msmdesign.nyc

data analytics & insights   information design

exploratorydataanalysis

> ### What is Exploratory Data Analysis?
>
> **Exploratory data analysis (EDA)** is a technique used by data scientists to inspect, characterize and briefly summarize the contents of a dataset. EDA is often the first step when encountering a new or unfamiliar dataset. EDA helps the data scientist become acquainted with a dataset and test some basic assumptions about the data. By the end of the EDA process, some initial insights can be drawn from the dataset and a framework for further analysis or modeling is established.

Swimming Beach Attendance

Dataset Analyzed: *Swimming Beach Attendance*

About This Dataset: Attendance records for NYC Parks swimming beaches. Each row is a daily beach attendance. Data provided by the Department of Parks and Recreation (DPR), the City of New York: https://data.cityofnewyork.us/Business/NYC-Business-Acceleration-Businesses-Served-and-Jo/9b9u-8989

Acknowledgements: NYC Open Data https://opendata.cityofnewyork.us/

EDA Catalogue Number: INS-009

EDA Publication Date: Monnday, January 9, 2023

Language: Python

Libraries Used: NumPy, pandas, matplotlib, seaborn

EDA Author: David White

Contact: david@msmdesign.nyc | msmdesign.nyc

---

0. Prepare the workspace

0.1 Import Python libraries, packages and functions

In [1]:
```
# import libraries for data wrangling, aggregate functions and basic descriptive stati
import numpy as np
import pandas as pd

# import data visualization packages
import matplotlib.pyplot as plt
import seaborn as sns
```

0.2 Adjust display options to make plots easier to read and understand

In [132…
```
# specify seaborn styling options
sns.set_theme(
    context='talk',
    style='whitegrid',
    palette='tab10',
    font='Courier New',
    font_scale=1.15)

# allow plots to display inline within the notebook
%matplotlib inline
```

0.3 Set Markdown tables to align-left within notebook cells

In [5]:
```
%%html
<style>
table {float:left}
</style>
```

0.4 Display all rows of output by default

In [6]:
```
pd.set_option('display.max_rows', None)

# to reset:
# pd.reset_option('display.max_rows')
```

0.5 Format large numbers and display floating point values to two decimal places

In [7]:
```
pd.set_option('display.float_format',  '{:,.2f}'.format)

# to reset:
# pd.reset_option('display.float_format')
```

0.6 Load the raw data file into the notebook and visually confirm that it has been read in as expected

In [8]:
```
# load the data from a csv file (stored locally) into a new DataFrame object

csv = r"F:\Creative Cloud Files\MSM Client 001 - Mister Shepherd Media LLC\MSM Design\

beach_attendance = pd.read_csv(csv, encoding='utf-8')
```

In [9]:
```
# glimpse the first three rows

beach_attendance.head(3)
```

Out[9]:

	Date	Beach	Attendance
0	05/27/2017	Orchard	550.00
1	05/27/2017	Coney Island	30,000.00
2	05/27/2017	Manhattan	3,800.00

In [10]:
```
# glimpse the last three rows

beach_attendance.tail(3)
```

Out[10]:

	Date	Beach	Attendance
4205	09/12/2021	South beach	3,400.00
4206	09/12/2021	Wolfe's Pond	0.00
4207	09/12/2021	Cedar Grove	0.00

In [11]:
```
# glimpse ten randomly selected rows

beach_attendance.sample(10, random_state=42)
```

Out[11]:

	Date	Beach	Attendance
1721	06/05/2019	Coney Island	7,000.00
3549	06/22/2021	South beach	700.00
2555	05/31/2020	Rockaway	21,500.00
4020	08/20/2021	Midland	2,588.00
3953	08/12/2021	Coney Island	78,500.00
3676	07/08/2021	Midland	840.00
2132	07/26/2019	Midland	1,370.00
2547	05/30/2020	Rockaway	40,000.00
2694	06/17/2020	Wolfe's Pond	300.00
3505	06/17/2021	Coney Island	9,500.00

The data has been loaded and has been read in as expected.

0.7. Check the data type of each column

In [13]:

```
# display a listing of each of the DataFrame's columns and its data type

beach_attendance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4208 entries, 0 to 4207
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        4208 non-null   object
 1   Beach       4208 non-null   object
 2   Attendance  4207 non-null   float64
dtypes: float64(1), object(2)
memory usage: 98.8+ KB
```

We'll need to change the data type of the 'Date' and 'Beach' columns

0.8 Refer to the data dictionary and make sure that our DataFrame's data types match the source data. Reassign data types where needed.

In [14]:

```
# cast column(s) containing dates to datetime data type

beach_attendance['Date'] = pd.to_datetime(beach_attendance['Date'], errors='coerce')
```

In [17]:

```
# cast column(s) containing categorical varibles to categorical data type

beach_attendance['Beach'] = beach_attendance['Beach'].astype('category')
```

In [18]:

```
# display the DataFrame info once again to confirm that the data type changes have bee
```

```
beach_attendance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4208 entries, 0 to 4207
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Date        4208 non-null   datetime64[ns]
 1   Beach       4208 non-null   category
 2   Attendance  4207 non-null   float64
dtypes: category(1), datetime64[ns](1), float64(1)
memory usage: 70.3 KB
```

---

1. Describe the characteristics of the dataset

1.1 How many rows and how many columns are in our dataset?

In [19]:
```
# display the number of rows and columns in the DataFrame

rows = beach_attendance.shape[0]
columns = beach_attendance.shape[1]

print(f'There are {rows} rows and {columns} columns in the dataset.')
```

```
There are 4208 rows and 3 columns in the dataset.
```

1.2 Identify the index of our DataFrame

In [20]:
```
# display the index of the DataFrame

beach_attendance.index
```

Out[20]:
```
RangeIndex(start=0, stop=4208, step=1)
```

Our DataFrame has an interger index. We know from the data dictionary that each row is an individual constituent case.

1.3 What are the column headings in our dataset?

In [21]:
```
# display a list of the DataFrame's columns

list(beach_attendance.columns)
```

Out[21]:
```
['Date', 'Beach', 'Attendance']
```

1.4 What are the data types of each column?

In [22]:
```
# display the data type of each column in the DataFrame

beach_attendance.dtypes
```

Out[22]:
```
Date          datetime64[ns]
Beach               category
Attendance           float64
dtype: object
```

1.5 How many null values are in each column?

In [23]:
```python
# display the number of missing values in each column of the DataFrame

beach_attendance.isna().sum()
```

Out[23]:
```
Date          0
Beach         0
Attendance    1
dtype: int64
```

1.6 How many unique values are there in each column?

In [24]:
```python
# display the count of unique elements in each column

beach_attendance.nunique(axis=0, dropna=True)
```

Out[24]:
```
Date           526
Beach            8
Attendance    1075
dtype: int64
```

---

2. Briefly summarize the contents of the dataset

2.1 Summarize the columns containing numerical variables

In [25]:
```python
# describe numeric columns only

num_cols = ['Attendance']

beach_attendance[num_cols].describe(include=[np.number])
```

Out[25]:

	Attendance
count	4,207.00
mean	15,815.81
std	47,829.87
min	0.00
25%	362.50
50%	1,700.00
75%	8,000.00
max	1,520,000.00

2.2 Summarize the columns containing datetime variables

In [26]:
```
# summarize the data contained in columns with the 'datetime' data type only

date_cols = ['Date']

beach_attendance[date_cols].describe(datetime_is_numeric=True)
```

Out[26]:

	Date
count	4208
mean	2019-07-25 21:23:57.262357504
min	2017-05-27 00:00:00
25%	2018-06-20 00:00:00
50%	2019-07-22 12:00:00
75%	2020-08-14 00:00:00
max	2021-09-12 00:00:00

2.3 Summarize the columns containing categorical variables

In [27]:
```
# summarize the data contained in columns with the 'category' data type only

beach_attendance.describe(include=['category'])
```

Out[27]:

	Beach
count	4208
unique	8
top	Cedar Grove
freq	526

In [ ]:
```
### examples of slicing and subsetting data ###

# select data by index location
# df3 = df2.iloc[100:111,2:5]


# select data based on a single condition
# df_females = df.loc[df['Sex']=='female']
# df_minors = df.loc[df['Age']<=18]


# select data based on multiple conditions
# df_women_and_children = df.loc[(df['Sex']=='female') | (df['Age'] < 18)]


# select data matching one of a set of values
```

```
# step 1 - create a mask
# bridge_and_tunnel = df_socrata['establishment_record_establishment_borough'].isin([
# step 2 - apply the mask to the original DataFrame
# df_socrata[bridge_and_tunnel]


# select data matching a substring
# step 1 - cast the column as a string dtype if it is not aslready
# df['Name'] = df['Name'].astype('string')
# step 2 - create a mask
# patricks = df['Name'].str.contains('Patrick')
# step 3 - apply the mask to the original DataFrame
# df[patricks]
```

---

3. Examine the individual variables in the dataset

3.1 Analysis of the 'Date' column

In [28]:

```
# what is the range of dates represented in the dataset?

print(beach_attendance['Date'].min())
print('to')
print(beach_attendance['Date'].max())
```

```
2017-05-27 00:00:00
to
2021-09-12 00:00:00
```

In [31]:

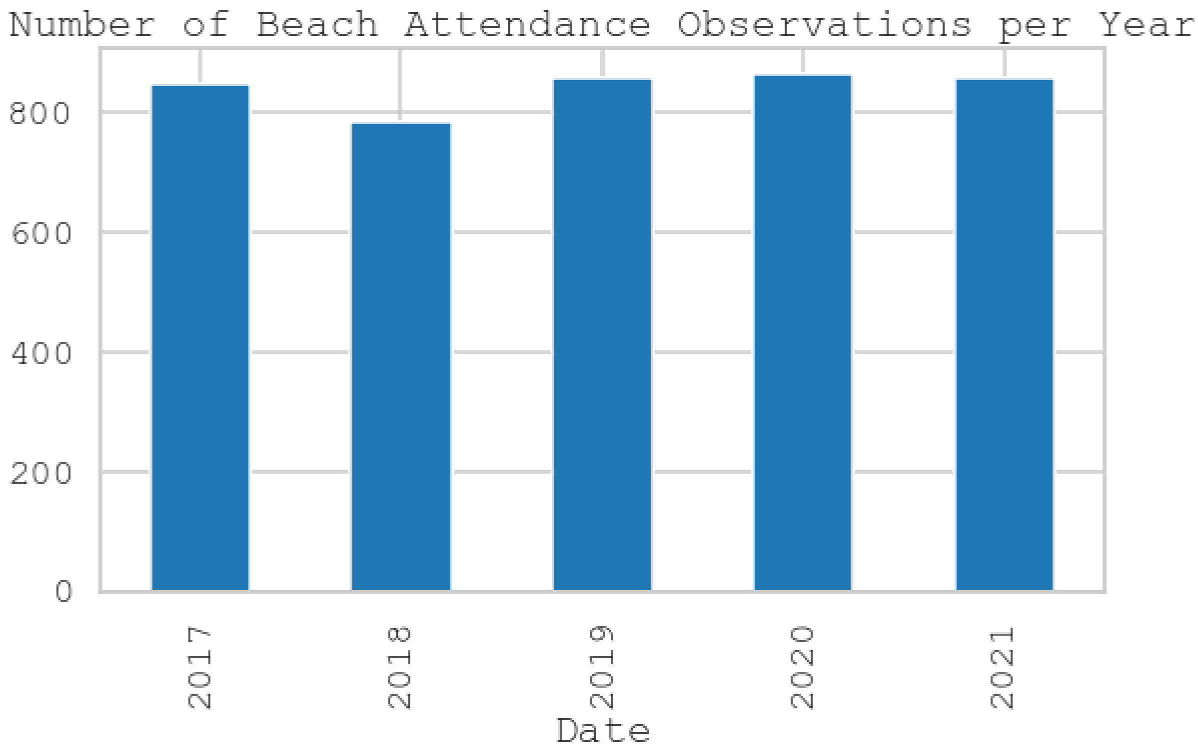
```
# how many obervations were made each year?

beach_attendance['Date'].groupby(beach_attendance['Date'].dt.year).count()
```

Out[31]:

```
Date
2017    848
2018    784
2019    856
2020    864
2021    856
Name: Date, dtype: int64
```

In [133…

```
beach_attendance['Date'].groupby(beach_attendance['Date'].dt.year).count().plot(kind='
                                                                                 figsiz
                                                                                 title=
```



In [41]:

```
# how many observations were made each year for each beach?

beach_attendance.groupby(['Beach', beach_attendance['Date'].dt.year])['Date'].count()
```

Out[41]:
```
Beach         Date
Cedar Grove   2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Coney Island  2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Manhattan     2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Midland       2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Orchard       2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Rockaway      2017    106
              2018     98
              2019    107
              2020    108
              2021    107
South beach   2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Wolfe's Pond  2017    106
              2018     98
              2019    107
              2020    108
              2021    107
Name: Date, dtype: int64
```

3.2 Analysis of the Beach Column

In [51]:
```
# how many different beaches are represented in the dataset?

beach_attendance['Beach'].nunique()
```

Out[51]:
```
8
```

In [52]:
```
# what are the names of each beach represented in the dataset?

beach_attendance['Beach'].unique()
```

Out[52]:
```
['Orchard', 'Coney Island', 'Manhattan', 'Rockaway', 'Midland', 'South beach', 'Wolf
e's Pond', 'Cedar Grove']
Categories (8, object): ['Cedar Grove', 'Coney Island', 'Manhattan', 'Midland', 'Orch
ard', 'Rockaway', 'South beach', 'Wolfe's Pond']
```

3.3 Analysis of the 'Attendance' column

In [58]:
```
# what are the highest attendance totals on record?

beach_attendance.nlargest(10, 'Attendance')
```

Out[58]:

	Date	Beach	Attendance
1161	2018-07-04	Coney Island	1,520,000.00
297	2017-07-03	Coney Island	705,000.00
1491	2018-08-14	Rockaway	581,500.00
1489	2018-08-14	Coney Island	426,145.00
2945	2020-07-19	Coney Island	400,000.00
393	2017-07-15	Coney Island	385,000.00
3641	2021-07-04	Coney Island	383,000.00
281	2017-07-01	Coney Island	370,000.00
121	2017-06-11	Coney Island	360,000.00
673	2017-08-19	Coney Island	345,000.00

In [57]:
```
# what are the lowesest attendance totals on record, exluding days of zero attendance?

beach_attendance.loc[beach_attendance['Attendance'] > 0].nsmallest(10, 'Attendance')
```

Out[57]:

	Date	Beach	Attendance
1071	2018-06-22	Cedar Grove	3.00
751	2017-08-28	Cedar Grove	5.00
927	2018-06-04	Cedar Grove	5.00
1079	2018-06-23	Cedar Grove	5.00
1183	2018-07-06	Cedar Grove	5.00
2495	2020-05-23	Cedar Grove	5.00
383	2017-07-13	Cedar Grove	8.00
2535	2020-05-28	Cedar Grove	9.00
39	2017-05-31	Cedar Grove	10.00
503	2017-07-28	Cedar Grove	10.00

In [120…
```
# what are the attendance totals per year?
```

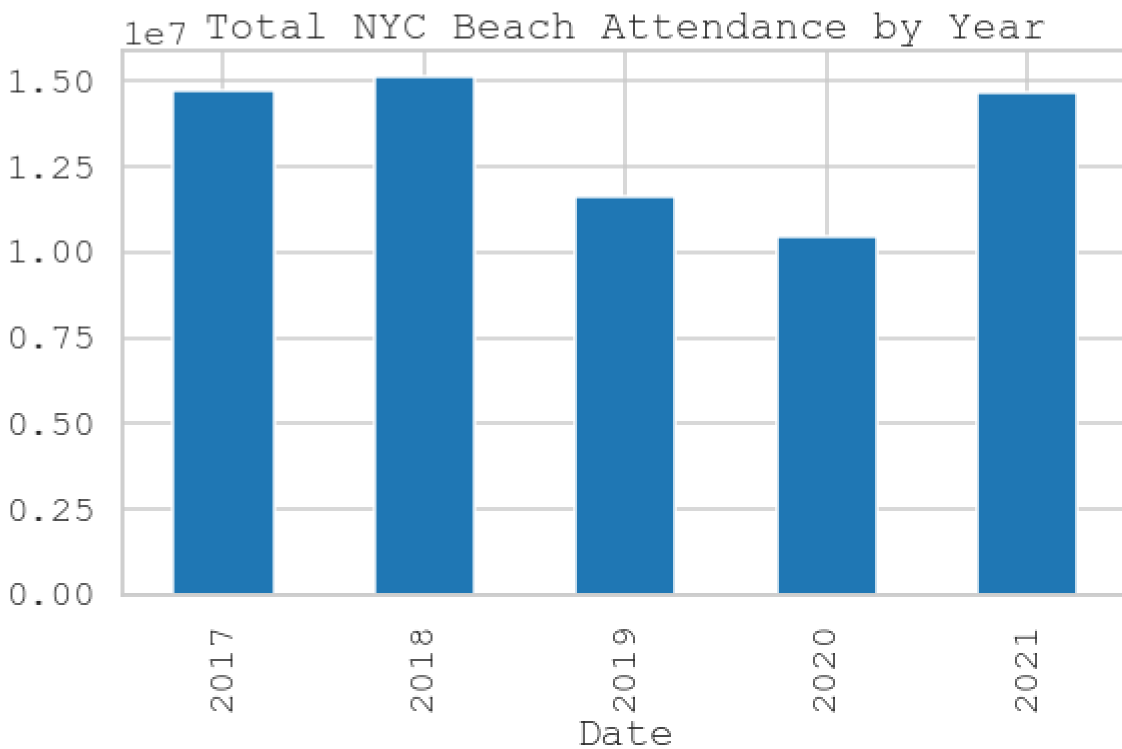
```
beach_attendance.groupby(beach_attendance['Date'].dt.year)['Attendance'].sum()
```

Out[120]:

```
Date
2017   14,733,428.00
2018   15,136,479.00
2019   11,604,986.00
2020   10,425,960.00
2021   14,636,259.00
Name: Attendance, dtype: float64
```

In [134…

```
beach_attendance.groupby(beach_attendance['Date'].dt.year)['Attendance'].sum().plot(ki
                                                                                   fi
                                                                                   ti
```



In [124…

```
# what are the attendance totals per beach, per year?

beach_attendance.groupby([beach_attendance['Date'].dt.year, 'Beach'])['Attendance'].su
```

Out[124]:
```
Date  Beach
2017  Cedar Grove       15,197.00
      Coney Island   6,675,385.00
      Manhattan        216,905.00
      Midland          345,250.00
      Orchard        1,969,148.00
      Rockaway       5,146,595.00
      South beach      333,710.00
      Wolfe's Pond      31,238.00
2018  Cedar Grove       19,553.00
      Coney Island   7,099,930.00
      Manhattan        250,835.00
      Midland          539,900.00
      Orchard        1,620,833.00
      Rockaway       5,042,498.00
      South beach      535,385.00
      Wolfe's Pond      27,545.00
2019  Cedar Grove       44,410.00
      Coney Island   4,181,550.00
      Manhattan        221,052.00
      Midland          391,910.00
      Orchard        1,566,670.00
      Rockaway       4,773,150.00
      South beach      390,289.00
      Wolfe's Pond      35,955.00
2020  Cedar Grove       24,794.00
      Coney Island   5,319,434.00
      Manhattan        177,376.00
      Midland          208,659.00
      Orchard        1,147,153.00
      Rockaway       3,197,162.00
      South beach      297,332.00
      Wolfe's Pond      54,050.00
2021  Cedar Grove       14,778.00
      Coney Island   7,991,535.00
      Manhattan        172,885.00
      Midland          297,326.00
      Orchard        1,693,505.00
      Rockaway       3,825,475.00
      South beach      449,105.00
      Wolfe's Pond     191,650.00
Name: Attendance, dtype: float64
```

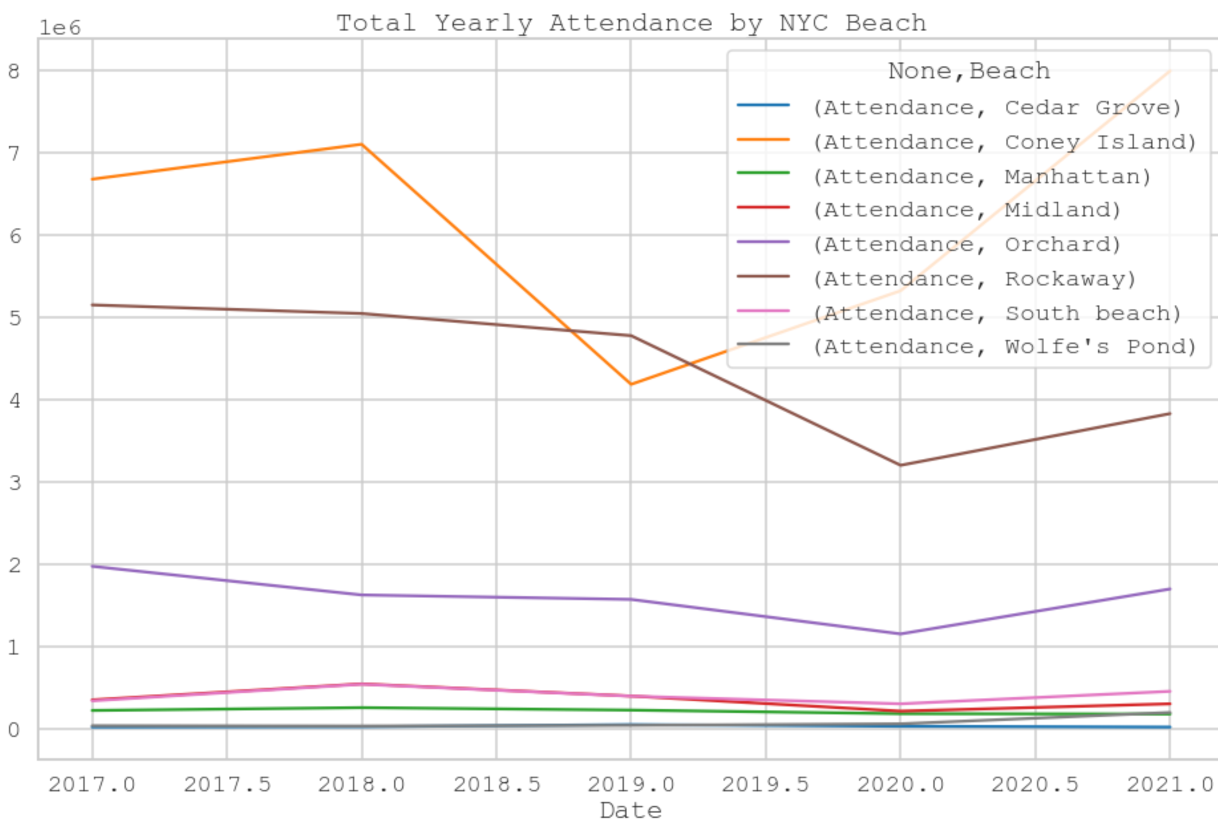
In [138…
```
beach_attendance.groupby([beach_attendance['Date'].dt.year, 'Beach']).sum().unstack()
```

Out[138]:

	Attendance							
Beach	**Cedar Grove**	**Coney Island**	**Manhattan**	**Midland**	**Orchard**	**Rockaway**	**South beach**	**Wolfe's Pond**
Date								
2017	15,197.00	6,675,385.00	216,905.00	345,250.00	1,969,148.00	5,146,595.00	333,710.00	31,238.00
2018	19,553.00	7,099,930.00	250,835.00	539,900.00	1,620,833.00	5,042,498.00	535,385.00	27,545.00
2019	44,410.00	4,181,550.00	221,052.00	391,910.00	1,566,670.00	4,773,150.00	390,289.00	35,955.00
2020	24,794.00	5,319,434.00	177,376.00	208,659.00	1,147,153.00	3,197,162.00	297,332.00	54,050.00
2021	14,778.00	7,991,535.00	172,885.00	297,326.00	1,693,505.00	3,825,475.00	449,105.00	191,650.00

In [139…

```
beach_attendance.groupby([beach_attendance['Date'].dt.year, 'Beach']).sum().unstack().
```



In [128…

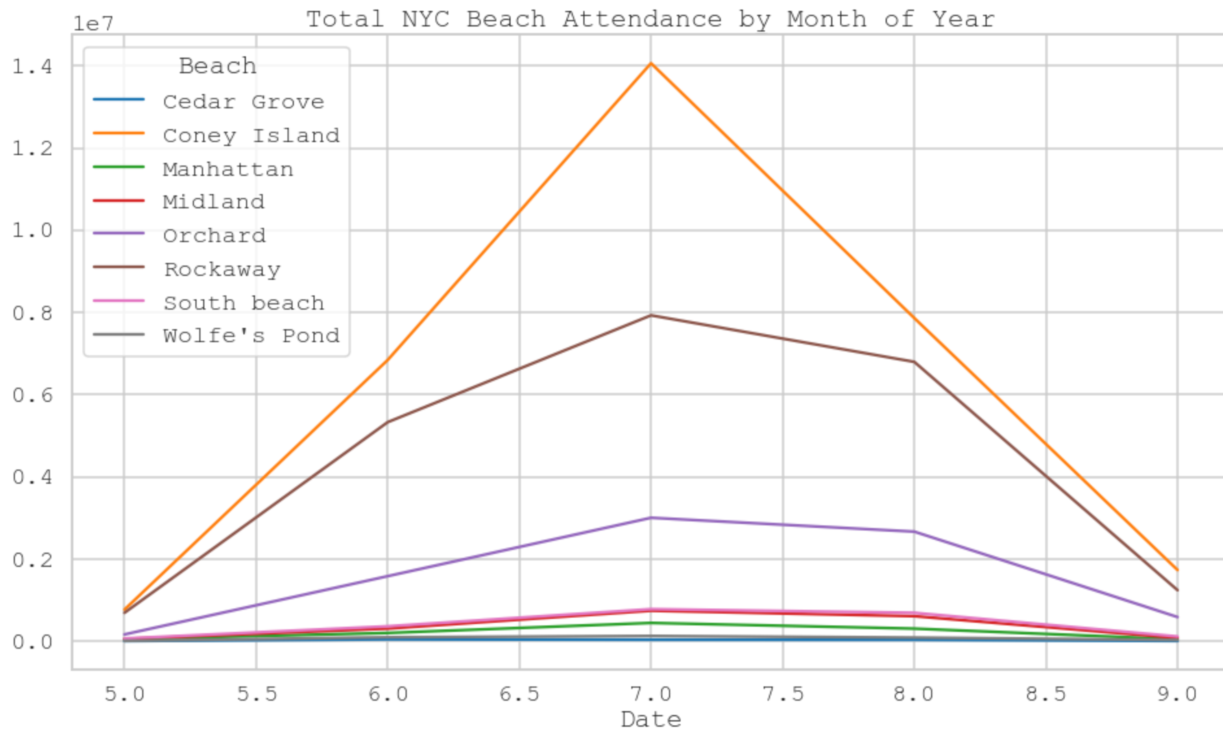
```
# what are the attendance totals per beach, per month of the year?

beach_attendance.groupby([beach_attendance['Date'].dt.month, 'Beach'])['Attendance'].s
```

Out[128]:

```
Date  Beach
5     Cedar Grove          2,155.00
      Coney Island       767,479.00
      Manhattan           47,067.00
      Midland             53,735.00
      Orchard            163,095.00
      Rockaway           689,773.00
      South beach         64,090.00
      Wolfe's Pond         6,872.00
6     Cedar Grove         43,973.00
      Coney Island     6,843,100.00
      Manhattan          198,895.00
      Midland            305,113.00
      Orchard          1,581,357.00
      Rockaway         5,329,414.00
      South beach        361,035.00
      Wolfe's Pond        93,815.00
7     Cedar Grove         35,204.00
      Coney Island    14,063,830.00
      Manhattan          443,006.00
      Midland            740,328.00
      Orchard          3,001,914.00
      Rockaway         7,928,850.00
      South beach        778,576.00
      Wolfe's Pond       126,001.00
8     Cedar Grove         33,215.00
      Coney Island     7,862,540.00
      Manhattan          304,745.00
      Midland            607,472.00
      Orchard          2,665,108.00
      Rockaway         6,796,593.00
      South beach        688,918.00
      Wolfe's Pond        86,795.00
9     Cedar Grove          4,185.00
      Coney Island     1,730,885.00
      Manhattan           45,340.00
      Midland             76,397.00
      Orchard            585,835.00
      Rockaway         1,240,250.00
      South beach        113,202.00
      Wolfe's Pond        26,955.00
Name: Attendance, dtype: float64
```

In [136…

```python
beach_attendance.groupby([beach_attendance['Date'].dt.month, 'Beach'])['Attendance'].s
```



Next steps

In [142...

```
# export data for data graphic creation

beach_attendance_trends = beach_attendance.groupby([beach_attendance['Date'].dt.year,
```

In [141...

```
beach_attendance_trends.to_csv('beach_attendance_trends.csv')
```
