Web Traffic Analysis

Phase 3

Collecting Data:

You can use libraries like requests to fetch data from web servers or APIs. Alternatively, you can work with log files from your web server.

Data Parsing:

Parse the data to extract relevant information. If you're working with log files, you'll need to extract fields like IP addresses, URLs, timestamps, and more.

Data Cleaning:

Clean and preprocess the data to handle missing values, outliers, and any other inconsistencies.

Data Analysis:

Analyze traffic patterns, such as hourly, daily, or monthly traffic.Identify popular pages or resources. Determine the geographical locations of your visitors. Detect unusual traffic or potential security threats.

Visualization:

Use libraries like Matplotlib, Seaborn, or Plotly to create visualizations that help you understand the data.

Reporting:

Generate reports or dashboards summarizing the key insights from your analysis.

Here's a simple example to get you started with Python code for web traffic analysis using a sample log file:

Coding :

```
import pandas as pd

# Load the log file into a DataFrame log_data =      pd.read_csv('web_access_log.txt', sep=' ',
names=['IP', '-', '-', 'Timestamp', 'Request', 'Status', 'Bytes'])
```

```
# Convert the timestamp to a datetime object log_data['Timestamp'] =
pd.to_datetime(log_data['Timestamp'], format='[%d/%b/%Y:%H:%M:%S')

# Analyze daily traffic daily_traffic = log_data.groupby(log_data['Timestamp'].dt.date).size()

 import matplotlib.pyplot as plt daily_traffic.plot(kind='line')

 plt.xlabel('Date')

plt.ylabel('Number of Requests')

plt.title('Daily Web Traffic')

plt.show()
```

This is a very basic example. For a comprehensive analysis, you might need to implement more complex algorithms and use additional libraries like NumPy, scikit-learn, and others, depending on your specific analysis goals.