	-2.0	437.0	8.0	3	21	5
1	277.0	-2.0	1721.0	22.0	6	1	7
2	114.0	-10.0	783.0	14.0	10	21	2
3	106.0	-7.0	529.0	13.0	10	21	2
4	101.0	-3.0	581.0	11.0	10	15	3
...	...	...	...	...	...	...	...
1028175	126.0	-6.0	837.0	11.0	4	19	6
1028176	137.0	3.0	780.0	12.0	11	2	7
1028177	125.0	-4.0	738.0	17.0	11	10	1
1028178	82.0	-2.0	480.0	9.0	11	23	7
1028179	88.0	-4.0	391.0	9.0	8	24	7

1028180 rows × 7 columns

Convert Categorical Features by Using One-hot Encoding

This coversion is use for logistic regression, other algorithm will use label encoder to conver categorical

In [66]:

```
categoricalFilted = dataSelected[listSelectedFeatures[4:]].copy()
```

In [67]:

```
categoricalFilted.describe()
```

Out[67]:

	Month	DayofMonth	DayOfWeek
count	1028180	1028180	1028180
unique	12	31	7
top	7	3	3
freq	92303	34622	152585

Only opt for category features <= 500 . **Choose all of them**

Convert opted Categorical features using one-hot and coding

In [68]:

```
print(categoricalFilted.columns)
categorical_OneHot = pd.DataFrame()
for i in categoricalFilted:
    #print(categoricalFilted[i])
    categorical_OneHot = pd.concat( [categorical_OneHot,
                                     pd.get_dummies(categoricalFilted[i], drop_first=True,prefix=i)
, axis=1)
categorical_OneHot.head()
```

```
Index(['Month', 'DayofMonth', 'DayOfWeek'], dtype='object')
```

Out[68]:

	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8	Month_9	Month_10	Month_11	...	DayofMonth_28	DayofMont
0	0	1	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	1	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	1	0	...	0	
3	0	0	0	0	0	0	0	0	1	0	...	0	
4	0	0	0	0	0	0	0	0	1	0	...	0	

5 rows × 47 columns

Get the final data afeter to go through model.

In [69]:

```
dataNoNull_withOneHot = pd.DataFrame()
dataNoNull_withOneHot = numerical[listSelectedFeatures[:4]].copy()
dataNoNull_withOneHot = pd.concat([dataNoNull_withOneHot,categorical_OneHot],axis=1)
dataNoNull_withOneHot
```

Out[69]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	...	DayofMonth_
0	72.0	-2.0	437.0	8.0	0	1	0	0	0	0	...	
1	277.0	-2.0	1721.0	22.0	0	0	0	0	1	0	...	
2	114.0	-10.0	783.0	14.0	0	0	0	0	0	0	...	
3	106.0	-7.0	529.0	13.0	0	0	0	0	0	0	...	
4	101.0	-3.0	581.0	11.0	0	0	0	0	0	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1028175	126.0	-6.0	837.0	11.0	0	0	1	0	0	0	...	
1028176	137.0	3.0	780.0	12.0	0	0	0	0	0	0	...	
1028177	125.0	-4.0	738.0	17.0	0	0	0	0	0	0	...	
1028178	82.0	-2.0	480.0	9.0	0	0	0	0	0	0	...	
1028179	88.0	-4.0	391.0	9.0	0	0	0	0	0	0	...	

1028180 rows × 51 columns

In [70]:

```
X = dataNoNull_withOneHot.copy()
```

Scaling Features

Step 1: convert the column of a dataframe to float

Step 2: create a min max processing object. Pass the float column to the min_max_scaler() which scales the dataframe by processing it as shown below m Step 3: Convert the scaled array to the dataframe.

-> The main purpose is to make the gradient descent more quickly. The formula is $\frac{x-min}{max-min}$

In [71]:

```
#Step1 All ready float
#Step2
mm_scaler = preprocessing.MinMaxScaler(feature_range=(0.01, 0.99))
#Step3
X_OneHotMinMax = pd.DataFrame(mm_scaler.fit_transform(X))
X_OneHotMinMax.columns = X.columns

X_OneHotMinMax.describe()
```

Out[71]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month_2	Month_3	Month_4	Month_5	M
count	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.0281
mean	1.811995e-01	3.224372e-01	1.498552e-01	6.579851e-02	8.828715e-02	9.554245e-02	9.420329e-02	9.541950e-02	9.5407
std	1.069299e-01	1.999704e-02	1.117772e-01	3.828270e-02	2.656927e-01	2.766119e-01	2.746436e-01	2.764321e-01	2.7641
min	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.0000
25%	1.044946e-01	3.144549e-01	6.993520e-02	4.391003e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.0000
50%	1.547900e-01	3.161782e-01	1.205427e-01	5.747405e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.0000
75%	2.264230e-01	3.213482e-01	1.945687e-01	7.442907e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.000000e-02	1.0000
max	9.900000e-01	9.900000e-01	9.900000e-01	9.900000e-01	9.900000e-01	9.900000e-01	9.900000e-01	9.900000e-01	9.9000

8 rows × 51 columns

Cross Validation Technique

This is to check and validate the data when running the code multiple times. Setting random_state a fixed value will guarantee that same sequence of random numbers are generated each time you run the code. And unless there is some other randomness present in the process, the results produced will be same as always. This helps in verifying the output.

20% test set and 80% train set.

In [72]:

```
XTrainHoldout, XTestHoldout, yTrainHoldout, yTestHoldout = train_test_split(X_OneHotMinMax, y, test
_size = 0.2, random_state = 10)
XTrainHoldout
```

Out[72]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	...	DayofMonth_
60143	0.107543	0.313880	0.074103	0.101557	0.01	0.01	0.01	0.99	0.01	0.01	...	0
261230	0.066392	0.314455	0.039968	0.040519	0.01	0.01	0.01	0.01	0.01	0.01	...	0
160524	0.145645	0.315604	0.095140	0.064256	0.01	0.01	0.01	0.01	0.99	0.01	...	0
894997	0.170031	0.317902	0.150510	0.064256	0.01	0.01	0.01	0.01	0.01	0.01	...	0
530734	0.269098	0.315604	0.246367	0.043910	0.01	0.01	0.99	0.01	0.01	0.01	...	0
...	...	...	...	...	...	...	...	...	...	...	...	
617841	0.188320	0.316178	0.141183	0.091384	0.01	0.01	0.01	0.01	0.01	0.01	...	0
443712	0.503810	0.320199	0.516472	0.121903	0.01	0.01	0.01	0.01	0.01	0.99	...	0
881167	0.374261	0.316753	0.347780	0.098166	0.99	0.01	0.01	0.01	0.01	0.01	...	0
760957	0.124308	0.317327	0.110024	0.050692	0.01	0.01	0.01	0.01	0.01	0.01	...	0
345353	0.445894	0.337433	0.490871	0.040519	0.01	0.01	0.01	0.01	0.99	0.01	...	0

822544 rows × 51 columns

822344 rows × 31 columns

Apply models in Logistic Regression, Naïve Bayes, Decision Tree, Random Forest, Gradient Boosting for Prediction

Robust algorithm dealing with imbalanced data as : NB, DT, RF, GB use Lable Encoder for categorical features, while Logistic Regression use One-Hot encoding.

Convert Categorical Features by Using Label Encoding

In [73]:

```
BayesdataSelected = dataSelected.copy()
BayesdataSelected.describe()
```

Out[73]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut
count	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06
mean	1.273279e+02	9.895836e+00	7.286988e+02	1.645487e+01
std	7.015911e+01	3.481117e+01	5.632204e+02	1.128949e+01
min	1.500000e+01	-5.340000e+02	2.400000e+01	0.000000e+00
25%	7.700000e+01	-4.000000e+00	3.260000e+02	1.000000e+01
50%	1.100000e+02	-1.000000e+00	5.810000e+02	1.400000e+01
75%	1.570000e+02	8.000000e+00	9.540000e+02	1.900000e+01
max	6.580000e+02	1.172000e+03	4.962000e+03	2.890000e+02

First of all, Numerical features should be convert to categorical features, and it is based on distribuion of graph above in **section 3.1.**

ActualElapsedTime

- U100 min < 100
- U200 100 <= min < 200
- 0200min >= 200

DepDelay:

- EarlyU10 : (min < -60)
- EarlyU1 : (-60 <= min < 0)
- Intime : (min = 0)
- LateU1 : (0 < min < 60)
- LateU10 : (60 <= min < 600)
- LateU20 : (>=600)

Distance

- S : (miles < 300)
- M : (300 <= miles < 600)
- L : (600 <= miles < 1500)
- XL : (1500 <= miles)

TaxiOut

- U30 : (min < 30)
- U1 : (30 < min < 60
- O1 : (min >= 60)

In [74]:

```
def ActualElapsedTime(minutes):
    if (minutes < 100) :
        return 'U100'
```

```
    elif ((minutes >= 100) and (minutes < 200)):
        return 'U200'
    elif (minutes >= 200):
        return 'O200'
    else:
        return -1

def DepDelay(minutes):
    if (minutes < -60):
        return 'EarlyU10'
    elif ((minutes >= -60) and (minutes < 0)):
        return 'EarlyU1'
    elif (minutes == 0):
        return 'Intime'
    elif ((minutes > 0) and (minutes < 60)):
        return 'LateU1'
    elif ((minutes >= 60) and (minutes < 600)):
        return  'LateU10'
    elif (minutes >= 600):
        return 'LateU20'
    else:
        return -1

def Distance(miles):
    if (miles < 300) :
        return 'S'
    elif ((miles >= 300) and (miles < 600)):
        return 'M'
    elif ((miles >= 600) and (miles < 1500)):
        return 'L'
    elif (miles >= 1500):
        return 'XL'
    else:
        return -1

def TaxiOut(minutes):
    if (minutes < 30) :
        return 'U30'
    elif ((minutes >= 30) and (minutes < 60)):
        return 'U1'
    elif (minutes >= 60):
        return 'O1'
    else:
        return -1
```

In [75]:

```
BayesdataSelected['ActualElapsedTime'] =
BayesdataSelected['ActualElapsedTime'].apply(ActualElapsedTime)
BayesdataSelected['DepDelay'] = BayesdataSelected['DepDelay'].apply(DepDelay)
BayesdataSelected['Distance'] = BayesdataSelected['Distance'].apply(Distance)
BayesdataSelected['TaxiOut'] = BayesdataSelected['TaxiOut'].apply(TaxiOut)
```

In [76]:

```
print(BayesdataSelected['ActualElapsedTime'].value_counts())
print(BayesdataSelected['DepDelay'].value_counts())
print(BayesdataSelected['Distance'].value_counts())
print(BayesdataSelected['TaxiOut'].value_counts())
```

```
U200    446597
U100    443062
O200    138521
Name: ActualElapsedTime, dtype: int64
EarlyU1     545087
LateU1      339982
Intime       79521
LateU10      63482
LateU20        105
EarlyU10         3
Name: DepDelay, dtype: int64
L     382244
M     315326
S     223540
```

```
XL     107070
Name: Distance, dtype: int64
U30     946001
U1       71156
O1       11023
Name: TaxiOut, dtype: int64
```

After that Lable Encoder is used to convert these features before feeding to algorithms

In [77]:

```
le = preprocessing.LabelEncoder()
BayesdataSelected['ActualElapsedTime'] = le.fit_transform(BayesdataSelected['ActualElapsedTime'])
BayesdataSelected['DepDelay'] = le.fit_transform(BayesdataSelected['DepDelay'])
BayesdataSelected['Distance'] = le.fit_transform(BayesdataSelected['Distance'])
BayesdataSelected['TaxiOut'] = le.fit_transform(BayesdataSelected['TaxiOut'])
```

Now this dataset is also split to 80% for training and 20% for testing as previous one

In [78]:

```
XBayes = BayesdataSelected.copy()
yBayes= y.copy()
```

In [79]:

```
XBayesTrainHoldout, XBayesTestHoldout, yBayesTrainHoldout, yBayesTestHoldout =
train_test_split(XBayes, yBayes, test_size = 0.2, random_state = 10)
XBayesTrainHoldout
```

Out[79]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month	DayofMonth	DayOfWeek
60143	1	0	1	2	5	26	1
261230	1	0	2	2	10	12	7
160524	2	0	1	2	6	19	4
894997	2	3	0	2	9	29	1
530734	2	0	0	2	4	29	2
...	...	...	...	...	...	...	...
617841	2	0	0	2	11	17	1
443712	0	3	3	1	7	13	7
881167	0	2	3	2	2	13	3
760957	1	3	1	2	10	9	4
345353	0	3	3	2	6	13	5

822544 rows × 7 columns

Naive Bayes

From the sklearn library it is recommended to use Complement Naive Bayes for imablance dataset; therefore two type of NB will be used in this case to make comparision.

In [80]:

```
from sklearn.naive_bayes import ComplementNB
CNB = ComplementNB()
print(CNB.fit(XBayesTrainHoldout,yBayesTrainHoldout))
y_CNB_pred = CNB.predict(XBayesTestHoldout)
from sklearn.metrics import classification_report
print(classification_report(yBayesTestHoldout, y_CNB_pred))
```

```
ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
```

```
ComplementNB(alpha=1.0, class_prior=None, fit_prior=True, norm=False)
              precision    recall  f1-score   support

           0       0.98      0.66      0.79    178568
           1       0.30      0.93      0.45     27068

    accuracy                           0.70    205636
   macro avg       0.64      0.80      0.62    205636
weighted avg       0.89      0.70      0.75    205636
```

In [ ]:

```

```

In [81]:

```
from sklearn.naive_bayes import GaussianNB
GNB = GaussianNB()
print(GNB.fit(XBayesTrainHoldout, yBayesTrainHoldout))
y_pred = GNB.predict(XBayesTestHoldout)
print(classification_report(yBayesTestHoldout, y_pred))
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
              precision    recall  f1-score   support

           0       0.89      0.98      0.93    178568
           1       0.60      0.23      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.75      0.61      0.64    205636
weighted avg       0.86      0.88      0.85    205636
```

=>As recommendation of sklearn ComplementNB give better reuslt than GaussianNB.

Decision Tree

In [82]:

```
from sklearn.tree import DecisionTreeClassifier
DTC = DecisionTreeClassifier()
print(DTC.fit(XBayesTrainHoldout,yBayesTrainHoldout))
y_DT_pred = DTC.predict(XBayesTestHoldout)
print(classification_report(yBayesTestHoldout, y_DT_pred))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       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')
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.57      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.83    205636
weighted avg       0.94      0.94      0.93    205636
```

=> F1_score for '0' is 0.96, and '1' is 70, it give result better than NB

Random Forest

In [83]:

```
from sklearn.ensemble import RandomForestClassifier
RDFC = RandomForestClassifier()
print(RDFC.fit(XBayesTrainHoldout, yBayesTrainHoldout))
y_RF_pred = RDFC.predict(XBayesTestHoldout)
print(classification_report(yBayesTestHoldout, y_RF_pred))
```

```
C:\Users\Annie Nguyen\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: T
he default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, 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=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.83    205636
weighted avg       0.93      0.94      0.93    205636
```

=> F1_score for '0' is 0.96, and '1' is 71, it give result better than NB and extremely small percentage improve comparing TD

Gradient Boosting

In [84]:

```
from sklearn.ensemble import GradientBoostingClassifier
GBC = GradientBoostingClassifier()
print(GBC.fit(XBayesTrainHoldout, yBayesTrainHoldout))
y_GBC_pred = GBC.predict(XBayesTestHoldout)
print(classification_report(yBayesTestHoldout, y_GBC_pred))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, 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)
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.92      0.57      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.84    205636
weighted avg       0.94      0.94      0.93    205636
```

=>Give Result Nearly the same as RF.

Logistic Regression.

In [85]:

```
from sklearn.linear_model import LogisticRegression
LogReg = LogisticRegression()
print(LogReg.fit(XTrainHoldout, yTrainHoldout))
y_LogReg_pred = LogReg.predict(XTestHoldout)
```

```
print(classification_report(yTestHoldout, y_LogReg_pred))
```

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

```
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)
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.95      0.79      0.87     27068

    accuracy                           0.97    205636
   macro avg       0.96      0.89      0.92    205636
weighted avg       0.97      0.97      0.97    205636
```

=> Logistic Regression use dataset One-Hot endcoding give the best resutl.

Apply PCA, SelectKBest and RFE for feature selections

PCA and Standard Scaler

For PCA it is suitable for using Standard scaler because PCA keep max variance (which compare to mean), and the scaler is to standardize features by removing the mean and scaling to unit variance.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

The formula is $\frac{x - mean}{standarddeviation}$

In [86]:

```
from sklearn.preprocessing import StandardScaler
Standscaler = StandardScaler()
X_StdScal = pd.DataFrame(Standscaler.fit_transform(XBayes))
X_StdScal.columns = XBayes.columns

X_StdScal.describe()
```

Out[86]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month	DayofMonth	DayOfWeek
count	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06
mean	-6.144620e-16	1.036306e-14	1.287841e-15	2.179078e-14	9.171558e-17	-4.029439e-16	2.585892e-18
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.877224e+00	-9.129315e-01	-1.052436e+00	-5.924288e+00	-1.583201e+00	-1.673959e+00	-1.472515e+00
25%	-4.327970e-01	-9.129315e-01	-1.052436e+00	2.812588e-01	-9.948495e-01	-8.787867e-01	-9.691181e-01
50%	-4.327970e-01	-9.129315e-01	-5.384088e-02	2.812588e-01	-1.123220e-01	2.998179e-02	3.767570e-02
75%	1.011630e+00	1.051550e+00	9.447542e-01	2.812588e-01	7.702055e-01	8.251542e-01	1.044469e+00
max	1.011630e+00	2.361204e+00	1.943349e+00	2.812588e-01	1.652733e+00	1.733923e+00	1.547866e+00

Firstly all components are chosen to draw the curve of explained variance to select the best number of components.

In [87]:

```
from sklearn.decomposition import PCA
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=7)
X_PCA_StdScal = pd.DataFrame(pca.fit_transform(X_StdScal))
X_PCA_StdScal
```

Out[87]:

	0	1	2	3	4	5	6
0	0.262805	-0.325011	-1.038835	-0.175899	-0.723494	0.898098	0.357725
1	2.699747	-0.648926	-0.277747	-2.079778	0.888483	0.721784	-0.032250
2	-1.463903	-1.307170	0.517113	1.162518	0.015575	0.001008	0.049203
3	-0.757812	-1.319448	0.516794	1.163845	0.014866	0.010594	-0.656759
4	-0.756817	-1.281040	0.546062	0.419690	0.410955	0.081080	-0.657285
...	...	...	...	...	...	...	...
1028175	-1.464843	-0.382725	-1.191317	-0.448566	-0.185014	0.902320	0.042366
1028176	-1.463389	0.164601	-0.007271	-1.455513	2.106345	-1.126195	0.002027
1028177	-1.460250	-1.510667	1.661948	0.472898	0.200808	-0.197422	0.051703
1028178	0.258912	-1.156923	-1.373316	0.435180	1.526159	0.293577	0.359133
1028179	0.259324	-0.798984	-1.633678	0.213308	0.846560	0.584723	0.357655

1028180 rows × 7 columns

In [88]:

```
pca.explained_variance_ratio_.cumsum()
```

Out[88]:

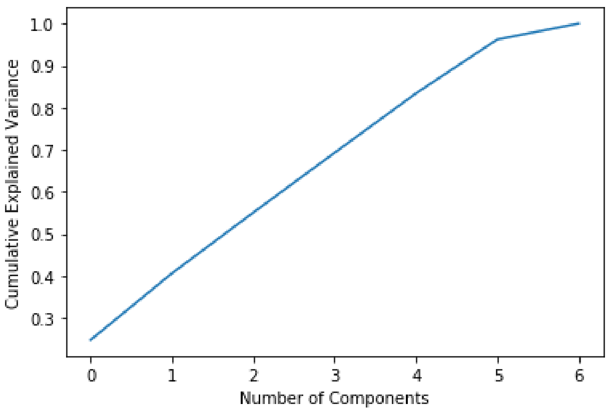
```
array([0.24788878, 0.40636015, 0.55074772, 0.69289267, 0.83407193,
       0.96221507, 1.        ])
```

In [89]:

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```

Out[89]:

```
Text(0, 0.5, 'Cumulative Explained Variance')
```



EVR > 80% is accepted, in this case components = 5 is chosen with EVR over 95%

In [90]:

```
pca = PCA(n_components=5)
X_PCA_StdScal = pd.DataFrame(pca.fit_transform(X_StdScal))
X_PCA_StdScal
```

Out[90]:

	0	1	2	3	4
0	0.262805	-0.325011	-1.038835	-0.175899	-0.723494
1	2.699747	-0.648926	-0.277747	-2.079778	0.888483
2	-1.463903	-1.307170	0.517113	1.162518	0.015575
3	-0.757812	-1.319448	0.516794	1.163845	0.014866
4	-0.756817	-1.281040	0.546062	0.419690	0.410955
...	...	...	...	...	...
1028175	-1.464843	-0.382725	-1.191317	-0.448566	-0.185014
1028176	-1.463389	0.164601	-0.007271	-1.455513	2.106345
1028177	-1.460250	-1.510667	1.661948	0.472898	0.200808
1028178	0.258912	-1.156923	-1.373316	0.435180	1.526159
1028179	0.259324	-0.798984	-1.633678	0.213308	0.846560

1028180 rows × 5 columns

Now the feature reduce from 7 to 5.

Naive Bayes

In [91]:

```
yPCA= y.copy()
```

In [92]:

```
XPCATrainHoldout, XPCATestHoldout, yPCATrainHoldout, yPCATestHoldout = train_test_split(X_PCA_StdSc
al, yPCA, test_size = 0.2, random_state = 10)
XPCATrainHoldout
```

Out[92]:

	0	1	2	3	4
60143	0.263815	-0.767625	0.213715	1.290130	-1.464516
261230	0.968395	-1.081016	-0.721047	-0.662708	1.540173
160524	-0.757761	-0.741575	-0.337223	0.163728	-0.283579
894997	-1.466310	0.146226	0.274189	1.987019	-0.517478
530734	-1.464803	-0.574811	-0.395044	1.261559	-1.471347
...	...	...	...	...	...
617841	-1.462260	-1.491441	1.205275	1.106145	0.051797
443712	2.740135	2.444158	-0.015443	-0.006219	2.053138
881167	2.699430	0.550014	0.115793	-0.602819	-1.234808
760957	0.264278	0.126272	0.543348	-0.327783	0.924699
345353	2.696686	0.642170	-0.354420	-0.587938	0.221353

822544 rows × 5 columns

In [93]:

```
# Naive Bayes
PCA_GNB = GaussianNB()
print(PCA_GNB.fit(XPCATrainHoldout, yPCATrainHoldout))
y_PCA_GNB_pred = PCA_GNB.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_GNB_pred))
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
              precision    recall  f1-score   support

           0       0.89      0.98      0.93    178568
           1       0.59      0.24      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.74      0.61      0.64    205636
weighted avg       0.85      0.88      0.85    205636
```

CNB cannot use in this case because it does not accept negative values, and the result of GNB is the same as without PCA.

Decision Tree

In [94]:

```
PCA_DTC = DecisionTreeClassifier()
print(PCA_DTC.fit(XPCATrainHoldout,yPCATrainHoldout))
y_PCA_DTC_pred = PCA_DTC.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_DTC_pred))
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       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')
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.92      0.57      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.83    205636
weighted avg       0.94      0.94      0.93    205636
```

The result of DT is the same as without PCA.

Random Forest

In [95]:

```
from sklearn.ensemble import RandomForestClassifier
PCA_RFC = RandomForestClassifier()
tick = time.time()
print(PCA_RFC.fit(XPCATrainHoldout, yPCATrainHoldout))
tock = time.time()
y_PCA_RFC_pred = PCA_RFC.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_RFC_pred))
```

```
C:\Users\Annie Nguyen\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: T
he default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, 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=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.84    205636
```

```
weighted avg       0.94      0.94      0.93    205636
```

The result of RF is the same as without PCA.

Gradient Boosting

In [96]:

```
PCA_GBC = GradientBoostingClassifier()
print(PCA_GBC.fit(XPCATrainHoldout, yPCATrainHoldout))
y_PCA_GBC_pred = PCA_GBC.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_GBC_pred))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, 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)
              precision    recall  f1-score   support

           0       0.91      0.99      0.95    178568
           1       0.80      0.38      0.51     27068

    accuracy                           0.91    205636
   macro avg       0.86      0.68      0.73    205636
weighted avg       0.90      0.91      0.89    205636
```

The result of GD is worse when comparing with case without PCA. F1_score of this case fall down by 20% in case predict delay

The number features reduce from 7 to 5 in PCA still keep the good performance as in case without PCA, except Gradient Boosting.

Logistic Regression.

Do the same things as above for datase One-hot

In [97]:

```
Standscaler_OneHot = StandardScaler()
X_StdScal_OneHot = pd.DataFrame(Standscaler_OneHot.fit_transform(X))
X_StdScal_OneHot.columns = X.columns

X_StdScal_OneHot.describe()
```

Out[97]:

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month_2	Month_3	Month_4	Month_5	M
count	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.028180e+06	1.0281
mean	-5.496467e-17	5.857794e-15	-5.587773e-16	4.647327e-16	-1.050044e-14	-1.041208e-14	6.634584e-16	-6.258826e-15	-1.45
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.0000
min	-1.601045e+00	-1.562418e+01	-1.251196e+00	-1.457539e+00	-2.946532e-01	-3.092509e-01	-3.065913e-01	-3.090073e-01	-3.08
25%	-7.173392e-01	-3.991777e-01	-7.149937e-01	-5.717593e-01	-2.946532e-01	-3.092509e-01	-3.065913e-01	-3.090073e-01	-3.08
50%	-2.469795e-01	-3.129984e-01	-2.622400e-01	-2.174473e-01	-2.946532e-01	-3.092509e-01	-3.065913e-01	-3.090073e-01	-3.08

	ActualElapsedTime	DepDelay	Distance	TaxiOut	Month_2	Month_3	Month_4	Month_5	M
75%	4.229267e-01	-5.[illegible]e-02	4.000233e-01	2.254427e-01	-2.9[illegible]e-01	-3.0[illegible]e-01	-3.0[illegible]e-01	-3.0[illegible]e-01	-3.0
max	7.563842e+00	3.338310e+01	7.516246e+00	2.414150e+01	3.393821e+00	3.233620e+00	3.261671e+00	3.236169e+00	3.2364

8 rows × 51 columns

In [98]:

```
pca_OneHot = PCA(n_components=51)
X_PCA_StdScal_OneHot = pd.DataFrame(pca_OneHot.fit_transform(X_StdScal_OneHot))
X_PCA_StdScal_OneHot
```

Out[98]:

	0	1	2	3	4	5	6	7	8	9	...	41	42
0	-1.046143	0.594496	1.582928	1.474979	0.313033	1.212834	-0.170205	-0.836590	0.693512	-1.763375	...	-0.114071	-1.865614
1	2.840984	-1.139434	-0.299239	0.789522	2.091451	-0.619642	-0.281183	0.161582	1.535104	1.388382	...	-0.402989	-0.309955
2	-0.297061	0.777460	0.194078	-1.124849	-0.068101	-0.600664	-0.939009	-1.359380	-1.062532	0.471484	...	0.414171	-0.047828
3	-0.693192	0.778627	0.208923	-1.103874	-0.037480	-0.589548	-0.851282	-1.335017	-1.055756	0.473483	...	0.441995	-0.001967
4	-0.735217	0.148296	-1.136495	-1.123450	-0.856519	1.133199	-0.635185	-1.101236	-1.709268	0.807898	...	0.356285	-1.017120
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1028175	-0.013478	0.924913	-1.612385	1.361140	-1.659897	-0.841433	-0.607373	1.374421	-0.419450	0.001615	...	0.051024	0.477756
1028176	0.017057	-1.357675	-1.476581	0.751257	1.669481	-0.429373	-1.392597	-0.376295	0.614516	-0.976130	...	-0.046942	1.232446
1028177	-0.078494	-0.886018	0.115955	-0.333310	-0.090322	-0.646966	-1.175583	0.993920	1.137883	-1.170445	...	0.409185	-0.252614
1028178	-0.942978	-1.358037	-1.430112	0.879735	1.695901	-0.462969	-1.354207	-0.222327	0.536786	-0.818289	...	-0.055728	0.902714
1028179	-0.948948	-1.814801	0.408080	0.887713	1.247815	-1.024392	-0.661036	-0.743317	-0.660346	-0.448297	...	0.267899	-0.146404

1028180 rows × 51 columns

In [99]:

```
pca_OneHot.explained_variance_ratio_.cumsum()
```

Out[99]:

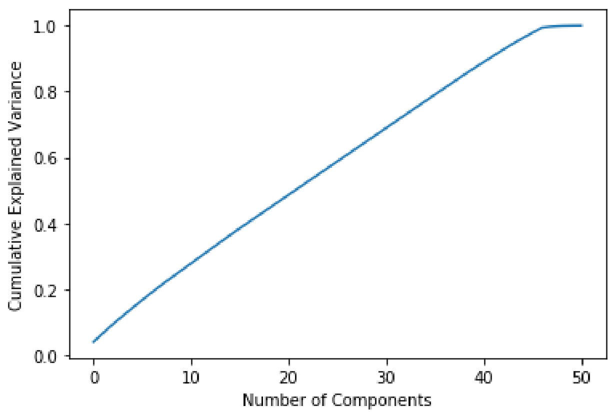
```
array([0.03992873, 0.06748428, 0.0943752 , 0.11897936, 0.14346629,
       0.16750533, 0.19093712, 0.21354361, 0.2352884 , 0.25688825,
       0.278427  , 0.29993313, 0.32138215, 0.34278198, 0.36406462,
       0.38489973, 0.4053013 , 0.42559246, 0.44587924, 0.4661657 ,
       0.48644859, 0.50672756, 0.52700512, 0.54727933, 0.56755207,
       0.58782448, 0.60809442, 0.6283631 , 0.64863064, 0.66889488,
       0.68915813, 0.70941711, 0.72967316, 0.74992799, 0.77017291,
       0.79035927, 0.81053524, 0.83067155, 0.85068793, 0.8699947 ,
       0.88904581, 0.90784522, 0.92654978, 0.94474153, 0.96179117,
       0.97816835, 0.99388768, 0.99713731, 0.99897381, 0.99963018,
       1.        ])
```

In [100]:

```
plt.plot(np.cumsum(pca_OneHot.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
```

Out[100]:

```
Text(0, 0.5, 'Cumulative Explained Variance')
```



EVR > 80% is accepted, in this case components = 45 is chosen with EVR over 95%

In [101]:

```
pca_OneHot = PCA(n_components=45)
X_PCA_StdScal_OneHot = pd.DataFrame(pca_OneHot.fit_transform(X_StdScal_OneHot))
X_PCA_StdScal_OneHot
```

Out[101]:

	0	1	2	3	4	5	6	7	8	9	...	35	36
0	-1.046143	0.594496	1.582928	1.474979	0.313033	1.212834	-0.170205	-0.836590	0.693512	-1.763375	...	-1.770704	-1.079344
1	2.840984	-1.139434	-0.299239	0.789522	2.091451	-0.619642	-0.281183	0.161582	1.535104	1.388382	...	-0.478136	-0.172651
2	-0.297061	0.777460	0.194078	-1.124849	-0.068101	-0.600664	-0.939009	-1.359380	-1.062532	0.471484	...	-1.468858	-0.735647
3	-0.693192	0.778627	0.208923	-1.103874	-0.037480	-0.589548	-0.851282	-1.335017	-1.055756	0.473483	...	-1.472245	-0.736144
4	-0.735217	0.148296	-1.136495	-1.123450	-0.856519	1.133199	-0.635185	-1.101236	-1.709268	0.807898	...	0.456949	0.008694
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1028175	-0.013478	0.924913	-1.612385	1.361140	-1.659897	-0.841433	-0.607373	1.374421	-0.419450	0.001615	...	0.560165	0.782107
1028176	0.017057	-1.357675	-1.476581	0.751257	1.669481	-0.429373	-1.392597	-0.376295	0.614516	-0.976130	...	-0.238971	-0.486737
1028177	-0.078494	-0.886018	0.115955	-0.333310	-0.090322	-0.646966	-1.175583	0.993920	1.137883	-1.170445	...	0.019637	-0.072314
1028178	-0.942978	-1.358037	-1.430112	0.879735	1.695901	-0.462969	-1.354207	-0.222327	0.536786	-0.818289	...	0.107946	0.137577
1028179	-0.948948	-1.814801	0.408080	0.887713	1.247815	-1.024392	-0.661036	-0.743317	-0.660346	-0.448297	...	1.249620	-0.339551

1028180 rows × 45 columns

Number of Features reduce from 51 to 45.

In [102]:

```
yPCA_OneHot= y.copy()
```

In [103]:

```
XPCATrainHoldout_OneHot, XPCATestHoldout_OneHot, yPCATrainHoldout_OneHot, yPCATestHoldout_OneHot =
train_test_split(X_PCA_StdScal_OneHot, yPCA_OneHot, test_size = 0.2, random_state = 10)
XPCATrainHoldout_OneHot
```

Out[103]:

	0	1	2	3	4	5	6	7	8	9	...	35	36	
60143	-0.740907	0.613245	-0.301772	0.820969	-1.376459	-0.000102	0.192758	0.242166	0.493895	1.802907	...	0.871396	-1.408840	0.
261230	-1.675211	-0.815318	-0.605419	1.166164	0.012698	-0.181360	-0.928262	-1.306709	-0.481775	0.393164	...	0.644062	1.045270	0.
160524	-0.559563	-0.387393	0.685354	-0.047933	-0.846373	-0.699652	0.527894	1.992820	2.013011	1.351582	...	-0.152639	0.254507	0.
894997	-0.235571	1.188443	-0.254977	-0.791269	0.050540	-1.247720	-1.337481	-0.281727	0.173109	-0.409151	...	-1.569469	-0.452817	1.
530734	0.907282	2.153111	-0.895044	-1.928941	0.705798	-1.903851	-0.904231	-0.490187	-1.175684	0.354629	...	-1.605211	-0.395463	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
617841	0.069287	-0.865147	0.158290	-0.393421	-0.045444	-0.623804	-1.223671	0.941275	1.143005	-1.124810	...	0.641264	-0.211041	1.
443712	4.715929	-1.972549	-0.223126	0.419010	1.572069	-0.813897	0.271586	-1.388134	-1.081793	-0.464045	...	0.941912	1.940286	2.
881167	2.651015	-1.134368	-1.005485	0.047435	0.488413	1.326845	-0.084422	1.923445	-2.432362	0.051991	...	0.855945	1.141019	2.
760957	-0.865528	-1.281845	0.414695	-1.045497	-1.603043	0.216201	-0.791148	-0.536245	-0.948666	0.923943	...	-1.032756	0.655699	0.
345353	3.726316	0.148904	0.712066	1.266378	1.327210	0.902349	-0.552466	0.397333	1.451776	1.549656	...	0.398781	1.071192	2.

822544 rows × 45 columns

In [104]:

```
PCA_LgR_OneHot = LogisticRegression()
print(PCA_LgR_OneHot.fit(XPCATrainHoldout_OneHot, yPCATrainHoldout_OneHot))
y_PCA_LgR_OneHot_pred = PCA_LgR_OneHot.predict(XPCATestHoldout_OneHot)
print(classification_report(yPCATestHoldout_OneHot, y_PCA_LgR_OneHot_pred))
```

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

```
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)
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.91      0.81      0.86     27068

    accuracy                           0.96    205636
   macro avg       0.94      0.90      0.92    205636
weighted avg       0.96      0.96      0.96    205636
```

=> Give the same result as in case without PCA with the Number of feature reduce from 51 to 46.

USING gridsearch EVALUATE BEST PARAMETERS FOR MODELS

Naive Bayes

In this case Gaussian NB Tree performance dominant other algorithms (F1_score 0: 0.93, 1:0.34).

- 'var_smoothing' :[0.00000000001,0.00000000003,0.0000000001,0.0000000003, 0.000000001,0.000000003,0.00000001]

In [112]:

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {'var_smoothing' :[0.00000000001,0.00000000003,0.0000000001,0.0000000003,
                                0.000000001,0.000000003,0.00000001]}
GNB_grid = GridSearchCV(estimator=GaussianNB(),
                        param_grid = param_grid,
                        scoring="f1",
                        cv=3,
                        n_jobs = -1)
tick = time.time()
GNB_grid.fit(XPCATrainHoldout, yPCATrainHoldout)
tock = time.time()
GNB_grid_best = GNB_grid.best_estimator_ #best estimator
```

In [113]:

```
print("Best Model Parameter for GNB: ",GNB_grid.best_params_)
```

```
Best Model Parameter for GNB:  {'var_smoothing': 1e-11}
```

In [ ]:

```
tock - tick
```

In [115]:

```
PCA_GNB_best = GNB_grid_best
tick = time.time()
print(PCA_GNB_best.fit(XPCATrainHoldout, yPCATrainHoldout))
tock = time.time()
y_PCA_GBN_best_pred = PCA_GNB_best.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_GBN_best_pred))
```

```
GaussianNB(priors=None, var_smoothing=1e-11)
              precision    recall  f1-score   support

           0       0.89      0.98      0.93    178568
           1       0.59      0.24      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.74      0.61      0.64    205636
weighted avg       0.85      0.88      0.85    205636
```

Although learning rate is now 10^-11 but GNB still give the same result as in PCA

Decision Tree

In this case Decision Tree performance dominant other algorithms (F1_score 0: 0.96, 1:0.70).

- 'criterion' :['gini', 'entropy']
- 'max_features': ['auto', 'log2'],
- 'max_depth' : [4,5],

In [116]:

```
param_grid = {'criterion' :['gini', 'entropy'],
              'max_features': ['auto', 'log2'],
              'max_depth' : [4,5]}
DT_grid = GridSearchCV(estimator=DecisionTreeClassifier(),
                        param_grid = param_grid,
                        scoring="f1",
                        cv=3,
                        n_jobs = -1)
tick = time.time()
DT_grid.fit(XPCATrainHoldout, yPCATrainHoldout)
tock = time.time()
DT_grid_best = DT_grid.best_estimator_ #best estimator
```

In [117]:

```
print("Best Model Parameter for DT: ",DT_grid.best_params_)
```

```
Best Model Parameter for DT:  {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2'}
```

In [ ]:

```
tock - tick
```

In [119]:

```
PCA_DTC_best = DT_grid_best
tick = time.time()
print(PCA_DTC_best.fit(XPCATrainHoldout, yPCATrainHoldout))
tock = time.time()
y_PCA_DTC_best_pred = PCA_DTC_best.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_DTC_best_pred))
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features='log2', 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')
              precision    recall  f1-score   support

           0       0.90      0.97      0.93    178568
           1       0.61      0.29      0.40     27068

    accuracy                           0.88    205636
   macro avg       0.76      0.63      0.67    205636
weighted avg       0.86      0.88      0.86    205636
```

=>Several changes of hyperparmeter, and it surprisingly gives the result worse than in PCA case

Random Forest

In this case Random Forest performance dominant other algorithms (F1_score 0: 0.96, 1:0.71). n_estimators is consider as importan hyperparameter of this algorithm so

- 'n_estimators' : [10,20,30]
- 'criterion' :['gini', 'entropy']
- 'max_features': ['auto', 'log2'],
- 'max_depth' : [4,5],

In [165]:

```
param_grid = {'n_estimators': [10,20,30],
              'criterion' :['gini', 'entropy'],
              'max_features': ['auto', 'log2'],
              'max_depth' : [4,5]}
RDF_grid = GridSearchCV(estimator=RandomForestClassifier(),
                          param_grid = param_grid,
                          scoring="f1",
                          cv=3,
                          n_jobs = -1)
tick = time.time()
RDF_grid.fit(XPCATrainHoldout, yPCATrainHoldout)
tock = time.time()
RDF_grid_best = RDF_grid.best_estimator_ #best estimator
```

In [166]:

```
print("Best Model Parameter for RDF: ",RDF_grid.best_params_)
```

```
Best Model Parameter for RDF:  {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto',
'n_estimators': 30}
```

In [167]:

```
tock - tick
```

Out[167]:

```
497.7357304096222
```

In [168]:

```
PCA_RFC_best = RDF_grid_best
tick = time.time()
print(PCA_RFC_best.fit(XPCATrainHoldout, yPCATrainHoldout))
tock = time.time()
y_PCA_RFC_best_pred = PCA_RFC_best.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_RFC_pred))
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=5, 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=30,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.84    205636
weighted avg       0.94      0.94      0.93    205636
```

=>Although RF is tuning with several change of hyperparmeter it still give the result similar as in PCA case

Gradient Boosting

In this case GBC performance (F1_score 0: 0.95, 1:0.51).

- 'learning_rate ':[0.01,0.03,0.1],
- 'n_estimators ':[10,20,30],
- 'max_depth':[4,5],
- 'max_features': ['auto', 'log2']

In [149]:

```
param_grid = {'n_estimators': [10,20,30],
              'learning_rate': [0.01,0.03,0.1],
              'max_features': ['auto', 'log2'],
              'max_depth' : [4,5]}
GBC_grid = GridSearchCV(estimator=GradientBoostingClassifier(),
                          param_grid = param_grid,
                          scoring="f1",
                          cv=3,
                          n_jobs = -1)
tick = time.time()
GBC_grid.fit(XPCATrainHoldout, yPCATrainHoldout)
tock = time.time()
GBC_grid_best = GBC_grid.best_estimator_ #best estimator
```

In [150]:

```
print("Best Model Parameter for GBC: ",GBC_grid.best_params_)
```

```
Best Model Parameter for GBC:  {'learning_rate': 0.1, 'max_depth': 5, 'max_features': 'auto',
'n_estimators': 30}
```

In [151]:

```
tock - tick
```

Out[151]:

```
1520.39133477211
```

In [152]:

```
PCA_GBC_best = GBC_grid_best
tick = time.time()
print(PCA_GBC_best.fit(XPCATrainHoldout, yPCATrainHoldout))
tock = time.time()
y_PCA_GBC_best_pred = PCA_GBC_best.predict(XPCATestHoldout)
print(classification_report(yPCATestHoldout, y_PCA_GBC_best_pred))
```

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           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=30,
                           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)
              precision    recall  f1-score   support

           0       0.92      0.99      0.95    178568
           1       0.86      0.41      0.55     27068

    accuracy                           0.91    205636
   macro avg       0.89      0.70      0.75    205636
weighted avg       0.91      0.91      0.90    205636
```

=> A little bit better than PCA, but it still worse than case without PCA.

Logistic Regression

In this case Logistic Regression performance dominant other algorithms (F1_score 0: 0.96, 1:0.71). n_estimators is consider as importan hyperparameter of this algorithm so

- 'C' : [0.7,0.8,0.9],
- 'solver': ['lbfgs', 'liblinear']

In [153]:

```
param_grid = {'C' : [0.7,0.8,0.9],
              'solver': ['lbfgs', 'liblinear']}
LgR_grid = GridSearchCV(estimator=LogisticRegression(),
                        param_grid = param_grid,
                        scoring="f1",
                        cv=3,
                        n_jobs = -1)
tick = time.time()
LgR_grid.fit(XPCATrainHoldout_OneHot, yPCATrainHoldout_OneHot)
tock = time.time()
LgR_grid_best = LgR_grid.best_estimator_ #best estimator
```

In [154]:

```
print("Best Model Parameter for LgR: ",LgR_grid.best_params_)
```

```
print('Best Model Parameter for LgR: ',LgR_grid.best_params_)
```

```
Best Model Parameter for LgR:  {'C': 0.8, 'solver': 'liblinear'}
```

In [155]:

```
tock - tick
```

Out[155]:

```
111.35945916175842
```

In [156]:

```
PCA_LgR_best = LgR_grid_best
tick = time.time()
print(PCA_LgR_best.fit(XPCATrainHoldout_OneHot, yPCATrainHoldout_OneHot))
tock = time.time()
y_PCA_LgR_best_pred = PCA_LgR_best.predict(XPCATestHoldout_OneHot)
print(classification_report(yPCATestHoldout_OneHot, y_PCA_LgR_best_pred))
```

```
LogisticRegression(C=0.8, 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='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.91      0.81      0.86     27068

    accuracy                           0.96    205636
   macro avg       0.94      0.90      0.92    205636
weighted avg       0.96      0.96      0.96    205636
```

=>Although LgR is tuning with several change of hyperparmeter it still give the result similar as in PCA case

CONCLUSIONS

Without PCA.

In [157]:

```
print('--------------------------CNB--------------------------')
print(classification_report(yBayesTestHoldout, y_CNB_pred))
print('--------------------------GNB--------------------------')
print(classification_report(yBayesTestHoldout, y_pred))
print('--------------------------DT--------------------------')
print(classification_report(yBayesTestHoldout, y_DT_pred))
print('--------------------------RF--------------------------')
print(classification_report(yBayesTestHoldout, y_RF_pred))
print('--------------------------GB--------------------------')
print(classification_report(yBayesTestHoldout, y_GBC_pred))
print('--------------------------LoG--------------------------')
print(classification_report(yTestHoldout, y_LogReg_pred))
```

```
--------------------------CNB--------------------------
              precision    recall  f1-score   support

           0       0.98      0.66      0.79    178568
           1       0.30      0.93      0.45     27068

    accuracy                           0.70    205636
   macro avg       0.64      0.80      0.62    205636
weighted avg       0.89      0.70      0.75    205636

--------------------------GNB--------------------------
              precision    recall  f1-score   support
```

```
           0       0.89      0.98      0.93    178568
           1       0.60      0.23      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.75      0.61      0.64    205636
weighted avg       0.86      0.88      0.85    205636

--------------------------DT-------------------------
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.57      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.83    205636
weighted avg       0.94      0.94      0.93    205636

--------------------------RF-------------------------
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.83    205636
weighted avg       0.93      0.94      0.93    205636

--------------------------GB-------------------------
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.92      0.57      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.84    205636
weighted avg       0.94      0.94      0.93    205636

--------------------------LoG-------------------------
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.95      0.79      0.87     27068

    accuracy                           0.97    205636
   macro avg       0.96      0.89      0.92    205636
weighted avg       0.97      0.97      0.97    205636
```

=>As recommendation of sklearn ComplementNB give better reuslt than GaussianNB.

=>DT give result better than NB

=>Give Result Nearly the same as RF.

=> Logistic Regression use dataset One-Hot endcoding give the best resutl.

With PCA.

In [158]:

```
print('--------------------------GNB-------------------------')
print(classification_report(yPCATestHoldout, y_PCA_GNB_pred))
print('--------------------------DT-------------------------')
print(classification_report(yPCATestHoldout, y_PCA_DTC_pred))
print('--------------------------RF-------------------------')
print(classification_report(yPCATestHoldout, y_PCA_RFC_pred))
print('--------------------------GBC-------------------------')
print(classification_report(yPCATestHoldout, y_PCA_GBC_pred))
print('--------------------------LgR-------------------------')
print(classification_report(yPCATestHoldout_OneHot, y_PCA_LgR_OneHot_pred))
```

```
--------------------------GNB--------------------------
              precision    recall  f1-score   support

           0       0.89      0.98      0.93    178568
           1       0.59      0.24      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.74      0.61      0.64    205636
weighted avg       0.85      0.88      0.85    205636

--------------------------DT--------------------------
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.92      0.57      0.70     27068

    accuracy                           0.94    205636
   macro avg       0.93      0.78      0.83    205636
weighted avg       0.94      0.94      0.93    205636

--------------------------RF--------------------------
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.84    205636
weighted avg       0.94      0.94      0.93    205636

--------------------------GBC--------------------------
              precision    recall  f1-score   support

           0       0.91      0.99      0.95    178568
           1       0.80      0.38      0.51     27068

    accuracy                           0.91    205636
   macro avg       0.86      0.68      0.73    205636
weighted avg       0.90      0.91      0.89    205636

--------------------------LgR--------------------------
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.91      0.81      0.86     27068

    accuracy                           0.96    205636
   macro avg       0.94      0.90      0.92    205636
weighted avg       0.96      0.96      0.96    205636
```

=>CNB cannot use in this case because it does not accept negative values, and the result of GNB is the same as without PCA.

=>The result of DT is the same as without PCA.

=>The result of RF is the same as without PCA.

=>The result of GD is worse when comparing with case without PCA. F1_score of this case fall down by 20% in case predict delay

=>The number features reduce from 7 to 5 in PCA still keep the good performance as in case without PCA, except Gradient Boosting.

=> Give the same result as in case without PCA with the Number of feature reduce from 51 to 46.

Best Model with Best Hyperparameter.

In [159]:

```
print("Best Model Parameter for GNB: ",GNB_grid.best_params_)
print(classification_report(yPCATestHoldout, y_PCA_GBN_best_pred))
```

```
Best Model Parameter for GNB:  {'var_smoothing': 1e-11}
              precision    recall  f1-score   support

           0       0.89      0.98      0.93    178568
           1       0.59      0.24      0.34     27068

    accuracy                           0.88    205636
   macro avg       0.74      0.61      0.64    205636
weighted avg       0.85      0.88      0.85    205636
```

In [160]:

```
print("Best Model Parameter for DT: ",DT_grid.best_params_)
print(classification_report(yPCATestHoldout, y_PCA_DTC_best_pred))
```

```
Best Model Parameter for DT:  {'criterion': 'entropy', 'max_depth': 4, 'max_features': 'log2'}
              precision    recall  f1-score   support

           0       0.90      0.97      0.93    178568
           1       0.61      0.29      0.40     27068

    accuracy                           0.88    205636
   macro avg       0.76      0.63      0.67    205636
weighted avg       0.86      0.88      0.86    205636
```

In [169]:

```
print("Best Model Parameter for RDF: ",RDF_grid.best_params_)
print(classification_report(yPCATestHoldout, y_PCA_RFC_pred))
```

```
Best Model Parameter for RDF:  {'criterion': 'gini', 'max_depth': 5, 'max_features': 'auto',
'n_estimators': 30}
              precision    recall  f1-score   support

           0       0.94      0.99      0.96    178568
           1       0.91      0.58      0.71     27068

    accuracy                           0.94    205636
   macro avg       0.92      0.78      0.84    205636
weighted avg       0.94      0.94      0.93    205636
```

In [162]:

```
print("Best Model Parameter for GBC: ",GBC_grid.best_params_)
print(classification_report(yPCATestHoldout, y_PCA_GBC_best_pred))
```

```
Best Model Parameter for GBC:  {'learning_rate': 0.1, 'max_depth': 5, 'max_features': 'auto',
'n_estimators': 30}
              precision    recall  f1-score   support

           0       0.92      0.99      0.95    178568
           1       0.86      0.41      0.55     27068

    accuracy                           0.91    205636
   macro avg       0.89      0.70      0.75    205636
weighted avg       0.91      0.91      0.90    205636
```

In [163]:

```
print("Best Model Parameter for LgR: ",LgR_grid.best_params_)
print(classification_report(yPCATestHoldout_OneHot, y_PCA_LgR_best_pred))
```

```
Best Model Parameter for LgR:  {'C': 0.8, 'solver': 'liblinear'}
              precision    recall  f1-score   support

           0       0.97      0.99      0.98    178568
           1       0.91      0.81      0.86     27068

    accuracy                           0.96    205636
   macro avg       0.94      0.90      0.92    205636
weighted avg       0.96      0.96      0.96    205636
```

=>Although learning rate is now 10^-11 but GNB still give the same result as in PCA

=>Several changes of hyperparmeter of DT , and it surprisingly gives the result worse than in PCA case

=>Although RF is tuning with several changes of hyperparmeter it still give the result similar as in PCA case

=> A little bit better than PCA, but it still worse than case without PCA.

=>Although LgR is tuning with several change of hyperparmeter it still give the result similar as in PCA case

Overal, The performance of these algorithm fowlling f1_score in this dataset as:

- Using Lable Encoding for Categorical (It is reccommended uses this converison for theses algorithm)
  - NB<DT<GB=RF (without PCA)
  - NB<GB<DT<RF (with PCA)
  - NB<DT<GB<RF (using gridsearch)
- Using One-Hot Encoding only for logistic regression

=>In PCA still keep the good performance as in case without PCA, except Gradient Boosting because the number of components opted for the explained variance is over 95%

=> RG gives better performance than DT because it combine independent DT to gether and these DT complement error for each others.

NOTE: If they are comparing f1_score with best condition for them Logistic Regresion in this case is the best model with F1_Score for Non-Delay is **98%** and Delay is **86%**.

REFERENCES

https://www.kaggle.com/cdabakoglu/heart-disease-classifications-machine-learning https://medium.com/@banikanusheela/flight-departure-delay-prediction-417240a72ea4

Target = Cate with Num and Cate in Classification: https://www.kaggle.com/mnassrib/titanic-logistic-regression-with-python#3.-Exploratory-Data-Analysis

Model: https://medium.com/@banikanusheela/flight-departure-delay-prediction-417240a72ea4

BoxPlot: https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

Countplot: https://www.kaggle.com/martin1234567890/logistic-regression

Sampling: https://www.kaggle.com/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets https://www.kaggle.com/joparga3/in-depth-skewed-data-classif-93-recall-acc-now

Naive Bayes: https://www.kaggle.com/jiazhuang/demonstrate-naive-bayes https://www.kaggle.com/dimitreoliveira/naive-bayes-probabilistic-ml-titanic-survival https://towardsdatascience.com/naive-bayes-explained-9d2b96f4a9c0

DecisionTree: Code implement https://www.kaggle.com/natevegh/decision-tree-from-scratch-theory-code-explained https://www.kaggle.com/vitorgamalemos/a-simple-decision-tree-tutorial

Random Forest: https://www.kaggle.com/zlatankr/titanic-random-forest-82-78 https://www.kaggle.com/aikinogard/random-forest-starter-with-numerical-features Theory https://towardsdatascience.com/understanding-random-forest-58381e0602d2 Tuning :https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Cycle: https://www.kaggle.com/tlapusan/titanic-decision-tree-implemented-as-lean-startup

Full: https://www.kaggle.com/praanj/titanic-decision-tree-complete-evaluation https://www.kaggle.com/hadend/tuning-random-forest-parameters https://www.kaggle.com/tavoosi/predicting-box-office-revenue-with-random-forest

In [ ]:

```

```