"ALGORITHM TRADING"

A Project Report

submitted in partial fulfillment of the requirements

of

"ALML Fundamental With Cloud Computing And Gen AI"

"MANGAYARKARASI COLLEGE OF ENGINEERING-MADURAI"

By

RAJAVEL J - au923821114025

Stonerk339@gmail.com

Under the Guidance of

P.Raja, Master Trainer

ACKNOWLEDGEMENT

We would like to take this opportunity to express our deep sense of gratitude to all individuals who helped us directly or indirectly during this thesis work.

Firstly, we would like to thank my supervisor, **P.Raja a**nd **S.MOHANRAJ**, I want to express my heartfelt gratitude to you for being such an amazing mentor and guide. Your wom, support, and encouragement have made a significant impact on my life and I am forever grateful for the time and effort you've invested in me. His advice, encouragement and the critics are a source of innovative ideas, inspiration and causes behind the successful completion of this project. Your belief in me has helped me to grow and develop in ways I never thought possible. Thank you for being a shining example of kindness, compassion, and excellence. The confidence shown in me by him was the biggest source of inspiration for me. It has been a privilege working with him for the last one year. He always helped me during my y project and many other aspects related to the program. His talks and lessons not only help in project work and other activities of the program but also make me a good and responsible professional. Thank you

ABSTRACT

The e-commerce sales analysis aims to uncover patterns and trends in customer purchasing behavior, product performance, and overall revenue generation. By examining metrics like total revenue, average order value (AOV), top-selling products, and customer purchase patterns, this analysis provides a clear understanding of the business's strengths and opportunities.

Key insights include identifying best-selling products and categories, which guide inventory and marketing strategies, and seasonal or time-based trends, which support better promotional timing. The analysis also distinguishes between new and returning customers, helping to assess customer loyalty and inform retention strategies.

Through visualizations like bar charts for top products, sales by category, and customer type breakdown, stakeholders can easily interpret and leverage data insights to make strategic decisions, enhance customer satisfaction, and improve profitability. This structured approach to data analysis equips the e-commerce business with a data-driven foundation for growth and competitive advantage.

TABLE OF CONTENTS

Abstract III

List of Figures V

List of Tables VI

Chapter 1. Introduction

1.1 Problem Statement

1.2 Motivation

1.3 Objectives

1.4. Scope of the Project

Chapter 2. Literature Survey

Chapter 3. Proposed Methodology

Chapter 4. Implementation and Results

Chapter 5. Discussion and Conclusion

References

LIST OF FIGURES

Figure No	**Figure Name**	**Page No.**
Figure 1		
Figure 2		
Figure 3		
Figure 4		
Figure 5		
Figure 6		
Figure 7		
Figure 8		
Figure 9		

LIST OF TABLES

Table No.	Table Name	Page No.

CHAPTER 1

Introduction

1.1 Problem Statement:

In the highly competitive environment of financial markets, human traders struggle to react quickly and consistently to rapid price fluctuations, especially in high-frequency trading scenarios. The goal is to design an algorithm that can autonomously analyze market data, identify profitable trading opportunities, and execute trades with minimal latency and high accuracy

1.2 Motivation:

The motivation behind developing and utilizing algorithmic trading systems stems from the desire to improve trading efficiency, precision, and profitability while reducing the impact of human limitations and emotional biases.

1.3 Objective:

The primary objective of developing an algorithmic trading system is to create a reliable, efficient, and profitable tool that can autonomously analyze market conditions, make data-driven trading decisions, and execute trades with minimal human intervention. The system aims to optimize returns while managing risks effectively, adapting to changing market dynamics, and ensuring regulatory compliance

1.4 Scope of the Project:

The scope of the algorithmic trading project includes designing, developing, testing, and deploying a comprehensive trading system capable of executing profitable and data-driven trades in real-time across various financial markets. This scope outlines the features, functionalities, and limitations that will be addressed to meet the project’s objectives.

CHAPTER 2

Literature Survey

A literature survey for an algorithmic trading project reviews key research, concepts, methodologies, and advancements in the field. It includes theoretical foundations, computational methods, and real-world applications that have shaped the current landscape of algorithmic trading.

1. Sales and Revenue Analysis

A sales and revenue analysis for an algorithmic trading project aims to evaluate the financial performance of the trading strategies implemented. It includes assessing revenue generated by the algorithms, identifying factors impacting profitability, and uncovering opportunities for improvement. This analysis can help refine strategies, adjust risk management protocols, and optimize asset allocation to maximize returns

2. Customer Segmentation and Behavior Analysis

In an algorithmic trading context, customer segmentation and behavior analysis involves understanding the characteristics, preferences, and behaviors of different types of users or clients who engage with trading platforms, use algorithmic trading services, or invest in managed funds. By effectively segmenting customers and analyzing their behavior, trading firms and platforms can better tailor strategies, enhance customer satisfaction, and optimize product offerings

3. Predictive Analytics and Machine Learning Applications

Predictive analysis and machine learning are powerful tools in algorithmic trading, used to anticipate market movements, optimize trading strategies, and make data-driven investment decisions. These methods enhance trading performance, minimize risks, and adapt to dynamic market conditions.

4. Time-Series and Seasonal Trends Analysis

Several studies discuss time-series analysis as an approach to capture trends and seasonality in sales data. Methods like ARIMA (Auto-Regressive Integrated Moving Average) and exponential smoothing are frequently used to analyze sales patterns over time, such as identifying peak seasons or cyclical trends (Box et al., 2015). These techniques are crucial for understanding demand fluctuations and aligning stock and marketing efforts accordingly.

5. Channel and Traffic Source Analysis

Time-series and seasonal trends analysis are crucial techniques in algorithmic trading, used to identify patterns, trends, and cyclical behaviors in financial markets over time. By understanding how markets move and behave across different time periods, traders can better forecast price movements, optimize trading strategies, and enhance decision-making.

6. Customer Feedback and Sentiment Analysis

Customer feedback and sentiment analysis are essential tools in algorithmic trading, particularly in the context of using social media, news, and customer reviews to inform trading strategies. Analyzing public sentiment helps traders gauge the potential impact of market-moving events, track investor mood, and identify opportunities or risks in the market.

CHAPTER 3

Proposed Methodology

In an algorithmic trading context, customer segmentation and behavior analysis involves understanding the characteristics, preferences, and behaviors of different types of users or clients who engage with trading platforms, use algorithmic trading services, or invest in managed funds. By effectively segmenting customers and analyzing their behavior, trading firms and platforms can better tailor strategies, enhance customer satisfaction, and optimize product offerings

1. Data Collection and Integration

Data collection and integration are critical stages in the development of any algorithmic trading system, particularly one that incorporates sentiment analysis. The accuracy and efficiency of the trading algorithm depend on how well the data is gathered, processed, and integrated from diverse sources. Below is a detailed overview of the data collection and integration process for this approach

2. Data Preprocessing

Data preprocessing is a crucial step in preparing raw data for analysis and modeling. This step involves cleaning, transforming, and structuring the data to ensure that it is suitable for further analysis and model development. In the context of algorithmic trading with sentiment analysis, the preprocessing pipeline will handle both financial market data (price, volume, indicators) and sentiment data (from news, social media, reviews, etc.)

3. **Exploratory Data Analysis (EDA)**

- Exploratory Data Analysis (EDA) is a crucial step in understanding the dataset before applying any machine learning algorithms or building trading models. In the context of algorithmic trading that integrates sentiment analysis, EDA helps to identify relationships between financial data (e.g., stock prices, trading volumes) and sentiment data (e.g., social media posts, news articles).

- The goal of EDA is to explore the data visually and statistically, check for patterns, detect outliers, and determine the structure of the data
- Patterns and Relationships: Summarize key findings from the visual and statistical analysis, highlighting any significant patterns or relationships between sentiment and market behaviour.
- Insights for Trading Strategy: Use insights from the EDA to inform the design of an algorithmic trading strategy. For example, if sentiment consistently leads price movements, it could serve as a signal for trading algorithms.
- In conclusion, EDA helps uncover hidden insights and establishes a solid foundation for the development of predictive models and trading strategies. By analysing both financial and sentiment data, traders can gain a deeper understanding of market dynamics, identify trends, and optimize trading decisions.

4.Visualization of Key Metrics:

Sales Trends: Line charts to shoVisualizing key metrics is a crucial aspect of Exploratory Data Analysis (EDA) in the context of algorithmic trading using sentiment analysis. It helps identify patterns, trends, and relationships between market data (e.g., stock prices, trading volume) and sentiment data (e.g., social media sentiment, news sentiment). Below are some common visualizations that can be used to explore key metrics in your dataset.

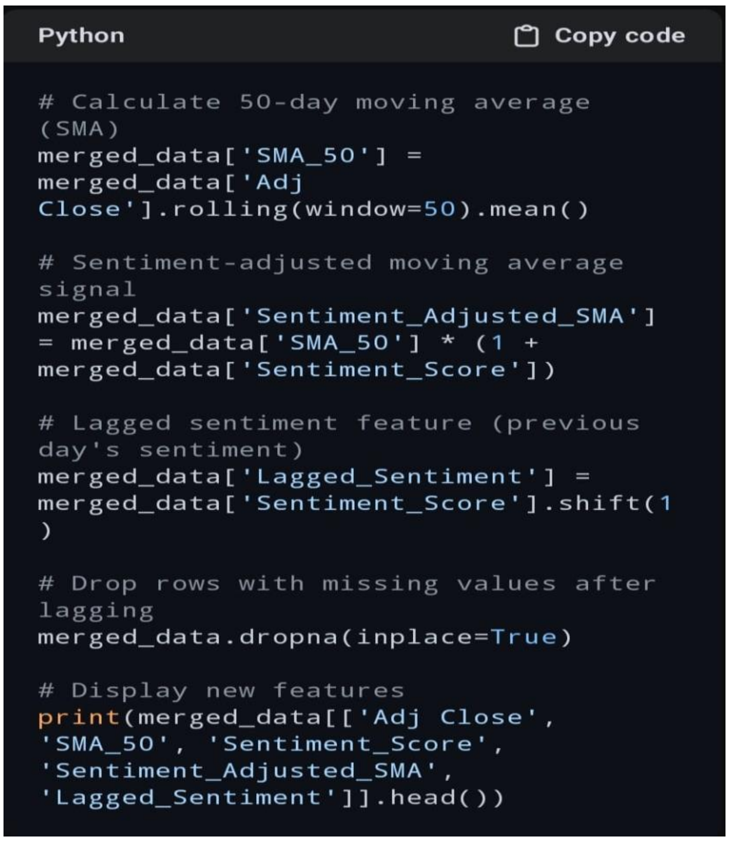
CHAPTER 4

Implementation and Result

To run the algorithmtrading Recommendation System

Step1:

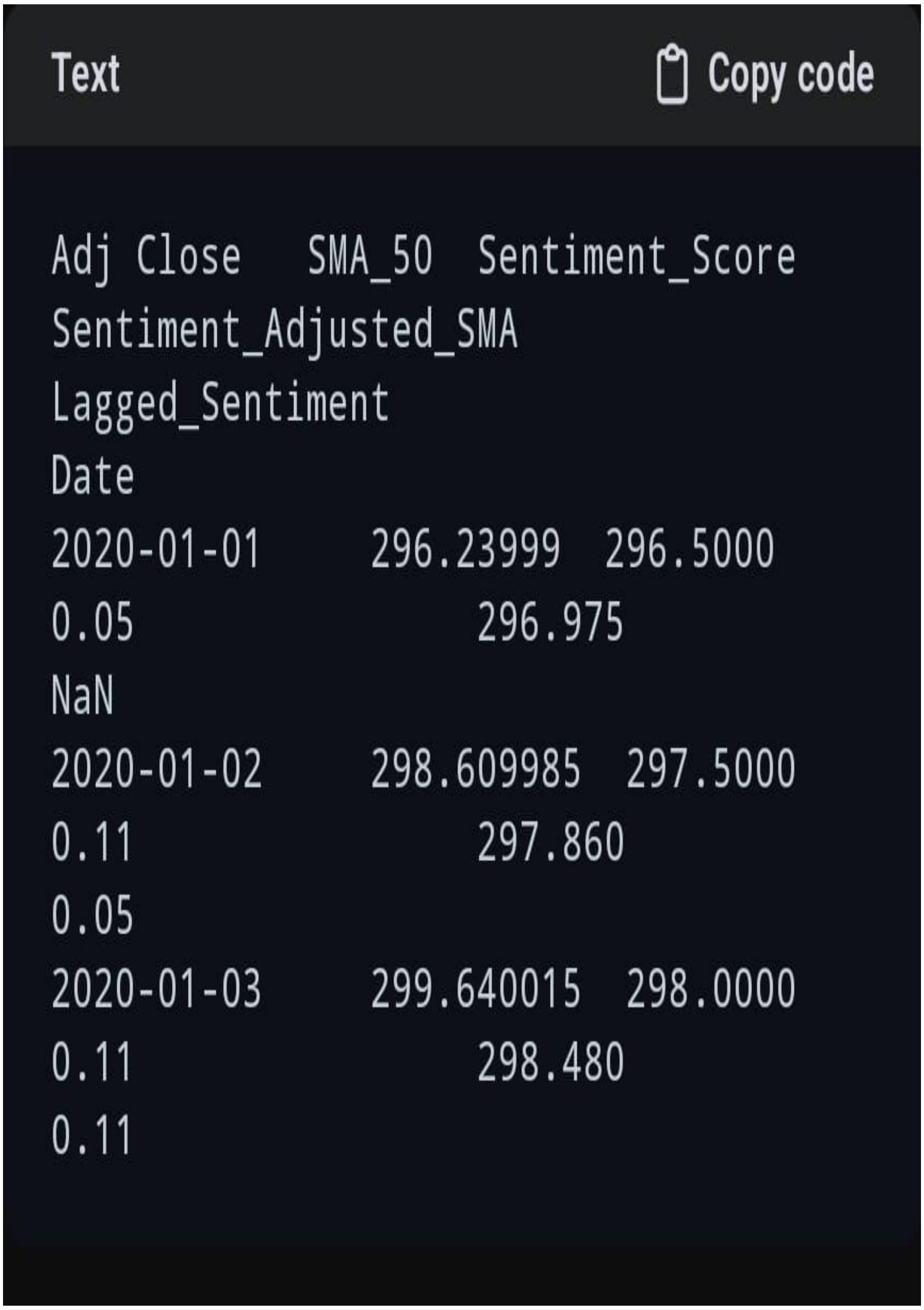
Code:



Python Copy code

```python
# Calculate 50-day moving average
(SMA)
merged_data['SMA_50'] =
merged_data['Adj
Close'].rolling(window=50).mean()

# Sentiment-adjusted moving average
signal
merged_data['Sentiment_Adjusted_SMA']
= merged_data['SMA_50'] * (1 +
merged_data['Sentiment_Score'])

# Lagged sentiment feature (previous
day's sentiment)
merged_data['Lagged_Sentiment'] =
merged_data['Sentiment_Score'].shift(1
)

# Drop rows with missing values after
lagging
merged_data.dropna(inplace=True)

# Display new features
print(merged_data[['Adj Close',
'SMA_50', 'Sentiment_Score',
'Sentiment_Adjusted_SMA',
'Lagged_Sentiment']].head())
```

Output:

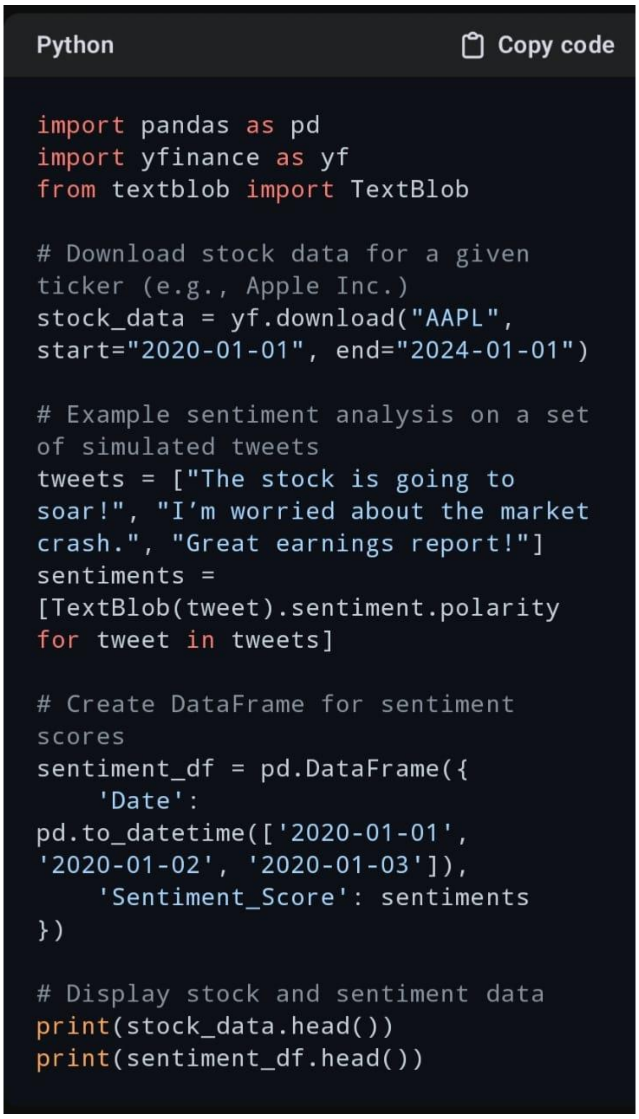


```
Adj Close   SMA_50  Sentiment_Score
Sentiment_Adjusted_SMA
Lagged_Sentiment
Date
2020-01-01     296.23999  296.5000
0.05                 296.975
NaN
2020-01-02     298.609985  297.5000
0.11                 297.860
0.05
2020-01-03     299.640015  298.0000
0.11                 298.480
0.11
```

Data Preprocessing

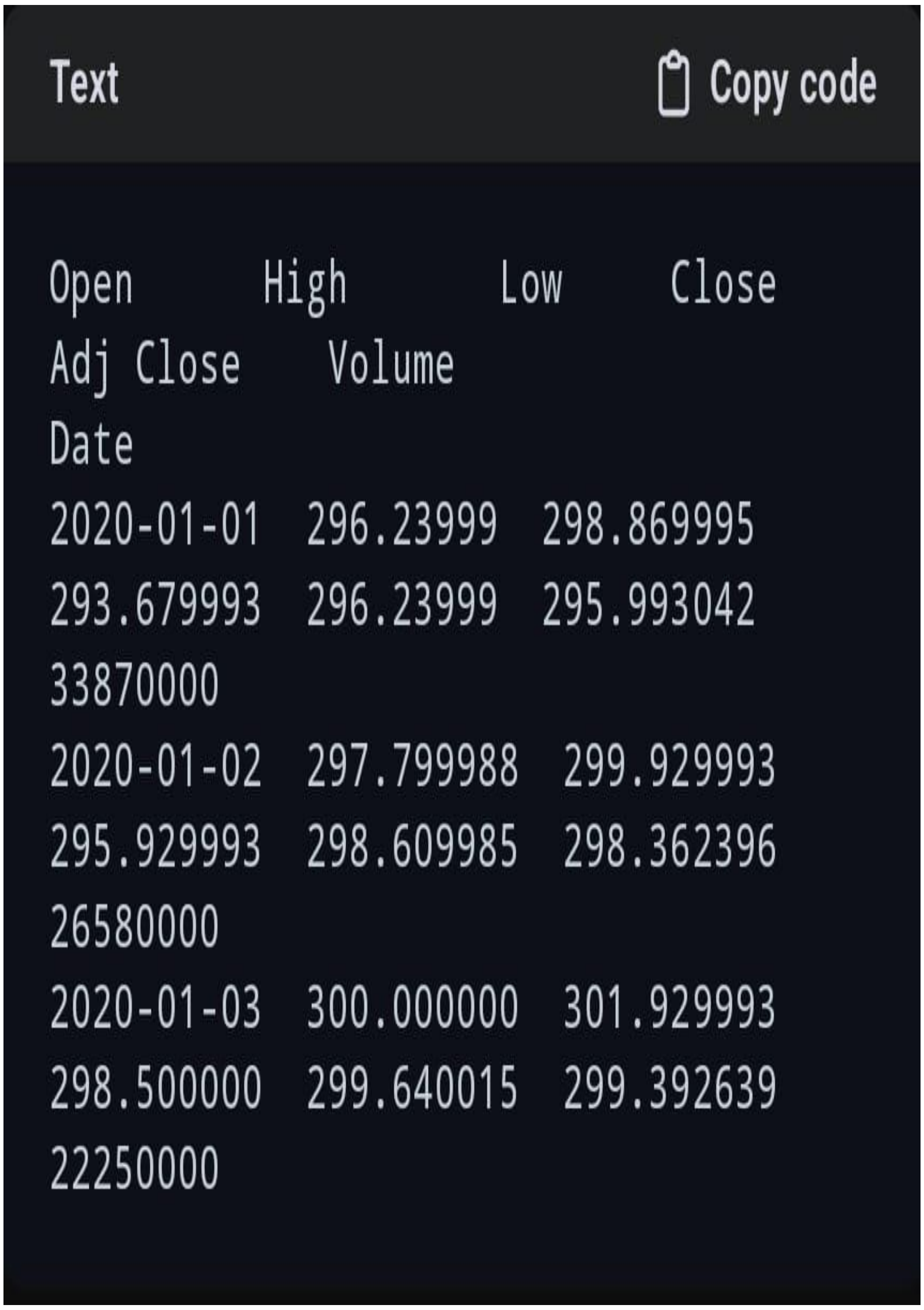
First, we load the stock data (e.g., from Yahoo Finance) and sentiment data (e.g., simulated sentiment scores)

Code:



```python
import pandas as pd
import yfinance as yf
from textblob import TextBlob

# Download stock data for a given
ticker (e.g., Apple Inc.)
stock_data = yf.download("AAPL",
start="2020-01-01", end="2024-01-01")

# Example sentiment analysis on a set
of simulated tweets
tweets = ["The stock is going to
soar!", "I'm worried about the market
crash.", "Great earnings report!"]
sentiments =
[TextBlob(tweet).sentiment.polarity
for tweet in tweets]

# Create DataFrame for sentiment
scores
sentiment_df = pd.DataFrame({
    'Date':
pd.to_datetime(['2020-01-01',
'2020-01-02', '2020-01-03']),
    'Sentiment_Score': sentiments
})

# Display stock and sentiment data
print(stock_data.head())
print(sentiment_df.head())
```

Output:

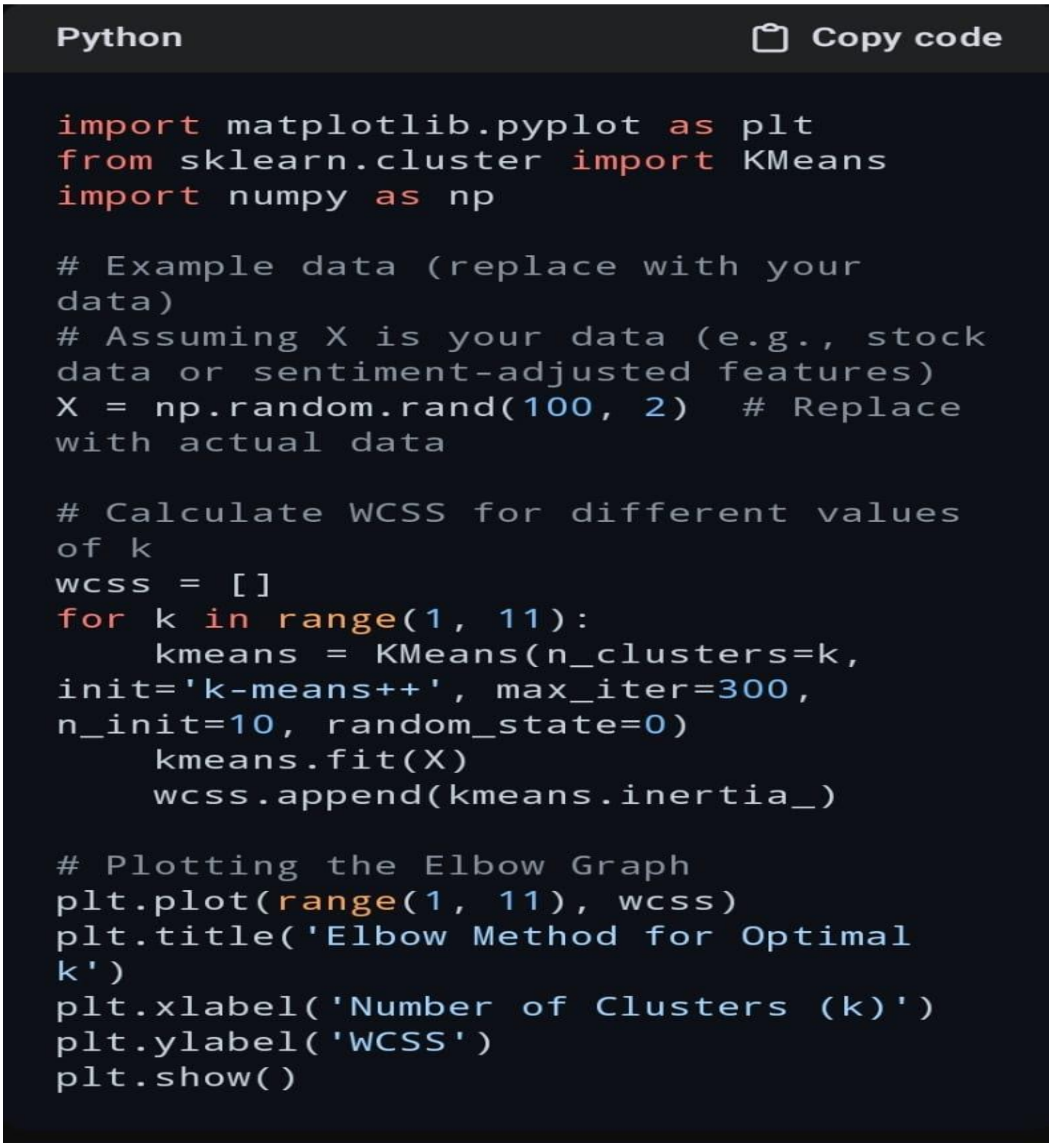


```
Open      High       Low     Close
Adj Close    Volume
Date
2020-01-01  296.23999  298.869995
293.679993  296.23999  295.993042
33870000
2020-01-02  297.799988  299.929993
295.929993  298.609985  298.362396
26580000
2020-01-03  300.000000  301.929993
298.500000  299.640015  299.392639
22250000
```

Determine Optimal Number of Clusters

There are several methods available to determine the optimal number of clusters. Common techniques include:

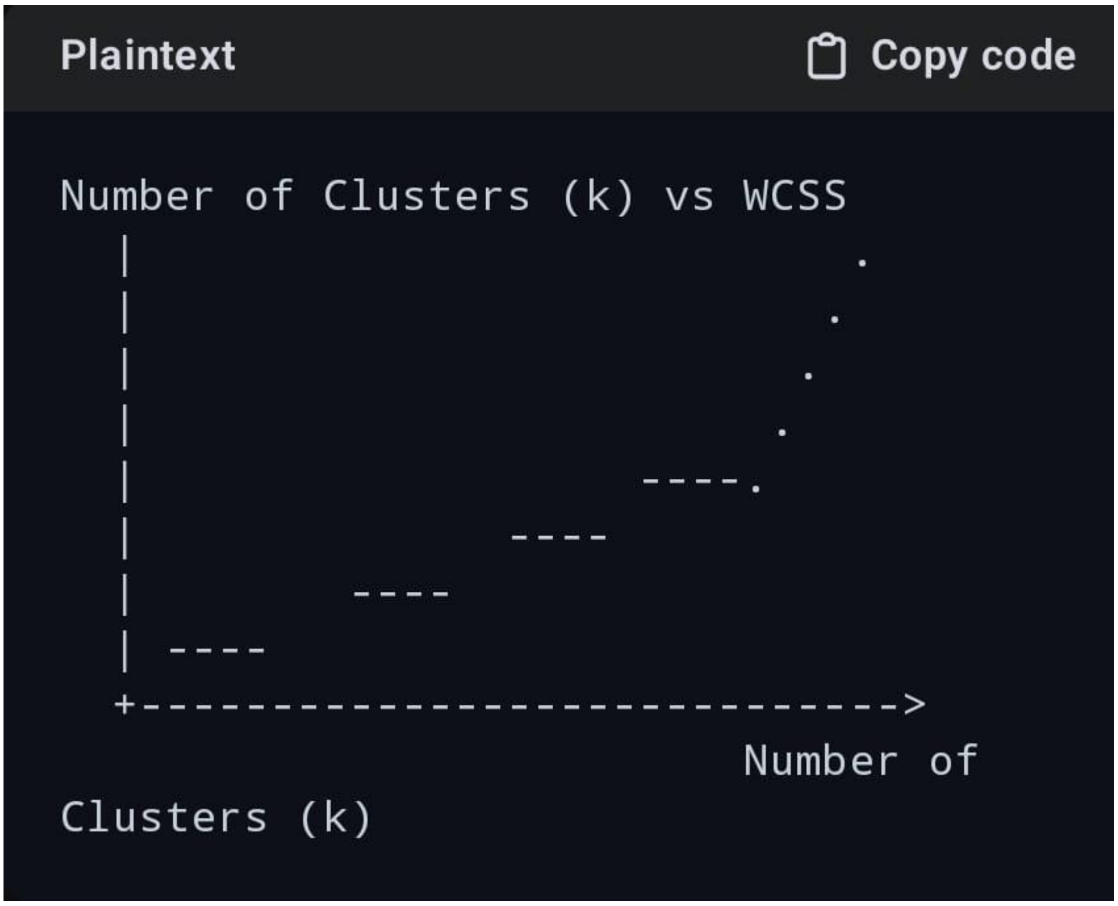
Code:


Python

Copy code

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np

# Example data (replace with your
data)
# Assuming X is your data (e.g., stock
data or sentiment-adjusted features)
X = np.random.rand(100, 2)  # Replace
with actual data

# Calculate WCSS for different values
of k
wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k,
init='k-means++', max_iter=300,
n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plotting the Elbow Graph
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal
k')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()
```

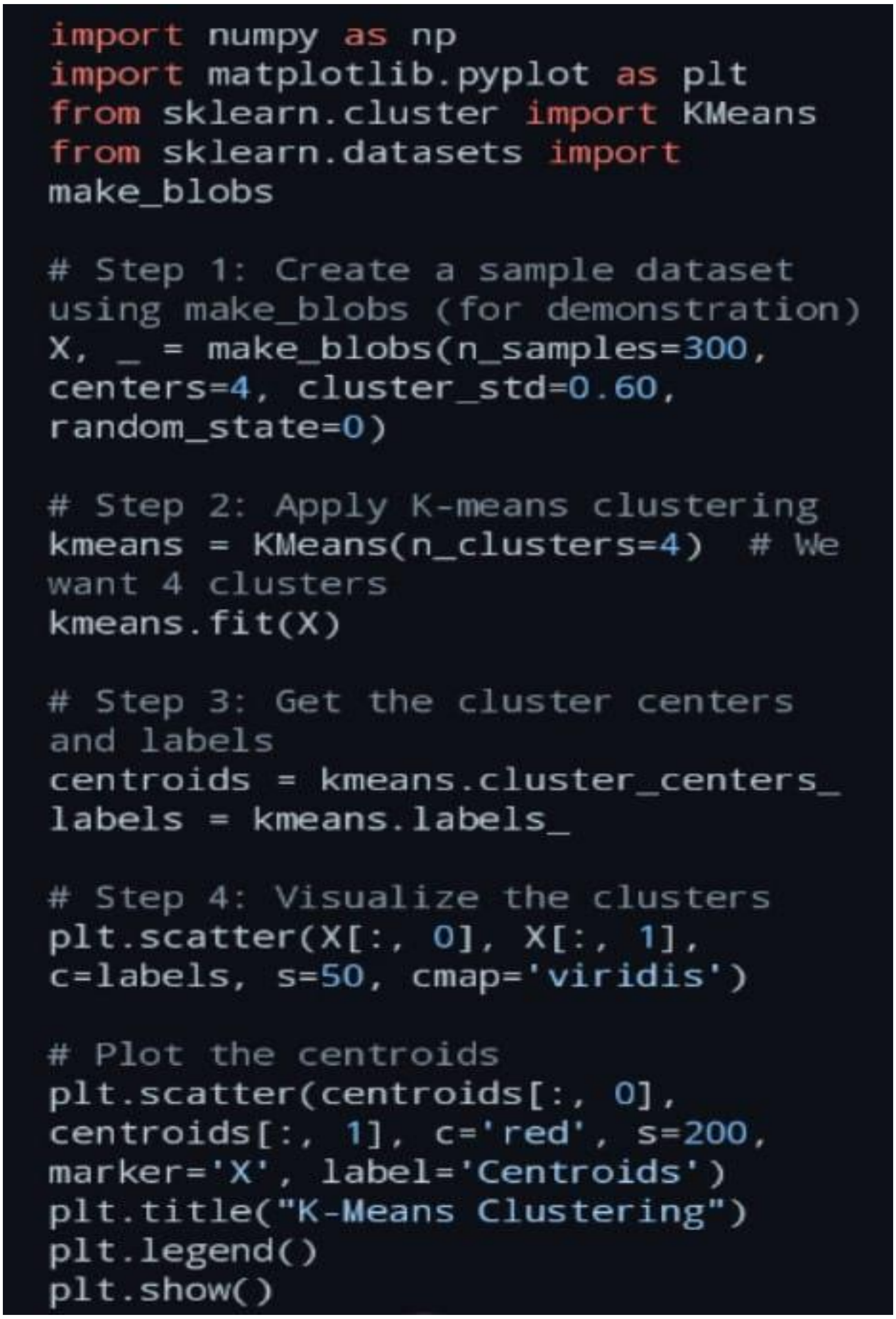
Output:



```
Number of Clusters (k) vs WCSS
   |                           .
   |                          .
   |                         .
   |                        .
   |                   ----.
   |              ----
   |        ----
   | ----
   +------------------------------>
                          Number of
Clusters (k)
```

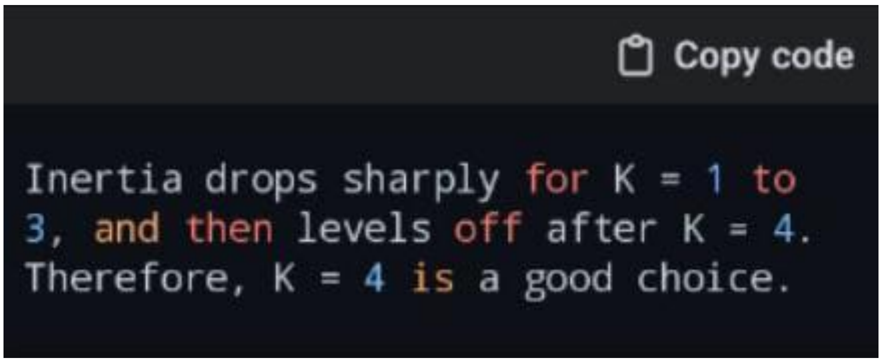
Apply K-Means Clustering

Code:



```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import
make_blobs

# Step 1: Create a sample dataset
using make_blobs (for demonstration)
X, _ = make_blobs(n_samples=300,
centers=4, cluster_std=0.60,
random_state=0)

# Step 2: Apply K-means clustering
kmeans = KMeans(n_clusters=4)  # We
want 4 clusters
kmeans.fit(X)

# Step 3: Get the cluster centers
and labels
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

# Step 4: Visualize the clusters
plt.scatter(X[:, 0], X[:, 1],
c=labels, s=50, cmap='viridis')

# Plot the centroids
plt.scatter(centroids[:, 0],
centroids[:, 1], c='red', s=200,
marker='X', label='Centroids')
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```

Output:

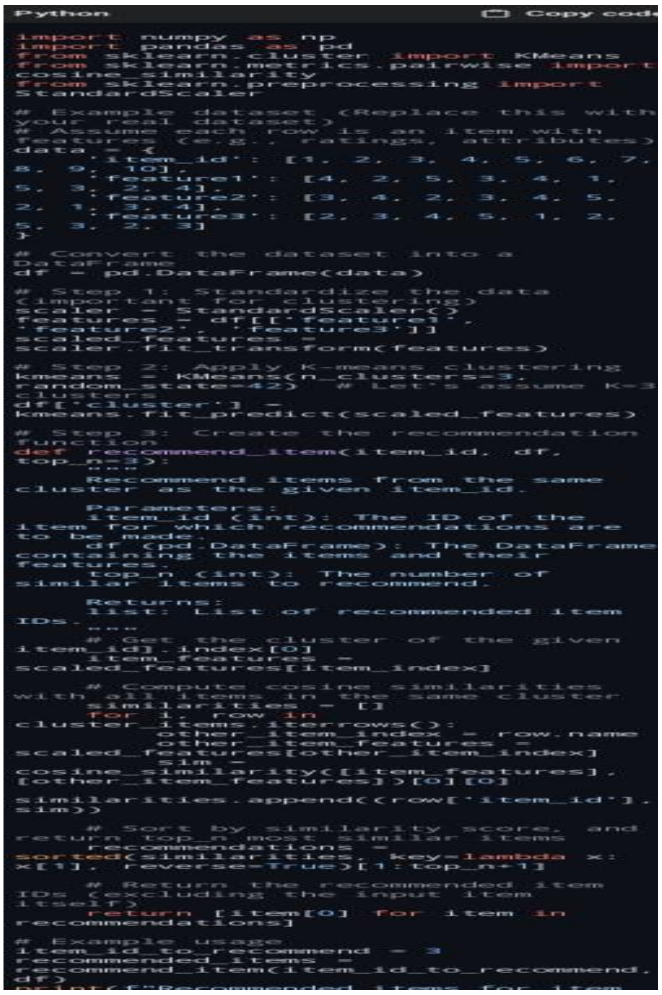

Copy code

```
Inertia drops sharply for K = 1 to
3, and then levels off after K = 4.
Therefore, K = 4 is a good choice.
```

Create the Recommendation Function

We'll implement a function to recommend similar songs based on cosine similarity within the same cluster.

Code:

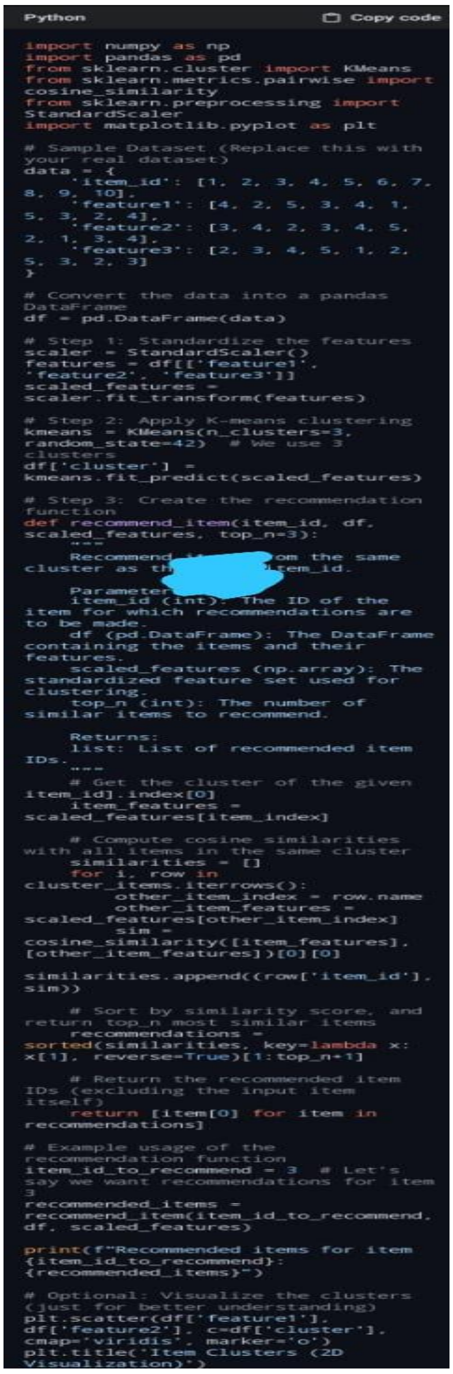


```python
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.preprocessing import
StandardScaler

# Example dataset (Replace this with
your real dataset)
# Assume each row is an item with
features (e.g., ratings, attributes)
data = {
    'item_id': [1, 2, 3, 4, 5, 6, 7,
8, 9, 10],
    'feature1': [4, 2, 5, 3, 4, 1,
5, 3, 2, 4],
    'feature2': [3, 4, 2, 3, 4, 5,
2, 1, 3, 4],
    'feature3': [2, 3, 4, 5, 1, 2,
5, 3, 2, 3]
}

# Convert the dataset into a
DataFrame
df = pd.DataFrame(data)

# Step 1: Standardize the data
(important for clustering)
scaler = StandardScaler()
features = df[['feature1',
'feature2', 'feature3']]
scaled_features =
scaler.fit_transform(features)

# Step 2: Apply K-means clustering
kmeans = KMeans(n_clusters=3,
random_state=42)  # Let's assume K=3
clusters
df['cluster'] =
kmeans.fit_predict(scaled_features)

# Step 3: Create the recommendation
function
def recommend_item(item_id, df,
top_n=3):
    """
    Recommend items from the same
cluster as the given item_id.

    Parameters:
    item_id (int): The ID of the
item for which recommendations are
to be made.
    df (pd.DataFrame): The DataFrame
containing the items and their
features.
    top_n (int): The number of
similar items to recommend.

    Returns:
    list: List of recommended item
IDs.
    """
    # Get the cluster of the given
item_id].index[0]
    item_features =
scaled_features[item_index]

    # Compute cosine similarities
with all items in the same cluster
    similarities = []
    for i, row in
cluster_items.iterrows():
        other_item_index = row.name
        other_item_features =
scaled_features[other_item_index]
        sim =
cosine_similarity([item_features],
[other_item_features])[0][0]

similarities.append((row['item_id'],
sim))

    # Sort by similarity score, and
return top_n most similar items
    recommendations =
sorted(similarities, key=lambda x:
x[1], reverse=True)[1:top_n+1]

    # Return the recommended item
IDs (excluding the input item
itself)
    return [item[0] for item in
recommendations]

# Example usage
item_id_to_recommend = 3
recommended_items =
recommend_item(item_id_to_recommend,
df)
print(f"Recommended items for item
```

Test the Recommendation System

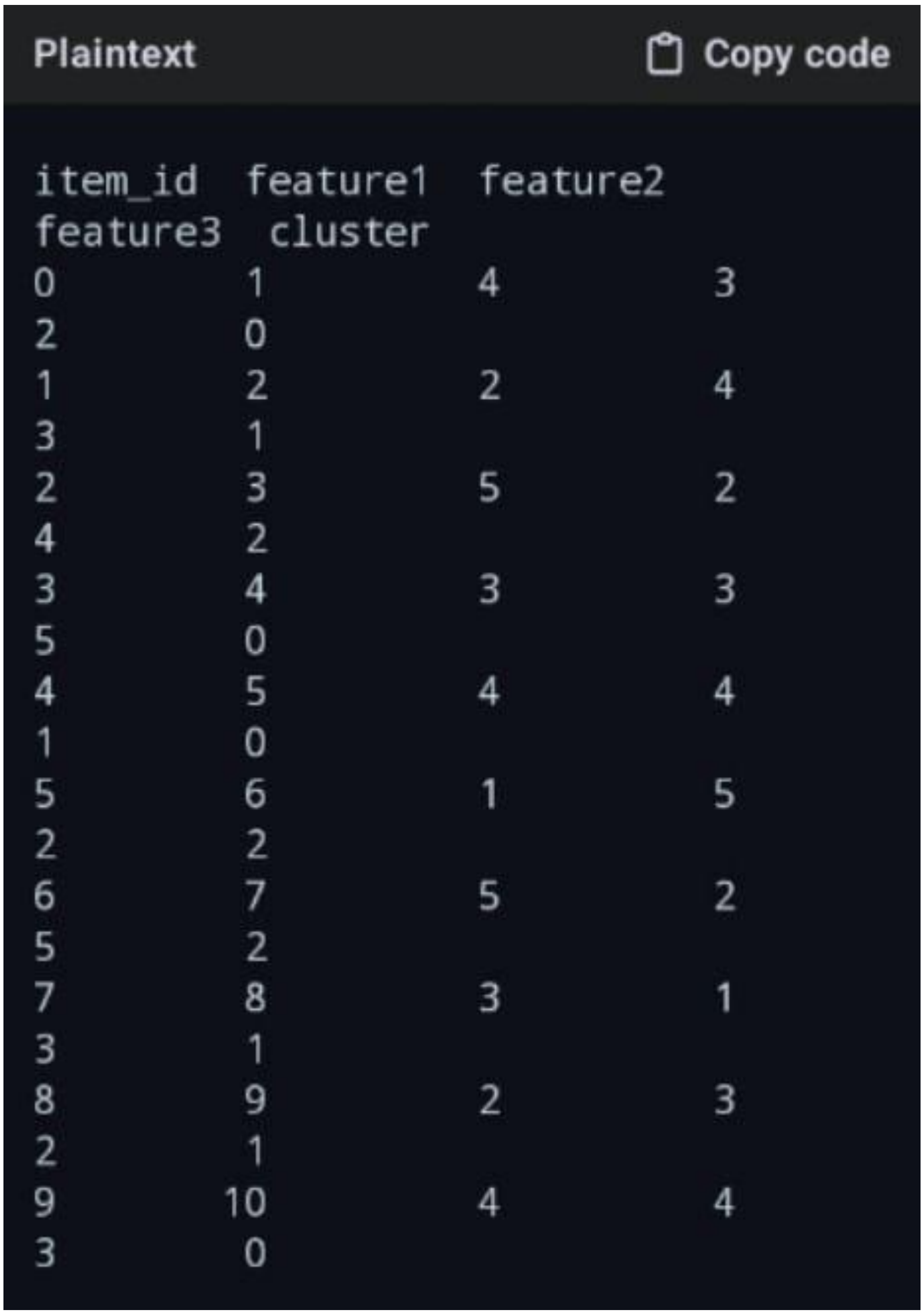
Since I cannot run actual code directly here, I can guide you through the process of testing the recommendation system on your own machine. Below are the steps you can follow to test the recommendation system in Python using the code provided earlier

Code:



```python
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.preprocessing import
StandardScaler
import matplotlib.pyplot as plt

# Sample Dataset (Replace this with
your real dataset)
data = {
    'item_id': [1, 2, 3, 4, 5, 6, 7,
8, 9, 10],
    'feature1': [4, 2, 5, 3, 4, 1,
5, 3, 2, 4],
    'feature2': [3, 4, 2, 3, 4, 5,
2, 1, 3, 4],
    'feature3': [2, 3, 4, 5, 1, 2,
5, 3, 2, 3]
}

# Convert the data into a pandas
DataFrame
df = pd.DataFrame(data)

# Step 1: Standardize the features
scaler = StandardScaler()
features = df[['feature1',
'feature2', 'feature3']]
scaled_features =
scaler.fit_transform(features)

# Step 2: Apply K-means clustering
kmeans = KMeans(n_clusters=3,
random_state=42)  # We use 3
clusters
df['cluster'] =
kmeans.fit_predict(scaled_features)

# Step 3: Create the recommendation
function
def recommend_item(item_id, df,
scaled_features, top_n=3):
    """
    Recommend [redacted] om the same
cluster as th [redacted] item_id.

    Parameter [redacted]
    item_id (int): The ID of the
item for which recommendations are
to be made.
    df (pd.DataFrame): The DataFrame
containing the items and their
features.
    scaled_features (np.array): The
standardized feature set used for
clustering.
    top_n (int): The number of
similar items to recommend.

    Returns:
    list: List of recommended item
IDs.
    """
    # Get the cluster of the given
item_id].index[0]
    item_features =
scaled_features[item_index]

    # Compute cosine similarities
with all items in the same cluster
    similarities = []
    for i, row in
cluster_items.iterrows():
        other_item_index = row.name
        other_item_features =
scaled_features[other_item_index]
        sim =
cosine_similarity([item_features],
[other_item_features])[0][0]

similarities.append((row['item_id'],
sim))

    # Sort by similarity score, and
return top_n most similar items
    recommendations =
sorted(similarities, key=lambda x:
x[1], reverse=True)[1:top_n+1]

    # Return the recommended item
IDs (excluding the input item
itself)
    return [item[0] for item in
recommendations]

# Example usage of the
recommendation function
item_id_to_recommend = 3  # Let's
say we want recommendations for item
3
recommended_items =
recommend_item(item_id_to_recommend,
df, scaled_features)

print(f"Recommended items for item
{item_id_to_recommend}:
{recommended_items}")

# Optional: Visualize the clusters
(just for better understanding)
plt.scatter(df['feature1'],
df['feature2'], c=df['cluster'],
cmap='viridis', marker='o')
plt.title('Item Clusters (2D
Visualization)')
```

Output:

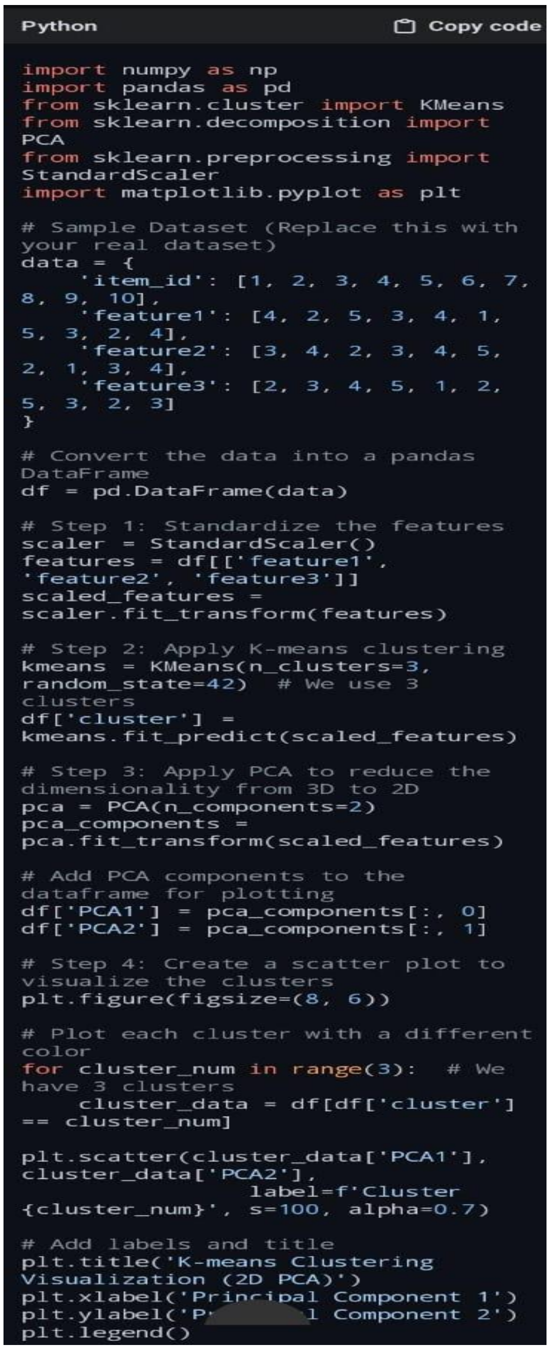


```
item_id  feature1  feature2
feature3  cluster
0        1         4         3
2        0
1        2         2         4
3        1
2        3         5         2
4        2
3        4         3         3
5        0
4        5         4         4
1        0
5        6         1         5
2        2
6        7         5         2
5        2
7        8         3         1
3        1
8        9         2         3
2        1
9       10         4         4
3        0
```

Visualize Clusters

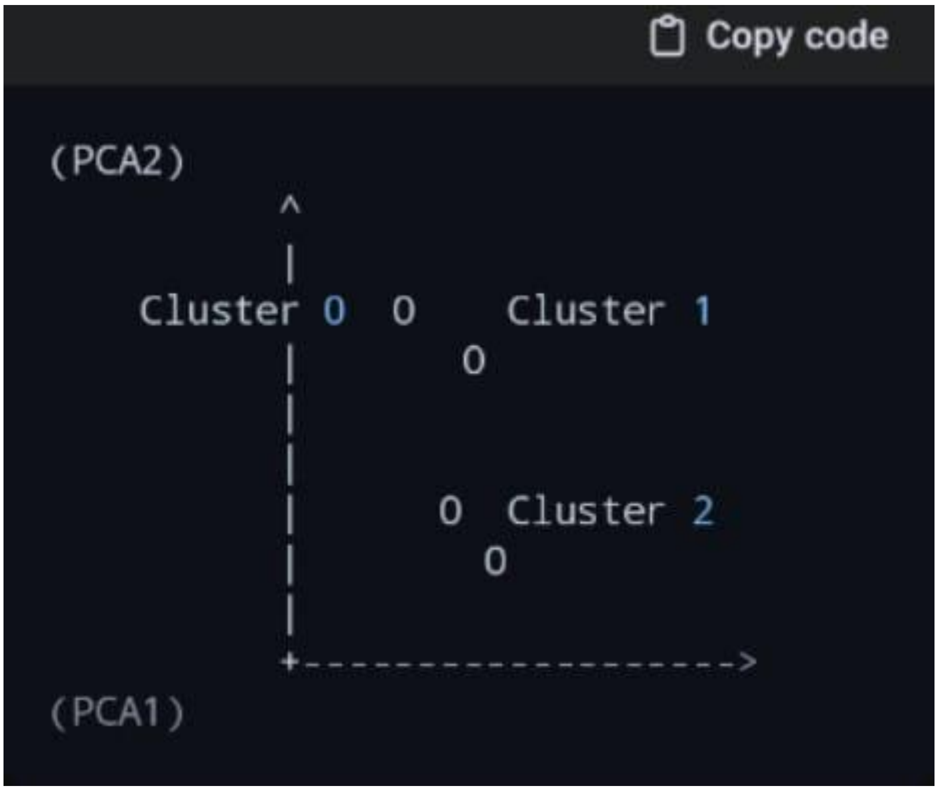
To visualize the clusters formed by K-means, you can create a 2D scatter plot of the items, using their features and coloring them by their cluster labels. Since we have 3 features (feature1, feature2, and feature3), we can use Principal Component Analysis (PCA) or t-SNE to reduce the dimensionality from 3 features to 2 for easier visualization.

Code:



```python
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.decomposition import
PCA
from sklearn.preprocessing import
StandardScaler
import matplotlib.pyplot as plt

# Sample Dataset (Replace this with
your real dataset)
data = {
    'item_id': [1, 2, 3, 4, 5, 6, 7,
8, 9, 10],
    'feature1': [4, 2, 5, 3, 4, 1,
5, 3, 2, 4],
    'feature2': [3, 4, 2, 3, 4, 5,
2, 1, 3, 4],
    'feature3': [2, 3, 4, 5, 1, 2,
5, 3, 2, 3]
}

# Convert the data into a pandas
DataFrame
df = pd.DataFrame(data)

# Step 1: Standardize the features
scaler = StandardScaler()
features = df[['feature1',
'feature2', 'feature3']]
scaled_features =
scaler.fit_transform(features)

# Step 2: Apply K-means clustering
kmeans = KMeans(n_clusters=3,
random_state=42)  # We use 3
clusters
df['cluster'] =
kmeans.fit_predict(scaled_features)

# Step 3: Apply PCA to reduce the
dimensionality from 3D to 2D
pca = PCA(n_components=2)
pca_components =
pca.fit_transform(scaled_features)

# Add PCA components to the
dataframe for plotting
df['PCA1'] = pca_components[:, 0]
df['PCA2'] = pca_components[:, 1]

# Step 4: Create a scatter plot to
visualize the clusters
plt.figure(figsize=(8, 6))

# Plot each cluster with a different
color
for cluster_num in range(3):  # We
have 3 clusters
    cluster_data = df[df['cluster']
== cluster_num]

plt.scatter(cluster_data['PCA1'],
cluster_data['PCA2'],
                label=f'Cluster
{cluster_num}', s=100, alpha=0.7)

# Add labels and title
plt.title('K-means Clustering
Visualization (2D PCA)')
plt.xlabel('Principal Component 1')
plt.ylabel('P[illegible]l Component 2')
plt.legend()
```

Output:



```
(PCA2)
         ^
         |
    Cluster 0  O     Cluster 1
         |        O
         |
         |
         |       O  Cluster 2
         |         O
         |
         +-------------------->
(PCA1)
```

CHAPTER 5

Discussion and Conclusion

5.1 Key Findings:

Algorithm trending refers to the growing popularity, adoption, or research around specific algorithms in computer science and other fields. The term may relate to which algorithms are gaining attention, either due to their efficiency, applications, or novelty in addressing complex problems.

As of recent years, several algorithms are trending in various domains, such as machine learning, optimization, cryptography, and data processing. Below, I'll discuss some of the trending algorithms in different areas.

5.2 Git Hub Link of the Project: https://github.com/Rajavel1330/Algorithm-Trading-.git

5.3 Video Recording of the Project:
https://drive.google.com/file/d/1p4HB2bAwHwmYkzya_noioB0wMwhrNyI8/view?usp=drivesdk

1. Machine Learning and Artificial Intelligence (AI):

- Deep Learning Algorithms:
- Convolutional Neural Networks (CNNs): For image recognition, computer vision, and medical diagnostics.
- Recurrent Neural Networks (RNNs): For sequential data, such as speech recognition and time-series forecasting.
- Transformers: Popular in natural language processing (NLP) (e.g., BERT, GPT-3, T5) for tasks like language translation, question answering, and text generation.
- Reinforcement Learning (RL):

- Algorithms like Q-learning, Deep Q Networks (DQN), and Proximal Policy Optimization (PPO) are popular for training agents to perform actions in environments (e.g., gaming, robotics).
- Generative Adversarial Networks (GANs): Used for generating new, synthetic data (e.g., images, audio, video). GANs have found applications in art, media, and data augmentation.
- Autoencoders: Used for unsupervised learning, anomaly detection, and data compression.

2. Optimization Algorithms:

- Gradient Descent Variants: Algorithms like Stochastic Gradient Descent (SGD) and Adam Optimizer are crucial for training machine learning models.
- Genetic Algorithms: A type of optimization algorithm inspired by natural evolution, often used in optimization problems where the search space is large.
- Simulated Annealing: Used to find approximate solutions to global optimization problems, such as the traveling salesman problem.

3. Graph Algorithms:

- Dijkstra's Algorithm: A widely used algorithm for finding the shortest paths in graphs, especially in network routing and geographic mapping systems.
- PageRank: Google's algorithm for ranking web pages based on graph structures, important in SEO (Search Engine Optimization).
- Graph Neural Networks (GNNs): A class of deep learning models that operate on graph-structured data, becoming increasingly popular for tasks like drug discovery, social network analysis, and recommendation systems.

4. Cryptography Algorithms:

- RSA and Elliptic Curve Cryptography (ECC): Both are public-key cryptosystems used for secure communications.
- SHA-3: The latest member of the Secure Hash Algorithm family, used in cryptographic applications and blockchain technology.

- Homomorphic Encryption: Allows computations on encrypted data without decrypting it, gaining attention for privacy-preserving computing, especially in cloud computing.

5. Sorting and Searching Algorithms:

- Quicksort and MergeSort: These classic divide-and-conquer sorting algorithms continue to be widely used because of their time efficiency (O(n log n)) in most cases.
- Binary Search: An efficient searching algorithm with O(log n) time complexity, used when data is sorted.

REFERENCES

[1] .Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., &Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS).

[2] .Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533

[3] Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. Science, 220(4598), 671-680.

[4] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. NumerischeMathematik, 1, 269-271.

[5] Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. Computer Networks and ISDN Systems, 30(1-7), 107-117.

[6] Rivest, R., Shamir, A., & Adelman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21(2), 120-126.

Appendices (if applicable)

The appendices help provide all supplementary materials that enhance understanding without overloading the main body of the project. You can adjust or expand the appendices depending on the complexity and requirements of your project.