ADS505_Group_Project

October 17, 2021

1 Final Team Project

- Hanmaro Song (hanmarosong@sandiego.edu)
- Eva Chow (echow@sandiego.edu)
- Jose Luis Estrada (joseestrada@sandiego.edu)

[64]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.linear_model import LogisticRegressionCV, SGDClassifier
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, BaggingClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.model_selection import RepeatedStratifiedKFold, cross_validate,␣
 ↪train_test_split, GridSearchCV

from imblearn.combine import SMOTETomek
from imblearn.under_sampling import TomekLinks
from imblearn.pipeline import Pipeline

from sklearn.metrics import plot_confusion_matrix

import time
import warnings

warnings.filterwarnings('ignore')

random_state = 123
```

1.0.1 Exploratory Data Analysis

Features	Description
year	Year
month	Month
day	Day
dep_time	Departure time, in Eastern time zone
dep_delay	Departure delay, in minutes
arr_time	Arrival time, in the local time zone
arr_delay	Arrival delay, in minutes
carrier	Carrier, abbreviated
tailnum	Tail number of the airplane
flight	Flight number
origin	Flight origin, airport code
dest	Flight destination, airport code
air_time	Time in the air, in minutes
distance	Distance between the departure and arrival airports, in miles
hour	Scheduled departure hour
minute	Scheduled departure minute

1.0.2 Business Objective

Three major airports of New York have been experiencing flight delays, which impacts profitability negatively. As a result, business partners search for alternatives to mitigate the financial impact and create a strategy to increase profitability. Using models, it's possible to predict if a plane will be delayed longer than 15 minutes or not. Utilizing such, businesses can set up their own shops of any kind to attract those whose planes got delayed and have nothing to do for a while.

1.0.3 Predictors and Target Variable

Suggested Variables to Drop: - 'year': all data is from 2013, making this irrelevant (unless we combine with month + day to create datetime) - 'arr_time': might not be useful for our objective? not sure - 'arr_delay': also not sure. could be useful for potential marketing opportunities of services for delayed flyers, but I suspect most flyers want to be out of the airport at this point and there is little to gain from marketing for services upon arrival - 'tailnum': unique plane identifier; this shouldn't have any impact on delays - 'flight': unique flight identifier; this shoudln't have any impact on delays - 'dest': a lot of unique values with uneven representation - 'hour': displays increase in delay time as the hours pass, but pattern is similar to dep_time and shows high multicollinearity. will only keep dep_time

Suggested Predictor Variables: - 'carrier': noted variation in departure delay across carriers - 'month': noted variation in departure delay according to month (possibly linked to holidays and may need to convert to categorical) - 'day': certain days are noted to show significant increase in departure delay time (as it relates to month, might need to combine with month into single variable for all encompassing datetime variable) - 'dep_time': scatterplot shows increase in delay time as the day progresses - 'origin': noted extended departure delay depending on which New York airport a flyer is traveling from - 'hour': displays increase in delay time as the hours pass, but pattern is similar to dep_time

Suggested Target Variable: - 'dep_delay': if we want to focus our business brief on suggesting marketing strategies as they relate to delays, we may need to set an arbitrary departure delay time at which we distinguish between no/short delay and long delays - i.e. 60 minute delay is the cutoff at which some flyers may consider looking around the airport for souvenirs/food/drink and 4 hours is the cutoff at which some flyers may consider looking for restaurants independent of the airport to dine at

The suggestions listed above do not necessarily mean we will follow. It's a preliminary listing of what we thought we might do later and could be different at the end of the notebook.

---

[2]:
```
data = pd.read_csv('nyc-flights.csv')

data.shape
```

[2]:
```
(32735, 16)
```

[3]:
```
data['dest'].unique().shape
```

[3]:
```
(102,)
```

[4]:
```
data.head()
```

[4]:
```
   year  month  day  dep_time  dep_delay  arr_time  arr_delay carrier tailnum  \
0  2013      6   30       940         15      1216         -4      VX  N626VA
1  2013      5    7      1657         -3      2104         10      DL  N3760C
2  2013     12    8       859         -1      1238         11      DL  N712TW
3  2013      5   14      1841         -4      2122        -34      DL  N914DL
4  2013      7   21      1102         -3      1230         -8      9E  N823AY

   flight origin dest  air_time  distance  hour  minute
0     407    JFK  LAX       313      2475     9      40
1     329    JFK  SJU       216      1598    16      57
2     422    JFK  LAX       376      2475     8      59
3    2391    JFK  TPA       135      1005    18      41
4    3652    LGA  ORF        50       296    11       2
```

---

dep_time & hour & minute

How different are dep_time from hour + minute

[5]:
```
data[['dep_time', 'hour', 'minute']].head(10)
```

[5]:
```
   dep_time  hour  minute
0       940     9      40
1      1657    16      57
2       859     8      59
3      1841    18      41
```

```
4       1102    11        2
5       1817    18       17
6       1259    12       59
7       1920    19       20
8        725     7       25
9       1323    13       23
```

[6]:

```python
data[['dep_time', 'hour', 'minute']].tail(10)
```

[6]:

```
       dep_time  hour  minute
32725      1437    14      37
32726      1558    15      58
32727      1716    17      16
32728      1923    19      23
32729       706     7       6
32730       752     7      52
32731       812     8      12
32732      1057    10      57
32733       844     8      44
32734      1813    18      13
```

When comparing, dep_time is the string representation of hour + minute (notice that single digit minute is concatenated by 0 in the front : 7hrs 6min -> 706). Because they these three columns are same, dropping hour and minute is fine

[7]:

```python
data.drop(columns=['hour', 'minute'], inplace=True)
```

---

year

[8]:

```python
data['year'].value_counts()
```

[8]:

```
2013    32735
Name: year, dtype: int64
```

Because the data is only from 2013 and no variance, dropping it won't have an impact

[9]:

```python
data.drop(columns='year', inplace=True)
```

---

month & day

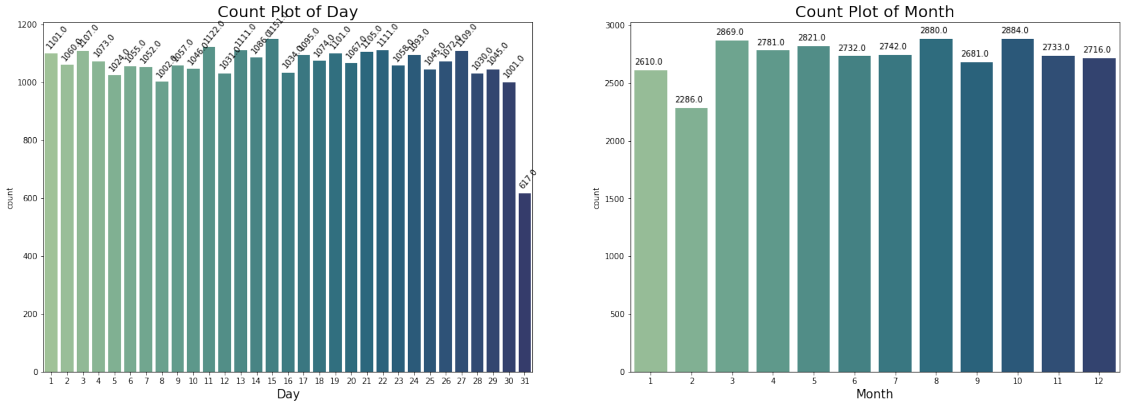
[10]:

```python
month_count = data['month'].value_counts().sort_index()
day_count = data['day'].value_counts().sort_index()
```

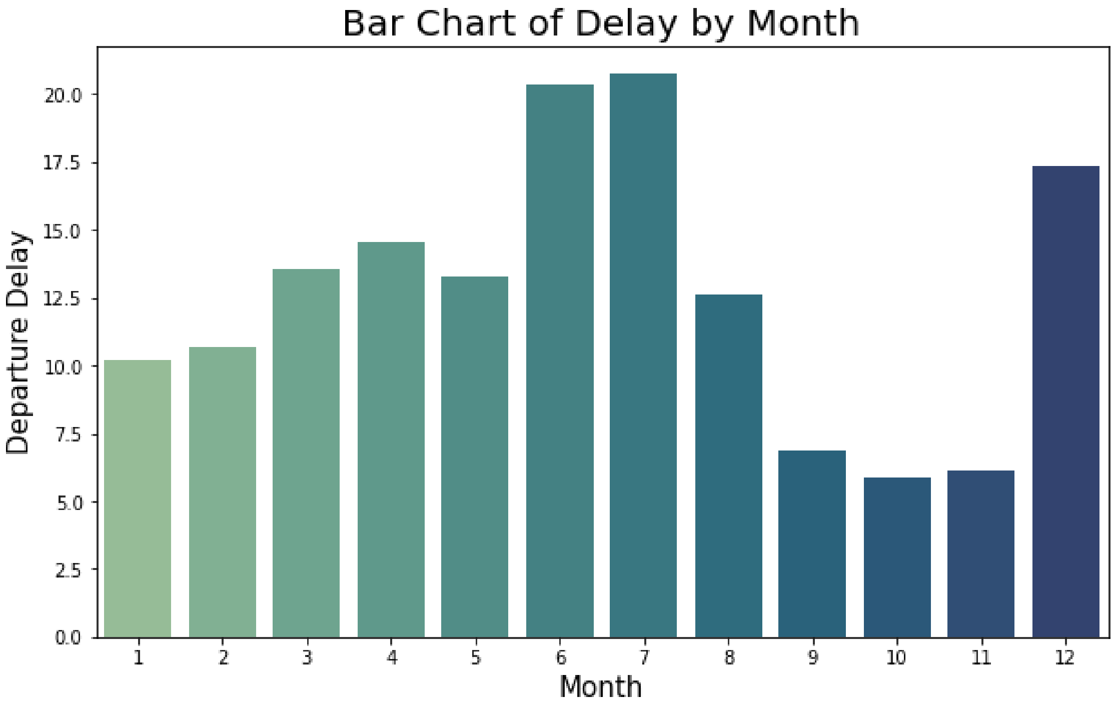
[11]:

```python
fig, ax = plt.subplots(1, 2, figsize=(24, 8))

ax1 = sns.countplot(x="day", data=data, ax=ax[0], palette='crest')
```

```
for p in ax1.patches:
    ax1.annotate('{:.1f}'.format(p.get_height()), (p.get_x(), p.
↪get_height()+15), rotation=50)
ax1.set_xlabel('Day', size=15)
ax1.set_title('Count Plot of Day', size=20)

ax2 = sns.countplot(x="month", data=data, ax=ax[1], palette='crest')

for p in ax2.patches:
    ax2.annotate('{:.1f}'.format(p.get_height()), (p.get_x(), p.
↪get_height()+50))
ax2.set_xlabel('Month', size=15)
ax2.set_title('Count Plot of Month', size=20)

plt.show();
```



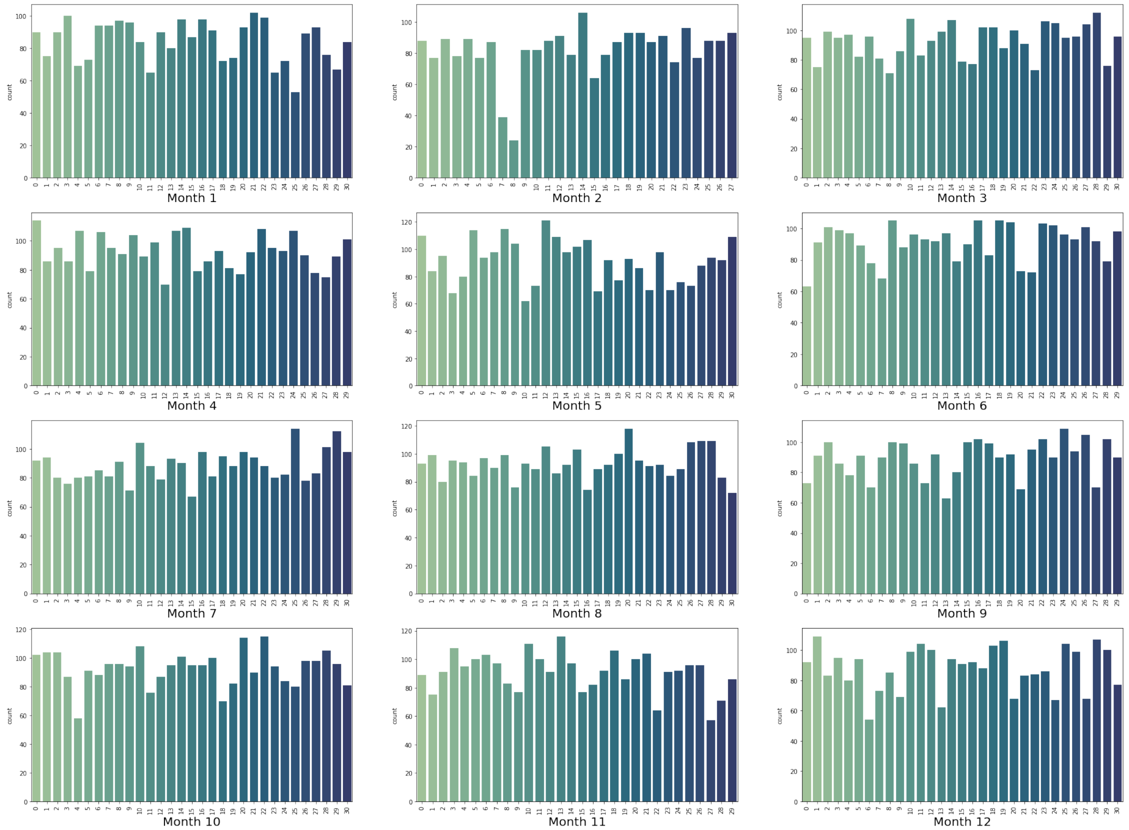
```
[12]: delay_month = data.groupby(['month'])[['dep_delay']].mean().
↪sort_values('dep_delay', ascending=False).reset_index()

fig, ax = plt.subplots(figsize=(10, 6))

ax = sns.barplot(x='month', y='dep_delay', data=delay_month, ax=ax,␣
↪palette='crest')

ax.set_xlabel('Month', size=15)
ax.set_ylabel('Departure Delay', size=15)
ax.set_title('Bar Chart of Delay by Month', size=20)
plt.show();
```



Larger range in departure delay with skew towards longer departure delays in June, July, and December. Steady increase in departure delays leading up until July, with a drop off until December. Are departure delays correlated with holidays/peak travel periods? Holiday seasons, spring break, and summer months have longer departure delays versus the fall months.

Any specific day(s) of month have more data than other days?

[13]:
```
fig, ax = plt.subplots(4, 3, figsize=(8*4, 24))

for i in range(12):

    row, col = i//3, i%3

    plot = sns.countplot(x="day", data=data[data['month']==i+1],␣
 ↪ax=ax[row][col], palette='crest')
    plot.set_xticklabels(plot.get_xticks(), rotation=90)
    plot.set_xlabel('Month {}'.format(i+1), size=20)
```



There seems to be no clear distinct patterns of days and months and they show quite similar distributions.

```
[14]: data[['day', 'month']].corr()
```

```
[14]:            day     month
       day    1.000000  0.010448
       month  0.010448  1.000000
```

And the correlation is also close to 0 and hence, it may be good to keep these two features.

```
[15]: dayspermonth = data.groupby(['day', 'month'])['dep_delay'].mean().unstack()
      dayspermonth.style.highlight_max(color = 'lightgreen', axis = 0)
```

```
[15]: <pandas.io.formats.style.Styler at 0x7fa571bc8610>
```

Longest departure delays around major holidays? - Feb: before Valentine's day - Apr: spring break - May: Memorial Day - Sep: Labor Day - Nov: Thanksgiving - Dec: Peak delay occurs around Dec 5th, but note increase in average flight delay in the week leading up to Christmas

---

origin & destination & tailnum & air time

[16]:

```
data['tailnum'].value_counts()
```

[16]:

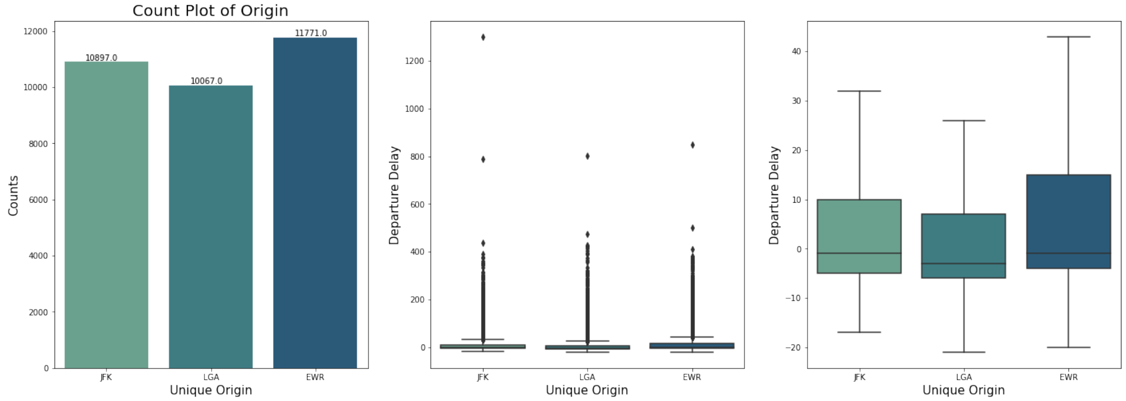
```
N725MQ    59
N713MQ    53
N711MQ    48
N723MQ    47
N722MQ    46
          ..
N359AA     1
N5FBAA     1
N8631E     1
N278AT     1
N924WN     1
Name: tailnum, Length: 3490, dtype: int64
```

[17]:

```
fig, ax = plt.subplots(1, 3, figsize=(24, 8))

ax1 = sns.countplot(x="origin", data=data, ax=ax[0], palette='crest')

for p in ax1.patches:
    ax1.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+.2, p.
 ↪get_height()+60))
ax1.set_xlabel('Unique Origin', size=15)
ax1.set_ylabel('Counts', size=15)
ax1.set_title('Count Plot of Origin', size=20)

ax2 = sns.boxplot(x='origin', y='dep_delay', data=data, ax=ax[1],␣
 ↪palette='crest')
ax2.set_ylabel('Departure Delay', size=15)
ax2.set_xlabel('Unique Origin', size=15)

ax3 = sns.boxplot(x='origin', y='dep_delay', data=data, ax=ax[2],␣
 ↪showfliers=False, palette='crest')
ax3.set_ylabel('Departure Delay', size=15)
ax3.set_xlabel('Unique Origin', size=15)
```

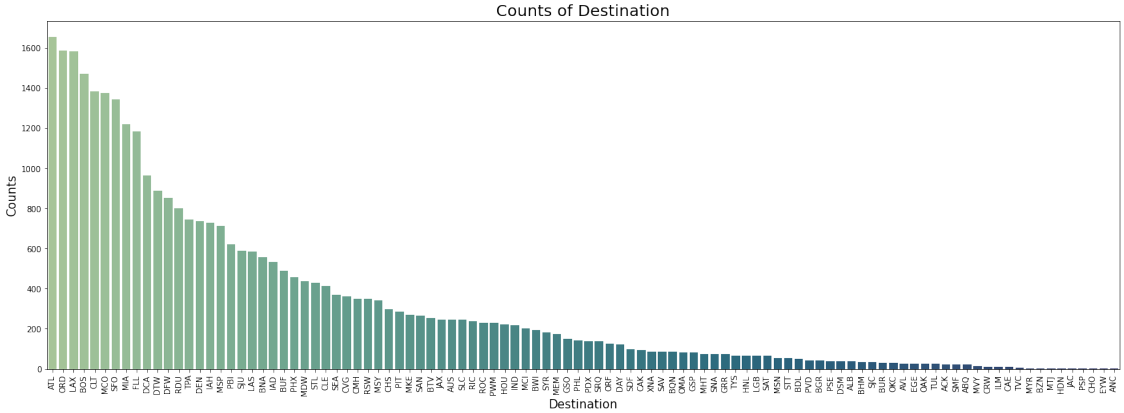
[17]:

```
Text(0.5, 0, 'Unique Origin')
```



We can see there are only 3 origin airports and they are quite evenly distributed and after removing outliers, EWR airport seems to experience longer departure delays, followed by JFK, then LGA.

```
[18]: fig, ax = plt.subplots(figsize=(24, 8))

      ax = sns.barplot(x=data['dest'].value_counts().index, y=data['dest'].
       ↪value_counts(), ax=ax, palette='crest')

      ax.set_xticklabels(data['dest'].value_counts().index, rotation=90)
      ax.set_xlabel('Destination', size=15)
      ax.set_ylabel('Counts', size=15)
      ax.set_title('Counts of Destination', size=20)

      plt.show();
```



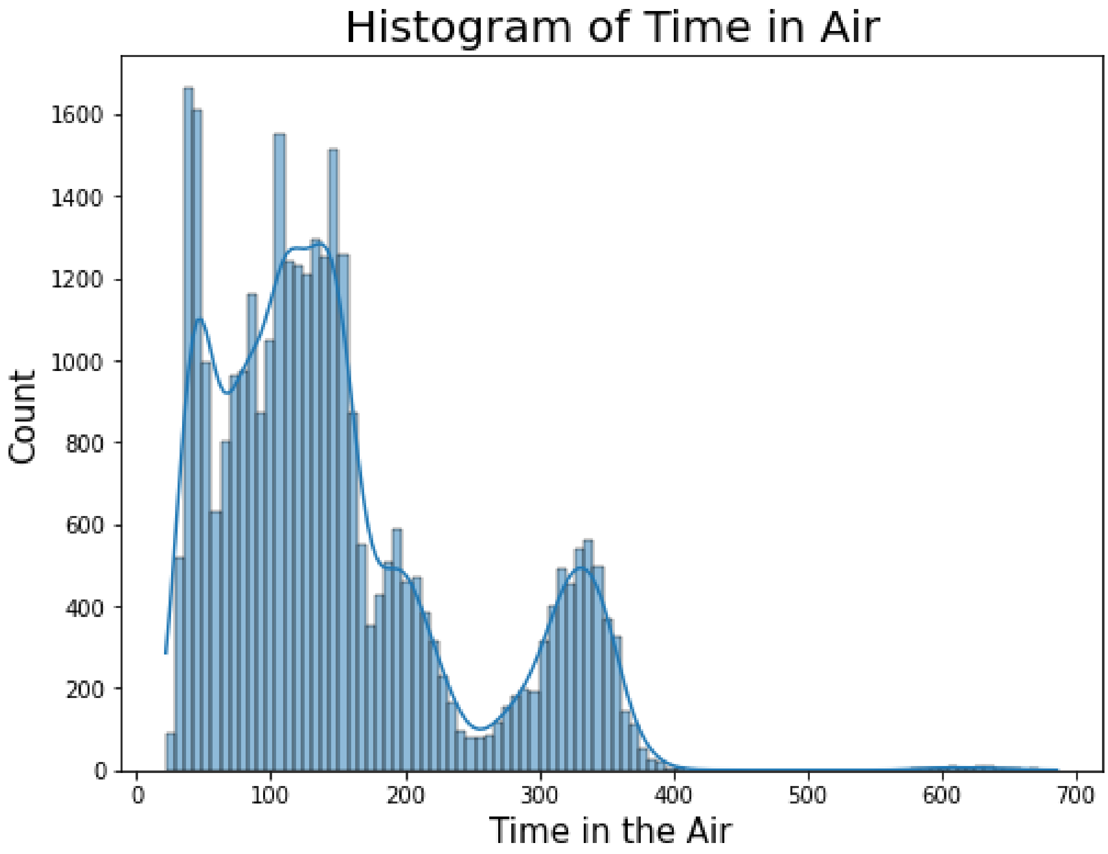
However when we look at the destination counts plot, they are skewed heavily. Also using this feature and One-Hot-Encoding will create a sparse matrix that could lead to unstable performance of models later so we drop this feature.

[19]:

```
data.drop(columns='dest', inplace=True)
```

[20]:

```
fig, ax = plt.subplots(figsize=(8, 6))

sns.histplot(x='air_time', data=data, ax=ax, kde=True, palette='crest')

ax.set_title('Histogram of Time in Air', size=20)
ax.set_xlabel('Time in the Air', size=15)
ax.set_ylabel('Count', size=15)

plt.show();
```



If departure is delayed, this will lead to a delay in arrival in most cases. Examination below

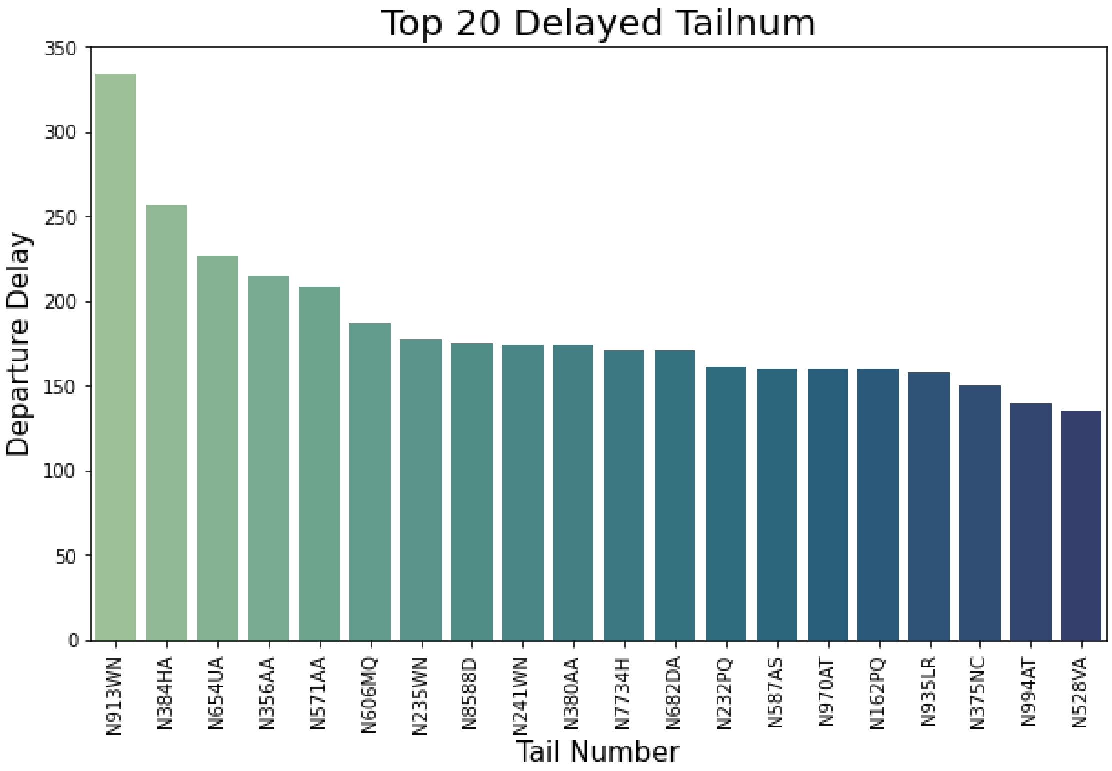
[21]:

```
delay_data = data[['tailnum', 'dep_delay', 'arr_delay']].groupby('tailnum').
↪mean().sort_values('dep_delay', ascending=False).reset_index()
```

[22]:

```
fig, ax = plt.subplots(figsize=(10, 6))
```

```
ax = sns.barplot(x='tailnum', y='dep_delay', data=delay_data.iloc[:20], ax=ax,␣
↪palette='crest')

ax.set_xticklabels(delay_data.iloc[:20]['tailnum'], rotation=90)
ax.set_xlabel('Tail Number', size=15)
ax.set_ylabel('Departure Delay', size=15)
ax.set_title('Top 20 Delayed Tailnum', size=20)
plt.show();
```



Seems that tailnum **N913WN** causes the most delay.

[23]:
```
delay_data.drop(columns='tailnum').corr()
```

[23]:
```
           dep_delay  arr_delay
dep_delay   1.000000   0.913046
arr_delay   0.913046   1.000000
```

There is actually highly correlated relationship between these two delays as mentioned above.

Also we are mainly focusing on departure delay, we can drop arrival delay feature

[24]:
```
data.drop(columns='arr_delay', inplace=True)
```

Because there are 3490 unique tailnum values, is it good to keep them as categorical feature or drop it? It's possible that some tailnum values have higher than others because they happen more.

[25]:

```
tailnum_count = data[['dep_delay', 'tailnum']].groupby('tailnum').count().
 ↪sort_index().rename(columns={'dep_delay':'count_delay'})
tailnum_count['avg_delay'] = data[['dep_delay', 'tailnum']].groupby('tailnum').
 ↪mean().sort_index()['dep_delay']
tailnum_count.sort_values('avg_delay', ascending=False).head()
```

[25]:

```
         count_delay  avg_delay
tailnum
N913WN             1      334.0
N384HA             5      257.0
N654UA             1      227.0
N356AA             1      215.0
N571AA             4      208.5
```

[26]:

```
tailnum_count.corr()
```

[26]:

```
             count_delay  avg_delay
count_delay     1.000000   0.012759
avg_delay       0.012759   1.000000
```

Based on the counts and average delays as well as the correlation between these two, tailnum doesn't seem to impact much on the delay. This also makes some sense because for example with N913WN, the delay happened only once with 334.0 value with highest delay. But this could just be due to air traffic of other planes and/or surrounding environment. With this reasnoning, try modeling without this tailnum feature

[27]:

```
data.drop(columns='tailnum', inplace=True)
```

Carrier & Departure Delay

Apply similar approach with carrier that's done to tailnum above

[28]:

```
carrier_count = data[['dep_delay', 'carrier']].groupby('carrier').count().
 ↪sort_index().rename(columns={'dep_delay':'count_delay'})
carrier_count['avg_delay'] = data[['dep_delay', 'carrier']].groupby('carrier').
 ↪mean().sort_index()['dep_delay']
carrier_count.sort_values('avg_delay', ascending=False)
```

[28]:

```
         count_delay  avg_delay
carrier
HA                34  38.529412
OO                 3  22.000000
EV              5142  20.066122
FL               307  18.368078
YV                53  18.264151
WN              1261  17.381443
```

```
9E                1696  17.285967
B6                5376  13.137091
F9                  69  12.811594
UA                5770  12.725650
VX                 497  12.722334
MQ                2507   9.617870
AA                3188   9.142409
DL                4751   8.529573
AS                  66   5.181818
US                2015   4.030769
```

Unlike tailnum, there are fixed number of carrier and not a lot, their counts are quite high and their average delay can be used. But is there correlation?

[29]:

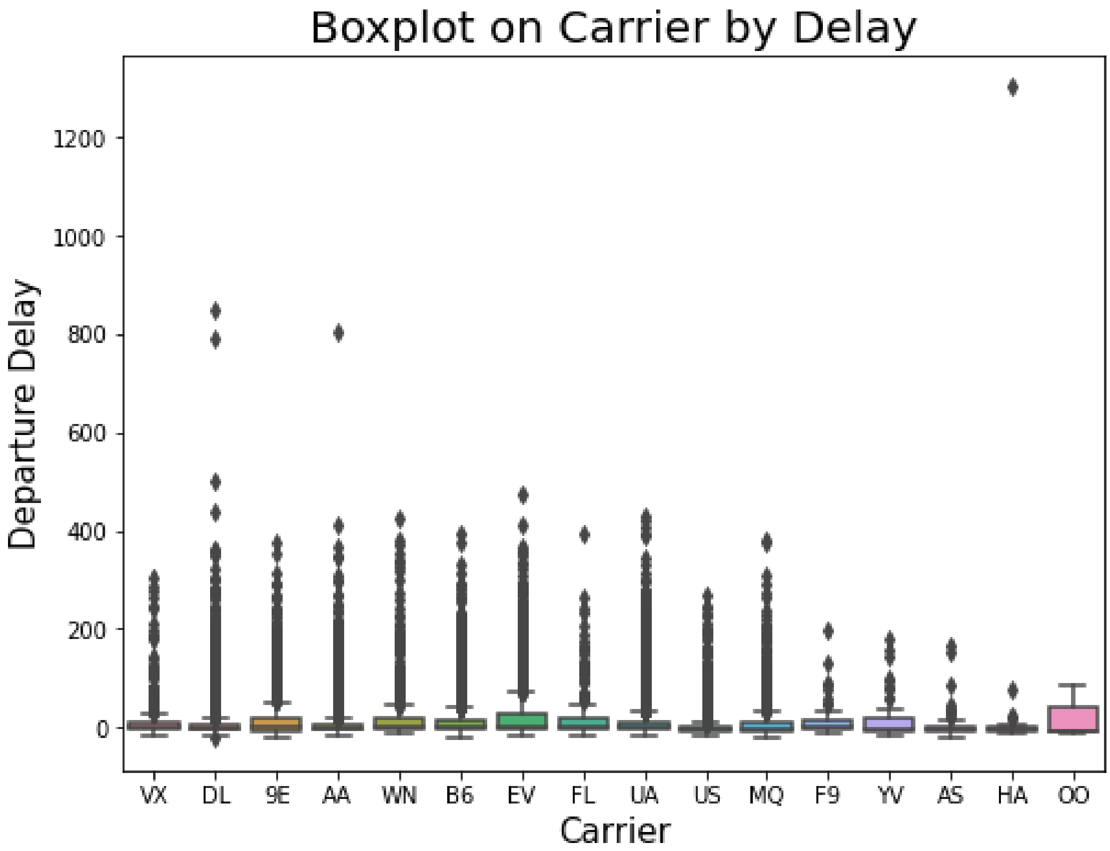
```
carrier_count.corr()
```

[29]:

```
             count_delay  avg_delay
count_delay     1.000000  -0.282994
avg_delay      -0.282994   1.000000
```

[30]:

```
plt.figure(figsize=(8, 6))
sns.boxplot(x='carrier', y='dep_delay', data=data)

plt.xlabel('Carrier', size=15)
plt.ylabel('Departure Delay', size=15)
plt.title('Boxplot on Carrier by Delay', size=20)

plt.show();
```



They have negative correlation but the numbers are not big enough to drop carrier predictor. Also it's actually possible carrier affects the delay as some may have more planes at NYC airport to cause traffic congestion so it could be a good idea to keep this feature.

Several outliers seen. Majority of delays don't exceed 400 minutes. A handful of airlines have delays less than 400 minutes

---

Departure Time

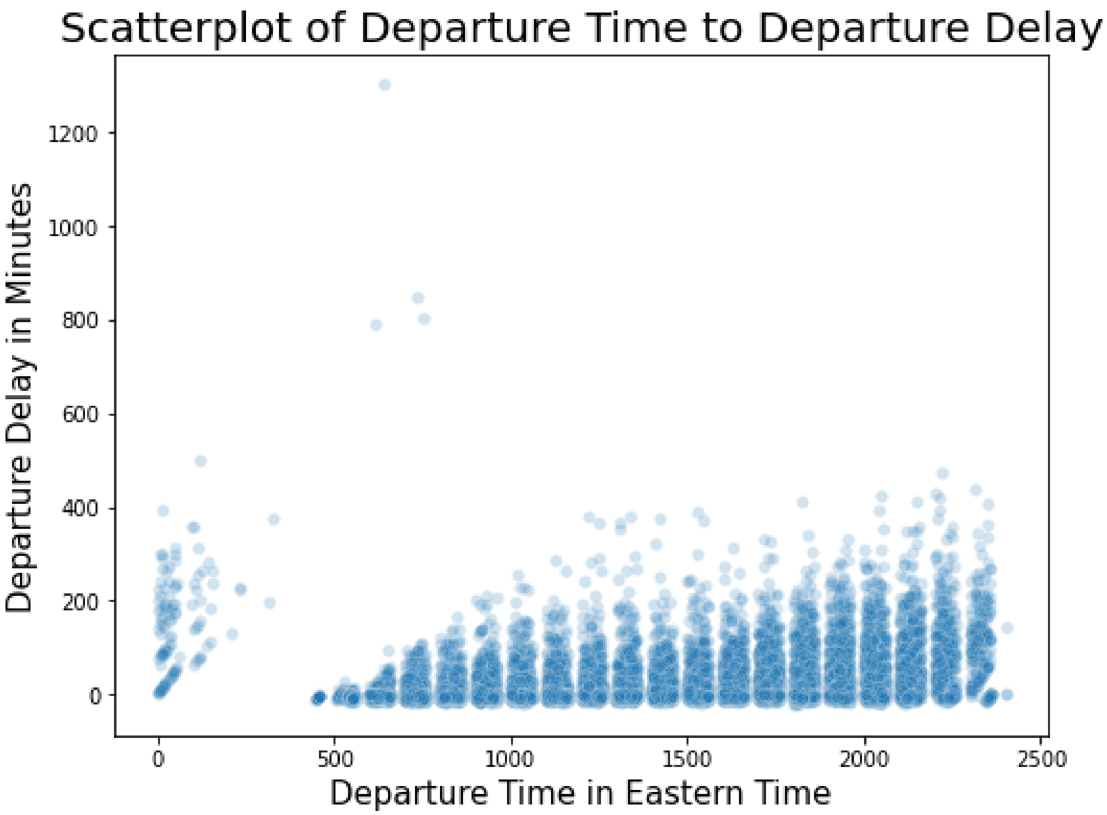
```
[31]: data['dep_time'].value_counts()
```

```
[31]: 755     101
      655      90
      556      87
      557      84
      656      79
             …
      152       1
      103       1
      119       1
```

```
117      1
232      1
Name: dep_time, Length: 1212, dtype: int64
```

[32]:

```python
# scatterplot of departure time and departure delays
# departure time noted in military time
fig, ax = plt.subplots(figsize=(8, 6))

ax = sns.scatterplot(x='dep_time', y='dep_delay', data=data, alpha=0.2, ax=ax)

ax.set_title('Scatterplot of Departure Time to Departure Delay', size=20)
ax.set_xlabel('Departure Time in Eastern Time', size=15)
ax.set_ylabel('Departure Delay in Minutes', size=15)

plt.show();
```



Steady increase in departure delay time as the day progresses from around 5AM to midnight. This is followed by a sharp decline in departure delay.

Arrival Time

Because arrival can be deduced from dep_time, dep_delay, and air_time, drop arr_time

[33]:

```
data.drop(columns='arr_time', inplace=True)
```

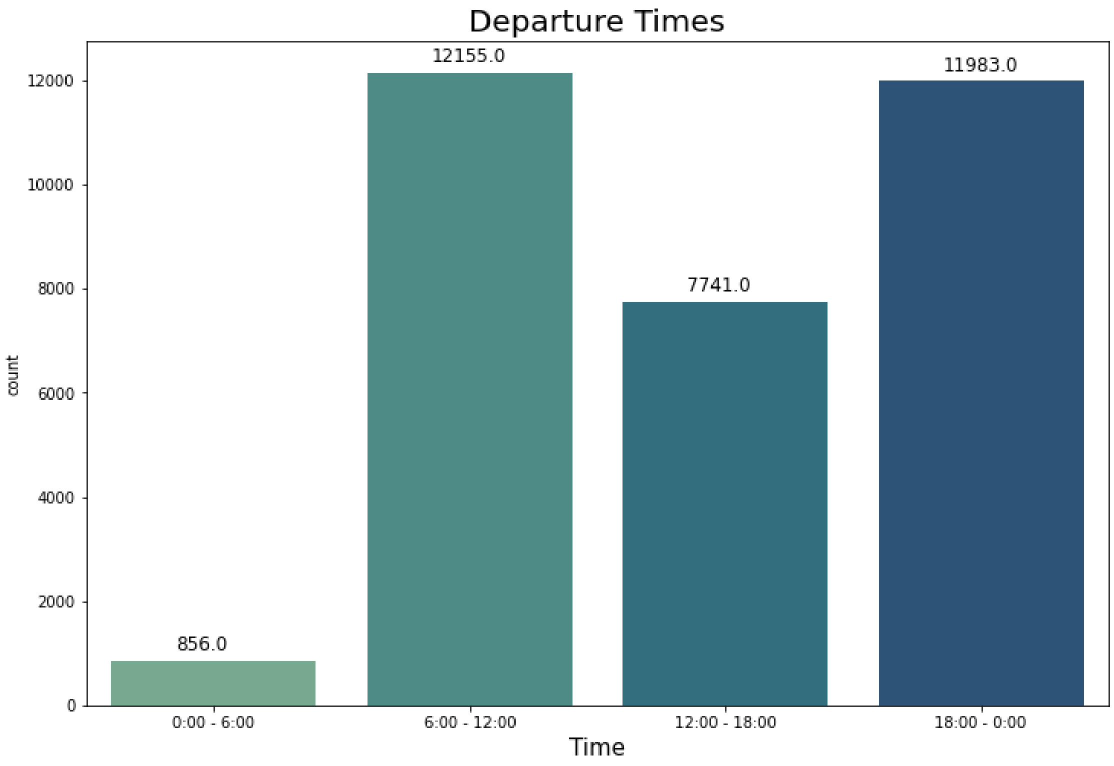
Transforming dep_time into a cetagorical predictor

[34]:

```
#Transform time variable into four categorical values
departure = []
for hour in data['dep_time']:
    aux = 0
    if hour < 600:
        aux = 0
    elif hour >= 600 and hour < 1200:
        aux = 1
    elif hour >= 1200 and hour < 1600:
        aux = 2
    else:
        aux = 3
    departure.append(aux)

data['dep_time'] = departure
```

[35]:

```
fig, ax = plt.subplots(figsize=(12, 8))

ax = sns.countplot(x="dep_time", data=data, palette='crest', ax=ax)

for p in ax.patches:
    ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x()+.25, p.
 ↪get_height()+200), size=12)
ax.set_xlabel('Time', size=15)
ax.set_title('Departure Times', size=20)
ax.set(xticklabels=["0:00 - 6:00", "6:00 - 12:00", "12:00 - 18:00", "18:00 - 0:
 ↪00"])

plt.show();
```



Instead of directly using the departure time as-is, it may be easier to predict the outcome with this transformed feature. Since we are implementing robust models that predict whether there will be less-than-15-minutes-delay or above, we don't have to have exact time of departure. By dividing this into 4 clusters, it could give a boost in the performance of models.

Data Transformation

[36]:
```
data.head()
```

[36]:
```
   month  day  dep_time  dep_delay carrier  flight origin  air_time  distance
0      6   30         1         15      VX     407    JFK       313      2475
1      5    7         3         -3      DL     329    JFK       216      1598
2     12    8         1         -1      DL     422    JFK       376      2475
3      5   14         3         -4      DL    2391    JFK       135      1005
4      7   21         1         -3      9E    3652    LGA        50       296
```

Convert Carrier and Origin into Cateogorical features

[37]:
```
carrier = pd.get_dummies(data['carrier'])
origin = pd.get_dummies(data['origin'])

data.drop(columns=['carrier', 'origin'], inplace=True)
```

```
data = pd.concat([data, carrier, origin], axis=1)
```

Normalize month and day

[38]:

```
# Max value of month is 12
data['month'] = data['month']/12
# Max value of day is 31
data['day'] = data['day']/31
# Convert airtime in hours
data['air_time'] = data['air_time']/60
```

[39]:

```
data.head()
```

[39]:

```
      month       day  dep_time  dep_delay  flight  air_time  distance  9E  \
0  0.500000  0.967742         1         15     407  5.216667      2475   0
1  0.416667  0.225806         3         -3     329  3.600000      1598   0
2  1.000000  0.258065         1         -1     422  6.266667      2475   0
3  0.416667  0.451613         3         -4    2391  2.250000      1005   0
4  0.583333  0.677419         1         -3    3652  0.833333       296   1

   AA  AS  …  MQ  OO  UA  US  VX  WN  YV  EWR  JFK  LGA
0   0   0  …   0   0   0   0   1   0   0    0    1    0
1   0   0  …   0   0   0   0   0   0   0    0    1    0
2   0   0  …   0   0   0   0   0   0   0    0    1    0
3   0   0  …   0   0   0   0   0   0   0    0    1    0
4   0   0  …   0   0   0   0   0   0   0    0    0    1

[5 rows x 26 columns]
```

---

[40]:

```
# multicollinearity check
multi = data.iloc[:, :7]

multi_df = pd.DataFrame()
multi_df['Variable'] = multi.columns

multi_df['VIF'] = [variance_inflation_factor(multi.values, i)
                   for i in range(len(multi.columns))]

multi_df.sort_values('VIF', ascending=False).head(5)
```

[40]:

```
   Variable         VIF
5  air_time  175.040532
6  distance  157.112718
2  dep_time    4.633030
0     month    3.873216
1       day    3.600684
```

The above shows the variance inflation factor for non-categorical features (those that were applied OHE)

[41]:

```
data.head()
```

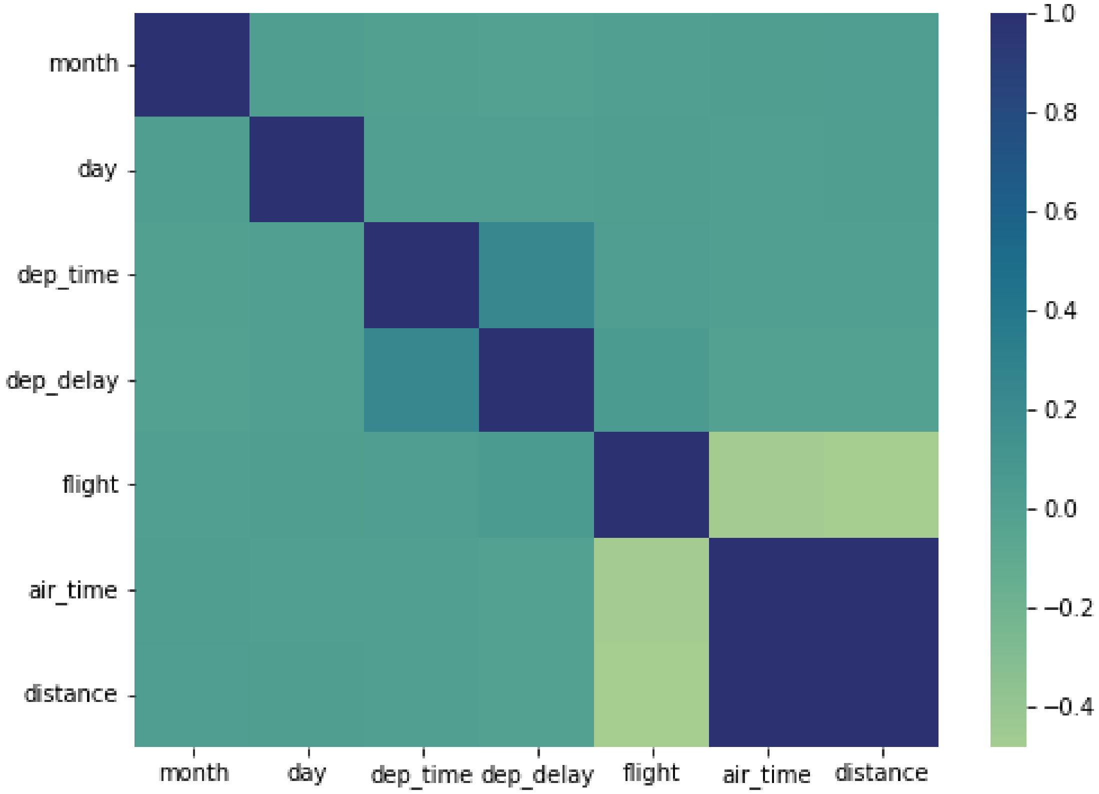
[41]:

```
      month       day  dep_time  dep_delay  flight  air_time  distance  9E  \
0  0.500000  0.967742         1         15     407  5.216667      2475   0
1  0.416667  0.225806         3         -3     329  3.600000      1598   0
2  1.000000  0.258065         1         -1     422  6.266667      2475   0
3  0.416667  0.451613         3         -4    2391  2.250000      1005   0
4  0.583333  0.677419         1         -3    3652  0.833333       296   1

   AA  AS  …  MQ  OO  UA  US  VX  WN  YV  EWR  JFK  LGA
0   0   0  …   0   0   0   0   1   0   0    0    1    0
1   0   0  …   0   0   0   0   0   0   0    0    1    0
2   0   0  …   0   0   0   0   0   0   0    0    1    0
3   0   0  …   0   0   0   0   0   0   0    0    1    0
4   0   0  …   0   0   0   0   0   0   0    0    0    1

[5 rows x 26 columns]
```

Show Correlation without carrier and origin features

[42]:

```
fig, ax = plt.subplots(figsize=(8, 6))

sns.heatmap(data.iloc[:, :7].corr(), cmap='crest', ax=ax)

plt.show();
```



We can see only air_time and distance have high correlation values which makes sense since the farther the distance is, the more time it takes to get there.

1.0.4 Model Implementation and Evaluation

Our goal is to predict if there will be a delay equal or more than 15 minutes

[44]:
```
data['long_delay'] = 0
data.loc[data['dep_delay']>=15, 'long_delay'] = 1

data.head()
```

[44]:
```
      month       day  dep_time  dep_delay  flight  air_time  distance  9E  \
0  0.500000  0.967742         1         15     407  5.216667      2475   0
1  0.416667  0.225806         3         -3     329  3.600000      1598   0
2  1.000000  0.258065         1         -1     422  6.266667      2475   0
3  0.416667  0.451613         3         -4    2391  2.250000      1005   0
4  0.583333  0.677419         1         -3    3652  0.833333       296   1

   AA  AS  …  OO  UA  US  VX  WN  YV  EWR  JFK  LGA  long_delay
0   0   0  …   0   0   0   1   0   0    0    1    0           1
```

```
1   0   0  …   0   0   0   0   0   0    0    1    0            0
2   0   0  …   0   0   0   0   0   0    0    1    0            0
3   0   0  …   0   0   0   0   0   0    0    1    0            0
4   0   0  …   0   0   0   0   0   0    0    0    1            0

[5 rows x 27 columns]
```

[45]:
```python
data['long_delay'].value_counts()
```

[45]:
```
0    25430
1     7305
Name: long_delay, dtype: int64
```

[46]:
```python
data.drop(columns='dep_delay', inplace=True)
```

Because there are only 7305 records for y=1, have 1000 from each y as test dataset

[47]:
```python
test_index = data[data['long_delay']==1].sample(1000,␣
 ↪random_state=random_state).index.tolist() \
           + data[data['long_delay']==0].sample(1000,␣
 ↪random_state=random_state).index.tolist()

train, test = data.drop(index=test_index), data.iloc[test_index]

X_train, X_test = train.drop(columns=['long_delay']), test.
 ↪drop(columns=['long_delay']),
y_train, y_test = train['long_delay'], test['long_delay']

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

[47]:
```
((30735, 25), (2000, 25), (30735,), (2000,))
```

[62]:
```python
# Using SmoteTomek for class imabalance problem
resample = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'))

# Use cross validation to validate models instead of separately creating␣
 ↪validation datasets using for loops
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=1, random_state=random_state)

scoring = ['accuracy','precision_macro','recall_macro']
```

Models

[72]:
```python
rfc = RandomForestClassifier(random_state=random_state)
sgd = SGDClassifier(random_state=random_state)
gbc = GradientBoostingClassifier()
bagging = BaggingClassifier(n_estimators=100)
lr = LogisticRegressionCV(random_state=random_state, max_iter=1000)
```

```
lda = LinearDiscriminantAnalysis()

rfc_pipeline = Pipeline(steps=[('resample', resample), ('rfc', rfc)])
sgd_pipeline = Pipeline(steps=[('resample', resample), ('sgd', sgd)])
gbc_pipeline = Pipeline(steps=[('resample', resample), ('gbc', gbc)])
bagging_pipeline = Pipeline(steps=[('resample', resample), ('bagging',␣
 ↪bagging)])
lr_pipeline = Pipeline(steps=[('resample', resample), ('lr', lr)])
lda_pipeline = Pipeline(steps=[('resample', resample), ('lda', lda)])

rfc_pipeline.fit(X_train, y_train)
sgd_pipeline.fit(X_train, y_train)
gbc_pipeline.fit(X_train, y_train)
bagging_pipeline.fit(X_train, y_train)
lr_pipeline.fit(X_train, y_train)
lda_pipeline.fit(X_train, y_train)

rfc_scores = cross_validate(rfc_pipeline, X_train, y_train, scoring=scoring,␣
 ↪cv=cv, n_jobs=-1)
sgd_scores = cross_validate(sgd_pipeline, X_train, y_train, scoring=scoring,␣
 ↪cv=cv, n_jobs=-1)
gbc_scores = cross_validate(gbc_pipeline, X_train, y_train, scoring=scoring,␣
 ↪cv=cv, n_jobs=-1)
bagging_scores = cross_validate(bagging_pipeline, X_train, y_train,␣
 ↪scoring=scoring, cv=cv, n_jobs=-1)
lr_scores = cross_validate(lr_pipeline, X_train, y_train, scoring=scoring,␣
 ↪cv=cv, n_jobs=-1)
lda_scores = cross_validate(lda_pipeline, X_train, y_train, scoring=scoring,␣
 ↪cv=cv, n_jobs=-1)
```

[73]:

```
scores = []

for score in [rfc_scores, sgd_scores, gbc_scores, bagging_scores, lr_scores,␣
 ↪lda_scores]:
    scores.append({
        'Accuracy':score['test_accuracy'].mean(),
        'Precision':score['test_precision_macro'].mean(),
        'Recall':score['test_recall_macro'].mean(),
    })

scores = pd.DataFrame(scores, index=['RFC', 'SGD', 'GBC', 'Bagging', 'LR',␣
 ↪'LDA'])
scores.sort_values('Accuracy', ascending=False)
```

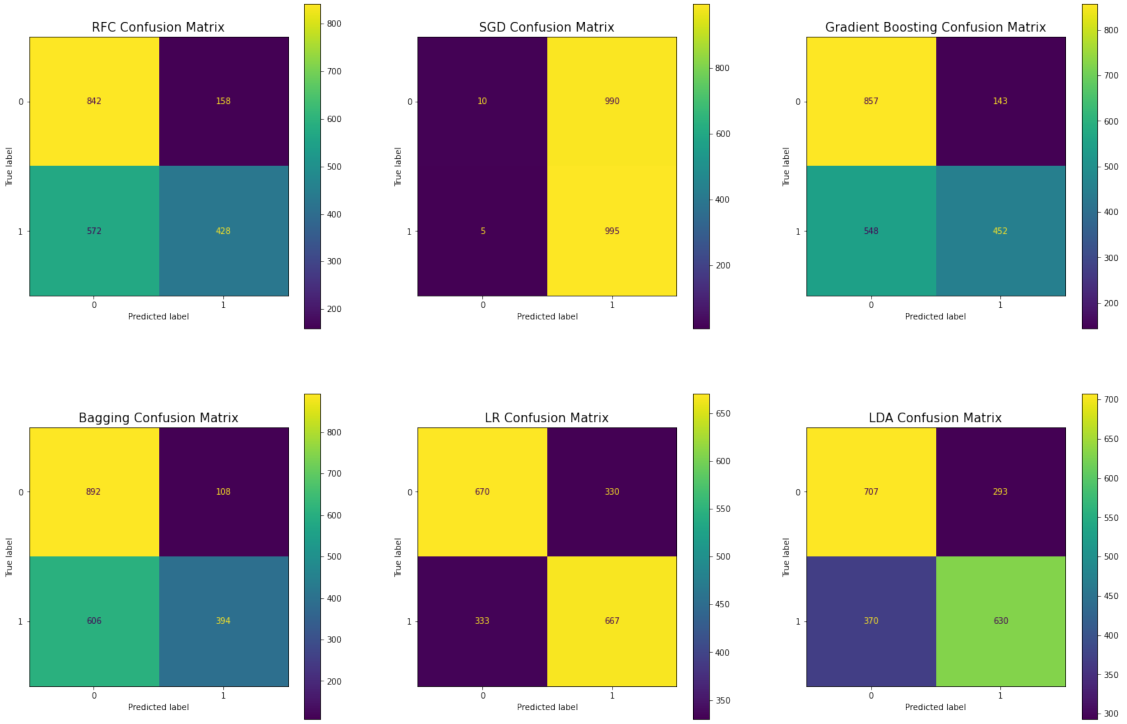
[73]:

```
         Accuracy  Precision    Recall
Bagging  0.783537   0.656096  0.631729
```

```
GBC       0.763787   0.636755  0.635604
RFC       0.746999   0.618310  0.623690
LDA       0.680690   0.609485  0.655699
LR        0.656385   0.608047  0.659885
SGD       0.515503   0.536169  0.531727
```

[74]:

```python
fig, ax = plt.subplots(2, 3, figsize=(24,16))

plot_confusion_matrix(rfc_pipeline, X_test, y_test, ax=ax[0][0])
ax[0][0].set_title('RFC Confusion Matrix', size=15)

plot_confusion_matrix(sgd_pipeline, X_test, y_test, ax=ax[0][1])
ax[0][1].set_title('SGD Confusion Matrix', size=15)

plot_confusion_matrix(gbc_pipeline, X_test, y_test, ax=ax[0][2])
ax[0][2].set_title('Gradient Boosting Confusion Matrix', size=15)

plot_confusion_matrix(bagging_pipeline, X_test, y_test, ax=ax[1][0])
ax[1][0].set_title('Bagging Confusion Matrix', size=15)

plot_confusion_matrix(lr_pipeline, X_test, y_test, ax=ax[1][1])
ax[1][1].set_title('LR Confusion Matrix', size=15)

plot_confusion_matrix(lda_pipeline, X_test, y_test, ax=ax[1][2])
ax[1][2].set_title('LDA Confusion Matrix', size=15);
```


RFC Confusion Matrix
SGD Confusion Matrix
Gradient Boosting Confusion Matrix
Bagging Confusion Matrix
LR Confusion Matrix
LDA Confusion Matrix
True label
Predicted label