REPORT

FLIGHT DELAY ANALYSIS

Team 03

Sonali Chaudhari
Madhumitha Prakash

EXPLORATORY DATA ANALYSIS (EDA):

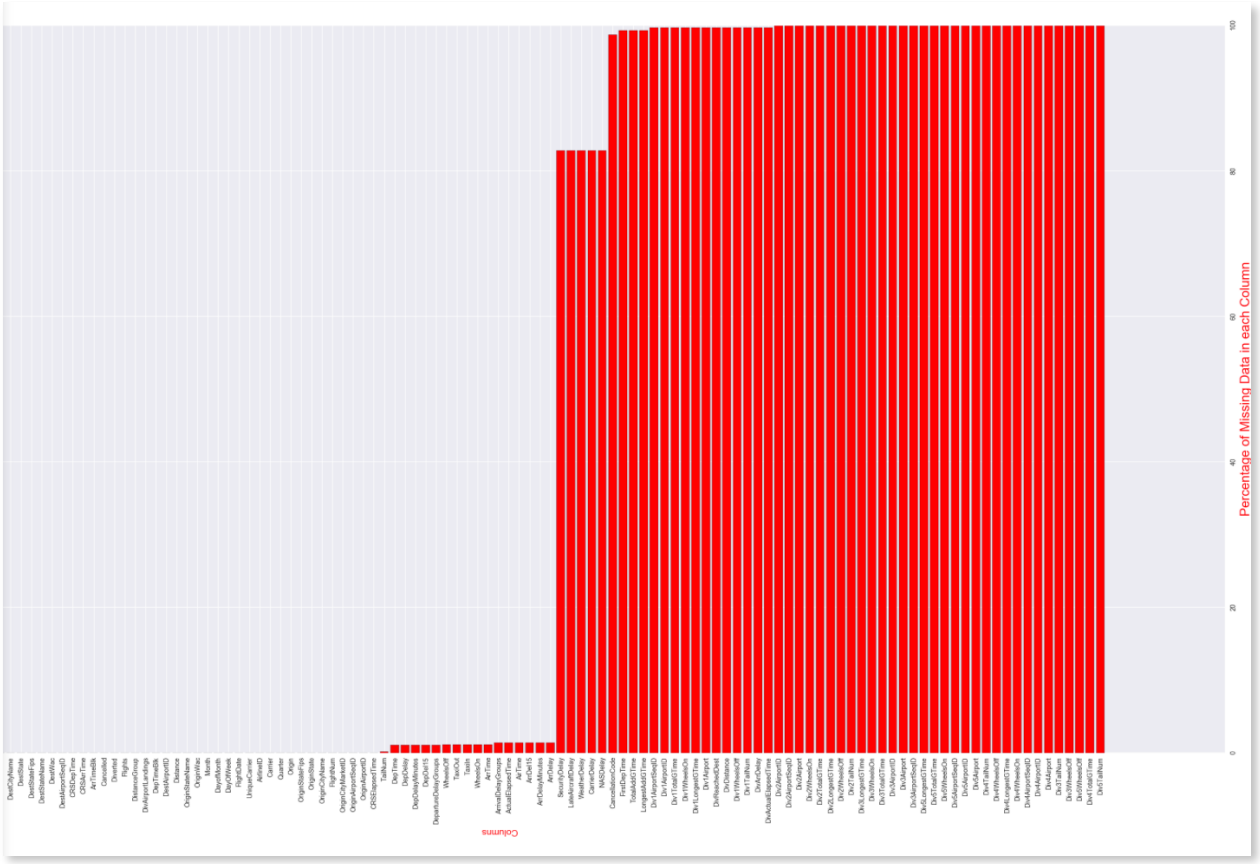
We worked on data of years 2016 and 2017. We performed EDA separately for 2016 and 2017 first and then EDA for combined data.

We have downloaded lookup tables for some columns for mapping the airlines name, airports names,etc.

```
cancel = pd.read_csv(cur_path+'/Lookup/CancellationCode.csv', low_memory=False)
airport = pd.read_csv(cur_path+'/Lookup/AirportID.csv', low_memory=False)
carrier = pd.read_csv(cur_path+'/Lookup/UniqueCarrier.csv', low_memory=False)
origin = pd.read_csv(cur_path+'/Lookup/Origin.csv', low_memory=False)
```

```
cancel_dict = cancel.set_index('Code').T.to_dict('list')
origin_dict = origin.set_index('Code').T.to_dict('list')
airport_dict = airport.set_index('Code').T.to_dict('list')
carrier_dict = carrier.set_index('Code').T.to_dict('list')
```

1. Missing Values

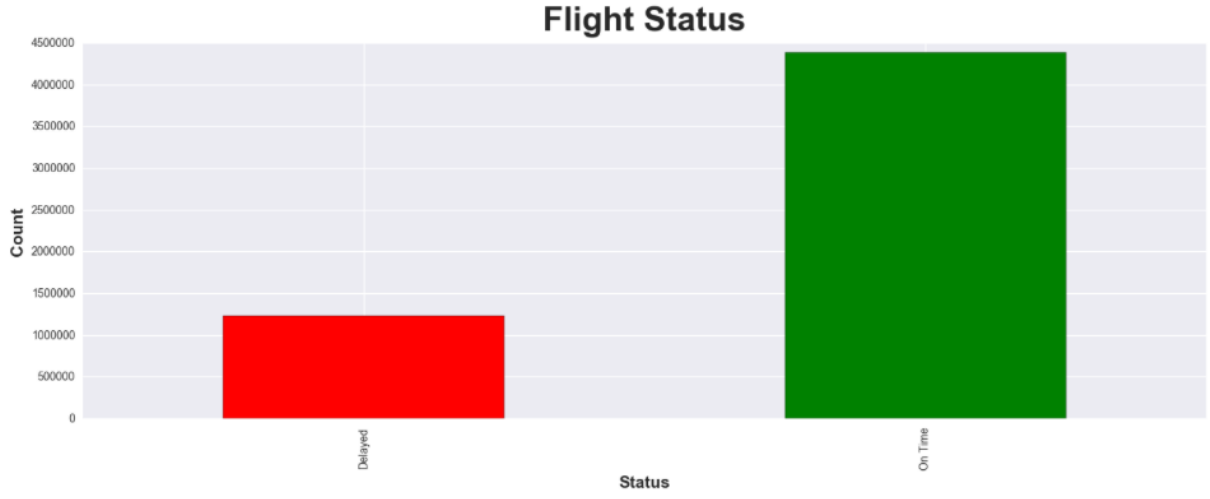


In the beginning we have 109 columns. From the EDA of missing data we se almost 46 columns have 98% missing data. We get rid of these columns

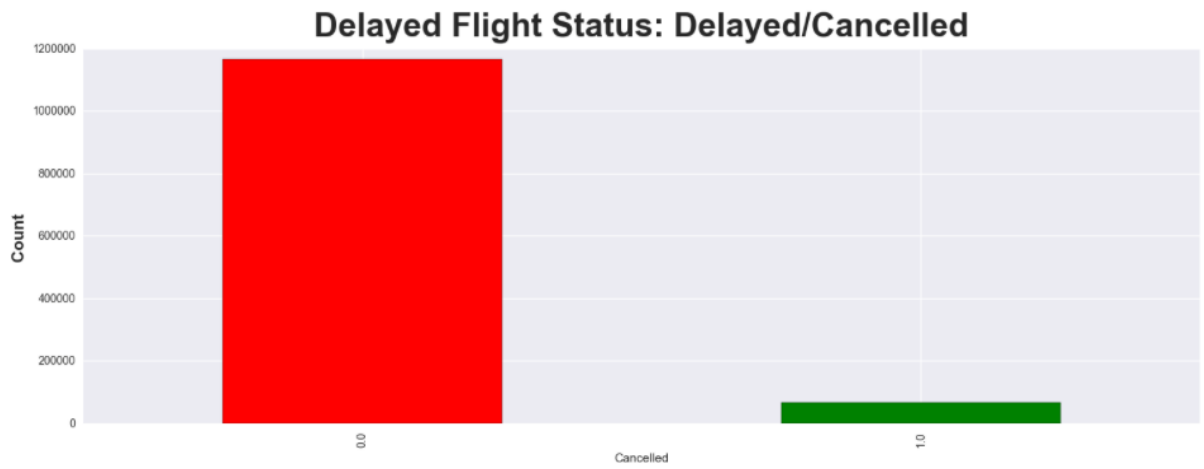
2. Count of flights Delayed and On-Time

Approximately 30% of the total flights were delayed in 2017.

```
Status
Delayed    1231265
On Time    4386393
Name: Status, dtype: int64
```

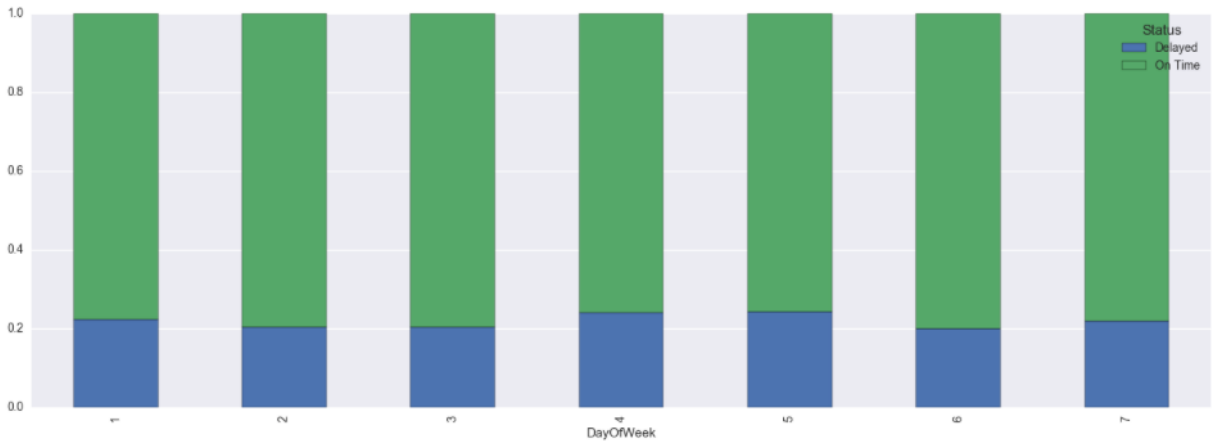


3. Count of delayed flights which were Cancelled



Out if the delayed flight very minimal flights are cancelled.

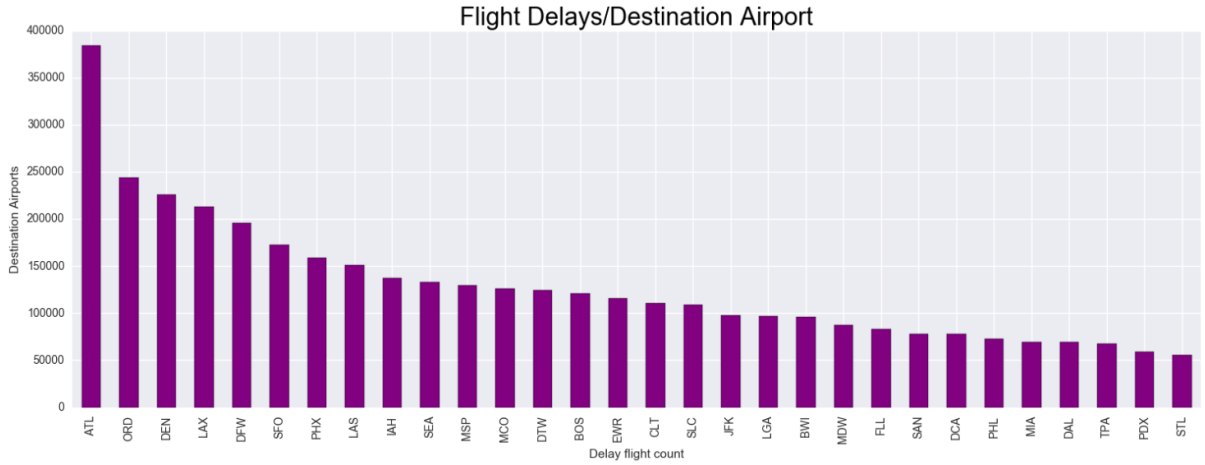
4. Percentage Distribution of Delayed/On-Time flights based on days of week.



Delay by Airports

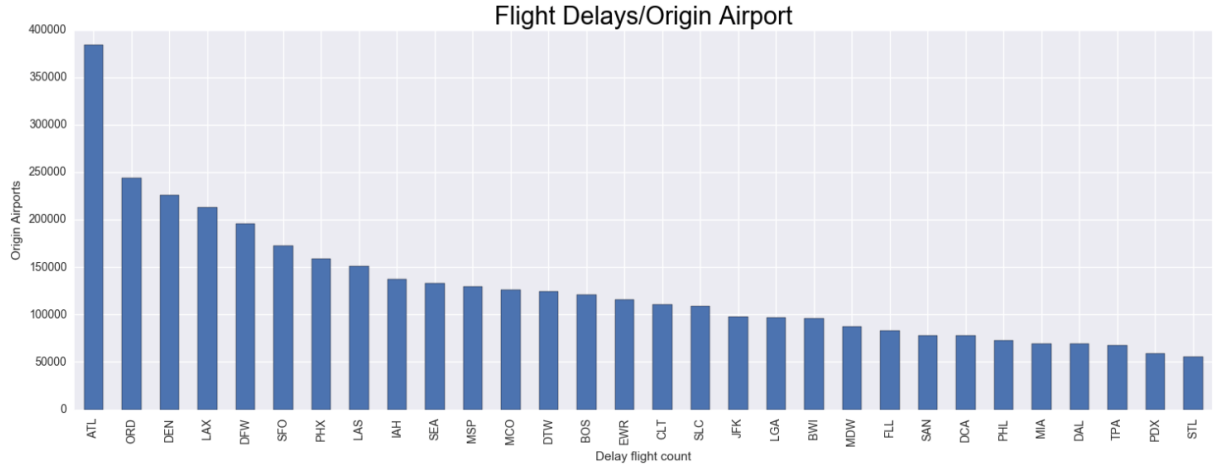
1- Denotes Monday
Thursday, Friday and Sunday have higher number of delayed flights comparatively.

5. Top 30 Destination Airports having flight delays

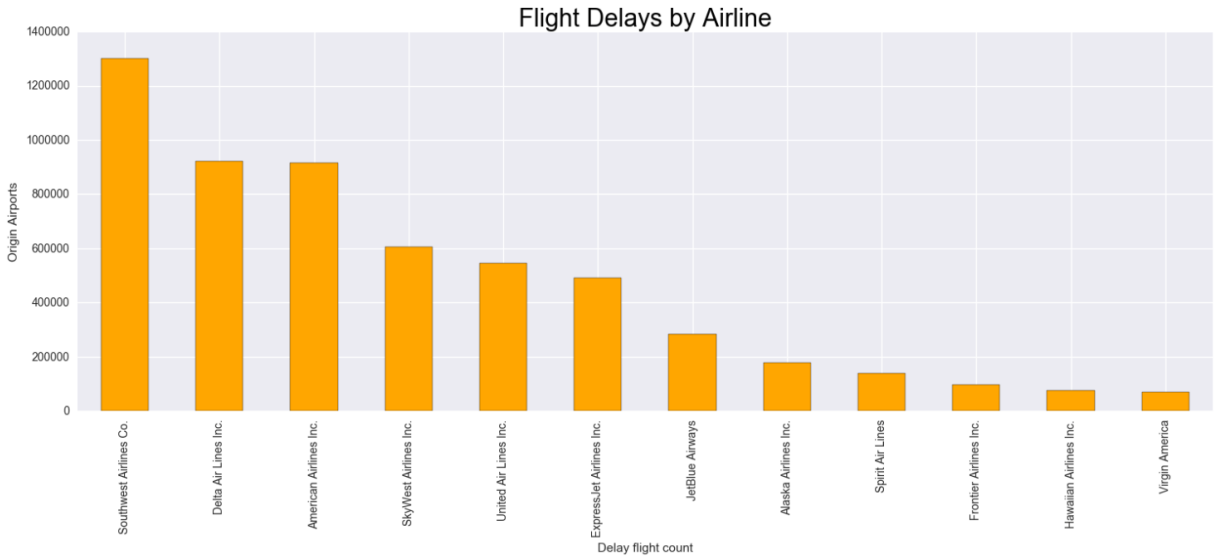


ATL- Atlanta has majority of delayed flights followed by ORD and DEN

6. Top 30 Origin Airports having flight delays:
   Same airport have the flight delay count as for departure airports flight delay

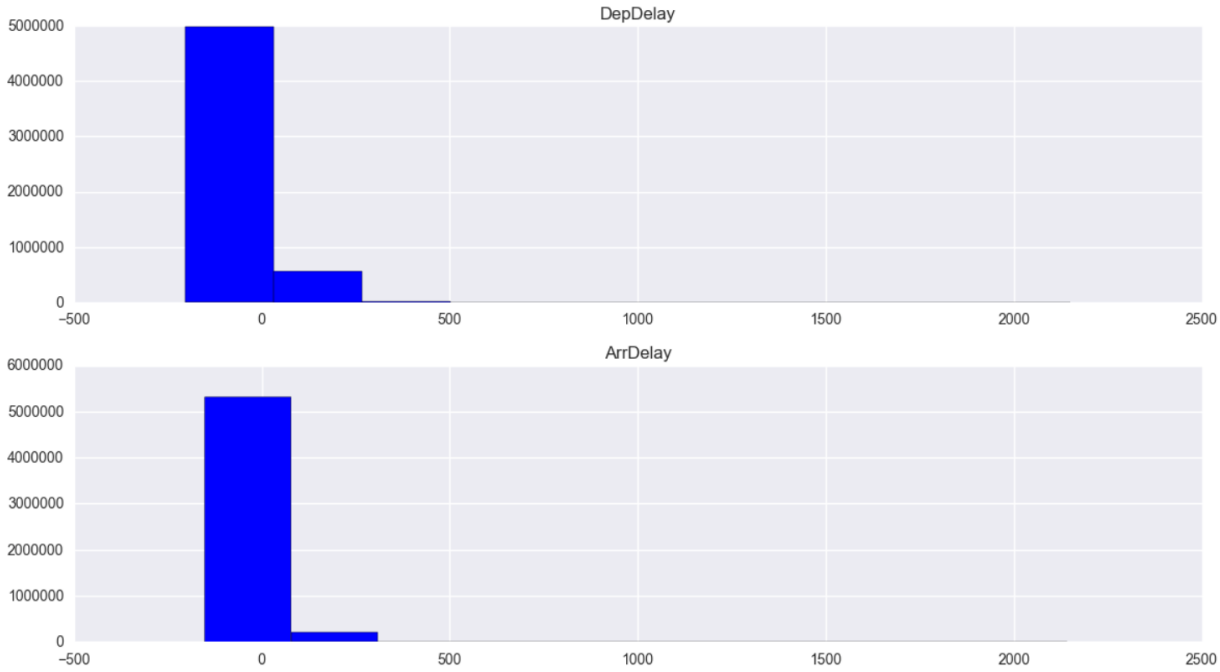


7. Flight Delays distribution by different Airlines:
   We can see that Southwest Airlines have the majority of flight delay in the year 2016



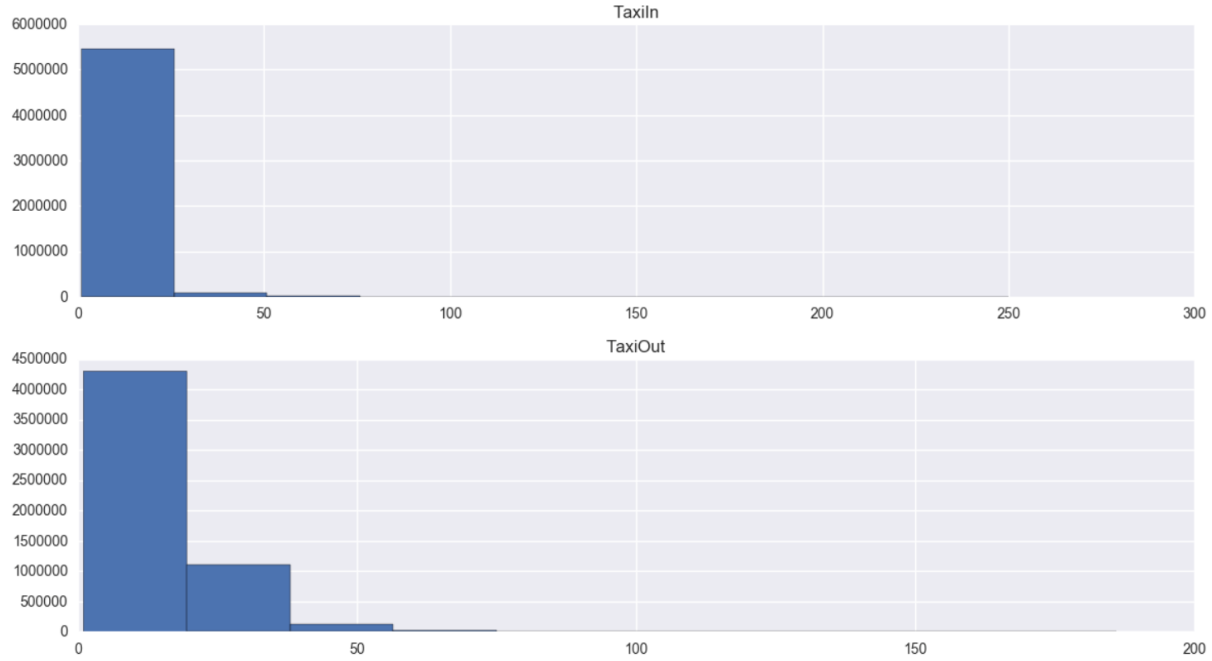
8. Departure Delay Distribution and Arrival Delay Distribution

```
Nan:  63456
Nan:  79513
```



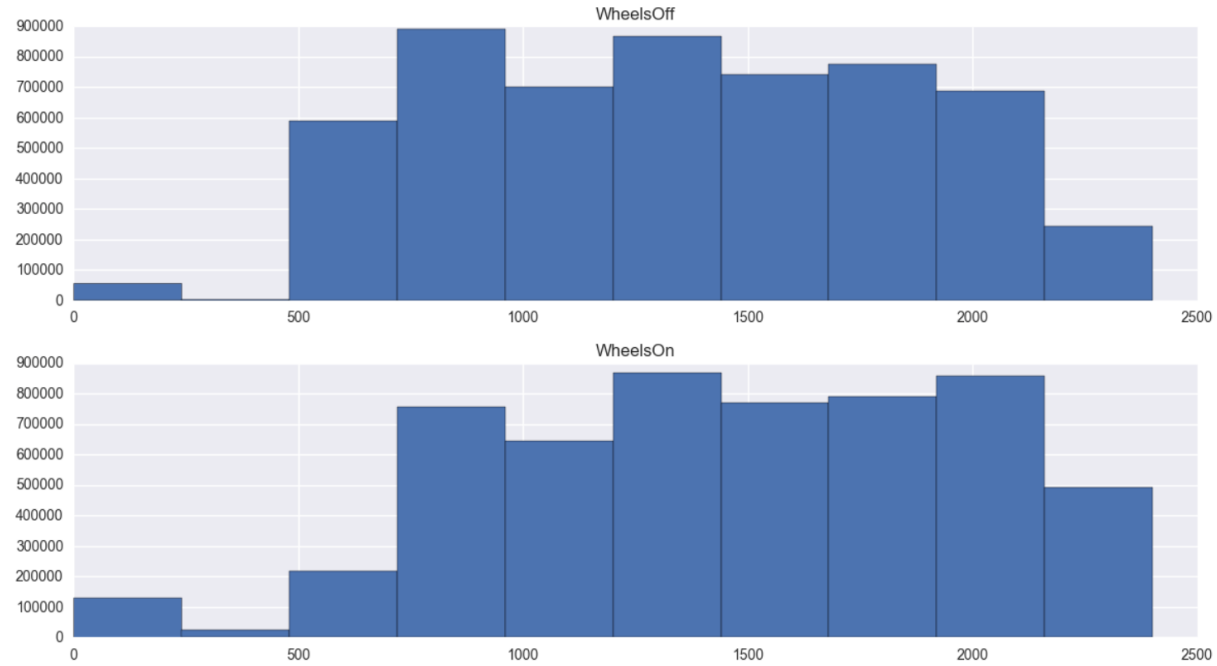
9. TaxiIn and TaxiOut Distribution

```
Nan:  65418
Nan:  67844
```



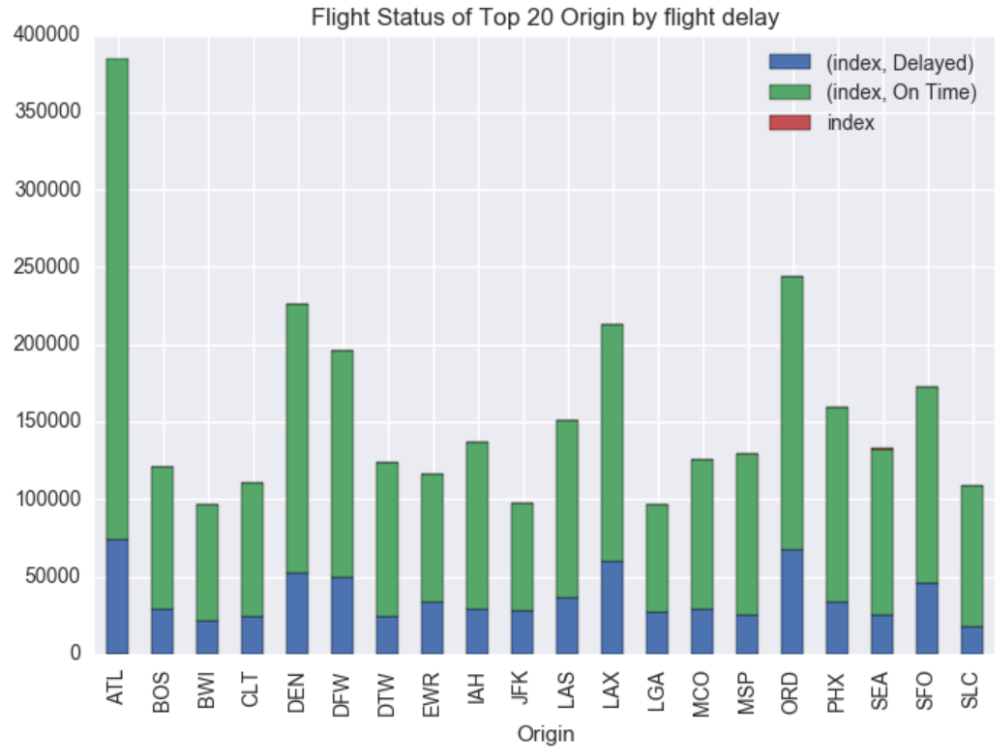
10. WheelsOn And WheelOf Distribution

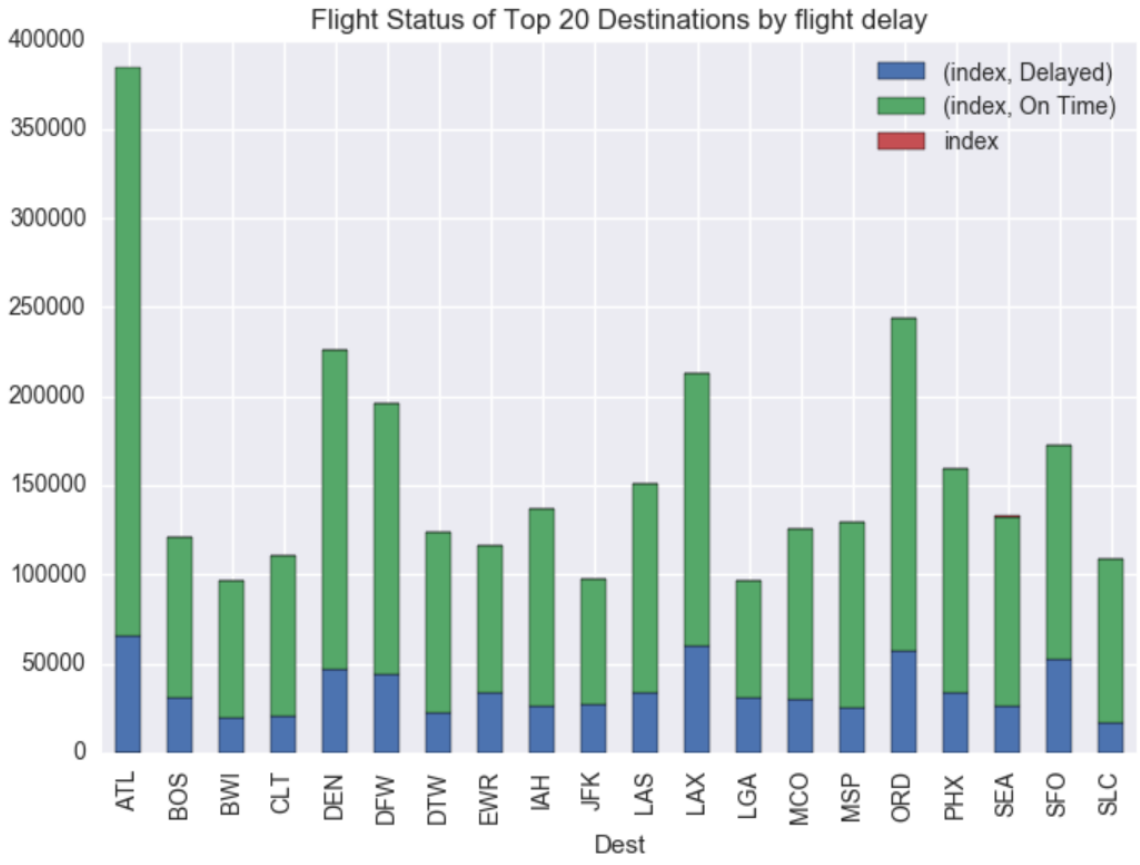
```
Nan:  65418
Nan:  67844
```



11. Flight Status of top 20 destinations and origin airports by flight delay:

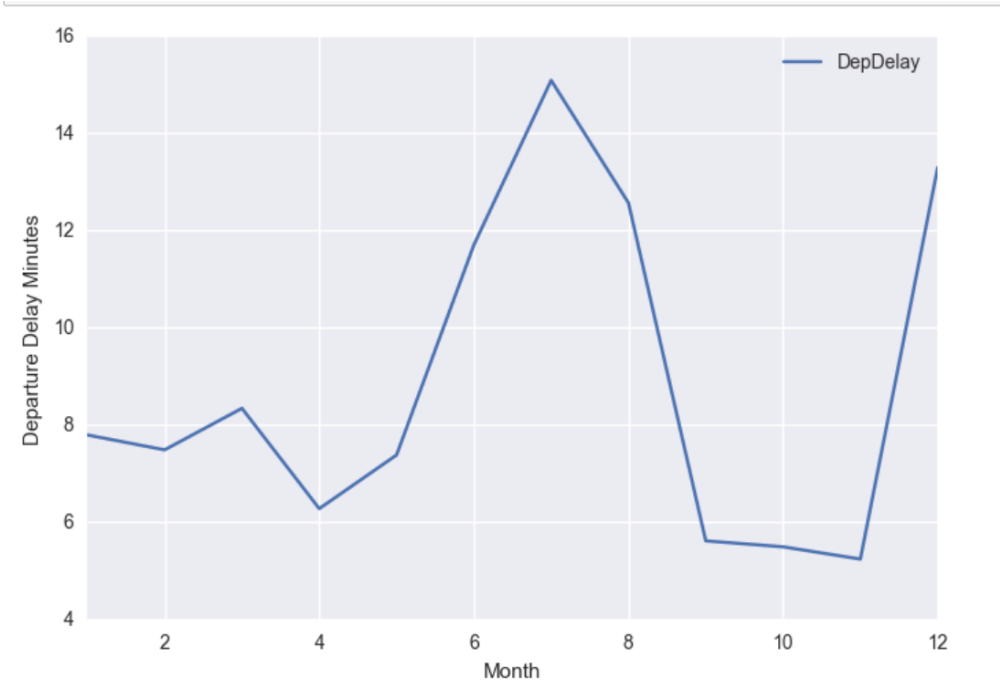
    Top 20 airports with total flights and onetime and departure delayed flights distribution. Atlanta and Chicago have the highest count of total and delayed flights. Similar goes for origin airports.



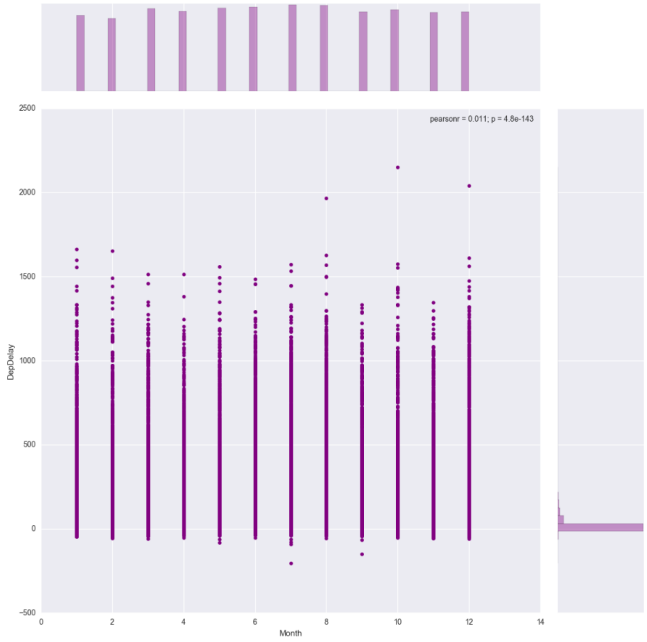


12. Departure delay trend over the months:

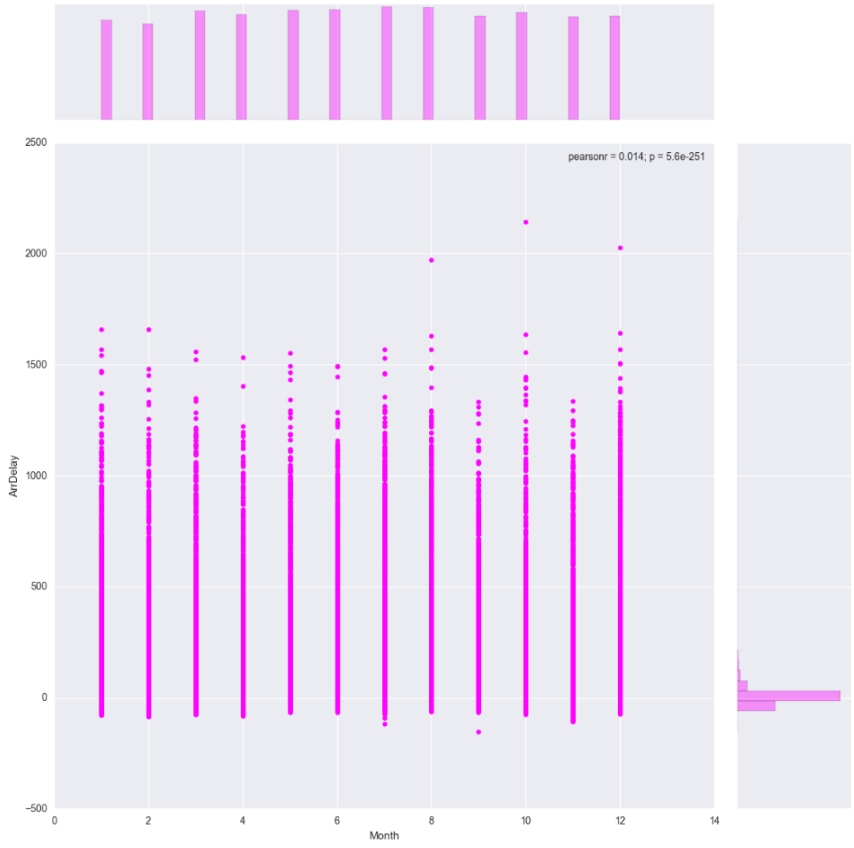
We plot the graph of mean departure delay over the months for 2016. We see the delay increases suddenly from April, it is hight in July after which it decrease and again increase for November and December. We can say that the delays are more during the Summer and Christmas breaks of the year during which people tend to travel a lot.



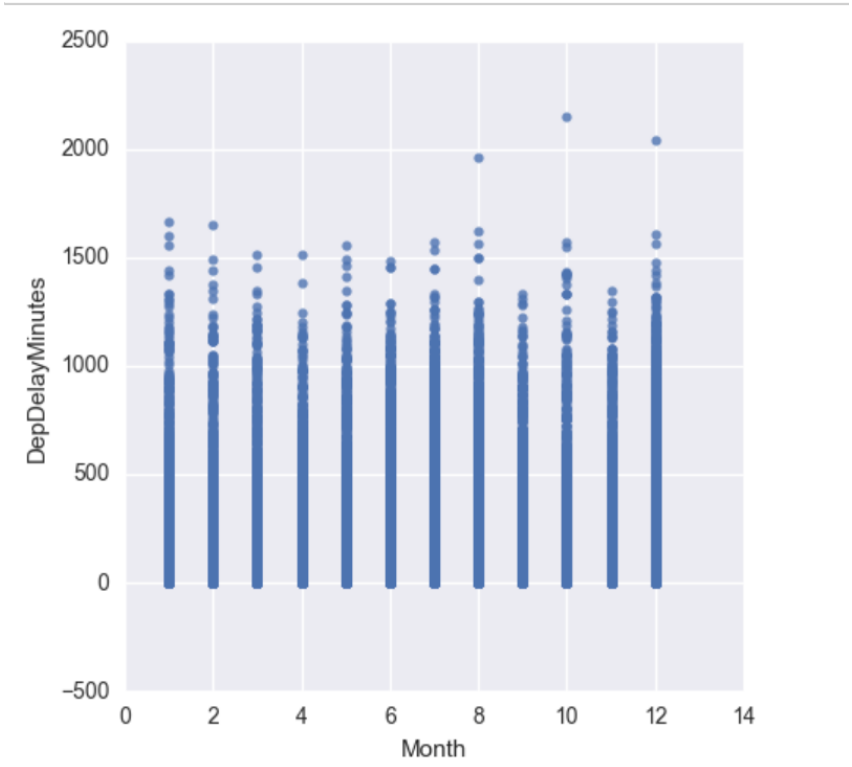
13. Flight delay during departure and arrival in minutes by month



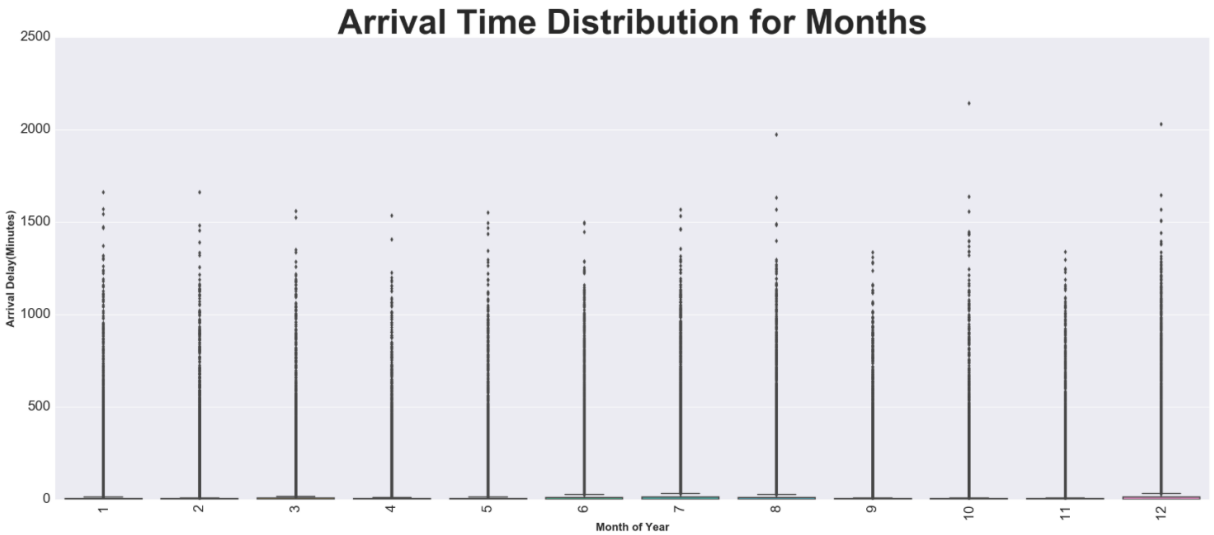
We plot a graph for distribution of departure delay by months. We see there are lot of outliers for every month. We have got git of these outliers for each month in the next part: data cleaning part

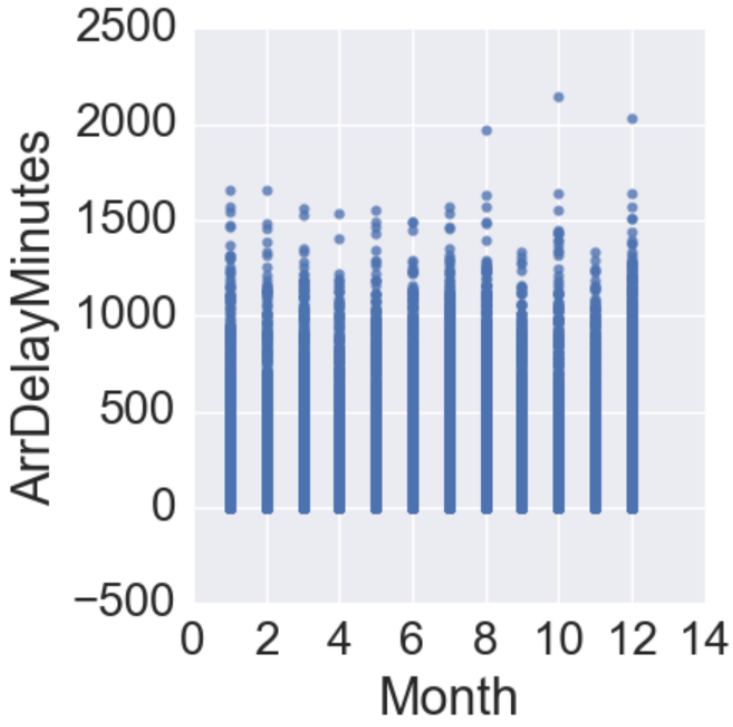


14. Distribution of departure delay minutes over the months

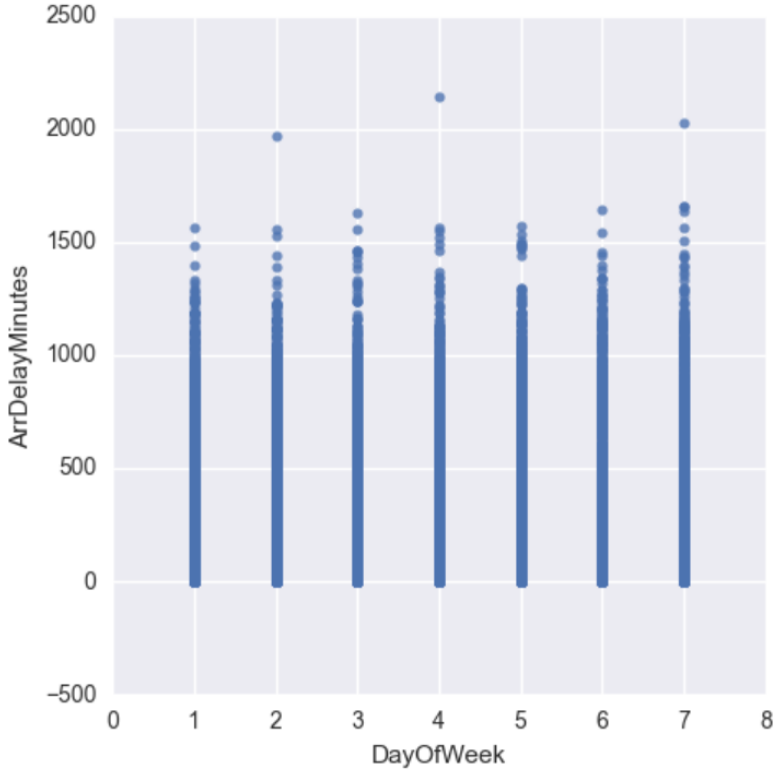


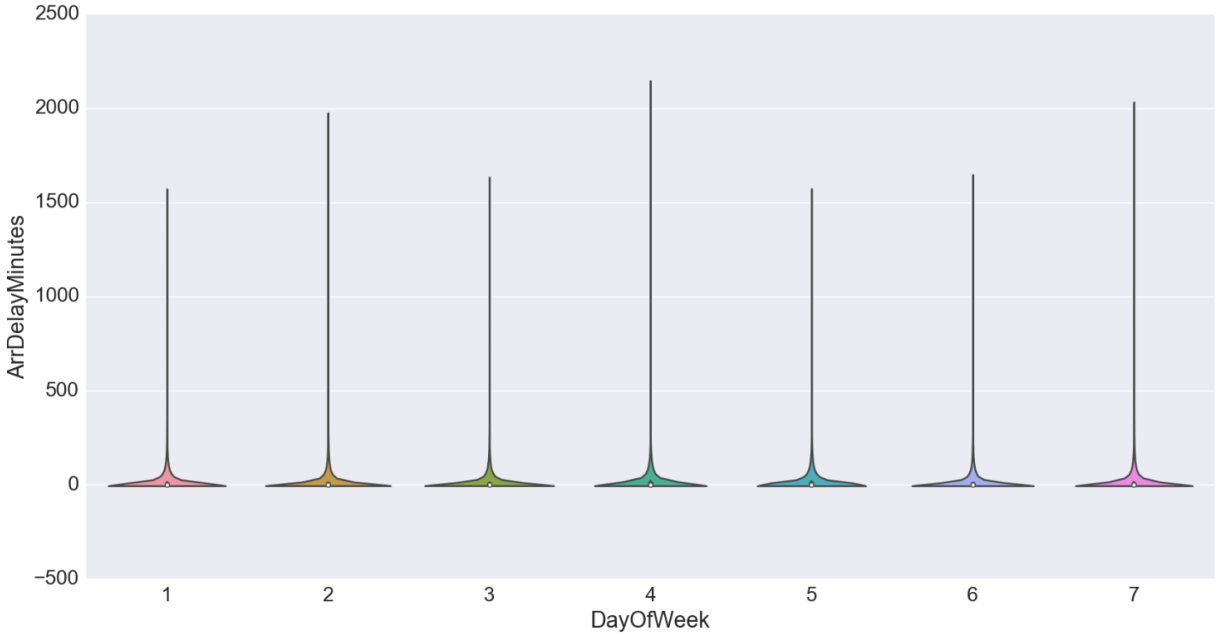
15. Distribution of arrival minutes over months



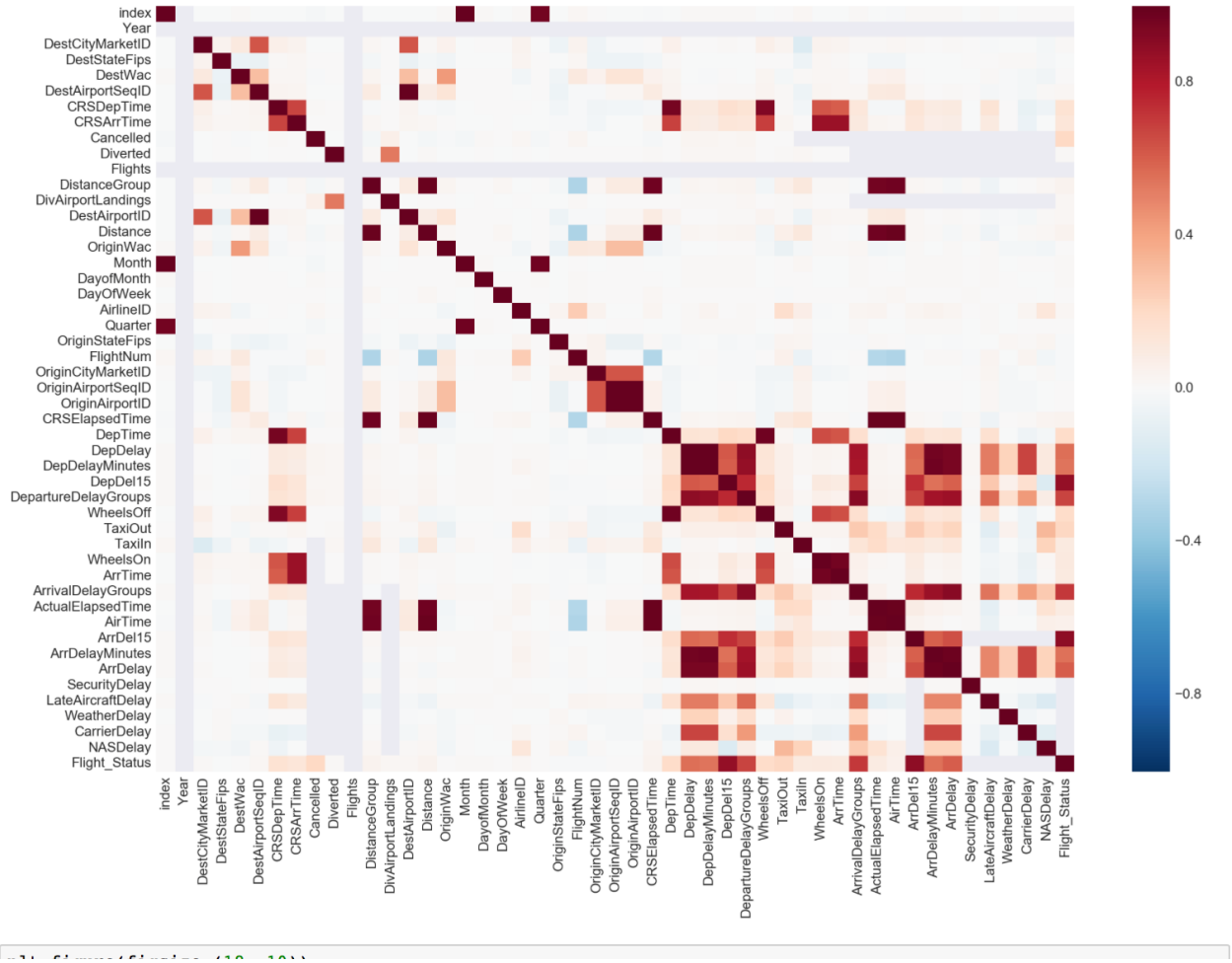


16. Delay distribution in minutes at the arrival end over the days of week

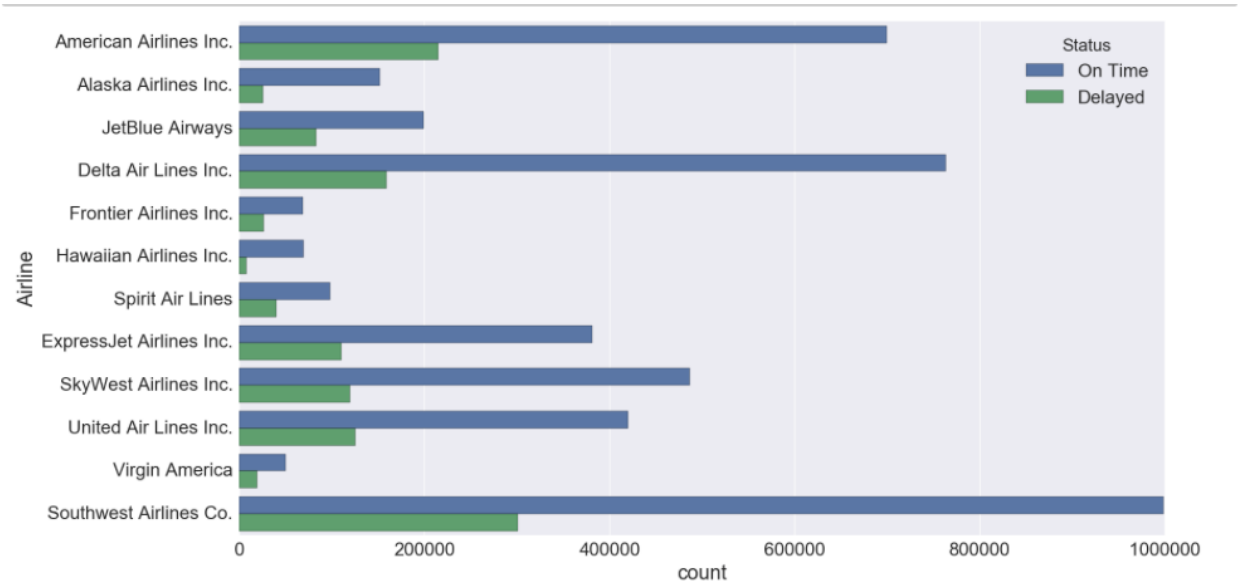




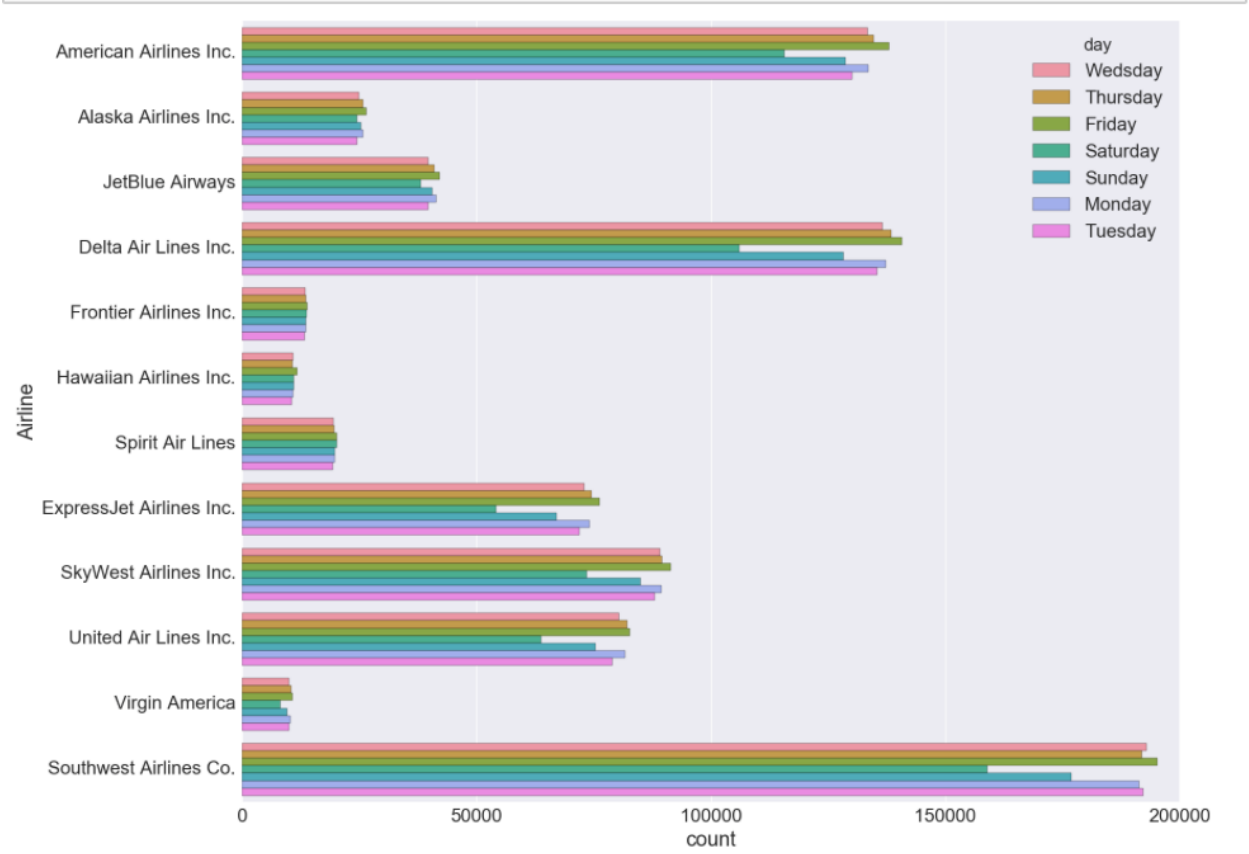
17. Heat map for correlation



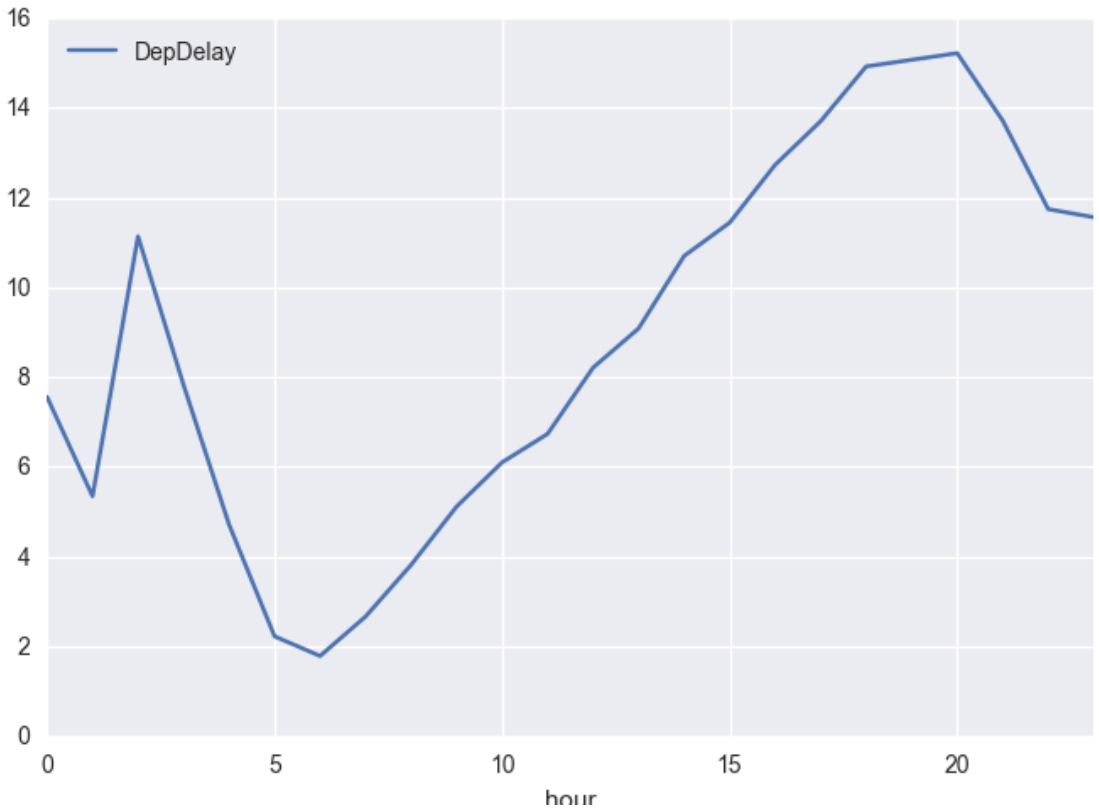
18. Flight Status based on various airlines



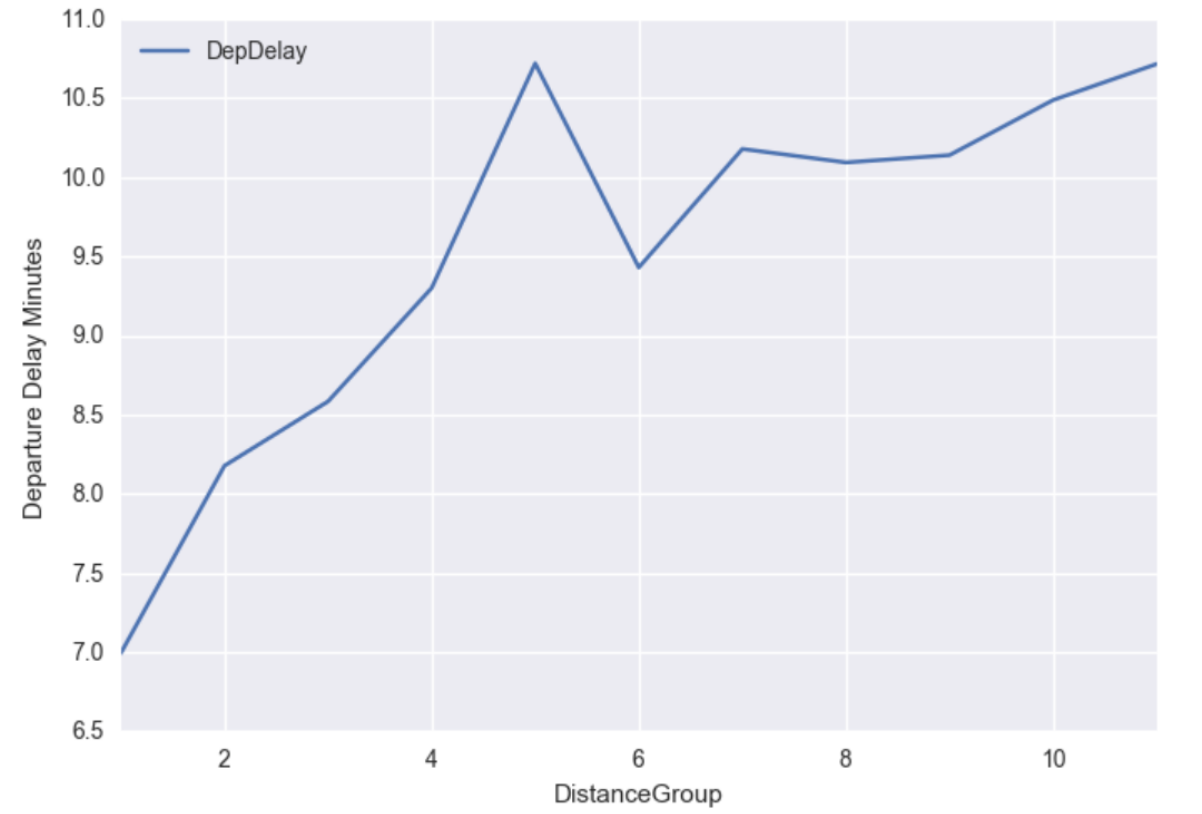
19. Count of flights for each day of week for various airlines



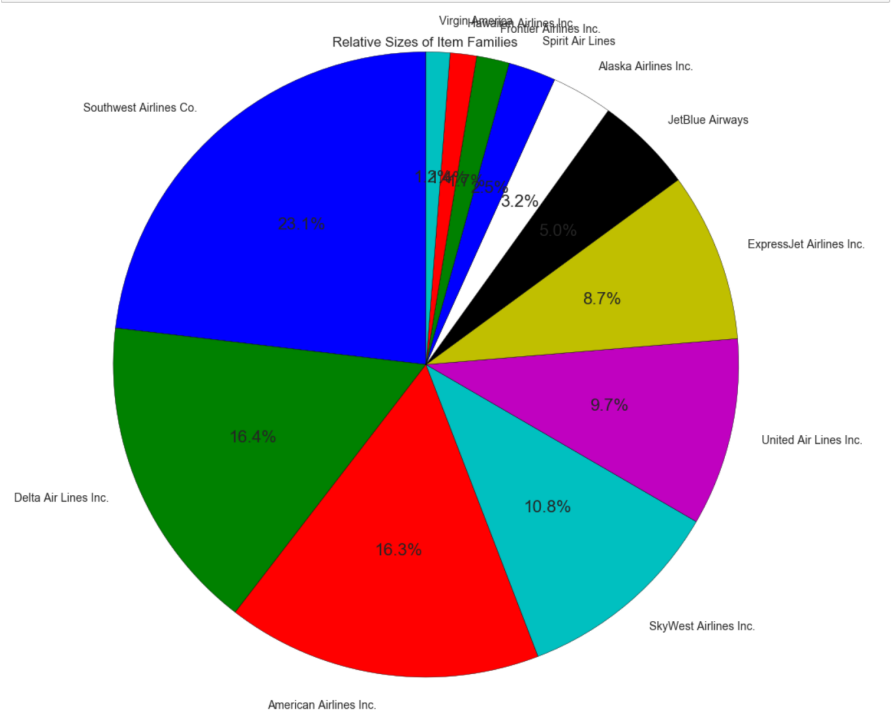
20. Departure Delay trend over hours of the day



21. Departure delay trend over the distance of the flight



22. Flight distribution over different airlines



IMPORTANT NOTES: Observation through the EDA

- **We get to know from the description of all the columns (column dictionary) provides on the website from where the data is scraped from that there are multiple columns have the same meaning.**
- **Also there are columns denoting the same information in different ways.**
  - **'Carrier' and 'UniqueCarrier' have the same meaning that is they denoted the various airlines. Carrier has numerical representation and UniqueCarrier has alphabetical. We should be considering only one of them.**

```
df.Carrier, df.UniqueCarrier
```

```
(0     AA
 1     AA
 2     AA
 3     AA
 4     AA
 5     AA
 6     AA
 7     AA
 8     AA
 9     AA
 10    AA
 11    AA
```

  - **Similarly for**
    - **Origin, OriginAirportID, OriginAirportSeqID**
    - **Dest, DestAirportID, DestAirportSeqID**
    - **OriginState, OriginStateName**
    - **OriginState, OriginStateName**
    - **DestState, DestStateName**
- **There are many columns that are not relevant for determining flight status or Deay minutes**

DATA WRANGLING:

We started data wrangling separately for 2016 and 2017 data. We have data for all 12 months for 2016 and data 2017 is unto September month.
FOR 2016 data:
Loaded the 'Data2016_Uncleaned.csv' into a data frame df.
Got rid of columns having missing values more than 98%

Loading the Csvs and combining into a single dataframe

```
cur_path = '/Users/sonalichaudhari/Desktop/ADS_Project/Data'
df = pd.read_csv(open(cur_path+'/Data2016_Uncleaned.csv','rU'), encoding='utf-8')
df.shape
```

```
(5617658, 109)
```

CLEANING

Getting rid of columns having 98% or more missing data

```
missing_data = df.isnull().sum().to_frame(name='Missing_Count').reset_index()
missing_data['Missing_percent'] = (missing_data['Missing_Count']/df.shape[0])*100
missing_data = missing_data.sort_values(by='Missing_percent', axis=0, ascending=True)
features = missing_data['index'][(missing_data.Missing_percent<99.38)]
len(features)
```

```
62
```

There are many columns that are not relevant for determining flight status or Delay minutes. As per the insights from EDA performed the following are columns we considered.

We change few columns name for easy understanding

```
# Renaming the columns
d = d.rename(columns = {'DayofMonth':'Day','DayOfWeek':'Week'})
```

columns =
['FlightDate','FlightNum','Year','Day','Month','Week','Origin','Dest','UniqueCarrier','Cancelled','ArrTime','ArrTimeBlk','DepTime','DepTimeBlk','CRSArrTime','CRSDepTime','DepDel15','ArrDel15','CRSElapsedTime','ActualElapsedTime','WheelsOn','WheelsOff','ArrDelayMinutes','DepDelayMinutes','OriginCityName','DestCityName','Distance']

We end up having 27 columns.

```
dt = d[columns]
dt.shape
```

```
(5617658, 27)
```

```
dt = dt[dt.Cancelled==0]
```

```
dt = dt[(dt.ArrTime.notnull()) &(dt.ArrDelayMinutes.notnull())]
```

```
dt.isnull().sum().sort_values()
```

There are lot of values missing in rows for Cancelled flights. Since we are dealing with Delayed flights only we get rid of all rows of Cancelled flights.
Also there are few row where ArrTime and ArrDealyMinutes is missing, there is no way we can fill the actual arrival time for the rows.

```
dt.isnull().sum().sort_values()
```

```
FlightDate                0
OriginCityName            0
DepDelayMinutes           0
ArrDelayMinutes           0
WheelsOff                 0
WheelsOn                  0
ActualElapsedTime         0
CRSElapsedTime            0
ArrDel15                  0
DepDel15                  0
DestCityName              0
CRSArrTime                0
DepTimeBlk                0
CRSDepTime                0
ArrTimeBlk                0
FlightNum                 0
Year                      0
Day                       0
DepTime                   0
Month                     0
Distance                  0
Origin                    0
Dest                      0
UniqueCarrier             0
Cancelled                 0
ArrTime                   0
Week                      0
CarrierDelay        4573906
WeatherDelay        4573906
NASDelay            4573906
SecurityDelay       4573906
LateAircraftDelay   4573906
dtype: int64
```

Adding a new column 'Flight_Status' to differentiate delayed and On-time flights.

Flight_Status =1 : Flight Delayed

```
dt['Flight_Status'] = 0
dt.loc[dt.DepDel15==1.0 ,'Flight_Status'] = 1
dt.loc[dt.ArrDel15==1.0,'Flight_Status'] = 1
dt['Flight_Status'] = dt['Flight_Status'].astype(int)
```

```
dt.shape
```

```
(5538145, 28)
```

Flight_Status =0 : On Time

Changed the data type of float to int because the values are integer for them.
From the EDA of delay minutes at arrival and departure airports distribution over months, we found lot of outliers. We get rid of outliers one by one for each month.

```
dd = dd[dd.DepDelayMinutes<1350]
```

```
dd = dd[dd.ArrDelayMinutes<1400]
dd.shape
```

```
(5538095, 28)
```

```
dd.Cancelled.unique()
```

```
array([0])
```

```
dd = dd[~((dd['Month'] == 1) & (dd['ArrDelayMinutes']>1320))]
dd = dd[~((dd['Month'] == 2) & (dd['ArrDelayMinutes']>1220))]
dd = dd[~((dd['Month'] == 3) & (dd['ArrDelayMinutes']>1220))]
dd = dd[~((dd['Month'] == 5) & (dd['ArrDelayMinutes']>1500))]
dd = dd[~((dd['Month'] == 6) & (dd['ArrDelayMinutes']>1200))]
dd = dd[~((dd['Month'] == 7) & (dd['ArrDelayMinutes']>1350))]
dd = dd[~((dd['Month'] == 9) & (dd['ArrDelayMinutes']>1120))]
dd = dd[~((dd['Month'] == 9) & (dd['ArrDelayMinutes']>1020))]
dd = dd[~((dd['Month'] == 10) & (dd['ArrDelayMinutes']>1270))]
dd = dd[~((dd['Month'] == 11) & (dd['ArrDelayMinutes']>1200))]
dd = dd[~((dd['Month'] == 12) & (dd['ArrDelayMinutes']>1350))]
```

```
dd.shape
```

```
(5538054, 28)
```

```
dd = dd[~((dd['Month'] == 2) & (dd['DepDelayMinutes']>1220))]
dd = dd[~((dd['Month'] == 4) & (dd['DepDelayMinutes']>1220))]
dd = dd[~((dd['Month'] == 5) & (dd['DepDelayMinutes']>1200))]
dd = dd[~((dd['Month'] == 10) & (dd['DepDelayMinutes']>1210))]
dd = dd[~((dd['Month'] == 11) & (dd['DepDelayMinutes']>1100))]
dd = dd[~((dd['Month'] == 12) & (dd['DepDelayMinutes']>1350))]
```

```
dd.shape
```

```
(5538041, 28)
```

The data wrangling is for 2017 data is the same as the 2016 data except the outliers removal.

```
dd1 = dt1
dd1.shape
```

```
(4193597, 28)
```

```
dd1 = dd1[~((dd1['Month'] == 1) & (dd1['ArrDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 3) & (dd1['ArrDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 9) & (dd1['ArrDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 5) & (dd1['ArrDelayMinutes']>1530))]
dd1 = dd1[~((dd1['Month'] == 6) & (dd1['ArrDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 7) & (dd1['ArrDelayMinutes']>1550))]
dd1 = dd1[~((dd1['Month'] == 9) & (dd1['ArrDelayMinutes']>1300))]
```

```
dd1 = dd1[~((dd1['Month'] == 1) & (dd1['DepDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 3) & (dd1['DepDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 4) & (dd1['DepDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 5) & (dd1['DepDelayMinutes']>1520))]
dd1 = dd1[~((dd1['Month'] == 6) & (dd1['DepDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 7) & (dd1['DepDelayMinutes']>1500))]
dd1 = dd1[~((dd1['Month'] == 9) & (dd1['DepDelayMinutes']>1300))]
```

```
dd1.shape
```

```
(4193575, 28)
```

Concatenating the the two data frame to one common data frame.

```
dd.shape
```

```
(5538041, 28)
```

click to scroll output; double click to hide

```
data = pd.concat([dd,dd1])
```

```
data.shape
```

```
(9731616, 28)
```

Combining with the weather data that we scraped.

```
# airports taken into consideration
airport = ['DFW', 'DTW', 'SEA', 'JFK', 'SJC', 'ORD', 'PHX', 'STL', 'LAX',
       'MCO', 'DEN', 'MIA', 'IAH','SLC', 'TUS','BOS', 'FLL', 'SFO', 'OGG', 'TPA', 'SNA', 'OKC', 'HNL',
       'PHL', 'LGA', 'RDU', 'DCA', 'RIC', 'ATL', 'LBB', 'CLT', 'ELP',
       'BNA', 'JAC', 'SMF', 'EWR', 'IAD', 'LIH', 'SJU', 'ABQ',
       'ORF', 'JAX', 'MSY','MCI', 'GUC', 'IND', 'PDX', 'BWI',
       'MSP', 'MKE', 'ONT', 'RSW', 'DSM', 'PSP',
       'EGE', 'PBI', 'SDF', 'PIT', 'DAY', 'STX', 'COS',
       'CMH', 'MTJ', 'HDN', 'BDL', 'MEM', 'CLE', 'HOU','OAK',
       'GEG', 'ANC', 'BUF', 'SYR', 'ALB', 'PVD','ILM', 'ICT',
       'PWM', 'GSO', 'CHS', 'MDT', 'BHM', 'ADQ', 'BRW', 'SCC',
       'JNU', 'KTN', 'YAK', 'CDV','PSG', 'WRG', 'OME',
       'OTZ', 'BUR', 'ADK', 'SWF', 'LGB', 'PSE', 'BQN', 'HPN',
       'SAV', 'SRQ', 'BTV', 'ORH','CVG', 'BIS', 'AVL', 'GRR',
       'FNT', 'MYR', 'JAN', 'FAR', 'PNS', 'AGS','LEX',
       'DAL', 'ATW', 'GPT', 'MLB', 'BZN', 'CHO', 'MSN', 'EYW',
       'TRI', 'LFT', 'ROA', 'ECP', 'VPS', 'XNA', 'EVV', 'AVP', 'MDW',
       'HSV', 'FAY', 'TYS', 'TLH', 'MSO','TTN',
       'PHF', 'FSD', 'LBE','BMI', 'CRW','PPG', 'IAG', 'ACT','SHV', 'FSM', 'MAF',
       'SAF', 'JLN', 'LRD', 'BRO', 'TYR', 'GJT', 'DLH',
       'SBA', 'ASE', 'IDA', 'RAP', 'FCA', 'LNK', 'AMA',
       'BFL', 'MLI', 'LSE', 'SBN', 'PSC','FLG', 'ISN', 'GFK',
       'GTF', 'MRY', 'MBS', 'SUN', 'TWF', 'SGF', 'CPR',
       'BTR', 'PBG', 'CRP', 'CID', 'SBP', 'RKS', 'CMX', 'MMH', 'PLN',
       'EKO', 'GCC', 'MFR', 'SMX', 'EUG', 'RST', 'TVC', 'SPI',
       'SGU', 'HLN', 'RDM', 'ACV', 'EAU', 'DVL', 'JMS', 'MKG', 'HYS',
       'COD', 'ITH', 'APN', 'ESC', 'BJI', 'MQT',
       'BGM', 'RHI', 'LWS', 'IMT', 'BRD', 'INL', 'PIH', 'GUM', 'HIB',
       'BTM', 'CDC', 'OTH', 'RDD', 'HRL', 'ISP', 'MHT', 'GNV',
       'MEI', 'PIB', 'BPT',  'AEX', 'TXK', 'ROW', 'CLL',
       'HOB', 'LCH', 'OAJ', 'ELM', 'VLD', 'MGM', 'BGR', 'GTR',
       'CSG', 'BQK', 'DHN', 'EWN', 'ABY', 'SPS', 'SJT', 'GGG',
       'ACK', 'MVY', 'HYA', 'GST', 'AKN', 'DLG', 'GCK', 'MHK',
       'ABI', 'GRI', 'EFD', 'PGD', 'SPN']
```

Out of 313 unique airports in our data we got weather data for 263. We will be dealing with the common 263 airports data. Merging the weather based on the Station ID, Date.

```
dt_both = data[data.Origin.isin(airport)]
dt_both = dt_both[dt_both.Dest.isin(airport)]
```

```
df_weather2016 = pd.read_csv(cur_path+'/weatherdata2016/weatherdata2016.csv', low_memory=False)
df_weather2017 = pd.read_csv(cur_path+'/weatherdata2017/weatherdata2017.csv', low_memory=False)
```

```
df_weather2016=df_weather2016.rename(columns = {'year':'Year','month':'Month','day':'Day'})
df_weather2017=df_weather2017.rename(columns = {'year':'Year','month':'Month','day':'Day'})
df_weather = pd.concat([df_weather2016,df_weather2017])
```

```
df_weather_origin = df_weather[['station', 'Year', 'Month', 'Day', 'Visibility', 'Wind', 'Precip']]
df_weather_origin.columns = ['Origin', 'Year', 'Month', 'Day', 'OriginVisibility', 'OriginWind', 'OriginPrecip']
```

```
result = dt_both.merge(df_weather_origin.drop_duplicates(subset=['Origin', 'Year','Month', 'Day']), how='left',on
```

```
result.isnull().sum().sort_values()
```

```
final_combined =result
final_combined.OriginVisibility = final_combined.OriginVisibility.fillna((final_combined.OriginVisibility.fillna(
final_combined.OriginWind = final_combined.OriginWind.fillna((final_combined.OriginWind.fillna(method='ffill') +f
final_combined.OriginPrecip = final_combined.OriginPrecip.fillna((final_combined.OriginPrecip.fillna(method='ffil
```

```
df_dec = final_combined.loc[(final_combined['Month']==9)]
```

```
mode_vis = df_dec.OriginVisibility.mode()[0]
mode_wind = df_dec.OriginWind.mode()[0]
mode_pre = df_dec.OriginPrecip.mode()[0]
```

```
final_combined.OriginVisibility = final_combined.OriginVisibility.fillna(mode_vis)
final_combined.OriginWind = final_combined.OriginWind.fillna(mode_wind)
final_combined.OriginPrecip = final_combined.OriginPrecip.fillna(mode_pre)
```

```
final_combined.isnull().sum().sort_values()
```

Finally we a dataset of (8058349 rows, 31columns)

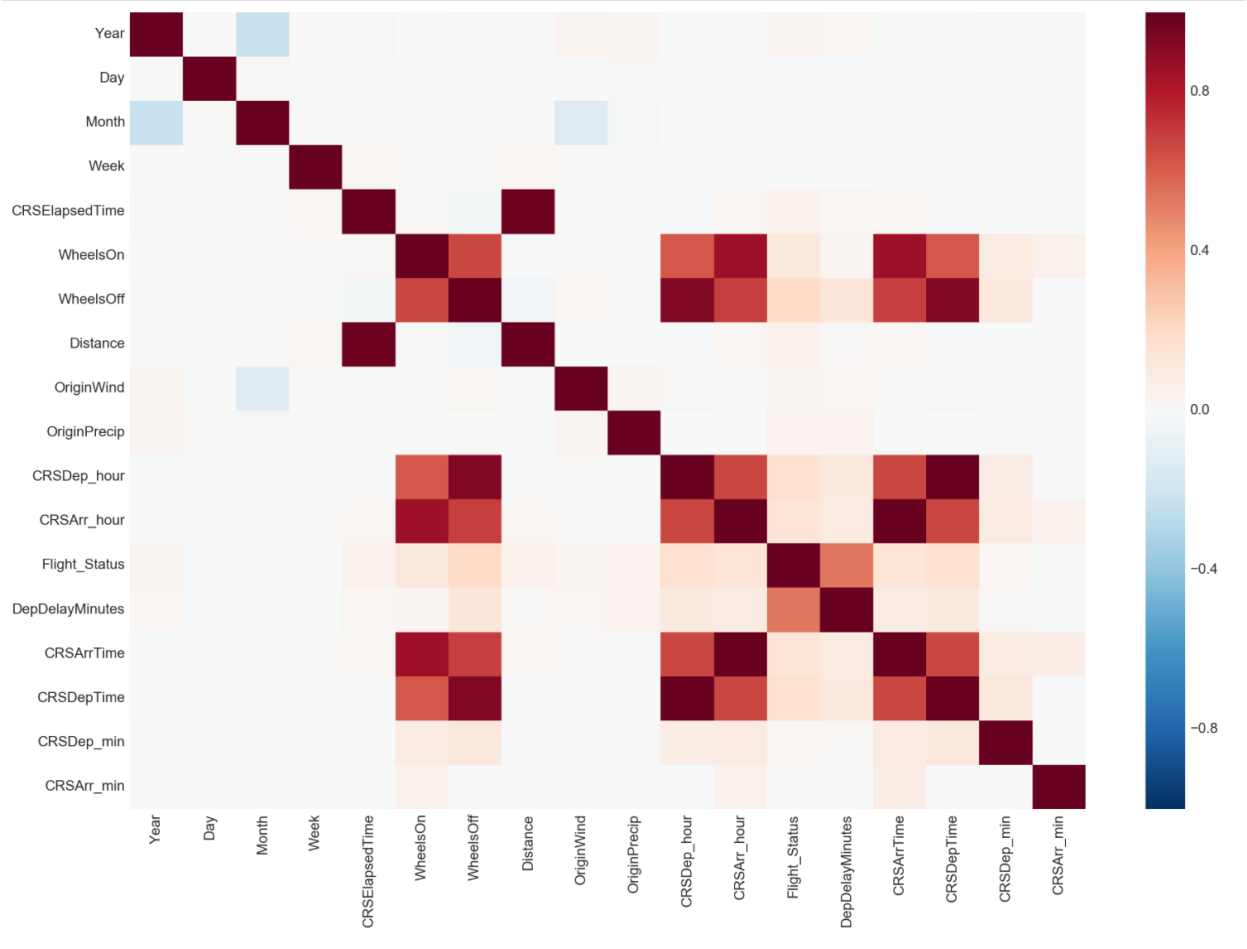
FEATURE ENGINEERING:

We change the time to hour format.

```
df['CRSDep_hour'] = df['CRSDepTime'].apply(lambda x: str(x)[:-2])
df['CRSDep_min'] = df['CRSDepTime'].apply(lambda x: str(x)[1:3] if len(str(x))==3 else  str(x)[2:4])
df = df.replace('', 0)
df['CRSArr_hour'] = df['CRSArrTime'].apply(lambda x: str(x)[:-2])
df['CRSArr_min'] = df['CRSArrTime'].apply(lambda x: str(x)[1:3] if len(str(x))==3 else  str(x)[2:4])
df = df.replace('', 0)
```

```
df['Scheduled_Dep_T'] = df.CRSDep_hour.astype(str).str.cat(df.CRSDep_min.astype(str), sep=':')
df['Scheduled_Arr_T'] = df.CRSArr_hour.astype(str).str.cat(df.CRSArr_min.astype(str), sep=':')
```

First we plot heat map to see the correlation between various columns.

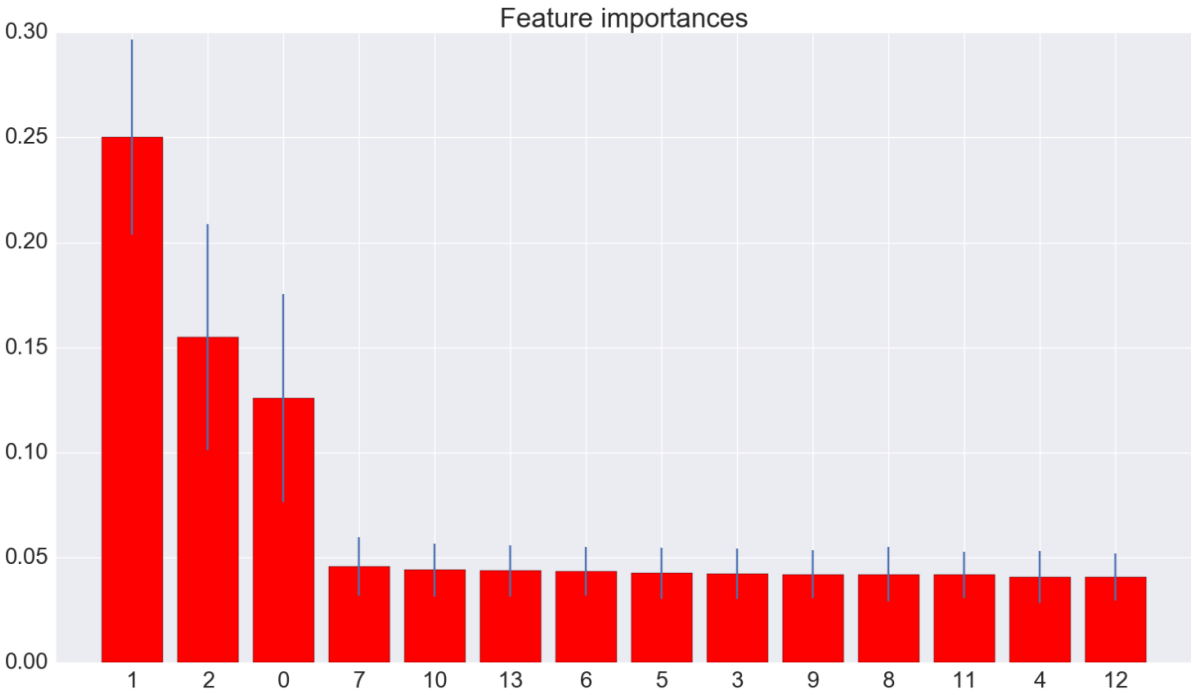


After that we consider only the columns that can filled from passenger data entry or computed in some way with input data from passenger.

For feature importance.we used ExtraTreesClassifier

```
import matplotlib.pyplot as plt

from sklearn.datasets import make_classification
from sklearn.ensemble import ExtraTreesClassifier

# Build a classification task using 3 informative features
X, y = make_classification(n_samples=1000,
                           n_features=14,
                           n_informative=3,
                           n_redundant=0,
                           n_repeated=0,
                           n_classes=2,
                           random_state=0,
                           shuffle=False)

# Build a forest and compute the feature importances
forest = ExtraTreesClassifier(n_estimators=250,
                              random_state=0)

forest.fit(X, y)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f],importances[indices[f]]))

# Plot the feature importances of the forest

plt.figure(figsize=(18, 10))
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. feature 1 (0.250092)
2. feature 2 (0.154778)
3. feature 0 (0.125871)
4. feature 7 (0.045717)
5. feature 10 (0.044092)
6. feature 13 (0.043659)
7. feature 6 (0.043440)
8. feature 5 (0.042660)
9. feature 3 (0.042191)
10. feature 9 (0.042055)
11. feature 8 (0.042045)
12. feature 11 (0.041782)
13. feature 4 (0.040920)
14. feature 12 (0.040699)
```



Columns we considered for classification. We generate a dataformodel.csv having required columns for data

CLASSIFICATION

```
colls = ['Year', 'Day', 'Month', 'Week', 'Origin','Dest', 'UniqueCarrier','CRSElapsedTime',
         'OriginWind', 'OriginPrecip','CRSDep_hour','CRSArr_hour','Flight_Status',
         'DepDelayMinutes','CRSArrTime','CRSDepTime']
len(colls)

colls.remove('Flight_Status')
colls.remove('CRSArrTime')
colls.remove('CRSDepTime')
len(colls)
```

```
13
```

```
X= df[colls]
y = df['Flight_Status']
```

Train and test data split and converting categorical data into numeric.

```
from sklearn import preprocessing, cross_validation

y = np.ravel(y)
# Splitting the data
X_train, X_test, y_train, y_test =  cross_validation.train_test_split(X, y,test_size=0.30,random_state=0)
```

```
/Users/sonalichaudhari/anaconda/lib/python3.5/site-packages/sklearn/cross_validation.py:41: DeprecationWarning:
This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored
classes and functions are moved. Also note that the interface of the new CV iterators are different from that o
f this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

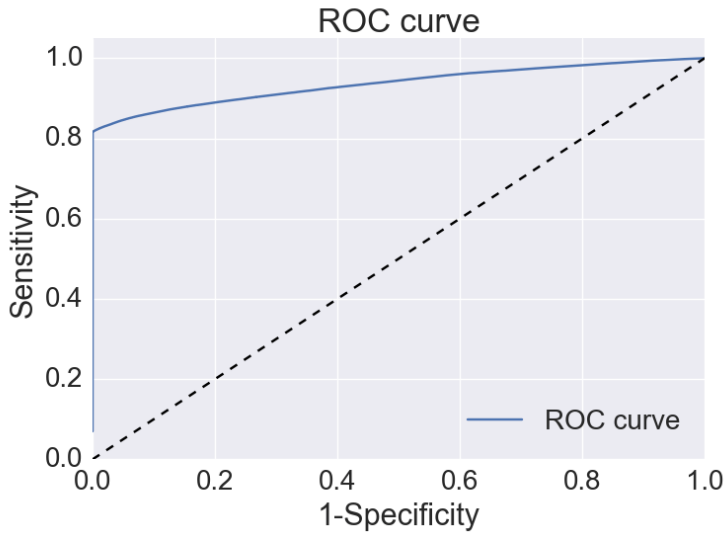
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
# Converting the categorical data to numeric

X_train['carrier'] = label_encoder.fit_transform(X_train['UniqueCarrier'])
X_train['origin'] = label_encoder.fit_transform(X_train['Origin'])
X_train['dest'] = label_encoder.fit_transform(X_train['Dest'])
# X_train['ori_city'] = label_encoder.fit_transform(X_train['OriginCityName'])
# X_train['dest_city'] = label_encoder.fit_transform(X_train['DestCityName'])

X_test['carrier'] = label_encoder.fit_transform(X_test['UniqueCarrier'])
X_test['origin'] = label_encoder.fit_transform(X_test['Origin'])
X_test['dest'] = label_encoder.fit_transform(X_test['Dest'])
# X_test['ori_city'] = label_encoder.fit_transform(X_test['OriginCityName'])
# X_test['dest_city'] = label_encoder.fit_transform(X_test['DestCityName'])

# Getting only the numerical columns

X_train = X_train._get_numeric_data()
X_test = X_test._get_numeric_data()
```

We implemented Neural Networks and Random Forest.

Random forest classifier worked best for classification
We tried various version of neural network trying out different layers and different values of estimators for Random forest.

In [40]:
```
# Creating the model
from sklearn.ensemble import RandomForestClassifier

rt = RandomForestClassifier(max_depth =7,random_state=0)
model1 = rt.fit(X_train, y_train)

accuracy = rt.score(X_train, y_train)
print('Accuracy: ',accuracy)

#Run the model on the test set
y_pred = rt.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve

preds = rt.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, preds)
plt.figure()
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```

```
Accuracy:  0.95902687612
[[1880009       0]
 [  98940  438556]]
```



PREDICTION:

For prediction we filtered the final data having only the flight delayed for departure.

```
pred = df[df.DepDel15==1]
```

click to expand output; double click to hide output

```
(1162752, 42)
```

```
pred = pred[pred.Flight_Status==1]
```

```
pred.columns
```

```
Index(['FlightDate', 'FlightNum', 'Year', 'Day', 'Month', 'Week', 'Origin',
       'Dest', 'UniqueCarrier', 'Cancelled', 'ArrTime', 'ArrTimeBlk',
       'DepTime', 'DepTimeBlk', 'CRSArrTime', 'CRSDepTime', 'DepDel15',
       'ArrDel15', 'CRSElapsedTime', 'ActualElapsedTime', 'WheelsOn',
       'WheelsOff', 'LateAircraftDelay', 'CarrierDelay', 'WeatherDelay',
       'NASDelay', 'SecurityDelay', 'ArrDelayMinutes', 'DepDelayMinutes',
       'OriginCityName', 'DestCityName', 'Distance', 'Flight_Status',
       'OriginVisibility', 'OriginWind', 'OriginPrecip', 'CRSDep_hour',
       'CRSDep_min', 'CRSArr_hour', 'CRSArr_min', 'Scheduled_Dep_T',
       'Scheduled_Arr_T'],
      dtype='object')
```

```
pred.shape
```

```
(1162752, 42)
```

Cross validation for splitting into train and test

```
from sklearn import preprocessing, cross_validation

y = np.ravel(y)
# Splitting the data
X_train, X_test, y_train, y_test =  cross_validation.train_test_split(X, y,test_size=0.30,random_state=0)
```

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
# Converting the categorical data to numeric

X_train['carrier'] = label_encoder.fit_transform(X_train['UniqueCarrier'])
X_train['origin'] = label_encoder.fit_transform(X_train['Origin'])
X_train['dest'] = label_encoder.fit_transform(X_train['Dest'])
# X_train['ori_city'] = label_encoder.fit_transform(X_train['OriginCityName'])
# X_train['dest_city'] = label_encoder.fit_transform(X_train['DestCityName'])

X_test['carrier'] = label_encoder.fit_transform(X_test['UniqueCarrier'])
X_test['origin'] = label_encoder.fit_transform(X_test['Origin'])
X_test['dest'] = label_encoder.fit_transform(X_test['Dest'])
# X_test['ori_city'] = label_encoder.fit_transform(X_test['OriginCityName'])
# X_test['dest_city'] = label_encoder.fit_transform(X_test['DestCityName'])
```

```
# Getting only the numerical columns

X_train = X_train._get_numeric_data()
X_test = X_test._get_numeric_data()
```

Random forest regressor worked best for prediction.
We tried various version of neural network trying out different layers and different values of estimators for Random forest.

```
from sklearn.ensemble import RandomForestRegressor

rtf3 = RandomForestRegressor(n_estimators=25)
rtf3.fit(X_train, y_train)

accuracy = rtf3.score(X_train, y_train)
print('Accuracy: ',accuracy)

# Run the model on the test set and train set
y_test_predicted = rtf3.predict(X_test)
y_train_predicted= rtf3.predict(X_train)

# RMS MAPE MAE
from sklearn.metrics import mean_squared_error,mean_absolute_error
from math import sqrt

rms = sqrt(mean_squared_error(y_train, y_train_predicted))
mae_train = mean_absolute_error(y_train, y_train_predicted)
mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
print("-----Train-----")
print('RMS: ',rms)
print('MAE: ',mae_train)
print('MAPE: ',mape_train)
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, a
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_test, y_test_predicted))
mae_test = mean_absolute_error(y_test, y_test_predicted)
mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100
print("-----Test-----")
print('RMS: ',rms)
print('MAE: ',mae_test)
print('MAPE: ',mape_test)
```

```
Accuracy:  0.845615044423
-----Train-----
RMS:  32.532554955455744
MAE:  18.942314865
MAPE:  37.9805380438
-----Test-----
RMS:  82.03817173269022
MAE:  49.4337994008
MAPE:  100.502043926
```