WEBSITE TRAFFIC ANALYSIS

INTRODUCTION:

To analyze website traffic data to understand user behaviour, popular pages and traffic sources , helping website owners to improve user experience. Website traffic analysis is a crucial aspect of data analytics, providing valuable insights into user behavior and interaction patterns on a website. Analyzing website traffic helps businesses and website owners understand their audience, optimize user experience, and make data-driven decisions to enhance their online presence.

DATASET:

DATA SOURCE:

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

PREPROCESSING:

PROGRAM:

```
import numpy as np
import pandas as pd
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
# Import Libraries
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import mode
data=pd.read_csv("/content/daily-website-visitors.csv")
data.head()
```

Row	Day	Day.Of.Week	Date	Page.Loads	Unique.Visits	First.Time.Visits	Returning.Visits
0	1	Sunday	9/14/2014	2,146	1,582	1,430	152
1	2	Monday	9/15/2014	3,621	2,528	2,297	231
2	3	Tuesday	9/16/2014	3,698	2,630	2,352	278
3	4	Wednesday	9/17/2014	3,667	2,614	2,327	287
4	5	Thursday	9/18/2014	3,316	3,366	2,130	236

Data preprocessing

* 1.Convert Date into Datetime format.
* 2.Removing ',' from Page.Loads, Unique.Visits, First.Time.Visits, Returning.Visits.
* 3.Convert the above values into float.

```
# Function to remove commas
def remove_commas(x):
    return float(x.replace(',', ''))
# Apply the preprocessing functions

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))
data.head()
```

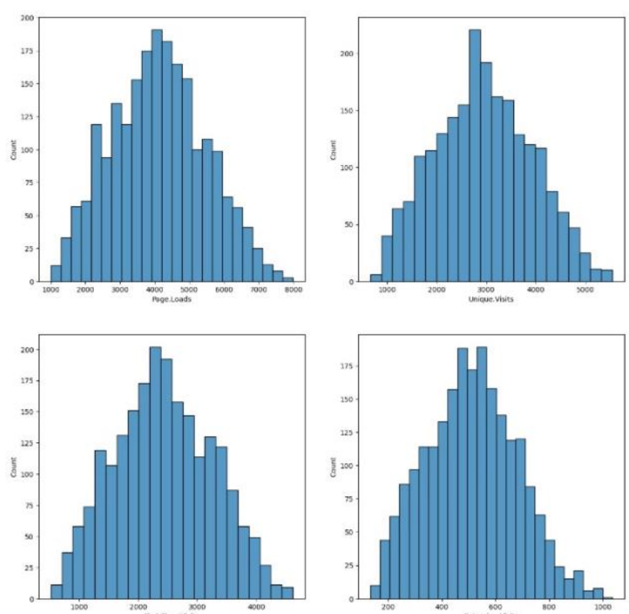
DATA VISUALIZATION:

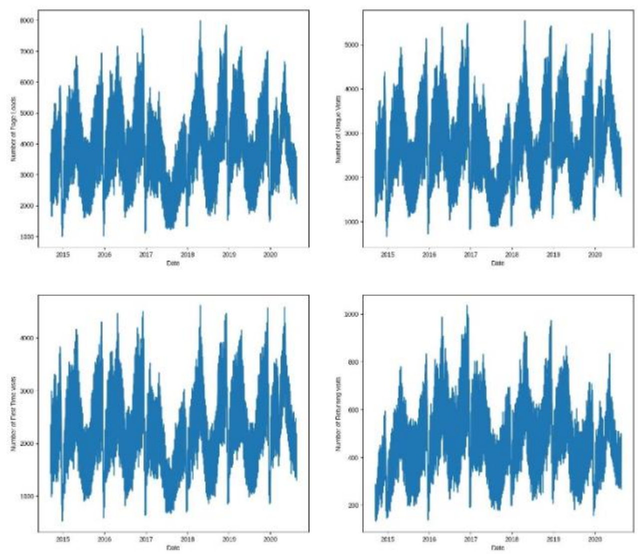
```
check_normality(data, col):

    # 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)
* mean : 4116 median : 4106 mode : 2948
* Page.Loads Distribution is skewed towards right
* mean : 2943 median : 2914 mode : 1197
* Unique.Visits Distribution is skewed towards right
* mean : 2431 median : 2400 mode : 3133
* First.Time.Visits Distribution is skewed towards left
* mean : 511 median : 509 mode : 552
* Returning.Visits Distribution is skewed towards left
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from datetime import datetime

# Load the dataset (replace 'website_traffic.csv' with your dataset)
data = pd.read_csv('website_traffic.csv')
```

```
# Select relevant columns (e.g., Date, PageViews, BounceRate, etc.)
selected_columns = ["Date", "PageViews", "BounceRate", "TimeOnSite"]
data = data[selected_columns]

# Convert the 'Date' column to datetime
data["Date"] = pd.to_datetime(data["Date"])

# Define X (features) and y (target)
X = data[["PageViews", "BounceRate"]]
y = data["TimeOnSite"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Data Visualization
# Scatter plot of actual vs. predicted TimeOnSite
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred)
plt.title('Actual TimeOnSite vs. Predicted TimeOnSite')
plt.xlabel('Actual TimeOnSite')
plt.ylabel('Predicted TimeOnSite')
plt.show()

# Residual plot to check for homoscedasticity
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
plt.scatter(y_pred, residuals)
plt.title('Residual Plot')
plt.xlabel('Predicted TimeOnSite')
plt.ylabel('Residuals')
plt.axhline(0, color='red', linestyle='--')
plt.show()
```





#Plot the correlation heatmap

Corr_matrix = data.corr()

Plt.figure(figsize=(12,12))

Sns.heatmap(corr_matrix, annot=True, cbar=False)

Plt.show()