Collage code:4212
421221423016
Phase-5

WEBSITE TRAFFIC ANALYSIS

DATA ANALYTICS WITH COGNOS –GROUP 2

In the previous phases we defined the problem statements, solving of those problems, data manipulation and the visualizing techniques using the python in this phase we are going to discuss about some problem statements that are given in phase5 they are:

Phase 5: Project Documentation & Submission

In this part you will document your project and prepare it for submission.

Document the customer churn prediction project and prepare it for submission.

Documentation

● Outline the project's objective, design thinking process, and development

phases.

● Describe the analysis objectives, data collection process, data visualization

using IBM Cognos, and predictive modeling.

Submission

● Share the GitHub repository link containing the project's code and files.

Collage code:4212
421221423016
Phase-5

● Provide instructions on how to replicate the analysis and generate visualizations using IBM Cognos and build the predictive model using Python.

● Include example outputs of the visualizations and model evaluation.

Problem Statement :

Develop a comprehensive website traffic analysis system that gathers, processes, and presents data to provide insights into user behaviour , source of traffic, popular content, and conversion rates, with the goal of optimizing the website's performance and user experience.

Objectives :

The objective of website traffic analysis is to gather and analyze data related to a website's performance and user interactions. It aims to assess metrics like user behavior, traffic sources, demographics, and conversion rates. By understanding these insights, businesses can optimize their website, improve user experience, enhance content strategies, and make informed decisions to increase engagement, conversions, and overall effectiveness. Traffic analysis enables businesses to adapt to changing user preferences and market trends, ultimately driving growth and achieving their online goals.

Collage code:4212
421221423016
Phase-5

DESIGN THINGING:

Design thinking is a user-centered approach to problem-solving and innovation. When applying design thinking to website traffic analysis, consider the following steps:

1. **Empathize:**

- Understand the needs and goals of website owners or stakeholders.

- Gather insights from users to understand their behaviors and expectations.

- Identify pain points in the current traffic analysis process.

2. **Define:**

- Clearly define the problem you want to address with traffic analysis.

- Create a user persona for website owners, analysts, and other stakeholders.

- Develop a specific problem statement, e.g., "Improve website traffic analysis to increase user engagement."

3. **Ideate:**

- Brainstorm creative solutions for traffic analysis.

- Explore various data sources and tools for collecting and analyzing website traffic.

- Encourage collaboration and generate diverse ideas.

4. **Prototype:**

- Create a prototype of the proposed traffic analysis system.

- Develop wireframes or mockups of the user interface for data visualization.

- Use rapid prototyping to test and refine the concept.

5. **Test:**

- Collect feedback from website owners and analysts on the prototype.

- Conduct usability testing to ensure the system is intuitive and efficient.

- Iterate and make improvements based on user feedback.

6. **Implement:**

- Develop the final website traffic analysis tool or platform.

- Integrate data sources such as Google Analytics, server logs, and user surveys.

- Ensure data security and compliance with privacy regulations.

Collage code:4212
421221423016
Phase-5

7. **Evaluate:**

- Monitor the performance of the traffic analysis system.
- Analyze key metrics and KPIs to measure its effectiveness.
- Continuously gather user feedback and make updates as needed.

8. Iterate:

- Use feedback and data-driven insights to make ongoing improvements.
- Adapt to changes in user behavior and technology trends.
- Consider scalability and future enhancements.

Throughout the design thinking process, keep the focus on improving the website's performance, user experience, and achieving the goals set in the definition phase. Engage stakeholders and end-users in every step to ensure the final traffic analysis solution meets their needs and expectations.

INTRODUCTION:

Website traffic analysis is a crucial aspect of understanding how users interact with the website. It involves the collection and interpretation of data related to the visitors, their behavior, and the performance of

your website. In this process, various metrics, tools, and techniques are used to gain insights into user demographics, page views, bounce rates, conversion rates, and more.

In this phase we are going to explain about design and ideology that are going to present to solve this problem.

DATASET LINK:

https://www.kaggle.com/datasets/bobnau/daily-website-visitors

To this problem this dataset is given to us so by using this dataset we are going to solve our problem.

In the phase1 we have defined certain steps to solve the problem step by step now we are going to explain which methodology we are going to use to solve this problem in each step.

Clearly define the problem:

Website traffic analysis is the process of examining user interactions and

behaviors on a website to gain insights into its performance.

Data collection:

The dataset is already given for us Dataset Link:
https://www.kaggle.com/datasets/bobnau/daily-website-visitors

Preparing of the data:

Collage code:4212
421221423016
Phase-5

Preparing data for website traffic analysis involves several steps. First, collect raw data using web analytics tools. Then, clean and format the data, removing duplicates or errors. Next, organize it into meaningful categories, like page views or user demographics.

Exploratory data analysis:

This was the most important step in this project so we have to represent our data in the understandable visualization tools like **pie chart, bar graph, histogram** to represent the relation between the two attributes in the given column.

Feature extraction:

In this step we are going to explain about the features or attributes that are going to select in the dataset and we have to represent the relationship between the data visualization.

Model validation and training:

Model validation in website traffic analysis involves comparing the predictions of a traffic analysis model with actual observed data to ensure accuracy. Model training in website traffic analysis entails using historical data to teach a machine learning model to make predictions and identify patterns in website traffic.

Model evaluation:

Collage code:4212
421221423016
Phase-5

Model evaluation in website traffic analysis assesses the performance of a predictive model. It involves comparing model predictions with actual traffic data using metrics like accuracy, precision, recall, or F1-

Result representation:

In website traffic analysis, result representation involves displaying analyzed data in visual formats like charts, graphs, and tables.

Reporting and Visualization:

This visual representation helps stakeholders quickly grasp key insights, such as traffic trends, visitor demographics, and conversion rates, making it easier to make informed decisions and optimize website prediction.

TOOLS AND SOFTWARE COMMANLY USED IN THE PROCESS:

Cognos is a business intelligence tool used for reporting and data analysis, while Python is a versatile programming language. To perform website traffic analysis and generate documents in various formats, you can use Python to collect, process, and analyze web traffic data, and then use Cognos to create reports or visualizations for further analysis. Here's a high-level overview of the process:

Collage code:4212
421221423016
Phase-5

Data Collection:

Use Python libraries like Requests or Scrapy to scrape website traffic data, or employ web analytics tools to gather relevant data.

Data Processing:

Process the collected data using Python for cleaning, transformation, and aggregation. Popular libraries for data processing include Pandas and NumPy.

Analysis:

Analyze the data with Python libraries like Matplotlib, Seaborn, or Plotly for creating visualizations, and perform statistical analysis or machine learning if necessary.

Report Generation:

You can export the analysis results to a document format like PDF, Excel, or HTML using Python libraries such as ReportLab, Pandas, or Jupyter Notebooks.

Collage code:4212
421221423016
Phase-5

Cognos Integration:

If your organization uses Cognos for reporting and visualization, you can import the generated documents or data into Cognos to create more sophisticated reports and dashboards.

This combination allows you to leverage Python's data manipulation and analysis capabilities while utilizing Cognos for enterprise-level reporting and visualization.

Jupyter Notebooks:

Jupyter is often used for exploratory data analysis, model development, and documentation in Python.

DATA PREPROCESSING:

This phase involves in designing of the steps that defining in each phase of the previous documentation this involves importing necessary functions, data processing and so on in this phase we have to begin our project by loading and preprocessing the dataset.

The IBM suggests using the jupyter notebook for loading and preprocess the dataset:

Here for this project title we need to define the loading the libraries,

understand the data and visualize the missing values. For this certain inputs are defined for this project.in this phase each of the input

Collage code:4212
421221423016
Phase-5

Codes of project is given below

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 bystep 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 predictivemodel.

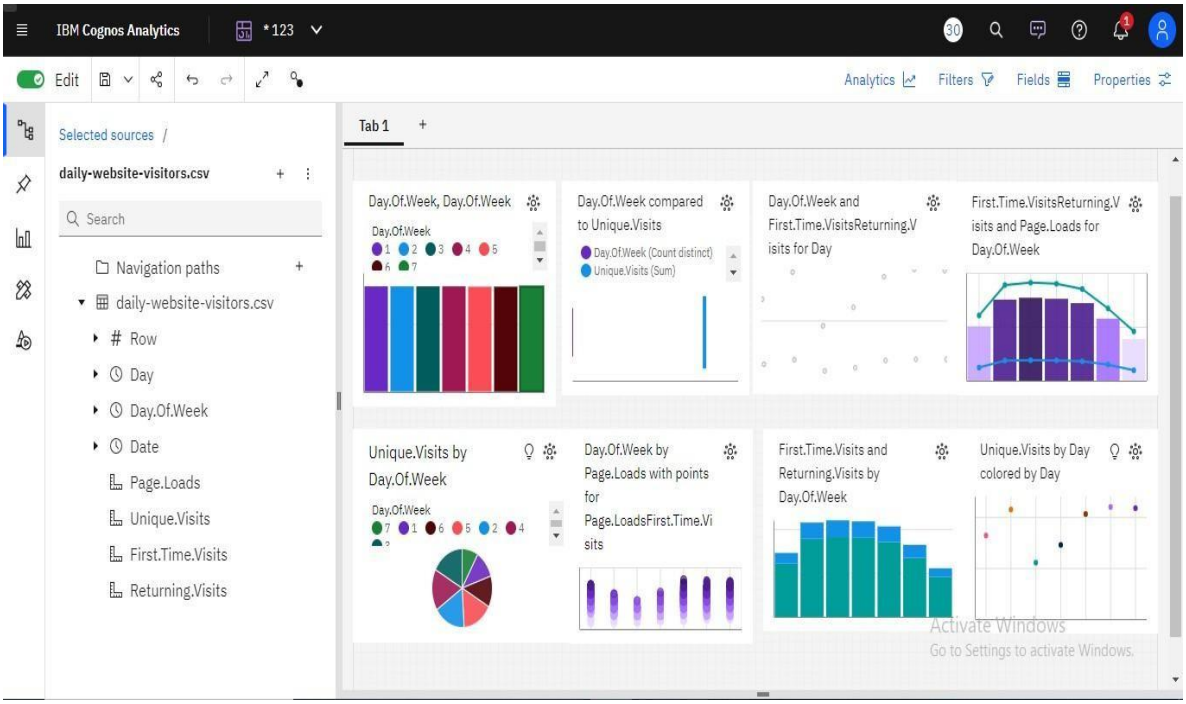
Problem:2 Create interactive dashboards and reports in IBM Cognos to visualize churn patterns, retention rates, and key factors influencing churn. Use machine learning algorithms to build a predictive model that identifies potential churners based on historical data and relevant features. Also use python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

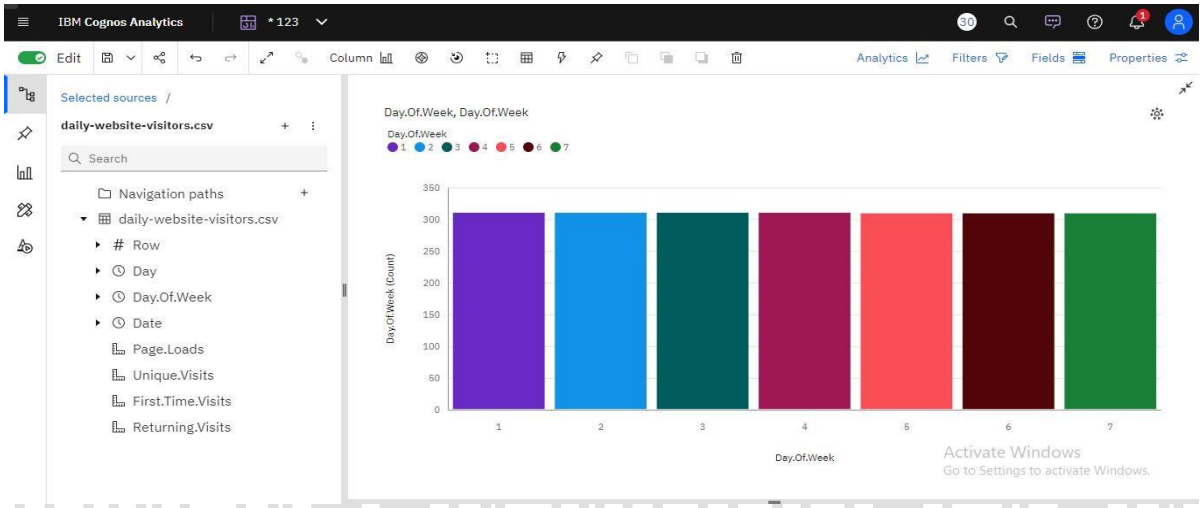
Collage code:4212
421221423016
Phase-5

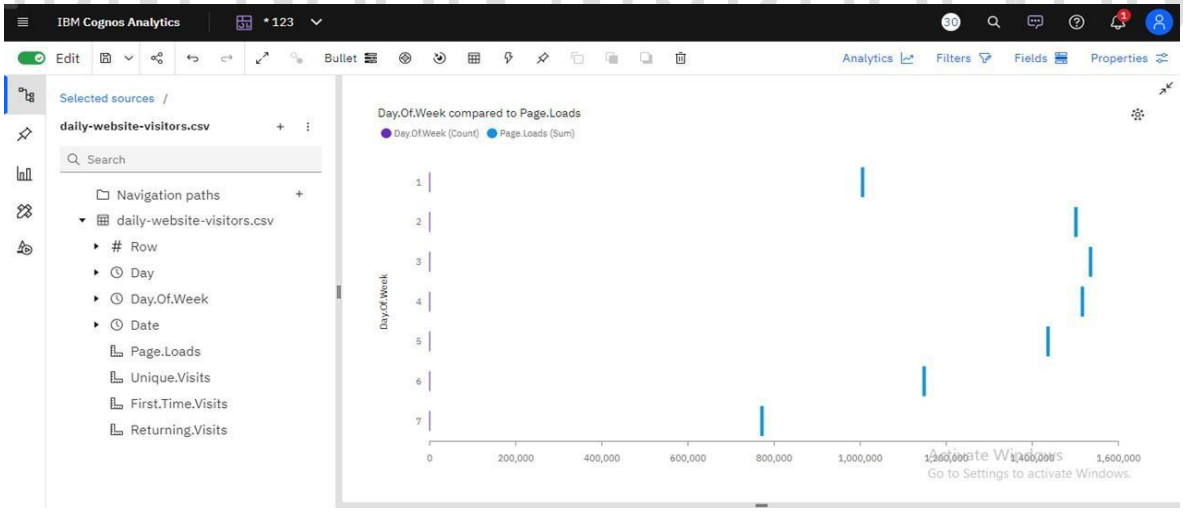
The visual insights that we have created are shown in the upcoming papers.so here we have prepared a necessaryvisualization and also the dashboard using the IBM Cognos.

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

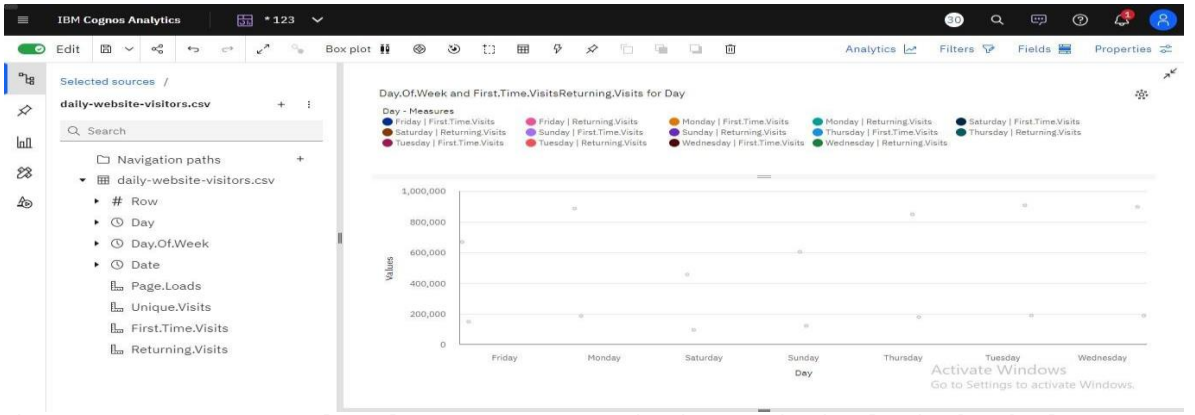


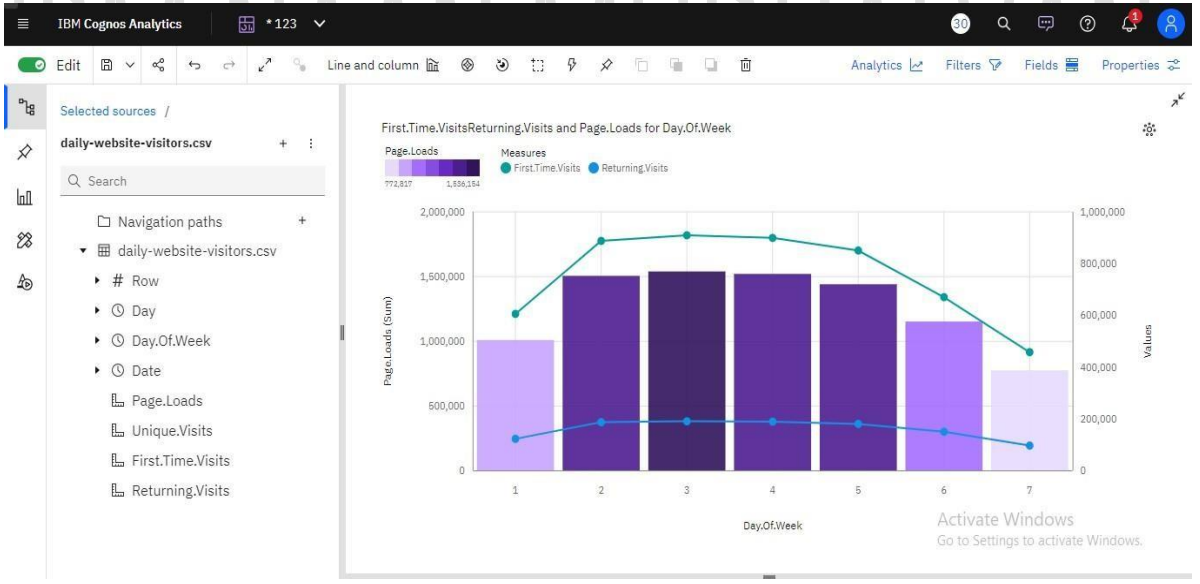
Collage code:4212
421221423016
Phase-5



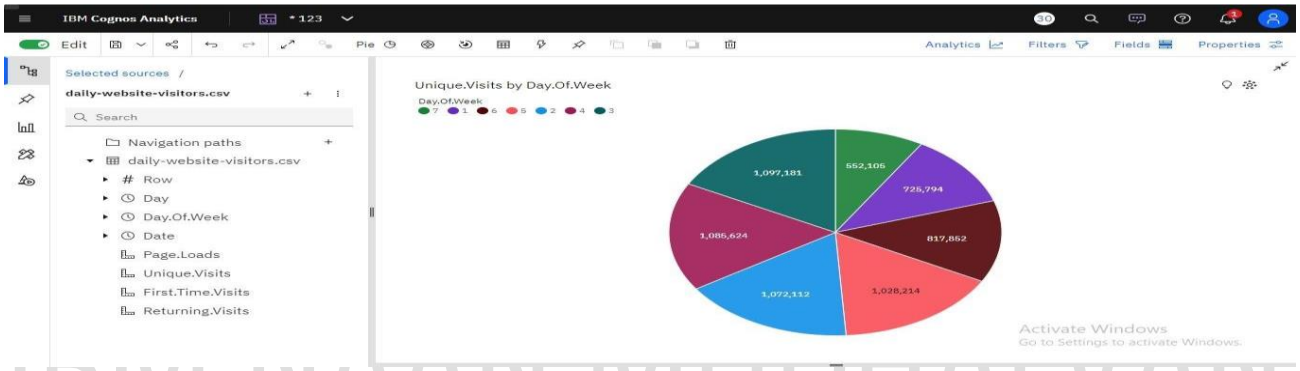


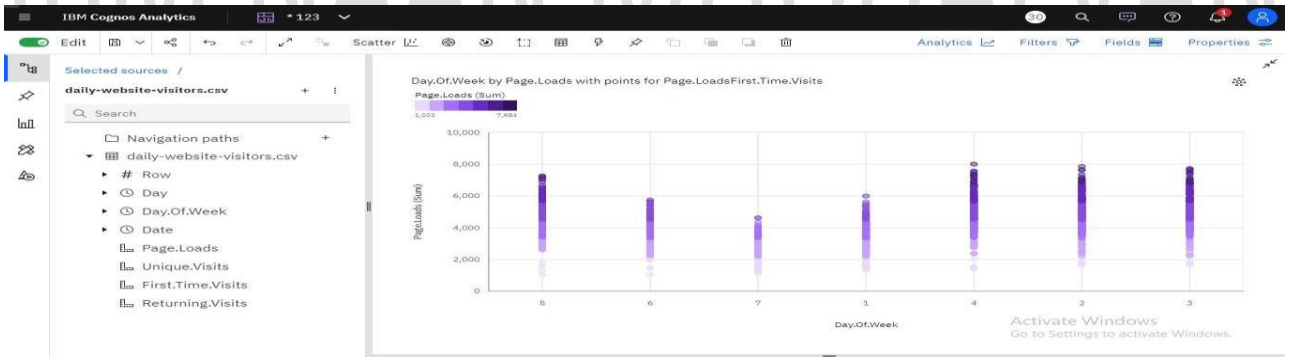
Collage code:4212
421221423016
Phase-5



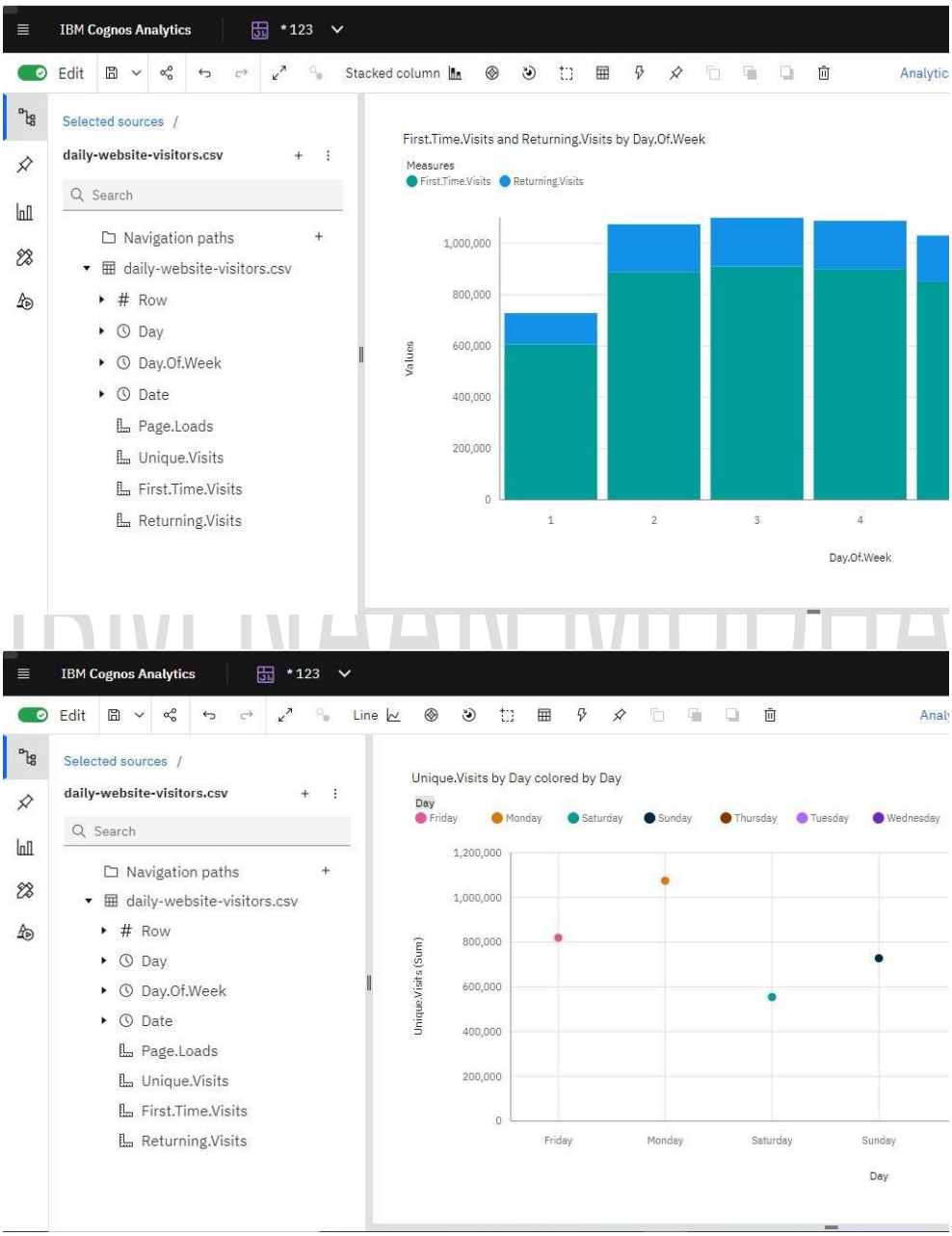


Collage code:4212
421221423016
Phase-5





Collage code:4212
421221423016
Phase-5



untitled7

October 18, 2023

[ ]:
```
PHASE 3
```

[1]:
```
import pandas as pd
import numpy as np
import missingno as msno
```

[2]:
```
df = pd.read_csv('daily-website-visitors.csv')
```

[3]:
```
df.head()
```

[3]:
```
   Row        Day  Day.Of.Week       Date Page.Loads Unique.Visits  \
0    1     Sunday            1  9/14/2014      2,146         1,582
1    2     Monday            2  9/15/2014      3,621         2,528
2    3    Tuesday            3  9/16/2014      3,698         2,630
3    4  Wednesday            4  9/17/2014      3,667         2,614
4    5   Thursday            5  9/18/2014      3,316         2,366

  First.Time.Visits Returning.Visits
0             1,430               152
1             2,297               231
2             2,352               278
3             2,327               287
4             2,130               236
```

[4]:
```
df.tail()
```

[4]:
```
        Row        Day  Day.Of.Week       Date Page.Loads Unique.Visits  \
2162  2163   Saturday            7  8/15/2020      2,221         1,696
2163  2164     Sunday            1  8/16/2020      2,724         2,037
2164  2165     Monday            2  8/17/2020      3,456         2,638
2165  2166    Tuesday            3  8/18/2020      3,581         2,683
2166  2167  Wednesday            4  8/19/2020      2,064         1,564

     First.Time.Visits Returning.Visits
2162             1,373               323
2163             1,686               351
2164             2,181               457
```

```
2165            2,184             499
2166            1,297             267
```

[5]:
```
df.shape
```

[5]:
```
(2167, 8)
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Row                2167 non-null   int64 
 1   Day                2167 non-null   object
 2   Day.Of.Week        2167 non-null   int64 
 3   Date               2167 non-null   object
 4   Page.Loads         2167 non-null   object
 5   Unique.Visits      2167 non-null   object
 6   First.Time.Visits  2167 non-null   object
 7   Returning.Visits   2167 non-null   object
dtypes: int64(2), object(6)
memory usage: 135.6+ KB
```

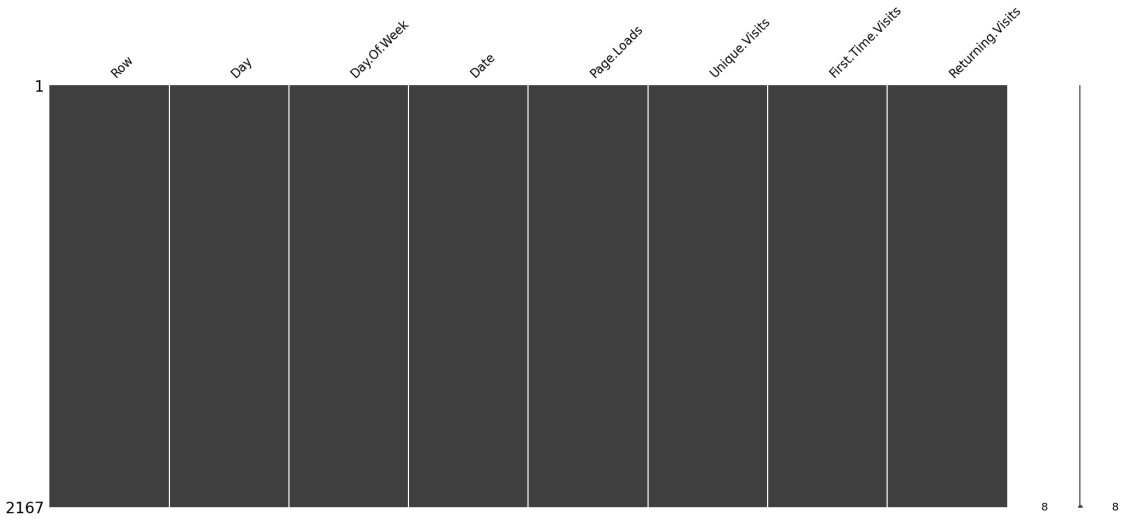
[7]:
```
df.columns.values
```

[7]:
```
array(['Row', 'Day', 'Day.Of.Week', 'Date', 'Page.Loads', 'Unique.Visits',
       'First.Time.Visits', 'Returning.Visits'], dtype=object)
```

[8]:
```
df.dtypes
```

[8]:
```
Row                 int64
Day                object
Day.Of.Week         int64
Date               object
Page.Loads         object
Unique.Visits      object
First.Time.Visits  object
Returning.Visits   object
dtype: object
```

[9]:
```
msno.matrix(df);
```



[10]:
```
df = df.drop(['Unique.Visits'],axis = 1)
df.head()
```

[10]:

	Row	Day	Day.Of.Week	Date	Page.Loads	First.Time.Visits	Returning.Visits
0	1	Sunday	1	9/14/2014	2,146	1,430	152
1	2	Monday	2	9/15/2014	3,621	2,297	231
2	3	Tuesday	3	9/16/2014	3,698	2,352	278
3	4	Wednesday	4	9/17/2014	3,667	2,327	287
4	5	Thursday	5	9/18/2014	3,316	2,130	236

[11]:
```
df.isnull()
```

[11]:

	Row	Day	Day.Of.Week	Date	Page.Loads	First.Time.Visits	\
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
…	…	…	…	…	…	…	
2162	False	False	False	False	False	False	
2163	False	False	False	False	False	False	
2164	False	False	False	False	False	False	

```
2165  False  False        False  False       False              False
2166  False  False        False  False       False              False

      Returning.Visits
0                False
1                False
2                False
3                False
4                False
…                …
2162             False
2163             False
2164             False
2165             False
2166             False

[2167 rows x 7 columns]
```

[12]:
```
df.isnull().sum()
```

[12]:
```
Row                 0
Day                 0
Day.Of.Week         0
Date                0
Page.Loads          0
First.Time.Visits   0
Returning.Visits    0
dtype: int64
```

[13]:
```
df['Row'] = pd.to_numeric(df.Row,errors='coerce')
df.isnull().sum()
```

[13]:
```
Row                 0
Day                 0
Day.Of.Week         0
Date                0
Page.Loads          0
First.Time.Visits   0
Returning.Visits    0
dtype: int64
```

[14]:
```
df[np.isnan(df['Row'])]
```

[14]:
```
Empty DataFrame
Columns: [Row, Day, Day.Of.Week, Date, Page.Loads, First.Time.Visits,
Returning.Visits]
Index: []
```

[15]:
```
df.fillna(df['Row'].mean())
```

[15]:

	Row	Day	Day.Of.Week	Date	Page.Loads	First.Time.Visits	Returning.Visits
0	1	Sunday	1	9/14/2014	2,146	1,430	152
1	2	Monday	2	9/15/2014	3,621	2,297	231
2	3	Tuesday	3	9/16/2014	3,698	2,352	278
3	4	Wednesday	4	9/17/2014	3,667	2,327	287
4	5	Thursday	5	9/18/2014	3,316	2,130	236
…	…	…	…	…	…	…	…
2162	2163	Saturday	7	8/15/2020	2,221	1,373	323
2163	2164	Sunday	1	8/16/2020	2,724	1,686	351
2164	2165	Monday	2	8/17/2020	3,456	2,181	457
2165	2166	Tuesday	3	8/18/2020	3,581	2,184	499
2166	2167	Wednesday	4	8/19/2020	2,064	1,297	267

[2167 rows x 7 columns]

[16]:
```
df["Date"] = pd.to_datetime(df["Date"],format="%m/%d/%Y")
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2167 entries, 0 to 2166
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Row                2167 non-null   int64
 1   Day                2167 non-null   object
 2   Day.Of.Week        2167 non-null   int64
 3   Date               2167 non-null   datetime64[ns]
 4   Page.Loads         2167 non-null   object
 5   First.Time.Visits  2167 non-null   object
 6   Returning.Visits   2167 non-null   object
dtypes: datetime64[ns](1), int64(2), object(4)
memory usage: 118.6+ KB
```

```
None
```

[17]:
```
df.isnull().sum()
```

[17]:
```
Row                 0
Day                 0
Day.Of.Week         0
Date                0
Page.Loads          0
First.Time.Visits   0
Returning.Visits    0
dtype: int64
```

[18]:
```
df["Returning.Visits"]=df['Returning.Visits'].map({0:"no", 1: "yes"})
df.head()
```

[18]:
```
   Row        Day  Day.Of.Week       Date Page.Loads First.Time.Visits  \
0    1     Sunday            1 2014-09-14      2,146             1,430
1    2     Monday            2 2014-09-15      3,621             2,297
2    3    Tuesday            3 2014-09-16      3,698             2,352
3    4  Wednesday            4 2014-09-17      3,667             2,327
4    5   Thursday            5 2014-09-18      3,316             2,130

  Returning.Visits
0              NaN
1              NaN
2              NaN
3              NaN
4              NaN
```

[19]:
```
df["Returning.Visits"].describe(include=['object','bool'])
```

[19]:
```
count       0
unique      0
top       NaN
freq      NaN
Name: Returning.Visits, dtype: object
```

[20]:
```
df[df['Row'] == 0].index
```

[20]:
```
Int64Index([], dtype='int64')
```

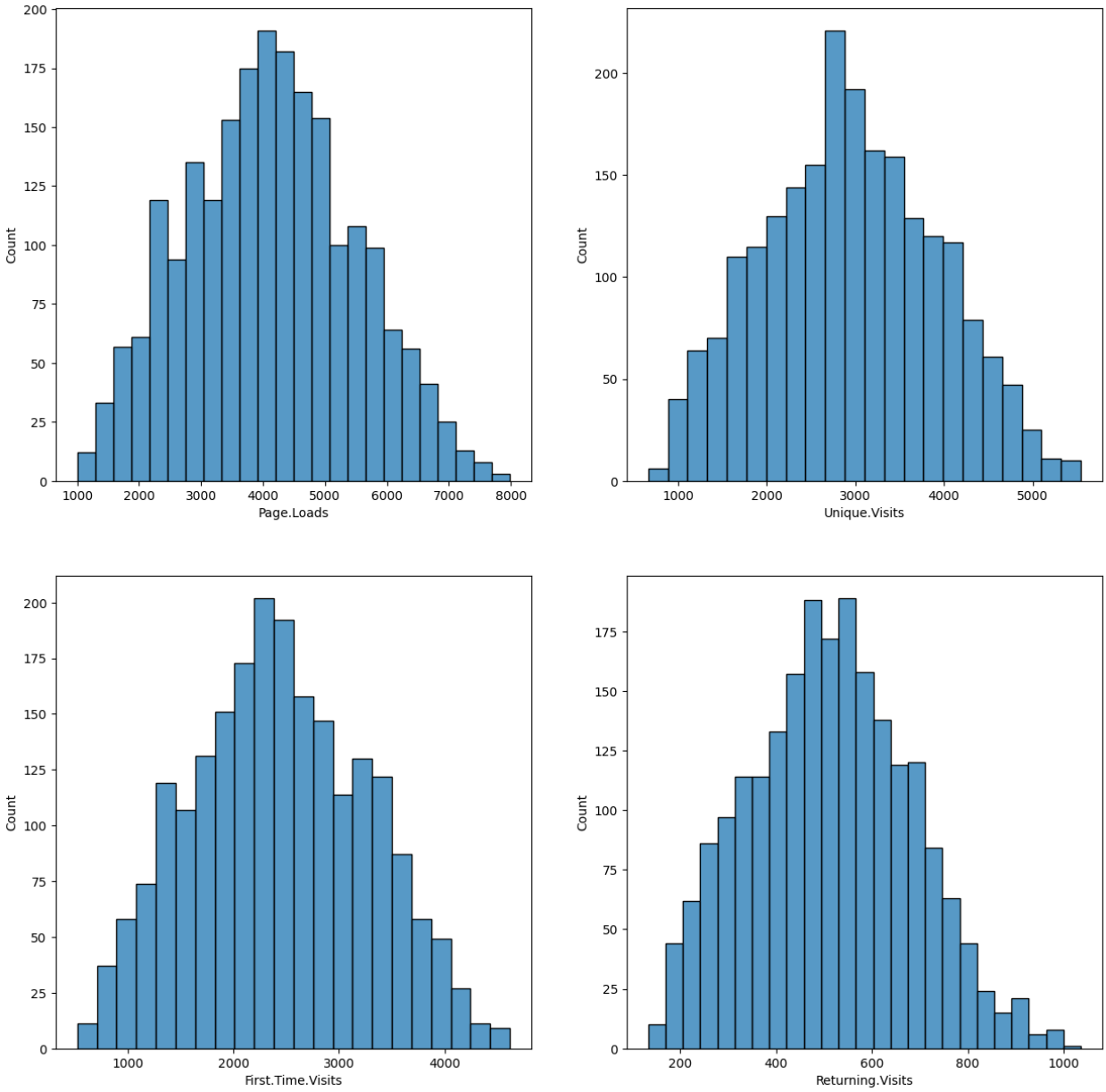
[21]:
```
numerical_cols = ['Row','First.Time.Visits','Returning.Visits']
df[numerical_cols].describe()
```

[21]:
```
                Row
count  2167.000000
```

```
mean   1084.000000
std     625.703338
min       1.000000
25%     542.500000
50%    1084.000000
75%    1625.500000
max    2167.000000
```

[ ]:

```
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)
```



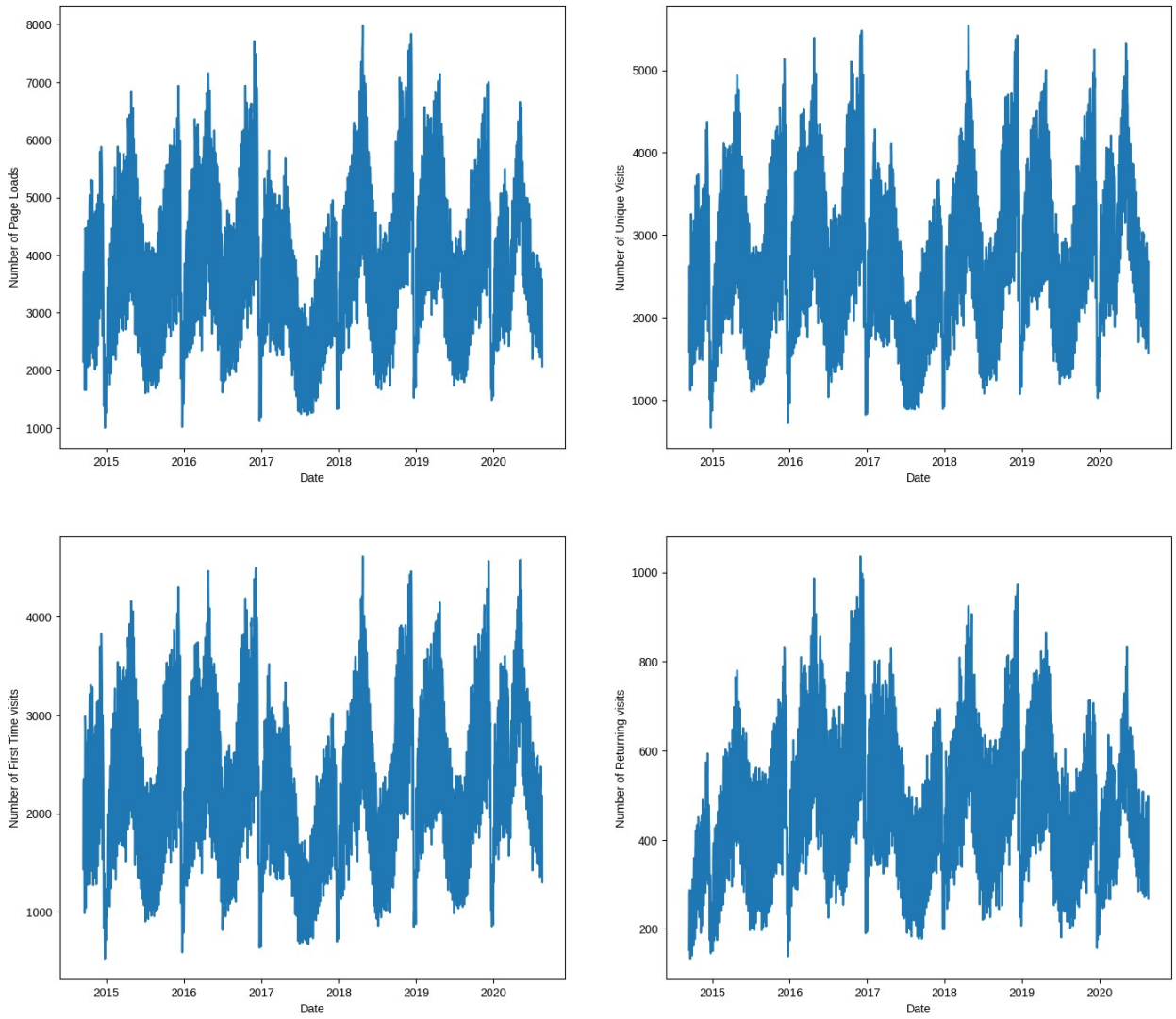

```
def check_normality(data, col):

    # 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()
```

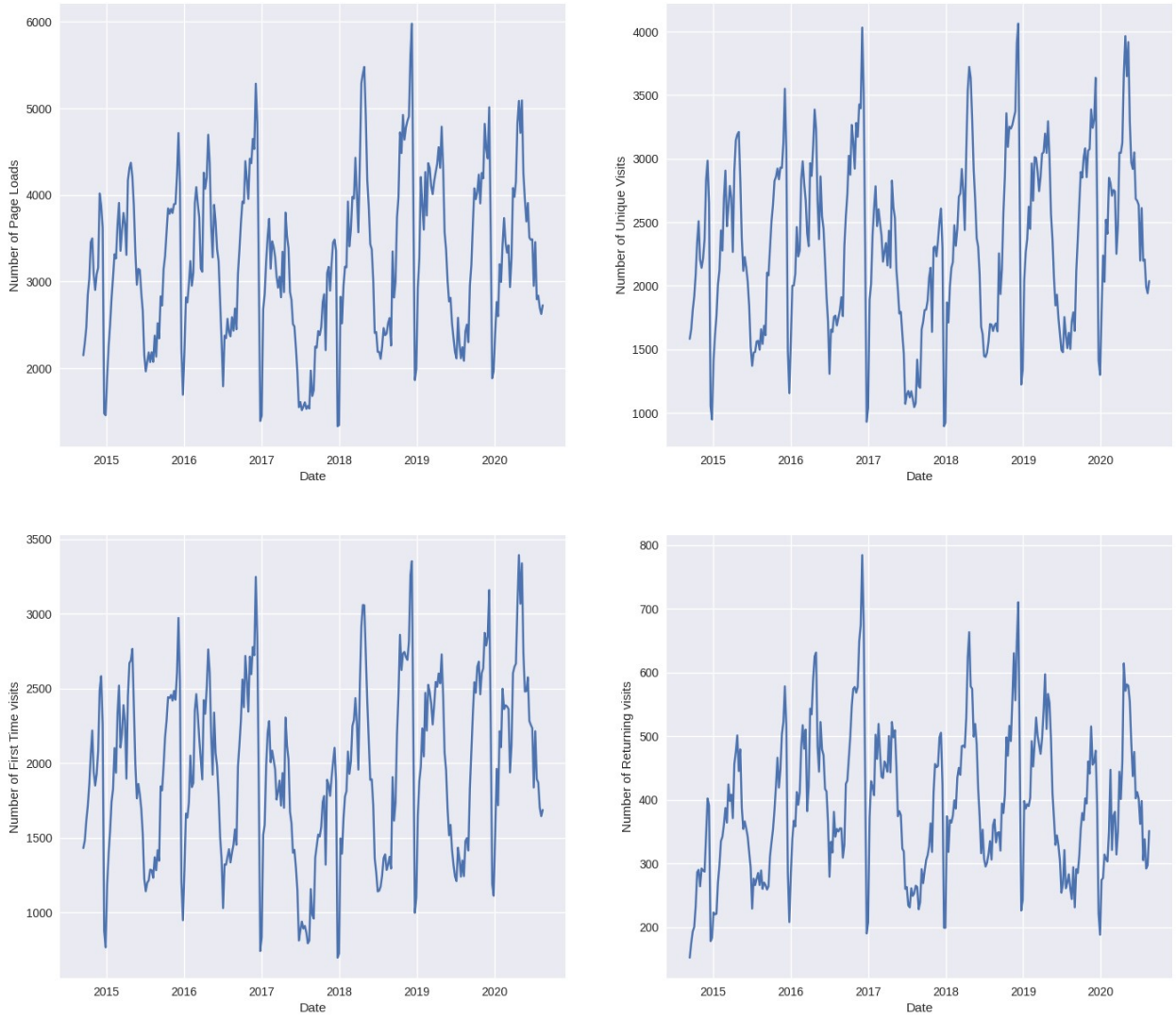
```
<ipython-input-17-45ff7e18345e>:2: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles shipped by seaborn. However, they will remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn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

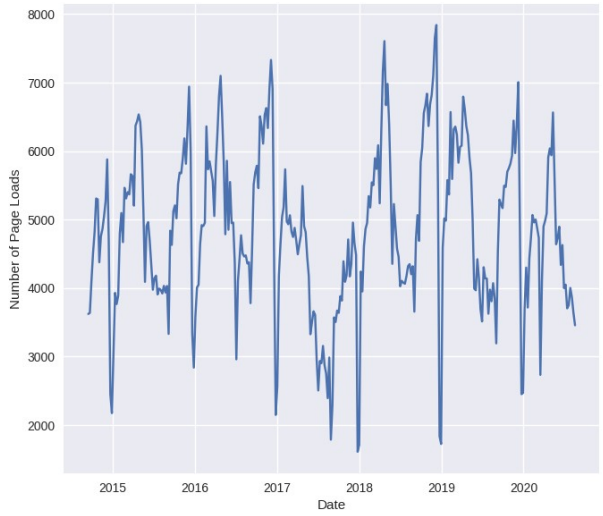
```
    # Plot the Number of Page Loads with time

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')
```

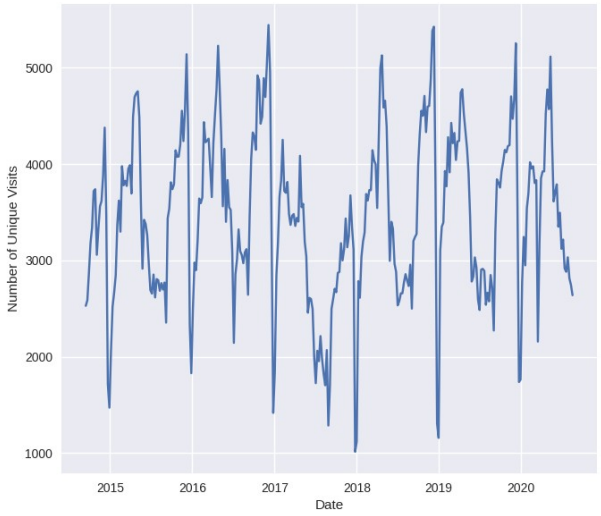
```
<ipython-input-19-0925a12011a4>:4: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they
no longer correspond to the styles shipped by seaborn. However, they
will remain available as 'seaborn-v0_8-<style>'. Alternatively,
directly use the seaborn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

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

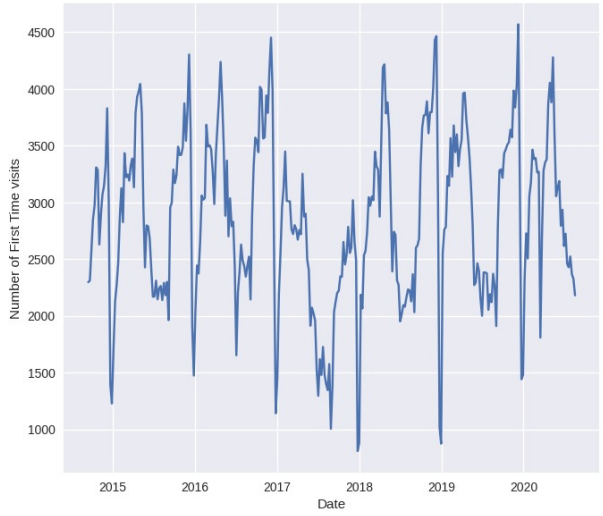


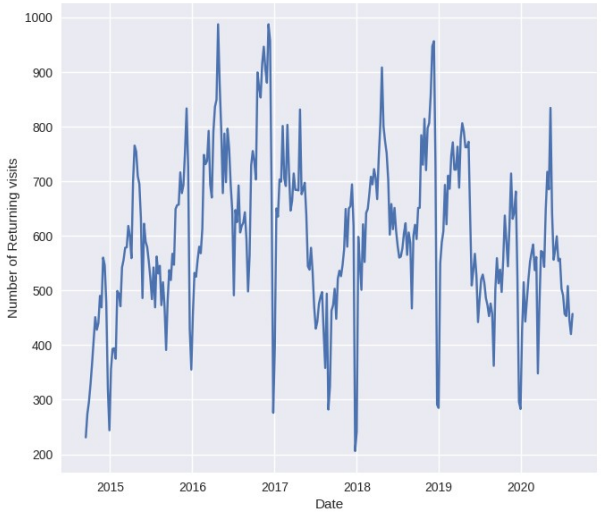

```
day_wise_EDA('Monday')
```

```
<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==========================================================
```



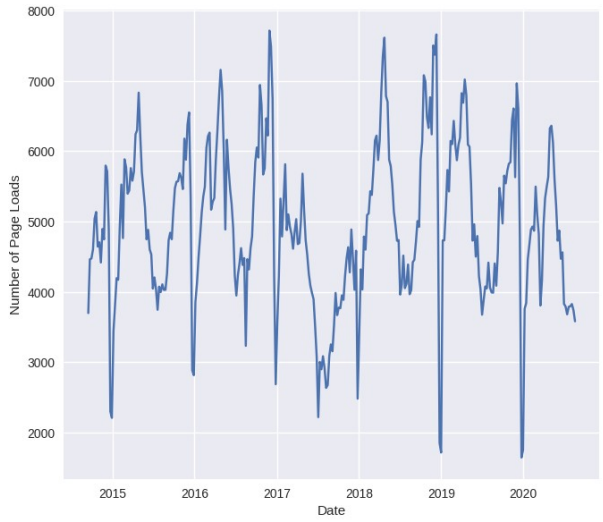


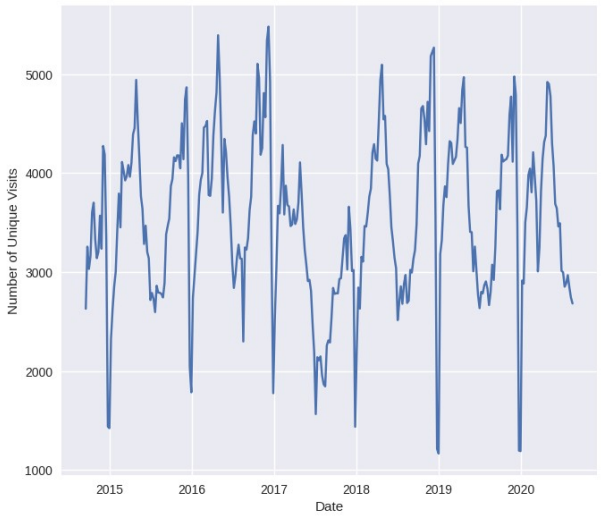


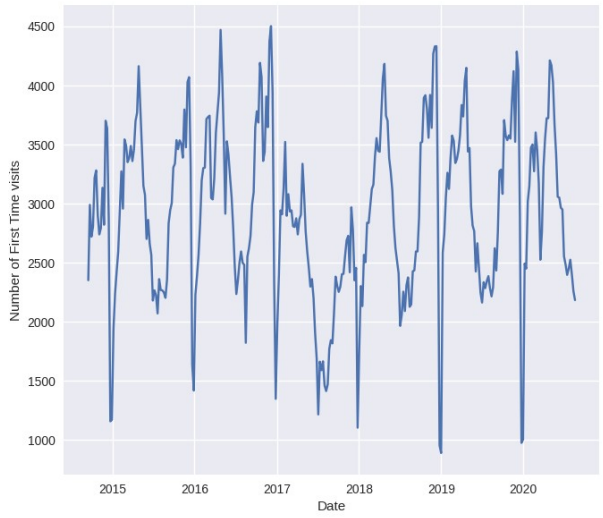


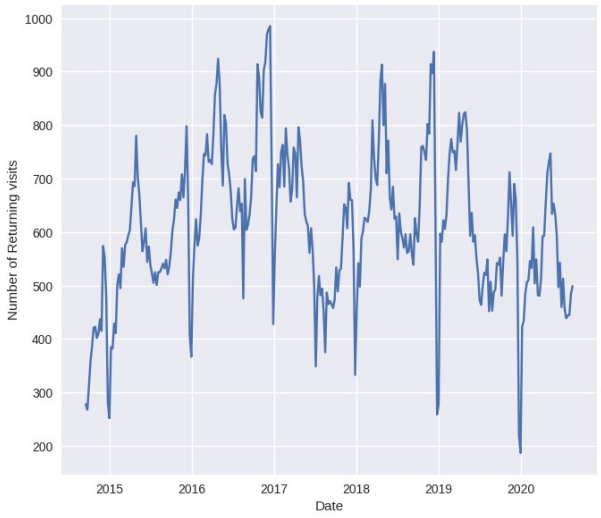

```
day_wise_EDA('Tuesday')
```

```
<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============================================================
```



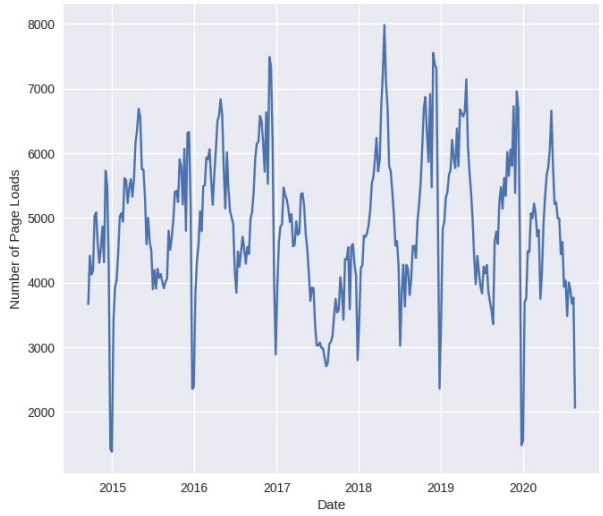


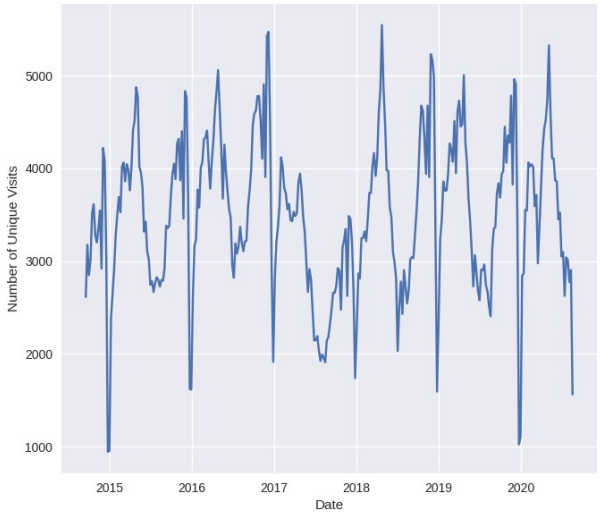


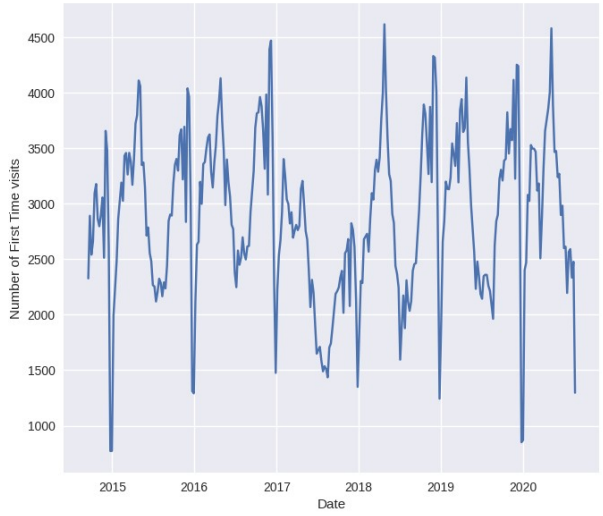


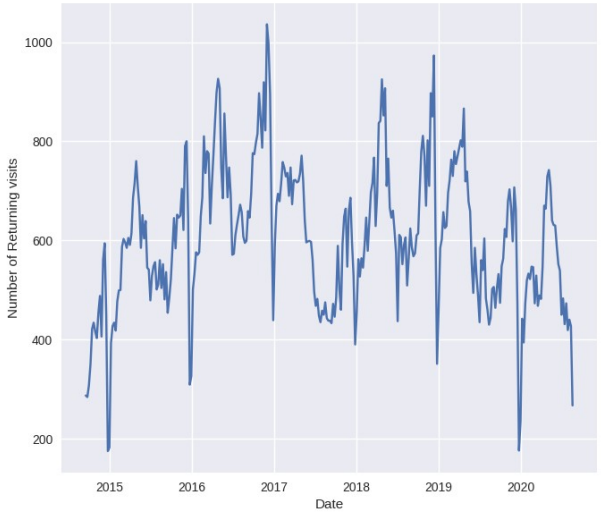

```
day_wise_EDA('Wednesday')
```

```
<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=========================================================
```









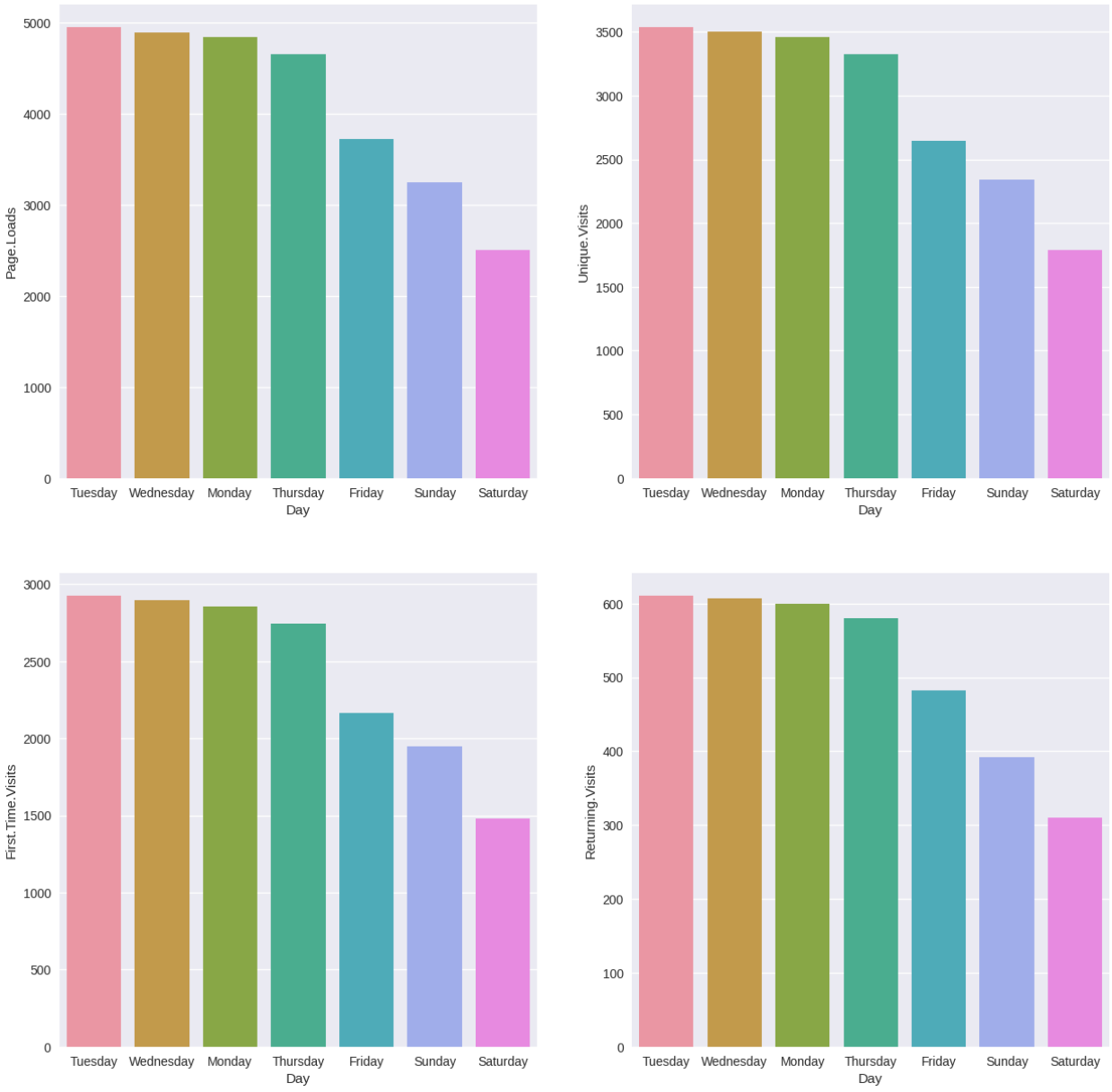

```
avg_day_data = day_grouped_data.mean().reset_index().drop('Row',
axis=1)
avg_day_data
```

```
<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)
```

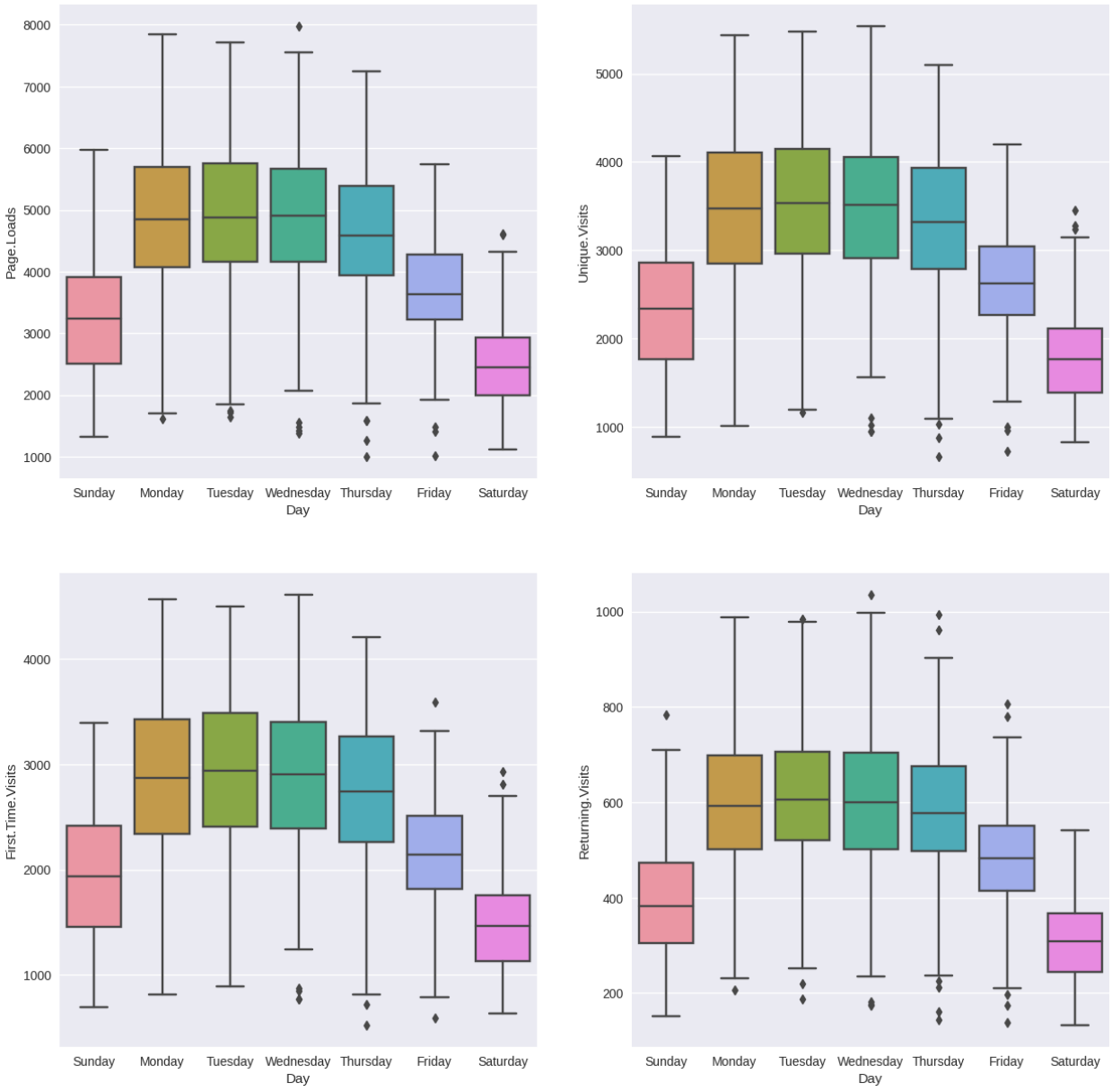
```
         Day  Day.Of.Week   Page.Loads  Unique.Visits
First.Time.Visits  \
0     Friday          6.0  3719.860841    2646.770227
2164.417476
1     Monday          2.0  4845.680645    3458.425806
```

```
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)
```

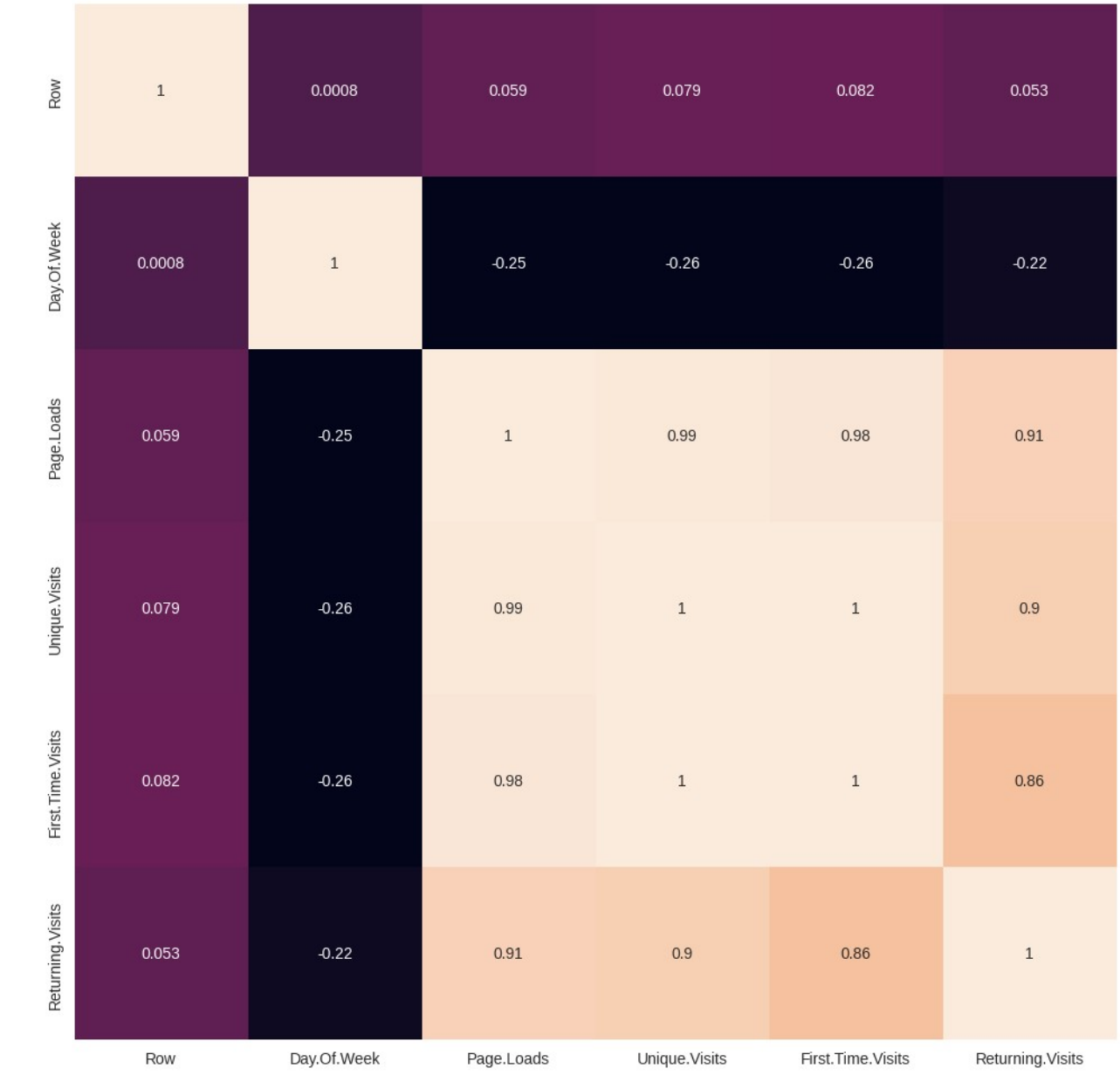



```
# Boxplots for all the continuous columns across day
cols_to_plot = ['Page.Loads', 'Unique.Visits', 'First.Time.Visits',
'Returning.Visits']
plt.figure(figsize=(15, 15))
for i, col in enumerate(cols_to_plot):
    plt.subplot(2, 2, i+1)
    sns.boxplot(data=data, x='Day', y=col)
```



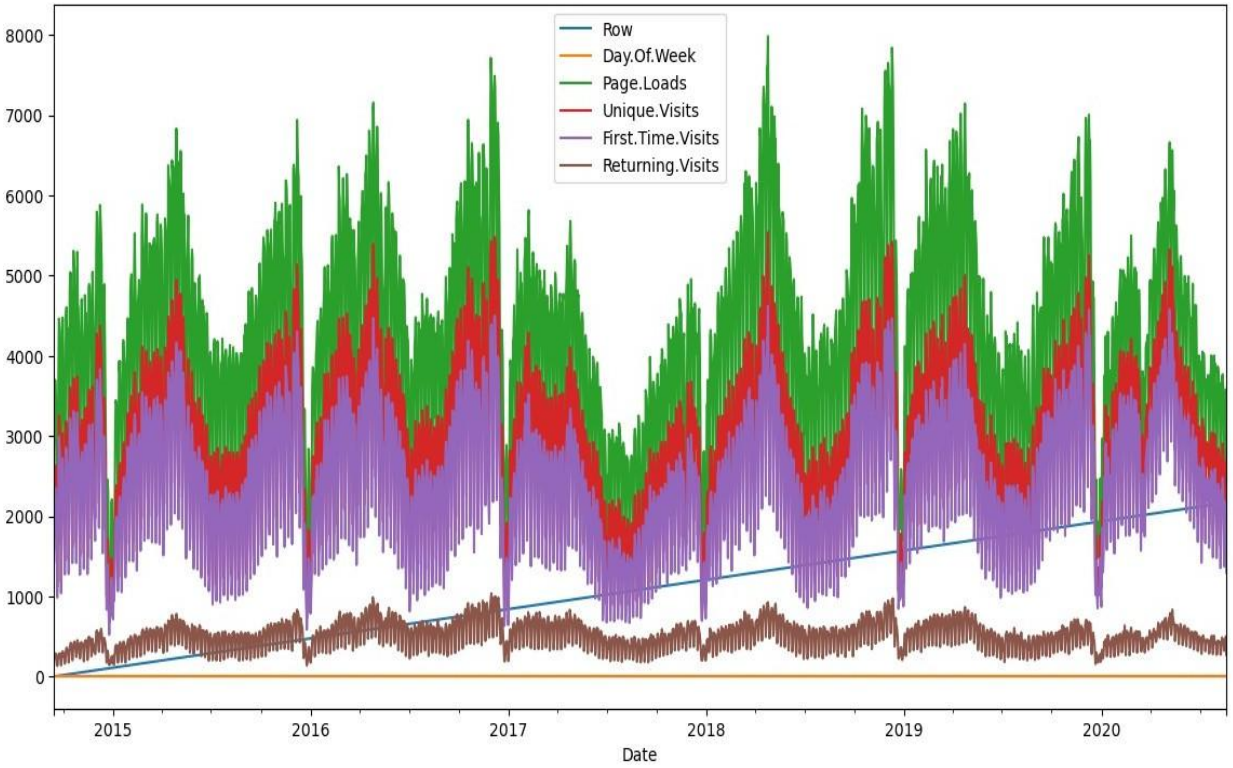

```
#Plot the correlation heatmap
corr_matrix = data.corr()
plt.figure(figsize=(12,12))
sns.heatmap(corr_matrix, annot=True, cbar=False)
plt.show()
```

```
<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()
```

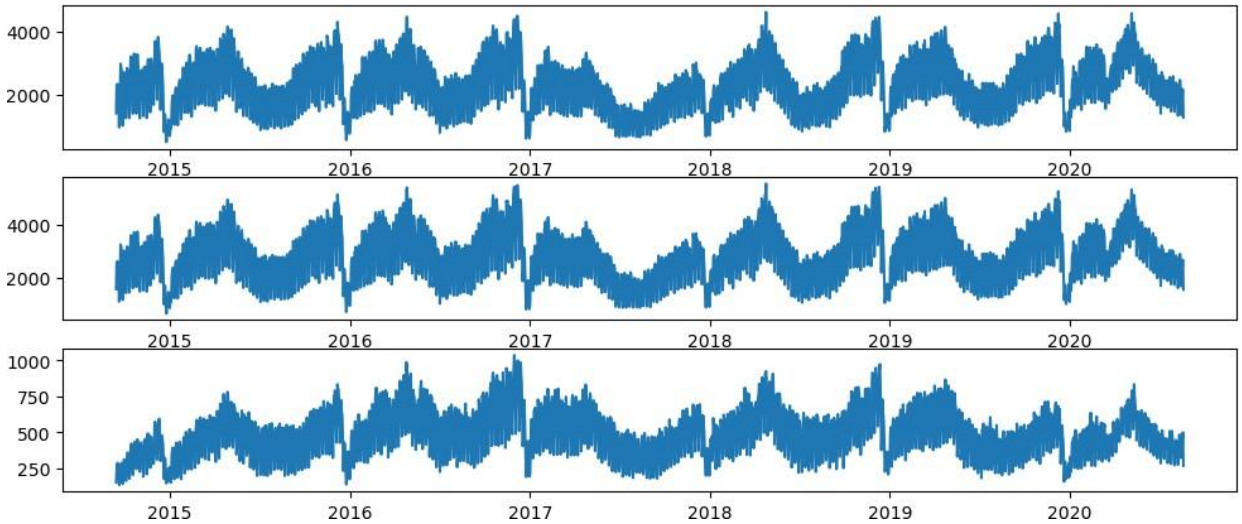

Row
Day.Of.Week
Page.Loads
Unique.Visits
First.Time.Visits
Returning.Visits
1 0.0008 0.059 0.079 0.082 0.053
0.0008 1 -0.25 -0.26 -0.26 -0.22
0.059 -0.25 1 0.99 0.98 0.91
0.079 -0.26 0.99 1 1 0.9
0.082 -0.26 0.98 1 1 0.86
0.053 -0.22 0.91 0.9 0.86 1
Row
Day.Of.Week
Page.Loads
Unique.Visits
First.Time.Visits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

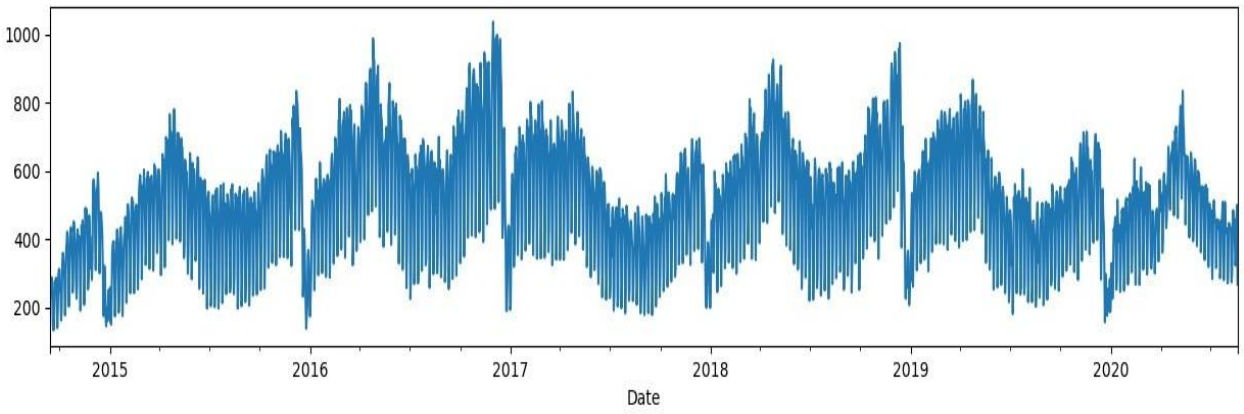
```
Date
2014-09-14    152
2014-09-15    231
2014-09-16    278
2014-09-17    287
2014-09-18    236
              ... 2020-08-15           323
2020-08-16    351
2020-08-17    457
2020-08-18    499
2020-08-19    267
Name: Returning.Visits, Length: 2167, dtype: int64
```

```
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]
```



Advantages:

Website traffic analysis offers several advantages, including:

Insight into User Behavior: It helps you understand how visitors interact with your website, what pages they visit, how long they stay, and what actions they take. This information can inform content and design decisions.

Audience Understanding: You can gain insights into your website's audience demographics, interests, and geographic locations, allowing you to tailor your content and marketing strategies accordingly.

Performance Measurement: Analyzing website traffic helps you track the success of your marketing campaigns, content efforts, and overall website performance. You can see which strategies are working and which need improvement.

Conversion Optimization: By monitoring conversion rates, you can identify where visitors drop off in your sales or signup process and make necessary adjustments to improve conversion rates.

Content Strategy: It aids in content planning by showing which pages and topics are most popular, helping you create more of what your audience is interested in.

SEO Improvement: Website traffic analysis can reveal the keywords that drive traffic to your site. This data helps in optimizing your content for search engines.

Competitive Analysis: You can benchmark your website's performance against competitors and identify areas where you can outperform them.

Security Monitoring: It can help detect unusual or suspicious website activity, potentially indicating security threats.

User Experience Enhancement: By understanding user behavior, you can enhance the overall user experience, making navigation smoother and more intuitive.

Data-Driven Decision Making: Website traffic data provides valuable insights for data-driven decision-making, ensuring your website evolves in response to user needs and market trends.

Cost Efficiency: It allows you to allocate resources more efficiently, focusing on strategies and content that drive the most traffic and conversions.

Disadvantage

Disadvantages of website traffic analysis can include:

Incomplete Data: Analysis may not capture all visitor interactions due to factors like ad blockers, JavaScript disabled, or privacy settings, leading to incomplete insights.

Interpretation Challenges: Interpreting data correctly requires expertise, and misinterpretation can lead to incorrect conclusions.

Privacy Concerns: Gathering user data for analysis may raise privacy concerns and legal compliance issues, especially with stricter data protection regulations.

Sample Bias: Small sample sizes can result in skewed insights, especially for newer websites with limited traffic.

Data Overload: Analyzing too much data can be overwhelming, making it challenging to focus on meaningful metrics.

Real-Time Limitations: Some analyses may not be real-time, which can impact immediate decision-making.

Limited Context: Analyzing data alone may not provide a complete picture, as it lacks context and qualitative insights from users.

Cost and Resource Intensive: Implementing and maintaining analytics tools and experts can be costly for smaller businesses.

Accuracy Challenges: Data can be affected by factors like bots, referral spam, or inaccurate tracking methods.

Time-Consuming: Analyzing website traffic data can be time-consuming, diverting resources from other tasks.