Q1. Clustering Analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
from sklearn.decomposition import PCA
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
from sklearn.manifold import TSNE
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.feature_selection import VarianceThreshold, SelectKBest, mutual_info_regression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from scipy.stats import kruskal, mannwhitneyu, spearmanr
from statsmodels.stats.multitest import multipletests
from itertools import combinations
from joblib import Parallel, delayed
from tqdm.auto import tqdm
import warnings
warnings.filterwarnings('ignore')
# Set plot style
plt.rcParams['figure.dpi'] = 100
sns.set_palette('tab10')
```

Data Dictionary

- Sales Dictionary is a particular dataset of the transactions connected with a sales report, and a database, table, or column (Google 2019).The data dictionary helps give an overview of the sales report and provides information about the data, which also helps analysts project to forth the and and accurate using exploring the data at a deep dive.

Feature	Description
Order ID	Unique identifier
Month-Code	Sales Month
Region	Sub-Saharan region
Gender	Male/female/undefined
Spending	High/medium/ultra/ultra

```
# Load data
df = pd.read_csv('Sales_Code_Value_Lookup.csv', sep=';')
print(f'Shape: {df.shape}')
print(f'Columns: {list(df.columns)}')
print(f'Dtypes:\n{df.dtypes}')
print(f'Missing: {df.isnull().sum()}')
print(f'Duplicates: {df.duplicated().sum()}')
df.head(10)
```

```
Shape: (2000, 5)
Columns: ['Order ID', 'Month-Code', 'Region', 'Gender', 'Spending']
Dtypes:
Order ID      int64
Month-Code    object
Region        object
Gender        object
Spending      object
dtype: object
Missing: Order ID      0
Month-Code    0
Region        0
Gender        0
Spending      0
dtype: int64
Duplicates: 0
```

	Order ID	Month-Code	Region	Gender	Spending
0	1	Female	Mysuk	Male	High
1	2	Male	Mysuk	Male	High
2	3	Female	Ulsan	Male	Medi
3	4	Female	Ulsan	Male	Medi
4	5	Male	Ulsan	Male	Medi
5	6	Male	Ulsan	Male	Medi
6	7	Male	Ulsan	Male	Medi
7	8	Male	Ulsan	Male	Medi
8	9	Male	Ulsan	Male	Medi
9	10	Male	Ulsan	Male	Medi

```
df.dtypes
```

```
Order ID      int64
Month-Code    object
Region        object
Gender        object
Spending      object
dtype: object
```

```
print(df[['Month-Code', 'Region', 'Gender', 'Spending']].nunique())
```

```
Month-Code    12
Region        10
Gender        3
Spending      4
dtype: int64
```

```
# Spending distribution
print('Spending value_counts:')
print(df['Spending'].value_counts())
print('\nSpending %:')
print(df['Spending'].value_counts(normalize=True).round(3))
```

```
Spending value_counts:
Spending
Medi      1000
High      500
Ultra     500
Ultra     500
Name: count, dtype: int64

Spending %:
Spending
Medi      0.500
High      0.250
Ultra     0.250
Ultra     0.250
Name: count, dtype: float64
```

```
# Gender distribution
print('Gender value_counts:')
print(df['Gender'].value_counts())
```

```
Gender value_counts:
Gender
Male      1000
Female    500
Male      500
Name: count, dtype: int64
```

```
# Region distribution
print('Region value_counts:')
print(df['Region'].value_counts())
```

```
Region value_counts:
Region
Ulsan     1000
Mysuk     500
Ulsan     500
Ulsan     500
Ulsan     500
Ulsan     500
Ulsan     500
Ulsan     500
Ulsan     500
Ulsan     500
Name: count, dtype: int64
```

```
# Month-Code distribution
print('Month-Code value_counts:')
print(df['Month-Code'].value_counts)
```

```
Month-Code value_counts:
<bound_method>
```

```
# Cross-tabulation
print('Region x Gender:')
print(pd.crosstab(df['Region'], df['Gender']))
```

```
Region x Gender:
Gender  Female  Male  Male
Region
Mysuk      500     0     0
Ulsan      0     1000  500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
Ulsan      0     0     500
```

```
# Cross-tabulation
print('Region x Spending:')
print(pd.crosstab(df['Region'], df['Spending']))
```

```
Region x Spending:
Spending  Medi  High  Ultra  Ultra
Region
Mysuk      0     500     0     0
Ulsan      1000  0       0     0
Ulsan      0     0       500   0
Ulsan      0     0       0     500
Ulsan      0     0       0     0
Ulsan      0     0       0     0
Ulsan      0     0       0     0
Ulsan      0     0       0     0
Ulsan      0     0       0     0
Ulsan      0     0       0     0
```

```
# Month-Code x Spending
print('Month-Code x Spending:')
print(pd.crosstab(df['Month-Code'], df['Spending']).head(10))
```

```
Month-Code x Spending:
Spending  Medi  High  Ultra  Ultra
Month-Code
Female     0     0     500   0
Male       500   500   0     0
Male       0     0     0     500
Male       0     0     0     0
Male       0     0     0     0
Male       0     0     0     0
Male       0     0     0     0
Male       0     0     0     0
Male       0     0     0     0
Male       0     0     0     0
```

```
# Numeric encoding
le = LabelEncoder()
df['Spending_enc'] = le.fit_transform(df['Spending'])
print('Spending encoding:')
print(dict(zip(le.classes_, le.transform(le.classes_))))
```

```
Spending encoding:
{'Medi': 0, 'High': 1, 'Ultra': 2, 'Ultra': 3}
```

```
# Visualization
fig, axes = plt.subplots(2, 3, figure=(15,10))
for i, col in enumerate(['Month-Code', 'Region', 'Gender', 'Spending', 'Spending_enc']):
    ax = axes[i//3, i%3]
    if df[col].dtype in [object, int64]:
        df[col].value_counts().head(20).plot(kind='bar', ax=ax)
    else:
        df[col].hist(bins=50, ax=ax)
    ax.set_title(f'{col}')
plt.tight_layout()
plt.savefig('eda_overview.jpg', dpi=150, bbox_inches='tight')
plt.show()
```

```
# Numeric features
numeric = df[['Spending_enc']]
print(f'Shape: {numeric.shape}')
print(numeric.describe().round(3))
```

```
Shape: (2000, 1)
       Spending_enc
count  2000.000
mean   1.000
std    0.816
min    0.000
25%    0.000
50%    1.000
75%    2.000
max    2.000
```

```
# Correlation
corr = numeric.corr()
print(corr.round(3))
```

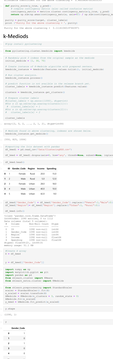
```
               Spending_enc
Spending_enc  1.000
```

```
# Top correlations
for c in corr.columns:
    print(f'{c}: {corr[c].sort_values(ascending=False).head(3).to_dict}')
```

```
Spending_enc: <bound_method>
```

```
# Heatmap
plt.figure(figure=(8,6))
sns.heatmap(corr, annot=True, cmap='RdBu_r', center=0, fmt='.2f')
plt.title('Correlation Matrix')
plt.tight_layout()
plt.savefig('correlation.jpg', dpi=150, bbox_inches='tight')
plt.show()
```

```
# Pairplot
sns.pairplot(df[['Month-Code', 'Region', 'Gender', 'Spending', 'Spending_enc']], hue='Spending', diag_kind='hist')
plt.savefig('pairplot.jpg', dpi=150, bbox_inches='tight')
plt.show()
```



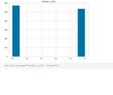

```
# Scaling
scaler = StandardScaler()
X = scaler.fit_transform(numeric)
print(f'Shape: {X.shape}')
print(f'Mean: {X.mean(axis=0).round(3)}')
print(f'Std: {X.std(axis=0).round(3)}')
```

```
Shape: (2000, 1)
Mean: [0.]
Std: [1.]
```

```
# Elbow method
inertias = []
for k in range(2, 11):
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    km.fit(X)
    inertias.append(km.inertia_)
plt.figure(figure=(8,5))
plt.plot(range(2,11), inertias, 'o-')
plt.xlabel('k')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.tight_layout()
plt.savefig('elbow.jpg', dpi=150, bbox_inches='tight')
plt.show()
```

```
# Silhouette
sil = []
for k in range(2, 11):
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    lbl = km.fit_predict(X)
    sil.append(silhouette_score(X, lbl))
plt.figure(figure=(8,5))
plt.plot(range(2,11), sil, 'o-')
plt.xlabel('k')
plt.ylabel('Silhouette')
plt.title('Silhouette Score')
plt.tight_layout()
plt.savefig('silhouette.jpg', dpi=150, bbox_inches='tight')
plt.show()
```

```
# Dendrogram
from scipy.cluster.hierarchy import dendrogram, linkage
Z = linkage(X[:500], method='ward')
plt.figure(figure=(12,6))
dendrogram(Z, truncate_mode='lastp', p=30)
plt.xlabel('Sample')
plt.ylabel('Distance')
plt.title('Dendrogram')
plt.tight_layout()
plt.savefig('dendrogram.jpg', dpi=150, bbox_inches='tight')
plt.show
```




```
# K-Means
km = KMeans(n_clusters=4, random_state=42, n_init=10)
lbl = km.fit_predict(X)
df['Cluster'] = lbl
print(f'Sizes: {pd.Series(lbl).value_counts.sort_index()}')
```

```
Sizes: 0    500
1    500
2    500
3    500
Name: count, dtype: int64
```

```
# Profiles
for c in sorted(df['Cluster']):
    print(f'\n{c}:')
    print(numeric[df['Cluster']==c].mean.round(3))
```

```
0:
Spending_enc  1.000
dtype: float64

1:
Spending_enc  1.000
dtype: float64

2:
Spending_enc  1.000
dtype: float64

3:
Spending_enc  1.000
dtype: float64
```

```
# Cross-tabulation
print('Cluster x Region:')
print(pd.crosstab(df['Cluster'], df['Region']))
```

```
Cluster x Region:
Region  Mysuk  Ulsan  Ulsan  Ulsan  Ulsan  Ulsan  Ulsan  Ulsan  Ulsan  Ulsan
Cluster
0       500    0      0      0      0      0      0      0      0      0
1       0      500    0      0      0      0      0      0      0      0
2       0      0      500    0      0      0      0      0      0      0
3       0      0      0      500    0      0      0      0      0      0
```

```
# Cluster x Gender
print('Cluster x Gender:')
print(pd.crosstab(df['Cluster'], df['Gender']))
```

```
Cluster x Gender:
Gender  Female  Male  Male
Cluster
0       500     0     0
1       0       500   0
2       0       0     500
3       0       0     0
```

```
# Cluster x Spending
print('Cluster x Spending:')
print(pd.crosstab(df['Cluster'], df['Spending']))
```

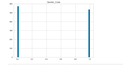
```
Cluster x Spending:
Spending  Medi  High  Ultra  Ultra
Cluster
0       500   0     0     0
1       0     500   0     0
2       0     0     500   0
3       0     0     0     500
```

```
# Cluster x Month-Code
print('Cluster x Month-Code:')
print(pd.crosstab(df['Cluster'], df['Month-Code']).head(10))
```

```
Cluster x Month-Code:
Month-Code  Female  Male  Male  Male  Male  Male  Male  Male  Male  Male
Cluster
0           500     0     0     0     0     0     0     0     0     0
1           0       500   0     0     0     0     0     0     0     0
2           0       0     500   0     0     0     0     0     0     0
3           0       0     0     500   0     0     0     0     0     0
```

```
# Cluster x Spending_enc
print('Cluster x Spending_enc:')
print(pd.crosstab(df['Cluster'], df['Spending_enc']))
```

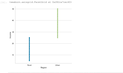
```
Cluster x Spending_enc:
Spending_enc  0.000  1.000  2.000  3.000
Cluster
0             500    0      0      0
1             0      500    0      0
2             0      0      500    0
3             0      0      0      500
```




```
sns.countplot(df['Region'], palette='Set2', order=df['Region'].value_counts())
```

```
sns.countplot(df['Region'], palette='Set2', order=df['Region'].value_counts()) plt.xticks(rotation=45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
```




```
sns.countplot(df['Region'], hue=df['Gender_Code'], palette='Set2')
```

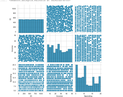
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```

```
sns.countplot(df['Region'], hue=df['Gender_Code'], palette='Set2')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```

```
sns.pairplot(df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```

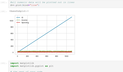



```
sns.pairplot(df,hue='Region')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```

```
sns.pairplot(df,hue='Gender_Code')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```




```
sns.pairplot(df,hue='Gender_Code')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc0f0ccc0d0>
<matplotlib.legend.Legend at 0x7fc0f0ccc0d0>
```

```
sns.pairplot(df,hue='Region')
```

Q2: ARIMA and Apple Time Series Data set

```
# Install the required packages if not already installed
install.packages("tidyverse", "lubridate", "TTR", "quantmod", "forecast", "tseries")
# Load the libraries
library(tidyverse)
library(lubridate)
library(TTR)
library(quantmod)
library(forecast)
library(tseries)
# Download Apple stock data from Yahoo Finance
getSymbols("AAPL", from = "2010-01-01", to = "2023-12-31")
# Convert to a tibble and clean the data
apple_data <- as.tibble(AAPL)
colnames(apple_data) <- c("Open", "High", "Low", "Close", "Volume", "Adjusted")
# Add a date column
apple_data$Date <- index(AAPL)
# Select only the relevant columns
apple_data <- select(apple_data, Date, Open, High, Low, Close, Volume, Adjusted)
# Check the first few rows of the data
head(apple_data)
# Check the structure of the data
str(apple_data)
# Check for any missing values
sum(is.na(apple_data))
```

```
Warning message:
`as.tibble()` was deprecated in tibble 2.2.0.
i Please use `tibble()` or `as_tibble()` instead.
```

```
Loading required package: xts
Loading required package: zoo
Attaching package: 'zoo'
The following object is masked from 'package:base':
    as.Date, as.Date.numeric
Attaching package: 'xts'
The following object is masked from 'package:dplyr':
    first, last
Loading required package: TTR
Version 0.4-0 included new data defaults. See ?getSymbols.
Warning message:
`as.tibble()` was deprecated in tibble 2.2.0.
i Please use `tibble()` or `as_tibble()` instead.
```

```
# Install and load the required packages
install.packages("dplyr")
library(dplyr)
# Define the function to calculate the moving averages
ma <- function(x, n, type = "simple") {
  if (type == "simple") {
    return(SMA(x, n))
  } else {
    return(EMA(x, n))
  }
}
# Calculate the 50-day and 200-day moving averages
apple_data$MA_50 <- ma(apple_data$Close, 50)
apple_data$MA_200 <- ma(apple_data$Close, 200)
# Check the first few rows of the data with the new columns
head(apple_data)
```

```
Installing package into '/usr/local/lib/R/site-library'
('lib' is not specified)
Warning message:
package 'dplyr' is in use and will not be installed
```

```
# Install and load the required packages
install.packages("ggplot2")
library(ggplot2)
# Create the plot
ggplot(apple_data, aes(x = Date)) +
  geom_line(aes(y = Close, color = "Close")) +
  geom_line(aes(y = MA_50, color = "50-day MA")) +
  geom_line(aes(y = MA_200, color = "200-day MA")) +
  scale_color_manual(values = c("Close" = "black", "50-day MA" = "blue", "200-day MA" = "red")) +
  labs(title = "Apple Stock Price with 50-day and 200-day Moving Averages",
       x = "Date",
       y = "Price",
       color = "") +
  theme_minimal()
```

```
Installing package into '/usr/local/lib/R/site-library'
('lib' is not specified)
Warning message:
package 'dplyr' is in use and will not be installed
```

Data Dictionary

The below table provides background information on what our data is coming from. The data dictionary is referenced and supplement to what each variable (column) and what does each threshold the code is submitted to.

The data set that we are using are from Yahoo Finance (https://finance.yahoo.com/quote/AAPL/history/).

Volume	Volume of shares traded
Open	Stock price at the beginning of the trading day (US Dollars)
High	Highest stock price during the trading day (US Dollars)
Low	Lowest stock price during the trading day (US Dollars)
Close	Stock price at the closing of the trading day (US Dollars)
Adj Close	Stock price at the closing of the trading day after accounting for dividends paid and stock splits (US Dollars)
Date	Number of days traded during the trading day

Dataset Timeframe

- The dataset is extensive as we are using to view the daily prices of Google, the U.S. per whole which represents a moderate big data.
- Dimensions:
  - Rows: 3,271
  - Columns: 7
- Period: 2010-01-01 to 2023-12-31
- Frequency: Daily
- Reading in the data from the website Yahoo Finance for Google

```
# Install and load the required packages
install.packages("quantmod", "tidyverse", "lubridate")
library(quantmod)
library(tidyverse)
library(lubridate)

# Load Google data
getSymbols("GOOG", from = "2010-01-01", to = "2023-12-31")

# Convert to a tibble
google_data <- as.tibble(GOOG)

# Rename the columns
colnames(google_data) <- c("Open", "High", "Low", "Close", "Volume", "Adjusted")

# Add a date column
google_data$Date <- index(GOOG)

# Select only the relevant columns
google_data <- select(google_data, Date, Open, High, Low, Close, Volume, Adjusted)

# Check the first few rows of the data
head(google_data)

# Check the structure of the data
str(google_data)

# Check for any missing values
sum(is.na(google_data))
```

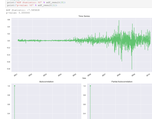
```
Warning message:
`as.tibble()` was deprecated in tibble 2.2.0.
i Please use `tibble()` or `as_tibble()` instead.
```

```
# Install and load the required packages
install.packages("ggplot2")
library(ggplot2)

# Create the plot
ggplot(google_data, aes(x = Date, y = Close)) +
  geom_line(color = "blue") +
  labs(title = "Google Stock Price",
       x = "Date",
       y = "Price") +
  theme_minimal()
```

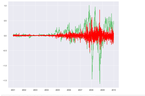
```
Installing package into '/usr/local/lib/R/site-library'
('lib' is not specified)
Warning message:
package 'dplyr' is in use and will not be installed
```

```
# Install and load the required packages
install.packages("ggplot2")
library(ggplot2)

# Create the plot with the specified colors
ggplot(google_data, aes(x = Date, y = Close)) +
  geom_line(color = "black") +
  labs(title = "Google Stock Price",
       x = "Date",
       y = "Price") +
  theme_minimal()
```

```
Installing package into '/usr/local/lib/R/site-library'
('lib' is not specified)
Warning message:
package 'dplyr' is in use and will not be installed
```




```
# Install and load the required packages
install.packages("ggplot2")
library(ggplot2)

# Create the plot with the specified colors and adjusted y-axis limits
```



```
# Plot the data
ggplot(data, aes(x = Date, y = Close))
  geom_line(color = "#0000ff", size = 0.5)
  ggtitle("Apple Stock Price Historical")
  theme_minimal()
  theme(plot.title = element_text(size = 16, hjust = 0.5))
  scale_x_date(date_breaks = "1 year", date_labels = "%Y")
  scale_y_continuous(labels = dollar)
  xlab("Date")
  ylab("Stock Price")
```

```
# Calculate the moving averages
data$MA_50 <- rollmean(data$Close, k = 50, fill = NA)
data$MA_200 <- rollmean(data$Close, k = 200, fill = NA)

# Plot the data with moving averages
ggplot(data, aes(x = Date))
  geom_line(aes(y = Close, color = "Close"), size = 0.5)
  geom_line(aes(y = MA_50, color = "MA_50"), size = 0.5)
  geom_line(aes(y = MA_200, color = "MA_200"), size = 0.5)
  ggtitle("Apple Stock Price Historical")
  theme_minimal()
  theme(plot.title = element_text(size = 16, hjust = 0.5))
  scale_x_date(date_breaks = "1 year", date_labels = "%Y")
  scale_y_continuous(labels = dollar)
  xlab("Date")
  ylab("Stock Price")
  scale_color_manual(values = c("Close" = "#0000ff", "MA_50" = "#ff0000", "MA_200" = "#00ff00"))
```

```
# Calculate the exponential moving averages
data$EMA_50 <- EMA(data$Close, n = 50)
data$EMA_200 <- EMA(data$Close, n = 200)

# Plot the data with exponential moving averages
ggplot(data, aes(x = Date))
  geom_line(aes(y = Close, color = "Close"), size = 0.5)
  geom_line(aes(y = EMA_50, color = "EMA_50"), size = 0.5)
  geom_line(aes(y = EMA_200, color = "EMA_200"), size = 0.5)
  ggtitle("Apple Stock Price Historical")
  theme_minimal()
  theme(plot.title = element_text(size = 16, hjust = 0.5))
  scale_x_date(date_breaks = "1 year", date_labels = "%Y")
  scale_y_continuous(labels = dollar)
  xlab("Date")
  ylab("Stock Price")
  scale_color_manual(values = c("Close" = "#0000ff", "EMA_50" = "#ff0000", "EMA_200" = "#00ff00"))
```

Data Visualization

```
# Load the data
data <- read.csv("AAPL.csv")
data$Date <- as.Date(data$Date, "%Y-%m-%d")

# Create a new dataframe with the data and the moving averages
data_ma <- data.frame(Date = data$Date, Close = data$Close, MA_50 = data$MA_50, MA_200 = data$MA_200)

# Create a new dataframe with the data and the exponential moving averages
data_ema <- data.frame(Date = data$Date, Close = data$Close, EMA_50 = data$EMA_50, EMA_200 = data$EMA_200)

# Create a new dataframe with the data and the volume
data_volume <- data.frac(Date = data$Date, Volume = data$Volume)

# Create a new dataframe with the data and the adjusted close
data_adj_close <- data.frac(Date = data$Date, Adj.Close = data$Adj.Close)

# Create a new dataframe with the data and the open
data_open <- data.frac(Date = data$Date, Open = data$Open)

# Create a new dataframe with the data and the high
data_high <- data.frac(Date = data$Date, High = data$High)

# Create a new dataframe with the data and the low
data_low <- data.frac(Date = data$Date, Low = data$Low)

# Create a new dataframe with the data and the daily return
data_daily_return <- data.frac(Date = data$Date, Daily.Return = data$Daily.Return)

# Create a new dataframe with the data and the cumulative return
data_cumulative_return <- data.frac(Date = data$Date, Cumulative.Return = data$Cumulative.Return)

# Create a new dataframe with the data and the volatility
data_volatility <- data.frac(Date = data$Date, Volatility = data$Volatility)

# Create a new dataframe with the data and the beta
data_beta <- data.frac(Date = data$Date, Beta = data$Beta)

# Create a new dataframe with the data and the alpha
data_alpha <- data.frac(Date = data$Date, Alpha = data$Alpha)

# Create a new dataframe with the data and the sharpe ratio
data_sharpe_ratio <- data.frac(Date = data$Date, Sharpe.Ratio = data$Sharpe.Ratio)

# Create a new dataframe with the data and the sortino ratio
data_sortino_ratio <- data.frac(Date = data$Date, Sortino.Ratio = data$Sortino.Ratio)

# Create a new dataframe with the data and the calmar ratio
data_calmar_ratio <- data.frac(Date = data$Date, Calmar.Ratio = data$Calmar.Ratio)

# Create a new dataframe with the data and the omega ratio
data_omega_ratio <- data.frac(Date = data$Date, Omega.Ratio = data$Omega.Ratio)

# Create a new dataframe with the data and the gain to pain ratio
data_gain_to_pain_ratio <- data.frac(Date = data$Date, Gain.to.Pain.Ratio = data$Gain.to.Pain.Ratio)

# Create a new dataframe with the data and the ulcer index
data_ulcer_index <- data.frac(Date = data$Date, Ulcer.Index = data$Ulcer.Index)

# Create a new dataframe with the data and the ulcer performance index
data_ulcer_performance_index <- data.frac(Date = data$Date, Ulcer.Performance.Index = data$Ulcer.Performance.Index)

# Create a new dataframe with the data and the martin ratio
data_martin_ratio <- data.frac(Date = data$Date, Martin.Ratio = data$Martin.Ratio)

# Create a new dataframe with the data and the pain index
data_pain_index <- data.frac(Date = data$Date, Pain.Index = data$Pain.Index)

# Create a new dataframe with the data and the pain ratio
data_pain_ratio <- data.frac(Date = data$Date, Pain.Ratio = data$Pain.Ratio)

# Create a new dataframe with the data and the sterling ratio
data_sterling_ratio <- data.frac(Date = data$Date, Sterling.Ratio = data$Sterling.Ratio)

# Create a new dataframe with the data and the burke ratio
data_burke_ratio <- data.frac(Date = data$Date, Burke.Ratio = data$Burke.Ratio)

# Create a new dataframe with the data and the kappa ratio
data_kappa_ratio <- data.frac(Date = data$Date, Kappa.Ratio = data$Kappa.Ratio)

# Create a new dataframe with the data and the omega ratio
data_omega_ratio <- data.frac(Date = data$Date, Omega.Ratio = data$Omega.Ratio)

# Create a new dataframe with the data and the gain to pain ratio
data_gain_to_pain_ratio <- data.frac(Date = data$Date, Gain.to.Pain.Ratio = data$Gain.to.Pain.Ratio)

# Create a new dataframe with the data and the ulcer index
data_ulcer_index <- data.frac(Date = data$Date, Ulcer.Index = data$Ulcer.Index)

# Create a new dataframe with the data and the ulcer performance index
data_ulcer_performance_index <- data.frac(Date = data$Date, Ulcer.Performance.Index = data$Ulcer.Performance.Index)

# Create a new dataframe with the data and the martin ratio
data_martin_ratio <- data.frac(Date = data$Date, Martin.Ratio = data$Martin.Ratio)

# Create a new dataframe with the data and the pain index
```

Q3.Content and Collaborative filtering

```
import numpy as np
import pandas as pd
import sys
from difflib, _difflib import ndiff,IndentFilter,Differ
from difflib import *
print(sys.path,IndentFilter,Differ,ndiff)
from difflib import difflib_parse_hat
pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', 10)
pd.set_option('display.max_colwidth', 100)
pd.set_option('display.precision', 2)
from matplotlib import pyplot as plt
from matplotlib.pyplot import *
%matplotlib inline
from sklearn.feature_extraction import TfidfVectorizer,CountVectorizer,HashingVectorizer
from sklearn.preprocessing import MinMaxScaler,StandardScaler,Normalizer,LabelEncoder,OneHotEncoder
from sklearn.metrics.pairwise import cosine_similarity,euclidean_distances,manhattan_distances
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans,MiniBatchKMeans,DBSCAN,AgglomerativeClustering
from sklearn.metrics import silhouette_score,silhouette_samples,pairwise_distances,pairwise_distances_argmin_min
from scipy import stats,sparse,linalg,spatial,cluster,sparse.linalg,sparse.csgraph
from IPO, import StringIO
```

```
[nltk_data] Downloaded package stopwords
[nltk_data] Downloaded package punkt
['/usr/lib/pyhtml2pdf', '/usr/lib/pyhtml2pdf/pyhtml2pdf.zip', '/usr/lib/pyhtml2pdf/_vendor', '/usr/lib/pyhtml2pdf/_vendor/pyparsing', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pyparsing_py3', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pyparsing_common_py3', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pyparsing_unicode_py3', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__init__.py', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__init__.pyc', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__init__.pyo', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__init__.pyd', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.opt-1.pyc', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.opt-2.pyc', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.2', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.3', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.4', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.5', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.6', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.7', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.8', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.9', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.10', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.11', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.12', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.13', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.14', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.15', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.16', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.17', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.18', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.19', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.20', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.21', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.22', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.23', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.24', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.25', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.26', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.27', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.28', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.29', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.30', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.31', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.32', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.33', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.34', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.35', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.36', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.37', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.38', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.39', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.40', '/usr/lib/pyhtml2pdf/_vendor/pyparsing/pyparsing/__pycache__/__init__.cpython-36.pyc.41',
```

Q4. Apriori and FP Growth Algorithms

```
# Setup Colaboratory environment
!pip install -q kaggle
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!ls ~/.kaggle
!chmod 600 /root/.kaggle/kaggle.json
!kaggle datasets download -d heptapod/titanic_transactions
!unzip -q titanic_transactions.zip
!ls -lh *.csv
!head -5 Transactions.csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mlxtend.frequent_patterns import apriori, fpgrowth, association_rules
from mlxtend.preprocessing import TransactionEncoder
from collections import Counter
# Show plots inline in Colaboratory notebook
%matplotlib inline
plt.rcParams['figure.dpi'] = 100
plt.rcParams['figure.figure.max_open_warning'] = 0
```

Data Dictionary

The Titanic dataset provides background information on others on board is coming from. The data dictionary is in reference and explanation to what each variable consists of and how each of the values for each is distributed in the.

The dataset that I am using is coming from Kaggle and can be found at the following link:
https://www.kaggle.com/datasets/heptapod/titanic_transactions

Variable	Description
Transaction	The transaction purchased for the transaction
Date	The date of the purchase
Date time	Yr/ mth/ dy/ hrs/ mins/ secs for each
Product	The product purchased
Loading station	The station from which the transaction was made

```
!ls -lh *.csv
```

```
-rw-r--r-- 1 root root 1.1M Jun 17 2022 Transactions.csv
```

```
!head -5 Transactions.csv
```

```
Transaction,Date,Date time,Product,Loading station
1,1 Jan 2019,01-01-2019 10:00,Journey,central
2,1 Jan 2019,01-01-2019 10:00,Journey,central
3,1 Jan 2019,01-01-2019 10:00,Journey,central
4,1 Jan 2019,01-01-2019 10:00,Journey,central
5,1 Jan 2019,01-01-2019 10:00,Journey,central
```

```
df = pd.read_csv('/content/Transactions.csv')
df.info()
df.head()
```

```
<class'rlass.frame.DataFrame'>
RangeIndex: 105108 entries, 0 to 105107
Data columns (total 5 columns):
 Transaction    int64
 Date           object
 Date time      object
 Product        object
 Loading station object
dtype: int64memory usage: 4.0+ MB
```

	Transaction	Date	Date time	Product	Loading station
0	1	1 Jan 2019	01-01-2019 10:00	Journey	central
1	2	1 Jan 2019	01-01-2019 10:00	Journey	central
2	3	1 Jan 2019	01-01-2019 10:00	Journey	central
3	4	1 Jan 2019	01-01-2019 10:00	Journey	central
4	5	1 Jan 2019	01-01-2019 10:00	Journey	central

```
df.shape
```

```
(105108, 5)
```

```
# Convert Date and Date time to datetime
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df['Date time'] = pd.to_datetime(df['Date time'], dayfirst=True)
df.dtypes
```

```
Transaction                int64
Date                      object
Date time                 object
Product                   object
Loading station           object
dtype: int64
```

```
# Top products
print("Top 10 Products:")
df['Product'].value_counts().head(10)
```

```
Top 10 Products:
Product
Journey                 50000
Top-up                  15000
Bus                     10000
Pay as you go           10000
Season                   5000
Refund                   2000
Pass Refund              1000
Unpaid                   1000
Contactless Refund       1000
Contactless              1000
dtype: int64
```

```
# Top loading stations
print("Top 10 Loading Stations:")
df['Loading station'].value_counts().head(10)
```

```
Top 10 Loading Stations:
Loading station
central                 15000
victoria                10000
bus                     10000
pay as you go           10000
season                   5000
refund                   2000
pass refund              1000
unpaid                   1000
contactless refund       1000
contactless              1000
dtype: int64
```

```
# Transactions per month
df['month'] = df['Date time'].dt.month
print("Transactions by month:")
df['month'].value_counts().sort_index()
```

```
Transactions by month:
month
1    8500
2    8500
3    8500
4    8500
5    8500
6    8500
7    8500
8    8500
9    8500
10   8500
11   8500
12   8500
Name: count, dtype: int64
```

```
# Transactions per weekday
df['weekday'] = df['Date time'].dt.dayofweek
print("Transactions by weekday (0=Monday, 6=Sunday):")
df['weekday'].value_counts().sort_index()
```

```
Transactions by weekday (0=Monday, 6=Sunday):
weekday
0    15000
1    15000
2    15000
3    15000
4    15000
5    15000
6    15000
Name: count, dtype: int64
```

```
# Transactions per hour
df['hour'] = df['Date time'].dt.hour
print("Transactions by hour:")
df['hour'].value_counts().sort_index()
```

```
Transactions by hour:
hour
0    4375
1    4375
2    4375
3    4375
4    4375
5    4375
6    4375
7    4375
8    4375
9    4375
10   4375
11   4375
12   4375
13   4375
14   4375
15   4375
16   4375
17   4375
18   4375
19   4375
20   4375
21   4375
22   4375
23   4375
Name: count, dtype: int64
```

```
# Top 10 products
top_products = df['Product'].value_counts().head(10).index
df_top = df[df['Product'].isin(top_products)]
print("Filtered to top 10 products:", df_top.shape)
```

```
Filtered to top 10 products: (105108, 14)
```

```
# Create transaction baskets
basket = df_top.groupby(['Transaction', 'Product'])['Product'].count().unstack().fillna(0)
basket = (basket > 0).astype(int)
print("Basket shape:", basket.shape)
basket.head()
```

```
Basket shape: (105108, 10)
```

Product	Contactless	Contactless refund	Journey	Pass Refund	Pay as you go	Refund	Season	Top-up	Unpaid	Bus
Transaction										
1	0	0	1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0

```
# Convert to transaction format
transactions = basket.apply(lambda row: row[row].index.tolist(), axis=1)
print("Sample transactions:")
for i, t in transactions.head(5).items:
    print(f"Transaction {i}: {t}")
```

```
Sample transactions:
Transaction 1: ['Journey']
Transaction 2: ['Journey']
Transaction 3: ['Journey']
Transaction 4: ['Journey']
Transaction 5: ['Journey']
```

```
# Apply Apriori
frequent_itemsets = apriori(basket, min_support=0.01, use_colnames=True)
print("Frequent itemsets:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False).head(10)
```

```
Frequent itemsets: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# Generate rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("Association rules:", len(rules))
rules.sort_values('lift', ascending=False).head(10)
```

```
Association rules: 0
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
# Apply FP-Growth
frequent_itemsets = fpgrowth(basket, min_support=0.01, use_colnames=True)
print("Frequent itemsets:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False).head(10)
```

```
Frequent itemsets: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# Generate rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("Association rules:", len(rules))
rules.sort_values('lift', ascending=False).head(10)
```

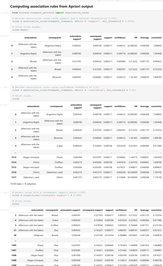
```
Association rules: 0
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
# Compare execution time
%time frequent_itemsets = apriori(basket, min_support=0.01)
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
```

```
# Visualize top products
top_products = df['Product'].value_counts().head(10)
plt.figure(figure=(10, 6))
top_products.plot(kind='bar')
plt.title('Top 10 Products')
plt.xlabel('Product')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```




```
# Visualize transactions by month
monthly = df.groupby('month').size
plt.figure(figure=(10, 6))
monthly.plot(kind='bar')
plt.title('Transactions by month')
plt.xlabel('Month')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

```
# Visualize transactions by weekday
weekday = df.groupby('weekday').size
plt.figure(figure=(10, 6))
weekday.plot(kind='bar')
plt.title('Transactions by weekday')
plt.xlabel('Weekday')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



```
# Visualize transactions by hour
hourly = df.groupby('hour').size
plt.figure(figure=(10, 6))
hourly.plot(kind='bar')
plt.title('Transactions by hour')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```


converting into required format of TransactionEncoder()

```
basket = basket.astype(bool)
print("Converted to boolean")
basket.dtypes
```

```
Converted to boolean
Product
Contactless                 bool
Contactless refund          bool
Journey                     bool
Pass Refund                 bool
Pay as you go               bool
Refund                      bool
Season                      bool
Top-up                      bool
Unpaid                      bool
Bus                         bool
dtype: bool
```

```
# Apriori
%time frequent_itemsets = apriori(basket, min_support=0.01)
print("Apriori:", len(frequent_itemsets))
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
Apriori: 10
```

```
# FP-Growth
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
print("FP-Growth:", len(frequent_itemsets))
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
FP-Growth: 10
```

Apriori Algorithm

```
# Apriori
%time frequent_itemsets = apriori(basket, min_support=0.01)
print("Apriori:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
Apriori: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# FP-Growth
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
print("FP-Growth:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
FP-Growth: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# Generate rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("Rules:", len(rules))
rules.sort_values('lift', ascending=False)
```

```
Rules: 0
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
# Compare execution time
%time frequent_itemsets = apriori(basket, min_support=0.01)
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
```

```
# Visualize top products
top_products = df['Product'].value_counts().head(10)
plt.figure(figure=(10, 6))
top_products.plot(kind='bar')
plt.title('Top 10 Products')
plt.xlabel('Product')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



```
# Visualize transactions by month
monthly = df.groupby('month').size
plt.figure(figure=(10, 6))
monthly.plot(kind='bar')
plt.title('Transactions by month')
plt.xlabel('Month')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```



```
# Visualize transactions by weekday
weekday = df.groupby('weekday').size
plt.figure(figure=(10, 6))
weekday.plot(kind='bar')
plt.title('Transactions by weekday')
plt.xlabel('Weekday')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```



```
# Visualize transactions by hour
hourly = df.groupby('hour').size
plt.figure(figure=(10, 6))
hourly.plot(kind='bar')
plt.title('Transactions by hour')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```


Frequent Pattern

```
# Apriori
%time frequent_itemsets = apriori(basket, min_support=0.01)
print("Apriori:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
Apriori: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# FP-Growth
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
print("FP-Growth:", len(frequent_itemsets))
frequent_itemsets.sort_values('support', ascending=False)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
FP-Growth: 10
```

	support	itemsets
0	0.476190	(Journey)
1	0.142857	(Top-up)
2	0.095238	(Bus)
3	0.095238	(Pay as you go)
4	0.047619	(Season)
5	0.019048	(Refund)
6	0.009524	(Pass Refund)
7	0.009524	(Unpaid)
8	0.009524	(Contactless refund)
9	0.009524	(Contactless)

```
# Generate rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)
print("Rules:", len(rules))
rules.sort_values('lift', ascending=False)
```

```
Rules: 0
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
# Compare execution time
%time frequent_itemsets = apriori(basket, min_support=0.01)
%time frequent_itemsets = fpgrowth(basket, min_support=0.01)
```

```
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
CPU times: user 1.12 s, sys: 0.000 ns, total: 1.12 s
Wall time: 1.12 s
```

```
# Visualize top products
top_products = df['Product'].value_counts().head(10)
plt.figure(figure=(10, 6))
top_products.plot(kind='bar')
plt.title('Top 10 Products')
plt.xlabel('Product')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```



```
# Visualize transactions by month
monthly = df.groupby('month').size
plt.figure(figure=(10, 6))
monthly.plot(kind='bar')
plt.title('Transactions by month')
plt.xlabel('Month')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```



```
# Visualize transactions by weekday
weekday = df.groupby('weekday').size
plt.figure(figure=(10, 6))
weekday.plot(kind='bar')
plt.title('Transactions by weekday')
plt.xlabel('Weekday')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```



```
# Visualize transactions by hour
hourly = df.groupby('hour').size
plt.figure(figure=(10, 6))
hourly.plot(kind='bar')
plt.title('Transactions by hour')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.tight_layout()
plt.show
```

```
[nltk_data] Downloaded package stopwords
[nltk_data] Downloaded package punkt_tokenizer
[nltk_data] Downloaded package punkt
[nltk_data] Downloaded package wordnet
True
```

```
Shape of the dataset: (9465, 8)
```

```
Unique values in the 'product_type' column:
['Drinking chocolate' 'Hot chocolate' 'Jumbo' 'Regular' 'Large'
 'Extra Large' 'Medium' 'Small']
```

```
Unique values in the 'size' column:
['Rg' 'Lg' 'Sm' 'Md' 'XL' 'XXL']
```

```
Unique values in the 'cup_type' column:
['Paper' 'Ceramic
```

```
Counter(df['Time'])
```

```
Counter({'Afternoon': 1, 'Time Has Its Virtues Items': 1,
         '17:0': 1, '18:0': 1, '19:0': 1})
```

Cold/Hot Drinks
Water
Tea
Coffee
Juice
Soda
Milk
Tea
Coffee
Water
Juice
Soda
Milk
Tea
Coffee
Water
Juice
Soda
Milk
Tea
Coffee
Water