**Website Traffic Analysis with IBM Cognos**

**P.BAKKIYAM**

**Objective:**

In this phase of the project, we will combine the strengths of IBM Cognos for creating interactive dashboards and reports to display key insights related to popular web pages, traffic sources, and user engagement metrics. Concurrently, we will leverage Python's Pandas and Matplotlib libraries to conduct advanced data analysis, including time series analysis and user segmentation. Additionally, we will explore machine learning-based predictions to gain a deeper understanding of our website's user behavior, ultimately enhancing decision-making and user experience.

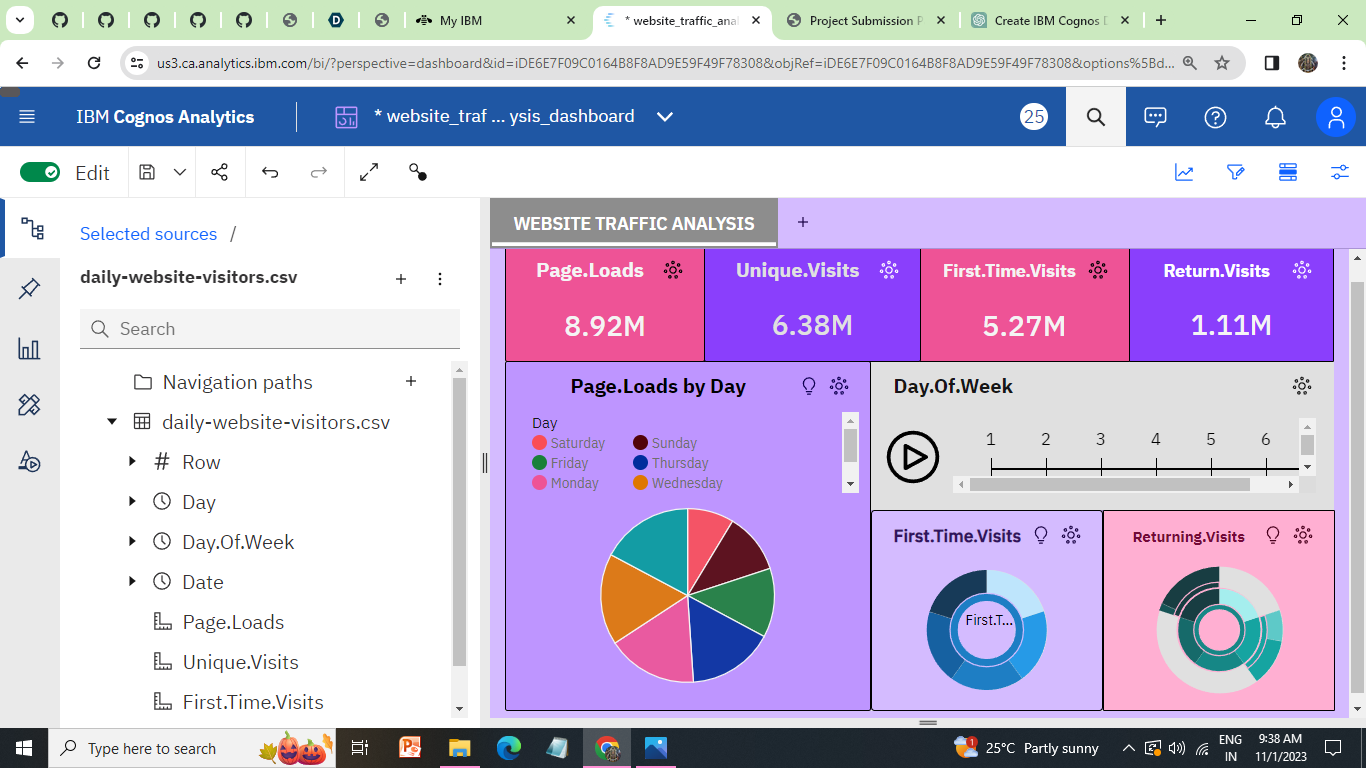
**Table of Contents:**

* **Dashboard and report creation using ibm cognos**
* **Analysis with Python**
  + Linear and non linear relationship
  + Linear Regression
  + User\_segmentation Analysis
  + Time\_series\_Decompostion
* **Making ML model**
* Split the Data into Training and Testing Sets
* Train the ARIMA Model
* Make Predictions on the Testing Set
* Evaluate the Model

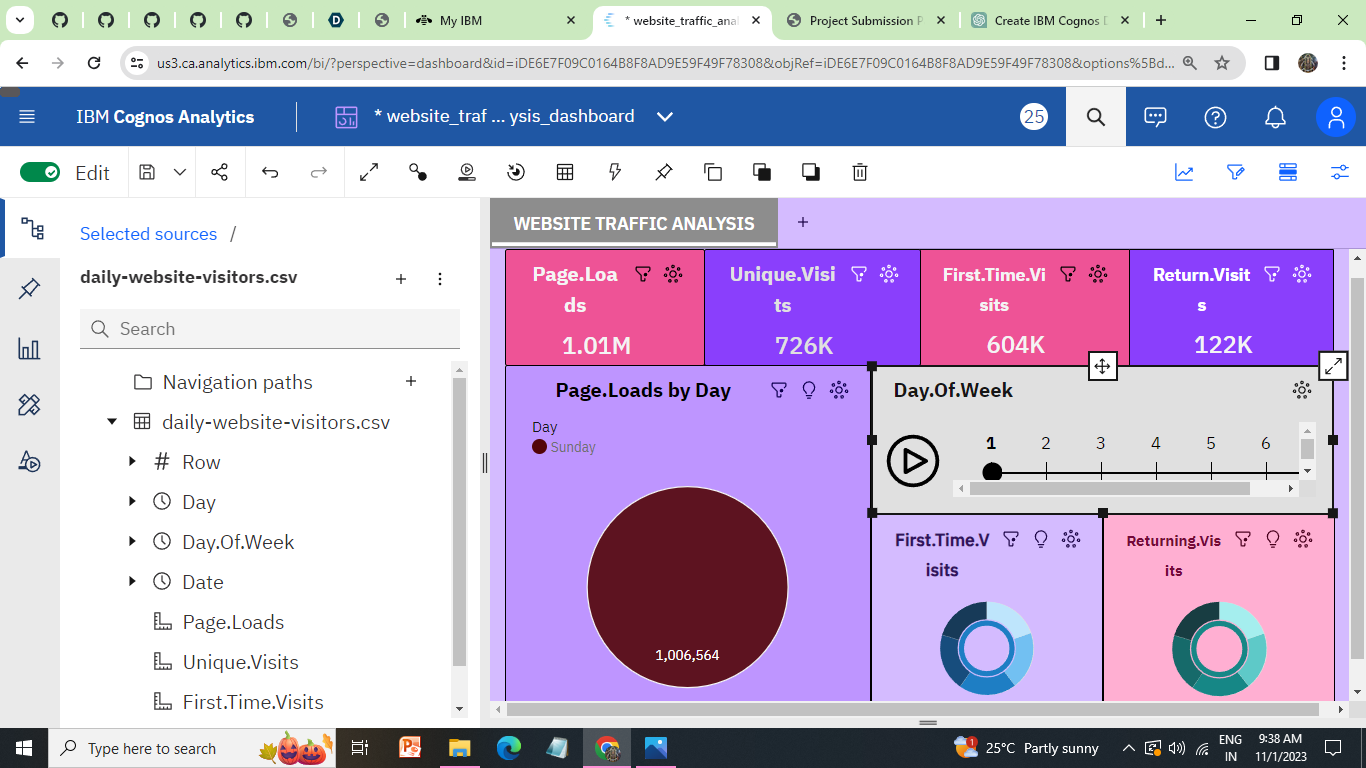
**Dashboard and Report Creation:**

In IBM Cognos, we can create interactive dashboards and detailed reports that offer a comprehensive view of our data. Dashboards provide at-a-glance insights with interactive visualizations, allowing users to monitor key metrics and drill down into details. Reports, on the other hand, offer in-depth analysis and documentation with tables, charts, and graphs, providing a more comprehensive understanding of our data. By combining both dashboards and reports, we can effectively cater to different user needs, from quick data exploration in dashboards to detailed analysis in reports, all within the same analytics platform.

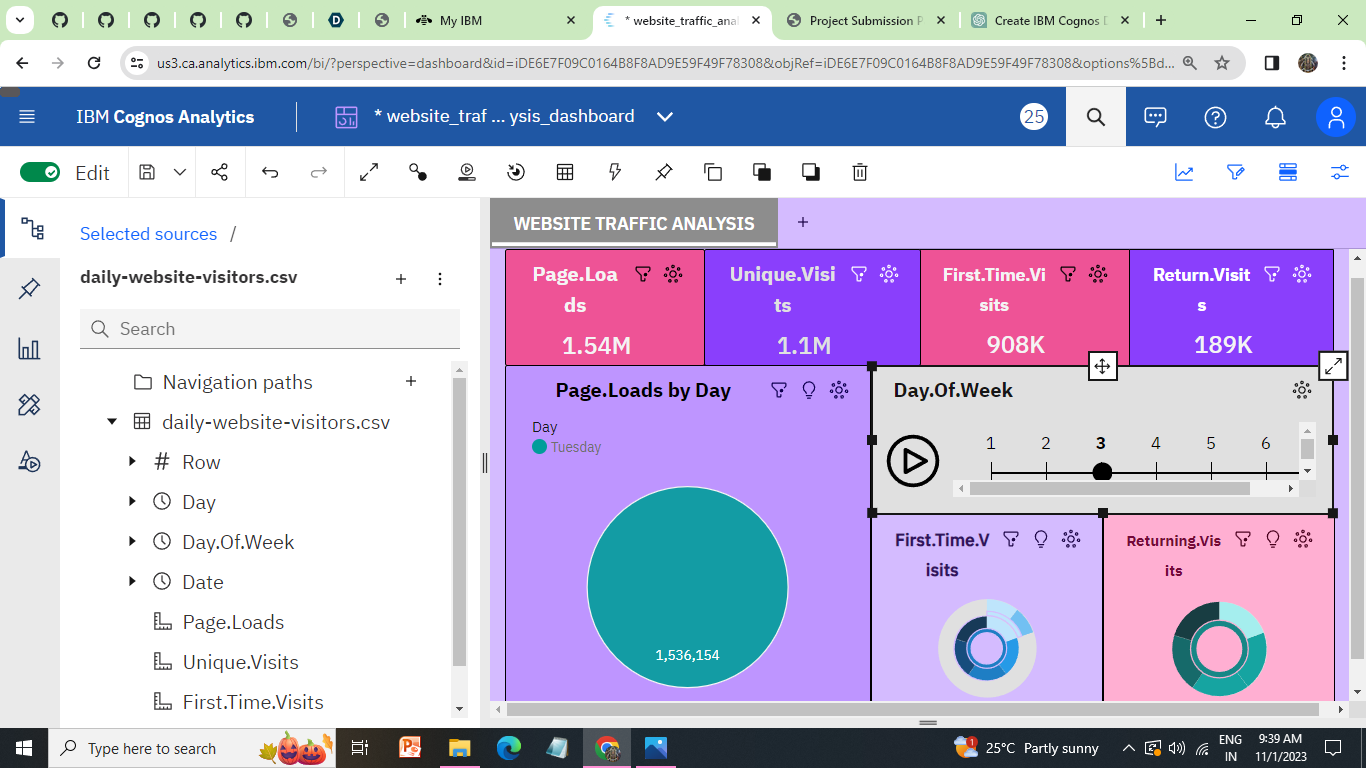
**Interactive Dashboard:**

****

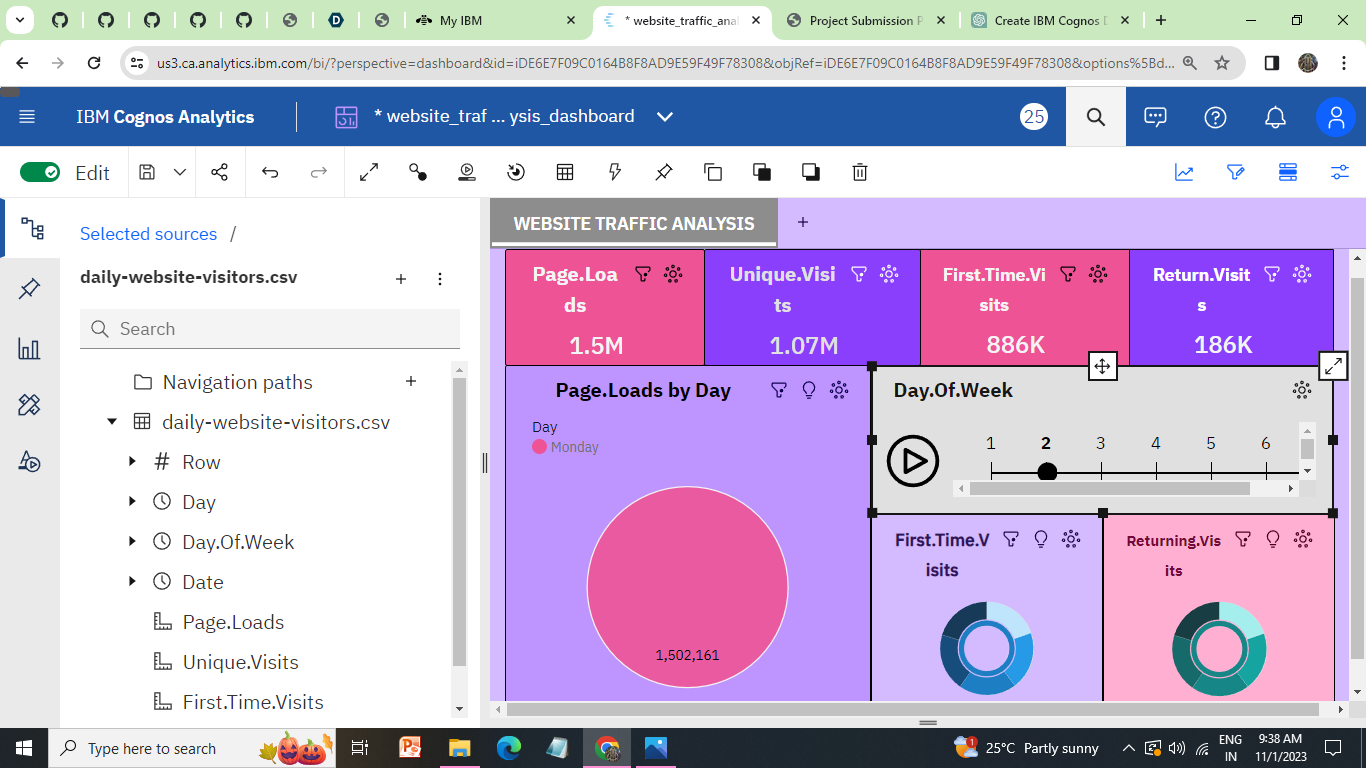
**SUNDAY:**

****

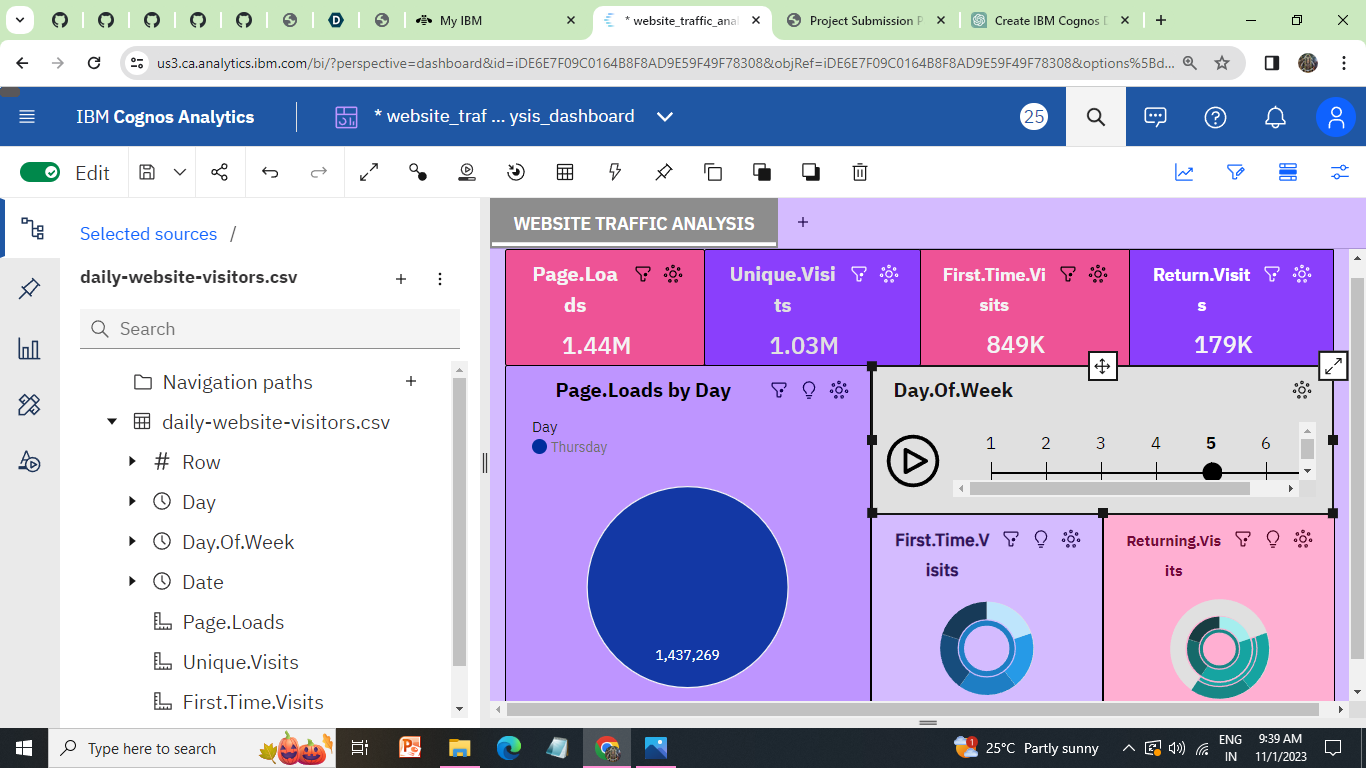
**MONDAY**

****

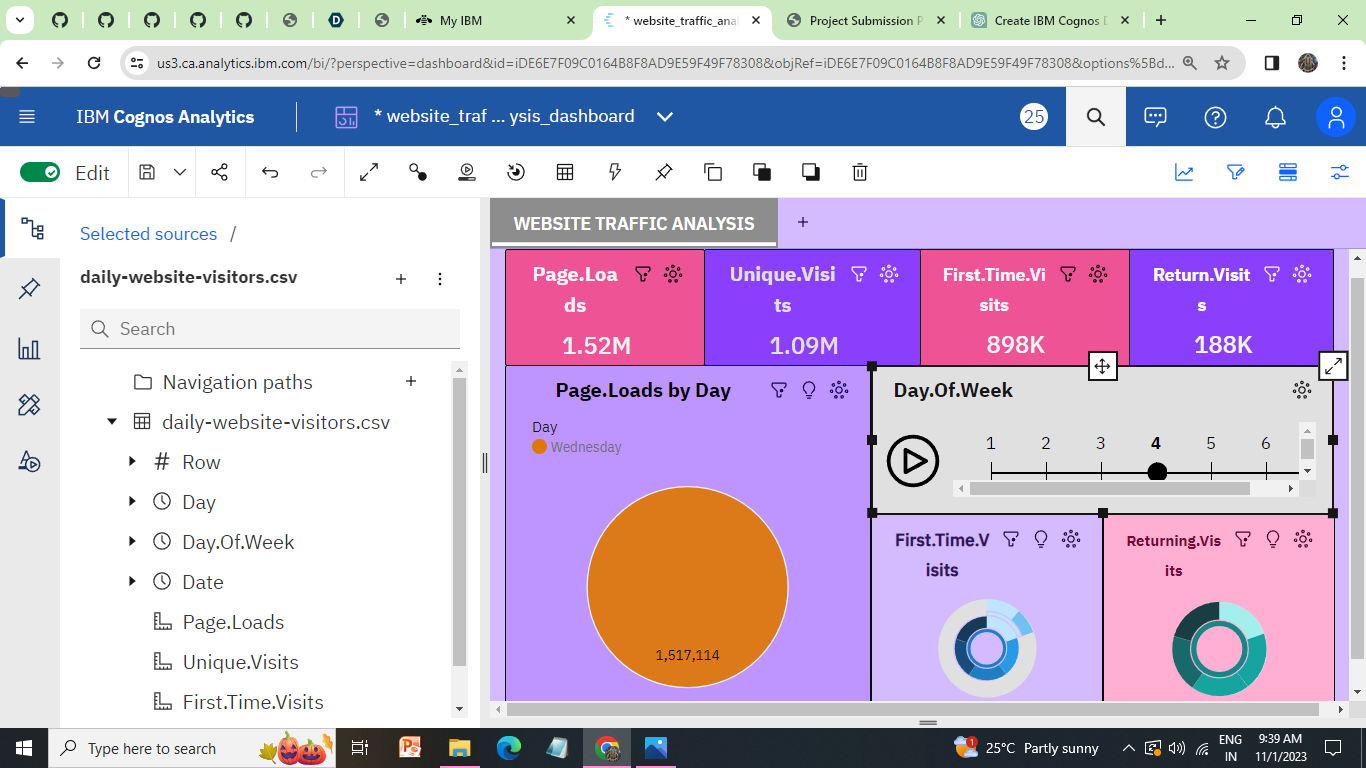
**TUESDAY**

****

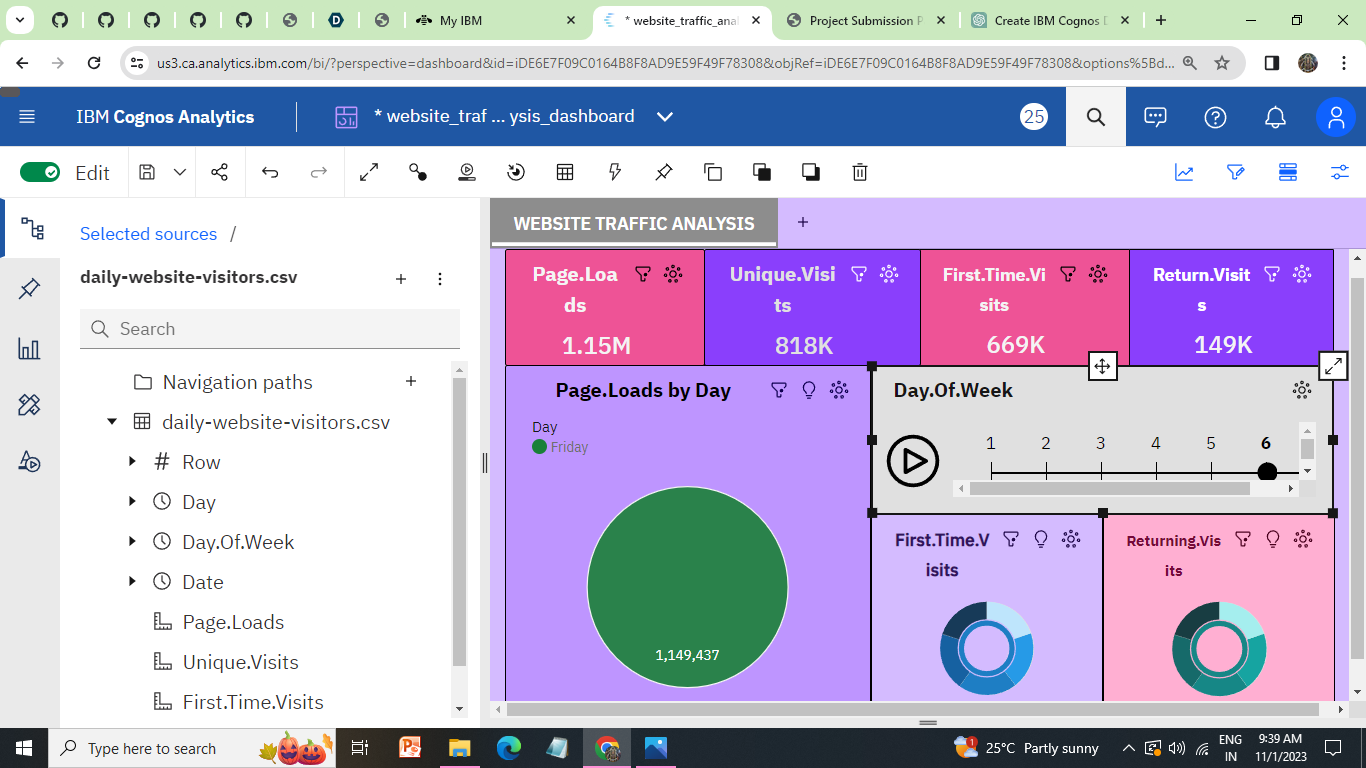
**WEDNESDAY**

****

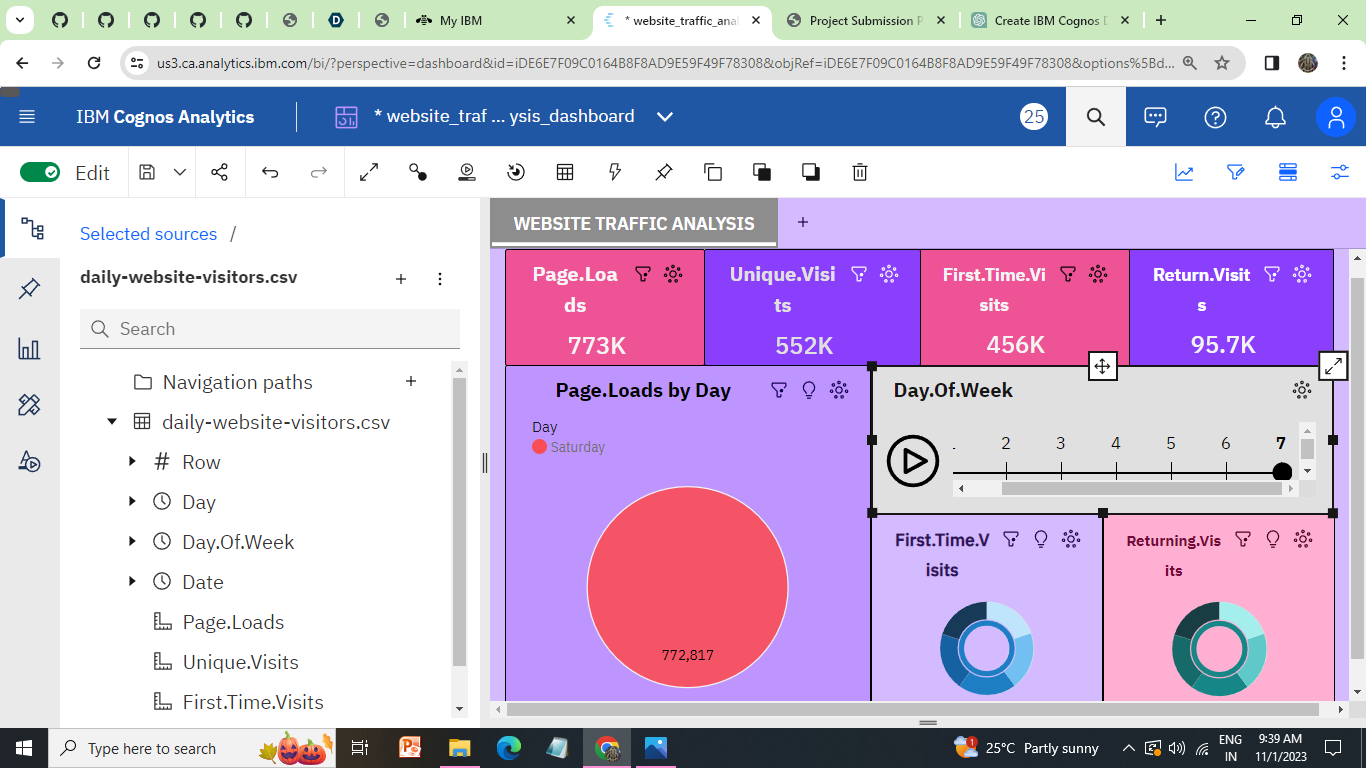
**THURSDAY:**

****

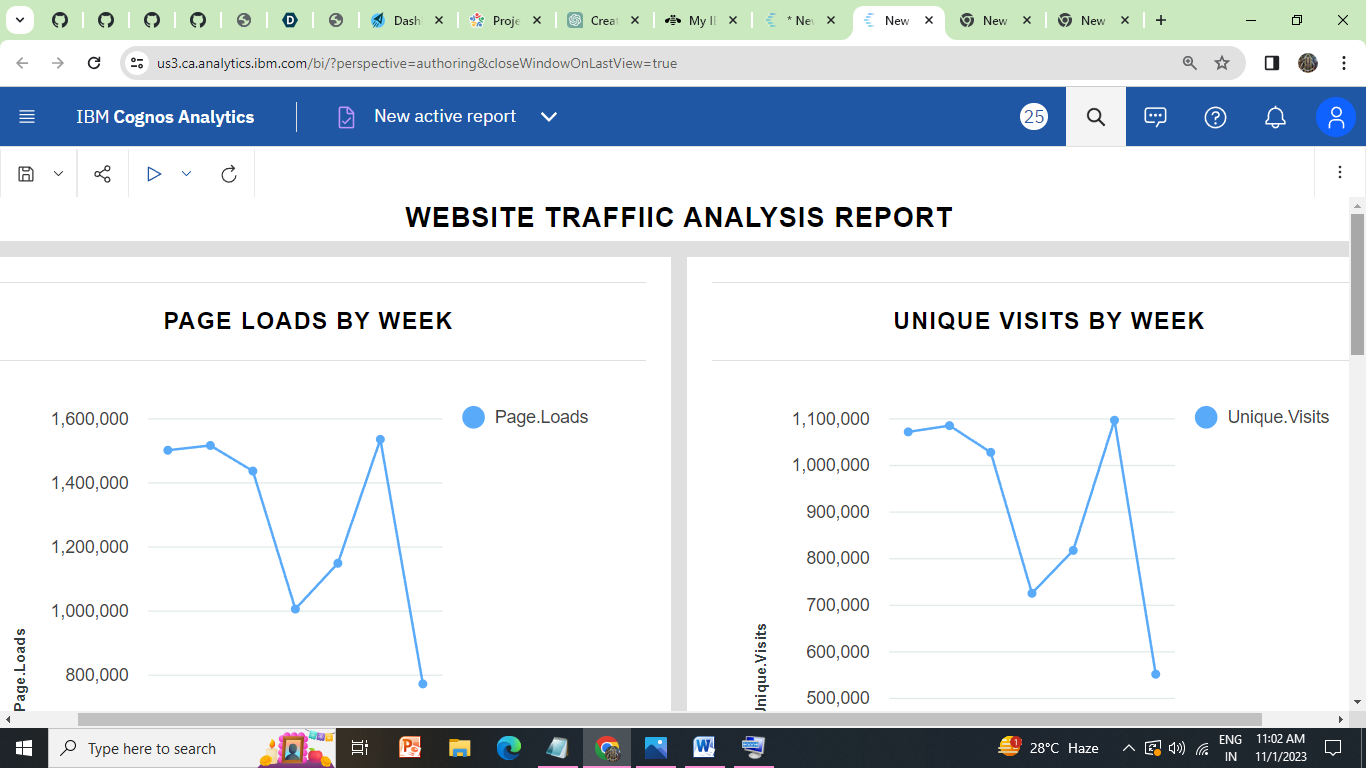
**FRIDAY:**

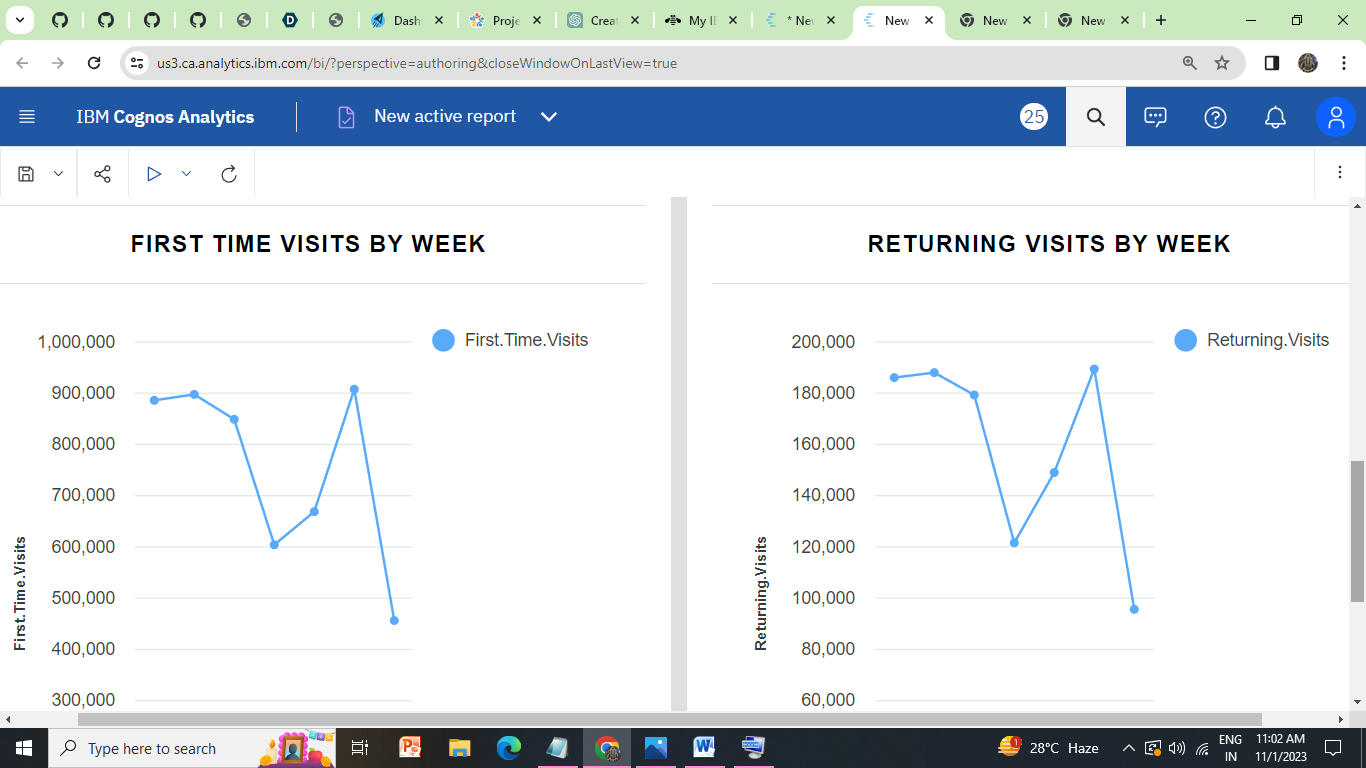
****

**SATURDAY:**

****

**Report:**

****

****

**Analysis:**

Analysis with Python refers to the process of using the Python programming language and various data analysis libraries and tools to explore, manipulate, visualize, and derive insights from data. Python provides a versatile environment for tasks such as data cleaning, statistical analysis, machine learning, time series forecasting, and data visualization, making it a popular choice for data scientists, analysts, and researchers across various domains. It allows users to uncover patterns, trends, and relationships within datasets, enabling data-driven decision-making and problem-solving.

**#Import Libraries**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.ensemble import RandomForestRegressor**

**from sklearn.metrics import mean\_squared\_error**

**from sklearn.cluster import KMeans**

**from sklearn import neighbors**

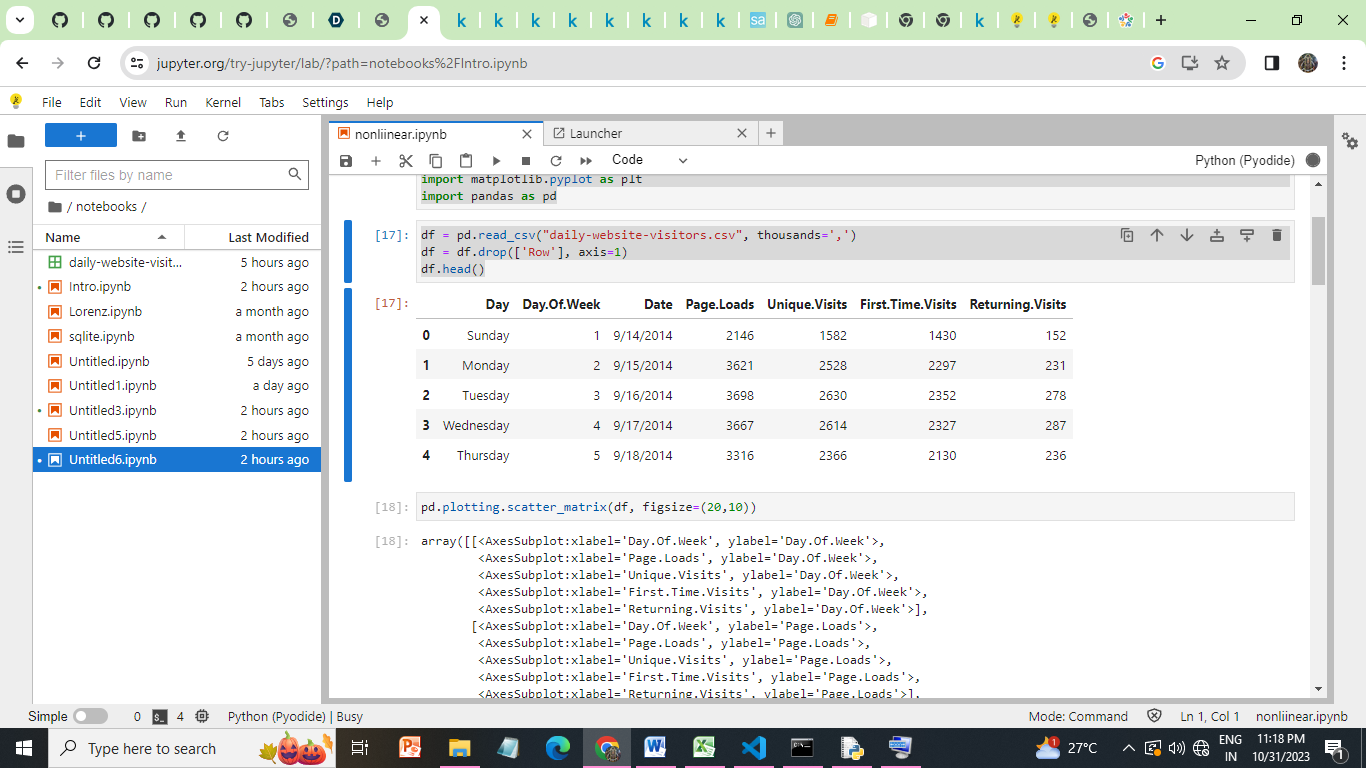
**#load the Dataset**

**df = pd.read\_csv("daily-website-visitors.csv", thousands=',')**

**df = df.drop(['Row'], axis=1)**

**df.head()**

**output:**

****

**pd.plotting.scatter\_matrix(df, figsize=(20,10))**

array([[<AxesSubplot:xlabel='Day.Of.Week', ylabel='Day.Of.Week'>,

<AxesSubplot:xlabel='Page.Loads', ylabel='Day.Of.Week'>,

<AxesSubplot:xlabel='Unique.Visits', ylabel='Day.Of.Week'>,

<AxesSubplot:xlabel='First.Time.Visits', ylabel='Day.Of.Week'>,

<AxesSubplot:xlabel='Returning.Visits', ylabel='Day.Of.Week'>],

[<AxesSubplot:xlabel='Day.Of.Week', ylabel='Page.Loads'>,

<AxesSubplot:xlabel='Page.Loads', ylabel='Page.Loads'>,

<AxesSubplot:xlabel='Unique.Visits', ylabel='Page.Loads'>,

<AxesSubplot:xlabel='First.Time.Visits', ylabel='Page.Loads'>,

<AxesSubplot:xlabel='Returning.Visits', ylabel='Page.Loads'>],

[<AxesSubplot:xlabel='Day.Of.Week', ylabel='Unique.Visits'>,

<AxesSubplot:xlabel='Page.Loads', ylabel='Unique.Visits'>,

<AxesSubplot:xlabel='Unique.Visits', ylabel='Unique.Visits'>,

<AxesSubplot:xlabel='First.Time.Visits', ylabel='Unique.Visits'>,

<AxesSubplot:xlabel='Returning.Visits', ylabel='Unique.Visits'>],

[<AxesSubplot:xlabel='Day.Of.Week', ylabel='First.Time.Visits'>,

<AxesSubplot:xlabel='Page.Loads', ylabel='First.Time.Visits'>,

<AxesSubplot:xlabel='Unique.Visits', ylabel='First.Time.Visits'>,

<AxesSubplot:xlabel='First.Time.Visits', ylabel='First.Time.Visits'>,

<AxesSubplot:xlabel='Returning.Visits', ylabel='First.Time.Visits'>],

[<AxesSubplot:xlabel='Day.Of.Week', ylabel='Returning.Visits'>,

<AxesSubplot:xlabel='Page.Loads', ylabel='Returning.Visits'>,

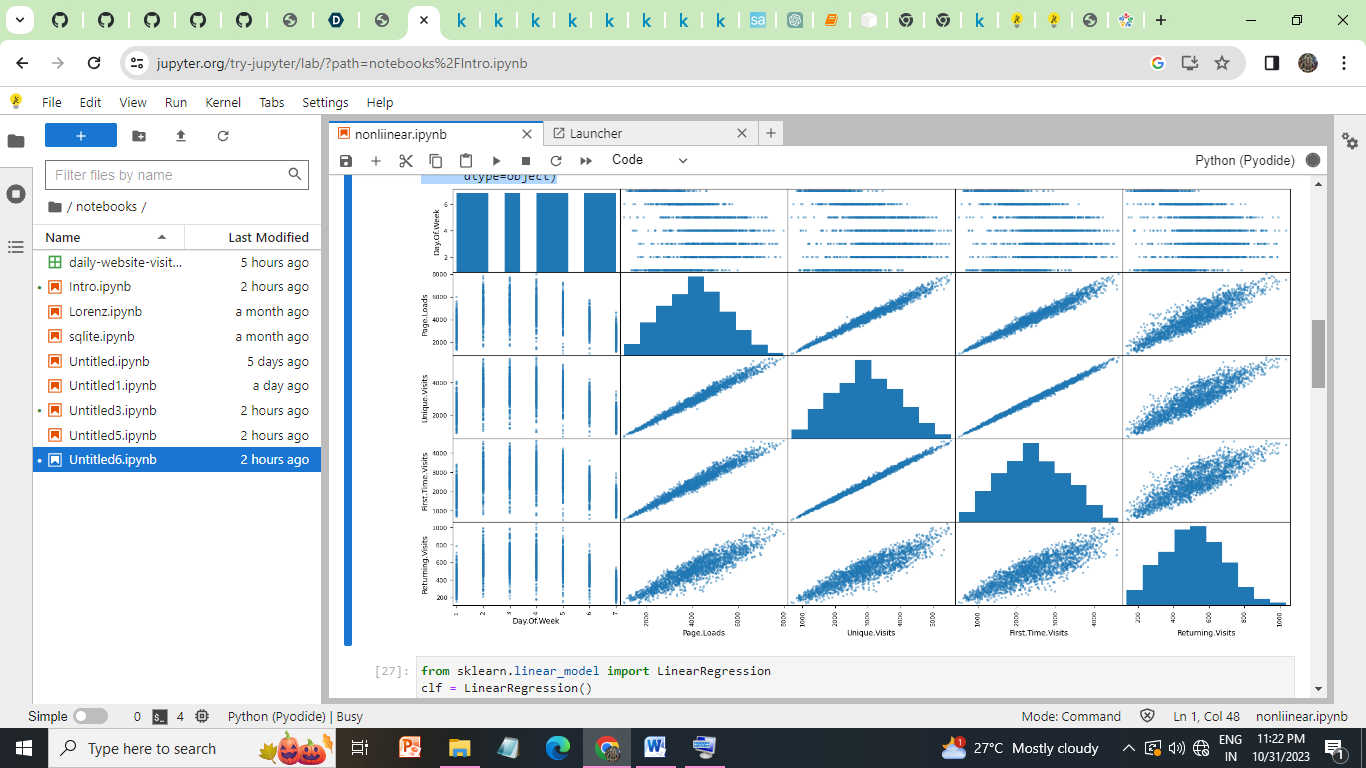
<AxesSubplot:xlabel='Unique.Visits', ylabel='Returning.Visits'>,

<AxesSubplot:xlabel='First.Time.Visits', ylabel='Returning.Visits'>,

<AxesSubplot:xlabel='Returning.Visits', ylabel='Returning.Visits'>]],

dtype=object)

**OUTPUT:**

****

**LINEAR REGRESSION:**

**from sklearn.linear\_model import LinearRegression**

**clf = LinearRegression()**

**clf.fit(x\_train,y\_train)**

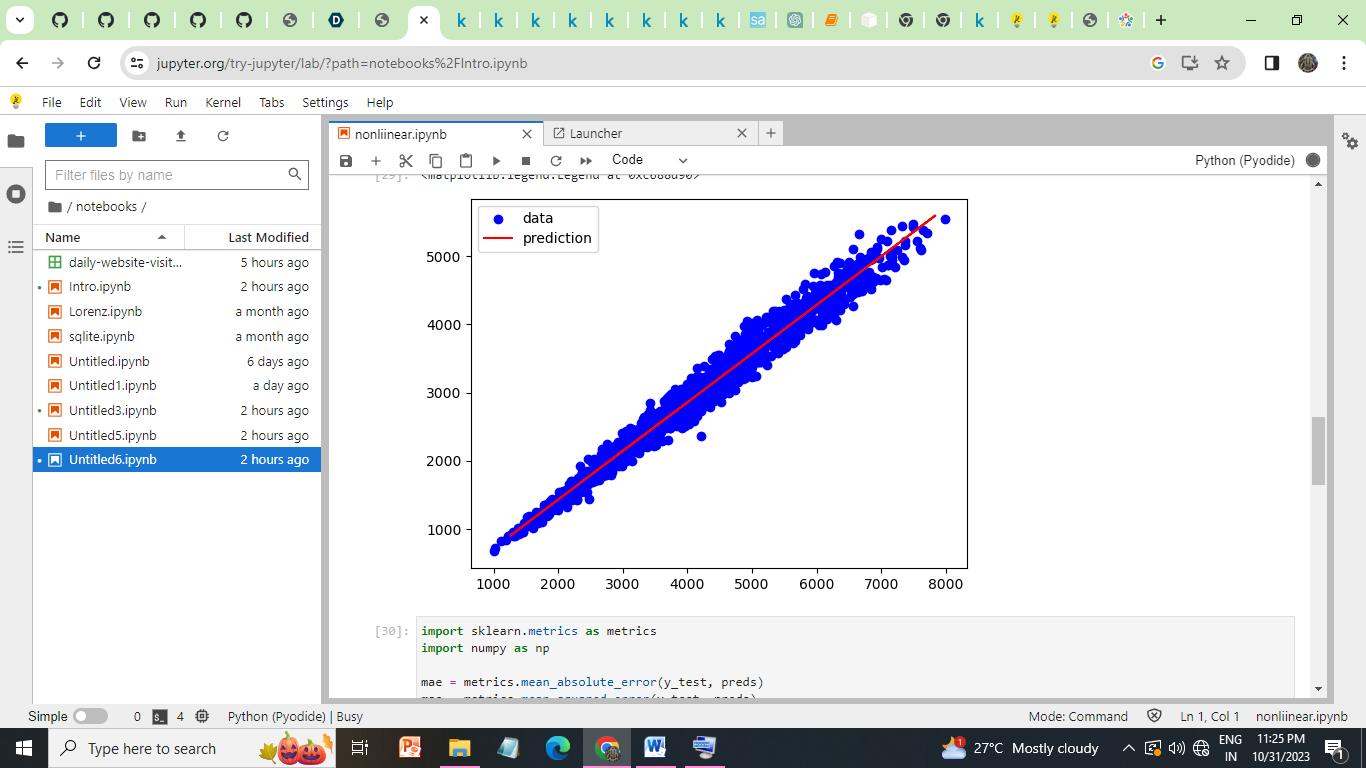
**preds = clf.predict(x\_test)**

**plt.scatter(x\_train, y\_train, color="blue", label="data")**

**plt.plot(x\_test, preds, color="red", label="prediction")**

**plt.legend()**

**OUTPUT:**

****

**METRICS FOR LINEAR REGRESSION:**

**import sklearn.metrics as metrics**

**import numpy as np**

**mae = metrics.mean\_absolute\_error(y\_test, preds)**

**mse = metrics.mean\_squared\_error(y\_test, preds)**

**rmse = np.sqrt(mse) # or mse\*\*(0.5)**

**r2 = metrics.r2\_score(y\_test,preds)**

**print("Results of Linear Regression:")**

**print("MAE:",mae)**

**print("MSE:", mse)**

**print("RMSE:", rmse)**

**print("R-Squared:", r2)**

**OUTPUT:**

Results of Linear Regression:

MAE: 113.01405534987238

MSE: 20710.290923291977

RMSE: 143.91070468624625

R-Squared: 0.977890439598449

**#TimeSeriesForPageLoads**

**import matplotlib.pyplot as plt**

**df['Date'] = pd.to\_datetime(df['Date']) # Convert Date to a datetime object**

**plt.figure(figsize=(12, 6))**

**plt.plot(df['Date'], df['Page.Loads'], label='Page Loads')**

**plt.xlabel('Date')**

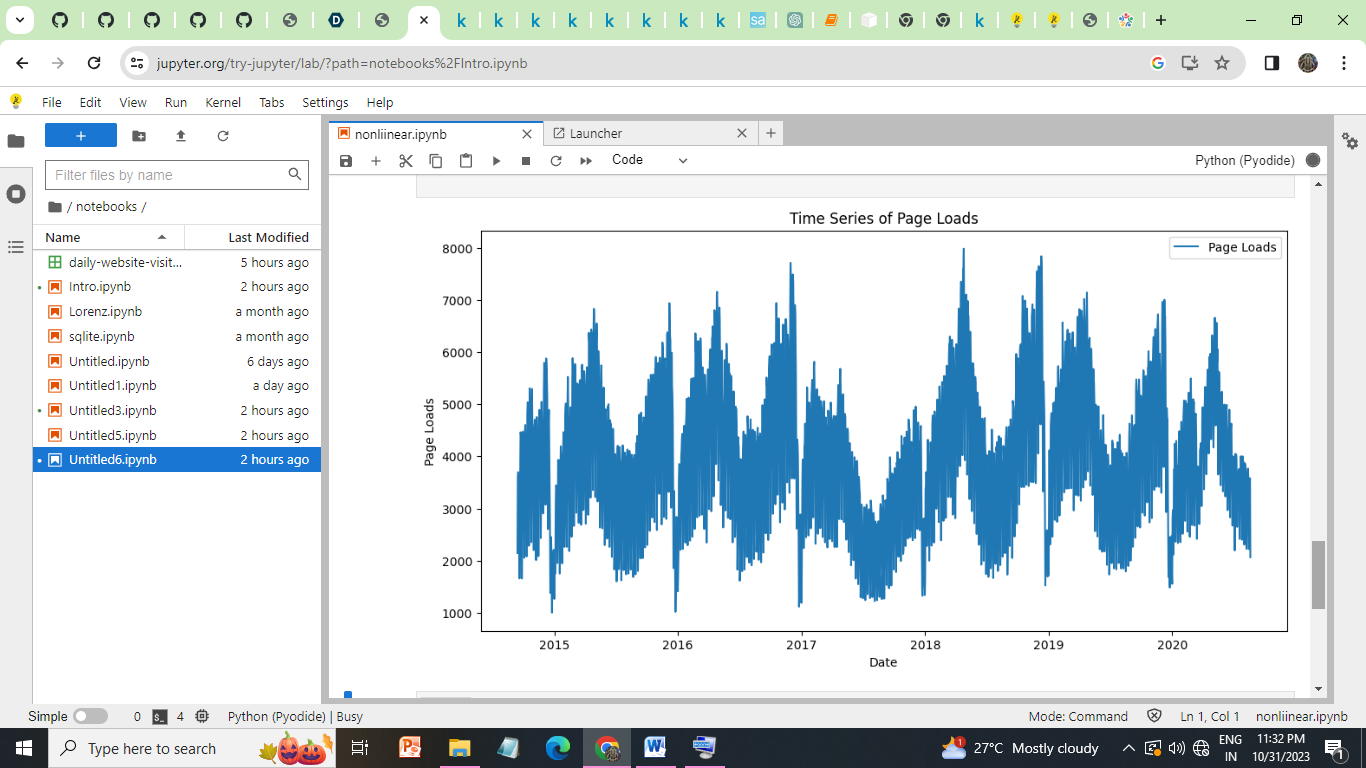
**plt.ylabel('Page Loads')**

**plt.title('Time Series of Page Loads')**

**plt.legend()**

**plt.show()**

**Output:**

****

**USER SEGMENTATION ANALYSIS:**

**import seaborn as sns**

**from sklearn.preprocessing import StandardScaler**

**from scipy.cluster.hierarchy import dendrogram, linkage, fcluster**

**from scipy.spatial.distance import pdist**

**# Load our dataset into a DataFrame**

**df = pd.read\_csv('daily-website-visitors.csv')**

**# Data Preprocessing**

**selected\_features = ['Page.Loads', 'Unique.Visits', 'First.Time.Visits', 'Returning.Visits']**

**# Remove any commas and convert columns to float**

**for feature in selected\_features:**

**df[feature] = df[feature].str.replace(',', '').astype(float)**

**# Standardize the selected features**

**scaler = StandardScaler()**

**X = scaler.fit\_transform(df[selected\_features])**

**# Calculate the linkage matrix for hierarchical clustering**

**linkage\_matrix = linkage(X, method='ward', metric='euclidean')**

**# Determine the optimal number of clusters using the dendrogram**

**plt.figure(figsize=(12, 6))**

**dendrogram(linkage\_matrix)**

**plt.title('User Segmentation Analysis')**

**plt.xlabel('users')**

**plt.ylabel('Distance')**

**plt.show()**

**# Based on the dendrogram, choose the optimal number of clusters (e.g., 3)**

**num\_clusters = 3**

**clusters = fcluster(linkage\_matrix, t=num\_clusters, criterion='maxclust')**

**# Add the cluster labels to the DataFrame**

**df['Cluster'] = clusters**

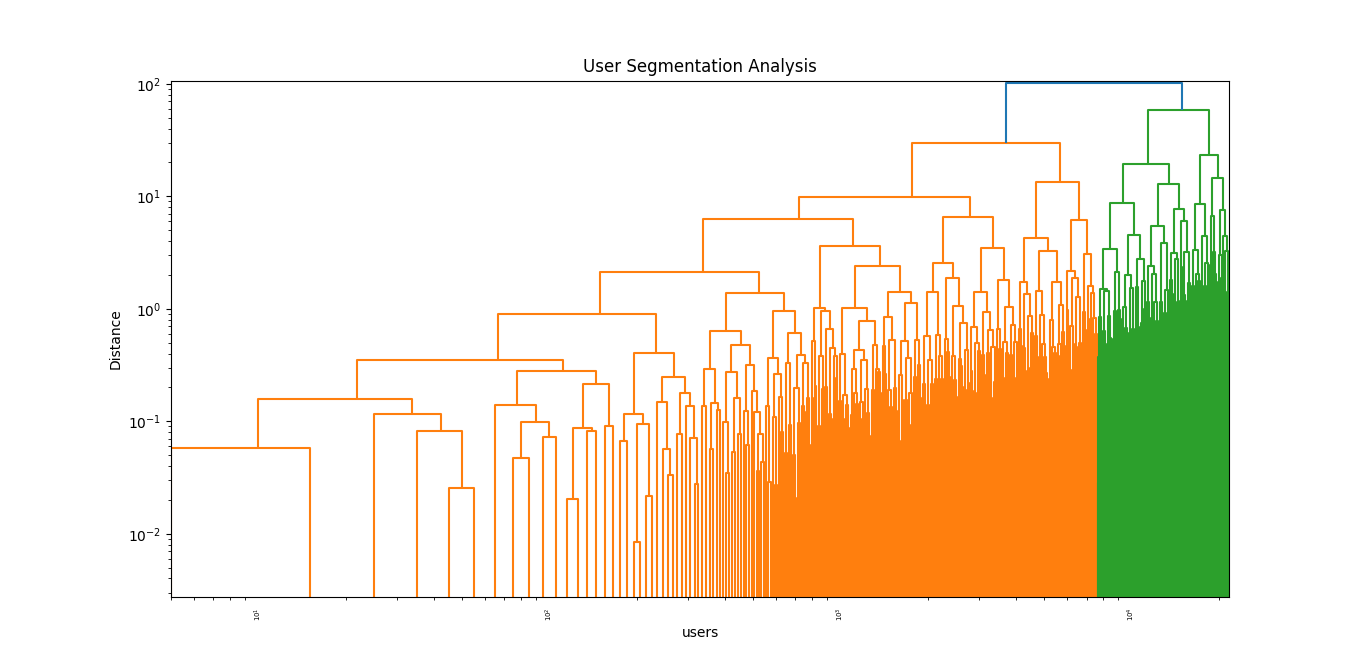
**# Visualize the user segments**

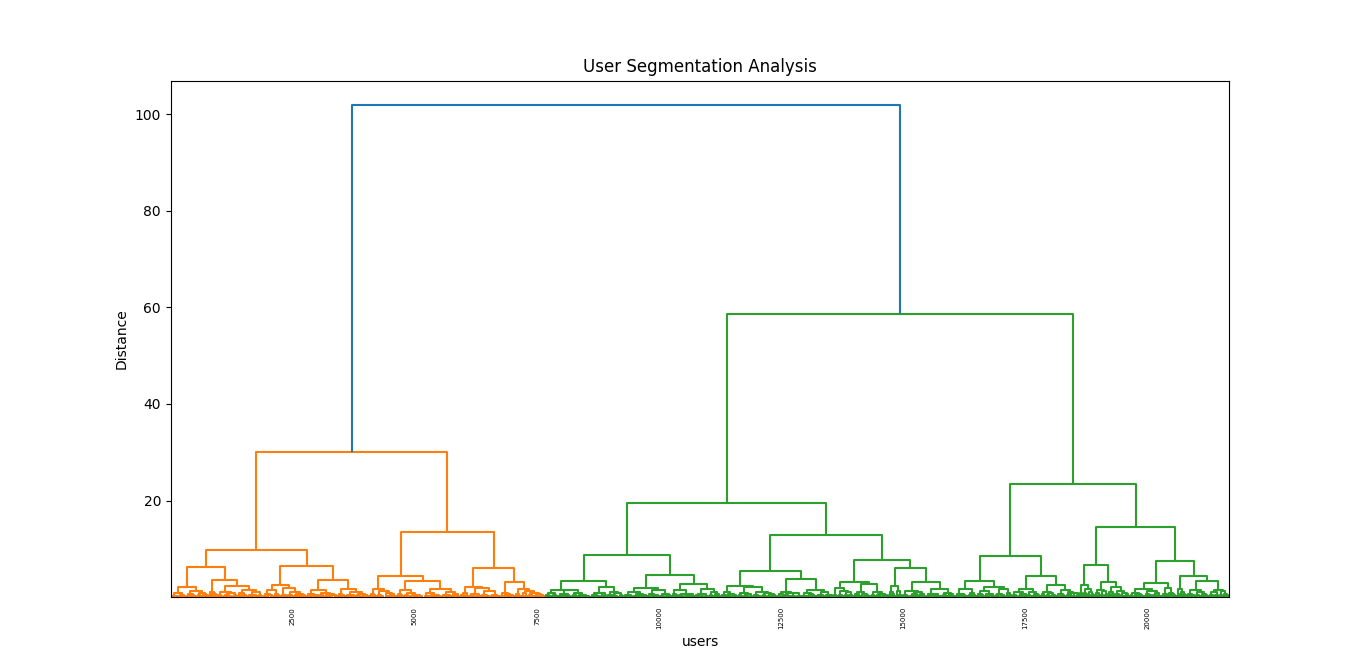
**sns.pairplot(df, hue='Cluster', vars=selected\_features, diag\_kind='kde')**

**plt.suptitle('User Segmentation Analysis')**

**plt.show()**

**OUTPUT:**

****

****

**TIME SERIES DECOMPOSITION:**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**from statsmodels.tsa.seasonal import seasonal\_decompose**

**# Load our dataset**

**df = pd.read\_csv("daily-website-visitors.csv")**

**# Clean and preprocess the "Page.Loads" column**

**df['Page.Loads'] = df['Page.Loads'].str.replace(',', '').astype(float)**

**# Convert the "Date" column to datetime and set it as the index**

**df['Date'] = pd.to\_datetime(df['Date'])**

**df.set\_index('Date', inplace=True)**

**# Specify the frequency (assuming daily data)**

**df.index.freq = 'D'**

**# Perform seasonal decomposition using statsmodels**

**result = seasonal\_decompose(df['Page.Loads'], model='additive')**

**# Plot the components**

**plt.figure(figsize=(12, 8))**

**plt.subplot(411)**

**plt.plot(result.observed, label='Observed')**

**plt.legend()**

**plt.subplot(412)**

**plt.plot(result.trend, label='Trend')**

**plt.legend()**

**plt.subplot(413)**

**plt.plot(result.seasonal, label='Seasonal')**

**plt.legend()**

**plt.subplot(414)**

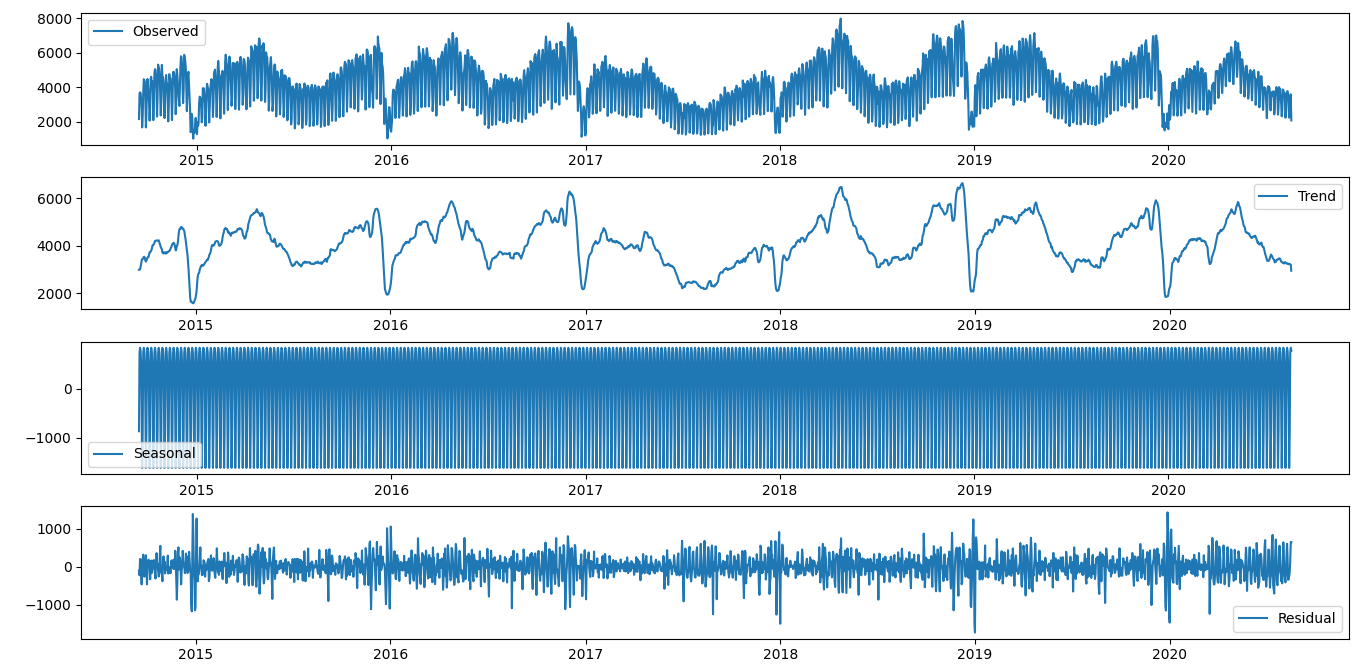
**plt.plot(result.resid, label='Residual')**

**plt.legend()**

**plt.tight\_layout()**

**plt.show()**

**OUTPUT:**

****

**DATA SPLITS:**

**from sklearn.model\_selection import train\_test\_split**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import seaborn as sns**

**# Load our dataset**

**df = pd.read\_csv("daily-website-visitors.csv")**

**# Split the data into training and testing sets**

**Y = df['Unique.Visits']**

**X = df[['Page.Loads']]**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2)**

**# Count the data points in each set**

**train\_count = y\_train.count()**

**test\_count = y\_test.count()**

**# Create a bar plot**

**plt.bar(x=['Train', 'Test'], height=[train\_count, test\_count])**

**# Add labels and title**

**plt.xlabel('Dataset')**

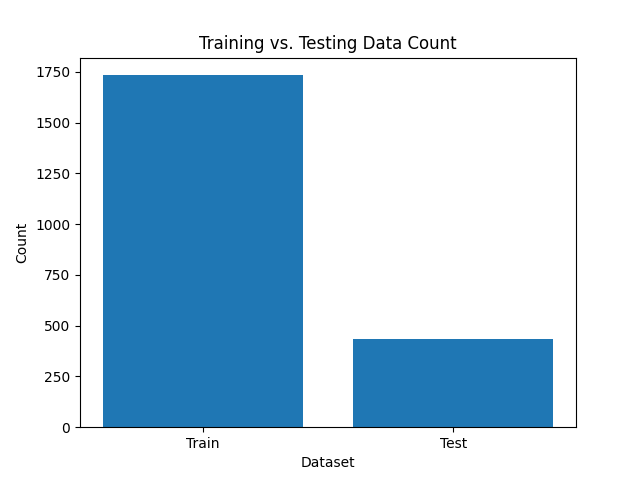
**plt.ylabel('Count')**

**plt.title('Training vs. Testing Data Count')**

**# Show the plot**

**plt.show()**

**OUTPUT:**

****

**Making ML Model**

**1)Split the Data into Training and Testing Sets:**

We can use a portion of our data for training the model and reserve the rest for testing. The typical split is around 70-80% for training and 20-30% for testing, but we can adjust this based on our data and requirements.

**2)Train the ARIMA Model:**

Train the ARIMA model using the training data. we can use the training data to find the best model parameters

**3)Make Predictions on the Testing Set:**

Use the trained model to make predictions on the testing set.

**4)Evaluate the Model:**

Evaluate the model's performance by comparing the predicted values with the actual values in the testing set. we can use metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and visualizations to assess the quality of the forecasts.

**OVERALL :**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**# Load our dataset**

**df = pd.read\_csv("daily-website-visitors.csv")**

**# Clean and preprocess the "Page.Loads" column**

**df['Page.Loads'] = df['Page.Loads'].str.replace(',', '').astype(float)**

**# Convert the "Date" column to datetime and set it as the index**

**df['Date'] = pd.to\_datetime(df['Date'])**

**df.set\_index('Date', inplace=True)**

**# Specify the frequency (assuming daily data)**

**df.index.freq = 'D'**

**# Split the data into training and testing sets**

**train\_size = int(len(df) \* 0.8) # Adjust the split ratio as needed**

**train, test = df[:train\_size], df[train\_size:]**

**from statsmodels.tsa.arima.model import ARIMA**

**# Fit an ARIMA model to the training data**

**model = ARIMA(train['Page.Loads'], order=(1, 1, 1))**

**model\_fit = model.fit()**

**forecast\_steps = len(test)**

**forecast = model\_fit.forecast(steps=forecast\_steps)**

**from sklearn.metrics import mean\_squared\_error**

**mse = mean\_squared\_error(test['Page.Loads'], forecast)**

**rmse = mse \*\* 0.5**

**print(f"Mean Squared Error: {mse}")**

**print(f"Root Mean Squared Error: {rmse}")**

**# Plot observed vs. predicted values**

**plt.figure(figsize=(12, 6))**

**plt.plot(test.index, test['Page.Loads'], label='Observed')**

**plt.plot(test.index, forecast, label='Forecast', linestyle='--')**

**plt.xlabel('Date')**

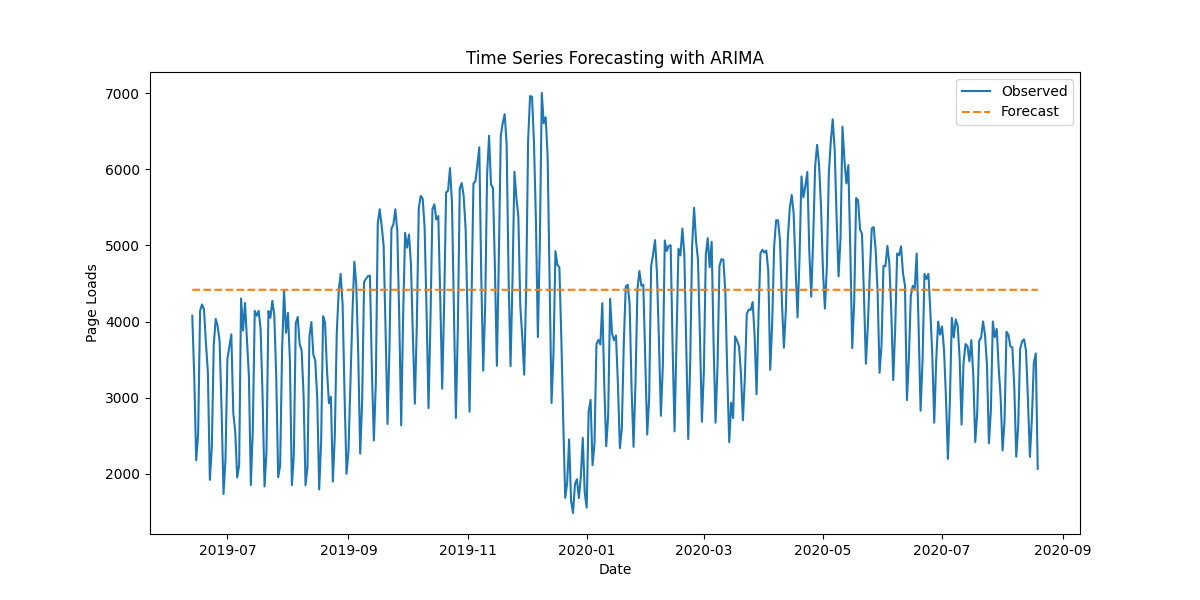
**plt.ylabel('Page Loads')**

**plt.title('Time Series Forecasting with ARIMA')**

**plt.legend()**

**plt.show()**

**OUTPUT:**

****

**All Python Files ,Jupyter Files ,Dashboard Link And Report Link Are Provided In My Github.**

**Conclusion:**

In this project phase, we have harnessed the power of IBM Cognos Analytics to craft interactive dashboards and detailed reports, enabling us to present key insights, including popular pages, traffic sources, and user engagement metrics. We have seamlessly integrated Python, leveraging libraries like Pandas and Matplotlib, to delve into more sophisticated data analyses, spanning time series analysis, user segmentation, and machine learning-driven predictions. This hybrid approach combines the strengths of Cognos for user-friendly data visualization with the flexibility of Python for advanced analytics, ultimately facilitating a comprehensive understanding of the data and the ability to make informed decisions.