

Dataset Link: <https://www.kaggle.com/datasets/tan5577/coffee-sales-csv>

Import libraries

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Upload Dataset to Google Colap

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))
```

Choose FilesCoffe_sales.csv

Coffe_sales.csv(text/csv) - 228608 bytes, last modified: 11/8/2025 - 100% done

Saving Coffe_sales.csv to Coffe_sales (1).csv

User uploaded file 'Coffe_sales (1).csv' with length 228608 bytes

Read the file content

```
df= pd.read_csv('Coffe_sales.csv')
df.head()
```

	hour_of_day	cash_type	money	coffee_name	Time_of_Day	Weekday	Month_name	Weekdaysort	Monthsort	Date	Time	
0	10	card	38.7	Latte	Morning	Fri	Mar	5	3	3/1/2024	15:50.5	
1	12	card	38.7	Hot Chocolate	Afternoon	Fri	Mar	5	3	3/1/2024	19:22.5	
2	12	card	38.7	Hot Chocolate	Afternoon	Fri	Mar	5	3	3/1/2024	20:18.1	
3	13	card	28.9	Americano	Afternoon	Fri	Mar	5	3	3/1/2024	46:33.0	
4	13	card	38.7	Latte	Afternoon	Fri	Mar	5	3	3/1/2024	48:14.6	

Next steps: [Generate code with df](#) [New interactive sheet](#)

df.dtypes

```

hour_of_day    int64
cash_type      object
money          float64
coffee_name    object
Time_of_Day    object
Weekday        object
Month_name     object
Weekdaysort   int64
Monthsort      int64
Date          object
Time          object
dtype: object
```

df.columns

```
Index(['hour_of_day', 'cash_type', 'money', 'coffee_name', 'Time_of_Day',
       'Weekday', 'Month_name', 'Weekdaysort', 'Monthsort', 'Date', 'Time'],
      dtype='object')
```

Getting the count of missing values

```
df.isnull().sum()

hour_of_day    0
cash_type      0
money          0
coffee_name    0
Time_of_Day    0
Weekday        0
Month_name     0
Weekdaysort   0
Monthsort      0
Date          0
Time          0
dtype: int64
```

Deleting columns that have 30% missing values

```
missing_percentage =df.isnull().mean()*100
Threshold =30

columns_to_drop = missing_percentage[missing_percentage > Threshold].index

df=df.drop(columns=columns_to_drop)

print(f"Columns dropped:{list(columns_to_drop)}")
print (f"Remaining columns: {df.columns}")

Columns dropped:[]
Remaining columns: Index(['hour_of_day', 'cash_type', 'money', 'coffee_name', 'Time_of_Day',
       'Weekday', 'Month_name', 'Weekdaysort', 'Monthsort', 'Date', 'Time'],
      dtype='object')
```

Detect Outliers

```
data=df.copy()
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Return a boolean series indicating if the value is an outlier
    return (column < lower_bound) | (column > upper_bound)

# Apply the IQR method to each numeric feature in the DataFrame
numeric_cols = data.select_dtypes(include=np.number).columns
for col in numeric_cols:
    data[f'{col}_Outlier'] = detect_outliers_iqr(data[col])

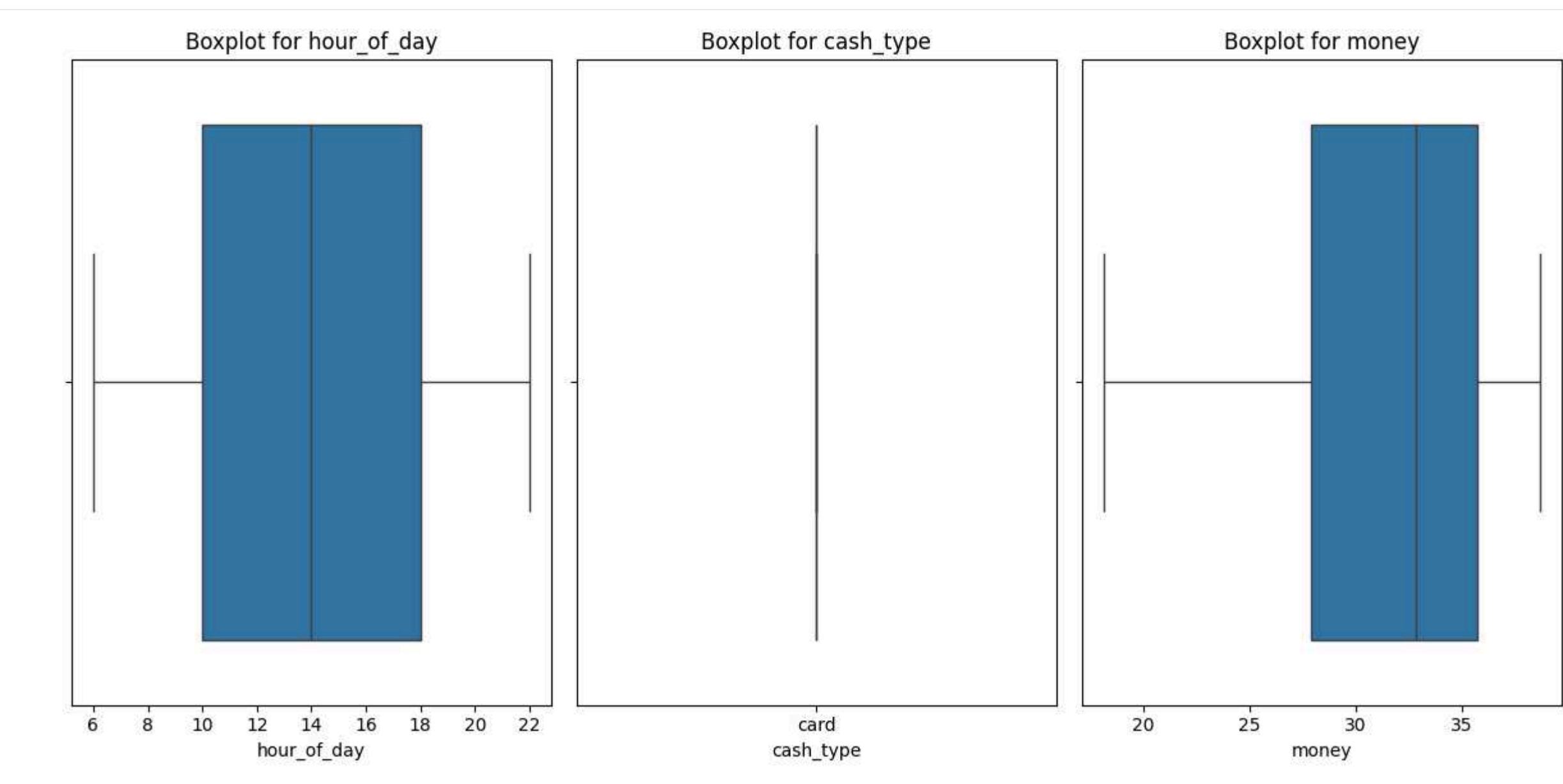
# Filter the DataFrame to display rows where any outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]

# Display the filtered DataFrame with outliers
outliers_only
```

	hour_of_day	cash_type	money	coffee_name	Time_of_Day	Weekday	Month_name	Weekdaysort	Monthsort	Date	Time	hour_of_day_Outlier	money_Outlier	Weekdaysort_Outlier	Monthsort_Outlier	
--	-------------	-----------	-------	-------------	-------------	---------	------------	-------------	-----------	------	------	---------------------	---------------	---------------------	-------------------	--

Visualize Outliers

```
plt.figure(figsize=(12, 6))
for i, col in enumerate(df.columns[:3]): # Only plotting the first 3 features
    plt.subplot(1, 3, i+1) # Create subplots for each feature
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot for {col}')
plt.tight_layout()
plt.show()
```



Using Z-Score

```
# Function to detect outliers in a column using Z-Score
data=df.copy()
def detect_outliers_zscore(column, threshold=3):
    mean = np.mean(column)
    std_dev = np.std(column)
    z_scores = (column - mean) / std_dev
    # Return a boolean series where True indicates an outlier
    return np.abs(z_scores) > threshold
# Apply the Z-Score method to each feature in the DataFrame
numeric_cols = data.select_dtypes(include=np.number).columns
for col in numeric_cols:
    data[f'{col}_Outlier'] = detect_outliers_zscore(data[col])
# Filter the DataFrame to display rows where any outlier flag is True
outliers_only = data[data.filter(like='_Outlier').any(axis=1)]
outliers_only
```

hour_of_day	cash_type	money	coffee_name	Time_of_Day	Weekday	Month_name	Weekdaysort	Monthsort	Date	Time	hour_of_day_Outlier	money_Outlier	Weekdaysort_Outlier	Monthsort_Outlier
-------------	-----------	-------	-------------	-------------	---------	------------	-------------	-----------	------	------	---------------------	---------------	---------------------	-------------------

Handling Outliers

```
data=df.copy()
def replace_outliers_with_nan(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Replace outliers with NaN
    return column.where((column >= lower_bound) & (column <=
upper_bound), np.nan)
# Replace outliers with NaN for each feature
numeric_cols = data.select_dtypes(include=np.number).columns
for col in numeric_cols:
    data[col] = replace_outliers_with_nan(data[col])
null_counts = data.isnull().sum()
print("Sum of null values in each column:")
print(null_counts)
```

Sum of null values in each column:

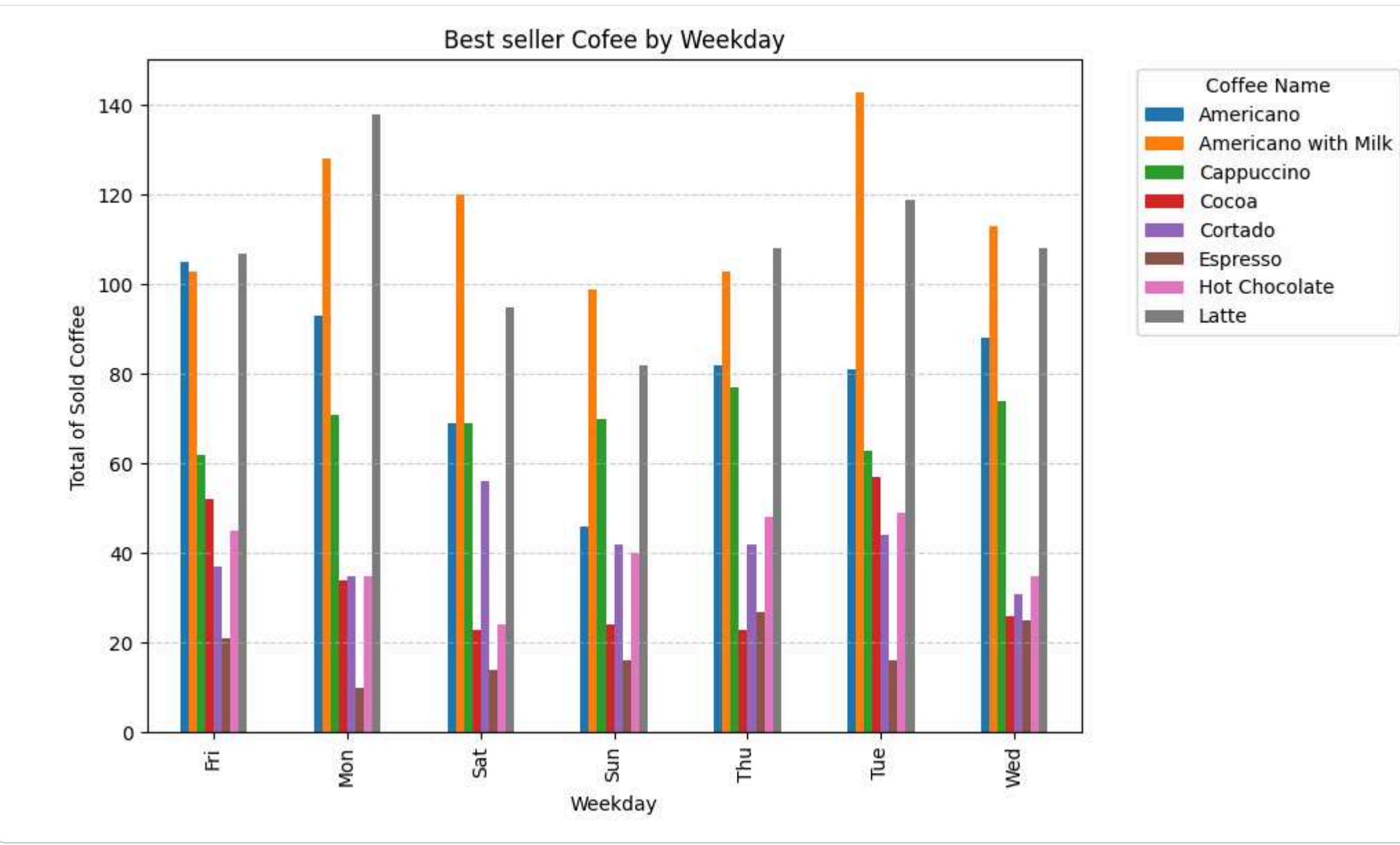
hour_of_day	0
cash_type	0
money	0
coffee_name	0
Time_of_Day	0
Weekday	0
Month_name	0
Weekdaysort	0
Monthsort	0
Date	0
Time	0
dtype:	int64

Data Visualization

```
coffee_sales = df.groupby(['Weekday', 'coffee_name']).size().unstack(fill_value=0)

# Plot
coffee_sales.plot(kind='bar', figsize=(10, 6))

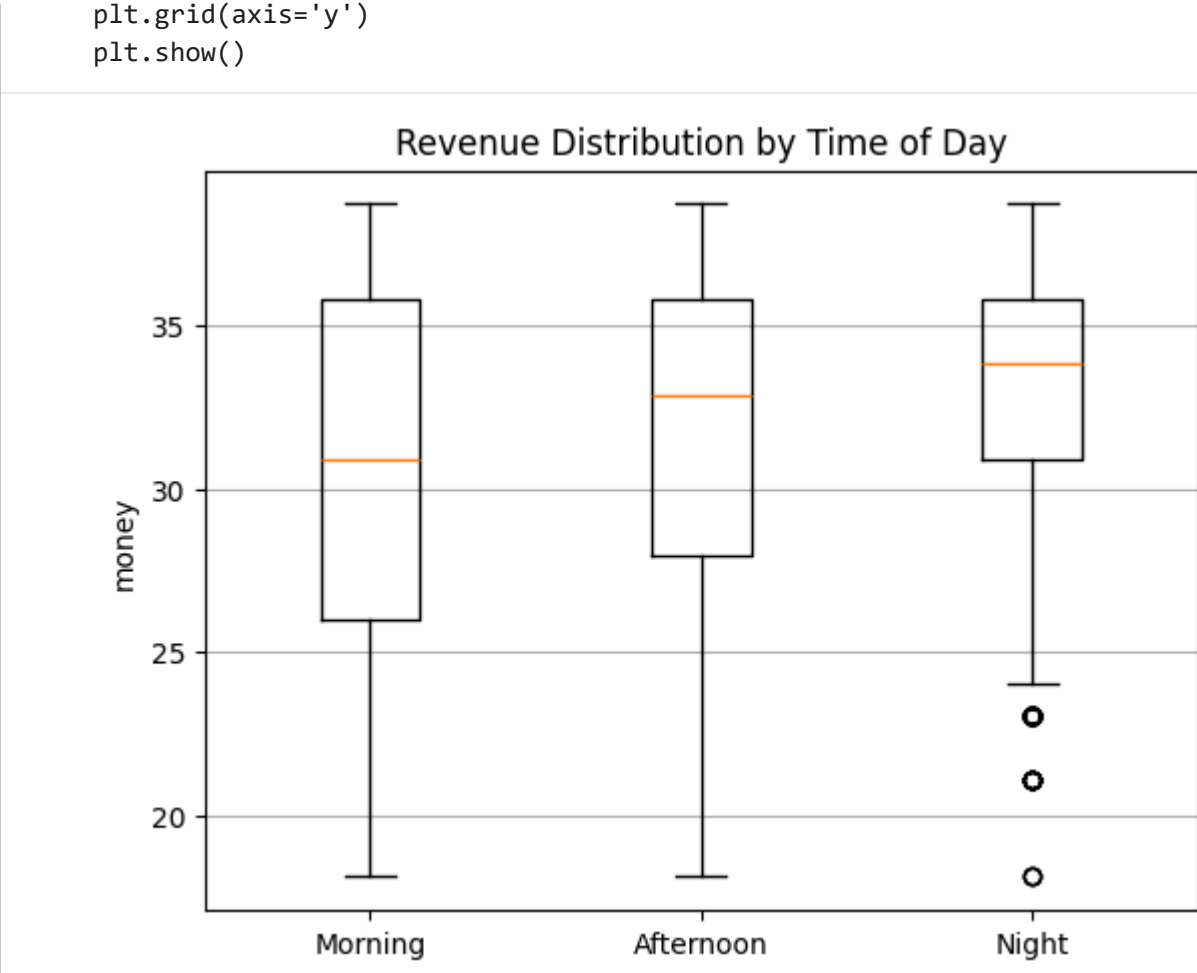
plt.title('Best seller Cofee by Weekday')
plt.xlabel('Weekday')
plt.ylabel('Total of Sold Coffee')
plt.legend(title='Coffee Name', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Boxplot

```
# Boxplot: Comparing revenue (money) across Time of Day
plt.boxplot(
    [df[df['Time_of_Day'] == 'Morning']['money'].dropna(),
    df[df['Time_of_Day'] == 'Afternoon']['money'].dropna(),
    df[df['Time_of_Day'] == 'Night']['money'].dropna()],
    labels=['Morning', 'Afternoon', 'Night'])
)

plt.title('Revenue Distribution by Time of Day')
plt.ylabel('money')
```



Scatter plot

```
# Scatter plot: Hour of the Day vs. Money Spent
plt.scatter(df['hour_of_day'], df['money'], color='teal', alpha=0.7)

plt.title('Coffee Sales Distribution by Hour of the Day')
plt.xlabel('Hour of Day')
plt.ylabel('Amount Spent')
plt.grid(True)
plt.show()
```



```
unique_coffees = df['coffee_name'].nunique()
print("Unique coffee types:", unique_coffees)

# Total number of items sold
total_items = df['coffee_name'].count()
print("Total items sold:", total_items)

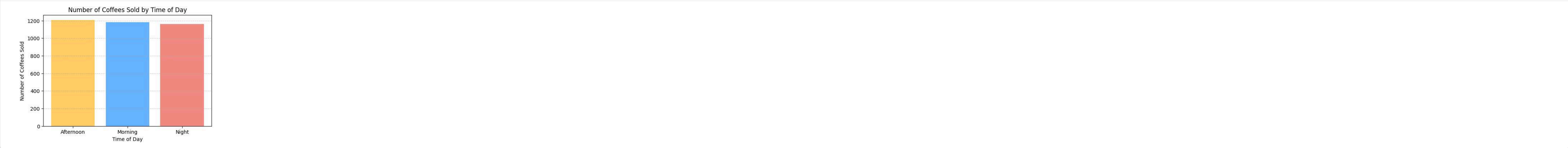
Unique coffee types: 8
Total items sold: 3547
```

```
time_sales = df['Time_of_Day'].value_counts()

# Plot bar chart
plt.figure(figsize=(6, 4))
bars = plt.bar(time_sales.index, time_sales.values, color=['#ffcc66', '#66b3ff', '#f28b82'])

plt.title('Number of Coffees Sold by Time of Day')
plt.xlabel('Time of Day')
plt.ylabel('Number of Coffees Sold')
plt.grid(axis='y', linestyle='--', alpha=0.6)

plt.show()
```



Best Seller Coffee by time of the Day

```
coffee_sales = df.groupby(['Time_of_Day', 'coffee_name']).size().unstack(fill_value=0)

# Plot
coffee_sales.plot(kind='bar', figsize=(10, 6))

plt.title('Best seller Coffee by time of the Day')
plt.xlabel('Time of Day')
plt.ylabel('Total of Sold Coffee')
plt.legend(title='Coffee Name', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



```
# Group by month and coffee, count sales
monthly_sales = df.groupby(['Month_name', 'coffee_name']).size().reset_index(name='sales')

# Get the best-selling coffee for each month
best_sellers = monthly_sales.loc[monthly_sales.groupby('Month_name')['sales'].idxmax()]

# Sort months (optional, if you want calendar order)
month_order = df['Month_name']
best_sellers = best_sellers.set_index('Month_name').reindex(month_order).dropna().reset_index()

# Plot
plt.figure(figsize=(10, 6))
plt.bar(best_sellers['Month_name'], best_sellers['sales'], color='skyblue')

# Add labels
for i, val in enumerate(best_sellers['sales']):
    plt.text(i, val + 0.5, f'{best_sellers["coffee_name"][i]}\n{(int(val))}',
             ha='center', va='bottom', fontsize=9)

plt.title('Best-Selling Coffee per Month')
plt.xlabel('Month')
plt.ylabel('Number of Items Sold')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

Exploded (Highlighted) Pie Chart

```
coffee_total = df['coffee_name'].value_counts()

# Plot pie chart
plt.figure(figsize=(8, 8))
plt.pie(
    coffee_total.values,
    labels=coffee_total.index,
    autopct='%1.1f%%',
    startangle=90,
)

plt.title('Total Best-Selling Coffees')
plt.show()
```

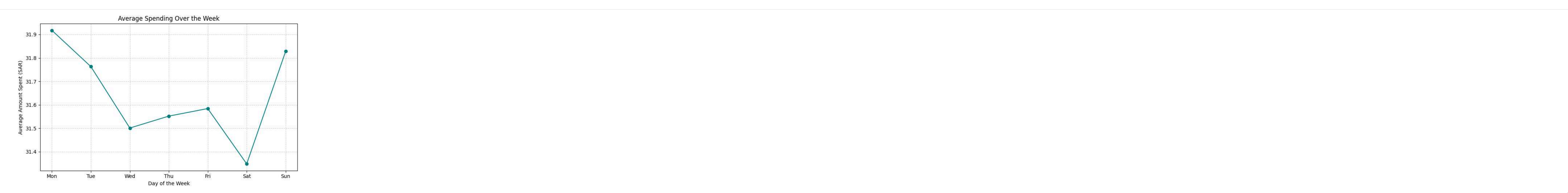


```
import matplotlib.pyplot as plt

# Group data by weekday and calculate average money spent
avg_spent = df.groupby('Weekday')['money'].mean().reindex(
    ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(avg_spent.index, avg_spent.values, marker='o', color='teal', linestyle='--')

# Labels and Title
plt.title('Average Spending Over the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Average Amount Spent (SAR)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



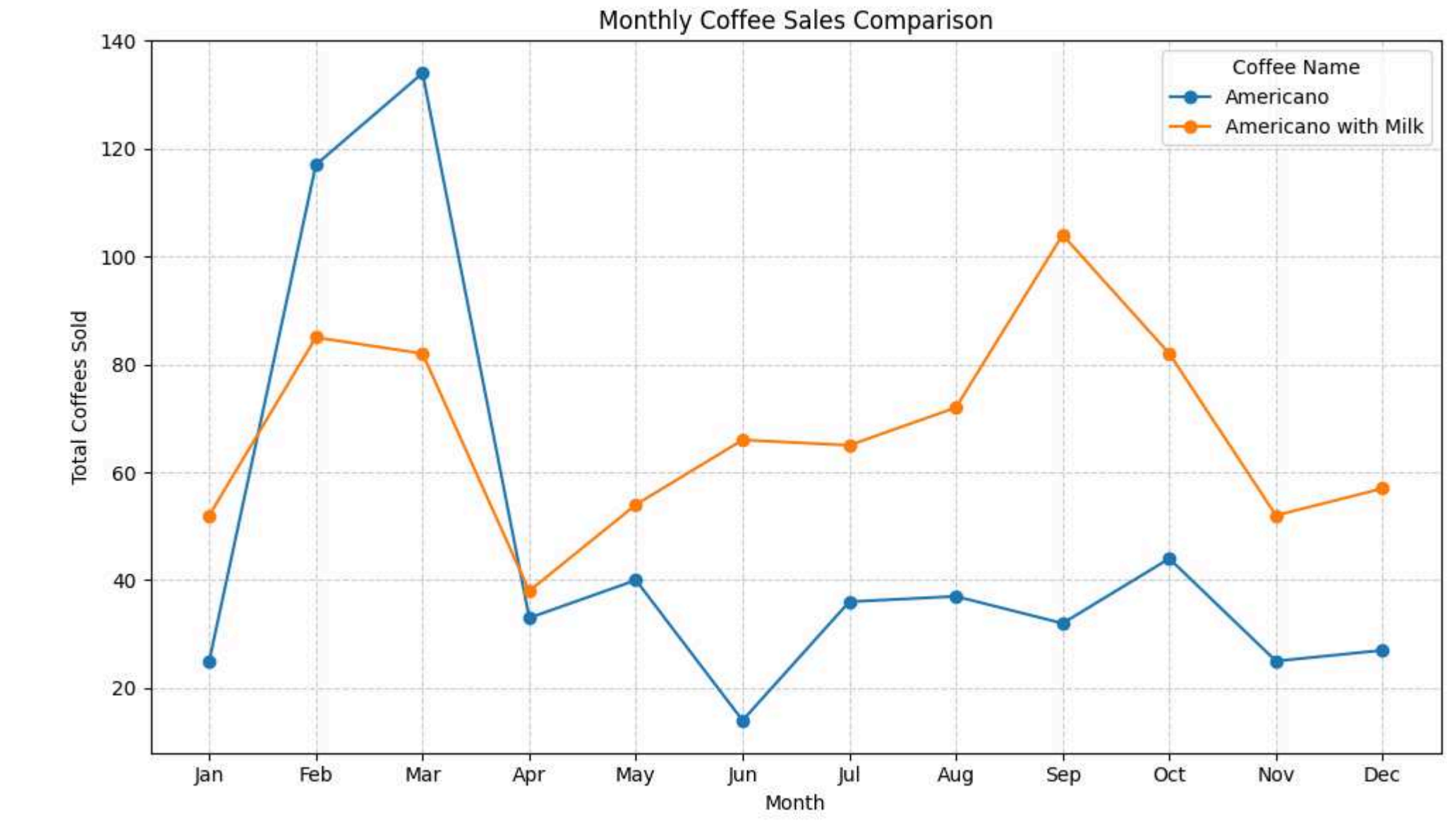
Line Graph

```
# Group data by month and coffee_name to get total sales count
monthly_sales = df.groupby(['Month_name', 'coffee_name']).size().unstack(fill_value=0)

# Sort months correctly if you have Monthsort or a proper order
if 'Monthsort' in df.columns:
    month_order = df[['Month_name', 'Monthsort']].drop_duplicates().sort_values('Monthsort')['Month_name']
    monthly_sales = monthly_sales.reindex(month_order)
else:
    monthly_sales = monthly_sales.reindex(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])

# Plot
plt.figure(figsize=(10, 6))
for coffee in monthly_sales.columns[2:]: # show two top coffee types for clarity
    plt.plot(monthly_sales.index, monthly_sales[coffee], marker='o', label=coffee)

plt.title('Monthly Coffee Sales Comparison')
plt.xlabel('Month')
plt.ylabel('Total Coffees Sold')
plt.legend(title='Coffee Name')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Pre-processing on dataset

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])
df.head()
```

	hour_of_day	cash_type	money	coffee_name	Time_of_Day	Weekday	Month_name	Weekdaysort	Monthsort	Date	Time
0	10	0	38.7	7	1	0	7	5	3	147	909
1	12	0	38.7	6	0	0	7	5	3	147	1144
2	12	0	38.7	6	0	0	7	5	3	147	1206
3	13	0	28.9	0	0	0	7	5	3	147	2641
4	13	0	38.7	7	0	0	7	5	3	147	2736

Next steps: [Generate code with df](#) [New interactive sheet](#)

Filling Missing Values

```
# Fill missing values in columns with mode
for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean() # Get the mode (most frequentvalue)
    df[column] = df[column].fillna(mean_value)
# Fill missing values in categorical columns with mode
for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0] # Get the mode (most frequentvalue)
    df[column] = df[column].fillna(mode_value)
df.isnull().sum()
```

```
0
hour_of_day    0
cash_type      0
money          0
coffee_name    0
Time_of_Day    0
Weekday        0
Month_name     0
Weekdaysort   0
Monthsort      0
Date           0
Time           0
dtype: int64
```

Correlation Heatmap

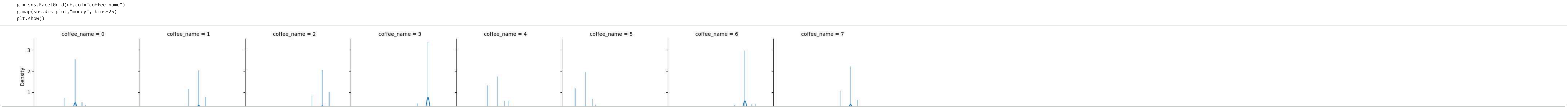
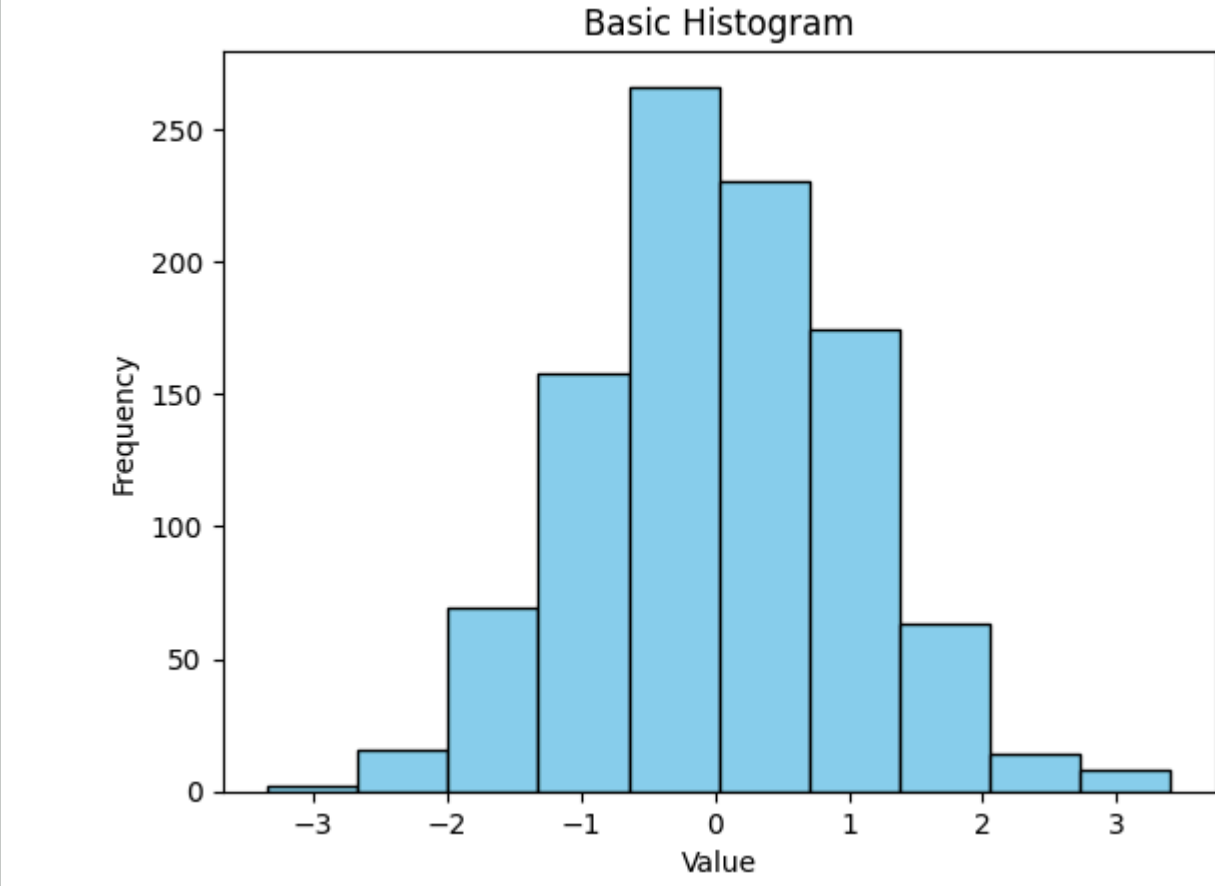
```
# Select only numeric columns (like money, hour_of_day, etc.)
#numeric_df = df.select_dtypes(include="number")

# Plot correlation heatmap
plt.figure(figsize=(18, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap of Coffee Sales Data')
plt.show()
```

Histogram

```
data = np.random.randn(1000)
plt.hist(data, color='skyblue', edgecolor='black')
plt.title('Basic Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



Boxplot

