In [2]:
```
pip install Django
```

```
Collecting DjangoNote: you may need to restart the kernel to use updated
packages.

  Obtaining dependency information for Django from https://files.pythonh
osted.org/packages/b9/45/707dfc56f381222c1c798503546cb390934ab246fc45b50
51ef66e31099c/Django-4.2.6-py3-none-any.whl.metadata (https://files.pyth
onhosted.org/packages/b9/45/707dfc56f381222c1c798503546cb390934ab246fc45
b5051ef66e31099c/Django-4.2.6-py3-none-any.whl.metadata)
  Downloading Django-4.2.6-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.6.0 (from Django)
  Obtaining dependency information for asgiref<4,>=3.6.0 from https://fi
les.pythonhosted.org/packages/9b/80/b9051a4a07ad231558fcd8ffc89232711b4e
618c15cb7a392a17384bbeef/asgiref-3.7.2-py3-none-any.whl.metadata (http
s://files.pythonhosted.org/packages/9b/80/b9051a4a07ad231558fcd8ffc89232
711b4e618c15cb7a392a17384bbeef/asgiref-3.7.2-py3-none-any.whl.metadata)
  Downloading asgiref-3.7.2-py3-none-any.whl.metadata (9.2 kB)
Collecting sqlparse>=0.3.1 (from Django)
  Downloading sqlparse-0.4.4-py3-none-any.whl (41 kB)
     ---------------------------------------- 0.0/41.2 kB ? eta -:--:--
     ---------------------------- --------- 30.7/41.2 kB 660.6 kB/s eta
0:00:01
     ---------------------------------------- 41.2/41.2 kB 658.4 kB/s eta
0:00:00
Collecting tzdata (from Django)
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
     ---------------------------------------- 0.0/341.8 kB ? eta -:--:--
     ------- -------------------------------- 61.4/341.8 kB 1.6 MB/s eta
0:00:01
     ------------------ ------------------ 174.1/341.8 kB 2.1 MB/s eta
0:00:01
     ------------------------------------  337.9/341.8 kB 2.6 MB/s eta
0:00:01
     ---------------------------------------- 341.8/341.8 kB 2.4 MB/s eta
0:00:00
Downloading Django-4.2.6-py3-none-any.whl (8.0 MB)
   ---------------------------------------- 0.0/8.0 MB ? eta -:--:--
   - -------------------------------------- 0.3/8.0 MB 5.2 MB/s eta 0:0
0:02
   -- ------------------------------------- 0.5/8.0 MB 4.7 MB/s eta 0:0
0:02
   --- ------------------------------------ 0.7/8.0 MB 4.9 MB/s eta 0:0
0:02
   ----- ---------------------------------- 1.0/8.0 MB 5.4 MB/s eta 0:0
0:02
   ----- ---------------------------------- 1.2/8.0 MB 5.0 MB/s eta 0:0
0:02
   ------ --------------------------------- 1.4/8.0 MB 5.1 MB/s eta 0:0
0:02
   -------- ------------------------------- 1.8/8.0 MB 5.4 MB/s eta 0:0
0:02
   --------- ------------------------------ 2.0/8.0 MB 5.3 MB/s eta 0:0
0:02
   ---------- ----------------------------- 2.1/8.0 MB 5.0 MB/s eta 0:0
0:02
   ----------- ---------------------------- 2.3/8.0 MB 4.9 MB/s eta 0:0
0:02
   ------------ --------------------------- 2.5/8.0 MB 4.9 MB/s eta 0:0
```

```
0:02
   ------------ ------------------------- 2.7/8.0 MB 4.8 MB/s eta 0:0
0:02
   ------------- ------------------------- 2.9/8.0 MB 4.8 MB/s eta 0:0
0:02
   -------------- ------------------------ 3.1/8.0 MB 4.7 MB/s eta 0:0
0:02
   --------------- ------------------------ 3.3/8.0 MB 4.6 MB/s eta 0:0
0:02
   ---------------- ----------------------- 3.5/8.0 MB 4.6 MB/s eta 0:0
0:01
   ----------------- ---------------------- 3.7/8.0 MB 4.6 MB/s eta 0:0
0:01
   ------------------ --------------------- 3.9/8.0 MB 4.6 MB/s eta 0:0
0:01
   ------------------- -------------------- 4.1/8.0 MB 4.6 MB/s eta 0:0
0:01
   -------------------- ------------------- 4.3/8.0 MB 4.6 MB/s eta 0:0
0:01
   --------------------- ------------------ 4.5/8.0 MB 4.6 MB/s eta 0:0
0:01
   ---------------------- ----------------- 4.8/8.0 MB 4.6 MB/s eta 0:0
0:01
   ----------------------- ---------------- 5.0/8.0 MB 4.6 MB/s eta 0:0
0:01
   ------------------------ --------------- 5.2/8.0 MB 4.6 MB/s eta 0:0
0:01
   ------------------------ --------------- 5.2/8.0 MB 4.6 MB/s eta 0:0
0:01
   ------------------------- -------------- 5.5/8.0 MB 4.4 MB/s eta 0:0
0:01
   -------------------------- ------------- 5.7/8.0 MB 4.5 MB/s eta 0:0
0:01
   -------------------------- ------------- 5.9/8.0 MB 4.5 MB/s eta 0:0
0:01
   --------------------------- ------------ 6.1/8.0 MB 4.5 MB/s eta 0:0
0:01
   ---------------------------- ----------- 6.4/8.0 MB 4.5 MB/s eta 0:0
0:01
   ---------------------------- ----------- 6.4/8.0 MB 4.5 MB/s eta 0:0
0:01
   ----------------------------- ---------- 6.7/8.0 MB 4.4 MB/s eta 0:0
0:01
   ----------------------------- ---------- 6.7/8.0 MB 4.4 MB/s eta 0:0
0:01
   ----------------------------- ---------- 6.7/8.0 MB 4.3 MB/s eta 0:0
0:01
   ----------------------------- ---------- 6.7/8.0 MB 4.3 MB/s eta 0:0
0:01
   ----------------------------- ---------- 6.8/8.0 MB 4.0 MB/s eta 0:0
0:01
   ------------------------------ --------- 6.8/8.0 MB 3.9 MB/s eta 0:0
0:01
   ------------------------------ --------- 6.8/8.0 MB 3.9 MB/s eta 0:0
0:01
   ------------------------------- -------- 7.0/8.0 MB 3.8 MB/s eta 0:0
0:01
```

```
   ---------------------------------- ---- 7.0/8.0 MB 3.8 MB/s eta 0:0
0:01
   ---------------------------------- ---- 7.0/8.0 MB 3.8 MB/s eta 0:0
0:01
   ---------------------------------- ---- 7.0/8.0 MB 3.6 MB/s eta 0:0
0:01
   ---------------------------------- ---- 7.1/8.0 MB 3.5 MB/s eta 0:0
0:01
   ---------------------------------- ---- 7.2/8.0 MB 3.4 MB/s eta 0:0
0:01
   ---------------------------------- --- 7.2/8.0 MB 3.4 MB/s eta 0:0
0:01
   ---------------------------------- --- 7.3/8.0 MB 3.4 MB/s eta 0:0
0:01
   ---------------------------------- -- 7.5/8.0 MB 3.4 MB/s eta 0:0
0:01
   ---------------------------------- - 7.6/8.0 MB 3.4 MB/s eta 0:0
0:01
   ---------------------------------- - 7.7/8.0 MB 3.3 MB/s eta 0:0
0:01
   ---------------------------------- 7.8/8.0 MB 3.3 MB/s eta 0:0
0:01
   ---------------------------------- 7.8/8.0 MB 3.3 MB/s eta 0:0
0:01
   ---------------------------------- 8.0/8.0 MB 3.2 MB/s eta 0:0
0:01
   ---------------------------------- 8.0/8.0 MB 3.2 MB/s eta 0:0
0:01
   ---------------------------------- 8.0/8.0 MB 3.2 MB/s eta 0:0
0:00
Downloading asgiref-3.7.2-py3-none-any.whl (24 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.2.6 asgiref-3.7.2 sqlparse-0.4.4 tzdata-
2023.3
```

In [3]:

```
from django.conf import settings
import shutil
import os
import advertools as adv
import pandas as pd
import pyarrow.parquet as pq
from io import BytesIO
import seaborn as sns
import matplotlib.pyplot as plt
from ua_parser import user_agent_parser
import ua_parser
from urllib.parse import urlparse
import base64
# sets no limit to number of columns displayed
pd.options.display.max_columns = None


class LogAnalyser():
    def _init_(self):
        self.logs_df = None
        self.host_df = None
        self.request_url_df = None
        self.referer_url_df = None
        self.ua_df = None

    def preprocess_log_file(self, output_file):
        try:
            if output_file and os.path.exists(output_file):
                # Now you can save output_media_path and errors_media_path
                self.logs_df = pd.read_parquet(output_file)
                # Now you can save output_media_path and errors_media_path
                self.logs_df = pd.read_parquet(output_file)

                # PREPROCESSING
                # Parse datetime column
                self.logs_df['datetime'] = pd.to_datetime(
                    self.logs_df['datetime'], format='%d/%b/%Y:%H:%M:%S %z
                # Reverse DNS lookup
                self.host_df = adv.reverse_dns_lookup(self.logs_df['client
                ip_host_dict = dict(
                    zip(self.host_df['ip_address'], self.host_df['hostname
                self.logs_df['hostname'] = [ip_host_dict[ip]
                                            for ip in self.logs_df['client

                # Parse URL components from 'request' column
                # Parse URL components using urlparse and create a DataFra
                request_url_components = self.logs_df['request'].apply(
                    lambda url: urlparse(url))
                self.request_url_df = pd.DataFrame(request_url_components.
                    'scheme', 'netloc', 'path', 'params', 'query', 'fragme
                # Add a prefix to column names
                self.request_url_df = self.request_url_df.add_prefix(
                    'request_')

                # Parse URL components from 'referer' column
                referer_url_components = self.logs_df['referer'].apply(
                    lambda url: urlparse(url))
```

```
                self.referer_url_df = pd.DataFrame(referer_url_components.
                    'scheme', 'netloc', 'path', 'params', 'query', 'fragme
                # Adding prefix
                self.referer_url_df = self.referer_url_df.add_prefix(
                    'referer_')

                # Parse user-agent strings
                self.ua_df = pd.json_normalize(
                    [user_agent_parser.Parse(ua) for ua in self.logs_df['u
                # Converting columns of ua_df to ua_ + name
                self.ua_df.columns = 'ua_' + \
                    self.ua_df.columns.str.replace(
                        'user_agent\.', '', regex=True)

                return self.logs_df, self.host_df, self.request_url_df, se
            except Exception as e:
                return None

        def save_plot_to_base64(plt):
            buffer = BytesIO()
            plt.savefig(buffer, format='png')
            buffer.seek(0)
            plt_base64 = base64.b64encode(buffer.read()).decode('utf-8')
            plt.close()  # Close the plot to free up resources
            return plt_base64

        def _generate_graph1(self, host_df):

            # Define the file path for the graph image
            save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph1.pr

            # Check if the image file already exists
            if os.path.exists(save_path):
                # If the file exists, return the URL path of the existing imag
                return os.path.join(settings.MEDIA_URL, 'images', 'graph1.png'

            # Generate the bar plot
            plt.figure(figsize=(12, 7))
            sns.barplot(data=host_df.head(50), x='ip_address', y='count')
            plt.xticks(rotation=45)
            plt.xlabel('IP Address')
            plt.ylabel('Count')
            plt.title('Top 50 IP Addresses by Count')
            plt.tight_layout()

            # Save the plot to the media/images directory
            plt.savefig(save_path)
            plt.close()  # Close the plot to free up resources

            # Return the URL path of the generated graph
            return os.path.join(settings.MEDIA_URL, 'images', 'graph1.png')

        def _generate_graph2(self, host_df):

            # Define the file path for the graph image
            save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph2.pr
```

```
        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph2.png'

        x_plot_data = host_df['hostname'].head(50)
        plt.figure(figsize=(12, 7))
        sns.countplot(data=host_df, x='hostname',
                      order=x_plot_data.value_counts().index)
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('Hostname')
        plt.ylabel('Count')
        plt.title('Distribution of Top 50 Hostnames')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources

        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph2.png')

    def _generate_graph3(self, host_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph3.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph3.png'

        # Fill None values with a label (e.g., 'No Error') before counting
        host_df['errors'].fillna('No Error', inplace=True)
        # Count the occurrences of distinct errors
        error_counts = host_df['errors'].value_counts()
        colors = ['red', 'green']
        # Create a pie chart
        plt.figure(figsize=(8, 8))
        plt.pie(error_counts, labels=error_counts.index,
                autopct='%1.1f%%', startangle=140, colors=colors)
        plt.title('Distribution of Errors')
        # Equal aspect ratio ensures that pie is drawn as a circle.
        plt.axis('equal')

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources

        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph3.png')

    def _generate_graph4(self, request_url_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph4.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
```

```
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph4.png'

        plt.figure(figsize=(12, 7))
        sns.countplot(data=request_url_df, x='request_path',
                      order=request_url_df['request_path'].value_counts().
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('Path')
        plt.ylabel('Count')
        plt.tight_layout()
        plt.title("Top 20 request path")

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph4.png')

    def _generate_graph5(self, referer_url_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph5.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph5.png'

        plt.figure(figsize=(12, 7))
        sns.countplot(data=referer_url_df, x='referer_path',
                      order=referer_url_df['referer_path'].value_counts().
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('Path')
        plt.ylabel('Count')
        plt.title("Top 20 Paths in REFERER URL")
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph5.png')

    def _generate_graph6(self, referer_url_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph6.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph6.png'
        plt.figure(figsize=(10, 8))
        sns.countplot(data=referer_url_df, x='referer_netloc',
                      order=referer_url_df['referer_netloc'].value_counts(
        plt.xlabel('Count')
        plt.xticks(rotation=45, ha='right')
        plt.ylabel('Network Locations')
        plt.title('Top 10 Network Locations in referer URL')
```

```
        plt.tight_layout()
        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph6.png')

    def _generate_graph7(self, referer_url_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph7.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph7.png'
        plt.figure(figsize=(8, 5))
        sns.countplot(data=referer_url_df, x='referer_scheme')
        plt.xlabel('Referrer Scheme')
        plt.ylabel('Count')
        plt.title('Distribution of Referrer Schemes')
        plt.xticks(rotation=45)
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph7.png')

    def _generate_graph8(self, ua_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph8.pr

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph8.png'
        plt.figure(figsize=(12, 7))
        sns.countplot(data=ua_df, x='ua_family',
                      order=ua_df['ua_family'].value_counts().index[:10])
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('User Agent Family')
        plt.ylabel('Count')
        plt.title('Top 10 User Agent Families')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph8.png')

    def _generate_graph9(self, ua_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph9.pr

        # Check if the image file already exists
```

```
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph9.png'
        plt.figure(figsize=(12, 7))
        sns.countplot(data=ua_df, x='ua_os.family',
                      order=ua_df['ua_os.family'].value_counts().index[:10
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('Operating System')
        plt.ylabel('Count')
        plt.title('Top 10 Operating Systems')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph9.png')

    def _generate_graph10(self, ua_df):

        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph10.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph10.png
        plt.figure(figsize=(12, 7))
        sns.countplot(data=ua_df, x='ua_device.family',
                      order=ua_df['ua_device.family'].value_counts().index
        plt.xticks(rotation=45, ha='right')
        plt.xlabel('User Agent Device Family')
        plt.ylabel('Count')
        plt.title('Top 10 User Agent Device Families')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph10.png')

    def _generate_graph11(self, ua_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph11.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph11.png
        # color palette
        palette = sns.color_palette("tab10")

        plt.figure(figsize=(12, 7))
        grouped_data = ua_df.groupby(
            ['ua_os.family', 'ua_device.brand']).size().unstack(fill_value
```

```
        # Create the stacked bar plot with the defined color palette
        grouped_data.plot(kind='bar', stacked=True,
                          figsize=(12, 7), color=palette)

        plt.xlabel('Operating System')
        plt.ylabel('Count')
        plt.title('Operating System Distribution by Device Brand')
        plt.xticks(rotation=45, ha='right')
        plt.legend(title='Device Brand')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph11.png')

    def _generate_graph12(self, referer_url_df, host_df):

        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph12.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph12.png
        merged_df = referer_url_df.merge(
            host_df, left_index=True, right_index=True)

        plt.figure(figsize=(12, 7))
        top_ip_addresses = merged_df['ip_address'].value_counts().index[:2
        sns.countplot(data=merged_df[merged_df['ip_address'].isin(
            top_ip_addresses)], x='referer_scheme', hue='ip_address')
        plt.xlabel('Referrer Scheme')
        plt.ylabel('Count')
        plt.title('Distribution of Referrer Schemes by IP Address (Top 20
        plt.xticks(rotation=45)
        plt.legend(title='IP Address')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph12.png')

    def _generate_graph13(self, logs_df):

        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph13.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph13.png
        status_df = logs_df.groupby('status').size().reset_index(name='cou
```

```
        plt.figure(figsize=(8, 8))
        plt.pie(status_df['count'], labels=status_df['status'], startangle
        plt.title('Distribution of Status Codes')
        plt.axis('equal')
        plt.legend(loc='lower right')

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph13.png')

    def _generate_graph14(self, logs_df):

        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph14.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph14.png
        # Convert 'datetime' column to datetime format
        logs_df['datetime'] = pd.to_datetime(logs_df['datetime'], utc=True
        # Set 'datetime' column as the DataFrame index
        logs_df.set_index('datetime', inplace=True)

        # Plot 1: Top 10 hits on hourly basis (Bar Plot)
        top_10_hourly_hits = logs_df['client'].resample(
            'H').size().nlargest(10)
        plt.figure(figsize=(12, 7))
        top_10_hourly_hits.plot(kind='bar', title='Top 10 Hits Hourly')
        plt.ylabel('Count')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph14.png')

    def _generate_graph15(self, logs_df):
        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph15.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph15.png
        # Resample at an hourly frequency and calculate hits per hour
        hourly_hits = logs_df.resample('H').size()

        # Plot hits per hour using an area plot
        plt.figure(figsize=(12, 2))
        hourly_hits.plot(
            kind='area', title='Hourly Hits Cumulative Distribution')
        plt.xlabel('Time')
        plt.ylabel('Cumulative Count')
```

```
        plt.xticks(rotation=45)

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph15.png')

    def _generate_graph16(self, logs_df):

        # Define the file path for the graph image
        save_path = os.path.join(settings.MEDIA_ROOT, 'images', 'graph16.p

        # Check if the image file already exists
        if os.path.exists(save_path):
            # If the file exists, return the URL path of the existing imag
            return os.path.join(settings.MEDIA_URL, 'images', 'graph16.png
            # Resample at a minute-level frequency and calculate hits per
        hits_per_minute = logs_df.resample('T').size()
        # Plot hits per minute
        plt.figure(figsize=(12, 6))
        hits_per_minute.plot(kind='line')
        plt.xlabel('Time')
        plt.ylabel('Hits')
        plt.title('Hits per Minute')
        plt.tight_layout()

        # Save the plot to the media/images directory
        plt.savefig(save_path)
        plt.close()  # Close the plot to free up resources
        # Return the URL path of the generated graph
        return os.path.join(settings.MEDIA_URL, 'images', 'graph16.png')

    def unique_ip_address(self, host_df):
        unique_ip_address = host_df['ip_address'].nunique()
        return unique_ip_address

    def most_common_ips(self, host_df):
        most_common_ips = host_df['ip_address'].value_counts().head(10)
        return most_common_ips

    def unique_hostname_count(self, host_df):
        unique_hostname_count = host_df['hostname'].nunique()
        return unique_hostname_count

    def most_common_hostnames(self, host_df):
        most_common_hostnames = host_df['hostname'].value_counts().head(10
        return most_common_hostnames

    def unique_error_count(self, host_df):
        unique_error_count = host_df['errors'].nunique()
        return unique_error_count

    def most_common_query(self, request_url_df):
        most_common_query = request_url_df['request_query'].value_counts()
            10)
        return most_common_query
```

```
    def most_common_paths(self, request_url_df):
        most_common_paths = request_url_df['request_path'].value_counts().
            10)
        return most_common_paths

    def status_df(self, logs_df):
        status_df = logs_df.groupby('status').size().reset_index(name='cou
        return status_df
```

In [4]:

```
from django import forms

class LogAnalysisForm(forms.Form):
    log_file = forms.FileField(label="Upload Log File" ,widget=forms.FileI


class userCheckboxes(forms.Form):
    Ip_Address=forms.BooleanField(label='Ip Address', required=False,widge
    HostName = forms.BooleanField(label='HostName', required=False,widget=
    Error = forms.BooleanField(label='Error', required=False,widget=forms.
    RequestURL = forms.BooleanField(label='Request URL', required=False,wi
    RefererURL = forms.BooleanField(label='Referer URL', required=False,wi
    UserAgent = forms.BooleanField(label='User Agent', required=False,widg
    StatusCode = forms.BooleanField(label='Status Code', required=False,wi
    Additional = forms.BooleanField(label='Additional', required=False,wid

class PersonalInfo(forms.Form):

    name = forms.CharField(widget=forms.TextInput(attrs={'id': 'i','class'
    Email = forms.EmailField(widget=forms.EmailInput(attrs={'id': 'i','cla
    phoneNumber = forms.IntegerField(widget=forms.NumberInput(attrs={'id':
```

In [1]:

```
"""
URL configuration for Log project.

The urlpatterns list routes URLs to views. For more information please see
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, pat
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from Log import views
# for media
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.homeForm),
    path('report/', views.Report, name='report_view')
]

if settings.DEBUG:
    urlpatterns+=static(settings.MEDIA_URL,document_root=settings.MEDIA_RO
```

```
---------------------------------------------------------------------------
---
ModuleNotFoundError                       Traceback (most recent call la
st)
Cell In[1], line 19
     17 from django.contrib import admin
     18 from django.urls import path
---> 19 from Log import views
     20 # for media
     21 from django.conf import settings

ModuleNotFoundError: No module named 'Log'
```

In [2]:

```
from .file import LogAnalyser
import pyarrow.parquet as pq
import tempfile
from datetime import datetime
from Log import settings
from django.http import HttpResponse, HttpResponseRedirect
from django.shortcuts import render, redirect
# importing form
from .forms import LogAnalysisForm, userCheckboxes, PersonalInfo
from django.urls import reverse
import logging
from LogFile.models import FileandCheckboxes
import os
import advertools as adv
import pandas as pd
import pyarrow.parquet as pq
from io import BytesIO
import seaborn as sns
import matplotlib.pyplot as plt
from ua_parser import user_agent_parser
import ua_parser
from urllib.parse import urlparse
import base64
# sets no limit to number of columns displayed
pd.options.display.max_columns = None


def homeForm(request):
    # storing form in variable
    if request.method == 'POST':
        # Handle form submission here
        formUpload = LogAnalysisForm(request.POST, request.FILES)
        formCheckboxes = userCheckboxes(request.POST)
        Info = PersonalInfo(request.POST)

        if formUpload.is_valid() and formCheckboxes.is_valid() and Info.is
            return HttpResponseRedirect(reverse('report_view'))
        else:
            # Print form errors for debugging
            print(formUpload.errors)
            print(formCheckboxes.errors)
            print(Info.errors)

    else:
        # Render the form if it's a GET request
        formUpload = LogAnalysisForm()
        formCheckboxes = userCheckboxes()
        Info = PersonalInfo()

    data = {'form1': formUpload, 'form2': formCheckboxes, 'form3': Info}

    return render(request, "home.html", data)

    # data ={} add here dynamic data and pass in render
    # use a for loop in html file to show normal elemnts of list
    # [list and andar dict h ]to display dictionaly method is through tabl
    # if condyion bhi laga sakte h .... now check kaise dekhe ki only chec
```

```
    # Render ki jagah httpresponseredirect krdo and a url like this
    # return HttpResponseRedirect('/url/')


def Report(request):
    context = {
        'graph1_data': None,
        'common_ips': None,
        'unique_ip_address': None,
        'graph2_data': None,
        'most_common_hostnames': None,
        'unique_hostname_count': None,
        'graph3_data': None,
        'unique_error_count': None,
        'graph4_data': None,
        'most_common_querry': None,
        'most_common_paths': None,
        'graph5_data': None,
        'graph6_data': None,
        'graph7_data': None,
        'graph8_data': None,
        'graph9_data': None,
        'graph10_data': None,
        'graph11_data': None,
        'graph12_data': None,
        'status_df': None,
        'graph13_data': None,
        'graph14_data': None,
        'graph15_data': None,
        'graph16_data': None,
        'error_message': None,
    }
    if request.method == 'POST':
        # Handle form submission here
        formUpload = LogAnalysisForm(request.POST, request.FILES)
        formCheckboxes = userCheckboxes(request.POST)
        Info = PersonalInfo(request.POST)

        if formUpload.is_valid() and formCheckboxes.is_valid() and Info.is
            # Save form data to the model
            data = FileandCheckboxes(
                # Assuming your file input field has the name 'log_file'
                log_file=request.FILES['log_file'],
                ip_address=formCheckboxes.cleaned_data['Ip_Address'],
                hostname=formCheckboxes.cleaned_data['HostName'],
                error=formCheckboxes.cleaned_data['Error'],
                request_url=formCheckboxes.cleaned_data['RequestURL'],
                referer_url=formCheckboxes.cleaned_data['RefererURL'],
                user_agent=formCheckboxes.cleaned_data['UserAgent'],
                status_code=formCheckboxes.cleaned_data['StatusCode'],
                additional=formCheckboxes.cleaned_data['Additional'],
                name=Info.cleaned_data['name'],
                email=Info.cleaned_data['Email'],
                phone_number=Info.cleaned_data['phoneNumber']
            )
            data.save()  # Save the data to the model
```

```
            selected_options = []
            for field_name, field_value in formCheckboxes.cleaned_data.ite
                if field_value is True:
                    selected_options.append(field_name)

            # Construct the absolute path to the Parquet log file in the '
            media_root = settings.MEDIA_ROOT
            output_file_rel_path = 'log/output_file.parquet'
            output_file = os.path.join(media_root, output_file_rel_path)

            # CLASS INSTANCE
            m = LogAnalyser()
            if os.path.exists(output_file):
                logs_df, host_df, request_url_df, referer_url_df, ua_df =
                    output_file)

                selected_options = formCheckboxes.cleaned_data.get(
                    'options', [])

                # Rest of your code for generating graphs and context

                if 'Ip_Address' in selected_options:

                    unique_ip_address = m.unique_ip_address(host_df)
                    graph1_data = m._generate_graph1(host_df)
                    most_common_ips = m.most_common_ips(host_df)

                    # Include the graph data and analysis results in the c
                    context['graph1_data'] = graph1_data
                    context['common_ips'] = most_common_ips
                    context['unique_ip_address'] = unique_ip_address

                if 'HostName' in selected_options:
                    unique_hostname_count = m.unique_hostname_count(host_d
                    graph2_data = m._generate_graph2(host_df)
                    most_common_hostnames = m.most_common_hostnames(host_d

                    context['unique_hostname_count'] = unique_hostname_cou
                    context['graph2_data'] = graph2_data
                    context['most_common_hostnames'] = most_common_hostnam

                if 'Error' in selected_options:

                    unique_error_count = m.unique_error_count(host_df)
                    graph3_data = m._generate_graph3(host_df)

                    context['unique_error_count'] = unique_error_count
                    context['graph3_data'] = graph3_data

                if 'RequestURL' in selected_options:
                    graph4_data = m._generate_graph4(request_url_df)
                    most_common_query = m.most_common_query(request_url_df
                    most_common_paths = m.most_common_paths(request_url_df

                    context['graph4_data'] = graph4_data
                    context['most_common_querry'] = most_common_query
```

```
                    context['most_common_paths'] = most_common_paths

                if 'RefererURL' in selected_options:
                    graph5_data = m._generate_graph5(referer_url_df)
                    graph6_data = m._generate_graph6(referer_url_df)
                    graph7_data = m._generate_graph7(referer_url_df)

                    context['graph5_data'] = graph5_data
                    context['graph6_data'] = graph6_data
                    context['graph7_data'] = graph7_data

                if 'UserAgent' in selected_options:
                    graph8_data = m._generate_graph8(ua_df)
                    graph9_data = m._generate_graph9(ua_df)
                    graph10_data = m._generate_graph10(ua_df)
                    graph11_data = m._generate_graph11(ua_df)
                    graph12_data = m._generate_graph12(referer_url_df, hos

                    context['graph8_data'] = graph8_data
                    context['graph9_data'] = graph9_data
                    context['graph10_data'] = graph10_data
                    context['graph11_data'] = graph11_data
                    context['graph12_data'] = graph12_data

                if 'StatusCode' in selected_options:
                    status_df = m.status_df(logs_df)
                    graph13_data = m._generate_graph13(logs_df)

                    context['status_df'] = status_df
                    context['graph13_data'] = graph13_data

                if 'Additional' in selected_options:
                    graph14_data = m._generate_graph14(logs_df)
                    graph15_data = m._generate_graph15(logs_df)
                    graph16_data = m._generate_graph16(logs_df)

                    context['graph14_data'] = graph14_data
                    context['graph15_data'] = graph15_data
                    context['graph16_data'] = graph16_data
            else:
                # Handle the case where the file does not exist
                context['error_message'] = 'Log file does not exist.'

    else:
        formUpload = LogAnalysisForm()
        formCheckboxes = userCheckboxes()
        Info = PersonalInfo()
        data = {'form1': formUpload, 'form2': formCheckboxes, 'form3': Inf
        return render(request, "home.html", data)

    return render(request, "report.html", context)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call la
st)
Cell In[2], line 1
----> 1 from .file import LogAnalyser
      2 import pyarrow.parquet as pq
      3 import tempfile

ImportError: attempted relative import with no known parent package
```

In [ ]: