**Question Set**

1. **Reena is a high school teacher and she wants to investigate if there's a relationship between the number of hours students spend studying for a math test and their resulting scores. She decides to collect data from a random sample of 20 students. She record each student's study hours (independent variable) and their corresponding math test scores (dependent variable).**

**Predict a student's math test score based on the number of hours they spend studying?**

|  |  |
| --- | --- |
| **hours\_studied** | **exam\_scores** |
| **2** | **60** |
| **3** | **70** |
| **4** | **80** |
| **5** | **85** |
| **6** | **75** |
| **7** | **90** |
| **8** | **95** |
| **9** | **85** |
| **10** | **80** |
| **11** | **75** |

# Import required libraries

import pandas as pd

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv(r"C:\Users\Deepu\OneDrive\Deepu\studyhours.csv")

data.columns = ['HoursStudied', 'ExamScores']

# Define independent (X) and dependent (y) variables

X = data[['HoursStudied']]

y = data['ExamScores']

# Create and train the Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Get user input for hours studied

hours = float(input("Enter number of study hours: "))

# Predict exam score

predicted\_score = model.predict(pd.DataFrame({'HoursStudied': [hours]}))

print(f"\nPredicted Exam Score for {hours} hours of study = {predicted\_score[0]:.2f}")

# Visualize the data and regression line

plt.scatter(X, y, color='blue', label='Actual Data')

plt.plot(X, model.predict(X), color='red', label='Regression Line')

plt.scatter(hours, predicted\_score, color='green', s=100, label='Predicted Point')

plt.xlabel('Hours Studied')

plt.ylabel('Exam Score')

plt.title('Study Hours vs Exam Score')

plt.legend()

plt.show()

1. **Consider you're a manager at an ice cream shop located in a tourist destination. You're interested in predicting the daily ice cream sales based on the temperature because you believe that warmer weather leads to higher ice cream sales. Over the past month, you've been keeping track of the daily temperature (in degrees Celsius) and the corresponding ice cream sales (in dollars) at your shop.**

**You have collected the following data:**

|  |  |
| --- | --- |
| **Temp**  **(degree Celsius)** | **Ice Cream Sale**  **(In litres)** |
| **20** | **17** |
| **25** | **23** |
| **30** | **28** |
| **33** | **31** |
| **35** | **38** |
| **38** | **41** |
| **40** | **53** |
| **47** | **62** |
| **52** | **90** |

**Using simple linear regression, analyze the relationship between temperature and ice cream sales at your shop.**

#### Import required libraries

import pandas as pd

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Load the dataset (FIXED file path issue using raw string)

data = pd.read\_csv(r"C:\Users\Deepu\OneDrive\Deepu\icecream.csv")

data.columns = ['Temperature', 'IceCreamSales']

# Display the dataset (optional: helps in viva)

print("Dataset Preview:")

print(data.head())

# Define independent (X) and dependent (y) variables

X = data[['Temperature']] # 2D array

y = data['IceCreamSales'] # 1D array

# Create and train the Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Get user input for temperature

temp\_value = float(input("\nEnter today's temperature (°C): "))

# Predict ice cream sales

predicted\_sales = model.predict(pd.DataFrame({'Temperature': [temp\_value]}))

print(f"\nPredicted Ice Cream Sales for {temp\_value}°C = {predicted\_sales[0]:.2f} litres")

# Visualize the data and regression line

plt.figure(figsize=(8, 5))

plt.scatter(X, y, label='Actual Data')

plt.plot(X, model.predict(X), label='Regression Line')

plt.scatter(temp\_value, predicted\_sales, s=100, label='Predicted Point')

plt.xlabel('Temperature (°C)')

plt.ylabel('Ice Cream Sales (litres)')

plt.title('Temperature vs Ice Cream Sales')

plt.legend()

plt.show()

1. **Consider you're an owner of a tea shop located in a tourist destination. You're interested in predicting the daily tea sales based on the temperature because you believe that cold weather leads to higher tea sales. Over the past month, you've been keeping track of the daily temperature (in degrees Celsius) and the corresponding tea sales (in dollars) at your shop.**

**You have collected the following data:**

|  |  |
| --- | --- |
| **Temp**  **(degree Celsius)** | **Tea Sale**  **(In litres)** |
| **20** | **45** |
| **25** | **37** |
| **30** | **31** |
| **33** | **28** |
| **35** | **26** |
| **38** | **23** |
| **40** | **17** |
| **47** | **9** |
| **52** | **7** |

**Using simple linear regression, analyze the relationship between temperature and tea sales at your shop.**

# Import required libraries

import pandas as pd

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

# Load the dataset using the uploaded tea.csv file

data = pd.read\_csv(r"C:\Users\Deepu\OneDrive\Deepu\tea.csv") # since you already uploaded it

# Rename columns to avoid KeyError (matching your expected style)

data.columns = ['Temperature', 'TeaSales']

# Display the dataset (optional: viva answer)

print("Dataset Preview:")

print(data.head())

# Define independent (X) and dependent (y) variables

X = data[['Temperature']] # 2D array

y = data['TeaSales'] # 1D array

# Create and train the Linear Regression model

model = LinearRegression()

model.fit(X, y)

# Get user input for temperature

temp\_value = float(input("\nEnter today's temperature (°C): "))

# Predict tea sales

predicted\_sales = model.predict(pd.DataFrame({'Temperature': [temp\_value]}))

print(f"\nPredicted Tea Sales for {temp\_value}°C = {predicted\_sales[0]:.2f} dollars")

# Visualize the data and regression line

plt.figure(figsize=(8, 5))

plt.scatter(X, y, label='Actual Data')

plt.plot(X, model.predict(X), label='Regression Line')

plt.scatter(temp\_value, predicted\_sales, s=100, label='Predicted Point')

plt.xlabel('Temperature (°C)')

plt.ylabel('Tea Sales (dollars)')

plt.title('Temperature vs Tea Sales')

plt.legend()

plt.show()

1. **Perform multiple linear regressions for below mentioned data set.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CAR** | **MODEL** | **VOLUME** | **WEIGHT** | **CO2** |
| **Toyato** | **Aygu** | **1000** | **790** | **99** |
| **Mitsubish** | **Space Star** | **1200** | **1160** | **95** |
| **Skoda** | **Citigo** | **1000** | **929** | **95** |
| **Flat** | **Gold** | **900** | **865** | **90** |
| **Mini** | **cooper** | **1500** | **1140** | **105** |

# Install required packages (only runs if not already installed)

!pip install pandas scikit-learn matplotlib --quiet

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Step 1: Load dataset from CSV file

df = pd.read\_csv("car.csv")

# Step 2: Display dataset

print("Dataset Loaded Successfully:\n")

print(df.head())

# Step 3: Define features (independent variables) and target (dependent variable)

X = df[['VOLUME', 'WEIGHT']]

y = df['CO2']

# Step 4: Create and train the model

model = LinearRegression()

model.fit(X, y)

print("\nModel trained successfully!")

print(f"Intercept: {model.intercept\_}")

print(f"Coefficients: {model.coef\_}")

# Step 5: Take user input for prediction

print("\nEnter new car details to predict CO2 emission:")

vol = float(input("Enter Engine Volume (e.g. 1200): "))

wt = float(input("Enter Car Weight (e.g. 1000): "))

# Step 6: Make prediction

predicted\_co2 = model.predict([[vol, wt]])

print(f"\nPredicted CO2 emission for car with Volume={vol} & Weight={wt} is: {predicted\_co2[0]:.2f}")

# Step 7: Visualizations

# ----------------------------------

# (1) Actual Weight vs CO2

plt.figure(figsize=(6,4))

plt.scatter(df['WEIGHT'], df['CO2'], color='blue', label='Actual CO2')

plt.xlabel('Car Weight')

plt.ylabel('CO2 Emission')

plt.title('Weight vs CO2 Emission (Actual)')

plt.legend()

plt.grid(True)

plt.show()

# (2) Predicted CO2 vs Weight (using model)

predicted = model.predict(X)

plt.figure(figsize=(6,4))

plt.scatter(df['WEIGHT'], y, color='blue', label='Actual CO2')

plt.plot(df['WEIGHT'], predicted, color='red', label='Predicted CO2')

plt.xlabel('Car Weight')

plt.ylabel('CO2 Emission')

plt.title('CO2 Prediction by Weight (Model Output)')

plt.legend()

plt.grid(True)

plt.show()

1. **Manu is a data scientist working for a financial institution, and her task is to develop a machine learning model for detecting credit card fraud. She has access to a dataset containing various features such as transaction amount, time of transaction, and other anonymized features, along with a label indicating whether each transaction is fraudulent or not. She decide to use a linear support vector machine (SVM) algorithm for this task due to its effectiveness in handling high-dimensional data and separating classes in binary classification problems.**

# Import required libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

# Step 1: Load dataset (make sure creditcard.csv is in same folder)

df = pd.read\_csv("creditcard.csv")

print("Dataset Loaded Successfully!\n")

print(df.head())

# Step 2: Separate features and target

X = df.drop('Class', axis=1)

y = df['Class']

# Step 3: Split 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)

# Step 4: Feature scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 5: Create and train Linear SVM model

model = SVC(kernel='linear')

model.fit(X\_train, y\_train)

# Step 6: Predict results

y\_pred = model.predict(X\_test)

# Step 7: Evaluate model

print("\nModel Evaluation Results:\n")

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nAccuracy:", accuracy\_score(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Step 8: Visualization (sample 500 points for clear plotting)

# Use first two features (V1 and V2) for visualization

sample\_df = df.sample(500, random\_state=42)

plt.figure(figsize=(8,6))

plt.scatter(sample\_df[sample\_df['Class'] == 0]['V1'], sample\_df[sample\_df['Class'] == 0]['V2'],

color='blue', label='Not Fraud (0)', alpha=0.6)

plt.scatter(sample\_df[sample\_df['Class'] == 1]['V1'], sample\_df[sample\_df['Class'] == 1]['V2'],

color='red', label='Fraud (1)', alpha=0.8)

plt.title(' Credit Card Fraud Detection (Linear SVM)')

plt.xlabel('Feature V1')

plt.ylabel('Feature V2')

plt.legend()

plt.grid(True)

plt.show()

1. **Consider the below mentioned sample dataset and apply linear binary classifier to decide whether the customer will purchase particular product or not.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User ID** | **Gender** | **Age** | **Estimated Salary** | **Purchased** |
| **15624510** | **Male** | **19** | **19000** | **0** |
| **15810944** | **Male** | **35** | **20000** | **0** |
| **15728773** | **Male** | **27** | **58000** | **0** |
| **15598044** | **Female** | **27** | **84000** | **0** |
| **15694829** | **Female** | **32** | **150000** | **1** |
| **15600575** | **Male** | **25** | **33000** | **0** |
| **15704987** | **Male** | **32** | **18000** | **0** |
| **15628972** | **Male** | **18** | **82000** | **0** |
| **15697686** | **Male** | **29** | **80000** | **0** |
| **15733883** | **Male** | **47** | **25000** | **1** |
| **15617482** | **Male** | **45** | **26000** | **1** |
| **15736760** | **Female** | **47** | **49000** | **1** |
| **15714658** | **Male** | **48** | **41000** | **1** |

# Import libraries

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

# Step 1: Load dataset

df = pd.read\_csv("customer\_product\_purchase.csv") # <-- your CSV file

print("Dataset Loaded Successfully!\n")

print(df.head())

# Step 2: Encode 'Gender' column (Male=1, Female=0)

le = LabelEncoder()

df['Gender'] = le.fit\_transform(df['Gender'])

# Step 3: Define features (X) and target (y)

X = df[['Gender', 'Age', 'Estimated Salary']]

y = df['Purchased']

# Step 4: Split the dataset into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=42

)

# Step 5: Feature scaling

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Step 6: Train Linear SVM model

model = SVC(kernel='linear')

model.fit(X\_train, y\_train)

# Step 7: Predictions

y\_pred = model.predict(X\_test)

# Step 8: Evaluate model

print("\n Model Evaluation Results:\n")

print("Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nAccuracy:", accuracy\_score(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

# Step 9: Visualization - Age vs Salary colored by Purchased

plt.figure(figsize=(8,6))

plt.scatter(df[df['Purchased']==0]['Age'], df[df['Purchased']==0]['Estimated Salary'],

color='red', label='Not Purchased (0)')

plt.scatter(df[df['Purchased']==1]['Age'], df[df['Purchased']==1]['Estimated Salary'],

color='green', label='Purchased (1)')

plt.title('Customer Purchase Prediction (Linear SVM)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.grid(True)

plt.show()

1. **Raj is tasked with building a machine learning model to determine whether an individual is eligible to receive a credit card. Using a Naive Bayes classifier and the provided dataset containing features such as age, income, employment status, credit score, how he would approach this task.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rid** | **Age** | **Income** | **Student** | **Credit** | **Buy** |
| **1** | **youth** | **high** | **no** | **fair** | **no** |
| **2** | **youth** | **high** | **no** | **excellent** | **no** |
| **3** | **middle\_aged** | **high** | **no** | **fair** | **yes** |
| **4** | **senior** | **medium** | **no** | **fair** | **yes** |
| **5** | **senior** | **low** | **yes** | **fair** | **yes** |
| **6** | **senior** | **low** | **yes** | **excellent** | **no** |
| **7** | **middle\_aged** | **low** | **yes** | **excellent** | **yes** |
| **8** | **youth** | **medium** | **no** | **fair** | **no** |
| **9** | **youth** | **low** | **yes** | **fair** | **yes** |
| **10** | **senior** | **medium** | **yes** | **fair** | **yes** |
| **11** | **youth** | **medium** | **yes** | **excellent** | **yes** |
| **12** | **middle\_aged** | **medium** | **no** | **excellent** | **yes** |
| **13** | **middle\_aged** | **high** | **yes** | **fair** | **yes** |
| **14** | **senior** | **medium** | **no** | **excellent** | **no** |

# Step 1: Import libraries

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Step 2: Load dataset

data = pd.read\_csv("person\_naive.csv")

print("Dataset Preview:")

print(data.head())

# Step 3: Encode categorical columns

label\_encoders = {}

for col in ['Age', 'Income', 'Student', 'Credit', 'Buy']:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le # store encoders for later use

print("\n Encoded Data:")

print(data.head())

# Step 4: Split features and target

X = data[['Age', 'Income', 'Student', 'Credit']]

y = data['Buy']

# Step 5: Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 6: Train Naive Bayes model

model = GaussianNB()

model.fit(X\_train, y\_train)

# Step 7: Evaluate

y\_pred = model.predict(X\_test)

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

# Step 8: Take user input

print("\nEnter details to check credit card eligibility ")

age = input("Enter Age (youth / middle\_aged / senior): ").strip().lower()

income = input("Enter Income (high / medium / low): ").strip().lower()

student = input("Is the person a student? (yes / no): ").strip().lower()

credit = input("Credit rating (fair / excellent): ").strip().lower()

# Step 9: Encode user input using stored encoders

user\_data = pd.DataFrame({

'Age': [age],

'Income': [income],

'Student': [student],

'Credit': [credit]

})

for col in user\_data.columns:

if col in label\_encoders:

le = label\_encoders[col]

if user\_data[col][0] in le.classes\_:

user\_data[col] = le.transform(user\_data[col])

else:

print(f" Warning: '{user\_data[col][0]}' not in training data for '{col}'. Defaulting to first known value.")

user\_data[col] = [0]

# Step 10: Predict

prediction = model.predict(user\_data)[0]

result = label\_encoders['Buy'].inverse\_transform([prediction])[0]

print("\nPrediction Result:", "Eligible for Credit Card" if result == "yes" else "Not Eligible for Credit Card")

1. **Apply naïve bayes algorithm to decide whether to play tennis on a given day (if Outlook=Rainy, Temperature=Mild, Humidity=High, Windy=True) based on weather conditions. You have a dataset of past instances where you recorded whether tennis was played or not, along with corresponding weather conditions.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Outlook** | **Temperature** | **Humidity** | **Windy** | **Play Tennis** |
| **Sunny** | **Hot** | **High** | **False** | **No** |
| **Sunny** | **Hot** | **High** | **True** | **No** |
| **Overcast** | **Hot** | **High** | **False** | **Yes** |
| **Rainy** | **Mild** | **High** | **False** | **Yes** |
| **Rainy** | **Cool** | **Normal** | **False** | **Yes** |
| **Rainy** | **Cool** | **Normal** | **True** | **No** |
| **Overcast** | **Cool** | **Normal** | **True** | **Yes** |
| **Sunny** | **Mild** | **High** | **False** | **No** |
| **Sunny** | **Cool** | **Normal** | **False** | **Yes** |
| **Rainy** | **Mild** | **Normal** | **False** | **Yes** |
| **Sunny** | **Mild** | **Normal** | **True** | **Yes** |
| **Overcast** | **Mild** | **High** | **True** | **Yes** |
| **Overcast** | **Hot** | **Normal** | **False** | **Yes** |

**([imports..]COMMON FOR 8,9,10 & 11)**

!pip install pandas scikit-learn numpy mlxtend

import pandas as pd

import numpy as np

from sklearn.preprocessing import OrdinalEncoder, LabelEncoder

from sklearn.naive\_bayes import CategoricalNB

from sklearn.model\_selection import cross\_val\_score, train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score, classification\_report

import matplotlib.pyplot as plt

from mlxtend.frequent\_patterns import apriori, association\_rules

from mlxtend.preprocessing import TransactionEncoder

# --- load your CSV (put tennis.csv in same folder as notebook) ---

df = pd.read\_csv("tennis.csv") # change path if needed

# --- prepare features & label ---

features = ["Outlook","Temperature","Humidity","Windy"]

X\_raw = df[features].copy()

X\_raw["Windy"] = X\_raw["Windy"].astype(str) # treat windy as categorical string

y\_raw = df["Play Tennis"]

# encode categories to integers

enc = OrdinalEncoder(dtype=int)

X = enc.fit\_transform(X\_raw)

lab = LabelEncoder()

y = lab.fit\_transform(y\_raw)

# train Categorical Naive Bayes

clf = CategoricalNB(alpha=1.0)

clf.fit(X, y)

# prepare query and predict

query = pd.DataFrame([{"Outlook":"Rainy","Temperature":"Mild","Humidity":"High","Windy":"True"}])

Xq = enc.transform(query[features])

pred = lab.inverse\_transform(clf.predict(Xq))[0]

probs = clf.predict\_proba(Xq)[0]

# print results (simple)

print("Prediction for (Outlook=Rainy, Temperature=Mild, Humidity=High, Windy=True) -->", pred)

print("Probabilities:")

for label, p in zip(lab.classes\_, probs):

print(f" {label}: {p:.4f}")

# also show feature encodings (optional, helps lab report)

print("\nFeature encodings (category -> int):")

for i, col in enumerate(features):

print(f" {col}:", {cat: idx for idx, cat in enumerate(enc.categories\_[i])})

1. **Suppose you're tasked with building a machine learning model to predict whether a patient is likely to have diabetes. Using a decision tree classifier and the provided dataset containing features such as glucose level, BMI, blood pressure, age, and family history, how would you approach this task? Consider the below mentioned dataset.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Glucose** | **Blood Pressure** | **Skin Thickness** | **Insulin** | **BMI** | **Age** | **Outcome** |
| **148** | **72** | **35** | **0** | **33.6** | **50** | **1** |
| **85** | **66** | **29** | **0** | **26.6** | **31** | **0** |
| **183** | **64** | **0** | **0** | **23.3** | **32** | **1** |
| **89** | **66** | **23** | **94** | **28.1** | **21** | **0** |
| **137** | **40** | **35** | **168** | **43.1** | **33** | **1** |
| **116** | **74** | **0** | **0** | **25.6** | **30** | **0** |
| **78** | **50** | **32** | **88** | **31** | **26** | **1** |
| **115** | **0** | **0** | **0** | **35.3** | **29** | **0** |
| **197** | **70** | **45** | **543** | **30.5** | **53** | **1** |
| **125** | **96** | **0** | **0** | **0** | **54** | **1** |
| **110** | **92** | **0** | **0** | **37.6** | **30** | **0** |
| **168** | **74** | **0** | **0** | **38** | **34** | **1** |

# ====== STEP 1: Load dataset (make sure file is in same folder) ======

df = pd.read\_csv("diabetes.csv") # change filename if needed

print("=== Dataset Loaded ===")

print(df.head(), "\n")

# ====== STEP 2: Split features & label ======

X = df.drop("Outcome", axis=1)

y = df["Outcome"]

# Split into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# ====== STEP 3: Train Decision Tree Classifier ======

clf = DecisionTreeClassifier(criterion="entropy", random\_state=42)

clf.fit(X\_train, y\_train)

# ====== STEP 4: Evaluate model ======

y\_pred = clf.predict(X\_test)

acc = accuracy\_score(y\_test, y\_pred)

print(f"Model Accuracy: {acc\*100:.2f}%\n")

print("Classification Report:\n", classification\_report(y\_test, y\_pred))

# ====== STEP 5: Predict for a new patient (example) ======

# You can change these values to test manually

new\_patient = pd.DataFrame([{

"Glucose": 120,

"Blood Pressure": 70,

"Skin Thickness": 25,

"Insulin": 80,

"BMI": 30.0,

"Age": 35

}])

prediction = clf.predict(new\_patient)[0]

print("=== Prediction for New Patient ===")

print(new\_patient.to\_string(index=False))

print("\nPredicted Outcome:", "Diabetic" if prediction == 1 else "Non-Diabetic")

# ====== STEP 6: Visualize the Decision Tree ======

plt.figure(figsize=(12,6))

plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=["Non-Diabetic", "Diabetic"], rounded=True)

plt.title("Decision Tree for Diabetes Prediction")

plt.show()

1. **Consider the below mentioned dataset and apply decision tree algorithm to decide**

**whether the particular day is preferable for playing tennis.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Day** | **Outlook** | **Temprature** | **Humidity** | **Wind** | **Play\_Tennis** |
| **D1** | **Sunny** | **Hot** | **High** | **Weak** | **No** |
| **D2** | **Sunny** | **Hot** | **High** | **Strong** | **No** |
| **D3** | **Overcast** | **Hot** | **High** | **Weak** | **Yes** |
| **D4** | **Rain** | **Mild** | **High** | **Weak** | **Yes** |
| **D5** | **Rain** | **Cool** | **Normal** | **Weak** | **Yes** |
| **D6** | **Rain** | **Cool** | **Normal** | **Strong** | **No** |
| **D7** | **Overcast** | **Cool** | **Normal** | **Strong** | **Yes** |
| **D8** | **Sunny** | **Mild** | **High** | **Weak** | **No** |
| **D9** | **Sunny** | **Cool** | **Normal** | **Weak** | **Yes** |
| **D10** | **Rain** | **Mild** | **Normal** | **Weak** | **Yes** |

# 1) Load dataset (keep file in same folder)

df = pd.read\_csv("ml10.csv") # ensure headers match: Day,Outlook,Temprature,Humidity,Wind,Play\_Tennis

print("Dataset:\n", df, "\n")

# 2) Select features (exclude Day) and target

feature\_cols = ['Outlook', 'Temprature', 'Humidity', 'Wind']

target\_col = 'Play\_Tennis'

# 3) Create a LabelEncoder for each categorical column and fit on training data

encoders = {}

df\_enc = df.copy()

for col in feature\_cols:

le = LabelEncoder()

df\_enc[col] = le.fit\_transform(df[col].astype(str))

encoders[col] = le

# encode target

le\_target = LabelEncoder()

df\_enc[target\_col] = le\_target.fit\_transform(df[target\_col].astype(str))

print("Encoded dataset:\n", df\_enc, "\n")

# 4) Train Decision Tree

X = df\_enc[feature\_cols]

y = df\_enc[target\_col]

clf = DecisionTreeClassifier(criterion='entropy', random\_state=1)

clf.fit(X, y)

print("Model trained.\n")

# 5) Prepare sample (use plain strings) and encode using stored encoders (DO NOT FIT)

sample\_raw = {"Outlook": "Rain", "Temprature": "Mild", "Humidity": "High", "Wind": "Weak"}

sample = pd.DataFrame([sample\_raw])

for col in feature\_cols:

sample[col] = encoders[col].transform(sample[col].astype(str)) # transform only

sample\_X = sample[feature\_cols]

print("Encoded sample:", sample\_X.to\_dict(orient='records')[0])

# 6) Predict and show readable label

pred\_idx = clf.predict(sample\_X)[0]

pred\_label = le\_target.inverse\_transform([pred\_idx])[0]

print(f"Prediction for {sample\_raw} --> Play\_Tennis = {pred\_label}\n")

# 7) (Optional) Visualize the tree

plt.figure(figsize=(8,5))

plot\_tree(clf, filled=True, feature\_names=feature\_cols, class\_names=le\_target.classes\_, rounded=True)

plt.show()

question 10

1. **Consider you're analyzing customer transaction data for a grocery store chain. Using the Apriori algorithm and the provided dataset of customer transactions, how would you**

**identify frequent item sets? Sample dataset is given below.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **T1** | **meatballs** | **milk** | **honey** | **french fries** | **protein bar** | |  |  |
| **T2** | **red wine** | **shrimp** | **pasta** | **pepper** | **eggs** | **chocolate** | **shampoo** |  |
| **T3** | **rice** | **sparkling water** | |  |  |  |  |  |
| **T4** | **spaghetti** | **mineral water** | **ham** | **body spray** | **pancakes** | **green tea** |  |  |
| **T5** | **burgers** | **grated cheese** | **shrimp** | **pasta** | **avocado** | **honey** | **white wine** | **toothpaste** |
| **T6** | **eggs** |  |  |  |  |  |  |  |
| **T7** | **parmesan cheese** | **spaghetti** | **soup** | **avocado** | **milk** | **fresh bread** | |  |

# ===== STEP 1: Load dataset =====

# Create CSV (transactions.csv) before running this

# Example:

# TID,Items

# T1,meatballs,milk,honey,french fries,protein bar

# T2,red wine,shrimp,pasta,pepper,eggs,chocolate,shampoo

# ...

df = pd.read\_csv("transactions.csv")

print("=== Dataset Loaded ===")

print(df, "\n")

# ===== STEP 2: Preprocess data =====

# Split items into lists

transactions = df.apply(lambda row: [x for x in row[1:] if str(x) != 'nan'], axis=1).tolist()

# Create one-hot encoded DataFrame

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()

te\_ary = te.fit(transactions).transform(transactions)

df\_encoded = pd.DataFrame(te\_ary, columns=te.columns\_)

print("=== One-Hot Encoded Transaction Data ===")

print(df\_encoded.head(), "\n")

# ===== STEP 3: Apply Apriori algorithm =====

frequent\_itemsets = apriori(df\_encoded, min\_support=0.3, use\_colnames=True)

print("=== Frequent Itemsets (min\_support=0.3) ===")

print(frequent\_itemsets, "\n")

# ===== STEP 4: Generate Association Rules =====

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.6)

print("=== Association Rules (confidence >= 0.6) ===")

print(rules[["antecedents", "consequents", "support", "confidence", "lift"]])

1. **Consider the below mentioned data types and apply partition clustering algorithm to group**

**the similar instances.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CustomerID** | **Genre** | **Age** | **Annual Income (k$)** | **Spending Score**  **(1-100)** |
| **1** | **Male** | **19** | **15** | **39** |
| **2** | **Male** | **21** | **15** | **81** |
| **3** | **Female** | **20** | **16** | **6** |
| **4** | **Female** | **23** | **16** | **77** |
| **5** | **Female** | **31** | **17** | **40** |
| **6** | **Female** | **22** | **17** | **76** |
| **7** | **Female** | **35** | **18** | **6** |
| **8** | **Female** | **23** | **18** | **94** |
| **9** | **Male** | **64** | **19** | **3** |
| **10** | **Female** | **30** | **19** | **72** |
| **11** | **Male** | **67** | **19** | **14** |
| **12** | **Female** | **35** | **19** | **99** |
| **13** | **Female** | **58** | **20** | **15** |
| **14** | **Female** | **24** | **20** | **77** |
| **15** | **Male** | **37** | **20** | **13** |

# Disable all warnings

import warnings

warnings.filterwarnings('ignore')

# Fix KMeans MKL memory warning

import os

os.environ["OMP\_NUM\_THREADS"] = "1"

# Import required libraries

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv('customers\_kmeans.csv')

# Encode 'Genre'

le = LabelEncoder()

data['Genre'] = le.fit\_transform(data['Genre'])

# Select features

X = data[['Genre', 'Age', 'Annual Income (k$)', 'Spending Score']]

# ----- ELBOW METHOD -----

inertia = []

max\_k = min(10, len(X))

K = range(1, max\_k + 1)

for k in K:

kmeans = KMeans(n\_clusters=k, random\_state=0)

kmeans.fit(X)

inertia.append(kmeans.inertia\_)

# Plot Elbow Method (No gridlines)

plt.figure(figsize=(6, 4))

plt.plot(K, inertia, marker='o', color='blue')

plt.title('Elbow Method to Find Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.grid(False)

plt.show()

# ----- K-MEANS CLUSTERING -----

k = 3 # Choose from elbow

kmeans = KMeans(n\_clusters=k, random\_state=0)

data['Cluster'] = kmeans.fit\_predict(X)

# Plot clusters (Age vs Spending Score)

plt.figure(figsize=(6, 4))

plt.scatter(data['Age'], data['Spending Score'],

c=data['Cluster'], cmap='viridis', s=100)

plt.xlabel('Age')

plt.ylabel('Spending Score')

plt.title('K-Means Clustering Result')

plt.grid(False)

plt.show()

1. **Consider the below mentioned data types and apply K-Means clustering algorithm to group the similar instances.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **status\_id** | **status\_type** | **num\_**  **commnts** | **num\_**  **shares** | **num\_**  **likes** | **num\_**  **wows** | **num\_**  **hahas** | **num\_sads** | **num\_angrys** |
| **7474** | **video** | **512** | **262** | **432** | **3** | **1** | **1** | **0** |
| **7757** | **photo** | **0** | **0** | **150** | **0** | **0** | **0** | **0** |
| **7397** | **video** | **236** | **57** | **204** | **1** | **1** | **0** | **0** |
| **9452** | **photo** | **0** | **0** | **111** | **0** | **0** | **0** | **0** |
| **3739** | **photo** | **0** | **0** | **204** | **0** | **0** | **0** | **0** |
| **8773** | **photo** | **6** | **0** | **211** | **1** | **0** | **0** | **0** |

# STEP 1: Import required libraries

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# STEP 2: Load the dataset

data = pd.read\_csv('status\_kmeans.csv')

# STEP 3: Convert 'status\_type' to numeric using LabelEncoder

le = LabelEncoder()

data['status\_type'] = le.fit\_transform(data['status\_type'])

# STEP 4: Select numeric columns for clustering

X = data[['status\_type', 'num\_commnts', 'num\_shares', 'num\_likes', 'num\_wows', 'num\_hahas', 'num\_sads', 'num\_angrys']]

# STEP 5: Elbow Method (limit k ≤ number of samples)

inertia = []

max\_k = min(10, len(X))

K = range(1, max\_k + 1)

for k in K:

kmeans = KMeans(n\_clusters=k, random\_state=0)

kmeans.fit(X)

inertia.append(kmeans.inertia\_)

# Elbow Method graph (no grid)

plt.figure(figsize=(6, 4))

plt.plot(K, inertia, marker='o', color='blue')

plt.title('Elbow Method to Find Optimal k')

plt.xlabel('Number of Clusters (k)')

plt.ylabel('Inertia')

plt.show()

# STEP 6: Apply K-Means with k=3 or less

k = min(3, len(X))

kmeans = KMeans(n\_clusters=k, random\_state=0)

data['Cluster'] = kmeans.fit\_predict(X)

# STEP 7: Cluster visualization (no grid)

plt.figure(figsize=(6, 4))

plt.scatter(data['num\_likes'], data['num\_commnts'], c=data['Cluster'], cmap='viridis', s=100)

plt.xlabel('Number of Likes')

plt.ylabel('Number of Comments')

plt.title('K-Means Clustering Result')

plt.show()

1. **Consider the below mentioned dataset and apply CURE algorithm to group the similar instances.**

**Sample Dataset is given below.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Invoice**  **No** | **Stock**  **Code** | **Description** | **Quantity** | **Invoice**  **Date** | **Unit**  **Price** | **Customer**  **ID** | **Country** |
| **536365** | **85123A** | **WHITE HANGING HEART T-LIGHT HOLDER** | **6** | **12-01-2010 08:26** | **2.55** | **17850** | **United Kingdom** |
| **536365** | **71053** | **WHITE METAL LANTERN** | **6** | **12-01-2010 08:26** | **3.39** | **17850** | **United Kingdom** |
| **536365** | **84406B** | **CREAM CUPID HEARTS COAT HANGER** | **8** | **12-01-2010 08:26** | **2.75** | **17850** | **France** |
| **536365** | **84029G** | **KNITTED UNION FLAG HOT WATER BOTTLE** | **6** | **12-01-2010 08:26** | **3.39** | **17850** | **Italy** |
| **536365** | **84029E** | **RED WOOLLY HOTTIE WHITE HEART.** | **6** | **12-01-2010 08:26** | **3.39** | **17850** | **German** |
| **536365** | **22752** | **SET 7 BABUSHKA NESTING BOXES** | **2** | **12-01-2010 08:26** | **7.65** | **17850** | **United Kingdom** |

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import StandardScaler

# --- Step 1: Read dataset ---

df = pd.read\_csv("cure.csv")

# --- Step 2: Select numeric features (Quantity, UnitPrice) ---

X = df[['Quantity', 'UnitPrice']].values

# --- Step 3: Standardize the features ---

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# --- Step 4: Simple CURE algorithm implementation ---

num\_clusters = 2

num\_representatives = 2

shrink\_factor = 0.5

clusters = [[x] for x in X\_scaled] # Each point is its own cluster

def cluster\_centroid(cluster):

return np.mean(cluster, axis=0)

def distance(a, b):

return np.linalg.norm(a - b)

# Merge clusters until the desired number is reached

while len(clusters) > num\_clusters:

min\_dist = float('inf')

pair = None

for i in range(len(clusters)):

for j in range(i + 1, len(clusters)):

d = np.min([distance(a, b) for a in clusters[i] for b in clusters[j]])

if d < min\_dist:

min\_dist = d

pair = (i, j)

i, j = pair

merged = clusters[i] + clusters[j]

del clusters[j]

del clusters[i]

clusters.append(merged)

# Shrink representative points toward centroid

final\_clusters = []

for cluster in clusters:

centroid = cluster\_centroid(cluster)

reps = cluster[:num\_representatives]

reps = [centroid + shrink\_factor \* (r - centroid) for r in reps]

final\_clusters.append(np.array(reps))

# Assign labels for silhouette score

labels = np.zeros(len(X\_scaled))

for i, cluster in enumerate(clusters):

for point in cluster:

idx = np.where((X\_scaled == point).all(axis=1))[0][0]

labels[idx] = i

# --- Step 5: Calculate silhouette score ---

score = silhouette\_score(X\_scaled, labels)

print("\nOUTPUT")

print(f"\nAverage Silhouette Score: {score}\n")

# --- Step 6: Plot Clustered Data ---

plt.figure(figsize=(6, 4))

plt.scatter(X\_scaled[:, 0], X\_scaled[:, 1], c=labels, cmap='coolwarm', s=50)

for i, cluster in enumerate(clusters):

center = cluster\_centroid(cluster)

plt.scatter(center[0], center[1], marker='X', s=200, c='black', label=f'Cluster {i+1} Center')

plt.title("Clustered Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.show()

# --- Step 7: Print cluster details (same as image format) ---

for i, cluster in enumerate(clusters):

cluster = np.array(cluster)

mean\_f1 = np.mean(cluster[:, 0])

mean\_f2 = np.mean(cluster[:, 1])

print(f"Cluster {i+1}:")

print(f"Number of points: {len(cluster)}")

print(f"Mean of feature 1: {round(mean\_f1, 2)}")

print(f"Mean of feature 2: {round(mean\_f2, 2)}\n")