Financial_Records_of_London_and_UK

September 11, 2024

1 BY - JAYENT SINGH PARIHAR

1.1 Project - Financial Records of London and UK

1.2 Identify Causes of Revenue Fluctuations

1.2.1 Load necessary libraries and dataset

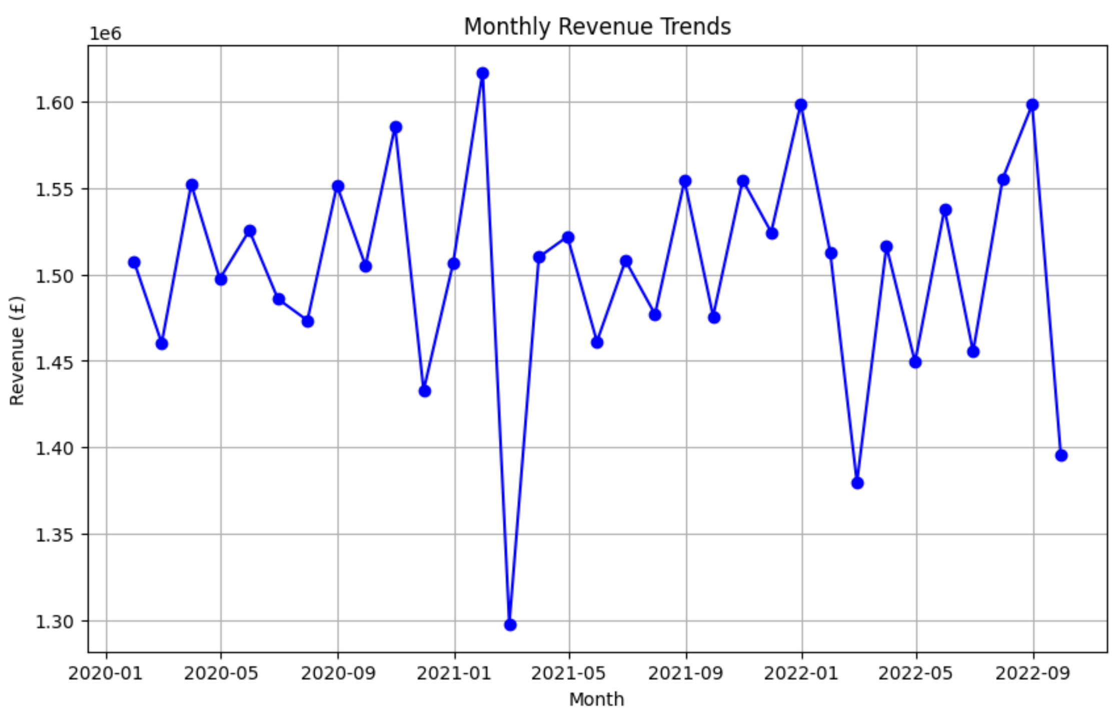
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/city1_financial_data.csv')
df = pd.read_csv('/content/city2_financial_data.csv')

# Convert 'Date' to datetime type for better analysis
df['Date'] = pd.to_datetime(df['Date'], format='%d/%m/%Y') # Specify the␣
↪correct date format

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

# Display the first few rows of the dataset
df.head()
```

[ ]:
```
           City      Revenue     Expenses       Profit  Customers  \
Date
2020-01-01   UK  54331.84411  24728.38845  14606.33550        395
2020-01-02   UK  54254.82683  20673.18542  24215.83602        406
2020-01-03   UK  38879.25182  33020.36529  14380.12070        131
2020-01-04   UK  38285.15911  41716.10057  17385.31555        147
2020-01-05   UK  61701.94743  28822.77867  24393.67087        426

            Transactions  Stock_Price  Market_Sentiment  Loan_Approval_Rate  \
Date
2020-01-01           118    82.460011          0.429902            0.741248
2020-01-02           188    91.905840          0.282992            0.705286
2020-01-03           148   118.247638          0.593407            0.833708
2020-01-04           123    85.339835          0.535897            0.972350
```

```
2020-01-05          152   112.866993         0.804489             0.791580

            Employee_Count  Marketing_Spend
Date
2020-01-01              45     12173.986810
2020-01-02              24      3108.474133
2020-01-03              62     13095.592560
2020-01-04              71      9559.047874
2020-01-05              57     16335.363610
```

1.2.2 Analyze Monthly Revenue Trends

[ ]:
```
# Resample data to get monthly revenue totals
monthly_revenue = df['Revenue'].resample('M').sum()

# Plot the monthly revenue to visualize trends
plt.figure(figsize=(10,6))
plt.plot(monthly_revenue, marker='o', linestyle='-', color='b')
plt.title('Monthly Revenue Trends')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.grid(True)
plt.show()

# Check if there are any seasonal patterns
monthly_revenue
```

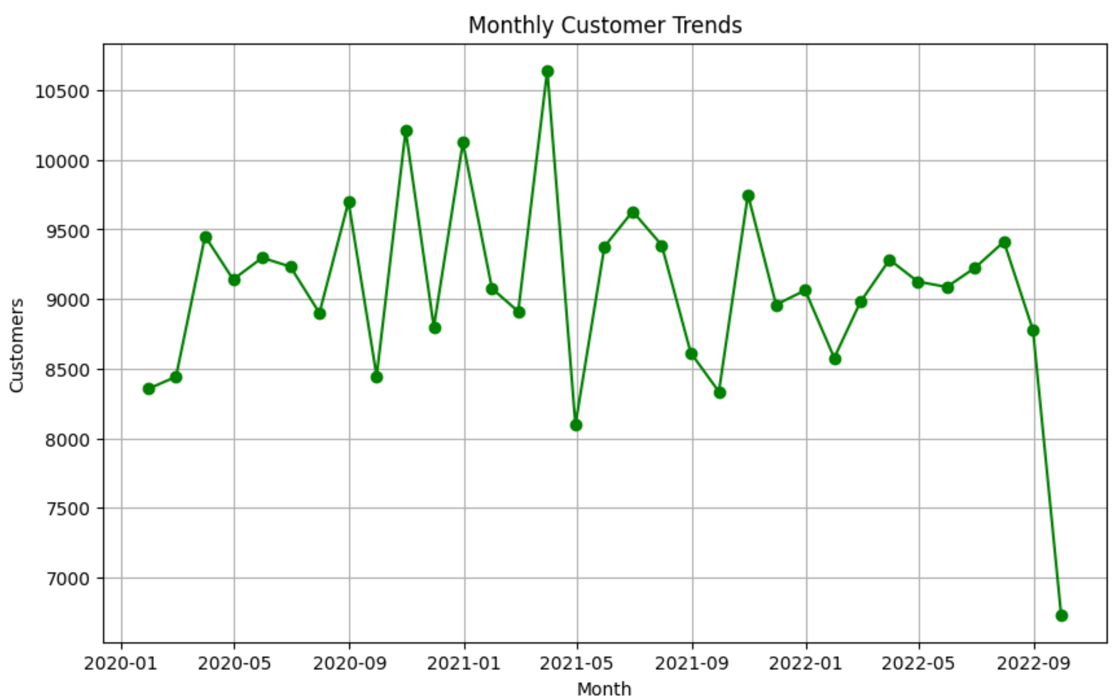


[ ]:
```
Date
2020-01-31    1.507439e+06
2020-02-29    1.460260e+06
2020-03-31    1.552317e+06
2020-04-30    1.497580e+06
2020-05-31    1.525569e+06
2020-06-30    1.485905e+06
2020-07-31    1.473424e+06
2020-08-31    1.551627e+06
2020-09-30    1.505105e+06
2020-10-31    1.585917e+06
2020-11-30    1.432909e+06
2020-12-31    1.506523e+06
2021-01-31    1.617188e+06
2021-02-28    1.297634e+06
2021-03-31    1.510101e+06
2021-04-30    1.522013e+06
2021-05-31    1.461207e+06
2021-06-30    1.508118e+06
2021-07-31    1.477009e+06
2021-08-31    1.554483e+06
2021-09-30    1.475830e+06
2021-10-31    1.554775e+06
2021-11-30    1.524514e+06
2021-12-31    1.598727e+06
2022-01-31    1.512679e+06
2022-02-28    1.379582e+06
2022-03-31    1.516347e+06
2022-04-30    1.449343e+06
2022-05-31    1.537972e+06
2022-06-30    1.455325e+06
2022-07-31    1.555469e+06
2022-08-31    1.598409e+06
2022-09-30    1.395925e+06
Freq: M, Name: Revenue, dtype: float64
```

1.2.3 Identify Customer Demand Trends

[ ]:
```python
# Analyze monthly customer trends
monthly_customers = df['Customers'].resample('M').sum()

# Plot the monthly customer count
plt.figure(figsize=(10,6))
plt.plot(monthly_customers, marker='o', linestyle='-', color='g')
```

```
plt.title('Monthly Customer Trends')
plt.xlabel('Month')
plt.ylabel('Customers')
plt.grid(True)
plt.show()

# You can compare customer trends with revenue to identify correlations
monthly_customers
```



[ ]:
```
Date
2020-01-31     8357
2020-02-29     8442
2020-03-31     9452
2020-04-30     9141
2020-05-31     9297
2020-06-30     9233
2020-07-31     8900
2020-08-31     9699
2020-09-30     8447
2020-10-31    10213
2020-11-30     8800
2020-12-31    10125
2021-01-31     9076
2021-02-28     8911
```

```
2021-03-31    10643
2021-04-30     8098
2021-05-31     9377
2021-06-30     9630
2021-07-31     9388
2021-08-31     8614
2021-09-30     8335
2021-10-31     9750
2021-11-30     8960
2021-12-31     9063
2022-01-31     8578
2022-02-28     8980
2022-03-31     9283
2022-04-30     9126
2022-05-31     9087
2022-06-30     9224
2022-07-31     9412
2022-08-31     8777
2022-09-30     6733
Freq: M, Name: Customers, dtype: int64
```

1.3 Increase Revenue in Low-Performing Months

1.3.1 Identify Low-Performing Months

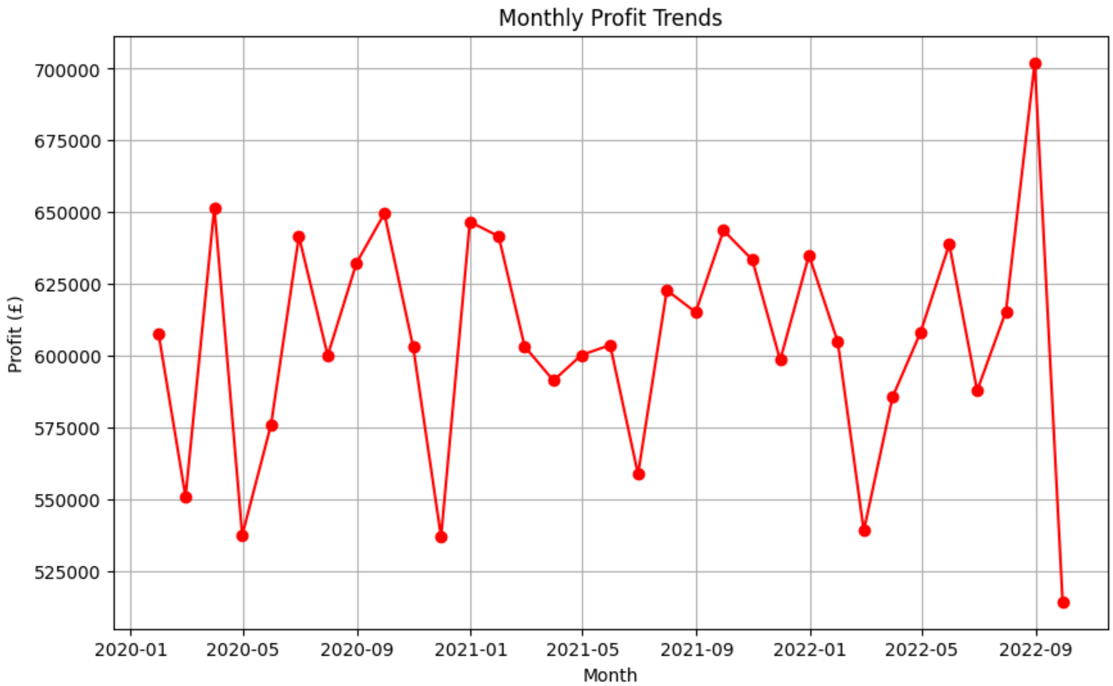
[ ]:
```
# Set a threshold to identify months with lower revenue
low_revenue_threshold = monthly_revenue.mean() * 0.75

# Filter months with revenue below the threshold
low_revenue_months = monthly_revenue[monthly_revenue < low_revenue_threshold]
low_revenue_months
```

[ ]:
```
Series([], Freq: M, Name: Revenue, dtype: float64)
```

1.4 Maintain or Improve Profitability

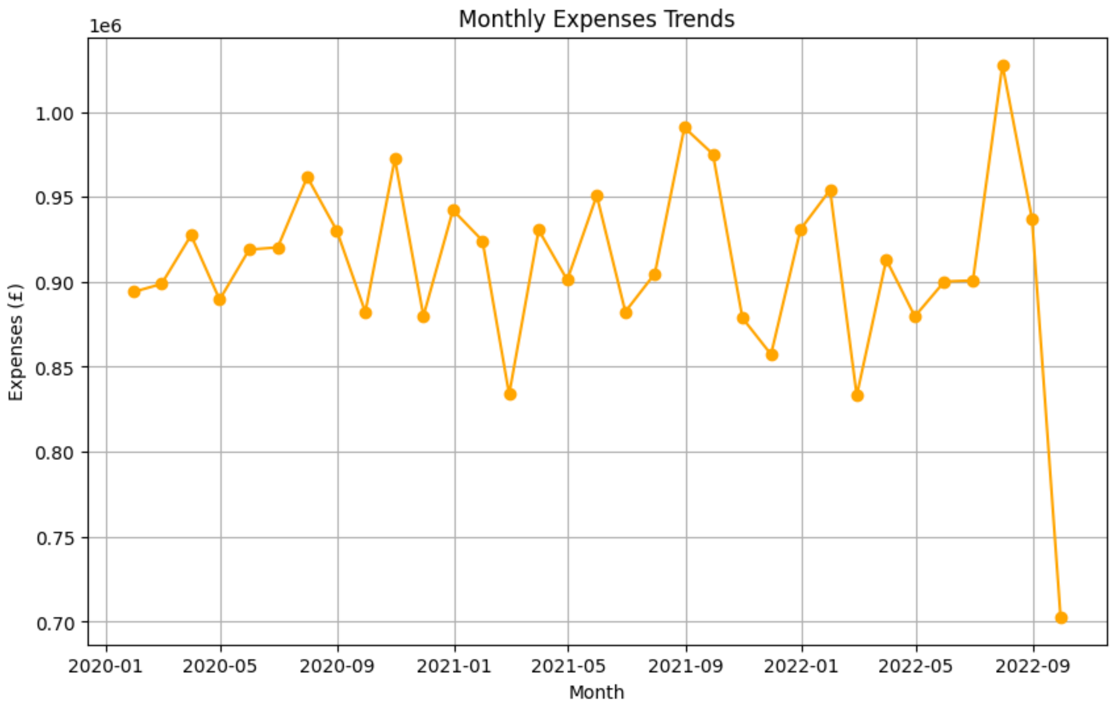
1.4.1 Analyze Profitability

[ ]:
```
# Calculate profit as (Revenue - Expenses) and plot it
monthly_profit = df['Profit'].resample('M').sum()

plt.figure(figsize=(10,6))
plt.plot(monthly_profit, marker='o', linestyle='-', color='r')
plt.title('Monthly Profit Trends')
plt.xlabel('Month')
plt.ylabel('Profit (£)')
plt.grid(True)
plt.show()
```



1.4.2 Cost Optimization Strategy

[ ]:
```
# Analyze expense trends and identify cost-saving opportunities
monthly_expenses = df['Expenses'].resample('M').sum()

plt.figure(figsize=(10,6))
plt.plot(monthly_expenses, marker='o', linestyle='-', color='orange')
plt.title('Monthly Expenses Trends')
plt.xlabel('Month')
plt.ylabel('Expenses (£)')
plt.grid(True)
plt.show()
```

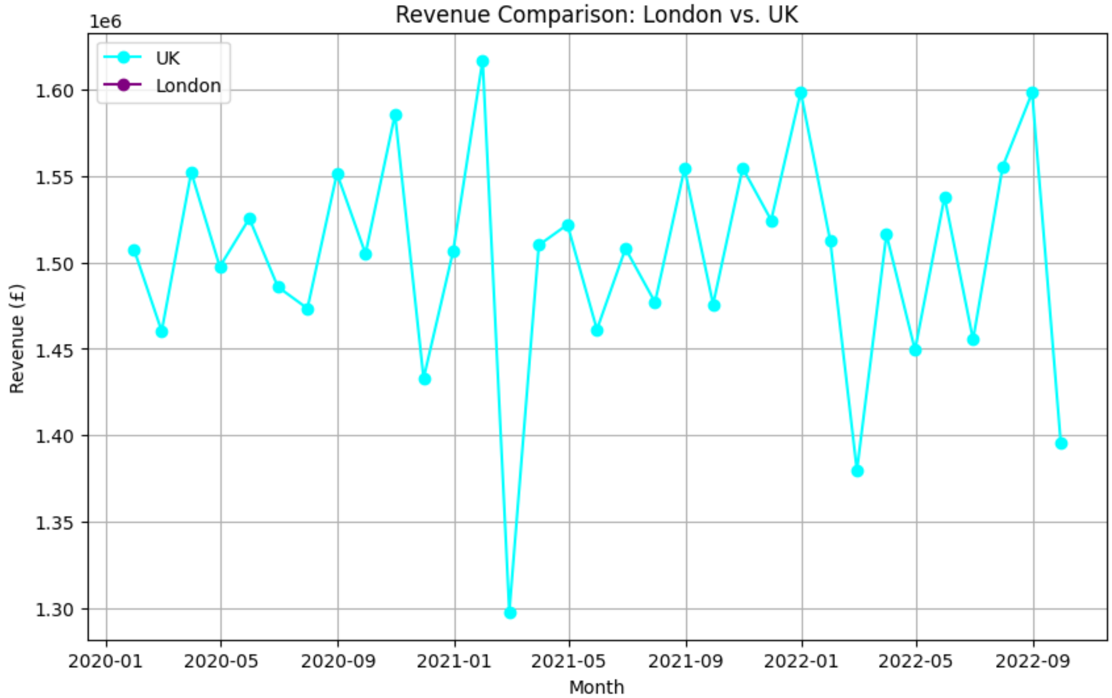


1.5 Leverage London's Economic Role

1.5.1 Capitalize on High Revenue Opportunities

[ ]:
```
# Analyze London-specific data (assuming there's a column for 'City')
london_data = df[df['City'] == 'London']

# Resample data for London to get monthly trends
monthly_revenue_london = london_data['Revenue'].resample('M').sum()

# Plot the revenue trends for London
plt.figure(figsize=(10,6))
plt.plot(monthly_revenue_london, marker='o', linestyle='-', color='purple')
plt.title('Monthly Revenue Trends for London')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.grid(True)
plt.show()

# Compare London's revenue with the rest of the UK
uk_data = df[df['City'] != 'London']
monthly_revenue_uk = uk_data['Revenue'].resample('M').sum()

plt.figure(figsize=(10,6))
```

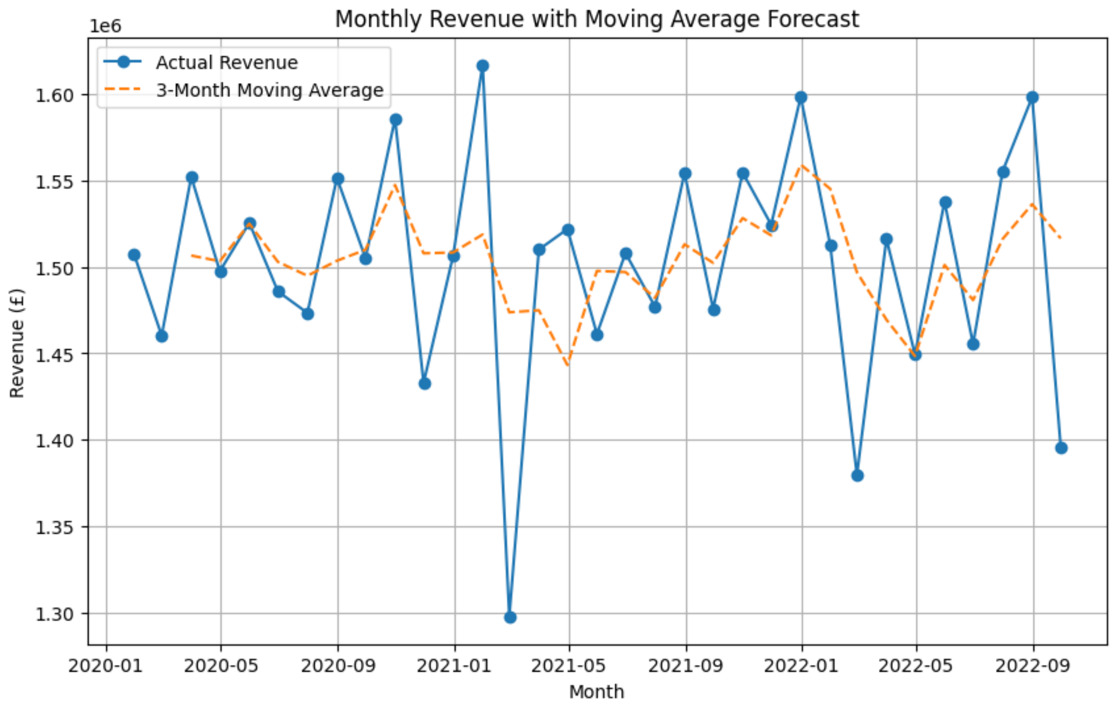
```python
plt.plot(monthly_revenue_uk, marker='o', linestyle='-', color='cyan',␣
 ↪label='UK')
plt.plot(monthly_revenue_london, marker='o', linestyle='-', color='purple',␣
 ↪label='London')
plt.title('Revenue Comparison: London vs. UK')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.legend()
plt.grid(True)
plt.show()
```



1.6 Forecast and Plan for Volatility

1.6.1 Build a Simple Forecasting Model (Moving Average)

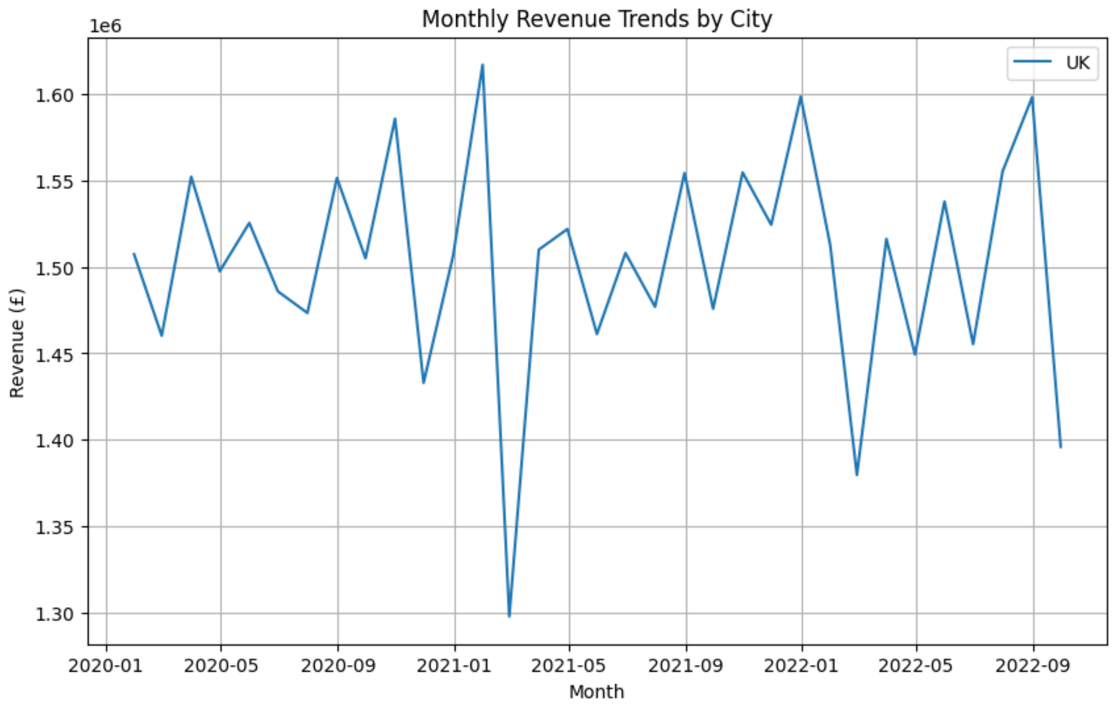
[ ]:
```python
# Calculate a simple moving average to smooth out fluctuations and forecast
rolling_avg_revenue = monthly_revenue.rolling(window=3).mean()

plt.figure(figsize=(10,6))
plt.plot(monthly_revenue, label='Actual Revenue', marker='o')
plt.plot(rolling_avg_revenue, label='3-Month Moving Average', linestyle='--')
plt.title('Monthly Revenue with Moving Average Forecast')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.legend()
plt.grid(True)
plt.show()
```



1.7 Expand Geographical Presence

1.7.1 Explore Revenue Trends in Other Cities

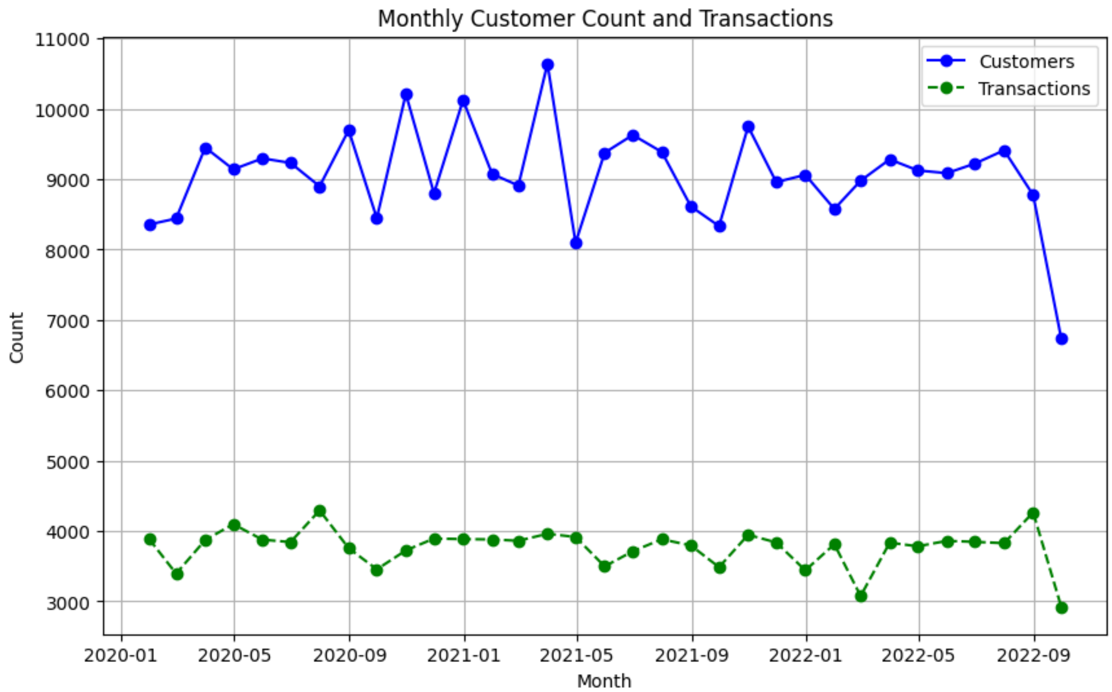
[ ]:
```
# Analyze revenue trends by city (assuming multiple cities)
cities = df['City'].unique()

# Plot revenue trends for each city
plt.figure(figsize=(10,6))

for city in cities:
    city_data = df[df['City'] == city]
    monthly_revenue_city = city_data['Revenue'].resample('M').sum()
    plt.plot(monthly_revenue_city, label=city)

plt.title('Monthly Revenue Trends by City')
plt.xlabel('Month')
plt.ylabel('Revenue (£)')
plt.legend()
plt.grid(True)
plt.show()
```



1.8 Customer Engagement Analysis

1.8.1 Analyze Overall Customer Count and Transactions

[ ]:
```python
# Resample data to get monthly customer count and transaction totals
monthly_customers = df['Customers'].resample('M').sum()
monthly_transactions = df['Transactions'].resample('M').sum()

# Plot monthly customer count trends
plt.figure(figsize=(10,6))
plt.plot(monthly_customers, marker='o', linestyle='-', color='b',␣
 ↪label='Customers')
plt.plot(monthly_transactions, marker='o', linestyle='--', color='g',␣
 ↪label='Transactions')
plt.title('Monthly Customer Count and Transactions')
plt.xlabel('Month')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
plt.show()

# View data
monthly_customers, monthly_transactions
```



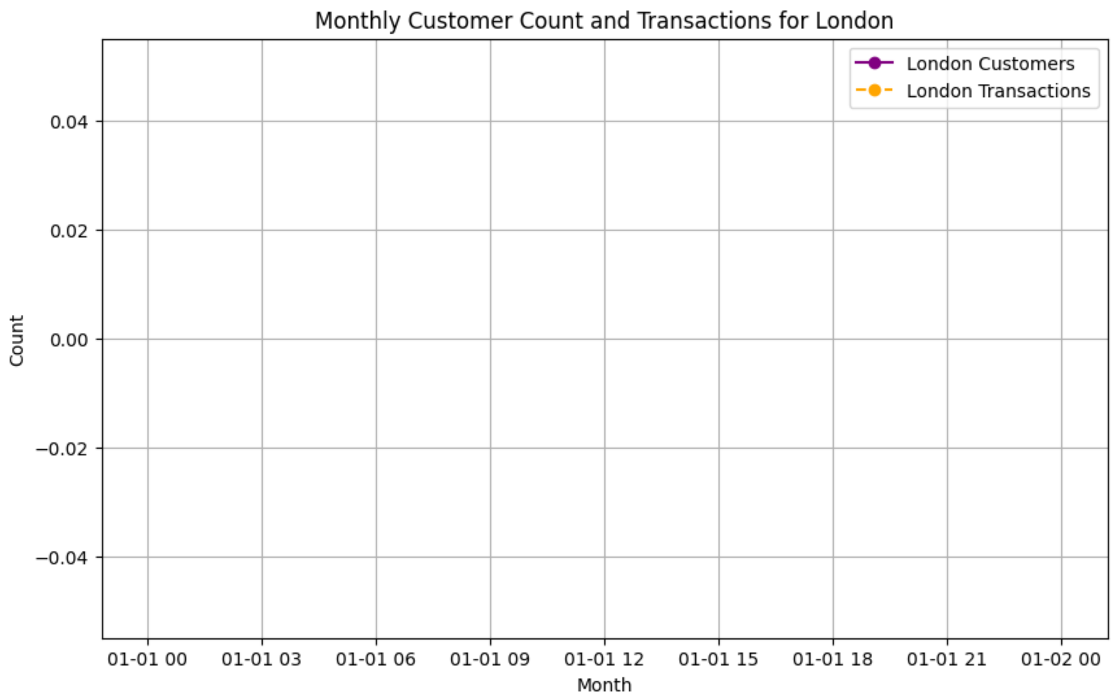

```
[ ]: (Date
      2020-01-31     8357
      2020-02-29     8442
      2020-03-31     9452
      2020-04-30     9141
      2020-05-31     9297
      2020-06-30     9233
      2020-07-31     8900
      2020-08-31     9699
      2020-09-30     8447
      2020-10-31    10213
      2020-11-30     8800
      2020-12-31    10125
      2021-01-31     9076
      2021-02-28     8911
      2021-03-31    10643
      2021-04-30     8098
      2021-05-31     9377
      2021-06-30     9630
      2021-07-31     9388
      2021-08-31     8614
      2021-09-30     8335
      2021-10-31     9750
      2021-11-30     8960
```

```
2021-12-31     9063
2022-01-31     8578
2022-02-28     8980
2022-03-31     9283
2022-04-30     9126
2022-05-31     9087
2022-06-30     9224
2022-07-31     9412
2022-08-31     8777
2022-09-30     6733
Freq: M, Name: Customers, dtype: int64,
Date
2020-01-31    3890
2020-02-29    3395
2020-03-31    3873
2020-04-30    4098
2020-05-31    3878
2020-06-30    3844
2020-07-31    4294
2020-08-31    3757
2020-09-30    3456
2020-10-31    3725
2020-11-30    3894
2020-12-31    3886
2021-01-31    3881
2021-02-28    3862
2021-03-31    3962
2021-04-30    3916
2021-05-31    3498
2021-06-30    3713
2021-07-31    3884
2021-08-31    3795
2021-09-30    3487
2021-10-31    3947
2021-11-30    3838
2021-12-31    3442
2022-01-31    3813
2022-02-28    3077
2022-03-31    3835
2022-04-30    3784
2022-05-31    3861
2022-06-30    3847
2022-07-31    3828
2022-08-31    4265
2022-09-30    2922
Freq: M, Name: Transactions, dtype: int64)
```

1.9 London-Specific Customer Engagement Analysis

1.9.1 Analyze Customer Trends for London

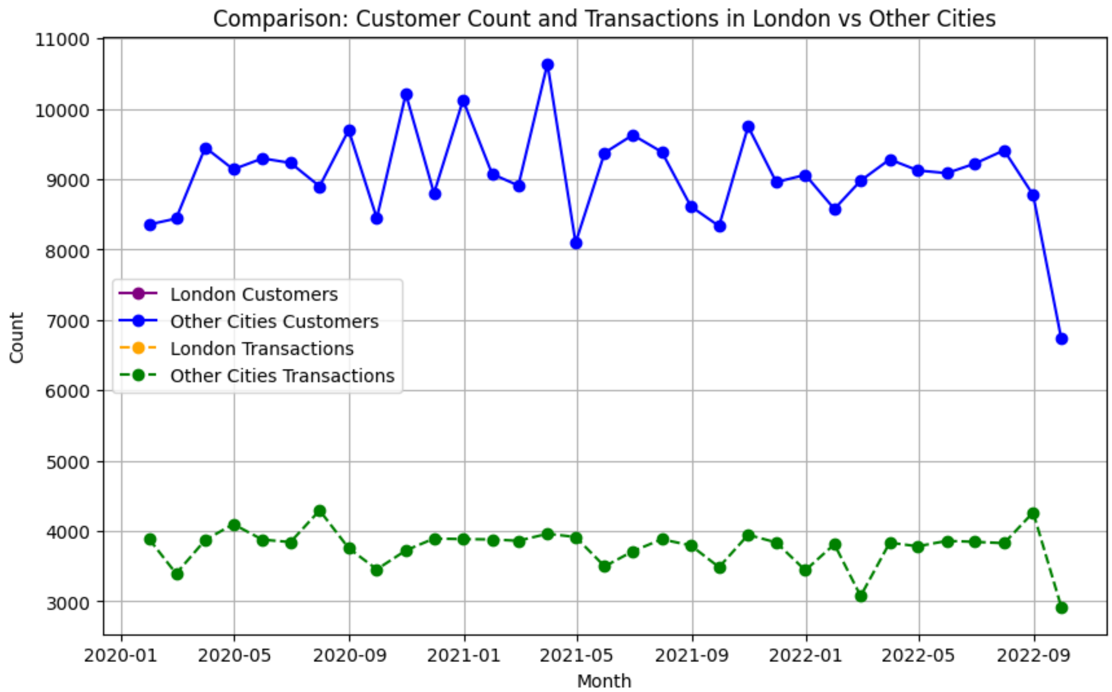
[ ]:
```
# Filter data for London
london_data = df[df['City'] == 'London']

# Resample London customer count and transactions
monthly_customers_london = london_data['Customers'].resample('M').sum()
monthly_transactions_london = london_data['Transactions'].resample('M').sum()

# Plot London customer count and transaction trends
plt.figure(figsize=(10,6))
plt.plot(monthly_customers_london, marker='o', linestyle='-', color='purple',␣
 ↪label='London Customers')
plt.plot(monthly_transactions_london, marker='o', linestyle='--',␣
 ↪color='orange', label='London Transactions')
plt.title('Monthly Customer Count and Transactions for London')
plt.xlabel('Month')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
plt.show()

# View London data
monthly_customers_london, monthly_transactions_london
```



```
[ ]: (Series([], Freq: M, Name: Customers, dtype: int64),
      Series([], Freq: M, Name: Transactions, dtype: int64))
```

1.9.2 Compare London to Other Cities

```
[ ]: # Filter data for all cities except London
other_cities_data = df[df['City'] != 'London']

# Resample data for customer count and transactions for all other cities
monthly_customers_other_cities = other_cities_data['Customers'].resample('M').
 ↪sum()
monthly_transactions_other_cities = other_cities_data['Transactions'].
 ↪resample('M').sum()

# Plot comparison between London and other cities
plt.figure(figsize=(10,6))
plt.plot(monthly_customers_london, marker='o', linestyle='-', color='purple',␣
 ↪label='London Customers')
plt.plot(monthly_customers_other_cities, marker='o', linestyle='-',␣
 ↪color='blue', label='Other Cities Customers')
plt.plot(monthly_transactions_london, marker='o', linestyle='--',␣
 ↪color='orange', label='London Transactions')
plt.plot(monthly_transactions_other_cities, marker='o', linestyle='--',␣
 ↪color='green', label='Other Cities Transactions')
plt.title('Comparison: Customer Count and Transactions in London vs Other␣
 ↪Cities')
plt.xlabel('Month')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
plt.show()
```



1.9.3 Customer Behavior Variability

[ ]:
```
# Calculate standard deviation of customer count and transactions for London
customer_variability_london = monthly_customers_london.std()
transaction_variability_london = monthly_transactions_london.std()

# Calculate standard deviation of customer count and transactions for other␣
↪cities
customer_variability_other_cities = monthly_customers_other_cities.std()
transaction_variability_other_cities = monthly_transactions_other_cities.std()

# Print the results
print(f"London Customer Variability: {customer_variability_london}")
print(f"London Transaction Variability: {transaction_variability_london}")
print(f"Other Cities Customer Variability: {customer_variability_other_cities}")
print(f"Other Cities Transaction Variability:␣
↪{transaction_variability_other_cities}")
```

```
London Customer Variability: nan
London Transaction Variability: nan
Other Cities Customer Variability: 698.195506938487
Other Cities Transaction Variability: 282.63169027638224
```

1.10 Targeting Customer Engagement

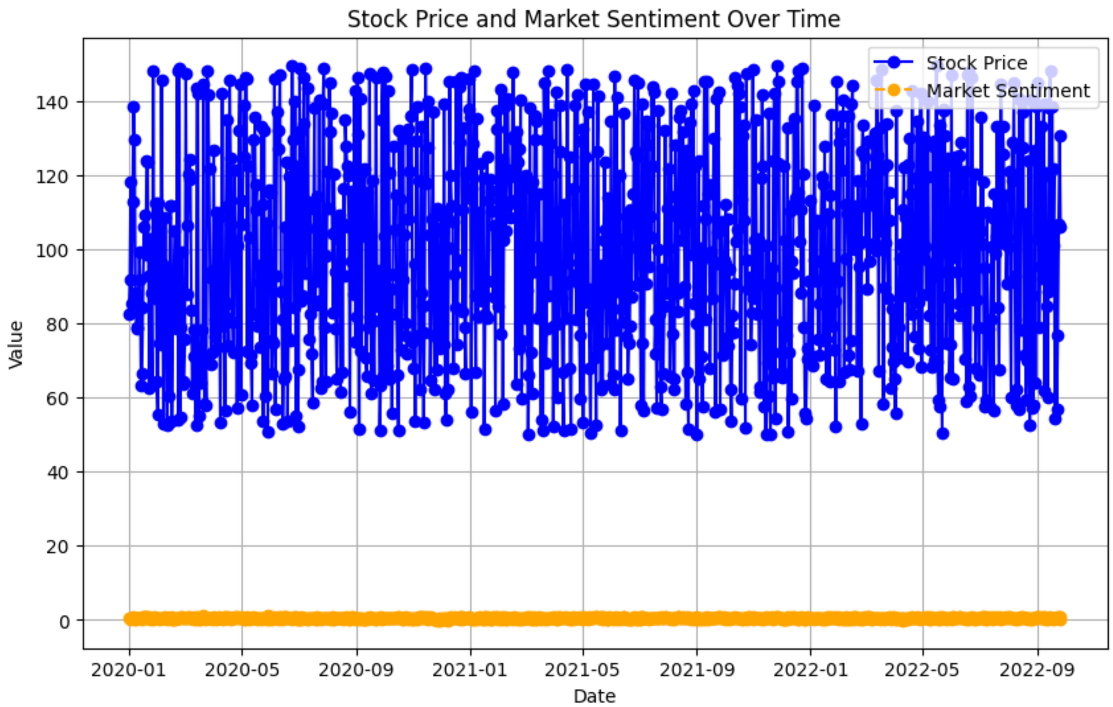
```
[ ]: # Set thresholds to identify months with low customer count and transactions
low_customer_threshold = monthly_customers.mean() * 0.75
low_transaction_threshold = monthly_transactions.mean() * 0.75

# Filter months with low customer count and transactions
low_customer_months = monthly_customers[monthly_customers <
 ↪low_customer_threshold]
low_transaction_months = monthly_transactions[monthly_transactions <
 ↪low_transaction_threshold]

# Print results
print("Low Customer Activity Months:")
print(low_customer_months)

print("\nLow Transaction Activity Months:")
print(low_transaction_months)
```

```
Low Customer Activity Months:
Date
2022-09-30    6733
Freq: M, Name: Customers, dtype: int64

Low Transaction Activity Months:
Series([], Freq: M, Name: Transactions, dtype: int64)
```

1.11 Stock Price and Market Sentiment Analysis

1.11.1 Analyze Stock Price and Market Sentiment

```
[ ]: # Plot stock price and market sentiment trends
plt.figure(figsize=(10,6))
plt.plot(df['Stock_Price'], marker='o', linestyle='-', color='blue',
 ↪label='Stock Price')
plt.plot(df['Market_Sentiment'], marker='o', linestyle='--', color='orange',
 ↪label='Market Sentiment')
plt.title('Stock Price and Market Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()

# Show basic statistics for stock price and market sentiment
df[['Stock_Price', 'Market_Sentiment']].describe()
```



[ ]:

	Stock_Price	Market_Sentiment
count	1000.000000	1000.000000
mean	99.487907	0.502056
std	28.280893	0.201009
min	50.017552	-0.098551
25%	75.589833	0.366848
50%	98.911444	0.501434
75%	123.062436	0.636046
max	149.681498	1.163517

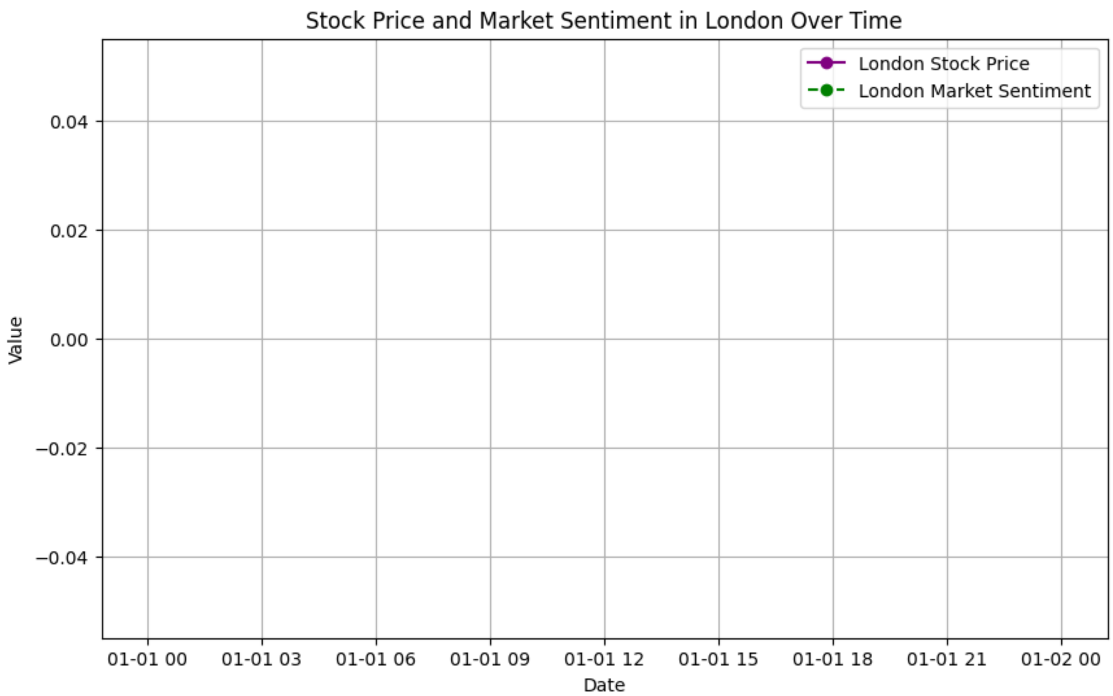
1.12 London-Specific Stock Performance

1.12.1 Analyze Stock Performance for London

[ ]:

```
# Filter data for London
london_data = df[df['City'] == 'London']

# Plot stock price and market sentiment for London
plt.figure(figsize=(10,6))
plt.plot(london_data['Stock_Price'], marker='o', linestyle='-', color='purple',␣
 ↪label='London Stock Price')
plt.plot(london_data['Market_Sentiment'], marker='o', linestyle='--',␣
 ↪color='green', label='London Market Sentiment')
plt.title('Stock Price and Market Sentiment in London Over Time')
```

```
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()

# Show basic statistics for London's stock price and market sentiment
london_data[['Stock_Price', 'Market_Sentiment']].describe()
```



[ ]:

	Stock_Price	Market_Sentiment
count	0.0	0.0
mean	NaN	NaN
std	NaN	NaN
min	NaN	NaN
25%	NaN	NaN
50%	NaN	NaN
75%	NaN	NaN
max	NaN	NaN

1.13 Stock Price Variability and Market Sentiment Correlation

1.13.1 Calculate Stock Price Variability

[ ]:
```
# Calculate stock price variability (standard deviation)
stock_price_variability = df['Stock_Price'].std()
london_stock_price_variability = london_data['Stock_Price'].std()

# Print the results
print(f"Overall Stock Price Variability: {stock_price_variability}")
print(f"London Stock Price Variability: {london_stock_price_variability}")
```

```
Overall Stock Price Variability: 28.28089255888583
London Stock Price Variability: nan
```

1.13.2 Correlation Between Stock Price and Market Sentiment

[ ]:
```
# Calculate correlation between stock price and market sentiment
correlation_overall = df['Stock_Price'].corr(df['Market_Sentiment'])
correlation_london = london_data['Stock_Price'].
 ↪corr(london_data['Market_Sentiment'])

# Print the results
print(f"Overall Correlation Between Stock Price and Market Sentiment:␣
 ↪{correlation_overall}")
print(f"London Correlation Between Stock Price and Market Sentiment:␣
 ↪{correlation_london}")
```

```
Overall Correlation Between Stock Price and Market Sentiment:
0.014569046308377364
London Correlation Between Stock Price and Market Sentiment: nan
```

1.14 Comparison of Stock Price and Market Sentiment Across Cities

1.14.1 Compare London's Stock Performance with Other Cities

[ ]:
```
# Filter data for all cities except London
other_cities_data = df[df['City'] != 'London']

# Calculate stock price variability and correlation for other cities
other_cities_stock_price_variability = other_cities_data['Stock_Price'].std()
correlation_other_cities = other_cities_data['Stock_Price'].
 ↪corr(other_cities_data['Market_Sentiment'])

# Print the comparison
print(f"Other Cities Stock Price Variability:␣
 ↪{other_cities_stock_price_variability}")
print(f"Other Cities Correlation Between Stock Price and Market Sentiment:␣
 ↪{correlation_other_cities}")
```

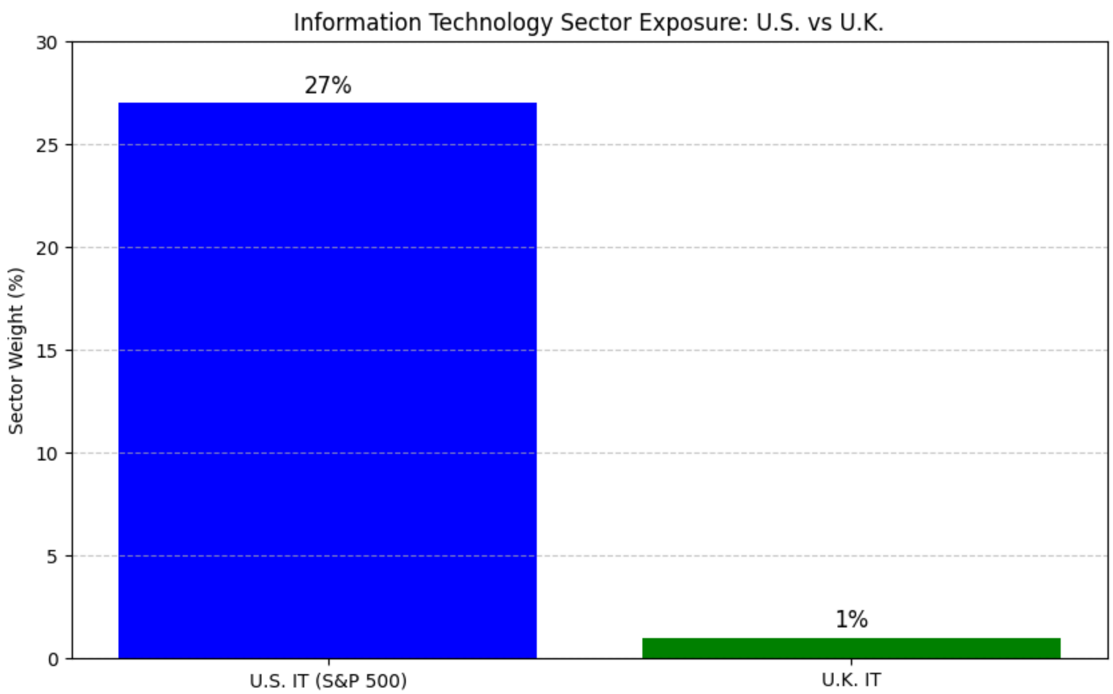
```
Other Cities Stock Price Variability: 28.28089255888583
Other Cities Correlation Between Stock Price and Market Sentiment:
0.014569046308377364
```

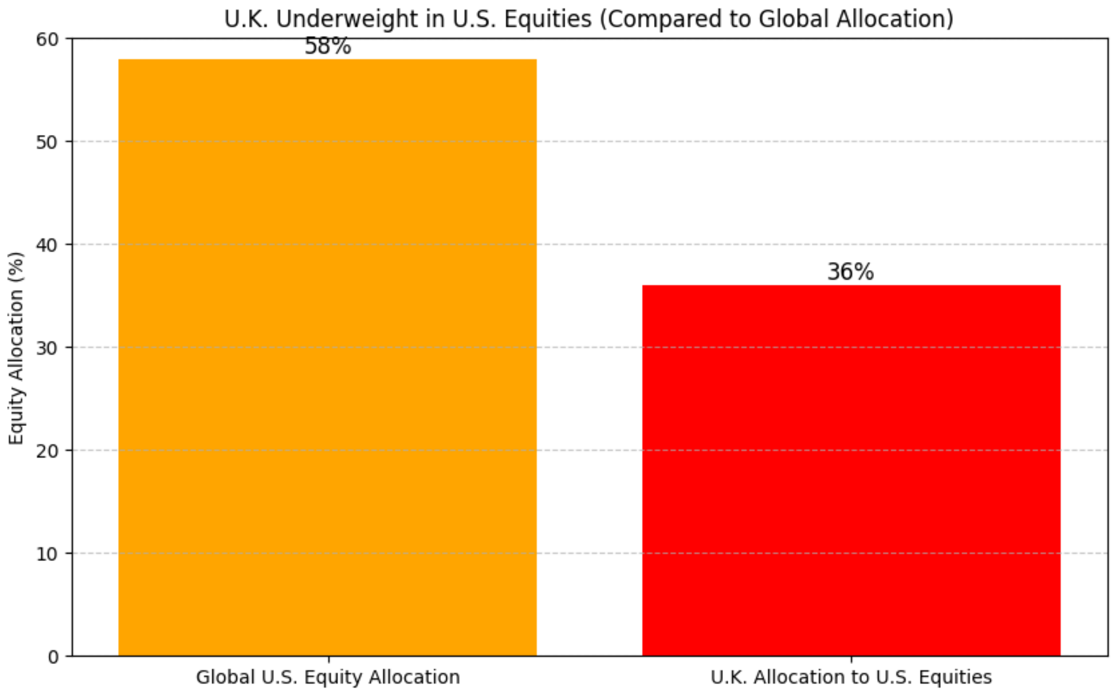
1.15 Understanding Global Financial Movements Impact on London

1.15.1 Assess Impact of Global Events on Stock Price in London

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Data: Global market capitalization breakdown
total_global_market = 100  # Assume global market is 100% for simplicity
sp500_market_cap_share = 50  # S&P 500 represents 50% of global market cap

# U.K. allocation to U.S. equities and sectors (underweight 22%)
uk_allocation_us_equities = 36  # U.K. allocates 36% of equity to U.S. (vs 58%␣
↪global weight)
underweight = 22  # Underweight by 22% relative to global exposure

# Data: Sectoral allocation (example)
# Global Information Technology sector weight and U.K. sector weight
us_it_sector_weight = 27  # IT sector in S&P 500
uk_it_sector_weight = 1  # IT sector in U.K. equities

# Diversification impact visualization
labels = ['U.S. IT (S&P 500)', 'U.K. IT']
sector_weights = [us_it_sector_weight, uk_it_sector_weight]

# Plot sector exposure
plt.figure(figsize=(10, 6))
plt.bar(labels, sector_weights, color=['blue', 'green'])
plt.title('Information Technology Sector Exposure: U.S. vs U.K.')
plt.ylabel('Sector Weight (%)')
plt.ylim(0, 30)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

# Annotate the chart with relevant data
for i, weight in enumerate(sector_weights):
    plt.text(i, weight + 0.5, f'{weight}%', ha='center', fontsize=12)

# Show the plot
plt.show()

# Underweight visualization
labels = ['Global U.S. Equity Allocation', 'U.K. Allocation to U.S. Equities']
allocations = [58, uk_allocation_us_equities]  # Global and U.K. allocation
```

```python
# Plot underweight
plt.figure(figsize=(10, 6))
plt.bar(labels, allocations, color=['orange', 'red'])
plt.title('U.K. Underweight in U.S. Equities (Compared to Global Allocation)')
plt.ylabel('Equity Allocation (%)')
plt.ylim(0, 60)
plt.grid(True, axis='y', linestyle='--', alpha=0.7)

# Annotate the chart with relevant data
for i, alloc in enumerate(allocations):
    plt.text(i, alloc + 0.5, f'{alloc}%', ha='center', fontsize=12)

# Show the plot
plt.show()
```

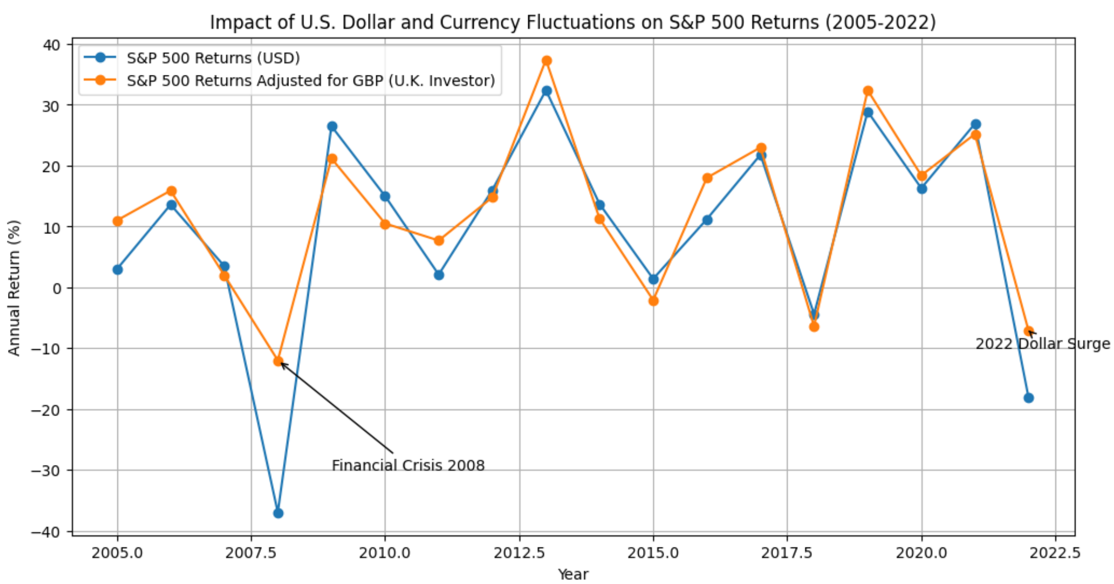




1.16 The impact of currency fluctuations on U.K. investments in the S&P 500

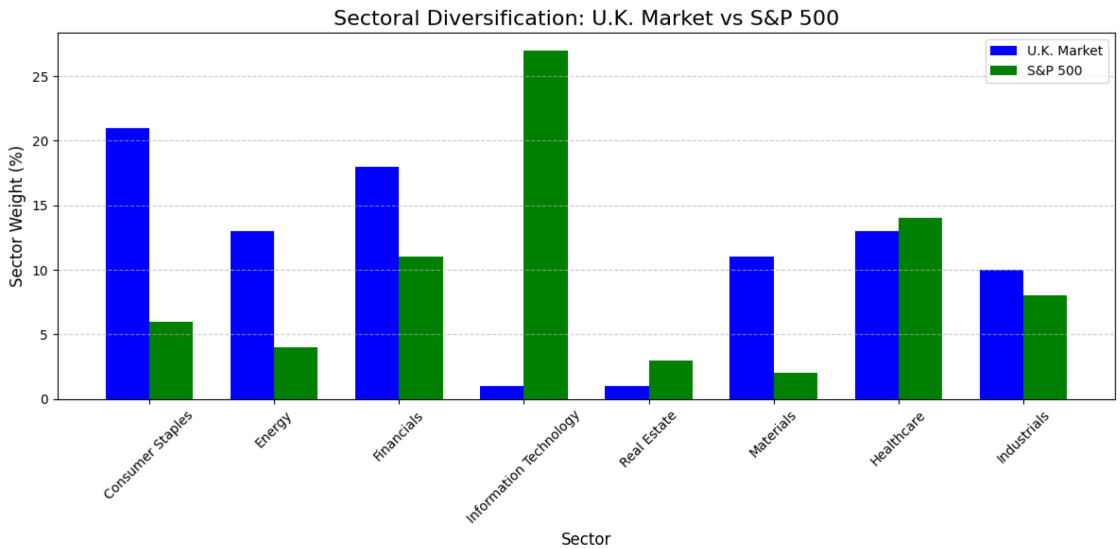
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Data for simulation
years = np.arange(2005, 2023)  # Adjusted to match the number of return data␣
↪points

# Simulated S&P 500 returns (in USD)
sp500_returns_usd = np.array([3.0, 13.6, 3.5, -37.0, 26.5, 15.0, 2.1, 16.0, 32.
↪4, 13.7, 1.4, 11.2, 21.8, -4.4, 28.9, 16.3, 26.9, -18.1])

# Simulated GBP/USD exchange rate fluctuations (% change in USD vs GBP)
# Positive values indicate USD appreciation (bad for U.K. investors), negative␣
↪indicates USD depreciation (good for U.K. investors)
usd_gbp_fluctuations = np.array([8.0, 2.3, -1.5, 25.0, -5.3, -4.5, 5.6, -1.2, 5.
↪0, -2.4, -3.5, 6.8, 1.2, -2.0, 3.5, 2.1, -1.7, 11.0])

# Adjusted S&P 500 returns for U.K. investors (taking into account currency␣
↪fluctuations)
sp500_returns_gbp = sp500_returns_usd + usd_gbp_fluctuations

# Plotting both the USD and GBP returns
```

```python
plt.figure(figsize=(12, 6))
plt.plot(years, sp500_returns_usd, label="S&P 500 Returns (USD)", marker='o')
plt.plot(years, sp500_returns_gbp, label="S&P 500 Returns Adjusted for GBP (U.K.
 ↪ Investor)", marker='o')
plt.title("Impact of U.S. Dollar and Currency Fluctuations on S&P 500 Returns␣
 ↪(2005-2022)")
plt.xlabel("Year")
plt.ylabel("Annual Return (%)")
plt.grid(True)
plt.legend()

# Annotate important financial crises (e.g., 2008, 2022)
plt.annotate("Financial Crisis 2008", xy=(2008, sp500_returns_gbp[3]),␣
 ↪xytext=(2009, -30),
             arrowprops=dict(facecolor='black', arrowstyle="->"), fontsize=10)
plt.annotate("2022 Dollar Surge", xy=(2022, sp500_returns_gbp[-1]),␣
 ↪xytext=(2021, -10),
             arrowprops=dict(facecolor='black', arrowstyle="->"), fontsize=10)

# Show the plot
plt.show()
```



1.17 The sectoral composition of both the U.K. equity market and the S&P 500

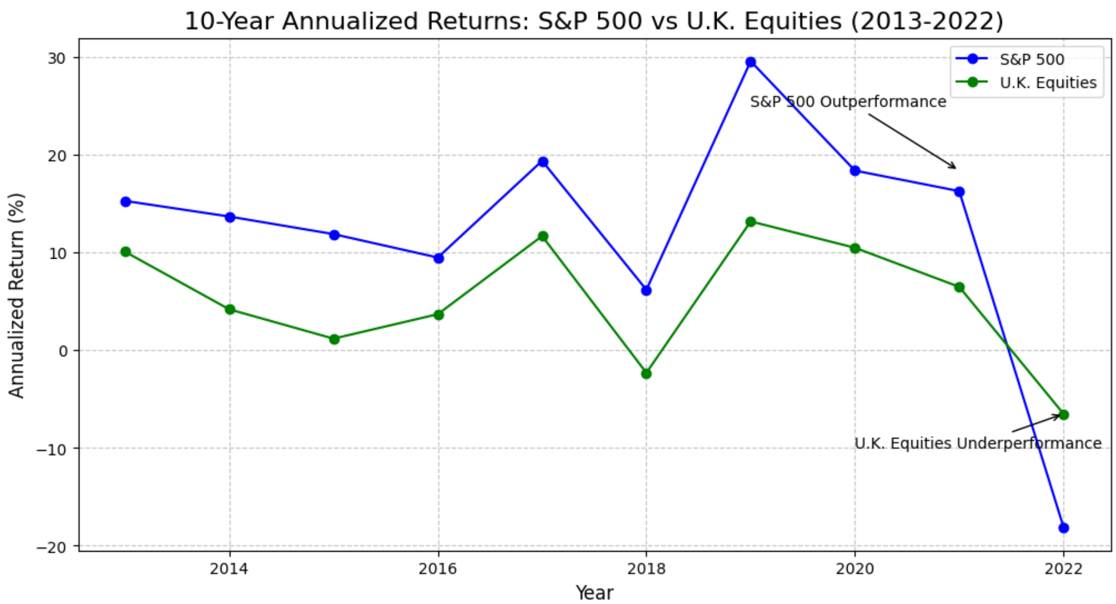
[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Data: Sectoral composition for U.K. and S&P 500
sectors = ['Consumer Staples', 'Energy', 'Financials', 'Information␣
↪Technology', 'Real Estate', 'Materials', 'Healthcare', 'Industrials']

# Sector weights for the U.K. market (as % of total market)
uk_sector_weights = [21, 13, 18, 1, 1, 11, 13, 10]

# Sector weights for the S&P 500 (as % of total market)
sp500_sector_weights = [6, 4, 11, 27, 3, 2, 14, 8]

# Plot settings
bar_width = 0.35
index = np.arange(len(sectors))

# Create figure
plt.figure(figsize=(12, 6))

# Plot the U.K. market sector weights
plt.bar(index, uk_sector_weights, bar_width, label='U.K. Market', color='blue')

# Plot the S&P 500 sector weights (shifted by bar width for separation)
plt.bar(index + bar_width, sp500_sector_weights, bar_width, label='S&P 500',␣
↪color='green')

# Titles and labels
plt.title('Sectoral Diversification: U.K. Market vs S&P 500', fontsize=16)
plt.xlabel('Sector', fontsize=12)
plt.ylabel('Sector Weight (%)', fontsize=12)
plt.xticks(index + bar_width / 2, sectors, rotation=45)
plt.legend()

# Display grid and plot
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



1.18 The S&P 500 and U.K. equities over a 10-year period

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

# Data: Annualized returns for S&P 500 and U.K. equities over a 10-year period
years = np.arange(2013, 2023)  # 10-year period
sp500_annual_returns = [15.3, 13.7, 11.9, 9.5, 19.4, 6.2, 29.6, 18.4, 16.3, -18.
↪1]  # S&P 500 returns in %
uk_equities_annual_returns = [10.1, 4.2, 1.2, 3.7, 11.7, -2.3, 13.2, 10.5, 6.5,␣
↪-6.5]  # U.K. equities returns in %

# Create the plot
plt.figure(figsize=(12, 6))
plt.plot(years, sp500_annual_returns, label="S&P 500", marker='o', color='blue')
plt.plot(years, uk_equities_annual_returns, label="U.K. Equities", marker='o',␣
↪color='green')

# Titles and labels
plt.title("10-Year Annualized Returns: S&P 500 vs U.K. Equities (2013-2022)",␣
↪fontsize=16)
plt.xlabel("Year", fontsize=12)
plt.ylabel("Annualized Return (%)", fontsize=12)
plt.legend()

# Annotate the key years
plt.annotate("S&P 500 Outperformance", xy=(2021, sp500_annual_returns[-3]),␣
↪xytext=(2019, 25),
```

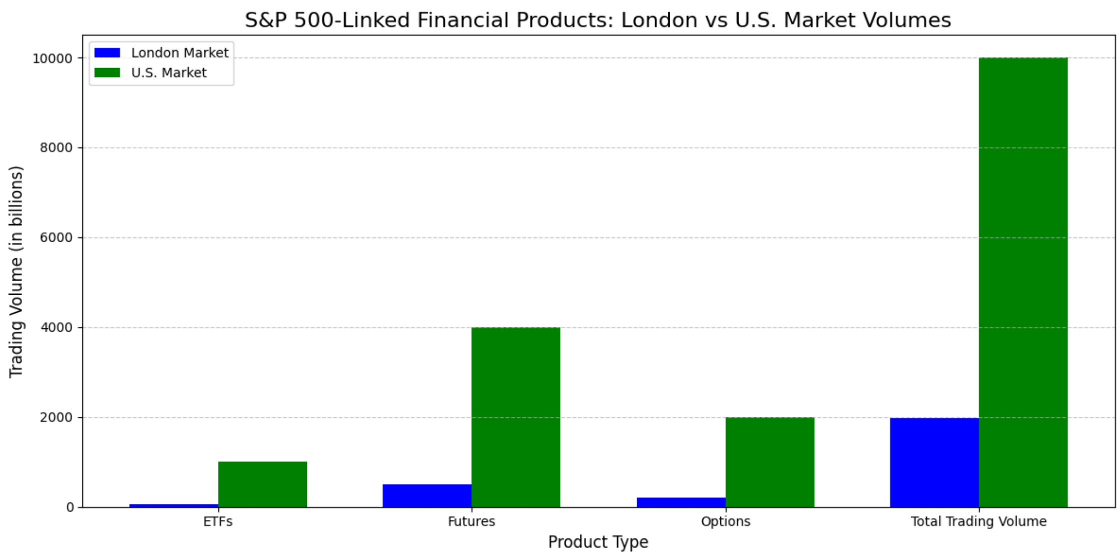
```
            arrowprops=dict(facecolor='black', arrowstyle="->"), fontsize=10)
plt.annotate("U.K. Equities Underperformance", xy=(2022,␣
 ↪uk_equities_annual_returns[-1]), xytext=(2020, -10),
            arrowprops=dict(facecolor='black', arrowstyle="->"), fontsize=10)

# Display grid and plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```



1.18.1 Visualizing London's Interconnectedness with Global Markets via S&P 500 Products

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np

# Data: Trading volumes of S&P 500-related products in London and U.S. markets
categories = ['ETFs', 'Futures', 'Options', 'Total Trading Volume']
london_volumes = [52, 500, 200, 1970]  # London-listed ETFs and related trading␣
 ↪volumes (in billions)
us_volumes = [1000, 4000, 2000, 10000]  # U.S. ETFs and related trading volumes␣
 ↪(in billions)

# Set up bar chart positions
index = np.arange(len(categories))
bar_width = 0.35

# Create figure
```

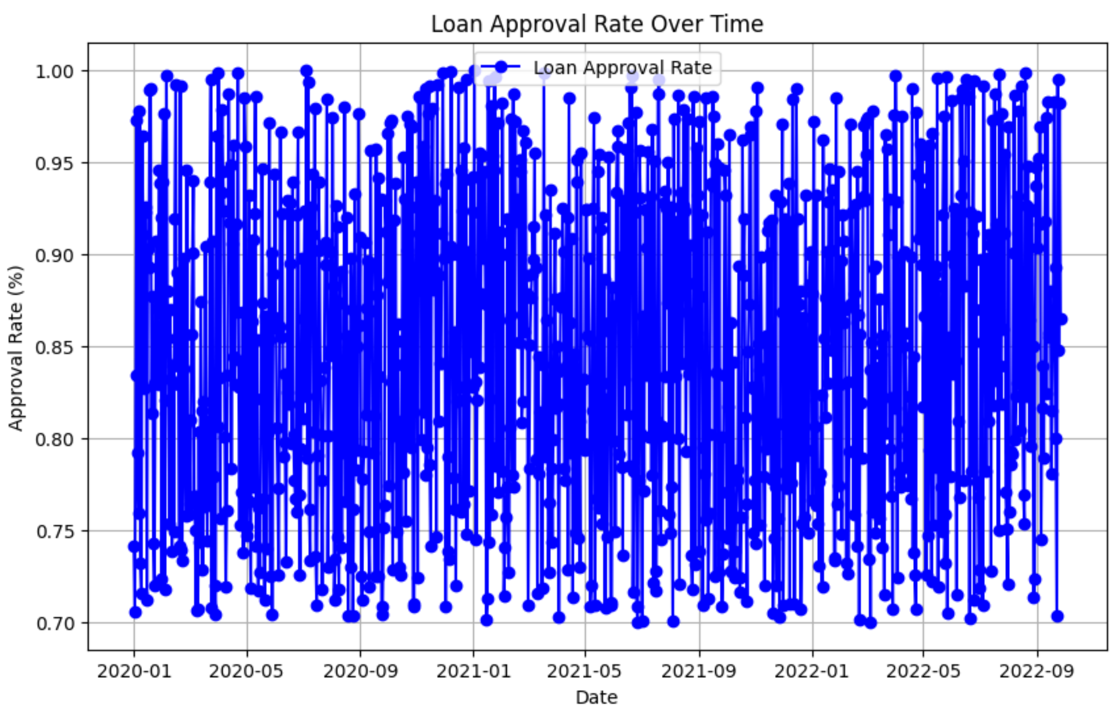
```
plt.figure(figsize=(12, 6))

# Plot London volumes
plt.bar(index, london_volumes, bar_width, label='London Market', color='blue')

# Plot U.S. volumes (shifted by bar width)
plt.bar(index + bar_width, us_volumes, bar_width, label='U.S. Market',␣
↪color='green')

# Titles and labels
plt.title("S&P 500-Linked Financial Products: London vs U.S. Market Volumes",␣
↪fontsize=16)
plt.xlabel("Product Type", fontsize=12)
plt.ylabel("Trading Volume (in billions)", fontsize=12)
plt.xticks(index + bar_width / 2, categories)
plt.legend()

# Display grid and plot
plt.grid(True, axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



1.19 Loan Approval Rate Analysis

1.19.1 Plot Loan Approval Rate Over Time

[ ]:
```
# Plot loan approval rate over time for the entire dataset
plt.figure(figsize=(10,6))
plt.plot(df['Loan_Approval_Rate'], marker='o', linestyle='-', color='blue',␣
 ↪label='Loan Approval Rate')
plt.title('Loan Approval Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Approval Rate (%)')
plt.legend()
plt.grid(True)
plt.show()

# Display basic statistics for loan approval rate
df['Loan_Approval_Rate'].describe()
```

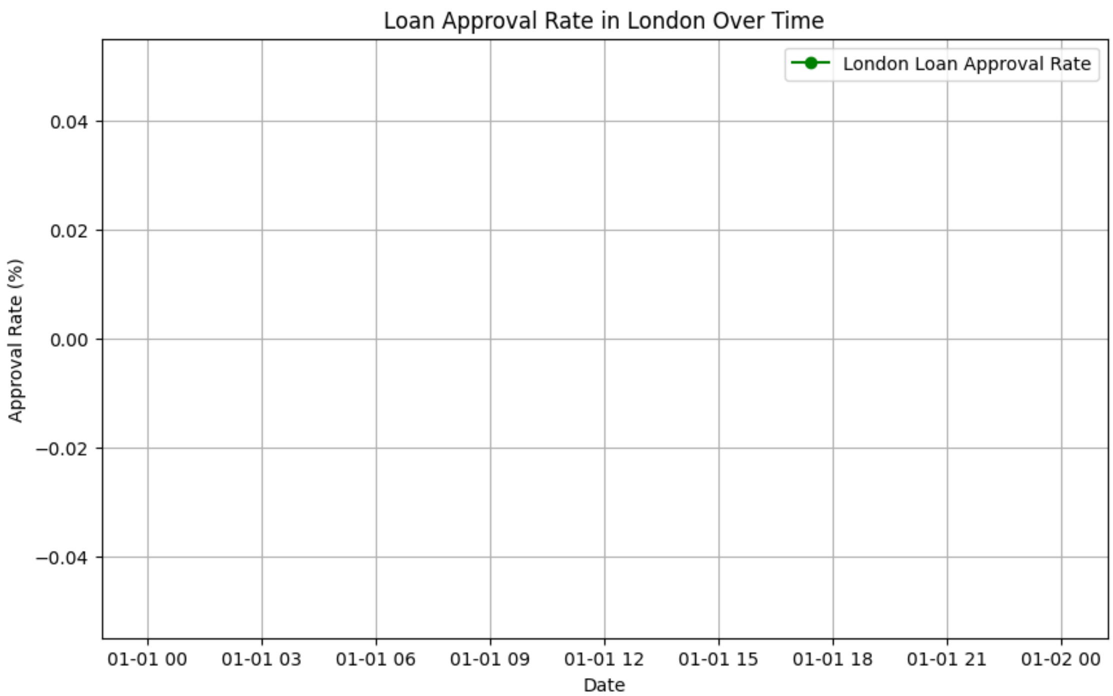


[ ]:
```
count    1000.000000
mean        0.848123
std         0.087595
min         0.700003
25%         0.772894
50%         0.847648
75%         0.923874
```

```
max          0.999908
Name: Loan_Approval_Rate, dtype: float64
```

1.19.2 Analyze London's Loan Approval Rate

[ ]:
```python
# Filter the dataset for London
london_data = df[df['City'] == 'London']

# Plot loan approval rate for London over time
plt.figure(figsize=(10,6))
plt.plot(london_data['Loan_Approval_Rate'], marker='o', linestyle='-',␣
↪color='green', label='London Loan Approval Rate')
plt.title('Loan Approval Rate in London Over Time')
plt.xlabel('Date')
plt.ylabel('Approval Rate (%)')
plt.legend()
plt.grid(True)
plt.show()

# Show basic statistics for London's loan approval rate
london_data['Loan_Approval_Rate'].describe()
```



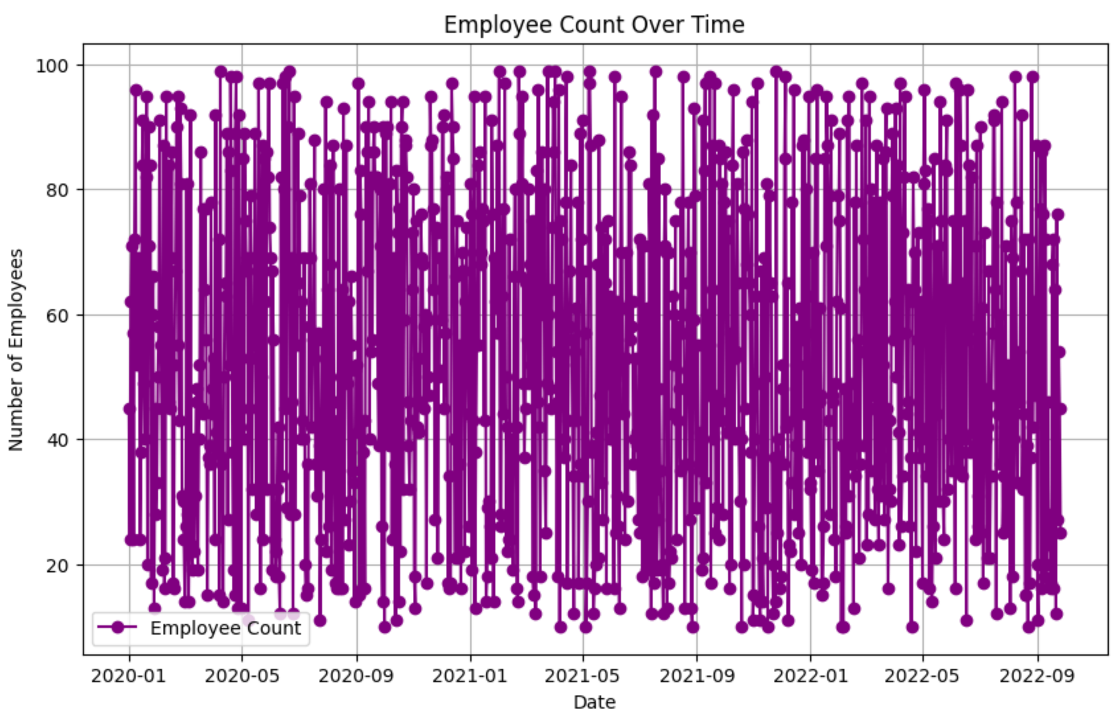
[ ]:
```
count    0.0
mean     NaN
```

```
std        NaN
min        NaN
25%        NaN
50%        NaN
75%        NaN
max        NaN
Name: Loan_Approval_Rate, dtype: float64
```

1.20 Employee Count Analysis

1.20.1 Plot Employee Count Over Time

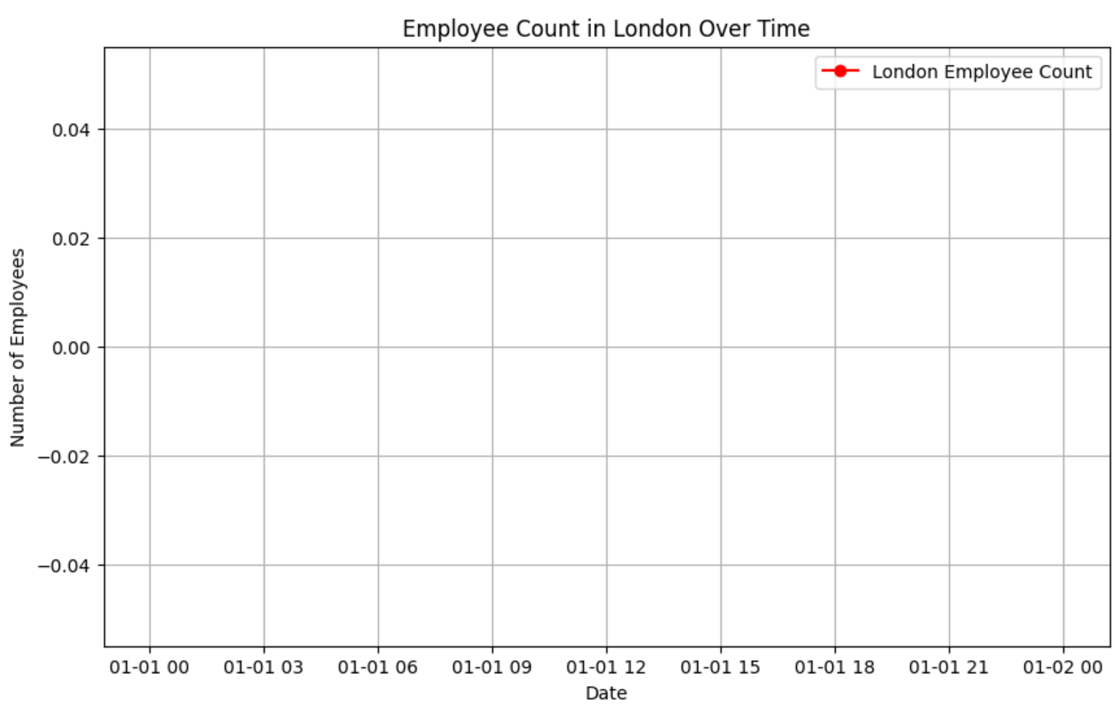
[ ]:
```python
# Plot employee count over time for the entire dataset
plt.figure(figsize=(10,6))
plt.plot(df['Employee_Count'], marker='o', linestyle='-', color='purple',␣
 ↪label='Employee Count')
plt.title('Employee Count Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Employees')
plt.legend()
plt.grid(True)
plt.show()

# Display basic statistics for employee count
df['Employee_Count'].describe()
```



```
[ ]: count    1000.000000
     mean       52.748000
     std        25.810903
     min        10.000000
     25%        31.000000
     50%        51.500000
     75%        75.000000
     max        99.000000
     Name: Employee_Count, dtype: float64
```

1.20.2 Analyze London's Employee Count

```python
[ ]: # Plot employee count for London over time
plt.figure(figsize=(10,6))
plt.plot(london_data['Employee_Count'], marker='o', linestyle='-', color='red',␣
 ↪label='London Employee Count')
plt.title('Employee Count in London Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Employees')
plt.legend()
plt.grid(True)
plt.show()

# Show basic statistics for London's employee count
london_data['Employee_Count'].describe()
```

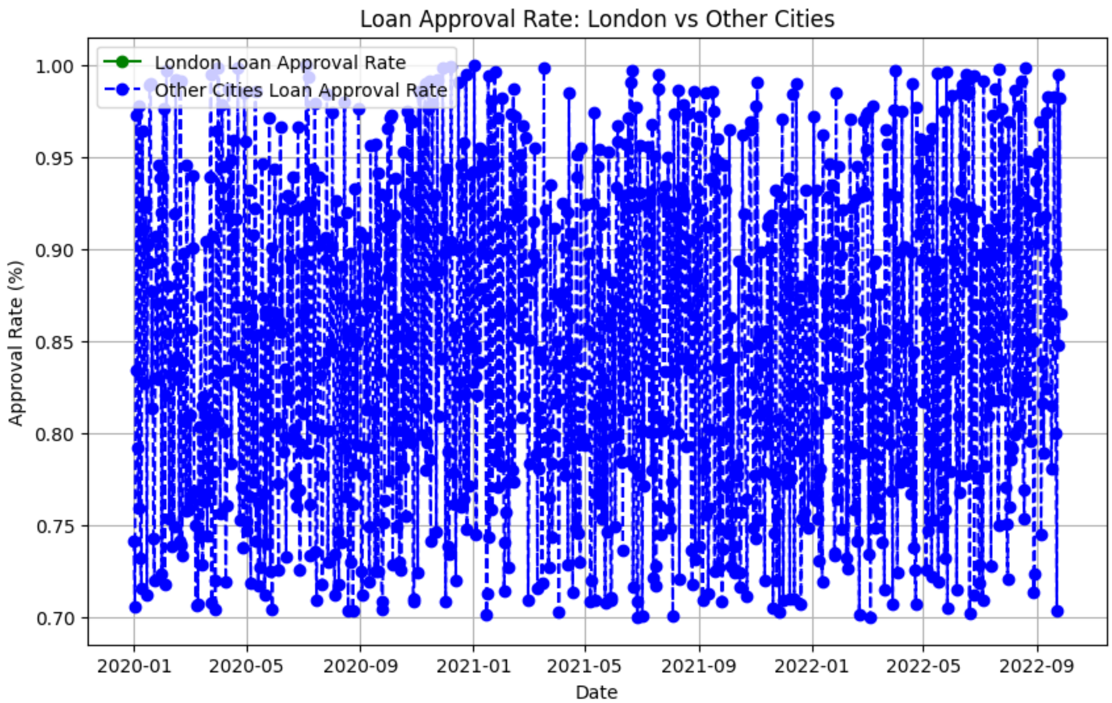


[ ]:
```
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: Employee_Count, dtype: float64
```

1.21 Compare London's Business Operations with Other Cities

1.21.1 Compare Loan Approval Rate and Employee Count for London vs Other Cities

[ ]:
```
# Filter data for all cities except London
other_cities_data = df[df['City'] != 'London']

# Plot comparison for loan approval rates
plt.figure(figsize=(10,6))
plt.plot(london_data['Loan_Approval_Rate'], marker='o', linestyle='-',␣
↪color='green', label='London Loan Approval Rate')
plt.plot(other_cities_data.groupby('Date')['Loan_Approval_Rate'].mean(),␣
↪marker='o', linestyle='--', color='blue', label='Other Cities Loan Approval␣
↪Rate')
plt.title('Loan Approval Rate: London vs Other Cities')
plt.xlabel('Date')
plt.ylabel('Approval Rate (%)')
plt.legend()
plt.grid(True)
plt.show()

# Plot comparison for employee count
plt.figure(figsize=(10,6))
plt.plot(london_data['Employee_Count'], marker='o', linestyle='-', color='red',␣
↪label='London Employee Count')
plt.plot(other_cities_data.groupby('Date')['Employee_Count'].mean(),␣
↪marker='o', linestyle='--', color='purple', label='Other Cities Employee␣
↪Count')
plt.title('Employee Count: London vs Other Cities')
plt.xlabel('Date')
plt.ylabel('Number of Employees')
plt.legend()
plt.grid(True)
plt.show()
```

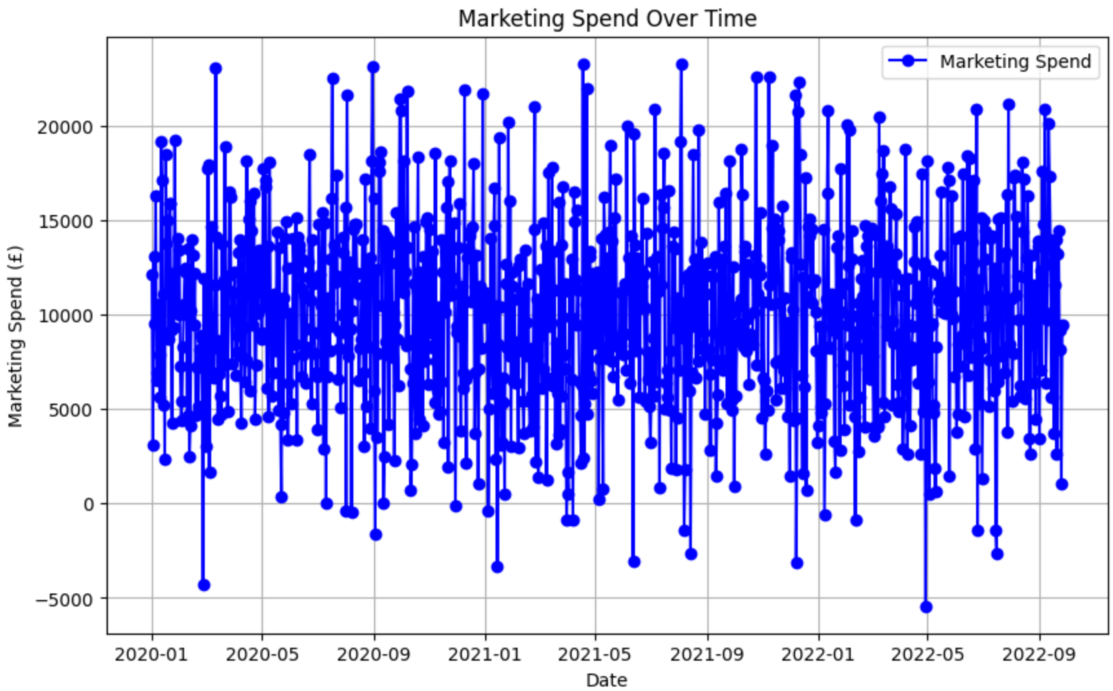

Loan Approval Rate: London vs Other Cities
London Loan Approval Rate
Other Cities Loan Approval Rate
Approval Rate (%)
Date
1.00
0.95
0.90
0.85
0.80
0.75
0.70
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09


Employee Count: London vs Other Cities
London Employee Count
Other Cities Employee Count
Number of Employees
Date
100
80
60
40
20
2020-01
2020-05
2020-09
2021-01
2021-05
2021-09
2022-01
2022-05
2022-09

1.22 Visualize Marketing Spend Trends

1.22.1 Load Data and Explore Marketing Spend

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load and prepare the data (assuming it's already loaded as df)
df = pd.read_csv('/content/city1_financial_data.csv')
df = pd.read_csv('/content/city2_financial_data.csv')

# Convert 'Date' to datetime with dayfirst=True
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)

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

# Display the first few rows to ensure the data is correctly loaded
df.head()

# Visualize marketing spend over time
plt.figure(figsize=(10,6))
plt.plot(df['Marketing_Spend'], marker='o', linestyle='-', color='blue',␣
 ↪label='Marketing Spend')
plt.title('Marketing Spend Over Time')
plt.xlabel('Date')
plt.ylabel('Marketing Spend (£)')
plt.legend()
plt.grid(True)
plt.show()
```



1.22.2 Analyze Marketing Spend for London

[ ]:
```
# Filter the dataset for London
london_data = df[df['City'] == 'London']

# Plot marketing spend for London over time
plt.figure(figsize=(10,6))
plt.plot(london_data['Marketing_Spend'], marker='o', linestyle='-',␣
↪color='green', label='London Marketing Spend')
plt.title('Marketing Spend in London Over Time')
plt.xlabel('Date')
plt.ylabel('Marketing Spend (£)')
plt.legend()
plt.grid(True)
plt.show()

# Show basic statistics for London's marketing spend
london_data['Marketing_Spend'].describe()
```

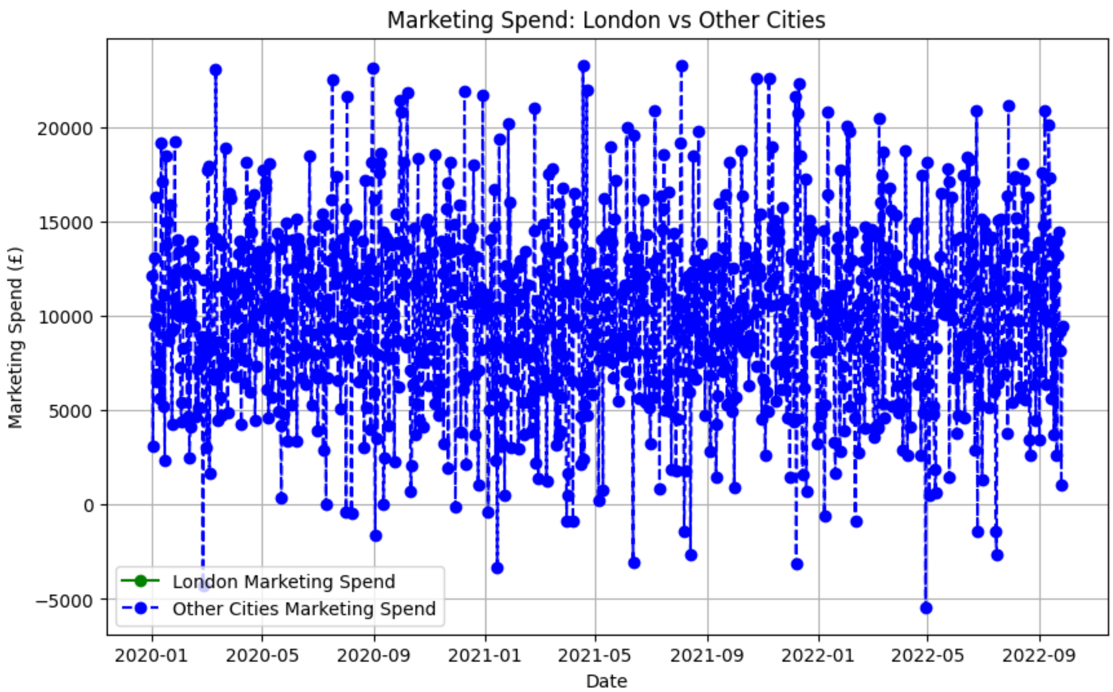
[ ]:
```
count    0.0
mean     NaN
std      NaN
min      NaN
25%      NaN
50%      NaN
75%      NaN
max      NaN
Name: Marketing_Spend, dtype: float64
```

1.23 Compare London's Marketing Spend with Other Cities

1.23.1 Compare Marketing Spend for London vs Other Cities

[ ]:
```
# Filter data for all cities except London
other_cities_data = df[df['City'] != 'London']

# Plot comparison of marketing spend
plt.figure(figsize=(10,6))
plt.plot(london_data['Marketing_Spend'], marker='o', linestyle='-',␣
 ↪color='green', label='London Marketing Spend')
plt.plot(other_cities_data.groupby('Date')['Marketing_Spend'].mean(),␣
 ↪marker='o', linestyle='--', color='blue', label='Other Cities Marketing␣
 ↪Spend')
```

```
plt.title('Marketing Spend: London vs Other Cities')
plt.xlabel('Date')
plt.ylabel('Marketing Spend (£)')
plt.legend()
plt.grid(True)
plt.show()
```



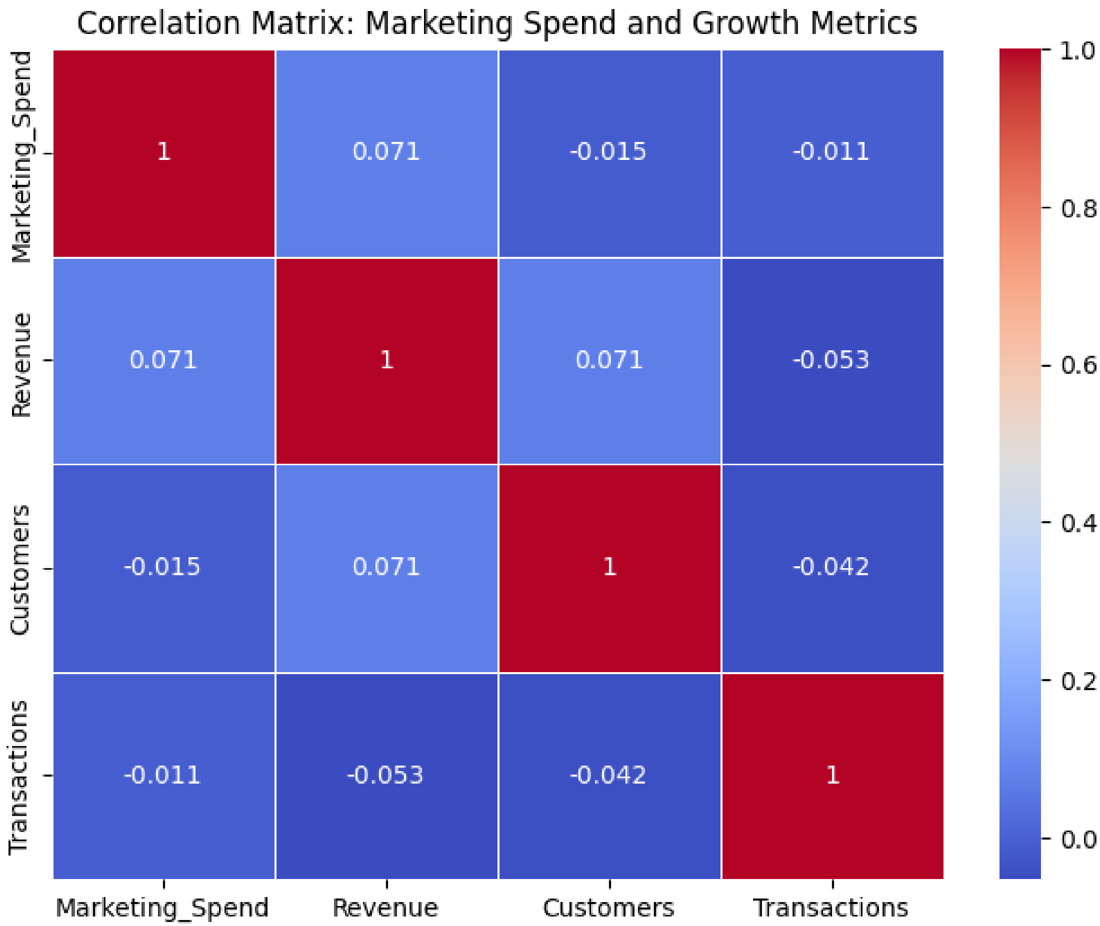
1.24 Analyze the Correlation Between Marketing Spend and Growth

1.24.1 Correlation Analysis Between Marketing Spend and Growth Metrics

[ ]:
```
# Calculate the correlation between marketing spend and other growth metrics
correlation_matrix = df[['Marketing_Spend', 'Revenue', 'Customers',␣
↪'Transactions']].corr()

# Display the correlation matrix
print(correlation_matrix)

# Visualize the correlation heatmap
import seaborn as sns

plt.figure(figsize=(8,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix: Marketing Spend and Growth Metrics')
```

```
plt.show()
```

```
                Marketing_Spend   Revenue  Customers  Transactions
Marketing_Spend        1.000000  0.071262  -0.015203     -0.011336
Revenue                0.071262  1.000000   0.071031     -0.052756
Customers             -0.015203  0.071031   1.000000     -0.042447
Transactions          -0.011336 -0.052756  -0.042447      1.000000
```



1.25 Deeper London-Specific Insights

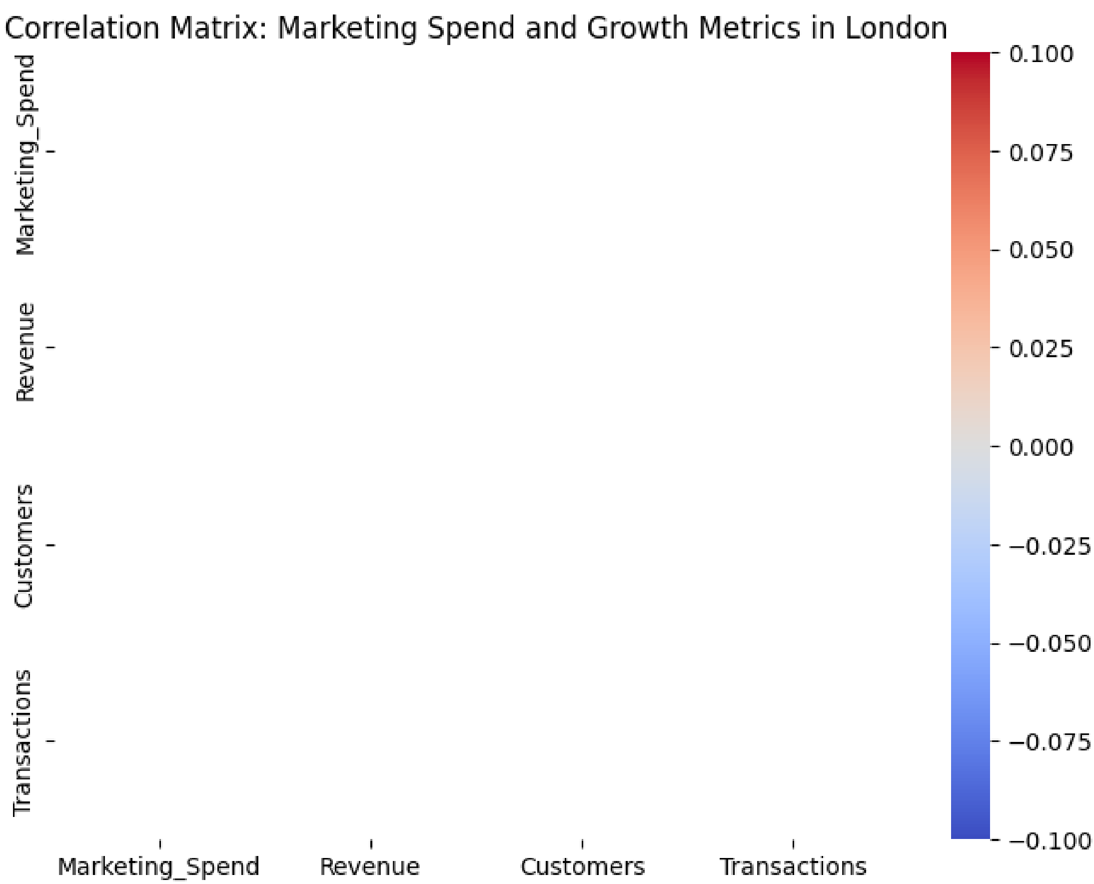
1.25.1 Correlation Analysis for London

[ ]:
```
# Calculate correlation for London-specific data
london_corr_matrix = london_data[['Marketing_Spend', 'Revenue', 'Customers',␣
↪'Transactions']].corr()

# Display the correlation matrix for London
print(london_corr_matrix)
```

```
# Visualize the correlation heatmap for London
plt.figure(figsize=(8,6))
sns.heatmap(london_corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix: Marketing Spend and Growth Metrics in London')
plt.show()
```

```
                Marketing_Spend  Revenue  Customers  Transactions
Marketing_Spend             NaN      NaN        NaN           NaN
Revenue                     NaN      NaN        NaN           NaN
Customers                   NaN      NaN        NaN           NaN
Transactions                NaN      NaN        NaN           NaN
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:202: RuntimeWarning:
All-NaN slice encountered
  vmin = np.nanmin(calc_data)
/usr/local/lib/python3.10/dist-packages/seaborn/matrix.py:207: RuntimeWarning:
All-NaN slice encountered
  vmax = np.nanmax(calc_data)
```

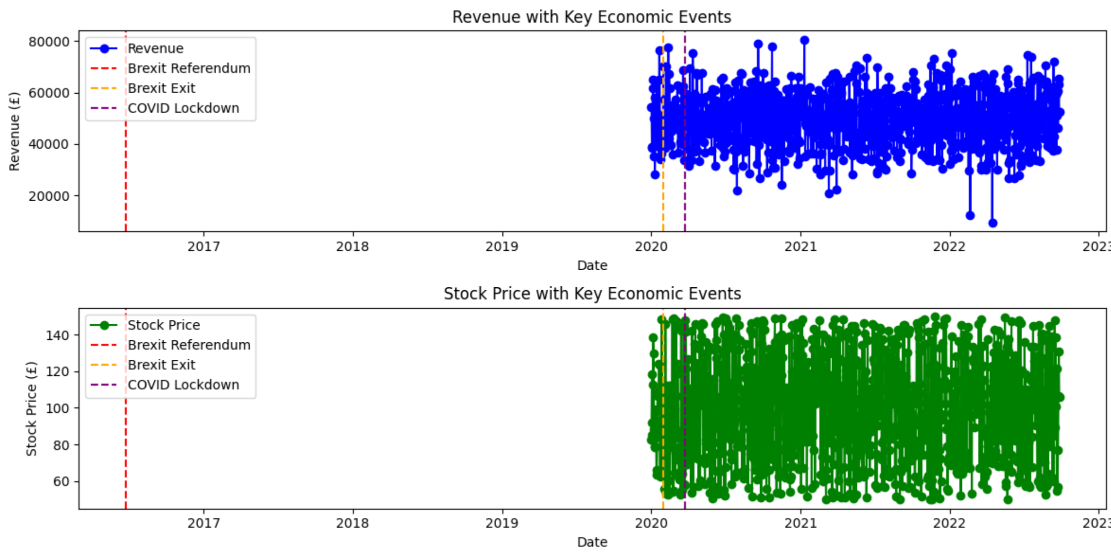


1.26 Economic Impact and Strategic Actions

1.26.1 Analyze the Influence of Broader UK Trends (Brexit, Global Shifts, Policy Changes)

1.26.2 Economic Analysis of Key Events

[ ]:
```python
# Sample event dates (you can customize these)
brexit_dates = ['2016-06-23', '2020-01-31']  # Brexit Referendum and UK␣
↪officially leaving the EU
covid_date = '2020-03-23'  # UK lockdown announcement

# Plot revenue and stock prices with event markers
plt.figure(figsize=(12, 6))

# Plot Revenue
plt.subplot(2, 1, 1)
plt.plot(df.index, df['Revenue'], marker='o', color='blue', label='Revenue')
plt.axvline(pd.to_datetime(brexit_dates[0]), color='red', linestyle='--',␣
↪label='Brexit Referendum')
plt.axvline(pd.to_datetime(brexit_dates[1]), color='orange', linestyle='--',␣
↪label='Brexit Exit')
plt.axvline(pd.to_datetime(covid_date), color='purple', linestyle='--',␣
↪label='COVID Lockdown')
plt.title('Revenue with Key Economic Events')
plt.xlabel('Date')
plt.ylabel('Revenue (£)')
plt.legend()

# Plot Stock Price
plt.subplot(2, 1, 2)
plt.plot(df.index, df['Stock_Price'], marker='o', color='green', label='Stock␣
↪Price')
plt.axvline(pd.to_datetime(brexit_dates[0]), color='red', linestyle='--',␣
↪label='Brexit Referendum')
plt.axvline(pd.to_datetime(brexit_dates[1]), color='orange', linestyle='--',␣
↪label='Brexit Exit')
plt.axvline(pd.to_datetime(covid_date), color='purple', linestyle='--',␣
↪label='COVID Lockdown')
plt.title('Stock Price with Key Economic Events')
plt.xlabel('Date')
plt.ylabel('Stock Price (£)')
plt.legend()

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



1.27 London's Sensitivity to Global Economic Trends

1.27.1 Regional Sensitivity Analysis

[ ]:
```
# Filter data for London
london_data = df[df['City'] == 'London']

# Calculate volatility as the standard deviation of key metrics
london_volatility = london_data[['Revenue', 'Stock_Price', 'Market_Sentiment']].
↪std()

# Compare with other cities (excluding London)
other_cities_data = df[df['City'] != 'London']
other_cities_volatility = other_cities_data.groupby('Date')[['Revenue',␣
↪'Stock_Price', 'Market_Sentiment']].std().mean()

# Print volatility comparison
print(f"London Volatility:\n{london_volatility}\n")
print(f"Other Cities Average Volatility:\n{other_cities_volatility}\n")
```

```
London Volatility:
Revenue             NaN
Stock_Price         NaN
Market_Sentiment    NaN
dtype: float64

Other Cities Average Volatility:
Revenue             NaN
Stock_Price         NaN
```

```
Market_Sentiment   NaN
dtype: float64
```

1.28 COVID-19 Influence and Strategic Actions

1.28.1 Assessing the Pandemic's Impact on Financial Performance

1.28.2 Analyze Revenue, Profit, and Customer Trends Before and After COVID-19 Lockdown

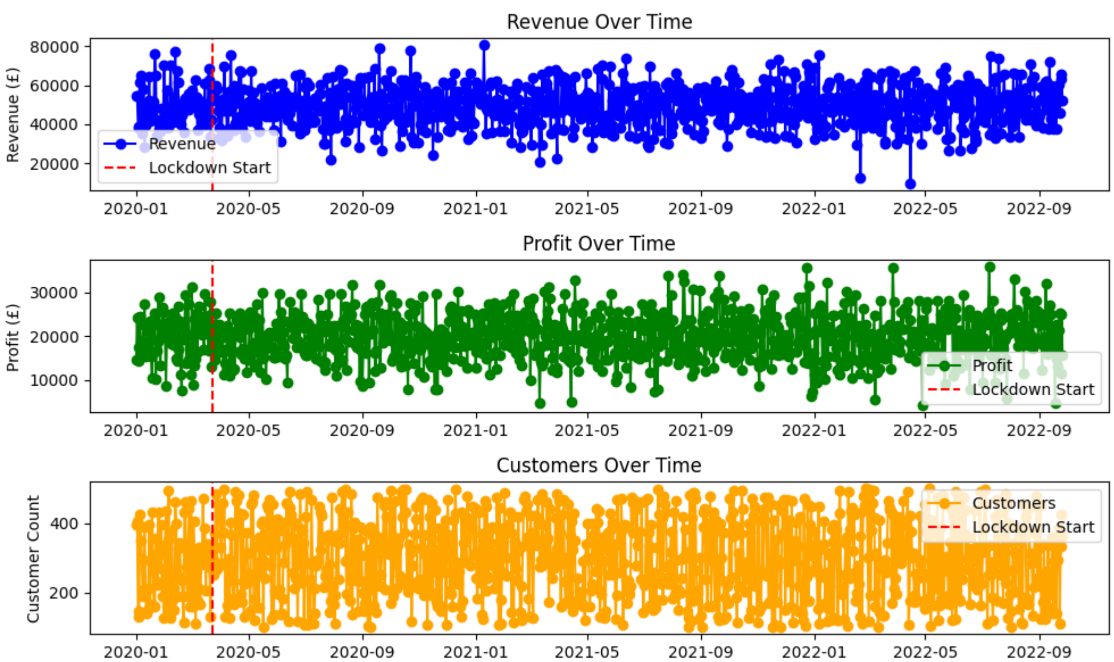
[ ]:
```python
# Define the lockdown date in the UK (March 23, 2020)
lockdown_date = '2020-03-23'

# Create separate dataframes for pre-lockdown and post-lockdown periods
pre_lockdown_data = df[df.index < pd.to_datetime(lockdown_date)]
post_lockdown_data = df[df.index >= pd.to_datetime(lockdown_date)]

# Calculate the average revenue, profit, and customer count for both periods
pre_lockdown_avg = pre_lockdown_data[['Revenue', 'Profit', 'Customers']].mean()
post_lockdown_avg = post_lockdown_data[['Revenue', 'Profit', 'Customers']].
↪mean()

# Print comparison of pre- and post-lockdown performance
print(f"Pre-Lockdown Averages:\n{pre_lockdown_avg}\n")
print(f"Post-Lockdown Averages:\n{post_lockdown_avg}\n")
```

```
Pre-Lockdown Averages:
Revenue      49573.686256
Profit       19849.120220
Customers      279.634146
dtype: float64

Post-Lockdown Averages:
Revenue      49588.434124
Profit       19986.699851
Customers      300.894336
dtype: float64
```

1.28.3 Visualize the Impact on Key Metrics

[ ]:
```python
# Plotting revenue, profit, and customers before and after lockdown
plt.figure(figsize=(10, 6))

# Revenue
plt.subplot(3, 1, 1)
plt.plot(df.index, df['Revenue'], color='blue', marker='o', label='Revenue')
```

```python
plt.axvline(pd.to_datetime(lockdown_date), color='red', linestyle='--',␣
    ↪label='Lockdown Start')
plt.title('Revenue Over Time')
plt.ylabel('Revenue (£)')
plt.legend()

# Profit
plt.subplot(3, 1, 2)
plt.plot(df.index, df['Profit'], color='green', marker='o', label='Profit')
plt.axvline(pd.to_datetime(lockdown_date), color='red', linestyle='--',␣
    ↪label='Lockdown Start')
plt.title('Profit Over Time')
plt.ylabel('Profit (£)')
plt.legend()

# Customers
plt.subplot(3, 1, 3)
plt.plot(df.index, df['Customers'], color='orange', marker='o',␣
    ↪label='Customers')
plt.axvline(pd.to_datetime(lockdown_date), color='red', linestyle='--',␣
    ↪label='Lockdown Start')
plt.title('Customers Over Time')
plt.ylabel('Customer Count')
plt.legend()

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



1.29 London's Unique Position

1.29.1 Regional Impact Analysis

[ ]:
```python
# Filter data for London and non-London cities
london_data = df[df['City'] == 'London']
non_london_data = df[df['City'] != 'London']

# Calculate percentage change in revenue and customer count during the pandemic␣
↪for both London and non-London
london_pct_change = ((post_lockdown_data[post_lockdown_data['City'] ==␣
↪'London'][['Revenue', 'Customers']].mean() -
                      pre_lockdown_data[pre_lockdown_data['City'] ==␣
↪'London'][['Revenue', 'Customers']].mean()) /
                     pre_lockdown_data[pre_lockdown_data['City'] ==␣
↪'London'][['Revenue', 'Customers']].mean()) * 100

non_london_pct_change = ((post_lockdown_data[post_lockdown_data['City'] !=␣
↪'London'][['Revenue', 'Customers']].mean() -
                          pre_lockdown_data[pre_lockdown_data['City'] !=␣
↪'London'][['Revenue', 'Customers']].mean()) /
                         pre_lockdown_data[pre_lockdown_data['City'] !=␣
↪'London'][['Revenue', 'Customers']].mean()) * 100

# Print percentage change comparison
print(f"London Percentage Change (Post-Lockdown):\n{london_pct_change}\n")
print(f"Non-London Percentage Change (Post-Lockdown):
↪\n{non_london_pct_change}\n")
```

```
London Percentage Change (Post-Lockdown):
Revenue      NaN
Customers    NaN
dtype: float64

Non-London Percentage Change (Post-Lockdown):
Revenue      0.029749
Customers    7.602859
dtype: float64
```