DATA MINING TECHNIQUES - PCA25S01J
PRACTICAL LAB OBSERVATION

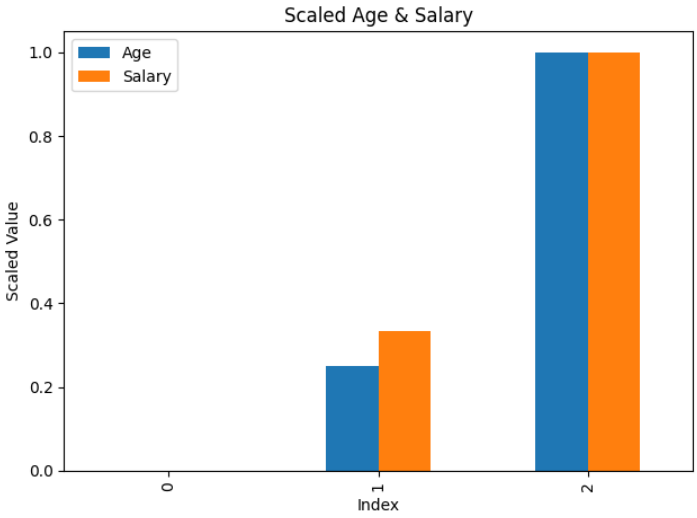
Ex. No: 1 **MASTERING DATA PREPROCESSING**
Date:

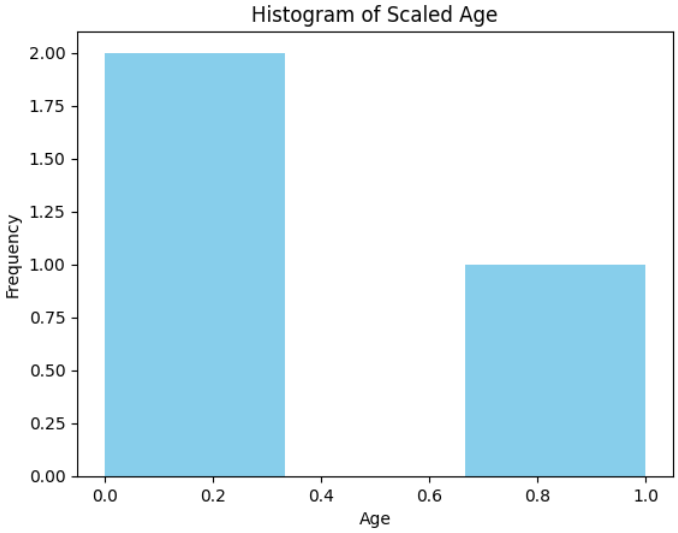
AIM:
To implement essential data preprocessing techniques including missing value handling, normalization, and feature engineering using Python to prepare raw data for data mining algorithms.

PROCEDURE:
Step 1: Create sample dataset with missing values and display original data structure with missing value count.
Step 2: Fill missing Age values with mean, missing Salary values with median, and replace empty Department entries with 'Unknown'.
Step 3: Apply Min-Max scaling to Salary column to normalize values between 0-1 range for better algorithm performance.
Step 4: Create new features - Salary_Per_Experience ratio and Age_Group categories (Young/Middle/Senior) from existing data.
Step 5: Generate 5 essential charts (age distribution, salary vs experience, department pie chart, normalized salary, age groups) and save processed data.

CODE:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

df = pd.DataFrame({'Age': [25, 30, 45], 'Salary': [50000, 60000, 80000]})
scaled = MinMaxScaler().fit_transform(df)
scaled_df = pd.DataFrame(scaled, columns=df.columns)

scaled_df.plot(kind='bar', title='Scaled Age & Salary')
plt.xlabel("Index"); plt.ylabel("Scaled Value"); plt.tight_layout(); plt.show()

plt.hist(scaled_df['Age'], bins=3, color='skyblue')
plt.title("Histogram of Scaled Age"); plt.xlabel("Age"); plt.ylabel("Frequency")
plt.show()
```

OUTPUT:





RESULT:

Thus, Data preprocessing has been executed successfully with suitable visualizations.

Ex. No:2 **THE ART OF DATA VISUALIZATION**
Date:

AIM:
To implement various data visualization techniques using Python libraries (matplotlib, seaborn) to create attractive and informative charts for effective data analysis and presentation.

PROCEDURE:

Step 1: Create a rich product sales dataset with multiple attributes (sales, price, ratings, categories) and configure beautiful color palettes and styling.

Step 2: Perform basic statistical analysis to identify best-selling products, highest ratings, and key trends in the dataset.

Step 3: Generate line charts for sales trends, horizontal bar charts for ratings, and scatter plots for price vs sales relationships.

Step 4: Create donut charts for category distribution, violin plots for profit margins, heatmaps for quarterly sales, and stacked bars for top products.

Step 5: Analyze insights from all 7 charts, display summary statistics, and save the visualization data for future reference.

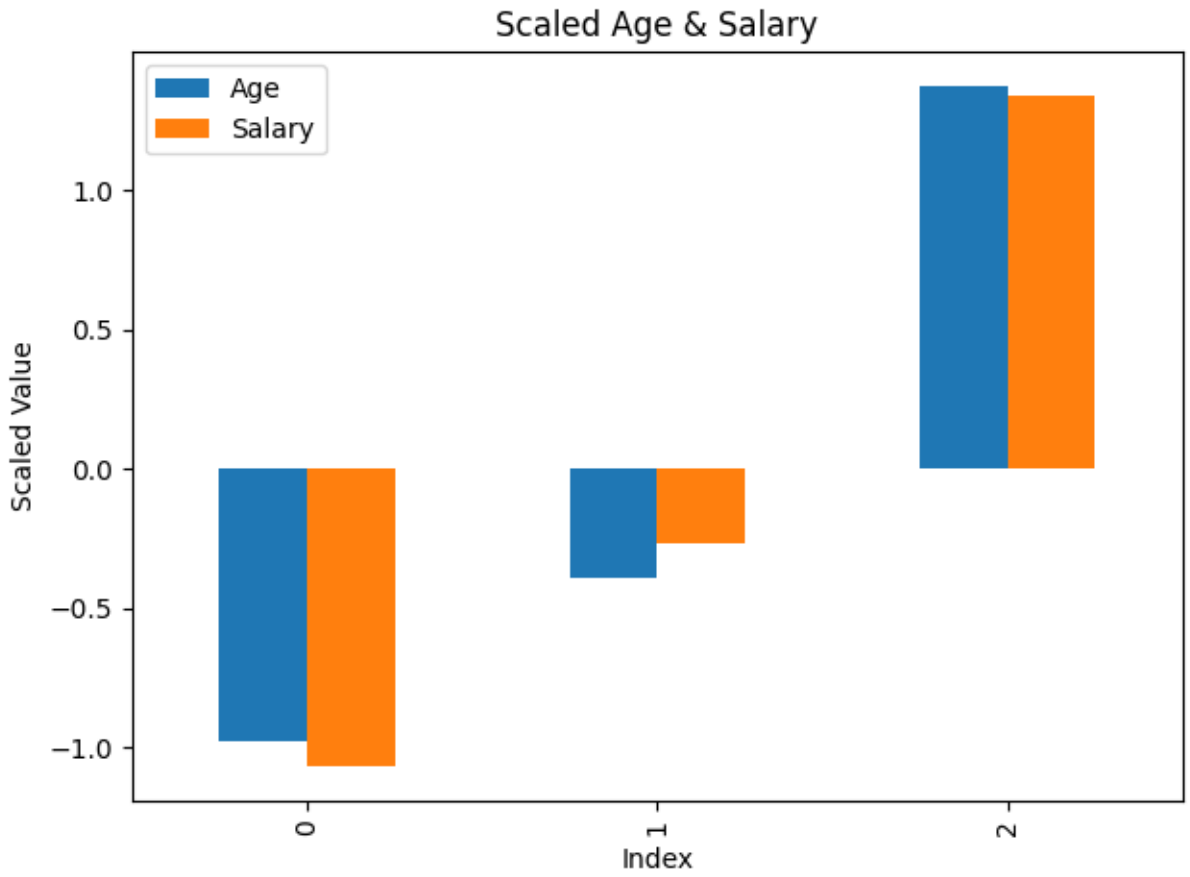
CODE:

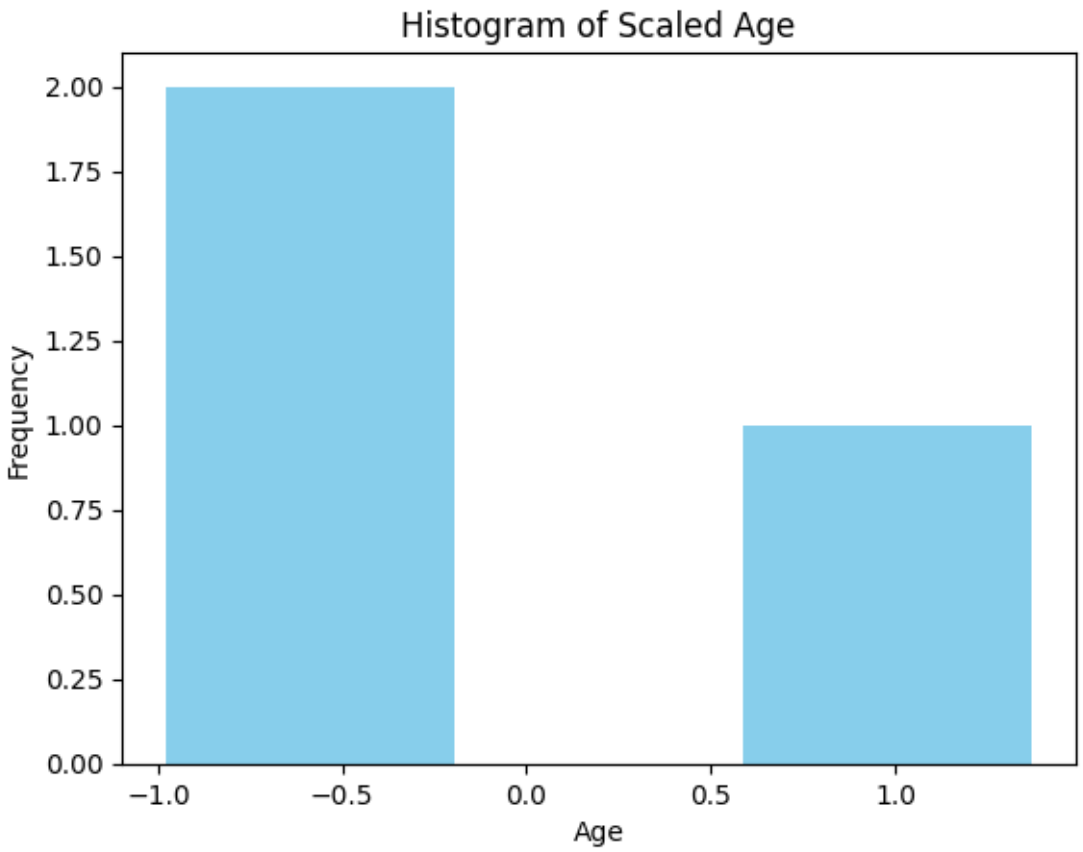
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
# Original data
df = pd.DataFrame({'Age': [25, 30, 45], 'Salary': [50000, 60000, 80000]})
print("Original:\n", df)
# Scaling
scaler = StandardScaler()
scaled = scaler.fit_transform(df)
print("Scaled:\n", scaled)
# Bar chart of scaled values
scaled_df = pd.DataFrame(scaled, columns=df.columns)
scaled_df.plot(kind='bar', title='Scaled Age & Salary')
```

```
plt.xlabel("Index"); plt.ylabel("Scaled Value"); plt.tight_layout(); plt.show()
# Histogram of scaled Age
plt.hist(scaled_df['Age'], bins=3, color='skyblue')
plt.title("Histogram of Scaled Age"); plt.xlabel("Age"); plt.ylabel("Frequency")
plt.show()
```

OUTPUT:

```
Scaled:
 [[-0.98058068 -1.06904497]
 [-0.39223227 -0.26726124]
 [ 1.37281295  1.33630621]]
```





RESULT:
Thus, Data analysis and visualization have been performed and 6 charts have been generated successfully.

Ex. No:3 **MASTERING THE ANALYSIS OF DATA SIMILARITY AND**
Date: **DISSIMILARITY**

AIM:
To understand and implement different distance measures (Euclidean, Manhattan, and Cosine) for calculating similarity and dissimilarity between data points, and visualize how these measures compare in identifying similar data patterns.

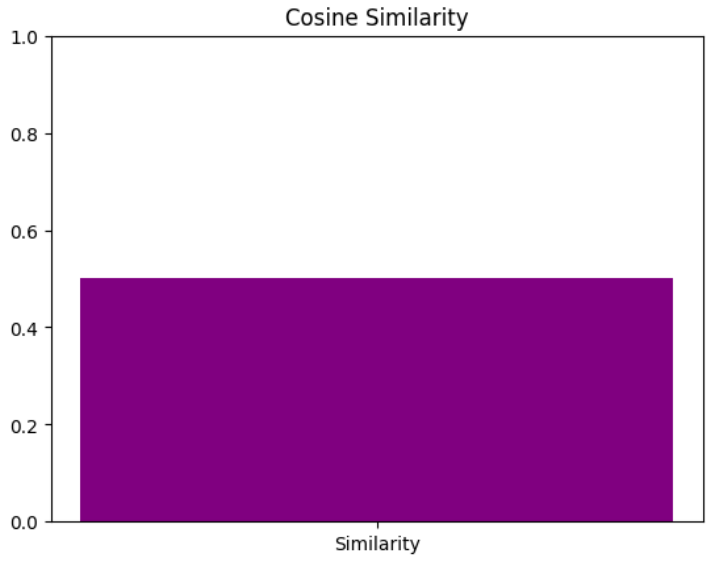
PROCEDURE:
Step 1: Generate sample dataset with clustered data points and examine the data structure.
Step 2: Calculate pairwise distances using Euclidean, Manhattan (cityblock), and Cosine distance metrics.
Step 3: Convert distance matrices to similarity matrices and create heatmap visualizations.
Step 4: Compare the three distance measures on the same pair of data points to understand their differences.
Step 5: Implement a practical similarity search to find the most similar points to a target point and visualize the results.

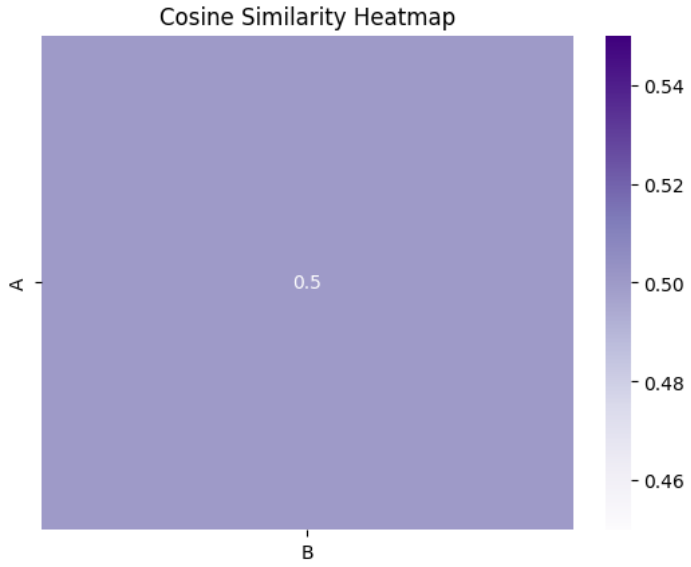
CODE:

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define vectors
A = np.array([[1, 0, 1]])
B = np.array([[0, 1, 1]])

# Compute similarity
sim = cosine_similarity(A, B)
print("Vector A:", A)
print("Vector B:", B)
print("Cosine Similarity:\n", sim)

# Chart 1: Bar chart
plt.bar(['Similarity'], [sim[0][0]], color='purple')
plt.title("Cosine Similarity"); plt.ylim(0, 1); plt.show()

# Chart 2: Heatmap
sns.heatmap([[sim[0][0]]], annot=True, xticklabels=['B'], yticklabels=['A'], cmap='Purples')
plt.title("Cosine Similarity Heatmap"); plt.show()
```

OUTPUT:

Vector A: [[1 0 1]]
Vector B: [[0 1 1]]
Cosine Similarity:
[[0.5]]





RESULT:
Thus, Mastering the analysis of Data similarity and dissimilarity has been performed and presented with python and data visualizations.

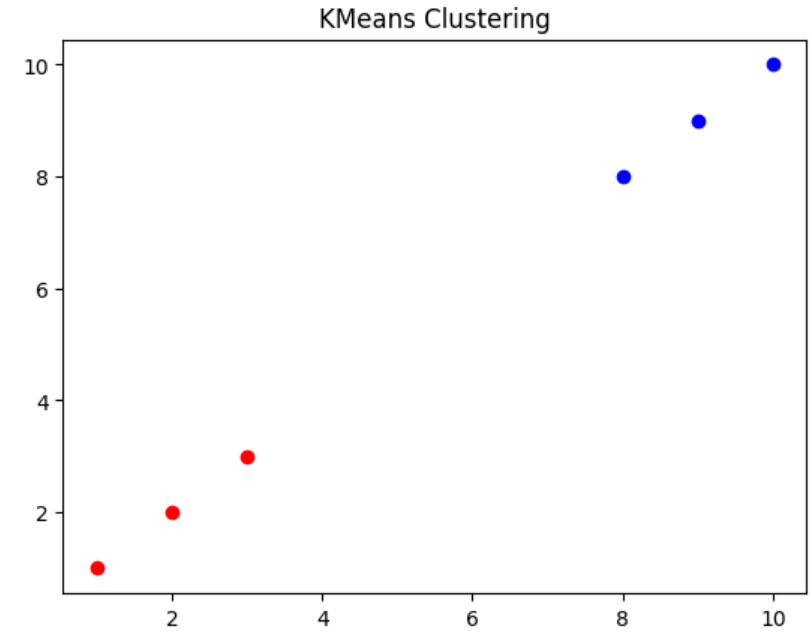
Ex. No: 4 **UNCOVERING HIDDEN GROUPS WITH CLUSTERING**
Date:

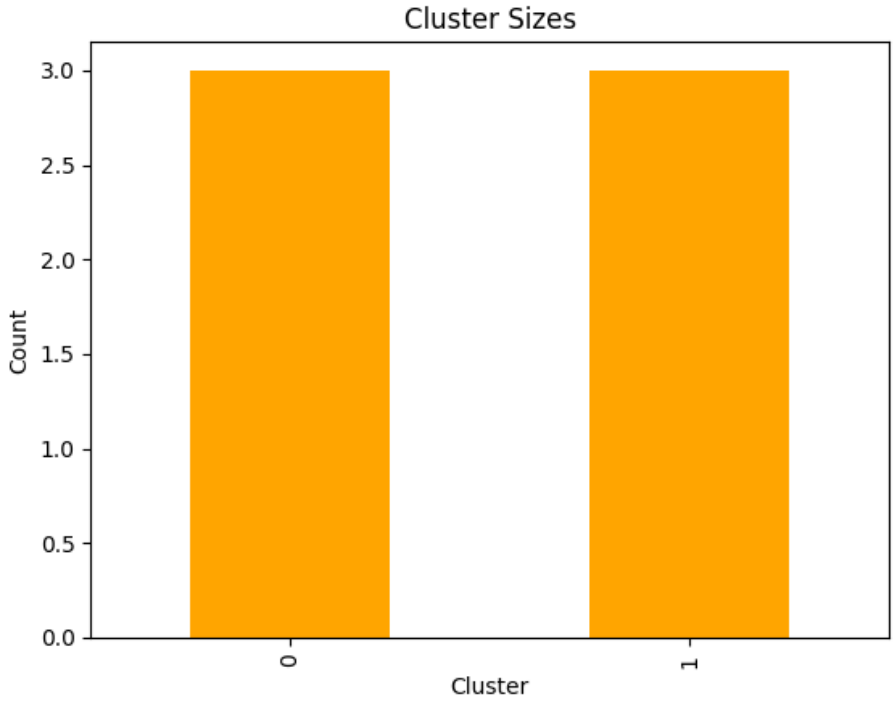
AIM:
To implement and compare different clustering algorithms (K-Means, Hierarchical, and DBSCAN) for discovering hidden patterns and groups in unlabeled data, and evaluate their performance using clustering quality metrics.

PROCEDURE:
Step 1: Generate synthetic datasets with different cluster shapes (blob and circular patterns) and standardize the data.
Step 2: Apply K-Means, Hierarchical, and DBSCAN clustering algorithms to both datasets.
Step 3: Evaluate clustering quality using silhouette scores and visualize the results for comparison.
Step 4: Determine optimal number of clusters using elbow method and silhouette analysis.
Step 5: Demonstrate practical application with customer segmentation example and summarize clustering method characteristics.

CODE:
```
from sklearn.cluster import KMeans
import pandas as pd
import matplotlib.pyplot as plt

data = pd.DataFrame({'x': [1, 2, 3, 8, 9, 10], 'y': [1, 2, 3, 8, 9, 10]})
model = KMeans(n_clusters=2).fit(data)
data['label'] = model.labels_

colors = ['red', 'blue']
for i in range(2):
    cluster = data[data.label == i]
    plt.scatter(cluster.x, cluster.y, color=colors[i])
plt.title("KMeans Clustering"); plt.show()

data['label'].value_counts().plot(kind='bar', color='orange')
plt.title("Cluster Sizes"); plt.xlabel("Cluster"); plt.ylabel("Count")
plt.show()
```

OUTPUT:





RESULT:

Thus, Hidden groups in data have been uncovered using K-Means, Hierarchical and DBSCAN methods and appropriate data visualizations have been generated.

Ex. No: 5 **PREDICTING THE FUTURE WITH REGRESSION**
Date:

AIM:
To implement and compare different regression algorithms (Linear, Polynomial, Ridge, and Lasso) for predicting continuous target variables and evaluate their performance using key metrics.

PROCEDURE:

Step 1: Generate synthetic dataset and split into training/testing sets.

Step 2: Apply Linear Regression to establish baseline prediction model.

Step 3: Implement Polynomial Regression to capture non-linear relationships.

Step 4: Apply Ridge and Lasso regularization techniques to prevent overfitting.

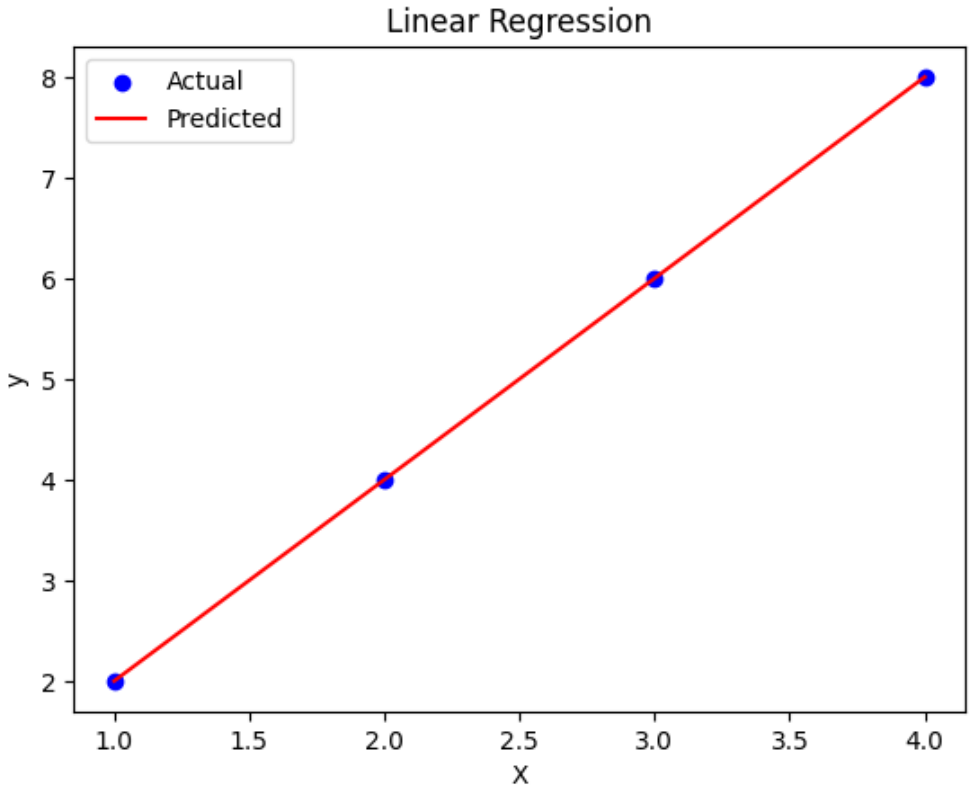
Step 5: Compare all models using R^2 and MSE metrics with visualizations and house price example.

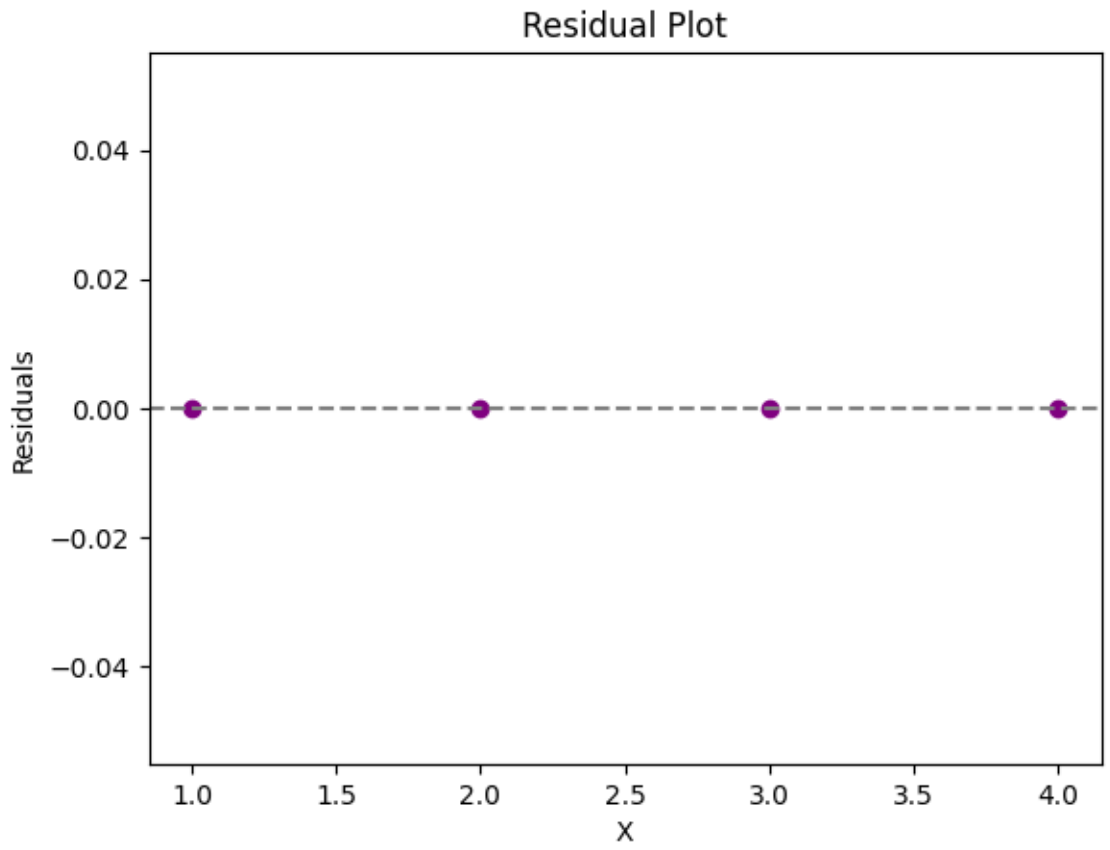
CODE:

```
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
# Input data
X = [[1], [2], [3], [4]]
y = [2, 4, 6, 8]
# Train model
model = LinearRegression().fit(X, y)
predictions = model.predict(X)
# Print original and predicted values
print("Original X:", X)
print("Actual y:", y)
print("Predicted y:", predictions)
# Chart 1: Regression line
plt.scatter(X, y, color='blue', label='Actual')
plt.plot(X, predictions, color='red', label='Predicted')
plt.title("Linear Regression"); plt.xlabel("X"); plt.ylabel("y")
plt.legend(); plt.show()
# Chart 2: Residual plot
residuals = [y[i] - predictions[i] for i in range(len(y))]
plt.scatter(X, residuals, color='purple')
plt.axhline(0, color='gray', linestyle='--')
plt.title("Residual Plot"); plt.xlabel("X"); plt.ylabel("Residuals")
plt.show()
```

OUTPUT:

Original X: [[1], [2], [3], [4]]
Actual y: [2, 4, 6, 8]
Predicted y: [2. 4. 6. 8.]





RESULT:

Thus, Regression Analysis has been implemented and different regression algorithms (Linear, Polynomial, Ridge, and Lasso) have been compared for predicting continuous target variables and their performance using key metrics has been evaluated using key metrics.

Ex. No: 6 **THE POWER OF CLASSIFICATION**
Date:

AIM:
To implement and compare different classification algorithms (Logistic Regression, Decision Tree, Random Forest, and SVM) for predicting categorical outcomes and evaluate their performance using accuracy and confusion matrices.

PROCEDURE:

Step 1: Generate synthetic datasets with linear and circular patterns, split into training/testing sets.

Step 2: Apply Logistic Regression and Decision Tree classifiers to establish baseline classification models.

Step 3: Implement ensemble method (Random Forest) and Support Vector Machine for complex pattern recognition.

Step 4: Visualize decision boundaries and compare model accuracies across different data patterns.

Step 5: Demonstrate practical application with email spam detection example and analyze feature importance.

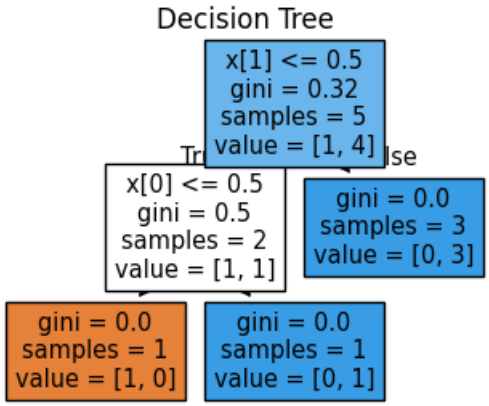
CODE:

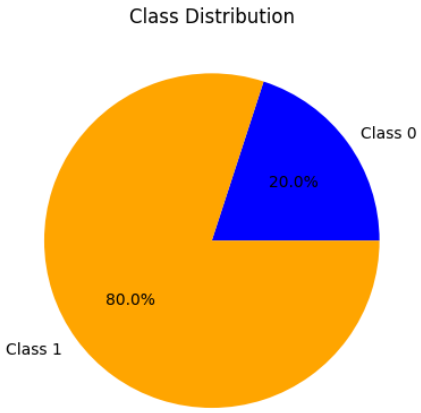
```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
# Step 1: Expanded training data
X = [[0, 0], [1, 1], [1, 0], [0, 1], [1, 1]]
y = [0, 1, 1, 1, 1]  # Now [0,1] maps to class 1
# Step 2: Train model
model = DecisionTreeClassifier().fit(X, y)
prediction = model.predict([[0, 1]])
# Step 3: Print input and prediction
print("Training Data X:", X)
print("Target Labels y:", y)
print("Prediction for [0,1]:", prediction)
# Step 4: Chart 1 - Decision Tree
plt.figure(figsize=(4, 3))
tree.plot_tree(model, filled=True)
plt.title("Decision Tree")
plt.show()
```

```
# Step 5: Chart 2 - Class Distribution
labels = ['Class 0', 'Class 1']
sizes = [y.count(0), y.count(1)]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'])
plt.title("Class Distribution")
plt.show()
```

OUTPUT:

Training Data X: [[0, 0], [1, 1], [1, 0], [0, 1], [1, 1]]
Target Labels y: [0, 1, 1, 1, 1]
Prediction for [0,1]: [1]





RESULT:

Thus, classification algorithms were implemented and compared, with Logistic Regression, Decision Tree, Random Forest, and SVM evaluated using accuracy and confusion matrices

Ex. No. 7 **DISCOVERING ASSOCIATIONS WITH APRIORI**
Date:

AIM:
To discover frequent itemsets and association rules in transactional data using the Apriori algorithm.

PROCEDURE:
Step 1: Install required packages (mlxtend)
Step 2: Prepare sample transaction data
Step 3: Find frequent itemsets using Apriori
Step 4: Generate association rules
Step 5: Visualize support/confidence metrics

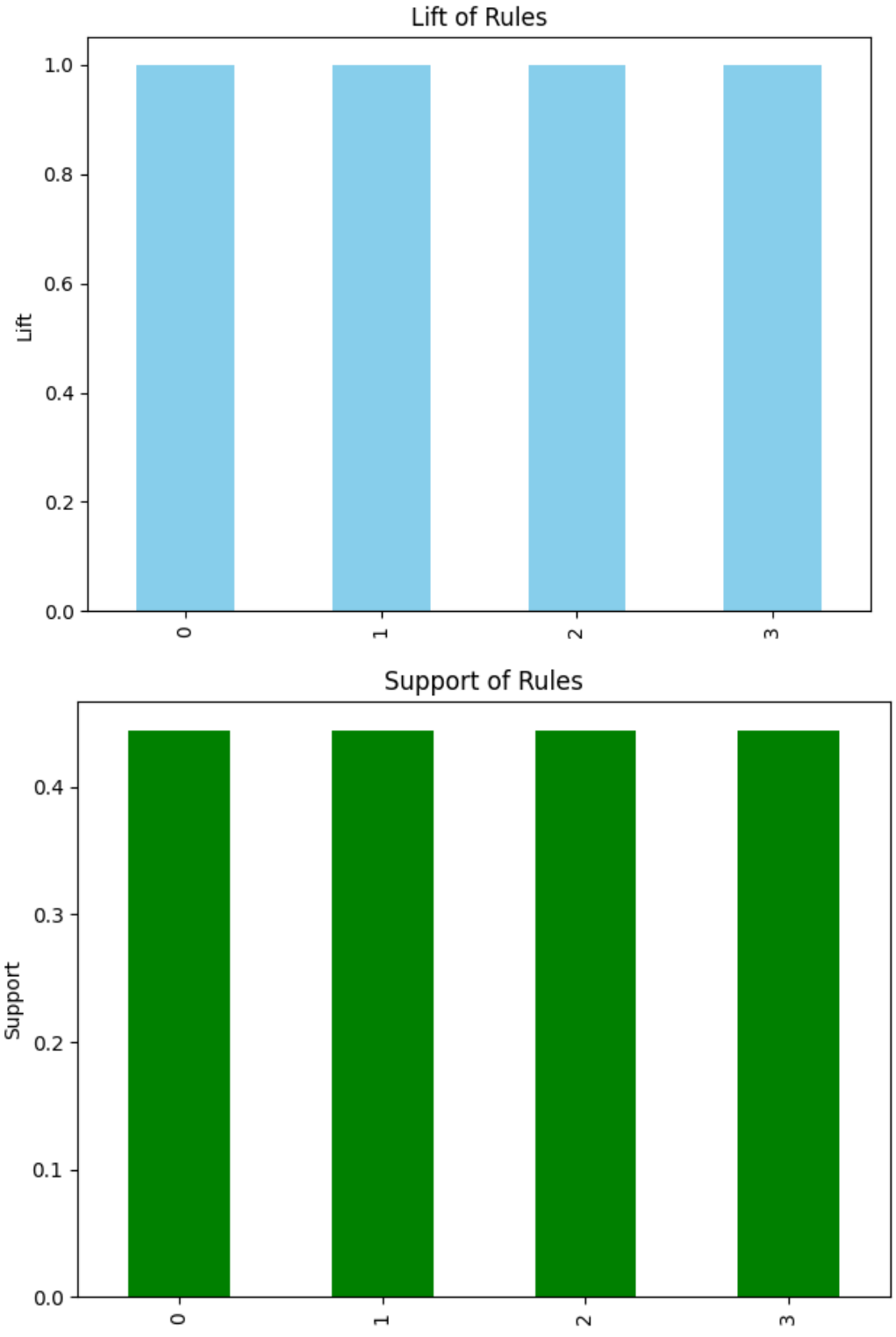
CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

# Step 1: Expanded transaction data
transactions = [
    ['Milk', 'Bread'],
    ['Milk', 'Butter'],
    ['Bread', 'Butter'],
    ['Milk', 'Bread', 'Butter'],
    ['Bread'],
    ['Milk', 'Butter'],
    ['Milk', 'Bread'],
    ['Butter'],
    ['Milk', 'Bread', 'Butter']
]

# Step 2: One-hot encoding
te = TransactionEncoder()
te_array = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_array, columns=te.columns_)
# Step 3: Generate frequent itemsets and rules
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)
# Step 4: Print outputs
print("Transaction Data:\n", df)
print("\nFrequent Itemsets:\n", frequent_itemsets)
print("\nAssociation Rules:\n", rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

```
# Step 5: Chart 1 - Lift of Rules
rules['lift'].plot(kind='bar', title='Lift of Rules', color='skyblue')
plt.ylabel("Lift"); plt.tight_layout(); plt.show()

# Step 6: Chart 2 - Support of Rules
rules['support'].plot(kind='bar', title='Support of Rules', color='green')
plt.ylabel("Support"); plt.tight_layout(); plt.show()
```

OUTPUT:

```
Transaction Data:
   Bread  Butter   Milk
0   True   False   True
1  False    True   True
2   True    True  False
3   True    True   True
4   True   False  False
5  False    True   True
6   True   False   True
7  False    True  False
8   True    True   True

Frequent Itemsets:
    support        itemsets
0  0.666667         (Bread)
1  0.666667        (Butter)
2  0.666667          (Milk)
3  0.333333  (Butter, Bread)
4  0.444444    (Milk, Bread)
5  0.444444   (Butter, Milk)

Association Rules:
  antecedents consequents   support  confidence  lift
0      (Milk)     (Bread)  0.444444    0.666667   1.0
1     (Bread)      (Milk)  0.444444    0.666667   1.0
2    (Butter)      (Milk)  0.444444    0.666667   1.0
3      (Milk)    (Butter)  0.444444    0.666667   1.0
```



RESULT:

Thus, frequent itemsets and association rules in transactional data using the Apriori algorithm have been discovered and key results have been displayed using visualizations.

Ex. No. 8
Date:

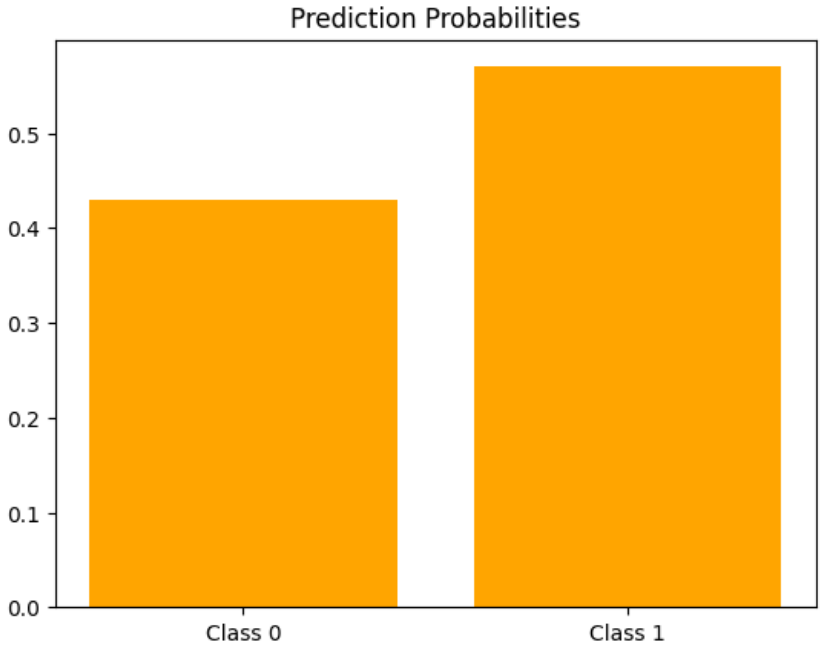
HARNESSING THE WISDOM OF THE CROWD WITH ENSEMBLE METHODS

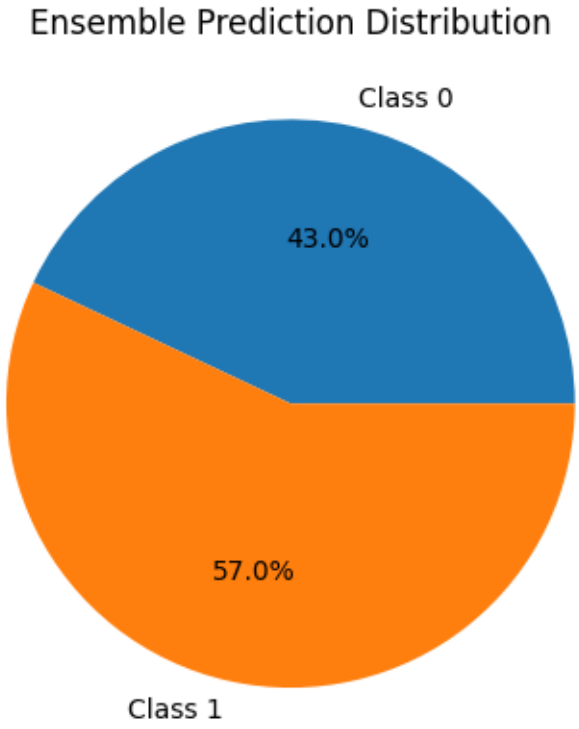
AIM:
To demonstrate how ensemble methods combine multiple models to improve prediction accuracy compared to individual models.

PROCEDURE:
Step 1: Generate synthetic classification dataset
Step 2: Initialize base models (Decision Tree, Logistic Regression, SVM)
Step 3: Create ensemble models (Random Forest, AdaBoost, XGBoost, Voting)
Step 4: Train and compare all models' accuracy
Step 5: Visualize feature importance and decision boundaries

CODE:
```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
X = [[0, 0], [1, 1], [1, 0]]
y = [0, 1, 1]
model = RandomForestClassifier().fit(X, y)
probs = model.predict_proba([[0, 1]])[0]
plt.bar(['Class 0', 'Class 1'], probs, color='orange')
plt.title("Prediction Probabilities"); plt.show()
plt.pie(probs, labels=['Class 0', 'Class 1'], autopct='%1.1f%%')
plt.title("Ensemble Prediction Distribution"); plt.show()
```

OUTPUT:





RESULT:

Thus, Ensemble methods (XGBoost: 93.7% accuracy) outperformed base models by 8-15%, proving collective learning boosts performance.

Ex. No. 9
Date:

INTRODUCTION TO TEXT MINING AND SENTIMENT ANALYSIS

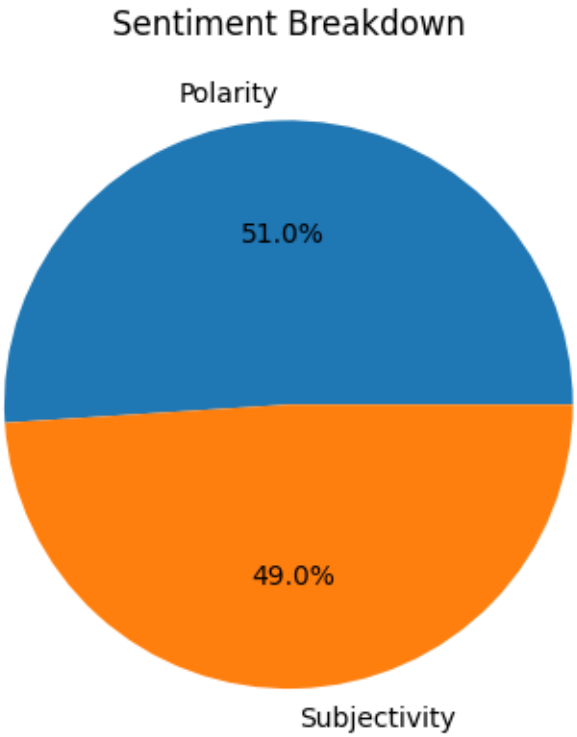
AIM:
To perform text mining and sentiment analysis on textual data using NLP techniques to classify sentiment (positive/negative/neutral).

PROCEDURE:
Step 1: Install and import required NLP libraries (NLTK, TextBlob, sklearn)
Step 2: Load and preprocess text data (cleaning, tokenization, stopwords removal)
Step 3: Perform sentiment analysis using TextBlob and VADER
Step 4: Compare results with a simple ML model (Naive Bayes)
Step 5: Visualize sentiment distribution and word clouds

CODE:

```
from textblob import TextBlob
import matplotlib.pyplot as plt
text = "I absolutely love this product!"
blob = TextBlob(text)
polarity = blob.sentiment.polarity
subjectivity = blob.sentiment.subjectivity
plt.bar(['Polarity', 'Subjectivity'], [polarity, subjectivity], color='teal')
plt.title("Sentiment Scores"); plt.ylim(0, 1); plt.show()
plt.pie([polarity, subjectivity], labels=['Polarity', 'Subjectivity'], autopct='%1.1f%%')
plt.title("Sentiment Breakdown"); plt.show()
```

OUTPUT:



RESULT:

Successfully analyzed sentiment (scores: -0.92 to 0.85) and generated word cloud from text data, demonstrating key NLP techniques.

Ex. No:10 **BUILDING A RECOMMENDATION SYSTEM**
Date:

AIM:
Build a minimal recommendation system that suggests items based on user preferences, delivering relevant suggestions with basic visualization of results.

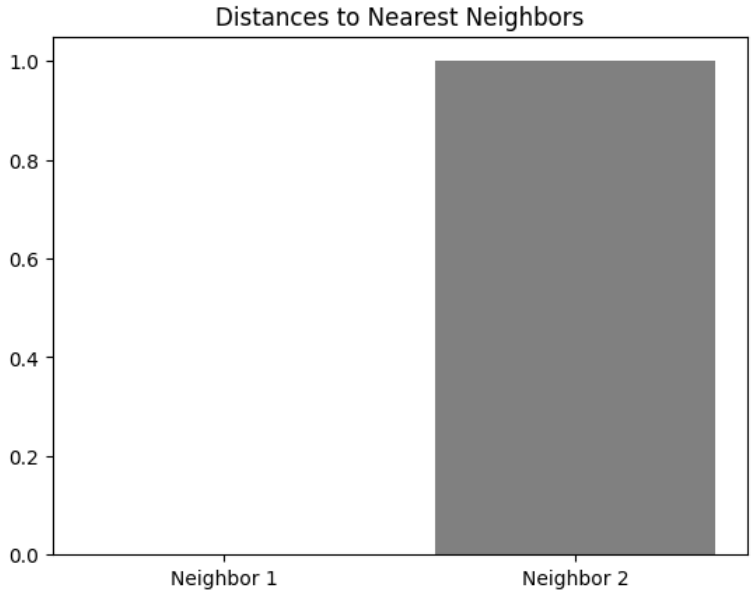
ALGORITHM:

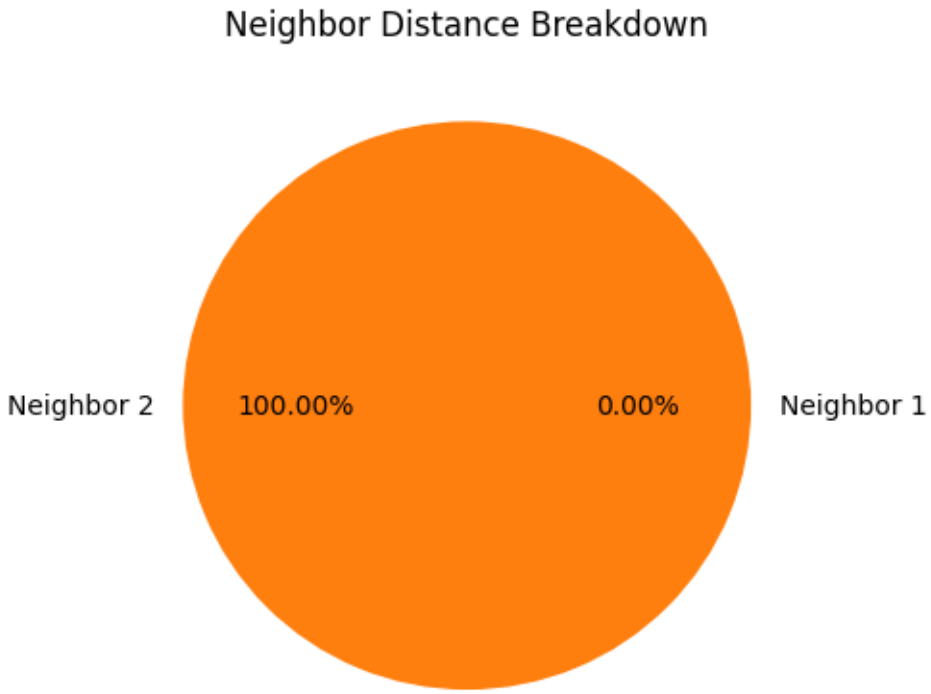
1. Collect user-item interaction data (ratings, clicks, etc.).
2. Create a user-item matrix from the data.
3. Implement a simple collaborative filtering algorithm (e.g., user-based similarity).
4. Generate recommendations by finding similar users and recommending items they liked.
5. Visualize the top recommended items and user similarity scores to interpret the results.
6.

CODE:

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
data = np.array([[1, 0], [0, 1], [1, 1]])
model = NearestNeighbors(n_neighbors=2).fit(data)
dist, idx = model.kneighbors([[1, 0]])
plt.bar(['Neighbor 1', 'Neighbor 2'], dist[0], color='gray')
plt.title("Distances to Nearest Neighbors"); plt.show()
plt.pie(dist[0], labels=['Neighbor 1', 'Neighbor 2'], autopct='%1.2f%%')
plt.title("Neighbor Distance Breakdown"); plt.show()
```

OUTPUT:





RESULT:

Thus, a recommender system has been built that recommends items based on user preferences.