Airline Data Challenge

The case given here is a case study which involves **Market Entry** Strategy for an Airline which wants to enter the US domestic market. The main focus market of this airline is involving 5 round trips between medium and large airports in US. The airlines motto is “On time for You”.

History of the Industry:-

The history of the airline industry is a fascinating narrative marked by significant milestones, challenges, and advancements that have shaped the way people and goods move across the globe. The industry's origins trace back to the pioneering efforts of aviation visionaries such as the Wright brothers, who achieved the first controlled powered flight in 1903. The post-World War I era witnessed the emergence of the first commercial airlines, albeit on a limited scale. The 1920s and 1930s marked the dawn of scheduled passenger services, with airlines like Pan American Airways and Imperial Airways initiating long-distance flights, linking continents and laying the groundwork for international air travel.

The mid-20th century witnessed a transformative shift with the introduction of jet engines, exemplified by the Boeing 707 and Douglas DC-8. This era marked the onset of the "Jet Age," bringing about faster and more efficient air travel. The 1978 Airline Deregulation Act in the United States triggered a wave of liberalization, leading to increased competition, reduced fares, and expanded route networks. The late 20th century and early 21st century saw the rise of mega-carriers, globalization of alliances, and the advent of low-cost carriers, contributing to the industry's dynamic landscape. Despite facing challenges such as economic downturns, security concerns, and global crises, the airline industry has consistently demonstrated resilience, adaptability, and a commitment to innovation, making it an integral force in shaping the interconnected world we experience today.

Introduction to the Industry:-

The commercial airline industry, since its inception, has undergone a transformative journey that has significantly shaped global transportation and connectivity. The roots of commercial aviation can be traced back to the early 20th century when pioneers like the Wright brothers successfully demonstrated controlled powered flight. However, it was in the post-World War I era that the commercial potential of aviation began to unfold. The first scheduled commercial airline service commenced on January 1, 1914, with the St. Petersburg-Tampa Airboat Line, marking a pivotal moment in the industry's history. Over the subsequent decades, technological advancements, such as the introduction of jet engines and the development of larger and more efficient aircraft, propelled the industry forward.

The commercial airline industry has played a central role in fostering global connectivity, economic growth, and cultural exchange. As air travel became more accessible and efficient, airlines expanded their routes, and airports evolved into vital hubs facilitating international travel. The industry's growth was further accelerated by deregulation in the late 20th century, leading to increased competition, lower fares, and a broader range of services. Today, the airline industry is a critical component of the global economy, connecting people and goods across continents, contributing to tourism, trade, and economic development on a scale unimaginable during its early years. The evolution of commercial aviation stands as a testament to human ingenuity and innovation, shaping the modern world's interconnected and accessible nature.

Given to analyze the 1Q2019 data for identifying(Remove)

a) Top 10 busiest round trip routes for the quarter, cancel flights excluded
b) Top 10 most profitable round trip routes(no upfront airlines cost) for the quarter. Along with profits, show total revenue, total cost, summary values of other key components and total round trip flights for 10 most profitable
c) Top 5 round trips I would recommend to invest in, based on the factors I would choose
d) No of round trip flights to breakeven
e) KPI for future tracking for measuring success of the round trip flights

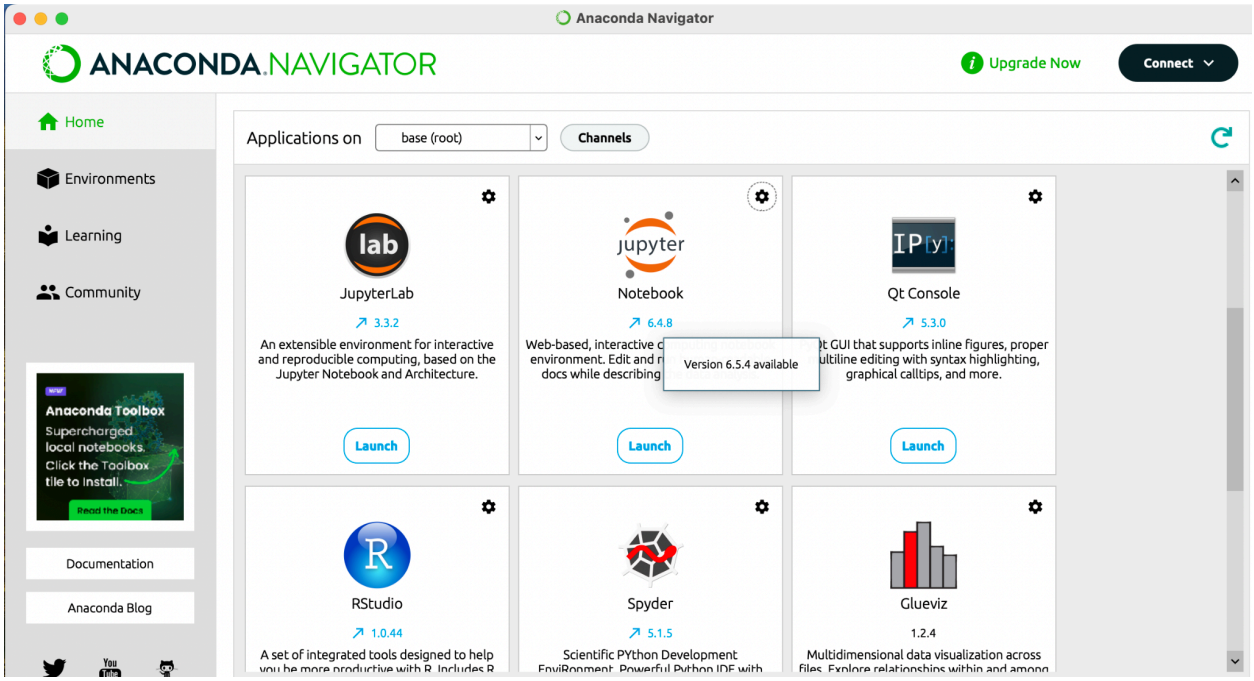
DataSets provided

a) Flights:- Available route from origin to destination, occupancy use this dataset
b) Tickets:- Ticket price data, round trip only in analysis.
c) Airport Codes:- Identify whether airport is medium or large. Only medium and large airports to be considere

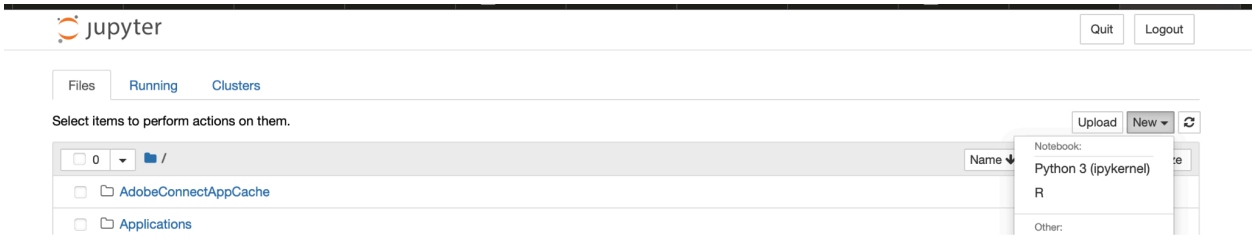
For the given data challenge i have decided to use Python

How to load Python

- Install Anaconda Software, this is available to download from Google
- Once downloaded, open Jupyter Notebook



- Once you open Jupyter choose Python3 Kernel(pykernel)



Load Datasets and Cleaning Datasets

Flights Data Set

- The first step is to load the dataset for the flights on the Jupyter notebook

- The next step is to to view the dataset loaded
  df_Flights

Out[3]:

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	D
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	
...	...	...	...	...	...	...	...	...	...	
1915881	3/23/19	AA	N903NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915882	3/24/19	AA	N965AN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915883	3/25/19	AA	N979NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915884	3/26/19	AA	N872NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915885	3/27/19	AA	N945AN	1433	15370	TUL	Tulsa, OK	11057	CLT	

```
#Duplicated values
df_Flights.duplicated()
#Check which rows are duplicated
```

```
0          False
1          False
2          False
3          False
4          False
           ...
1915881     True
1915882     True
1915883     True
1915884     True
1915885     True
Length: 1915886, dtype: bool
```

  - The above code is used to know which of the rows are duplicated.
- Data cleaning step here is to remove the null values from the dataset

The next step is to drop duplicate values and calculate the null values in each column

In [6]:
```
#Drop duplicates
df_Flights = df_Flights.drop_duplicates()
```

In [7]:
```
# Calculate the sum of null values in each column of the df_Flights
null_values = df_Flights.isnull().sum()
print("Null values in each column:")
print(null_values)
```

```
Null values in each column:
FL_DATE                 0
OP_CARRIER              0
TAIL_NUM            12111
OP_CARRIER_FL_NUM       0
ORIGIN_AIRPORT_ID       0
ORIGIN                  0
ORIGIN_CITY_NAME        0
DEST_AIRPORT_ID         0
DESTINATION             0
DEST_CITY_NAME          0
DEP_DELAY           50216
ARR_DELAY           55847
CANCELLED               0
AIR_TIME            55903
DISTANCE               63
OCCUPANCY_RATE         31
dtype: int64
```

The next step is to extract and store the values from Air_Time column

In [8]:
```
# Extract and store the values from the 'AIR_TIME' column of the df_Flights DataFrame into the variable k
k = df_Flights['AIR_TIME'].values
```

In [9]:
```
#Looping statment goes through each statement
s=[] # Initialize an empty list named 's'
for i in k: # Start a loop over each element in the iterable 'k'
  #print(type(i))
  s.append(type(i)) # Append the type of the current element 'i' to the list 's'
```

In [10]:
```
from collections import Counter
s = Counter(s)   # Import the Counter class from the collections module
 # Create a Counter object from the list 's'.
               # This will count the occurrences of each element in the list and store them as a dictionary-like ob
```

In [11]:
```
s
# This line transforms 's' into a Counter object.
# Originally, 's' is a list containing the types of elements from another iterable 'k' (as seen in the previous code
# By applying Counter to 's', it now becomes a dictionary-like object where each key is a unique type found in 's',
# and the corresponding value is the number of times that type appears in the original list 's'.
```

Out[11]:
```
Counter({float: 1900867, str: 10474})
```

Next step is to convert the values to numeric

In [12]:
```
string_values = df_Flights[df_Flights['AIR_TIME'].apply(lambda x: isinstance(x, str))]['AIR_TIME']
print(string_values)
# This line first filters 'df_Flights' DataFrame to select only those rows where 'AIR_TIME' is of string type.
# It uses the 'apply' method with a lambda function to check if each element in 'AIR_TIME' column is an instance of
# Then, it extracts the 'AIR_TIME' column from these filtered rows.
# The resulting Series 'string_values' contains all 'AIR_TIME' values that are strings.
# Finally, it prints the 'string_values' Series to the console.
```

```
1900544    39.0
1900545    36.0
1900546    31.0
1900547    33.0
1900548    32.0
           ...
1911336     112
1911337     106
1911338     106
1911339     112
1911340     117
Name: AIR_TIME, Length: 10474, dtype: object
```

In [13]:
```
# This line converts the 'AIR_TIME' column in the 'df_Flights' DataFrame to a numeric type.
# The 'pd.to_numeric' function is used for the conversion. If it encounters any value in 'AIR_TIME' that cannot be c
# the parameter 'errors='coerce'' instructs it to replace such non-convertible values with NaN (Not a Number).
# As a result, 'AIR_TIME' will have all its values as numeric (integers or floats), and any non-numeric strings or o
df_Flights['AIR_TIME'] = pd.to_numeric(df_Flights['AIR_TIME'], errors='coerce')
```

The next step is to impute the null values with median the reason why I choose median to update null values is since the data has outliers and median is also less sensitive to outliers.

In [14]:
```
# This function iterates through all columns of a DataFrame 'df', replacing NaN values in numeric columns with thei

import pandas as pd

def replace_na_with_median(df):
    # Calculate the median for each column
    medians = df.median()

    # Replace NaNs with the median of each column
    df_filled = df.fillna(medians)

    return df_filled

#Decided to use median since the data is skewed or may have outliers hence median
```

In [15]:
```
df_Flights = replace_na_with_median(df_Flights)
# Apply the 'replace_na_with_median' function to the 'df_Flights' DataFrame to replace NaNs in numeric columns with
```

```
/var/folders/_q/583vq_k14zq8_cnpc5x9g7700000gn/T/ipykernel_34730/484178123.py:7: FutureWarning: Dropping of nuisanc
e columns in DataFrame reductions (with 'numeric_only=None') is deprecated; in a future version this will raise Typ
eError.  Select only valid columns before calling the reduction.
  medians = df.median()
```

In [16]:
```
# Calculate and print the count of null values in each column of the 'df_Flights' DataFrame.

null_values = df_Flights.isnull().sum()
print("Null values in each column:")
print(null_values)
```

```
Null values in each column:
FL_DATE                 0
OP_CARRIER              0
TAIL_NUM            12111
OP_CARRIER_FL_NUM       0
ORIGIN_AIRPORT_ID       0
ORIGIN                  0
ORIGIN_CITY_NAME        0
DEST_AIRPORT_ID         0
DESTINATION             0
DEST_CITY_NAME          0
DEP_DELAY               0
```

In [4]:
```
#Data cleaning to remove null values
df_Flights = df_Flights.dropna()
```

In [5]:
```
df_Flights
```

Out[5]:

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	D
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	
...	...	...	...	...	...	...	...	...	...	
1915881	3/23/19	AA	N903NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915882	3/24/19	AA	N965AN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915883	3/25/19	AA	N979NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915884	3/26/19	AA	N872NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915885	3/27/19	AA	N945AN	1433	15370	TUL	Tulsa, OK	11057	CLT	

1858595 rows × 16 columns

Next step we create a df where airtime values are converted to numeric values

In [19]:
```
# Filter the 'df_Flights' DataFrame to include only rows where 'AIR_TIME' can be successfully converted to numeric v

df_Flights[pd.to_numeric(df_Flights['AIR_TIME'], errors='coerce').notnull()]
```

Out[19]:

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	D
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	
...	...	...	...	...	...	...	...	...	...	
1911336	3/23/19	AA	N903NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1911337	3/24/19	AA	N965AN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1911338	3/25/19	AA	N979NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1911339	3/26/19	AA	N872NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1911340	3/27/19	AA	N945AN	1433	15370	TUL	Tulsa, OK	11057	CLT	

1899168 rows × 16 columns

- The next step is to filter out the canceled flights

```
In [14]: #Filtering out cancelling flights
df_active_flights = df_Flights[df_Flights['CANCELLED'] == 0]
df_active_flights
```

Out[14]:

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	D
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	
...	...	...	...	...	...	...	...	...	...	
1915881	3/23/19	AA	N903NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915882	3/24/19	AA	N965AN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915883	3/25/19	AA	N979NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915884	3/26/19	AA	N872NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915885	3/27/19	AA	N945AN	1433	15370	TUL	Tulsa, OK	11057	CLT	

The next step is to create the correlation and descriptive statistics

```
In [22]: df_active_flights.corr()
```

Out[22]:

	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	DEP_DELAY	ARR_DELAY	CANCELLED	AIR_TIME	OCCUPANCY_RATE
ORIGIN_AIRPORT_ID	1.000000	0.019837	0.001447	0.003001	NaN	0.048812	-0.000369
DEST_AIRPORT_ID	0.019837	1.000000	0.006987	0.013156	NaN	0.099137	-0.001098
DEP_DELAY	0.001447	0.006987	1.000000	0.959487	NaN	0.008864	-0.001100
ARR_DELAY	0.003001	0.013156	0.959487	1.000000	NaN	0.004915	-0.000962
CANCELLED	NaN	NaN	NaN	NaN	NaN	NaN	NaN
AIR_TIME	0.048812	0.099137	0.008864	0.004915	NaN	1.000000	-0.000131
OCCUPANCY_RATE	-0.000369	-0.001098	-0.001100	-0.000962	NaN	-0.000131	1.000000

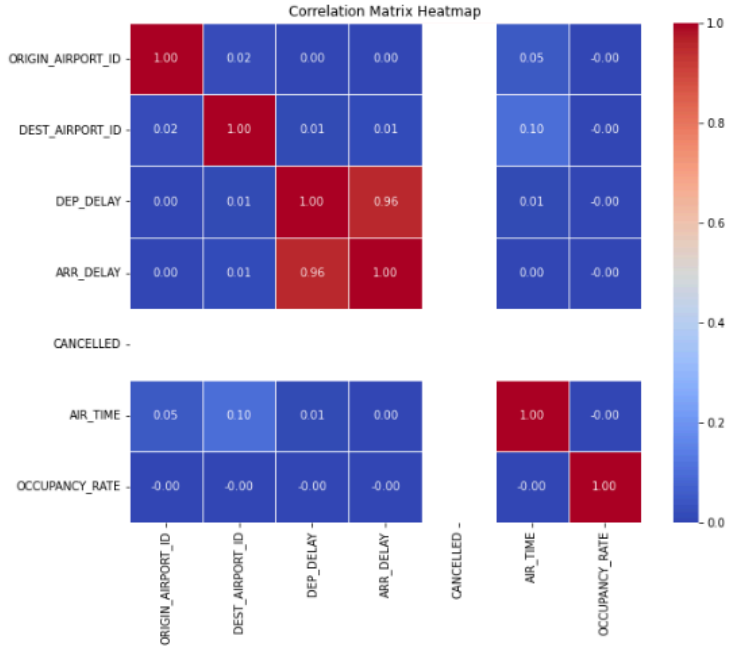
```
In [23]: df_active_flights.describe()
```

Out[23]:

	ORIGIN_AIRPORT_ID	DEST_AIRPORT_ID	DEP_DELAY	ARR_DELAY	CANCELLED	AIR_TIME	OCCUPANCY_RATE
count	1.859801e+06	1.859801e+06	1.859801e+06	1.859801e+06	1859801.0	1.859801e+06	1.859801e+06
mean	1.268601e+04	1.268615e+04	1.078100e+01	5.633136e+00	0.0	1.092961e+02	6.502111e-01
std	1.521702e+03	1.521887e+03	5.009254e+01	5.237371e+01	0.0	7.017855e+01	2.019938e-01
min	1.013500e+04	1.013500e+04	-6.300000e+01	-9.400000e+01	0.0	-1.210000e+02	3.000000e-01
25%	1.129200e+04	1.129200e+04	-6.000000e+00	-1.500000e+01	0.0	5.900000e+01	4.800000e-01
50%	1.288900e+04	1.288900e+04	-2.000000e+00	-6.000000e+00	0.0	9.100000e+01	6.500000e-01
75%	1.405700e+04	1.405700e+04	7.000000e+00	8.000000e+00	0.0	1.390000e+02	8.200000e-01
max	1.621800e+04	1.621800e+04	2.941000e+03	2.923000e+03	0.0	2.222000e+03	1.000000e+00

```
In [24]: corr_matrix = df_active_flights.corr()
```

In [25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
#The correlation matrix below indicates a strong positive correlation
#(0.96) between DEP_DELAY and ARR_DELAY,
#and a moderate correlation between DEST_AIRPORT_ID and AIR_TIME,
#with other variables showing weak or negligible correlations.
```



Airport Codes

The first step is to load the Airport_Codes dataset

1858595 rows × 16 columns

```
In [12]: #Loading Airport Codes
         df_Airport_Codes = pd.read_csv("/Users/jayantvelichety/Desktop/Capital One Data Challenge Jayant Velichety/DA-Airlin
```

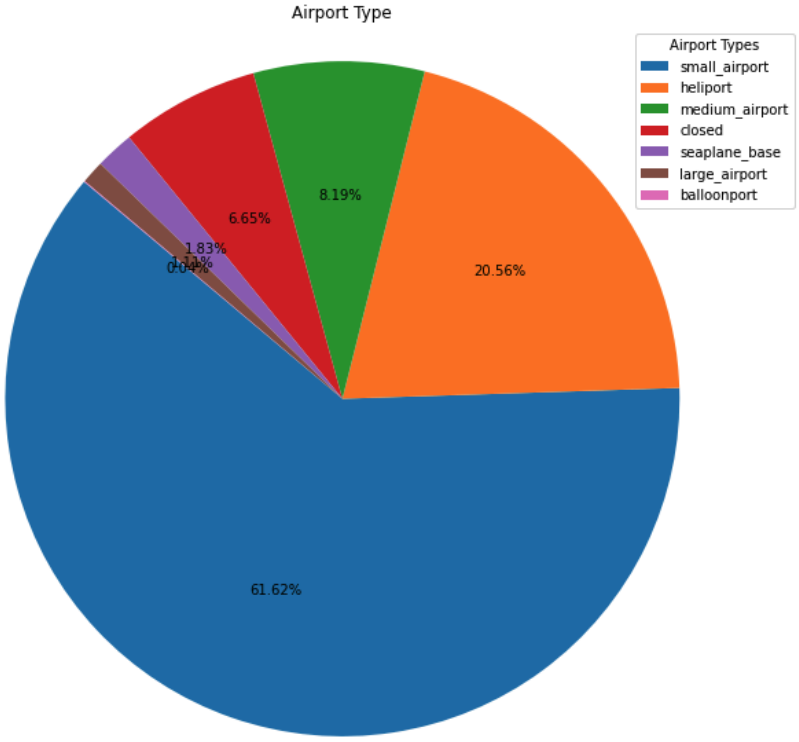
```
In [13]: df_Airport_Codes
```

Out[13]:

	TYPE	NAME	ELEVATION_FT	CONTINENT	ISO_COUNTRY	MUNICIPALITY	IATA_CODE	COORDINATES
0	heliport	Total Rf Heliport	11.0	NaN	US	Bensalem	NaN	-74.93360137939453, 40.07080078125
1	small_airport	Aero B Ranch Airport	3435.0	NaN	US	Leoti	NaN	-101.473911, 38.704022
2	small_airport	Lowell Field	450.0	NaN	US	Anchor Point	NaN	-151.695999146, 59.94919968
3	small_airport	Epps Airpark	820.0	NaN	US	Harvest	NaN	-86.77030181884766, 34.86479949951172
4	closed	Newport Hospital & Clinic Heliport	237.0	NaN	US	Newport	NaN	-91.254898, 35.6087
...	...	...	...	...	...	...	...	...
55364	medium_airport	Yingkou Lanqi Airport	0.0	AS	CN	Yingkou	YKH	122.3586, 40.542524
55365	medium_airport	Shenyang Dongta Airport	NaN	AS	CN	Shenyang	NaN	123.49600219726562, 41.784400939941406
55366	heliport	Sealand Helipad	40.0	EU	GB	Sealand	NaN	1.4825, 51.894444
55367	small_airport	Glorioso Islands Airstrip	11.0	AF	TF	Grande Glorieuse	NaN	47.296388888900005, -11.584277777799999
55368	small_airport	Satsuma IÅ□jima Airport	338.0	AS	JP	Mishima-Mura	NaN	130.270556, 30.784722

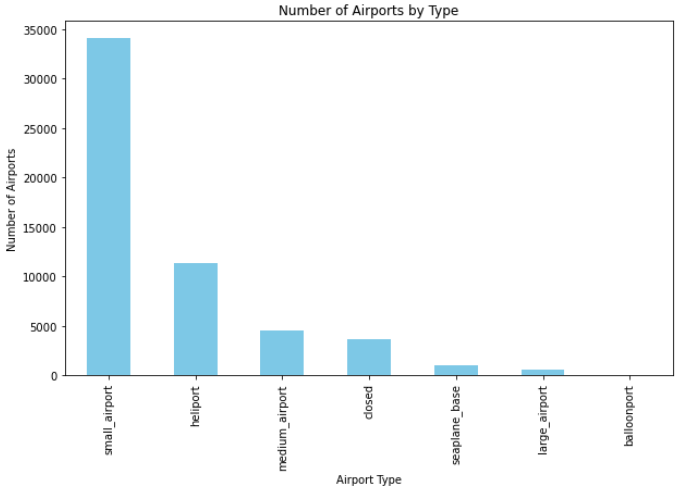
The next step I plotted a pie chart to understand the percentage split of the airport type

```
#Plotting Pie chart by Airport type to show percentage split
import matplotlib.pyplot as plt

# Pie chart of Airport Types before filtering
airport_type = df_Airport_Codes['TYPE'].value_counts()

# Distribution of Airports by country
plt.figure(figsize=(12, 10))
plt.pie(airport_type, autopct='%1.2f%%', startangle=140)

# Set aspect ratio to be equal, ensuring that pie is drawn as a circle
plt.axis('equal')

# Move legend to the top right corner
plt.legend(airport_type.index, title="Airport Types", loc="upper right")

plt.title('Airport Type')
plt.show()
```


Airport Type
Airport Types
small_airport
heliport
medium_airport
closed
seaplane_base
large_airport
balloonport
8.19%
6.65%
1.83%
20.56%
61.62%

The next step is to plot the bar chart to understand the number of total airports in each type

```
In [10]: #Bar chart of Number of Airports
airport_type_counts = df_Airport_Codes['TYPE'].value_counts()
#No of aiports by type bar chart
plt.figure(figsize=(10, 6))
airport_type_counts.plot(kind='bar', color='skyblue')
plt.title('Number of Airports by Type')
plt.xlabel('Airport Type')
plt.ylabel('Number of Airports')
plt.show()
```



Filtering the airport based on the country = ‘US’ and TYPE = ‘Medium’ and ‘Large’, since we are analyzing for US market and the airline wants to enter the US market.

```
In [11]: #Filtering based on country, medium and large
df_Airport_Codes_US = df_Airport_Codes[(df_Airport_Codes['TYPE'].isin(['medium_airport', 'large_airport'])) &
                                       (df_Airport_Codes['ISO_COUNTRY'] == 'US')]
df_Airport_Codes_US
```

Out[11]:

	TYPE	NAME	ELEVATION_FT	CONTINENT	ISO_COUNTRY	MUNICIPALITY	IATA_CODE	COORDINATES
6194	medium_airport	Aleknagik / New Airport	66.0	NaN	US	Aleknagik	WKK	-158.617996216, 59.2826004028
25963	medium_airport	South Alabama Regional At Bill Benton Field Ai...	310.0	NaN	US	Andalusia/Opp	NaN	-86.393799, 31.3088
26143	medium_airport	Lehigh Valley International Airport	393.0	NaN	US	Allentown	ABE	-75.44080352783203, 40.652099609375
26144	medium_airport	Abilene Regional Airport	1791.0	NaN	US	Abilene	ABI	-99.68190002440001, 32.4113006592
26145	large_airport	Albuquerque International Sunport	5355.0	NaN	US	Albuquerque	ABQ	-106.609001, 35.040199
...	...	...	...	...	...	...	...	...
49351	medium_airport	Jim's Private Airport	890.0	NaN	US	Conyers	NaN	-84.14440155, 33.59790039
49519	medium_airport	Yuma Auxiliary AAF #2	NaN	NaN	US	NaN	NaN	-114.511383, 32.548984
49569	large_airport	atl	NaN	NaN	US	NaN	NaN	-84.375, 33.137551
49837	medium_airport	Williston Basin International Airport (U.C.)	2344.0	NaN	US	Williston	NaN	-103.748797, 48.258387
50008	medium_airport	34S Airport	NaN	NaN	US	NaN	NaN	-16.875, 19.145168

858 rows × 8 columns

The next step is to check null values and drop in IATA code

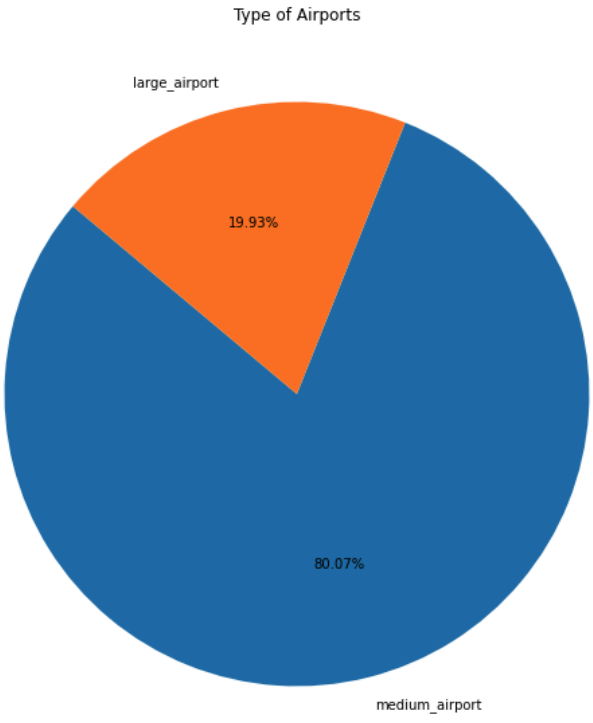
```
In [32]: #Checking null values
         null_values = df_Airport_Codes_US.isnull().sum()
         print("Null values in each column:")
         print(null_values)
```

```
Null values in each column:
TYPE              0
NAME              0
ELEVATION_FT      3
CONTINENT       858
ISO_COUNTRY       0
MUNICIPALITY      3
IATA_CODE        37
COORDINATES       0
dtype: int64
```

```
In [33]: # Assuming df_Airport_Codes_US is already loaded
         # If not, you would need to load it from its source

         # Dropping rows where 'IATA code' column has null values
         df_Airport_Codes_US = df_Airport_Codes_US.dropna(subset=['IATA_CODE'])

         # Display the cleaned DataFrame
         print(df_Airport_Codes_US)
```

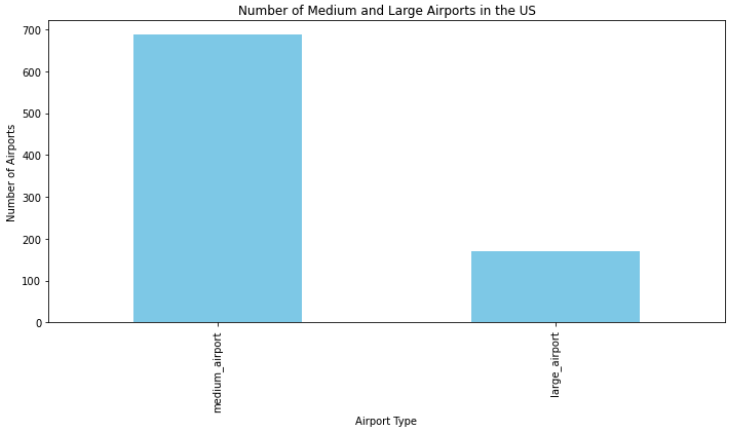
The next step in the process is to plot pie charts for medium and large airports to understand percentage split of medium and large airports

```
In [18]: airport_type = df_Airport_Codes_US['TYPE'].value_counts()
         #Percentage of airports post filtering
         plt.figure(figsize=(10, 10))
         plt.pie(airport_type, labels= airport_type.index, autopct='%1.2f%%', startangle=140)
         plt.title('Type of Airports')
         plt.show()
```



The next step is to plot bar graph to show the number of medium and large airports in US

```
In [19]: airport_type_counts = df_Airport_Codes_US['TYPE'].value_counts()
         #Bar chart to show number of medium and large airports in USA
         plt.figure(figsize=(10, 6))
         airport_type_counts.plot(kind='bar', color='skyblue')
         plt.title('Number of Medium and Large Airports in the US')
         plt.xlabel('Airport Type')
         plt.ylabel('Number of Airports')
         plt.tight_layout()
         plt.show()
```



Tickets

The next step is to load the tickets dataset

```
In [32]: #Importing Tickets
         df_Tickets = pd.read_csv("/Users/jayantvelichety/Desktop/Capital One Data Challenge Jayant Velichety/DA-Airline-Data
         df_Tickets
```

Out[32]:

	ITIN_ID	YEAR	QUARTER	ORIGIN	ORIGIN_COUNTRY	ORIGIN_STATE_ABR	ORIGIN_STATE_NM	ROUNDTRIP	REPORTING_CARRIER	PASSEN
0	201912723049	2019	1	ABI	US	TX	Texas	1.0	MQ	
1	201912723085	2019	1	ABI	US	TX	Texas	1.0	MQ	
2	201912723491	2019	1	ABI	US	TX	Texas	1.0	MQ	
3	201912723428	2019	1	ABI	US	TX	Texas	1.0	MQ	
4	201912723509	2019	1	ABI	US	TX	Texas	0.0	MQ	
...	...	...	...	...	...	...	...	...	...	
1167280	201911284909	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167281	201911284959	2019	1	YAK	US	AK	Alaska	1.0	AS	
1167282	201911284940	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167283	201911284914	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167284	201911284952	2019	1	YAK	US	AK	Alaska	1.0	AS	

1167285 rows × 12 columns

The next step is to check the duplicates and null values and drop them

In [37]:
```
df_Tickets.duplicated()
#Checking for duplicate values
```

Out[37]:
```
0          False
1          False
2          False
3          False
4          False
           ...
1167280     True
1167281     True
1167282     True
1167283     True
1167284     True
Length: 1167285, dtype: bool
```

In [38]:
```
df_Tickets = df_Tickets.drop_duplicates()
#drop duplicates
```

In [39]:
```
df_Tickets
```

Out[39]:

	ITIN_ID	YEAR	QUARTER	ORIGIN	ORIGIN_COUNTRY	ORIGIN_STATE_ABR	ORIGIN_STATE_NM	ROUNDTRIP	REPORTING_CARRIER	PASSEN
0	201912723049	2019	1	ABI	US	TX	Texas	1.0	MQ	
1	201912723085	2019	1	ABI	US	TX	Texas	1.0	MQ	
2	201912723491	2019	1	ABI	US	TX	Texas	1.0	MQ	
3	201912723428	2019	1	ABI	US	TX	Texas	1.0	MQ	
4	201912723509	2019	1	ABI	US	TX	Texas	0.0	MQ	
...	...	...	...	...	...	...	...	...	...	
1115066	201913420834	2019	1	YUM	US	AZ	Arizona	1.0	OO	
1115067	201913421041	2019	1	YUM	US	AZ	Arizona	1.0	OO	
1115068	201913420763	2019	1	YUM	US	AZ	Arizona	0.0	OO	
1115069	201915099580	2019	1	YUM	US	AZ	Arizona	0.0	YV	
1115070	201913421222	2019	1	YUM	US	AZ	Arizona	0.0	OO	

1095387 rows × 12 columns

In [40]:
```
#Checking null values in each column
null_values = df_Tickets.isnull().sum()
print("Null values in each column:")
print(null_values)
```

```
Null values in each column:
ITIN_ID                 0
YEAR                    0
QUARTER                 0
ORIGIN                  0
ORIGIN_COUNTRY          0
ORIGIN_STATE_ABR        0
ORIGIN_STATE_NM         0
```

Create a variable k, looping statement through k, understand the number of strings and float. After which we impute the median values to the rows having null values.

In [41]:
```
# Extract and store the values from the 'ITIN_FARE' column of the df_Tickets DataFrame into the variable k
k = df_Tickets['ITIN_FARE'].values
```

In [42]:
```
# Loop through 'k', appending the type of each element to the list 's'.

s=[]
for i in k:
  #print(type(i))
  s.append(type(i))
```

In [43]:
```

# Convert the list 's' into a Counter object to count the occurrences of each element's type.
from collections import Counter
s = Counter(s)
```

In [44]:
```
# 's' now represents a Counter object containing the frequency of each unique element type from the original list.
s
```

Out[44]:
```
Counter({str: 1094600, float: 787})
```

In [45]:
```
string_values = df_Tickets[df_Tickets['ITIN_FARE'].apply(lambda x: isinstance(x, str))]['ITIN_FARE']
print(string_values)
# Extract and print all 'ITIN_FARE' values from 'df_Tickets' that are stored as strings.
```

```
0          736.0
1          570.0
2          564.0
3          345.0
4          309.0
           ...
1115065    958.0
1115066    705.0
1115067    580.0
1115069    117.0
1115070    366.0
Name: ITIN_FARE, Length: 1094600, dtype: object
```

In [46]:
```
# Convert the 'ITIN_FARE' column in 'df_Tickets' to a numeric type, setting non-convertible values to NaN.
df_Tickets['ITIN_FARE'] = pd.to_numeric(df_Tickets['ITIN_FARE'], errors='coerce')
```

```
/var/folders/_q/583vq_k14zq8_cnpc5x9g7700000gn/T/ipykernel_34730/1836836365.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_Tickets['ITIN_FARE'] = pd.to_numeric(df_Tickets['ITIN_FARE'], errors='coerce')
```

In [47]:
```
#Replace null values with median, since median helps us with identifying outliers
df_Tickets = replace_na_with_median(df_Tickets)
```

In [49]:
```
#Correlation
df_Tickets.corr()
```

Out[49]:

	ITIN_ID	YEAR	QUARTER	ROUNDTRIP	PASSENGERS	ITIN_FARE
ITIN_ID	1.000000	NaN	NaN	0.030488	0.020660	-0.023304
YEAR	NaN	NaN	NaN	NaN	NaN	NaN
QUARTER	NaN	NaN	NaN	NaN	NaN	NaN
ROUNDTRIP	0.030488	NaN	NaN	1.000000	-0.028273	0.135988
PASSENGERS	0.020660	NaN	NaN	-0.028273	1.000000	-0.054376
ITIN_FARE	-0.023304	NaN	NaN	0.135988	-0.054376	1.000000

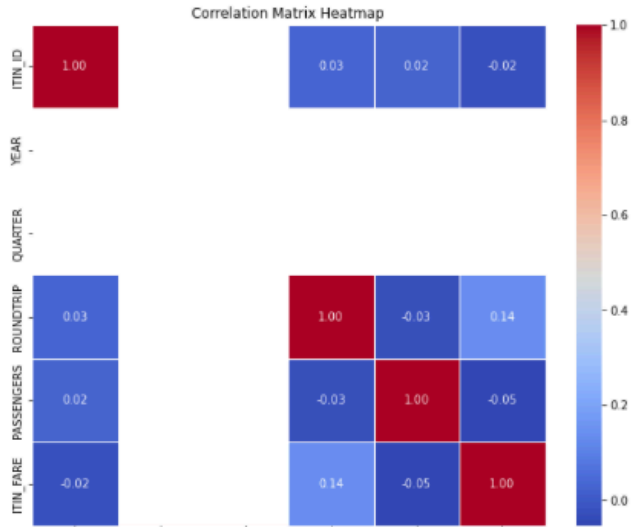
In [50]:
```
#Descriptive stats
df_Tickets.describe()
```

Out[50]:

	ITIN_ID	YEAR	QUARTER	ROUNDTRIP	PASSENGERS	ITIN_FARE
count	1.095387e+06	1095387.0	1095387.0	1.095387e+06	1.095387e+06	1.095387e+06
mean	1.707017e+11	2019.0	1.0	6.034726e-01	2.096440e+00	4.057071e+02
std	6.910635e+10	0.0	0.0	4.891765e-01	5.960456e+00	6.098009e+02
min	2.019120e+05	2019.0	1.0	0.000000e+00	1.000000e+00	0.000000e+00
25%	2.019114e+11	2019.0	1.0	0.000000e+00	1.000000e+00	2.060000e+02
50%	2.019128e+11	2019.0	1.0	1.000000e+00	1.000000e+00	3.460000e+02
75%	2.019140e+11	2019.0	1.0	1.000000e+00	1.000000e+00	5.250000e+02
max	2.019153e+11	2019.0	1.0	1.000000e+00	7.690000e+02	3.774000e+05

In [51]:
```
corr_matrix = df_Tickets.corr()
```

In [52]:
```
#Correlation graph plot
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```



The next step is I have taken the assumption of the ticket prices to be between $30 and $1,000 since there were tickets at around $37,000, and other astronomical figures which are not possible for a route.

In [35]:
```
import pandas as pd

# Assuming df is your DataFrame
df = df_Tickets
# Remove '$' from a specific column
df['ITIN_FARE'] = df['ITIN_FARE'].replace('[\$,]', '', regex=True).astype(float)

# Filter values within the range [30, 1000]
df_Tickets = df[(df['ITIN_FARE'] >= 30) & (df['ITIN_FARE'] <= 1000)]
```

In [36]:
```
#Filtered prices set minimum as $30 and Max $1,000
df_Tickets
```

Out[36]:

	ITIN_ID	YEAR	QUARTER	ORIGIN	ORIGIN_COUNTRY	ORIGIN_STATE_ABR	ORIGIN_STATE_NM	ROUNDTRIP	REPORTING_CARRIER	PASSEN
0	201912723049	2019	1	ABI	US	TX	Texas	1.0	MQ	
1	201912723085	2019	1	ABI	US	TX	Texas	1.0	MQ	
2	201912723491	2019	1	ABI	US	TX	Texas	1.0	MQ	
3	201912723428	2019	1	ABI	US	TX	Texas	1.0	MQ	
4	201912723509	2019	1	ABI	US	TX	Texas	0.0	MQ	
...	...	...	...	...	...	...	...	...	...	
1167280	201911284909	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167281	201911284959	2019	1	YAK	US	AK	Alaska	1.0	AS	
1167282	201911284940	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167283	201911284914	2019	1	YAK	US	AK	Alaska	0.0	AS	
1167284	201911284952	2019	1	YAK	US	AK	Alaska	1.0	AS	

1047695 rows × 12 columns

The next step is to focus on the roundtrip data only for the flights since that is the ask in order to know what is top 10 busiest route round trips

In [55]:
```
# Filter 'df_Tickets' to include only rows where 'ROUNDTRIP' is equal to 1.

df_Tickets = df_Tickets[df_Tickets['ROUNDTRIP'] == 1]
```

In [56]:
```
df_Tickets
```

Out[56]:

	ITIN_ID	YEAR	QUARTER	ORIGIN	ORIGIN_COUNTRY	ORIGIN_STATE_ABR	ORIGIN_STATE_NM	ROUNDTRIP	REPORTING_CARRIER	PASSE
0	201912723049	2019	1	ABI	US	TX	Texas	1.0	MQ	
1	201912723085	2019	1	ABI	US	TX	Texas	1.0	MQ	
2	201912723491	2019	1	ABI	US	TX	Texas	1.0	MQ	
3	201912723428	2019	1	ABI	US	TX	Texas	1.0	MQ	
11	201912723447	2019	1	ABI	US	TX	Texas	1.0	MQ	
...	...	...	...	...	...	...	...	...	...	
1115053	201915099493	2019	1	YUM	US	AZ	Arizona	1.0	YV	
1115057	201913421100	2019	1	YUM	US	AZ	Arizona	1.0	OO	
1115058	201913421044	2019	1	YUM	US	AZ	Arizona	1.0	OO	
1115066	201913420834	2019	1	YUM	US	AZ	Arizona	1.0	OO	
1115067	201913421041	2019	1	YUM	US	AZ	Arizona	1.0	OO	

591437 rows × 12 columns

The first step we create a roundtrip column, we do the round trip counts and also do it for reverse direction, to be noted round trip is the flight goes from origin to destination and back to origin.

In [39]:
```
#Create column RoundTrip Route
df_active_flights['ROUNDTRIPROUTE'] = df_active_flights.apply(lambda x: '-'.join(sorted([x['ORIGIN'], x['DESTINATION
df_active_flights
```

Out[39]:

	FL_DATE	OP_CARRIER	TAIL_NUM	OP_CARRIER_FL_NUM	ORIGIN_AIRPORT_ID	ORIGIN	ORIGIN_CITY_NAME	DEST_AIRPORT_ID	DESTINATION	D
0	2019-03-02	WN	N955WN	4591	14635	RSW	Fort Myers, FL	11042	CLE	
1	2019-03-02	WN	N8686A	3231	14635	RSW	Fort Myers, FL	11066	CMH	
2	2019-03-02	WN	N201LV	3383	14635	RSW	Fort Myers, FL	11066	CMH	
3	2019-03-02	WN	N413WN	5498	14635	RSW	Fort Myers, FL	11066	CMH	
4	2019-03-02	WN	N7832A	6933	14635	RSW	Fort Myers, FL	11259	DAL	
...	...	...	...	...	...	...	...	...	...	
1915881	3/23/19	AA	N903NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915882	3/24/19	AA	N965AN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915883	3/25/19	AA	N979NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915884	3/26/19	AA	N872NN	1433	15370	TUL	Tulsa, OK	11057	CLT	
1915885	3/27/19	AA	N945AN	1433	15370	TUL	Tulsa, OK	11057	CLT	

1858595 rows × 17 columns

In [59]:
```
# Group 'df_active_flights' by 'ORIGIN' and 'DESTINATION', count occurrences, and reset index to form 'df_round_trip

round_trip_counts = df_active_flights.groupby(['ORIGIN','DESTINATION']).size()
df_round_trip = round_trip_counts.reset_index(name='count')
```

In [60]:
```
df_round_trip
```

Out[60]:

	ORIGIN	DESTINATION	count
0	ABE	ATL	217
1	ABE	CLT	251
2	ABE	DTW	248
3	ABE	FLL	20
4	ABE	ORD	159
...	...	...	...
5912	YAK	CDV	83
5913	YAK	JNU	88
5914	YKM	SEA	305
5915	YUM	DFW	28
5916	YUM	PHX	347

5917 rows × 3 columns

In [61]:
```
## Add a 'reverse_count' column to 'df_round_trip', counting reverse direction trips from 'DESTINATION' to 'ORIGIN'.
df_round_trip['reverse_count'] = df_round_trip.apply(lambda row: round_trip_counts.get((row['DESTINATION'], row['ORI
```

In [62]:
```
df_round_trip
```

Out[62]:

	ORIGIN	DESTINATION	count	reverse_count
0	ABE	ATL	217	217
1	ABE	CLT	251	251
2	ABE	DTW	248	249
3	ABE	FLL	20	20
4	ABE	ORD	159	161
...	...	...	...	...
5912	YAK	CDV	83	88
5913	YAK	JNU	88	85
5914	YKM	SEA	305	305
5915	YUM	DFW	28	29
5916	YUM	PHX	347	347

5917 rows × 4 columns

und Trip Routes fo

Checking the total number of trips

In [63]:
```
# Calculate 'round_trips' in 'df_round_trip' as the minimum of 'count' and '

df_round_trip['round_trips'] = df_round_trip[['count', 'reverse_count']].min(axis=1)
```

In [64]:
```
df_round_trip
```

Out[64]:

	ORIGIN	DESTINATION	count	reverse_count	round_trips
0	ABE	ATL	217	217	217
1	ABE	CLT	251	251	251
2	ABE	DTW	248	249	248
3	ABE	FLL	20	20	20
4	ABE	ORD	159	161	159
...	...	...	...	...	...
5912	YAK	CDV	83	88	83
5913	YAK	JNU	88	85	85
5914	YKM	SEA	305	305	305
5915	YUM	DFW	28	29	28
5916	YUM	PHX	347	347	347

5917 rows × 5 columns

In [65]:
```
# Create a 'ROUTE' column in 'df_round_trip', sort by 'round_trips', and return the highest value of 'round_trips' a

df_round_trip['ROUTE'] = df_round_trip.apply(lambda row: '-'.join([row['ORIGIN'], row['DESTINATION']]), axis=1)
round_trip_routes = df_round_trip.sort_values(by='round_trips', ascending=False)
#return
float(df_round_trip['round_trips'].values[0])
```

Out[65]:
```
217.0
```

In [66]:
```
round_trip_routes
```

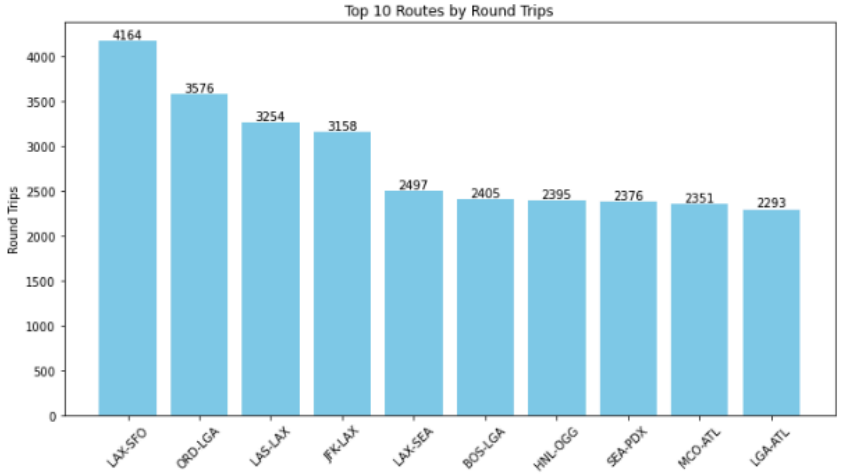
Out[66]:

	ORIGIN	DESTINATION	count	reverse_count	round_trips	ROUTE
3010	LAX	SFO	4164	4176	4164	LAX-SFO
5284	SFO	LAX	4176	4164	4164	SFO-LAX
4066	ORD	LGA	3580	3576	3576	ORD-LGA
3115	LGA	ORD	3576	3580	3576	LGA-ORD
2869	LAS	LAX	3254	3257	3254	LAS-LAX
...	...	...	...	...	...	...
1101	CPR	RAP	2	0	0	CPR-RAP
5845	TWF	DEN	26	0	0	TWF-DEN
3416	MEI	DFW	89	0	0	MEI-DFW
5846	TWF	LAX	28	0	0	TWF-LAX
5380	SJC	MCI	4	0	0	SJC-MCI

Next step is we plot the bar graph to see the top 10 busiest round trip routes in the country, the busiest being LAX-SFO

```
In [69]: # Select the top 10 rows with the largest values in 'round_trips' from 'round_trip_routes' to create 'top_10'.

         top_10 = round_trip_routes.nlargest(10, 'round_trips')
```

```
In [70]: # Plot a bar chart of the top 10 routes by round trips, with labels on each bar, using 'ROUTE' and 'round_trips' from

         mport matplotlib.pyplot as plt

         # Assuming top_10['ROUTE'] and top_10['round_trips'] are your data
         lt.figure(figsize=(10, 6))
         ars = plt.bar(top_10['ROUTE'], top_10['round_trips'], color='skyblue')

         # Add labels
         or bar in bars:
             yval = bar.get_height()
             plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), va='bottom', ha='center')

         lt.xlabel('Route')
         lt.ylabel('Round Trips')
         lt.title('Top 10 Routes by Round Trips')
         lt.xticks(rotation=45)
         lt.tight_layout()
         lt.show()
```



2) Top 10 most profitable round trip routes

The first step is to understand and create subset from each of the columns, we choose specific values pertinent to our analysis

```
In [76]: # Create a subset 'df_active_flights_subset' from 'df_active_flights' with selected columns for specific flight info
         df_active_flights_subset = df_active_flights[['FL_DATE','ORIGIN', 'DESTINATION', 'OCCUPANCY_RATE', 'DISTANCE', 'DEP_
```

```
In [77]: df_Airport_Codes_US_subset = df_Airport_Codes_US[['TYPE', 'IATA_CODE']]
```

```
In [78]: df_Tickets_subset = df_Tickets[['ORIGIN', 'DESTINATION', 'ITIN_FARE']]
```

```
In [79]: df_Tickets_subset
```

We start of by performing JOINS

591437 rows × 3 columns

In [80]:
```
#This line of code performs a left join of df_active_flights_subset with df_Airport_Codes_US_subset,
#matching the ORIGIN column from the former with the IATA_CODE column from the latter,
#and stores the result in df_profit_merge_origin.
df_profit_merge_origin=df_active_flights_subset.merge(df_Airport_Codes_US_subset, left_on= 'ORIGIN',right_on = 'IATA
```

In [81]:
```
df_profit_merge_origin
```

0	2019-03-02	RSW	CLE	0.970000	1025.0	-8.0	-6.0	large_airport	RSW
1	2019-03-02	RSW	CMH	0.550000	930.0	1.0	5.0	large_airport	RSW
2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	RSW
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	RSW
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	RSW
...	...	...	...	...	...	...	...	...	...
1859796	3/23/19	TUL	CLT	0.794884	****	-9.0	-6.0	large_airport	TUL
1859797	3/24/19	TUL	CLT	0.538399	****	-2.0	-1.0	large_airport	TUL
1859798	3/25/19	TUL	CLT	0.955579	****	-8.0	-25.0	large_airport	TUL
1859799	3/26/19	TUL	CLT	0.595344	****	-9.0	-6.0	large_airport	TUL
1859800	3/27/19	TUL	CLT	0.350192	****	-8.0	5.0	large_airport	TUL

1859801 rows × 9 columns

In [82]:
```
import pandas as pd

# Assuming df_profit_merge_origin is already defined
df_profit_merge_origin = df_profit_merge_origin.rename(columns={'TYPE': 'TYPE_ORIGIN'})

# Now the 'TYPE' column in df_profit_merge_origin is renamed to 'TYPE_ORIGIN'
```

Renaming type columns as TYPE_ORIGIN

In [82]:
```
import pandas as pd

# Assuming df_profit_merge_origin is already defined
df_profit_merge_origin = df_profit_merge_origin.rename(columns={'TYPE': 'TYPE_ORIGIN'})

# Now the 'TYPE' column in df_profit_merge_origin is renamed to 'TYPE_ORIGIN'
```

In [83]:
```
df_profit_merge_origin
```

Out[83]:

	FL_DATE	ORIGIN	DESTINATION	OCCUPANCY_RATE	DISTANCE	DEP_DELAY	ARR_DELAY	TYPE_ORIGIN	IATA_CODE
0	2019-03-02	RSW	CLE	0.970000	1025.0	-8.0	-6.0	large_airport	RSW
1	2019-03-02	RSW	CMH	0.550000	930.0	1.0	5.0	large_airport	RSW
2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	RSW
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	RSW
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	RSW
...	...	...	...	...	...	...	...	...	...
1859796	3/23/19	TUL	CLT	0.794884	****	-9.0	-6.0	large_airport	TUL
1859797	3/24/19	TUL	CLT	0.538399	****	-2.0	-1.0	large_airport	TUL
1859798	3/25/19	TUL	CLT	0.955579	****	-8.0	-25.0	large_airport	TUL
1859799	3/26/19	TUL	CLT	0.595344	****	-9.0	-6.0	large_airport	TUL
1859800	3/27/19	TUL	CLT	0.350192	****	-8.0	5.0	large_airport	TUL

1859801 rows × 9 columns

In [84]:

More joins now on Destination

In [84]:
```
#The line of code performs a left join of the df_active_flights_subset DataFrame with the df_Airport_Codes_US_subse
#DataFrame, matching the DESTINATION column from the first DataFrame with the IATA_CODE column from the
#second DataFrame, and stores the resulting merged DataFrame in
#df_profit_merge_destination.
#This operation is typically used to enrich flight data with additional airport information.
df_profit_merge_destination=df_active_flights_subset.merge(df_Airport_Codes_US_subset, left_on= 'DESTINATION',right
```

In [85]:
```
df_profit_merge_destination
```

0	2019-03-02	RSW	CLE	0.970000	1025.0	-8.0	-6.0	large_airport	CLE
1	2019-03-02	RSW	CMH	0.550000	930.0	1.0	5.0	large_airport	CMH
2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	CMH
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	CMH
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	DAL
...	...	...	...	...	...	...	...	...	...
1859796	3/23/19	TUL	CLT	0.794884	****	-9.0	-6.0	large_airport	CLT
1859797	3/24/19	TUL	CLT	0.538399	****	-2.0	-1.0	large_airport	CLT
1859798	3/25/19	TUL	CLT	0.955579	****	-8.0	-25.0	large_airport	CLT
1859799	3/26/19	TUL	CLT	0.595344	****	-9.0	-6.0	large_airport	CLT
1859800	3/27/19	TUL	CLT	0.350192	****	-8.0	5.0	large_airport	CLT

1859801 rows × 9 columns

Performing the merge functions

In [86]:
```
import pandas as pd

# Assuming df_profit_merge_origin and df_profit_merge_destination are already defined
# Renaming the IATA_CODE and TYPE columns for clarity
df_profit_merge_origin = df_profit_merge_origin.rename(columns={"IATA_CODE": "IATA_CODE_ORIGIN", "TYPE": "TYPE_ORIGI
df_profit_merge_destination = df_profit_merge_destination.rename(columns={"IATA_CODE": "IATA_CODE_DESTINATION", "TYP

# Performing the merge
df_merged = pd.merge(df_profit_merge_origin, df_profit_merge_destination, on=["FL_DATE", "ORIGIN", "DESTINATION", "O

# The resulting dataframe 'merged_df' will have combined data from both the dataframes
```

In [87]:
```
df_merged
```

Out[87]:

	FL_DATE	ORIGIN	DESTINATION	OCCUPANCY_RATE	DISTANCE	DEP_DELAY	ARR_DELAY	TYPE_ORIGIN	IATA_CODE_ORIGIN	TYPE_DESTINATION
0	2019-03-02	RSW	CLE	0.970000	1025.0	-8.0	-6.0	large_airport	RSW	large_airport
1	2019-03-02	RSW	CMH	0.550000	930.0	1.0	5.0	large_airport	RSW	large_airport
2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	RSW	large_airport
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	RSW	large_airport
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	RSW	large_airport
...	...	...	...	...	...	...	...	...	...	...
1860124	3/23/19	TUL	CLT	0.794884	****	-9.0	-6.0	large_airport	TUL	large_airport
1860125	3/24/19	TUL	CLT	0.538399	****	-2.0	-1.0	large_airport	TUL	large_airport
1860126	3/25/19	TUL	CLT	0.955579	****	-8.0	-25.0	large_airport	TUL	large_airport
1860127	3/26/19	TUL	CLT	0.595344	****	-9.0	-6.0	large_airport	TUL	large_airport
1860128	3/27/19	TUL	CLT	0.350192	****	-8.0	5.0	large_airport	TUL	large_airport

1860129 rows × 11 columns

Adding ITIN fare

```
In [88]: df_Tickets_subset_unique = df_Tickets_subset.drop_duplicates(subset=['ORIGIN', 'DESTINATION'])

         # Now perform the merge
         df_merged_with_itin_fare = df_merged.merge(df_Tickets_subset_unique, on=['ORIGIN', 'DESTINATION'], how='left')
```

```
In [89]: df_merged_with_itin_fare
```

Out[89]:

	FL_DATE	ORIGIN	DESTINATION	OCCUPANCY_RATE	DISTANCE	DEP_DELAY	ARR_DELAY	TYPE_ORIGIN	IATA_CODE_ORIGIN	TYPE_DESTINATION
0	2019-03-02	RSW	CLE	0.970000	1025.0	-8.0	-6.0	large_airport	RSW	large_airport
1	2019-03-02	RSW	CMH	0.550000	930.0	1.0	5.0	large_airport	RSW	large_airport
2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	RSW	large_airport
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	RSW	large_airport
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	RSW	large_airport
...	...	...	...	...	...	...	...	...	...	...
1860124	3/23/19	TUL	CLT	0.794884	****	-9.0	-6.0	large_airport	TUL	large_airport
1860125	3/24/19	TUL	CLT	0.538399	****	-2.0	-1.0	large_airport	TUL	large_airport

```
In [93]: df_merged_profit = df_merged_with_itin_fare
```

```
In [95]: # Convert 'DISTANCE' column to numeric, setting errors='coerce' to turn non-numeric values to NaN
         df_merged_profit['DISTANCE'] = pd.to_numeric(df_merged_profit['DISTANCE'], errors='coerce')

         # Calculate the median of the 'DISTANCE' column, excluding NaN values
         median_distance = df_merged_profit['DISTANCE'].median()

         # Replace NaN values in 'DISTANCE' with the median
         df_merged_profit['DISTANCE'].fillna(median_distance, inplace=True)
```

```
In [96]: df_merged_profit
```

2	2019-03-02	RSW	CMH	0.910000	930.0	0.0	4.0	large_airport	RSW	large_airport
3	2019-03-02	RSW	CMH	0.670000	930.0	11.0	14.0	large_airport	RSW	large_airport
4	2019-03-02	RSW	DAL	0.620000	1005.0	0.0	-17.0	large_airport	RSW	large_airport
...	...	...	...	...	...	...	...	...	...	...
1860124	3/23/19	TUL	CLT	0.794884	612.0	-9.0	-6.0	large_airport	TUL	large_airport
1860125	3/24/19	TUL	CLT	0.538399	612.0	-2.0	-1.0	large_airport	TUL	large_airport
1860126	3/25/19	TUL	CLT	0.955579	612.0	-8.0	-25.0	large_airport	TUL	large_airport
1860127	3/26/19	TUL	CLT	0.595344	612.0	-9.0	-6.0	large_airport	TUL	large_airport
1860128	3/27/19	TUL	CLT	0.350192	612.0	-8.0	5.0	large_airport	TUL	large_airport

1860129 rows × 12 columns

Now onto adding all the costs associated such as fuel, depreciation etc.

In [98]:
```
# Assuming a fuel consumption rate and depreciation of 8+1.18 per unit of distance
fuel_rate = 8 + 1.18

# Ensure 'DISTANCE' column is numeric
df_merged_profit['DISTANCE'] = pd.to_numeric(df_merged_profit['DISTANCE'], errors='coerce')

# Create the 'fuel' column
df_merged_profit['fuel'] = df_merged_profit['DISTANCE'] * fuel_rate

# Optionally, view the first few rows to confirm the new column
print(df_merged_profit.head())
```

Defining the delay and calculating arrival and departure delay we are defining the variable and creating if else statements.

In [100]:
```
def calculate_delay_cost(row, delay_type):
    delay = row[delay_type]
    if pd.isna(delay) or delay <= 15:
        return 0
    else:
        return (delay - 15) * 75

# Apply the function for both DEP_DELAY and ARR_DELAY and create new columns
df_merged_profit['Departure_Delay_Cost'] = df_merged_profit.apply(lambda row: calculate_delay_cost(row, 'DEP_DELAY')
df_merged_profit['Arrival_Delay_Cost'] = df_merged_profit.apply(lambda row: calculate_delay_cost(row, 'ARR_DELAY'),

# Optionally, sum up both costs in a new column 'Total_Operational_Cost'
df_merged_profit['Total_Operational_Cost'] = df_merged_profit['Departure_Delay_Cost'] + df_merged_profit['Arrival_De

# View the first few rows to confirm the new columns
print(df_merged_profit.head())
df_merged_profit
```

Calculating the number of passengers based on occupancy rates

In [101]:
```
def calculate_passengers(row):
    # Assuming 'OCCUPANCY_RATE' is given as a decimal (e.g., 0.8 for 80%)
    if pd.notna(row['OCCUPANCY_RATE']):
        return round(row['OCCUPANCY_RATE'] * 200)
    else:
        # Handle cases where occupancy rate might be NaN
        return None

# Apply the function to each row to create a new column 'Number_of_Passengers'
df_merged_profit['Number_of_Passengers'] = df_merged_profit.apply(calculate_passengers, axis=1)

# View the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

```
      FL_DATE ORIGIN DESTINATION  OCCUPANCY_RATE  DISTANCE  DEP_DELAY  \
```

Understanding the money made from baggage fees

In [102]:
```
# Define a function to calculate baggage fees based on the number of passengers and whether it's a round trip
# Apply the function to each row and create a new column 'Baggage_Fees' in df_merged_rowwise_cleaned
def calculate_baggage_fee(row):
    if pd.notna(row['Number_of_Passengers']):
        # Check if the flight is a round trip
        if row.get('ROUNDTRIP', 0) == 1:
            # Double the fee for round trips
            return row['Number_of_Passengers'] * 35
        else:
            # Fee for one-way trips
            return row['Number_of_Passengers'] * 17.5
    else:
        return None

# Apply the function to each row to create a new column 'Baggage_Fees'
df_merged_profit['Baggage_Fees'] = df_merged_profit.apply(calculate_baggage_fee, axis=1)

# View the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

```
      FL_DATE ORIGIN DESTINATION  OCCUPANCY_RATE  DISTANCE  DEP_DELAY  \
```

Operational costs for airport based on size

1860129 rows × 18 columns

In [103]:
```
# Define a function to calculate operational cost based on airport type
def calculate_operational_cost(TYPE_ORIGIN, TYPE_DESTINATION):
    cost = 0
    if TYPE_ORIGIN == 'medium_airport':
        cost += 5000
    elif TYPE_ORIGIN == 'large_airport':
        cost += 10000

    if TYPE_DESTINATION == 'medium_airport':
        cost += 5000
    elif TYPE_DESTINATION == 'large_airport':
        cost += 10000

    return cost

# Apply the function to each row in the dataframe
df_merged_profit['Airport Operational Costs'] = df_merged_profit.apply(
    lambda row: calculate_operational_cost(row['TYPE_ORIGIN'], row['TYPE_DESTINATION']), axis=1)
df_merged_profit
```

Out[103]:

Total ticket price, this is got by multiplying passengers with ticket ITIN_FARE

1860129 rows × 19 columns

In [104]:
```
# Create a new column 'ticket_price' by multiplying 'ITIN_FARE' and 'Number_of_Passengers'
df_merged_profit['ticket_price'] = df_merged_profit.apply(lambda row: row['ITIN_FARE'] * row['Number_of_Passengers']

# Optionally, view the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

```
      FL_DATE ORIGIN DESTINATION  OCCUPANCY_RATE  DISTANCE  DEP_DELAY  \
```

Finding the total operating costs

In [105]:
```
# Create a new column 'total_costs' by summing the specified columns
df_merged_profit['total_costs'] = df_merged_profit.apply(
    lambda row: row['fuel'] + row['Total_Operational_Cost'] + row.get('origin_cost', 0) + row.get('dest_cost', 0),
    axis=1
)

# Optionally, view the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

In [106]:
```
# Create a new column 'total_revenue' by adding 'Baggage_Fees' and 'ticket_price'
df_merged_profit['total_revenue'] = df_merged_profit.apply(lambda row: row['Baggage_Fees'] + row['ticket_price'], ax

# Optionally, view the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

```
FL_DATE ORIGIN DESTINATION  OCCUPANCY_RATE  DISTANCE  DEP_DELAY  \
```

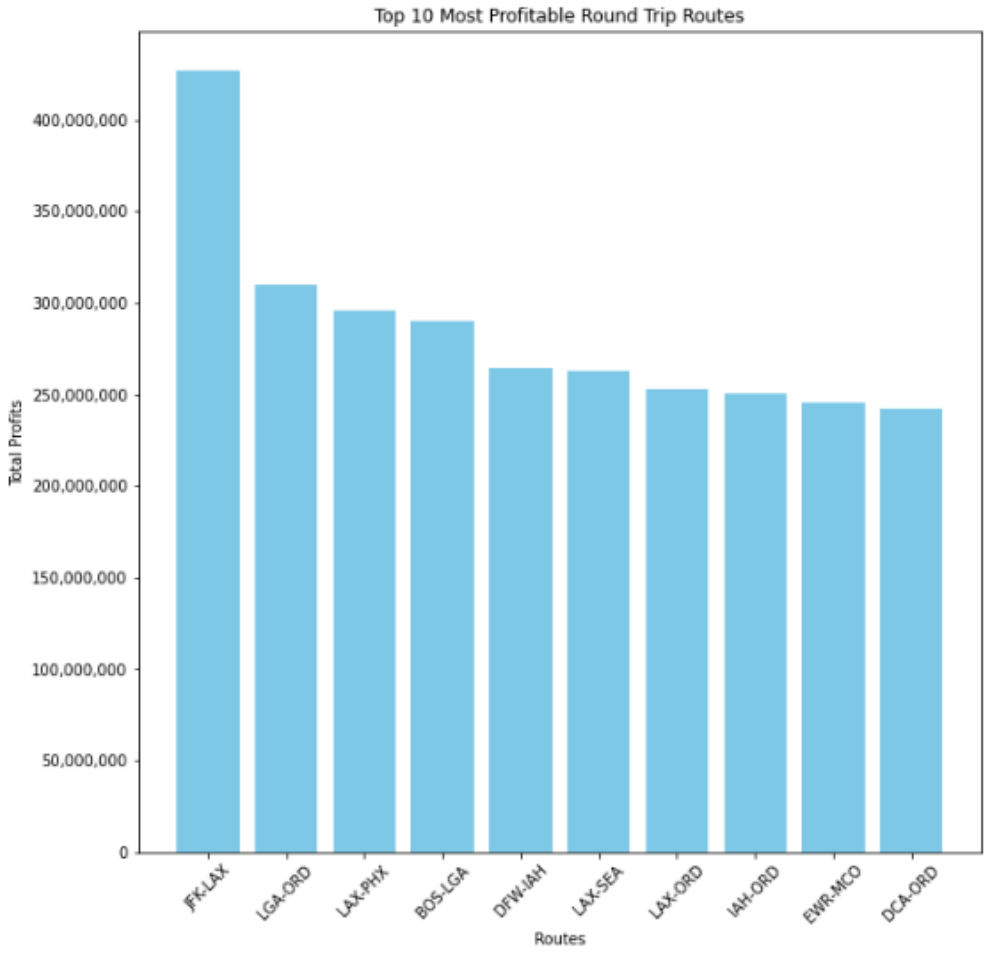
Total revenue generated

Computing total profits got by Total Revenue - Total Costs

In [107]:
```
# Create a new column 'Total_Profits' by subtracting 'total_costs' from 'total_revenue'
df_merged_profit['Total_Profits'] = df_merged_profit['total_revenue'] - df_merged_profit['total_costs']

# Optionally, view the first few rows to confirm the new column
print(df_merged_profit.head())
df_merged_profit
```

Calculating the total amount of profits generated by top 10 profitable routes

In [112]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Assuming df_merged_profit is your DataFrame and it contains columns 'ORIGIN', 'DESTINATION', and 'Total_Profits'
# Step 1: Identify unique round trip routes
df_merged_profit['ROUTE'] = df_merged_profit.apply(lambda x: '-'.join(sorted([x['ORIGIN'], x['DESTINATION']])), axis

# Step 2: Calculate total profits for each route
route_profits = df_merged_profit.groupby('ROUTE')['Total_Profits'].sum().reset_index()

# Step 3: Sort and select top 10 profitable routes
top_routes = route_profits.sort_values(by='Total_Profits', ascending=False).head(10)

# Step 4: Plot the bar graph
plt.figure(figsize=(10, 10))
plt.bar(top_routes['ROUTE'], top_routes['Total_Profits'], color='skyblue')

# Format y-axis to show full numbers
plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.xlabel('Routes')
plt.ylabel('Total Profits')
plt.title('Top 10 Most Profitable Round Trip Routes')
plt.xticks(rotation=45)
plt.show()
```


Top 10 Most Profitable Round Trip Routes
Total Profits
400,000,000
350,000,000
300,000,000
250,000,000
200,000,000
150,000,000
100,000,000
50,000,000
0
JFK-LAX
LGA-ORD
LAX-PHX
BOS-LGA
DFW-IAH
LAX-SEA
LAX-ORD
IAH-ORD
EWR-MCO
DCA-ORD
Routes

The next step we calculate the total costs incurred for the top profitable trips

In [113]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Assuming df_merged_profit is your DataFrame and it contains columns 'ORIGIN', 'DESTINATION', 'Total_Profits', and
# Step 1: Identify unique round trip routes
df_merged_profit['ROUTE'] = df_merged_profit.apply(lambda x: '-'.join(sorted([x['ORIGIN'], x['DESTINATION']])), axis

# Step 2: Calculate total profits and total costs for each route
route_metrics = df_merged_profit.groupby('ROUTE').agg(
    Total_Profits=pd.NamedAgg(column='Total_Profits', aggfunc='sum'),
    total_costs=pd.NamedAgg(column='total_costs', aggfunc='sum')
).reset_index()

# Step 3: Sort by total profits and select top 10 routes
top_profitable_routes = route_metrics.sort_values(by='Total_Profits', ascending=False).head(10)

# Step 4: Plot the bar graph for Total Costs of top 10 profitable routes
plt.figure(figsize=(12, 6))
plt.bar(top_profitable_routes['ROUTE'], top_profitable_routes['total_costs'], color='red')
plt.xlabel('Routes')
plt.ylabel('Total Costs')

# Format y-axis to show full numbers
plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.title('Total Costs for Top 10 Most Profitable Routes')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

The next step is to understand the total revenue generated by the profitable routes

In [114]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

# Assuming df_merged_profit is your DataFrame and it contains columns 'ORIGIN', 'DESTINATION', and 'total_revenue'
# Step 1: Identify unique round trip routes
df_merged_profit['ROUTE'] = df_merged_profit.apply(lambda x: '-'.join(sorted([x['ORIGIN'], x['DESTINATION']])), axis

# Step 2: Calculate total revenue for each route
route_revenues = df_merged_profit.groupby('ROUTE')['total_revenue'].sum().reset_index()

# Step 3: Sort and select top 10 routes by total revenue
top_routes_revenue = route_revenues.sort_values(by='total_revenue', ascending=False).head(10)

# Step 4: Plot the bar graph for Total Revenue
plt.figure(figsize=(10, 10))
plt.bar(top_routes_revenue['ROUTE'], top_routes_revenue['total_revenue'], color='green')

# Format y-axis to show full numbers
plt.gca().yaxis.set_major_formatter(ticker.StrMethodFormatter('{x:,.0f}'))

plt.xlabel('Routes')
plt.ylabel('Total Revenue')
plt.title('Top 10 Routes by Total Revenue')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

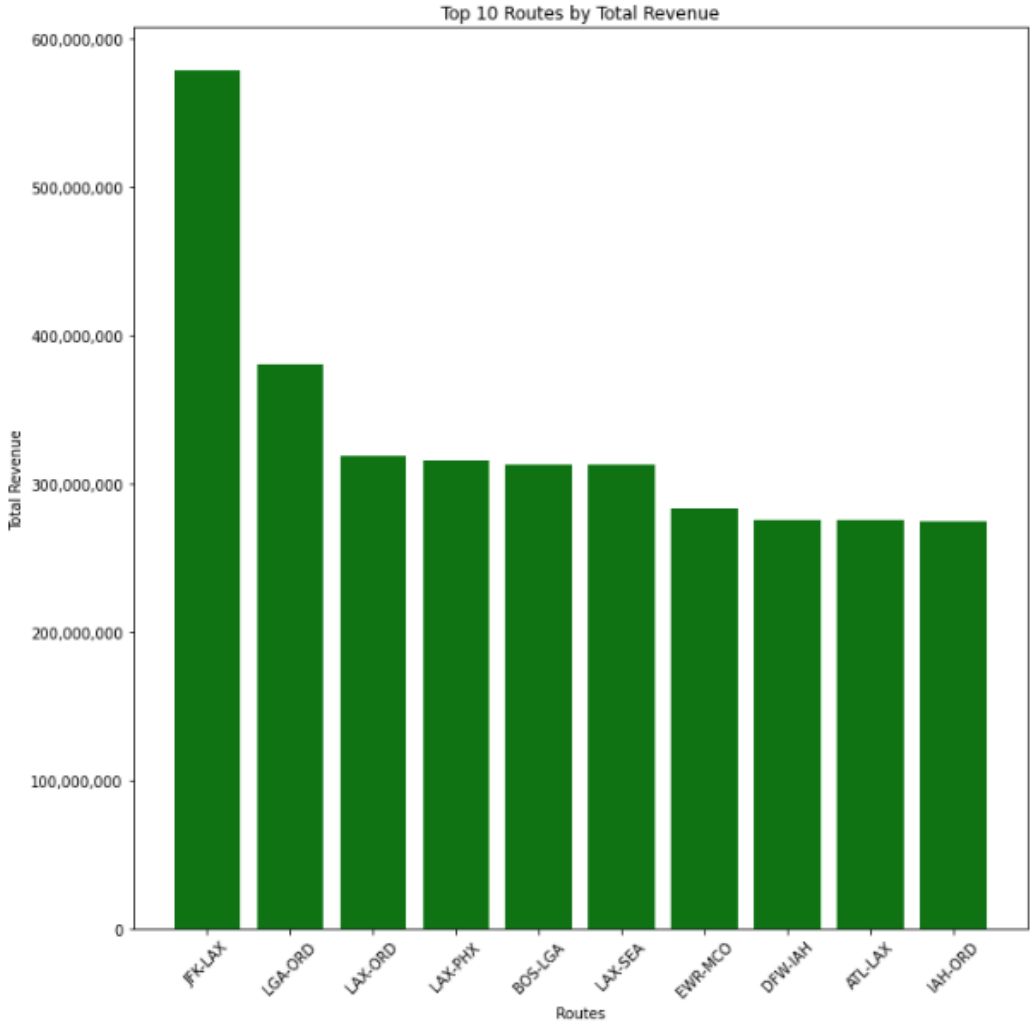
To see how many number of round trip flights are there in the round trip route

In [129]:
```python
import pandas as pd

# Assuming df_merged_profit is your existing DataFrame

# Step 1: Group by 'ROUTE' and sum 'Total_Profits'
profit_per_route = df_merged_profit.groupby('ROUTE')['Total_Profits'].sum()

# Step 2: Sort the sums in descending order
profit_per_route = profit_per_route.sort_values(ascending=False)

# Step 3: Select the top 10 most profitable routes
top_10_profitable_routes = profit_per_route.head(10)

# Step 4: Join with the 'round_trips' information
# Extract 'round_trips' for each 'ROUTE' from the original DataFrame
route_round_trips = df_merged_profit[['ROUTE', 'round_trips']].drop_duplicates()

# Merge the top 10 profitable routes with the round_trips information
top_10_profitable_routes_with_round_trips = top_10_profitable_routes.to_frame().join(route_round_trips.set_index('RO

# Display the top 10 profitable routes with round trips information
print(top_10_profitable_routes_with_round_trips)
```

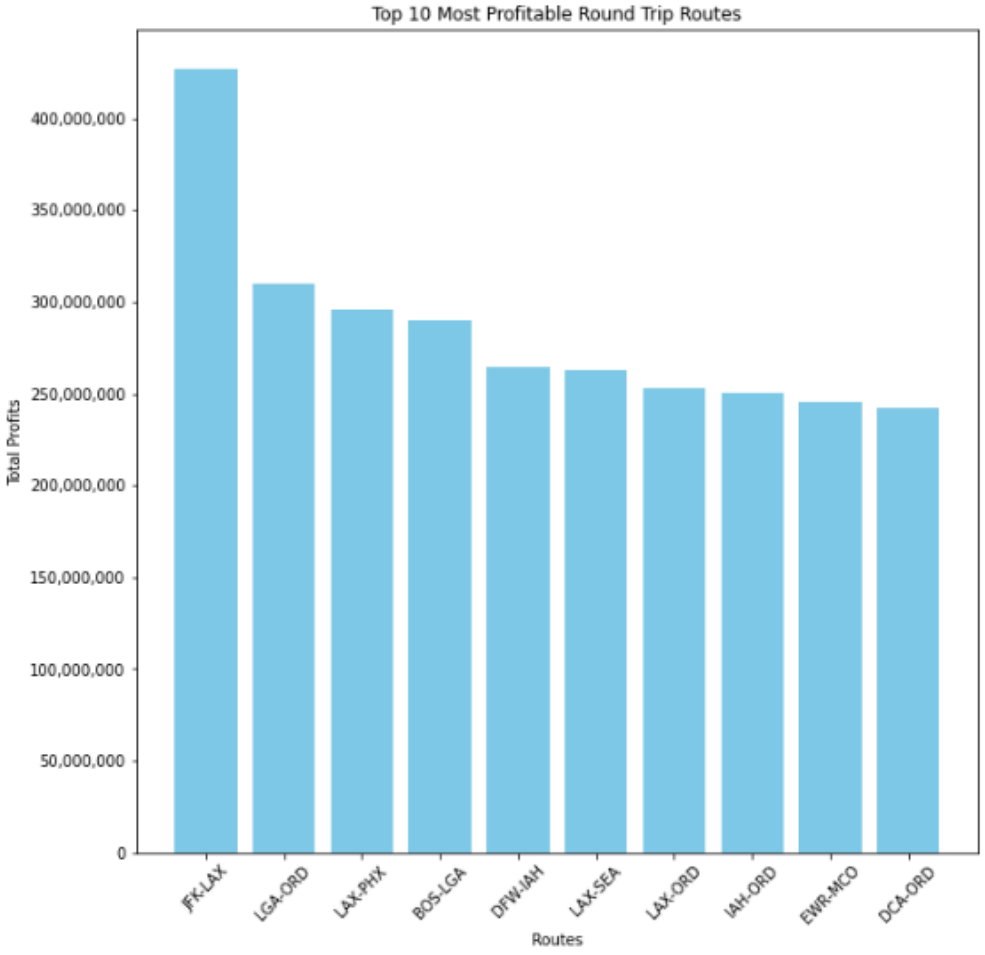
Adding the number of round_trips columns by using join

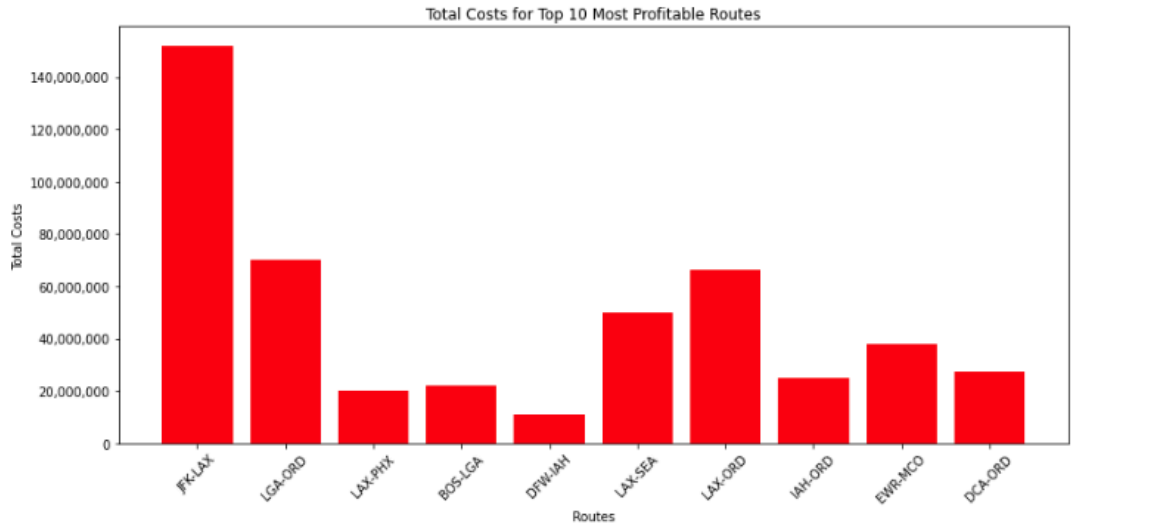
In [118]:
```python
import pandas as pd

# Assuming df_merged_profit and round_trip_routes are your existing DataFrames

# Step 1: Check the 'ROUTE' column in both DataFrames
print(df_merged_profit['ROUTE'].head())
print(round_trip_routes['ROUTE'].head())

# Step 2: Perform the left join operation
df_merged_profits = pd.merge(df_merged_profit, round_trip_routes[['ROUTE', 'round_trips']], on='ROUTE', how='left')

# Step 3: Check if the 'round_trips' column is added successfully
print(df_merged_profit.head())

# Step 4: Handle missing values if any (Optional based on your requirement)
# For example, replacing NaN with 0 in 'round_trips' column
df_merged_profit['round_trips'].fillna(0, inplace=True)
```

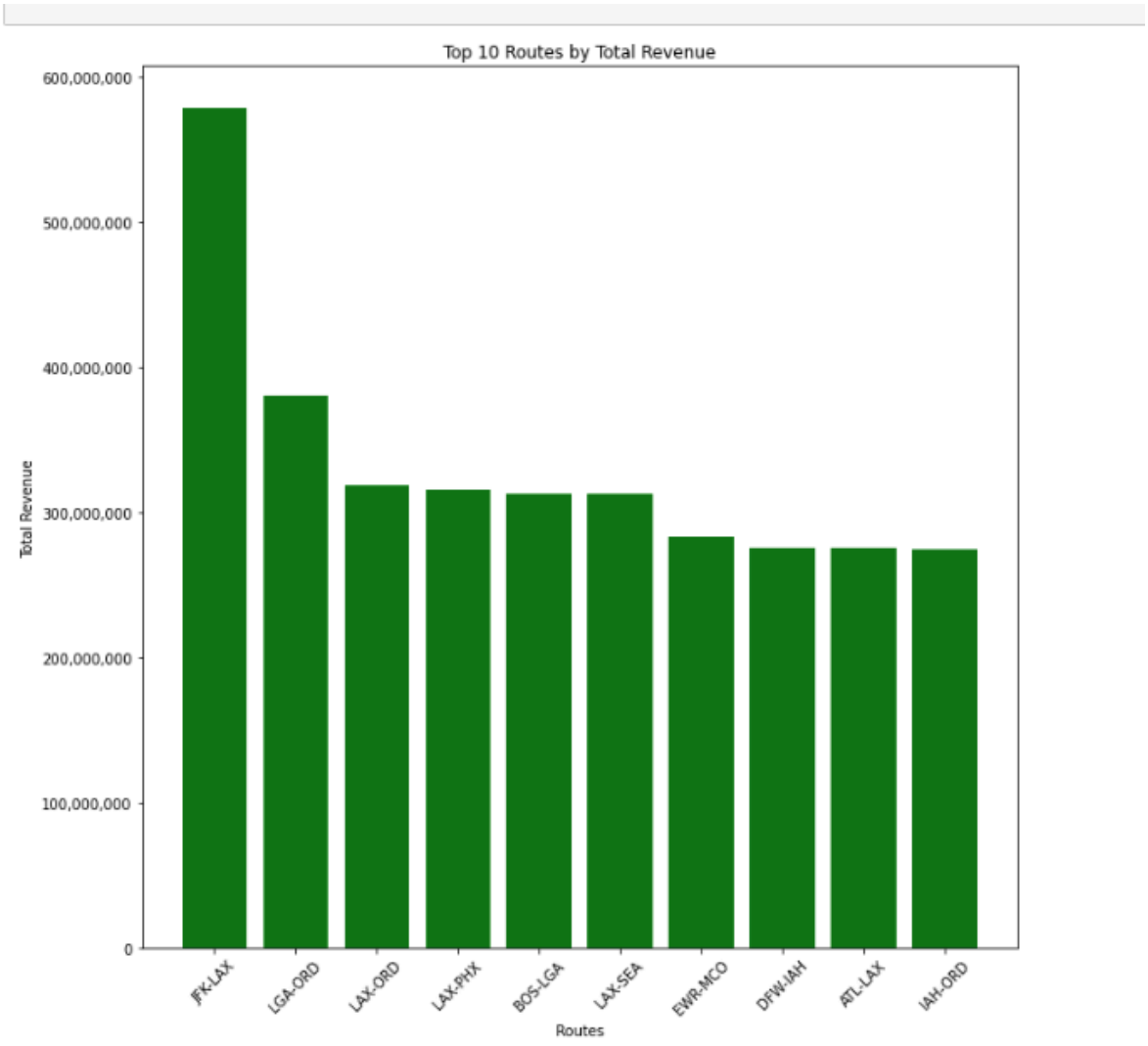
```
0    CLE-RSW
```


Top 10 Routes by Total Revenue
Total Revenue
600,000,000
500,000,000
400,000,000
300,000,000
200,000,000
100,000,000
0
JFK-LAX
LGA-ORD
LAX-ORD
LAX-PHX
BOS-LGA
LAX-SEA
EWR-MCO
DFW-IAH
ATL-LAX
IAH-ORD
Routes

Q3) Top 5 routes I would recommend to invest in







I would like to choose the following routes

a) JFK-LAX:- - Based on the initial analysis this route already helps the airline to achieve its profits and cross the thresh hold margin $400 million in order to recover the airplane costs over time, this route is extremely popular among business travelers, tourists, celebrities so I wouldn't mind charging a premium for this route. Currently the Trans continental routes don't serve hot meals for such a long flight I would like to introduce these once I am able to recover my airplane costs. I woudnt mind utilizing 2 aircrafts for this route given the distance and density of the aircraft. I would want to target having early morning flights from JFK to LAX, and one in the evening as well. From LAX to JFK i would want to target a red eye flight and also an afternoon flight from there.

b) LGA-ORD:- This flight is the second most popular in terms of being profitable, this flight could generate good load factors and also connects few of the major financial and consulting markets in the country. Both of these cities are famous for their consulting presence attracting lots of travelers.

c) BOS-LGA:- This would be a trunk route, recommending this route because the cost of running this route would be less and the profits for this route are high. The reason why

the cost to operate of this route route are less but the revenue and profits being generated are high. Another reason I choose this route is that I can operate 1 flight between LGA-ORD and BOS-LGA saving costs for the company and also optimizing time. I would also want to focus on reducing the turn around time of the aircraft, ie making it ready within 40-50 mins as against to 45 minutes- 1 hour. This route is catered heavily by business travelers, students, consultants, and also tourists. Both these cities also have lots of historical significance. One thing to note is that this route has flights, trains, buses on an average every 30-45 minutes so would need to time the flight well during the day.

d) EWR-MCO:- This route would be mainly for tourists, one of the major reasons I am introducing this route is because the amount of families and people present in the area. Queens being the most diverse community in the area. Also Orlando being home to disney land would attract lots of tourists, kids, parents alike. The profits in this flight generated arent the best but would still want to go for it. One more option I am thinking of having is a disney themed flight for this route, and giving kids out disney themed gifts such as soft toys, drawing and coloring books, butter beer(kids would love these),
e) LAX-PHX:- You have chosen the LAX-PHX route due to its potential for profitability and its short flying distance. This route connects two major economic hubs, Los Angeles and Phoenix, which have strong business ties. Regular flights on this route will support business travel and contribute to the economic growth of both cities. Los Angeles, with its entertainment industry and beaches, and Phoenix, known for its desert landscapes and proximity to the Grand Canyon, are both popular tourist destinations. The direct flight between these cities will boost tourism opportunities. The short distance of this route is ideal for short-haul flights, making it more cost-effective in terms of fuel and operational costs. Additionally, both LAX and PHX are major connecting hubs for other domestic and international flights, so adding this route can enhance your airline's network and provide more options for passengers connecting through these airports.

5) A few KPI major of success I would recommend

a) Financial Performance Metrics: This includes Yield, Revenue per Available Seat Mile (RASM), and Cost per Available Seat Mile (CASM). These metrics will provide a comprehensive view of the profitability and cost efficiency of each route, such as the premium JFK-LAX route and the high-density LGA-ORD route.
b) Operational Efficiency and Reliability: This encompasses Fleet Utilization, Aircraft Turnaround Time, and Schedule Reliability. By using a single fleet type, operational processes can be streamlined, leading to better utilization of the aircraft, faster turnaround times, and more reliable scheduling - critical for routes like BOS-LGA that require quick turnarounds.
c) Customer Satisfaction and Market Position: This involves assessing customer feedback and market share. It includes measuring customer satisfaction through feedback on in-flight services and overall experience, especially for unique offerings like Disney-themed flights on the EWR-MCO route. Market share will indicate your competitive position in the market, especially in routes with multiple transportation options.
d) Environmental and Fuel Efficiency: Focus on Fuel Efficiency and Carbon Footprint Reduction, crucial for measuring the environmental impact and cost savings from using the fuel-efficient Airbus A321Neo. This is particularly relevant given increasing environmental concerns and the need for sustainable operations.
e) Cost Savings from a Single Fleet Type: Evaluate Maintenance and Operational Costs, Training Costs for Crew and Technicians, and Inventory Management Efficiency. Operating a single fleet type like the A321Neo should lead to reduced complexity and cost in these areas, contributing to overall cost-effectiveness and streamlined operations.