CoffeeSales

September 20, 2024

1 Importing Libariries

[ ]:
```
import pandas as pd
```

2 Laoding Dataset

[4]:
```
# Load the dataset to inspect the columns and first few rows
file_path = r"C:\Users\Admin\Downloads\index.csv"
coffee_sales_data = pd.read_csv(file_path)
```

[8]:
```
# Display the first few rows to understand the structure of the data
data_head = coffee_sales_data.head()
coffee_sales_data
```

[8]:
```
            date                 datetime cash_type                 card  \
0     2024-03-01  2024-03-01 10:15:50.520      card  ANON-0000-0000-0001
1     2024-03-01  2024-03-01 12:19:22.539      card  ANON-0000-0000-0002
2     2024-03-01  2024-03-01 12:20:18.089      card  ANON-0000-0000-0002
3     2024-03-01  2024-03-01 13:46:33.006      card  ANON-0000-0000-0003
4     2024-03-01  2024-03-01 13:48:14.626      card  ANON-0000-0000-0004
…            …                        …       …                    …
1128  2024-07-31  2024-07-31 20:53:35.077      card  ANON-0000-0000-0443
1129  2024-07-31  2024-07-31 20:59:25.013      card  ANON-0000-0000-0040
1130  2024-07-31  2024-07-31 21:26:26.000      card  ANON-0000-0000-0444
1131  2024-07-31  2024-07-31 21:54:11.824      card  ANON-0000-0000-0445
1132  2024-07-31  2024-07-31 21:55:16.570      card  ANON-0000-0000-0446

      money          coffee_name
0     38.70                Latte
1     38.70        Hot Chocolate
2     38.70        Hot Chocolate
3     28.90            Americano
4     38.70                Latte
…     …                    …
1128  23.02              Cortado
1129  27.92  Americano with Milk
1130  32.82                Latte
```

```
1131  32.82                Latte
1132  32.82                Latte

[1133 rows x 6 columns]
```

[10]:
```python
# Display the columns and their data types
data_info = coffee_sales_data.info()

data_head, data_info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1133 entries, 0 to 1132
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         1133 non-null   object
 1   datetime     1133 non-null   object
 2   cash_type    1133 non-null   object
 3   card         1044 non-null   object
 4   money        1133 non-null   float64
 5   coffee_name  1133 non-null   object
dtypes: float64(1), object(5)
memory usage: 53.2+ KB
```

[10]:
```
(         date                 datetime cash_type                 card  money  \
 0  2024-03-01  2024-03-01 10:15:50.520      card  ANON-0000-0000-0001   38.7
 1  2024-03-01  2024-03-01 12:19:22.539      card  ANON-0000-0000-0002   38.7
 2  2024-03-01  2024-03-01 12:20:18.089      card  ANON-0000-0000-0002   38.7
 3  2024-03-01  2024-03-01 13:46:33.006      card  ANON-0000-0000-0003   28.9
 4  2024-03-01  2024-03-01 13:48:14.626      card  ANON-0000-0000-0004   38.7

      coffee_name
 0          Latte
 1  Hot Chocolate
 2  Hot Chocolate
 3      Americano
 4          Latte  ,
 None)
```

3 Data Cleaning

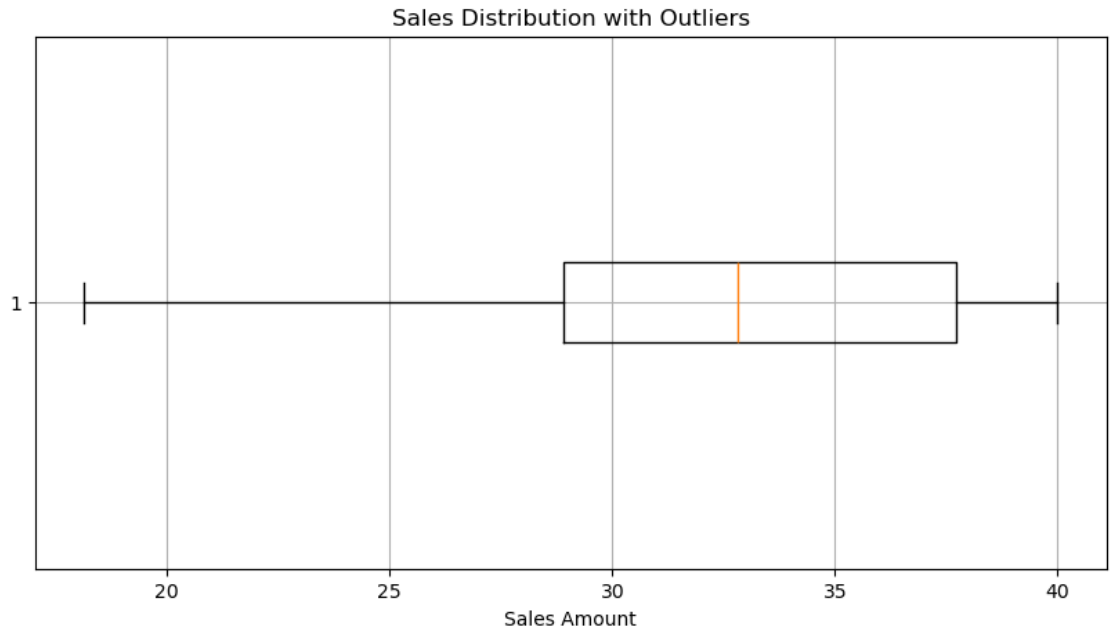
[12]:
```python
# 1. Identify missing data
missing_data_summary = coffee_sales_data.isnull().sum()

# Display the summary of missing values in each column
missing_data_summary
```

```
[12]: date            0
      datetime        0
      cash_type       0
      card           89
      money           0
      coffee_name     0
      dtype: int64
```

```python
[14]: # Replace missing values in the 'card' column with 'Unknown'
coffee_sales_data['card'].fillna('Cash Payment', inplace=True)

# Verify that there are no more missing values
missing_data_summary_after_cleaning = coffee_sales_data.isnull().sum()

# Display the summary of missing values after cleaning
missing_data_summary_after_cleaning
```

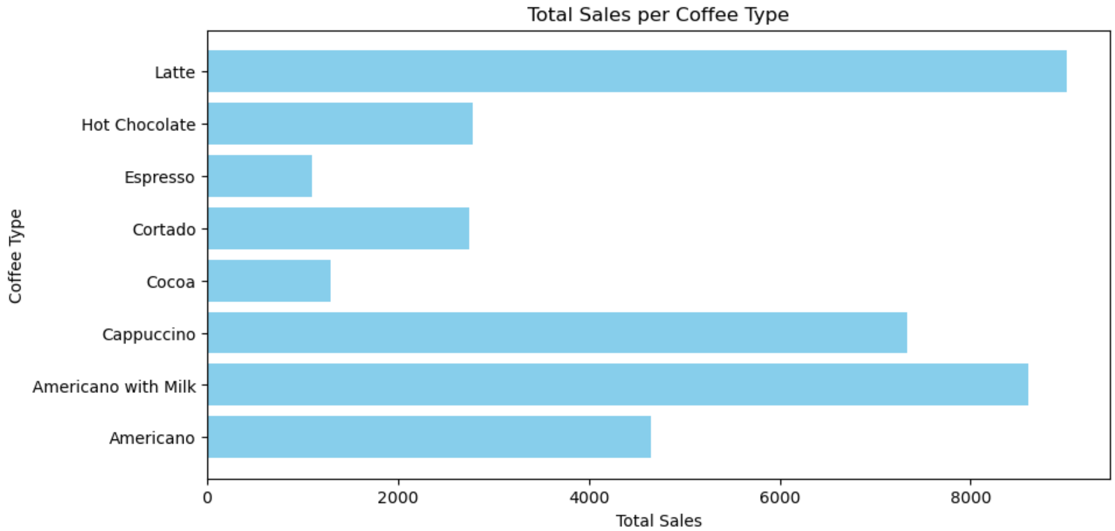
```
[14]: date           0
      datetime       0
      cash_type      0
      card           0
      money          0
      coffee_name    0
      dtype: int64
```

```python
[16]: # Outlier Detection in the 'money' column using Interquartile Range (IQR)
import matplotlib.pyplot as plt
Q1 = coffee_sales_data['money'].quantile(0.25)
Q3 = coffee_sales_data['money'].quantile(0.75)
IQR = Q3 - Q1

# Determine the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = coffee_sales_data[(coffee_sales_data['money'] < lower_bound) |␣
 ↪(coffee_sales_data['money'] > upper_bound)]

# Visualize sales distribution with outliers using a boxplot
plt.figure(figsize=(10, 5))
plt.boxplot(coffee_sales_data['money'], vert=False)
plt.title('Sales Distribution with Outliers')
plt.xlabel('Sales Amount')
plt.grid(True)
plt.show()
```



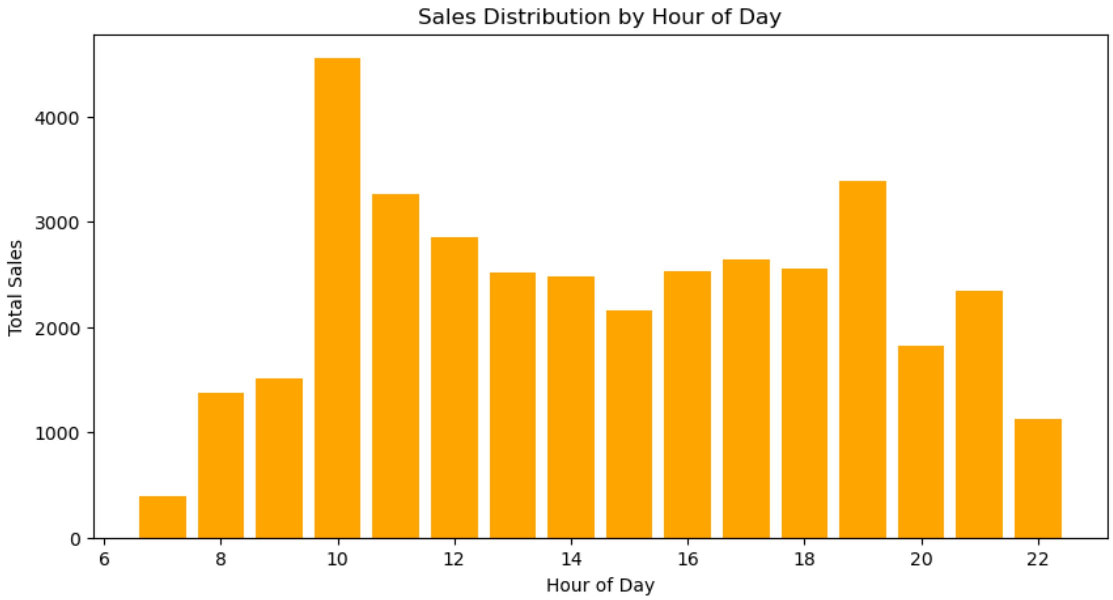
4 Exploratory Data Analysis

[25]:
```
# Distribution per Coffee Type
coffee_type_sales = coffee_sales_data.groupby('coffee_name')['money'].sum()

# Visualize the distribution
plt.figure(figsize=(10, 5))
plt.barh(coffee_type_sales.index, coffee_type_sales.values, color='skyblue')
plt.title('Total Sales per Coffee Type')
plt.xlabel('Total Sales')
plt.ylabel('Coffee Type')
plt.show()

# Display coffee sales per coffee type
coffee_type_sales
```



[25]:
```
coffee_name
Americano              4644.54
Americano with Milk    8601.94
Cappuccino             7333.14
Cocoa                  1295.94
Cortado                2745.08
Espresso               1100.62
Hot Chocolate          2778.48
Latte                  9009.14
Name: money, dtype: float64
```

[29]:
```
# Convert 'date' and 'datetime' columns to datetime objects
coffee_sales_data['date'] = pd.to_datetime(coffee_sales_data['date'])
coffee_sales_data['datetime'] = pd.to_datetime(coffee_sales_data['datetime'])
# Extract the hour from 'datetime' to analyze distribution by time of day
coffee_sales_data['hour'] = coffee_sales_data['datetime'].dt.hour
purchasing_time_distribution = coffee_sales_data.groupby('hour')['money'].sum()

# Visualize the distribution per purchasing time
plt.figure(figsize=(10, 5))
plt.bar(purchasing_time_distribution.index, purchasing_time_distribution.
 ↪values, color='orange')
plt.title('Sales Distribution by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Total Sales')
plt.show()

# Display sales distribution by hour
purchasing_time_distribution
```



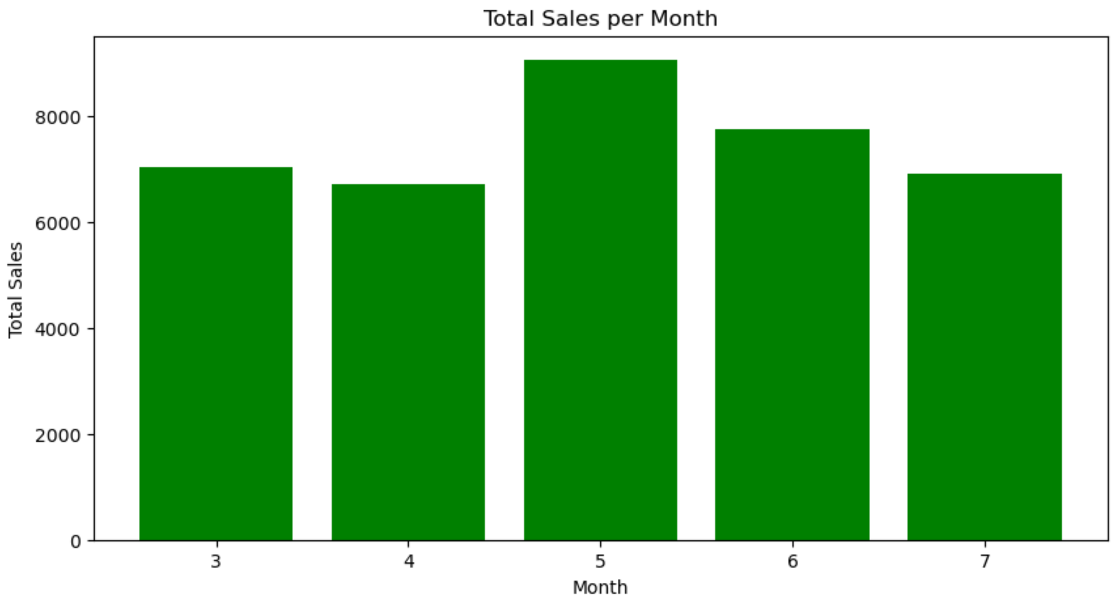
[29]:

```
hour
7        392.80
8       1380.38
9       1515.48
10      4553.18
11      3258.64
12      2850.60
13      2511.60
14      2484.92
15      2158.76
16      2525.36
17      2639.08
18      2558.04
19      3388.32
20      1819.92
21      2343.86
22      1127.94
Name: money, dtype: float64
```

[31]:

```python
# Extract the month from 'datetime' to analyze sales by month
coffee_sales_data['month'] = coffee_sales_data['datetime'].dt.month
monthly_sales = coffee_sales_data.groupby('month')['money'].sum()

# Visualize coffee sales per month
plt.figure(figsize=(10, 5))
plt.bar(monthly_sales.index, monthly_sales.values, color='green')
plt.title('Total Sales per Month')
```

```
plt.xlabel('Month')
plt.ylabel('Total Sales')
plt.show()

# Display monthly sales
monthly_sales
```



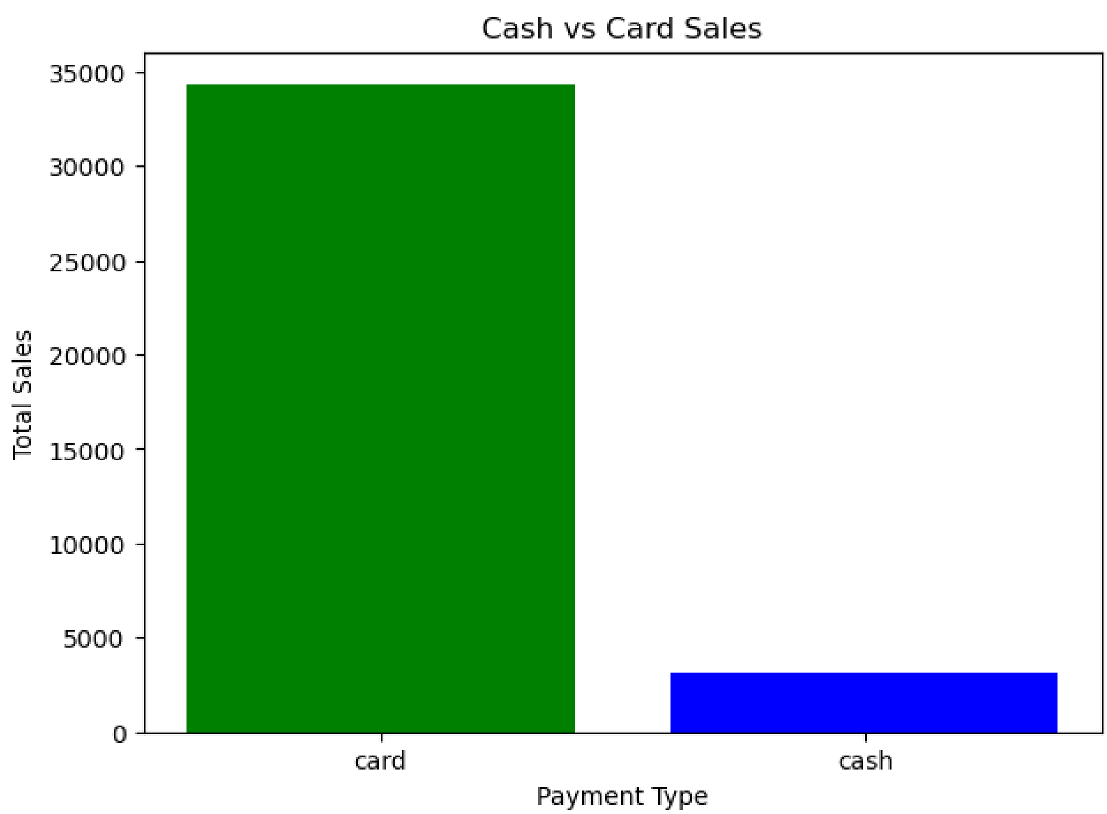
[31]:
```
month
3    7050.20
4    6720.56
5    9063.42
6    7758.76
7    6915.94
Name: money, dtype: float64
```

[39]:
```
total_money = coffee_sales_data['money'].sum()
total_money
```

[39]:
```
37508.880000000005
```

[37]:
```
# Cash Money vs Card Money
cash_vs_card_sales = coffee_sales_data.groupby('cash_type')['money'].sum()

# Visualize Cash vs Card Sales
plt.figure(figsize=(7, 5))
plt.bar(cash_vs_card_sales.index, cash_vs_card_sales.values, color=['green',␣
↪'blue'])
```

```
plt.title('Cash vs Card Sales')
plt.xlabel('Payment Type')
plt.ylabel('Total Sales')
plt.show()

# Display cash vs card sales
cash_vs_card_sales
```



[37]:

```
cash_type
card    34322.88
cash     3186.00
Name: money, dtype: float64
```

[41]:

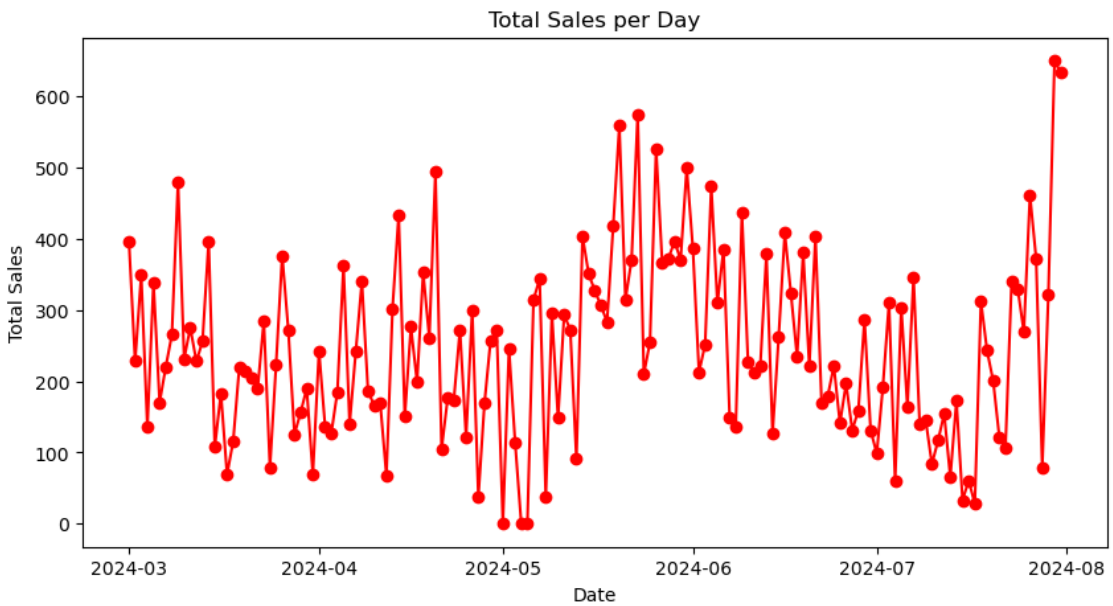
```
# Sales per week
coffee_sales_data['week'] = coffee_sales_data['datetime'].dt.isocalendar().week
weekly_sales = coffee_sales_data.groupby('week')['money'].sum()

# Visualize weekly sales
plt.figure(figsize=(10, 5))
plt.plot(weekly_sales.index, weekly_sales.values, marker='o', color='purple')
plt.title('Total Sales per Week')
```

```
plt.xlabel('Week Number')
plt.ylabel('Total Sales')
plt.show()

# Display weekly sales
weekly_sales
```

[41]:
```
week
9        973.50
10      1840.50
11      1516.30
12      1307.80
13      1412.10
14      1434.50
15      1666.00
16      1838.84
17      1251.20
18       890.18
19      1705.80
20      2180.26
21      2811.80
22      2605.00
23      2143.52
24      1835.98
25      1911.42
26      1268.24
27      1475.42
```

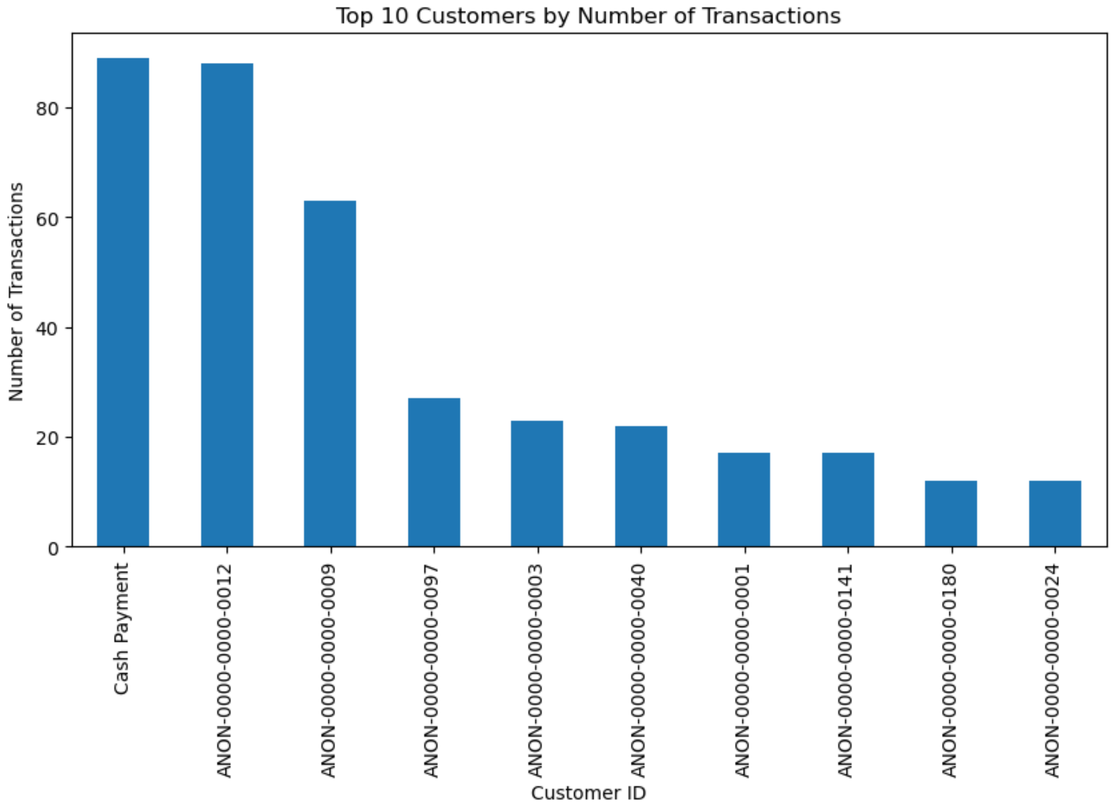
```
28      876.80
29      998.28
30     1959.30
31     1606.14
Name: money, dtype: float64
```

[68]:
```
# Sales per day
daily_sales = coffee_sales_data['money'].resample('D').sum()

# Visualize daily sales
plt.figure(figsize=(10, 5))
plt.plot(daily_sales.index, daily_sales.values, marker='o', color='red')
plt.title('Total Sales per Day')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.show()

# Display daily sales (sample)
daily_sales.head()
```



[68]:
```
date
2024-03-01    396.3
2024-03-02    228.1
2024-03-03    349.1
2024-03-04    135.2
2024-03-05    338.5
Freq: D, Name: money, dtype: float64
```

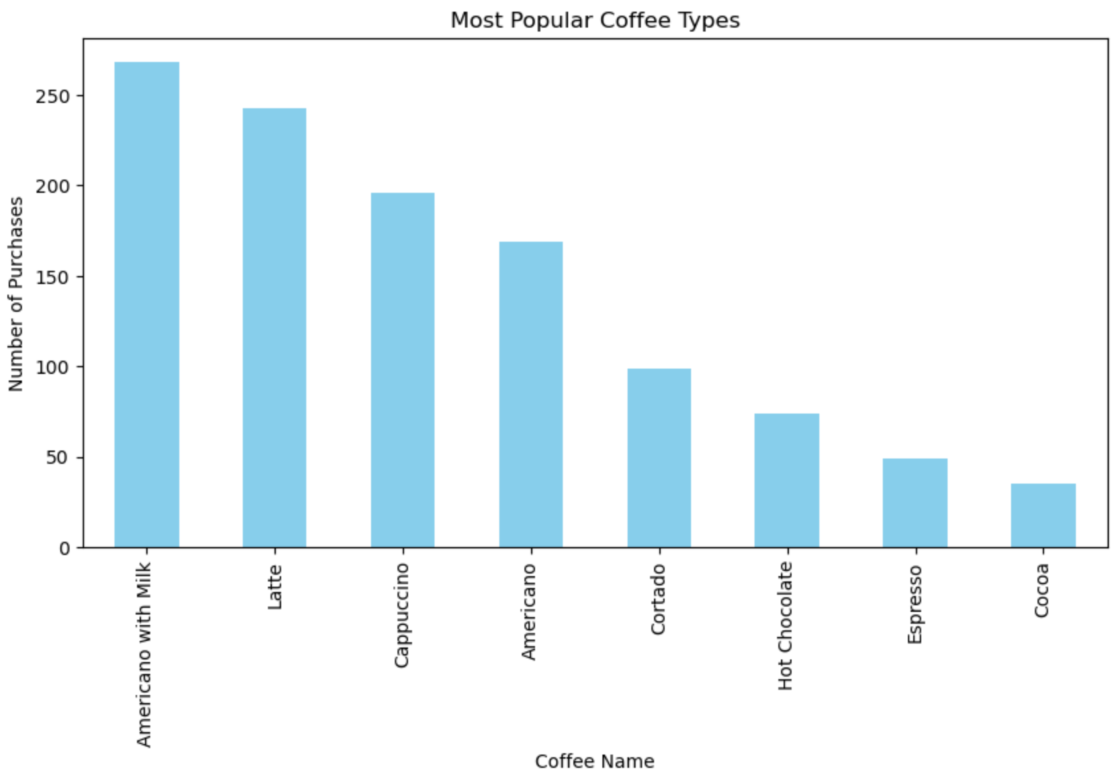
[45]:
```python
# Top Customers by Number of Transactions
top_customers = coffee_sales_data['card'].value_counts().head(10)

# Visualize the top customers
plt.figure(figsize=(10, 5))
top_customers.plot(kind='bar')
plt.title('Top 10 Customers by Number of Transactions')
plt.xlabel('Customer ID')
plt.ylabel('Number of Transactions')
plt.show()

# Display top customers
top_customers
```



[45]:
```
card
Cash Payment            89
ANON-0000-0000-0012     88
ANON-0000-0000-0009     63
ANON-0000-0000-0097     27
ANON-0000-0000-0003     23
ANON-0000-0000-0040     22
```

```
ANON-0000-0000-0001    17
ANON-0000-0000-0141    17
ANON-0000-0000-0180    12
ANON-0000-0000-0024    12
Name: count, dtype: int64
```

[49]:
```python
# Popular Coffee Types
popular_coffees = coffee_sales_data['coffee_name'].value_counts()

# Visualize the most popular coffee types
plt.figure(figsize=(10, 5))
popular_coffees.plot(kind='bar', color='skyblue')
plt.title('Most Popular Coffee Types')
plt.xlabel('Coffee Name')
plt.ylabel('Number of Purchases')
plt.show()

# Display popular coffee types
popular_coffees
```

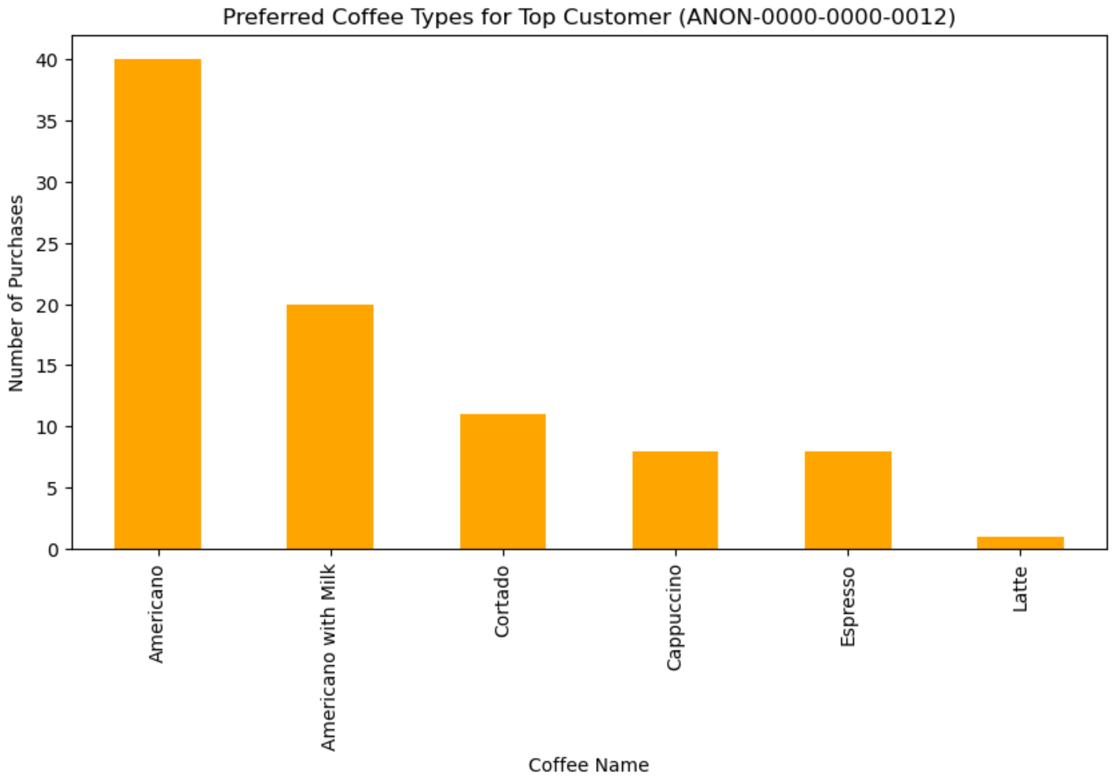


[49]:
```
coffee_name
Americano with Milk    268
```

```
Latte              243
Cappuccino         196
Americano          169
Cortado             99
Hot Chocolate       74
Espresso            49
Cocoa               35
Name: count, dtype: int64
```

[65]:

```python
# Exclude "Cash Payment" from the top customer analysis
valid_customers = coffee_sales_data[coffee_sales_data['card'] != 'Cash Payment']

# Analyze Purchasing Patterns for the Top Customer (excluding 'Cash Payment')
top_customer_id = valid_customers['card'].value_counts().index[0]
top_customer_purchases = valid_customers[valid_customers['card'] ==␣
 ↪top_customer_id]['coffee_name'].value_counts()

# Visualize the preferred coffee types for the top customer
plt.figure(figsize=(10, 5))
top_customer_purchases.plot(kind='bar', color='orange')
plt.title(f'Preferred Coffee Types for Top Customer ({top_customer_id})')
plt.xlabel('Coffee Name')
plt.ylabel('Number of Purchases')
plt.show()

# Display preferred coffee types for the top customer
top_customer_purchases
```

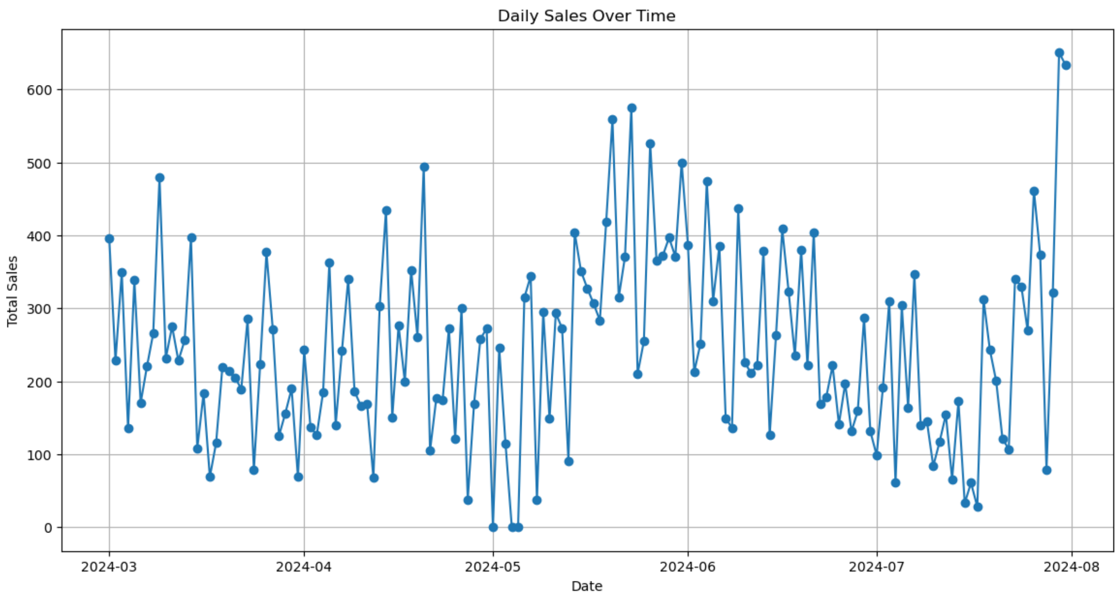


[65]:
```
coffee_name
Americano              40
Americano with Milk    20
Cortado                11
Cappuccino              8
Espresso                8
Latte                   1
Name: count, dtype: int64
```

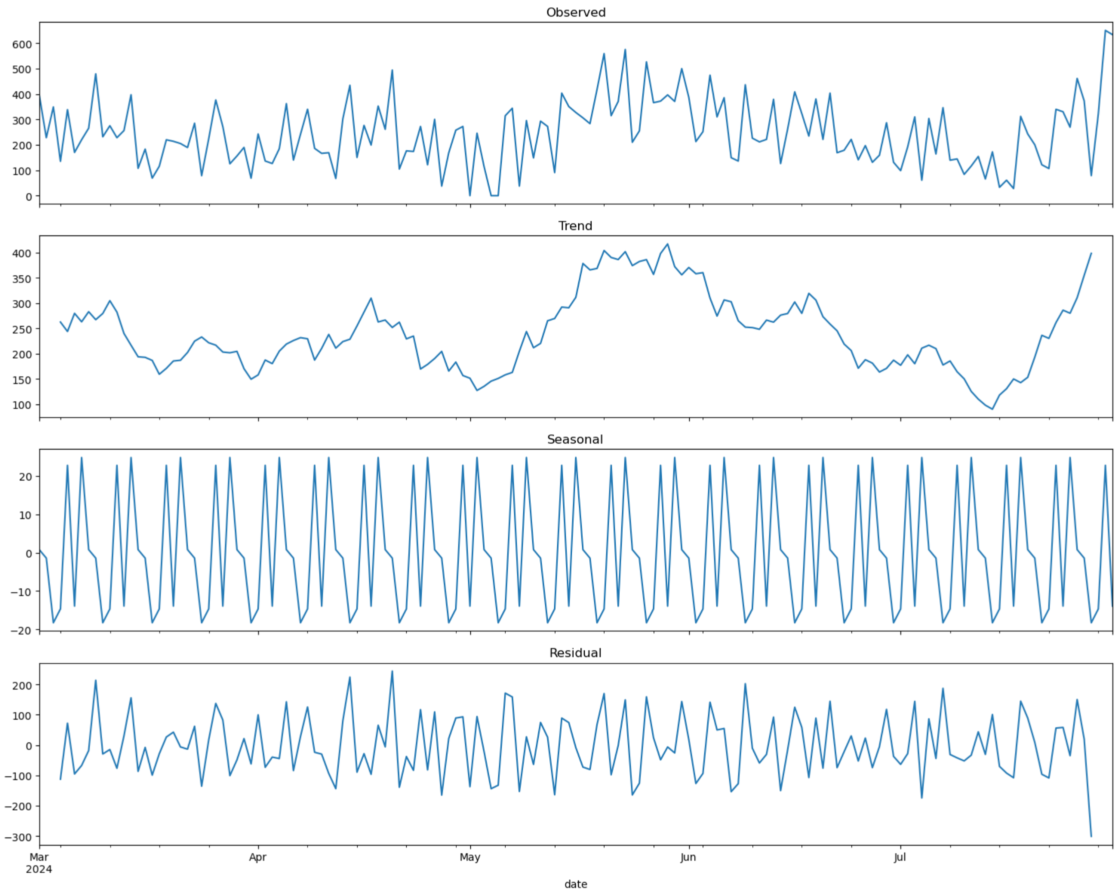
5 Time Series EDA

[53]:
```python
import matplotlib.pyplot as plt

# Convert 'date' and 'datetime' columns to datetime objects
coffee_sales_data['date'] = pd.to_datetime(coffee_sales_data['date'])
coffee_sales_data['datetime'] = pd.to_datetime(coffee_sales_data['datetime'])

# Set 'date' as the index for time series analysis
coffee_sales_data.set_index('date', inplace=True)

# Resample the data to daily sales sums
```

```
daily_sales = coffee_sales_data['money'].resample('D').sum()

# Plot the daily sales to explore trends over time
plt.figure(figsize=(14, 7))
plt.plot(daily_sales, marker='o', linestyle='-')
plt.title('Daily Sales Over Time')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.grid(True)
plt.show()
```



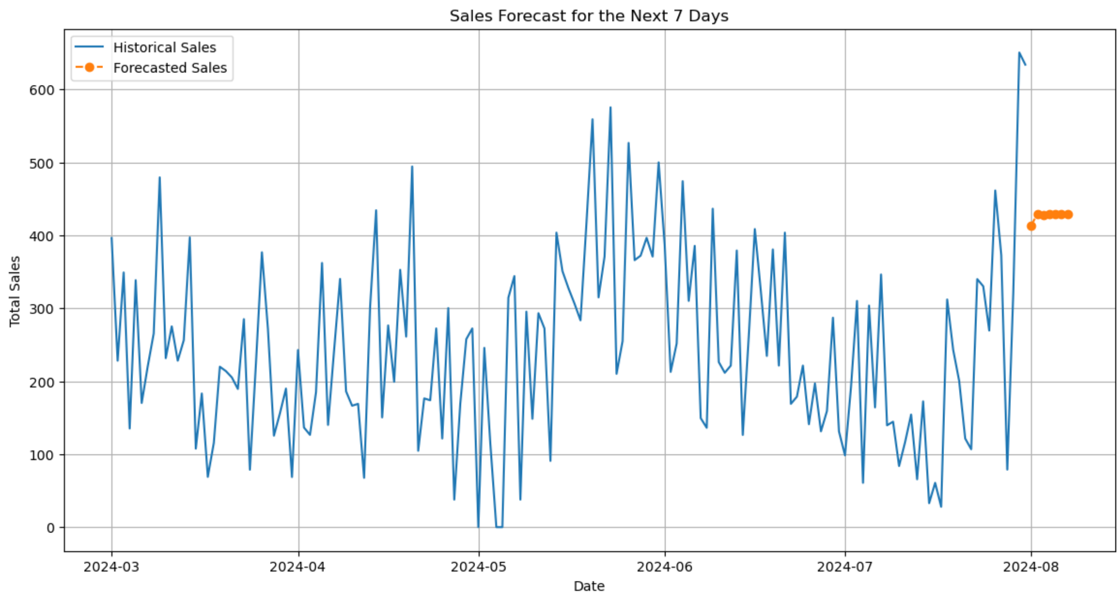
[55]:
```
from statsmodels.tsa.seasonal import seasonal_decompose

# Perform seasonal decomposition
decomposition = seasonal_decompose(daily_sales, model='additive', period=7)  #␣
↪Assuming weekly seasonality

# Plot the decomposed components
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(15, 12), sharex=True)
decomposition.observed.plot(ax=ax1, title='Observed')
decomposition.trend.plot(ax=ax2, title='Trend')
decomposition.seasonal.plot(ax=ax3, title='Seasonal')
decomposition.resid.plot(ax=ax4, title='Residual')

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



6 Next Day / Week / Month Sales

[57]:
```
from statsmodels.tsa.arima.model import ARIMA

# Fit an ARIMA model to the daily sales data
# Setting the order parameter based on the decomposition: (p,d,q) = (1,1,1) as␣
↪a starting point
arima_model = ARIMA(daily_sales, order=(1, 1, 1))
arima_fit = arima_model.fit()

# Forecast the next 7 days
forecast = arima_fit.forecast(steps=7)

# Plot the historical sales data along with the forecast
plt.figure(figsize=(14, 7))
plt.plot(daily_sales, label='Historical Sales')
plt.plot(forecast.index, forecast, label='Forecasted Sales', linestyle='--',␣
↪marker='o')
```

```python
plt.title('Sales Forecast for the Next 7 Days')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True)
plt.show()

# Display the forecasted values
forecast
```



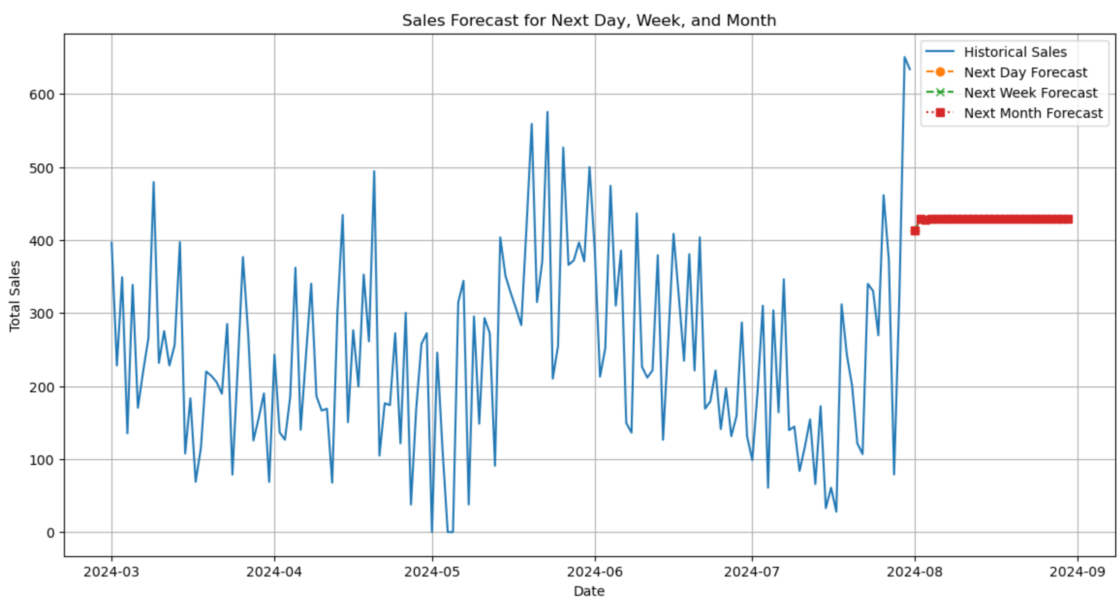
[57]:

```
2024-08-01    413.709083
2024-08-02    429.494824
2024-08-03    428.362817
2024-08-04    428.443994
2024-08-05    428.438173
2024-08-06    428.438590
2024-08-07    428.438560
Freq: D, Name: predicted_mean, dtype: float64
```

[56]:

```python
# Extend ARIMA forecasting to predict next day, week, and month sales

# Forecast the next day (1 step ahead)
next_day_forecast = arima_fit.forecast(steps=1)

# Forecast the next week (7 days ahead)
next_week_forecast = arima_fit.forecast(steps=7)
```

```python
# Forecast the next month (30 days ahead)
next_month_forecast = arima_fit.forecast(steps=30)

# Plot the historical sales data along with the forecasts
plt.figure(figsize=(14, 7))
plt.plot(daily_sales, label='Historical Sales')
plt.plot(next_day_forecast.index, next_day_forecast, label='Next Day Forecast',
 ↪marker='o', linestyle='--')
plt.plot(next_week_forecast.index, next_week_forecast, label='Next Week
 ↪Forecast', marker='x', linestyle='--')
plt.plot(next_month_forecast.index, next_month_forecast, label='Next Month
 ↪Forecast', linestyle=':', marker='s')
plt.title('Sales Forecast for Next Day, Week, and Month')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True)
plt.show()

# Display forecasted values
next_day_forecast[0], next_week_forecast, next_month_forecast
```



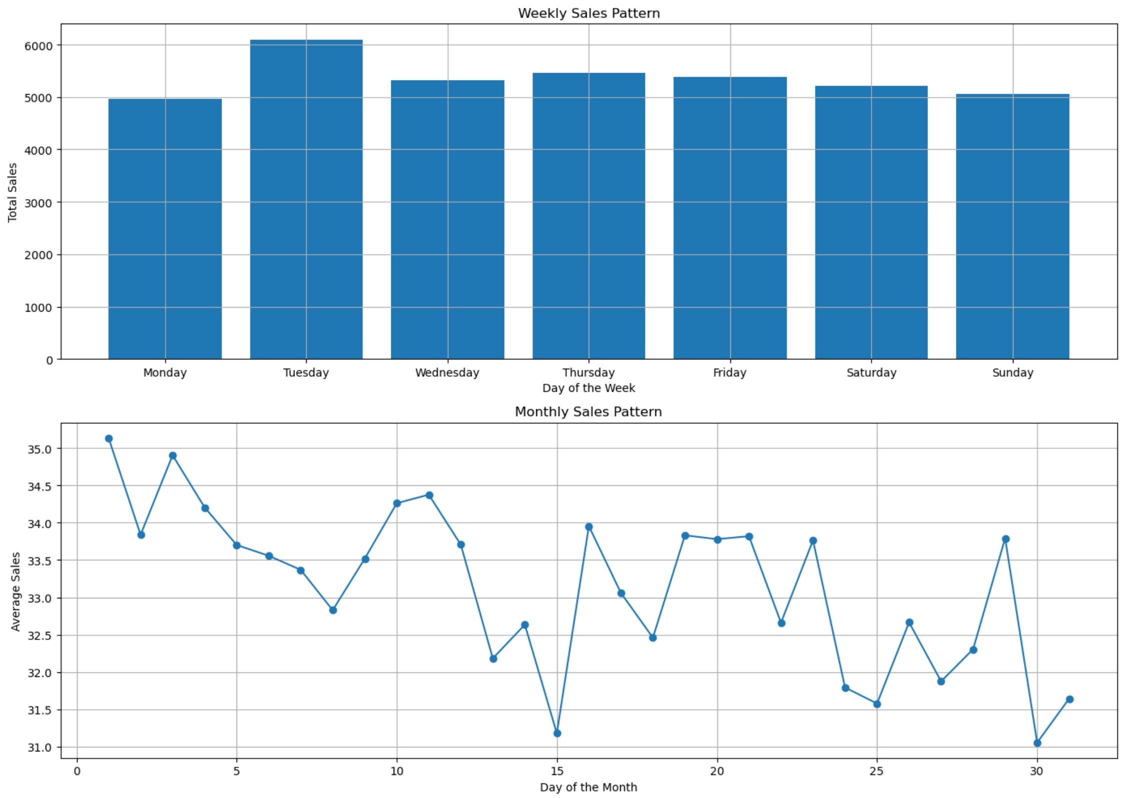

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_9440\3078544631.py:26:
FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In a
future version, integer keys will always be treated as labels (consistent with
DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  next_day_forecast[0], next_week_forecast, next_month_forecast
```

[56]: (413.7090825714696,
```
 2024-08-01    413.709083
 2024-08-02    429.494824
 2024-08-03    428.362817
 2024-08-04    428.443994
 2024-08-05    428.438173
 2024-08-06    428.438590
 2024-08-07    428.438560
 Freq: D, Name: predicted_mean, dtype: float64,
 2024-08-01    413.709083
 2024-08-02    429.494824
 2024-08-03    428.362817
 2024-08-04    428.443994
 2024-08-05    428.438173
 2024-08-06    428.438590
 2024-08-07    428.438560
 2024-08-08    428.438563
 2024-08-09    428.438562
 2024-08-10    428.438562
 2024-08-11    428.438562
 2024-08-12    428.438562
 2024-08-13    428.438562
 2024-08-14    428.438562
 2024-08-15    428.438562
 2024-08-16    428.438562
 2024-08-17    428.438562
 2024-08-18    428.438562
 2024-08-19    428.438562
 2024-08-20    428.438562
 2024-08-21    428.438562
 2024-08-22    428.438562
 2024-08-23    428.438562
 2024-08-24    428.438562
 2024-08-25    428.438562
 2024-08-26    428.438562
 2024-08-27    428.438562
 2024-08-28    428.438562
 2024-08-29    428.438562
 2024-08-30    428.438562
 Freq: D, Name: predicted_mean, dtype: float64)
```

7 Seasonal Sales Pattern

[52]:
```python
# 1. Weekly Seasonality Analysis
# Aggregate sales by day of the week
coffee_sales_data['day_of_week'] = coffee_sales_data.index.day_name()
weekly_sales = coffee_sales_data.groupby('day_of_week')['money'].sum()
```

```
# Order days of the week for visualization
ordered_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
↪'Saturday', 'Sunday']
weekly_sales = weekly_sales.reindex(ordered_days)

# 2. Monthly Seasonality Analysis
# Aggregate sales by day of the month
coffee_sales_data['day_of_month'] = coffee_sales_data.index.day
monthly_sales = coffee_sales_data.groupby('day_of_month')['money'].mean()

# 3. Visualizations for Seasonal Patterns
fig, ax = plt.subplots(2, 1, figsize=(14, 10))

# Plot weekly sales pattern
ax[0].bar(weekly_sales.index, weekly_sales)
ax[0].set_title('Weekly Sales Pattern')
ax[0].set_xlabel('Day of the Week')
ax[0].set_ylabel('Total Sales')
ax[0].grid(True)

# Plot monthly sales pattern
ax[1].plot(monthly_sales.index, monthly_sales, marker='o')
ax[1].set_title('Monthly Sales Pattern')
ax[1].set_xlabel('Day of the Month')
ax[1].set_ylabel('Average Sales')
ax[1].grid(True)

plt.tight_layout()
plt.show()

# Display the weekly and monthly sales data
weekly_sales, monthly_sales
```



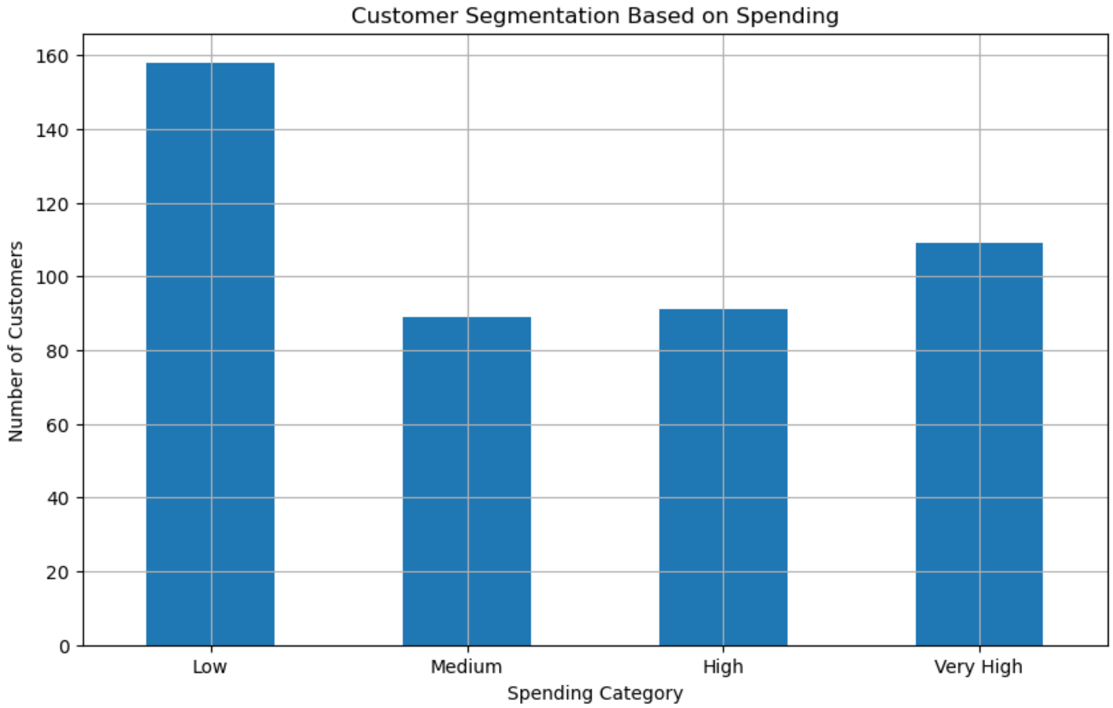

```
[52]: (day_of_week
       Monday       4969.68
       Tuesday      6092.48
       Wednesday    5327.20
       Thursday     5466.74
       Friday       5386.32
       Saturday     5216.26
       Sunday       5050.20
       Name: money, dtype: float64,
       day_of_month
       1     35.136250
       2     33.838000
       3     34.903636
       4     34.204800
       5     33.702051
       6     33.557143
       7     33.369231
       8     32.828571
       9     33.518261
       10    34.260800
       11    34.375484
       12    33.706429
```

```
13    32.182353
14    32.634894
15    31.179310
16    33.954595
17    33.060000
18    32.460000
19    33.831556
20    33.779600
21    33.819412
22    32.660000
23    33.763913
24    31.793143
25    31.577500
26    32.665263
27    31.875135
28    32.305714
29    33.791429
30    31.054615
31    31.644737
Name: money, dtype: float64)
```

8 Customer Segmentation

[87]:
```python
import numpy as np

#Calculate total spending for each customer
customer_spending = coffee_sales_data.groupby('card')['money'].sum()

#Segment customers into quartiles based on total spending
# Define quantile-based segmentation: Low, Medium, High, Very High spenders
spending_labels = ['Low', 'Medium', 'High', 'Very High']
customer_spending_segments = pd.qcut(customer_spending, q=4,␣
↪labels=spending_labels)

#Count the number of customers in each segment
spending_segment_counts = customer_spending_segments.value_counts().sort_index()

#Visualize the customer segments
plt.figure(figsize=(10, 6))
spending_segment_counts.plot(kind='bar')
plt.title('Customer Segmentation Based on Spending')
plt.xlabel('Spending Category')
plt.ylabel('Number of Customers')
plt.xticks(rotation=0)
plt.grid(True)
plt.show()
```

```
# Display the segmentation information
spending_segment_counts
```

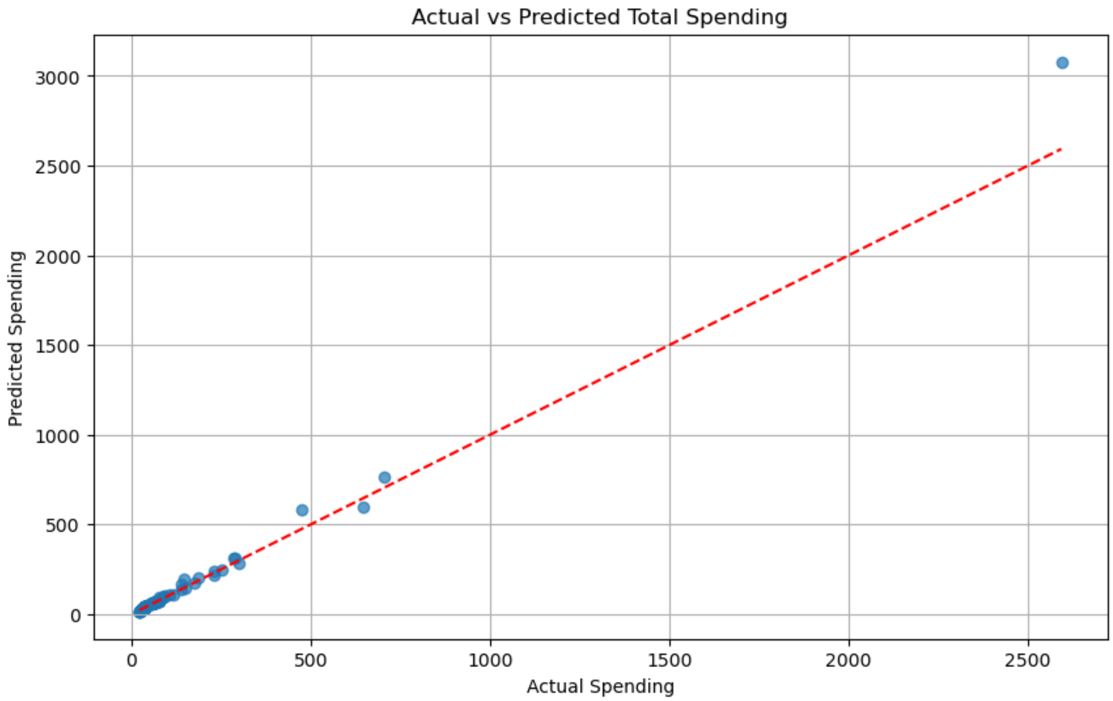


[87]:
```
money
Low          158
Medium        89
High          91
Very High    109
Name: count, dtype: int64
```

9 Predictive Analysis of Spenders

[91]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


# 1. Prepare the data for predictive analysis
# Feature engineering: Total spending, average spending per transaction, and␣
 ↪purchase frequency
customer_features = coffee_sales_data.groupby('card').agg(
    total_spending=('money', 'sum'),
```

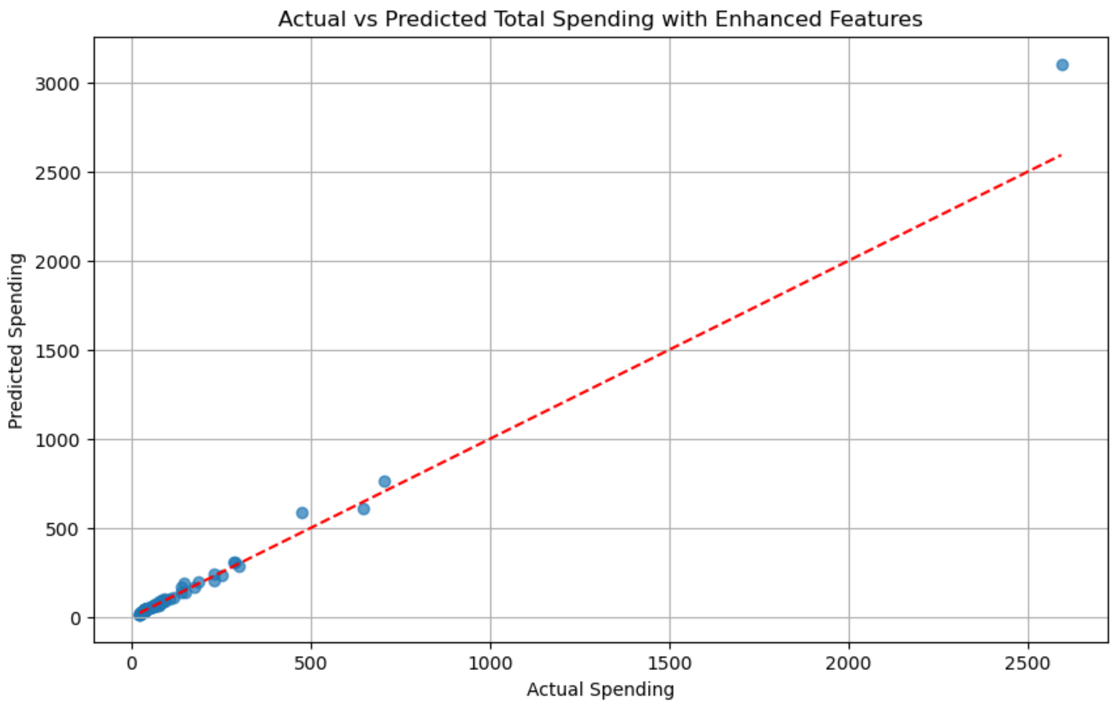
```
    avg_spending=('money', 'mean'),
    purchase_frequency=('money', 'count')
).reset_index()

# 2. Define the target and features
X = customer_features[['avg_spending', 'purchase_frequency']]
y = customer_features['total_spending']

# 3. Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# 4. Train a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# 5. Make predictions on the test set
y_pred = model.predict(X_test)

# 6. Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# 7. Visualize actual vs predicted total spending
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.title('Actual vs Predicted Total Spending')
plt.xlabel('Actual Spending')
plt.ylabel('Predicted Spending')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()],␣
 ↪color='red', linestyle='--')
plt.grid(True)
plt.show()

# Display model evaluation metrics
mse, r2
```



[91]: (1923.8154997029321, 0.96576529483049)

[95]:
```
# 1. Enhance features with additional customer metrics
customer_features_enhanced = coffee_sales_data.groupby('card').agg(
    total_spending=('money', 'sum'),
    avg_spending=('money', 'mean'),
    purchase_frequency=('money', 'count'),
    distinct_days=('datetime', lambda x: x.dt.date.nunique()),
    unique_coffees=('coffee_name', 'nunique'),
    recency=('datetime', lambda x: (coffee_sales_data.index.max() - x.max()).
  ↪days)
).reset_index()

# 2. Define the target and new enhanced features
X_enhanced = customer_features_enhanced[['avg_spending', 'purchase_frequency',␣
  ↪'distinct_days', 'unique_coffees', 'recency']]
y_enhanced = customer_features_enhanced['total_spending']

# 3. Split the enhanced data into training and testing sets
X_train_enhanced, X_test_enhanced, y_train_enhanced, y_test_enhanced =␣
  ↪train_test_split(X_enhanced, y_enhanced, test_size=0.3, random_state=42)

# 4. Train a linear regression model with enhanced features
model_enhanced = LinearRegression()
```

```
model_enhanced.fit(X_train_enhanced, y_train_enhanced)

# 5. Make predictions on the enhanced test set
y_pred_enhanced = model_enhanced.predict(X_test_enhanced)

# 6. Evaluate the enhanced model
mse_enhanced = mean_squared_error(y_test_enhanced, y_pred_enhanced)
r2_enhanced = r2_score(y_test_enhanced, y_pred_enhanced)

# 7. Visualize actual vs predicted total spending with enhanced features
plt.figure(figsize=(10, 6))
plt.scatter(y_test_enhanced, y_pred_enhanced, alpha=0.7)
plt.title('Actual vs Predicted Total Spending with Enhanced Features')
plt.xlabel('Actual Spending')
plt.ylabel('Predicted Spending')
plt.plot([y_test_enhanced.min(), y_test_enhanced.max()], [y_test_enhanced.
 ↪min(), y_test_enhanced.max()], color='red', linestyle='--')
plt.grid(True)
plt.show()

# Display enhanced model evaluation metrics
mse_enhanced, r2_enhanced
```



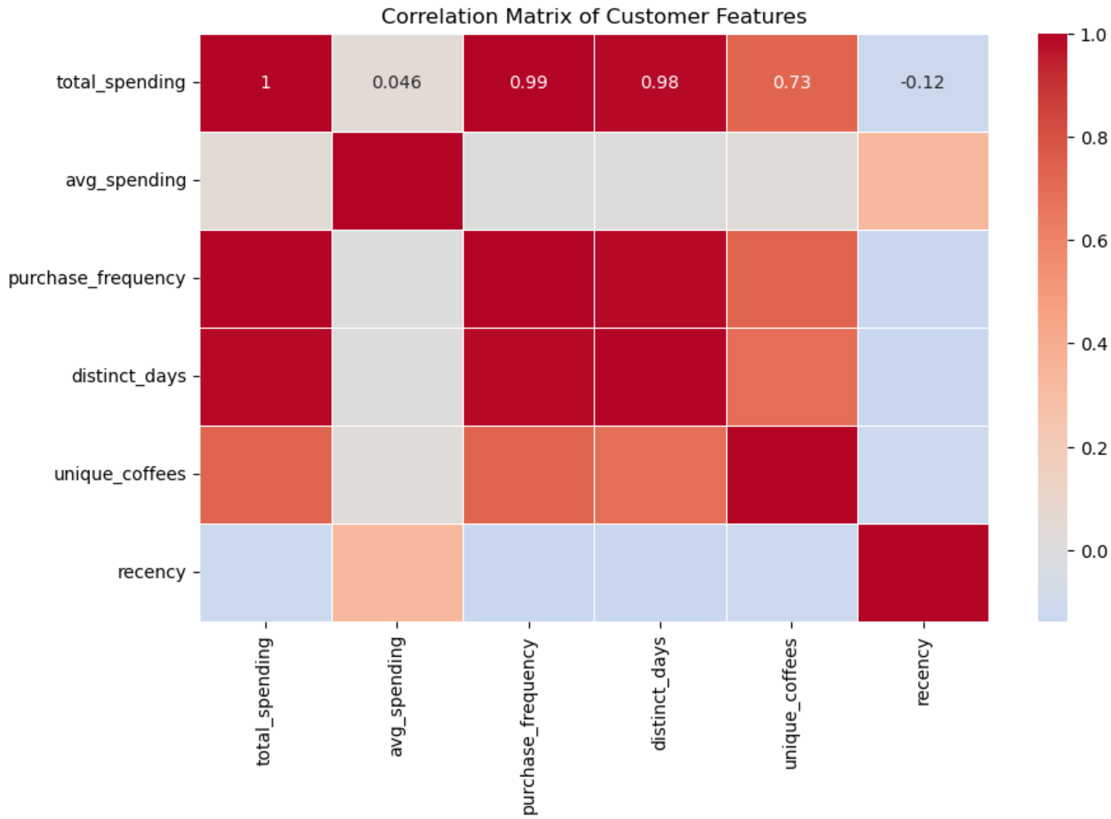
[95]: (2103.871985214473, 0.9625611514517213)

[113]:

```
# Step 1: Ensure 'datetime' column is properly converted to datetime
coffee_sales_data['datetime'] = pd.to_datetime(coffee_sales_data['datetime'],␣
↪errors='coerce')

# Step 2: Check for any missing values in 'datetime' and handle them
missing_datetime_count = coffee_sales_data['datetime'].isnull().sum()
print(f"Missing 'datetime' values: {missing_datetime_count}")

# Remove rows with missing 'datetime'
data_cleaned = coffee_sales_data.dropna(subset=['datetime'])

# Step 3: Basic check for unique 'card' values to ensure consistency
unique_cards_count = data_cleaned['card'].nunique()
print(f"Unique 'card' values: {unique_cards_count}")

# Step 4: Basic calculation for 'total_spending' to verify data integrity
total_spending_check = data_cleaned.groupby('card')['money'].sum()
print(f"Total Spending Sample:\n{total_spending_check.head()}")

# Step 5: Recreate 'customer_features_enhanced' step-by-step
# Calculate distinct_days safely
customer_features_step1 = data_cleaned.groupby('card').agg(
    total_spending=('money', 'sum'),
    avg_spending=('money', 'mean'),
    purchase_frequency=('money', 'count'),
    distinct_days=('datetime', lambda x: pd.to_datetime(x).dt.floor('D').
↪nunique())
).reset_index()

# Debug: Check if 'distinct_days' is correctly calculated
print(f"Distinct Days Sample:\n{customer_features_step1[['card',␣
↪'distinct_days']].head()}")

# Add 'unique_coffees' feature
customer_features_step2 = data_cleaned.groupby('card').agg(
    unique_coffees=('coffee_name', 'nunique')
).reset_index()

# Merge the two intermediate DataFrames
customer_features_merged = pd.merge(customer_features_step1,␣
↪customer_features_step2, on='card')

# Calculate 'recency' separately and merge it
recency = data_cleaned.groupby('card').agg(
    recency=('datetime', lambda x: (pd.to_datetime(data_cleaned['datetime'].
↪max()) - x.max()).days)
).reset_index()
```

```
# Merge the recency information
customer_features_enhanced = pd.merge(customer_features_merged, recency,␣
↪on='card')

# Step 6: Perform correlation analysis
correlation_features = customer_features_enhanced[['total_spending',␣
↪'avg_spending', 'purchase_frequency',
                                                    'distinct_days',␣
↪'unique_coffees', 'recency']]

# Calculate the correlation matrix
correlation_matrix = correlation_features.corr()

# Visualize the correlation matrix using a heatmap
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0,␣
↪linewidths=0.5)
plt.title('Correlation Matrix of Customer Features')
plt.show()

# Display the correlation matrix
correlation_matrix
```

```
Missing 'datetime' values: 0
Unique 'card' values: 447
Total Spending Sample:
card
ANON-0000-0000-0001    646.14
ANON-0000-0000-0002     77.40
ANON-0000-0000-0003    651.96
ANON-0000-0000-0004    289.50
ANON-0000-0000-0005     33.80
Name: money, dtype: float64
Distinct Days Sample:
                  card  distinct_days
0  ANON-0000-0000-0001             17
1  ANON-0000-0000-0002              1
2  ANON-0000-0000-0003             13
3  ANON-0000-0000-0004              6
4  ANON-0000-0000-0005              1
```



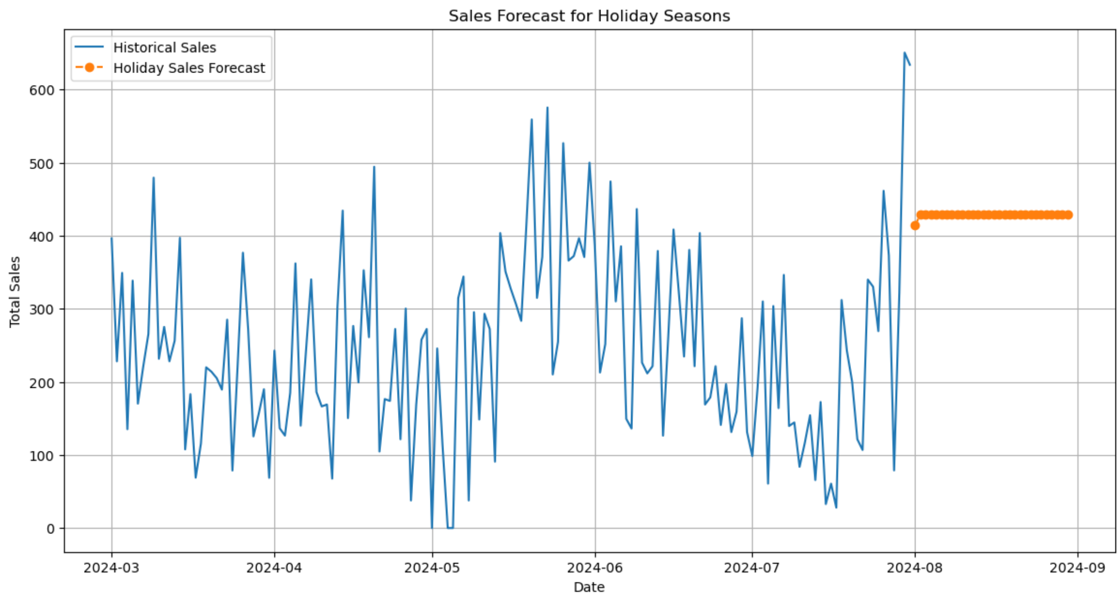
[113]:

```
                    total_spending  avg_spending  purchase_frequency  \
total_spending            1.000000      0.045674            0.994912
avg_spending              0.045674      1.000000            0.007374
purchase_frequency        0.994912      0.007374            1.000000
distinct_days             0.984794      0.011410            0.982740
unique_coffees            0.729213      0.021728            0.727147
recency                  -0.115951      0.325229           -0.131678

                    distinct_days  unique_coffees   recency
total_spending           0.984794        0.729213 -0.115951
avg_spending             0.011410        0.021728  0.325229
purchase_frequency       0.982740        0.727147 -0.131678
distinct_days            1.000000        0.691335 -0.137934
unique_coffees           0.691335        1.000000 -0.109815
recency                 -0.137934       -0.109815  1.000000
```

[117]:

```python
from pandas.tseries.holiday import USFederalHolidayCalendar

# Identify holidays within the dataset's timeframe
calendar = USFederalHolidayCalendar()
```

```python
holidays = calendar.holidays(start=coffee_sales_data.index.min(), end=coffee_sales_data.index.max())

#Create an indicator for holidays in the dataset
coffee_sales_data['is_holiday'] = coffee_sales_data.index.isin(holidays).astype(int)

#Analyze historical sales during holidays
holiday_sales = coffee_sales_data[coffee_sales_data['is_holiday'] == 1]['money'].resample('D').sum()
non_holiday_sales = coffee_sales_data[coffee_sales_data['is_holiday'] == 0]['money'].resample('D').sum()

#Prepare the dataset for time series forecasting with holiday effects
# Use ARIMA with external regressors (holidays)
exog = coffee_sales_data[['is_holiday']].resample('D').max().fillna(0)

# Fit an ARIMA model with holiday indicator as an exogenous variable
arima_model_holiday = ARIMA(daily_sales, order=(1, 1, 1), exog=exog)
arima_fit_holiday = arima_model_holiday.fit()

# Forecast sales for the next 30 days, assuming some days will be holidays
future_exog = pd.DataFrame({'is_holiday': [1 if date in holidays else 0 for date in pd.date_range(start=daily_sales.index.max(), periods=30)]})
holiday_sales_forecast = arima_fit_holiday.forecast(steps=30, exog=future_exog)

# 5. Visualize the historical and forecasted sales for holidays
plt.figure(figsize=(14, 7))
plt.plot(daily_sales, label='Historical Sales')
plt.plot(holiday_sales_forecast.index, holiday_sales_forecast, label='Holiday Sales Forecast', linestyle='--', marker='o')
plt.title('Sales Forecast for Holiday Seasons')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.legend()
plt.grid(True)
plt.show()

# Display forecasted holiday sales values
holiday_sales_forecast
```



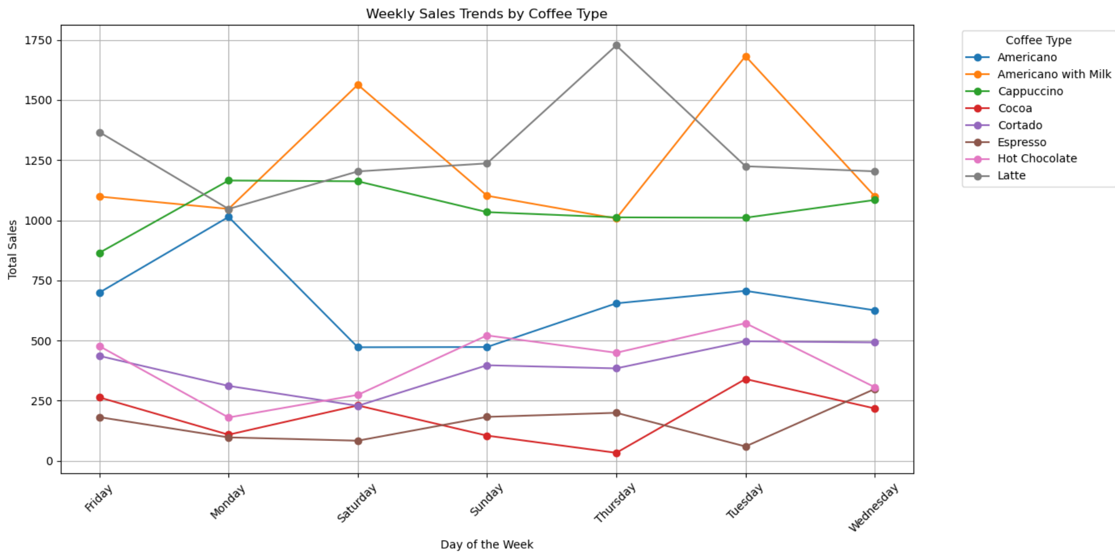

```
[117]: 2024-08-01    413.942151
       2024-08-02    429.573437
       2024-08-03    428.462298
       2024-08-04    428.541282
       2024-08-05    428.535668
       2024-08-06    428.536067
       2024-08-07    428.536039
       2024-08-08    428.536041
       2024-08-09    428.536041
       2024-08-10    428.536041
       2024-08-11    428.536041
       2024-08-12    428.536041
       2024-08-13    428.536041
       2024-08-14    428.536041
       2024-08-15    428.536041
       2024-08-16    428.536041
       2024-08-17    428.536041
       2024-08-18    428.536041
       2024-08-19    428.536041
       2024-08-20    428.536041
       2024-08-21    428.536041
       2024-08-22    428.536041
       2024-08-23    428.536041
       2024-08-24    428.536041
       2024-08-25    428.536041
       2024-08-26    428.536041
       2024-08-27    428.536041
```

```
2024-08-28    428.536041
2024-08-29    428.536041
2024-08-30    428.536041
Freq: D, Name: predicted_mean, dtype: float64
```

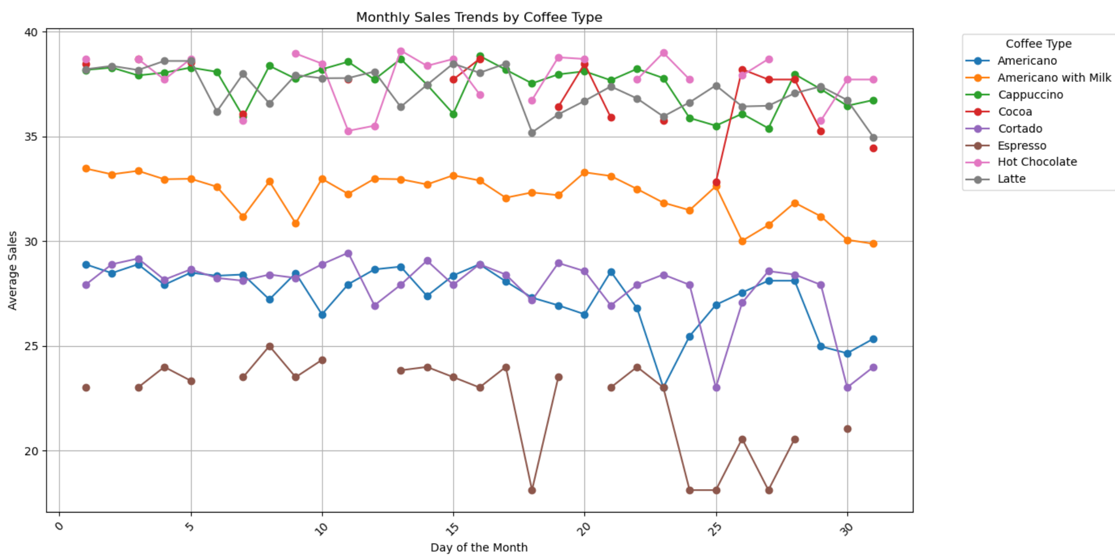
10 Seasonal Product Trends

[119]:
```python
# 1. Weekly Product Trends
# Group sales by day of the week and coffee name
weekly_product_sales = coffee_sales_data.groupby(['day_of_week',␣
↪'coffee_name'])['money'].sum().unstack()

# 2. Monthly Product Trends
# Group sales by day of the month and coffee name
monthly_product_sales = coffee_sales_data.groupby(['day_of_month',␣
↪'coffee_name'])['money'].mean().unstack()

# 3. Visualize the weekly product trends
plt.figure(figsize=(14, 7))
weekly_product_sales.plot(kind='line', marker='o', figsize=(14, 7))
plt.title('Weekly Sales Trends by Coffee Type')
plt.xlabel('Day of the Week')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend(title='Coffee Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# 4. Visualize the monthly product trends
plt.figure(figsize=(14, 7))
monthly_product_sales.plot(kind='line', marker='o', figsize=(14, 7))
plt.title('Monthly Sales Trends by Coffee Type')
plt.xlabel('Day of the Month')
plt.ylabel('Average Sales')
plt.xticks(rotation=45)
plt.grid(True)
plt.legend(title='Coffee Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Display the first few rows of weekly and monthly sales data
weekly_product_sales.head(), monthly_product_sales.head()
```

```
<Figure size 1400x700 with 0 Axes>
```



<Figure size 1400x700 with 0 Axes>



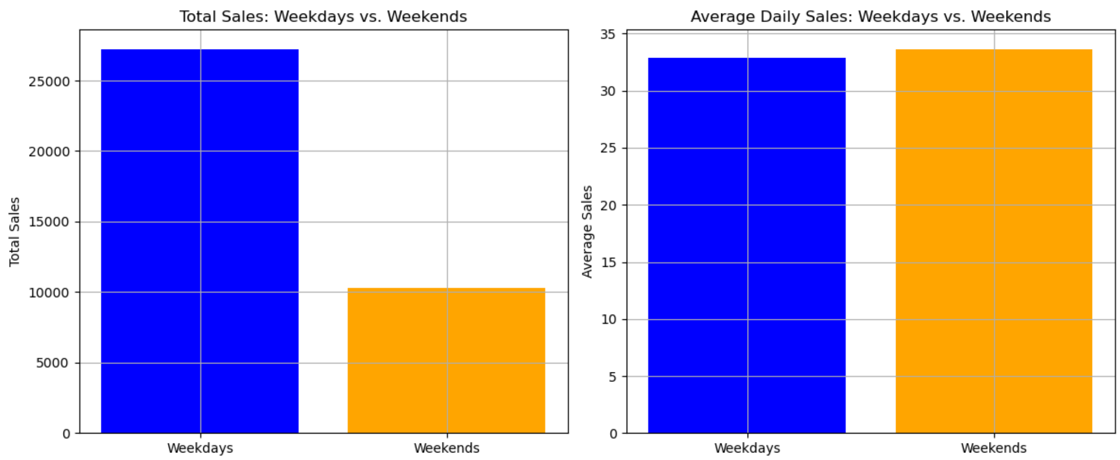
[119]:
```
(coffee_name  Americano  Americano with Milk  Cappuccino   Cocoa  Cortado  \
 day_of_week
 Friday          699.10              1098.64      864.60  263.38   435.94
 Monday         1013.96              1047.16     1165.36  108.26   311.16
 Saturday        471.90              1563.82     1162.10  229.86   228.26
 Sunday          472.92              1102.04     1033.92  104.34   397.00
 Thursday        654.52              1007.62     1012.20   32.82   384.02
```

```
coffee_name  Espresso  Hot Chocolate    Latte
day_of_week
Friday         181.22         476.64  1366.80
Monday          97.02         179.78  1046.98
Saturday        83.26         273.82  1203.24
Sunday         182.20         521.22  1236.56
Thursday       199.34         449.02  1727.20  ,
coffee_name   Americano  Americano with Milk  Cappuccino      Cocoa  Cortado  \
day_of_month
1             28.900000            33.473333   38.176000  38.473333   27.920
2             28.480000            33.194545   38.290000        NaN   28.900
3             28.900000            33.360000   37.916000  38.700000   29.175
4             27.920000            32.960000   38.046667        NaN   28.165
5             28.497143            32.983333   38.290000  38.535000   28.655

coffee_name    Espresso  Hot Chocolate      Latte
day_of_month
1             23.020000          38.70  38.210000
2                   NaN            NaN  38.368571
3             23.020000          38.70  38.174000
4             24.000000          37.72  38.605714
5             23.346667          38.70  38.605714  )
```

[121]:

```python
# 1. Categorize sales into weekdays and weekends
coffee_sales_data['is_weekend'] = coffee_sales_data['day_of_week'].
 ↪isin(['Saturday', 'Sunday'])

# 2. Aggregate sales for weekdays and weekends
weekday_sales = coffee_sales_data[coffee_sales_data['is_weekend'] ==␣
 ↪False]['money'].sum()
weekend_sales = coffee_sales_data[coffee_sales_data['is_weekend'] ==␣
 ↪True]['money'].sum()

# Calculate average daily sales for weekdays and weekends
average_weekday_sales = coffee_sales_data[coffee_sales_data['is_weekend'] ==␣
 ↪False]['money'].mean()
average_weekend_sales = coffee_sales_data[coffee_sales_data['is_weekend'] ==␣
 ↪True]['money'].mean()

# 3. Visualize the comparison
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Total sales bar chart
ax[0].bar(['Weekdays', 'Weekends'], [weekday_sales, weekend_sales],␣
 ↪color=['blue', 'orange'])
ax[0].set_title('Total Sales: Weekdays vs. Weekends')
ax[0].set_ylabel('Total Sales')
```

```
ax[0].grid(True)

# Average sales bar chart
ax[1].bar(['Weekdays', 'Weekends'], [average_weekday_sales,␣
↪average_weekend_sales], color=['blue', 'orange'])
ax[1].set_title('Average Daily Sales: Weekdays vs. Weekends')
ax[1].set_ylabel('Average Sales')
ax[1].grid(True)

plt.tight_layout()
plt.show()

# Display sales figures
weekday_sales, weekend_sales, average_weekday_sales, average_weekend_sales
```



[121]: (27242.42, 10266.46, 32.901473429951686, 33.66052459016393)

[ ]:
```
# 3. Review the impact of payment method (Cash vs. Card)
# Aggregate sales by payment method
payment_method_sales = coffee_sales_data.groupby('cash_type')['money'].sum()
average_payment_method_sales = coffee_sales_data.groupby('cash_type')['money'].
↪mean()

# Visualize the impact of payment method
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

# Total sales by payment method
ax[0].bar(payment_method_sales.index, payment_method_sales, color=['green',␣
↪'blue'])
ax[0].set_title('Total Sales by Payment Method')
ax[0].set_ylabel('Total Sales')
```

```
ax[0].grid(True)

# Average sales by payment method
ax[1].bar(average_payment_method_sales.index, average_payment_method_sales,␣
 ↪color=['green', 'blue'])
ax[1].set_title('Average Sales by Payment Method')
ax[1].set_ylabel('Average Sales')
ax[1].grid(True)

plt.tight_layout()
plt.show()

# Display summary of outliers and payment method impact
outliers_summary = outliers.describe()
payment_method_sales, average_payment_method_sales, outliers_summary
```