Coffe_Sales_UM

December 9, 2024

1 Importing Libraries and Data

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
url = "https://drive.google.com/uc?id=1QPlFQU8Z-H2Joq6NYe_2HAOpNc1B9VDl"
df = pd.read_csv(url, encoding='utf-8')
df.head()
```

[3]:

	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

2 Data Summarization

[ ]:
```
df.dtypes
```

[ ]:
```
date            object
datetime        object
cash_type       object
card            object
```

```
money          float64
coffee_name     object
dtype: object
```

[ ]:
```
df['datetime'] = pd.to_datetime(df['datetime'])
```

[ ]:
```
df.dtypes
```

[ ]:
```
date                    object
datetime        datetime64[ns]
cash_type               object
card                    object
money                  float64
coffee_name             object
dtype: object
```

[ ]:
```
df.head(3)
```

[ ]:
```
         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

     coffee_name
0          Latte
1  Hot Chocolate
2  Hot Chocolate
```

[ ]:
```
df.drop('date', axis=1, inplace=True)
```

[ ]:
```
df['coffee_name'].value_counts()
```

[ ]:
```
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
```

[ ]:
```
df['date'] = df['datetime'].dt.date
df['hour'] = df['datetime'].dt.hour
df['day'] = df['datetime'].dt.day
df['month_num'] = df['datetime'].dt.month
```

```
df['year'] = df['datetime'].dt.year
df['weekday'] = df['datetime'].dt.day_name()
df['month_name'] = df['datetime'].dt.month_name()
df['year_month'] = df['datetime'].dt.strftime('%Y-%m')
```

[ ]:
```
df.head(3)
```

[ ]:
```
                 datetime cash_type                 card  money  \
0 2024-03-01 10:15:50.520      card  ANON-0000-0000-0001   38.7
1 2024-03-01 12:19:22.539      card  ANON-0000-0000-0002   38.7
2 2024-03-01 12:20:18.089      card  ANON-0000-0000-0002   38.7

     coffee_name        date  hour  day  month_num  year weekday month_name  \
0          Latte  2024-03-01    10    1          3  2024  Friday      March
1  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March
2  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March

  year_month
0    2024-03
1    2024-03
2    2024-03
```

[ ]:
```
df['date'].min()
```

[ ]:
```
datetime.date(2024, 3, 1)
```

[ ]:
```
df['date'].max()
```

[ ]:
```
datetime.date(2024, 7, 31)
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
datetime        0
cash_type       0
card           89
money           0
coffee_name     0
date            0
hour            0
day             0
month_num       0
year            0
weekday         0
month_name      0
year_month      0
dtype: int64
```

[ ]:
```
df.duplicated().sum()
```

[ ]: 0

[ ]:
```
df.shape
```

[ ]: (1133, 13)

[ ]:
```
df.describe()
```

[ ]:
```
                           datetime        money         hour           day  \
count                          1133  1133.000000  1133.000000  1133.000000
mean   2024-05-20 02:38:39.053382912    33.105808    14.552515    16.793469
min       2024-03-01 10:15:50.520000    18.120000     7.000000     1.000000
25%    2024-04-14 10:55:27.406000128    28.900000    11.000000     9.000000
50%    2024-05-23 12:22:06.604999936    32.820000    14.000000    18.000000
75%    2024-06-22 08:39:50.272999936    37.720000    18.000000    25.000000
max       2024-07-31 21:55:16.570000    40.000000    22.000000    31.000000
std                              NaN     5.035366     4.084588     8.921907

         month_num    year
count  1133.000000  1133.0
mean      5.082083  2024.0
min       3.000000  2024.0
25%       4.000000  2024.0
50%       5.000000  2024.0
75%       6.000000  2024.0
max       7.000000  2024.0
std       1.390073     0.0
```

[ ]:
```
df['money'].max()
```

[ ]: 40.0

[ ]:
```
df['coffee_name'].value_counts()
```

[ ]:
```
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
```

Americano with Milk and Latte has Higher sales than every other product

[ ]:
```
df[df['money'] == df.money.max()][['coffee_name', 'money']].value_counts()
```

[ ]:
```
coffee_name    money
Latte          40.0     16
Cappuccino     40.0     10
Hot Chocolate  40.0      5
Cocoa          40.0      1
Name: count, dtype: int64
```

[ ]:
```
df[df['money'] == df.money.min()][['coffee_name', 'money']].value_counts()
```

[ ]:
```
coffee_name  money
Espresso     18.12    10
Name: count, dtype: int64
```

Expresso is the Cheapest coffee product

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1133 entries, 0 to 1132
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   datetime     1133 non-null   datetime64[ns]
 1   cash_type    1133 non-null   object
 2   card         1044 non-null   object
 3   money        1133 non-null   float64
 4   coffee_name  1133 non-null   object
 5   date         1133 non-null   object
 6   hour         1133 non-null   int32
 7   day          1133 non-null   int32
 8   month_num    1133 non-null   int32
 9   year         1133 non-null   int32
 10  weekday      1133 non-null   object
 11  month_name   1133 non-null   object
 12  year_month   1133 non-null   object
dtypes: datetime64[ns](1), float64(1), int32(4), object(7)
memory usage: 97.5+ KB
```

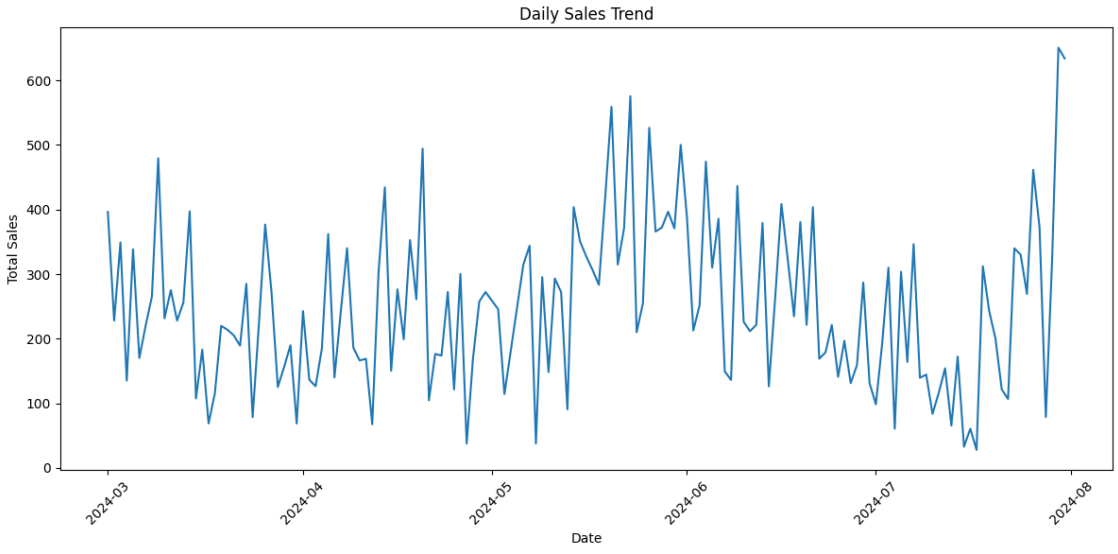
[ ]:
```
df.head(3)
```

[ ]:
```
                 datetime cash_type                 card  money  \
0 2024-03-01 10:15:50.520      card  ANON-0000-0000-0001   38.7
1 2024-03-01 12:19:22.539      card  ANON-0000-0000-0002   38.7
2 2024-03-01 12:20:18.089      card  ANON-0000-0000-0002   38.7

      coffee_name        date  hour  day  month_num  year weekday month_name  \
```

```
0          Latte  2024-03-01    10    1          3  2024  Friday      March
1  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March
2  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March

  year_month
0    2024-03
1    2024-03
2    2024-03
```

3 EDA - Sales Analysis

[ ]:
```
# Aggregate sales by date
daily_sales = df.groupby('date')['money'].sum().reset_index()

# Plot the daily sales trend
plt.figure(figsize=(12, 6))
sns.lineplot(data=daily_sales, x='date', y='money')
plt.title('Daily Sales Trend')
plt.xlabel('Date')
plt.ylabel('Total Sales')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



[ ]:
```
daily_sales
```

[ ]:
```
            date   money
0     2024-03-01  396.30
1     2024-03-02  228.10
2     2024-03-03  349.10
3     2024-03-04  135.20
4     2024-03-05  338.50
..           …     …
145   2024-07-27  372.76
146   2024-07-28   78.86
147   2024-07-29  321.82
148   2024-07-30  650.48
149   2024-07-31  633.84

[150 rows x 2 columns]
```

[ ]:
```
df.shape
```

[ ]:
```
(1133, 13)
```

[ ]:
```
df.isnull().sum()
```

[ ]:
```
datetime        0
cash_type       0
card           89
money           0
coffee_name     0
date            0
hour            0
day             0
month_num       0
year            0
weekday         0
month_name      0
year_month      0
dtype: int64
```

[ ]:
```
df['card'].fillna(df['card'].mode()[0], inplace=True)
df.head()
```

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

     coffee_name        date  hour  day  month_num  year weekday month_name  \
```

```
0          Latte  2024-03-01    10    1          3  2024  Friday      March
1  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March
2  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March
3      Americano  2024-03-01    13    1          3  2024  Friday      March
4          Latte  2024-03-01    13    1          3  2024  Friday      March

  year_month
0    2024-03
1    2024-03
2    2024-03
3    2024-03
4    2024-03
```

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
0
```

[ ]:
```
df.dtypes
```

[ ]:
```
datetime       datetime64[ns]
cash_type              object
card                   object
money                 float64
coffee_name            object
date                   object
hour                    int32
day                     int32
month_num               int32
year                    int32
weekday                object
month_name             object
year_month             object
dtype: object
```

[ ]:
```
# Removing Outliers with the help of Z-Scores
from scipy.stats import zscore

df = df[(np.abs(zscore(df[['money']])) < 3).all(axis=1)]
```

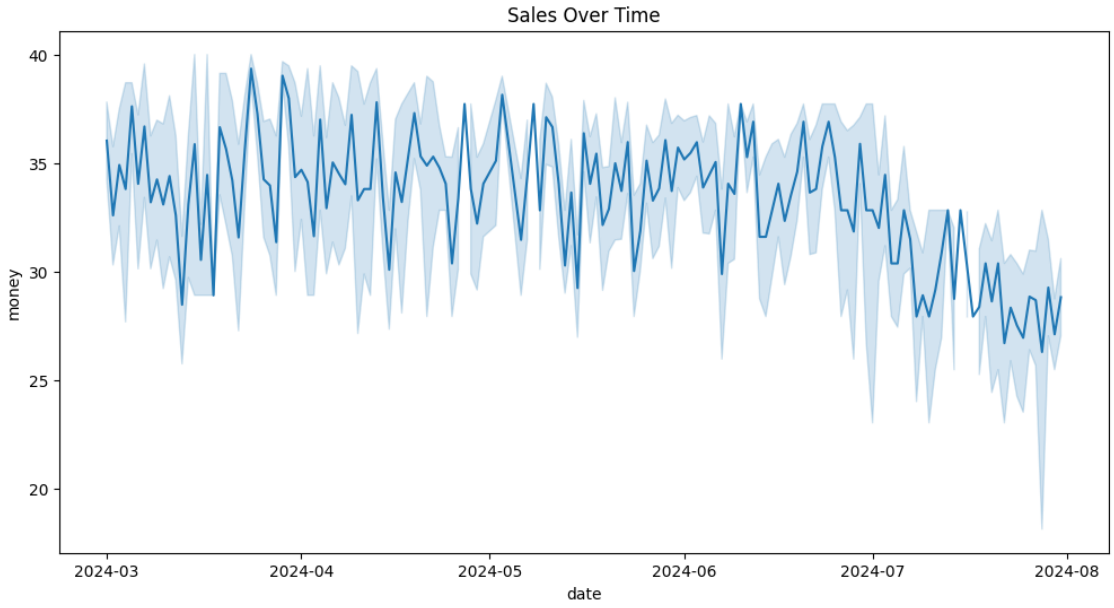
[ ]:
```
np.abs(zscore(df[['money']])) < 3
```

[ ]:
```
      money
0      True
1      True
2      True
3      True
4      True
```

```
…      …
1128   True
1129   True
1130   True
1131   True
1132   True

[1133 rows x 1 columns]
```

3.1 Sales plot

[ ]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(data=df, x='date', y='money')
plt.title('Sales Over Time')
plt.show()
```

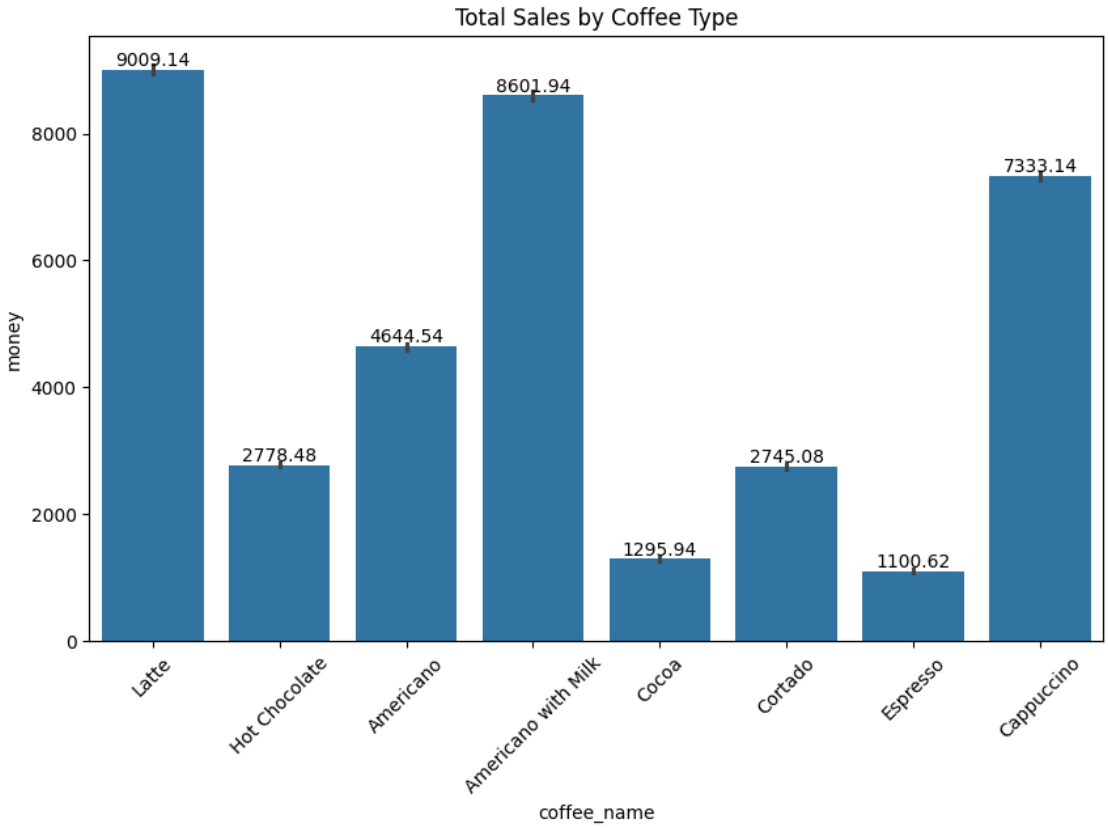


The sales starts to decline from the month of June

3.2 Sales by Coffee Type

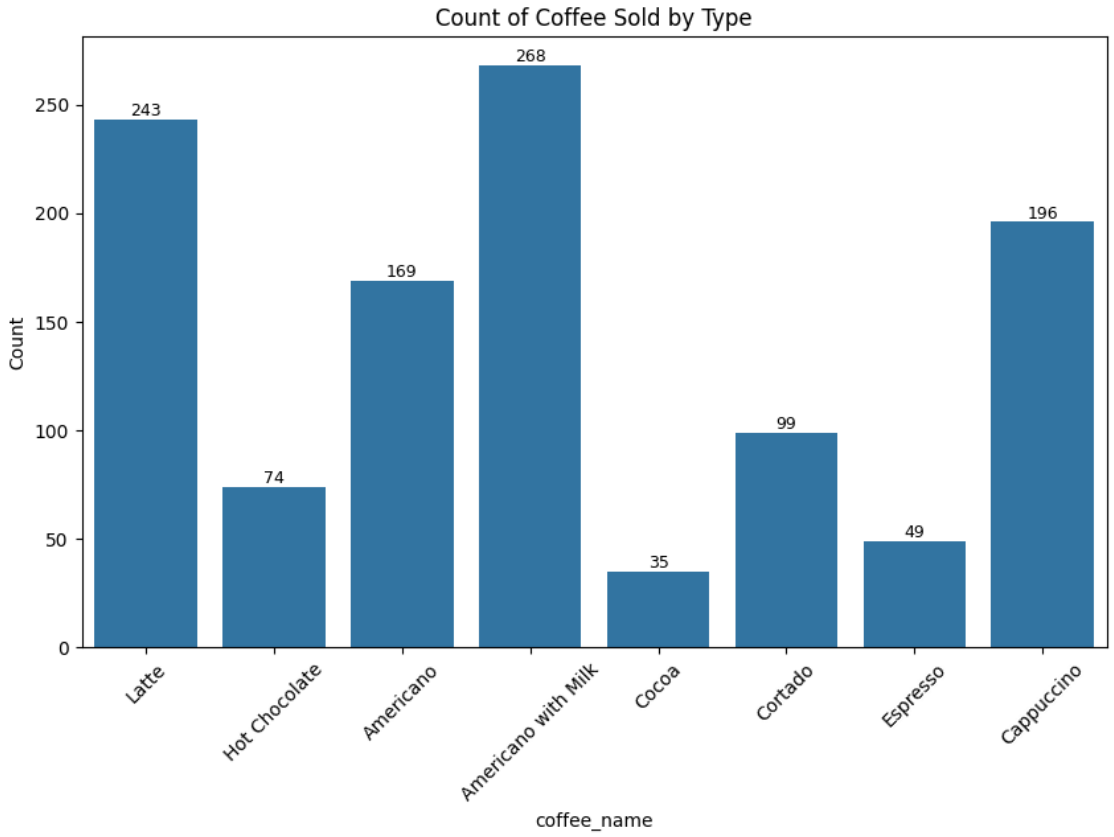
[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming your data is in a DataFrame named 'df'
plt.figure(figsize=(10, 6))
ax = sns.barplot(data=df, x='coffee_name', y='money', estimator=np.sum)  #␣
 ↪Default estimator is 'sum' for total sales
```

```
# # Customize data label formatting (optional)
# for bar in plt.gca().containers[0]:
#     yval = bar.get_height()
#     # Format the label (e.g., "{:.2f}".format(yval) for two decimal places)
#     label = f"{yval:.2f}"  # Example with two decimal places
#     plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.1, label,
#              ha='center', va='bottom', fontsize=12)  # Adjust position and font size

ax.bar_label(ax.containers[0], fontsize=10)

plt.title('Total Sales by Coffee Type')
plt.xticks(rotation=45)
# plt.tight_layout()  # Adjust spacing to avoid clipping data labels
plt.show()
```



Latte and Americano With Milk has booked Higher sales whie Expresso and cocoa has booked minimum sales

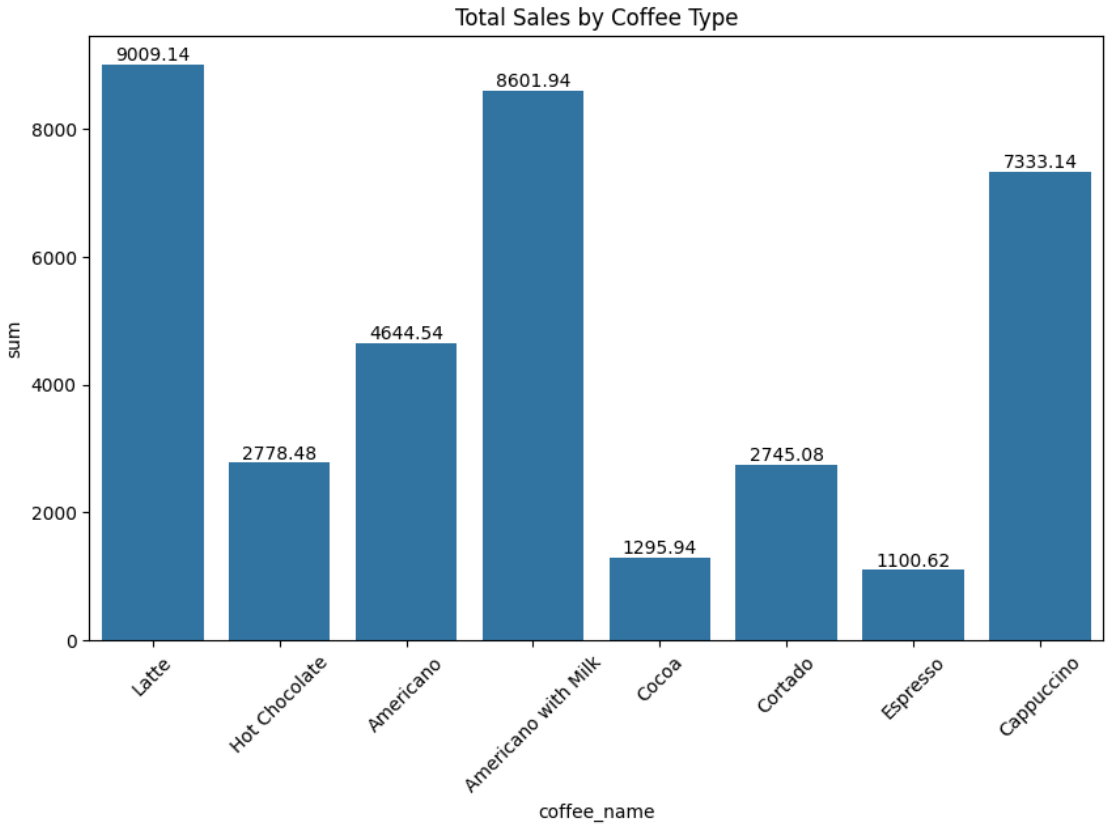
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Create the bar plot
plt.figure(figsize=(10, 6))
ax = sns.barplot(data=df, x='coffee_name', y='money', estimator=len,␣
 ↪errorbar=None)

# # Add data labels to each bar
# for bar in ax.patches:  # Loop through each bar in the plot
#     height = bar.get_height()  # Get the height of the bar
#     ax.text(
#         bar.get_x() + bar.get_width() / 2,  # X-coordinate: center of the bar
#         height,  # Y-coordinate: just above the bar
#         f'{int(height)}',  # Convert the height to an integer and label
#         ha='center', va='bottom', fontsize=10  # Align text and set font size
#     )

ax.bar_label(ax.containers[0], fontsize=9)

# Customize plot appearance
plt.title('Count of Coffee Sold by Type')
plt.xticks(rotation=45)
plt.ylabel('Count')  # Update the y-axis label
plt.show()
```



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

def plot_sales_by_day(df, agg_func=np.sum):
    plt.figure(figsize=(10, 6))
    ax = sns.barplot(data=df, x='coffee_name', y='money', estimator=agg_func,␣
 ↪errorbar=None)
    ax.bar_label(ax.containers[0], fontsize=10)

    # # Customize data label formatting (optional)
    # for bar in plt.gca().containers[0]:
    #     yval = bar.get_height()
    #     # Format the label (e.g., "{:.2f}".format(yval) for two decimal␣
 ↪places)
    #     label = f"{yval:.2f}"  # Example with two decimal places
    #     plt.text(bar.get_x() + bar.get_width() / 2, yval + 0.1, label,
    #              ha='center', va='bottom', fontsize=12)  # Adjust position and␣
 ↪font size
```

```
    plt.title('Total Sales by Coffee Type')
    plt.xticks(rotation=45)
    plt.ylabel('sum')  # Update the y-axis label
    plt.show()

# Mapping string input to the respective NumPy function
agg_func_input = input("Insert aggregation function (e.g., 'sum' or 'mean'): ")

# Map the input to the corresponding NumPy function
if agg_func_input == 'sum':
    agg_func = np.sum
elif agg_func_input == 'mean':
    agg_func = np.mean
elif agg_func_input == 'median':
    agg_func = np.median
elif agg_func_input == 'min':
    agg_func = np.min
elif agg_func_input == 'max':
    agg_func = np.max
elif agg_func_input == 'count':
    agg_func = len
else:
    print("Invalid function name. Using default: np.sum")
    agg_func = np.sum

# Call the function with the selected aggregation function
plot_sales_by_day(df, agg_func=agg_func)
```

```
Insert aggregation function (e.g., 'sum' or 'mean'): sum
```



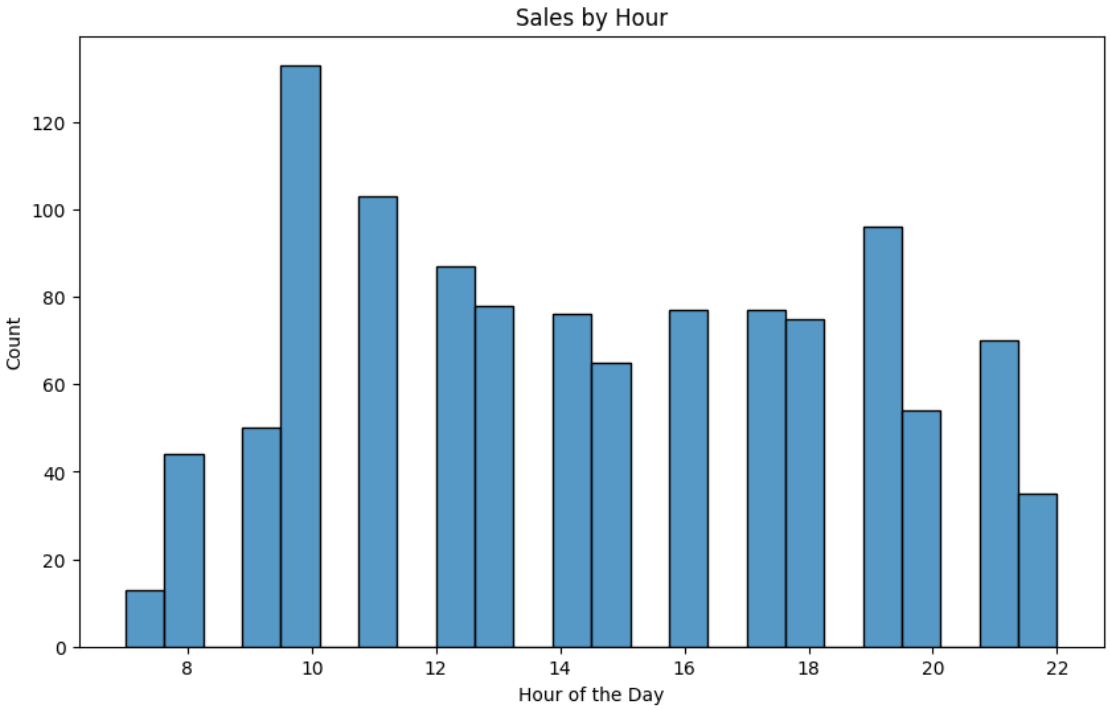
Latte Has maximum sales amount and Espresso has minimum Sales amount sum

[ ]:
```
df[['hour']].max()
```

[ ]:
```
hour    22
dtype: int32
```

3.3 Peak Sales Hours

[ ]:
```
# Peak Sales Hours

plt.figure(figsize=(10, 6))
sns.histplot(df['hour'], bins=24, kde = False)
plt.title('Sales by Hour')
plt.xlabel('Hour of the Day')
plt.show()
```



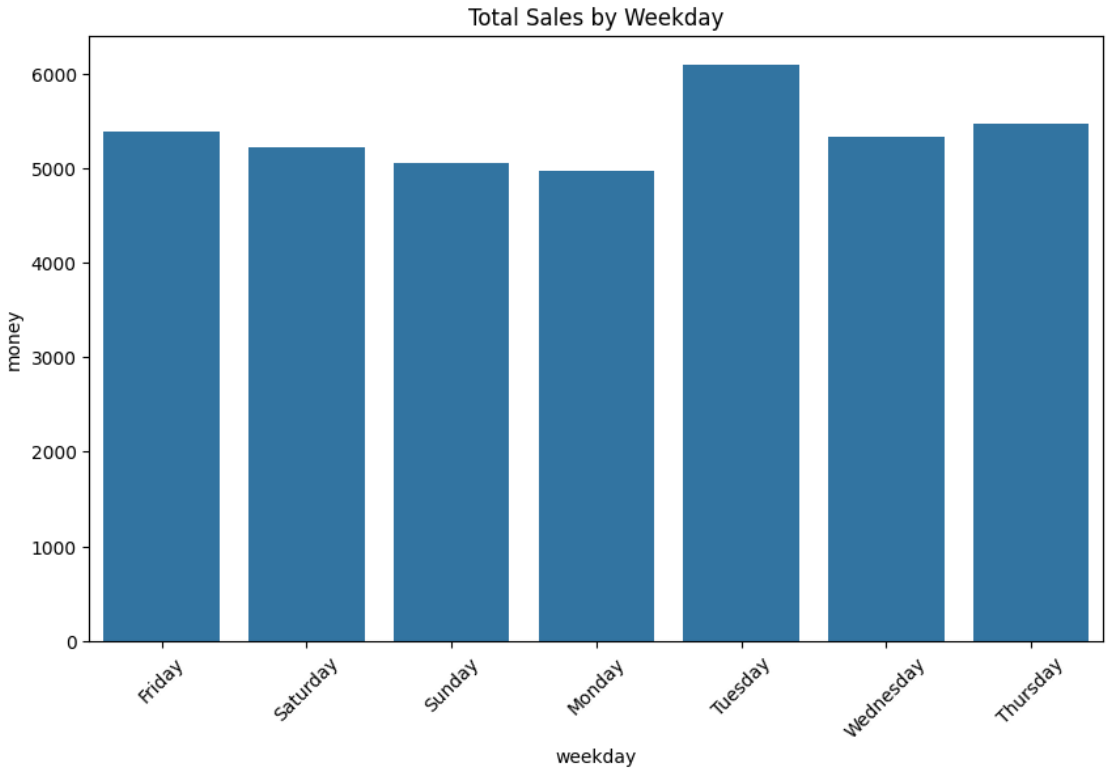
Morning 10'O clock has peak sales and between 12 to 18 sales is steady while morning 8'O clock has less sales

[ ]:
```
df[['weekday', 'coffee_name']]
```

[ ]:
```
        weekday          coffee_name
0        Friday                Latte
1        Friday        Hot Chocolate
2        Friday        Hot Chocolate
3        Friday            Americano
4        Friday                Latte
…          …                   …
1128  Wednesday              Cortado
1129  Wednesday  Americano with Milk
1130  Wednesday                Latte
1131  Wednesday                Latte
1132  Wednesday                Latte

[1133 rows x 2 columns]
```

3.4 sales by Week

[ ]:
```
# Sales by weekday

plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='weekday', y='money', estimator=np.sum, errorbar=None)
plt.title('Total Sales by Weekday')
plt.xticks(rotation=45)
plt.show()
```

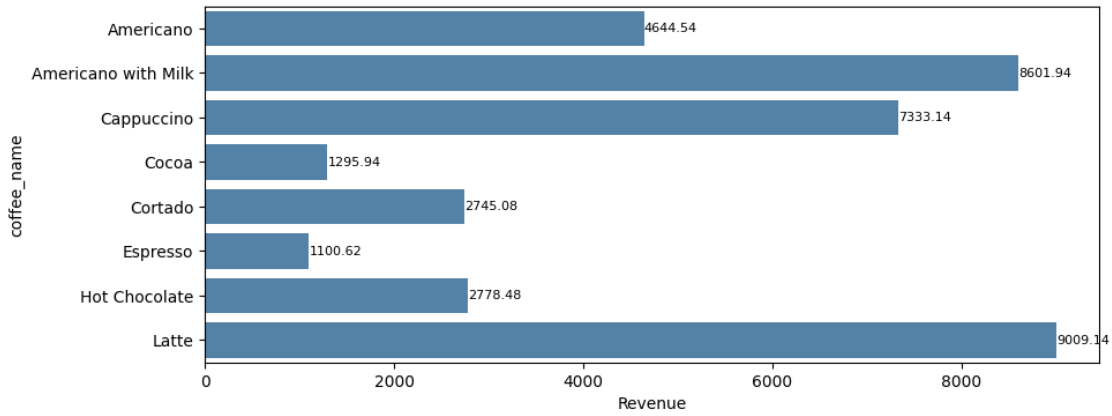


Tuesday has Highest Sales and Monday has Minimum Sales

[ ]:
```
# Overall Revenue of Each coffee

coffee_data = df.groupby('coffee_name')['money'].sum().reset_index()
```

[ ]:
```
plt.figure(figsize=(10,4))
ax = sns.barplot(data=coffee_data,x='money',y='coffee_name',color='steelblue')
ax.bar_label(ax.containers[0], fontsize=8)
plt.xlabel('Revenue')
```

[ ]:
```
Text(0.5, 0, 'Revenue')
```



[ ]:
```
monthly_sales = df.groupby(['coffee_name','month_name']).count()['date'].
↪unstack()
monthly_sales
```

[ ]:
month_name	April	July	June	March	May
coffee_name					
Americano	35	36	14	36	48
Americano with Milk	42	65	69	34	58
Cappuccino	43	32	46	20	55
Cocoa	6	9	5	6	9
Cortado	19	14	19	30	17
Espresso	7	14	10	10	8
Hot Chocolate	13	11	14	22	14
Latte	31	56	50	48	58

[ ]:
```
monthly_sales = df.groupby(['coffee_name','month_name'])['date'].count().
↪reset_index().pivot(index='month_name', columns='coffee_name',␣
↪values='date').reset_index()
monthly_sales
```

[ ]:
coffee_name	month_name	Americano	Americano with Milk	Cappuccino	Cocoa	\
0	April	35	42	43	6	
1	July	36	65	32	9	
2	June	14	69	46	5	
3	March	36	34	20	6	
4	May	48	58	55	9	

coffee_name	Cortado	Espresso	Hot Chocolate	Latte
0	19	7	13	31
1	14	14	11	56
2	19	10	14	50
3	30	10	22	48

```
4                17         8              14     58
```

[ ]:
```
monthly_sales.columns
```

[ ]:
```
Index(['month_name', 'Americano', 'Americano with Milk', 'Cappuccino', 'Cocoa',
       'Cortado', 'Espresso', 'Hot Chocolate', 'Latte'],
      dtype='object', name='coffee_name')
```

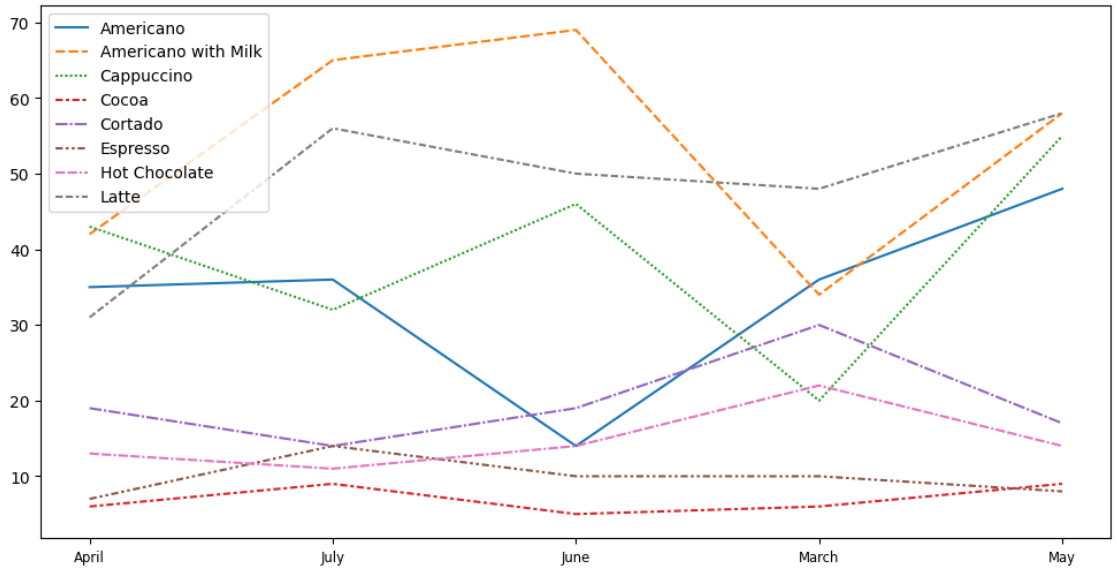
[ ]:
```
monthly_sales.describe().T.loc[:,['min','max']]
```

[ ]:
```
                      min   max
coffee_name
Americano            14.0  48.0
Americano with Milk  34.0  69.0
Cappuccino           20.0  55.0
Cocoa                 5.0   9.0
Cortado              14.0  30.0
Espresso              7.0  14.0
Hot Chocolate        11.0  22.0
Latte                31.0  58.0
```

4 Sales Plot By Coffee Type

[ ]:
```
plt.figure(figsize=(12,6))
sns.lineplot(data=monthly_sales)
plt.legend(loc='upper left')
plt.
↪xticks(range(len(monthly_sales['month_name'])),monthly_sales['month_name'],size='small')
```

[ ]:
```
([<matplotlib.axis.XTick at 0x7b6d504db610>,
  <matplotlib.axis.XTick at 0x7b6d504dbc40>,
  <matplotlib.axis.XTick at 0x7b6d4eaa0220>,
  <matplotlib.axis.XTick at 0x7b6d4eab1f00>,
  <matplotlib.axis.XTick at 0x7b6d4eab29b0>],
 [Text(0, 0, 'April'),
  Text(1, 0, 'July'),
  Text(2, 0, 'June'),
  Text(3, 0, 'March'),
  Text(4, 0, 'May')])
```



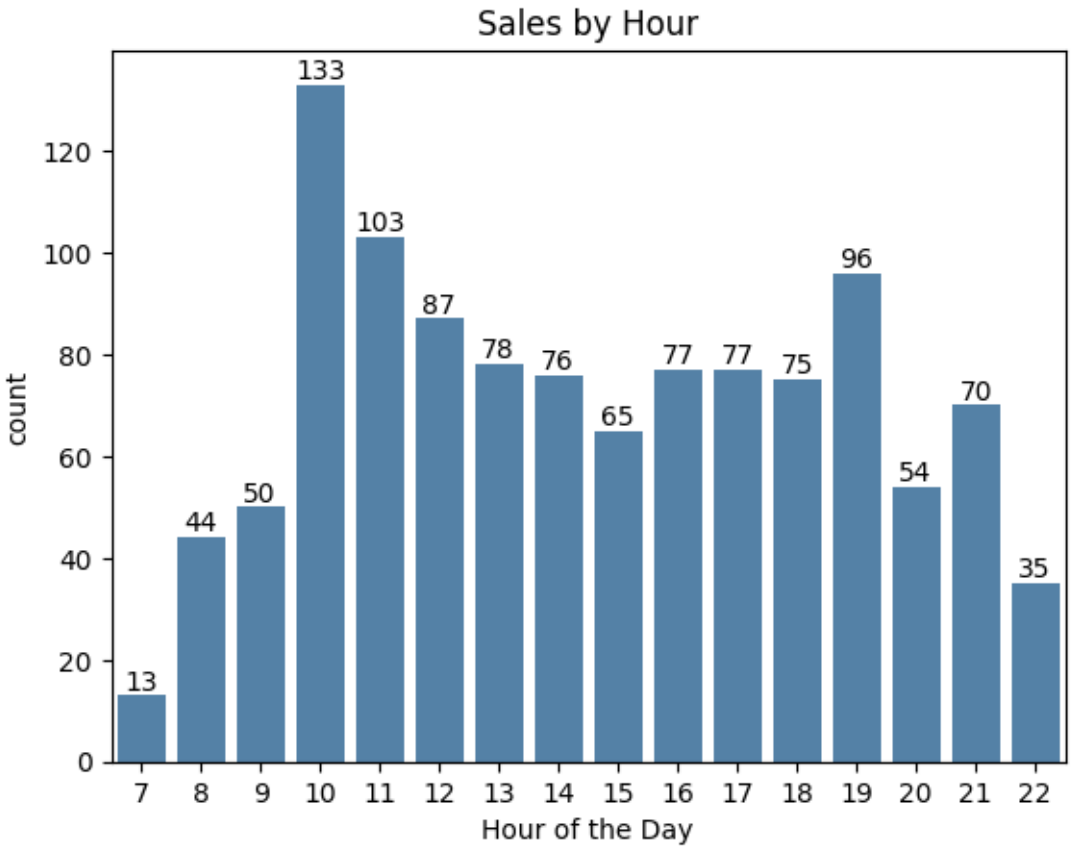
1. **Latte and Americano was the popular one with maximum revenue**
2. **Americano With Milk Was popular till june but after it declined**
3. **The Demand of Latte has increased over time**
4. **Cortado was lagging at the begining but its sales has increased with time**
5. **Cuppacino was fluctuating and it but it shows growth**

[ ]:
```
hourly_sales = df.groupby(['hour']).count()['date'].reset_index().
↪rename(columns={'date':'count'})
hourly_sales
```

[ ]:

	hour	count
0	7	13
1	8	44
2	9	50
3	10	133
4	11	103
5	12	87
6	13	78
7	14	76
8	15	65
9	16	77
10	17	77
11	18	75
12	19	96
13	20	54
14	21	70
15	22	35

[ ]:
```
ax = sns.barplot(data=hourly_sales,x='hour',y='count',color='steelblue')
ax.bar_label(ax.containers[0], fontsize=10)
plt.title('Sales by Hour')
plt.xlabel('Hour of the Day')
plt.show()
```



[ ]:
```
hourly_sales_by_coffee = df.groupby(['hour', 'coffee_name'])['date'].count().
  ↪reset_index().pivot(index='hour', columns='coffee_name', values='date').
  ↪reset_index().fillna(0)
hourly_sales_by_coffee
```

[ ]:

coffee_name	hour	Americano	Americano with Milk	Cappuccino	Cocoa	Cortado	\
0	7	5.0	4.0	1.0	0.0	1.0	
1	8	10.0	7.0	8.0	1.0	6.0	
2	9	8.0	16.0	6.0	1.0	5.0	
3	10	20.0	31.0	10.0	4.0	8.0	
4	11	21.0	25.0	16.0	1.0	13.0	
5	12	14.0	26.0	15.0	3.0	7.0	
6	13	18.0	18.0	10.0	2.0	12.0	
7	14	15.0	18.0	13.0	4.0	6.0	

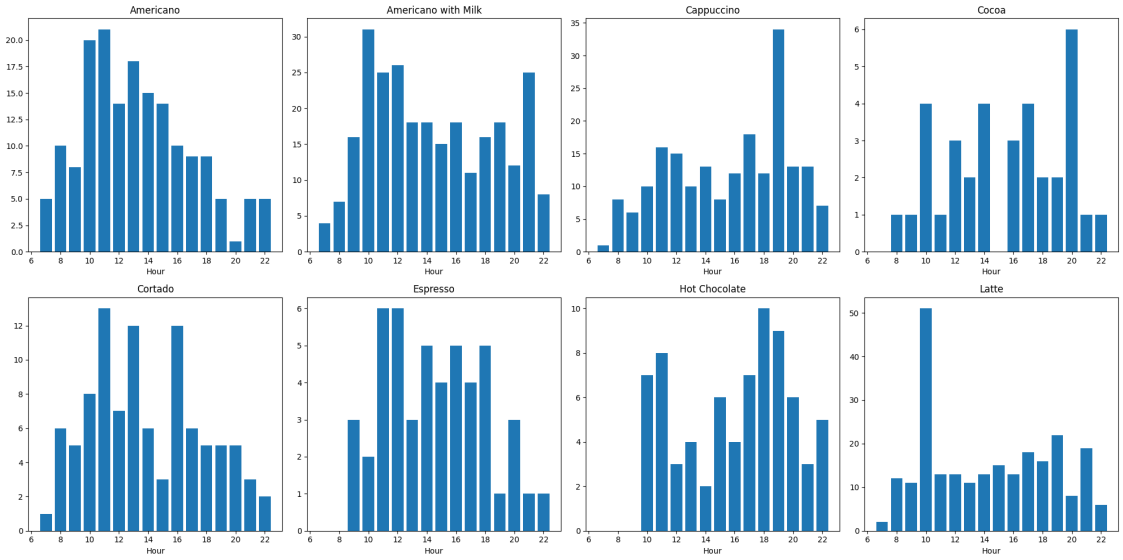
8	15	14.0	15.0	8.0	0.0	3.0
9	16	10.0	18.0	12.0	3.0	12.0
10	17	9.0	11.0	18.0	4.0	6.0
11	18	9.0	16.0	12.0	2.0	5.0
12	19	5.0	18.0	34.0	2.0	5.0
13	20	1.0	12.0	13.0	6.0	5.0
14	21	5.0	25.0	13.0	1.0	3.0
15	22	5.0	8.0	7.0	1.0	2.0

coffee_name	Espresso	Hot Chocolate	Latte
0	0.0	0.0	2.0
1	0.0	0.0	12.0
2	3.0	0.0	11.0
3	2.0	7.0	51.0
4	6.0	8.0	13.0
5	6.0	3.0	13.0
6	3.0	4.0	11.0
7	5.0	2.0	13.0
8	4.0	6.0	15.0
9	5.0	4.0	13.0
10	4.0	7.0	18.0
11	5.0	10.0	16.0
12	1.0	9.0	22.0
13	3.0	6.0	8.0
14	1.0	3.0	19.0
15	1.0	5.0	6.0

4.1 Sales Hour of Each Coffee Type

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

# Create a 2x4 grid of subplots
fig, axs = plt.subplots(2, 4, figsize=(20, 10))

# Flatten the array of subplots for easy iteration
axs = axs.flatten()

# Determine the number of columns you have (excluding the 'Index' column)
num_columns = len(hourly_sales_by_coffee.columns[1:])

# Loop through each column in the DataFrame, making sure not to exceed the␣
 ↪number of subplots
for i, column in enumerate(hourly_sales_by_coffee.columns[1:num_columns+1]):  #␣
 ↪Adjust the range as needed
    axs[i].bar(hourly_sales_by_coffee['hour'], hourly_sales_by_coffee[column])
    axs[i].set_title(f'{column}')
```

```
    axs[i].set_xlabel('Hour')

# Adjust the layout to prevent overlap
plt.tight_layout()

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



1. **Cuppacino is selling most at the evening**
2. **Cortado has steady sales at noon time**
3. **Expresso has higer steady sales and it maintains its pace**
4. **Latte is selling most at 10'O clock evening**

[ ]:
```
hourly_sales_by_coffee.columns[1:]
```

[ ]:
```
Index(['Americano', 'Americano with Milk', 'Cappuccino', 'Cocoa', 'Cortado',
       'Espresso', 'Hot Chocolate', 'Latte'],
      dtype='object', name='coffee_name')
```

[ ]:
```
for i, column in enumerate(hourly_sales_by_coffee.columns[1:num_columns+1]):
  print(i,column)
```

```
0 Americano
1 Americano with Milk
2 Cappuccino
3 Cocoa
4 Cortado
5 Espresso
6 Hot Chocolate
```

```
7 Latte
```

[ ]:
```
df['card'].nunique()
```

[ ]:
```
446
```

[ ]:
```
daily_sales
```

[ ]:
```
            date   money
0     2024-03-01  396.30
1     2024-03-02  228.10
2     2024-03-03  349.10
3     2024-03-04  135.20
4     2024-03-05  338.50
..           …     …
145   2024-07-27  372.76
146   2024-07-28   78.86
147   2024-07-29  321.82
148   2024-07-30  650.48
149   2024-07-31  633.84

[150 rows x 2 columns]
```

Forecasting sales

[ ]:
```
# Aggregate by day
daily_sales = df.groupby('date')['money'].sum().reset_index()
```

[ ]:
```
!pip install statsmodels --quiet
```

```
                         0.0/10.8 MB
? eta -:--:--

0.3/10.8 MB 8.6 MB/s eta 0:00:02

3.0/10.8 MB 43.5 MB/s eta 0:00:01

10.3/10.8 MB 120.2 MB/s eta 0:00:01
                       10.8/10.8 MB
161.0 MB/s eta 0:00:01
                       10.8/10.8 MB
87.3 MB/s eta 0:00:00
                          0.0/232.9
kB ? eta -:--:--
                       232.9/232.9 kB
17.0 MB/s eta 0:00:00
```

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

# Example of fitting an ARIMA model
model = ARIMA(daily_sales['money'], order=(1, 1, 1))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  money   No. Observations:                  150
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -922.976
Date:                Sun, 01 Dec 2024   AIC                           1851.951
Time:                        12:02:55   BIC                           1860.963
Sample:                             0   HQIC                          1855.613
                                  - 150
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0777      0.114     -0.681      0.496      -0.301       0.146
ma.L1         -0.7575      0.078     -9.702      0.000      -0.910      -0.604
sigma2      1.398e+04   1717.177      8.139      0.000    1.06e+04    1.73e+04
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
4.85
Prob(Q):                              0.92   Prob(JB):
0.09
Heteroskedasticity (H):               1.39   Skew:
0.44
Prob(H) (two-sided):                  0.25   Kurtosis:
2.85
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

4.2 Analysing Customer prefrence

[ ]:
```
coffee_preference = df['coffee_name'].value_counts(normalize=True) * 100
coffee_preference
```

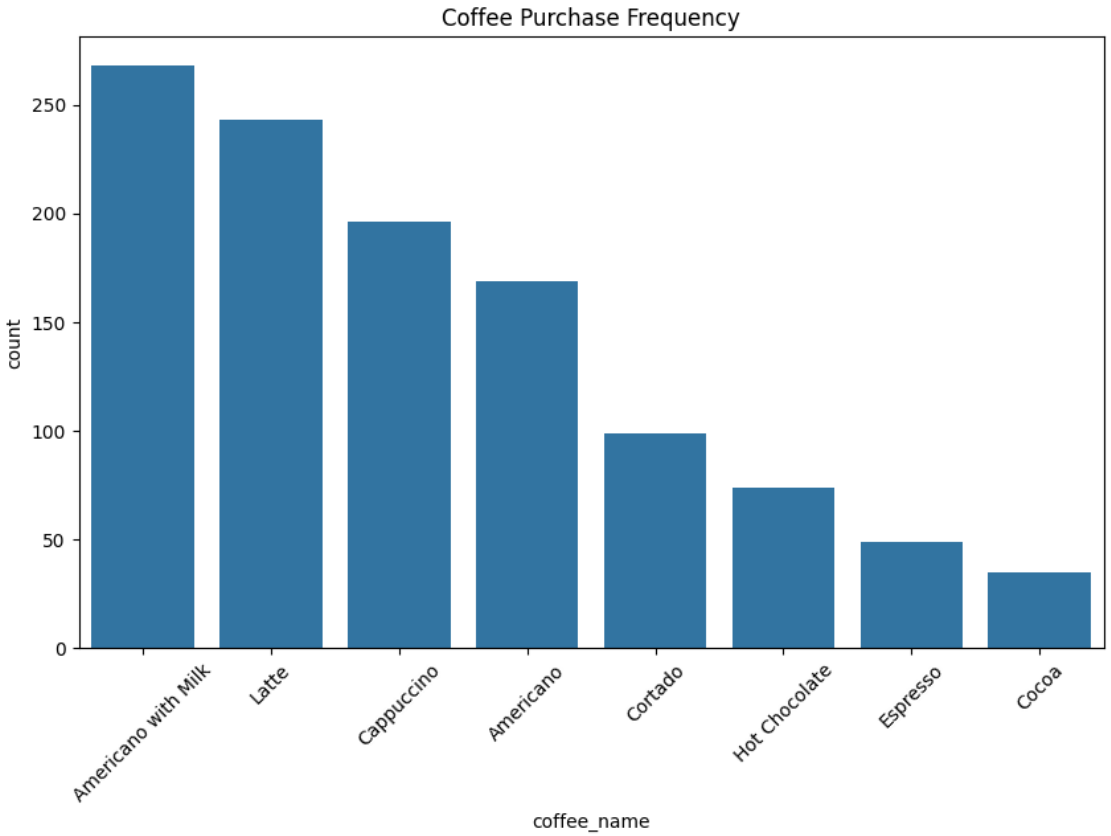
[ ]:
```
coffee_name
Americano with Milk    23.654016
Latte                  21.447485
```

```
Cappuccino              17.299206
Americano               14.916152
Cortado                  8.737864
Hot Chocolate            6.531333
Espresso                 4.324801
Cocoa                    3.089144
Name: proportion, dtype: float64
```

[ ]:
```
od = df['coffee_name'].value_counts().index
od
```

[ ]:
```
Index(['Americano with Milk', 'Latte', 'Cappuccino', 'Americano', 'Cortado',
       'Hot Chocolate', 'Espresso', 'Cocoa'],
      dtype='object', name='coffee_name')
```

[ ]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='coffee_name', order=df['coffee_name'].value_counts().
 ↪index)
plt.title('Coffee Purchase Frequency')
plt.xticks(rotation=45)
plt.show()
```



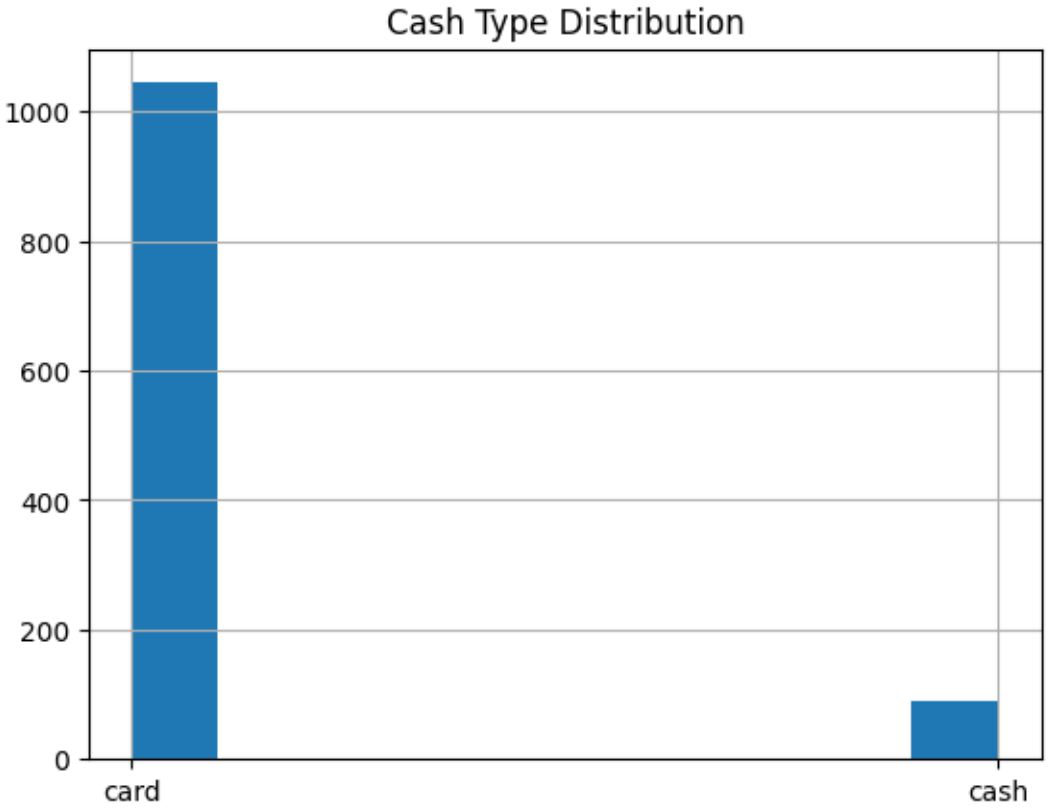
[ ]:
```
df.loc[:,['cash_type','card','coffee_name']].describe().T
```

[ ]:

	count	unique	top	freq
cash_type	1133	2	card	1044
card	1133	446	ANON-0000-0000-0012	177
coffee_name	1133	8	Americano with Milk	268

1. Americano with Milk is the most popular product
2. maximum payment occured by card
3. Mostly ANON card is used here

[ ]:
```
df['cash_type'].hist().set_title('Cash Type Distribution')
plt.show()
```



5 Time Series Anaysis For the Sales of the Coffee

[ ]:
```
df.head(3)
```

[ ]:
```
                 datetime cash_type                 card  money  \
0 2024-03-01 10:15:50.520      card  ANON-0000-0000-0001   38.7
1 2024-03-01 12:19:22.539      card  ANON-0000-0000-0002   38.7
2 2024-03-01 12:20:18.089      card  ANON-0000-0000-0002   38.7

     coffee_name        date  hour  day  month_num  year weekday month_name  \
0          Latte  2024-03-01    10    1          3  2024  Friday      March
1  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March
2  Hot Chocolate  2024-03-01    12    1          3  2024  Friday      March

  year_month
0    2024-03
1    2024-03
2    2024-03
```

[ ]:
```
DF = df.groupby('date')['money'].sum().reset_index()
DF
```

[ ]:
```
           date   money
0    2024-03-01  396.30
1    2024-03-02  228.10
2    2024-03-03  349.10
3    2024-03-04  135.20
4    2024-03-05  338.50
..          …     …
145  2024-07-27  372.76
146  2024-07-28   78.86
147  2024-07-29  321.82
148  2024-07-30  650.48
149  2024-07-31  633.84

[150 rows x 2 columns]
```

[ ]:
```
DF.shape
```

[ ]:
```
(150, 2)
```

[ ]:
```
print(type(DF))
```

```
<class 'pandas.core.frame.DataFrame'>
```

[ ]:
```
DF['date'].min(), DF['date'].max()
```

[ ]:
```
(datetime.date(2024, 3, 1), datetime.date(2024, 7, 31))
```

[ ]:
```
# print(DF.index.min())
# print(DF.index.max())
```

[ ]:
```
DF.isnull().sum()
```

[ ]:
```
date     0
money    0
dtype: int64
```

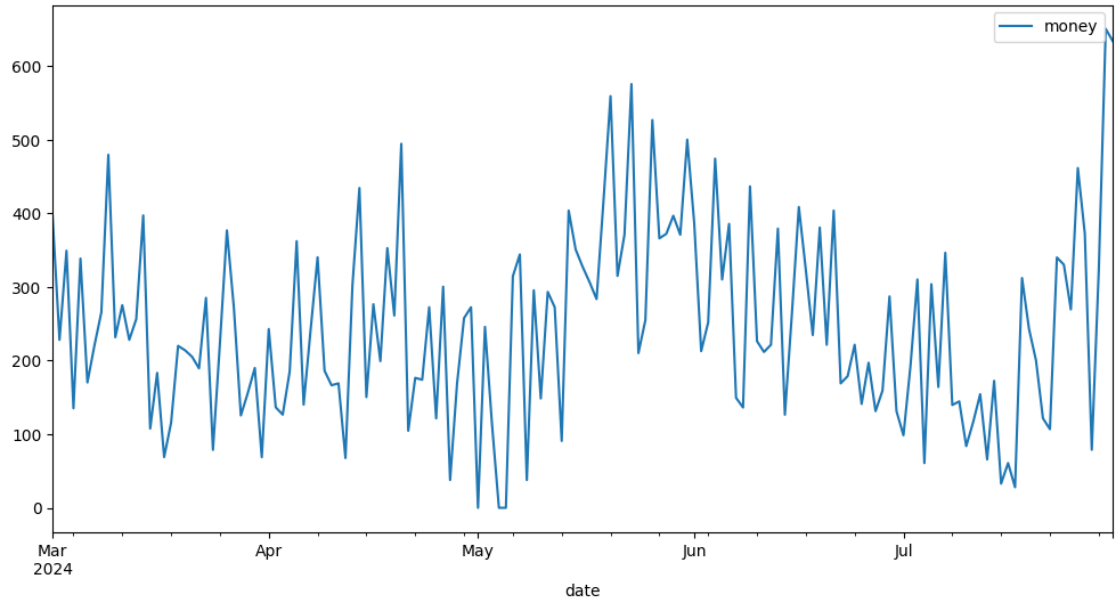
[ ]:
```
# Display rows where there are null values
print(DF[DF.isnull()])
```

```
    date  money
0    NaN    NaN
1    NaN    NaN
2    NaN    NaN
3    NaN    NaN
4    NaN    NaN
..   …    …
145  NaN    NaN
146  NaN    NaN
147  NaN    NaN
148  NaN    NaN
149  NaN    NaN

[150 rows x 2 columns]
```

[ ]:
```
DF.dtypes
```

[ ]:
```
date      object
money    float64
dtype: object
```

[ ]:
```
DF.plot(figsize=(12,6))
plt.show()
```



[ ]:
```
DF.set_index('date', inplace=True)
```

Seasonal Decompose

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

[ ]:
```
# Example: Convert your Series to have a DatetimeIndex
DF.index = pd.to_datetime(DF.index)  # Ensure the index is in datetime format
DF = DF.asfreq('D')  # Set the frequency (e.g., 'D' for daily data)
```

[ ]:
```
DF.fillna(0, inplace=True)
```

[ ]:
```
print(type(DF))
```

```
<class 'pandas.core.frame.DataFrame'>
```

[ ]:
```
DF.head()
```

[ ]:
```
            money
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
```

[ ]:
```
DF.shape
```

[ ]: (153, 1)

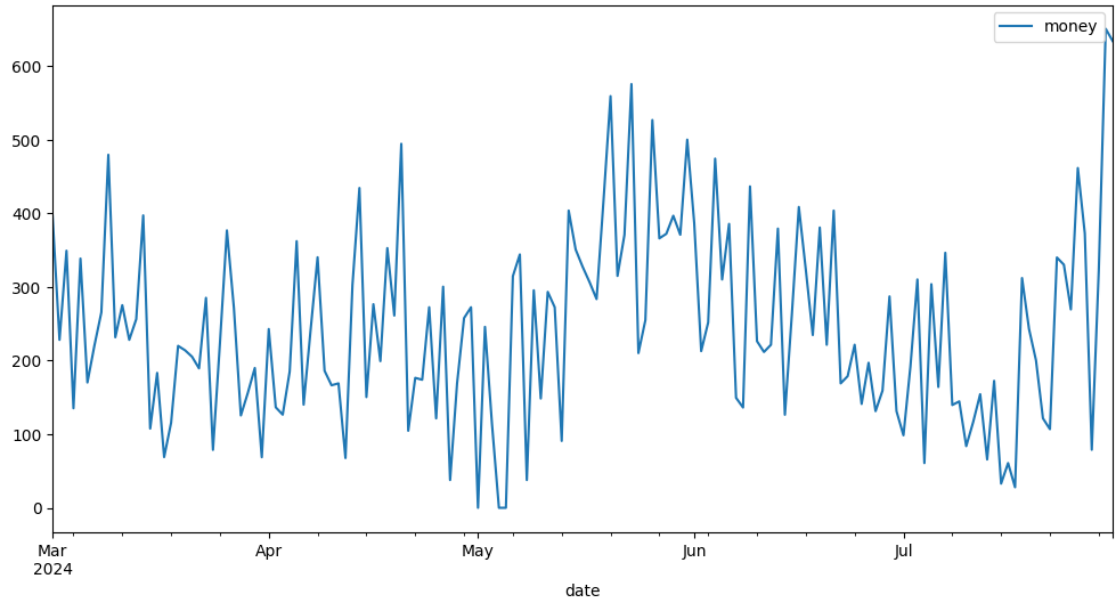
[ ]:
```
DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 153 entries, 2024-03-01 to 2024-07-31
Freq: D
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   money   153 non-null    float64
dtypes: float64(1)
memory usage: 2.4 KB
```

[ ]:
```
df1 = DF.copy()
df2 = DF.copy()
df3 = DF.copy()
```

[ ]:
```
from statsmodels.tsa.stattools import adfuller

def adf_test(data):
    res = adfuller(data)
    print('test_stat',res[0])
    print('p_val',res[1])
    alpha = 0.05
    if res[1]> alpha:
        print('Ho accepted: Data is not stationary')
    else:
        print('H1 accepted: Data is stationary')
```

[ ]:
```
# plt.figure()
df1.plot(figsize=(12,6))
plt.show()
```



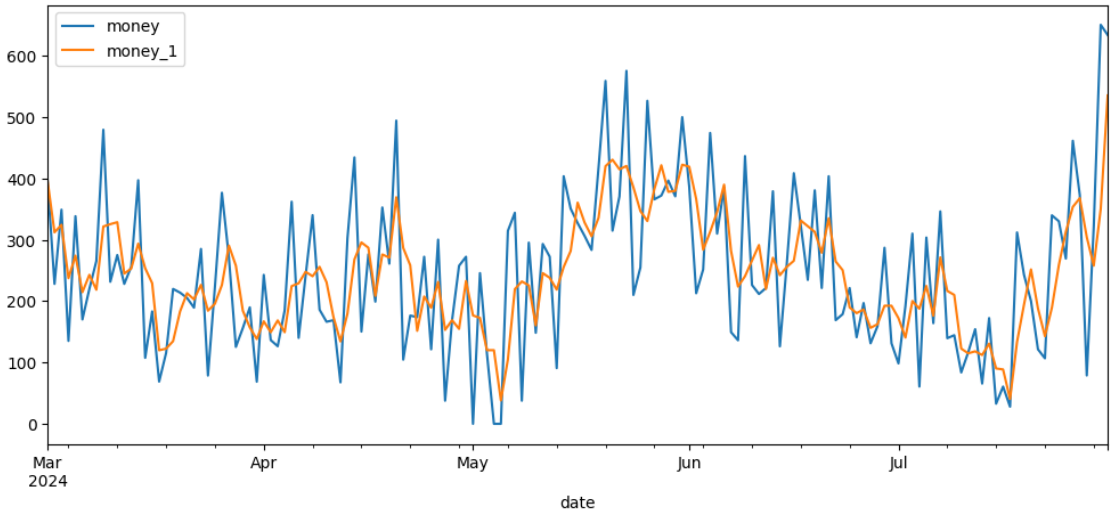
[ ]:
```
df1['money_1'] = df1['money'].rolling(window=3, min_periods=1).mean()
```

[ ]:
```
df1
```

[ ]:
```
             money     money_1
date
2024-03-01  396.30  396.300000
2024-03-02  228.10  312.200000
2024-03-03  349.10  324.500000
2024-03-04  135.20  237.466667
2024-03-05  338.50  274.266667
…              …         …
2024-07-27  372.76  367.860000
2024-07-28   78.86  304.346667
2024-07-29  321.82  257.813333
2024-07-30  650.48  350.386667
2024-07-31  633.84  535.380000

[153 rows x 2 columns]
```

[ ]:
```
df1[['money', 'money_1']].plot(figsize=(12,5))
```

[ ]:
```
<Axes: xlabel='date'>
```



[ ]:
```
adf_test(df1['money'])
```

```
test_stat -2.3950547665243396
p_val 0.1431370317996271
Ho accepted: Data is not stationary
```

[ ]:
```
adf_test(df1['money'].diff(1).dropna()) # d = 1
```

```
test_stat -9.087181511770668
p_val 3.911567323624255e-15
H1 accepted: Data is stationary
```

[ ]:
```
adf_test(df1['money'].diff(2).dropna()) # d = 1
```

```
test_stat -4.345448293269364
p_val 0.0003702475050956096
H1 accepted: Data is stationary
```

[ ]:
```
adf_test(df1['money'].diff(1).diff(1).dropna()) # d = 1
```

```
test_stat -6.964886368850894
p_val 8.967792091186672e-10
H1 accepted: Data is stationary
```

[ ]:
```
fcs = df1['money'].diff(1).diff(1).dropna()
```

[ ]:
```
adf_test(df1['money_1'])
```

```
test_stat -1.9937783876748023
p_val 0.28930770475893386
```

```
Ho accepted: Data is not stationary
```

[ ]:
```
adf_test(df1['money_1'].diff(1).dropna())
```

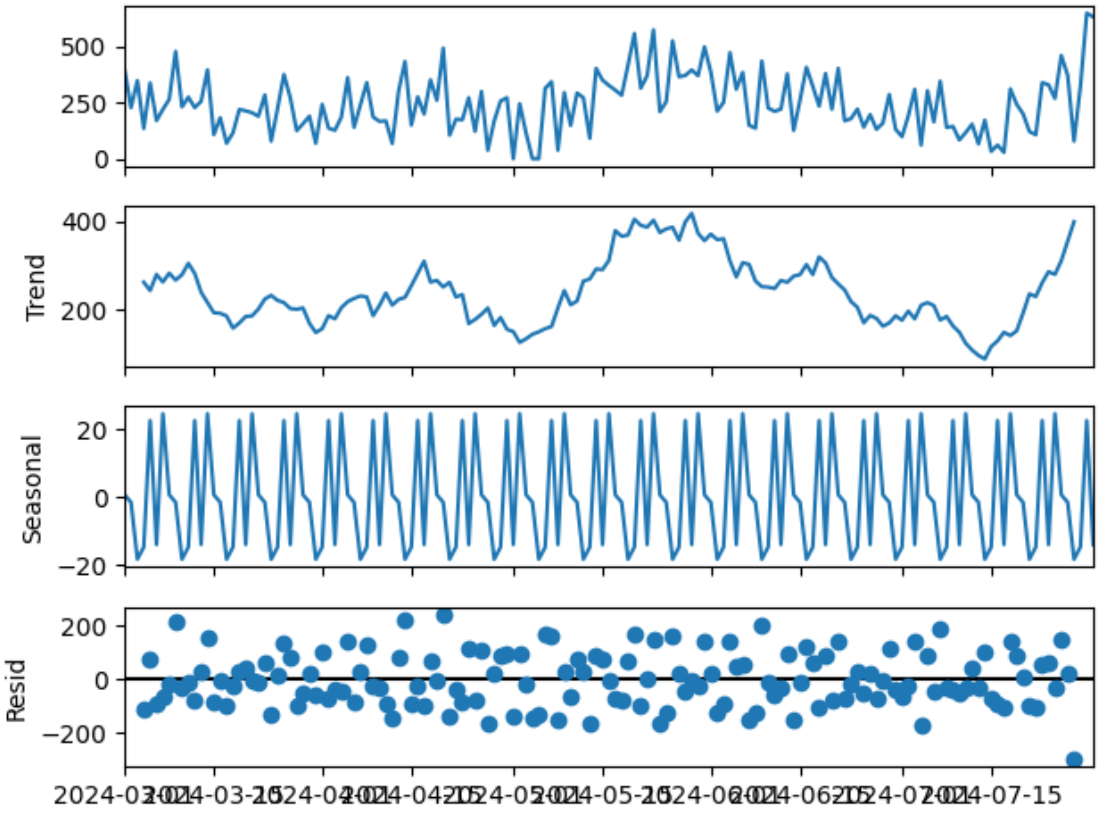
```
test_stat -4.371627408321249
p_val 0.00033309807852419695
H1 accepted: Data is stationary
```

[ ]:
```
adf_test(df1['money_1'].diff(1).diff(1).dropna())
```

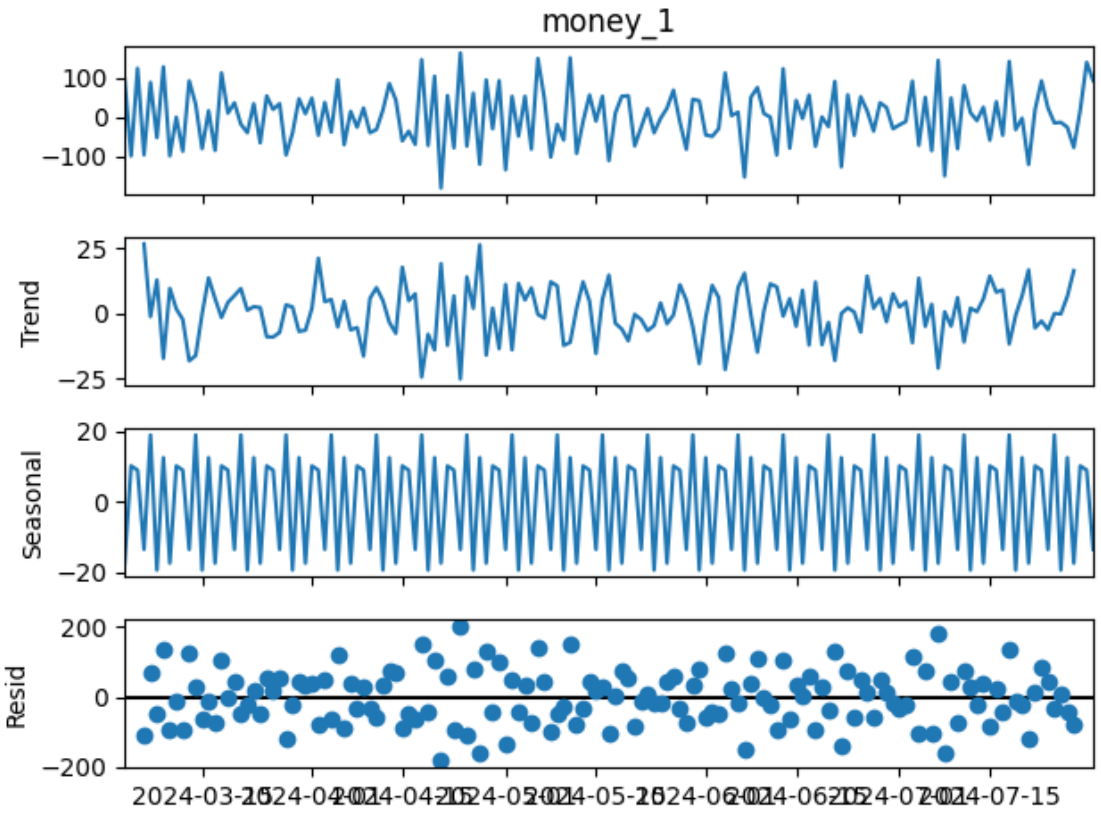
```
test_stat -6.742474178487956
p_val 3.093772645066475e-09
H1 accepted: Data is stationary
```

[ ]:
```
df_sm = df1['money_1'].diff(1).diff(1).dropna()
```

[ ]:
```
# Decomposition of Normal sales data with group averaging

decom = seasonal_decompose(DF, model='additive')
decom.plot()
plt.show()
```

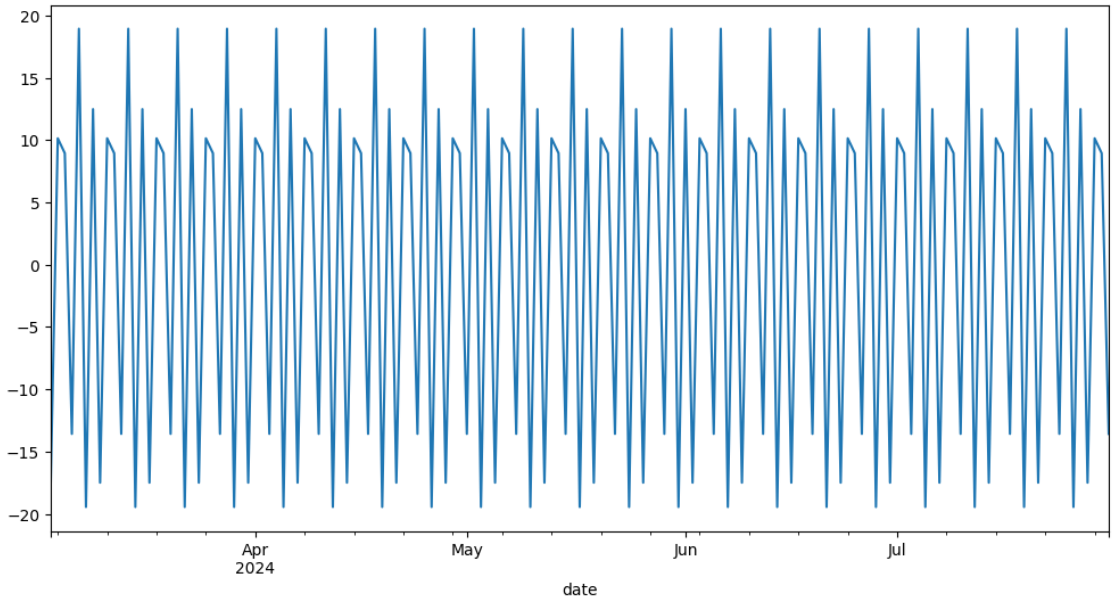


[ ]:
```
# Decomposition with Smoothened Data

decom = seasonal_decompose(df_sm, model='additive')

original = decom.observed
trend = decom.trend
seasonal = decom.seasonal
resid = decom.resid

decom.plot()
plt.show()
```



[ ]:
```
seasonal.plot(figsize=(12,6))
```

[ ]:
```
<Axes: xlabel='date'>
```



[ ]:
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import *

def eval_model(test,fcast):
    mae = mean_absolute_error(test,fcast)
    mse = mean_squared_error(test,fcast)
    rmse = np.sqrt(mse)
    return mae,mse,rmse

def plot_model(train,test,fcast):
    plt.figure(figsize=(12,6))
    plt.plot(train,label='Train')
    plt.plot(test,label='Test')
    plt.plot(fcast,label='Forecast')
    plt.legend()
    plt.show()

def custom_arima(train,test,p,d,q):
  model = ARIMA(np.log(train),order=(p,d,q))
  model_fit = model.fit()
  fcast = np.exp(model_fit.forecast(len(test)))
  plot_model(train,test,fcast)
  mae,mse,rmse = eval_model(test,fcast)
  res_df = pd.DataFrame({'MAE':mae,'MSE':mse,'RMSE':rmse},
                        index=[f'ARIMA({p,d,q})'])
  return res_df
```

[ ]:
```
print(DF.index.dtype)  # Displays the index (in this case, the dates)
print(DF.values.dtype)  # Displays the values (in this case, the money sums)
```

```
datetime64[ns]
float64
```

[ ]:
```
fcs.head()
```

[ ]:
```
date
2024-03-03    289.2
2024-03-04   -334.9
2024-03-05    417.2
2024-03-06   -371.6
2024-03-07    218.2
Freq: D, Name: money, dtype: float64
```

[ ]:
```
fcs.shape
```

[ ]:
```
(151,)
```

[ ]:
```
train = DF.iloc[:-30]
test = DF.iloc[-30:]

print(train.shape)
print(test.shape)
print(type(train))
print(type(test))
```

```
(123, 1)
(30, 1)
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.frame.DataFrame'>
```

[ ]:
```
train2 = fcs.iloc[:-30]
test2 = fcs.iloc[-30:]

print(train2.shape)
print(test2.shape)
print(type(train2))
print(type(test2))
```

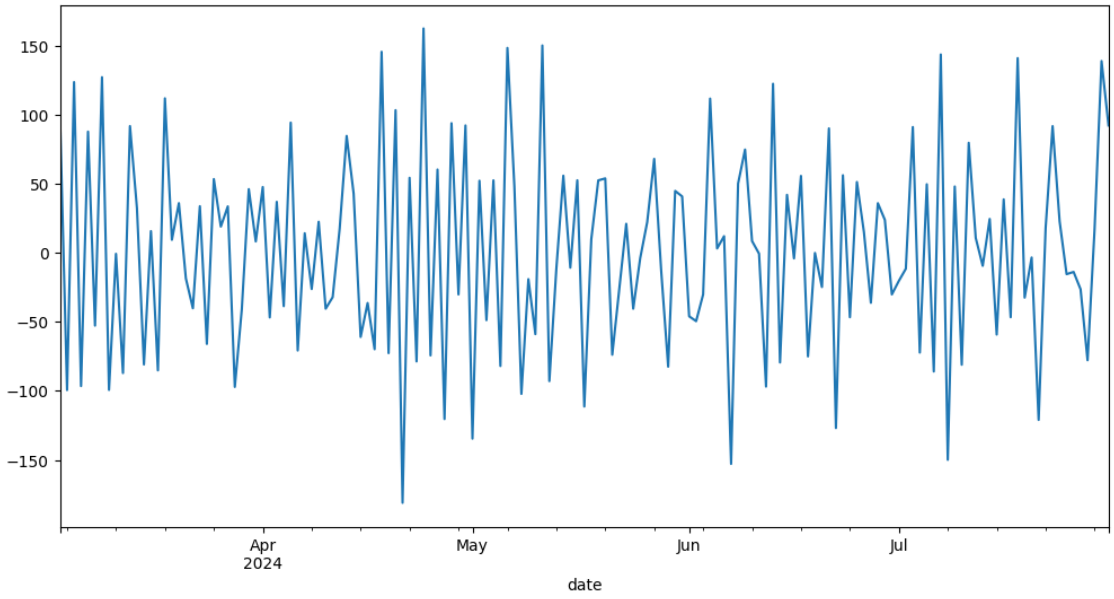
```
(121,)
(30,)
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[ ]:
```
df_sm.shape
```

[ ]: (151,)

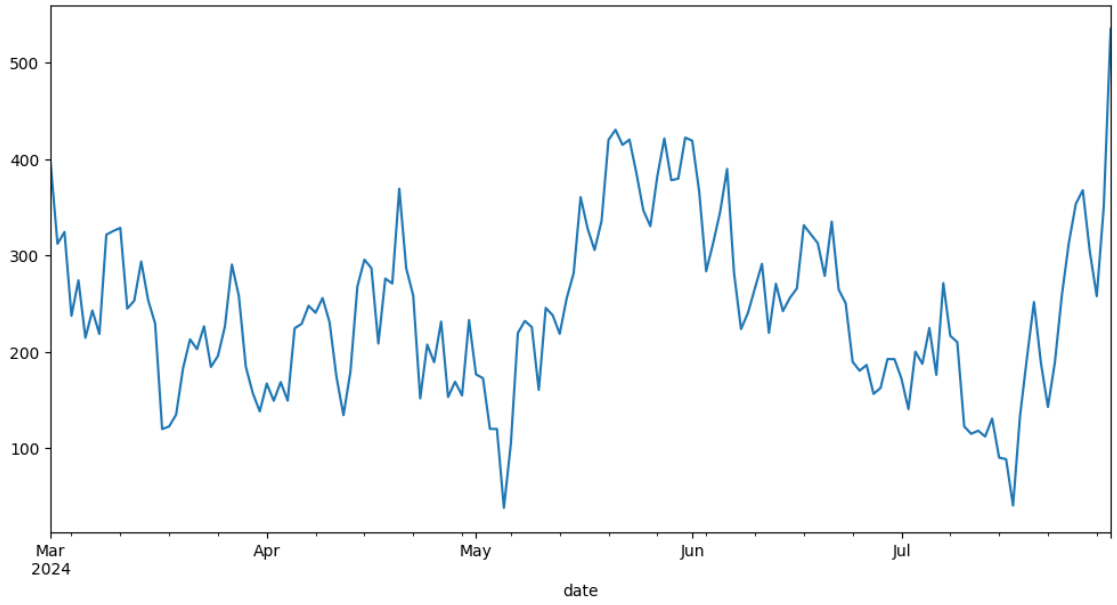
[ ]:
```
df_sm.plot(figsize=(12,6))
```

[ ]: <Axes: xlabel='date'>



[ ]:
```
sm_data = df1['money_1']
sm_data.plot(figsize=(12,6))
```

[ ]: <Axes: xlabel='date'>



[ ]:
```
train3 = df_sm.iloc[:-30]
test3 = df_sm.iloc[-30:]

print(train3.shape)
print(test3.shape)
print(type(train3))
print(type(test3))
```

```
(121,)
(30,)
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[ ]:
```
train3.shape
```

[ ]: (121,)

[ ]:
```
train4 = seasonal.iloc[:-30]
test4 = seasonal.iloc[-30:]

print(train4.shape)
print(test4.shape)
print(type(train4))
print(type(test4))
```

```
(121,)
(30,)
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

[ ]:
```
train5 = sm_data.iloc[:-30]
test5 = sm_data.iloc[-30:]

print(train5.shape)
print(test5.shape)
print(type(train5))
print(type(test5))
```

```
(123,)
(30,)
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

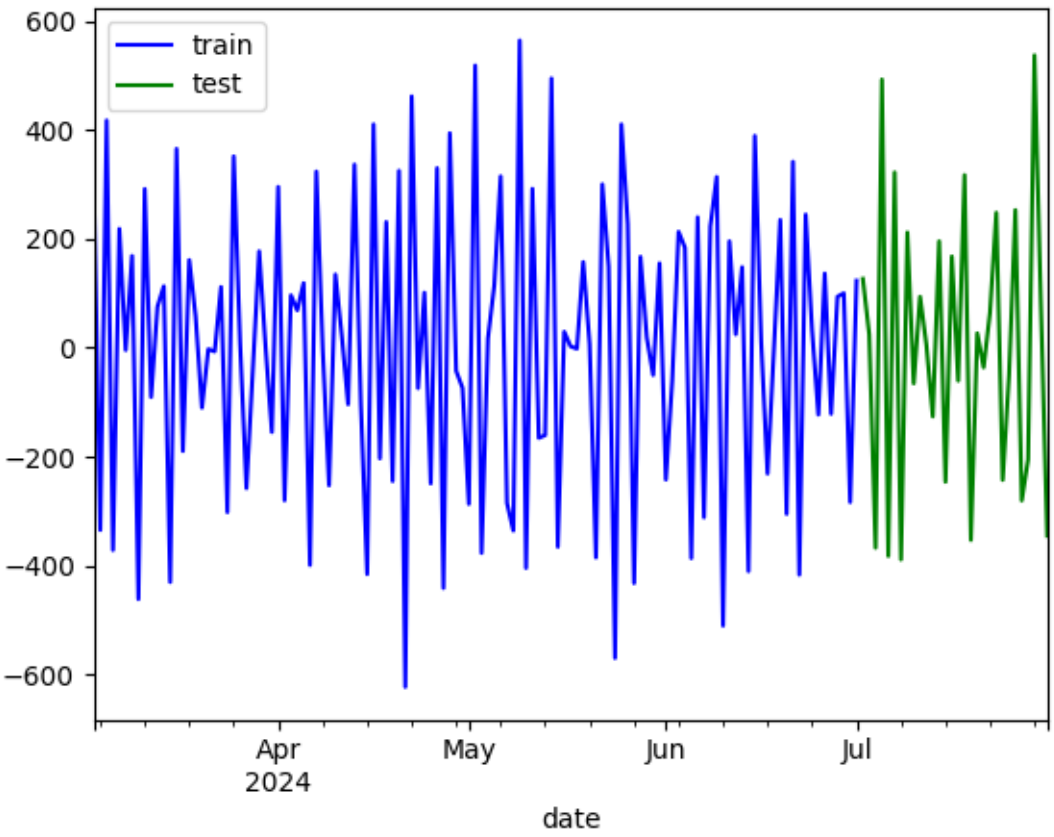
[ ]:
```
train.tail()
```

[ ]:
```
date
2024-06-27   -121.48
2024-06-28     93.56
2024-06-29     99.94
2024-06-30   -283.64
2024-07-01    122.96
Freq: D, Name: money, dtype: float64
```

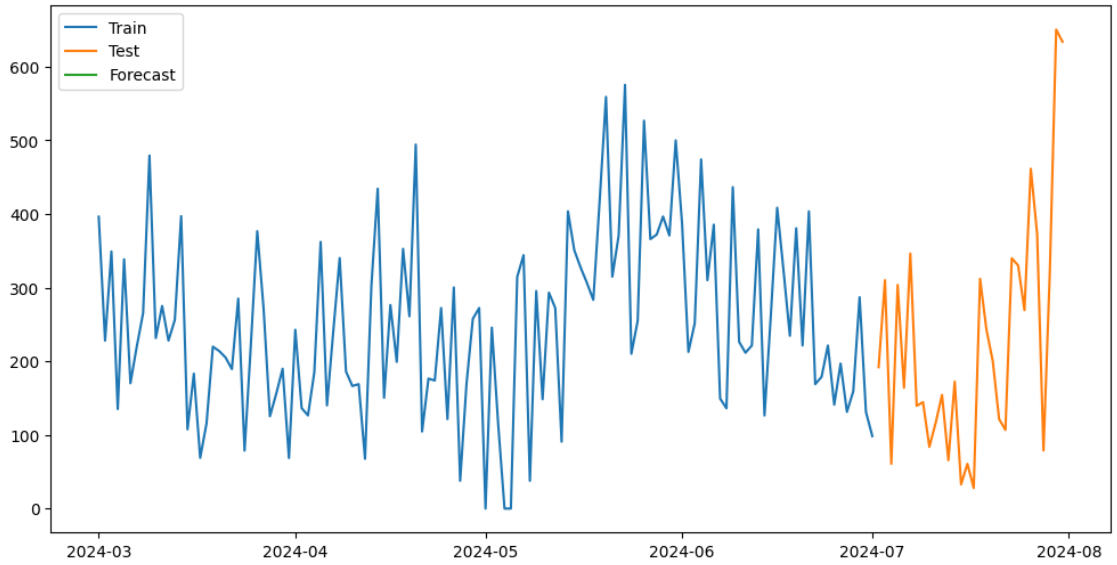
[ ]:
```
test.head()
```

[ ]:
```
date
2024-07-02    126.38
2024-07-03     24.50
2024-07-04   -367.40
2024-07-05    492.30
2024-07-06   -382.56
Freq: D, Name: money, dtype: float64
```

[ ]:
```
train.plot(color='blue',label='train')
test.plot(color='green', label='test')
plt.legend()
plt.show()
```



[ ]:
```
custom_arima(train,test,1,1,1)
```

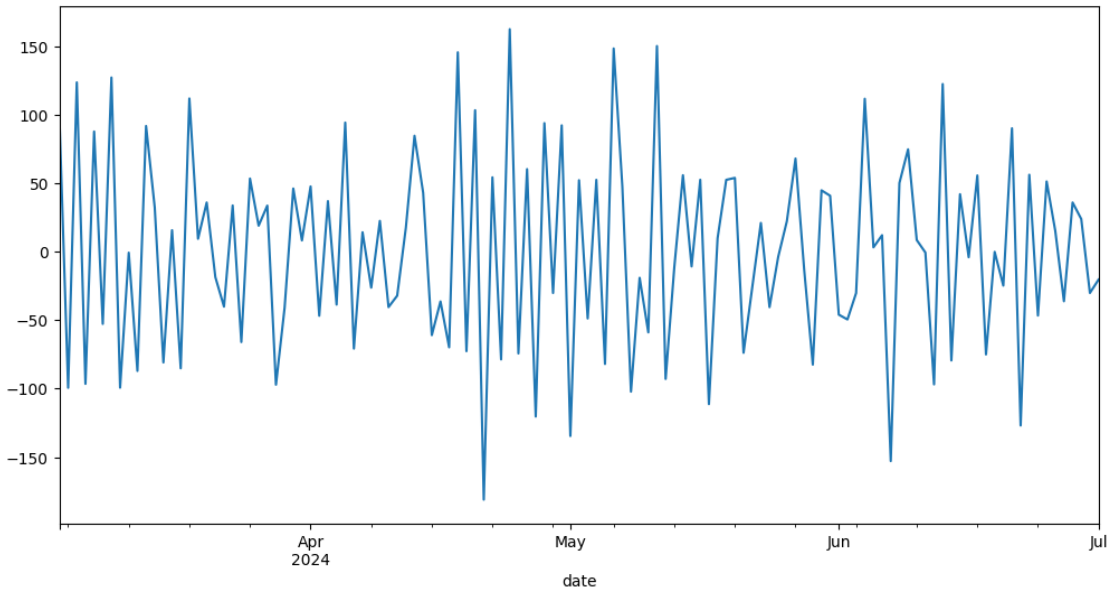


```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-181-a4333e8e25d4> in <cell line: 1>()
----> 1 custom_arima(train,test,1,1,1)

<ipython-input-180-ac8867a8a068> in custom_arima(train, test, p, d, q)
     21   fcast = np.exp(model_fit.forecast(len(test)))
     22   plot_model(train,test,fcast)
---> 23   mae,mse,rmse = eval_model(test,fcast)
     24   res_df = pd.DataFrame({'MAE':mae,'MSE':mse,'RMSE':rmse},
     25                         index=[f'ARIMA({p,d,q})'])

<ipython-input-180-ac8867a8a068> in eval_model(test, fcast)
      3
      4 def eval_model(test,fcast):
----> 5     mae = mean_absolute_error(test,fcast)
      6     mse = mean_squared_error(test,fcast)
      7     rmse = np.sqrt(mse)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py in␣
 ↪wrapper(*args, **kwargs)
    211                     )
    212                 ):
--> 213                     return func(*args, **kwargs)
    214             except InvalidParameterError as e:
    215                 # When the function is just a wrapper around an␣
 ↪estimator, we allow

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py in␣
 ↪mean_absolute_error(y_true, y_pred, sample_weight, multioutput)
    214     np.float64(0.85…)
    215     """
--> 216     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    217         y_true, y_pred, multioutput
    218     )

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_regression.py in␣
 ↪_check_reg_targets(y_true, y_pred, multioutput, dtype, xp)
    111     check_consistent_length(y_true, y_pred)
    112     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
--> 113     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
    114
    115     if y_true.ndim == 1:
```

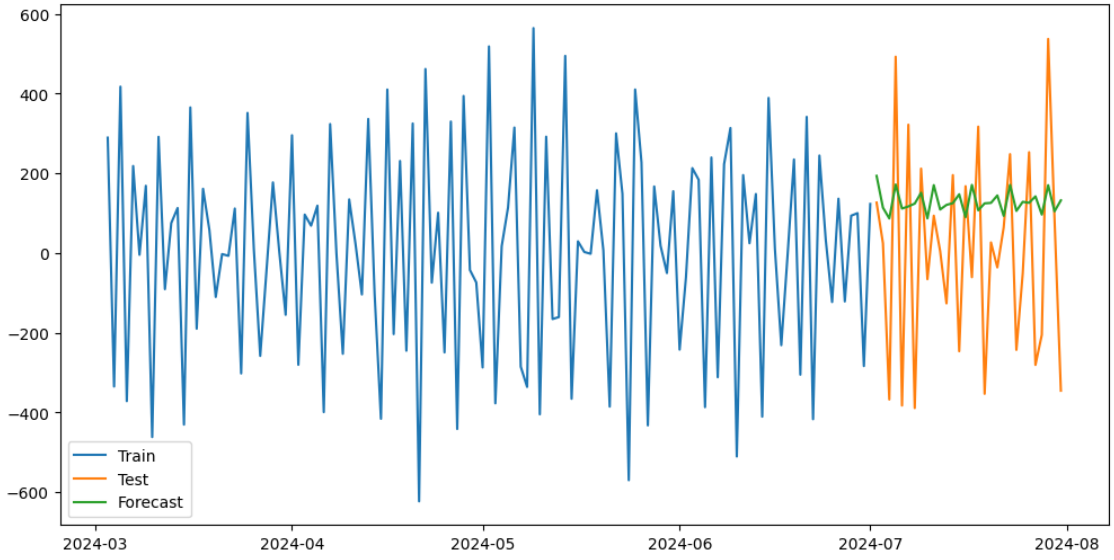
```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
↪check_array(array, accept_sparse, accept_large_sparse, dtype, order, copy,␣
↪force_writeable, force_all_finite, ensure_2d, allow_nd, ensure_min_samples,␣
↪ensure_min_features, estimator, input_name)
   1062
   1063         if force_all_finite:
-> 1064             _assert_all_finite(
   1065                 array,
   1066                 input_name=input_name,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
↪_assert_all_finite(X, allow_nan, msg_dtype, estimator_name, input_name)
    121         return
    122
--> 123     _assert_all_finite_element_wise(
    124         X,
    125         xp=xp,

/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py in␣
↪_assert_all_finite_element_wise(X, xp, allow_nan, msg_dtype, estimator_name,␣
↪input_name)
    170                 "#estimators-that-handle-nan-values"
    171             )
--> 172         raise ValueError(msg_err)
    173
    174

ValueError: Input contains NaN.
```

[ ]:
```
train3.plot(figsize=(12,6))
```

[ ]:
```
<Axes: xlabel='date'>
```



[ ]:
```
custom_arima(train2,test2,3,2,10)
```

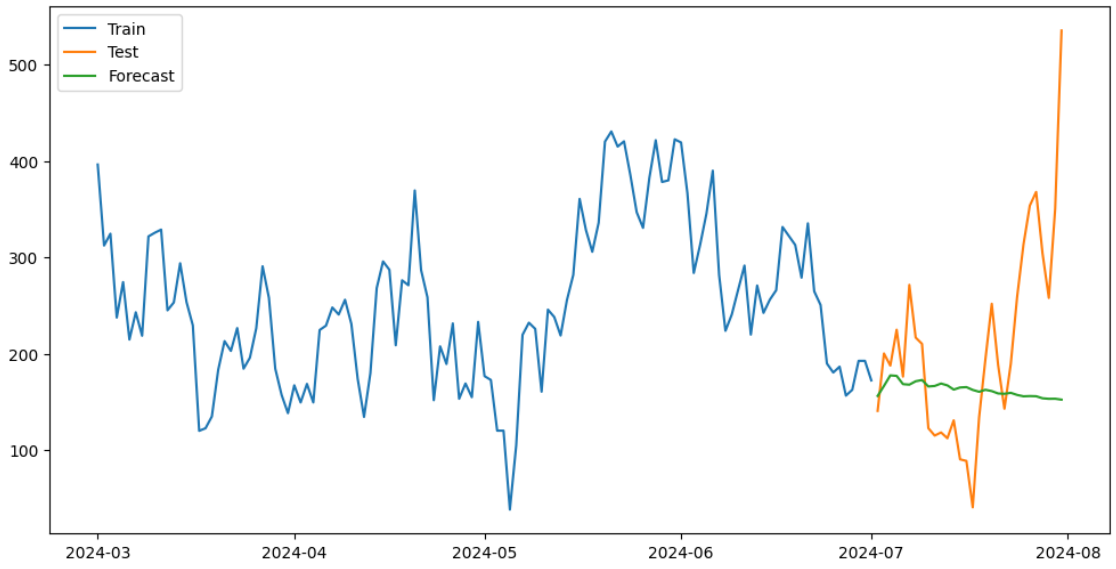


[ ]:

	MAE	MSE	RMSE
ARIMA((3, 2, 10))	229.059941	77267.496169	277.970315

[ ]:
```

```

[ ]:
```
custom_arima(train5,test5,3,2,10)
```



[ ]:

	MAE	MSE	RMSE
ARIMA((3, 2, 10))	84.536477	13531.374968	116.324438

[ ]:

[ ]:
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_pred)
    print(f"MSE: {mse}, MAE: {mae}, RMSE: {rmse}, R2: {r2}")
```

[ ]:
```
m1 = ARIMA(train, order=(1,1,1))
m1_fit = m1.fit()
```

[ ]:
```
print(test.shape)
```

```
(30,)
```

[ ]:
```
forcast_m1 = m1_fit.forecast(30)
forcast_m1
```

[ ]:
```
2024-07-02   -79.672134
2024-07-03    51.132653
2024-07-04   -33.305545
2024-07-05    21.201706
```

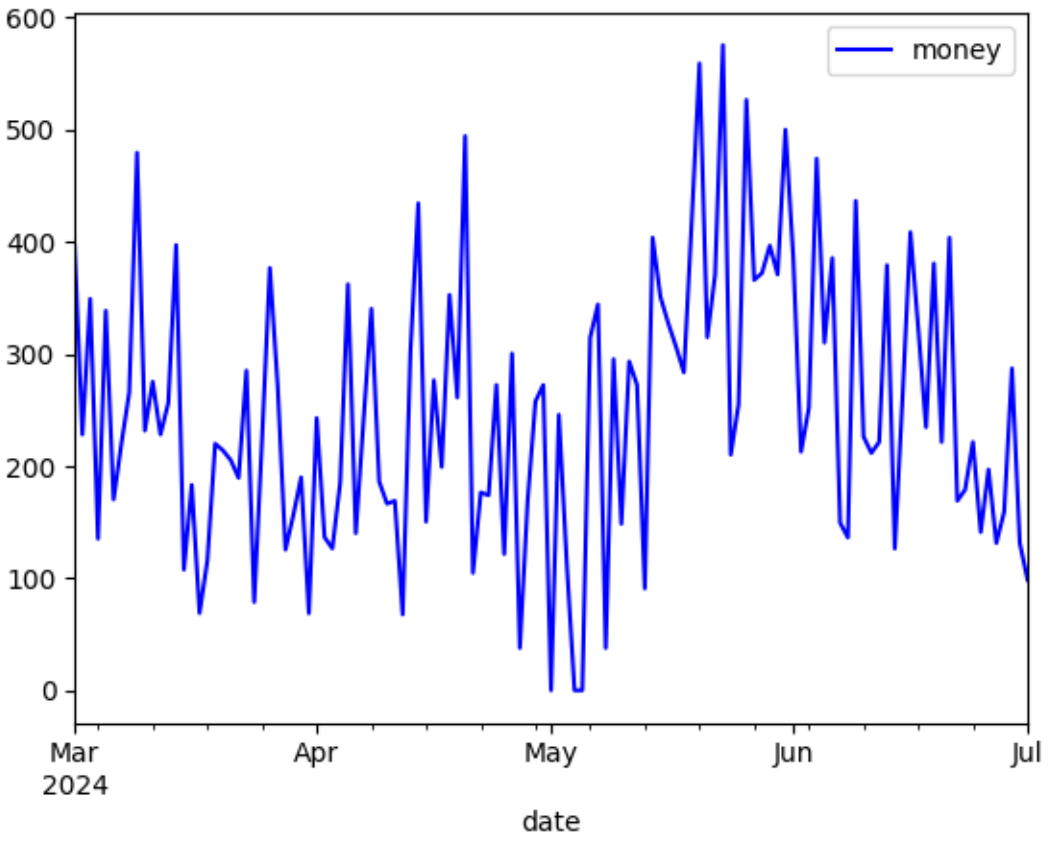
```
2024-07-06   -13.984270
2024-07-07     8.729275
2024-07-08    -5.932962
2024-07-09     3.531927
2024-07-10    -2.577927
2024-07-11     1.366157
2024-07-12    -1.179861
2024-07-13     0.463666
2024-07-14    -0.597277
2024-07-15     0.087592
2024-07-16    -0.354511
2024-07-17    -0.069121
2024-07-18    -0.253348
2024-07-19    -0.134425
2024-07-20    -0.211193
2024-07-21    -0.161637
2024-07-22    -0.193627
2024-07-23    -0.172976
2024-07-24    -0.186307
2024-07-25    -0.177702
2024-07-26    -0.183257
2024-07-27    -0.179671
2024-07-28    -0.181985
2024-07-29    -0.180491
2024-07-30    -0.181456
2024-07-31    -0.180833
Freq: D, Name: predicted_mean, dtype: float64
```

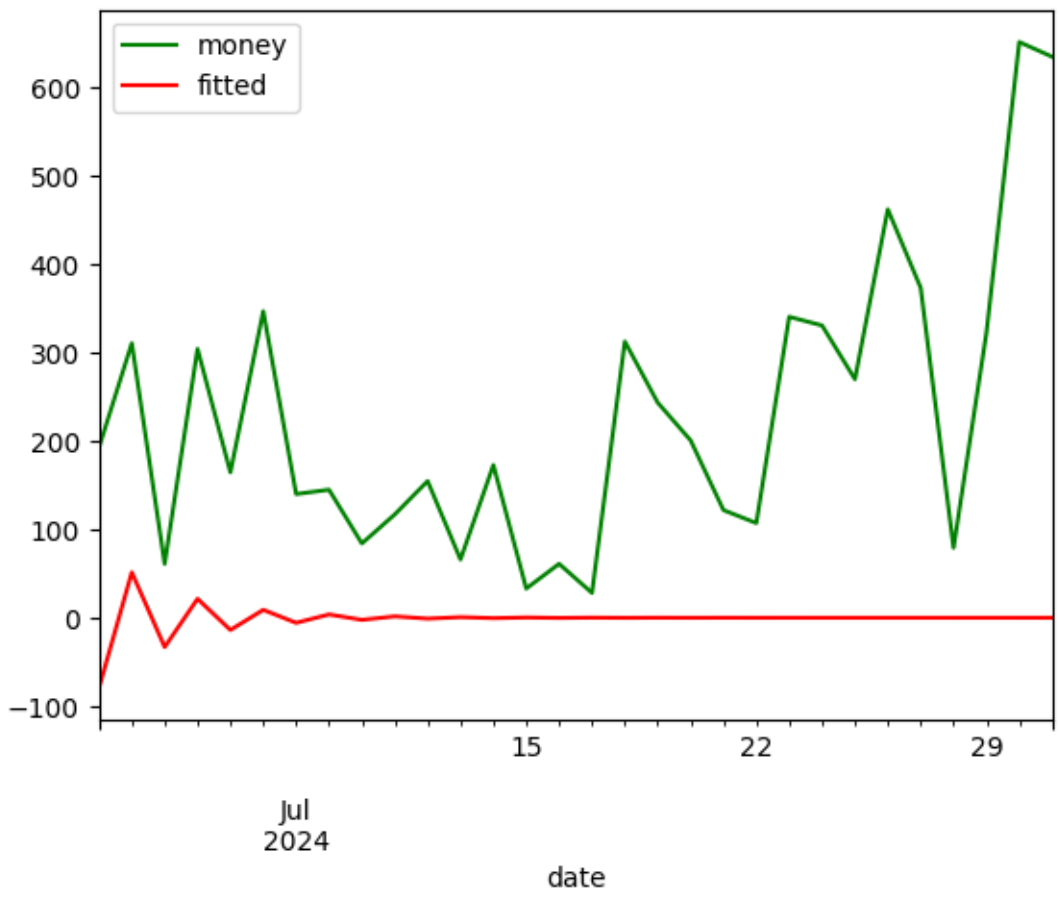
[ ]:
```
print(m1_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  money   No. Observations:                  121
Model:                 ARIMA(1, 1, 1)   Log Likelihood                -813.313
Date:                Sun, 01 Dec 2024   AIC                           1632.625
Time:                        12:02:59   BIC                           1640.988
Sample:                    03-03-2024   HQIC                          1636.021
                         - 07-01-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.6455      0.081     -7.963      0.000      -0.804      -0.487
ma.L1         -0.9995      6.259     -0.160      0.873     -13.267      11.268
sigma2      4.296e+04   2.69e+05      0.160      0.873   -4.83e+05    5.69e+05
==============================================================================
===
```

```
Ljung-Box (L1) (Q):                   14.56   Jarque-Bera (JB):
1.33
Prob(Q):                               0.00   Prob(JB):
0.51
Heteroskedasticity (H):                1.66   Skew:
-0.21
Prob(H) (two-sided):                   0.11   Kurtosis:
2.71
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[ ]:
```
train.plot(color='blue',label='train')
test.plot(color='green', label='test')
forcast_m1.plot(color='red',label='fitted')
plt.legend()
plt.show()
```





[ ]:
```
evaluate_model(test,forcast_m1)
```

```
MSE: 76963.24479402533, MAE: 229.04065139005567, RMSE: 277.4225023209641, R2:
-2.0490876397607716
```

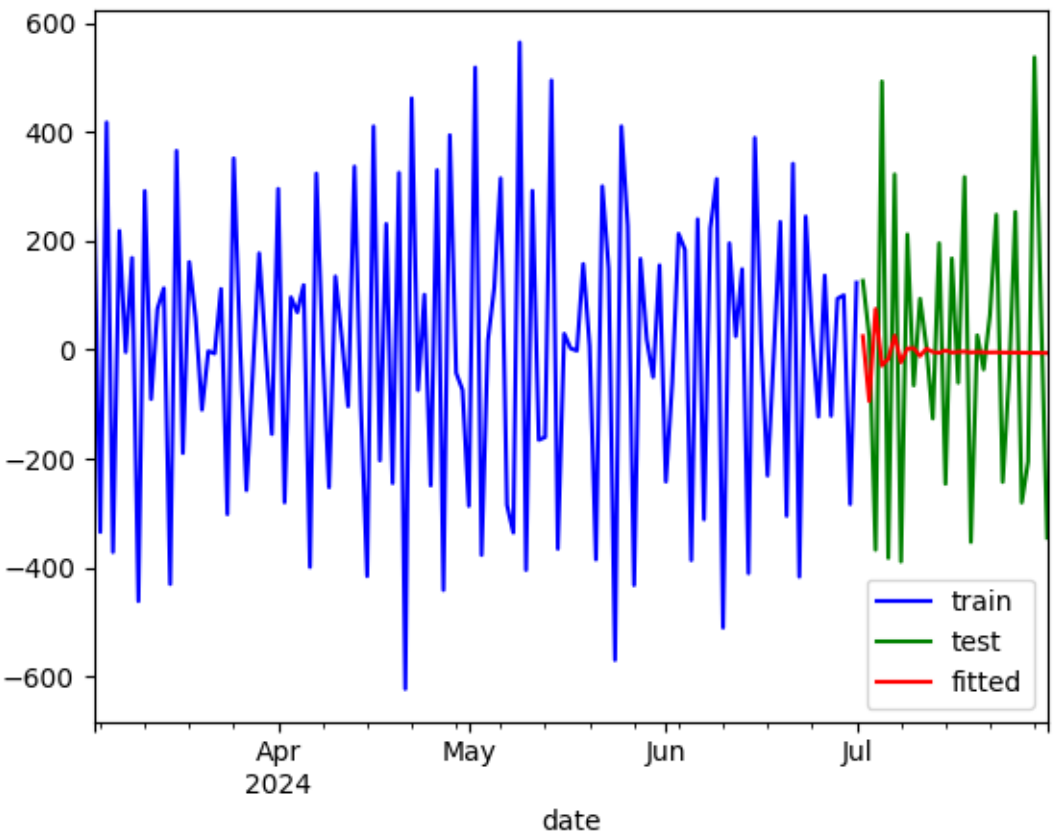
[ ]:
```
m2 = ARIMA(train, order=(2,2,2))
m2_fit = m2.fit()
```

[ ]:
```
forcast_m2 = m2_fit.forecast(30)
forcast_m2
```

[ ]:
```
2024-07-02    24.711237
2024-07-03   -94.480714
2024-07-04    74.732301
2024-07-05   -29.193998
2024-07-06   -16.884006
2024-07-07    25.734106
2024-07-08   -23.168840
2024-07-09     2.054939
```

```
2024-07-10     2.761927
2024-07-11   -11.683362
2024-07-12     1.808500
2024-07-13    -4.197515
2024-07-14    -5.804157
2024-07-15    -1.405388
2024-07-16    -5.258811
2024-07-17    -4.164181
2024-07-18    -3.569442
2024-07-19    -5.106387
2024-07-20    -4.277244
2024-07-21    -4.649467
2024-07-22    -5.092579
2024-07-23    -4.828639
2024-07-24    -5.223960
2024-07-25    -5.344533
2024-07-26    -5.386231
2024-07-27    -5.651389
2024-07-28    -5.738143
2024-07-29    -5.882209
2024-07-30    -6.064412
2024-07-31    -6.178630
Freq: D, Name: predicted_mean, dtype: float64
```

[ ]:
```
train.plot(color='blue',label='train')
test.plot(color='green', label='test')
forcast_m2.plot(color='red',label='fitted')
plt.legend()
plt.show()
```



5.1 2nd Approach

[ ]:
```
p = list(range(1,15))
d = [1,2]
q = list(range(1,15))
print(len(p))
print(len(d))
print(len(q))
print(14*2*14)
```

```
14
2
14
392
```

[ ]:
```
df1
```

[ ]:
```
              money
date
2024-03-01  396.30
```

```
2024-03-02  228.10
2024-03-03  349.10
2024-03-04  135.20
2024-03-05  338.50
…            …
2024-07-27  372.76
2024-07-28   78.86
2024-07-29  321.82
2024-07-30  650.48
2024-07-31  633.84

[153 rows x 1 columns]
```

[ ]:
```
train.shape
```

[ ]:
```
(121,)
```

[ ]:
```
import numpy as np

# Check the data type and shape
print(type(train))
print(train.shape)

# If train is a NumPy array, you can reshape it:
if isinstance(train, np.ndarray):
    # Reshape to a 2D array with 121 rows and 1 column
    train = train.reshape(-1, 1)
    print(train.shape)  # Should print (121, 1)

# If train is a Pandas Series or DataFrame:
# Convert to NumPy array and reshape as needed
if isinstance(train, pd.Series) or isinstance(train, pd.DataFrame):
    train_array = train.values.reshape(-1, 1)
    print(train_array.shape)  # Should print (121, 1)

# Now you can use train for evaluation, depending on your model's input shape␣
 ↪requirements.
```

```
<class 'pandas.core.series.Series'>
(121,)
(121, 1)
```

[ ]:
```
pdq_list = []
rmse_list = []
for i in p:
    for j in d:
        for k in q:
```

```
            model = ARIMA(np.log(train5),order=(i,j,k))
            model_fit = model.fit()
            fcast = np.exp(model_fit.forecast(len(test5)))
            rmse = np.sqrt(mean_squared_error(test5,fcast))
            print(f'p={i},d={j},q={k},RMSE={rmse}')
            rmse_list.append(rmse)
            pdq_list.append([i,j,k])
```

```
p=1,d=1,q=1,RMSE=103.47799301682322
p=1,d=1,q=2,RMSE=103.17574686208793
p=1,d=1,q=3,RMSE=106.48068390544883
p=1,d=1,q=4,RMSE=107.84522665833775
p=1,d=1,q=5,RMSE=107.89630082389522
p=1,d=1,q=6,RMSE=98.97026424911716
p=1,d=1,q=7,RMSE=102.72470637009512
p=1,d=1,q=8,RMSE=102.88509271188555
p=1,d=1,q=9,RMSE=104.88541049451875
p=1,d=1,q=10,RMSE=104.51765384403699
p=1,d=1,q=11,RMSE=104.76584578962768
p=1,d=1,q=12,RMSE=107.23793259914542
p=1,d=1,q=13,RMSE=96.21462227684769
p=1,d=1,q=14,RMSE=96.86843562072679
p=1,d=2,q=1,RMSE=119.82233127727527
p=1,d=2,q=2,RMSE=120.50219481309182
p=1,d=2,q=3,RMSE=121.44051336600555
p=1,d=2,q=4,RMSE=115.7305413586629
p=1,d=2,q=5,RMSE=114.51969677805849
p=1,d=2,q=6,RMSE=116.82328473908635
p=1,d=2,q=7,RMSE=119.06630176800772
p=1,d=2,q=8,RMSE=117.22840441329046
p=1,d=2,q=9,RMSE=117.31078629263253
p=1,d=2,q=10,RMSE=113.19581125434456
p=1,d=2,q=11,RMSE=120.91461464539668
p=1,d=2,q=12,RMSE=122.06138398125239
p=1,d=2,q=13,RMSE=121.00716873584909
p=1,d=2,q=14,RMSE=128.36446003448484
p=2,d=1,q=1,RMSE=103.05058241720502
p=2,d=1,q=2,RMSE=102.69537986270088
p=2,d=1,q=3,RMSE=107.31049666932337
p=2,d=1,q=4,RMSE=106.29006352180205
p=2,d=1,q=5,RMSE=103.20309184718327
p=2,d=1,q=6,RMSE=107.15954209427476
p=2,d=1,q=7,RMSE=102.78585235573695
p=2,d=1,q=8,RMSE=102.9769014708401
p=2,d=1,q=9,RMSE=105.32207821298536
p=2,d=1,q=10,RMSE=105.34078400846367
p=2,d=1,q=11,RMSE=104.57309562643677
```

p=2,d=1,q=12,RMSE=105.59715621514485
p=2,d=1,q=13,RMSE=107.10542341150907
p=2,d=1,q=14,RMSE=103.51323788990202
p=2,d=2,q=1,RMSE=119.76294264987735
p=2,d=2,q=2,RMSE=120.90212786563312
p=2,d=2,q=3,RMSE=124.23712914611582
p=2,d=2,q=4,RMSE=117.00832102931544
p=2,d=2,q=5,RMSE=115.80677568283693
p=2,d=2,q=6,RMSE=119.64650719786223
p=2,d=2,q=7,RMSE=117.1247408618275
p=2,d=2,q=8,RMSE=117.2063936671621
p=2,d=2,q=9,RMSE=118.07250141714064
p=2,d=2,q=10,RMSE=113.84485740191745
p=2,d=2,q=11,RMSE=114.50687815075466
p=2,d=2,q=12,RMSE=116.00669472308812
p=2,d=2,q=13,RMSE=122.72223809728497
p=2,d=2,q=14,RMSE=120.3393475424737
p=3,d=1,q=1,RMSE=109.40544422568429
p=3,d=1,q=2,RMSE=109.45800452042943
p=3,d=1,q=3,RMSE=108.31487751043561
p=3,d=1,q=4,RMSE=108.053354823647
p=3,d=1,q=5,RMSE=106.8752266891494
p=3,d=1,q=6,RMSE=107.55329107020928
p=3,d=1,q=7,RMSE=103.96587526834227
p=3,d=1,q=8,RMSE=108.60084921196507
p=3,d=1,q=9,RMSE=107.4973726669377
p=3,d=1,q=10,RMSE=106.01835414724795
p=3,d=1,q=11,RMSE=104.90377538870122
p=3,d=1,q=12,RMSE=106.61150204818914
p=3,d=1,q=13,RMSE=107.29051187180957
p=3,d=1,q=14,RMSE=102.93938932957263
p=3,d=2,q=1,RMSE=119.45602185625611
p=3,d=2,q=2,RMSE=119.38993017114798
p=3,d=2,q=3,RMSE=120.06889091919983
p=3,d=2,q=4,RMSE=119.6289272031346
p=3,d=2,q=5,RMSE=119.63113649790051
p=3,d=2,q=6,RMSE=115.75073041870235
p=3,d=2,q=7,RMSE=116.66884962373743
p=3,d=2,q=8,RMSE=117.63693301618015
p=3,d=2,q=9,RMSE=118.29443522336449
p=3,d=2,q=10,RMSE=116.32443839534815
p=3,d=2,q=11,RMSE=117.67822752040384
p=3,d=2,q=12,RMSE=117.94451037286366
p=3,d=2,q=13,RMSE=115.78524712677452
p=3,d=2,q=14,RMSE=118.52682269394117
p=4,d=1,q=1,RMSE=109.40701895797089
p=4,d=1,q=2,RMSE=99.32130135065837
p=4,d=1,q=3,RMSE=101.00104012513113

p=4,d=1,q=4,RMSE=105.5339203429448
p=4,d=1,q=5,RMSE=107.17548333757028
p=4,d=1,q=6,RMSE=107.92904122947012
p=4,d=1,q=7,RMSE=107.83338690485557
p=4,d=1,q=8,RMSE=108.38624051340207
p=4,d=1,q=9,RMSE=103.83652542683598
p=4,d=1,q=10,RMSE=105.67423587750342
p=4,d=1,q=11,RMSE=105.89466500586074
p=4,d=1,q=12,RMSE=107.27426824926316
p=4,d=1,q=13,RMSE=107.51618736541803
p=4,d=1,q=14,RMSE=97.36254179503356
p=4,d=2,q=1,RMSE=119.4193437532332
p=4,d=2,q=2,RMSE=119.53496430589804
p=4,d=2,q=3,RMSE=120.33212216268568
p=4,d=2,q=4,RMSE=120.2778312683033
p=4,d=2,q=5,RMSE=119.43790370270143
p=4,d=2,q=6,RMSE=116.84827333296415
p=4,d=2,q=7,RMSE=117.36185518947015
p=4,d=2,q=8,RMSE=117.1143401860871
p=4,d=2,q=9,RMSE=117.10386245559847
p=4,d=2,q=10,RMSE=118.46115582189377
p=4,d=2,q=11,RMSE=117.79443250387365
p=4,d=2,q=12,RMSE=117.29746250266258
p=4,d=2,q=13,RMSE=117.36817142576531
p=4,d=2,q=14,RMSE=110.6774285768455
p=5,d=1,q=1,RMSE=99.75805878273313
p=5,d=1,q=2,RMSE=111.85116638158665
p=5,d=1,q=3,RMSE=109.74312617763283
p=5,d=1,q=4,RMSE=101.66302834685341
p=5,d=1,q=5,RMSE=106.98100601630676
p=5,d=1,q=6,RMSE=108.7425450187092
p=5,d=1,q=7,RMSE=107.72304033538728
p=5,d=1,q=8,RMSE=108.45034231570703
p=5,d=1,q=9,RMSE=101.14889780237358
p=5,d=1,q=10,RMSE=105.18405841502899
p=5,d=1,q=11,RMSE=105.38114741493735
p=5,d=1,q=12,RMSE=117.91353363637519
p=5,d=1,q=13,RMSE=115.59172051406685
p=5,d=1,q=14,RMSE=108.8800064678609
p=5,d=2,q=1,RMSE=119.39982604123132
p=5,d=2,q=2,RMSE=119.39033168743248
p=5,d=2,q=3,RMSE=119.06404707216178
p=5,d=2,q=4,RMSE=120.65482897898171
p=5,d=2,q=5,RMSE=120.57654178675607
p=5,d=2,q=6,RMSE=118.90357305504715
p=5,d=2,q=7,RMSE=117.64929527244341
p=5,d=2,q=8,RMSE=116.66072485850712
p=5,d=2,q=9,RMSE=116.12194572642923

p=5,d=2,q=10,RMSE=117.72069482129145
p=5,d=2,q=11,RMSE=117.8107623186981
p=5,d=2,q=12,RMSE=115.35377662683155
p=5,d=2,q=13,RMSE=119.7001270113119
p=5,d=2,q=14,RMSE=115.31760356843131
p=6,d=1,q=1,RMSE=107.63414492817282
p=6,d=1,q=2,RMSE=107.3484967479291
p=6,d=1,q=3,RMSE=99.20381900458804
p=6,d=1,q=4,RMSE=100.2796868881698
p=6,d=1,q=5,RMSE=109.71795007832732
p=6,d=1,q=6,RMSE=107.14816187735919
p=6,d=1,q=7,RMSE=106.33357340694727
p=6,d=1,q=8,RMSE=108.29470028682698
p=6,d=1,q=9,RMSE=105.55925872585998
p=6,d=1,q=10,RMSE=105.36449817466502
p=6,d=1,q=11,RMSE=104.29965667715861
p=6,d=1,q=12,RMSE=113.99984493929658
p=6,d=1,q=13,RMSE=115.92090862309578
p=6,d=1,q=14,RMSE=113.58343039348455
p=6,d=2,q=1,RMSE=118.96503720176602
p=6,d=2,q=2,RMSE=119.35026954930156
p=6,d=2,q=3,RMSE=120.0948667384398
p=6,d=2,q=4,RMSE=120.52062945492524
p=6,d=2,q=5,RMSE=119.29577086652799
p=6,d=2,q=6,RMSE=120.1098364723762
p=6,d=2,q=7,RMSE=120.85309524595188
p=6,d=2,q=8,RMSE=117.91462712201678
p=6,d=2,q=9,RMSE=117.7884093389665
p=6,d=2,q=10,RMSE=118.16999944777736
p=6,d=2,q=11,RMSE=121.43054541423416
p=6,d=2,q=12,RMSE=120.10286755382039
p=6,d=2,q=13,RMSE=118.73466799848897
p=6,d=2,q=14,RMSE=117.62562774434733
p=7,d=1,q=1,RMSE=109.50879364945078
p=7,d=1,q=2,RMSE=109.26875194129677
p=7,d=1,q=3,RMSE=99.83424114961848
p=7,d=1,q=4,RMSE=99.3343200246926
p=7,d=1,q=5,RMSE=100.6246573938702
p=7,d=1,q=6,RMSE=105.5963241192096
p=7,d=1,q=7,RMSE=104.19850749643051
p=7,d=1,q=8,RMSE=109.04209184255076
p=7,d=1,q=9,RMSE=104.80925943115624
p=7,d=1,q=10,RMSE=100.59738880380273
p=7,d=1,q=11,RMSE=103.34188467167426
p=7,d=1,q=12,RMSE=108.43528162646454
p=7,d=1,q=13,RMSE=104.62245264606648
p=7,d=1,q=14,RMSE=108.4003118013945
p=7,d=2,q=1,RMSE=118.72511040067198

p=7,d=2,q=2,RMSE=119.7386767535364
p=7,d=2,q=3,RMSE=120.00820217446673
p=7,d=2,q=4,RMSE=120.3928255118533
p=7,d=2,q=5,RMSE=121.20045275935722
p=7,d=2,q=6,RMSE=120.2551856754916
p=7,d=2,q=7,RMSE=120.59368734081508
p=7,d=2,q=8,RMSE=120.85835476874206
p=7,d=2,q=9,RMSE=120.17250165579239
p=7,d=2,q=10,RMSE=120.95794519677409
p=7,d=2,q=11,RMSE=120.74874329724493
p=7,d=2,q=12,RMSE=119.91883897012524
p=7,d=2,q=13,RMSE=118.62095935196187
p=7,d=2,q=14,RMSE=121.74382184091873
p=8,d=1,q=1,RMSE=110.11267705919957
p=8,d=1,q=2,RMSE=109.96981647153184
p=8,d=1,q=3,RMSE=98.87075608623088
p=8,d=1,q=4,RMSE=99.76960342810749
p=8,d=1,q=5,RMSE=100.49737092264608
p=8,d=1,q=6,RMSE=105.68644530851739
p=8,d=1,q=7,RMSE=104.55491104997196
p=8,d=1,q=8,RMSE=107.67587368310326
p=8,d=1,q=9,RMSE=103.99612299072122
p=8,d=1,q=10,RMSE=102.05371611322617
p=8,d=1,q=11,RMSE=103.46635015322099
p=8,d=1,q=12,RMSE=103.00997389144342
p=8,d=1,q=13,RMSE=102.12142430631985
p=8,d=1,q=14,RMSE=104.83289455419099
p=8,d=2,q=1,RMSE=120.15280714289796
p=8,d=2,q=2,RMSE=120.26982111632232
p=8,d=2,q=3,RMSE=119.81315081557199
p=8,d=2,q=4,RMSE=120.64876545069812
p=8,d=2,q=5,RMSE=120.73718455972508
p=8,d=2,q=6,RMSE=121.176851558917
p=8,d=2,q=7,RMSE=117.94063222559558
p=8,d=2,q=8,RMSE=117.37234869452725
p=8,d=2,q=9,RMSE=118.37643110004993
p=8,d=2,q=10,RMSE=121.0496650887946
p=8,d=2,q=11,RMSE=121.2068707522405
p=8,d=2,q=12,RMSE=120.18905619147971
p=8,d=2,q=13,RMSE=118.27196714788762
p=8,d=2,q=14,RMSE=118.43256055377188
p=9,d=1,q=1,RMSE=106.294636908982
p=9,d=1,q=2,RMSE=107.53554886112408
p=9,d=1,q=3,RMSE=111.14210498445095
p=9,d=1,q=4,RMSE=109.42026072061375
p=9,d=1,q=5,RMSE=110.37338584781719
p=9,d=1,q=6,RMSE=115.79952881008168
p=9,d=1,q=7,RMSE=114.74284493022043

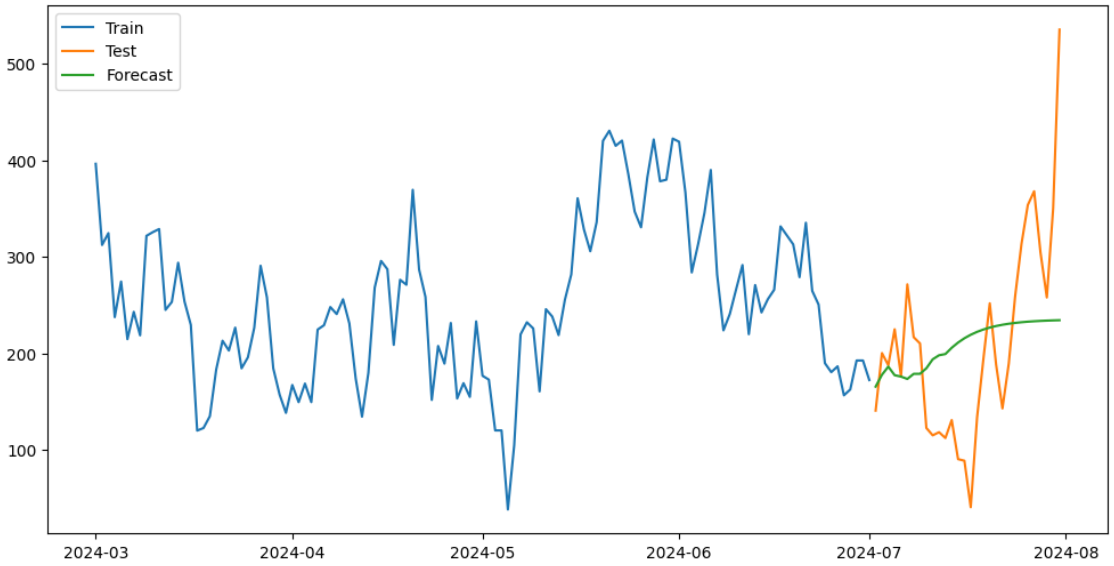
p=9,d=1,q=8,RMSE=114.822626322062
p=9,d=1,q=9,RMSE=111.18986027968667
p=9,d=1,q=10,RMSE=115.1341362034233
p=9,d=1,q=11,RMSE=114.38560198776564
p=9,d=1,q=12,RMSE=113.06125676473087
p=9,d=1,q=13,RMSE=117.9620382357481
p=9,d=1,q=14,RMSE=116.64358186316056
p=9,d=2,q=1,RMSE=116.87964063067035
p=9,d=2,q=2,RMSE=117.13271295993346
p=9,d=2,q=3,RMSE=119.78881043657874
p=9,d=2,q=4,RMSE=119.74344308446622
p=9,d=2,q=5,RMSE=120.78269273454332
p=9,d=2,q=6,RMSE=120.92270314193631
p=9,d=2,q=7,RMSE=117.70679595403598
p=9,d=2,q=8,RMSE=105.25448273275092
p=9,d=2,q=9,RMSE=120.78707728093923
p=9,d=2,q=10,RMSE=120.33393636433013
p=9,d=2,q=11,RMSE=119.83651632338197
p=9,d=2,q=12,RMSE=118.12901839657398
p=9,d=2,q=13,RMSE=118.95826590941894
p=9,d=2,q=14,RMSE=118.34901572245934
p=10,d=1,q=1,RMSE=106.08182160020287
p=10,d=1,q=2,RMSE=109.51223405054827
p=10,d=1,q=3,RMSE=108.8641287125037
p=10,d=1,q=4,RMSE=112.98561309086222
p=10,d=1,q=5,RMSE=112.55938921937965
p=10,d=1,q=6,RMSE=110.2201973263621
p=10,d=1,q=7,RMSE=110.48001922750709
p=10,d=1,q=8,RMSE=114.370683436412
p=10,d=1,q=9,RMSE=109.51225285067864
p=10,d=1,q=10,RMSE=116.05270285680082
p=10,d=1,q=11,RMSE=115.40216368433379
p=10,d=1,q=12,RMSE=117.58078679414766
p=10,d=1,q=13,RMSE=112.63338252356245
p=10,d=1,q=14,RMSE=119.9544971976159
p=10,d=2,q=1,RMSE=116.49878220532803
p=10,d=2,q=2,RMSE=116.34515587601689
p=10,d=2,q=3,RMSE=115.16919709742692
p=10,d=2,q=4,RMSE=117.25521780705387
p=10,d=2,q=5,RMSE=119.49311624676538
p=10,d=2,q=6,RMSE=120.23561081687278
p=10,d=2,q=7,RMSE=120.19457867618672
p=10,d=2,q=8,RMSE=119.60838380163777
p=10,d=2,q=9,RMSE=124.17977310030307
p=10,d=2,q=10,RMSE=119.38462830275257
p=10,d=2,q=11,RMSE=127.47003487775815
p=10,d=2,q=12,RMSE=119.26722398340297
p=10,d=2,q=13,RMSE=126.91701358322965

p=10,d=2,q=14,RMSE=112.64643788893487
p=11,d=1,q=1,RMSE=110.23677426953361
p=11,d=1,q=2,RMSE=107.71000674058742
p=11,d=1,q=3,RMSE=112.24276100464006
p=11,d=1,q=4,RMSE=113.06497420537424
p=11,d=1,q=5,RMSE=110.80885485513747
p=11,d=1,q=6,RMSE=114.39513986868775
p=11,d=1,q=7,RMSE=113.43722154089762
p=11,d=1,q=8,RMSE=108.74925172192742
p=11,d=1,q=9,RMSE=109.70471367429222
p=11,d=1,q=10,RMSE=116.40004447617349
p=11,d=1,q=11,RMSE=116.42715277278917
p=11,d=1,q=12,RMSE=116.98030562262089
p=11,d=1,q=13,RMSE=114.21639823702367
p=11,d=1,q=14,RMSE=123.188601024936
p=11,d=2,q=1,RMSE=167.98274903881034
p=11,d=2,q=2,RMSE=114.8310985068505
p=11,d=2,q=3,RMSE=148.8033434399573
p=11,d=2,q=4,RMSE=123.93083161511169
p=11,d=2,q=5,RMSE=116.90328703462079
p=11,d=2,q=6,RMSE=119.12341711757331
p=11,d=2,q=7,RMSE=122.58560352901648
p=11,d=2,q=8,RMSE=121.79670749758294
p=11,d=2,q=9,RMSE=121.19557537785018
p=11,d=2,q=10,RMSE=120.90050844614524
p=11,d=2,q=11,RMSE=126.94041496386778
p=11,d=2,q=12,RMSE=125.65019441468087
p=11,d=2,q=13,RMSE=126.35041450930797
p=11,d=2,q=14,RMSE=125.079374061405
p=12,d=1,q=1,RMSE=112.17514518463952
p=12,d=1,q=2,RMSE=111.48819174059054
p=12,d=1,q=3,RMSE=112.47655736822597
p=12,d=1,q=4,RMSE=115.51491548191325
p=12,d=1,q=5,RMSE=108.68563645153219
p=12,d=1,q=6,RMSE=115.39377268693042
p=12,d=1,q=7,RMSE=116.30882625542822
p=12,d=1,q=8,RMSE=115.78017008098514
p=12,d=1,q=9,RMSE=106.41296704627884
p=12,d=1,q=10,RMSE=115.70389251829221
p=12,d=1,q=11,RMSE=102.6010856836599
p=12,d=1,q=12,RMSE=105.23065397591965
p=12,d=1,q=13,RMSE=102.65128152334938
p=12,d=1,q=14,RMSE=103.81884591421489
p=12,d=2,q=1,RMSE=119.48531120891802
p=12,d=2,q=2,RMSE=160.45096066099006
p=12,d=2,q=3,RMSE=113.90708736925366
p=12,d=2,q=4,RMSE=115.85169183771991
p=12,d=2,q=5,RMSE=117.40377071300904

p=12,d=2,q=6,RMSE=117.90018705837058
p=12,d=2,q=7,RMSE=118.93951168029291
p=12,d=2,q=8,RMSE=119.52206492940483
p=12,d=2,q=9,RMSE=120.46979341497385
p=12,d=2,q=10,RMSE=121.83376387866811
p=12,d=2,q=11,RMSE=127.02845023617918
p=12,d=2,q=12,RMSE=129.2053533269082
p=12,d=2,q=13,RMSE=125.61186836794163
p=12,d=2,q=14,RMSE=124.5590639090723
p=13,d=1,q=1,RMSE=112.10938496010824
p=13,d=1,q=2,RMSE=109.72178913680604
p=13,d=1,q=3,RMSE=111.23857487938132
p=13,d=1,q=4,RMSE=110.65349630621984
p=13,d=1,q=5,RMSE=117.23464654932364
p=13,d=1,q=6,RMSE=118.46264979171136
p=13,d=1,q=7,RMSE=116.50905322758071
p=13,d=1,q=8,RMSE=117.40416989549321
p=13,d=1,q=9,RMSE=116.5311340521918
p=13,d=1,q=10,RMSE=115.42014739784925
p=13,d=1,q=11,RMSE=99.72695955954306
p=13,d=1,q=12,RMSE=103.66225446484171
p=13,d=1,q=13,RMSE=105.49907573121683
p=13,d=1,q=14,RMSE=105.17499597606508
p=13,d=2,q=1,RMSE=125.76977149178663
p=13,d=2,q=2,RMSE=120.51220815375127
p=13,d=2,q=3,RMSE=118.86996466500372
p=13,d=2,q=4,RMSE=115.37808329622565
p=13,d=2,q=5,RMSE=114.99545800297773
p=13,d=2,q=6,RMSE=123.32566448848934
p=13,d=2,q=7,RMSE=121.8196055824996
p=13,d=2,q=8,RMSE=126.64843799919299
p=13,d=2,q=9,RMSE=124.72118905589656
p=13,d=2,q=10,RMSE=120.60463391603209
p=13,d=2,q=11,RMSE=124.0091025233612
p=13,d=2,q=12,RMSE=128.57843008274102
p=13,d=2,q=13,RMSE=127.85336731457747
p=13,d=2,q=14,RMSE=128.247971333683
p=14,d=1,q=1,RMSE=111.55619943642616
p=14,d=1,q=2,RMSE=110.89004154536075
p=14,d=1,q=3,RMSE=112.59457267032694
p=14,d=1,q=4,RMSE=113.06081390573078
p=14,d=1,q=5,RMSE=114.9314716788465
p=14,d=1,q=6,RMSE=113.40380160572269
p=14,d=1,q=7,RMSE=115.36378170451692
p=14,d=1,q=8,RMSE=116.23566896952477
p=14,d=1,q=9,RMSE=115.79478582472154
p=14,d=1,q=10,RMSE=115.24214702150265
p=14,d=1,q=11,RMSE=100.51522343583284

```
p=14,d=1,q=12,RMSE=103.03631384953843
p=14,d=1,q=13,RMSE=106.83282677161031
p=14,d=1,q=14,RMSE=107.8064422159907
p=14,d=2,q=1,RMSE=124.5838659858703
p=14,d=2,q=2,RMSE=124.88096977445039
p=14,d=2,q=3,RMSE=124.17822927638754
p=14,d=2,q=4,RMSE=122.0319374415645
p=14,d=2,q=5,RMSE=117.04671647162756
p=14,d=2,q=6,RMSE=127.04861797875591
p=14,d=2,q=7,RMSE=128.44056011752292
p=14,d=2,q=8,RMSE=130.47085394663114
p=14,d=2,q=9,RMSE=123.51021374690107
p=14,d=2,q=10,RMSE=125.04372547612907
p=14,d=2,q=11,RMSE=125.23798585006749
p=14,d=2,q=12,RMSE=126.48439085693431
p=14,d=2,q=13,RMSE=129.29427211682335
p=14,d=2,q=14,RMSE=131.1542990037607
```

[ ]:
```
custom_arima(train5,test5,1,1,13)
```



[ ]:
```
                          MAE          MSE       RMSE
ARIMA((1, 1, 13))  75.370692  9257.25354  96.214622
```

[ ]:
```
sm_data.head()
```

[ ]:
```
date
2024-03-01    396.300000
2024-03-02    312.200000
```

```
2024-03-03    324.500000
2024-03-04    237.466667
2024-03-05    274.266667
Freq: D, Name: money_1, dtype: float64
```

The result states that this (1,1,13) parameter best suited for the ARIMA model here