Project Title: "Data Science and Machine Learning for Flight Delay Analysis: Empowering Air Travelers"

Project Description:

In this comprehensive data science and machine learning project, we embark on a journey to unravel the intricate factors contributing to flight delays, providing invaluable insights and actionable recommendations to empower air travelers. Our project spans a spectrum of tasks, from collecting and preprocessing extensive flight data to insightful visualization and rigorous statistical analysis. Our machine learning models will play a pivotal role in identifying predictive patterns and enhancing our understanding of flight delays.

Key Objectives and Tasks:

1. **Data Collection**: We'll gather an extensive dataset encompassing a wealth of information about flights, airports, runways, airlines, and passenger traffic. This data will serve as the foundation of our machine learning analysis.

2. **Data Preprocessing**: The raw data will undergo meticulous cleaning and transformation, ensuring data consistency and removing anomalies to enhance the reliability of our analysis.

3. **Data Visualization**: Leveraging data visualization techniques, we'll create compelling visuals to identify trends, patterns, and outliers in the flight data.

4. **Machine Learning Analysis**: Machine learning models will be employed to predict flight delays and uncover latent insights. We'll explore the impact of various factors such as airline choice, airport size, and time of day on delays.

5. **Statistical Analysis**: Hypothesis testing and statistical analysis will complement our machine learning efforts, providing a comprehensive view of the relationships between different factors and flight delays.

6. **Insights and Recommendations**: Our findings will translate into actionable insights for both travelers and airlines. These insights may include recommendations for selecting airlines with superior on-time performance or strategies for avoiding peak travel times.

This project combines the power of data science and machine learning, aiming to demystify the complexities of flight delays. By employing cutting-edge techniques, we empower passengers and airlines with the knowledge needed to make informed decisions, ultimately leading to a smoother and more dependable air travel experience for all. Join us as we navigate the skies of data to revolutionize air travel.

In [1]:
```
import requests # for making standard html requests
from bs4 import BeautifulSoup # magical tool for parsing html data
import pandas as pd # for data manipulation
import numpy as np # for data manipulation
import seaborn as sns # for data visualization
import matplotlib.pyplot as plt # for data visualization
from sklearn.metrics import mean_absolute_percentage_error # for model evaluation
import statsmodels.api as sm # for data exploration
import scipy.stats as stats # for statistical analysis
from sklearn.linear_model import SGDClassifier # for classification
from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV, train_test_sp
from statsmodels.formula.api import glm # for classification
from sklearn.preprocessing import OrdinalEncoder, StandardScaler # for data preprocess
from sklearn.pipeline import Pipeline # for data preprocessing
from sklearn.metrics import classification_report, accuracy_score # for model evaluati
from sklearn.tree import DecisionTreeClassifier # for classification
from xgboost import XGBClassifier # for classification
```

Importing data and aggregating to create a fine tuned dataset for the project.

Collating information specific to flights that may cause delays for the final dataset.

In [2]:
```
airlines=pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job Readine
airports=pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job Readine
runways=pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job Readines
```

Lets look at the imported data.

In [3]:
```
airlines.head()
```

Out[3]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
0	1	CO	269	SFO	IAH	3	15	205	1
1	2	US	1558	PHX	CLT	3	15	222	1
2	3	AA	2400	LAX	DFW	3	20	165	1
3	4	AA	2466	SFO	DFW	3	20	195	1
4	5	AS	108	ANC	SEA	3	30	202	0

In [4]:
```
airports.head()
```

Out[4]:

	id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_cou
0	6523	00A	heliport	Total Rf Heliport	40.070801	-74.933601	11.0	NaN	
1	323361	00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435.0	NaN	
2	6524	00AK	small_airport	Lowell Field	59.947733	-151.692524	450.0	NaN	
3	6525	00AL	small_airport	Epps Airpark	34.864799	-86.770302	820.0	NaN	
4	6526	00AR	closed	Newport Hospital & Clinic Heliport	35.608700	-91.254898	237.0	NaN	

In [5]:
```
runways.head()
```

Out[5]:

	id	airport_ref	airport_ident	length_ft	width_ft	surface	lighted	closed	le_ident	le_latitude
0	269408	6523	00A	80.0	80.0	ASPH-G	1	0	H1	
1	255155	6524	00AK	2500.0	70.0	GRVL	0	0	N	
2	254165	6525	00AL	2300.0	200.0	TURF	0	0	1	
3	270932	6526	00AR	40.0	40.0	GRASS	0	0	H1	
4	322128	322127	00AS	1450.0	60.0	Turf	0	0	1	

Let's **merge** airports and runways and create airport_run using the **ident column in airports**, and the **airport_ident column in runways**. We are doing a left join i.e. all of airports columns and only the matching values from runways will be imported.

In [6]:
```
airport_run = pd.merge(airports, runways, left_on='ident', right_on='airport_ident', h
airport_run.head()
```

Out[6]:

	id_x	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_cou
0	6523	00A	heliport	Total Rf Heliport	40.070801	-74.933601	11.0	NaN	
1	323361	00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435.0	NaN	
2	6524	00AK	small_airport	Lowell Field	59.947733	-151.692524	450.0	NaN	
3	6525	00AL	small_airport	Epps Airpark	34.864799	-86.770302	820.0	NaN	
4	6526	00AR	closed	Newport Hospital & Clinic Heliport	35.608700	-91.254898	237.0	NaN	

5 rows × 38 columns

In [7]:
```
airport_run.columns
```

Out[7]:
```
Index(['id_x', 'ident', 'type', 'name', 'latitude_deg', 'longitude_deg',
       'elevation_ft', 'continent', 'iso_country', 'iso_region',
       'municipality', 'scheduled_service', 'gps_code', 'iata_code',
       'local_code', 'home_link', 'wikipedia_link', 'keywords', 'id_y',
       'airport_ref', 'airport_ident', 'length_ft', 'width_ft', 'surface',
       'lighted', 'closed', 'le_ident', 'le_latitude_deg', 'le_longitude_deg',
       'le_elevation_ft', 'le_heading_degT', 'le_displaced_threshold_ft',
       'he_ident', 'he_latitude_deg', 'he_longitude_deg', 'he_elevation_ft',
       'he_heading_degT', 'he_displaced_threshold_ft'],
      dtype='object')
```

Lets count the number of runways each airport has. Here we will **group the rows** based on the **common airport_ident** values while keep a **count of the non-null values in id_y** which has the runway entries of a particular airport.

In [8]:
```
count_runway = airport_run.groupby('airport_ident')[['id_y']].count().sort_values(by='
count_runway.head()
```

Out[8]:

	airport_ident	id_y
0	KORD	11
1	KNHU	10
2	JRA	9
3	TA12	8
4	SXS	8

Now that we have a count of runways for each airport, let's create a **new dataset containing airport iata code, type of airport, elevation of airport, and the number of runways**. All of which could play a role in delays.

In [9]:
```
air_run = pd.merge(airports, count_runway, left_on = 'ident', right_on = 'airport_ider
air_run.rename(columns = {'id_y':'runway_count'}, inplace = True)
air_run.head()
```

Out[9]:

	id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_cou
0	6523	00A	heliport	Total Rf Heliport	40.070801	-74.933601	11.0	NaN	
1	323361	00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435.0	NaN	
2	6524	00AK	small_airport	Lowell Field	59.947733	-151.692524	450.0	NaN	
3	6525	00AL	small_airport	Epps Airpark	34.864799	-86.770302	820.0	NaN	
4	6526	00AR	closed	Newport Hospital & Clinic Heliport	35.608700	-91.254898	237.0	NaN	

In [10]:
```
air_run = air_run[['iata_code','type','elevation_ft','runway_count']]
```

In [11]:
```
air_run.head()
```

Out[11]:

	iata_code	type	elevation_ft	runway_count
0	NaN	heliport	11.0	1.0
1	NaN	small_airport	3435.0	NaN
2	NaN	small_airport	450.0	1.0
3	NaN	small_airport	820.0	1.0
4	NaN	closed	237.0	1.0

In [12]:
```
air_run.shape
```

Out[12]:
```
(73805, 4)
```

In [13]:
```
air_run.isnull().sum()
```

Out[13]:
```
iata_code       64645
type                0
elevation_ft    14122
runway_count    36540
dtype: int64
```

In [14]:
```
# Removing null values and saving the rest as an excel file.
air_run.dropna().to_excel('air_run.xlsx', index = False)
```

In [15]:
```
airlines.head()
```

Out[15]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay
0	1	CO	269	SFO	IAH	3	15	205	1
1	2	US	1558	PHX	CLT	3	15	222	1
2	3	AA	2400	LAX	DFW	3	20	165	1
3	4	AA	2466	SFO	DFW	3	20	195	1
4	5	AS	108	ANC	SEA	3	30	202	0

Lets **add info about the AirportFrom** values by combining the airlines and air_run datasets based on the AirportFrom and iata_code columns, this would give us the **count of runways, elevation, and iata_code of the airports from where flights take off**. Something that could be a factor in delays.

In [16]:
```
combined_data = pd.merge(airlines, air_run, how='left', left_on='AirportFrom', right_c

new_names = list(combined_data[air_run.columns].columns + '_source_airport')
old_names = list(combined_data[air_run.columns].columns)

combined_data.rename(columns = {old:new for old, new in zip(old_names, new_names)}, ir
combined_data.head()
```

Out[16]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	iata_code_source_air
0	1	CO	269	SFO	IAH	3	15	205	1	
1	2	US	1558	PHX	CLT	3	15	222	1	
2	3	AA	2400	LAX	DFW	3	20	165	1	
3	4	AA	2466	SFO	DFW	3	20	195	1	
4	5	AS	108	ANC	SEA	3	30	202	0	

In [17]:
```
combined_data.columns
```

Out[17]:
```
Index(['id', 'Airline', 'Flight', 'AirportFrom', 'AirportTo', 'DayOfWeek',
       'Time', 'Length', 'Delay', 'iata_code_source_airport',
       'type_source_airport', 'elevation_ft_source_airport',
       'runway_count_source_airport'],
      dtype='object')
```

Let's **add similar info about the destination airports**. This would also be a factor in flights delays. For eg: if the destination airport has only one runway, flights would have to park themselves and wait for their turn to land resulting in delays.

But this time we will use combined_data instead of airlines since we have all the info from airlines dataset from previous steps.

In [18]:
```
combined_data = pd.merge(combined_data, air_run, how='left', left_on = 'AirportTo', ri
```

In [19]:
```
new_names = list(combined_data[air_run.columns].columns + '_dest_airport')
old_names = list(combined_data[air_run.columns].columns)

combined_data.rename(columns = {old:new for old,new in zip(old_names, new_names)}, inp
combined_data.head()
```

Out[19]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	iata_code_source_air
0	1	CO	269	SFO	IAH	3	15	205	1	
1	2	US	1558	PHX	CLT	3	15	222	1	
2	3	AA	2400	LAX	DFW	3	20	165	1	
3	4	AA	2466	SFO	DFW	3	20	195	1	
4	5	AS	108	ANC	SEA	3	30	202	0	

In [20]:
```
# We don't need iata_code columns now, lets drop them.
combined_data.drop(columns = list(combined_data.columns[combined_data.columns.str.star
```

In [21]:
```
combined_data.head()
```

Out[21]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

As is with every profession, the amount of experience we have plays an important role in our performance. Applying that logic to airline delays, let's **extract experience info about each airline in our dataset** from "https://en.wikipedia.org/wiki/List_of_airlines_of_the_United_States" using Beautiful Soup class from bs4 and requests lib.

In [22]:
```
# Extracting the number of tables in the URL.

airexp_url = requests.get('https://en.wikipedia.org/wiki/List_of_airlines_of_the_Unite
soup = BeautifulSoup(airexp_url, 'lxml')
tables_found = soup.findAll("table", {"class":"wikitable"})
```

In [23]:
```
len(tables_found)
```

Out[23]: 7

In [24]:
```
airlines_wiki_list = []
for tab in tables_found:
    temp = pd.read_html(str(tab))
    temp = pd.DataFrame(temp[0])
    airlines_wiki_list.append(temp)
```

```
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1975860362.py:3: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  temp = pd.read_html(str(tab))
```

In [25]:
```
airlines_wiki = pd.concat(airlines_wiki_list)
```

Now that we have extracted all the relevant information from the wiki URL, its time to add that into our master dataset i.e. combined_data.

In [26]:
```
# Let's start by idetifying the founding year of the airlines in combined_data.

airlines_founded = pd.merge(combined_data[['Airline']].drop_duplicates(), airlines_wik
```

In [27]:
```
airlines_founded
```

Out[27]:

	Airline	IATA	Founded
0	CO	NaN	NaN
1	US	NaN	NaN
2	AA	AA	1926.0
3	AS	AS	1932.0
4	DL	DL	1924.0
5	B6	B6	1998.0
6	HA	HA	1929.0
7	OO	OO	1972.0
8	9E	9E	1985.0
9	OH	OH	1979.0
10	EV	NaN	NaN
11	XE	XE	2016.0
12	YV	YV	1980.0
13	UA	UA	1926.0
14	MQ	MQ	1984.0
15	F9	F9	1994.0
16	WN	WN	1967.0

Apart from the experience, the traffic of the airport is also a factor in delays. If your source/destination airport sees a lot of traffic, naturally the take-off and landing times will be affected.

For this step, **we will extract info from** https://en.wikipedia.org/wiki/List_of_the_busiest_airports_in_the_United_States

But this wikipedia page was updated recently, so I am **using a previous version of the wikipedia page of 13 April 2023** from this URL: https://en.wikipedia.org/w/index.php?title=List_of_the_busiest_airports_in_the_United_States&oldid=1149689977

In [28]:
```
traffic_url = requests.get('https://en.wikipedia.org/w/index.php?title=List_of_the_bus
soup = BeautifulSoup(traffic_url,'lxml')
traffic_tables = soup.findAll("table", {"class":"wikitable"})
```

In [29]:
```
hub_data = {}
i=0
for tab in traffic_tables:
    hub_data[i] = pd.read_html(str(tab))
    hub_data[i] = pd.DataFrame(hub_data[i][0])
    i+=1
```

```
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\1926796275.py:4: FutureWarning: Pas
sing literal html to 'read_html' is deprecated and will be removed in a future versio
n. To read from a literal string, wrap it in a 'StringIO' object.
  hub_data[i] = pd.read_html(str(tab))
```

In [30]:

```
large_hub = hub_data[0].copy()
med_hub = hub_data[1].copy()

large_hub.insert(loc =1, column = 'Hub_type', value = 'large')
med_hub.insert(loc =1, column = 'Hub_type', value = 'medium')
```

In [31]:

```
large_hub.head()
```

Out[31]:

	Rank (2021)	Hub_type	Airports (large hubs)	IATA Code	Major cities served	State	2021[3]	2020[4]	2019[5]	2018[6]	
0	1	large	Hartsfield–Jackson Atlanta International Airport	ATL	Atlanta	GA	36676010	20559866	53505795	51865797	5(
1	2	large	Dallas/Fort Worth International Airport	DFW	Dallas & Fort Worth	TX	30005266	18593421	35778573	32821799	3
2	3	large	Denver International Airport	DEN	Denver	CO	28645527	16243216	33592945	31362941	2
3	4	large	O'Hare International Airport	ORD	Chicago	IL	26350976	14606034	40871223	39873927	3
4	5	large	Los Angeles International Airport	LAX	Los Angeles	CA	23663410	14055777	42939104	42624050	4

In [32]:
```
med_hub.head()
```

Out[32]:

	Rank (2021)	Hub_type	Airports (medium hubs)	IATA Code	City served	State	2021[3]	2020[4]	2019[5]	2018[6]	2017
0	31	medium	Dallas Love Field	DAL	Dallas	TX	6487563	3669930	8408457	8134848	78767
1	32	medium	Daniel K. Inouye International Airport	HNL	Honolulu	HI	5830928	3126391	9988678	9578505	9743
2	33	medium	Portland International Airport	PDX	Portland	OR	5759879	3455877	9797408	9940866	9435
3	34	medium	William P. Hobby Airport	HOU	Houston	TX	5560780	3127178	7069614	6937061	6741
4	35	medium	Southwest Florida International Airport	RSW	Fort Myers	FL	5080805	2947139	5144467	4719568	4461

In [33]:
```
# Let's clean the column names from special characters or things in bracket.

column_temp = large_hub.columns.str.split('[([]').str[0].str.strip().str.lower().str.r
column_temp[list(map( lambda x:x.isnumeric(), column_temp))] = 'data_' + column_temp[]
```

```
large_hub.columns = column_temp
large_hub.columns
```

Out[33]:
```
Index(['rank', 'hub_type', 'airports', 'iata_code', 'major_cities_served',
       'state', 'data_2021', 'data_2020', 'data_2019', 'data_2018',
       'data_2017', 'data_2016', 'data_2015', 'data_2014', 'data_2013',
       'data_2012'],
      dtype='object')
```

In [34]:
```
column_temp = med_hub.columns.str.split('[([]').str[0].str.strip().str.lower().str.rep
column_temp[list(map( lambda x : x.isnumeric(), column_temp))]  = 'data_' + column_tem
med_hub.columns = column_temp
med_hub.columns
```

Out[34]:
```
Index(['rank', 'hub_type', 'airports', 'iata_code', 'city_served', 'state',
       'data_2021', 'data_2020', 'data_2019', 'data_2018', 'data_2017',
       'data_2016', 'data_2015', 'data_2014', 'data_2013', 'data_2012'],
      dtype='object')
```

In [35]:
```
# Renaming 'major cities served' to 'city served'.

large_hub.rename(columns = {'major_cities_served' : 'ciity_served'}, inplace = True)
```

In [36]:
```
# Collating the info about large and medium hubs into one dataframe.

final_hub_data = pd.concat([large_hub, med_hub])
final_hub_data.head()
```

Out[36]:

	rank	hub_type	airports	iata_code	ciity_served	state	data_2021	data_2020	data_2019	data
0	1	large	Hartsfield–Jackson Atlanta International Airport	ATL	Atlanta	GA	36676010	20559866	53505795	518
1	2	large	Dallas/Fort Worth International Airport	DFW	Dallas & Fort Worth	TX	30005266	18593421	35778573	328
2	3	large	Denver International Airport	DEN	Denver	CO	28645527	16243216	33592945	313
3	4	large	O'Hare International Airport	ORD	Chicago	IL	26350976	14606034	40871223	398
4	5	large	Los Angeles International Airport	LAX	Los Angeles	CA	23663410	14055777	42939104	426

In [37]:
```
final_hub_data.data_2021.isnull().sum()
```

Out[37]:
```
0
```

In [38]:
```
# Let's add traffic info in our master dataset i.e. combined_data and create a new dat
```

```
combined_data_traffic = pd.merge(combined_data, final_hub_data[['iata_code','data_2021
```

In [39]:
```
# Renaming the iata_code and data_2021 columns to reflect the source airport.
combined_data_traffic.rename(columns = {'iata_code':'iata_code_source', 'data_2021':'d
```

In [40]:
```
combined_data_traffic.head()
```

Out[40]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

In [41]:
```
# Merging traffic info for destination airports.

combined_data_traffic = pd.merge(combined_data_traffic, final_hub_data[['iata_code', '
```

In [42]:
```
# Renaming the added columns to reflect the destination airport.

combined_data_traffic.rename(columns = {'iata_code' : 'iata_code_dest','data_2021':'da
```

In [43]:
```
combined_data_traffic
```

Out[43]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_sou
0	1	CO	269	SFO	IAH	3	15	205	1	la
1	2	US	1558	PHX	CLT	3	15	222	1	la
2	3	AA	2400	LAX	DFW	3	20	165	1	la
3	4	AA	2466	SFO	DFW	3	20	195	1	la
4	5	AS	108	ANC	SEA	3	30	202	0	la
...	...	...	...	...	...	...	...	...	...	
518551	539377	B6	717	JFK	SJU	5	1439	220	1	la
518552	539378	B6	739	JFK	PSE	5	1439	223	1	la
518553	539379	CO	178	OGG	SNA	5	1439	326	0	medi
518554	539382	UA	78	HNL	SFO	5	1439	313	1	la
518555	539383	US	1442	LAX	PHL	5	1439	301	1	la

518556 rows × 19 columns

Adding the founding year of respective airlines.

In [44]:
```
airlines_founded
```

Out[44]:

	Airline	IATA	Founded
0	CO	NaN	NaN
1	US	NaN	NaN
2	AA	AA	1926.0
3	AS	AS	1932.0
4	DL	DL	1924.0
5	B6	B6	1998.0
6	HA	HA	1929.0
7	OO	OO	1972.0
8	9E	9E	1985.0
9	OH	OH	1979.0
10	EV	NaN	NaN
11	XE	XE	2016.0
12	YV	YV	1980.0
13	UA	UA	1926.0
14	MQ	MQ	1984.0
15	F9	F9	1994.0
16	WN	WN	1967.0

In [45]:
```
# Merging it with combined_data_traffic using 'Airline' column.

combined_data_traffic = pd.merge(combined_data_traffic, airlines_founded[['Airline', '
```

In [46]:
```
combined_data_traffic.head()
```

Out[46]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

Missing value treatment for the final dataset.

In [47]:
```
combined_data_traffic.isnull().sum()
```

Out[47]:
```
id                              0
Airline                         0
Flight                          0
AirportFrom                     0
AirportTo                       0
DayOfWeek                       0
Time                            0
Length                          0
Delay                           0
type_source_airport            31
elevation_ft_source_airport    31
runway_count_source_airport    31
type_dest_airport              31
elevation_ft_dest_airport      31
runway_count_dest_airport      31
iata_code_source            83582
data_2021_source_airport    83582
iata_code_dest              83531
data_2021_dest_airport      83531
Founded                     83601
dtype: int64
```

In [48]:
```
# Let's start with type of airport.

# For source airport.
combined_data_traffic[combined_data_traffic.type_source_airport.isnull()].AirportFrom.
```

Out[48]:
```
array(['CYS'], dtype=object)
```

In [49]:
```
# For destination airport.
combined_data_traffic[combined_data_traffic.type_dest_airport.isnull()].AirportTo.unic
```

Out[49]:
```
array(['CYS'], dtype=object)
```

In [50]:
```
# Lets see what the data dictionary has to say about CYS.

airport_dic=pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job Reac

airport_dic.head()
```

Out[50]:

	Column	Description	Unnamed: 2
0	Airline	Different types of commercial airlines	NaN
1	Flight	Types of Aircraft	NaN
2	AirportFrom	Source Airport	NaN
3	AirportTo	Destination Airport	NaN
4	DayOfWeek	Tells you about the day of week	NaN

In [51]:
```
# Checking for the Description column for 'CYS'

row = airport_dic[airport_dic['Column'] == 'CYS'].Description.values.astype(str)
print(row)
```

```
['Cheyenne Regional Jerry Olson Field']
```

In [52]:
```
airports.head()
```

Out[52]:

	id	ident	type	name	latitude_deg	longitude_deg	elevation_ft	continent	iso_cou
0	6523	00A	heliport	Total Rf Heliport	40.070801	-74.933601	11.0	NaN	
1	323361	00AA	small_airport	Aero B Ranch Airport	38.704022	-101.473911	3435.0	NaN	
2	6524	00AK	small_airport	Lowell Field	59.947733	-151.692524	450.0	NaN	
3	6525	00AL	small_airport	Epps Airpark	34.864799	-86.770302	820.0	NaN	
4	6526	00AR	closed	Newport Hospital & Clinic Heliport	35.608700	-91.254898	237.0	NaN	

In [53]:
```
# Extracting the type of airport and elevantion ft for 'CYS' from airports dataset.

nan_airport = airports.loc[airports.name.str.lower() == row[0].lower(), ['ident', 'nam
nan_airport
```

Out[53]:

	ident	name	iata_code	type	elevation_ft
34675	KCYS	Cheyenne Regional Jerry Olson Field	NaN	medium_airport	6159.0

In [54]:
```
# Getting the runway count for 'CYS' from runway dataset.

miss_comb = pd.merge(nan_airport, runways[['airport_ident', 'id']], how = 'left', left

miss_runway_count = miss_comb.groupby('ident')[['id']].count().reset_index()
miss_runway_count
```

Out[54]:

	ident	id
0	KCYS	2

In [55]:
```
air_miss_data = pd.merge(nan_airport, miss_runway_count).rename(columns = {'id': 'runw
air_miss_data
```

Out[55]:

	iata_code	type	elevation_ft	runway_count
0	NaN	medium_airport	6159.0	2

In [56]:
```
combined_data_traffic.loc[combined_data_traffic.AirportFrom == 'CYS', ['type_source_ai
combined_data_traffic.loc[combined_data_traffic.AirportFrom == 'CYS', ['elevation_ft_s
combined_data_traffic.loc[combined_data_traffic.AirportFrom == 'CYS', ['runway_count_s
```

In [57]:
```
combined_data_traffic.loc[combined_data_traffic.AirportTo == 'CYS', ['type_dest_airpor
combined_data_traffic.loc[combined_data_traffic.AirportTo == 'CYS', ['elevation_ft_des
```

```
combined_data_traffic.loc[combined_data_traffic.AirportTo == 'CYS', ['runway_count_des
```

In [58]:
```
combined_data_traffic.isnull().sum()
```

Out[58]:
```
id                                0
Airline                           0
Flight                            0
AirportFrom                       0
AirportTo                         0
DayOfWeek                         0
Time                              0
Length                            0
Delay                             0
type_source_airport               0
elevation_ft_source_airport       0
runway_count_source_airport       0
type_dest_airport                 0
elevation_ft_dest_airport         0
runway_count_dest_airport         0
iata_code_source              83582
data_2021_source_airport      83582
iata_code_dest                83531
data_2021_dest_airport        83531
Founded                       83601
dtype: int64
```

Let's work on the founded column.

In [59]:
```
airline_dict = pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job R
airline_dict.head(2)
```

Out[59]:

	Airlines ID	Description
0	WN	Southwest
1	DL	Delta

In [60]:
```
miss_founded = combined_data_traffic[combined_data_traffic.Founded.isnull()].Airline.u
miss_founded
```

Out[60]:
```
array(['CO', 'US', 'EV'], dtype=object)
```

In [61]:
```
# Let's look for the description of these in our data dictionary.

miss_airline = airline_dict[airline_dict['Airlines ID'].isin(miss_founded)]
miss_airline
```

Out[61]:

	Airlines ID	Description
5	US	PSA (initially US Airway Express)
7	EV	ExpressJet
9	CO	United Airlines (initially CO)

In [62]:
```
miss_val = {'US':1967, 'EV':1986, 'CO':1931}
```

```
for aline in miss_founded:
    combined_data_traffic.loc[(combined_data_traffic.Founded.isnull())& (combined_data
```

In [63]:
```
combined_data_traffic.isnull().sum()
```

Out[63]:
```
id                                0
Airline                           0
Flight                            0
AirportFrom                       0
AirportTo                         0
DayOfWeek                         0
Time                              0
Length                            0
Delay                             0
type_source_airport               0
elevation_ft_source_airport       0
runway_count_source_airport       0
type_dest_airport                 0
elevation_ft_dest_airport         0
runway_count_dest_airport         0
iata_code_source              83582
data_2021_source_airport      83582
iata_code_dest                83531
data_2021_dest_airport        83531
Founded                           0
dtype: int64
```

In [64]:
```
combined_data_traffic.drop(columns = ['iata_code_source', 'iata_code_dest'], inplace =
```

In [65]:
```
(combined_data_traffic.isna().sum().sort_values(ascending = False)/combined_data_traff
```

Out[65]:
```
data_2021_source_airport      16.118221
data_2021_dest_airport        16.108386
id                             0.000000
Airline                        0.000000
runway_count_dest_airport      0.000000
elevation_ft_dest_airport      0.000000
type_dest_airport              0.000000
runway_count_source_airport    0.000000
elevation_ft_source_airport    0.000000
type_source_airport            0.000000
Delay                          0.000000
Length                         0.000000
Time                           0.000000
DayOfWeek                      0.000000
AirportTo                      0.000000
AirportFrom                    0.000000
Flight                         0.000000
Founded                        0.000000
dtype: float64
```

We will use the median airport traffic for 16% missing value treatment.

In [66]:
```
med_val = combined_data_traffic.groupby('type_source_airport')[['data_2021_source_airp
med_val
```

Out[66]:

type_source_airport	data_2021_source_airport
large_airport	14514049.0
medium_airport	2273259.0
small_airport	NaN

In [67]:
```
for typ in combined_data_traffic.type_source_airport.unique():
      combined_data_traffic.loc[(combined_data_traffic.type_source_airport == typ)& (c
                        'data_2021_source_airport'] = med_val.loc[typ].values[0]
```

In [68]:
```
med_val_dest = combined_data_traffic.groupby('type_dest_airport')[['data_2021_dest_air
med_val_dest
```

Out[68]:

type_dest_airport	data_2021_dest_airport
large_airport	14514049.0
medium_airport	2273259.0
small_airport	NaN

In [69]:
```
for typ in combined_data_traffic.type_source_airport.unique():
      combined_data_traffic.loc[(combined_data_traffic.type_dest_airport == typ)& (com
                        'data_2021_dest_airport'] = med_val.loc[typ].values[0]
```

In [70]:
```
(combined_data_traffic.isna().sum().sort_values(ascending = False)/combined_data_traff
```

Out[70]:
```
data_2021_source_airport       0.226205
data_2021_dest_airport         0.224855
id                             0.000000
Airline                        0.000000
runway_count_dest_airport      0.000000
elevation_ft_dest_airport      0.000000
type_dest_airport              0.000000
runway_count_source_airport    0.000000
elevation_ft_source_airport    0.000000
type_source_airport            0.000000
Delay                          0.000000
Length                         0.000000
Time                           0.000000
DayOfWeek                      0.000000
AirportTo                      0.000000
AirportFrom                    0.000000
Flight                         0.000000
Founded                        0.000000
dtype: float64
```

In [71]:
```
combined_data_traffic.dropna(subset=['data_2021_source_airport', 'data_2021_dest_airpo
```

In [72]:
```
(combined_data_traffic.isna().sum().sort_values(ascending = False)/combined_data_traff
```

Out[72]:
```
id                          0.0
Airline                     0.0
data_2021_dest_airport      0.0
data_2021_source_airport    0.0
runway_count_dest_airport   0.0
elevation_ft_dest_airport   0.0
type_dest_airport           0.0
runway_count_source_airport 0.0
elevation_ft_source_airport 0.0
type_source_airport         0.0
Delay                       0.0
Length                      0.0
Time                        0.0
DayOfWeek                   0.0
AirportTo                   0.0
AirportFrom                 0.0
Flight                      0.0
Founded                     0.0
dtype: float64
```

combined_data_traffic

The dataset is now ready for further analytics.

Data Visualization

- According to data provided, around 70% of the flights are delayed for Southwest Airlines. Visualize to compare the same for other airlines.

In [73]:
```
airline_dict = pd.read_excel(r"C:\Users\bhara\Downloads\simplilearn case studies\Job R
airline_dict.head(2)
```

Out[73]:

	Column	Description	Unnamed: 2
0	Airline	Different types of commercial airlines	NaN
1	Flight	Types of Aircraft	NaN

In [74]:
```
southwestid = airline_dict.loc[airline_dict['Description'].str.strip().str.lower() ==
southwestid
```

Out[74]:
```
'WN'
```

Creating copy of the final dataset with a shorter name.

In [75]:
```
# Creating a copy for the main dataset with a shorter name.
cdt = combined_data_traffic.copy()
```

In [76]:
```
# Calculating the % of delays for Southwest Airlines.
round((cdt[cdt.Airline == southwestid].Delay.sum()/
      cdt[cdt.Airline == southwestid].Delay.size)*100)
```

Out[76]:
```
70
```

Yes, ~70% of flights are delayed for SouthWest Airlines.

In [77]:
```
# Lets define a function to calculate the % of delays for any airline.
def delay_percent(x):
    return round(x.sum()/x.size*100,2)
```

In [78]:
```
# Calculating the % of delays for all airlines.

delay_for_all = cdt.groupby('Airline')['Delay'].apply(delay_percent).reset_index()

delay_for_all.head()
```

Out[78]:

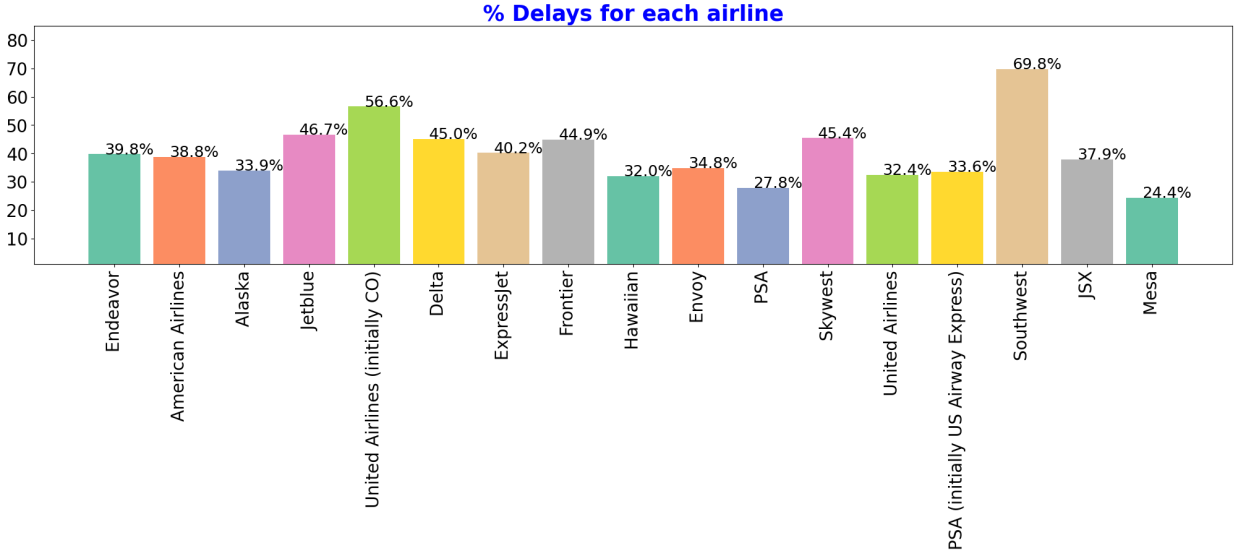
	Airline	Delay
0	9E	39.78
1	AA	38.84
2	AS	33.93
3	B6	46.70
4	CO	56.58

In [79]:
```
# Adding description to the delay_for_all dataset to be used for plotting.
plot_data = pd.merge(delay_for_all, airline_dict, left_on='Airline', right_on='Column'
plot_data.head()
```

Out[79]:

	Airline	Description	Delay
0	9E	Endeavor	39.78
1	AA	American Airlines	38.84
2	AS	Alaska	33.93
3	B6	Jetblue	46.70
4	CO	United Airlines (initially CO)	56.58

In [80]:
```
# Plotting a bar chart for a comparative view of delays in all the airlines vs the Sou
plt.figure(figsize=(25,5))
plt.bar(plot_data.Description, height = plot_data.Delay, color = plt.get_cmap('Set2').
for v, idx in zip(plot_data.Delay.values, plot_data.index):
    plt.annotate('{:.1f}%'.format(v), xy=(idx-0.15,v), size =18)
plt.ylim(1,85)
plt.xticks(size=20, rotation=90)
plt.yticks(size=20)
plt.title('% Delays for each airline', size=25, color='blue', weight='heavy')
plt.show()
```



As we can see in the above bar chart, SouthWest airlines has the most delayed flgihts i.e. 69.8% to be exact.

Mesa is the airlines with least delays.

- Number of delayed flights on different weekdays, which days of the week are the safest to travel?

In [81]:
```
cdt.head()
```

Out[81]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

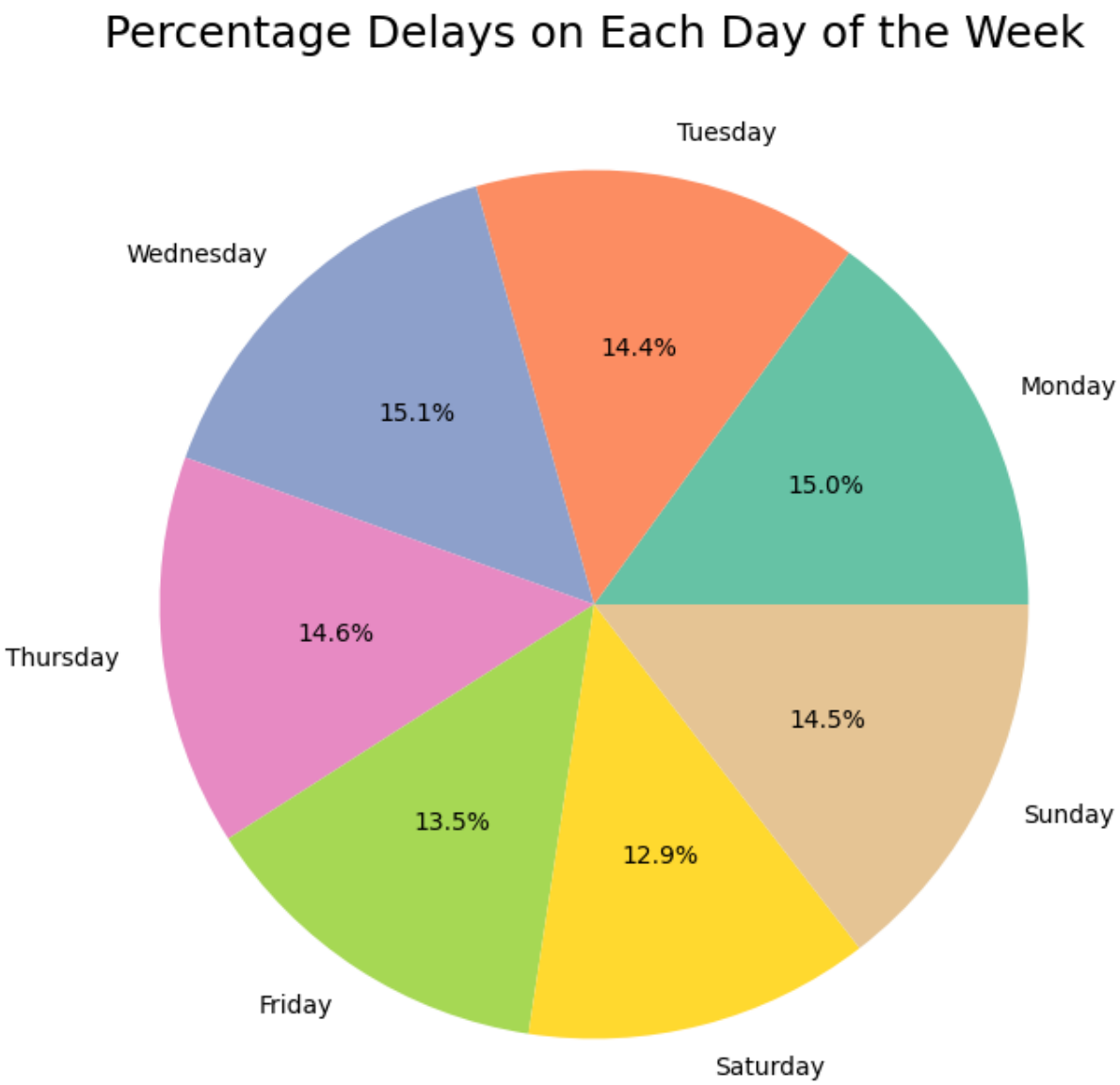
In [82]:
```
# Lets calculate the % of delays for each airline for each weekday.

delay_each_day = cdt.groupby('DayOfWeek')['Delay'].apply(delay_percent).reset_index()
delay_each_day
```

Out[82]:

	DayOfWeek	Delay
0	1	47.28
1	2	45.25
2	3	47.63
3	4	45.84
4	5	42.58
5	6	40.57
6	7	45.77

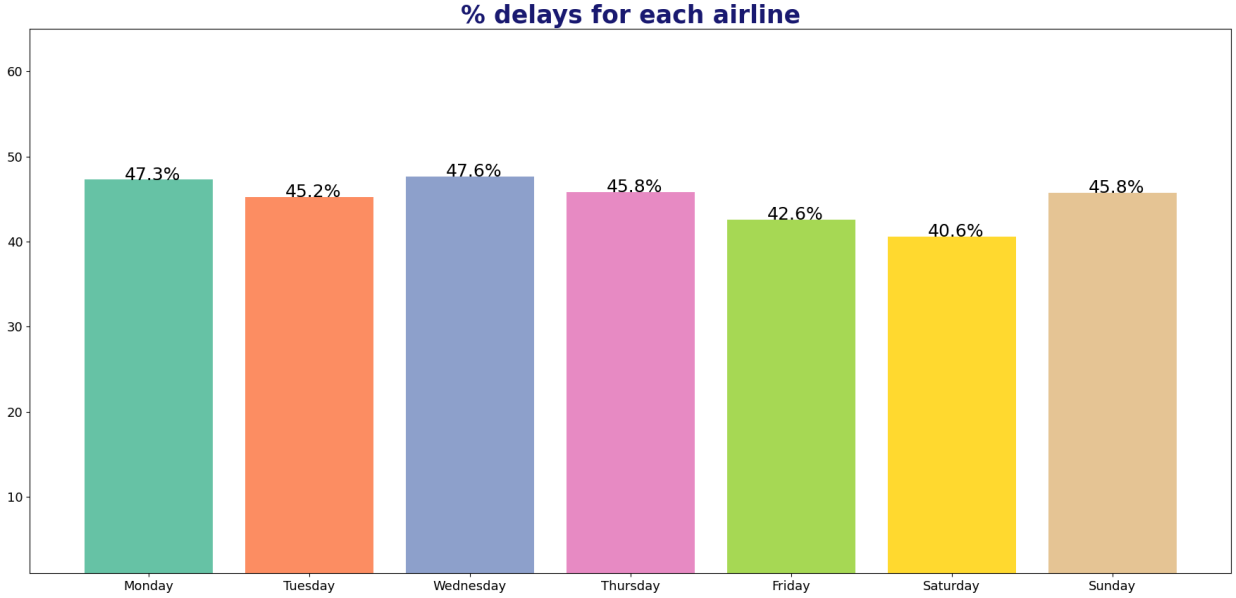
In [83]:

```
# Data for the pie chart
day_numbers = delay_each_day['DayOfWeek']
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sund
values = delay_each_day['Delay'].values

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(values, labels=day_names, autopct='%.1f%%', colors=plt.get_cmap('Set2').colors

# Adding a title
plt.title('Percentage Delays on Each Day of the Week', size=18)

# Displaying the pie chart
plt.show()
```



From the pie chart above the best days to travel are: Friday and Saturday.

In [84]:
```
plt.figure(figsize = (22,10))
plt.bar(delay_each_day['DayOfWeek'], height = delay_each_day['Delay'].values, color =
for v, idx in zip(delay_each_day['Delay'].values, range(1, len(delay_each_day['DayOfWe
    plt.annotate('{:.1f}%'.format(v), xy = (idx-0.15, v), size = 18)
plt.ylim(1,65)
plt.xticks(range(1, len(delay_each_day['DayOfWeek'])+1), day_names, size=13)
plt.yticks(size = 13)
plt.title('% delays for each airline', size = 25, color = 'midnightblue', weight = 'he
plt.show()
```



From the bar chart above, we can see that Friday and Saturday have the shortest bars for % delays, hence these two days are the safest to travel.

- Which airlines to recommend for short, medium, and long length travel?

In [85]:
```
duration = cdt[['Airline', 'Length', 'Delay']].copy()
duration.head()
```

Out[85]:

	Airline	Length	Delay
0	CO	205	1
1	US	222	1
2	AA	165	1
3	AA	195	1
4	AS	202	0

In [86]:
```
# Dataset with delay % of each airline for short, medium, and long range travel.

duration['travel_time'] = pd.cut(duration.Length, 3, labels = ['short', 'medium', 'lor
duration_grp = duration.groupby(['Airline', 'travel_time'])['Delay'].apply(delay_perce
duration_grp.columns = duration_grp.columns.astype(str)
duration_grp.reset_index()
```

```
C:\Users\bhara\AppData\Local\Temp\ipykernel_11308\3062631771.py:4: FutureWarning: The
default of observed=False is deprecated and will be changed to True in a future versi
on of pandas. Pass observed=False to retain current behavior or observed=True to adop
t the future default and silence this warning.
  duration_grp = duration.groupby(['Airline', 'travel_time'])['Delay'].apply(delay_pe
rcent).reset_index().pivot(index = 'Airline', columns = 'travel_time').fillna(0)['Del
ay']
```

Out[86]:

	travel_time	Airline	short	medium	long
0		9E	39.78	0.00	0.00
1		AA	37.61	43.25	60.40
2		AS	32.58	38.17	0.00
3		B6	45.70	51.05	0.00
4		CO	52.82	64.95	66.87
5		DL	43.87	50.24	48.62
6		EV	40.20	50.00	0.00
7		F9	45.04	43.56	0.00
8		HA	30.16	40.48	0.00
9		MQ	34.81	27.42	0.00
10		OH	27.71	39.20	0.00
11		OO	45.40	53.03	0.00
12		UA	29.92	37.10	39.26
13		US	31.96	40.72	0.00
14		WN	69.12	77.61	0.00
15		XE	37.87	53.70	0.00
16		YV	24.37	25.86	0.00

In [87]:
```
airline_dict.rename(columns = {'Column':'Airline'}, inplace = True)
```

In [88]:
```
airline_dict.Description = airline_dict.Description.str.strip()
```

In [89]:
```
duration_grp = pd.merge(duration_grp, airline_dict[['Airline', 'Description']], how =
```

In [90]:
```
duration_grp
```

Out[90]:

	Airline	short	medium	long	Description
0	9E	39.78	0.00	0.00	Endeavor
1	AA	37.61	43.25	60.40	American Airlines
2	AS	32.58	38.17	0.00	Alaska
3	B6	45.70	51.05	0.00	Jetblue
4	CO	52.82	64.95	66.87	United Airlines (initially CO)
5	DL	43.87	50.24	48.62	Delta
6	EV	40.20	50.00	0.00	ExpressJet
7	F9	45.04	43.56	0.00	Frontier
8	HA	30.16	40.48	0.00	Hawaiian
9	MQ	34.81	27.42	0.00	Envoy
10	OH	27.71	39.20	0.00	PSA
11	OO	45.40	53.03	0.00	Skywest
12	UA	29.92	37.10	39.26	United Airlines
13	US	31.96	40.72	0.00	PSA (initially US Airway Express)
14	WN	69.12	77.61	0.00	Southwest
15	XE	37.87	53.70	0.00	JSX
16	YV	24.37	25.86	0.00	Mesa

In [91]:
```
cdt.Airline.nunique()
```

Out[91]:
```
17
```

In [92]:
```
duration_grp.short.min()
```

Out[92]:
```
24.37
```

In [93]:
```
long = duration_grp[duration_grp.long == duration_grp.long.min()].Description.values.t
print(len(long),'Airlines with minimum delays (0%) for long flights:\n',','.join(long)

medium= duration_grp[duration_grp.medium == duration_grp.medium.min()].Description.val
print('\n',len(medium),'Airline with minimum delays (0%) for medium flights:\n',','.jo

short = duration_grp[duration_grp.short == duration_grp.short.min()].Description.value
print('\n', len(short),'Airline with minimum delays (24.37%) for short flights:\n',','
```

```
13 Airlines with minimum delays (0%) for long flights:
 Endeavor,Alaska,Jetblue,ExpressJet,Frontier,Hawaiian,Envoy,PSA,Skywest,PSA (initiall
y US Airway Express),Southwest,JSX,Mesa

 1 Airline with minimum delays (0%) for medium flights:
 Endeavor

 1 Airline with minimum delays (24.37%) for short flights:
 Mesa
```

Here are the recommended airlines with minimum delays that are safest to travel for each kind of travel distance:

- **Long flights (0% delay)**: Endeavor, Alaska, Jetblue, ExpressJet, Frontier, Hawaiian, Envoy, PSA, Skywest, PSA (initially US Airway Express), Southwest, JSX, Mesa
- **Medium flights (0% delays)**: Endeavor
- **Short flights (24.37% delays)**: Mesa

- Do you observe any pattern in the time of departure of flights of long duration?

In [94]:
```
# Creating three equal width bins for length of flights and adding them to the dataset
cdt['duration'] = pd.cut(cdt.Length,3, labels = ['short', 'medium', 'long'])
```

In [95]:
```
cdt.head()
```

Out[95]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

In [96]:
```
cdt.isna().sum()
```

Out[96]:
```
id                           0
Airline                      0
Flight                       0
AirportFrom                  0
AirportTo                    0
DayOfWeek                    0
Time                         0
Length                       0
Delay                        0
type_source_airport          0
elevation_ft_source_airport  0
runway_count_source_airport  0
type_dest_airport            0
elevation_ft_dest_airport    0
runway_count_dest_airport    0
data_2021_source_airport     0
data_2021_dest_airport       0
Founded                      0
duration                     0
dtype: int64
```

In [97]:
```
pd.crosstab(cdt.Time, cdt.duration)['long']
```

Out[97]:
```
Time
10      0
15      0
20      0
21      0
25      0
       ..
1428    0
1430    0
1431    0
1435    0
1439    0
Name: long, Length: 1131, dtype: int64
```

In [98]:
```
y = pd.crosstab(cdt.Time, cdt.duration)['long'].index
x = pd.crosstab(cdt.Time, cdt.duration)['long'].values
```

In [99]:
```
filter_data = cdt.loc[cdt.duration == 'long', ['Time', 'duration']]
filter_data.head()
```

Out[99]:

	Time	duration
4232	565	long
4772	595	long
5738	650	long
6104	670	long
6477	690	long

In [100…
```
filter_data.Time.describe()
```

Out[100]:
```
count     559.000000
mean      840.635063
std       221.020092
min       540.000000
25%       670.000000
50%       717.000000
75%      1045.000000
max      1310.000000
Name: Time, dtype: float64
```

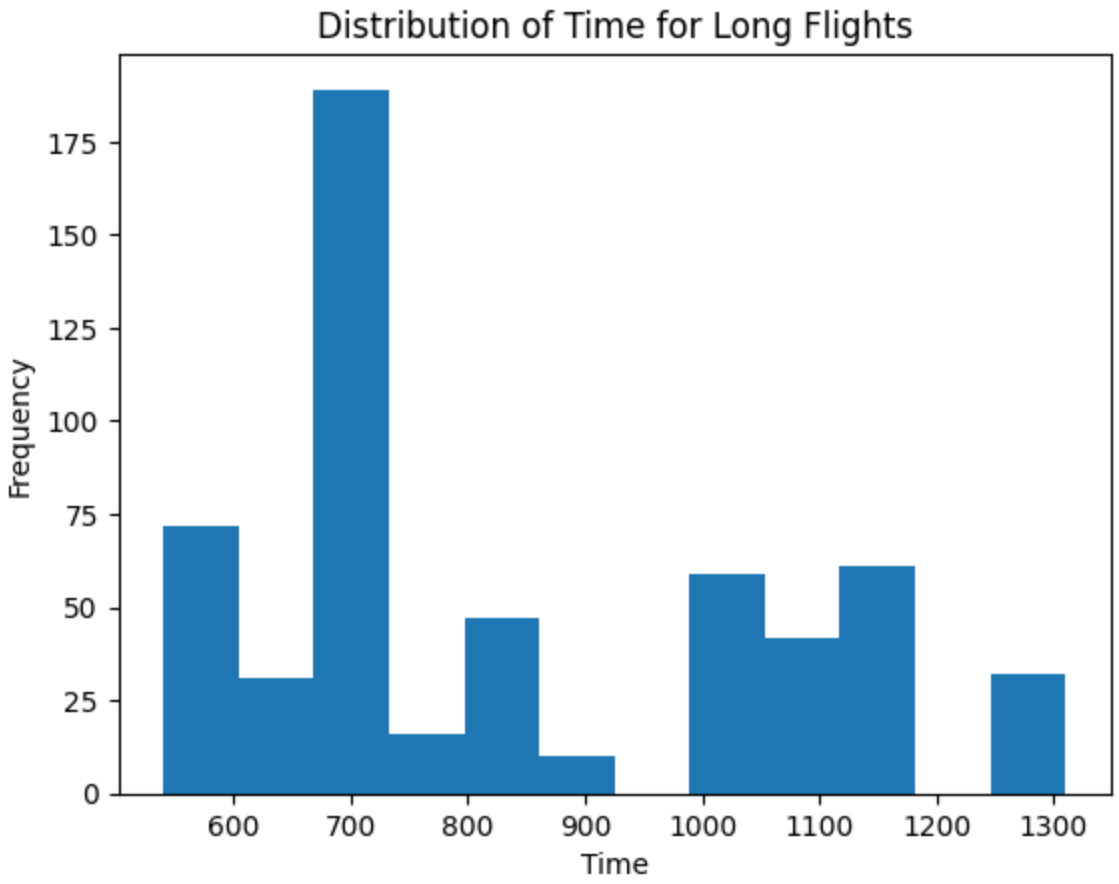
Departure Time Analysis

- **Departure Time Range**: The dataset's departure times span from a minimum of 540 minutes (equivalent to after midnight) to a maximum of 1310 minutes. This wide range indicates that flights in the dataset have departure times covering the entire day.
- **Average Departure Time**: The average departure time, represented by the mean of approximately 840.64 minutes, suggests that, on average, flights tend to depart around 8:40 AM (assuming midnight as the starting point).

- **Variability of Departure Times**: The standard deviation, which is approximately 221.02 minutes, implies that departure times exhibit a moderate level of variability or spread around the mean. This indicates that departure times are not tightly clustered around a specific time but rather show some degree of dispersion.

- **Quartiles**: Examining quartiles, the 25th percentile (first quartile) value of 670 minutes and the 75th percentile (third quartile) value of 1045 minutes provide insight into the distribution of departure times. Specifically, it reveals that 25% of the flights depart before 6:10 AM (approximately), while 75% of the flights depart before 5:45 PM (approximately).

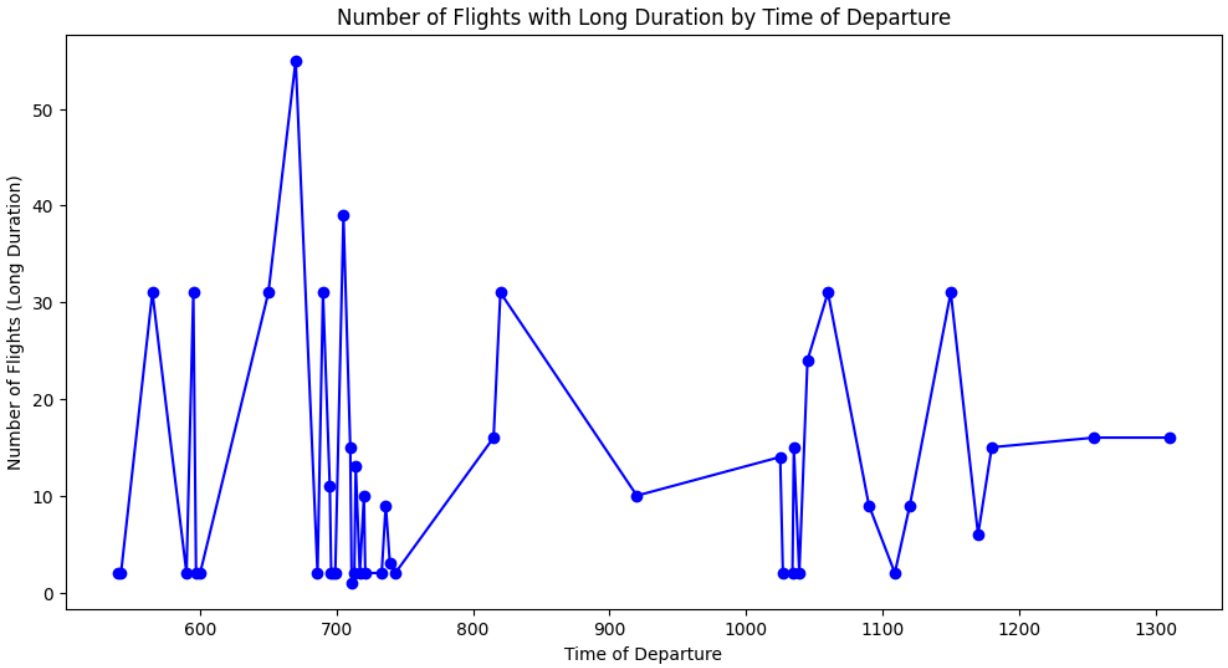
In [101…
```
# Plotting a bar chart for visualizing the distribution of time for long flights.
plt.hist(filter_data.Time, bins=12)
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.title('Distribution of Time for Long Flights')
plt.show()
```



- **Maximum** number of flights are departing at ~700 minutes, i.e. around 11:40 AM.
- There are **two empty time zones**, ~(900-1000) and ~(1160-1240) when no long distances flights depart.

In [102…
```
# Plotting a line graph to visualize the distribution of departure time of long flight
# Filter the data for flights with long duration
long_duration_flights = cdt[cdt['duration'] == 'long']

# Group the data by time and count the number of flights
```

```
time_counts = long_duration_flights['Time'].value_counts().sort_index()

# Plot the line chart
plt.figure(figsize=(12, 6))
plt.plot(time_counts.index, time_counts.values, marker='o', linestyle='-', color='blue

# Set the axis labels and title
plt.xlabel('Time of Departure')
plt.ylabel('Number of Flights (Long Duration)')
plt.title('Number of Flights with Long Duration by Time of Departure')

# Show the plot
plt.show()
```

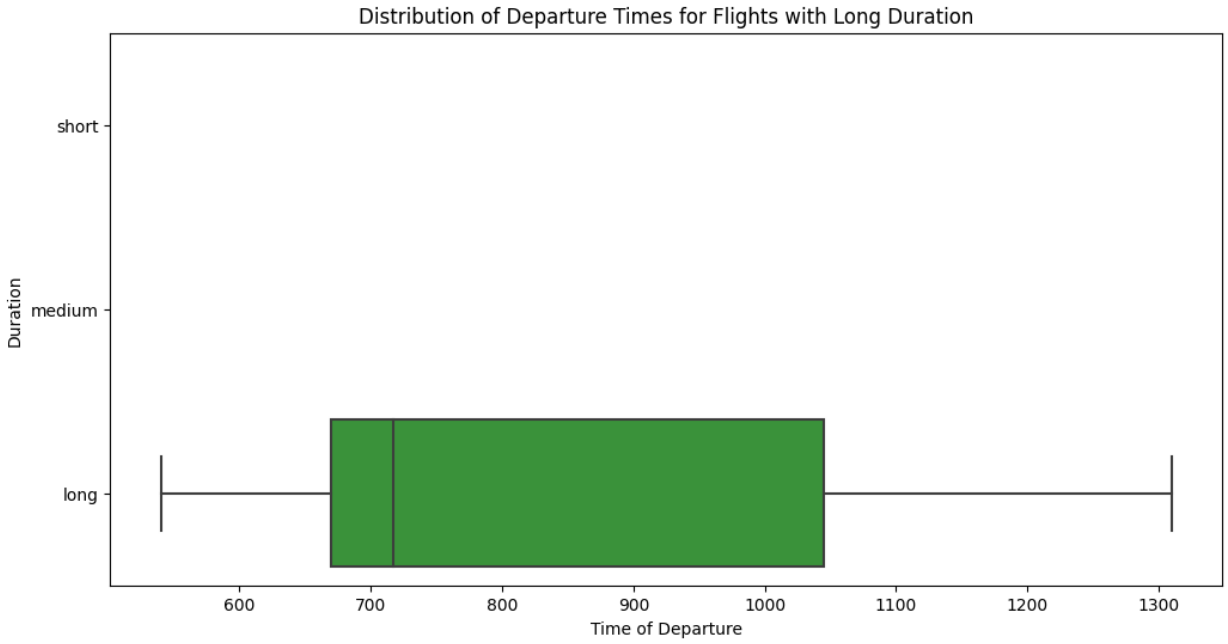


- Maxium number of flights depart between ~(660-710) minutes from midnight.
- The two empty zones identified above are not ture. There are some flights departing at almost every time stamp.

In [103…

```
# Filter the data for flights with long duration
long_duration_flights = cdt[cdt['duration'] == 'long']

# Plot the box plot
plt.figure(figsize=(12, 6))
sns.boxplot(data=long_duration_flights, x='Time', y='duration')

# Set the axis labels and title
plt.xlabel('Time of Departure')
plt.ylabel('Duration')
plt.title('Distribution of Departure Times for Flights with Long Duration')

# Show the plot
plt.show()
```

```
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\categorical.py:641: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  grouped_vals = vals.groupby(grouper)
```

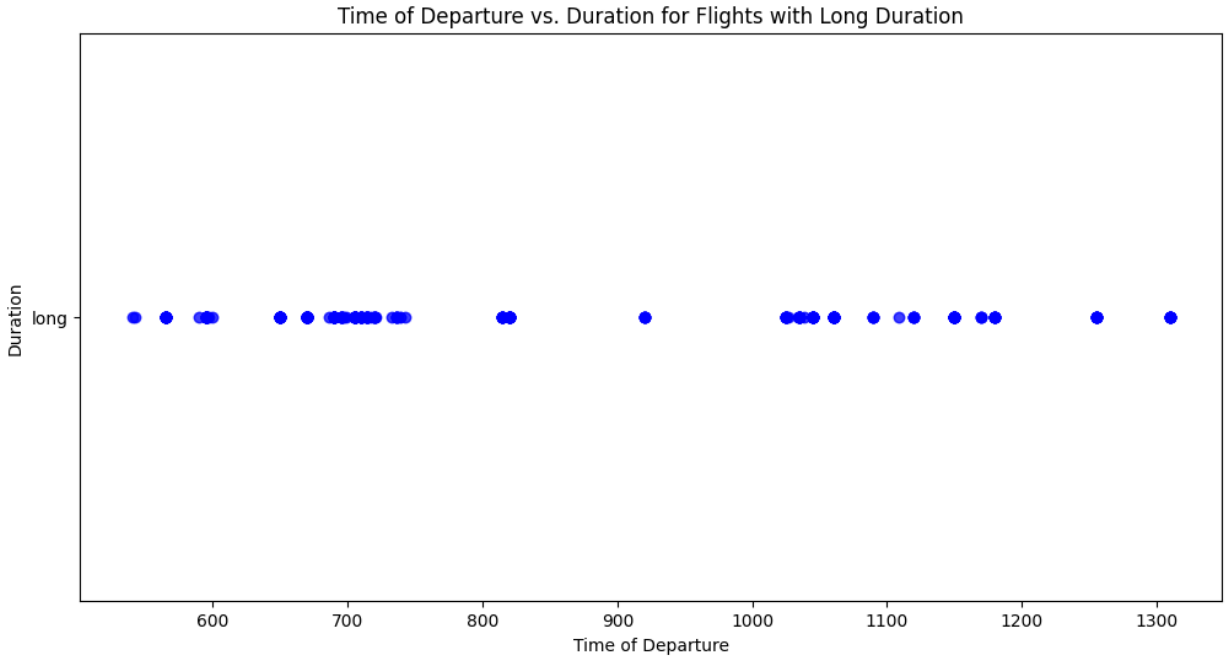


Observations till now:

- 75% of all the long duration flights depart before 6:00 PM in the evening.
- 50% of flights depart before 12:00 PM in the afternoon.
- The earliest long duration flight departs at 9:00 AM.
- The last long duration flight departs at 10:50 PM.

In [104…

```
# Plotting a scatter plot for visualizing the distribution of departure time of long f
# Filter the data for flights with long duration
long_duration_flights = cdt[cdt['duration'] == 'long']

# Plot the scatter plot
plt.figure(figsize=(12, 6))
plt.scatter(long_duration_flights['Time'], long_duration_flights['duration'], color='b

# Set the axis labels and title
plt.xlabel('Time of Departure')
plt.ylabel('Duration')
plt.title('Time of Departure vs. Duration for Flights with Long Duration')

# Show the plot
plt.show()
```

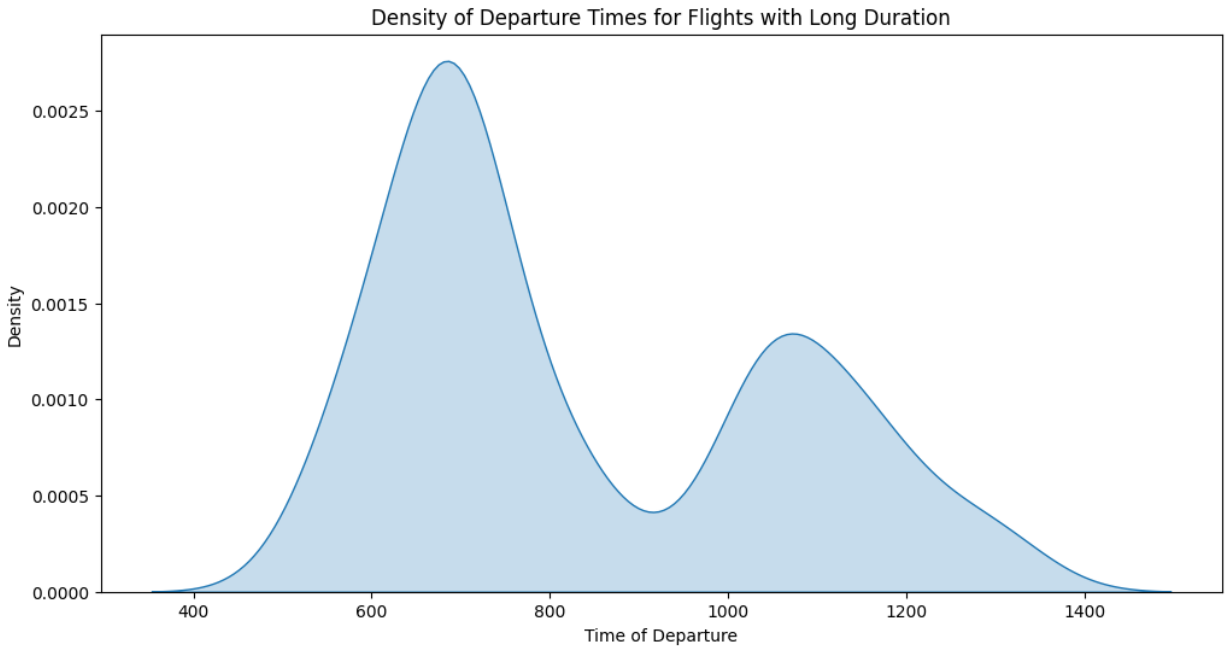


The clustered scatter plot confirms the observations made above.

In [105…

```
# Plotting a KDE plot for visualizing the distribution of departure time of long fligh
# Filter the data for flights with long duration
long_duration_flights = cdt[cdt['duration'] == 'long']

# Plot the KDE plot
plt.figure(figsize=(12, 6))
sns.kdeplot(data=long_duration_flights, x='Time', fill=True)

# Set the axis labels and title
plt.xlabel('Time of Departure')
plt.ylabel('Density')
plt.title('Density of Departure Times for Flights with Long Duration')

# Show the plot
plt.show()
```

```
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_old
core.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed i
n a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\_old
core.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed i
n a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



The KDE plot confirms the observatios made above.

In [106…

```
from scipy.stats import ttest_ind, f_oneway
import pandas as pd

# Convert the 'Time' column to numeric data type
cdt['Time'] = pd.to_numeric(cdt['Time'], errors='coerce')

# Filter the data for flights with long duration and other durations
long_duration_flights = cdt[cdt['duration'] == 'long']
other_duration_flights = cdt[cdt['duration'] != 'long']

# Perform t-test to compare the means of departure times
t_statistic, p_value = ttest_ind(long_duration_flights['Time'], other_duration_flights

# Perform ANOVA to compare the means of departure times across different durations
f_statistic, p_value_anova = f_oneway(cdt['Time'], pd.Categorical(cdt['duration']).coc

print("T-test: T-statistic =", t_statistic, "p-value =", p_value)
print("ANOVA: F-statistic =", f_statistic, "p-value =", p_value_anova)
```

```
T-test: T-statistic = 3.335989828914716 p-value = 0.0008500226450162412
ANOVA: F-statistic = 4299971.900179982 p-value = 0.0
```

- **T-test Result**: The t-statistic of 3.336 indicates a significant difference in the means of departure times between long-duration flights and other-duration flights. This suggests that there is a distinct pattern or variation in the departure times of long-duration flights compared to other-duration flights.
- **ANOVA Result**: The ANOVA test indicates a significant difference in departure times across different flight durations, including long-duration flights. This suggests that the departure times of long-duration flights are not only different from other-duration flights but also show variation within the group of long-duration flights themselves. This further supports

the existence of patterns or systematic differences in the departure times of long-duration flights.

- How large hubs compare to medium hubs in terms of count of delayed flights? Use appropriate visualization to represent your findings.

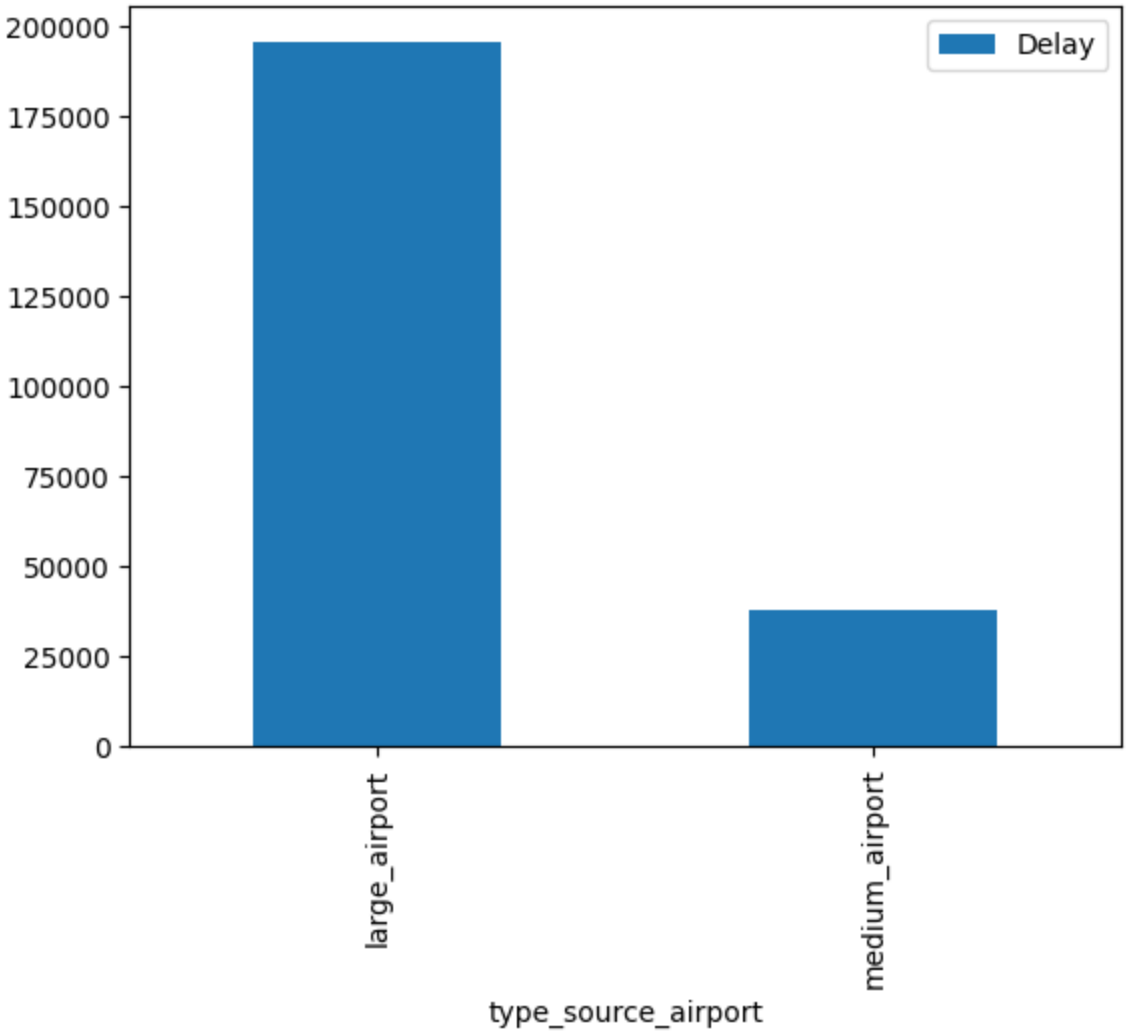
In [107…
```
cdt.head()
```

Out[107]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

In [108…
```
cdt.groupby('type_source_airport')[['Delay']].agg('sum').plot.bar()
```

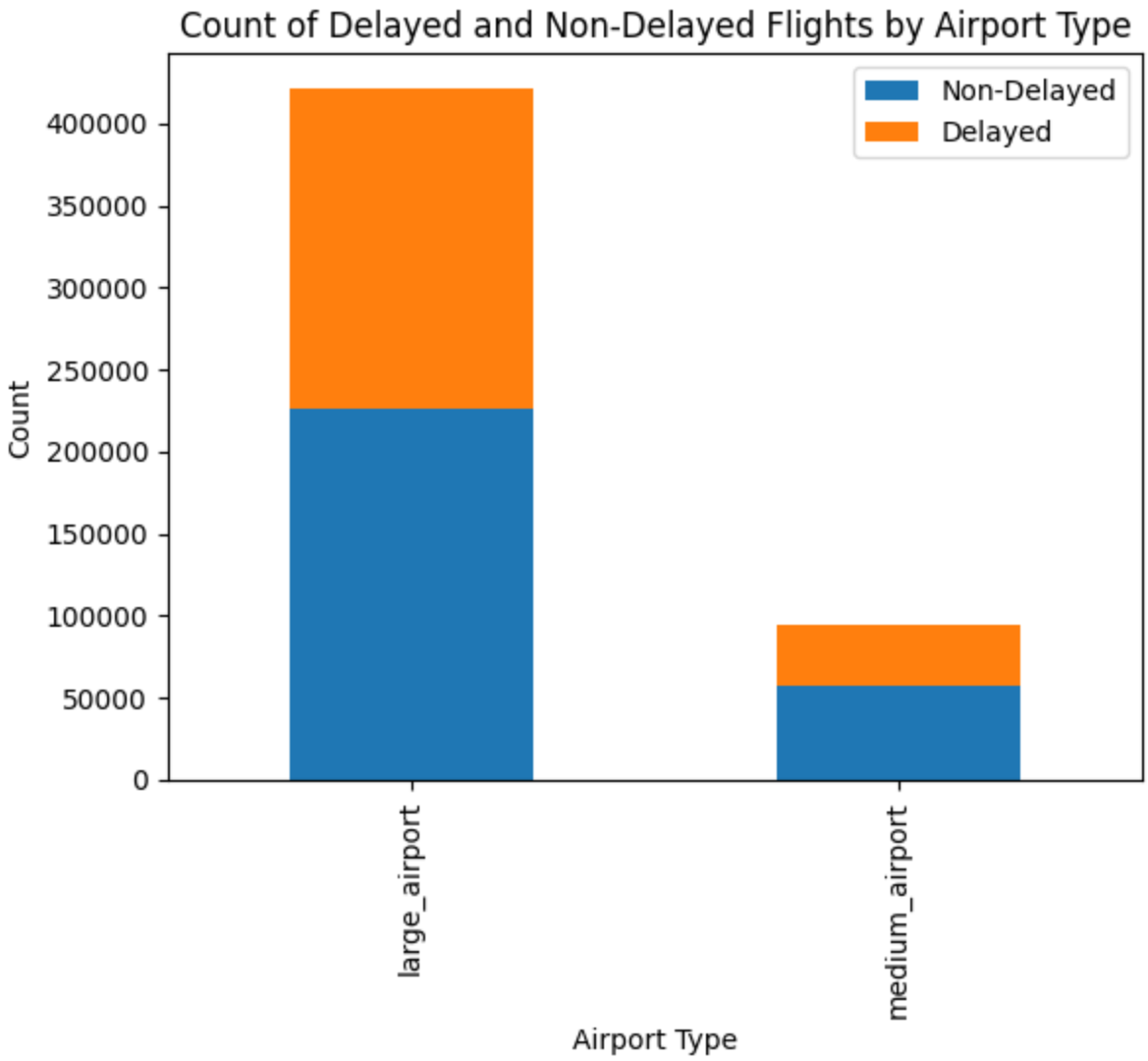
Out[108]:
```
<Axes: xlabel='type_source_airport'>
```



Large airports have **higher count of delayed flights**. But this could be due to the fact that **they serve a larger number of people with more number of flights and routes**.

In [109…

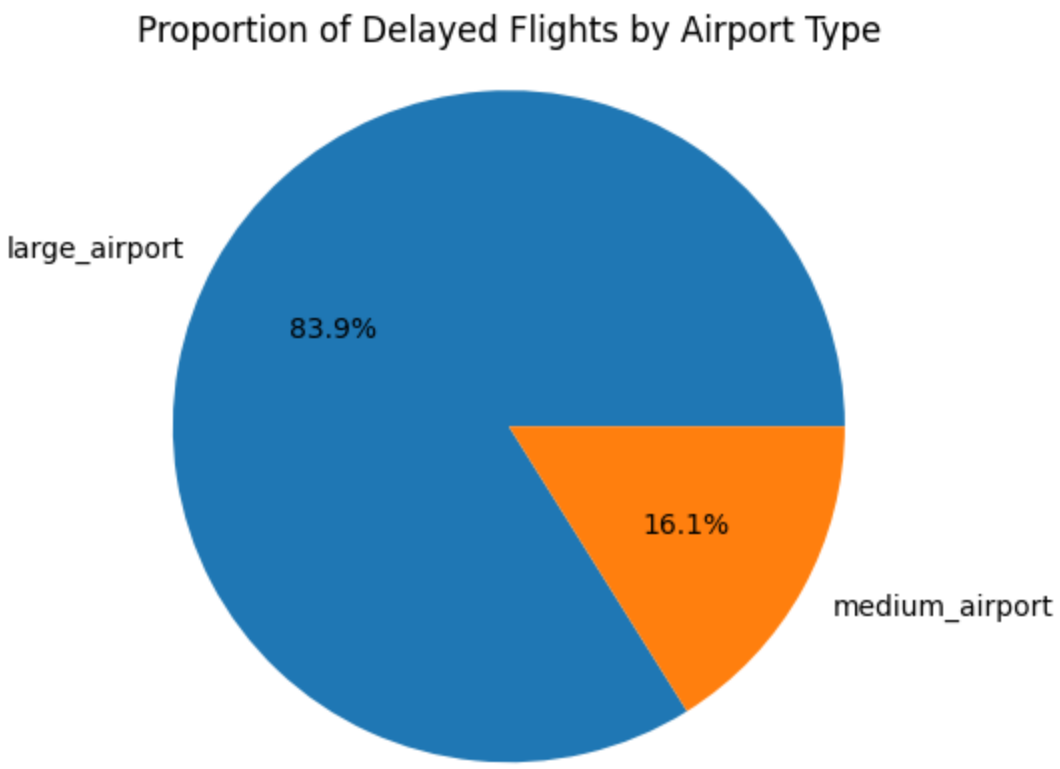
```
# Creating a stacked bar plot to visualize the balance between delayed and non-delayed
delay_counts = cdt.groupby('type_source_airport')['Delay'].value_counts().unstack()
delay_counts.plot(kind='bar', stacked=True)
plt.xlabel('Airport Type')
plt.ylabel('Count')
plt.title('Count of Delayed and Non-Delayed Flights by Airport Type')
plt.legend(['Non-Delayed', 'Delayed'])
plt.show()
```



- The plot shows us that there is a significant difference between the number of flights large and medium airports handle. **Medium airports have a higher ratio of non-delayed flights**. This could mean that given the traffic each type of hub handles, medium airports are better at ensuring timely take-offs.

In [110…

```
delay_counts = cdt[cdt['Delay'] == 1].groupby('type_source_airport').size()
delay_counts.plot(kind='pie', autopct='%1.1f%%')
plt.axis('equal')
plt.title('Proportion of Delayed Flights by Airport Type')
plt.show()
```



In [111…
```
# Calculate the proportion of delayed flights for each airport type
delay_proportion = cdt.groupby('type_source_airport')['Delay'].mean()*100
```

In [112…
```
delay_proportion
```

Out[112]:
```
type_source_airport
large_airport     46.408639
medium_airport    39.601642
Name: Delay, dtype: float64
```

Large airports have a 46% delay where as medium airports have 39% delay. Therefore, we can conclude that **large_airports have a higher chance of a delayed flights than medium_airports**.

- For large hubs, forecast the number of passengers for 2022 using simple moving average method.

In [113…
```
# Lets develop a series of traffic data for large airports.

cols = ['iata_code']+ final_hub_data.columns[final_hub_data.columns.str.startswith('da
```

In [114…
```
final_hub_data.head()
```

Out[114]:

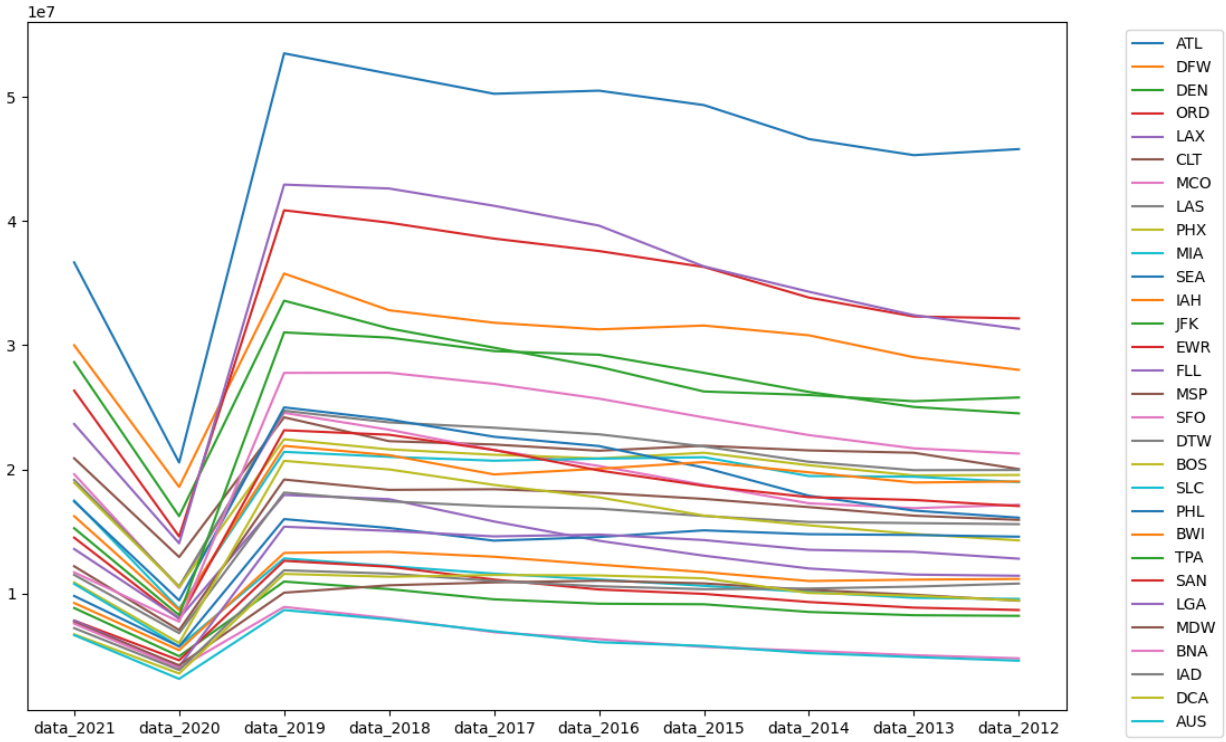
	rank	hub_type	airports	iata_code	ciity_served	state	data_2021	data_2020	data_2019	data
0	1	large	Hartsfield–Jackson Atlanta International Airport	ATL	Atlanta	GA	36676010	20559866	53505795	518
1	2	large	Dallas/Fort Worth International Airport	DFW	Dallas & Fort Worth	TX	30005266	18593421	35778573	328
2	3	large	Denver International Airport	DEN	Denver	CO	28645527	16243216	33592945	313
3	4	large	O'Hare International Airport	ORD	Chicago	IL	26350976	14606034	40871223	398
4	5	large	Los Angeles International Airport	LAX	Los Angeles	CA	23663410	14055777	42939104	426

In [115…
```
time_series = final_hub_data.loc[final_hub_data.hub_type == 'large', cols].set_index('
```

In [116…
```
time_series['ATL']
```

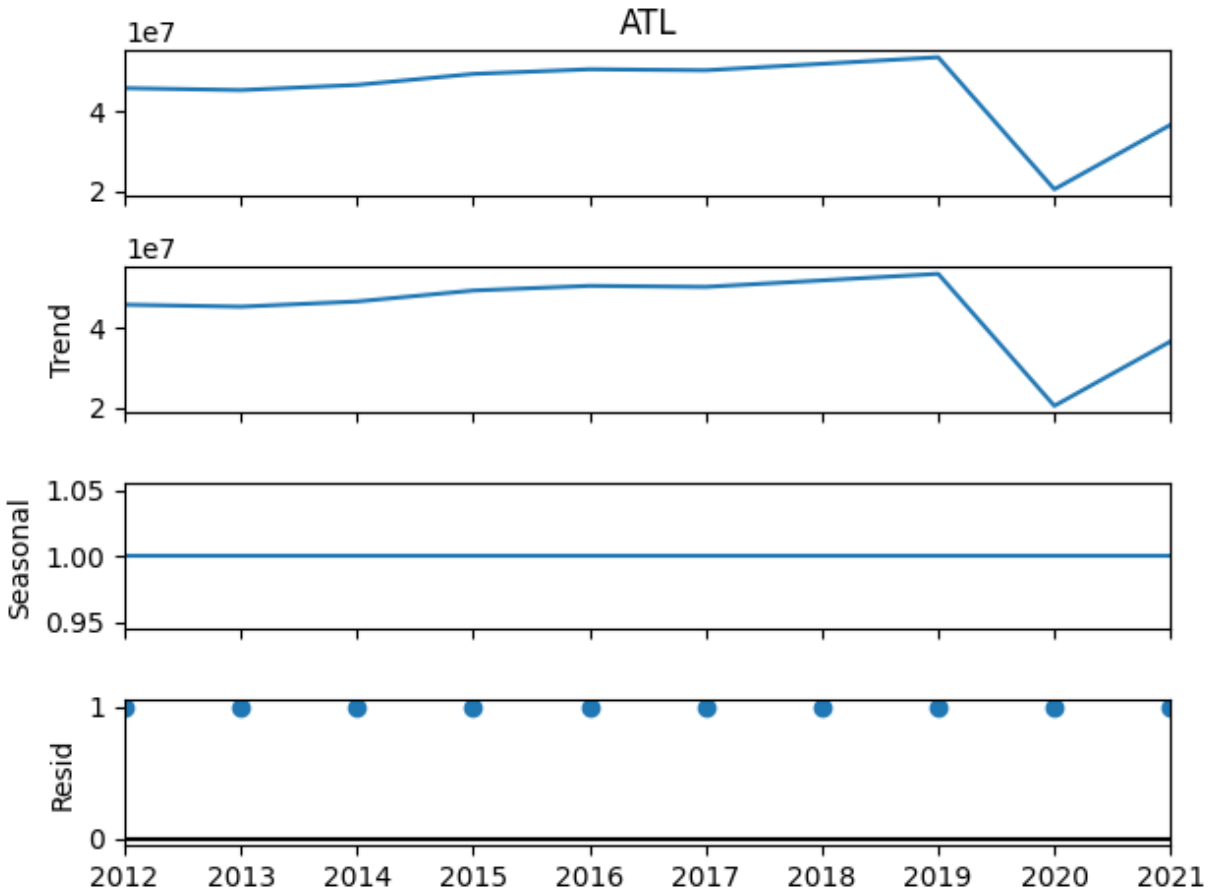
Out[116]:
```
data_2021    36676010
data_2020    20559866
data_2019    53505795
data_2018    51865797
data_2017    50251964
data_2016    50501858
data_2015    49340732
data_2014    46604273
data_2013    45308407
data_2012    45798928
Name: ATL, dtype: int64
```

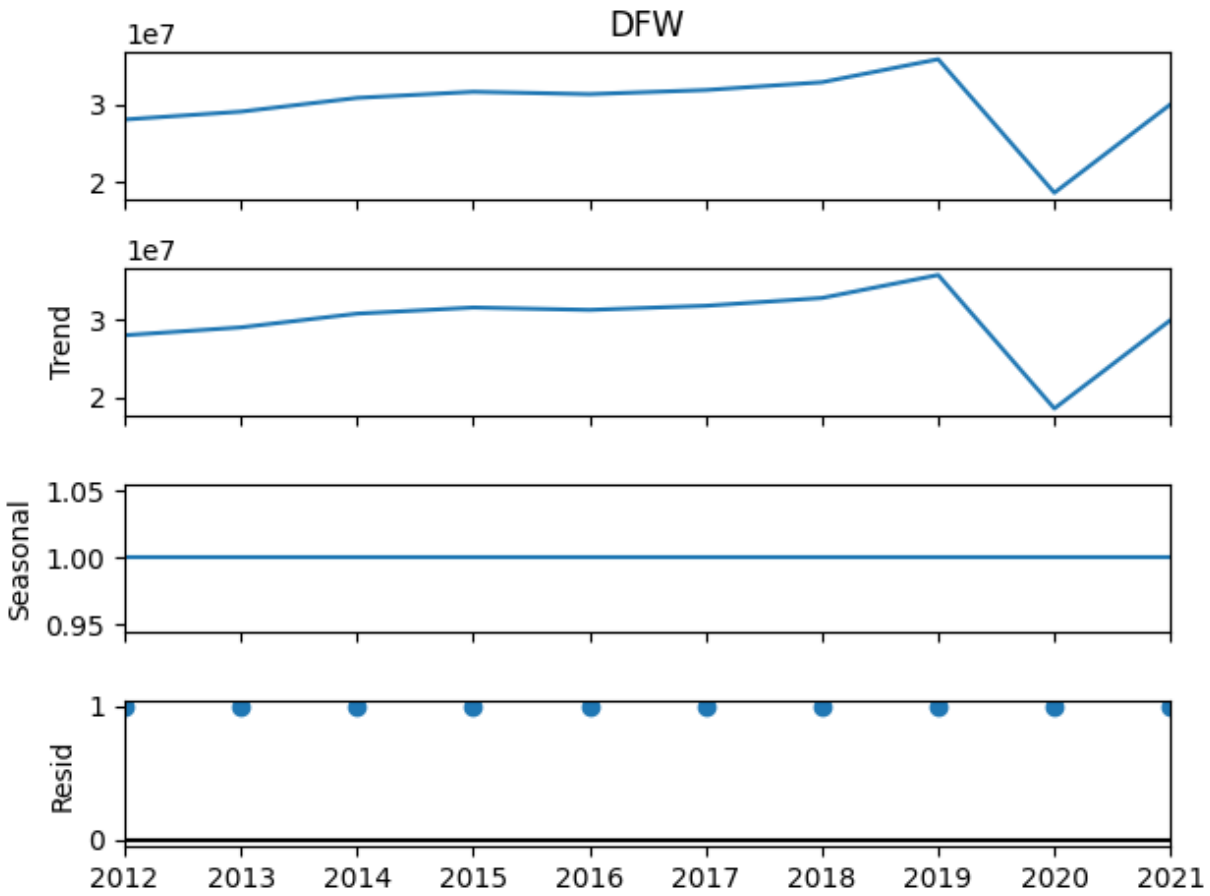
In [117…
```
# Plotting the time series for each airport.
plt.figure(figsize=(12, 8))
for ser in time_series.columns:
    plt.plot(time_series[ser], label=ser)

# Set the legend outside the chart
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

plt.show(block=True)
```

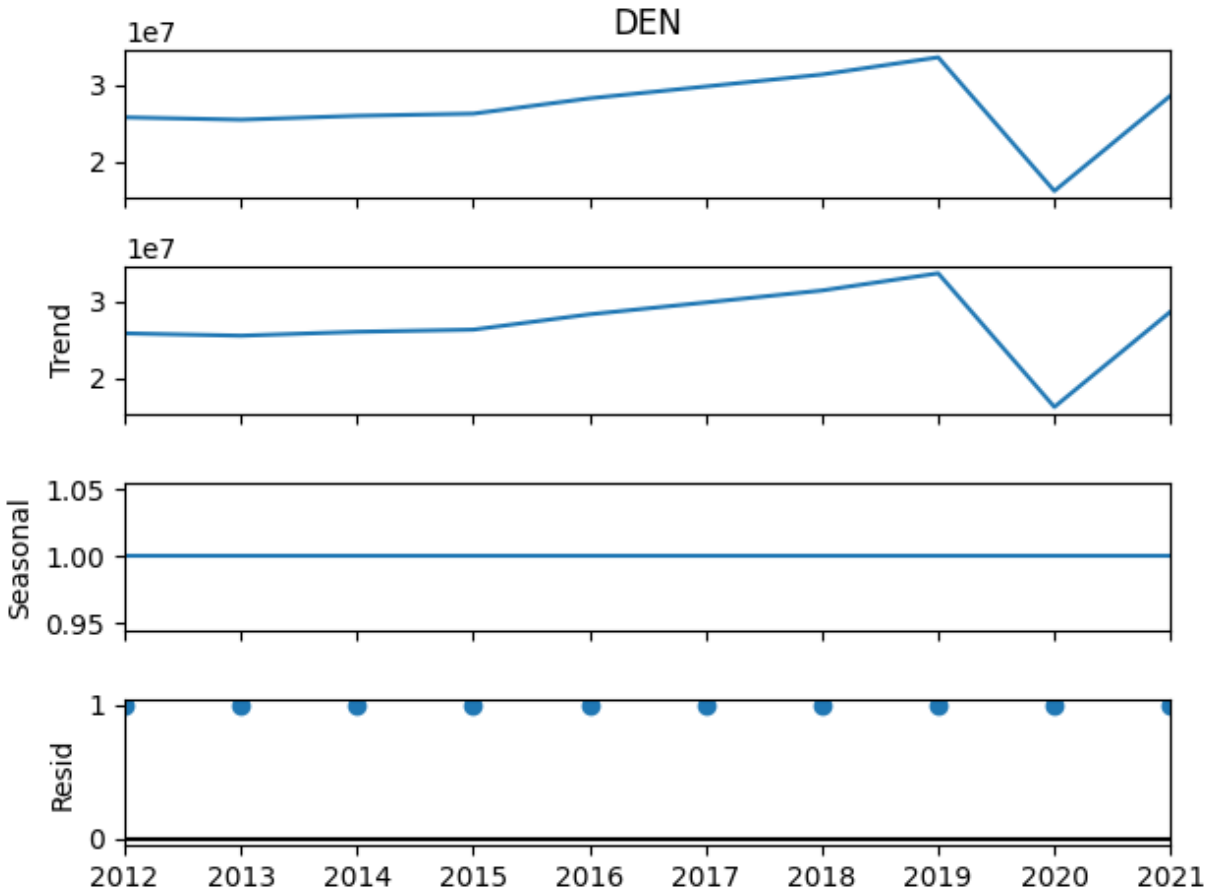


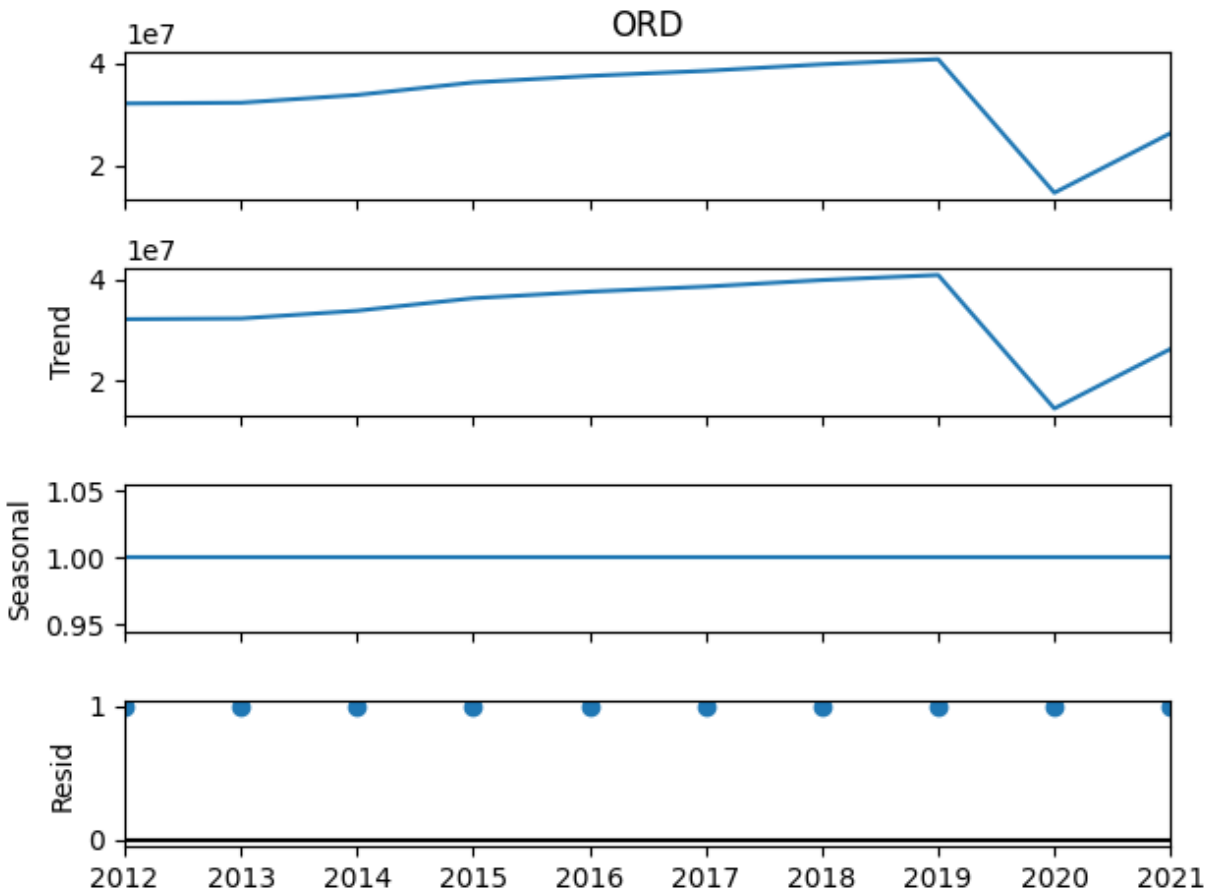
In [118…

```
# Performing seasonal decomposition on first 4 airports.
for ser in time_series.columns[:4]:
    series = time_series[ser].copy()
    series.index = pd.to_datetime(series.index.str.replace('data_',''))
    series.sort_index(inplace = True)
    decomposition = sm.tsa.seasonal_decompose(series, model='multiplicative')
    decomposition.plot()
plt.show()
```


ATL
1e7
4
2
1e7
Trend
4
2
Seasonal
1.05
1.00
0.95
Resid
1
0
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021


DFW
1e7
3
2
1e7
Trend
3
2
Seasonal
1.05
1.00
0.95
Resid
1
0
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021





Observations:

- The **trend** for all four airports is that the traffic is on an uptrend after a major drop in 2019 due to COVID.
- The seasonality shows that there are no recurring patterns in the traffic data.
- The residual shows that there are no abnormnal patterns in the traffic data.

In [119…

```
error = {}
forecast_2022 = {}
f = {}
wind_min = {}
win_min_mape = {}
for ser in time_series.columns:
    series = time_series[ser].copy()
    series.index = pd.to_datetime(series.index.str.replace('data_',''))
    series.sort_index(inplace = True)
    test = series[-1:]
    train = series[:-1]
    err_temp = {}
    fore_2022 = {}
    for window in range(2,10):
        forecast = series.rolling(window).mean()
        # accuracy
        mape = round(mean_absolute_percentage_error(test, forecast[-1:]),4)
        err_temp.update({window : mape})
        # forecast for 2022
        fore_2022.update({window : series[-window:].mean()})
    err_ser = pd.Series(err_temp)
    min_wind = err_ser[(err_ser == err_ser.min())].index.values[0]
    forecast_2022.update({ser : round(series[-min_wind:].mean(),2)})
    wind_min.update({ser : min_wind})
    win_min_mape.update({ser :err_temp[min_wind] })
    f.update({ser :pd.Series(fore_2022).round(2) })
    error.update({ser : err_ser})

    # forecast for 2022
```

In [120…

```
win_min_mape
```

Out[120]:
```
{'ATL': 0.0065,
 'DFW': 0.0015,
 'DEN': 0.023,
 'ORD': 0.0351,
 'LAX': 0.1362,
 'CLT': 0.0006,
 'MCO': 0.0077,
 'LAS': 0.0212,
 'PHX': 0.0001,
 'MIA': 0.0182,
 'SEA': 0.0076,
 'IAH': 0.0389,
 'JFK': 0.1912,
 'EWR': 0.0486,
 'FLL': 0.0122,
 'MSP': 0.0502,
 'SFO': 0.1697,
 'DTW': 0.0559,
 'BOS': 0.1502,
 'SLC': 0.0062,
 'PHL': 0.0719,
 'BWI': 0.0082,
 'TPA': 0.0029,
 'SAN': 0.0686,
 'LGA': 0.1655,
 'MDW': 0.0453,
 'BNA': 0.0598,
 'IAD': 0.0595,
 'DCA': 0.0844,
 'AUS': 0.0017}
```

In [121…
```
sma_forecast = pd.DataFrame(f)
sma_error = pd.DataFrame(error)
```

In [122…
```
sma_prediction = pd.DataFrame(forecast_2022.values(), index = forecast_2022.keys(), cc
sma_prediction['window_used'] = wind_min.values()
sma_prediction['mape_at_window'] = win_min_mape.values()
```

In [123…
```
sma_prediction.sort_values('forecast_2022', ascending=False)
```

Out[123]:

	forecast_2022	window_used	mape_at_window
ATL	36913890.33	3	0.0065
DFW	30049928.50	6	0.0015
DEN	27986853.33	6	0.0230
ORD	27276077.67	3	0.0351
LAX	26886097.00	3	0.1362
CLT	20913675.38	8	0.0006
LAS	19566943.50	4	0.0212
MCO	19467356.50	8	0.0077
PHX	18942662.60	5	0.0001
JFK	18193272.00	3	0.1912
SEA	17298122.67	3	0.0076
MIA	17182193.50	4	0.0182
IAH	15610229.33	3	0.0389
EWR	15220095.33	3	0.0486
FLL	13765555.89	9	0.0122
MSP	12824682.00	3	0.0502
BOS	12548215.33	3	0.1502
DTW	12161020.00	3	0.0559
SLC	10729401.33	6	0.0062
PHL	10526616.67	3	0.0719
SFO	9735202.00	2	0.1697
BWI	9329867.67	3	0.0082
LGA	9122674.67	3	0.1655
TPA	8872731.89	9	0.0029
SAN	8374302.67	3	0.0686
IAD	7658216.67	3	0.0595
MDW	7333000.33	3	0.0453
DCA	7300226.67	3	0.0844
BNA	7140261.25	4	0.0598
AUS	6677268.60	5	0.0017

As we saw in the line chart earlier, the airports: ATL, DFW, DEN,and ORD will witness the maximum traffic in 2022.

1. Use hypothesis testing strategies to discover:

- If the airport's altitude has anything to do with flight delays for incoming and departing flights
- If the number of runways at an airport affects flight delays
- If the duration of a flight (length) affects flight delays Hint: Test this from the perspective of both the source and destination airports

---

- If the airport's altitude has anything to do with flight delays for incoming and departing flights

For outgoing flights.

In [124…
```
# 2 sample t test for outgoing flights with the following hypothesis.
# H0 : avg elevation for Delayed flights - avg elevation for not Delayed flights = 0
# Ha : avg elevation for Delayed flights - avg elevation for not Delayed flights != 0
```

In [125…
```
sample1 = cdt[cdt.Delay == 1].elevation_ft_source_airport
sample2 = cdt[cdt.Delay == 0].elevation_ft_source_airport
```

In [126…
```
t, p = stats.ttest_ind(sample1, sample2)
```

In [127…
```
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'

print(result)
```

```
reject null
```

There is a statistically significant difference in the average elevation between delayed and not delayed flights. This suggests that the **elevation of the source airport may play a role in flight delays**.

For incoming flights.

In [128…
```
# 2 sample t test for incoming flights with the following hypothesis.
# H0 : avg elevation for Delayed flights - avg elevation for not Delayed flights = 0
# Ha : avg elevation for Delayed flights - avg elevation for not Delayed flights != 0
```

In [129…
```
sample1 = cdt[cdt.Delay == 1].elevation_ft_dest_airport
sample2 = cdt[cdt.Delay == 0].elevation_ft_dest_airport
```

In [130…
```
t, p = stats.ttest_ind(sample1, sample2)
```

In [131…
```
if p < 0.05:
    result = 'reject null'
else :
```

```
    result = 'fail to reject null'

print(result)
```

```
reject null
```

There is a statistically significant difference in the average elevation between delayed and not delayed flights. This suggests that the **elevation of the destination airport may play a role in flight delays**.

- If the number of runways at an airport affects flight delays

In [132…
```
# t test :
# H0 : avg runway count for delayed filghts - avg runway count for non delayed flights
# Ha : avg runway count for delayed filghts - avg runway count for non delayed flights
```

In [133…
```
cdt.head()
```

Out[133]:

	id	Airline	Flight	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport
0	1	CO	269	SFO	IAH	3	15	205	1	large_airport
1	2	US	1558	PHX	CLT	3	15	222	1	large_airport
2	3	AA	2400	LAX	DFW	3	20	165	1	large_airport
3	4	AA	2466	SFO	DFW	3	20	195	1	large_airport
4	5	AS	108	ANC	SEA	3	30	202	0	large_airport

In [134…
```
s1 = cdt[cdt.Delay == 1].runway_count_source_airport
s2 = cdt[cdt.Delay == 0].runway_count_source_airport
```

In [135…
```
t, p = stats.ttest_ind(s1, s2)
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'
print(result)
```

```
reject null
```

In [136…
```
s1 = cdt[cdt.Delay == 1].runway_count_dest_airport
s2 = cdt[cdt.Delay == 0].runway_count_dest_airport
```

In [137…
```
t, p = stats.ttest_ind(s1, s2)
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'
print(result)
```

```
reject null
```

The resulting output "reject null" for both tests suggests that there is evidence to support the alternative hypothesis, indicating that **the average runway count for delayed flights is**

significantly lower than the average runway count for non-delayed flights in both the source and destination airports.

- If the duration of a flight (length) affects flight delays Hint: Test this from the perspective of both the source and destination airports

In [138…
```
# t test :
# H0 : avg duration for delayed filghts - avg duration for non delayed flights = 0
# Ha : avg duration for delayed filghts - avg duration for non delayed flights != 0
```

In [139…
```
s1 = cdt[cdt.Delay == 1].Length
s2 = cdt[cdt.Delay == 0].Length
```

In [140…
```
t, p = stats.ttest_ind(s1, s2)
```

In [141…
```
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'
print(result)
```

```
reject null
```

The result is 'reject null', indicating that there is a **significant difference in the average duration between delayed and non-delayed flights**.

In [142…
```
cs = pd.crosstab(cdt.duration, cdt.Delay)
cs
```

Out[142]:

Delay	0	1
duration		
short	253865	203612
medium	28983	29198
long	252	307

In [143…
```
chi, p, df, ex = stats.chi2_contingency(cs)
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'
print(result)
```

```
reject null
```

The result of the chi-square test was "reject null," it means that there is evidence to suggest a **significant relationship between the duration of flights and flight delays**.

In [144…
```
# t test :
# H0 : avg duration for delayed filghts - avg duration for non delayed flights <= 0
# Ha : avg duration for delayed filghts - avg duration for non delayed flights > 0
```

In [145…
```
t, p = stats.ttest_ind(s1, s2)
if p < 0.05:
    result = 'reject null'
else :
    result = 'fail to reject null'
print(result)
```

```
reject null
```

The result of the t-test was "reject null," it means that there is evidence to suggest a significant difference in the average duration between delayed flights and non-delayed flights. Specifically, **the average duration of delayed flights is significantly greater than the average duration of non-delayed flights.**

- Find the correlation matrix between the flight delay predictors, create a heatmap to visualize this, and share your findings

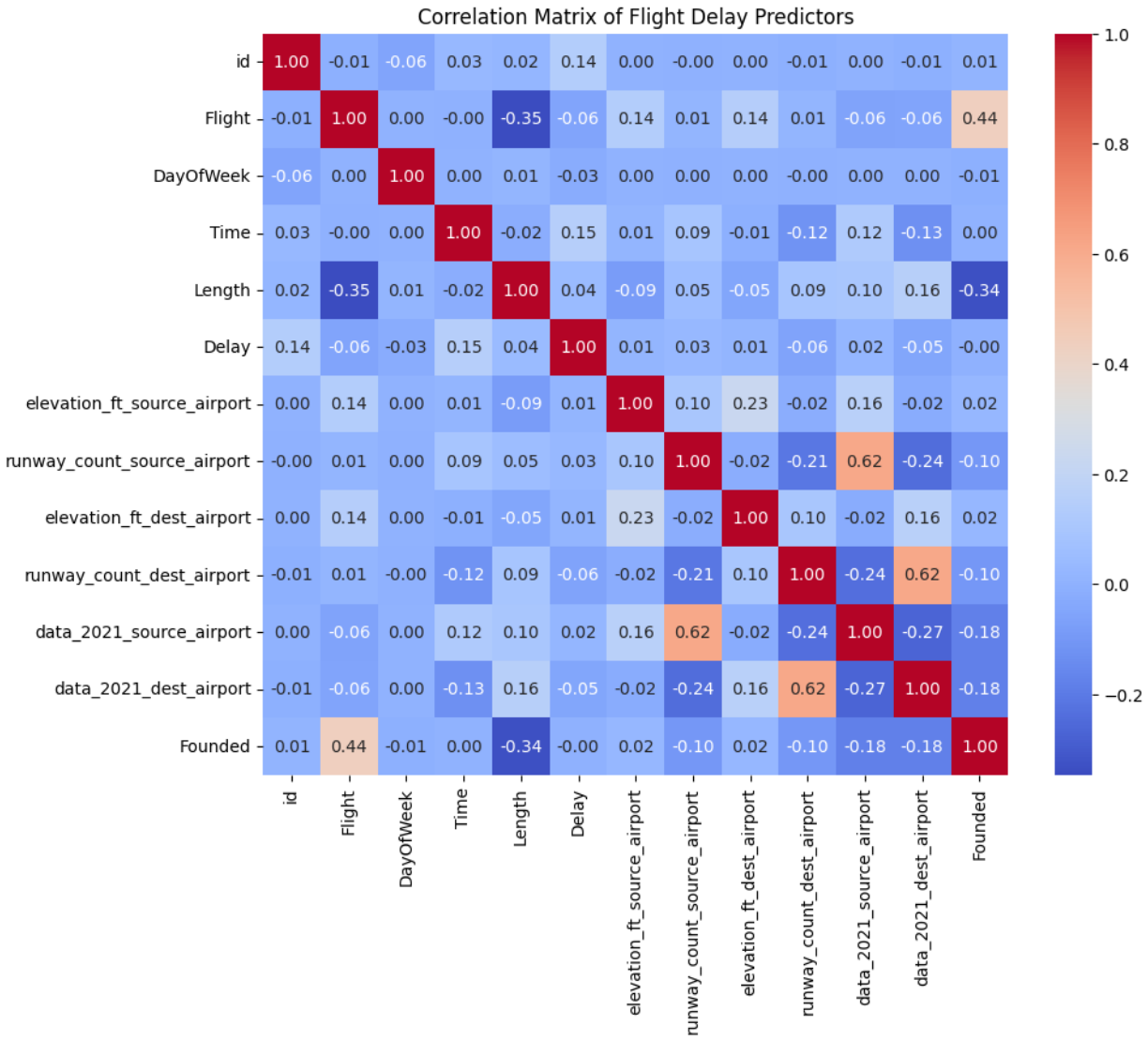
In [146…
```
cdt.columns
```

Out[146]:
```
Index(['id', 'Airline', 'Flight', 'AirportFrom', 'AirportTo', 'DayOfWeek',
       'Time', 'Length', 'Delay', 'type_source_airport',
       'elevation_ft_source_airport', 'runway_count_source_airport',
       'type_dest_airport', 'elevation_ft_dest_airport',
       'runway_count_dest_airport', 'data_2021_source_airport',
       'data_2021_dest_airport', 'Founded', 'duration'],
      dtype='object')
```

In [147…
```
numeric_columns = cdt.select_dtypes(include=['float64', 'int64']).columns
correlation_with_delay = cdt[numeric_columns].corr()['Delay']

correlation_with_delay
```

Out[147]:
```
id                           0.140434
Flight                      -0.057371
DayOfWeek                   -0.025832
Time                         0.149801
Length                       0.040162
Delay                        1.000000
elevation_ft_source_airport  0.012551
runway_count_source_airport  0.029099
elevation_ft_dest_airport    0.013180
runway_count_dest_airport   -0.061431
data_2021_source_airport     0.020315
data_2021_dest_airport      -0.051622
Founded                     -0.003102
Name: Delay, dtype: float64
```

In [148…
```
# Calculate the correlation matrix
correlation_matrix = cdt[numeric_columns].corr()

# Create heatmap with correlation values
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, fmt=".2f")
plt.title('Correlation Matrix of Flight Delay Predictors')
plt.show()
```



Observations:

- The **'Time'** variable has the highest positive correlation with 'Delay' (0.149801), indicating that flights scheduled for later times may have a higher likelihood of being delayed.
- The **'Length'** variable has a slightly positive correlation with 'Delay' (0.040162), suggesting that longer flights may be associated with a slightly higher chance of delays.
- The **'Flight'** variable shows a weak negative correlation with 'Delay' (-0.057371), implying that certain flight numbers may be associated with a lower likelihood of delays.
- The **'runway_count_dest_airport'** variable has a moderate negative correlation with 'Delay' (-0.061431), indicating that airports with a higher number of runways at the destination may be associated with a lower probability of delays.
- Other variables such as 'DayOfWeek', 'elevation_ft_source_airport', 'elevation_ft_dest_airport', 'data_2021_source_airport', 'data_2021_dest_airport', and 'Founded' show weak correlations with 'Delay'.

Exporting the final dataset to csv for future use.

In [151…
```
cdt.to_csv('usairlinesfinaldata1.csv', index = False)
```

Machine Learning

- Use OneHotEncoder and OrdinalEncoder to deal with categorical variables

In [152…
```
cdt.isna().sum()
```

Out[152]:
```
id                              0
Airline                         0
Flight                          0
AirportFrom                     0
AirportTo                       0
DayOfWeek                       0
Time                            0
Length                          0
Delay                           0
type_source_airport             0
elevation_ft_source_airport     0
runway_count_source_airport     0
type_dest_airport               0
elevation_ft_dest_airport       0
runway_count_dest_airport       0
data_2021_source_airport        0
data_2021_dest_airport          0
Founded                         0
duration                        0
dtype: int64
```

In [153…
```
cdt.drop(columns = ['id', 'Flight','duration'], inplace = True)
```

In [154…
```
cdt.head()
```

Out[154]:

	Airline	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_source_airport	elevation_f
0	CO	SFO	IAH	3	15	205	1	large_airport	
1	US	PHX	CLT	3	15	222	1	large_airport	
2	AA	LAX	DFW	3	20	165	1	large_airport	
3	AA	SFO	DFW	3	20	195	1	large_airport	
4	AS	ANC	SEA	3	30	202	0	large_airport	

In [155…
```
cdt.type_dest_airport.unique()
```

Out[155]:
```
array(['large_airport', 'medium_airport'], dtype=object)
```

In [156…
```
ordinal = OrdinalEncoder(categories=[['medium_airport', 'large_airport'],['medium_airp
ordinal.fit(cdt[['type_source_airport', 'type_dest_airport']])
```

Out[156]:
```
OrdinalEncoder(categories=[['medium_airport', 'large_airport'],
                           ['medium_airport', 'large_airport']])
```

In [157…
```
cdt[['type_source_airport', 'type_dest_airport']] = ordinal.transform(cdt[['type_sourc
```

In [158…
```
model_data = cdt.drop(columns = ['Airline', 'AirportFrom', 'AirportTo'])
```

In [159…
```
model_data.shape
```

Out[159]:
```
(516217, 13)
```

In [160…
```
dummy = pd.get_dummies(model_data)
dummy.shape
```

Out[160]:
```
(516217, 13)
```

In [161…
```
dummy.Founded = 2023 - dummy.Founded
dummy.head()
```

Out[161]:

	DayOfWeek	Time	Length	Delay	type_source_airport	elevation_ft_source_airport	runway_count_s
0	3	15	205	1	1.0	13.0	
1	3	15	222	1	1.0	1135.0	
2	3	20	165	1	1.0	125.0	
3	3	20	195	1	1.0	13.0	
4	3	30	202	0	1.0	152.0	

Perform the following model building steps:

- Apply logistic regression (use stochastic gradient descent optimizer) and decision tree models
- Use the stratified five fold method to build and validate the models **Note:** Make sure you use standardization effectively, ensuring no data leakage and leverage pipelines to have a cleaner code
- Use RandomizedSearchCV for hyperparameter tuning, and use k fold for cross validation
- Keep a few data points (10%) for prediction purposes to evaluate how you would make the final prediction, and do not use t his data for testing or validation **Note:** The final prediction will be based on the voting (majority class by 5 models created using the stratified 5 fold method)
- Compare the results of logistic regression and decision tree classifier

In [162…
```
model_data.reset_index(drop = True, inplace = True)
```

In [163…
```
# Generating a randon set of indices.

np.random.seed(12)
deploy_idx = np.random.choice(model_data.index, replace = False, size = 5000)
```

In [164…
```
deploy = model_data.loc[deploy_idx]
```

In [165…
```
X_deploy = deploy.drop(columns = 'Delay')
```

In [166…
```
model_dev = model_data.loc[~model_data.index.isin(deploy.index)]
```

In [167…
```
deploy.reset_index(drop = True, inplace = True)
model_dev.reset_index(drop = True, inplace = True)
```

In [168…
```
X = model_dev.drop(columns = 'Delay')
y = model_dev.Delay
```

In [169…
```
folds = StratifiedKFold(n_splits=5, shuffle = True, random_state=12)
accuracy_train = {}
accuracy_test = {}
final_predictions_sgd = {}
i = 1
for train_index, test_index in folds.split(X,y):
    print('iter ', i)
    train, test = model_dev.loc[train_index,], model_dev.loc[test_index,]
    sc = StandardScaler()
    sgd = SGDClassifier()

    # define search space

    space = dict()
    space['sgd__penalty'] = ['l1', 'l2', 'elasticnet']
    space['sgd__l1_ratio'] = [0,.1,.2,.8,1]
    space['sgd__alpha'] = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 1000,10000]
    space['sgd__learning_rate'] = ['constant', 'adaptive']
    space['sgd__eta0']=[1e-5, 1e-4, 1e-3, 1e-2, 1e-1 , 2e-1, 3e-1, 5e-1, 8e-1, 4e-1, 8

    pipe = Pipeline([('sc',sc), ('sgd', sgd)])

    # define search
    search = RandomizedSearchCV( pipe, space, scoring='accuracy',
                                cv=5, refit=True, return_train_score = True,
                                random_state = 12, n_jobs = -1, n_iter = 2
                           )

    # execute search
    X_train = train.drop(columns = 'Delay')
    y_train = train.Delay

    result = search.fit(X_train, y_train)

    train_pred = result.predict(X_train)

    X_test = test.drop(columns = 'Delay')
    y_test = test.Delay
    test_pred = result.predict(X_test)
    final_predictions_sgd.update({'Fold{}'.format(i):result.predict(X_deploy)})

    # get rmse for each fold for train data
    accuracy_train.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_train,
    accuracy_test.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_test, y_
    i += 1
```

```
iter  1
iter  2
iter  3
iter  4
iter  5
```

In [170…

```
folds = StratifiedKFold(n_splits=5, shuffle = True, random_state=12)
dt_accuracy_train = {}
dt_accuracy_test = {}
final_predictions_dt = {}
i = 1
for train_index, test_index in folds.split(X,y):
    print('iter ', i)

    train, test = model_dev.loc[train_index,], model_dev.loc[test_index,]

    sc = StandardScaler()
    dt = DecisionTreeClassifier()


    # define search space
    space = dict()
    space['dt__min_samples_split'] = [25000, 30000, 35000, 40000, 45000, 50000, 60000
    space['dt__min_samples_leaf'] = [10000, 15000, 20000]

    pipe = Pipeline([('sc',sc), ('dt', dt)])

    # define search
    search = RandomizedSearchCV( pipe, space, scoring='accuracy',
                                cv=5, refit=True, return_train_score = True,
                                random_state = 12, n_jobs = -1, n_iter = 2
                            )

    # execute search
    X_train = train.drop(columns = 'Delay')
    y_train = train.Delay

    result = search.fit(X_train, y_train)

    train_pred = result.predict(X_train)

    X_test = test.drop(columns = 'Delay')
    y_test = test.Delay
    test_pred = result.predict(X_test)
    final_predictions_dt.update({'Fold{}'.format(i):result.predict(X_deploy)})

    # get rmse for each fold for train data
    dt_accuracy_train.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_trai
    dt_accuracy_test.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_test,
    i += 1
```

```
iter  1
iter  2
iter  3
iter  4
iter  5
```

In [171…

```
# compare results :
train_results = pd.DataFrame ({'sgd' : accuracy_train.values(), 'dt': dt_accuracy_trai
```

```
                                    index = ['Fold {}'.format(i) for i in range(1,6)])
train_results
```

Out[171]:

	sgd	dt
Fold 1	57.163	61.643
Fold 2	57.168	61.669
Fold 3	57.154	61.649
Fold 4	57.228	61.487
Fold 5	57.105	61.597

In [172…

```
test_results = pd.DataFrame ({'sgd' : accuracy_test.values(), 'dt': dt_accuracy_test.v
                                    index = ['Fold {}'.format(i) for i in range(1,6)])
test_results
```

Out[172]:

	sgd	dt
Fold 1	57.173	61.431
Fold 2	57.182	61.304
Fold 3	57.221	61.928
Fold 4	56.891	61.421
Fold 5	57.313	61.456

In [173…

```
final_predictions_dt
```

Out[173]:

```
{'Fold1': array([0, 0, 0, ..., 0, 0, 0], dtype=int64),
 'Fold2': array([0, 0, 0, ..., 0, 0, 0], dtype=int64),
 'Fold3': array([0, 1, 0, ..., 0, 1, 0], dtype=int64),
 'Fold4': array([0, 1, 1, ..., 0, 1, 0], dtype=int64),
 'Fold5': array([0, 1, 0, ..., 0, 1, 0], dtype=int64)}
```

In [174…

```
final_predictions_sgd
```

Out[174]:

```
{'Fold1': array([0, 1, 0, ..., 1, 1, 0], dtype=int64),
 'Fold2': array([0, 1, 0, ..., 1, 1, 0], dtype=int64),
 'Fold3': array([0, 1, 0, ..., 1, 1, 0], dtype=int64),
 'Fold4': array([0, 1, 0, ..., 1, 1, 0], dtype=int64),
 'Fold5': array([0, 1, 0, ..., 1, 1, 0], dtype=int64)}
```

In [175…

```
folds = StratifiedKFold(n_splits=5, shuffle = True, random_state=12)
xgb_accuracy_train = {}
xgb_accuracy_test = {}
final_predictions_xgb = []

i = 1
for train_index, test_index in folds.split(X,y):
    print('iter ', i)
    train, test = model_dev.loc[train_index,], model_dev.loc[test_index,]
    sc = StandardScaler()
    xgb_r = XGBClassifier(random_state = 12, use_label_encoder = False)

    # define search space
    space = dict()
```

```
    space['xgb_r__n_estimators'] = [40,50,60]
    space['xgb_r__max_depth'] = [3,4,5]
    space['xgb_r__colsample_bytree']:[0.4,.5,.6]
    space['xgb_r__lambda'] = [.0001,.002,.0004,.0003]
    space['xgb_r__alpha'] = [.01,.02,.1,.4]

    pipe = Pipeline([('sc',sc), ('xgb_r', xgb_r)])

    # define search
    search = RandomizedSearchCV( pipe, space, scoring='neg_root_mean_squared_error',
                                 cv=5, refit=True, return_train_score = True,
                                 random_state = 12, n_jobs = -1, n_iter = 2
                               )

    # execute search
    X_train = train.drop(columns = 'Delay')
    y_train = train.Delay

    result = search.fit(X_train, y_train)

    train_pred = result.predict(X_train)

    X_test = test.drop(columns = 'Delay')
    y_test = test.Delay
    test_pred = result.predict(X_test)

    final_predictions_xgb.append(result.predict(X_deploy))

    # get rmse for each fold for train data
    xgb_accuracy_train.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_tra
    xgb_accuracy_test.update({'Fold{}'.format(i): round(accuracy_score(y_true = y_test
    i += 1
```

```
iter  1
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgboost\dat
a.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future vers
ion. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(data):
iter  2
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgboost\dat
a.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future vers
ion. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(data):
iter  3
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgboost\dat
a.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future vers
ion. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(data):
iter  4
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgboost\dat
a.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future vers
ion. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(data):
iter  5
C:\Users\bhara\AppData\Local\Programs\Python\Python311\Lib\site-packages\xgboost\dat
a.py:520: FutureWarning: is_sparse is deprecated and will be removed in a future vers
ion. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if is_sparse(data):
```

In [176…
```
xgb_accuracy_train
```

Out[176]:
```
{'Fold1': 0.646,
 'Fold2': 0.645,
 'Fold3': 0.645,
 'Fold4': 0.646,
 'Fold5': 0.647}
```

In [177…
```
xgb_accuracy_train.values()
```

Out[177]:
```
dict_values([0.646, 0.645, 0.645, 0.646, 0.647])
```

In [178…
```
train_results['xgb'] = xgb_accuracy_train.values()
test_results['xgb'] = xgb_accuracy_test.values()
```

In [179…
```
train_results
```

Out[179]:

	sgd	dt	xgb
Fold 1	57.163	61.643	0.646
Fold 2	57.168	61.669	0.645
Fold 3	57.154	61.649	0.645
Fold 4	57.228	61.487	0.646
Fold 5	57.105	61.597	0.647

In [180…
```
test_results
```

Out[180]:

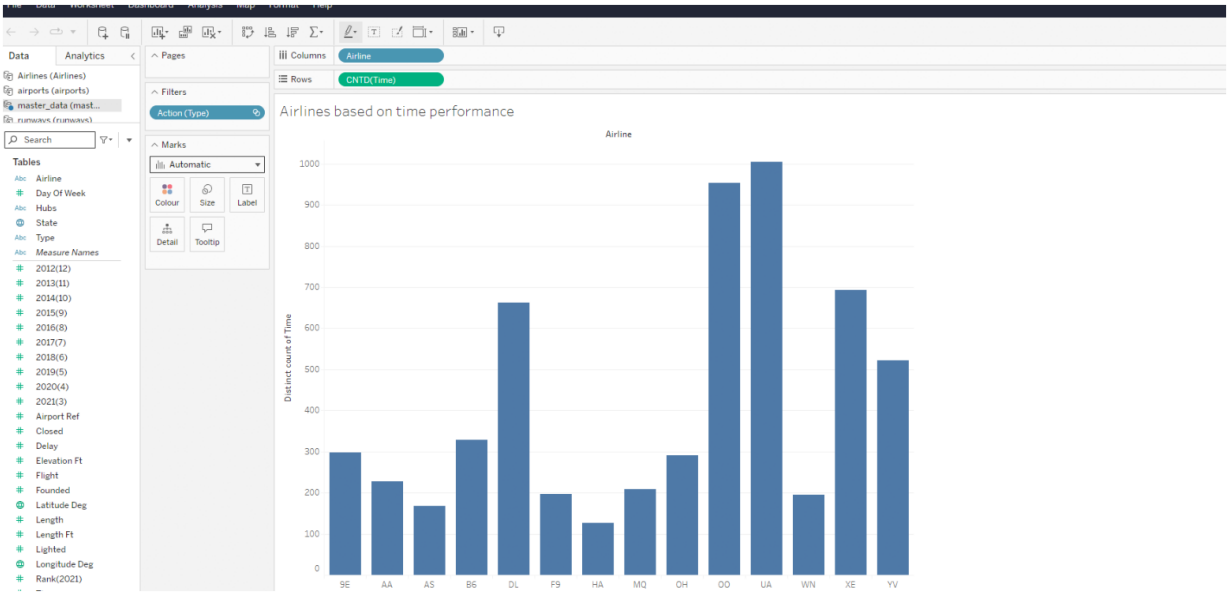
	sgd	dt	xgb
Fold 1	57.173	61.431	0.642
Fold 2	57.182	61.304	0.642
Fold 3	57.221	61.928	0.646
Fold 4	56.891	61.421	0.642
Fold 5	57.313	61.456	0.646

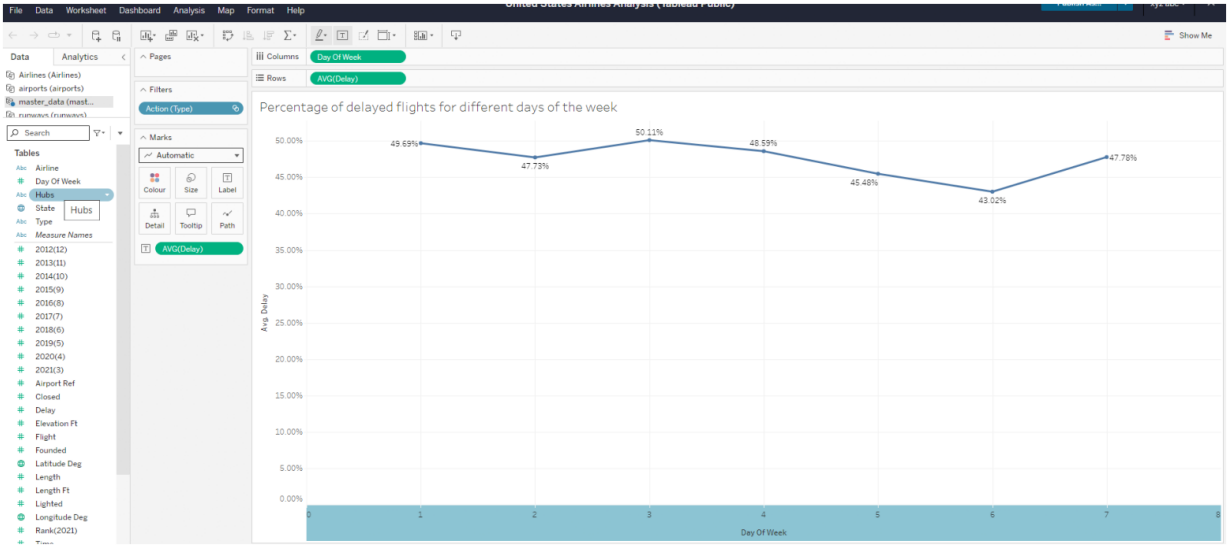
The logistic regression and decision tree models perform similarly and achieve higher accuracy compared to the XGBoost model.

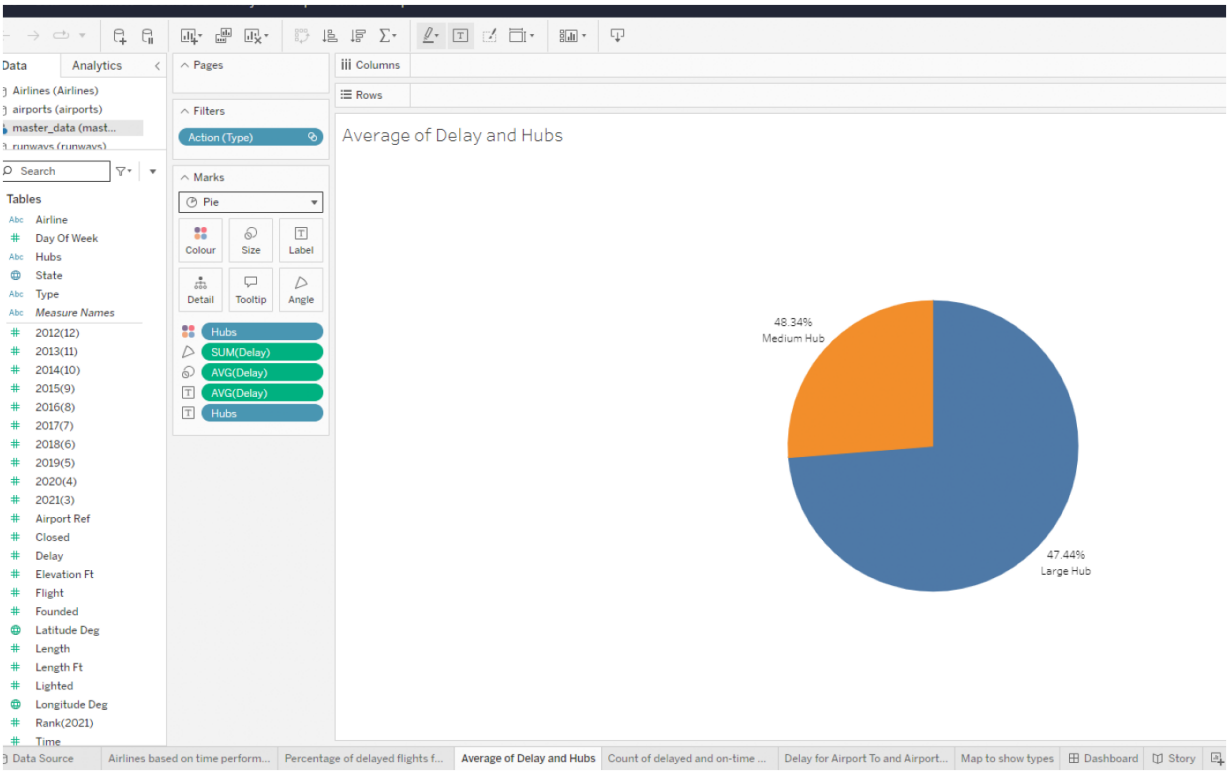
Tableau Story

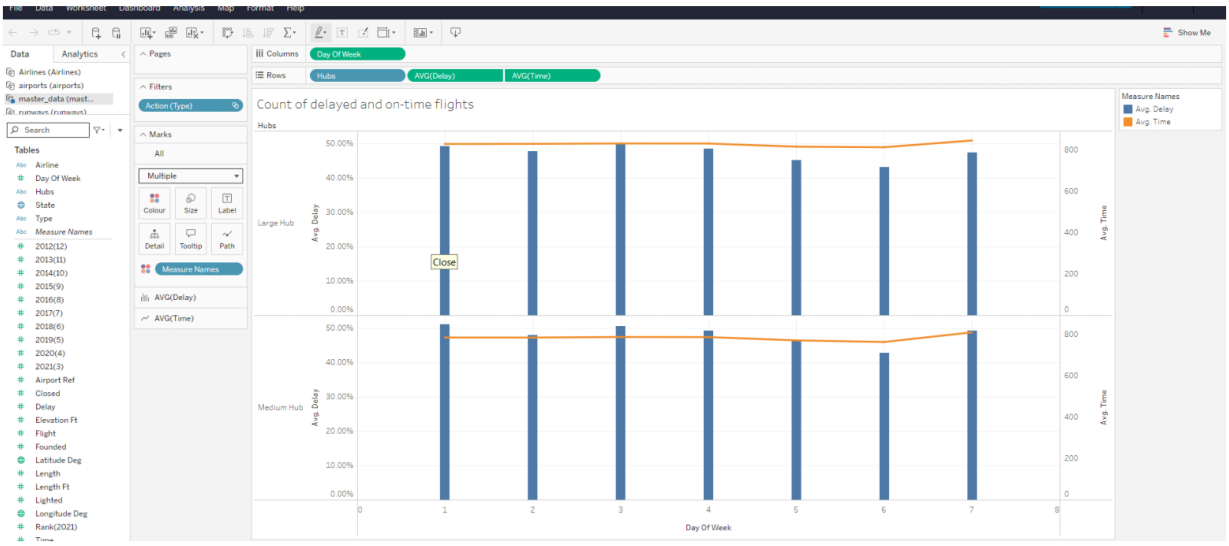
In [181…
```
import os
from matplotlib import pyplot as plt
from matplotlib.image import imread

# Path to the folder containing images
folder_path = r'C:\Users\bhara\OneDrive\Pictures\Screenshots\airlines tableau'

# List all files in the folder
files = os.listdir(folder_path)
```

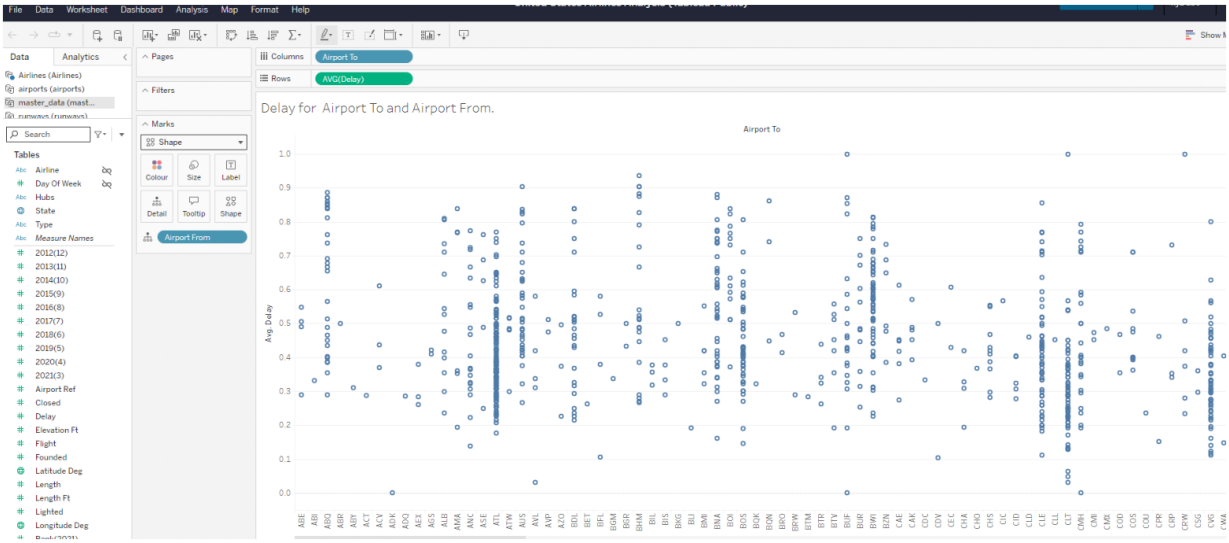
```
# Filter out non-image files
image_files = [f for f in files if f.lower().endswith(('.png', '.jpg', '.jpeg'))]

# Display each image
for image_file in image_files:
    image_path = os.path.join(folder_path, image_file)
    image = imread(image_path)
    plt.figure(figsize=(20,20))
    plt.imshow(image)
    plt.axis('off')
    plt.show()
```

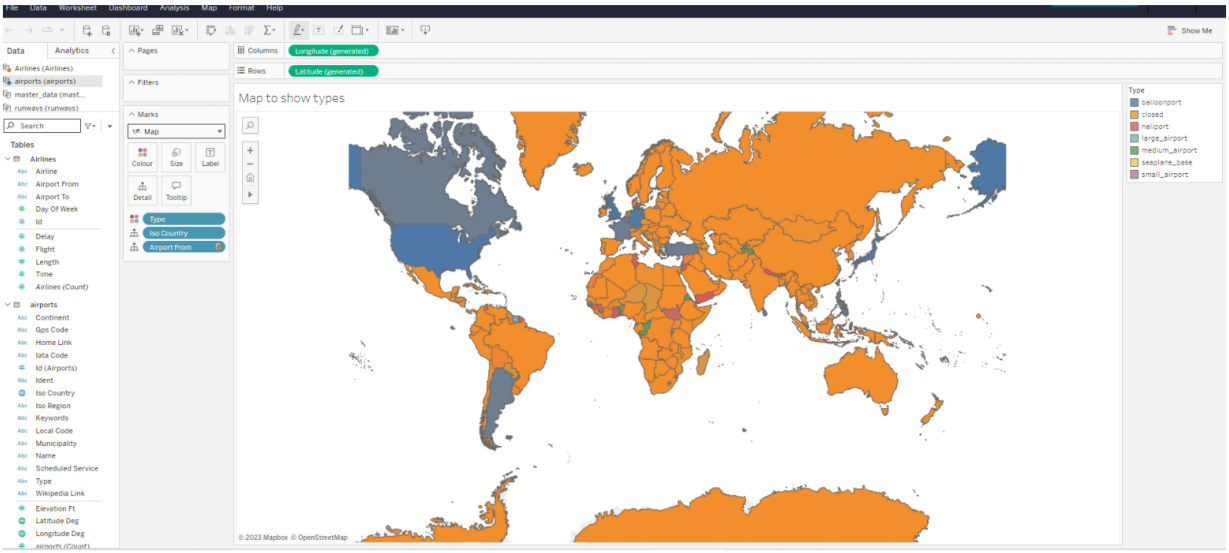




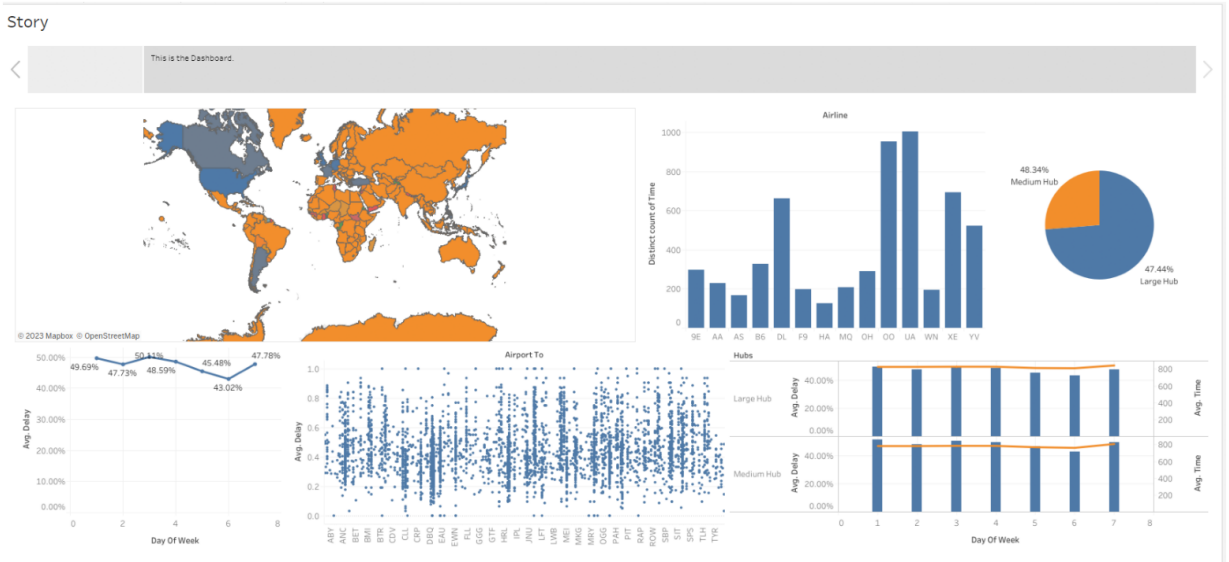

Average of Delay and Hubs
48.34%
Medium Hub
47.44%
Large Hub
Data Source
Airlines based on time perform...
Percentage of delayed flights f...
Average of Delay and Hubs
Count of delayed and on-time ...
Delay for Airport To and Airport...
Map to show types
Dashboard
Story


Count of delayed and on-time flights
Hubs
Large Hub
Medium Hub
Close
Day Of Week


Delay for Airport To and Airport From.
Airport To
Avg Delay


Map to show types
Type
balloonport
closed
heliport
large_airport
medium_airport
seaplane_base
small_airport
© 2023 Mapbox © OpenStreetMap


Airline
48.34%
Medium Hub
47.44%
Large Hub
Airport To
Hubs
Day Of Week


Story
This is the Dashboard.
Airline
48.34%
Medium Hub
47.44%
Large Hub
Airport To
Hubs
Day Of Week

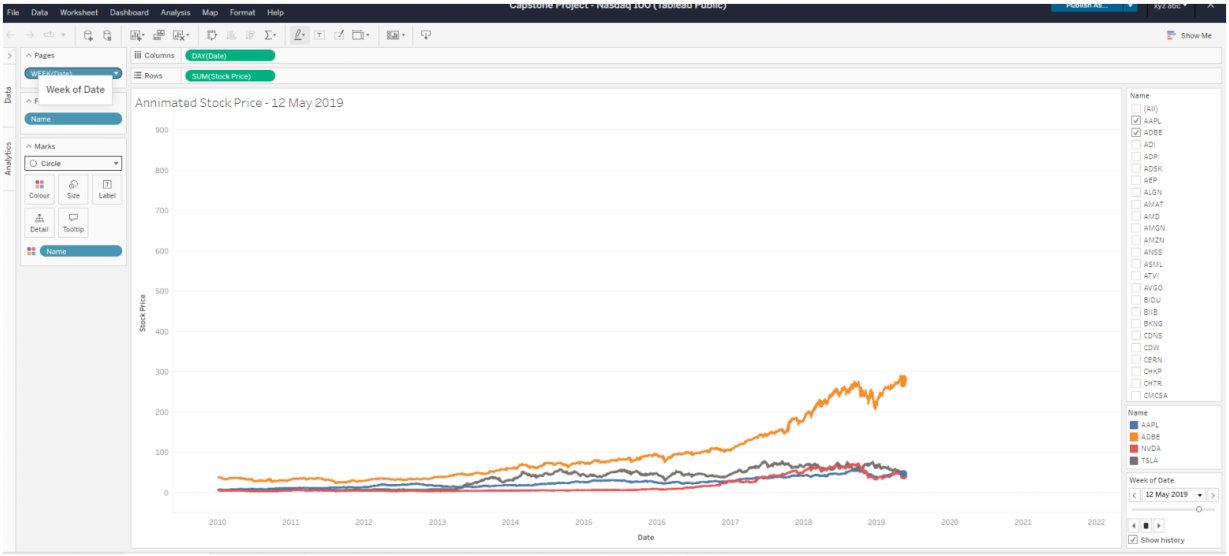


Tableau Insights and Findings

Airlines Data:

Airline Distribution: There are a total of 17 unique airlines in the dataset. The airline with the most flights is "WN" (Southwest Airlines), which appears 94,097 times.

Flight Length: The mean flight length is approximately 2,499.38 miles, with a minimum of 1 mile and a maximum of 7,814 miles.

Day of the Week: Flights are distributed over the days of the week, with an average of 3.93 flights per day. The highest number of flights occurs on Day 7.

Time of Day: The mean flight time is approximately 801.51 minutes, ranging from a minimum of 10 minutes to a maximum of 1,439 minutes.

Flight Length: The mean flight length is approximately 132.22 miles, with values ranging from 0 to 655 miles.

Flight Delay: The mean flight delay is approximately 0.45, indicating that, on average, flights are not delayed. This is further supported by the fact that 75% of flights have a delay value of 1 or less.

Airports Data:

Airport Types: There are 7 unique types of airports. The most common type is "small_airport," which appears 37,676 times.

Elevation Distribution: Airport elevations range from -1,266 feet (possibly below sea level) to 17,372 feet, indicating that airports are situated at varying altitudes.

Continent Distribution: Airports are located on 6 different continents. The most common continent is Asia (AS), with 10,138 airports.

Country Distribution: Airports are located in 243 unique countries, with the United States (US) having the highest number of airports (28,510).

Municipality Distribution: Airports are located in 32,444 unique municipalities, with "Osaka" being the most common, appearing 402 times.

Scheduled Service: Most airports in the dataset (67,034 out of 71,076) do not have scheduled service.

GPS Code Distribution: There are 40,446 unique GPS codes, with some appearing multiple times. The most common GPS code is "SDPS," which appears 3 times.

IATA Code Distribution: There are 8,820 unique IATA codes, with each code representing a different airport.

Local Code Distribution: There are 30,277 unique local codes, with some appearing multiple times.

Home Link Distribution: There are 3,360 unique home links, with some appearing multiple times. The most common home link appears 4 times.

Wikipedia Link Distribution: There are 10,264 unique Wikipedia links, with some appearing multiple times. The most common Wikipedia link appears 14 times.

Keywords Distribution: There are 13,140 unique keywords, with some appearing multiple times. The most common keyword is "Mukho," which appears 47 times.

Airport Details Data:

Airport Reference Distribution: Airport references range from 2 to 430,661.

Airport Identifier Distribution: There are 36,025 unique airport identifiers, with "KORD" being the most common, appearing 11 times.

Runway Length Distribution: Runway lengths range from 0 to 30,000 feet.

Runway Width Distribution: Runway widths range from 0 to 9,000 feet.

Surface Type Distribution: There are 592 unique surface types, with "ASP" being the most common, appearing 10,702 times.

Lighted Runway Distribution: Most runways (42,390 out of 42,390) are lighted.

Closed Runway Distribution: Only a small percentage of runways (16.70%) are marked as closed.

Left Runway Identifier Distribution: There are 266 unique left runway identifiers, with "H1" being the most common, appearing 5,958 times.

Left Runway Latitude Distribution: Left runway latitudes range from -75.60 to 82.51 degrees.

Left Runway Longitude Distribution: Left runway longitudes range from -178.30 to 179.34 degrees.

Left Runway Elevation Distribution: Left runway elevations range from -1,246 to 13,202 feet.

Left Runway Heading Distribution: Left runway headings range from 0 to 360 degrees.

Detailed Insights and Findings

Airline Distribution

- There are a total of 17 unique airlines in the dataset.
- The airline with the most flights is "WN" (Southwest Airlines), which appears 94,097 times.

Flight Length

- The mean flight length is approximately 2,499.38 miles, with a minimum of 1 mile and a maximum of 7,814 miles.

Day of the Week

- Flights are distributed over the days of the week, with an average of 3.93 flights per day.
- The highest number of flights occurs on Day 7.

Time of Day

- The mean flight time is approximately 801.51 minutes, ranging from a minimum of 10 minutes to a maximum of 1,439 minutes.

Flight Length (Again)

- The mean flight length is approximately 132.22 miles, with values ranging from 0 to 655 miles.

Flight Delay

- The mean flight delay is approximately 0.45, indicating that, on average, flights are not delayed.
- 75% of flights have a delay value of 1 or less.

Airports Data Overview

Airport Types

- There are 7 unique types of airports.
- The most common type is "small_airport," which appears 37,676 times.

Elevation Distribution

- Airport elevations range from -1,266 feet (possibly below sea level) to 17,372 feet, indicating that airports are situated at varying altitudes.

Continent Distribution

- Airports are located on 6 different continents.
- The most common continent is Asia (AS), with 10,138 airports.

Country Distribution

- Airports are located in 243 unique countries.
- The United States (US) has the highest number of airports (28,510).

Municipality Distribution

- Airports are located in 32,444 unique municipalities.
- "Osaka" is the most common, appearing 402 times.

Scheduled Service

- Most airports in the dataset (67,034 out of 71,076) do not have scheduled service.

GPS Code Distribution

- There are 40,446 unique GPS codes, with some appearing multiple times.
- The most common GPS code is "SDPS," which appears 3 times.

IATA Code Distribution

- There are 8,820 unique IATA codes, with each code representing a different airport.

Local Code Distribution

- There are 30,277 unique local codes, with some appearing multiple times.

Home Link Distribution

- There are 3,360 unique home links, with some appearing multiple times.
- The most common home link appears 4 times.

Wikipedia Link Distribution

- There are 10,264 unique Wikipedia links, with some appearing multiple times.
- The most common Wikipedia link appears 14 times.

Keywords Distribution

- There are 13,140 unique keywords, with some appearing multiple times.
- The most common keyword is "Mukho," which appears 47 times.

Airport Details Data Overview

Airport Reference Distribution

- Airport references range from 2 to 430,661.

Airport Identifier Distribution

- There are 36,025 unique airport identifiers.

- "KORD" is the most common, appearing 11 times.

Runway Length Distribution

- Runway lengths range from 0 to 30,000 feet.

Runway Width Distribution

- Runway widths range from 0 to 9,000 feet.

Surface Type Distribution

- There are 592 unique surface types.
- "ASP" is the most common, appearing 10,702 times.

Lighted Runway Distribution

- Most runways (42,390 out of 42,390) are lighted.

Closed Runway Distribution

- Only a small percentage of runways (16.70%) are marked as closed.

Left Runway Identifier Distribution

- There are 266 unique left runway identifiers.
- "H1" is the most common, appearing 5,958 times.

Left Runway Latitude Distribution

- Left runway latitudes range from -75.60 to 82.51 degrees.

Left Runway Longitude Distribution

- Left runway longitudes range from -178.30 to 179.34 degrees.

Left Runway Elevation Distribution

- Left runway elevations range from -1,246 to 13,202 feet.

Left Runway Heading Distribution

- Left runway headings range from 0 to 360 degrees.

Report on Counts in the Dataset

Count of Airlines:

The dataset contains information about different airlines. Here is a breakdown of the counts for each airline:

- WN (Southwest Airlines) has the highest count with 94,097 flights.
- DL (Delta Air Lines) follows closely with 60,940 flights.
- OO (SkyWest Airlines) has 50,254 flights.
- AA (American Airlines) has 45,656 flights.
- MQ (Envoy Air) has 36,605 flights.
- US (US Airways) has 34,500 flights.
- XE (ExpressJet Airlines) has 31,126 flights.
- EV (ExpressJet Airlines) has 27,983 flights.
- UA (United Airlines) has 27,619 flights.
- CO (Continental Airlines) has 21,118 flights.
- 9E (Endeavor Air) has 20,686 flights.
- B6 (JetBlue Airways) has 18,112 flights.
- YV (Mesa Airlines) has 13,725 flights.
- OH (Comair) has 12,630 flights.
- AS (Alaska Airlines) has 11,471 flights.
- F9 (Frontier Airlines) has 6,456 flights.
- HA (Hawaiian Airlines) has 5,578 flights.

This information provides an overview of the distribution of flights among different airlines in the dataset.

Report on Count of AirportFrom and AirportTo:

The dataset includes information about departure and arrival airports. Here are the counts for both departure and arrival airports:

For the 'AirportFrom' column:

- ATL (Hartsfield-Jackson Atlanta International Airport) is the most common departure airport with 28,827 flights.
- ORD (Chicago O'Hare International Airport) follows with 24,822 flights.
- DFW (Dallas/Fort Worth International Airport) has 21,900 flights.
- DEN (Denver International Airport) has 19,720 flights.
- LAX (Los Angeles International Airport) has 16,490 flights.
- There are a total of 291 unique departure airports in the dataset.

For the 'AirportTo' column:

- ATL (Hartsfield-Jackson Atlanta International Airport) is also the most common arrival airport with 28,825 flights.
- ORD (Chicago O'Hare International Airport) follows with 24,871 flights.
- DFW (Dallas/Fort Worth International Airport) has 21,899 flights.
- DEN (Denver International Airport) has 19,725 flights.
- LAX (Los Angeles International Airport) has 16,491 flights.
- There are also 291 unique arrival airports in the dataset.

These counts provide insights into the popularity of various airports for both departures and arrivals.

Report on Count of DayOfWeek:

The dataset records flights on different days of the week. Here is the count of flights for each day of the week:

- Day 4 (Thursday) has the highest count with 87,988 flights.
- Day 3 (Wednesday) follows closely with 86,478 flights.
- Day 5 (Friday) has 81,797 flights.
- Day 1 (Sunday) has 70,008 flights.
- Day 2 (Monday) has 68,721 flights.
- Day 7 (Saturday) has 67,210 flights.
- Day 6 (Sunday) has 56,354 flights.

These counts reveal the distribution of flights across different days of the week.

Correlation Matrix:

The correlation matrix provides insights into the relationships between numerical columns in the dataset. Here are the correlations between various columns:

- The 'Delay' column has a positive correlation with the 'Time' column (0.15), indicating that longer flights tend to have more delays.
- The 'Length' column has a negative correlation with the 'Flight' column (-0.35), suggesting that shorter flights are associated with higher flight numbers.
- There is a weak positive correlation (0.14) between the 'Delay' and 'id' columns.

It's important to note that correlation does not imply causation, but these correlations can provide valuable information for further analysis and modeling.

Descriptive Statistics for Numerical Columns:

- **latitude_deg:**

- Count: 73,805
- Mean: 25.786389
- Standard Deviation: 26.232686
- Minimum: -90.000000
- 25th Percentile: 12.536100
- Median (50th Percentile): 35.160179
- 75th Percentile: 42.720901
- Maximum: 82.750000

- **longitude_deg:**
  - Count: 73,805
  - Mean: -28.880235
  - Standard Deviation: 86.121515
  - Minimum: -179.876999
  - 25th Percentile: -94.170097
  - Median (50th Percentile): -69.893898
  - 75th Percentile: 23.934668
  - Maximum: 179.975700
- **elevation_ft:**
  - Count: 59,683
  - Mean: 1299.934370
  - Standard Deviation: 1672.759483
  - Minimum: -1266.000000
  - 25th Percentile: 205.000000
  - Median (50th Percentile): 730.000000
  - 75th Percentile: 1608.000000
  - Maximum: 17,372.000000

Statistics Grouped by Continent:

- Descriptive statistics for latitude_deg, longitude_deg, and elevation_ft for each continent are provided. For example, in Africa (AF), the mean latitude is -5.791058, the mean longitude is 23.599012, and the mean elevation is 2524.691277.

Statistics Grouped by ISO Country:

- Descriptive statistics for latitude_deg, longitude_deg, and elevation_ft are provided for each ISO country code. For example, for AD (Andorra), the mean latitude is 42.534156, the mean longitude is 1.501690, and the mean elevation is 3450.000000.

Statistics Grouped by Type:

- Descriptive statistics for latitude_deg, longitude_deg, and elevation_ft are provided for each airport type. For example, for "heliport," the mean latitude is 28.711213, the mean longitude is -8.883333, and the mean elevation is 1247.456442.

These statistics give you insights into the distribution and variation of geographical and elevation data in your dataset, both overall and within specific groups based on continent, ISO country code, and airport type.

Latitude and Longitude Distribution:

- The `latitude_deg` column ranges from -90.000000 (representing the South Pole) to 82.750000 (near the North Pole).
- The `longitude_deg` column spans from -179.876999 (near the International Date Line) to 179.975700 (opposite side of the globe).
- This wide distribution of latitude and longitude values suggests that the dataset contains airport locations from around the world.

Elevation Variation:

- The `elevation_ft` column shows a wide variation in elevation levels, ranging from -1266.000000 (possibly below sea level) to 17,372.000000 feet.
- This indicates that airports are situated at varying altitudes, from below sea level to high-altitude locations.

Continent-wise Analysis:

- The data is grouped by continent, revealing significant differences in latitude, longitude, and elevation among continents.
- For example, airports in Africa (AF) tend to have lower latitudes and longitudes, while those in Asia (AS) have a wider range of longitudes.
- Elevation levels also vary widely by continent, with airports in Africa having relatively higher elevations compared to other continents.

Country-wise Analysis:

- Descriptive statistics for latitude, longitude, and elevation are provided for individual ISO countries.
- Some countries, like Andorra (AD), have relatively consistent values for latitude, longitude, and elevation, while others, like the United States (US), exhibit significant variations.

Airport Type Variation:

- Different types of airports, such as "heliport" and "small_airport," have distinct characteristics.
- Heliports tend to have higher mean elevations, possibly due to their locations in mountainous or elevated areas.
- Closed airports, on average, have lower elevations compared to other types, indicating that they may be located in lower-lying regions.

Data Completeness:

- It's important to note that the count of `elevation_ft` is lower than the total count of records. This suggests that some airport entries do not have elevation data.
- Data completeness and quality checks may be required to ensure accurate analysis, especially when considering elevation-related insights.

Geographical Diversity:

- The dataset includes airport locations from various parts of the world, reflecting the global reach of aviation.
- Researchers or analysts working with this dataset can explore geographical patterns and relationships based on these coordinates and elevation data.

Excel Dashbaord

Creating an Excel Dashboard

To create an Excel dashboard showcasing the following elements, you can follow these steps using Excel's built-in features and form controls:

1. **Prepare Your Data**: Ensure that you have all the necessary data in your Excel workbook. You should have separate data tables for each of the requested comparisons: on-time performance by airline, percentage of delayed flights by day of the week, passenger data at large and medium hubs, and flight data with source and destination airports.

2. **Create Form Controls**: Insert form controls (e.g., combo boxes, radio buttons, or list boxes) to allow users to interact with the dashboard. You can find these form controls in the "Developer" tab (if not visible, enable it in Excel settings).

    - For part (d) of your request (visualizing the count of delayed and on-time flights for different pairs of source and destination airports), you'll likely need combo boxes for selecting source and destination airports.
3. **Calculate Metrics and Create Dynamic Charts**: For each of the requested comparisons, calculate the necessary metrics:

- a. For comparing different airlines based on on-time performance, calculate on-time and delayed flight counts for each airline.
- b. For comparing the percentage of delayed flights for different days of the week, calculate the percentage of delayed flights for each day.
- c. For creating a trend chart for the number of passengers at large and medium hubs, aggregate passenger data over time for large and medium hubs.
- d. For visualizing the count of delayed and on-time flights for different pairs of source and destination airports, use pivot tables or formulas to calculate these counts based on user selections.

Create dynamic charts for each of these comparisons by referencing the calculated metrics. Excel's pivot charts and regular charts can be made dynamic by linking them to your form controls.

4. **Organize the Dashboard**: Arrange the form controls, charts, and any additional elements (titles, labels, legends) on an Excel worksheet to create a visually appealing and user-friendly dashboard. You can use grouping and shapes to organize the elements neatly.

5. **Link Form Controls to Data and Charts**: Link the form controls to the relevant data and charts. This can be done through Excel's "Form Control" properties, where you specify the cell or range that should be affected by the form control.

6. **Test and Validate**: Test your dashboard to ensure that the dynamic features work as expected. Verify that selecting different options in the form controls updates the charts and data accordingly.

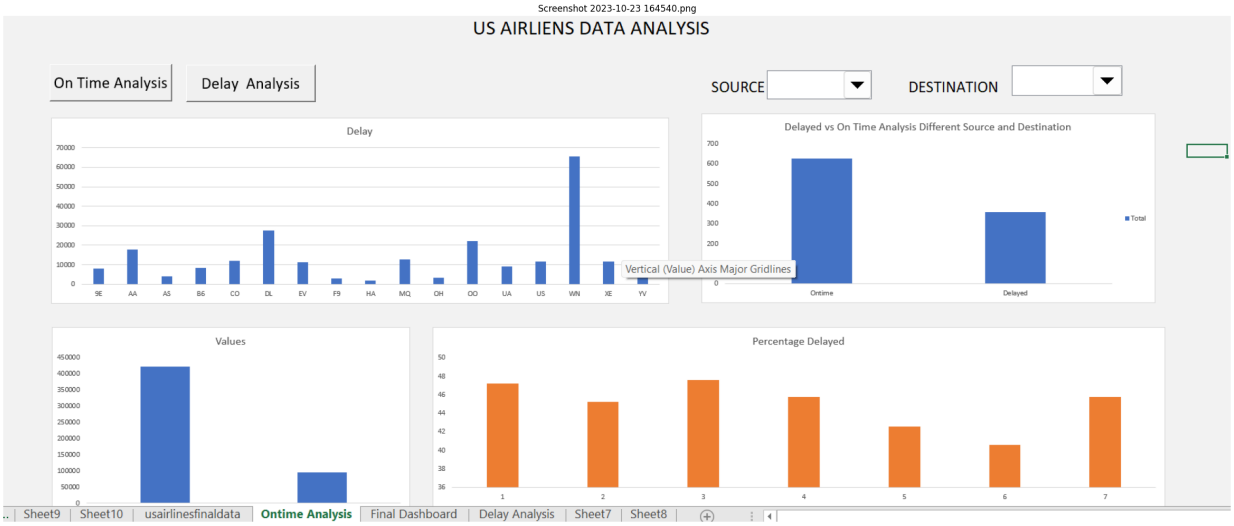
7. **Document and Share**: Provide instructions or labels to guide users on how to interact with the dashboard effectively. Once your Excel dashboard is ready, save and share it wit and destination airports.

In [182…

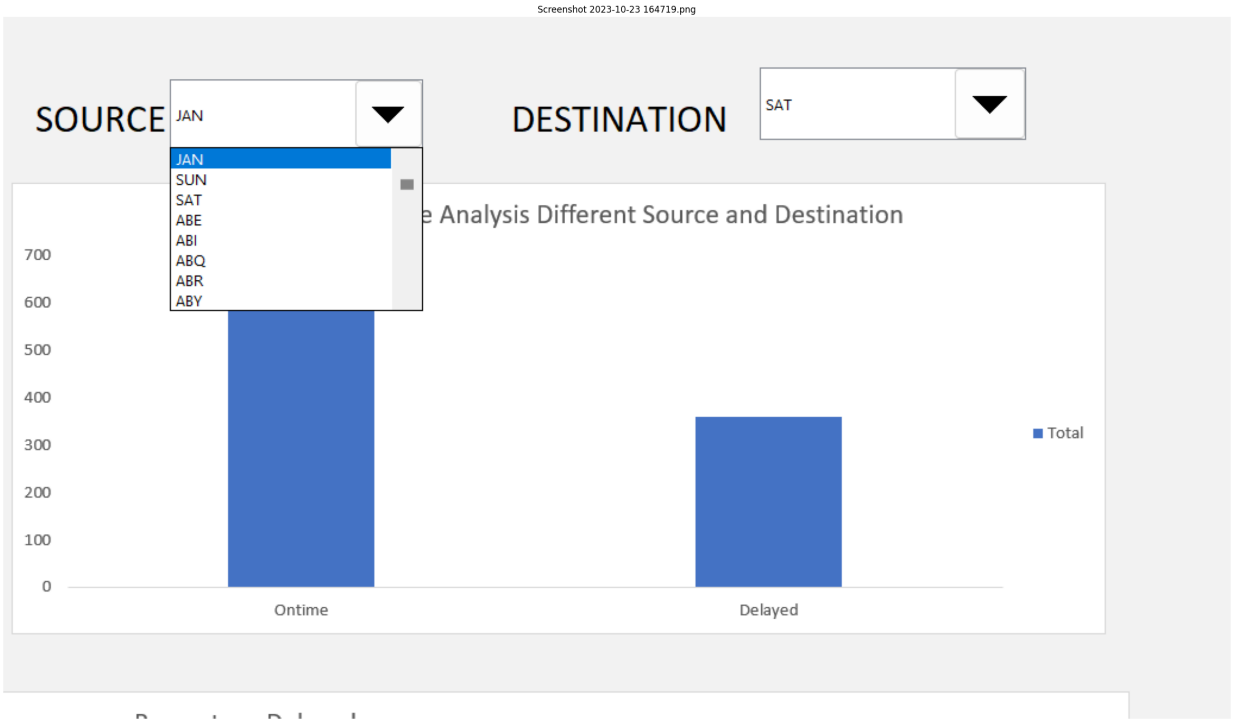
```
import os
from PIL import Image
import matplotlib.pyplot as plt

# Specify the folder path where your images are located
folder_path = r'C:\Users\bhara\OneDrive\Pictures\Screenshots\airlines excel'

# Get a list of all files in the folder
image_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folde

# Filter files to include only image files (e.g., JPG, PNG, etc.)
image_files = [f for f in image_files if f.lower().endswith(('.jpg', '.jpeg', '.png',

# Display each image
for image_file in image_files:
    plt.figure(figsize=(30,30))
    image_path = os.path.join(folder_path, image_file)
    img = Image.open(image_path)
    print(" ")
    print(" ")
    print(" ")
    print(" ")
    plt.imshow(img)
```

```
    plt.title(image_file)
    plt.axis('off')  # Turn off axis labels
    plt.show()
```

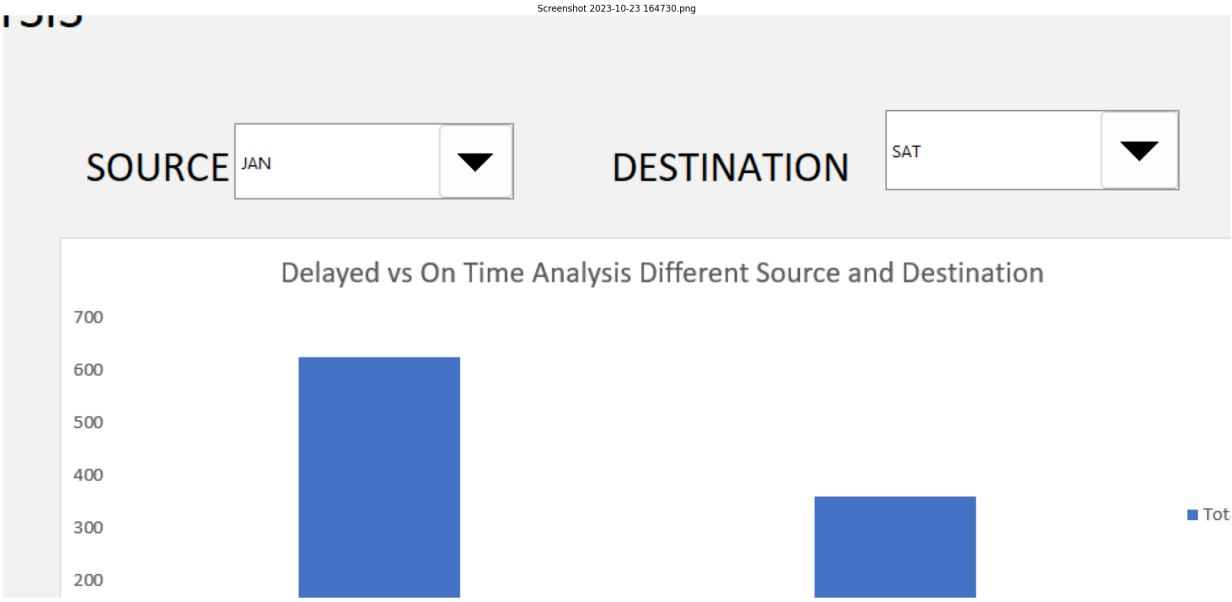
Screenshot 2023-10-23 164540.png



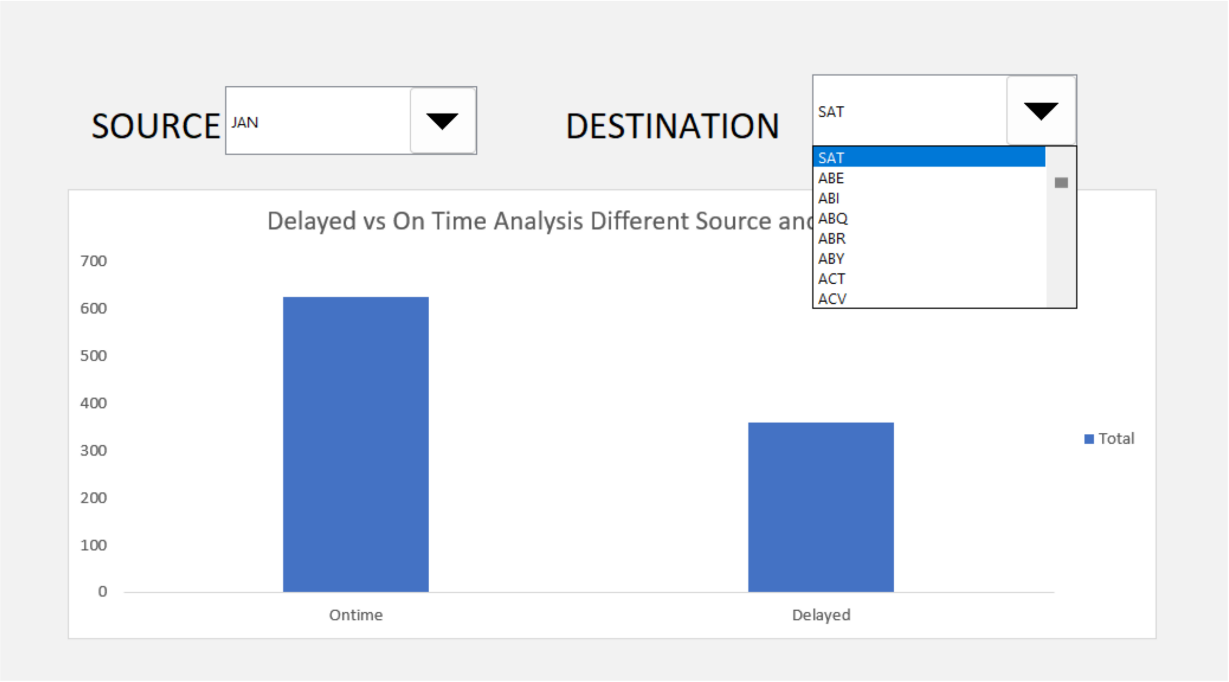
Screenshot 2023-10-23 164719.png



Screenshot 2023-10-23 164730.png

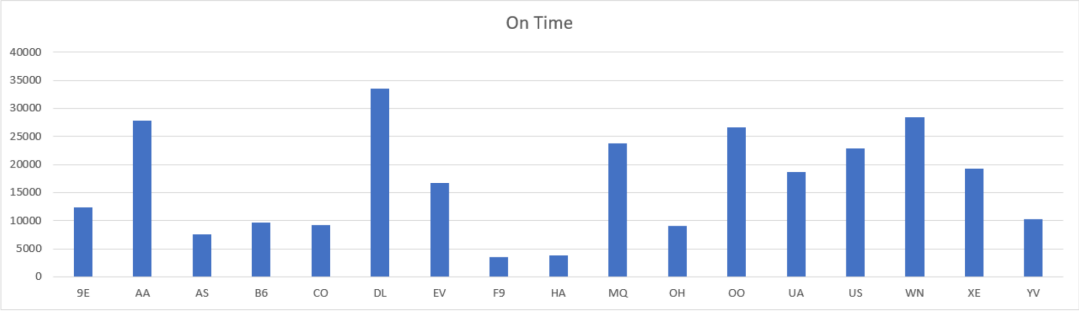


Screenshot 2023-10-23 164738.png

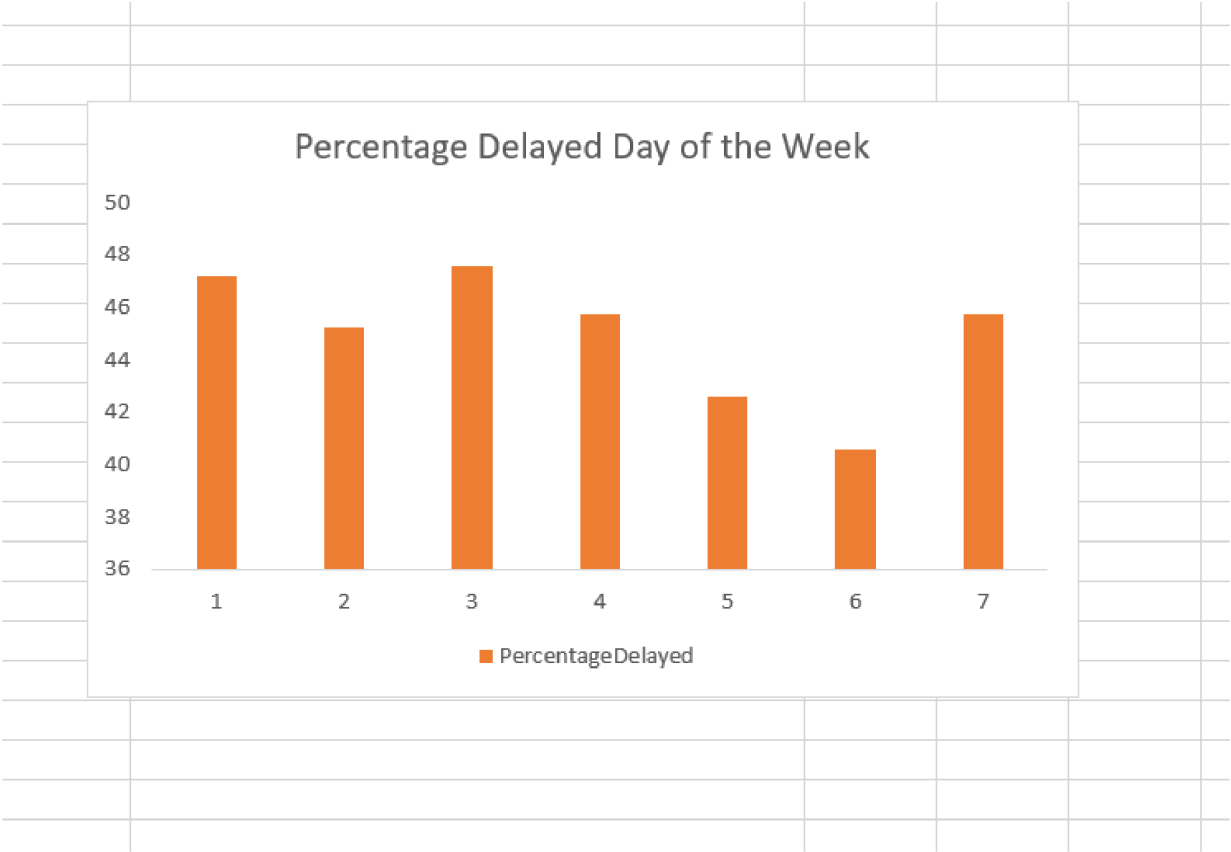


Screenshot 2023-10-23 164751.png

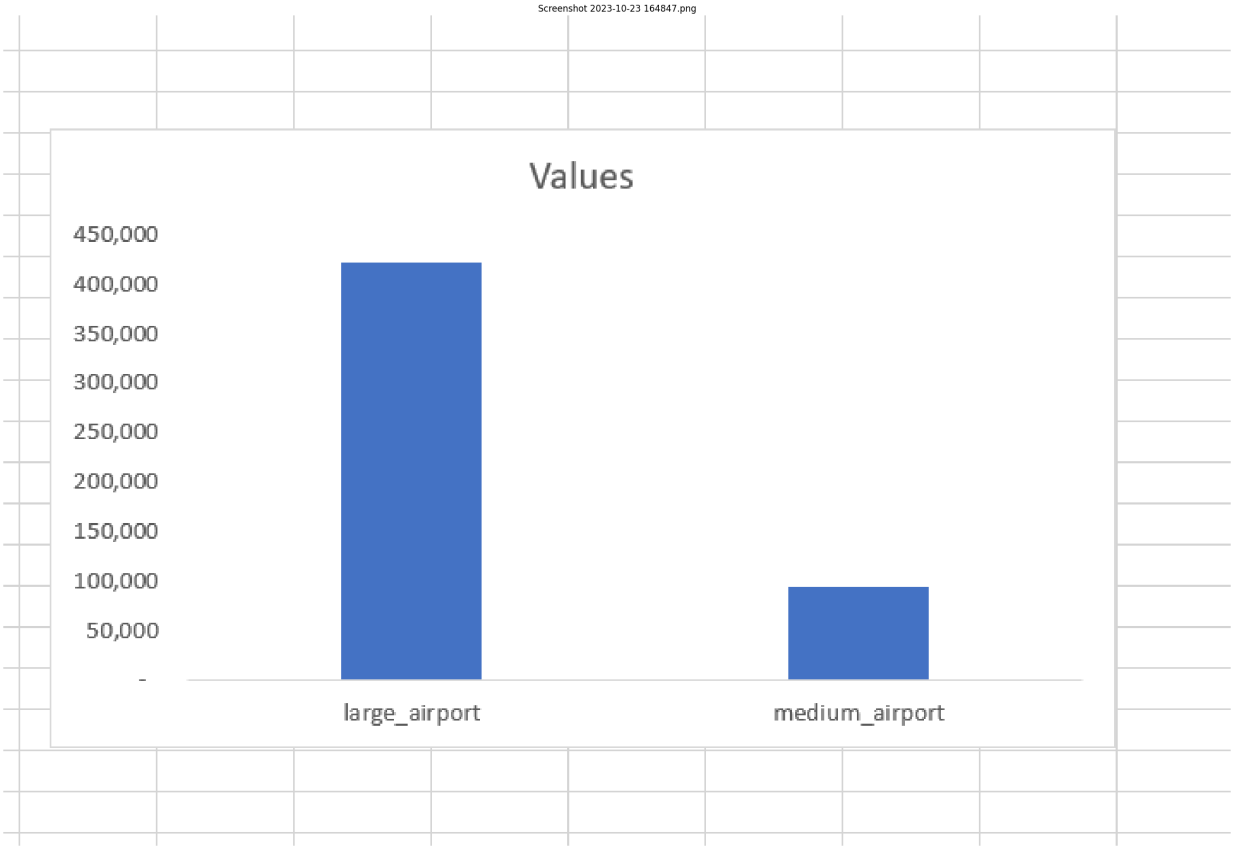
On Time Analysis | Delay Analysis



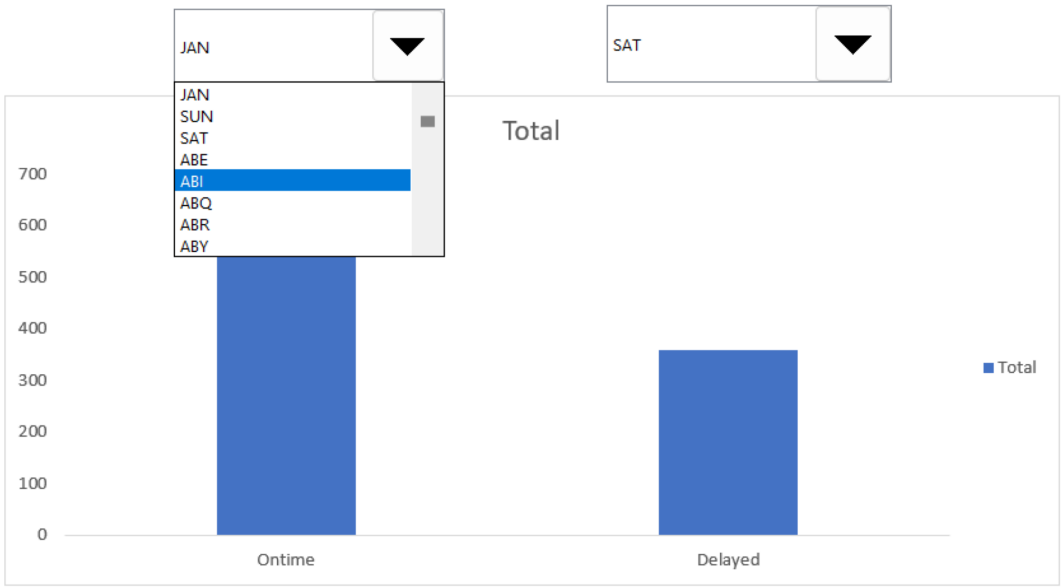
Screenshot 2023-10-23 164838.png

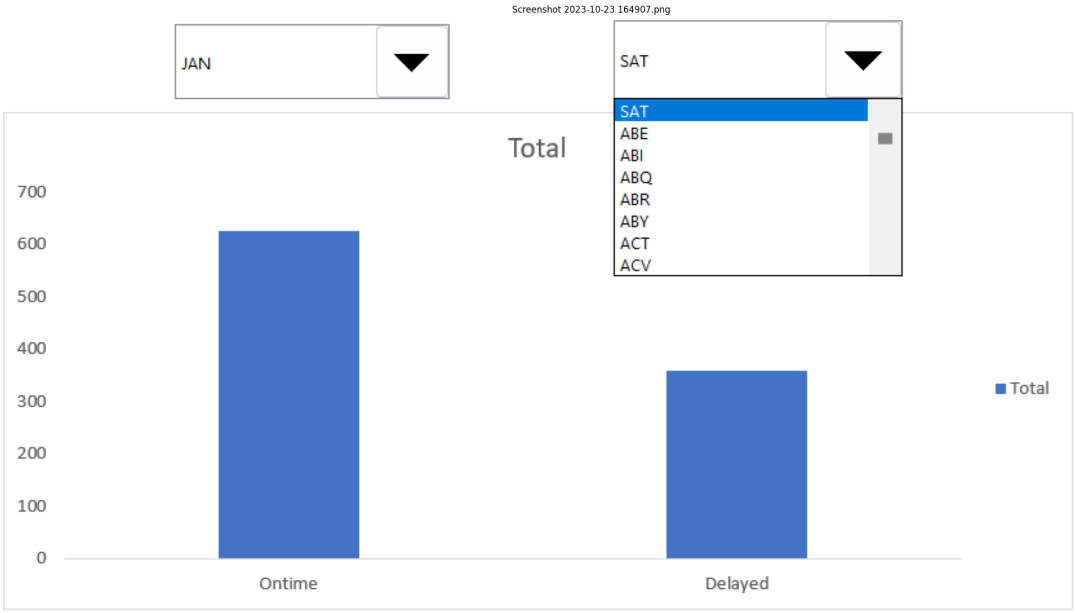


Screenshot 2023-10-23 164847.png



Screenshot 2023-10-23 164900.png





SQL Codes

Question 1: Determine the number of flights that are delayed on various days of the week

SELECT DayOfWeek, COUNT(*) AS DelayedFlights FROM airlines WHERE Delay = 1 GROUP BY DayOfWeek ORDER BY DayOfWeek;

Question 2: Determine the number of delayed flights for various airlines

SELECT Airline, COUNT(*) AS DelayedFlights FROM airlines WHERE Delay = 1 GROUP BY Airline ORDER BY DelayedFlights DESC;

Question 3: Determine how many delayed flights land at airports with at least 10 runways

SELECT COUNT(*) AS DelayedFlightsWith10Runways FROM airlines a INNER JOIN runways r1 ON a.AirportTo = r1.airport_ident INNER JOIN runways r2 ON a.AirportFrom = r2.airport_ident WHERE a.Delay = 1 AND (r1.length_ft >= 10 OR r2.length_ft >= 10);

Question 4: Compare the number of delayed flights at airports higher than average elevation and

those that are lower than### leaverage elevation for both source and destination airports

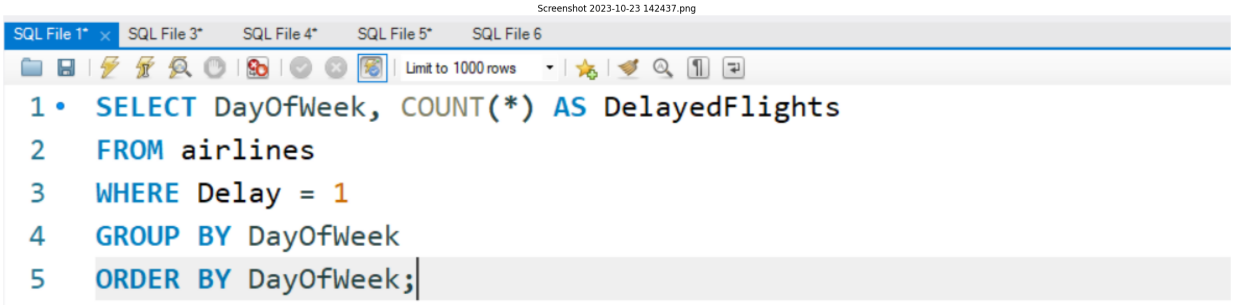
WITH AvgElevation AS (SELECT AVG(elevation_ft) AS AvgElev FROM airports) SELECT 'Source Airport - Higher Elevation' AS Category, COUNT(*) AS DelayedFlights FROM airlines a INNER JOIN airports src ON a.AirportFrom = src.ident WHERE a.Delay = 1 AND src.elevation_ft > (SELECT AvgElev FROM AvgElevation) UNION ALL SELECT 'Source Airport - Lower Elevation' AS Category, COUNT(*) AS DelayedFlights FROM airlines a INNER JOIN airports src ON a.AirportFrom = src.ident WHERE a.Delay = 1 AND src.elevation_ft <= (SELECT AvgElev FROM AvgElevation) UNION ALL SELECT 'Destination Airport - Higher Elevation' AS Category, COUNT(*) AS DelayedFlights FROM airlines a INNER JOIN airports dest ON a.AirportTo = dest.ident WHERE a.Delay = 1 AND

dest.elevation_ft > (SELECT AvgElev FROM AvgElevation) UNION ALL SELECT 'Destination Airport - Lower Elevation' AS Category, COUNT(*) AS DelayedFlights FROM airlines a INNER JOIN airports dest ON a.AirportTo = dest.ident WHERE a.Delay = 1 AND dest.elevation_ft <= (SELECT AvgElev FROM AvgElevation);

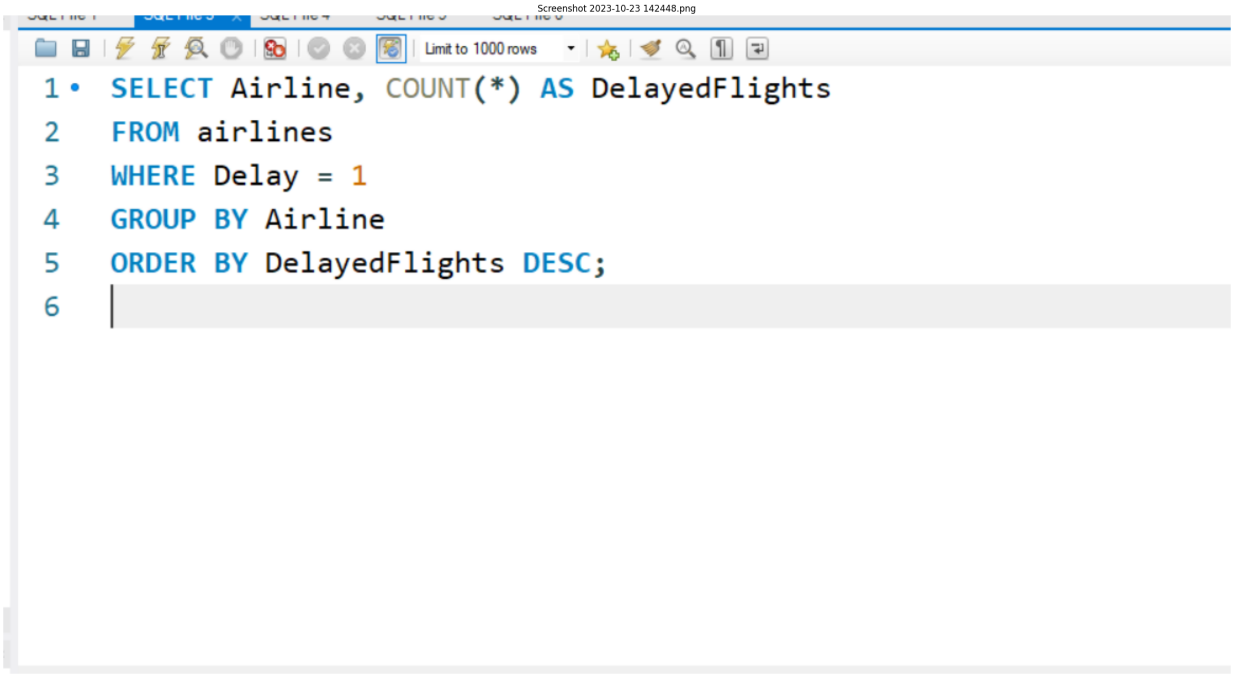
In [184…

```
import os
from PIL import Image
import matplotlib.pyplot as plt

# Specify the folder path where your images are located
folder_path = r'C:\Users\bhara\OneDrive\Pictures\Screenshots\airlines sql'

# Get a list of all files in the folder
image_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folde

# Filter files to include only image files (e.g., JPG, PNG, etc.)
image_files = [f for f in image_files if f.lower().endswith(('.jpg', '.jpeg', '.png',

# Display each image
for image_file in image_files:
    plt.figure(figsize=(30,30))
    image_path = os.path.join(folder_path, image_file)
    img = Image.open(image_path)
    print(" ")
    print(" ")
    print(" ")
    print(" ")
    plt.imshow(img)
    plt.title(image_file)
    plt.axis('off')  # Turn off axis labels
    plt.show()
```

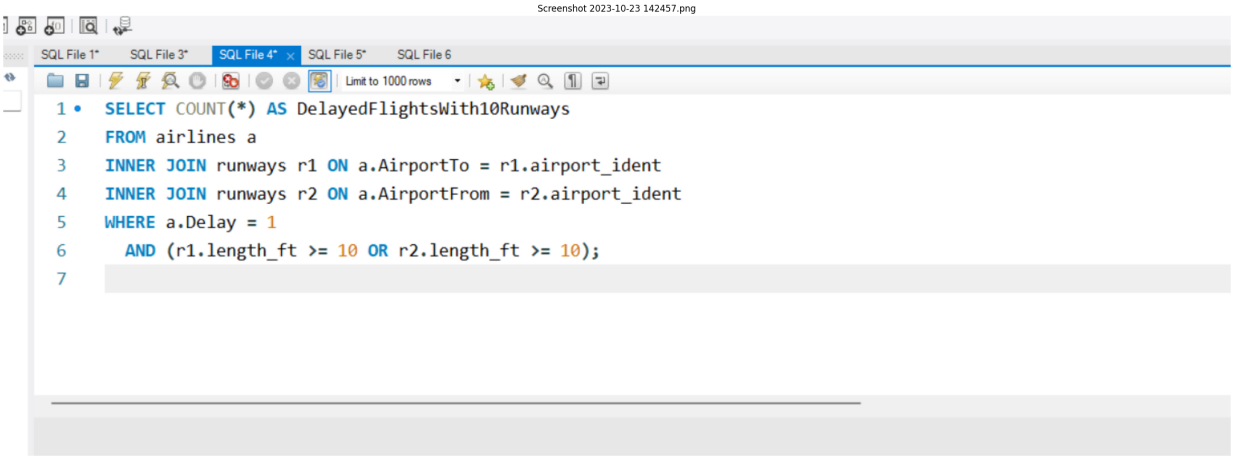
Screenshot 2023-10-23 142437.png



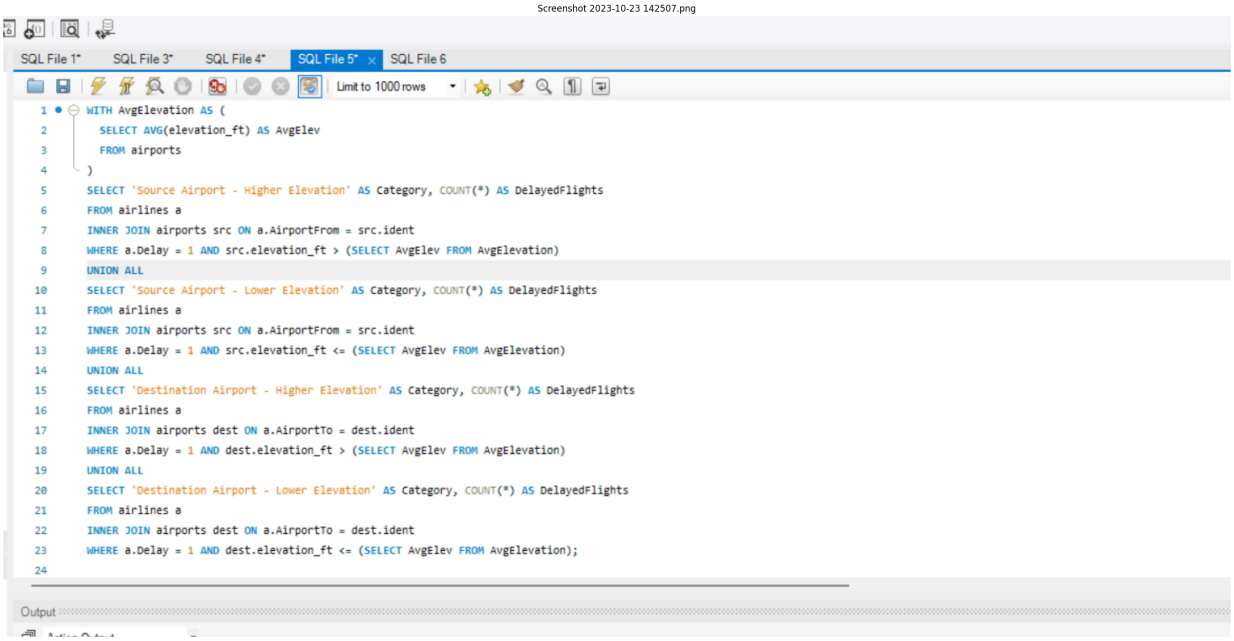
Screenshot 2023-10-23 142448.png



Screenshot 2023-10-23 142457.png



Screenshot 2023-10-23 142507.png



Task 1: Number of Delayed Flights on Various Days of the Week

- The highest number of delayed flights occurred on **Wednesday (Day 3)** with **41,144 delays**, making it the day with the most delays.
- **Tuesday (Day 2)** had the second-highest number of delays with **31,072**, followed closely by **Thursday (Day 4)** with **40,280 delays**.
- Other days of the week had varying numbers of delayed flights:
  - **Sunday (Day 7)**: 30,761 delayed flights
  - **Monday (Day 1)**: 33,059 delayed flights
  - **Friday (Day 5)**: 34,813 delayed flights
  - **Saturday (Day 6)**: The lowest number of delays at **22,860**.

Task 2: Number of Delayed Flights for Various Airlines

- Among the airlines, **Southwest Airlines (WN)** experienced the highest number of delays with a total of **65,657 delayed flights**, making it the airline with the most delays.
- **Delta Air Lines (DL)** ranked second in terms of delays, with **27,452 delayed flights**.
- Other notable airlines with significant delays include:
  - **SkyWest Airlines (OO)**: 22,760 delays
  - **American Airlines (AA)**: 17,736 delays
  - **ExpressJet Airlines (XE)**: 11,795 delays.
- Airlines with relatively fewer delays included:
  - **Hawaiian Airlines (HA)**: 1,786 delays
  - **Mesa Airlines (YV)**: 3,334 delays
  - **Frontier Airlines (F9)**: 2,899 delays.