COLLLEGE CODE:4212
REG NO-421221243030

DATA ANALYTICS WITH COGNOS-GROUP2

WEBSITE TRAFFIC ANALYSIS-PHASE4

In the previous phases we have discussed about the step-by-step process, Design thinking and at the phase3have discussed about the data preprocessing techniques and many more in the last steps and in this step we have given some problem statements to solve in the IBM COGNOS In the previous phases we have discussed about the step by step processes design thinking and at phase3 we ANALYTICS.

In this part we will continue building our project, Building the analysis by creating visualizations using IBM Cognos.

Problem:1 Continue building the analysis by creating visualizations using IBM Cognos and developing a predictive model.

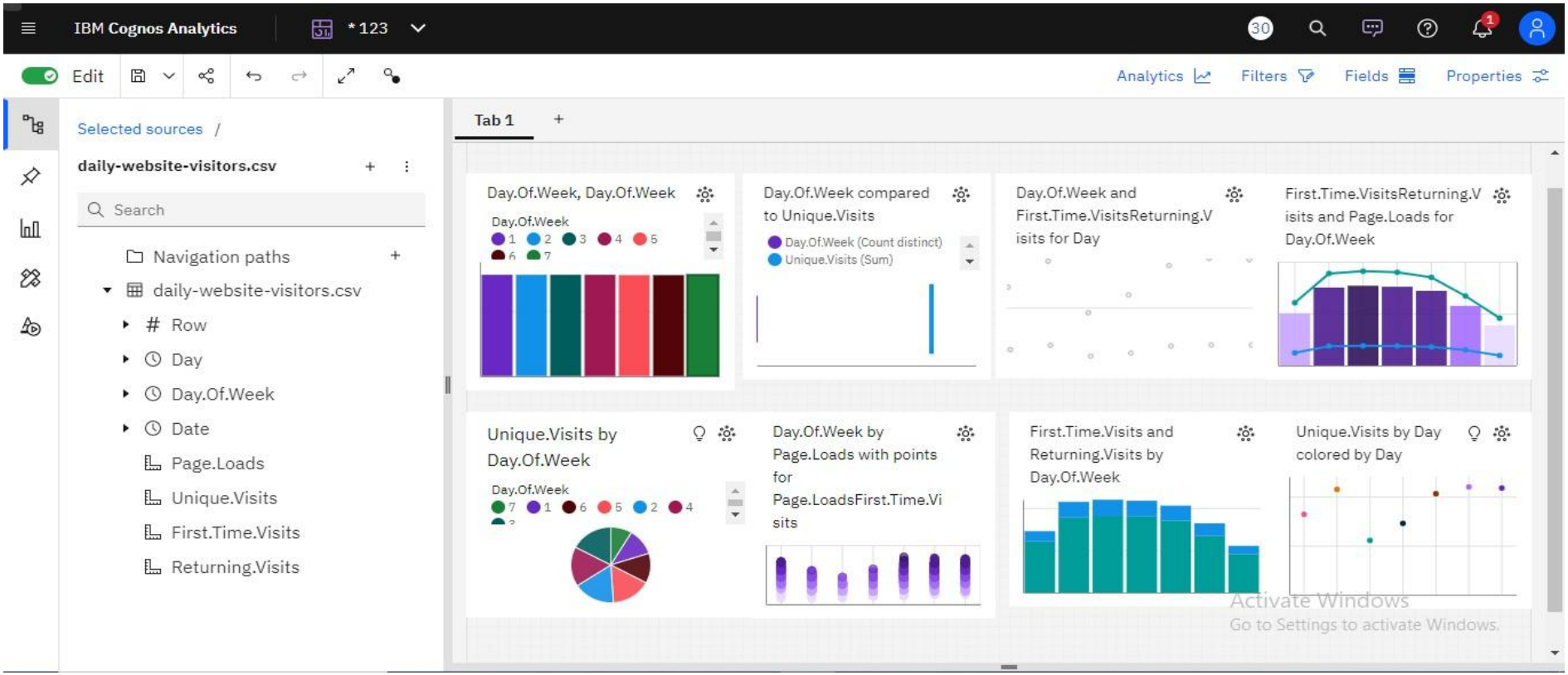
Problem:2 Create interactive dashboards and reports in IBM Cognos to visualize churn patterns, retention rates, and key factors influencing churn. Use machine learning algorithms to build a predictive model that identifies potential churners based on historical data and relevant features. Also use python libraries to perform more complex analysis on data such as time series analysis.

According to the problems we come to know that we have to find the relation between the two variables so for that we need to visualize the relation between the two variables using the IBM Cognos .

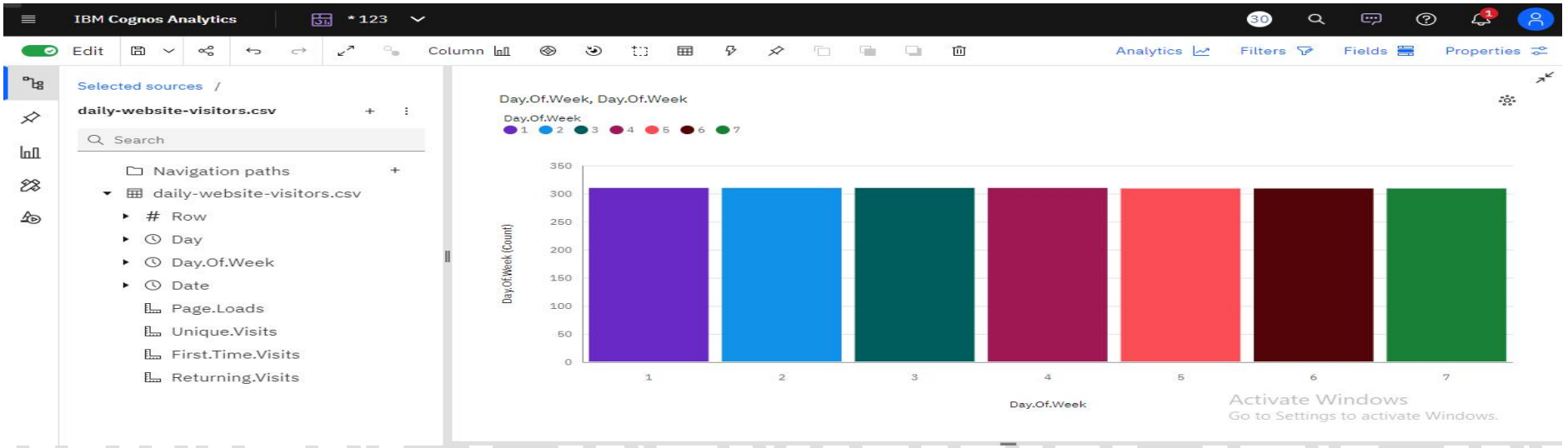
COLLLEGE CODE:4212
REG NO-421221243030

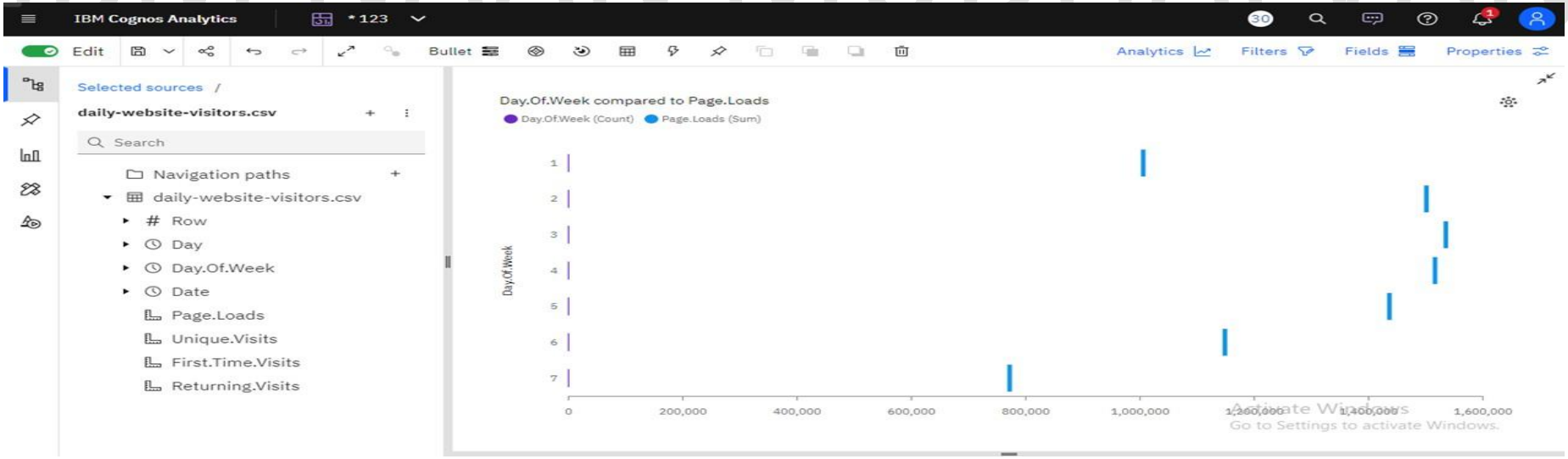
The visual insights that we have created are shown in the upcoming papers.so here we have prepared a necessary visualization and also the dashboard using the IBM Cognos.

Note: The Narrative insights(i.e. The explanation of the visualization) can be at the right side of the picture. HOW TO USE IBM COGNOS: for this we need to login in the IBM Cognos the IBM Cognos is free for one month

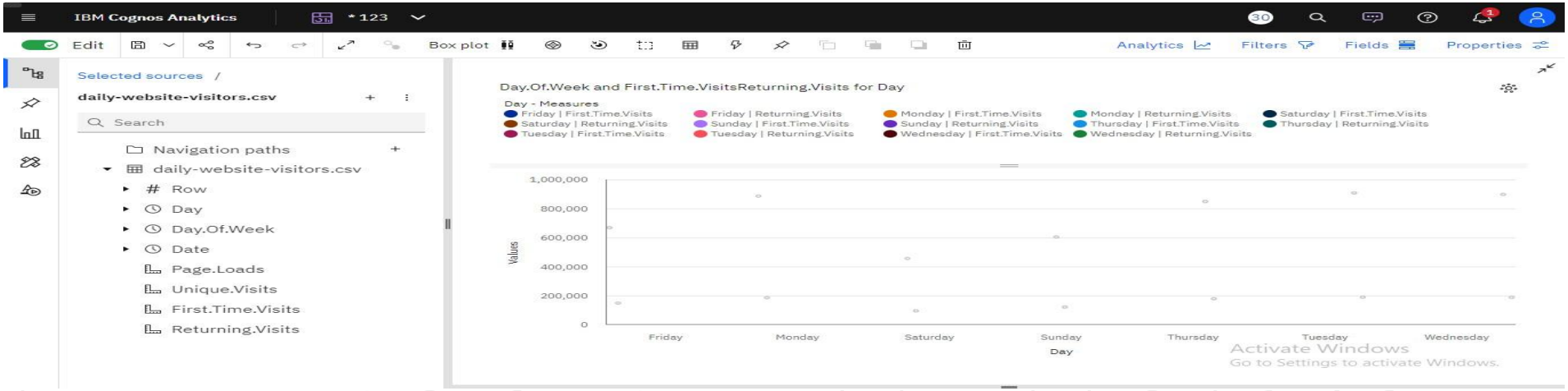


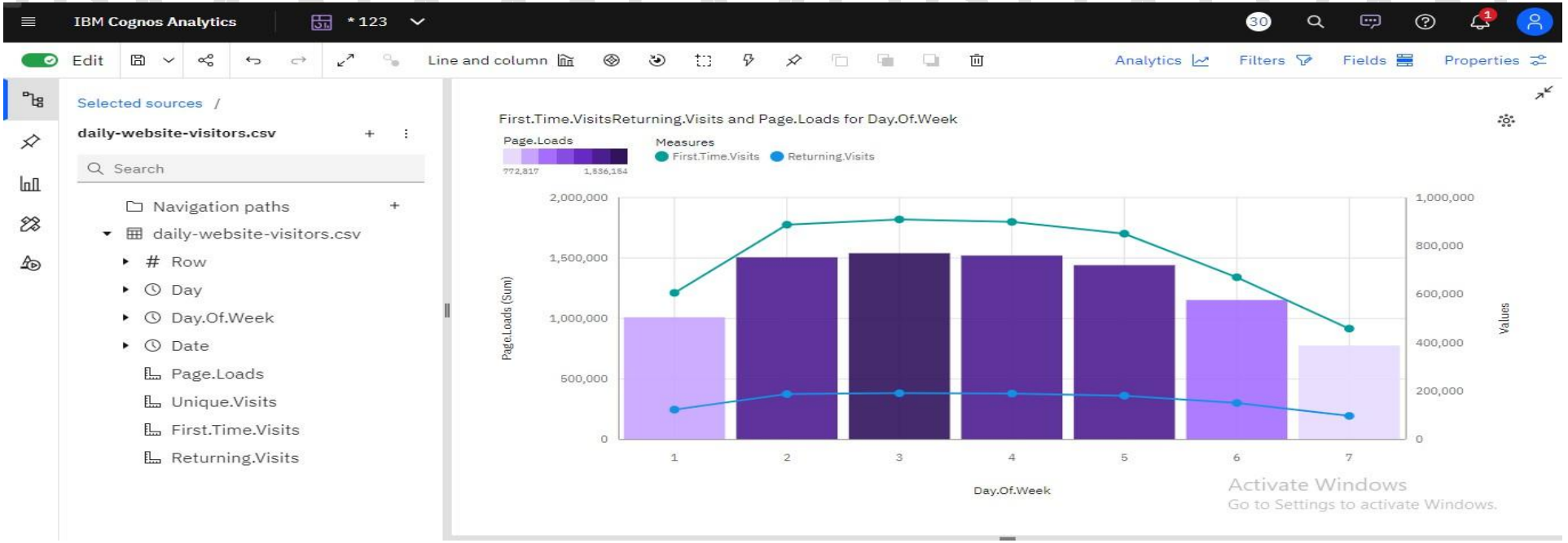
COLLLEGE CODE:4212
REG NO-421221243030



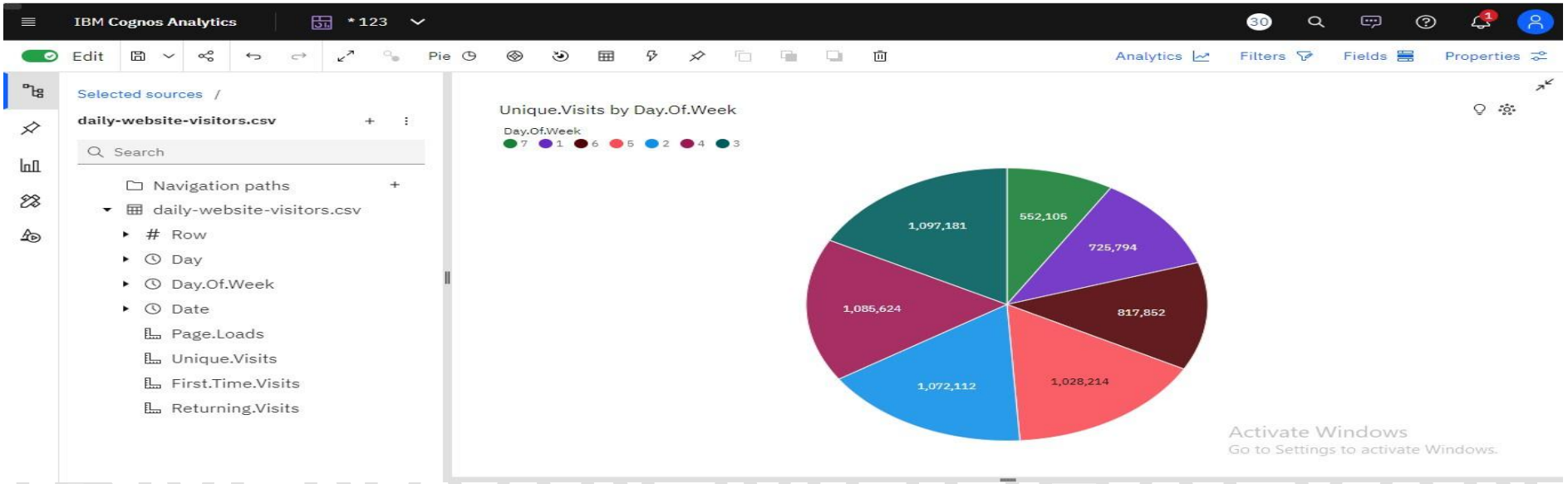


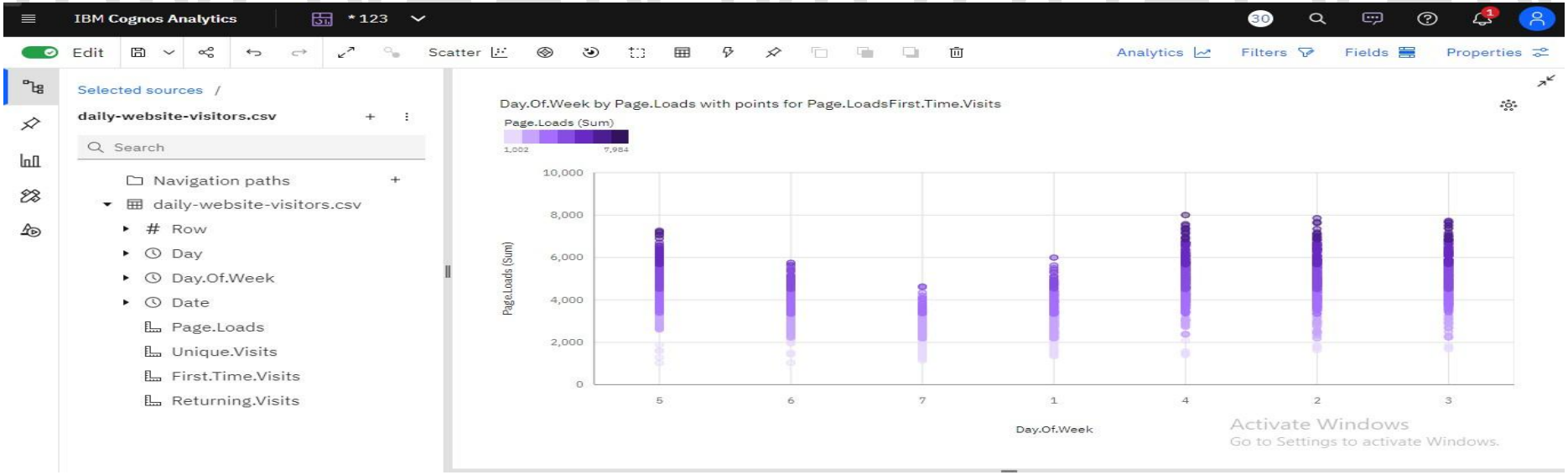
COLLLEGE CODE:4212
REG NO-421221243030

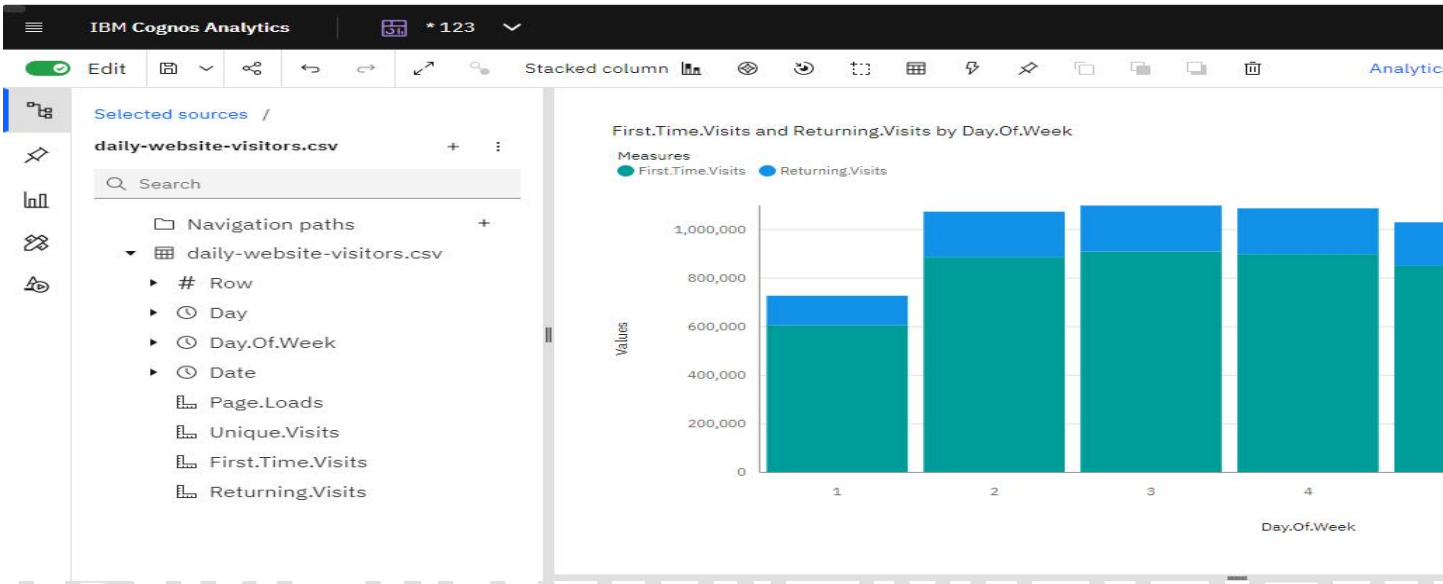


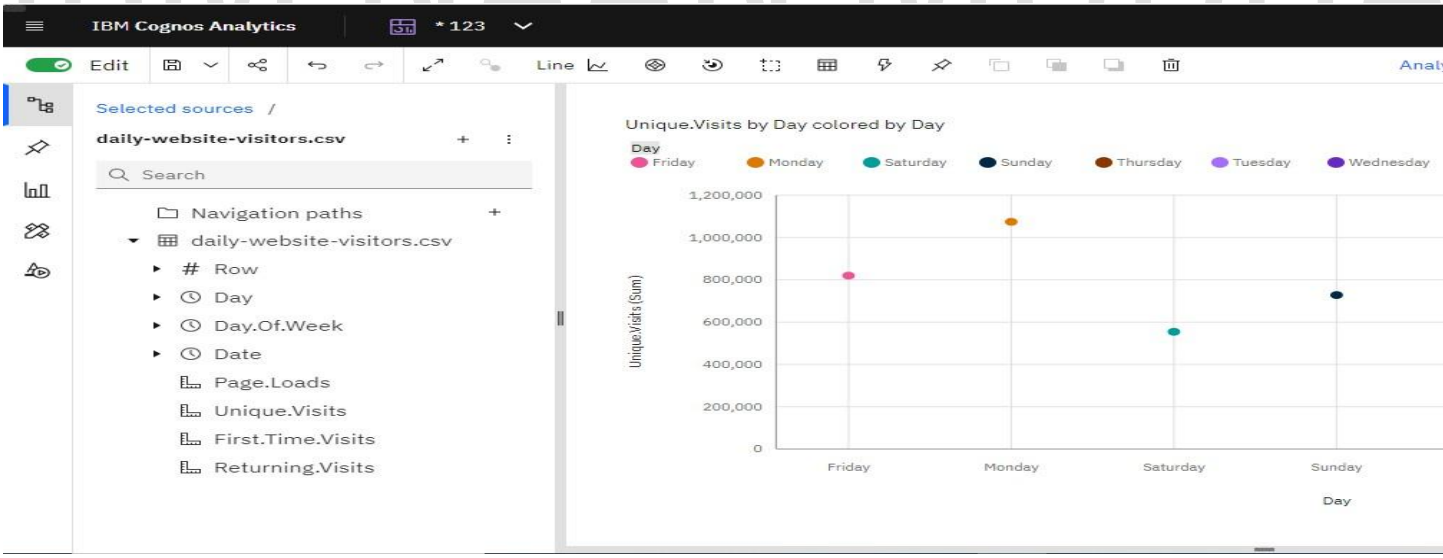


COLLLEGE CODE:4212
REG NO-421221243030

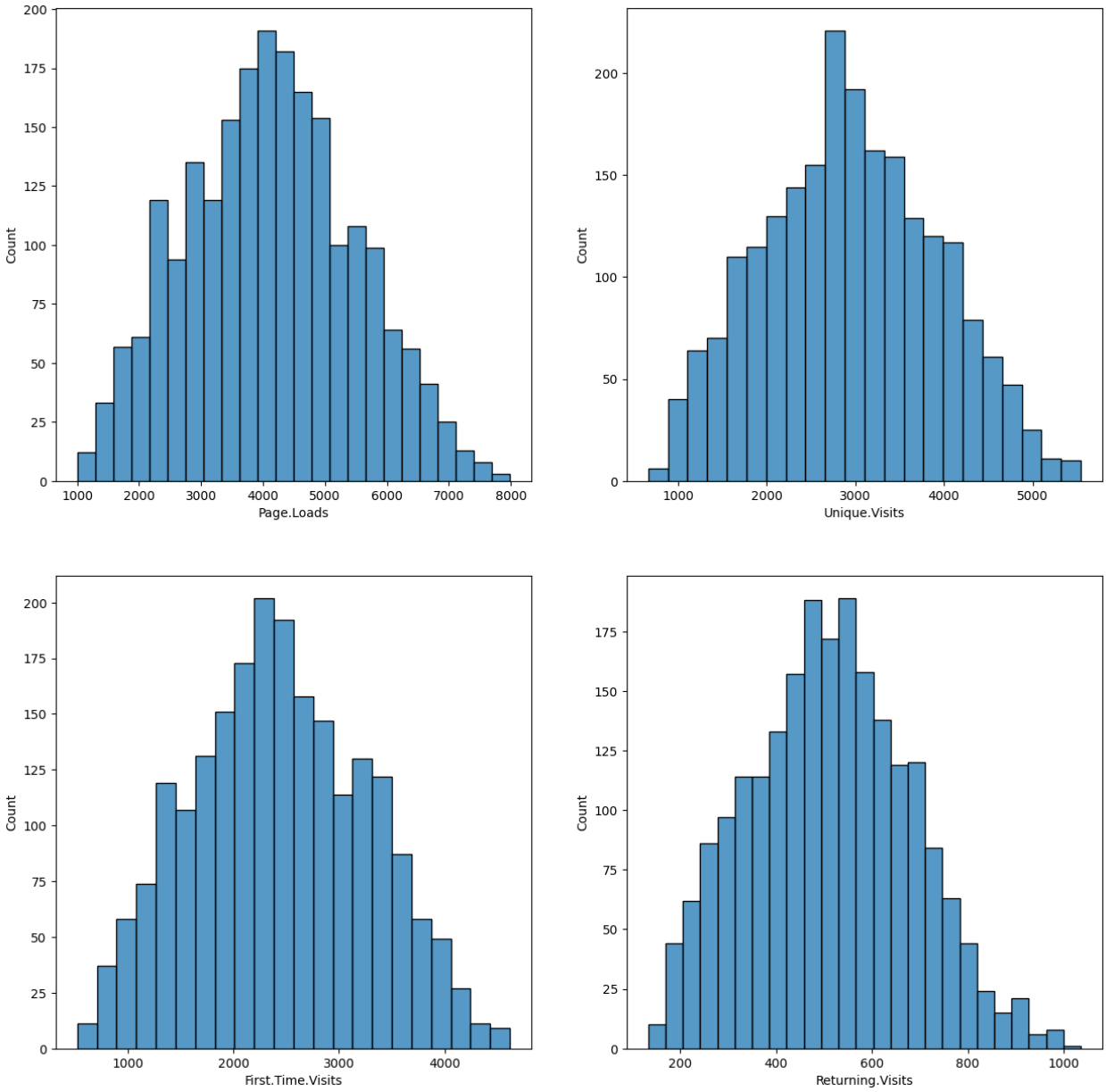





IBM Cognos Analytics
* 123
Edit
Stacked column
Analytic
Selected sources /
daily-website-visitors.csv
Search
Navigation paths
daily-website-visitors.csv
Row
Day
Day.Of.Week
Date
Page.Loads
Unique.Visits
First.Time.Visits
Returning.Visits
First.Time.Visits and Returning.Visits by Day.Of.Week
Measures
First.Time.Visits
Returning.Visits
Values
1,000,000
800,000
600,000
400,000
200,000
0
1
2
3
4
Day.Of.Week


IBM Cognos Analytics
* 123
Edit
Line
Anal
Selected sources /
daily-website-visitors.csv
Search
Navigation paths
daily-website-visitors.csv
Row
Day
Day.Of.Week
Date
Page.Loads
Unique.Visits
First.Time.Visits
Returning.Visits
Unique.Visits by Day colored by Day
Day
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
Unique.Visits (Sum)
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Friday
Monday
Saturday
Sunday
Day

```
import numpy as np
import pandas as pd


import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv('/content/daily-website-visitors.csv')

def remove_commas(x):
    return float(x.replace(',', ''))

data['Date'] = pd.to_datetime(data['Date'])
data['Page.Loads'] = data['Page.Loads'].apply(lambda x :
remove_commas(x))
data['Unique.Visits'] = data['Unique.Visits'].apply(lambda x :
remove_commas(x))
data['First.Time.Visits'] = data['First.Time.Visits'].apply(lambda x :
remove_commas(x))
data['Returning.Visits'] = data['Returning.Visits'].apply(lambda x :
remove_commas(x))

cols_to_plot = ['Page.Loads', 'Unique.Visits', 'First.Time.Visits',
'Returning.Visits']
plt.figure(figsize=(15, 15))
for i, col in enumerate(cols_to_plot):
    plt.subplot(2, 2, i+1)
    sns.histplot(data=data, x=col)
```



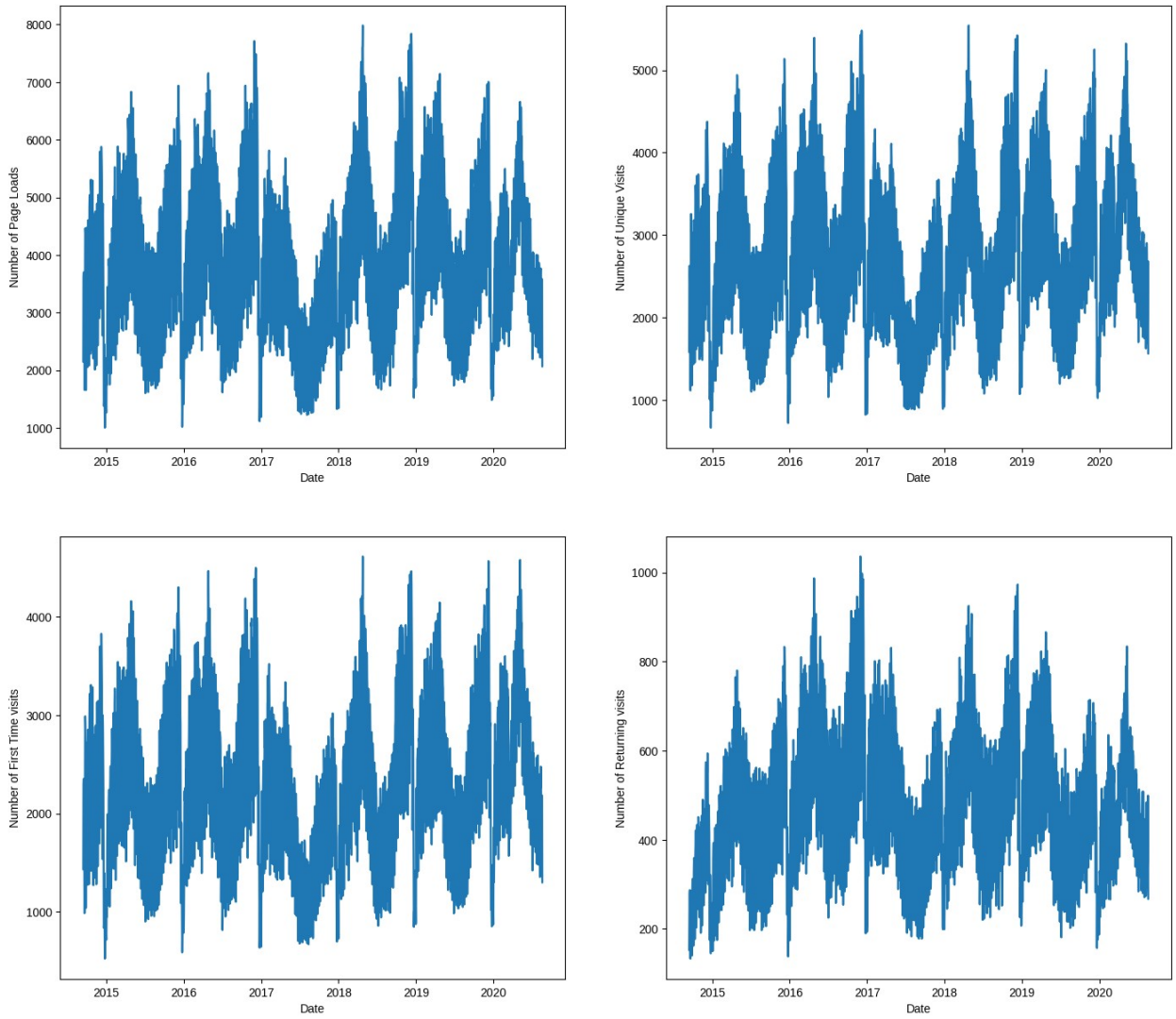

```
def check_normality(data, col):

    # Compute mean
    mean = int(np.mean(data[col]))
    median = int(np.median(data[col]))
    mode_ = int(mode(data[col])[0][0])

    print("mean", ":", mean, "median", ":", median, "mode", ":",
mode_)
    if mean == median == mode_:
        print("{} Distribution is Normal".format(col))
    elif mean > median and mean > mode_ and mode_ < median:
        print("{} Distribution is skewed towards right".format(col))
```

```
    else:
        print("{} Distribution is skewed towards left".format(col))

for col in cols_to_plot:
    check_normality(data, col)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-f077af3790a3> in <cell line: 1>()
      1 for col in cols_to_plot:
----> 2     check_normality(data, col)

<ipython-input-14-5ea3b9df35db> in check_normality(data, col)
      4     mean = int(np.mean(data[col]))
      5     median = int(np.median(data[col]))
----> 6     mode_ = int(mode(data[col])[0][0])
      7
      8     print("mean", ":", mean, "median", ":", median, "mode", ":", mode_)

NameError: name 'mode' is not defined
```

```
figure, ax = plt.subplots(2, 2, figsize=(17, 15))
plt.style.use('seaborn')

ax1 = ax[0]
ax2 = ax[1]

# Plot the Number of Page Loads with time
ax1[0].plot(data['Date'], data['Page.Loads'])
ax1[0].set_xlabel("Date")
ax1[0].set_ylabel("Number of Page Loads")

# Plot the Number of Unique Visits with time
ax1[1].plot(data['Date'], data['Unique.Visits'])
ax1[1].set_xlabel("Date")
ax1[1].set_ylabel("Number of Unique Visits")
# Plot the Number of First Time visits with time
ax2[0].plot(data['Date'], data['First.Time.Visits'])
ax2[0].set_xlabel("Date")
ax2[0].set_ylabel("Number of First Time visits")

# Plot the Number of Returning visits with time
ax2[1].plot(data['Date'], data['Returning.Visits'])
ax2[1].set_xlabel("Date")
ax2[1].set_ylabel("Number of Returning visits")

figure.show()
```

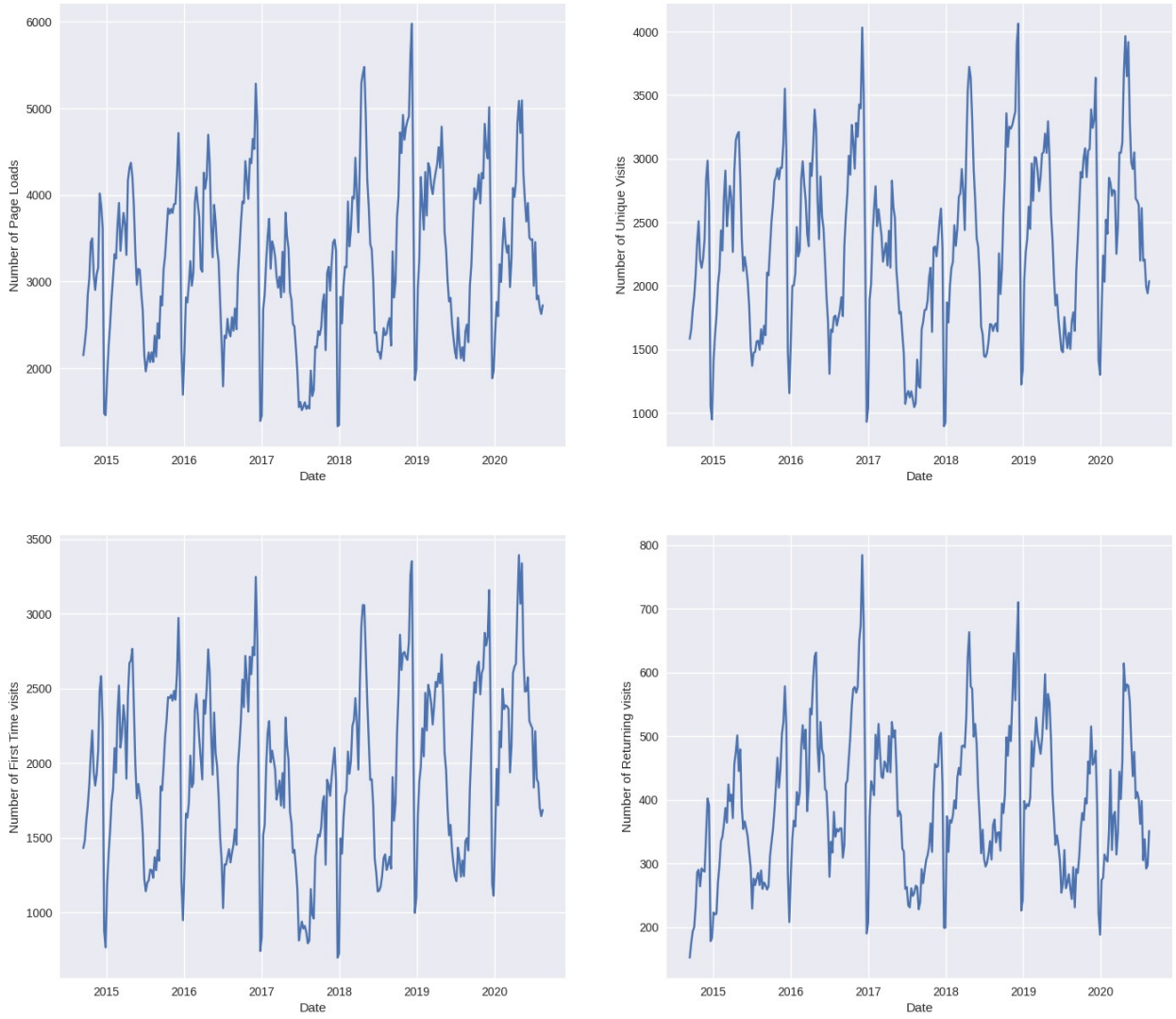
```
<ipython-input-17-45ff7e18345e>:2: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn')
```




```
# group the data by day and draw insights
day_grouped_data = data.groupby('Day')

def day_wise_EDA(day):
    sun_data = day_grouped_data.get_group(day)
    figure, ax = plt.subplots(2, 2, figsize=(17, 15))
    plt.style.use('seaborn')
    ax1 = ax[0]
    ax2 = ax[1]
```

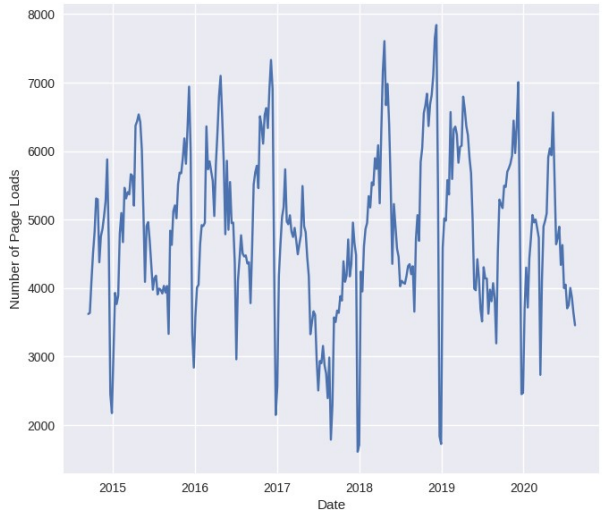
```
    # Plot the Number of Page Loads with time

print("============================================================
===={}
ANALYSIS=====================================================".format
(day.upper()))
    ax1[0].plot(sun_data['Date'], sun_data['Page.Loads'])
    ax1[0].set_xlabel("Date")
    ax1[0].set_ylabel("Number of Page Loads")
    ax1[1].plot(sun_data['Date'], sun_data['Unique.Visits'])
    ax1[1].set_xlabel("Date")
    ax1[1].set_ylabel("Number of Unique Visits")

    # Plot the Number of First Time visits with time
    ax2[0].plot(sun_data['Date'], sun_data['First.Time.Visits'])
    ax2[0].set_xlabel("Date")
    ax2[0].set_ylabel("Number of First Time visits")

    # Plot the Number of Returning visits with time
    ax2[1].plot(sun_data['Date'], sun_data['Returning.Visits'])
    ax2[1].set_xlabel("Date")
    ax2[1].set_ylabel("Number of Returning visits")

    figure.show()

day_wise_EDA('Sunday')
```

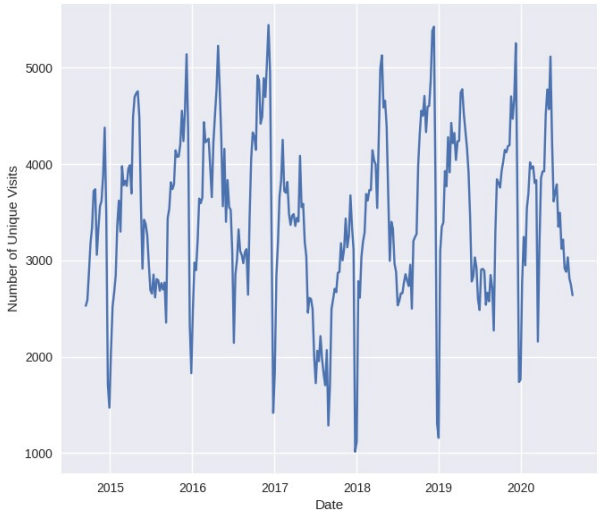
```
<ipython-input-19-0925a12011a4>:4: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they
no longer correspond to the styles shipped by seaborn. However, they
will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn API instead.
  plt.style.use('seaborn')

============================================================SUND
AY ANALYSIS=============================================
```

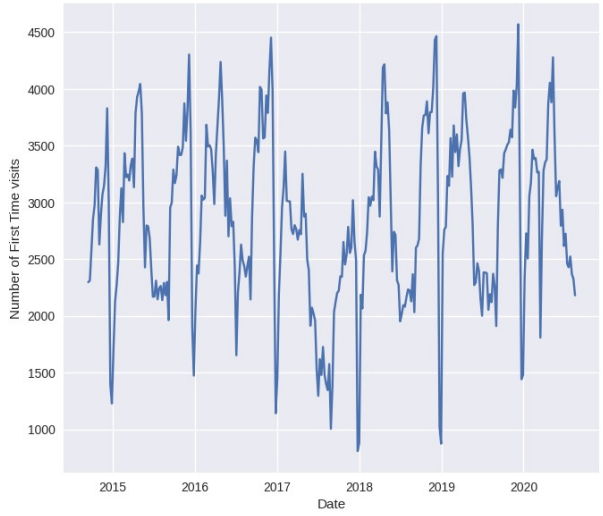


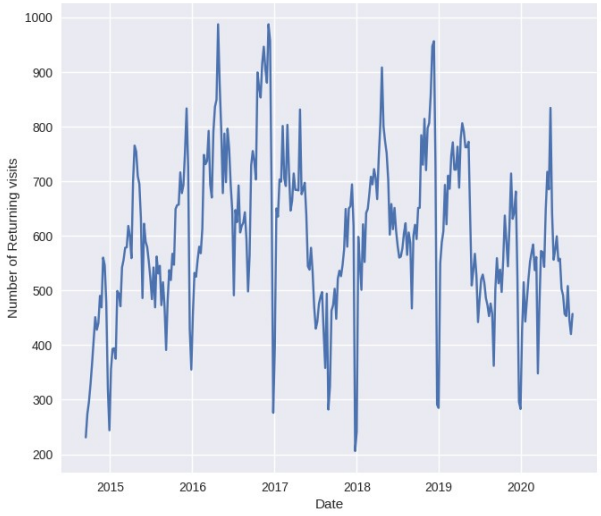

```
day_wise_EDA('Monday')
```

```
<ipython-input-19-0925a12011a4>:4: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
  plt.style.use('seaborn')

==========================================================================MONDAY ANALYSIS==========================================================
```



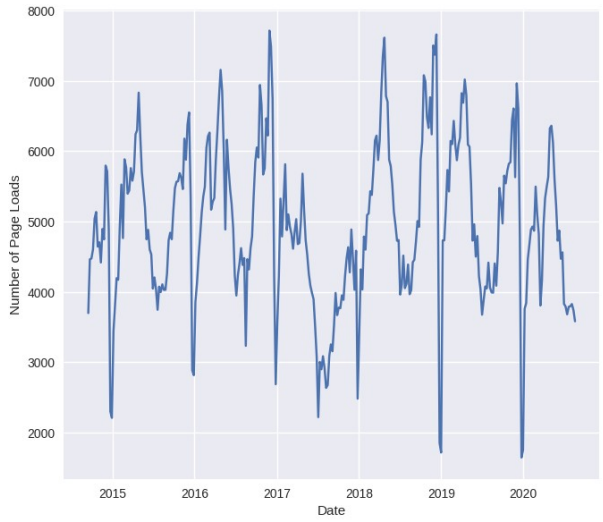


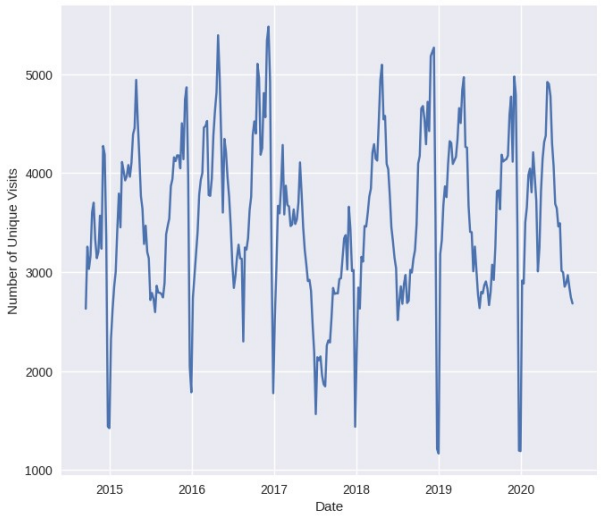


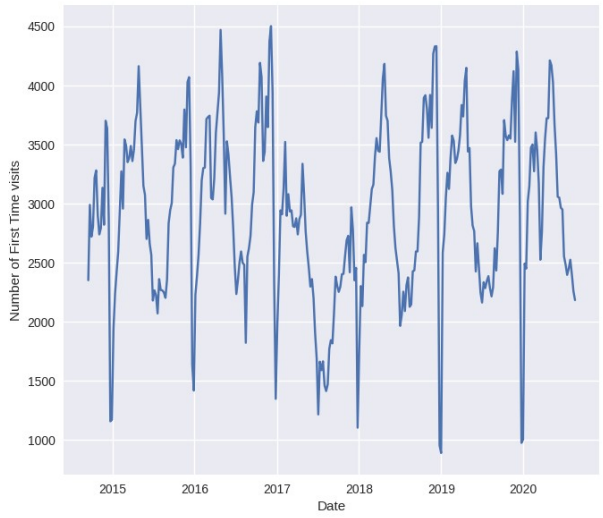


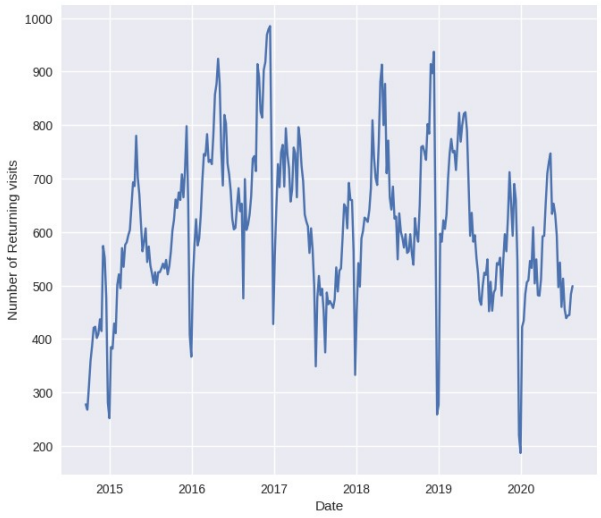

```
day_wise_EDA('Tuesday')
```

```
<ipython-input-19-0925a12011a4>:4: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they
no longer correspond to the styles shipped by seaborn. However, they
will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn API instead.
  plt.style.use('seaborn')

============================================================TUES
DAY ANALYSIS============================================================
```



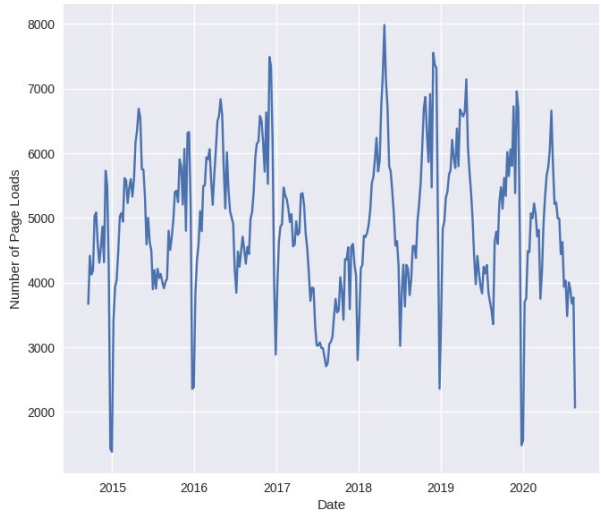


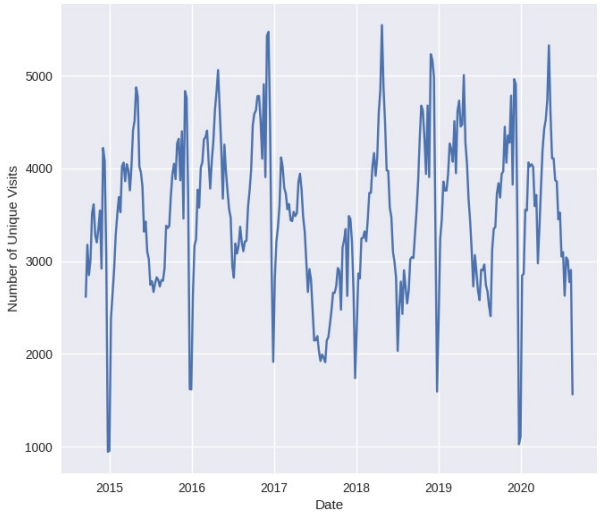


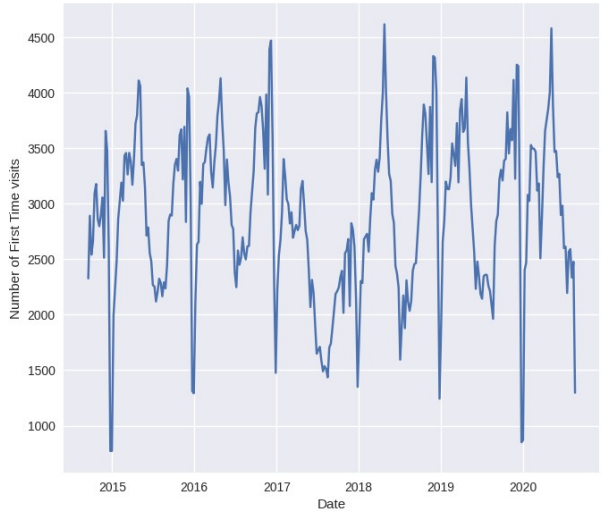


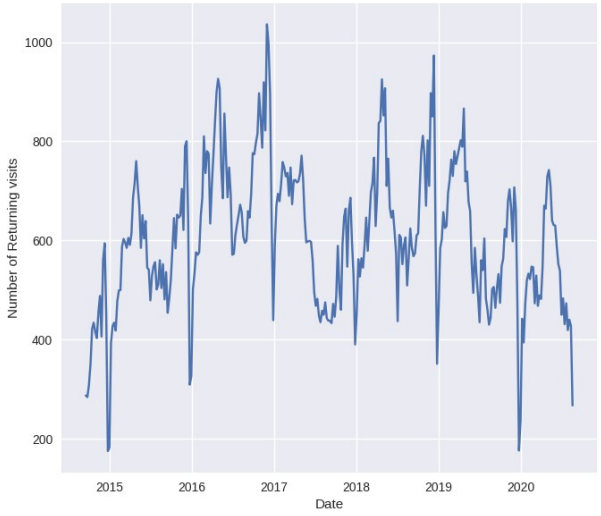

```
day_wise_EDA('Wednesday')
```

```
<ipython-input-19-0925a12011a4>:4: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they
no longer correspond to the styles shipped by seaborn. However, they
will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn API instead.
  plt.style.use('seaborn')

==========================================================================WEDN
ESDAY ANALYSIS=========================================================
```









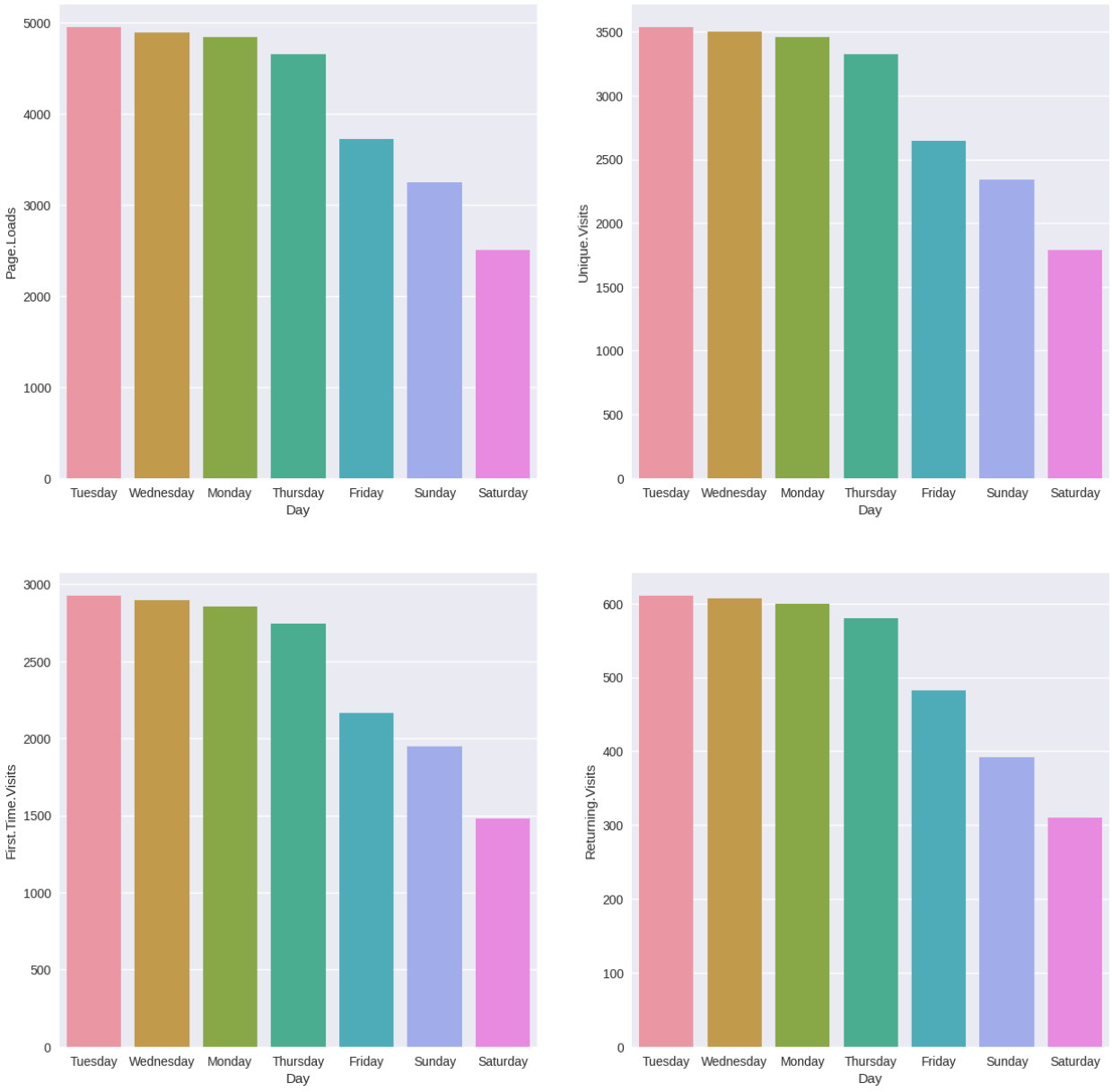

```
avg_day_data = day_grouped_data.mean().reset_index().drop('Row',
axis=1)
avg_day_data
```

```
<ipython-input-24-ed8e1f4e0f5f>:1: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.mean is deprecated. In a future
version, numeric_only will default to False. Either specify
numeric_only or select only columns which should be valid for the
function.
  avg_day_data = day_grouped_data.mean().reset_index().drop('Row',
axis=1)
```

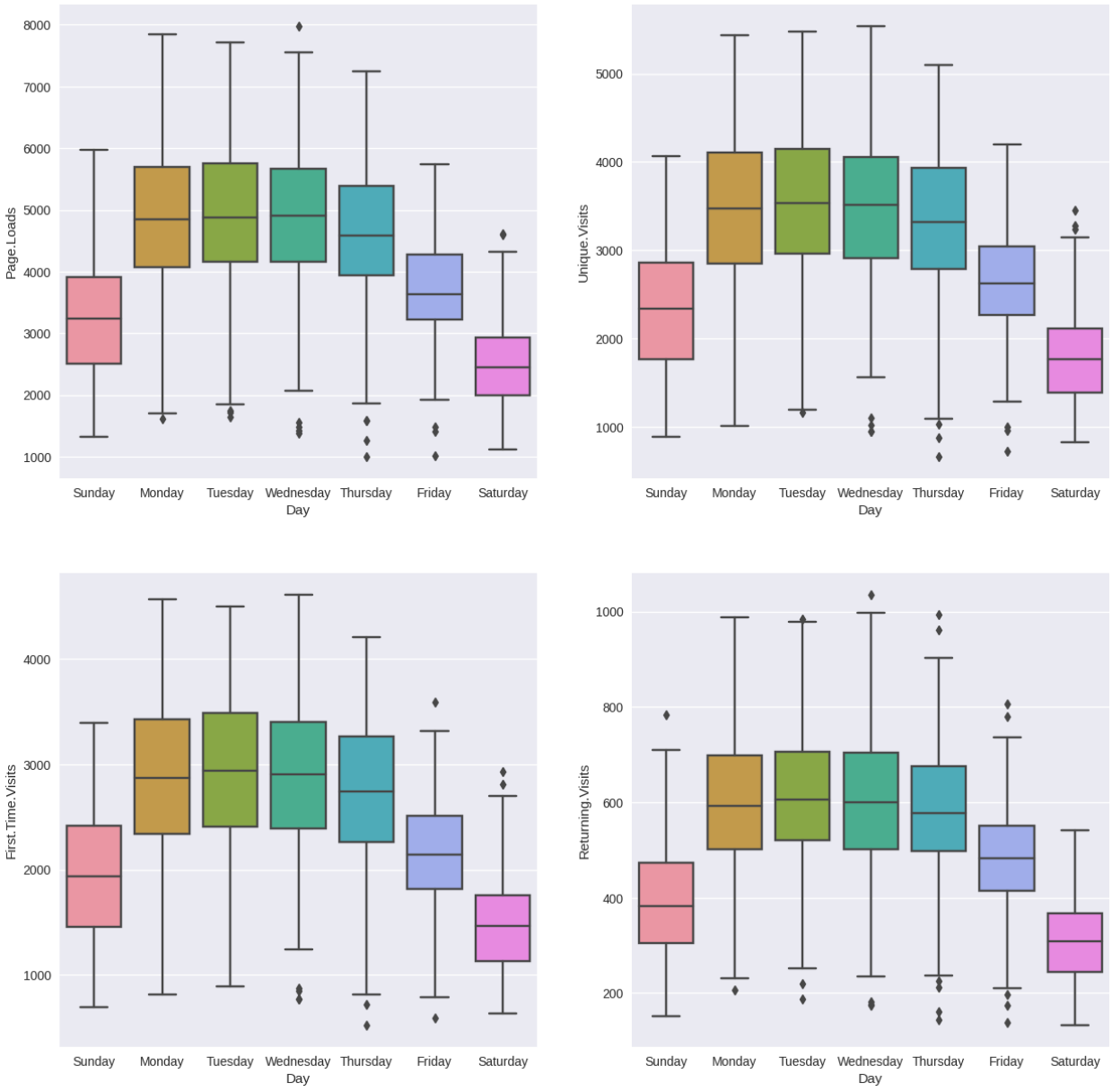
```
         Day  Day.Of.Week   Page.Loads  Unique.Visits
First.Time.Visits  \
0     Friday          6.0  3719.860841    2646.770227
2164.417476
1     Monday          2.0  4845.680645    3458.425806
```

```
2858.180645
2   Saturday          7.0  2501.025890    1786.747573
1477.181230
3     Sunday          1.0  3246.980645    2341.270968
1949.025806
4   Thursday          5.0  4651.355987    3327.553398
2747.317152
5    Tuesday          3.0  4955.335484    3539.293548
2928.232258
6  Wednesday          4.0  4893.916129    3502.012903
2895.490323

   Returning.Visits
0        482.352751
1        600.245161
2        309.566343
3        392.245161
4        580.236246
5        611.061290
6        606.522581
```

```
# Plot the Bargraph for every continuous variable across day
cols_to_plot = ['Page.Loads', 'Unique.Visits', 'First.Time.Visits',
'Returning.Visits']
plt.figure(figsize=(15, 15))
for i, col in enumerate(cols_to_plot):
    plt.subplot(2, 2, i+1)
    sns.barplot(data=avg_day_data.sort_values(by=col,
ascending=False), x='Day', y=col)
```

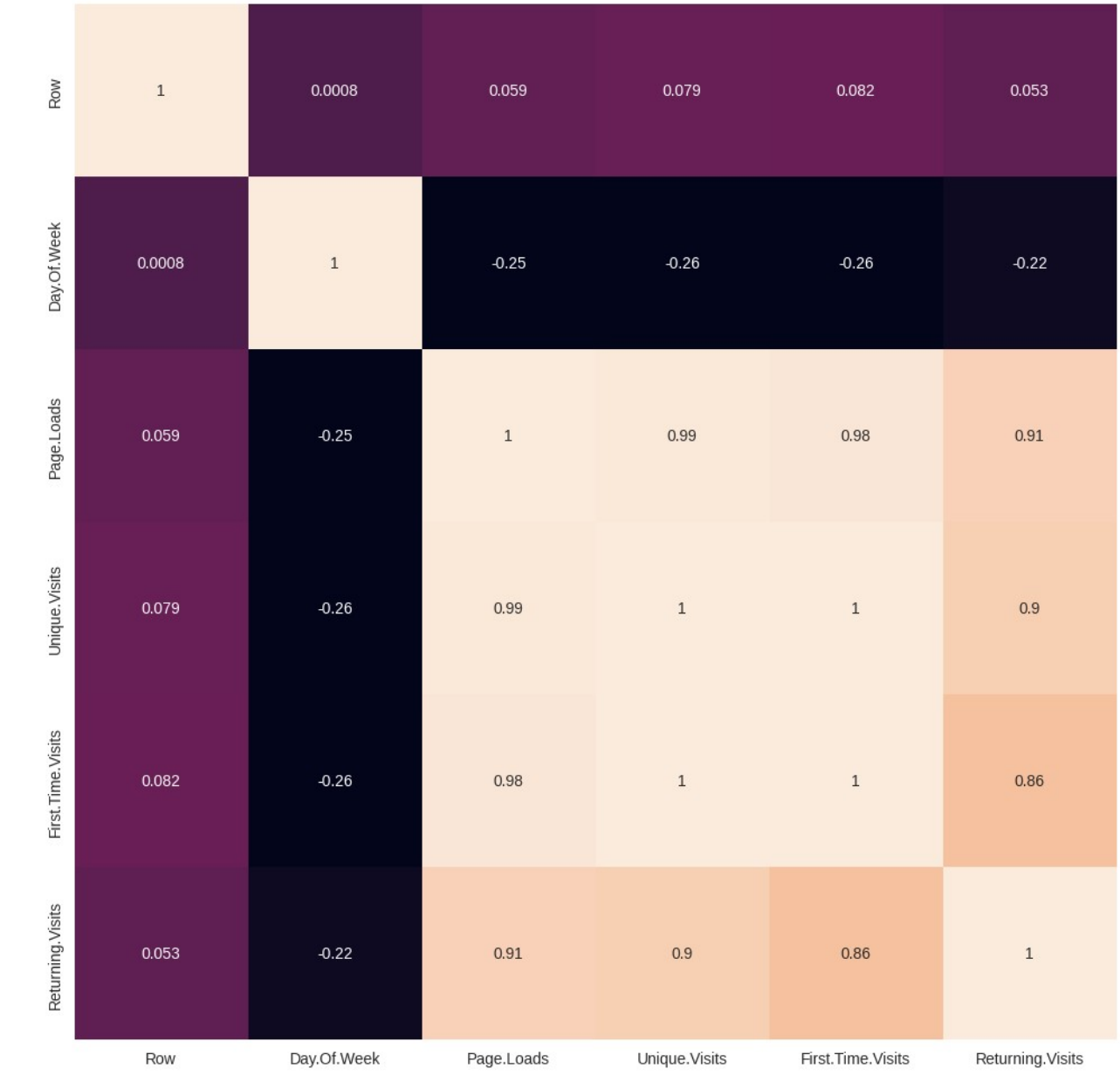



```
# Boxplots for all the continuous columns across day
cols_to_plot = ['Page.Loads', 'Unique.Visits', 'First.Time.Visits',
'Returning.Visits']
plt.figure(figsize=(15, 15))
for i, col in enumerate(cols_to_plot):
    plt.subplot(2, 2, i+1)
    sns.boxplot(data=data, x='Day', y=col)
```



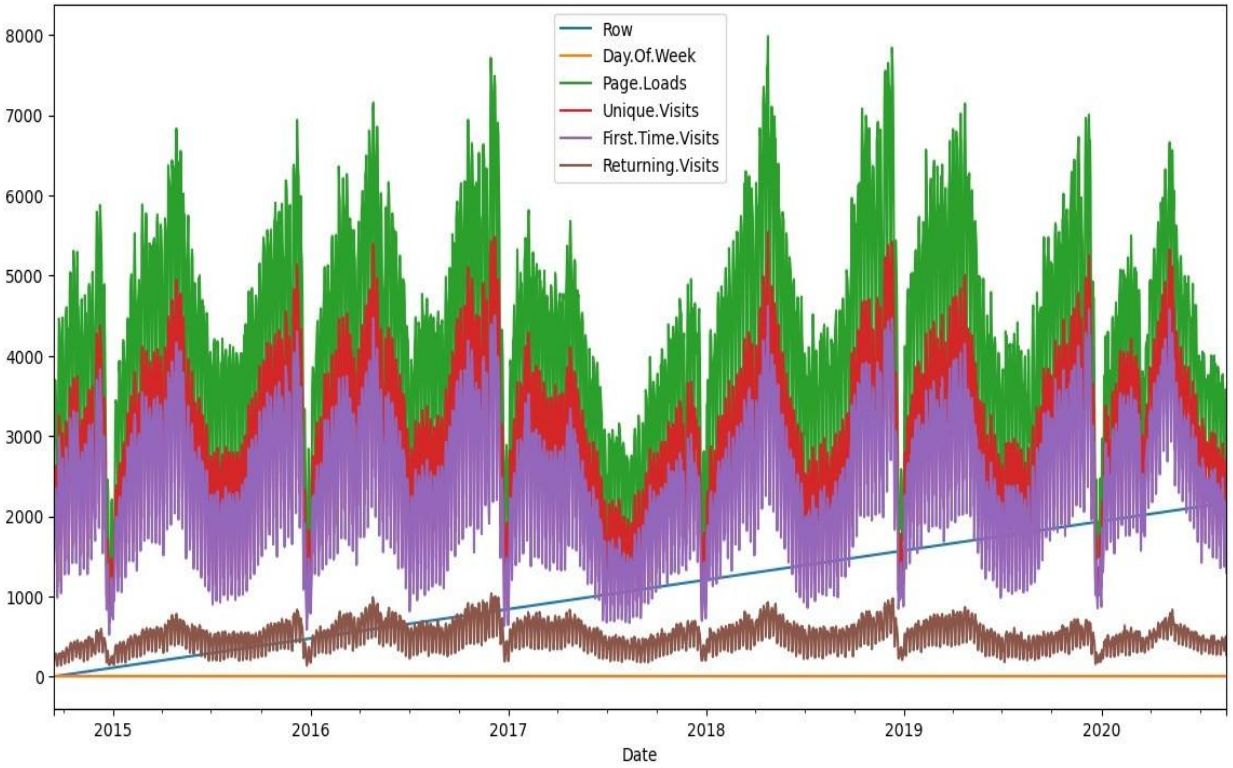

```
#Plot the correlation heatmap
corr_matrix = data.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr_matrix, annot=True, cbar=False)
plt.show()
```

```
<ipython-input-27-78d9e94c18ef>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corr_matrix = data.corr()
```

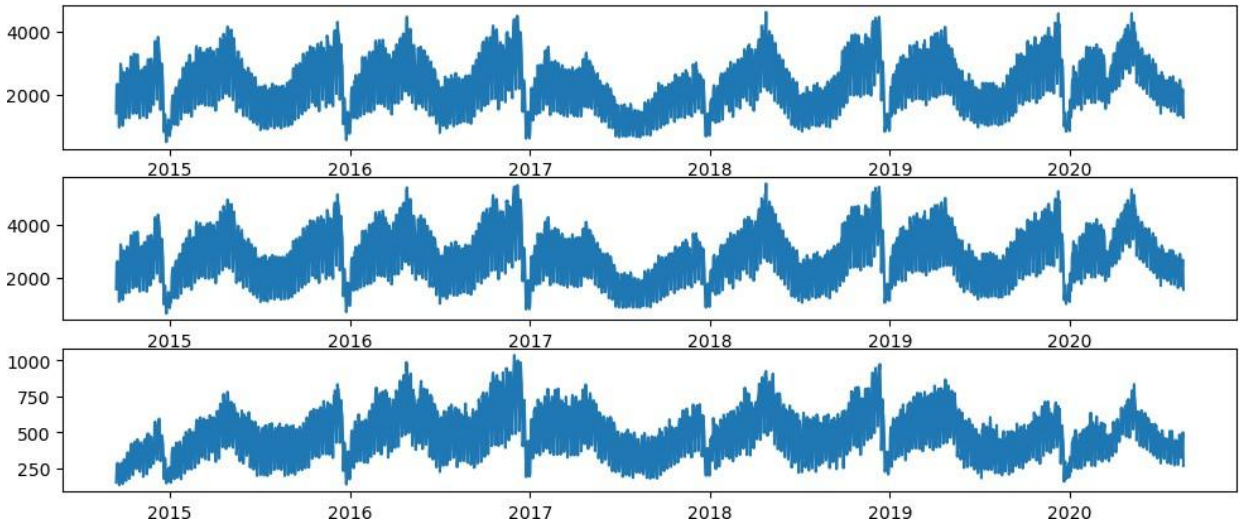

Row
Day.Of.Week
Page.Loads
Unique.Visits
First.Time.Visits
Returning.Visits
1 0.0008 0.059 0.079 0.082 0.053
0.0008 1 -0.25 -0.26 -0.26 -0.22
0.059 -0.25 1 0.99 0.98 0.91
0.079 -0.26 0.99 1 1 0.9
0.082 -0.26 0.98 1 1 0.86
0.053 -0.22 0.91 0.9 0.86 1
Row
Day.Of.Week
Page.Loads
Unique.Visits
First.Time.Visits
Returning.Visits

```
df = pd.read_csv('daily-website-visitors.csv',\
                 index_col = 'Date', thousands = ',',
parse_dates=True)

df.plot(figsize=(14,7))
```

```
<AxesSubplot:xlabel='Date'>
```




```
size=(12, 5)) axs[0].plot(df['First.Time.Visits'])
axs[1].plot(df['Unique.Visits'])
axs[2].plot(df['Returning.Visits']
) plt.show()
```




```
target_column =
df['Returning.Visits']
target_column
```

```
Date
2014-09-14    152
2014-09-15    231
2014-09-16    278
2014-09-17    287
2014-09-18    236
              ... 2020-08-15           323
2020-08-16    351
2020-08-17    457
2020-08-18    499
2020-08-19    267
Name: Returning.Visits, Length: 2167, dtype: int64
```

```
target_column.plot(figsize=(15,3))
plt.show()
```

```
TEST_DATA_PERCENTAGE = 0.1

TEST_DATA_BOUNDARY_INDEX = int((1 - TEST_DATA_PERCENTAGE) *
len(target_column))
print(f"Train data:\tReturning Visits
[:{TEST_DATA_BOUNDARY_INDEX}] ({TEST_DATA_BOUNDARY_INDEX +
1})")
print(f"Test data:\tReturning Visits
[{TEST_DATA_BOUNDARY_INDEX}:] ({len(target_column) -
TEST_DATA_BOUNDARY_INDEX})") print(f"\nLast target on
train data:
{target_column[TEST_DATA_BOUNDARY_INDEX]}")
```

```
Train data:      Returning Visits
[:1950] (1951) Test data: Returning
```

```
Visits [1950:] (217)

Last target on train data: 441

print(f"Train dataset ending

values:
{target_column[TEST_DATA_BOUNDARY_INDEX - 10:
TEST_DATA_BOUNDARY_INDEX].values}")
print(f"Test dataset starting values:
{target_column[TEST_DATA_BOUNDARY_INDEX:
TEST_DATA_BOUNDARY_INDEX +
10].values}")

Train dataset ending values: [429 423 442 464 372 253 277 515
434 394]
Test dataset starting values: [441 413 246 314 443 484 473 490
353
249]
```