TAMILNADU MARGINAL WORKERS ASSESSMENT

Data Analytics with cognos – Phase 3

DOCUMENTATION

Team Members:

1.Kowsalya.A(au613021205028)

2.Dhanusha.R(au613021205006)

3.Mownika.M(au613021205034)

4.Vimali.D(au613021205060)

5.Priyadharsini.S(au613021205037)

Phase 3: Development Part 1

Problem Definition:

Start the Website Traffic analysis by loading and preprocessing the dataset. Load the dataset using python and data manipulation libraries (e.g., pandas).

Dataset Link:

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

Overview of the process:

1.Libraries and APIs:

Use libraries in programming languages like Python to access web analytics data through the APIs provided by your chosen web analytics tool. For example,

you can use libraries like google-api-python-client for Google Analytics or requests to interact with other APIs.

2.Load the Dataset:

Use pd.read_csv() or other appropriate methods to load your dataset into a pandas DataFrame.

3.Data Collection:

Implement web analytics tools: Start by setting up web analytics tools like Google Analytics, Adobe Analytics, or other third-party tools. These tools help track and collect data on website traffic.

4.Define Goals and KPIs:

Clearly define your website's goals and key performance indicators (KPIs). Common KPIs include conversion rates, page views, bounce rates, and average session duration.

5.Performance Monitoring:

Continuously monitor your website's performance by tracking KPIs over time. Look for trends, spikes, or dips in traffic and user engagement.

6.Actionable Insights:

Use the insights gained from your analysis to make data-driven decisions and implement changes to improve your website's performance, user experience, and conversions.

6.Data Privacy and Compliance:

Ensure that you're compliant with data privacy regulations, such as GDPR and CCPA, and handle user data responsibly and securely.

Loading the dataset:

1.Importing libraries

Here, for preprocessing the dataset and manipulate the data, pandas is the library used to frame the data.

Code:

```
import pandas as pd
```

2.Loading the dataset

In this step, we are framing the data into the table using DataFrame in pandas, and display the head or 5 rows of the dataset.

Code:

```
# Replace with the actual filename
data=pd.read_csv("C:/Users/dhanu/OneDrive/Desktop/PHASE 3.csv")
```

Preprocessing the dataset

3.Explore the dataset:

After framing data, the first few or five rows of the data in displayed using the head() function.

Code:

```
data
```

Output:

Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits

0	1	Sunday	1	9/14/2014	2,146	1,582	1,430	152
1	2	Monday	2	9/15/2014	3,621	2,528	2,297	231
2	3	Tuesday	3	9/16/2014	3,698	2,630	2,352	278
3	4	Wednesday	4	9/17/2014	3,667	2,614	2,327	287
4	5	Thursday	5	9/18/2014	3,316	2,366	2,130	236
...	...	...	...	...	...	...	...	...
2162	2163	Saturday	7	8/15/2020	2,221	1,696	1,373	323
2163	2164	Sunday	1	8/16/2020	2,724	2,037	1,686	351
2164	2165	Monday	2	8/17/2020	3,456	2,638	2,181	457
2165	2166	Tuesday	3	8/18/2020	3,581	2,683	2,184	499
2166	2167	Wednesday	4	8/19/2020	2,064	1,564	1,297	267

2167 rows × 8 columns

4.Check for missing values:

In this step, the missing values or null values, if it present in the data are separated and number of null values are shown through this code.

Code:

```
print("Missing values:\n", data.isnull().sum())
```

Output:

```
Missing values:
Missing values:/n Row                0
Day                  0
Day.Of.Week          0
Date                 0
Page.Loads           0
Unique.Visits        0
First.Time.Visits    0
Returning.Visits     0
dtype: int64
```

5.Check datatype:

In this step, the data type of the columns are discussed

Code: print("Data Types:\n", data.dtypes)

Output:

```
Data Types:
Data Types:/n 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

6.Check basic statistics:

the statistics of the columns such as count, mean, std, min, max, 25%, 50%, 75% are shown through the describe() function command.

Code:

```
print("Summary Statistics:\n", data.describe())
```

Output:

Summary Statistics:

Summary Statistics:

	Row	Day.Of.Week
count	2167.000000	2167.000000
mean	1084.000000	3.997231
std	625.703338	2.000229
min	1.000000	1.000000

25% 542.500000 2.000000

50% 1084.000000 4.000000

75% 1625.500000 6.000000
max 2167.000000 7.000000

7.Additional Preprocessing steps:

Perform any other preprocessing steps that are specific to your dataset and analysis goals. This may include scaling numeric features, handling outliers, or creating new features.

8.Saving Preprocessed dataset:

In this step, if we made substantial changes to the dataset and want to save the preprocessed version, you can use the following Code.
Code:

```
# Save the preprocessed dataset to a new CSV file
data.to_csv('preprocessed_dataset.csv', index=False)
```

9.Vizualization:

```
import numpy as np

import pandas as pd

import pandas_profiling

import warnings

warnings.filterwarnings('ignore')

import datetime
```

```
from datetime import date


import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

sns.set_style("whitegrid")


# import chart_studio.plotly as py

import cufflinks as cf

import plotly.express as px


from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot

init_notebook_mode(connected=True)


cf.go_offline()


import pandas_profiling

import plotly.graph_objects as go


from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV

from sklearn.metrics import accuracy_score

from sklearn.svm import SVR
```

```
from sklearn.linear_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

import xgboost as xg

# from prophet import Prophet
```

Importing the required dataset, renaming the columns, removing the commas from the columns and converting their data types

In [2]:

```
df=pd.read_csv('../input/daily-website-visitors/daily-website-visitors.csv')


df.rename(columns = {'Day.Of.Week':'day_of_week'

                    ,'Page.Loads':'page_loads'

                    ,'Unique.Visits':'unique_visits'

                    ,'First.Time.Visits':'first_visits'

                    ,'Returning.Visits':'returning_visits'}, inplace = True)


df=df.replace(',','',regex=True)


df['page_loads']=df['page_loads'].astype(int)

df['unique_visits']=df['unique_visits'].astype(int)

df['first_visits']=df['first_visits'].astype(int)

df['returning_visits']=df['returning_visits'].astype(int)
```

df

Out[2]:

	Row	Day	day_of_week	Date	page_loads	unique_visits	first_visits	returning_visits
0	1	Sunday	1	9/14/2014	2146	1582	1430	152
1	2	Monday	2	9/15/2014	3621	2528	2297	231
2	3	Tuesday	3	9/16/2014	3698	2630	2352	278
3	4	Wednesday	4	9/17/2014	3667	2614	2327	287
4	5	Thursday	5	9/18/2014	3316	2366	2130	236
...	...	...	...	...	...	...	...	...
2162	2163	Saturday	7	8/15/2020	2221	1696	1373	323
2163	2164	Sunday	1	8/16/2020	2724	2037	1686	351

2164	2165	Monday	2	8/17/2020	3456	2638	2181	457
2165	2166	Tuesday	3	8/18/2020	3581	2683	2184	499
2166	2167	Wednesday	4	8/19/2020	2064	1564	1297	267

2167 rows × 8 columns

Checking for the null values if any

In [3]:

```
df.isna().sum()
```

Out[3]:

```
Row              0
Day              0
day_of_week      0
Date             0
page_loads       0
unique_visits    0
first_visits     0
returning_visits 0
dtype: int64
```

Checking for duplicate values if any

In [4]:

```
df.duplicated().sum()
```

Out[4]:

```
0
```

In [5]:

```
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   int64 
 5   unique_visits   2167 non-null   int64 
 6   first_visits    2167 non-null   int64 
```

7 returning_visits 2167 non-null int64

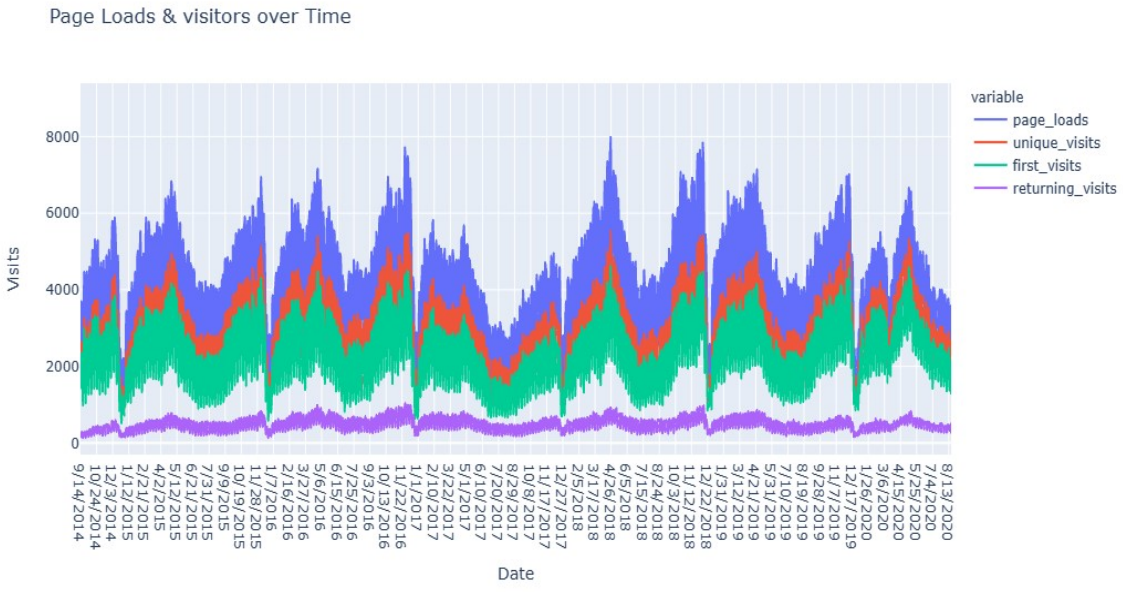
dtypes: int64(6), object(2)

memory usage: 135.6+ KB

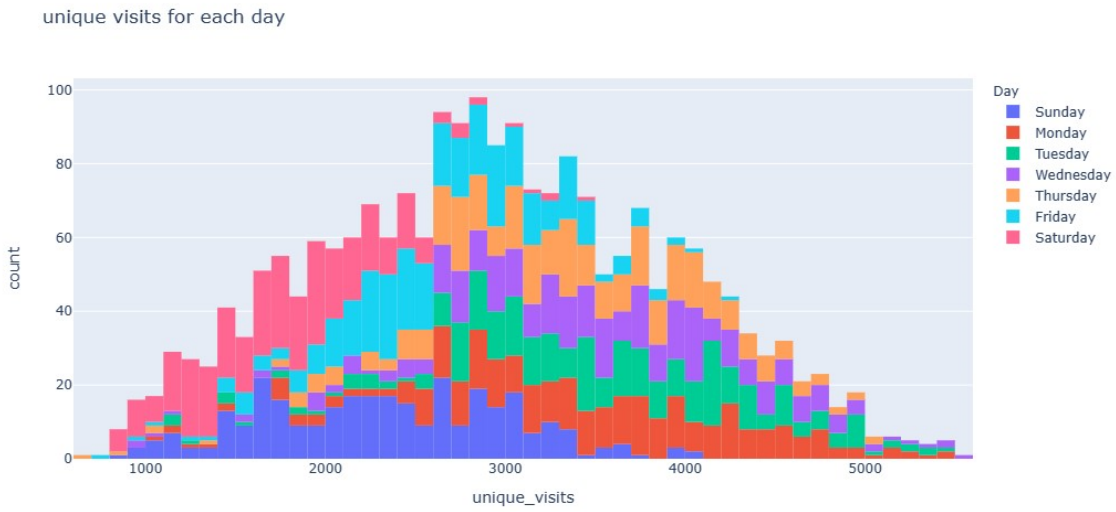
generating line plot for visualizing the trend of page loads and visits over time series, it seems that page loads and visits have a constant fluctuation, means they have trend over time and are correlated to each other.

In [6]:

```
px.line(df,x='Date',y=['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits'],
    labels={'value':'Visits'}
    ,title='Page Loads & visitors over Time')
```

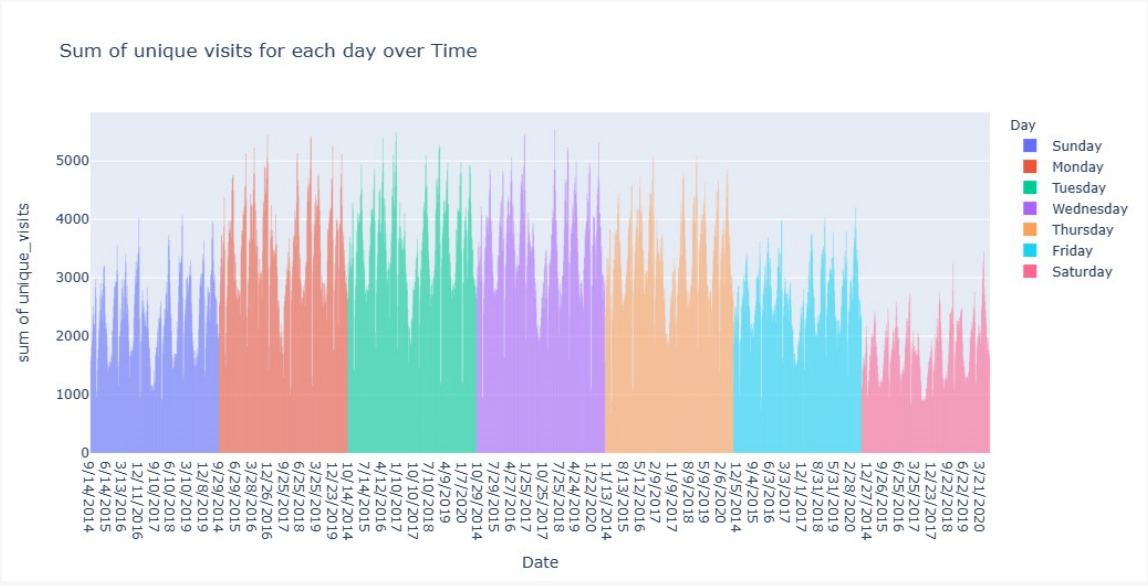



```
px.histogram(df,x='unique_visits',color='Day',title='unique visits for each day')
```




```
day_imp=df.groupby(['Day'])['unique_visits'].agg(['sum']).sort_values(by='sum',ascending=False)
px.bar(day_imp,labels={'value':'sum of unique visits'},title='Sum of Unique visits for each day')
```


```
px.histogram(df,x='Date',y='unique_visits',color='Day',title='Sum of unique visits for each day over Time')
```

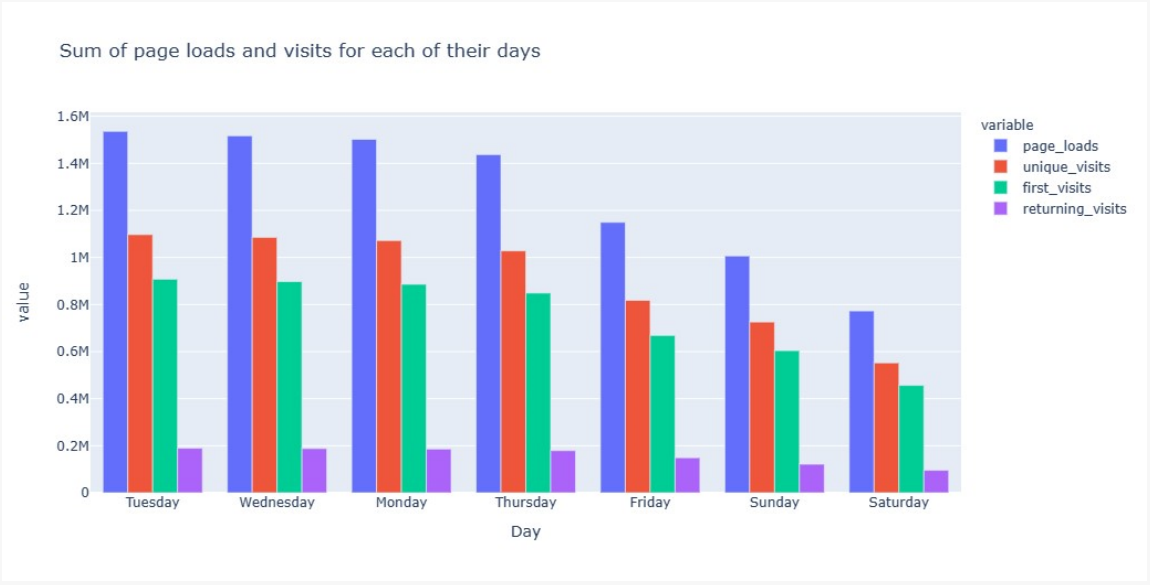



```
sums=df.groupby(['Day'])[['page_loads' ,'unique_visits' ,'first_visits'
,'returning_visits']].sum().sort_values(
   by='unique_visits',ascending=False)
sums
```

page_loads	unique_visits	first_visits	returning_visits	
Day				
Tuesday	1536154	1097181	907752	189429
Wednesday	1517114	1085624	897602	188022
Monday	1502161	1072112	886036	186076
Thursday	1437269	1028214	848921	179293
Friday	1149437	817852	668805	149047
Sunday	1006564	725794	604198	121596

Saturday	772817	552105	456449	95656

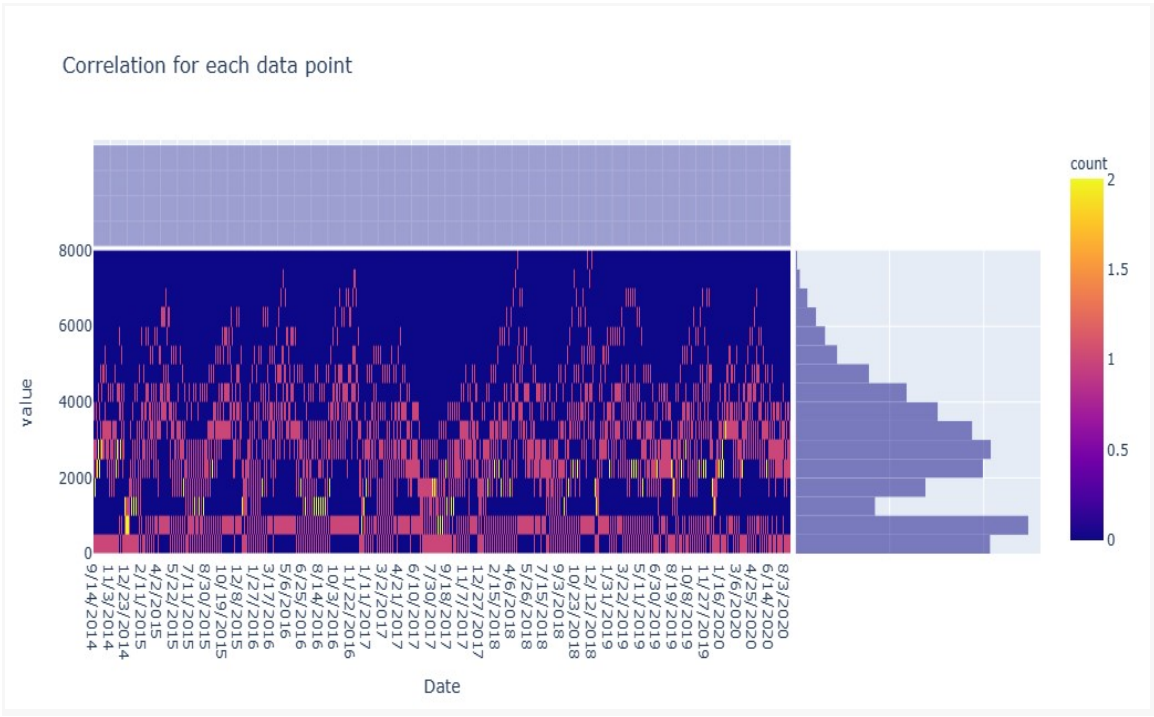
px.bar(sums,barmode='group',title='Sum of page loads and visits for each of their days')



px.density_heatmap(df, x='Date',y=['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']

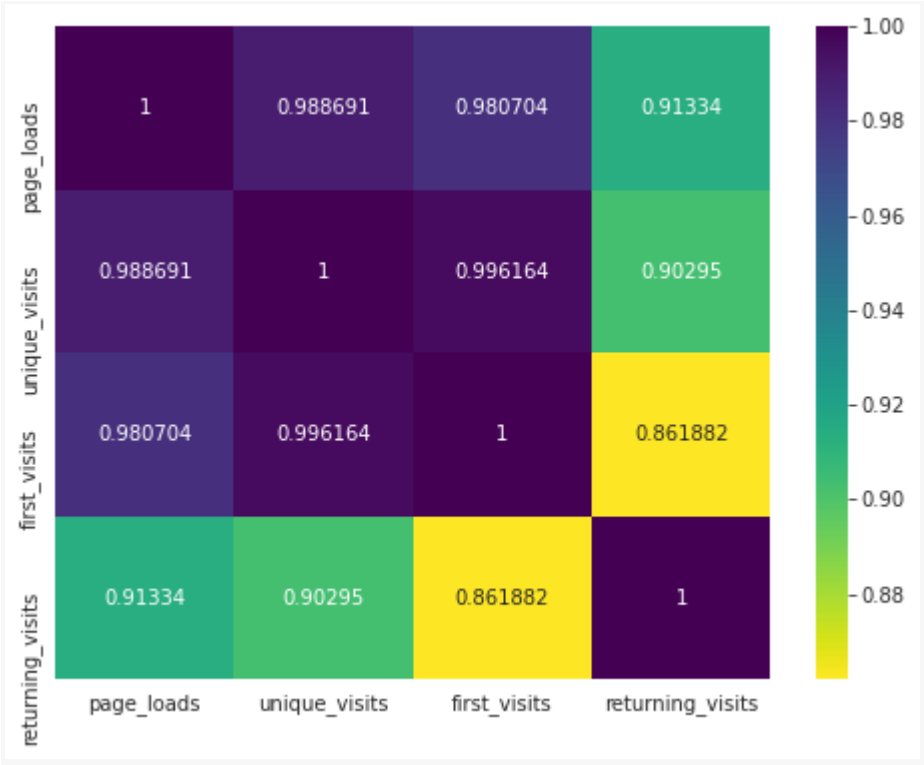
color_continuous_scale="Viridis"

,marginal_x="histogram", marginal_y="histogram",title='Correlation for each data point')




```
fig, ax = plt.subplots()
fig.set_size_inches(8, 6)
sns.heatmap(df[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits']].corr(),
            annot=True,
            cmap='viridis_r',
            fmt='g')
```

```
<AxesSubplot:>
```




```
px.scatter(
    df, x='first_visits', y='unique_visits',opacity=0.4,
    trendline='ols', trendline_color_override='purple',title="Regression line for unique visits and first visits"
)
```

pred_df=df[['page_loads' ,'unique_visits' ,'first_visits' ,'returning_visits','Day']]

Tuesday, wednesday, thursday and monday are the days when our website received the most traffic so we will create a feature days_f of them 1 value will define their existence and 0 will define the rest of the days.

In [18]:

```
pred_df['days_f']=np.where((df['Day']=='Tuesday') |  (df['Day']=='Wednesday') |
(df['Day']=='Thursday') |
(df['Day']=='Monday'),1,0)
```

pred_df

	page_loads	unique_visits	first_visits	returning_visits	Day	days_f
0	2146	1582	1430	152	Sunday	0
1	3621	2528	2297	231	Monday	1
2	3698	2630	2352	278	Tuesday	1
3	3667	2614	2327	287	Wednesday	1
4	3316	2366	2130	236	Thursday	1
...	...	...	...	...	...	...
2162	2221	1696	1373	323	Saturday	0

2163	2724	2037	1686	351	Sunday	0
2164	3456	2638	2181	457	Monday	1
2165	3581	2683	2184	499	Tuesday	1
2166	2064	1564	1297	267	Wednesday	1

2167 rows × 6 columns

Multi Linear Regression model

```
pred_df.drop('Day',axis=1,inplace=True)
# drop the days column as we don't need it anymore
```

```
pred_df.head(5)
```

	page_loads	unique_visits	first_visits	returning_visits	days_f
0	2146	1582	1430	152	0
1	3621	2528	2297	231	1
2	3698	2630	2352	278	1
3	3667	2614	2327	287	1

4	3316	2366	2130	236	1

separate the independent variable and dependent / target variable

```
X2=pred_df[['page_loads','first_visits' ,'returning_visits','days_f']]
y2=pred_df['unique_visits']
```

split the dataset in train and test samples now

```
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.3,random_state=42)
```

train the model with train sample

```
regressor2 = LinearRegression(fit_intercept=False,normalize=True)
regressor2.fit(X_train, y_train)
```

```
LinearRegression(fit_intercept=False, normalize=True)
```

```
y_pred2 = regressor2.predict(X_test)
```

```
lr2 = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred2})
lr2
```

	Actual	Predicted
1486	4173	4173.0

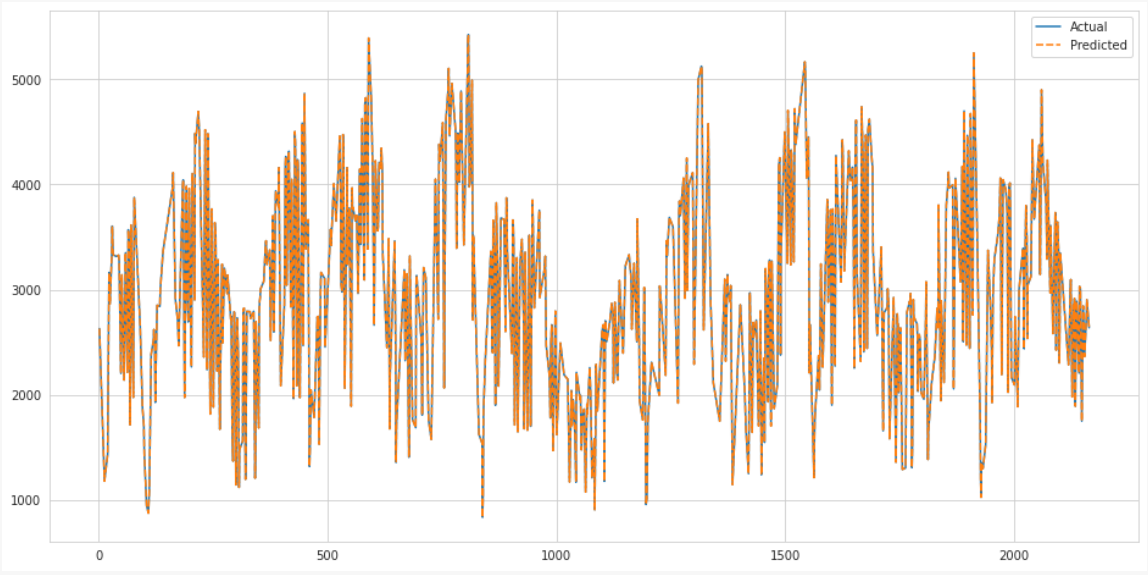
1602	1902	1902.0
1460	2870	2870.0
1134	2142	2142.0
1513	4329	4329.0
...	...	...
439	2579	2579.0
271	2494	2494.0
244	1818	1818.0
1159	3332	3332.0
1701	2565	2565.0

651 rows × 2 columns

visualize the actual and predicted values

```
plt.figure(figsize=(16,8))
sns.lineplot(data=lr2)
```

<AxesSubplot:>



get the accuacy score of the model.

regressor2.score(X_test,y_test)*100

100.0

Support Vector Regression

svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.00001)

svr_rbf.fit(X_train, y_train)

SVR(C=1000.0, gamma=1e-05)

```
y_pred3 = svr_rbf.predict(X_test)
```

```
svr = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred3})
svr
```

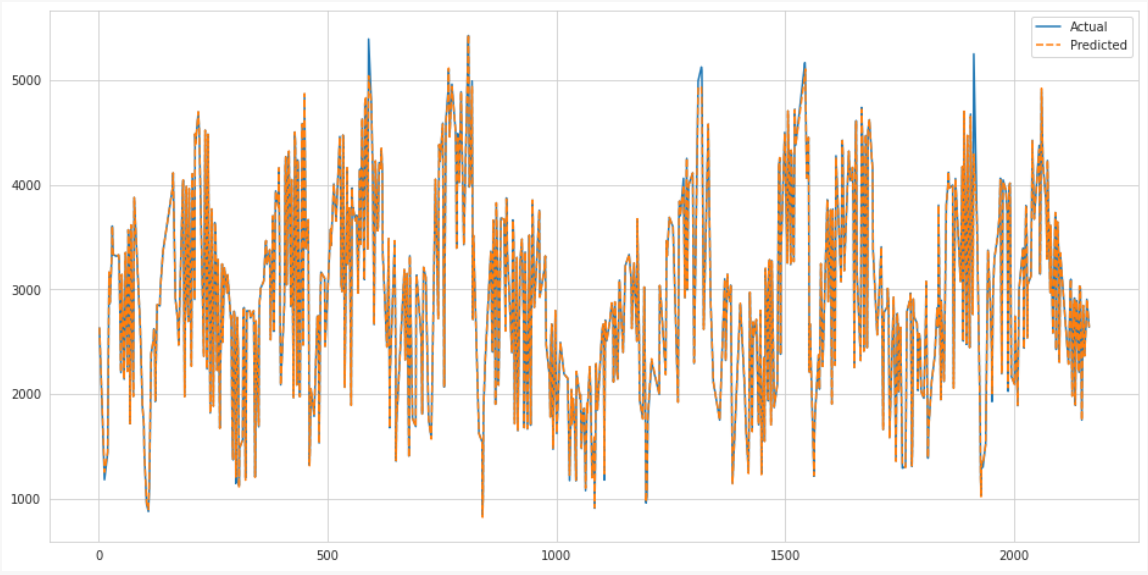
	Actual	Predicted
1486	4173	4173.783532
1602	1902	1904.847560
1460	2870	2870.181094
1134	2142	2142.904123
1513	4329	4328.316673
...	...	...
439	2579	2578.897313
271	2494	2493.887467
244	1818	1816.932763
1159	3332	3331.902324

1701	2565	2564.972314

651 rows × 2 columns

```
plt.figure(figsize=(16,8))
sns.lineplot(data=svr)
```

<AxesSubplot:>




```
svr_rbf.score(X_test,y_test)*100
```

99.80054455767926

Decision Tree Regression

```
dtr = DecisionTreeRegressor(random_state=0)
dtr.fit(X_train, y_train)
```

```
DecisionTreeRegressor(random_state=0)
```

```
dtr_pred = dtr.predict(X_test)
```

```
dtr_g = pd.DataFrame({'Actual': y_test, 'Predicted': dtr_pred})
dtr_g
```

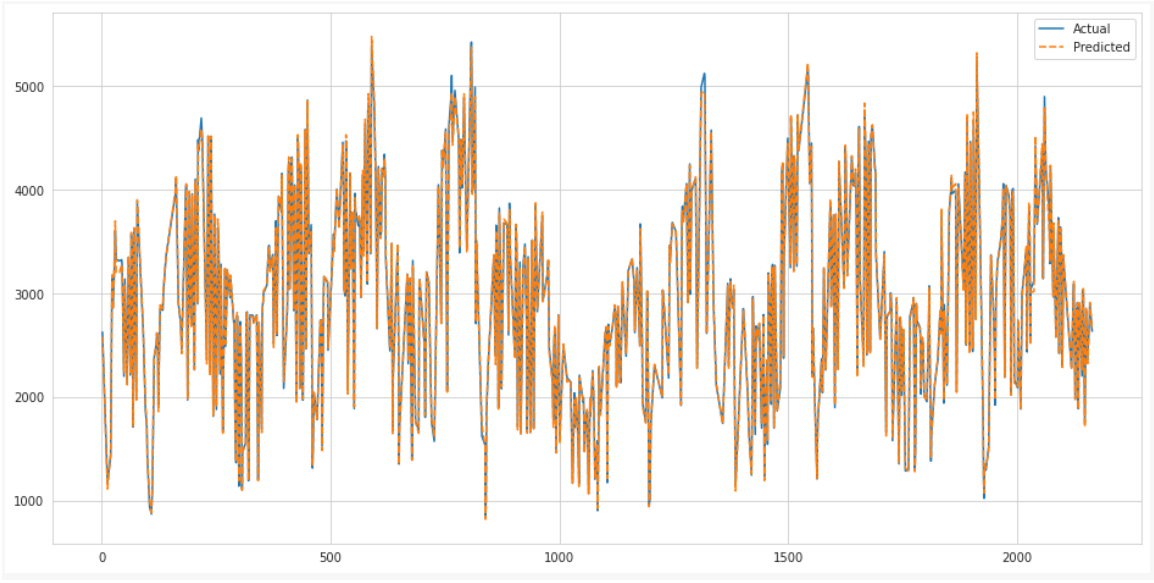
	Actual	Predicted
1486	4173	4140.0
1602	1902	1929.0
1460	2870	2871.0
1134	2142	2198.0
1513	4329	4330.0
...	...	...
439	2579	2572.0

271	2494	2518.0
244	1818	1826.0
1159	3332	3341.0
1701	2565	2559.0

651 rows × 2 columns

```
plt.figure(figsize=(16,8))
sns.lineplot(data=dtr_g)
```

<AxesSubplot:>




```
dtr.score(X_test,y_test)*100
```

99.85504139672305

XGboost regression

```
xgb_r = xg.XGBRegressor(objective ='reg:squarederror',n_estimators = 10, seed = 123)
xgb_r.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=10, n_jobs=4, num_parallel_tree=1, random_state=123,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=123,
             subsample=1, tree_method='exact', validate_parameters=1,

             verbosity=None)
```

```
xgb_pred = xgb_r.predict(X_test)
```

```
xgb_df = pd.DataFrame({'Actual': y_test, 'Predicted': xgb_pred})
xgb_df
```

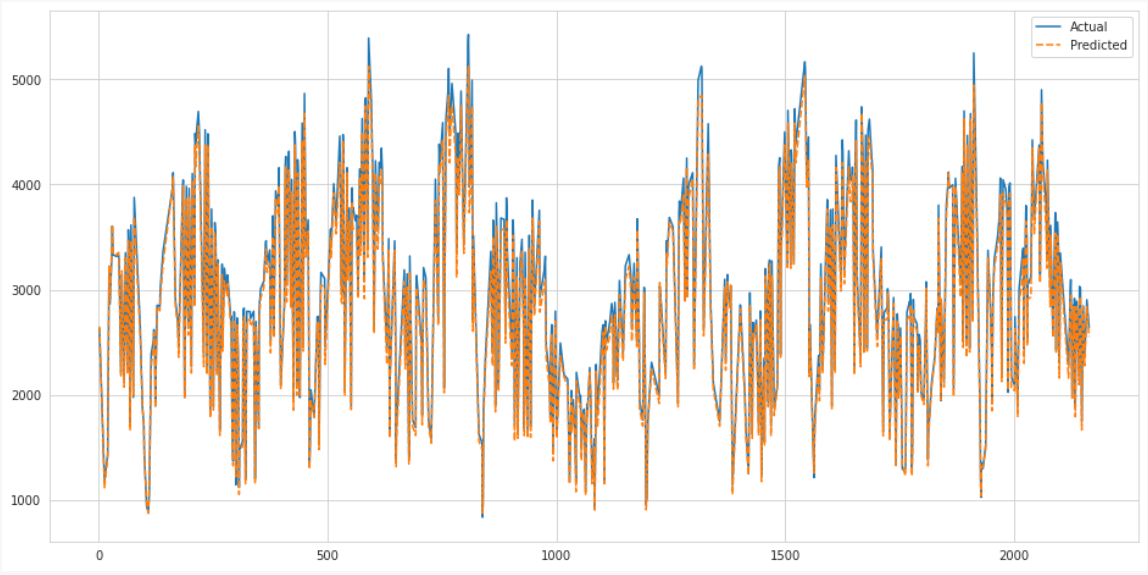
	Actual	Predicted

1486	4173	4069.691895
1602	1902	1860.709961
1460	2870	2798.565430
1134	2142	2050.101318
1513	4329	4167.435547
...	...	...
439	2579	2427.022705
271	2494	2415.062012
244	1818	1780.136475
1159	3332	3223.888916
1701	2565	2459.920410

651 rows × 2 columns

```
plt.figure(figsize=(16,8))
sns.lineplot(data=xgb_df)
```

<AxesSubplot:>




```
xgb_r.score(X_test,y_test)*100
```

98.7655882096893

Conclusion:

In conclusion, the outlined data loading and preprocessing steps provide a foundational framework for preparing a dataset for analysis in Python using the pandas library. By following these steps, you can ensure that your data is in a suitable format and quality for further exploration and visualization tasks.