assignment

August 31, 2024

Assignment by Karthik N P

[1]:

```
!pip install pymongo
import pandas as pd
import pymongo
flight_df = pd.read_csv('Flights_Delay.csv')
```

```
Requirement already satisfied: pymongo in c:\programdata\anaconda3\lib\site-
packages (4.8.0)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in
c:\programdata\anaconda3\lib\site-packages (from pymongo) (2.6.1)
```

0.0.1 Data Cleaning

[2]:

```
mean_num = flight_df['ARRIVAL_DELAY'].mean()
flight_df['ARRIVAL_DELAY'] = flight_df['ARRIVAL_DELAY'].fillna(mean_num)

mean_num = flight_df['DEPARTURE_DELAY'].mean()
flight_df['DEPARTURE_DELAY'] = flight_df['DEPARTURE_DELAY'].fillna(mean_num)

mean_num = flight_df['AIR_TIME'].mean()
flight_df['AIR_TIME'] = flight_df['AIR_TIME'].fillna(mean_num)

flight_df['DEPARTURE_TIME'] = flight_df['DEPARTURE_TIME'].fillna(0)
flight_df['ARRIVAL_TIME'] = flight_df['ARRIVAL_TIME'].fillna(0)
flight_df['ARRIVAL_TIME']

flight_df['DEPARTURE_TIME'] = flight_df['DEPARTURE_TIME'].astype('int')
flight_df['DEPARTURE_TIME'] = flight_df['DEPARTURE_TIME'].apply(lambda x : x //100 )

flight_df['ARRIVAL_TIME'] = flight_df['ARRIVAL_TIME'].astype('int')
flight_df['ARRIVAL_TIME'] = flight_df['ARRIVAL_TIME'].apply(lambda x : x //100 )
```

a) Create collections “flights” inside database “airline_delayDB” b) How would you insert this entire dataset into a MongoDB collection named flights? Describe the structure of each document.

[3]:

```
client = pymongo.MongoClient('localhost:27017')
db = client['flight_db']

flight_docs = flight_df.to_dict('records')

collection_name = 'flight'

if collection_name in db.list_collection_names():
    db[collection_name].drop()

flights = db[collection_name]

if flight_docs:
    flights.insert_many(flight_docs)
```

.

0.0.2 Description of Document Structure

- **flightNumber**: The unique flight number assigned to the flight .
- **airline**: The airline operating the flight.
- **originAirport**: The IATA code of the airport where the flight originated .
- **destinationAirport**: The IATA code of the destination airport .
- **scheduledDeparture**: The scheduled departure date and time .
- **actualDeparture**: The actual departure date and time.
- **departureDelay**: The difference in minutes between the scheduled and actual departure times.
- **arrivalDelay**: The difference in minutes between the scheduled and actual arrival times.
- **diverted**: A boolean flag indicating whether the flight was diverted.
- **canceled**: A boolean flag indicating whether the flight was canceled.
- **aircraftType**: The type of aircraft used for the flight
- **distance**: The distance of the flight in miles.
- **dayOfWeek**: The day of the week the flight occurred (1 = Monday, 7 = Sunday).
- **month**: The month the flight occurred .
- **year**: The year the flight occurred.

c) Write a MongoDB command to insrt a single flight record from the dataset.

[4]:

```
single_flight = {
    "ID": 1,
    "YEAR": 2015,
    "MONTH": 1,
    "DAY": 1,
    "DAY_OF_WEEK": 4,
    "AIRLINE": "AA",
    "FLIGHT_NUMBER": 2548,
    "TAIL_NUMBER": "N3KUAA",
    "ORIGIN_AIRPORT": "LAX",
    "DESTINATION_AIRPORT": "SFO",
    "SCHEDULED_DEPARTURE": 830,
    "DEPARTURE_TIME": 850,
    "DEPARTURE_DELAY": 20,
    "TAXI_OUT": 15,
    "WHEELS_OFF": 865,
    "SCHEDULED_TIME": 150,
    "ELAPSED_TIME": 140,
    "AIR_TIME": 120,
    "DISTANCE": 337,
    "WHEELS_ON": 1010,
    "TAXI_IN": 10,
    "SCHEDULED_ARRIVAL": 1000,
    "ARRIVAL_TIME": 1020,
    "ARRIVAL_DELAY": 20,
    "DIVERTED": 0,
    "CANCELLED": 0,
}

flights.insert_one(single_flight)
```

```
[4]: InsertOneResult(ObjectId('66d345e885a6b507e69972a4'), acknowledged=True)
```

0.0.3 D) Write a MongoDB query to find all flights that were delayed by more than 60 minutes.

```
[5]: delayed_flights = flights.find({'$or': [{'ARRIVAL_DELAY': {'$gt' :␣
      ↪60}},{'DEPARTURE_DELAY' : {'$gt' : 60}}]},
                                   {'_id':0,'FLIGHT_NUMBER':1,'AIRLINE':1}).
      ↪limit(10)
     for flight in delayed_flights:
         print(flight)
```

```
{'AIRLINE': 'B6', 'FLIGHT_NUMBER': 716}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 6196}
{'AIRLINE': 'US', 'FLIGHT_NUMBER': 1756}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 2699}
{'AIRLINE': 'F9', 'FLIGHT_NUMBER': 661}
{'AIRLINE': 'US', 'FLIGHT_NUMBER': 686}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 4544}
{'AIRLINE': 'WN', 'FLIGHT_NUMBER': 1165}
{'AIRLINE': 'EV', 'FLIGHT_NUMBER': 3936}
{'AIRLINE': 'DL', 'FLIGHT_NUMBER': 1088}
```

0.0.4 E) How would you query all flights that were cancelled and return only the AIRLINE, ORIGIN_AIRPORT, and CANCELLATION_REASON fields?

[6]:
```
cancelled_flights = flights.find(
    {"CANCELLED": 1},
    {"AIRLINE": 1, "ORIGIN_AIRPORT": 1, "CANCELLATION_REASON": 1, "_id": 0}
).limit(10)
##Limit given to reduce the output size
for flight in cancelled_flights:
    print(flight)
```

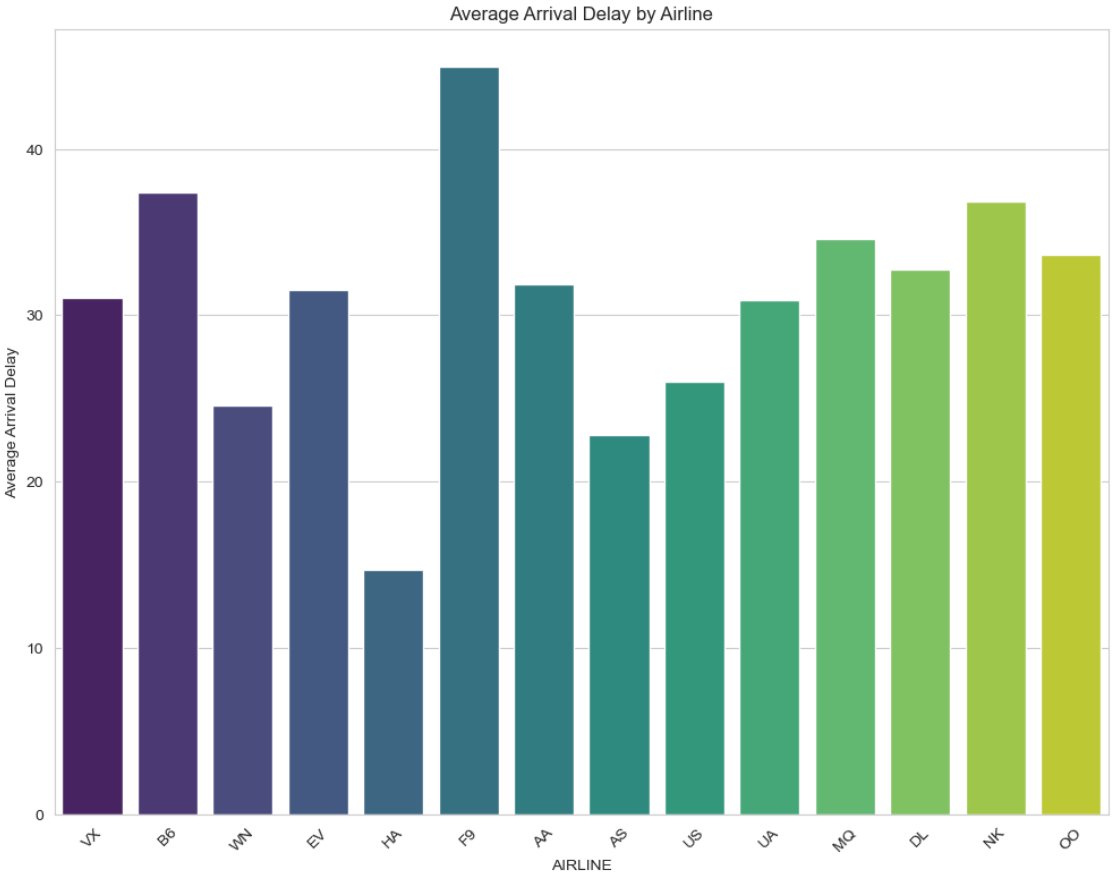
```
{'AIRLINE': 'EV', 'ORIGIN_AIRPORT': 'MLI', 'CANCELLATION_REASON': 'C'}
{'AIRLINE': 'WN', 'ORIGIN_AIRPORT': 'BWI', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'DL', 'ORIGIN_AIRPORT': 'SFO', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'AA', 'ORIGIN_AIRPORT': 'DFW', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'MQ', 'ORIGIN_AIRPORT': 'LGA', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'AA', 'ORIGIN_AIRPORT': 'BDL', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'WN', 'ORIGIN_AIRPORT': 'MKE', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'US', 'ORIGIN_AIRPORT': 'DCA', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'WN', 'ORIGIN_AIRPORT': 'FLL', 'CANCELLATION_REASON': 'B'}
{'AIRLINE': 'EV', 'ORIGIN_AIRPORT': 'ORF', 'CANCELLATION_REASON': 'B'}
```

F)Using MongoDB's aggregation framework, how would you calculate the average arrival delay) for each airline]

[8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
avg_flight_delays = flights.aggregate([
    {"$match": {"ARRIVAL_DELAY": {"$exists": True, "$ne": float('nan'),"$gte":
 ↪0}}},
    {'$group': {'_id':'$AIRLINE', 'avgDelay': {'$avg':'$ARRIVAL_DELAY'}}},
    {'$project': {'AIRLINE':1,'avgDelay' :{'$round': ['$avgDelay',2]}}}
])

delay_df = pd.DataFrame(list(avg_flight_delays))

plt.figure(figsize=(12, 9))
sns.set_style('whitegrid')
sns.barplot(data=delay_df, x='_id', y='avgDelay', palette='viridis')
plt.xlabel('AIRLINE')
plt.ylabel('Average Arrival Delay')
plt.title('Average Arrival Delay by Airline')
plt.xticks(rotation=45)
plt.show()
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_13196\3868619237.py:13:
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=delay_df, x='_id', y='avgDelay', palette='viridis')
```

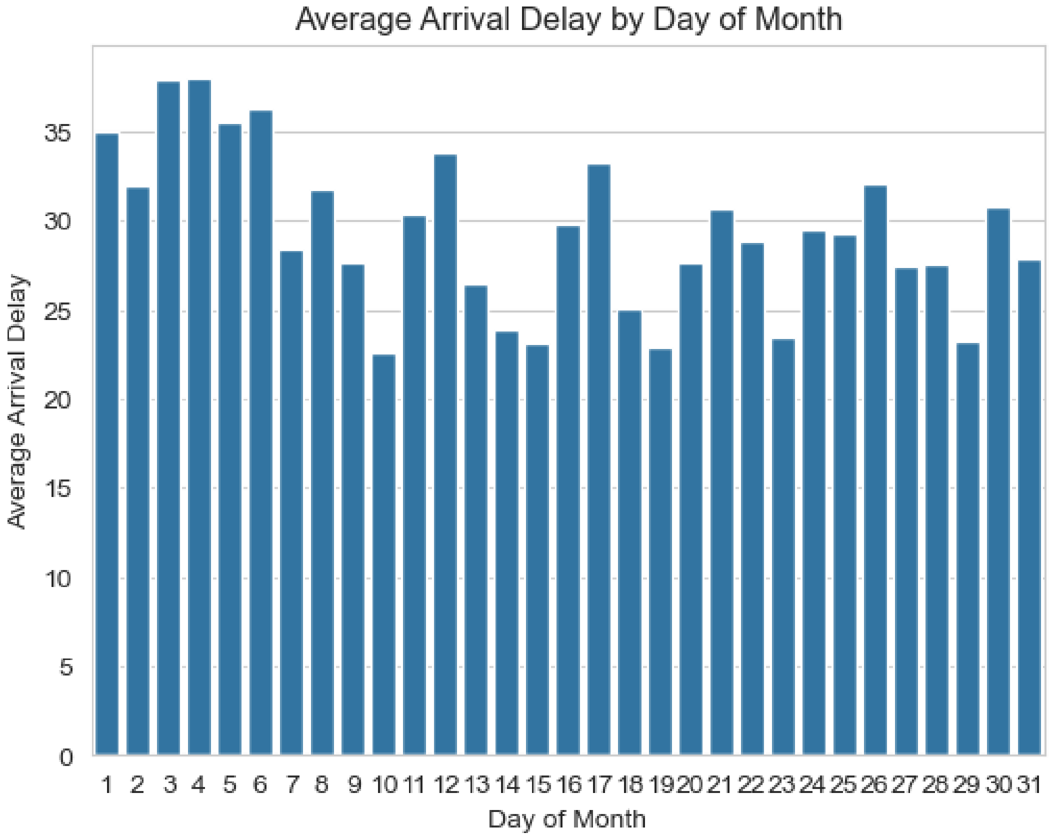


G) Days of months with espect to average of arrival delays. [Create a suitable plot using matplotlib/seaborn]

[9]:

```
pipeline = [
     {"$match": {"ARRIVAL_DELAY": {"$exists": True, "$ne": float('nan'),"$gte":
 ↪0}}},
    {"$group": {"_id": "$DAY", "avg_arrival_delay": {"$avg":␣
 ↪"$ARRIVAL_DELAY"}}},
    {"$sort": {"_id": 1}}
]

result = list(flights.aggregate(pipeline))

# Plotting
```

```
days = [r['_id'] for r in result]
avg_delays = [r['avg_arrival_delay'] for r in result]

sns.barplot(x=days, y=avg_delays)
plt.xlabel('Day of Month')
plt.ylabel('Average Arrival Delay')
plt.title('Average Arrival Delay by Day of Month')
plt.show()
```



H) Write a MongoDB aggregation pipeline to find the top 10 airports with the highest average total delay (DEPARTURE_DELAY + ARRIVAL DELAY).

[10]:
```
# Define the aggregation pipeline
pipeline = [
    {
        "$project": {
            "ORIGIN_AIRPORT": 1,
            "total_delay": {
                "$add": ["$DEPARTURE_DELAY", "$ARRIVAL_DELAY"]  # Calculate␣
 ↪total delay
```

```
            }
        }
    },
    {
        "$group": {
            "_id": "$ORIGIN_AIRPORT",  # Group by origin airport
            "average_total_delay": { "$avg": "$total_delay" }  # Calculate␣
 ↪average total delay
        }
    },
    {
        "$sort": { "average_total_delay": -1 }  # Sort by average total delay␣
 ↪in descending order
    },
    {
        "$limit": 10  # Limit the results to the top 10
    }
]

# Execute the aggregation pipeline
top_airports = list(flights.aggregate(pipeline))

for x in top_airports:
    print(x)
```

```
{'_id': 'HOB', 'average_total_delay': 250.66666666666666}
{'_id': 'CDC', 'average_total_delay': 210.2}
{'_id': 'PIH', 'average_total_delay': 191.2}
{'_id': 'ILG', 'average_total_delay': 168.66666666666666}
{'_id': 'HIB', 'average_total_delay': 168.25}
{'_id': 'SCE', 'average_total_delay': 128.71428571428572}
{'_id': 'TTN', 'average_total_delay': 97.89143550201578}
{'_id': 'JLN', 'average_total_delay': 95.74200070079935}
{'_id': 'SPS', 'average_total_delay': 93.33922129665704}
{'_id': 'MBS', 'average_total_delay': 77.91064986808563}
```

I) Explain how you would create an index on the ORIGIN_AIRPORT and DESTINATION_AIRPORT fields to optimize queries filtering by these fields.

[11]:

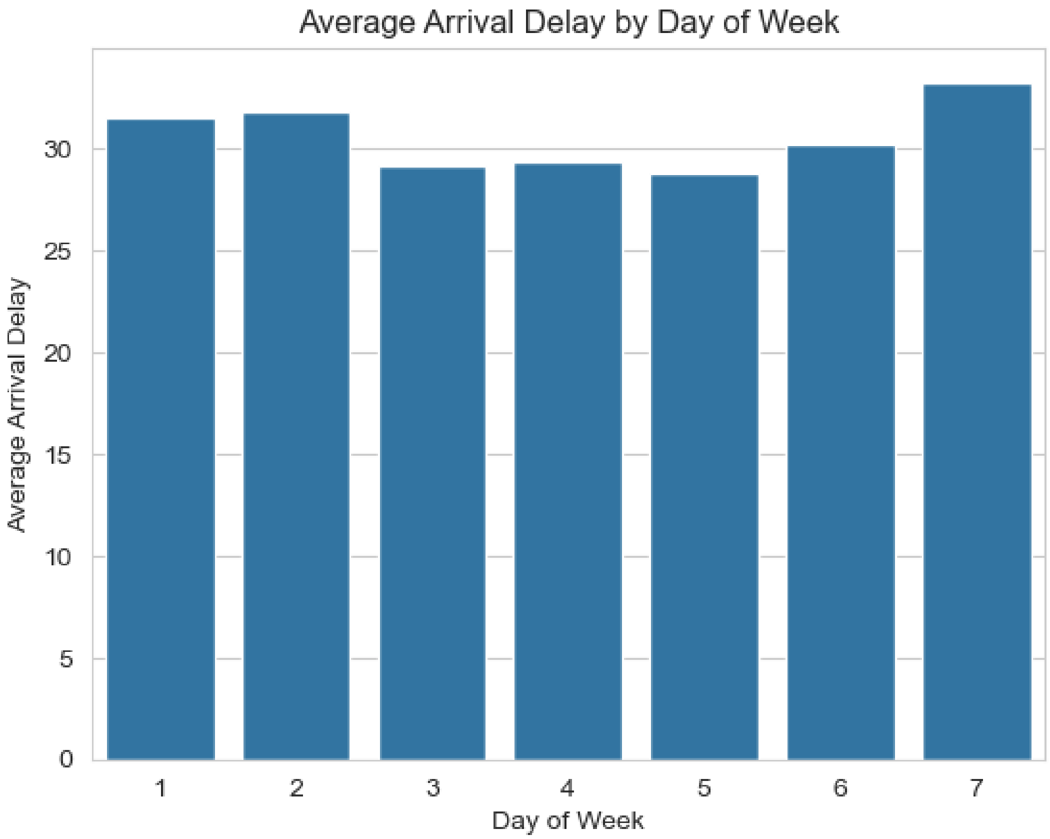
```
flights.create_index(['ORIGIN_AIRPORT','DESTINATION_AIRPORT'])

for i in flights.find({},{'_id':0,'ORIGIN_AIRPORT':1,'DESTINATION_AIRPORT':1}).
 ↪limit(10):
    print(i)
```

```
{'ORIGIN_AIRPORT': 'CVG', 'DESTINATION_AIRPORT': 'XNA'}
{'ORIGIN_AIRPORT': 'DFW', 'DESTINATION_AIRPORT': 'SPS'}
{'ORIGIN_AIRPORT': 'JAX', 'DESTINATION_AIRPORT': 'DCA'}
```

```
{'ORIGIN_AIRPORT': 'COS', 'DESTINATION_AIRPORT': 'IAH'}
{'ORIGIN_AIRPORT': 'ATL', 'DESTINATION_AIRPORT': 'AVL'}
{'ORIGIN_AIRPORT': 'IAH', 'DESTINATION_AIRPORT': 'SFO'}
{'ORIGIN_AIRPORT': 'HDN', 'DESTINATION_AIRPORT': 'DEN'}
{'ORIGIN_AIRPORT': 'ATL', 'DESTINATION_AIRPORT': 'CAK'}
{'ORIGIN_AIRPORT': 'HOU', 'DESTINATION_AIRPORT': 'MEM'}
{'ORIGIN_AIRPORT': 'DAL', 'DESTINATION_AIRPORT': 'MAF'}
```

J)Arrange weekdays with respect to the average arrival delays caused. [Create a suitable plot using matplotlib/seaborn]

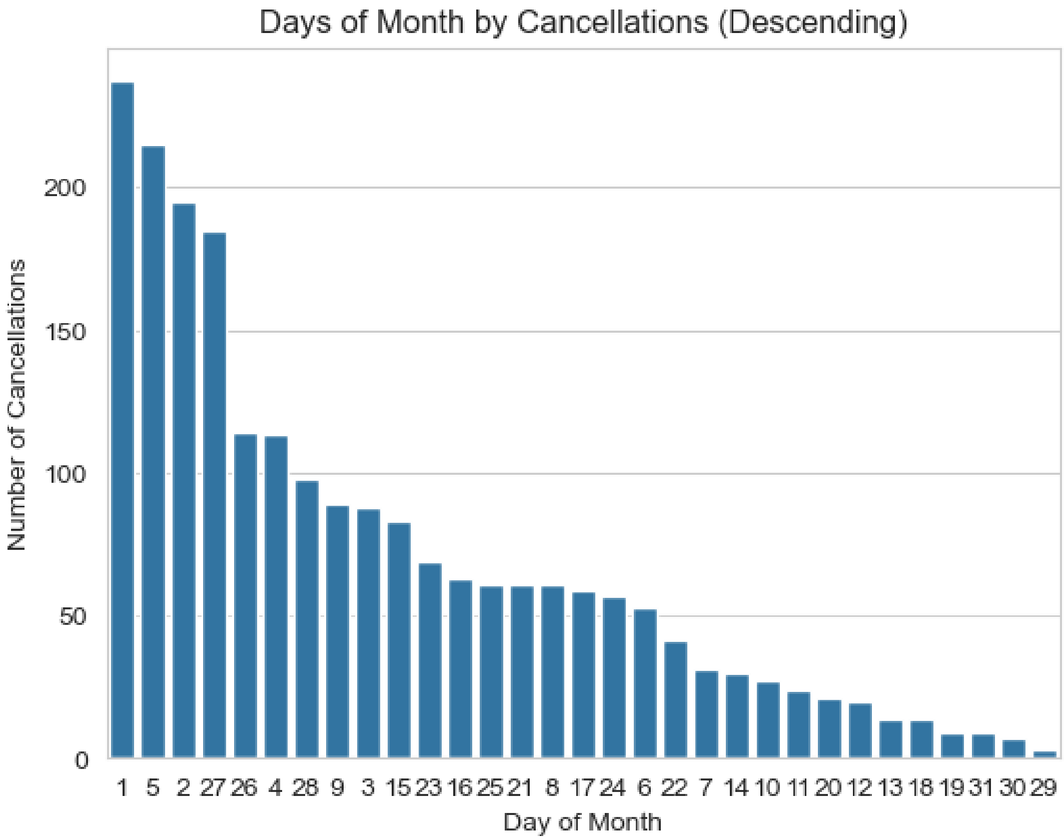
[13]:
```
pipeline = [
     {"$match": {"ARRIVAL_DELAY": {"$exists": True, "$ne": float('nan'),"$gte":
 ↪0}}},
    {"$group": {"_id": "$DAY_OF_WEEK", "avg_arrival_delay": {"$avg":␣
 ↪"$ARRIVAL_DELAY"}}},
    {"$sort": {"_id": 1}}
]

result = list(flights.aggregate(pipeline))

# Plotting
weekdays = [r['_id'] for r in result]
avg_delays = [r['avg_arrival_delay'] for r in result]

sns.barplot(x=weekdays, y=avg_delays)
plt.xlabel('Day of Week')
plt.ylabel('Average Arrival Delay')
plt.title('Average Arrival Delay by Day of Week')
plt.show()
```



K) Arrange Days of month as per cancellations done in descending order. [Create a suitable plot using matplotlib/seaborn]

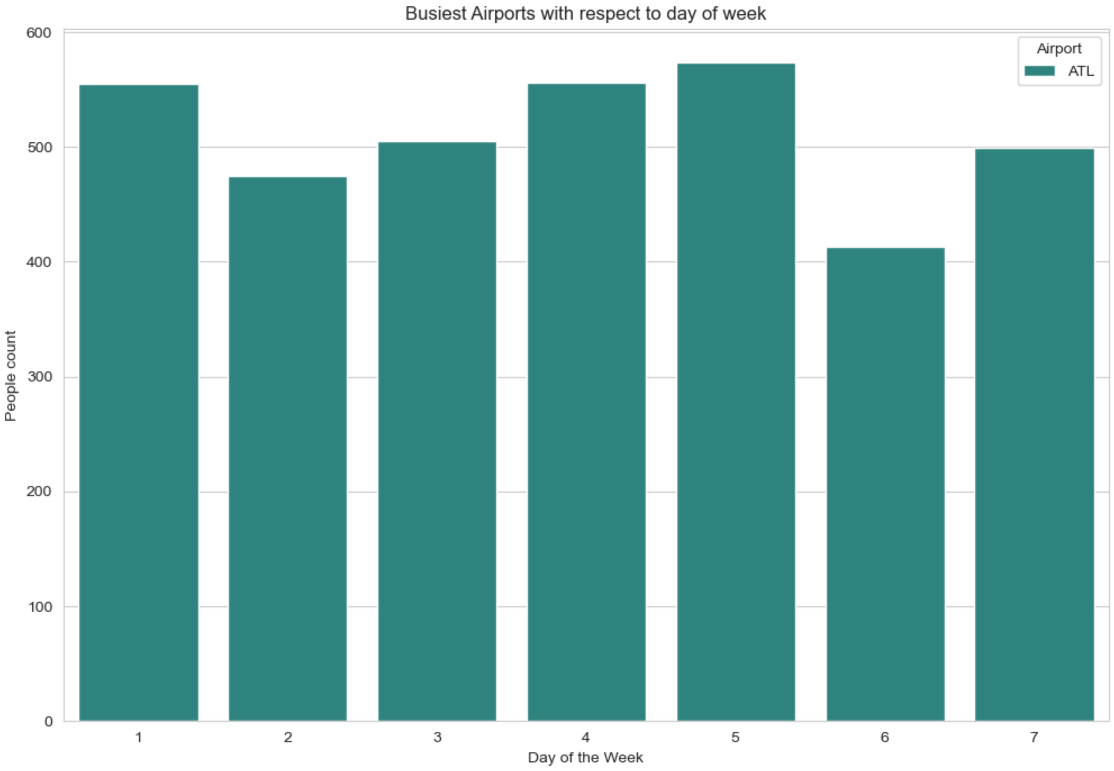
[15]:
```
# Your existing pipeline and result fetching code
pipeline = [
     {"$match": {"CANCELLED": {"$exists": True, "$ne": float('nan'), "$gte":␣
  ↪0}}},
    {"$group": {"_id": "$DAY", "cancellations": {"$sum": "$CANCELLED"}}},
    {"$sort": {"cancellations": -1}}
]

result = list(flights.aggregate(pipeline))

# Extracting data
days = [r['_id'] for r in result]
cancellations = [r['cancellations'] for r in result]

# Creating a DataFrame for plotting
import pandas as pd
data = pd.DataFrame({'Day': days, 'Cancellations': cancellations})
```

```
# Plotting
sns.barplot(x='Day', y='Cancellations', data=data, order=days)
plt.xlabel('Day of Month')
plt.ylabel('Number of Cancellations')
plt.title('Days of Month by Cancellations (Descending)')
plt.show()
```



L) Find the busiest airports with respect to day of week. Represent it by using suitable plot.

[16]:
```
busiest_airports = flights.aggregate([
    {'$group': {'_id': {'AIRPORT': '$ORIGIN_AIRPORT', 'DAY_OF_WEEK':␣
 ↪'$DAY_OF_WEEK'},'count': {'$sum': 1}}},
    {'$project': {'_id': 0,'AIRPORT': '$_id.AIRPORT','DAY_OF_WEEK': '$_id.
 ↪DAY_OF_WEEK','count': 1}},
    {'$sort': {'DAY_OF_WEEK': 1,'count': -1}},
    {'$group': {'_id': '$DAY_OF_WEEK','AIRPORT': {'$first': '$AIRPORT'},'COUNT':
 ↪ {'$first': '$count'}}},
    {'$project': {'_id': 0,'DAY_OF_WEEK': '$_id','AIRPORT': 1,'COUNT': 1}}
])

buzy_airport_df = pd.DataFrame(busiest_airports)

plt.figure(figsize=(12, 8))
sns.barplot(data=buzy_airport_df, x='DAY_OF_WEEK', y='COUNT', hue='AIRPORT',␣
 ↪palette='viridis')
plt.title('Busiest Airports with respect to day of week')
plt.xlabel('Day of the Week')
plt.ylabel('People count')
plt.legend(title='Airport')
plt.show()
```

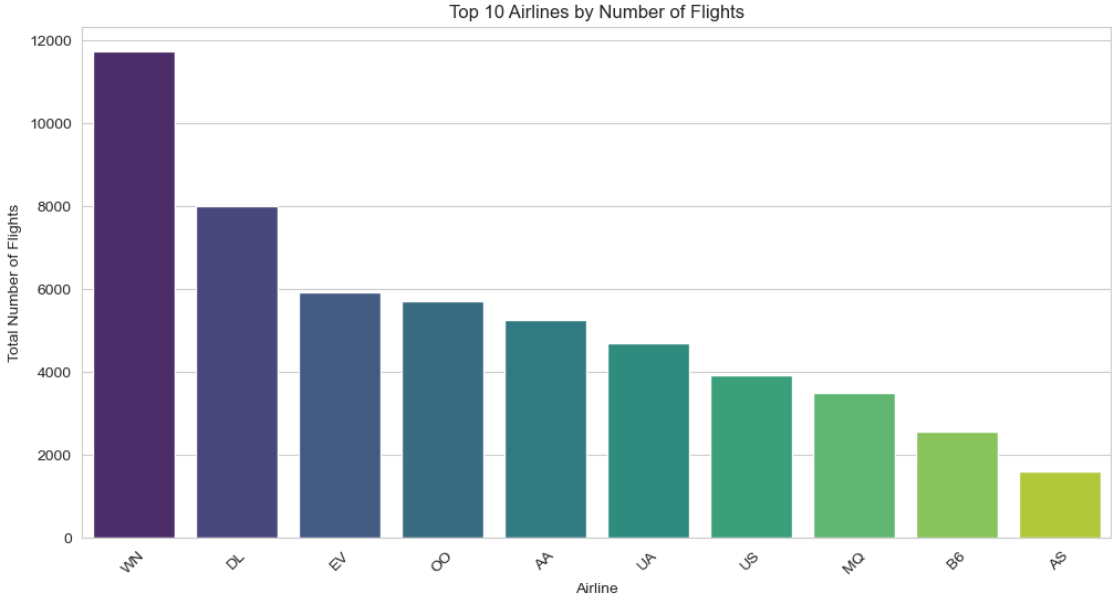


M) Find top 10 Airlines of US. Represent it by using suitable plot.

[17]:
```
# Define the aggregation pipeline
pipeline = [
    {
        "$group": {
            "_id": "$AIRLINE",  # Group by airline
            "total_flights": {"$sum": 1}  # Count the total number of flights␣
 ↪for each airline
```

```
        }
    },
    {
        "$sort": {
            "total_flights": -1  # Sort by total flights in descending order
        }
    },
    {
        "$limit": 10  # Limit the results to the top 10 airlines
    }
]

# Execute the aggregation pipeline
top_airlines = list(flights.aggregate(pipeline))

# Convert the results to a DataFrame
df = pd.DataFrame(top_airlines)

# Rename columns for clarity
df.columns = ['Airline', 'Total_Flights']

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(data=df, x='Airline', y='Total_Flights', palette='viridis')
plt.title('Top 10 Airlines by Number of Flights')
plt.xlabel('Airline')
plt.ylabel('Total Number of Flights')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_13196\4135795927.py:30:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=df, x='Airline', y='Total_Flights', palette='viridis')
```



N) Finding airlines that make the maximum, minimum number of cancellations.

[19]:

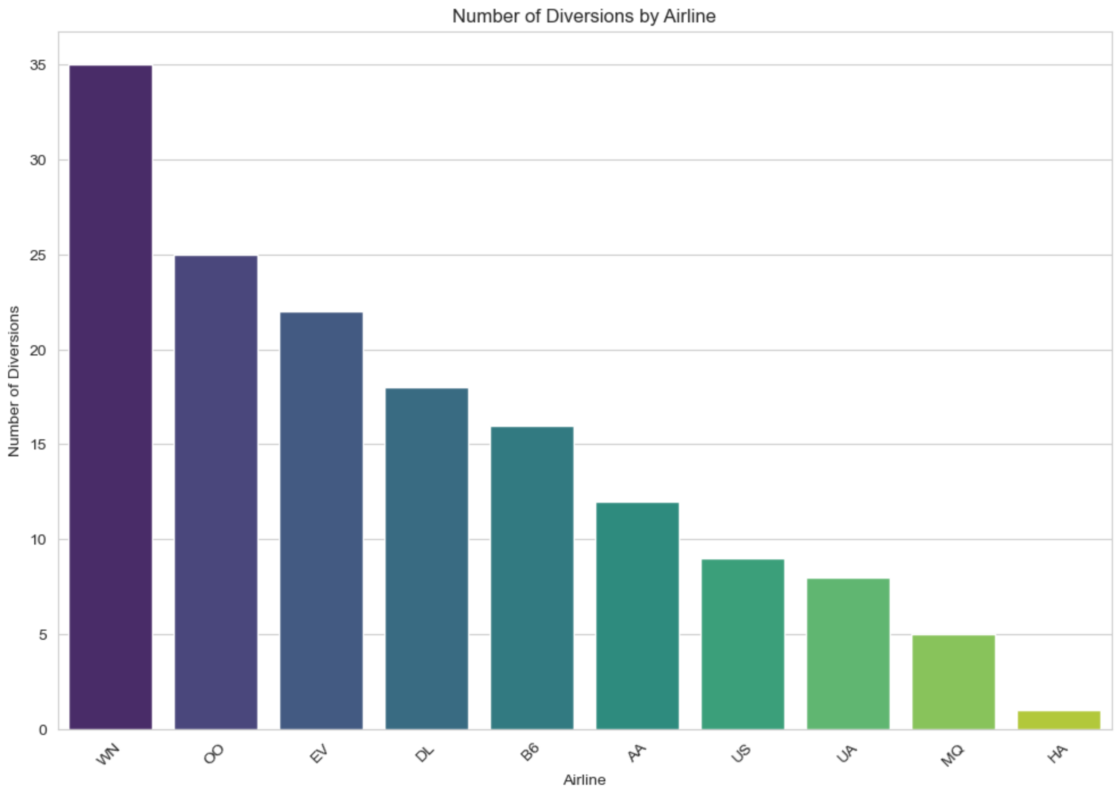
```
# Aggregation pipeline to find airlines with maximum and minimum cancellations
pipeline = [
    {"$group": {
        "_id": "$AIRLINE",
        "total_cancellations": {"$sum": {"$cond": ["$CANCELLED", 1, 0]}}
    }},
    {"$sort": {"total_cancellations": -1}}
]


results = flights.aggregate(pipeline)


df_cancellations = pd.DataFrame(list(results))

# Find maximum and minimum cancellations
max_cancellations = df_cancellations.iloc[0]  # Airline with maximum␣
 ↪cancellations
min_cancellations = df_cancellations.iloc[-1]  # Airline with minimum␣
 ↪cancellations


print("Airline with Maximum Cancellations:")
print(max_cancellations)
print("\nAirline with Minimum Cancellations:")
print(min_cancellations)
```

```
Airline with Maximum Cancellations:
_id                      MQ
total_cancellations    414
Name: 0, dtype: object

Airline with Minimum Cancellations:
_id                     HA
total_cancellations      3
Name: 13, dtype: object
```

0.0.5 o)Find and show airlines names in descending that make the most number of diversions made. [Create a suitable plot using matplotlib/seaborn]

[20]:
```
pipeline = [
    {
        "$match": {"DIVERTED": 1}  # Filter for diverted flights
    },
    {
        "$group": {
            "_id": "$AIRLINE",  # Group by airline
            "diversion_count": {"$sum": 1}  # Count diversions
        }
    },
    {
        "$sort": {"diversion_count": -1}  # Sort by diversion count in␣
 ↪descending order
    }
]


diversion_counts = list(flights.aggregate(pipeline))


df = pd.DataFrame(diversion_counts)


df.columns = ['Airline', 'Diversion Count']

# Plotting
plt.figure(figsize=(12, 8))
sns.barplot(data=df, x='Airline', y='Diversion Count', palette='viridis')
plt.title('Number of Diversions by Airline')
plt.xlabel('Airline')
plt.ylabel('Number of Diversions')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_13196\2971107579.py:27:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=df, x='Airline', y='Diversion Count', palette='viridis')
```



P) Finding days of month that see the most number of diversion and delays.

```
[21]: number_of_diversions = flights.aggregate([
          {'$group': {'_id': '$DAY','total_diversions': { '$sum': '$DIVERTED' },
                      'total_delays': { '$sum': { '$add':␣
        ↪['$ARRIVAL_DELAY','$DEPARTURE_DELAY']}}}},
          {'$project': {'_id': 0,'DAY': '$_id','total_diversions': 1,'total_delays':␣
        ↪1}},
          {'$sort': {'total_diversions': -1,'total_delays': -1}}
      ])
```

```
for i in number_of_diversions:
    print(i)
```

```
{'total_diversions': 15, 'total_delays': 83444.06075661737, 'DAY': 2}
{'total_diversions': 13, 'total_delays': 69783.05544624574, 'DAY': 1}
{'total_diversions': 12, 'total_delays': 93482.38226739684, 'DAY': 4}
{'total_diversions': 11, 'total_delays': 87888.04081527858, 'DAY': 5}
{'total_diversions': 9, 'total_delays': 36363.75671576428, 'DAY': 9}
{'total_diversions': 8, 'total_delays': 11124.27104460393, 'DAY': 14}
{'total_diversions': 7, 'total_delays': 66690.84021543432, 'DAY': 6}
{'total_diversions': 6, 'total_delays': 25587.409222091712, 'DAY': 7}
{'total_diversions': 6, 'total_delays': 20852.61663387372, 'DAY': 23}
{'total_diversions': 5, 'total_delays': 89013.04215267168, 'DAY': 3}
{'total_diversions': 5, 'total_delays': 37673.087509893914, 'DAY': 8}
{'total_diversions': 5, 'total_delays': 20776.716467495356, 'DAY': 11}
{'total_diversions': 5, 'total_delays': 18703.97097672936, 'DAY': 18}
{'total_diversions': 5, 'total_delays': 10682.520042047961, 'DAY': 30}
{'total_diversions': 4, 'total_delays': 42470.34372211236, 'DAY': 12}
{'total_diversions': 4, 'total_delays': 36335.29115011572, 'DAY': 16}
{'total_diversions': 4, 'total_delays': 23647.871143107728, 'DAY': 21}
{'total_diversions': 4, 'total_delays': 20973.560088898546, 'DAY': 20}
{'total_diversions': 4, 'total_delays': 18519.887641232315, 'DAY': 28}
{'total_diversions': 3, 'total_delays': 43927.00587738131, 'DAY': 26}
{'total_diversions': 3, 'total_delays': 33324.24749587793, 'DAY': 17}
{'total_diversions': 3, 'total_delays': 23647.769770674655, 'DAY': 27}
{'total_diversions': 3, 'total_delays': 2888.507315483577, 'DAY': 31}
{'total_diversions': 2, 'total_delays': 20210.005511789976, 'DAY': 13}
{'total_diversions': 1, 'total_delays': 22466.56386045875, 'DAY': 25}
{'total_diversions': 1, 'total_delays': 17531.80394014394, 'DAY': 15}
{'total_diversions': 1, 'total_delays': 13801.416399620788, 'DAY': 19}
{'total_diversions': 1, 'total_delays': 9446.158282999577, 'DAY': 10}
{'total_diversions': 1, 'total_delays': 4191.169105161192, 'DAY': 29}
{'total_diversions': 0, 'total_delays': 27578.527420995368, 'DAY': 22}
{'total_diversions': 0, 'total_delays': 23988.849297366163, 'DAY': 24}
```

0.0.6 Q) Write a MongoDB query to find the flights with the shortest and longest AIR_TIME. Return the flightNumber, airline, and AIR_TIME.

[22]:

```
long_short_flights = flights.aggregate([
    {'$facet':{
        'longest_time': [
            {'$project': {'_id': 0,'FLIGHT_NUMBER': 1,'AIRLINE':1,'AIR_TIME':
 ↪1}},
            {'$sort': {'AIR_TIME':-1}},
            {'$limit': 1}
        ],
        'shortest_time': [
            {'$project': {'_id': 0,'FLIGHT_NUMBER': 1,'AIRLINE':1,'AIR_TIME':
 ↪1}},
            {'$sort': {'AIR_TIME':1}},
            {'$limit': 1}
        ],
    }}
])
print(list(long_short_flights))
```

```
[{'longest_time': [{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 15, 'AIR_TIME': 654.0}],
'shortest_time': [{'AIRLINE': 'AS', 'FLIGHT_NUMBER': 65, 'AIR_TIME': 9.0}]}]
```

0.0.7 R) Finding all diverted Route from a source to destination Airport & which route is the most diverted route.

[23]:

```
most_diverted = flights.aggregate([
    {'$match' : {'DIVERTED':1}},
    {'$group' : {'_id':{'ORIGIN_AIRPORT':
 ↪'$ORIGIN_AIRPORT','DESTINATION_AIRPORT':'$DESTINATION_AIRPORT'},'COUNT':␣
 ↪{'$sum':1}}},
    {'$project': {'_id':0,'ORIGIN_AIRPORT':'$_id.
 ↪ORIGIN_AIRPORT','DESTINATION_AIRPORT':'$_id.DESTINATION_AIRPORT','COUNT':1}},
    {'$sort':{'COUNT':-1}},
    {'$limit':1}
])

for i in most_diverted:
    print(i)
```

```
{'COUNT': 2, 'ORIGIN_AIRPORT': 'JFK', 'DESTINATION_AIRPORT': 'SEA'}
```

0.0.8 S) Write a MongoDB aggregation pipeline to calculate the all aggregated values for departure delay (DEPARTURE_DELAY) and arrival delay (ARRIVAL_DELAY) for each airline, excluding flights that were either cancelled or diverted.

[24]:

```
# Define the aggregation pipeline
pipeline = [
    {
        "$match": {
            "CANCELLED": { "$ne": 1 },  # Exclude cancelled flights
            "DIVERTED": { "$ne": 1 }    # Exclude diverted flights
        }
    },
    {
        "$group": {
            "_id": "$AIRLINE",  # Group by airline
            "total_departure_delay": { "$sum": "$DEPARTURE_DELAY" },
            "average_departure_delay": { "$avg": "$DEPARTURE_DELAY" },
            "max_departure_delay": { "$max": "$DEPARTURE_DELAY" },
            "min_departure_delay": { "$min": "$DEPARTURE_DELAY" },

        }
    }
]

# Execute the aggregation pipeline
results = flights.aggregate(pipeline)

# Print the results
for result in results:
    print(result)
```

```
{'_id': 'DL', 'total_departure_delay': 77338.0, 'average_departure_delay':
9.922761098280729, 'max_departure_delay': 1166.0, 'min_departure_delay': -26.0}
{'_id': 'UA', 'total_departure_delay': 64760.0, 'average_departure_delay':
14.167578210457231, 'max_departure_delay': 473.0, 'min_departure_delay': -23.0}
{'_id': 'AS', 'total_departure_delay': 3640.0, 'average_departure_delay':
2.312579415501906, 'max_departure_delay': 400.0, 'min_departure_delay': -42.0}
{'_id': 'US', 'total_departure_delay': 29051.0, 'average_departure_delay':
7.753135842006939, 'max_departure_delay': 327.0, 'min_departure_delay': -21.0}
{'_id': 'EV', 'total_departure_delay': 63771.0, 'average_departure_delay':
11.424399856682193, 'max_departure_delay': 526.0, 'min_departure_delay': -24.0}
{'_id': 'WN', 'total_departure_delay': 114180.0, 'average_departure_delay':
10.064345526663729, 'max_departure_delay': 490.0, 'min_departure_delay': -15.0}
{'_id': 'B6', 'total_departure_delay': 37925.0, 'average_departure_delay':
15.888144113950565, 'max_departure_delay': 468.0, 'min_departure_delay': -24.0}
{'_id': 'F9', 'total_departure_delay': 18412.0, 'average_departure_delay':
23.514687100893997, 'max_departure_delay': 499.0, 'min_departure_delay': -32.0}
{'_id': 'HA', 'total_departure_delay': 855.0, 'average_departure_delay':
1.190807799442897, 'max_departure_delay': 715.0, 'min_departure_delay': -17.0}
{'_id': 'OO', 'total_departure_delay': 63435.0, 'average_departure_delay':
11.471066907775768, 'max_departure_delay': 540.0, 'min_departure_delay': -36.0}
{'_id': 'AA', 'total_departure_delay': 57060.0, 'average_departure_delay':
11.41656662665066, 'max_departure_delay': 1264.0, 'min_departure_delay': -23.0}
{'_id': 'VX', 'total_departure_delay': 5520.0, 'average_departure_delay':
9.857142857142858, 'max_departure_delay': 309.0, 'min_departure_delay': -15.0}
{'_id': 'NK', 'total_departure_delay': 15947.0, 'average_departure_delay':
15.527750730282376, 'max_departure_delay': 546.0, 'min_departure_delay': -22.0}
{'_id': 'MQ', 'total_departure_delay': 51679.0, 'average_departure_delay':
16.762568926370417, 'max_departure_delay': 494.0, 'min_departure_delay': -25.0}
```

T) Write a MongoDB query to find all flights that were delayed due to WEATHER_DELAY but were notcancelled or diverted. Include the flightNumber, airline, originAirport, and destinationAirport in the results.

[25]:

```
# Query to find flights delayed due to WEATHER_DELAY, not cancelled or diverted
query = {
    "WEATHER_DELAY": {"$gt": 0},  # Flights delayed due to WEATHER_DELAY
    "CANCELLED": 0,               # Flights that were not cancelled
    "DIVERTED": 0                 # Flights that were not diverted
}

# Fields to include in the results
projection = {
    "FLIGHT_NUMBER": 1,
    "AIRLINE": 1,
    "ORIGIN_AIRPORT": 1,
    "DESTINATION_AIRPORT": 1,
    "_id": 0
}

# Execute the query
results = flights.find(query, projection).limit(10)

# Print the results
for flight in results:
    print(flight)
```

```
{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 532, 'ORIGIN_AIRPORT': 'ORD',
'DESTINATION_AIRPORT': 'DCA'}
{'AIRLINE': 'US', 'FLIGHT_NUMBER': 1784, 'ORIGIN_AIRPORT': 'BWI',
'DESTINATION_AIRPORT': 'PHX'}
{'AIRLINE': 'MQ', 'FLIGHT_NUMBER': 3019, 'ORIGIN_AIRPORT': 'ORD',
'DESTINATION_AIRPORT': 'OKC'}
{'AIRLINE': 'MQ', 'FLIGHT_NUMBER': 3564, 'ORIGIN_AIRPORT': 'GSO',
'DESTINATION_AIRPORT': 'LGA'}
{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 1667, 'ORIGIN_AIRPORT': 'ORD',
'DESTINATION_AIRPORT': 'PDX'}
{'AIRLINE': 'DL', 'FLIGHT_NUMBER': 1788, 'ORIGIN_AIRPORT': 'ATL',
'DESTINATION_AIRPORT': 'MEM'}
{'AIRLINE': 'DL', 'FLIGHT_NUMBER': 424, 'ORIGIN_AIRPORT': 'JFK',
'DESTINATION_AIRPORT': 'LAX'}
{'AIRLINE': 'MQ', 'FLIGHT_NUMBER': 3201, 'ORIGIN_AIRPORT': 'ORD',
'DESTINATION_AIRPORT': 'BNA'}
{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 1718, 'ORIGIN_AIRPORT': 'LAX',
'DESTINATION_AIRPORT': 'KOA'}
{'AIRLINE': 'DL', 'FLIGHT_NUMBER': 338, 'ORIGIN_AIRPORT': 'DTW',
'DESTINATION_AIRPORT': 'ATL'}
```

0.0.9 U) Write a MongoDB query to find all flights that were delayed both at departure (DEPARTURE_DELAY)and arrival (ARRIVAL_DELAY). Return the count of such Flights which are delayed.

[32]:

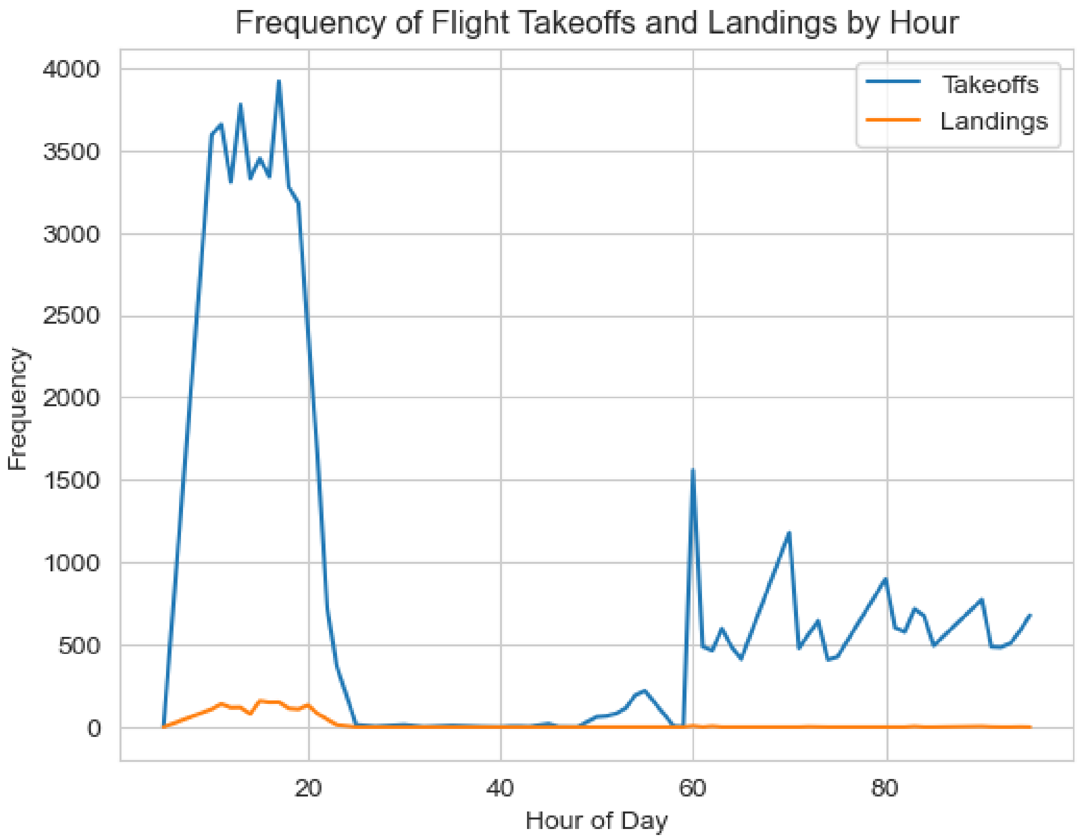
```
# Query to find flights delayed at both departure and arrival
query = {
    "DEPARTURE_DELAY": {"$gt": 0},  # Delayed at departure
    "ARRIVAL_DELAY": {"$gt": 0}     # Delayed at arrival
}

# Find all such flights
delayed_flights = flights.find(query, {
    "FLIGHT_NUMBER": 1,
    "AIRLINE": 1,
    "DEPARTURE_DELAY": 1,
    "ARRIVAL_DELAY": 1,
    "_id": 0
}).limit(20)

# Print the results
print("Flights delayed at both departure and arrival:")
for flight in delayed_flights:
    print(flight)
```

```
Flights delayed at both departure and arrival:
{'AIRLINE': 'EV', 'FLIGHT_NUMBER': 5170, 'DEPARTURE_DELAY': 19.0,
'ARRIVAL_DELAY': 33.0}
{'AIRLINE': 'MQ', 'FLIGHT_NUMBER': 3584, 'DEPARTURE_DELAY': 36.0,
'ARRIVAL_DELAY': 32.0}
{'AIRLINE': 'B6', 'FLIGHT_NUMBER': 716, 'DEPARTURE_DELAY': 90.0,
'ARRIVAL_DELAY': 96.0}
{'AIRLINE': 'WN', 'FLIGHT_NUMBER': 518, 'DEPARTURE_DELAY': 10.0,
'ARRIVAL_DELAY': 9.0}
{'AIRLINE': 'HA', 'FLIGHT_NUMBER': 371, 'DEPARTURE_DELAY': 30.0,
'ARRIVAL_DELAY': 33.0}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 6196, 'DEPARTURE_DELAY': 65.0,
'ARRIVAL_DELAY': 56.0}
{'AIRLINE': 'US', 'FLIGHT_NUMBER': 1756, 'DEPARTURE_DELAY': 68.0,
'ARRIVAL_DELAY': 54.0}
{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 792, 'DEPARTURE_DELAY': 6.0, 'ARRIVAL_DELAY':
35.0}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 2699, 'DEPARTURE_DELAY': 298.0,
'ARRIVAL_DELAY': 293.0}
{'AIRLINE': 'UA', 'FLIGHT_NUMBER': 532, 'DEPARTURE_DELAY': 10.0,
'ARRIVAL_DELAY': 31.0}
{'AIRLINE': 'VX', 'FLIGHT_NUMBER': 251, 'DEPARTURE_DELAY': 31.0,
'ARRIVAL_DELAY': 31.0}
{'AIRLINE': 'F9', 'FLIGHT_NUMBER': 661, 'DEPARTURE_DELAY': 80.0,
'ARRIVAL_DELAY': 82.0}
{'AIRLINE': 'US', 'FLIGHT_NUMBER': 686, 'DEPARTURE_DELAY': 100.0,
'ARRIVAL_DELAY': 100.0}
{'AIRLINE': 'OO', 'FLIGHT_NUMBER': 4544, 'DEPARTURE_DELAY': 140.0,
'ARRIVAL_DELAY': 153.0}
{'AIRLINE': 'WN', 'FLIGHT_NUMBER': 1165, 'DEPARTURE_DELAY': 130.0,
'ARRIVAL_DELAY': 130.0}
{'AIRLINE': 'EV', 'FLIGHT_NUMBER': 3936, 'DEPARTURE_DELAY': 278.0,
'ARRIVAL_DELAY': 344.0}
{'AIRLINE': 'MQ', 'FLIGHT_NUMBER': 3337, 'DEPARTURE_DELAY': 44.0,
'ARRIVAL_DELAY': 50.0}
{'AIRLINE': 'AA', 'FLIGHT_NUMBER': 1419, 'DEPARTURE_DELAY': 22.0,
'ARRIVAL_DELAY': 33.0}
{'AIRLINE': 'EV', 'FLIGHT_NUMBER': 5950, 'DEPARTURE_DELAY': 4.0,
'ARRIVAL_DELAY': 8.0}
{'AIRLINE': 'EV', 'FLIGHT_NUMBER': 6172, 'DEPARTURE_DELAY': 11.329091145205275,
'ARRIVAL_DELAY': 7.545457931394093}
```

V) Write a MongoDB query to calculate the frequency of flight takeoffs and landings within defined timeintervals (e.g., every hour) throughout the day. Generate a Suitable Plot.

[28]:

```
pipeline = [
    {"$project": {
        "takeoff_hour": {"$toInt": {"$substr": [{"$toString":␣
  ↪"$SCHEDULED_DEPARTURE"}, 0, 2]}},
        "landing_hour": {"$toInt": {"$substr": [{"$toString":␣
  ↪"$SCHEDULED_ARRIVAL"}, 0, 2]}}
    }},
    {"$group": {
        "_id": "$takeoff_hour",
        "takeoffs": {"$sum": 1},
        "landings": {"$sum": {"$cond": [{"$eq": ["$landing_hour",␣
  ↪"$takeoff_hour"]}, 1, 0]}}
    }},
    {"$sort": {"_id": 1}}
]

results = flights.aggregate(pipeline)
df_time_intervals = pd.DataFrame(list(results))

# Plot (If needed):
plt.plot(df_time_intervals['_id'], df_time_intervals['takeoffs'],␣
  ↪label='Takeoffs')
plt.plot(df_time_intervals['_id'], df_time_intervals['landings'],␣
  ↪label='Landings')
plt.xlabel('Hour of Day')
plt.ylabel('Frequency')
plt.title('Frequency of Flight Takeoffs and Landings by Hour')
plt.legend()
plt.show()
```

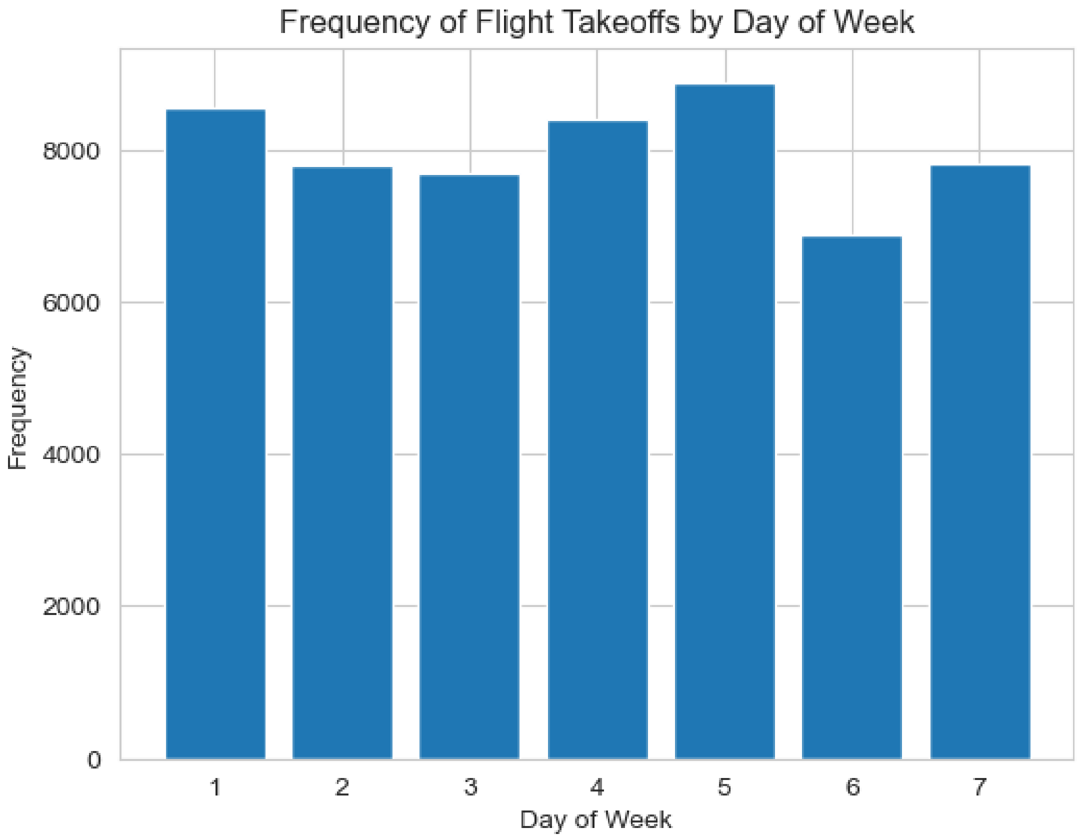


0.0.10 W) Write a MongoDB query to calculate the frequency of flight takeoffs and landings within defined weekof day. Generate a Suitable Plot.

[29]:

```
pipeline = [
    {"$project": {
        "day_of_week": "$DAY_OF_WEEK"
    }},
    {"$group": {
        "_id": "$day_of_week",
        "takeoffs": {"$sum": 1}
    }},
    {"$sort": {"_id": 1}}
]

results = flights.aggregate(pipeline)
df_weekly_intervals = pd.DataFrame(list(results))
```

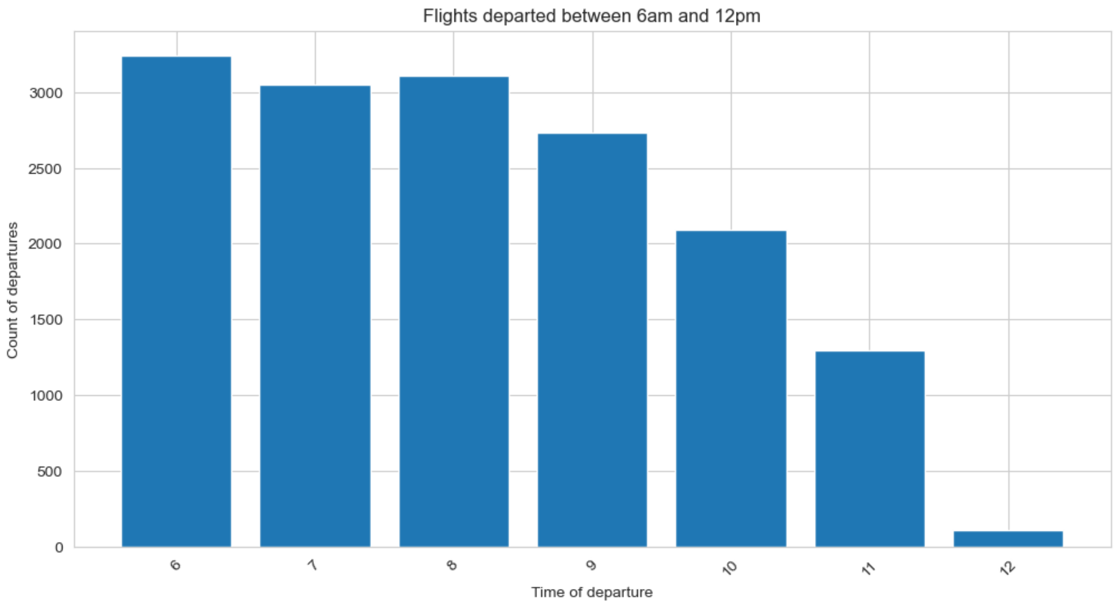
```
# Plot (If needed):
plt.bar(df_weekly_intervals['_id'], df_weekly_intervals['takeoffs'])
plt.xlabel('Day of Week')
plt.ylabel('Frequency')
plt.title('Frequency of Flight Takeoffs by Day of Week')
plt.show()
```



X) Write a MongoDB query to find all flights that departed between 6 AM and 12 PM (noon) local time, regardless of the date. Return the flightNumber, airline, and departureTime. Generate a Bar Plot usingTime (x-axis) and Frequency (y-axis).

[30]:
```
flight_time = flights.aggregate([
    {'$match': {'DEPARTURE_TIME': {'$gte': 6, '$lte': 12}, 'ARRIVAL_TIME':␣
 ↪{'$gte': 6, '$lte':12}}},
    {'$project' : {'_id':0,'FLIGHT_NUMBER':1,'AIRLINE':1,'DEPARTURE_TIME':1}}
])
flight_time_df = pd.DataFrame(flight_time)
freq = flight_time_df.groupby('DEPARTURE_TIME')['DEPARTURE_TIME'].value_counts()
```

```
plt.figure(figsize=(12, 6))
plt.bar(height=freq.values, x=freq.index)
plt.xlabel('Time of departure')
plt.ylabel('Count of departures')
plt.title('Flights departed between 6am and 12pm')
plt.xticks(rotation=45)

plt.show()
```



0.0.11 Y) When is the best time of day/day of week/time of a year to fly with minimum delays?

[31]:
```
pipeline = [
    {
        "$match": {
            "$and": [
                {"DEPARTURE_DELAY": {"$ne": None}},
                {"ARRIVAL_DELAY": {"$ne": None}},
                {"DEPARTURE_DELAY": {"$ne": float('nan')}},
                {"ARRIVAL_DELAY": {"$ne": float('nan')}},
                {"SCHEDULED_DEPARTURE": {"$ne": None}},
                {"$expr": {"$gte": [{"$toInt": {"$substr": [{"$toString":␣
 ↪"$SCHEDULED_DEPARTURE"}, 0, 2]}}, 0]}},
                {"$expr": {"$lte": [{"$toInt": {"$substr": [{"$toString":␣
 ↪"$SCHEDULED_DEPARTURE"}, 0, 2]}}, 23]}},
            ]
        }
    },
    {
        "$project": {
            "hour_of_day": {
                "$toInt": {"$substr": [{"$toString": "$SCHEDULED_DEPARTURE"},␣
 ↪0, 2]}
            },
            "day_of_week": "$DAY_OF_WEEK",
            "month_of_year": "$MONTH",
            "total_delay": {"$add": ["$DEPARTURE_DELAY", "$ARRIVAL_DELAY"]}
        }
    },
    {
        "$match": {
            "total_delay": {"$gte": 0}  # Filter out negative total delays
        }
    },
    {
        "$group": {
            "_id": {
                "hour_of_day": "$hour_of_day",
                "day_of_week": "$day_of_week",
                "month_of_year": "$month_of_year"
            },
            "average_delay": {"$avg": "$total_delay"}
        }
    },
    {"$sort": {"average_delay": 1}},
    {"$limit": 1}
]

results = flights.aggregate(pipeline)
best_time = list(results)[0]

# Display result
print("Best time to fly with minimum delays:")
print(best_time)
```

```
Best time to fly with minimum delays:
{'_id': {'hour_of_day': 23, 'day_of_week': 5, 'month_of_year': 2},
'average_delay': 13.222222222222221}
```

[ ]: