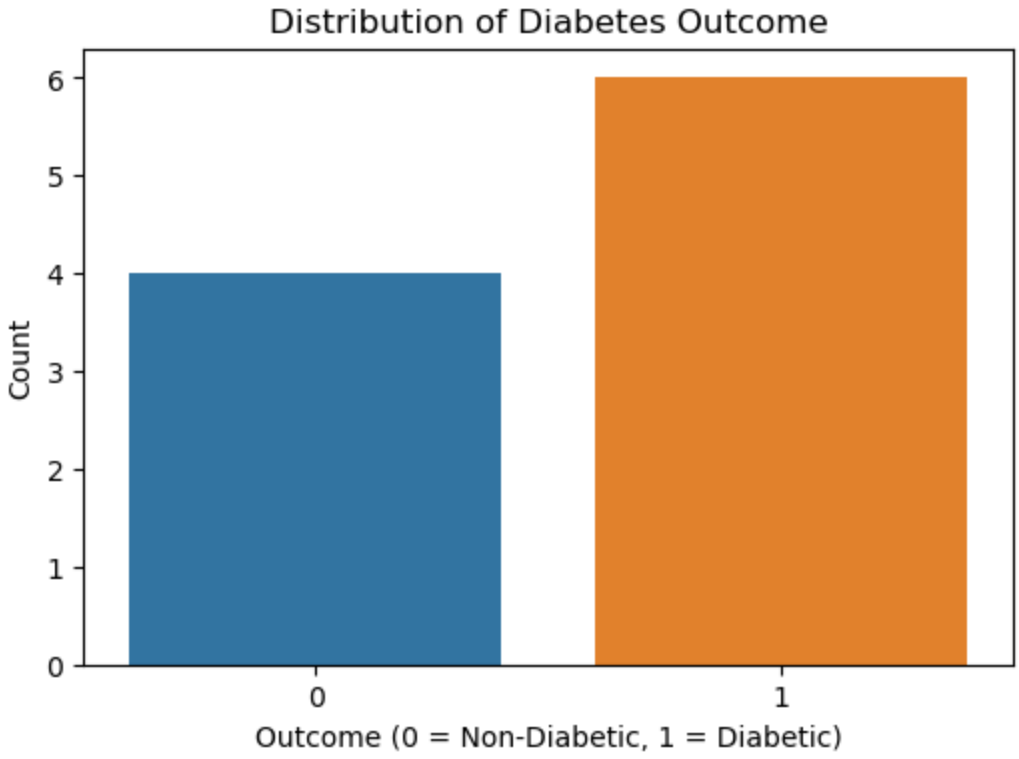
1) Write a Python program to collect, load, and perform initial exploration of the Diabetes
dataset using Pandas. Display the first few records of the dataset and summarize its
structure and contents.

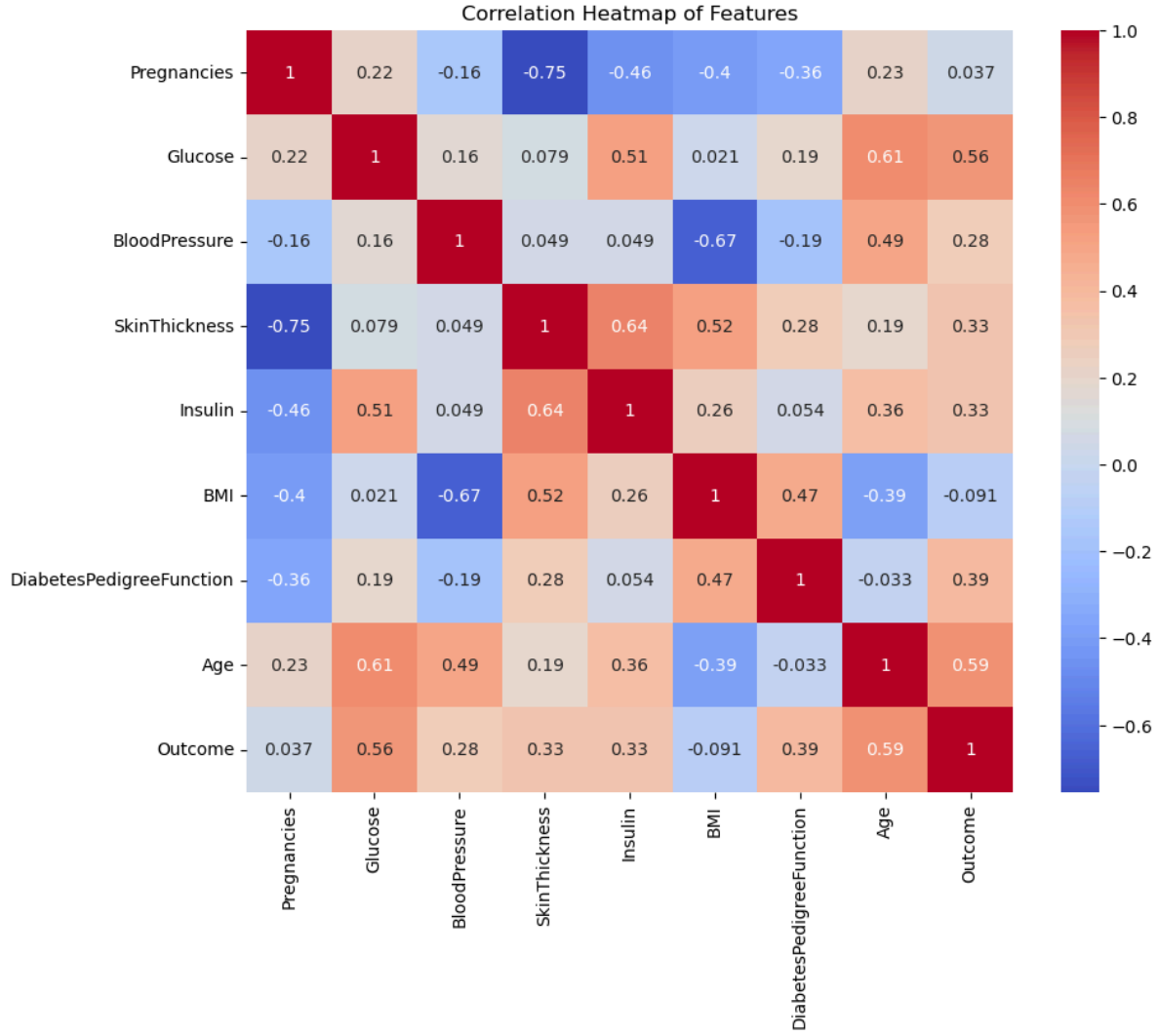
# Exp No: 5 - Data Collection and Initial Exploration
# Objective: To collect, load, and perform initial exploration of the diabetes dataset

In [6]:
```
# Exp No: 5 - Data Collection and Initial Exploration
# Objective: To collect, load, and perform initial exploration of the diabetes

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Create the diabetes.csv file manually
data = {
    "Pregnancies": [6,1,8,1,0,5,3,10,2,8],
    "Glucose": [148,85,183,89,137,116,78,115,197,125],
    "BloodPressure": [72,66,64,66,40,74,50,0,70,96],
    "SkinThickness": [35,29,0,23,35,0,32,0,45,0],
    "Insulin": [0,0,0,94,168,0,88,0,543,0],
    "BMI": [33.6,26.6,23.3,28.1,43.1,25.6,31.0,35.3,30.5,0.0],
    "DiabetesPedigreeFunction": [0.627,0.351,0.672,0.167,2.288,0.201,0.248,0.13
    "Age": [50,31,32,21,33,30,26,29,53,54],
    "Outcome": [1,0,1,0,1,0,1,0,1,1]
}

df = pd.DataFrame(data)
df.to_csv("diabetes.csv", index=False)
print("diabetes.csv file created successfully!")

# Step 2: Load the dataset from the created CSV file
diabetes_df = pd.read_csv("diabetes.csv")

# Step 3: Display the first few rows
print("\nFirst 5 rows of the dataset:")
print(diabetes_df.head())

# Step 4: Display basic information about the dataset
print("\nDataset Information:")
print(diabetes_df.info())

# Step 5: Display statistical summary
print("\nStatistical Summary:")
print(diabetes_df.describe())

# Step 6: Check for missing values
print("\nMissing Values in Each Column:")
print(diabetes_df.isnull().sum())

# Step 7: Visualize the distribution of the target variable (Outcome)
plt.figure(figsize=(6,4))
sns.countplot(x='Outcome', data=diabetes_df)
plt.title('Distribution of Diabetes Outcome')
plt.xlabel('Outcome (0 = Non-Diabetic, 1 = Diabetic)')
plt.ylabel('Count')
plt.show()

# Step 8: Display correlation heatmap
plt.figure(figsize=(10,8))
sns.heatmap(diabetes_df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```

```
diabetes.csv file created successfully!

First 5 rows of the dataset:
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               10 non-null     int64
 1   Glucose                   10 non-null     int64
 2   BloodPressure             10 non-null     int64
 3   SkinThickness             10 non-null     int64
 4   Insulin                   10 non-null     int64
 5   BMI                       10 non-null     float64
 6   DiabetesPedigreeFunction  10 non-null     float64
 7   Age                       10 non-null     int64
 8   Outcome                   10 non-null     int64
dtypes: float64(2), int64(7)
memory usage: 848.0 bytes
None

Statistical Summary:
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count     10.00000   10.000000      10.000000      10.000000   10.000000
mean       4.40000  127.300000      59.800000      19.900000   89.300000
std        3.50238   40.075068      25.654976      17.978073  169.937276
min        0.00000   78.000000       0.000000       0.000000    0.000000
25%        1.25000   95.500000      53.500000       0.000000    0.000000
50%        4.00000  120.500000      66.000000      26.000000    0.000000
75%        7.50000  145.250000      71.500000      34.250000   92.500000
max       10.00000  197.000000      96.000000      45.000000  543.000000

             BMI  DiabetesPedigreeFunction        Age    Outcome
count  10.000000                 10.000000  10.000000  10.000000
mean   27.710000                  0.507800  35.900000   0.600000
std    11.259016                  0.654114  11.873874   0.516398
min     0.000000                  0.134000  21.000000   0.000000
25%    25.850000                  0.175500  29.250000   0.000000
50%    29.300000                  0.240000  31.500000   1.000000
75%    32.950000                  0.558000  45.750000   1.000000
max    43.100000                  2.288000  54.000000   1.000000
```

```
Missing Values in Each Column:
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```





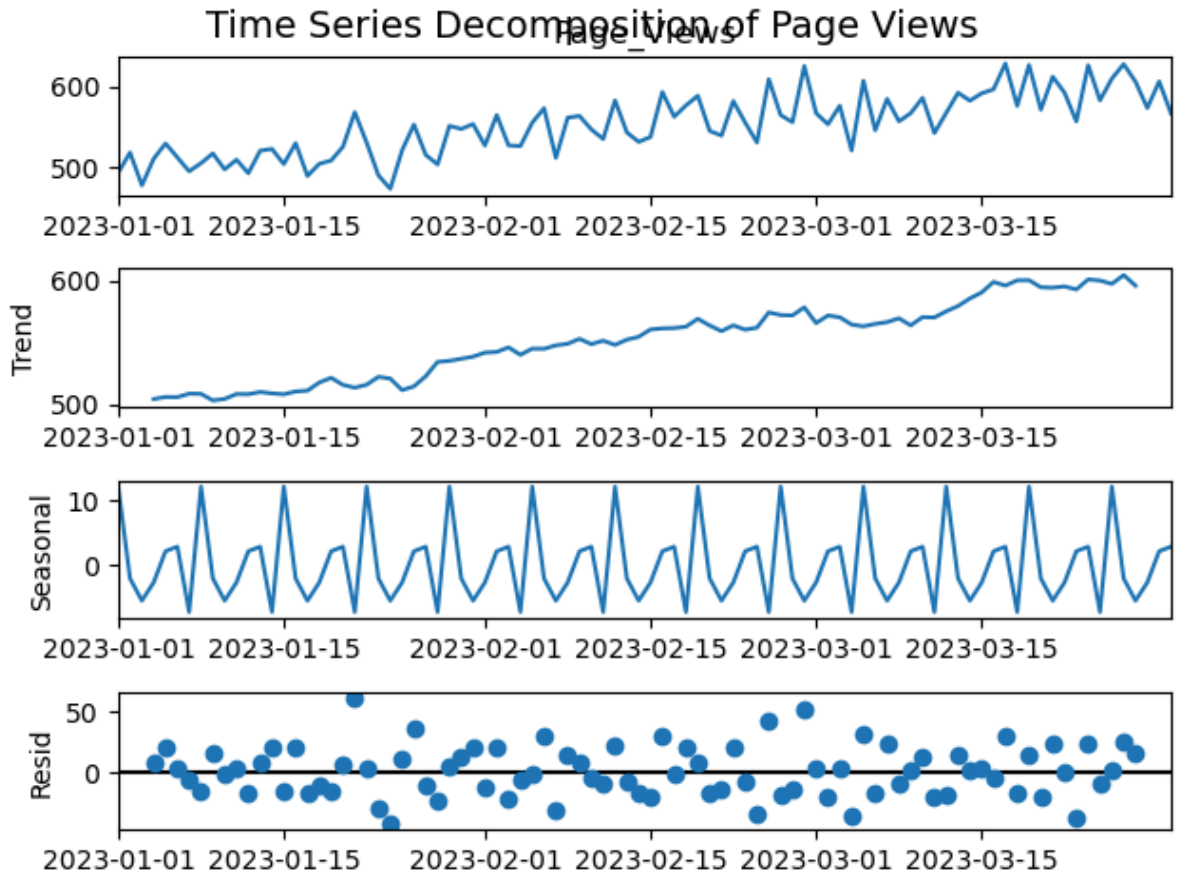

2.Write a Python program to perform Time Series Analysis on a website traffic dataset. The
program should load and clean the dataset, decompose the time series to identify trends
and seasonality, visualize key metrics using moving averages and seasonal plots, and detect
anomalies using statistical methods.
Dataset:
A dataset containing daily website traffic data with the following columns:
• Date – Date of observation
• Page_Views – Total number of page views per day
• Unique_Visitors – Number of unique visitors per day
• Bounce_Rate – Percentage of visitors who leave after viewing only one page
(You can create a sample CSV file like website_traffic.csv for practice.)
Methodology:
1. Load and Clean Data:
o Import the dataset and parse the Date column as a datetime object.
o Handle missing values by interpolation.
2. Time Series Decomposition:
o Decompose the time series data of Page_Views to identify trend, seasonality, and
residuals.
3. Visualization:
o Plot the time series data for Page_Views, Unique_Visitors, and

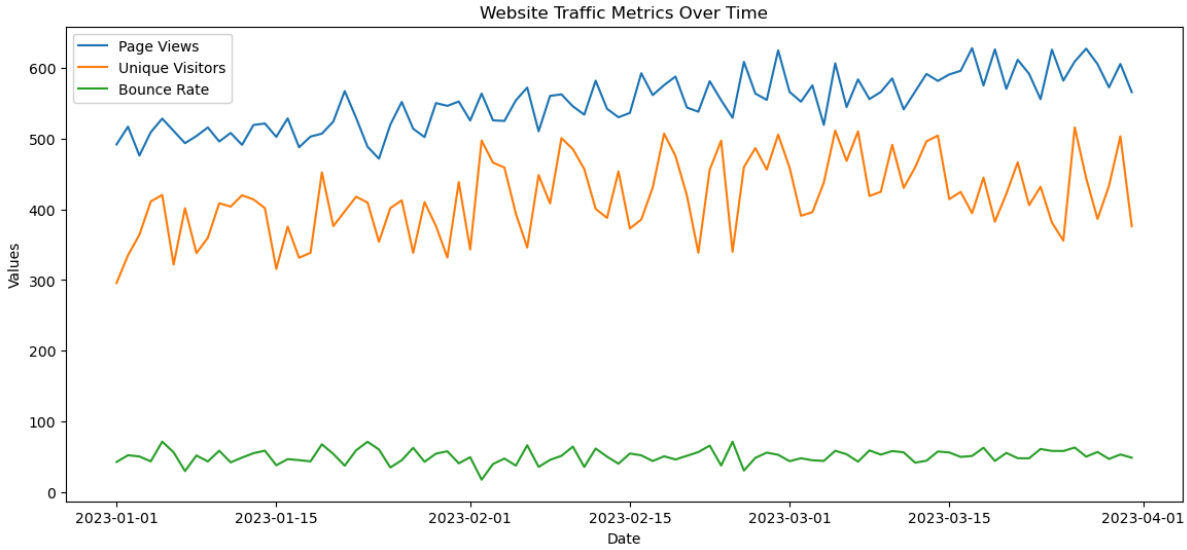
Bounce_Rate.
o Use moving averages to smooth the time series and highlight trends.
o Create seasonal plots to visualize patterns over time.
4. Anomaly Detection:
o Identify and visualize anomalies in the time series data using statistical methods such as
the Z-score.

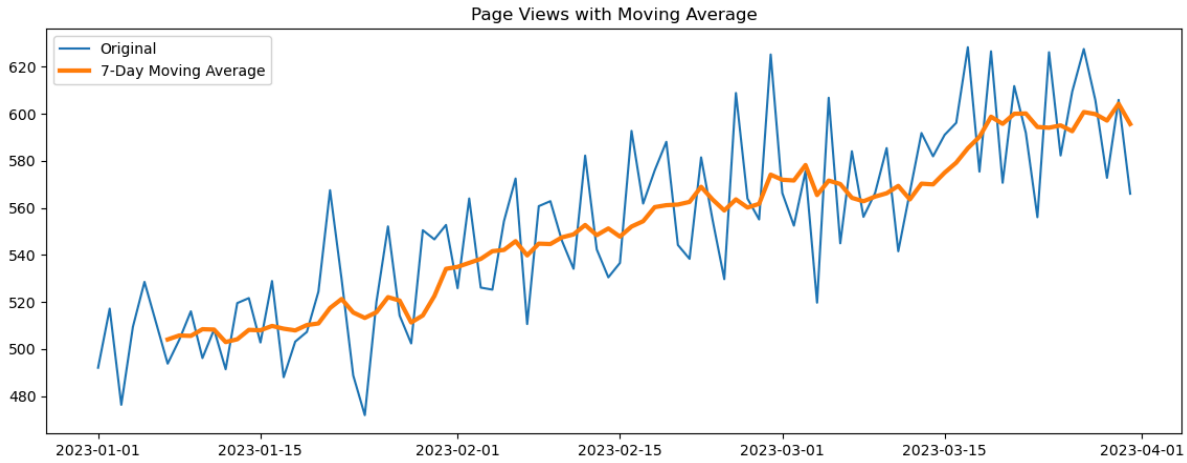
In [7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy import stats

# --- Step 0: Generate Synthetic Website Traffic Data and Save as CSV ---

# Create a date range for 90 days
dates = pd.date_range(start='2023-01-01', periods=90, freq='D')

# Generate synthetic data with float arrays to allow NaN
np.random.seed(42)
page_views = (np.random.poisson(lam=500, size=90) + np.linspace(0, 100, 90)).as
unique_visitors = (page_views * np.random.uniform(0.6, 0.9, size=90)).astype(fl
bounce_rate = np.random.normal(loc=50, scale=10, size=90)

# Introduce some missing values
page_views[5] = np.nan
unique_visitors[20] = np.nan
bounce_rate[50] = np.nan

# Create dataframe
df_sample = pd.DataFrame({
    'Date': dates,
    'Page_Views': page_views,
    'Unique_Visitors': unique_visitors,
    'Bounce_Rate': bounce_rate
})

# Save to CSV
df_sample.to_csv('website_traffic.csv', index=False)
print("Sample 'website_traffic.csv' created successfully!")

# --- Step 1: Load and Clean Data ---

# Load dataset with Date parsing
df = pd.read_csv('website_traffic.csv', parse_dates=['Date'])

# Sort by Date
df = df.sort_values('Date')

# Set Date as index
df.set_index('Date', inplace=True)

# Handle missing values with interpolation
df = df.interpolate()

print("\nFirst few rows of the dataset:")
print(df.head())

print("\nDataset Information:")
print(df.info())

# --- Step 2: Time Series Decomposition (Page_Views) ---
```

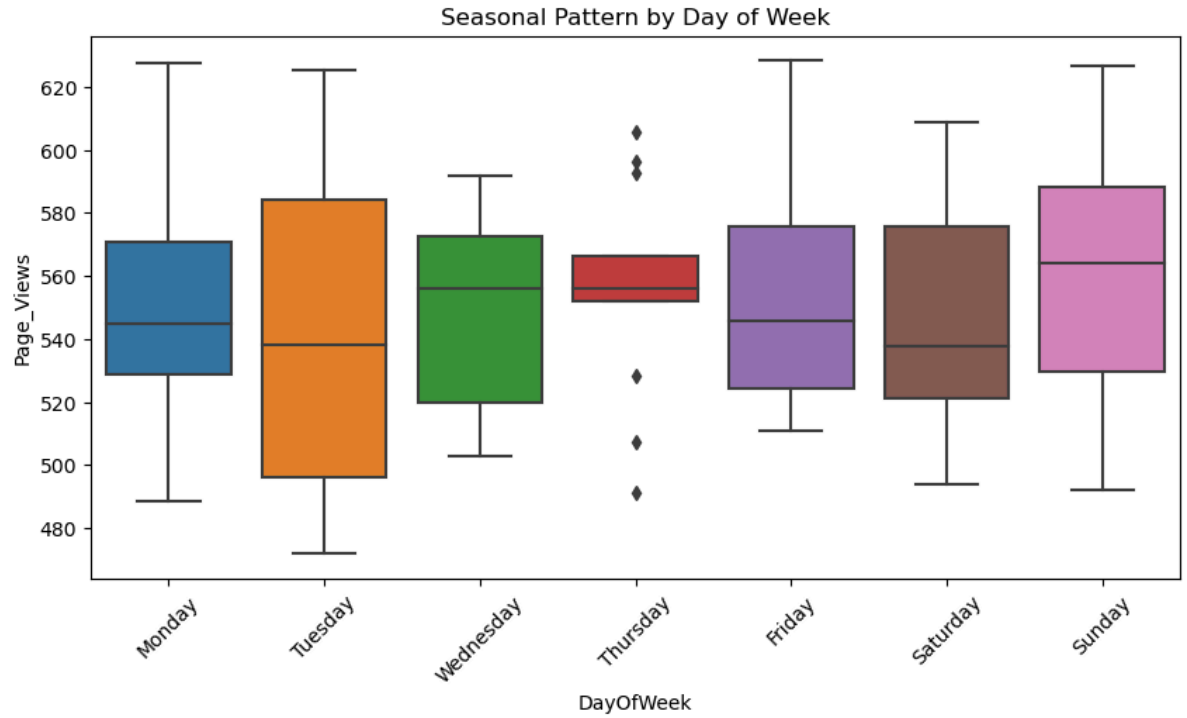
```
decomposition = seasonal_decompose(df['Page_Views'], model='additive', period=7

plt.figure(figsize=(12, 8))
decomposition.plot()
plt.suptitle('Time Series Decomposition of Page Views', fontsize=14)
plt.show()

# --- Step 3: Visualization ---

# (a) Plot all metrics
plt.figure(figsize=(14, 6))
plt.plot(df.index, df['Page_Views'], label='Page Views')
plt.plot(df.index, df['Unique_Visitors'], label='Unique Visitors')
plt.plot(df.index, df['Bounce_Rate'], label='Bounce Rate')
plt.title('Website Traffic Metrics Over Time')
plt.xlabel('Date')
plt.ylabel('Values')
plt.legend()
plt.show()

# (b) Moving Average on Page Views
df['PageViews_MA7'] = df['Page_Views'].rolling(window=7).mean()

plt.figure(figsize=(14, 5))
plt.plot(df.index, df['Page_Views'], label='Original')
plt.plot(df.index, df['PageViews_MA7'], label='7-Day Moving Average', linewidth
plt.title('Page Views with Moving Average')
plt.legend()
plt.show()

# (c) Seasonal plot - Weekly pattern of Page Views
df['DayOfWeek'] = df.index.day_name()

plt.figure(figsize=(10, 5))
sns.boxplot(x='DayOfWeek', y='Page_Views', data=df.reset_index(),
            order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Sat
plt.title('Seasonal Pattern by Day of Week')
plt.xticks(rotation=45)
plt.show()

# --- Step 4: Anomaly Detection using Z-score ---

df['Z_Score'] = np.abs(stats.zscore(df['Page_Views']))

# Define anomalies as points with Z-score > 3
anomalies = df[df['Z_Score'] > 3]

plt.figure(figsize=(14, 5))
plt.plot(df.index, df['Page_Views'], label='Page Views')
plt.scatter(anomalies.index, anomalies['Page_Views'], color='red', label='Anoma
plt.title('Anomaly Detection in Page Views')
plt.legend()
plt.show()

print("\nDetected Anomalies:")
print(anomalies[['Page_Views', 'Z_Score']])
```

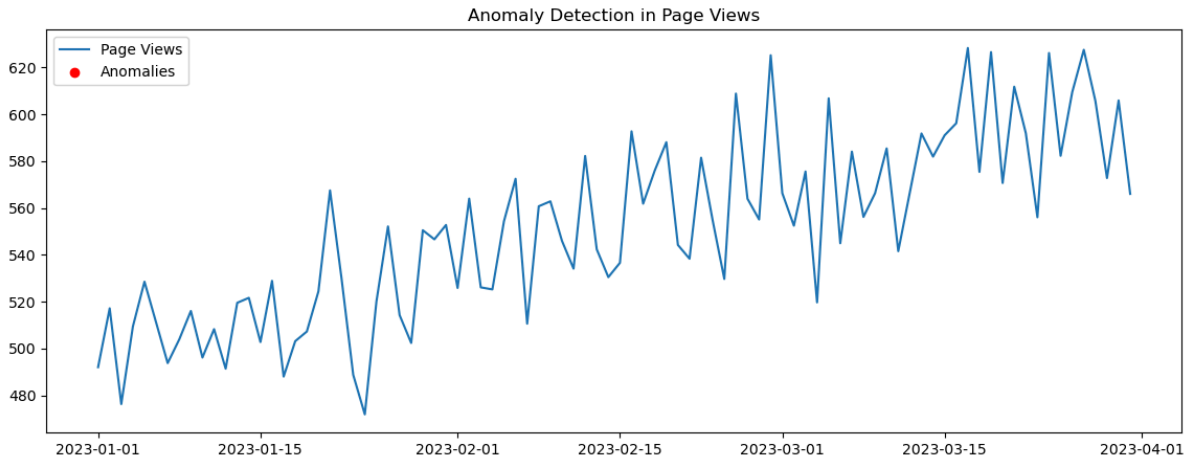
```
Sample 'website_traffic.csv' created successfully!

First few rows of the dataset:
            Page_Views  Unique_Visitors  Bounce_Rate
Date
2023-01-01  492.000000       295.947090    42.696334
2023-01-02  517.123596       335.221449    52.164586
2023-01-03  476.247191       364.148192    50.455718
2023-01-04  509.370787       411.351832    43.483997
2023-01-05  528.494382       420.463988    71.439441

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 90 entries, 2023-01-01 to 2023-03-31
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Page_Views       90 non-null     float64
 1   Unique_Visitors  90 non-null     float64
 2   Bounce_Rate      90 non-null     float64
dtypes: float64(3)
memory usage: 2.8 KB
None

<Figure size 1200x800 with 0 Axes>
```




Website Traffic Metrics Over Time
Page Views
Unique Visitors
Bounce Rate
Values
Date
0
100
200
300
400
500
600
2023-01-01
2023-01-15
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01


Page Views with Moving Average
Original
7-Day Moving Average
480
500
520
540
560
580
600
620
2023-01-01
2023-01-15
2023-02-01
2023-02-15
2023-03-01
2023-03-15
2023-04-01


Seasonal Pattern by Day of Week
Page_Views
DayOfWeek
480
500
520
540
560
580
600
620
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday




```
Detected Anomalies:
Empty DataFrame
Columns: [Page_Views, Z_Score]
Index: []
```

### 3) Random Sampling and Sampling Distribution

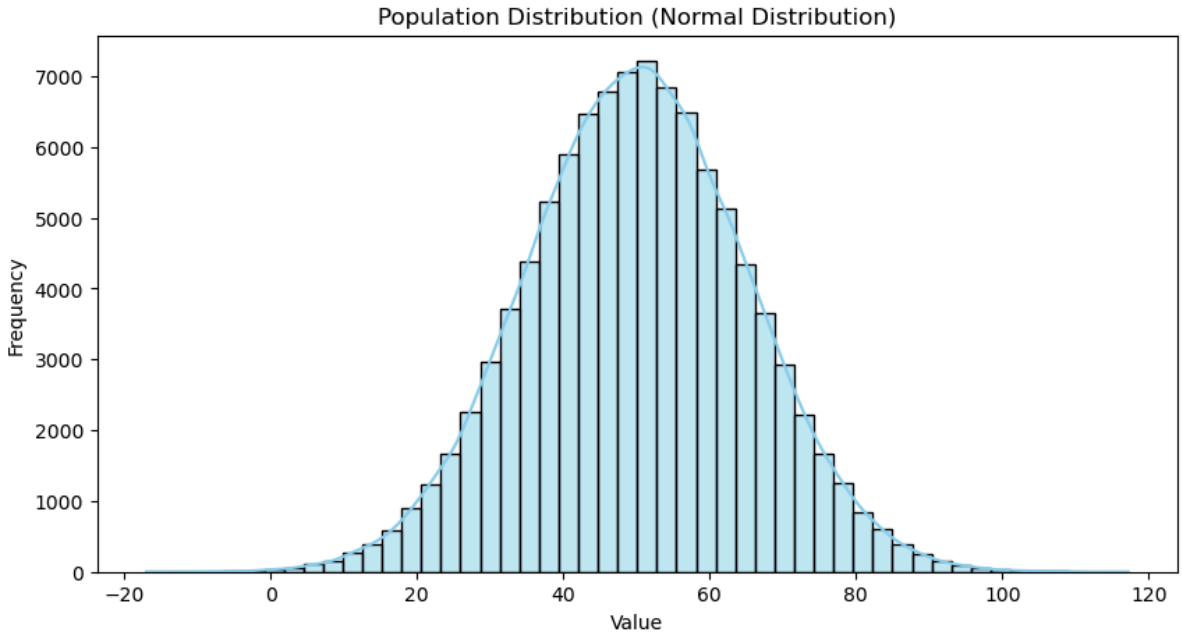
To explore random sampling from a population and understand the concept of sampling
distribution using Python in Jupyter Notebook.

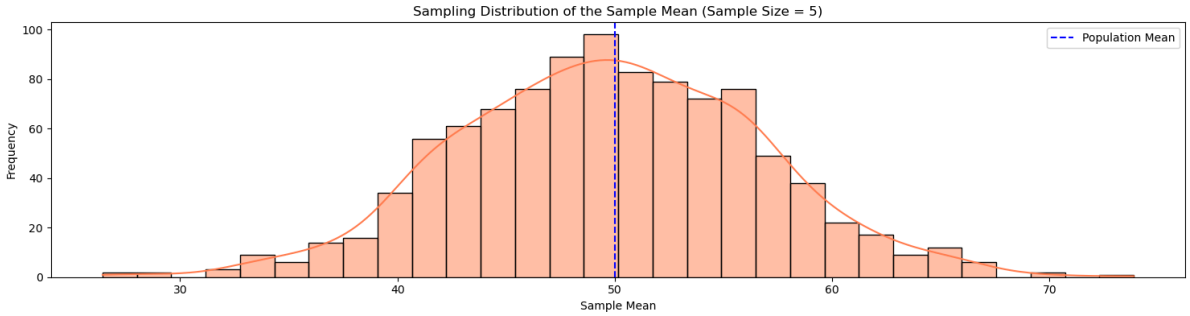
Steps:

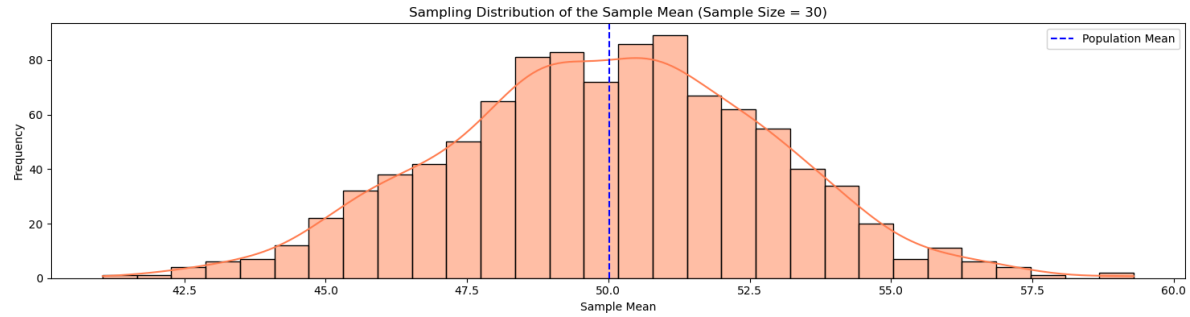
1. Generate a Population:
   - Create a population of data with a specified distribution (e.g., normal distribution).
2. Random Sampling:
   - Perform random sampling from the population to create multiple samples of different sizes.
   - Compute sample statistics (mean, standard deviation, etc.) for each sample.
3. Sampling Distribution:
   - Plot histograms or density plots of sample statistics (e.g., sample means).
   - Compare the sampling distribution of the sample statistic (mean) with the population distribution.
4. Central Limit Theorem (Optional):
   - Demonstrate the Central Limit Theorem by showing that as sample size increases, the sampling distribution of the sample mean approaches a normal distribution regardless of the population distribution.

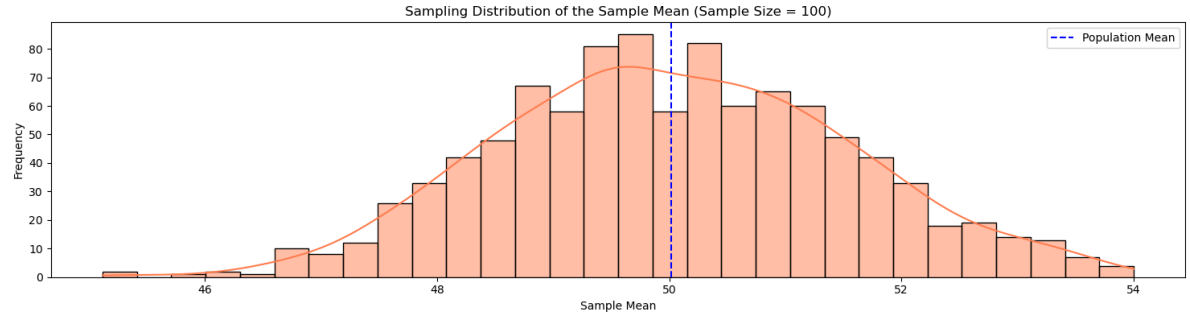
In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Generate a Population
np.random.seed(42)  # For reproducibility

population_size = 100000
# Population from a normal distribution with mean=50, std=15
population = np.random.normal(loc=50, scale=15, size=population_size)

plt.figure(figsize=(10, 5))
sns.histplot(population, bins=50, kde=True, color='skyblue')
plt.title("Population Distribution (Normal Distribution)")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

# Step 2 & 3: Random Sampling and Sampling Distribution
sample_sizes = [5, 30, 100]
num_samples = 1000  # Number of samples to draw for each sample size

plt.figure(figsize=(15, 12))

for i, size in enumerate(sample_sizes, 1):
    sample_means = []

    for _ in range(num_samples):
        sample = np.random.choice(population, size=size, replace=False)
        sample_means.append(np.mean(sample))

    plt.subplot(3, 1, i)
    sns.histplot(sample_means, bins=30, kde=True, color='coral')
    plt.title(f"Sampling Distribution of the Sample Mean (Sample Size = {size})
    plt.xlabel("Sample Mean")
    plt.ylabel("Frequency")
    plt.axvline(np.mean(population), color='blue', linestyle='--', label='Popul
    plt.legend()

plt.tight_layout()
plt.show()

# Step 4: Central Limit Theorem Demonstration with Non-Normal Population
# Generate a population with a uniform distribution (non-normal)
uniform_population = np.random.uniform(low=0, high=100, size=population_size)

plt.figure(figsize=(10, 5))
sns.histplot(uniform_population, bins=50, kde=True, color='lightgreen')
plt.title("Population Distribution (Uniform Distribution)")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(15, 12))

for i, size in enumerate(sample_sizes, 1):
    sample_means = []
```

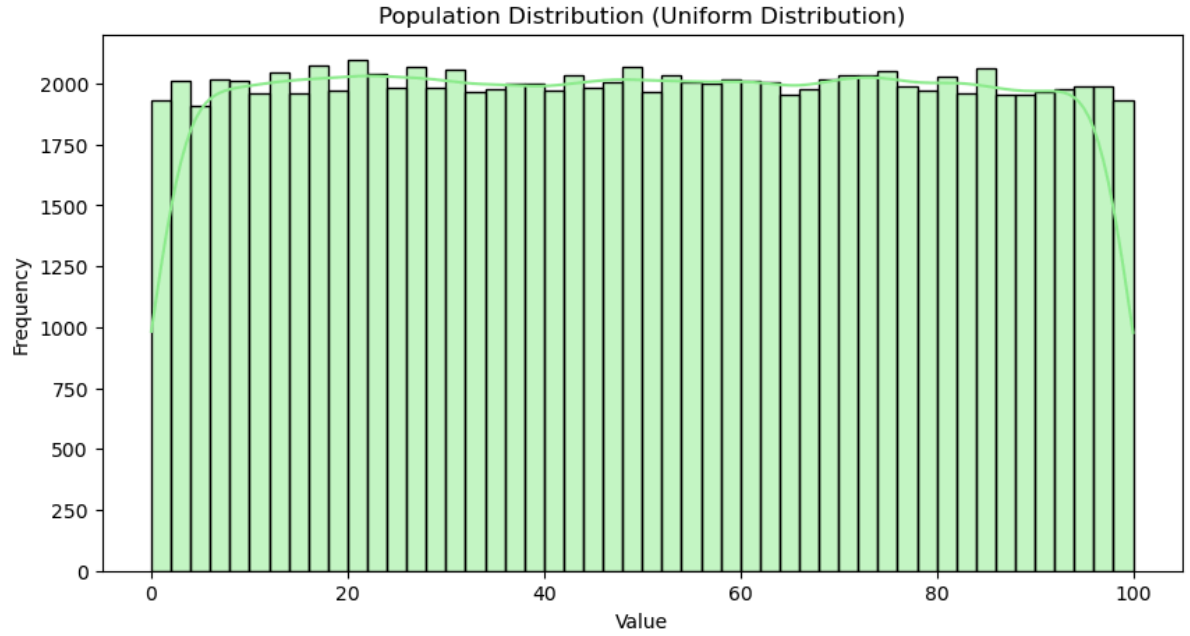
```
    for _ in range(num_samples):
        sample = np.random.choice(uniform_population, size=size, replace=False)
        sample_means.append(np.mean(sample))

    plt.subplot(3, 1, i)
    sns.histplot(sample_means, bins=30, kde=True, color='purple')
    plt.title(f"Sampling Distribution of the Sample Mean (Sample Size = {size})")
    plt.xlabel("Sample Mean")
    plt.ylabel("Frequency")
    plt.axvline(np.mean(uniform_population), color='blue', linestyle='--', label
    plt.legend()

plt.tight_layout()
plt.show()
```

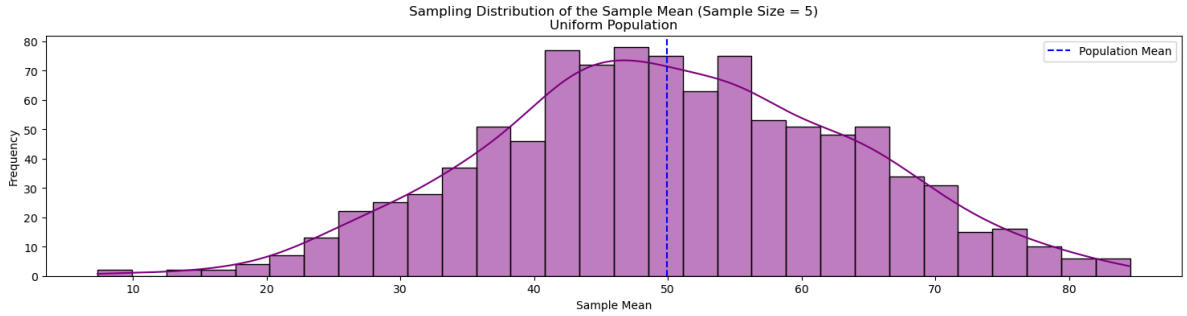


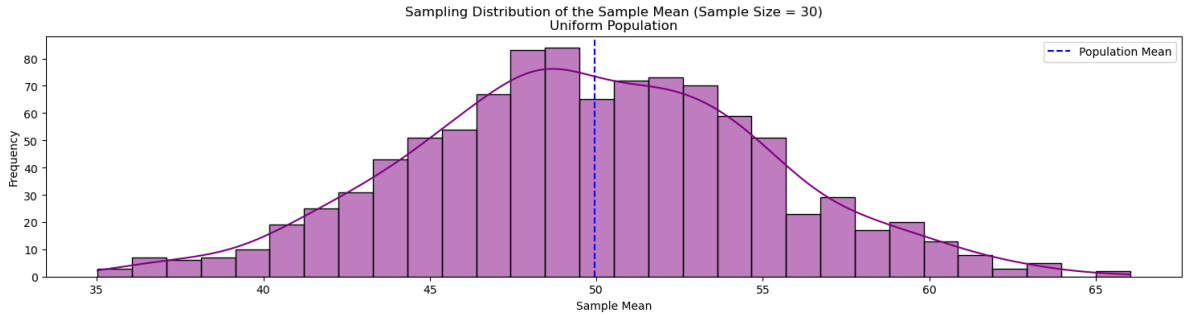

Sampling Distribution of the Sample Mean (Sample Size = 5)
Population Mean
Frequency
Sample Mean

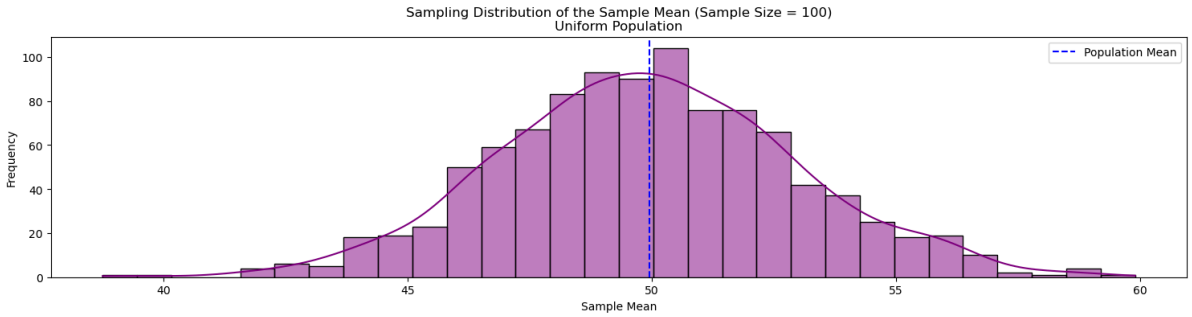

Sampling Distribution of the Sample Mean (Sample Size = 30)
Population Mean
Frequency
Sample Mean


Sampling Distribution of the Sample Mean (Sample Size = 100)
Population Mean
Frequency
Sample Mean


Population Distribution (Uniform Distribution)
Frequency
Value


Sampling Distribution of the Sample Mean (Sample Size = 5)
Uniform Population
Population Mean
Frequency
Sample Mean


Sampling Distribution of the Sample Mean (Sample Size = 30)
Uniform Population
Population Mean
Frequency
Sample Mean


Sampling Distribution of the Sample Mean (Sample Size = 100)
Uniform Population
Population Mean
Frequency
Sample Mean