WEBSITE TRAFFIC ANALYSIS

INTRODUCTION:

The goal of this project is to leverage machine learning algorithms to gain actionable insights from the data. By harnessing predictive analytics, businesses can anticipate user behavior, detect trends, and optimize website performance. This process aids in making informed decisions to enhance user experiences, boost conversions, and achieve organizational objectives, such as increasing revenue and audience engagement.

OBJECTIVE:

1. Traffic Sources: Analyzing where your web traffic is coming from is crucial. It could be from search engines, social media, referral sites, or direct visits. Understanding this helps you allocate resources effectively.

2. Visitor Demographics: Knowing the demographics of your website visitors (age, gender, location, etc.) can help tailor your content and marketing efforts to your target audience.

3. User Behavior: Tracking user behavior includes understanding which pages they visit, how long they stay, and what actions they take on your site (e.g., signing up for newsletters, making purchases).

4. Bounce Rate: Bounce rate measures the percentage of visitors who leave your site after viewing only one page. A high bounce rate may indicate issues with your site's content or user experience.

5. Conversion Rate: This metric indicates the percentage of visitors who complete a desired action on your site, such as making a purchase or signing up for a newsletter. Improving the conversion rate is often a primary objective.

6. Page Load Times: Slow-loading pages can lead to high bounce rates and dissatisfied visitors. Analyzing page load times and optimizing them is essential.

7. Content Performance: Identifying which types of content perform best (e.g., blog posts, videos, infographics) can help you create more of what resonates with your audience.

DESIGN THINKING PROCESS:

1. Empathize: This stage involves understanding the needs, thoughts, and feelings of the people you are designing for. It often starts with user research, interviews, and observations to gain empathy and insight into their experiences.

2. Define: In this stage, you define the problem you are trying to solve based on the insights gathered during the empathize stage. It's about framing the problem in a way that guides your design efforts.

3. Ideate: Here, you generate a wide range of creative ideas to address the defined problem. It's a brainstorming phase where no idea is considered too wild, and the focus is on quantity and diversity of ideas.

4. Prototype: This stage involves creating tangible representations of your ideas. These can be in the form of sketches, mock-ups, or even simple prototypes to test and experiment with various solutions.

5. Test: Testing is about gathering feedback on your prototypes and ideas from real users. It helps you understand what works, what doesn't, and how to refine and improve your design. This stage often leads back to the empathize stage as you learn more about user needs and iterate on your design.

DEVELOPMENT PHASE:

1. Requirements Gathering: In this phase, the development team works with stakeholders to gather and document the project's requirements. This involves understanding the objectives, functionality, and constraints of the software to be developed.

2. Planning and Design: Once requirements are clear, the team plans the project. This includes defining the project scope, creating a timeline, allocating resources, and designing the system architecture. The design phase involves creating detailed specifications and system blueprints.

3. Implementation (Coding): During this phase, developers write the actual code for the software, following the design specifications. This is where the software is built and its features are implemented.

4. Testing: After coding, the software undergoes testing to identify and fix any bugs or issues. This includes unit testing (testing individual components), integration testing (testing how components work together), and user acceptance testing (ensuring it meets user requirements).

5. Deployment and Maintenance: Once the software is tested and ready, it's deployed to the production environment for actual use. Maintenance includes ongoing support, updates, and bug fixes to ensure the software remains functional and up to date.

OBJECTIVIES FOR DATA ANALYSIS:

1. Insight Generation: The primary objective of data analysis is to generate meaningful insights from the data that can inform decision-making, strategy, or problem-solving.

2. Pattern Recognition: Data analysis aims to identify patterns, trends, and correlations within the data to help uncover hidden relationships or dependencies.

3. Anomaly Detection: It helps in identifying outliers or anomalies within the data, which could be errors or exceptional cases requiring special attention.

4. Performance Evaluation: Data analysis can be used to evaluate the performance of a system, process, or product by comparing it against predefined metrics or benchmarks.

5. Predictive Modeling: Data analysis can support predictive modeling to forecast future trends or outcomes, allowing organizations to plan and adapt accordingly.

DATA COLLECTION PROCESS:

1. Define Objectives: Start by clearly defining the objectives and goals of data collection. What specific information do you need to collect, and what will it be used for?

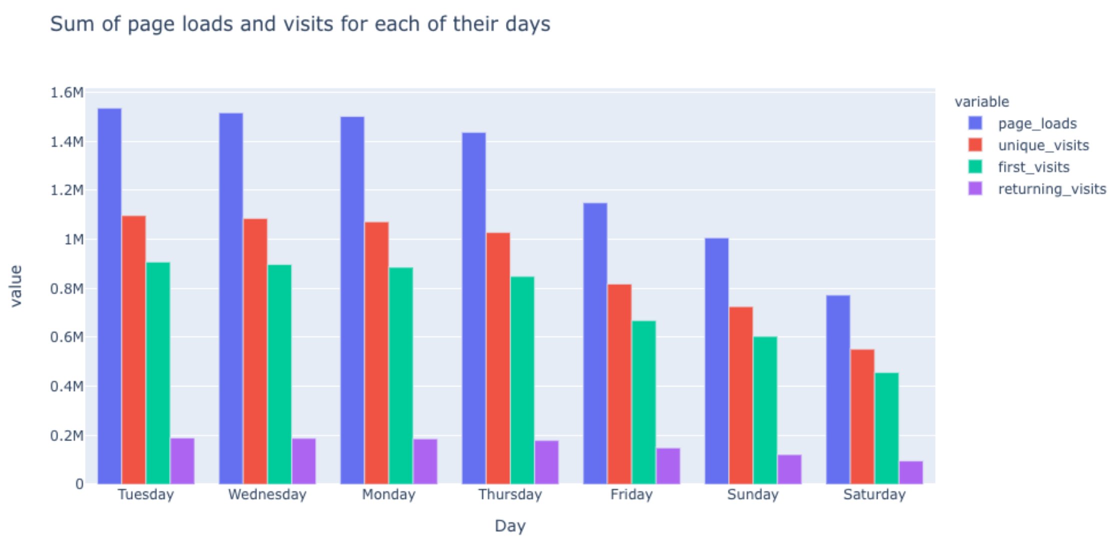
2. Select Data Sources: Identify the sources of data, which could include surveys, sensors, databases, web scraping, or any other relevant means. Ensure that the data sources align with your objectives.

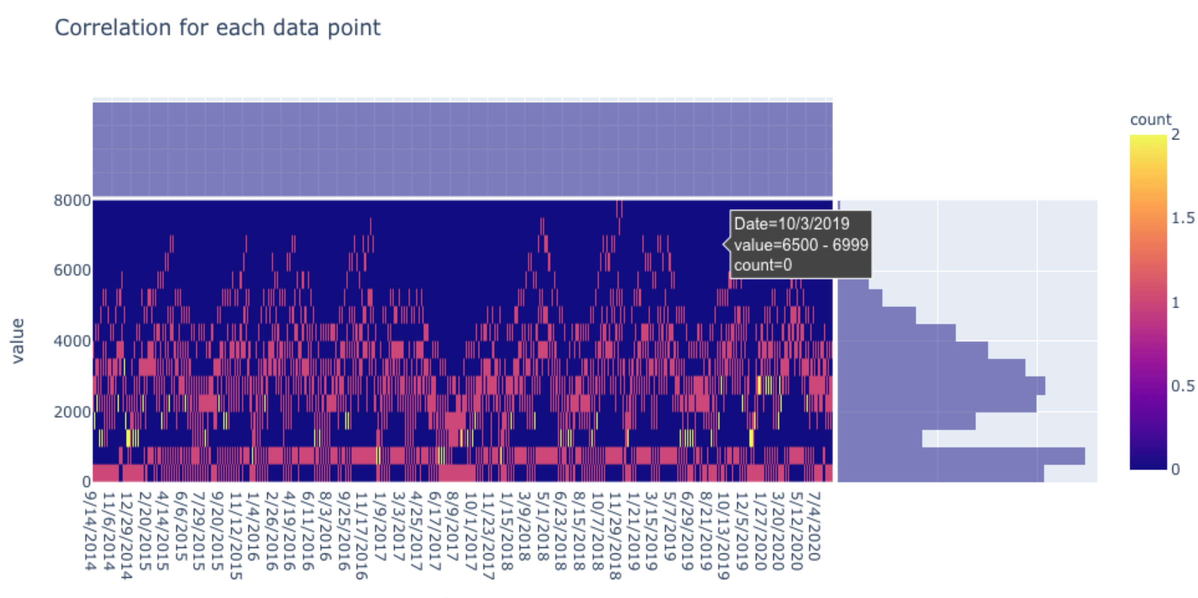
3. Data Collection Methods: Choose the appropriate data collection methods, whether it's through surveys, interviews, observations, automated data capture, or a combination of methods.

4. Data Quality Assurance: Implement measures to ensure data accuracy and reliability. This may involve data validation, cleaning, and error correction.

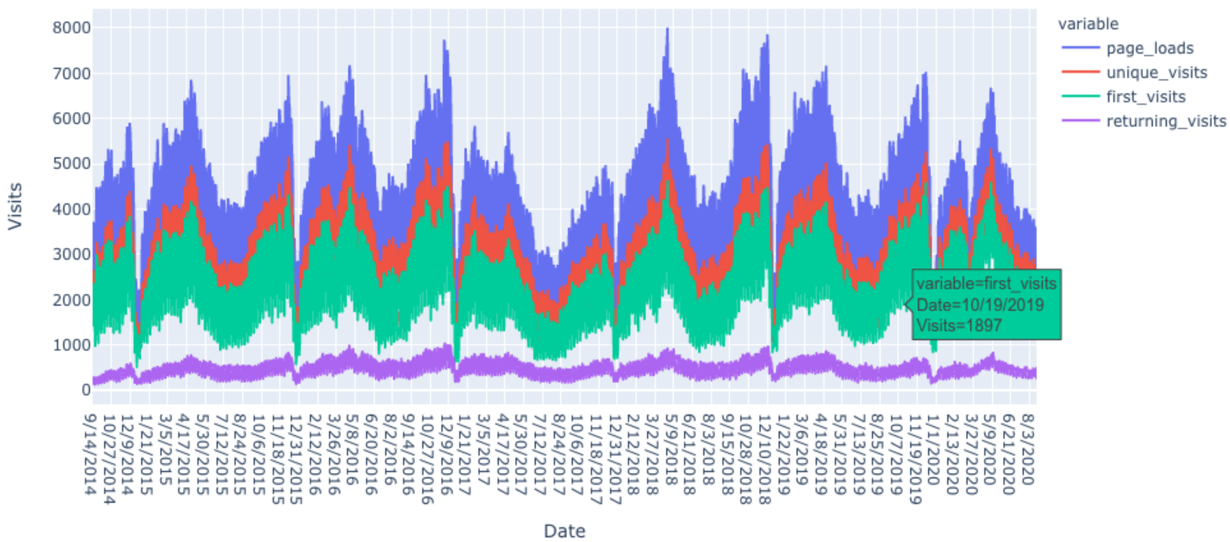
5. Ethical and Legal Considerations: Be mindful of privacy and legal issues related to data collection. Ensure compliance with data protection regulations and ethical guidelines, especially when dealing with sensitive data.

DATA VISUALIZATION:

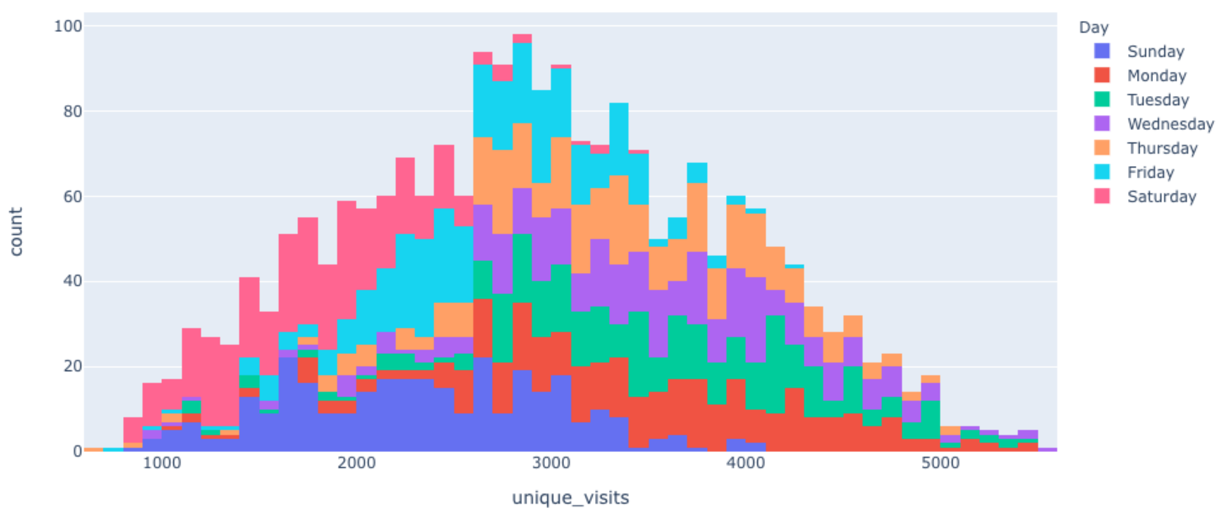


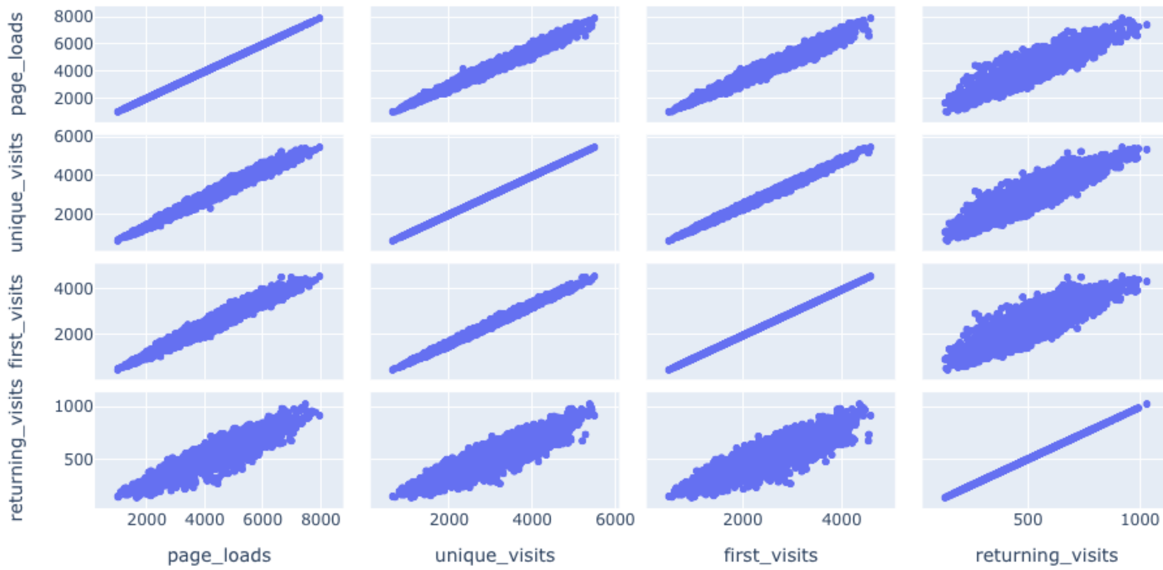


Page Loads & visitors over Time



unique visits for each day





PYTHON CODE INTEGRATION:

```
In [1]:
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()
```

```python
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]:

```python
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]:

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

9/14/201410/27/201412/9/20141/21/20153/5/20154/17/20155/30/2015 7/12/20158/24/201510/6/201511/18/201512/31/20152/12/20163/26/20 165/8/20166/20/20168/2/20169/14/201610/27/201612/9/20161/21/201 73/5/20174/17/20175/30/20177/12/20178/24/201710/6/201711/18/201 712/31/20172/12/20183/27/20185/9/20186/21/20188/3/20189/15/2018 10/28/201812/10/20181/22/20193/6/20194/18/20195/31/20197/13/201 98/25/201910/7/201911/19/20191/1/20202/13/20203/27/20205/9/2020 6/21/20208/3/202010002000300040005000600070008000

variablepage_loadsunique_visitsfirst_visitsreturning_visitsPage Loads & visitors over TimeDateVisits

This histogram plot represent the sum of unique visits for each day in the week against count of unique visits for each day in the week.

but from this plot it's hard to estimate which day had the most unique visitors, so we will explore more deeper.

In [7]:

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

10002000300040005000020406080100

DaySundayMondayTuesdayWednesdayThursdayFridaySaturdayunique visits for each dayunique_visitscount

With this bar plot it is clear that tuesday, wednesday, monday and thursday are the days in a week when extensive amount of traffic come to this website

In [8]:

```
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')
```

TuedayWednesdayMondayThursdayFridaySundaySaturday00.2M0.4M0.6M0.8M1M

variablesumSum of Unique visits for each dayDaysum of unique visits

sum of unique visits for each week day over time series, we know which days get the most traffic but on what time intervals ? this graph answers to that question.

time intervals are grouped according to their relation with unique visits and days, now we can understand that in which days, months and years did the website get the most traffic.

In [9]:

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

9/14/20146/28/20154/10/20161/22/201711/5/20178/19/20186/2/20193/15/20201/19/201511/2/20158/15/20165/29/20173/12/201812/24/201810/7/20197/20/20205/26/20153/8/201612/20/201610/3/20177/17/20184/30/20192/11/202012/17/20149/30/20157/13/20164/26/20172/7/201811/21/20189/4/20196/17/20204/23/20152/4/201611/17/20168/31/20176/14/20183/28/20191/9/202011/21/20149/4/20156/17/20163/31/20171

/12/201810/26/20188/9/20195/22/20204/4/20151/16/201610/29/20168/12/20175/26/20183/9/201912/21/20191000200030004000500

DaySundayMondayTuesdayWednesdayThursdayFridaySaturdaySum of unique visits for each day over TimeDatesum of unique_visits

get the sum of page_loads unique_visits first_visits returning_visits related to each of their days

In [10]:

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

Out[10]:

	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

this grouped bar chart comes from the crosstab above and it shows the sum of page_loads, unique_visits, first_visits, returning_visits for each day

In [11]:

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

TuesdayWednesdayMondayThursdayFridaySundaySaturday00.2M0.4M0.6M0.8M1M1.2M1.4M1.6M

variablepage_loadsunique_visitsfirst_visitsreturning_visitsSum of page loads and visits for each of their daysDayvalue

This is a heatmap graph that shows the correlation of each datapoint from page loads, unique_visits, first_visits , returning_visits columns, first visits seems to have a great correlation with unique visits.

The Yellow points indicate a great correlation between first visits and unique visits, but we don't how much let's find that out

In [12]:

```
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')
```

9/14/201411/6/201412/29/20142/20/20154/14/20156/6/20157/29/2015 9/20/201511/12/20151/4/20162/26/20164/19/20166/11/20168/3/20169 /25/201611/17/20161/9/20173/3/20174/25/20176/17/20178/9/201710/ 1/201711/23/20171/15/20183/9/20185/1/20186/23/20188/15/201810/7 /201811/29/20181/21/20193/15/20195/7/20196/29/20198/21/201910/1 3/201912/5/20191/27/20203/20/20205/12/20207/4/20200200040006000 8000

00.511.52countCorrelation for each data pointDatevalue

this shows the paired correlation of page_loads unique_visits first_visits returning_visits columns with annotated values we know that first visits and unique visits are correlated by 0.99 which is a great correlation and page loads have a good correlation with our target variable as well.

let's see how the correlation looks like in our next plot.

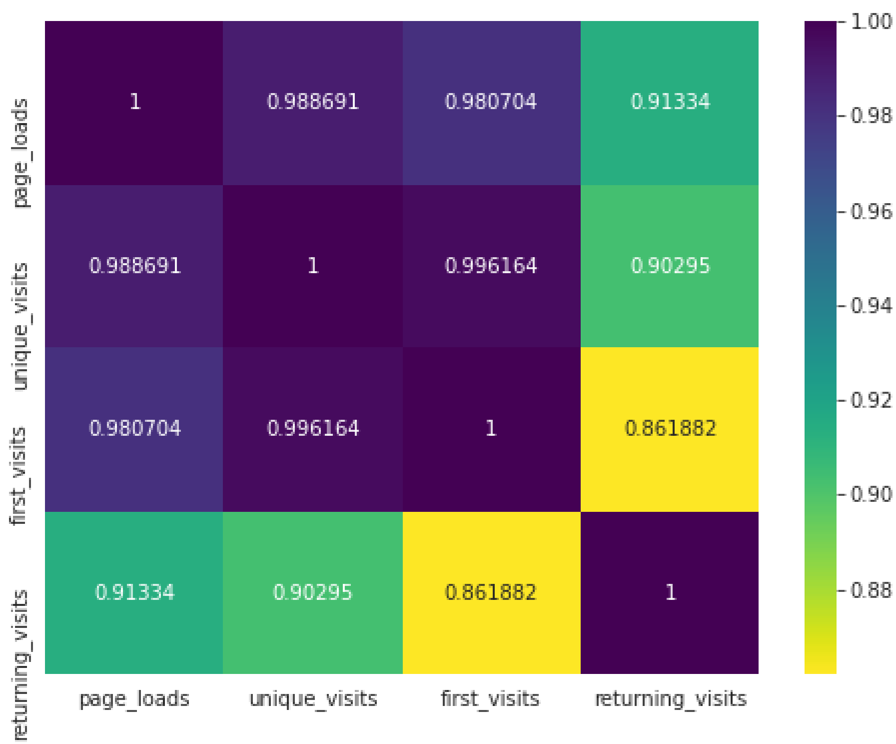
In [13]:

```
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')
```

Out[13]:

```
<AxesSubplot:>
```



this scatter matrix plot shows the paired plot of page_loads unique_visits first_visits returning_visits we can see that unique visits and first visits have a straight upward line, that means that first visits are increasing as the unique visits increase. we can also other pairs and identify their level of correlation visualy.

The last thing we need is to visualize the trend line.

In [14]:

```
px.scatter_matrix(df[['page_loads'     ,'unique_visits'     ,'first_visits'
,'returning_visits']])
```

200040006000800020004000600020004000200040006000800050010002000400060002000400050010000

page_loadsunique_visitsfirst_visitsreturning_visitspage_loadsunique_visitsfirst_visitsreturning_visits

Okay now we have the regression line pointing upward which confirms the trend between these two columns

In [15]:

```
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"
)
```

50010001500200025003000350040004500100020003000400050000

Regression line for unique visits and first visitsfirst_visitsunique_visits

there are no outliears that need to be dealt with, data is tightly packed with no dispersion except for returning visits, this column was also less correlated with our target variable.

In [16]:

```
px.violin(df,y=['page_loads'        ,'unique_visits'        ,'first_visits'
,'returning_visits'],box=True,points='all')
```

page_loadsunique_visitsfirst_visitsreturning_visits0100020003000400050006000700080009000

variablevalue

starting the feature engineering.

we only need these columns

In [17]:

```
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
```

Out[18]:

	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

In [19]:

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

In [20]:

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

Out[20]:

	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

In [21]:

```
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

In [22]:

```
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

In [23]:

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

Out[23]:

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

In [24]:

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

In [25]:

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

Out[25]:

	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

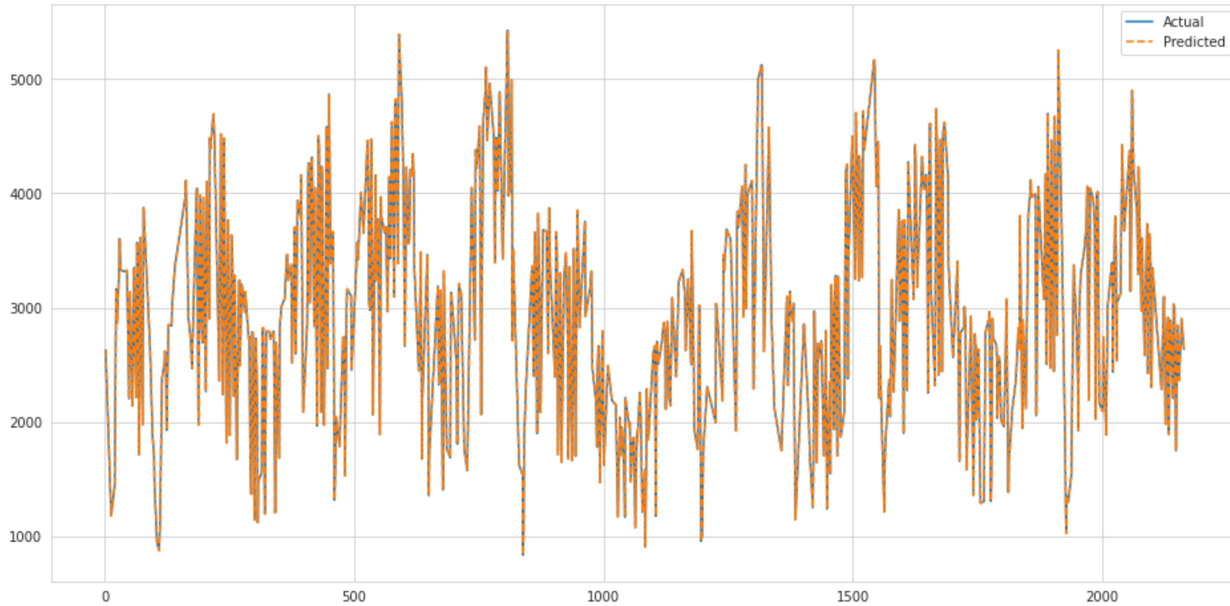
visualize the actual and predicted values

In [26]:

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

Out[26]:

```
<AxesSubplot:>
```



get the accuacy score of the model.

In [27]:

```
regressor2.score(X_test,y_test)*100
```

Out[27]:

```
100.0
```

Support Vector Regression

In [28]:

```
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.00001)
svr_rbf.fit(X_train, y_train)
```

Out[28]:

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

In [29]:

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

In [30]:

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

Out[30]:

	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

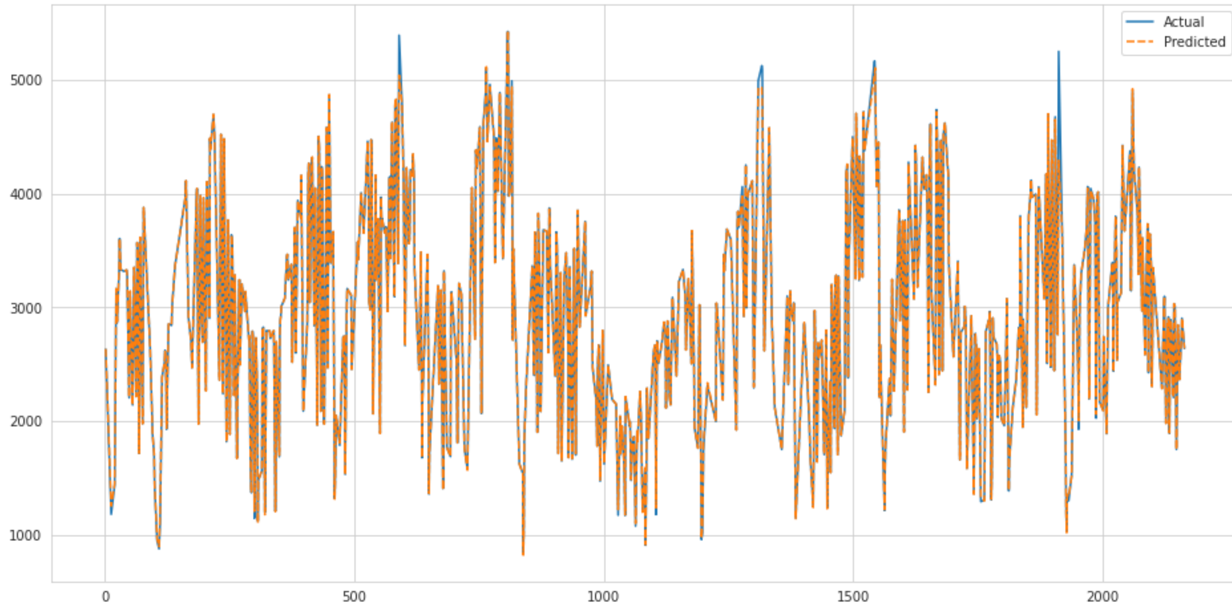
651 rows × 2 columns

In [31]:

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

Out[31]:

```
<AxesSubplot:>
```



In [32]:

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

Out[32]:

```
99.80054455767926
```

Decision Tree Regression

In [33]:

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

Out[33]:

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

In [34]:

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

In [35]:

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

Out[35]:

	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

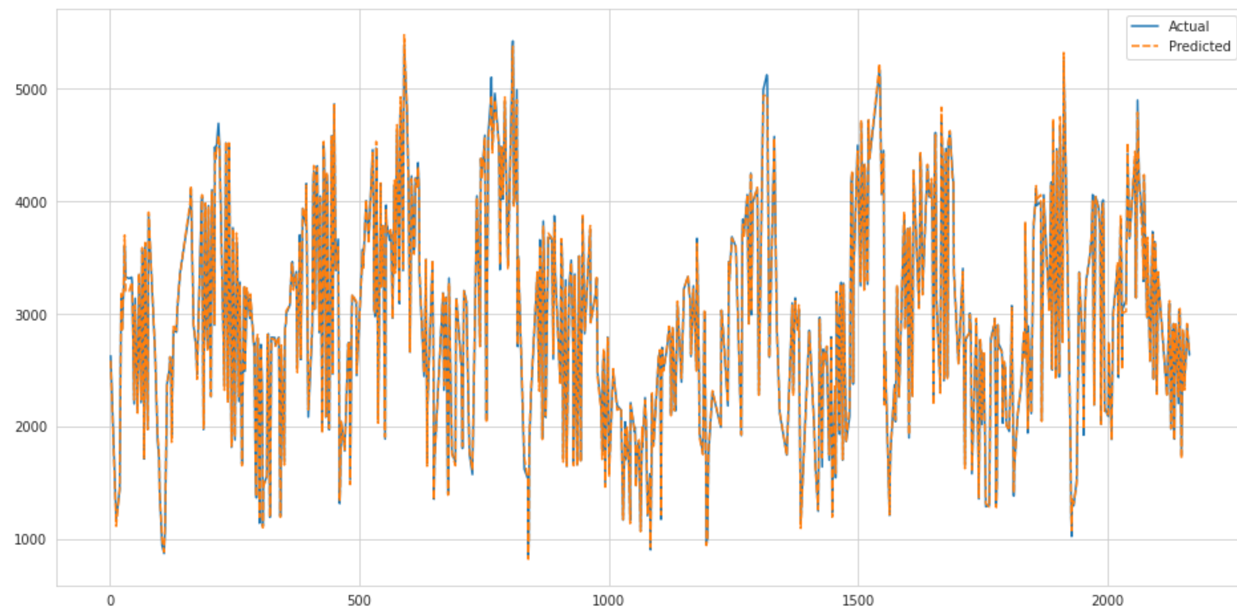
651 rows × 2 columns

In [36]:

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

Out[36]:

```
<AxesSubplot:>
```



In [37]:

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

Out[37]:

```
99.85504139672305
```

XGboost regression

In [38]:

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

Out[38]:

```
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)
```

In [39]:

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

In [40]:

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

Out[40]:

	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

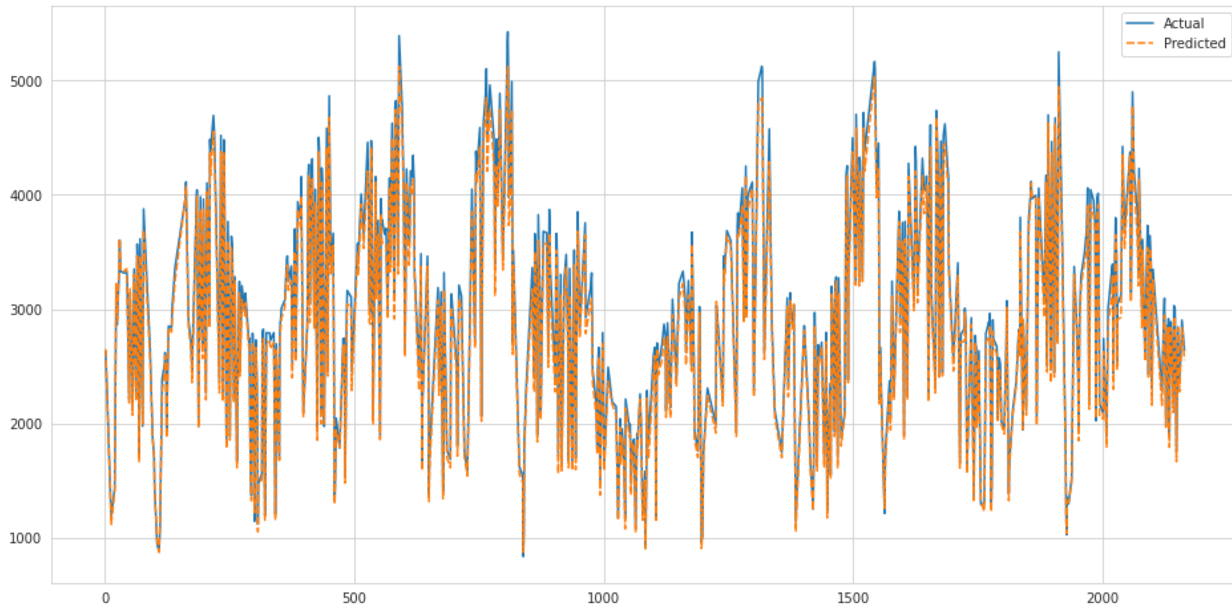
651 rows × 2 columns

In [41]:

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

Out[41]:

<AxesSubplot:>



In [42]:

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

Out[42]:

98.7655882096893