DMBI Practical list
Sem: VI
TE IT A&B

#Data Preprocessing & Exploration

1. Missing Value Handling:
○ Practical: Use techniques like mean imputation, median imputation, or predictive imputation to handle missing values in datasets.
○ Dataset: You can use the "Adult Income" dataset from the UCI Machine Learning Repository, which contains missing values in various attributes such as education and occupation.
Dataset link

```
import pandas as pd

# Load the dataset
url = 
'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.dat
a'
columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 
'marital-status',
           'occupation', 'relationship', 'race', 'sex', 'capital-gain', 
'capital-loss',
           'hours-per-week', 'native-country', 'income']
df = pd.read_csv(url, names=columns, na_values=' ?', 
skipinitialspace=True)

# Check for missing values
missing_values = df.isnull().sum()
print(missing_values)
# Impute numeric columns with mean or median
numeric_cols = ['age', 'education-num', 'capital-gain', 'capital-loss', 
'hours-per-week']
for col in numeric_cols:
    if df[col].isnull().sum() > 0:
        median_val = df[col].median()
        df[col].fillna(median_val, inplace=True)

# Impute categorical columns with mode (most frequent value)
```

```
categorical_cols = ['workclass', 'education', 'marital-status',
'occupation',
                    'relationship', 'race', 'sex', 'native-country']
for col in categorical_cols:
    if df[col].isnull().sum() > 0:
        mode_val = df[col].mode()[0]
        df[col].fillna(mode_val, inplace=True)

# Check if all missing values are handled
missing_values_after = df.isnull().sum()
print(missing_values_after)
```

2. Outlier Detection:

○ **Practical: Identify and handle outliers in the data using methods like z-score, IQR (Interquartile Range), or visualization techniques.**

○ **Dataset: The "Credit Card Fraud Detection" dataset from Kaggle contains transactions with potential outliers representing fraudulent activities. Dataset link**

```
import pandas as pd
import numpy as np

# Load the dataset
df = pd.read_csv('creditcard.csv')

# Display basic information about the dataset
print(df.info())
from scipy import stats

# Calculate z-scores for numeric columns
numeric_cols = df.select_dtypes(include=['float64']).columns
z_scores = stats.zscore(df[numeric_cols])

# Absolute z-score values greater than 3 are considered outliers
threshold = 3
outlier_indices = (np.abs(z_scores) > threshold).any(axis=1)
outliers_zscore = df[outlier_indices]

# Remove or handle outliers based on your analysis
# For example, you can remove outliers from the dataset
df_no_outliers_zscore = df[~outlier_indices]
# Calculate IQR for numeric columns
```

```
Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1

# Identify outliers using IQR method
outliers_iqr = df[((df[numeric_cols] < (Q1 - 1.5 * IQR)) |
(df[numeric_cols] > (Q3 + 1.5 * IQR))).any(axis=1)]

# Remove or handle outliers based on your analysis
# For example, you can remove outliers from the dataset
df_no_outliers_iqr = df[~df.index.isin(outliers_iqr.index)]
import matplotlib.pyplot as plt

# Box plot for a specific numeric column
plt.figure(figsize=(8, 6))
plt.boxplot(df['Amount'])
plt.title('Box plot for Amount')
plt.show()

# Scatter plot for two numeric columns
plt.figure(figsize=(8, 6))
plt.scatter(df['Time'], df['Amount'])
plt.xlabel('Time')
plt.ylabel('Amount')
plt.title('Scatter plot for Time vs Amount')
plt.show()
```

3. Feature Scaling:

○ **Practical: Normalize or standardize features in the dataset to ensure fair comparisons and improve machine learning model performance.**

○ **Dataset: The "Wine Quality" dataset from the UCI Machine Learning Repository includes features related to wine properties that can benefit from feature scaling. Dataset link**

```
import pandas as pd

# Load the dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/wi
nequality-red.csv"
df = pd.read_csv(url, sep=";")
```

```python
# Display basic information about the dataset
print(df.info())
from sklearn.preprocessing import MinMaxScaler

# Select features to be normalized
features_to_normalize = df.columns[:-1]  # Exclude the target variable

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Normalize the selected features
df_normalized = df.copy()
df_normalized[features_to_normalize] =
scaler.fit_transform(df[features_to_normalize])

# Display the normalized dataset
print(df_normalized.head())
from sklearn.preprocessing import StandardScaler

# Initialize StandardScaler
scaler = StandardScaler()

# Standardize the selected features
df_standardized = df.copy()
df_standardized[features_to_normalize] =
scaler.fit_transform(df[features_to_normalize])

# Display the standardized dataset
print(df_standardized.head())
```

4. Data Visualization:

○ **Practical: Explore the dataset visually using plots such as histograms, scatter plots, and box plots to understand data distributions and relationships.**

○ **Dataset: The "Iris" dataset is a classic dataset often used for data visualization tasks, showcasing features of different iris flower species. Dataset link**

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Iris dataset
```

```
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'species']
df = pd.read_csv(url, names=columns)

# Display basic information about the dataset
print(df.info())
# Plot histograms for each numerical feature
plt.figure(figsize=(10, 6))
for i, feature in enumerate(df.columns[:-1]):
    plt.subplot(2, 2, i + 1)
    sns.histplot(df[feature], kde=True)
    plt.title(f'Histogram of {feature}')
plt.tight_layout()
plt.show()
# Scatter plot of sepal length vs sepal width colored by species
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x='sepal_length', y='sepal_width', hue='species')
plt.title('Scatter Plot of Sepal Length vs Sepal Width')
plt.show()
# Box plot of petal length for each species
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='species', y='petal_length')
plt.title('Box Plot of Petal Length by Species')
plt.show()
```

Association Rule Mining - Apriori algorithm

1. Frequent Itemset Mining:

○ Experiment: Use the Apriori algorithm to mine frequent itemsets from transactional data.

○ Dataset: The "Online Retail" dataset from the UCI Machine Learning Repository contains transactional data from an online retail store, suitable for frequent itemset mining. Dataset link

```
pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the Online Retail dataset
```

```python
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20
Retail.xlsx"
df = pd.read_excel(url)

# Explore the dataset
print(df.head())
print(df.info())

# Preprocessing: Remove spaces in the description column
df['Description'] = df['Description'].str.strip()

# Perform one-hot encoding for market basket analysis
basket = df.groupby(['InvoiceNo',
'Description'])['Quantity'].sum().unstack().reset_index().fillna(0)
basket_sets = basket.drop('InvoiceNo', axis=1)

# Convert quantities to binary values (0 or 1)
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket_sets.applymap(encode_units)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket_sets, min_support=0.05,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1)
rules.sort_values(by='lift', ascending=False, inplace=True)

# Display frequent itemsets and association rules
print("Frequent Itemsets:")
print(frequent_itemsets)

print("\nAssociation Rules:")
```

```
print(rules)
```

2. Association Rule Generation:

○ **Experiment: Generate association rules with specified support and confidence thresholds from the mined frequent itemsets.**

○ **Dataset: The "Groceries" dataset from the UCI Machine LearningRepository includes transactional data from a grocery store, ideal for association rule generation tasks. Dataset link**

```
pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the Groceries dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/00352/Online%20
Retail.xlsx"
df = pd.read_excel(url)

# Explore the dataset
print(df.head())
print(df.info())

# Preprocessing: Remove spaces in the description column
df['Description'] = df['Description'].str.strip()

# Perform one-hot encoding for market basket analysis
basket = df.groupby(['InvoiceNo',
'Description'])['Quantity'].sum().unstack().reset_index().fillna(0)
basket_sets = basket.drop('InvoiceNo', axis=1)

# Convert quantities to binary values (0 or 1)
def encode_units(x):
    if x <= 0:
        return 0
    if x >= 1:
        return 1

basket_sets = basket_sets.applymap(encode_units)

# Apply Apriori algorithm to find frequent itemsets
```

```
frequent_itemsets = apriori(basket_sets, min_support=0.01,
use_colnames=True)

# Generate association rules with specified support and confidence
thresholds
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)

# Display generated association rules
print("Generated Association Rules:")
print(rules)
```

3. Rule Evaluation and Pruning:

○ **Experiment: Evaluate generated association rules based on metrics like lift, Support & confidence. Prune rules based on predefined criteria.**

○ **Dataset: The "Mushroom" dataset from the UCI Machine Learning Repository contains data about mushroom species, suitable for association rule evaluation and pruning. Dataset link**

```
import pandas as pd

# Load the Mushroom dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaric
us-lepiota.data"
columns = ["class", "cap-shape", "cap-surface", "cap-color", "bruises",
           "odor", "gill-attachment", "gill-spacing", "gill-size",
           "gill-color", "stalk-shape", "stalk-root",
"stalk-surface-above-ring",
           "stalk-surface-below-ring", "stalk-color-above-ring",
"stalk-color-below-ring",
           "veil-type", "veil-color", "ring-number", "ring-type",
"spore-print-color",
           "population", "habitat"]
df = pd.read_csv(url, names=columns)

# Display the first few rows and inspect the data
print(df.head())
print(df.info())
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
```

```
le = LabelEncoder()

# Apply label encoding to each column
for col in df.columns:
    df[col] = le.fit_transform(df[col])

# Display the encoded DataFrame
print(df.head())
from mlxtend.frequent_patterns import apriori, association_rules

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(df, min_support=0.2, use_colnames=True)

# Generate association rules with specified support and confidence
thresholds
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.5)

# Display generated association rules
print("Generated Association Rules:")
print(rules)
```

4. Rule Visualization and Interpretation:

○ **Experiment: Visualize the generated association rules using graphs or charts for better understanding and interpretation.**

○ **Dataset: The "Market Basket Optimisation" dataset from Kaggle consists of transactional data from a grocery store, providing opportunities for rule visualization and interpretation**

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules

# Load the Market Basket Optimisation dataset
df =
pd.read_csv('https://raw.githubusercontent.com/stedy/Machine-Learning-with
-R-datasets/master/groceries.csv', header=None)

# Explore the dataset
print(df.head())
print(df.info())

# Perform one-hot encoding for market basket analysis
```

```python
basket_sets = pd.get_dummies(df, dtype=bool)

# Apply Apriori algorithm to find frequent itemsets
frequent_itemsets = apriori(basket_sets, min_support=0.01,
use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)

# Display generated association rules
print("Generated Association Rules:")
print(rules)
import matplotlib.pyplot as plt

# Scatter plot for support vs confidence
plt.figure(figsize=(10, 6))
plt.scatter(rules['support'], rules['confidence'], alpha=0.5)
plt.xlabel('Support')
plt.ylabel('Confidence')
plt.title('Support vs Confidence')
plt.show()

# Scatter plot for support vs lift
plt.figure(figsize=(10, 6))
plt.scatter(rules['support'], rules['lift'], alpha=0.5)
plt.xlabel('Support')
plt.ylabel('Lift')
plt.title('Support vs Lift')
plt.show()

# Scatter plot for confidence vs lift
plt.figure(figsize=(10, 6))
plt.scatter(rules['confidence'], rules['lift'], alpha=0.5)
plt.xlabel('Confidence')
plt.ylabel('Lift')
plt.title('Confidence vs Lift')
plt.show()
```

Classification: Naive Bayes Algorithm

1. Spam Email Classification:
○ Practical: Train a Naive Bayes classifier to distinguish between spam and non-spam emails based on text features.
○ Dataset: The "Spambase" dataset from the UCI Machine Learning Repository contains email spam and non-spam data, ideal for spam classification tasks. Dataset link

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Load the Spambase dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spamba
se.data"
columns = [
    "word_freq_make", "word_freq_address", "word_freq_all",
"word_freq_3d", "word_freq_our",
    "word_freq_over", "word_freq_remove", "word_freq_internet",
"word_freq_order", "word_freq_mail",
    "word_freq_receive", "word_freq_will", "word_freq_people",
"word_freq_report", "word_freq_addresses",
    "word_freq_free", "word_freq_business", "word_freq_email",
"word_freq_you", "word_freq_credit",
    "word_freq_your", "word_freq_font", "word_freq_000",
"word_freq_money", "word_freq_hp",
    "word_freq_hpl", "word_freq_george", "word_freq_650", "word_freq_lab",
"word_freq_labs",
    "word_freq_telnet", "word_freq_857", "word_freq_data",
"word_freq_415", "word_freq_85",
    "word_freq_technology", "word_freq_1999", "word_freq_parts",
"word_freq_pm", "word_freq_direct",
    "word_freq_cs", "word_freq_meeting", "word_freq_original",
"word_freq_project", "word_freq_re",
    "word_freq_edu", "word_freq_table", "word_freq_conference",
"char_freq_;", "char_freq_(",
    "char_freq_[", "char_freq_!", "char_freq_$", "char_freq_#",
"capital_run_length_average",
```

```
    "capital_run_length_longest", "capital_run_length_total", "is_spam"
]
df = pd.read_csv(url, names=columns)

# Split the dataset into features (X) and target variable (y)
X = df.drop('is_spam', axis=1)
y = df['is_spam']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

2. Sentiment Analysis:

○ **Practical: Perform sentiment analysis using a Naive Bayes classifier to categorize text data into positive, negative, or neutral sentiments.**

○ **Dataset: The "Sentiment140" dataset from Kaggle consists of tweets labeled with sentiments (positive or negative), suitable for sentiment analysis tasks. Dataset link**

```
import pandas as pd

# Load the Sentiment140 dataset
df =
pd.read_csv('https://storage.googleapis.com/kaggle-datasets/133598/344967/
training.1600000.processed.noemoticon.csv?X-Goog-Algorithm=GOOG4-RSA-SHA25
6&X-Goog-Credential=gcp-kaggle-com%40kaggle-161607.iam.gserviceaccount.com
```

```
%2F20220425%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20220425T165347Z&
X-Goog-Expires=259199&X-Goog-SignedHeaders=host&X-Goog-Signature=2ba3edbc9
4a6d8397d148d47854beea649a1b31f014b2715374a7f501b500efdd717cf57be1907a39b8
c26905f75dbd8ba9db1b489fda8354c3f190eb05ebf35d0d6ba7f53a4991df7d6661a5270e
8b79085d8c69e9bcf4fbdac57e264580df979531c638ad104f31392b8f4ed1792d1b3d17db
9138b0c17e8fc4e7fb8a882fc13083189a84d3b022ee4fa7b9c6160376f2b1c08ff01596e9
6ecae2d4f9c5739186a23bbce0fc9209bc546586db75c966f9574e4eac4902fa71b5d17f74
e87b29065f407eb87503c37f07afdc743d7d31828c6c728f84f2b5e09851ff0b17ccf7cb98
6e5868ab3e9d7066079e231de05b70367b76523d6b')
# Assign meaningful column names
df.columns = ['target', 'id', 'date', 'flag', 'user', 'text']

# Explore the dataset
print(df.head())
print(df.info())

# Map sentiment target values to readable labels
sentiment_mapping = {0: 'Negative', 2: 'Neutral', 4: 'Positive'}
df['sentiment'] = df['target'].map(sentiment_mapping)

# Drop unnecessary columns
df.drop(['target', 'id', 'date', 'flag', 'user'], axis=1, inplace=True)

# Filter out neutral sentiment (if needed)
# df = df[df['sentiment'] != 'Neutral']

# Display the updated dataset
print(df.head())
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Prepare the text data and labels
X = df['text']
y = df['sentiment']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```
# Initialize the CountVectorizer to convert text into a numerical format
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vec, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test_vec)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

3. Document Classification:

○ Practical: Classify documents into categories (e.g., news articles into topics) using a Naive Bayes classifier.

○ Dataset: The "20 Newsgroups" dataset from scikit-learn provides a collection of news articles categorized into 20 different topics, suitable for document classification experiments. Dataset link

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split

# Load the "20 Newsgroups" dataset
newsgroups_data = fetch_20newsgroups(subset='all', remove=('headers',
'footers', 'quotes'))

# Prepare the text data and labels
X = newsgroups_data.data
y = newsgroups_data.target
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the CountVectorizer to convert text into a numerical format
vectorizer = CountVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Initialize and train the Naive Bayes classifier
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_vec, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test_vec)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=newsgroups_data.target_names))
```

4. Medical Diagnosis:

○ **Practical: Build a Naive Bayes model to assist in medical diagnosis by predicting the likelihood of a disease based on symptoms.**

○ **Dataset: The "Breast Cancer Wisconsin (Diagnostic)" dataset from the UCI Machine Learning Repository includes features derived from breast cancer cell images, useful for medical diagnosis experiments. Dataset link**

```
import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Load the Breast Cancer Wisconsin (Diagnostic) dataset
data = load_breast_cancer()
```

```
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Gaussian Naive Bayes classifier
nb_classifier = GaussianNB()

# Train the classifier on the training data
nb_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=data.target_names))
```

Classification: Decision Tree

1. Customer Churn Prediction:

○ Practical: Build a decision tree classifier to predict customer churn (whether customers will leave or stay) based on historical customer data, such as usage patterns, demographics, and customer interactions.

○ Dataset: The "Telco Customer Churn" dataset from Kaggle contains customer data from a telecommunications company, including features related to customer behavior and churn status. Dataset link

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the Telco Customer Churn dataset
```

```
url =
"https://raw.githubusercontent.com/stedy/Machine-Learning-with-R-datasets/
master/telco-churn.csv"
df = pd.read_csv(url)

# Explore the dataset
print(df.head())
print(df.info())

# Preprocess the data: handle categorical variables and missing values
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')  #
Convert TotalCharges to numeric
df.dropna(inplace=True)  # Drop rows with missing values
df.drop(['customerID'], axis=1, inplace=True)  # Drop customerID column

# Convert categorical variables to numerical using one-hot encoding
df = pd.get_dummies(df, drop_first=True)

# Separate features (X) and target variable (y)
X = df.drop('Churn_Yes', axis=1)  # Features
y = df['Churn_Yes']  # Target variable (Churn_Yes indicates churn)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = dt_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

2. Loan Approval Prediction:

○ **Practical: Develop a decision tree model to predict whether a loan application will be approved or rejected based on applicant information like credit score, income, loan amount, and other relevant factors.**

○ **Dataset: The "German Credit" dataset from the UCI Machine Learning Repository includes attributes related to loan applicants and their creditworthiness, making it suitable for loan approval prediction experiments. Dataset link**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load the German Credit dataset
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/german/
german.data"
# Define column names based on dataset description
columns = [
    "existing_account_status", "duration_month", "credit_history",
"purpose",
    "credit_amount", "savings_account", "employment_status",
"installment_rate",
    "personal_status_sex", "other_debtors", "residence_since", "property",
    "age", "other_installment_plans", "housing", "existing_credits",
    "job", "num_dependents", "telephone", "foreign_worker",
"credit_approval"
]
df = pd.read_csv(url, sep=' ', names=columns)

# Explore the dataset
print(df.head())
print(df.info())

# Preprocess the data: handle categorical variables
# Convert categorical variables to numerical using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)
```

```
# Separate features (X) and target variable (y)
X = df_encoded.drop('credit_approval_good', axis=1)  # Features
y = df_encoded['credit_approval_good']  # Target variable
(credit_approval_good indicates approval)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
dt_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = dt_classifier.predict(X_test)

# Evaluate the classifier's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

#Clustering: K means algorithm

1. Customer Segmentation:

○ Practical: Use K-means clustering to segment customers based on their purchasing behavior and demographics.

○ Dataset: The "Mall Customer Segmentation Data" from Kaggle contains information about customers such as age, income, and spending score, suitable for customer segmentation tasks.

Dataset link

```
import pandas as pd

# Load the Mall Customer Segmentation Data
df =
pd.read_csv('https://raw.githubusercontent.com/stedy/Machine-Learning-with
-R-datasets/master/mall_customers.csv')
```

```python
# Display the first few rows and info about the dataset
print(df.head())
print(df.info())
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Selecting relevant features for clustering (e.g., income and spending
score)
X = df[['Annual Income (k$)', 'Spending Score (1-100)']]

# Determine the optimal number of clusters using the Elbow Method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method to find the optimal number of clusters
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o', linestyle='--')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS (Within-cluster Sum of Squares)')
plt.title('Elbow Method for Optimal K')
plt.xticks(range(1, 11))
plt.grid(True)
plt.show()
# Initialize and fit K-means clustering with the chosen number of clusters
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
kmeans.fit(X)

# Assign clusters to each data point
df['Cluster'] = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(10, 6))
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'],
c=df['Cluster'], cmap='viridis')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
```

```
plt.title('Customer Segmentation using K-means Clustering')
plt.show()

# Print the cluster centers (centroid) for interpretation
print("Cluster Centers (Centroids):")
print(pd.DataFrame(kmeans.cluster_centers_, columns=['Annual Income (k$)',
'Spending Score (1-100)']))
```

2. Image Compression:

○ **Practical: Apply K-means clustering to compress images by reducing the number of colors while preserving image quality.**

○ **Dataset: You can use images from public repositories or create your own dataset of images for image compression experiments.**

```
pip install numpy matplotlib scikit-learn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from PIL import Image  # Pillow library for image processing

# Load the image (replace 'your_image.jpg' with the path to your image)
image_path = 'your_image.jpg'
original_image = Image.open(image_path)
original_width, original_height = original_image.size

# Display the original image
plt.figure(figsize=(6, 6))
plt.imshow(original_image)
plt.title('Original Image')
plt.axis('off')
plt.show()

# Convert the image to a NumPy array
image_array = np.array(original_image)
height, width, channels = image_array.shape

# Reshape the array to a 2D matrix of pixels (each row represents a pixel)
pixels = image_array.reshape((-1, channels))

# Number of colors/clusters for K-means
n_colors = 16  # Adjust this parameter for desired compression level
```

```python
# Fit K-means clustering on the pixels
kmeans = KMeans(n_clusters=n_colors, random_state=42)
kmeans.fit(pixels)

# Assign each pixel to its nearest cluster center and create compressed
image
compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
compressed_image = compressed_pixels.reshape((height, width,
channels)).astype(np.uint8)

# Create a PIL image from the compressed array
compressed_image = Image.fromarray(compressed_image)

# Display the compressed image
plt.figure(figsize=(6, 6))
plt.imshow(compressed_image)
plt.title(f'Compressed Image ({n_colors} colors)')
plt.axis('off')
plt.show()

# Calculate compression ratio
original_size = original_width * original_height * channels
compressed_size = len(kmeans.labels_) * (n_colors + 1)  # Each pixel needs
n_colors + 1 bits (index + cluster center)
compression_ratio = original_size / compressed_size
print(f"Compression Ratio: {compression_ratio:.2f}")
```

3. Anomaly Detection:

○ **Practical: Detect anomalies or outliers in data using K-means clustering to identify unusual patterns or data points.**

○ **Dataset: The "Credit Card Fraud Detection" dataset from Kaggle includes transactional data with potential anomalies related to fraudulent activities. Dataset link**

```python
pip install numpy pandas matplotlib scikit-learn seaborn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

```python
import seaborn as sns

# Load the Credit Card Fraud Detection dataset
df =
pd.read_csv('https://storage.googleapis.com/download.tensorflow.org/data/c
reditcard.csv')

# Explore the dataset
print(df.head())
print(df.info())

# Drop the 'Time' column as it's not necessary for clustering
df.drop('Time', axis=1, inplace=True)

# Standardize the features (mean=0, std=1) using StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(df.drop('Class', axis=1))  # 'Class' is the
target variable (0: non-fraud, 1: fraud)

# Reduce dimensionality using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Initialize K-means clustering for anomaly detection
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X)

# Assign each data point to its nearest cluster center
df['Cluster'] = kmeans.labels_

# Visualize the clusters in 2D (PCA-transformed space)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=df['Cluster'],
palette='viridis', alpha=0.7)
plt.title('Anomaly Detection using K-means Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(['Cluster 0', 'Cluster 1'], loc='best')
plt.show()
```

```
# Check the distribution of fraud and non-fraud transactions in each
cluster
fraud_distribution = df.groupby(['Cluster', 'Class']).size().unstack()
print(fraud_distribution)
```

4. Market Segmentation:

○ **Practical: Cluster market data (e.g., products, customers) using K-means clustering to identify distinct market segments.**

○ **Dataset: The "Online Retail" dataset from the UCI Machine Learning Repository contains transactional data from an online retail store, suitable for market segmentation analysis. Dataset link**

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# Load the Online Retail dataset
df =
pd.read_excel('http://archive.ics.uci.edu/ml/machine-learning-databases/00
352/Online%20Retail.xlsx')

# Explore the dataset
print(df.head())
print(df.info())

# Preprocess the data
df_cleaned = df.dropna()  # Drop rows with missing values
df_cleaned = df_cleaned[df_cleaned['Quantity'] > 0]  # Keep only positive
quantity values

# Create a new column for total sales (quantity * unit price)
df_cleaned['TotalSales'] = df_cleaned['Quantity'] *
df_cleaned['UnitPrice']

# Group data by CustomerID and calculate total sales per customer
customer_sales =
df_cleaned.groupby('CustomerID')['TotalSales'].sum().reset_index()
```

```python
# Select relevant features for clustering (total sales per customer)
X = customer_sales[['TotalSales']]

# Standardize the features (mean=0, std=1) using StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Initialize K-means clustering for market segmentation
kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_scaled)

# Assign each customer to its cluster
customer_sales['Cluster'] = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x='CustomerID', y='TotalSales', hue='Cluster',
data=customer_sales, palette='viridis')
plt.title('Market Segmentation using K-means Clustering')
plt.xlabel('CustomerID')
plt.ylabel('Total Sales')
plt.legend(title='Cluster', loc='best')
plt.show()

# Analyze cluster statistics
cluster_stats = customer_sales.groupby('Cluster')['TotalSales'].describe()
print(cluster_stats)
```