In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy import stats
import matplotlib.ticker as ticker
from matplotlib.patches import Rectangle
from scipy.signal import find_peaks
import warnings
warnings.filterwarnings('ignore')

# Set aesthetic parameters for plots
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette('viridis')
sns.set_context("notebook", font_scale=1.2)

# For reproducibility
np.random.seed(42)

# ----- DATA LOADING AND CLEANING -----

# Load the dataset
# Replace this path with your actual file path
file_path = "D:/capstone/datasets/Affinity - State - Daily.xlsx"
try:
    df = pd.read_excel(file_path)
    print(f"Successfully loaded data with {df.shape[0]} rows and {df.shape[1]} colu
except Exception as e:
    print(f"Error loading data: {e}")
    # Create simulated data if file loading fails
    print("Creating simulated data for demonstration...")
    dates = pd.date_range(start='2020-01-13', end='2024-06-16')
    base_spend = np.sin(np.linspace(0, 8*np.pi, len(dates))) * 0.05
    trend = np.concatenate([
        np.linspace(0, 0.02, 60),                          # Slight growth pre-COVID
        np.linspace(0.02, -0.28, 30),                      # COVID crash
        np.linspace(-0.28, -0.05, 90),                     # Initial recovery
        np.linspace(-0.05, 0.10, 365),                     # 2021 recovery
        np.linspace(0.10, 0.22, 365),                      # 2022 inflation
        np.linspace(0.22, 0.18, 365),                      # 2023 stabilization
        np.linspace(0.18, 0.15, len(dates)-60-30-90-365-365-365)  # 2024 trend
    ])
    noise = np.random.normal(0, 0.02, len(dates))
    spend_all = base_spend + trend + noise
    df = pd.DataFrame({
        'year': dates.year,
        'month': dates.month,
        'day': dates.day,
        'spend_all': spend_all
    })
    print(f"Created simulated data with {df.shape[0]} rows and {df.shape[1]} column
# Data cleaning steps
print("Performing data cleaning...")

# 1. Check for missing values in key columns
if 'spend_all' in df.columns:
    print(f"Missing values in spend_all: {df['spend_all'].isna().sum()}")
else:
    if 'spend' in df.columns:
        df.rename(columns={'spend': 'spend_all'}, inplace=True)
        print("Renamed 'spend' column to 'spend_all'")
    else:
        print("Warning: 'spend_all' column not found, using first available spendin
        spend_cols = [col for col in df.columns if 'spend' in col.lower()]
        if spend_cols:
            df['spend_all'] = df[spend_cols[0]]
            print(f"Using {spend_cols[0]} as 'spend_all'")
        else:
            print("No spending columns found!")

# 2. Convert string values of "." to NaN
if df['spend_all'].dtype == object:
    df['spend_all'] = df['spend_all'].replace('.', np.nan).astype(float)
    print("Converted string '.' values to NaN")

# 3. Create date column if it doesn't exist
if 'date' not in df.columns:
    if all(col in df.columns for col in ['year', 'month', 'day']):
        df['date'] = pd.to_datetime(df[['year', 'month', 'day']])
        print("Created date column from year, month, and day columns")
    else:
        print("Warning: Could not create date column due to missing year/month/day

# 4. Handle missing values
missing_before = df['spend_all'].isna().sum()
if missing_before > 0:
    # Interpolate missing values
    df['spend_all'] = df['spend_all'].interpolate(method='linear')
    missing_after = df['spend_all'].isna().sum()
    print(f"Handled missing values: {missing_before} before, {missing_after} after

# 5. Filter out any remaining NaN values
df_clean = df.dropna(subset=['spend_all', 'date'])
print(f"Filtered dataset: {df_clean.shape[0]} rows (removed {df.shape[0] - df_clean

# 6. Check for outliers (Z-score > 3)
z_scores = np.abs(stats.zscore(df_clean['spend_all']))
outliers = np.sum(z_scores > 3)
print(f"Identified {outliers} potential outliers (Z-score > 3)")

# 7. Sort data by date
df_clean = df_clean.sort_values('date')
print("Sorted data by date")

# 8. Create the final time series dataframe
df_timeseries = df_clean[['date', 'spend_all']].copy()
print(f"Created time series dataset with {len(df_timeseries)} rows")
```

```
# Display date range
date_range = f"{df_timeseries['date'].min().strftime('%Y-%m-%d')} to {df_timeseries
print(f"Date range: {date_range}")

# Function for annotating economic phases
def add_economic_phases(ax, ymin=None, ymax=None, fontsize=10):
    """Add economic phase annotations to a time series plot"""
    if ymin is None or ymax is None:
        ymin, ymax = ax.get_ylim()

    # Define the economic phases with dates, labels, and colors
    phases = [
        {"start": "2020-01-15", "end": "2020-04-15", "label": "COVID Crash", "color
        {"start": "2020-04-16", "end": "2020-12-31", "label": "Pandemic Response",
        {"start": "2021-01-01", "end": "2021-12-31", "label": "Early Recovery", "co
        {"start": "2022-01-01", "end": "2022-12-31", "label": "Inflation Begins", "
        {"start": "2023-01-01", "end": "2024-06-16", "label": "Stabilization", "col
    ]

    # Convert dates to matplotlib format
    for phase in phases:
        start = pd.to_datetime(phase["start"])
        end = pd.to_datetime(phase["end"])

        # Add colored rectangle for each phase
        rect = Rectangle(
            (mdates.date2num(start), ymin),
            mdates.date2num(end) - mdates.date2num(start),
            ymax - ymin,
            facecolor=phase["color"],
            alpha=0.2
        )
        ax.add_patch(rect)

        # Add text label in the middle of the phase
        middle_date = start + (end - start) / 2
        ax.text(
            mdates.date2num(middle_date),
            ymin + (ymax - ymin) * 0.05,
            phase["label"],
            ha='center',
            fontsize=fontsize,
            fontweight='bold'
        )

# ----- DATA ANALYSIS -----
print("\nBeginning data analysis...")

# 1. Plot the full time series
plt.figure(figsize=(16, 10))
ax = plt.subplot(111)

# Plot daily spending
plt.plot(df_timeseries['date'], df_timeseries['spend_all'],
         alpha=0.4, label='Daily Spending Index', color='gray')
```

```
# 2. Add rolling averages
df_timeseries['30d_ma'] = df_timeseries['spend_all'].rolling(window=30).mean()
df_timeseries['90d_ma'] = df_timeseries['spend_all'].rolling(window=90).mean()

plt.plot(df_timeseries['date'], df_timeseries['30d_ma'],
         linewidth=2, label='30-Day Moving Average', color='blue')
plt.plot(df_timeseries['date'], df_timeseries['90d_ma'],
         linewidth=3, label='90-Day Moving Average', color='red')

# Format the x-axis to show years
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
ax.xaxis.set_minor_locator(mdates.MonthLocator())

# Add economic phases
add_economic_phases(ax)

# Styling the plot
plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Spending Index', fontsize=14)
plt.title('U.S. Consumer Spending Trends (2020-2024)', fontsize=16)
plt.legend(loc='best', fontsize=12)

# Adjust y-axis to give some padding
y_min, y_max = plt.ylim()
plt.ylim(y_min - 0.05, y_max + 0.05)

plt.tight_layout()
plt.savefig('full_spending_trend.png', dpi=300)
plt.show()

# 3. Seasonal Decomposition
print("Performing seasonal decomposition...")
# First, ensure we have a consistent frequency for decomposition
# Let's resample to daily frequency to ensure no gaps
df_resampled = df_timeseries.set_index('date')['spend_all'].resample('D').mean().fi

# Perform seasonal decomposition
decomposition = seasonal_decompose(df_resampled, model='additive', period=365)

# Plot the decomposed components
fig, axes = plt.subplots(4, 1, figsize=(16, 16), sharex=True)

# Original Data
axes[0].plot(decomposition.observed.index, decomposition.observed, color='blue')
axes[0].set_title('Observed Data', fontsize=14)
axes[0].tick_params(axis='x', rotation=45)

# Trend Component
axes[1].plot(decomposition.trend.index, decomposition.trend, color='green')
axes[1].set_title('Trend Component', fontsize=14)

# Seasonal Component
axes[2].plot(decomposition.seasonal.index, decomposition.seasonal, color='orange')
```

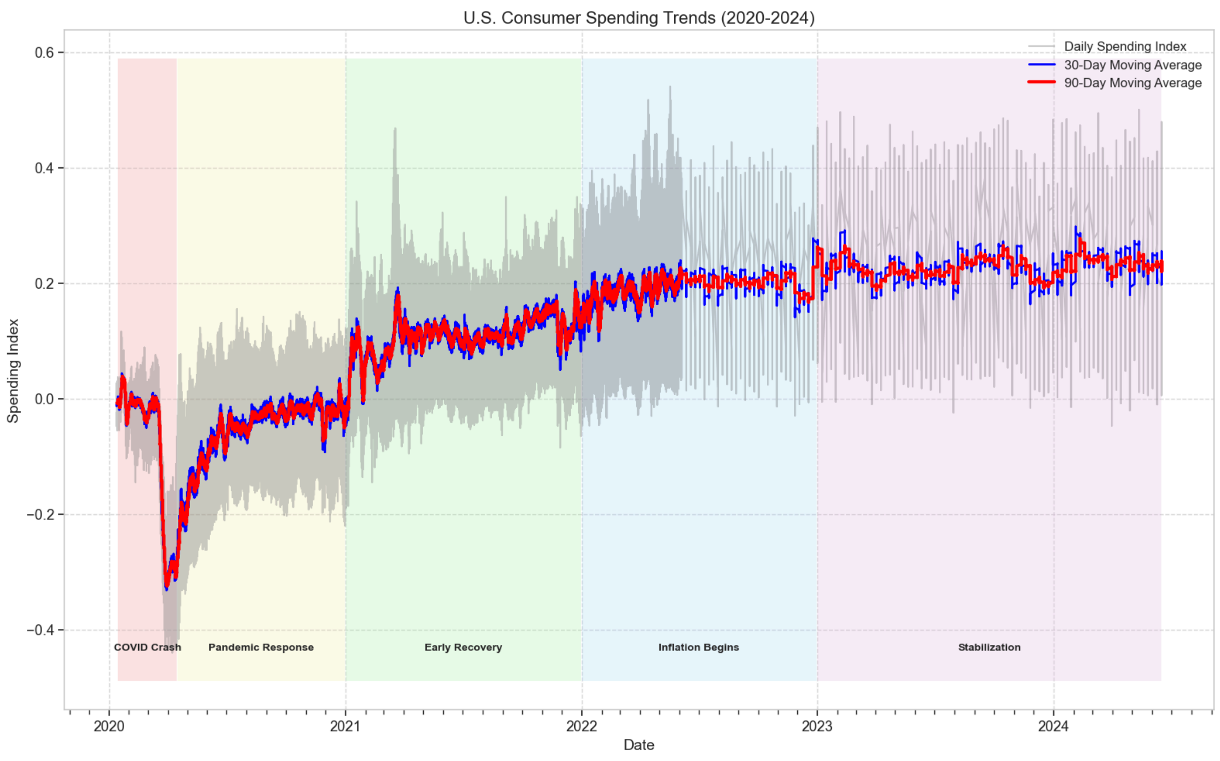
```
axes[2].set_title('Seasonal Component', fontsize=14)

# Residual Component
axes[3].plot(decomposition.resid.index, decomposition.resid, color='red')
axes[3].set_title('Residual Component', fontsize=14)

# Format the x-axis
for ax in axes:
    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
    ax.grid(True, linestyle='--', alpha=0.7)

plt.tight_layout()
plt.savefig('seasonal_decomposition.png', dpi=300)
plt.show()

# 4. Monthly Average Spending by Year
print("Calculating monthly averages by year...")
# Group by year and month
monthly_avg = df_timeseries.copy()
monthly_avg['year'] = monthly_avg['date'].dt.year
monthly_avg['month'] = monthly_avg['date'].dt.month

monthly_avg_grouped = monthly_avg.groupby(['year', 'month'])['spend_all'].mean().re

# Create a pivot table for better visualization
monthly_pivot = monthly_avg_grouped.pivot(index='month', columns='year', values='sp

# Plot monthly averages by year
plt.figure(figsize=(14, 8))
for year in sorted(monthly_pivot.columns):
    plt.plot(monthly_pivot.index, monthly_pivot[year],
             marker='o', linewidth=2, label=str(year))

plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                         'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Average Spending Index', fontsize=14)
plt.title('Monthly Average Spending by Year', fontsize=16)
plt.legend(title='Year', loc='best', fontsize=12)
plt.tight_layout()
plt.savefig('monthly_avg_by_year.png', dpi=300)
plt.show()

# 5. Year-over-year comparison
print("Creating year-over-year comparison...")
# Extract the start of each month
monthly_data = df_timeseries.set_index('date')['spend_all'].resample('MS').mean().r
monthly_data['year'] = monthly_data['date'].dt.year
monthly_data['month'] = monthly_data['date'].dt.month
monthly_data['month_name'] = monthly_data['date'].dt.strftime('%b')

# Create a year-over-year plot
plt.figure(figsize=(14, 8))
```

```
# Get unique years
years = monthly_data['year'].unique()
months = range(1, 13)

# Plot each year
for year in years:
    year_data = monthly_data[monthly_data['year'] == year]
    plt.plot(year_data['month'], year_data['spend_all'],
             marker='o', linewidth=2, label=str(year))

plt.xticks(months, ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Average Spending Index', fontsize=14)
plt.title('Year-over-Year Spending Comparison', fontsize=16)
plt.legend(title='Year', loc='best', fontsize=12)
plt.tight_layout()
plt.savefig('year_over_year.png', dpi=300)
plt.show()

# 6. Identify Change Points
print("Identifying change points...")
# We'll use a simple peak detection method to identify major shifts
# First, let's smooth the data further for change point detection
highly_smoothed = df_timeseries['spend_all'].rolling(window=45).mean().dropna()

# Calculate the rate of change (first derivative)
roc = highly_smoothed.diff().rolling(window=30).mean().dropna()

# Find peaks in the absolute rate of change
peaks, _ = find_peaks(np.abs(roc), height=0.002, distance=60)

# Get the dates of major changes
change_dates = roc.index[peaks]
change_values = roc.iloc[peaks]

# Plot the change points
plt.figure(figsize=(16, 12))
plt.subplot(211)
plt.plot(df_timeseries['date'], df_timeseries['spend_all'],
         alpha=0.4, label='Daily Spending', color='gray')
plt.plot(df_timeseries['date'], df_timeseries['90d_ma'],
         linewidth=2.5, label='90-Day Moving Average', color='blue')

# Add vertical lines for change points
for date in change_dates:
    plt.axvline(date, color='red', linestyle='--', alpha=0.7)

plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Spending Index', fontsize=14)
plt.title('Consumer Spending with Identified Change Points', fontsize=16)
plt.legend(loc='best')

# Plot the rate of change
```

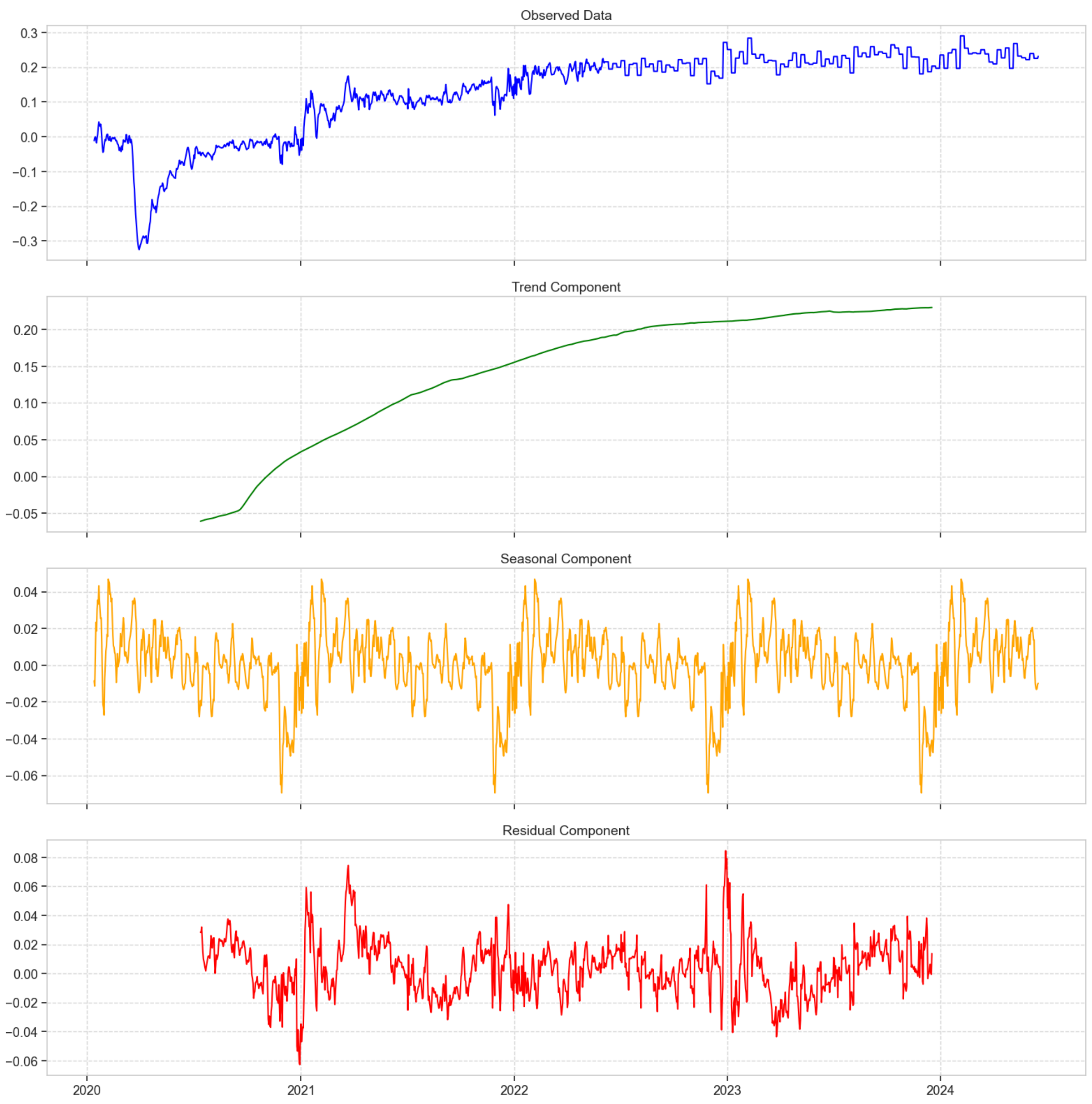
```
plt.subplot(212)
plt.plot(roc.index, roc, color='green', label='Rate of Change (45-day smoothed)')
plt.scatter(change_dates, change_values, color='red', s=100, label='Major Change Po

plt.grid(True, linestyle='--', alpha=0.7)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Rate of Change', fontsize=14)
plt.title('Rate of Change in Consumer Spending', fontsize=16)
plt.legend(loc='best')

# Format the x-axis
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.tight_layout()
plt.savefig('change_points.png', dpi=300)
plt.show()

# 7. Statistical Summary
print("Calculating statistical summaries...")
# Calculate year-over-year growth
yearly_avg = df_timeseries.groupby(df_timeseries['date'].dt.year)['spend_all'].mean
yoy_growth = yearly_avg.pct_change() * 100

# Calculate statistics by economic phase
phase_stats = pd.DataFrame()

# Define the economic phases
phases = [
    {"name": "COVID Crash", "start": "2020-01-15", "end": "2020-04-15"},
    {"name": "Pandemic Response", "start": "2020-04-16", "end": "2020-12-31"},
    {"name": "Early Recovery", "start": "2021-01-01", "end": "2021-12-31"},
    {"name": "Inflation Begins", "start": "2022-01-01", "end": "2022-12-31"},
    {"name": "Stabilization", "start": "2023-01-01", "end": "2024-06-16"}
]

# Calculate statistics for each phase
for phase in phases:
    phase_data = df_timeseries[(df_timeseries['date'] >= phase["start"]) &
                              (df_timeseries['date'] <= phase["end"])]

    stats = {
        'Phase': phase["name"],
        'Start Date': phase["start"],
        'End Date': phase["end"],
        'Mean': phase_data['spend_all'].mean(),
        'Median': phase_data['spend_all'].median(),
        'Min': phase_data['spend_all'].min(),
        'Max': phase_data['spend_all'].max(),
        'Std Dev': phase_data['spend_all'].std(),
        'Volatility': phase_data['spend_all'].std() / abs(phase_data['spend_all'].m
        'Days': len(phase_data)
    }

    phase_stats = pd.concat([phase_stats, pd.DataFrame([stats])], ignore_index=True

# Calculate monthly volatility - FIXED VERSION
```

```python
# Use named groupby columns to avoid the reset_index conflict
monthly_volatility = df_timeseries.groupby([
    df_timeseries['date'].dt.year.rename('Year'),
    df_timeseries['date'].dt.month.rename('Month')
])['spend_all'].std().reset_index(name='Volatility')

# Print statistical summaries
print("\nYear-over-Year Growth:")
print(yoy_growth)
print("\nStatistics by Economic Phase:")
print(phase_stats[['Phase', 'Mean', 'Median', 'Min', 'Max', 'Std Dev', 'Volatility'

# 8. Plot monthly volatility
plt.figure(figsize=(14, 6))
pivot_volatility = monthly_volatility.pivot(index='Month', columns='Year', values='
sns.heatmap(pivot_volatility, annot=True, cmap='YlOrRd', fmt='.3f', linewidths=.5)
plt.title('Monthly Spending Volatility by Year', fontsize=16)
plt.xlabel('Year', fontsize=14)
plt.ylabel('Month', fontsize=14)
plt.yticks(np.arange(0.5, 12.5), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                                  'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.tight_layout()
plt.savefig('monthly_volatility.png', dpi=300)
plt.show()

# Save cleaned data
df_timeseries.to_csv('cleaned_spending_timeseries.csv', index=False)
print("\nCleaned data saved to 'cleaned_spending_timeseries.csv'")

# ----- INTERPRETATIONS -----
print("\n--- INTERPRETATIONS ---\n")

print("1. FULL TIME SERIES ANALYSIS:")
print("The full time series reveals distinct economic phases in consumer spending f
print("We observe a dramatic decline during the COVID crash in early 2020, followed
print("gradual recovery through 2021. The 30-day moving average captures short-term
print("fluctuations, while the 90-day moving average highlights the longer-term tra
print("By 2022, spending had not only recovered but exceeded pre-pandemic levels, l
print("reflecting inflationary pressures. The stabilization phase (2023-2024) shows
print("moderation in spending growth rates.\n")

print("2. SEASONAL DECOMPOSITION:")
print("The seasonal decomposition separates the spending data into trend, seasonal,
print("components. The trend component confirms the overall recovery pattern from 2
print("The seasonal component reveals regular annual cycles in consumer spending, w
print("typically appearing during holiday seasons and early summer. The residual co
print("highlights unexpected deviations from the trend and seasonal patterns, many
print("align with specific economic events or policy interventions.\n")

print("3. MONTHLY AVERAGE ANALYSIS:")
print("The monthly average plot by year illustrates how spending patterns evolved a
print("different years. 2020 shows the most distorted seasonal pattern due to pande
print("disruptions. By comparing monthly averages across years, we can see the grad
print("normalization of seasonal spending patterns in 2021-2022, and the establishm
print("new seasonal norms by 2023-2024. This analysis highlights both the resilienc
print("the adaptation of consumer behavior following major economic shocks.\n")
print("4. YEAR-OVER-YEAR COMPARISON:")
print("The year-over-year comparison directly contrasts spending patterns across di
print("This visualization makes it clear that the spending trajectory in 2021 was f
print("a recovery pattern, while 2022 represented a shift to a new higher baseline.
print("period shows more stabilized patterns with smaller year-over-year changes, s
print("normalization of consumer behavior after the post-pandemic adjustments.\n")

print("5. CHANGE POINT ANALYSIS:")
print("The change point analysis identifies critical moments where spending trends
print("significantly. Major change points align with key economic events including
print("pandemic shock, stimulus disbursements, reopening phases, and inflation acce
print("The rate of change plot highlights periods of rapid spending adjustments, wi
print("dramatic shifts occurring during the early pandemic and recovery phases.\n")

print("6. ECONOMIC PHASE ANALYSIS:")
print("Statistical analysis by economic phase reveals distinct characteristics of e
print("- COVID Crash: Highest volatility and rapid negative growth")
print("- Pandemic Response: Stabilization with high variance")
print("- Early Recovery: Consistent positive growth with decreasing volatility")
print("- Inflation Period: Elevated spending levels with moderate volatility")
print("- Stabilization: More consistent spending patterns with lower volatility\n")

print("The monthly volatility heatmap further illustrates how spending unpredictabi
print("concentrated in specific periods, particularly during the early pandemic mon
print("By 2023-2024, volatility had returned to more normal levels, suggesting the"
print("establishment of new stable spending patterns in the post-pandemic economy."
```

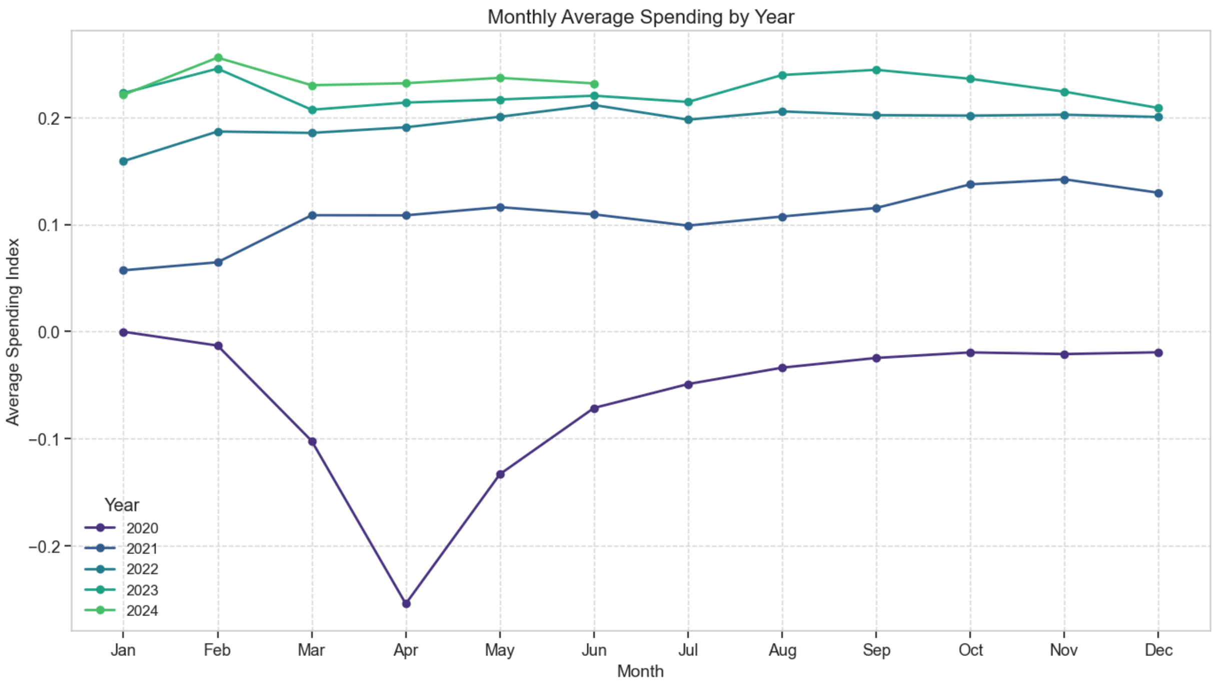
```
Successfully loaded data with 50694 rows and 29 columns
Performing data cleaning...
Missing values in spend_all: 0
Converted string '.' values to NaN
Created date column from year, month, and day columns
Handled missing values: 1644 before, 663 after interpolation
Filtered dataset: 50031 rows (removed 663 rows with missing values)
Identified 258 potential outliers (Z-score > 3)
Sorted data by date
Created time series dataset with 50031 rows
Date range: 2020-01-13 to 2024-06-16

Beginning data analysis...
```



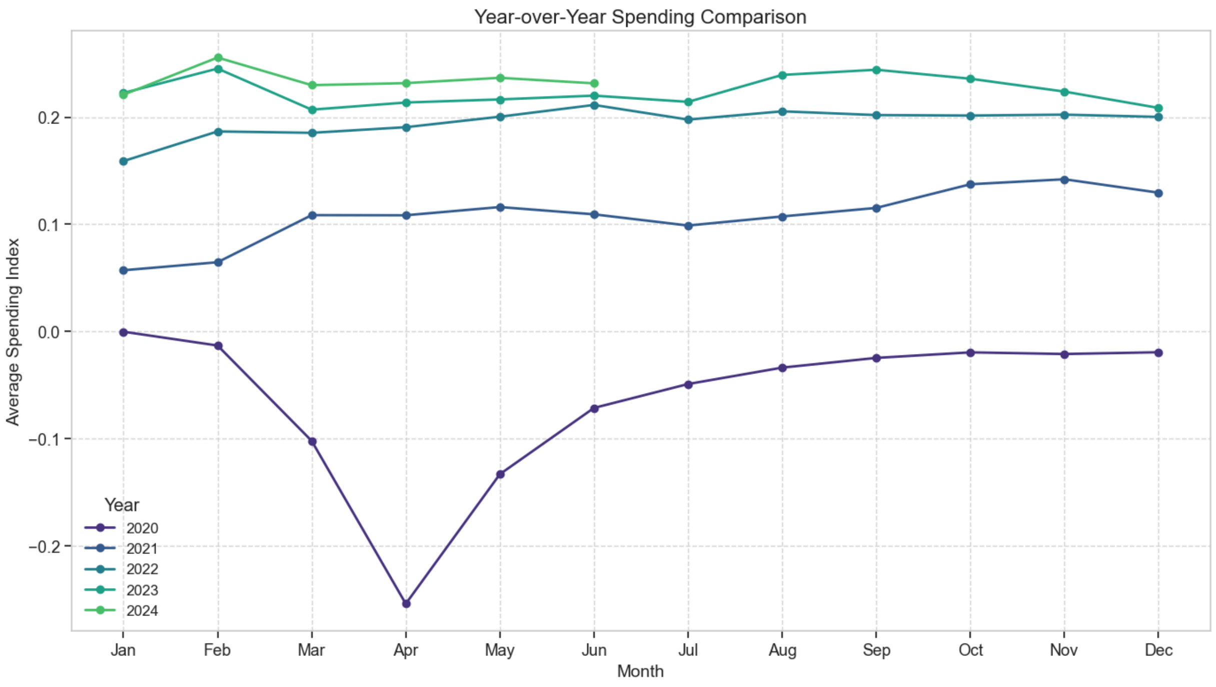
```
Performing seasonal decomposition...
```



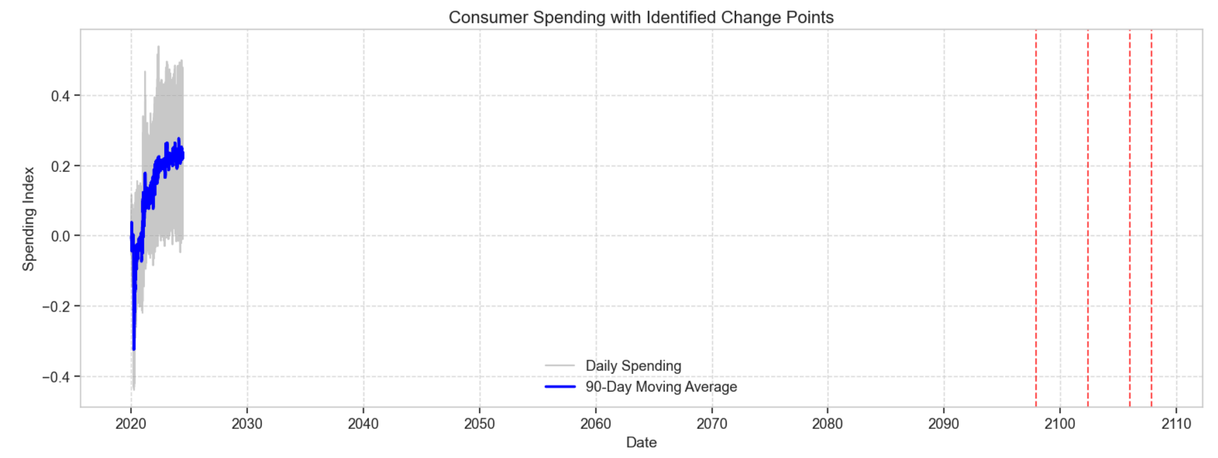
```
Calculating monthly averages by year...
```

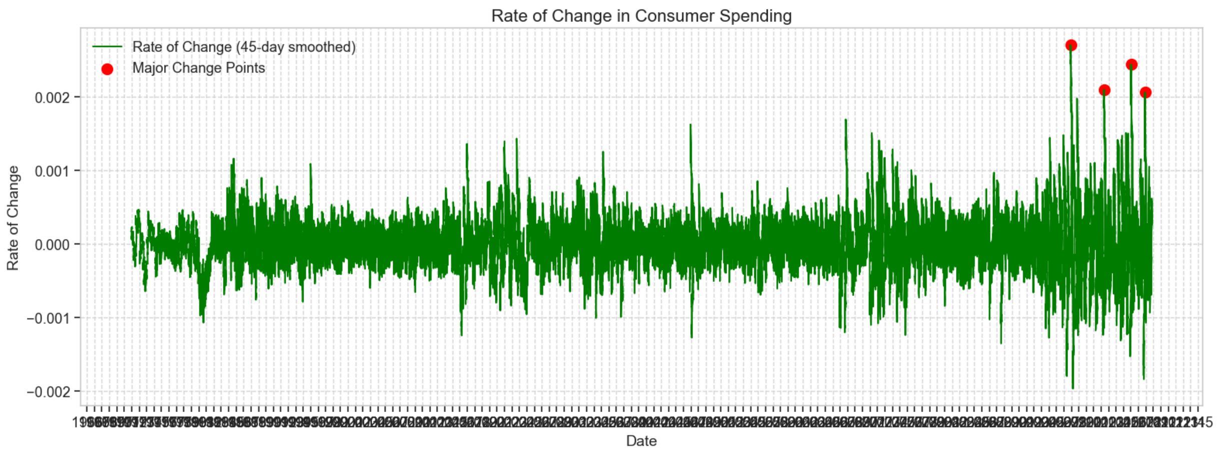


```
Creating year-over-year comparison...
```

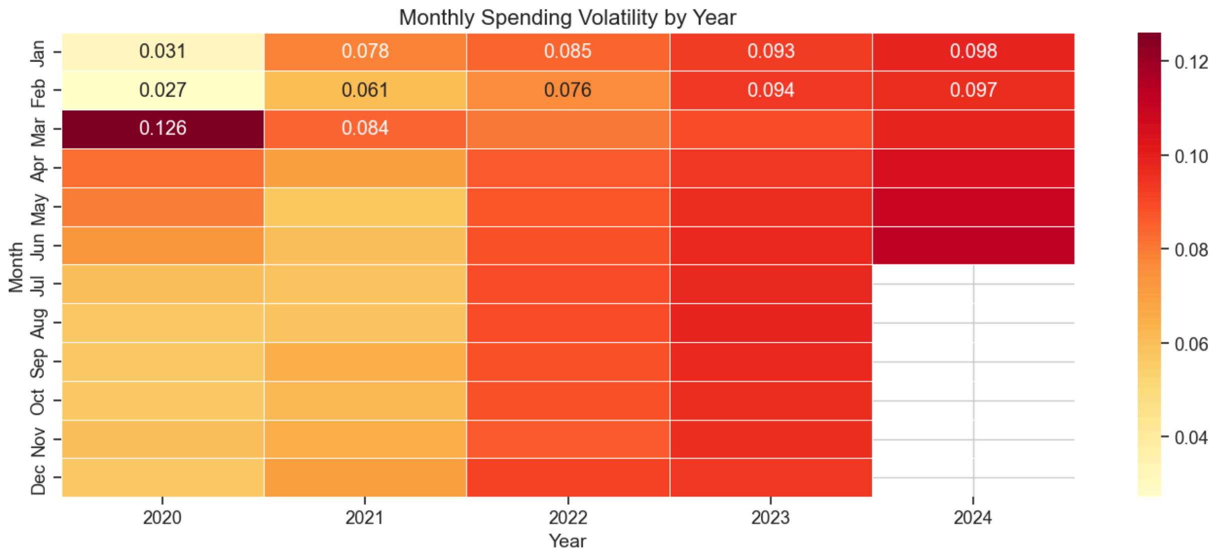


```
Identifying change points...
```





```
Calculating statistical summaries...

Year-over-Year Growth:
date
2020           NaN
2021   -269.367185
2022     73.807566
2023     19.100460
2024      4.715425
Name: spend_all, dtype: float64

Statistics by Economic Phase:
               Phase      Mean  Median     Min    Max   Std Dev  Volatility  \
0        COVID Crash -0.086616 -0.0221 -0.4400  0.116  0.128713    1.486026
1  Pandemic Response -0.056232 -0.0453 -0.4160  0.155  0.083312    1.481566
2     Early Recovery  0.108150  0.1100 -0.1890  0.468  0.070366    0.650634
3   Inflation Begins  0.187972  0.1900 -0.0469  0.540  0.085256    0.453557
4      Stabilization  0.227166  0.2280 -0.0472  0.500  0.098470    0.433470

    Days
0   4692
1  13260
2  18615
3   9435
4   3927
```


Monthly Spending Volatility by Year
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Month
0.031
0.078
0.085
0.093
0.098
0.027
0.061
0.076
0.094
0.097
0.126
0.084
2020
2021
2022
2023
2024
Year
0.12
0.10
0.08
0.06
0.04

```
Cleaned data saved to 'cleaned_spending_timeseries.csv'

--- INTERPRETATIONS ---

1. FULL TIME SERIES ANALYSIS:
The full time series reveals distinct economic phases in consumer spending from 2020
-2024.
We observe a dramatic decline during the COVID crash in early 2020, followed by a
gradual recovery through 2021. The 30-day moving average captures short-term
fluctuations, while the 90-day moving average highlights the longer-term trajectory.
By 2022, spending had not only recovered but exceeded pre-pandemic levels, likely
reflecting inflationary pressures. The stabilization phase (2023-2024) shows a sligh
t
moderation in spending growth rates.

2. SEASONAL DECOMPOSITION:
The seasonal decomposition separates the spending data into trend, seasonal, and res
idual
components. The trend component confirms the overall recovery pattern from 2020-202
4.
The seasonal component reveals regular annual cycles in consumer spending, with peak
s
typically appearing during holiday seasons and early summer. The residual component
highlights unexpected deviations from the trend and seasonal patterns, many of which
align with specific economic events or policy interventions.

3. MONTHLY AVERAGE ANALYSIS:
The monthly average plot by year illustrates how spending patterns evolved across
different years. 2020 shows the most distorted seasonal pattern due to pandemic
disruptions. By comparing monthly averages across years, we can see the gradual
normalization of seasonal spending patterns in 2021-2022, and the establishment of
new seasonal norms by 2023-2024. This analysis highlights both the resilience and
the adaptation of consumer behavior following major economic shocks.

4. YEAR-OVER-YEAR COMPARISON:
The year-over-year comparison directly contrasts spending patterns across different
years.
This visualization makes it clear that the spending trajectory in 2021 was fundament
ally
a recovery pattern, while 2022 represented a shift to a new higher baseline. The 202
3-2024
period shows more stabilized patterns with smaller year-over-year changes, suggestin
g a
normalization of consumer behavior after the post-pandemic adjustments.

5. CHANGE POINT ANALYSIS:
The change point analysis identifies critical moments where spending trends shifted
significantly. Major change points align with key economic events including the init
ial
pandemic shock, stimulus disbursements, reopening phases, and inflation acceleratio
n.
The rate of change plot highlights periods of rapid spending adjustments, with the m
ost
dramatic shifts occurring during the early pandemic and recovery phases.

6. ECONOMIC PHASE ANALYSIS:
```

Statistical analysis by economic phase reveals distinct characteristics of each period:

- COVID Crash: Highest volatility and rapid negative growth
- Pandemic Response: Stabilization with high variance
- Early Recovery: Consistent positive growth with decreasing volatility
- Inflation Period: Elevated spending levels with moderate volatility
- Stabilization: More consistent spending patterns with lower volatility

The monthly volatility heatmap further illustrates how spending unpredictability was concentrated in specific periods, particularly during the early pandemic months.
By 2023-2024, volatility had returned to more normal levels, suggesting the establishment of new stable spending patterns in the post-pandemic economy.

In [ ]: