In [129…
```
!pip install -q pandasql
!pip install -q plotly
```

GOAL: To recommend 5 round trip routes between medium and large US airports for an airline company looking to enter US domestic market
CLIENT's Moto: "On time for you" **Dataset for analysis:** 2019 Q1 flights data

In [130…
```
import pandas as pd
import numpy as np
import seaborn as sns
from statistics import mean
from pandasql import sqldf
import matplotlib.pyplot as plt
import statsmodels.api as sm

%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import plotly.io as pio
pio.renderers.default='notebook'
pd.set_option('display.float_format', lambda x: '%.2f' % x)
```

In [131…
```
airport_raw_df  = pd.read_csv('data/airport_codes.csv')
flights_raw_df  = pd.read_csv('data/flights.csv')
tickets_raw_df  = pd.read_csv('data/tickets.csv')
```

Generic functions for data quality

In [132…
```
''' Function : To check if nulls are in the aggregates post grouping for a passed dataframe
    Params   : Dataframe, Columns to be grouped, Aggregate column
    Returns  : Dataframe with nulls '''

def find_nulls_post_grouping(df, groupCols, resultCol):
    result_df = df.groupby(groupCols, as_index=False)[resultCol].mean()
    return (result_df[result_df[resultCol].isna()].reset_index(drop=True) if result_df[resultCol].isna().sum() else 'No nulls in '
```

In [133…
```
''' Function : To calculate feature-wise null percentage for a passed dataframe for all/ selected features
    Params   : Input Dataframe, Columns - optional
    Returns  : Dataframe with feature wise missing percent for all features/ selected features '''

def find_missing_percent(df, cols=[]):
    input_df = df[cols] if cols else df
    missing_airport_df = input_df.isnull().sum(axis=0).reset_index()
    missing_airport_df.columns = ['VARIABLE', 'MISSING']
    missing_airport_df['PERCENT_MISSING']=(missing_airport_df['MISSING'])/input_df.shape[0]*100
    missing_airport_df.sort_values('PERCENT_MISSING', ascending=False, inplace=True, ignore_index=True)
    print("Total records: ", df.shape[0])
    return missing_airport_df[missing_airport_df.PERCENT_MISSING>0]
```

In [134…
```
''' Function : To check if value is numeric else returns non-numeric invalid values
    Params   : value
    Returns  : Returns non-numeric values whereas returns null if numeric '''

def check_numeric(value):
    try:
        pd.to_numeric(value)
        return np.NaN
    except ValueError:
        return value
```

In [135…
```
''' Function : To show box and dist plots for  passed dataframe
    Params   : Dataframe, Columns to be plotted
    Returns  : None '''

def show_distribution(data,col):
    fig,(ax1,ax2)=plt.subplots(2, 1)
    sns.distplot(data[col],ax=ax1)
    sns.boxplot(data[col],ax=ax2,color='orange')
```

In [136…
```
''' Function : To find outliers of passed dataframe column
    Params   : Dataframe, Columns to be plotted
    Returns  : None '''

def find_outliers(df, col):
    q3 = df[col].quantile(0.75)
    q1 = df[col].quantile(0.25)
    IQR = q3 - q1
    upper_bound = q3 + (1.5 * IQR)
    lower_bound = q1 - (1.5 * IQR)
    return upper_bound
```

In [137…
```
''' Function : Drop specified cols from passed dataframe
    Params   : Dataframe, Columns to be dropped
    Returns  : None '''

def drop_columns(df, cols):
```

```
    for col in cols:
        df.drop(col, axis = 1, inplace = True)
```

Data Quality Check

1. Airport File

In [138…

```
airport_raw_df.head()
```

Out[138…

	TYPE	NAME	ELEVATION_FT	CONTINENT	ISO_COUNTRY	MUNICIPALITY	IATA_CODE	COORDINATES
0	heliport	Total Rf Heliport	11.00	NaN	US	Bensalem	NaN	-74.93360137939453, 40.07080078125
1	small_airport	Aero B Ranch Airport	3435.00	NaN	US	Leoti	NaN	-101.473911, 38.704022
2	small_airport	Lowell Field	450.00	NaN	US	Anchor Point	NaN	-151.695999146, 59.94919968
3	small_airport	Epps Airpark	820.00	NaN	US	Harvest	NaN	-86.77030181884766, 34.86479949951172
4	closed	Newport Hospital & Clinic Heliport	237.00	NaN	US	Newport	NaN	-91.254898, 35.6087

In [139…

```
airport_raw_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55369 entries, 0 to 55368
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   TYPE          55369 non-null  object 
 1   NAME          55369 non-null  object 
 2   ELEVATION_FT  48354 non-null  float64
 3   CONTINENT     27526 non-null  object 
 4   ISO_COUNTRY   55122 non-null  object 
 5   MUNICIPALITY  49663 non-null  object 
 6   IATA_CODE     9182 non-null   object 
 7   COORDINATES   55369 non-null  object 
dtypes: float64(1), object(7)
memory usage: 3.4+ MB
```

In [140…

```
# Percentage missing values in airport_df
find_missing_percent(airport_raw_df)
```

```
Total records:  55369
```

Out[140…

	VARIABLE	MISSING	PERCENT_MISSING
0	IATA_CODE	46187	83.42
1	CONTINENT	27843	50.29
2	ELEVATION_FT	7015	12.67
3	MUNICIPALITY	5706	10.31
4	ISO_COUNTRY	247	0.45

In [141…

```
# As we are interested in analyzing US airports. Let's explore ISO_COUNTRY field first
# All 247 missing entries in Country variable belongs to the continent 'AF' ie Africa and not 'North America'
# Such entries will be filtered out in next stepsince only US airports are of interest. So we are good to go
airport_raw_df[airport_raw_df['ISO_COUNTRY'].isna()]['CONTINENT'].value_counts()
```

Out[141…

```
AF    247
Name: CONTINENT, dtype: int64
```

In [142…

```
# Checking if there is any mismatch in the continent names for US country airports (NA for NorthAmerica/ nan are accepted)
airport_raw_df.loc[(airport_raw_df['ISO_COUNTRY']=='US') & (airport_raw_df['TYPE'].isin(['medium_airport','large_airport'])) ]['CO
```

Out[142…

```
Series([], Name: CONTINENT, dtype: int64)
```

In [143…

```
# Since we can extract all US airports by country column as US and also as all the 247 nulls in that col belong to
# non-NorthAmerica continent. We dont have to worry abt continent column
```

In [144…

```
# There aren't any nulls in Type column. So lets narrow down by type and country for our analysis
airport_raw_df[airport_raw_df.ISO_COUNTRY=='US']['TYPE'].value_counts(dropna=False)
```

Out[144…

```
small_airport     13708
heliport           6268
closed             1392
medium_airport      687
seaplane_base       566
large_airport       171
balloonport          18
Name: TYPE, dtype: int64
```

In [145…

```
# Narrowing down the search as we are interested in analysis of medium and large US domestic airports
airport_df = airport_raw_df.loc[(airport_raw_df['ISO_COUNTRY']=='US') & (airport_raw_df['TYPE'].isin(['medium_airport','large_airp
```

In [146…

```
# 858 entries are US medium/ large airports
airport_df.shape
```

Out[146… (858, 8)

In [147…
```
# Sorting the iata codes such that nulls appear in the end
# So taht when dropping the duplicates we can retain the first match and get rid of the rest
airport_df = airport_df.sort_values(by='IATA_CODE', ascending=True, na_position='last').reset_index(drop=True)
```

In [148…
```
# Duplicates
airport_df[airport_df.duplicated(subset = ['TYPE', 'NAME', 'MUNICIPALITY'])]
```

Out[148…

	TYPE	NAME	ELEVATION_FT	CONTINENT	ISO_COUNTRY	MUNICIPALITY	IATA_CODE	COORDINATES
839	medium_airport	Columbus Municipal Airport	1447.00	NaN	US	Columbus	NaN	-97.34259796, 41.44800186

In [149…
```
airport_df.drop_duplicates( subset=['TYPE', 'NAME', 'MUNICIPALITY'], keep='first', inplace=True )
```

In [150…
```
# Missing % for filtered data of US domestic medium and large airports
find_missing_percent(airport_df)
```

```
Total records:  857
```

Out[150…

	VARIABLE	MISSING	PERCENT_MISSING
0	CONTINENT	857	100.00
1	IATA_CODE	36	4.20
2	ELEVATION_FT	3	0.35
3	MUNICIPALITY	3	0.35

In [151…
```
# Here the IATA code column is only of interest which has 36 nulls. Lets see if can fill the nulls based on municipality.
# We have municipalities in airports file and cities in flights file but same city/municipality can have multiple airports
# Eg. Newyork can have 2 airports - JFK and LGA. Since we are limited not to use extra datasets, we will drop the nulls in iata
```

In [152…
```
airport_df = airport_df.dropna(subset=['IATA_CODE'])
```

In [153…
```
airport_df.shape[0]
```

Out[153… 821

2. Flights File

In [154…
```
flights_raw_df.head()
```

Out[154…

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	DEST_CI
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	Clev
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	Colu
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	Colu
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	Colu
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	

In [155…
```
# Excluding cancelled flights from our analysis
flights_df = flights_raw_df[flights_raw_df['CANCELLED']==0.0]
```

In [156…
```
# Flights file has only got 2019-1Q data and in entirety can be used for our analysis
print('Flights data is from date: ' + flights_raw_df['FL_DATE'].min() + ' and ' + flights_raw_df['FL_DATE'].max())
```

```
Flights data is from date: 1/1/19 and 3/9/19
```

In [157…
```
# Check for invalid non-numeric values by calling custom function in the following cols which is required for aggregation
cols = ['DISTANCE','AIR_TIME','ARR_DELAY','DEP_DELAY','OCCUPANCY_RATE']

for col in cols:
    print(flights_df[col].apply(check_numeric).dropna().reset_index(drop=True).value_counts())
```

```
****       2030
NAN          20
Hundred      10
Twenty       10
Name: DISTANCE, dtype: int64
$$$     1810
Two       10
NAN       10
Name: AIR_TIME, dtype: int64
Series([], Name: ARR_DELAY, dtype: int64)
Series([], Name: DEP_DELAY, dtype: int64)
Series([], Name: OCCUPANCY_RATE, dtype: int64)
```

In [158…
```
# Converting reqd cols to numeric which will also get rid of above invalid values
```

```
# Distance has invalid values

cols = ['DISTANCE','AIR_TIME','ARR_DELAY','DEP_DELAY','OCCUPANCY_RATE']
flights_df[cols] = flights_df[cols].apply(pd.to_numeric, errors='coerce')
```

In [159…

```
flights_df.describe()
```

Out[159…

	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	DEP_DELAY	ARR_DELAY	CANCELLED	AIR_TIME	DISTANCE	OCCUPANCY_RATE
count	1864272.00	1864272.00	1864272.00	1859895.00	1864272.00	1857415.00	1861592.00	1863962.00
mean	12685.80	12687.10	10.77	5.65	0.00	109.39	772.38	0.65
std	1522.33	1521.97	50.07	52.41	0.00	70.47	582.68	0.20
min	10135.00	10135.00	-63.00	-94.00	0.00	-121.00	-1947.00	0.30
25%	11292.00	11292.00	-6.00	-15.00	0.00	59.00	344.00	0.48
50%	12889.00	12889.00	-2.00	-6.00	0.00	91.00	612.00	0.65
75%	14057.00	14057.00	7.00	8.00	0.00	139.00	1013.00	0.83
max	16218.00	16218.00	2941.00	2923.00	0.00	2222.00	9898.00	1.00

In [160…

```
# From below, overall there are < 1% nulls in each of the above four variables which is very minimal
```

In [161…

```
find_missing_percent(flights_df)
```

```
Total records:  1864272
```

Out[161…

	VARIABLE	MISSING	PERCENT_MISSING
0	AIR_TIME	6857	0.37
1	ARR_DELAY	4377	0.23
2	DISTANCE	2680	0.14
3	OCCUPANCY_RATE	310	0.02

Handling missing values, outliers, negatives and skewness for below quantitative columns

- DISTANCE
- AIR_TIME
- ARR_DELAY
- DEP_DELAY
- OCCUPANCY_RATE

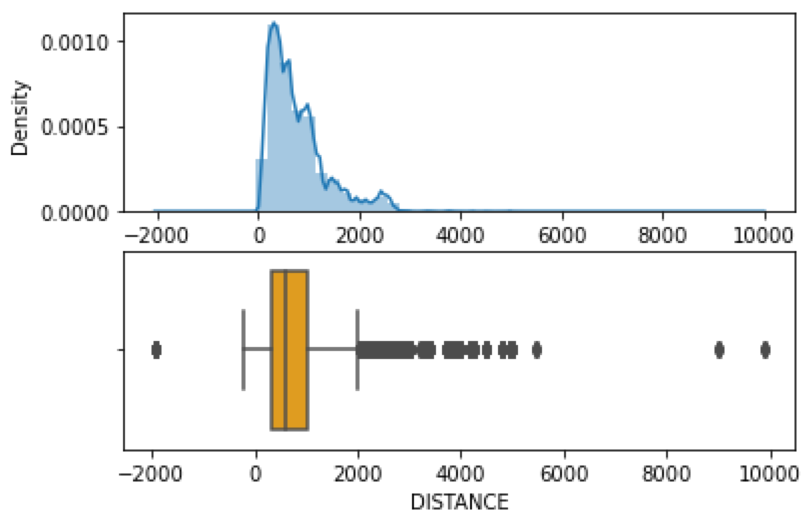
a) DISTANCE variable

In [162…

```
# From below, Distance here spans from negative values -2000 until 10000 which is unusual and 2680 are nulls
# Since we have 1.8 Milllion records, we shall group by origin-dest to identify mode and perform
# mode level imputation for distance which might fix above issues as well as skewness
```

In [163…

```
show_distribution(flights_df,'DISTANCE')
```



In [164…

```
# Lets chk this for example pre and post imputation
# Here we see different distance for the same route 'RSW-STL' whihc should ideally be the most frequent ones ie 979
flights_df[(flights_df['ORIGIN'] == 'RSW') & (flights_df['DESTINATION'] == 'STL')]['DISTANCE'].value_counts()
```

Out[164…

```
979.00     238
9000.00     10
Name: DISTANCE, dtype: int64
```

In [165…

```
# Mode level distance imputation
route_groups  = flights_df.groupby(['ORIGIN','DESTINATION'], as_index=False)
mode_by_group = route_groups['DISTANCE'].agg(lambda x: x.mode()[0])[['ORIGIN','DESTINATION','DISTANCE']]
flights_df = flights_df.merge(mode_by_group, how='left', on=['ORIGIN','DESTINATION'])[['FL_DATE','OP_CARRIER','TAIL_NUM','OP_CARRI
```

In [166…

```
# This is fixed post-imputation
flights_df[(flights_df['ORIGIN'] == 'RSW') & (flights_df['DESTINATION'] == 'STL')]['DISTANCE'].value_counts()
```

Out[166…
```
979.00    248
Name: DISTANCE, dtype: int64
```

In [167…
```
# Nulls are also handled with mode level imputation
flights_df['DISTANCE'].isna().sum()
```

Out[167…
```
0
```

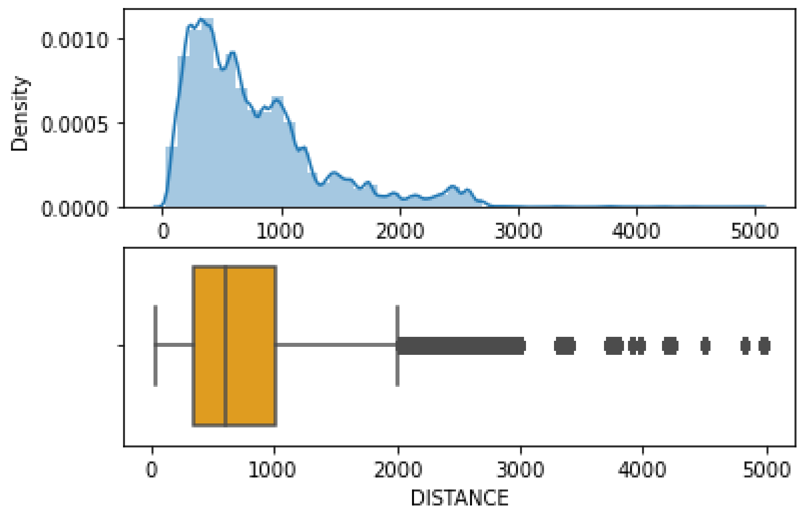
As we know, Boston to Honolulu is the longest domestic flight in US - 11 hours - 5095 miles. There is just one origin-dest pair ie, MIA-ABQ which is said to have a distance of 10000 miles way beyond 6000 miles. MIA-ABQ is actually ~2000 miles and here its 5X of its original value which is clearly an outlier.

In [168…
```
# As we know, Boston to Honolulu is the longest domestic flight in US - 11 hours - 5095 miles
# There is just one origin-dest pair ie, MIA-ABQ which is said to have a distance of
# 10000 miles way beyond 6000 miles. MIA-ABQ is actually ~2000 miles and here its 5X of its original value
# which is clearly an outlier.
flights_df[flights_df['DISTANCE'] > 6000][['ORIGIN','DESTINATION','DISTANCE']].value_counts()
```

Out[168…
```
ORIGIN  DESTINATION  DISTANCE
MIA     ABQ          9898.00     10
dtype: int64
```

In [169…
```
# Hence removing the above record
flights_df = flights_df.loc[~(flights_df['DISTANCE'] > 6000)]
```

In [170…
```
# POST-IMPUTATION for DISTANCE (Invalids, nulls are all handled and the distribution has become less skewed
show_distribution(flights_df, 'DISTANCE')
```



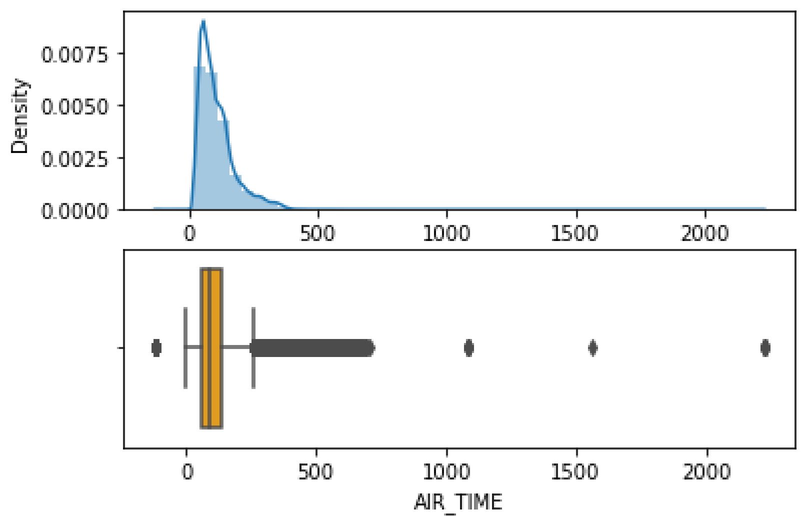
b) AIR_TIME variable

In [171…
```
# When both the air time AND arrival delay fields are null, its equivalent to flight getting cancelled
(flights_df.loc[((flights_df['AIR_TIME'].isna()) & (flights_df['ARR_DELAY'].isna()))].shape[0])/(flights_df.shape[0])*100
```

Out[171…
```
0.23478459572742458
```

In [172…
```
# 4377 records of ~1.8Million are probably cancelled flights marked instead as not cancelled

# So dropping probably cancelled flights from our analysis
flights_df_cleaned = flights_df.loc[~((flights_df['AIR_TIME'].isna()) & (flights_df['ARR_DELAY'].isna()))]
```

In [173…
```
show_distribution(flights_df_cleaned, 'AIR_TIME')
```



In [174…
```
flights_df_cleaned[flights_df_cleaned['AIR_TIME']<0].shape
```

Out[174…
```
(100, 12)
```

In [175…
```
pd.unique(flights_df_cleaned[flights_df_cleaned['AIR_TIME']<0][['ORIGIN', 'DESTINATION']].values.ravel('K'))
```

Out[175…
```
array(['JFK', 'ORD'], dtype=object)
```

In [176…
```
# Abnormal for airtime to be negative which is 0.005% of dataset and that too for 1 route, JFK-ORD and hence removing them.
# There are records with <10 min Air_time but its still feasible. So leaving them as such eg. Actual air_time for Alaska-Petersbur
flights_df_cleaned = flights_df_cleaned[~(flights_df_cleaned['AIR_TIME']<0)]
```

In [177…
```
flights_df_cleaned['AIR_TIME'].describe()
```

Out[177…
```
count    1857305.00
mean         109.40
std           70.45
min            1.00
25%           59.00
50%           91.00
75%          139.00
max         2222.00
Name: AIR_TIME, dtype: float64
```

In [178…
```
# Certain extremities have airtime of 18 hrs which is huge for US domestic flights
# But by arr_delay we can say that majority of flights arrived within more or less 5 mins of expected arrival,
# so these can be considered as expected air times
flights_df_cleaned[ flights_df_cleaned['AIR_TIME']>1000 ]
```

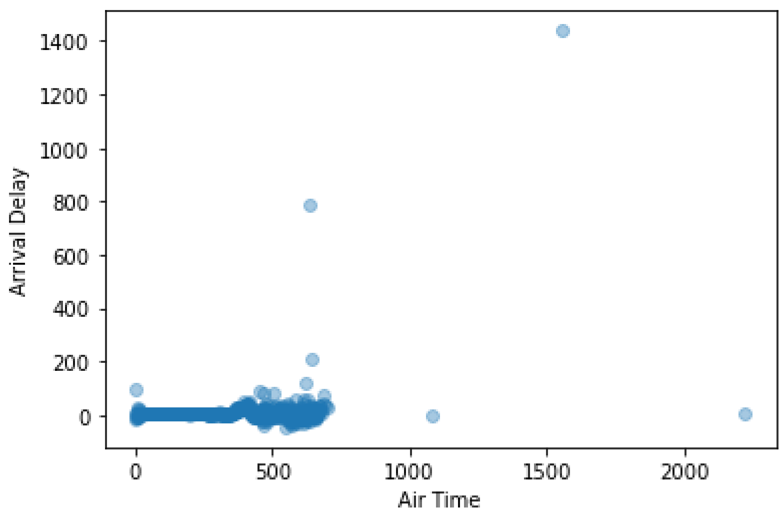
Out[178…

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN	DESTINATION	DISTANCE	DEP_DELAY	ARR_DELAY	CANCELLED	AIR_TIME
76718	2019-03-27	EV	N14542	4304	JAX	EWR	820.00	-9.00	1439.00	0.00	1557.00
1859374	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1859379	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1859864	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1859869	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1860354	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1860359	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1860844	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1860849	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1861334	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1861339	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1861824	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1861829	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1862314	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1862319	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1862804	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1862809	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1863294	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1863299	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00
1863784	3/2/19	WN	N937WN	4434	RSW	MCI	1155.00	-7.00	-5.00	0.00	1080.00
1863789	2/23/19	AS	N7845A	3434	RSW	MDW	1105.00	-1.00	5.00	0.00	2222.00

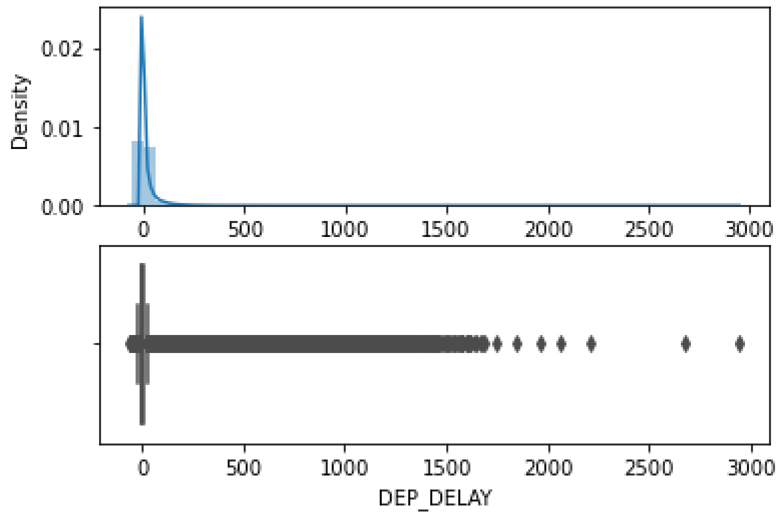
c) DEP_DELAY and ARR_DELAY variables

In [179…
```
# Air time vs arrival delay
# Air time range is between 0-800 predominantly with variation delay between 0 and 200. There is not much variation in arr_delay w
dist_delay = flights_df_cleaned.groupby('AIR_TIME', as_index = False)['ARR_DELAY'].mean()
x = dist_delay.AIR_TIME
y = dist_delay.ARR_DELAY
plt.xlabel("Air Time")
plt.ylabel("Arrival Delay")
plt.scatter(x, y, alpha=0.4)
```

Out[179…
```
<matplotlib.collections.PathCollection at 0x204b620c2b0>
```

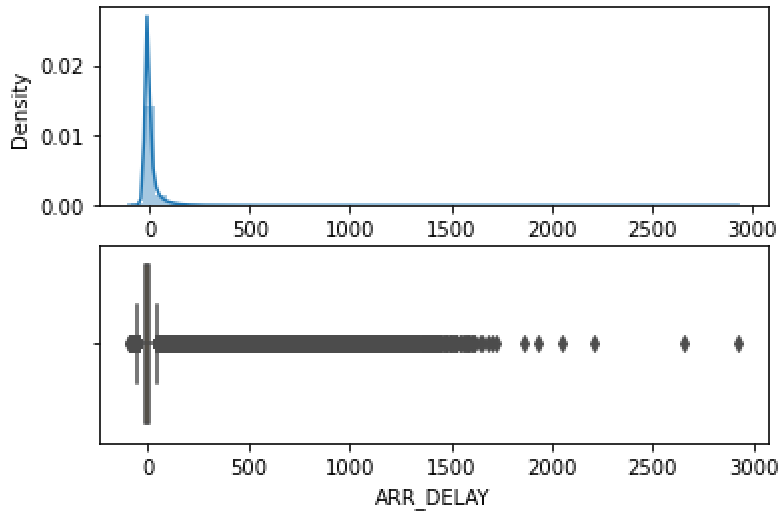


In [180…
```
show_distribution(flights_df_cleaned, 'DEP_DELAY')
```



In [181…
```
# Since departure and arrival delays are highly correlated they doesnt seem to much vary in range from above
```

In [182…
```
show_distribution(flights_df, 'ARR_DELAY')
```



In [183…
```
flights_df_cleaned.describe()
```

Out[183…

	DISTANCE	DEP_DELAY	ARR_DELAY	CANCELLED	AIR_TIME	OCCUPANCY_RATE
count	1859785.00	1859785.00	1859785.00	1859785.00	1857305.00	1859475.00
mean	772.46	10.72	5.65	0.00	109.40	0.65
std	581.11	49.96	52.41	0.00	70.45	0.20
min	31.00	-63.00	-94.00	0.00	1.00	0.30
25%	344.00	-6.00	-15.00	0.00	59.00	0.48
50%	612.00	-2.00	-6.00	0.00	91.00	0.65
75%	1013.00	7.00	8.00	0.00	139.00	0.83
max	4983.00	2941.00	2923.00	0.00	2222.00	1.00

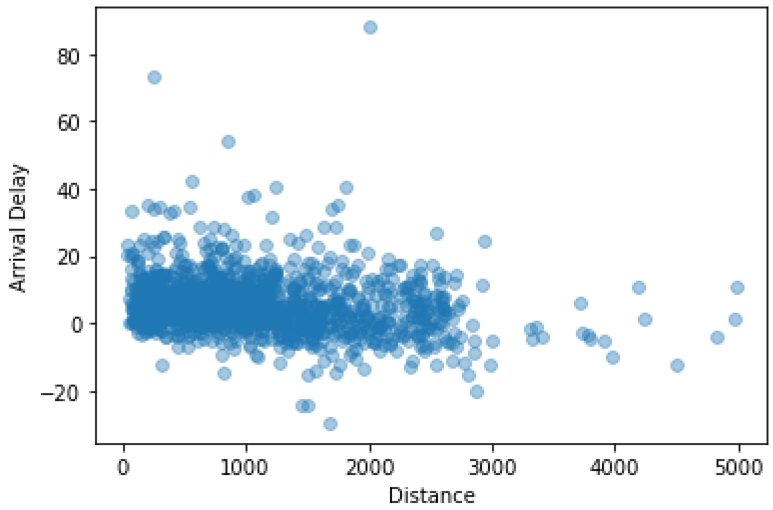
In [184…
```
# Very early departures
(flights_df_cleaned[flights_df_cleaned['DEP_DELAY'] < -15].shape[0]) / (flights_df_cleaned.shape[0]) * 100
```

Out[184… 0.7481509959484564

In [185…
```
# Flights can depart few mins early due to quick onboarding but otherwise very early departure is very less likely to happen. Henc
flights_df_cleaned = flights_df_cleaned.loc[~(flights_df_cleaned['DEP_DELAY'] < -15)]
```

In [186…
```
# Long haul flights has lesser variation in arrival delay over short haul flights
%matplotlib inline
dist_delay = flights_df_cleaned.groupby('DISTANCE')['ARR_DELAY'].mean().reset_index()

# plot a scatter plot that takes distance as predictor, and arrival delay as response
x = dist_delay['DISTANCE']
y = dist_delay['ARR_DELAY']
plt.xlabel("Distance")
plt.ylabel("Arrival Delay")
plt.scatter(x, y, alpha=0.4)
```

Out[186… <matplotlib.collections.PathCollection at 0x204b61ae250>



In [187…
```
find_outliers(flights_df_cleaned, 'ARR_DELAY')
```
Out[187… 45.0

In [188…
```
# Percent of >24hrs DEP_DELAY data out of total
(flights_df_cleaned[flights_df_cleaned['DEP_DELAY']>1440].shape[0])/ (flights_df_cleaned.shape[0]) * 100
```
Out[188… 0.0023836985358131747

In [189…
```
# Almost 44 flights have departed well later than 24 hrs/ 1440 mins which is less likely to happen without being cancelled
# Since we dont have delay reasons  excluding such data (0.002%) from analysis
flights_df_cleaned = flights_df_cleaned.loc[~(flights_df_cleaned['DEP_DELAY']>1440)]
```

In [190…
```
# Create a new column as ON_TIME where, if ARR_DELAY less than 15 mins its on-time else as delayed (as per Beaureau of Transportat
flights_df_cleaned['ON_TIME'] = flights_df_cleaned['ARR_DELAY'].apply(lambda x: 1 if x < 15 else 0)
```

In [191…
```
flights_df_cleaned['DELAY_GRT_30MINS'] = flights_df_cleaned['ARR_DELAY'].apply(lambda x: 1 if x > 30 else 0)
```

In [192…
```
find_nulls_post_grouping(flights_df_cleaned, ['ORIGIN','DESTINATION'], 'ARR_DELAY')
```
Out[192… 'No nulls in ARR_DELAY column, post grouping and mean aggregation'

In [193…
```
flights_df['OCCUPANCY_RATE'].isna().sum()
```
Out[193… 310

In [194…
```
find_nulls_post_grouping(flights_df, ['ORIGIN','DESTINATION'],'OCCUPANCY_RATE')
```
Out[194… 'No nulls in OCCUPANCY_RATE column, post grouping and mean aggregation'

3. Tickets File

In [195…
```
tickets_raw_df  = pd.read_csv('data/tickets.csv')
```

In [196…
```
# Provided dataset is for 2019-Q1
tickets_raw_df.head()
```
Out[196…

	ITIN_ID	YEAR	QUARTER	ORIGIN	ORIGIN_COUNTRY	ORIGIN_STATE_ABR	ORIGIN_STATE_NM	ROUNDTRIP	REPORTING_CARRIER	PASSENGERS
0	201912723049	2019	1	ABI	US	TX	Texas	1.00	MQ	1.00
1	201912723085	2019	1	ABI	US	TX	Texas	1.00	MQ	1.00
2	201912723491	2019	1	ABI	US	TX	Texas	1.00	MQ	1.00
3	201912723428	2019	1	ABI	US	TX	Texas	1.00	MQ	1.00
4	201912723509	2019	1	ABI	US	TX	Texas	0.00	MQ	1.00

In [197…
```
# Rountrips should oly be considered for analysis
# Columns interested are origin, destination and itin_fare alone
tickets_df = tickets_raw_df[tickets_raw_df['ROUNDTRIP']==1.0]
tickets_df = tickets_df[['ORIGIN','DESTINATION','ITIN_FARE']]
```

In [198…
```
# 560 fare details ~0.08% are nulls
find_missing_percent(tickets_df)
```
Total records: 708600

Out[198…

	VARIABLE	MISSING	PERCENT_MISSING
0	ITIN_FARE	560	0.08

In [199…
```
# Invalid fare detials are there for ~1200 records ~0.17% which is handled during numeric conversion
tickets_df['ITIN_FARE'].apply(check_numeric).dropna().value_counts()
```
Out[199…
```
200 $       713
$ 100.00    305
```

```
820$$$      272
Name: ITIN_FARE, dtype: int64
```

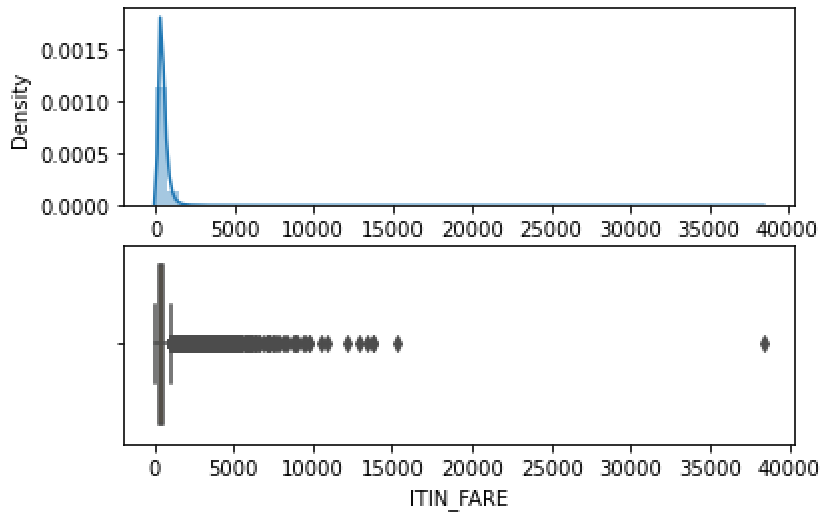
In [200…
```
tickets_df['ITIN_FARE'] = tickets_df['ITIN_FARE'].apply(pd.to_numeric, errors='coerce')
```

In [201…
```
tickets_df.describe()
```

Out[201…

	ITIN_FARE
count	706750.00
mean	473.69
std	344.27
min	0.00
25%	280.00
50%	416.00
75%	596.00
max	38400.00

In [202…
```
show_distribution(tickets_df, 'ITIN_FARE')
```



Averages do not include frequent-flyer or 'zero fares' or a few abnormally high reported fares.

In [203…
```
# As per Beareau of Transportation Statistics (BTS) for 2019Q1, fares above 10000 seems too expensive for roundtrip fare for US do
tickets_df[tickets_df['ITIN_FARE']>10000].shape[0]
```

Out[203… 9

In [204…
```
# Its unknown from the data if the passenger had applied coupons or which travel classes(first class or not) or bulk booking
# So as per BTS 2019Q1 and owing to above assumption, excluding <$50 round trip fare that is too less for US domestic flights
tickets_df[tickets_df['ITIN_FARE']<50].shape[0]
```

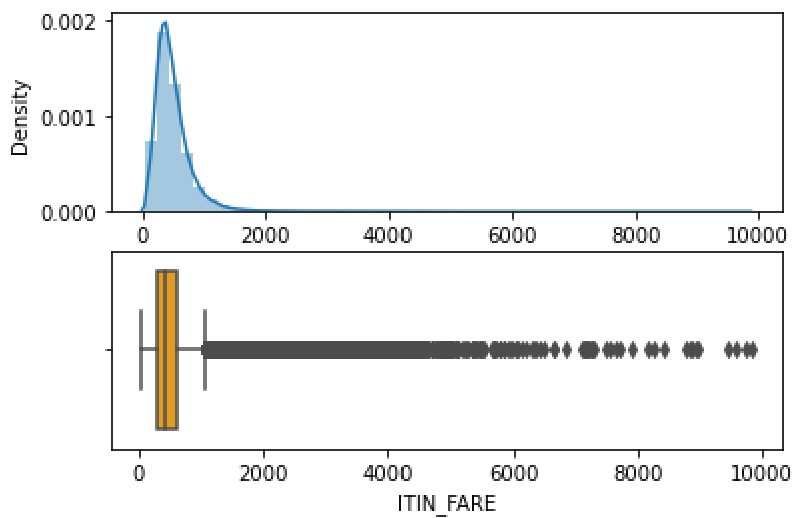
Out[204… 40520

In [205…
```
abnormal_fares_percent = ((tickets_df[tickets_df['ITIN_FARE']<50].shape[0] + tickets_df[tickets_df['ITIN_FARE']>10000].shape[0] )/
print("Records with round-trip-fares between 2 domestic airports with abnormal figures are (<50$ and >10000$) " + str(round(abnorm
```

```
Records with round-trip-fares between 2 domestic airports with abnormal figures are (<50$ and >10000$) 5.72%
```

In [206…
```
# Lets nullify such abnormal fares so that we get fair enough average fare by grouping ORIGIN-DEST pairs
tickets_df['ITIN_FARE'].mask((tickets_df['ITIN_FARE'] < 50) | (tickets_df['ITIN_FARE'] > 10000), np.nan, inplace=True)
```

In [207…
```
show_distribution(tickets_df,'ITIN_FARE')
```



In [208…
```
find_missing_percent(tickets_df)
```

```
Total records:  708600
```

Out[208…

	VARIABLE	MISSING	PERCENT_MISSING
0	ITIN_FARE	42379	5.98

In [209…

```
tickets_df['ITIN_FARE'].describe()
```

Out[209…

```
count    666221.00
mean        501.54
std         328.59
min          50.00
25%         304.00
50%         433.00
75%         611.00
max        9816.00
Name: ITIN_FARE, dtype: float64
```

In [210…

```
# Conditionally join two dataframes airport_df and tickets_df
```

In [211…

```
# Aggregation at leg level  for each origin-destination pair
flights_legs = flights_df_cleaned.groupby(['ORIGIN', 'DESTINATION'], as_index=False).agg({'OP_CARRIER': ['nunique'], 'TAIL_NUM': [
flights_legs.columns = ['origin', 'dest', 'carriers', 'flights', 'distance', 'occupancy_rate', 'avg_dep_delay' , 'avg_arr_delay',

flights_legs['on_time_percent'] = (flights_legs['on_time'] / flights_legs['flights']) * 100
flights_legs['delay_grt_30mins_percent'] = (flights_legs['delay_grt_30mins'] / flights_legs['flights']) * 100
flights_legs['on_time_percent'] = flights_legs['on_time_percent'].astype(int)
flights_legs['delay_grt_30mins_percent'] = flights_legs['delay_grt_30mins_percent'].astype(int)

drop_columns(flights_legs,['on_time', 'delay_grt_30mins'])

flights_legs[['flights', 'distance']] = flights_legs[['flights', 'distance']].apply(pd.to_numeric, errors="coerce")
```

In [212…

```
flights_legs
```

Out[212…

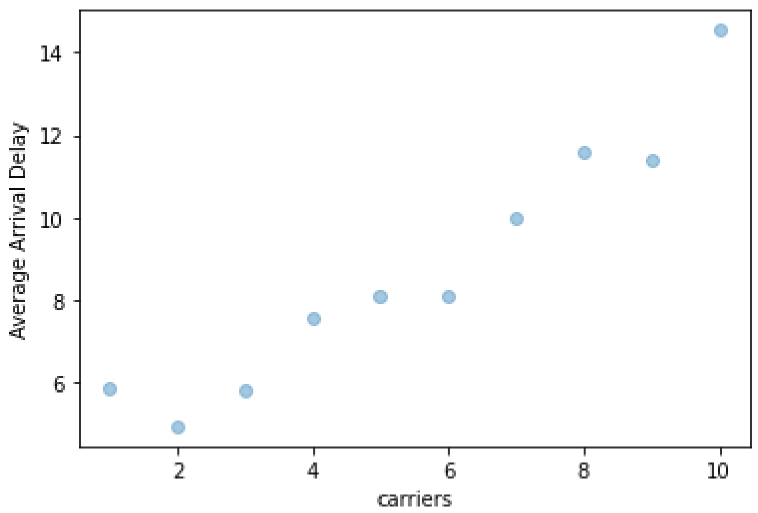
	origin	dest	carriers	flights	distance	occupancy_rate	avg_dep_delay	avg_arr_delay	on_time_percent	delay_grt_30mins_percent
0	ABE	ATL	2	217	692.00	0.64	5.98	5.46	78	13
1	ABE	CLT	1	248	481.00	0.67	4.06	4.51	81	8
2	ABE	DTW	2	248	425.00	0.64	15.97	10.69	78	12
3	ABE	FLL	1	20	1041.00	0.58	13.60	10.85	75	10
4	ABE	ORD	3	156	654.00	0.67	22.56	15.85	69	23
...	...	...	...	...	...	...	...	...	...	...
5905	YAK	CDV	1	41	213.00	0.62	9.22	11.83	75	12
5906	YAK	JNU	1	27	198.00	0.77	16.96	18.41	70	14
5907	YKM	SEA	1	305	103.00	0.66	11.62	11.50	74	14
5908	YUM	DFW	1	28	1022.00	0.64	16.00	15.46	89	7
5909	YUM	PHX	2	341	160.00	0.65	0.87	-2.52	89	5

5910 rows × 10 columns

In [213…

```
%matplotlib inline
dist_delay = flights_legs.groupby('carriers')['avg_arr_delay'].mean().reset_index()
x = dist_delay['carriers']
y = dist_delay['avg_arr_delay']
plt.xlabel("carriers")
plt.ylabel("Average Arrival Delay")
plt.scatter(x, y, alpha=0.4)

# Arrival delay seems higher with increase in careers
```

Out[213… <matplotlib.collections.PathCollection at 0x204b3f0f820>



In [214…

```
# Merging of airports and tickets by iata_code to that of origin and destination
tickets_airport_origin_merge  = tickets_df.merge(airport_df, how='inner', left_on='ORIGIN', right_on='IATA_CODE')[['TYPE','ORIGIN'
tickets_airport_dest_merge    = tickets_airport_origin_merge.merge(airport_df, how='inner', left_on='DESTINATION', right_on='IATA_
```

In [215…

```
tickets_airport_merge_filtered = tickets_airport_dest_merge.loc[(tickets_airport_dest_merge['ORIGIN_TYPE'].isin(['medium_airport',
tickets_airport_merge_filtered['route'] = np.where(tickets_airport_merge_filtered.ORIGIN > tickets_airport_merge_filtered.DESTINAT
tickets_airport_merge_filtered['ITIN_FARE']=tickets_airport_merge_filtered['ITIN_FARE'].apply(lambda x : pd.to_numeric(x, errors='
```

In [216…

```
roundtrip_fares = tickets_airport_merge_filtered.groupby('route').agg({ 'ORIGIN_TYPE':['first'],'ORIGIN':['min'], 'DESTINATION':['
```

```
roundtrip_fares.columns = ['origin_type', 'origin', 'dest', 'dest_type', 'avg_rtr_fare']
```

In [217…

```
# Avg rountrip fares for each origin-dest pair between medium and large airports
roundtrip_fares
```

Out[217…

	origin_type	origin	dest	dest_type	avg_rtr_fare
0	medium_airport	ABI	ABE	medium_airport	758.00
1	medium_airport	ABE	ABQ	large_airport	534.00
2	large_airport	ABE	AGS	medium_airport	391.00
3	medium_airport	ABE	AMA	large_airport	654.00
4	medium_airport	ABE	ASE	medium_airport	742.00
...	...	...	...	...	...
24040	medium_airport	VLD	XNA	medium_airport	778.67
24041	medium_airport	VPS	XNA	large_airport	270.90
24042	large_airport	VPS	YUM	medium_airport	796.00
24043	medium_airport	YAK	WRG	medium_airport	745.00
24044	medium_airport	YKM	XNA	medium_airport	461.00

24045 rows × 5 columns

In [218…

```
find_missing_percent(roundtrip_fares)
```

```
Total records:  24045
```

Out[218…

	VARIABLE	MISSING	PERCENT_MISSING
0	avg_rtr_fare	365	1.52

In [219…

```
# Since %nulls is minimal for avg_rtr_fare in the order of ~2% for grouped origin-dest pair, dropping them as profit cant be calcu
roundtrip_fares = roundtrip_fares.loc[roundtrip_fares['avg_rtr_fare'].notna()]
```

In [220…

```
roundtrip_fares['avg_rtr_fare'].describe()
```

Out[220…

```
count    23680.00
mean       586.40
std        250.23
min         54.00
25%        440.46
50%        534.09
75%        662.94
max       5960.00
Name: avg_rtr_fare, dtype: float64
```

In [221…

```
flights_legs[flights_legs['flights']<50].shape[0]
```

Out[221…

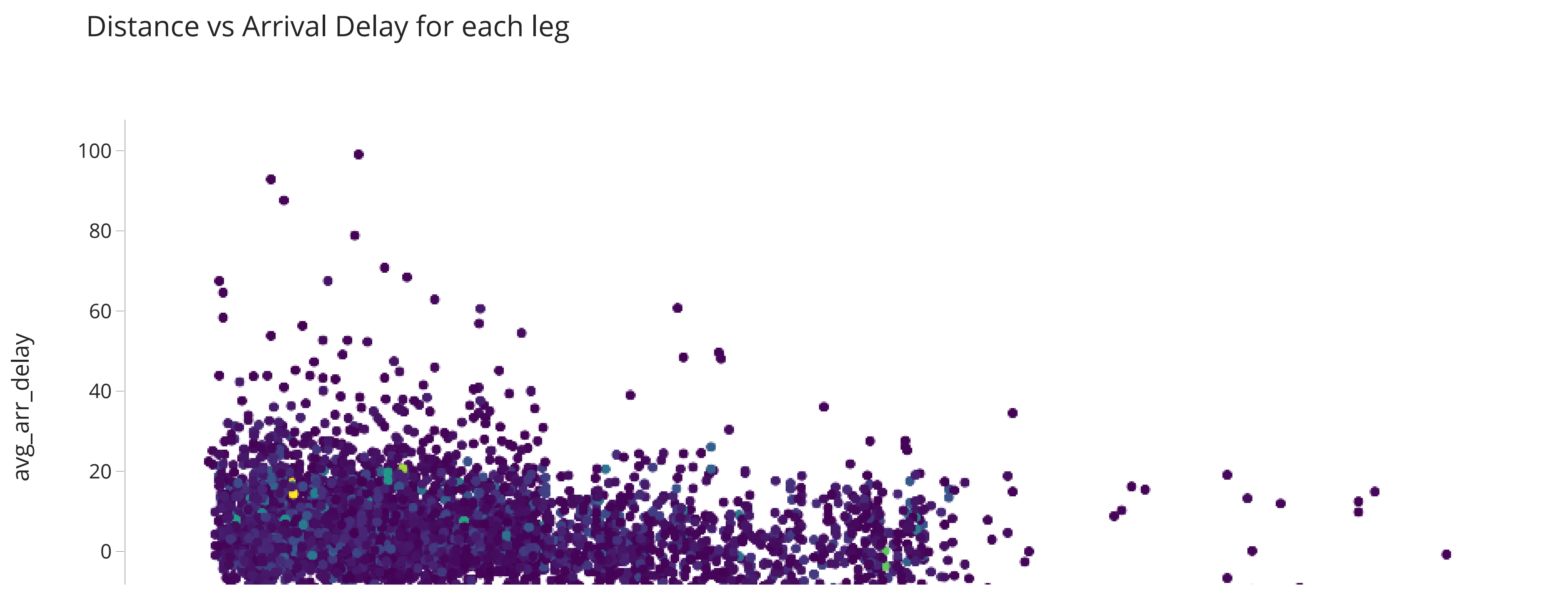
```
1028
```

In [222…

```
# Considering only the routes which has flights > 50 for Q1 2019
flights_legs_filtd = flights_legs[flights_legs['flights']>50]
```

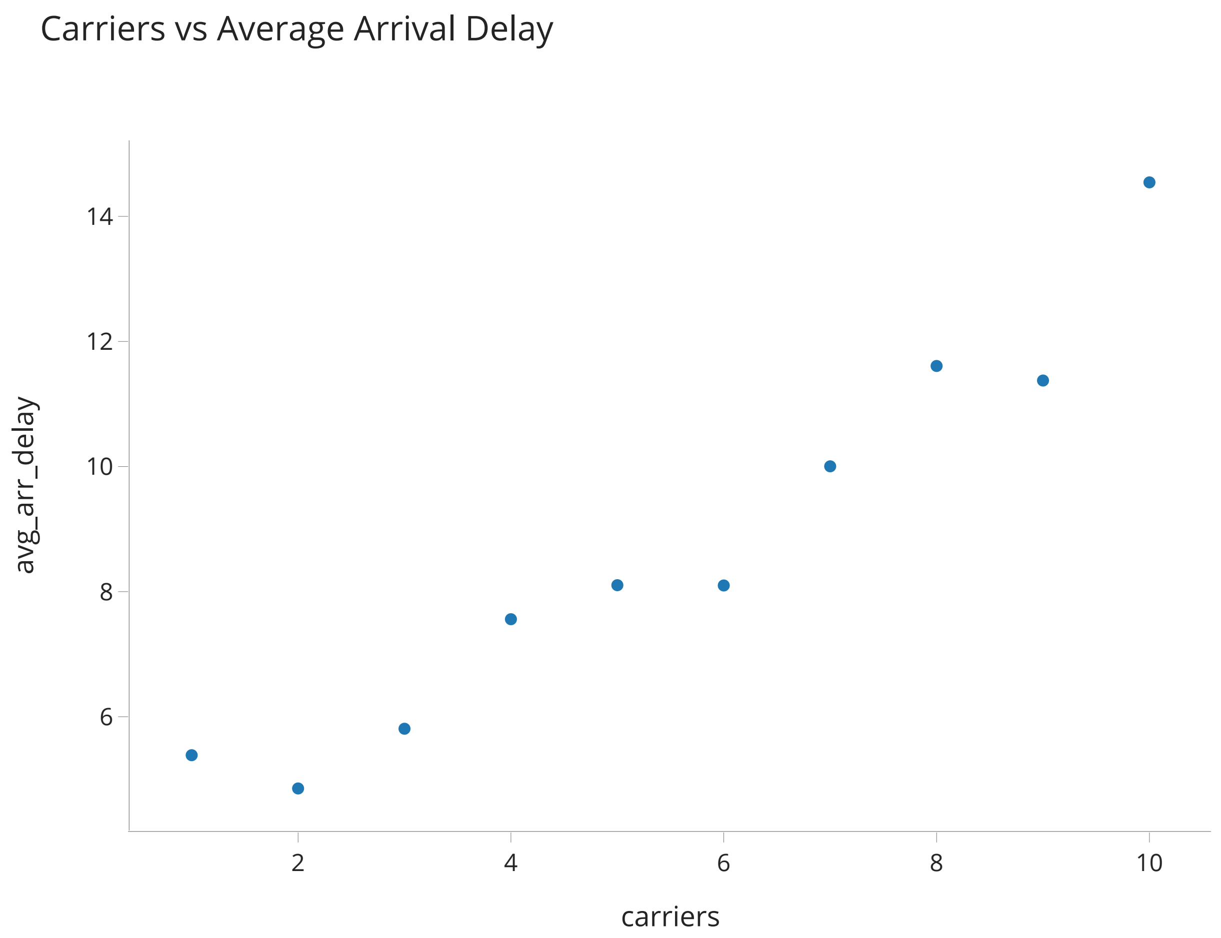
In [223…

```
# Distance vs Arrival Delay for each leg
import plotly.express as px

fig = px.scatter(flights_legs_filtd, x="distance", y="avg_arr_delay",color='flights',
                title="Distance vs Arrival Delay for each leg",template="simple_white")

fig.update_layout(barmode='stack', xaxis={'categoryorder':'total descending'})
fig.show("notebook")
```



In [224…

```
# Carriers vs Average Arrival Delay
dist_delay = flights_legs_filtd.groupby('carriers')['avg_arr_delay'].mean().reset_index()

fig1 = px.scatter(dist_delay, x="carriers", y="avg_arr_delay",#,color='flights',
                title="Carriers vs Average Arrival Delay",template="simple_white")
fig1.update_layout(width=700)
fig1.show("notebook")
```



In [225…

```
# Lets merge roundtrip fares dataframe with that of flights dataframe to find avg fare, avg distance, avg delays at each leg level
```

In [226…

```
flights_rtr_legs_query = '''
SELECT f.origin,
f.dest,
f.carriers,
f.flights,
f.distance,
f.occupancy_rate,
f.avg_dep_delay,
f.avg_arr_delay,
d.avg_rtr_fare,
d.origin_type,
d.dest_type,
f.on_time_percent,
f.delay_grt_30mins_percent
FROM roundtrip_fares d JOIN flights_legs_filtd f
on ((d.origin=f.origin and d.dest=f.dest) or (d.origin=f.dest and d.dest=f.origin)) '''
```

In [227…

```
flights_rtr_legs = sqldf(flights_rtr_legs_query, locals())
```

In [228…

```
# Merged dataframe consisiting of roundtrip routes at each leg level with aggregated metrics
flights_rtr_legs
```

Out[228…

	origin	dest	carriers	flights	distance	occupancy_rate	avg_dep_delay	avg_arr_delay	avg_rtr_fare	origin_type	dest_type	on_time_percent	c
0	ABE	ATL	2	217	692.00	0.64	5.98	5.46	576.91	medium_airport	large_airport	78	
1	ATL	ABE	2	217	692.00	0.67	7.76	-0.35	576.91	medium_airport	large_airport	87	
2	ABE	CLT	1	248	481.00	0.67	4.06	4.51	499.24	medium_airport	large_airport	81	
3	CLT	ABE	1	251	481.00	0.66	3.86	-1.32	499.24	medium_airport	large_airport	87	
4	ABE	DTW	2	248	425.00	0.64	15.97	10.69	449.50	medium_airport	large_airport	78	
...	...	...	...	...	...	...	...	...	...	...	...	...	
4675	TPA	STL	1	211	869.00	0.67	10.80	2.90	399.75	large_airport	large_airport	80	

	origin	dest	carriers	flights	distance	occupancy_rate	avg_dep_delay	avg_arr_delay	avg_rtr_fare	origin_type	dest_type	on_time_percent	c
4676	STL	TUL	1	163	351.00	0.65	9.47	4.58	434.72	large_airport	large_airport	78	
4677	TUL	STL	1	161	351.00	0.64	2.55	-4.10	434.72	large_airport	large_airport	92	
4678	TPA	TTN	1	77	955.00	0.63	22.32	12.22	183.09	medium_airport	large_airport	74	
4679	TTN	TPA	1	71	955.00	0.63	10.82	6.00	183.09	medium_airport	large_airport	78	

4680 rows × 13 columns

In [229…

```
# Roundtrip route identifier
flights_rtr_legs['route'] = np.where(flights_rtr_legs.origin > flights_rtr_legs.dest, flights_rtr_legs['dest']+'-'+flights_rtr_leg
```

In [230…

```
# Group corresponding leg level flights data as single roundtrip route
flights_rtr_grp         = flights_rtr_legs.groupby('route').agg({  'flights': ['min'], 'carriers': ['mean'] , 'distance': ['sum']
'delay_grt_30mins_percent':'mean', 'route': 'count'}).round(2).reset_index()
flights_rtr_grp.columns = ['roundtrip_route', 'avg_flights', 'avg_carriers', 'distance', 'occupancy_rate', 'avg_dep_delay' , 'avg_

flights_rtr_grp['on_time%'] = flights_rtr_grp['on_time%'].astype(int)
flights_rtr_grp['delay_grt_30mins%'] = flights_rtr_grp['delay_grt_30mins%'].astype(int)
flights_rtr_grp = flights_rtr_grp.loc[flights_rtr_grp['cnt']>1] # means filter rountrips
drop_columns(flights_rtr_grp, ['cnt'])
```

In [231…

```
flights_rtr_grp[['roundtrip_route','avg_flights','avg_carriers','distance','occupancy_rate','avg_dep_delay','avg_arr_delay','on_ti
```

Out[231…

	roundtrip_route	avg_flights	avg_carriers	distance	occupancy_rate	avg_dep_delay	avg_arr_delay	on_time%	delay_grt_30mins%	avg_rtr_fare
0	ABE-ATL	217	2.00	1384.00	0.65	6.87	2.56	82	10	576.91
1	ABE-CLT	248	1.00	962.00	0.66	3.96	1.60	84	7	499.24
2	ABE-DTW	248	2.00	850.00	0.64	13.70	5.98	79	12	449.50
3	ABE-ORD	156	3.00	1308.00	0.65	29.40	24.08	64	29	585.79
4	ABE-SFB	111	1.00	1764.00	0.66	6.26	3.36	83	9	205.35
...	...	...	...	...	...	...	...	...	...	...
2346	SMF-STL	88	1.00	3358.00	0.66	9.19	0.64	85	7	502.27
2347	SNA-STS	87	1.00	866.00	0.64	8.98	3.81	83	9	328.26
2348	STL-TPA	209	1.00	1738.00	0.66	11.33	5.08	79	9	399.75
2349	STL-TUL	161	1.00	702.00	0.65	6.01	0.24	85	7	434.72
2350	TPA-TTN	71	1.00	1910.00	0.63	16.57	9.11	76	16	183.09

2329 rows × 10 columns

In [232…

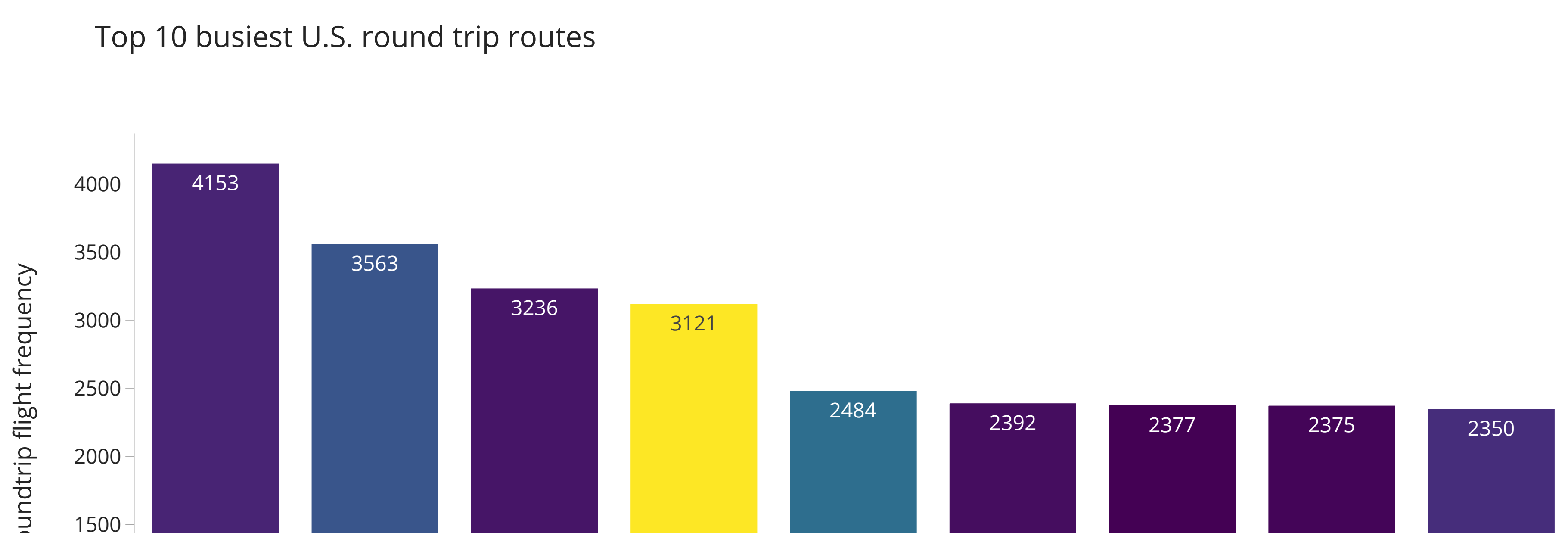
```
flights_rtr_grp_top10 = flights_rtr_grp.nlargest(10,'avg_flights').reset_index(drop=True)
```

In [233…

```
flights_rtr_grp_top10["Rank"] = flights_rtr_grp_top10["avg_flights"].rank(method='dense',ascending=False).astype(int)
```

In [234…

```
# Bar plot - Top 10 busiest U.S. round trip routes
import plotly.express as px

fig = px.bar( flights_rtr_grp_top10, x="roundtrip_route", y="avg_flights",text='avg_flights',color='distance',
              title="Top 10 busiest U.S. round trip routes",
              labels={
                     "roundtrip_route": "roundtrip routes",
                     "avg_flights": "avg roundtrip flight frequency"
                 },
              template="simple_white")
fig.update_layout(barmode='stack', xaxis={'categoryorder':'total descending'})
fig.show("notebook")
```



In [235…
```
# Tabular data - Top 10 busiest U.S. round trip routes
import plotly.graph_objects as go
col = ['<b>RANK</b>', '<b>ROUNDTRIP ROUTES</b>', '<b>AVERAGE FLIGHTS</b>', '<b>ON TIME%</b>', '<b>AVERAGE CARRIERS</b>', '<b>DISTA
df = flights_rtr_grp_top10

fig = go.Figure(data=[go.Table(
    columnwidth = [40,80],
    header=dict(values=col,
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[df.Rank, df.roundtrip_route, df.avg_flights, df['on_time%'], df.avg_carriers, df.distance],
               fill_color='lavender',
               align='left'))
])
fig.update_layout(width=720)

fig.show("notebook")
```

RANK	**ROUNDTRIP ROUTES**	**AVERAGE FLIGHTS**	**ON TIME%**	**AVERAGE CARRIERS**	**DISTANCE (MILES)**
1	LAX-SFO	4153	69	7	674
2	LGA-ORD	3563	66	6	1466
3	LAS-LAX	3236	78	8	472
4	JFK-LAX	3121	82	4	4950
5	LAX-SEA	2484	78	6	1908
6	BOS-LGA	2392	71	5	368
7	HNL-OGG	2377	91	1	200
8	PDX-SEA	2375	77	5	258
9	ATL-MCO	2350	85	5	808
10	ATL-LGA	2291	77	5	1524

Generic functions for key metrics calculation

In [236…
```
TOTAL_PASSENGERS = 200
FUEL_COST_PER_MILE = 8
DELAY_COST_PER_MIN = 75
INSURANCE_COST_PER_MILE = 1.18
UPFRONT_PLANE_COST = 90000000

OPERATIONAL_COST_MEDIUM_AIRPORT = 5000
OPERATIONAL_COST_LARGE_AIRPORT = 10000
ROUNDTRIP_BAGGAGE_PER_CUSTOMER = 70
ALLOWED_DELAY_IN_MINS=15

def calculateRevenue(df):
    baggage_fees = 0.50*(df.avg_flights*df.occupancy_rate*TOTAL_PASSENGERS)*ROUNDTRIP_BAGGAGE_PER_CUSTOMER
    tickets_fares = df.avg_flights*df.occupancy_rate*TOTAL_PASSENGERS*df.avg_rtr_fare
    return (baggage_fees + tickets_fares)
```

In [237…
```
def calculateTotalCost(df):
    total_fuel_cost = df.avg_flights * ( (df.distance*FUEL_COST_PER_MILE) + (df.distance*INSURANCE_COST_PER_MILE) )
    total_operational_cost = np.where(df.origin_type=='medium_airport',OPERATIONAL_COST_MEDIUM_AIRPORT,OPERATIONAL_COST_LARGE_AIRP
    total_delay_cost = np.where(df.avg_dep_delay>ALLOWED_DELAY_IN_MINS, (df.avg_dep_delay-ALLOWED_DELAY_IN_MINS)*DELAY_COST_PER_MI
    return (total_fuel_cost + total_operational_cost + total_delay_cost)
```

In [238…
```
def convertToMillions(df,col):
```

```
    return '$'+(df[col].astype(float)/1000000).round(1).astype(str) + 'M'
```

In [239…

```
def calculate_breakeven(df):
    return (UPFRONT_PLANE_COST / (df['total_profit_usd']/df['avg_flights']))
```

In [240…

```
flights_rtr_grp['total_revenue_usd'] = calculateRevenue(flights_rtr_grp)
flights_rtr_grp['total_cost_usd'] = calculateTotalCost(flights_rtr_grp)
flights_rtr_grp['total_profit_usd'] = flights_rtr_grp['total_revenue_usd'] - flights_rtr_grp['total_cost_usd']
flights_rtr_grp['total_profit_per_carrier_usd'] = flights_rtr_grp['total_profit_usd'] / flights_rtr_grp['avg_carriers']
flights_rtr_profit_top10 =  flights_rtr_grp.nlargest(10,'total_profit_usd').reset_index(drop=True)

# Conversion to necessary types
flights_rtr_profit_top10[['avg_carriers', 'distance', 'avg_rtr_fare', 'total_revenue_usd', 'total_cost_usd', 'total_profit_usd', '

# Formatting as reqd
flights_rtr_profit_top10['total_revenue'] = convertToMillions(flights_rtr_profit_top10,'total_revenue_usd')
flights_rtr_profit_top10['total_cost']    = convertToMillions(flights_rtr_profit_top10,'total_cost_usd')
flights_rtr_profit_top10['total_profit']  = convertToMillions(flights_rtr_profit_top10,'total_profit_usd')
flights_rtr_profit_top10['profit_share']  = convertToMillions(flights_rtr_profit_top10,'total_profit_per_carrier_usd')

# Ranking based on most profittable
flights_rtr_profit_top10[["Rank"]] = flights_rtr_profit_top10[["total_profit_usd"]].rank(method='dense',ascending=False).astype(in
flights_rtr_profit_top10 = flights_rtr_profit_top10.sort_values("Rank")
```

In [241…

```
# Tabular data - Top10 most profitable routes
import plotly.graph_objects as go
col = ['<b>RANK</b>', '<b>ROUNDTRIP ROUTES</b>', '<b>ROUNDTRIP FLIGHTS</b>','<b>TOTAL PROFIT</b>','<b>TOTAL REVENUE</b>','<b>TOTAL
df = flights_rtr_profit_top10

fig = go.Figure(data=[go.Table(
    columnwidth =[20,40],
    header=dict(values=col,
                fill_color='powderblue',
                align='left'),
    cells=dict(values=[df.Rank, df.roundtrip_route, df.avg_flights,df.total_profit,df.total_revenue,df.total_cost, df['on_time%'],
               fill_color='paleturquoise',
               align='left'))
])
fig.update_layout(width=990)
fig.show("notebook")
```

RANK	ROUNDTRIP ROUTES	ROUNDTRIP FLIGHTS	TOTAL PROFIT	TOTAL REVENUE	TOTAL COST	ON TIME%	AVERAGE CARRIERS	DISTANCE (MILES)	AVERAGE FARE
1	JFK-LAX	3121	$275.5M	$417.3M	$141.8M	82	4	4950	993
2	LAX-SFO	4153	$157.2M	$182.9M	$25.7M	69	7	674	303
3	LGA-ORD	3563	$141.1M	$189.1M	$48.0M	66	6	1466	373
4	JFK-SFO	1838	$133.0M	$220.3M	$87.3M	74	4	5172	886
5	EWR-SFO	1187	$121.7M	$177.7M	$55.9M	69	2	5130	1116
6	ATL-LGA	2291	$119.0M	$151.1M	$32.1M	77	5	1524	472
7	BOS-LGA	2392	$117.5M	$125.6M	$8.1M	71	5	368	368
8	DCA-ORD	1832	$115.2M	$135.8M	$20.6M	78	6	1224	535
9	LAS-LAX	3236	$108.0M	$122.0M	$14.0M	78	8	472	255
10	DCA-LGA	1672	$105.7M	$112.3M	$6.6M	69	2	428	481

In [242…

```
# Breakeven analysis
flights_rtr_profit_top10['breakeven'] = calculate_breakeven(flights_rtr_profit_top10)
flights_rtr_profit_top10['breakeven'] = flights_rtr_profit_top10['breakeven'].astype(int)

# Breakeven in terms of roundtrip flights
flights_rtr_profit_top10[['roundtrip_route','avg_flights',  'total_revenue', 'total_cost', 'total_profit', 'breakeven', 'avg_carri
```

Out[242…

	roundtrip_route	avg_flights	total_revenue	total_cost	total_profit	breakeven	avg_carriers	distance
0	JFK-LAX	3121	$417.3M	$141.8M	$275.5M	1019	4	4950
1	LAX-SFO	4153	$182.9M	$25.7M	$157.2M	2377	7	674
2	LGA-ORD	3563	$189.1M	$48.0M	$141.1M	2272	6	1466

	roundtrip_route	avg_flights	total_revenue	total_cost	total_profit	breakeven	avg_carriers	distance
3	JFK-SFO	1838	$220.3M	$87.3M	$133.0M	1243	4	5172
4	EWR-SFO	1187	$177.7M	$55.9M	$121.7M	877	2	5130
5	ATL-LGA	2291	$151.1M	$32.1M	$119.0M	1733	5	1524
6	BOS-LGA	2392	$125.6M	$8.1M	$117.5M	1831	5	368
7	DCA-ORD	1832	$135.8M	$20.6M	$115.2M	1431	6	1224
8	LAS-LAX	3236	$122.0M	$14.0M	$108.0M	2696	8	472
9	DCA-LGA	1672	$112.3M	$6.6M	$105.7M	1423	2	428

In [243…

```
# Recommended routes
recommended_routes = flights_rtr_profit_top10
recommended_routes["Rank"] = recommended_routes[["on_time%","delay_grt_30mins%","total_profit_usd"]].apply(tuple,axis=1)\
                 .rank(method='dense',ascending=False).astype(int)
recommended_routes = recommended_routes.sort_values("Rank")
#recommended_routes[['Rank','roundtrip_route','avg_flights', 'distance', 'occupancy_rate', 'avg_dep_delay', 'avg_arr_delay', 'on_t
```

In [244…

```
# Recommended Routes
import plotly.graph_objects as go
col = ['<b>RANK</b>', '<b>ROUNDTRIP ROUTES</b>', '<b>AVERAGE ROUNDTRIP FLIGHTS</b>','<b>AVERAGE ARRIVAL DELAY</b>','<b>ON TIME%</b
df = recommended_routes

fig = go.Figure(data=[go.Table(
    columnwidth =[30,60],
    header=dict(values=col,
                fill_color='powderblue',
                align='left'),
    cells=dict(values=[df.Rank, df.roundtrip_route, df.avg_flights,df.avg_arr_delay,df['on_time%'],df['delay_grt_30mins%'], df.tot
               fill_color='lavender',
               align='left'))
])
fig.update_layout(width=900)
fig.show("notebook")
```

RANK	ROUNDTRIP ROUTES	AVERAGE ROUNDTRIP FLIGHTS	AVERAGE ARRIVAL DELAY	ON TIME%	ARRIVAL DELAY>30%	TOTAL PROFIT	AVERAGE CARRIERS	DISTANCE (MILES)
1	JFK-LAX	3121	-1.8	82	10	$275.5M	4	4950
2	DCA-ORD	1832	6.54	78	13	$115.2M	6	1224
3	LAS-LAX	3236	7.02	78	12	$108.0M	8	472
4	ATL-LGA	2291	5.47	77	14	$119.0M	5	1524
5	JFK-SFO	1838	6.96	74	17	$133.0M	4	5172
6	BOS-LGA	2392	14.49	71	20	$117.5M	5	368
7	LAX-SFO	4153	15.79	69	22	$157.2M	7	674
8	DCA-LGA	1672	12.09	69	20	$105.7M	2	428
9	EWR-SFO	1187	14.83	69	19	$121.7M	2	5130
10	LGA-ORD	3563	19.71	66	23	$141.1M	6	1466

In [245…

```
#Breakeven table
import plotly.graph_objects as go
col = ['<b>ROUND TRIP ROUTES</b>', '<b>AVERAGE ROUNDTRIP FLIGHTS</b>','<b>TOTAL REVENUE</b>','<b>TOTAL COST</b>','<b>TOTAL PROFIT<
df = recommended_routes.nsmallest(5,'Rank').reset_index(drop=True)

fig = go.Figure(data=[go.Table(
    columnwidth =[30,30],
    header=dict(values=col,
                fill_color='powderblue',
                align='left'),
    cells=dict(values=[ df.roundtrip_route, df.avg_flights,df.total_revenue,df.total_cost, df.total_profit,df.occupancy_rate, df.b
               fill_color='paleturquoise',
               align='left'))
])
fig.update_layout(width=750)
fig.show("notebook")
```