**WEBSITE TRAFFIC ANALYSIS PROJECT PHASE 4-DOCUMENTATION**

**ARUN P**

**PHASE 4:**DEVELOPMENT PART 2

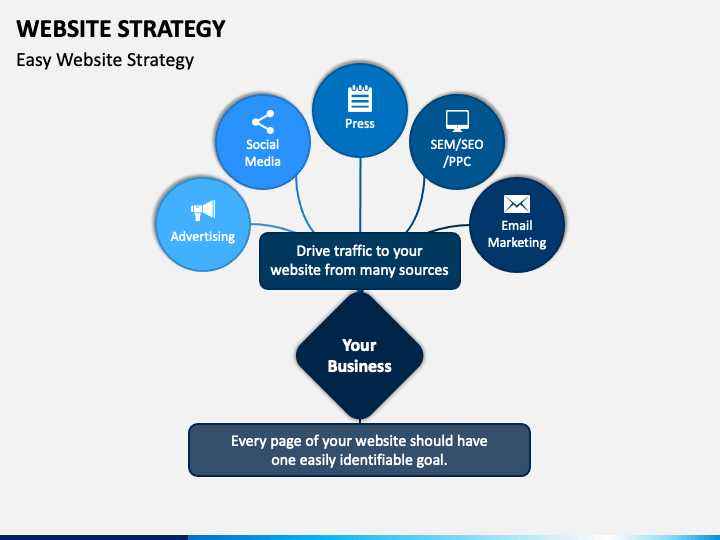
**AIM:**

To build website traffic analysis with the integration of IBM Cognos for interactive dashboards and Python for advanced analysis. Website traffic analysis is to understand, optimize, and enhance website performance. This involves examining user behavior and patterns to make data-driven decisions for improved user experience and achieving specific goals.

Creating a comprehensive website traffic analysis involves several steps, and it typically requires a combination of tools and technologies. Here's a general guide on how you can approach this project:

**1. DATA COLLECTION AND STORAGE:**

* Ensure you have a robust data collection mechanism in place. Common tools for web analytics include Google Analytics, Adobe Analytics, or your own custom tracking system.
* Store the collected data in a structured format, such as a relational database, data warehouse, or flat files.



**2. DATA PREPROCESSING:**

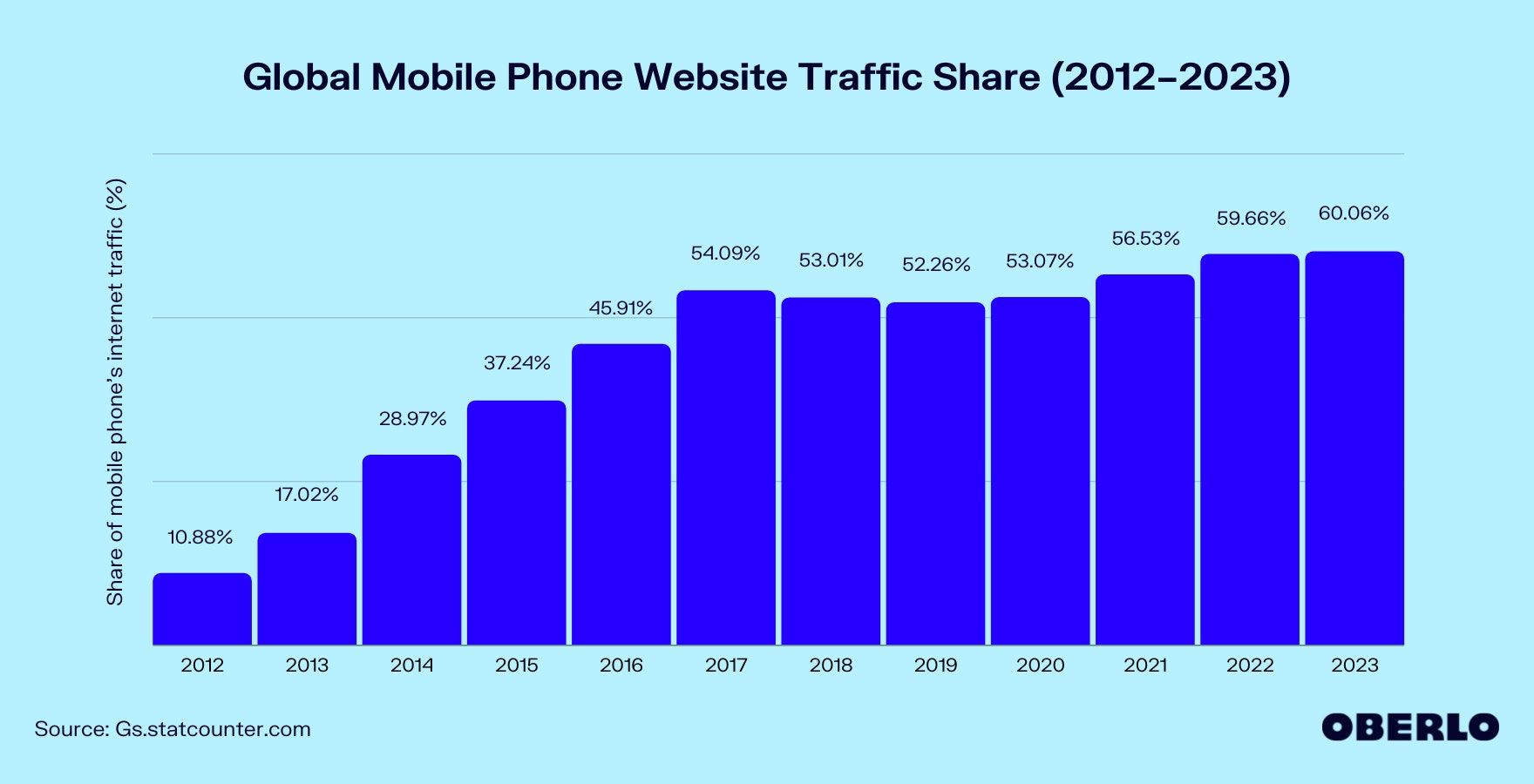
* Extract the relevant data from your data source.
* Preprocess and clean the data, handling missing values, outliers, and data quality issues.
* Convert the data into a format that can be easily imported into IBM Cognos and Python (e.g., CSV, Excel, or a database).

**3. IBM Cognos INTEGRATION:**

* Import the preprocessed data into IBM Cognos.
* Use Cognos to create interactive dashboards and reports. You can follow these steps:
* Define data connections: Configure connections to your data sources.
* Create data modules: Build data models to represent your website traffic data.
* Design interactive dashboards: Create visualizations and reports that display key metrics, such as popular pages, traffic sources, and user engagement.
* Add interactivity: Use filters, drill-through, and prompts to allow users to explore the data.
* Schedule and share reports: Set up automated report delivery and distribution.

**4. PYTHON INTEGRATION:**

* Use Python to perform more advanced analysis and generate additional insights from the data.
* Import the necessary libraries, such as Pandas, Matplotlib, and any machine learning libraries you plan to use.
* Perform various analyses based on your objectives:
* Time Series Analysis: Analyze trends and seasonality in website traffic over time.
* User Segmentation: Segment users based on demographics, behavior, or other criteria.
* Machine Learning: Build predictive models for things like user behavior, click-through rates, or user retention.
* Create visualizations using Matplotlib or more specialized libraries like Seaborn or Plotly to enhance your analysis.



**5. COMBINE COGNOS AND PYTHON OUTPUTS:**

* Combine insights from IBM Cognos dashboards and Python analysis to get a holistic view of your website traffic.
* You can use Python to perform in-depth analyses and then import the results into IBM Cognos for inclusion in your dashboards.

**6. AUTOMATION AND MONITORING:**

* Consider setting up automated processes for data extraction, preprocessing, and report generation.
* Continuously monitor the performance of your website and the accuracy of your analysis to ensure your insights remain relevant.

**7. DOCUMENTATION AND KNOWLEDGE SHARING:**

* Document your data sources, analysis methods, and insights for future reference.
* Share your findings with relevant stakeholders within your organization.

# **Python Libraries**

Python libraries are collections of modules that contain useful codes and functions, eliminating the need to write them from scratch. There are tens of thousands of Python libraries that help machine learning developers, as well as professionals working in data science, data visualization, and more.

### 1. NumPy

NumPy is a popular Python library for multi-dimensional array and matrix processing because it can be used to perform a great variety of mathematical operations. Its capability to handle linear algebra, Fourier transform, and more, makes NumPy ideal for machine learning and artificial intelligence (AI) projects, allowing users to manipulate the matrix to easily improve machine learning performance. NumPy is faster and easier to use than most other Python libraries.

### 2. Scikit-learn

Scikit-learn is a very popular machine learning library that is built on NumPy and SciPy. It supports most of the classic supervised and unsupervised learning algorithms, and it can also be used for data mining, modeling, and analysis. Scikit-learns simple design offers a user-friendly library for those new to machine learning.

### 3. Pandas

Pandas is another Python library that is built on top of NumPy, responsible for preparing high-level data sets for machine learning and training. It relies on two types of data structures, one-dimensional (series) and two-dimensional (DataFrame). This allows Pandas to be applicable in a variety of industries including finance, engineering, and statistics. Unlike the slow-moving animals themselves, the Pandas library is quick, compliant, and flexible.

### 4.Matplotlib

Matplotlib is a Python library focused on data visualization and primarily used for creating beautiful graphs, plots, histograms, and bar charts. It is compatible for plotting data from SciPy, NumPy, and Pandas. If you have experience using other types of graphing tools, Matplotlib might be the most intuitive choice for you.

# Preprocess data:

This file contains 5 years of daily time series data for several measures of traffic on a statistical forecasting teaching notes website whose alias is statforecasting.com. The variables have complex seasonality that is keyed to the day of the week and to the academic calendar. The patterns you see here are similar in principle to what you would see in other daily data with day-of-week and time-of-year effects. Some good exercises are to develop a 1-day-ahead forecasting model, a 7-day ahead forecasting model, and an entire-next-week forecasting model (i.e., next 7 days) for unique visitors.

We take 2 attributes is the Row represents for the day count, and the Unique Visit for the number of people visit the website

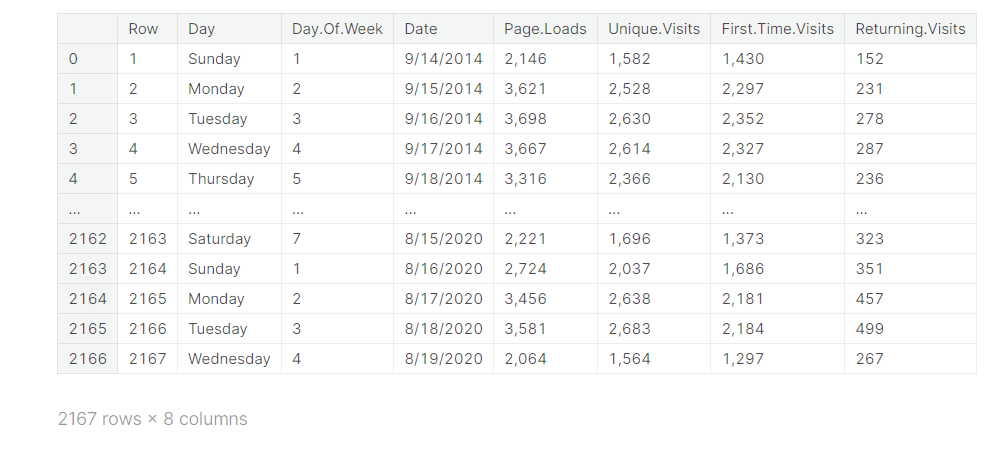
**In [1]:** import numpy as np  
 import pandas as pd  
 import matplotlib.pyplot as plt

**In [2]:** my\_data = pd.read\_csv("/kaggle/input/daily-website-visitors/daily- website-visitors.csv", delimiter=",")

Displaying the website traffic data:

**In [3]:** my\_data

Out [1]:



**In [4]:** my\_data["Page.Loads"].map(lambda x: float(x.replace(",", "")))

Out [4]:

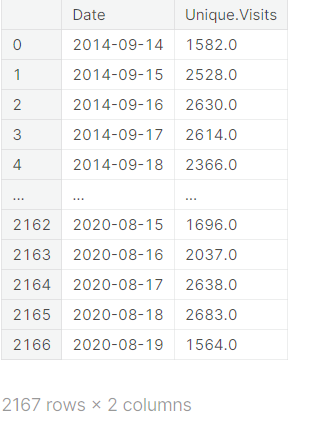
0 2146.0  
1 3621.0  
2 3698.0  
3 3667.0  
4 3316.0  
 ...   
2162 2221.0  
2163 2724.0  
2164 3456.0  
2165 3581.0  
2166 2064.0  
Name: Page.Loads, Length: 2167, dtype: float64

**In [5]:**

X = (my\_data["Unique.Visits"].map(lambda x: float(x.replace(",", ""))))  
 # X = my\_data["Unique.Visits"]

**In [6]:** X1 = pd.to\_datetime(my\_data["Date"])  
 X2 = pd.concat([X1, X], axis = 1)  
 X2

Out [6]:



**In [7]:**

X3 = pd.read\_csv("/kaggle/input/daily-website-visitors/daily-website-visitors.csv", index\_col = "Date", parse\_dates = True)  
X3 = X3["Unique.Visits"].map(lambda x: float(x.replace(",", "")))  
X3

Out [7]:

Date  
2014-09-14 1582.0  
2014-09-15 2528.0  
2014-09-16 2630.0  
2014-09-17 2614.0  
2014-09-18 2366.0  
 ...   
2020-08-15 1696.0  
2020-08-16 2037.0  
2020-08-17 2638.0  
2020-08-18 2683.0  
2020-08-19 1564.0  
Name: Unique.Visits, Length: 2167, dtype: float64

**In [8]:**

X2.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2167 entries, 0 to 2166  
Data columns (total 2 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Date 2167 non-null datetime64[ns]  
 1 Unique.Visits 2167 non-null float64   
dtypes: datetime64[ns](1), float64(1)  
memory usage: 34.0 KB

**In [9]:** X

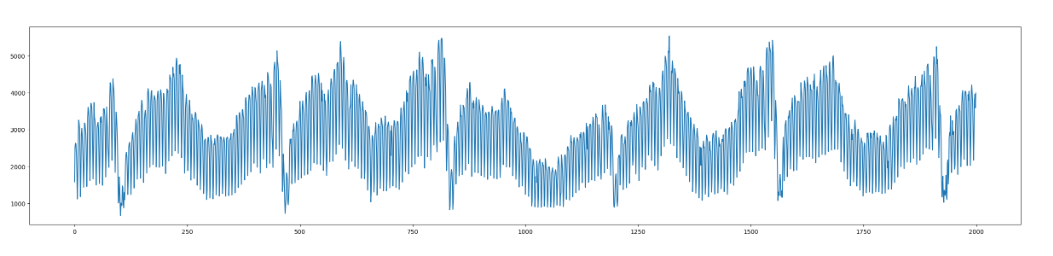
Out [9]:

0 1582.0  
1 2528.0  
2 2630.0  
3 2614.0  
4 2366.0  
 ...   
2162 1696.0  
2163 2037.0  
2164 2638.0  
2165 2683.0  
2166 1564.0  
Name: Unique.Visits, Length: 2167, dtype: float64

**In [10]:**

x = range(0,2000)  
y = X[0:2000]  
plt.figure(figsize=(30,6))  
plt.plot(x,y)  
plt.show()

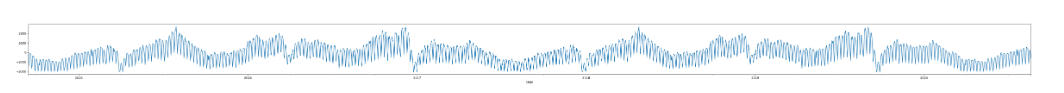
Out [10]:



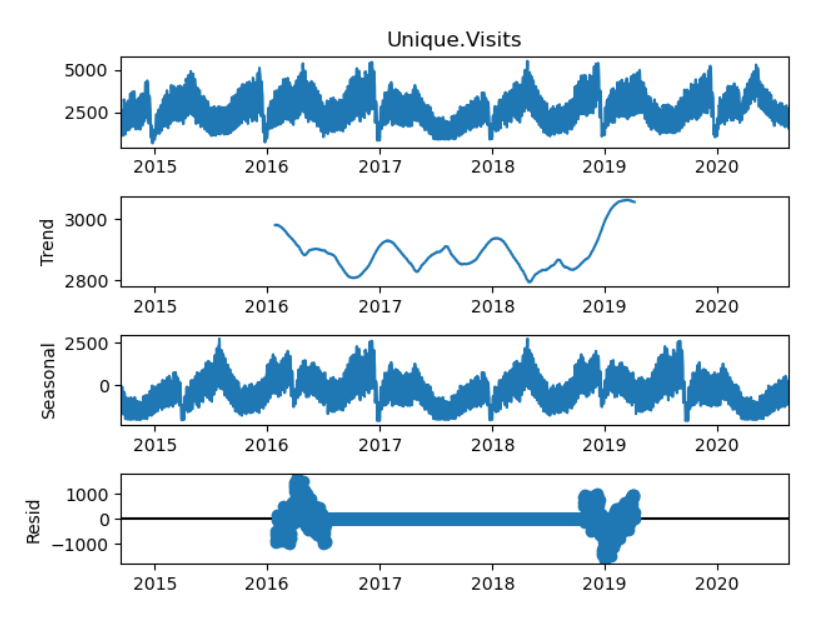
**In [11]:** import statsmodels.tsa.seasonal as statseason

**In [12]:**

plt.figure(figsize=(60,3))  
season = statseason.seasonal\_decompose(X3, model = "additive", period = 1000)  
season.seasonal.plot()  
plt.show()



**In [13]:** season.plot()  
 plt.show()



# Stationarity Testing:

**In [14]:** import statsmodels.tsa.stattools as stattools

In [15]:

adf\_test = stattools.adfuller(X)  
# Print the results  
print(adf\_test)

(-4.475968574445399, 0.00021726409300080682, 26, 2140, {'1%': -3.4334094211542983, '5%': -2.8628915360971003, '10%': -2.5674894918770197}, 29336.11247026125)

**In [16]:**

*# ADF Test*  
*result* = stattools.adfuller(X, autolag='AIC')  
#Extracting the values from the results:

print('ADF Statistic: **%f**' % result[0])  
print('p-value: **%f**' % result[1])  
print('Critical Values:')  
for key, value **in** result[4].items():  
 print('**\t%s**: **%.3f**' % (key, value))  
if result[0] < result[4]["5%"]:  
 print ("Reject H0 - Time Series is Stationary")  
else:  
 print ("Failed to Reject Ho - Time Series is Non-Stationary")

ADF Statistic: -4.475969  
p-value: 0.000217  
Critical Values:  
 1%: -3.433  
 5%: -2.863  
 10%: -2.567  
Reject H0 - Time Series is Stationary

# Seasonality Testing:

**In [17]:** X

Out [17]:

0 1582.0  
1 2528.0  
2 2630.0  
3 2614.0  
4 2366.0  
 ...   
2162 1696.0  
2163 2037.0  
2164 2638.0  
2165 2683.0  
2166 1564.0  
Name: Unique.Visits, Length: 2167, dtype: float64

**CONCLUSION**:

Website traffic analysis is to boost user engagement, conversion rates, and overall site performance, contributing to the success of the website and the achievement of its objectives. The above analysis details help in many fields to enlarge their business by knowing the interest of people.