Slip 1

1. Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

# Install mlxtend if you haven't already

# !pip install mlxtend

import pandas as pd

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

# Sample groceries dataset

# Replace 'groceries.csv' with the path to your dataset

# Ensure dataset is in a format with each item in one transaction

data = pd.read\_csv('groceries.csv')

# One-hot encoding

basket = pd.get\_dummies(data, prefix='', prefix\_sep='').groupby(level=0, axis=1).sum()

# Applying Apriori algorithm

frequent\_itemsets = apriori(basket, min\_support=0.25, use\_colnames=True)

# Display frequent itemsets

print("Frequent itemsets:")

print(frequent\_itemsets)

# Generate association rules

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1)

print("\nAssociation Rules:")

print(rules)

2. Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical values in numeric format for a dataset.

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Load the Iris dataset

data = load\_iris()

df = pd.DataFrame(data.data, columns=data.feature\_names)

df['species'] = data.target # Convert categorical target to numeric format

# Scatter plot of sepal length vs sepal width

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

sns.scatterplot(x='sepal length (cm)', y='sepal width (cm)', hue='species', data=df, palette='viridis')

plt.title('Sepal Length vs Sepal Width')

plt.show()

# Scatter plot of petal length vs petal width

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

sns.scatterplot(x='petal length (cm)', y='petal width (cm)', hue='species', data=df, palette='viridis')

plt.title('Petal Length vs Petal Width')

plt.show()

Slip 2

Q.1. Write a python program to implement simple Linear Regression for predicting house price. First find all null values in a given dataset and remove them.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Load dataset (replace 'house\_prices.csv' with your dataset file path)

data = pd.read\_csv('house\_prices.csv')

# Check for and remove null values

print("Null values in each column before removing:")

print(data.isnull().sum())

data = data.dropna()

print("\nNull values after removing:")

print(data.isnull().sum())

# Assume dataset has columns 'Size' (feature) and 'Price' (target)

X = data[['Size']] # Independent variable (feature)

y = data['Price'] # Dependent variable (target)

# 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)

# Create and train the Linear Regression model

model = LinearRegression()

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

# Predict house prices

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

# Calculate and display Mean Squared Error

mse = mean\_squared\_error(y\_test, y\_pred)

print("\nMean Squared Error:", mse)

# Display coefficients

print("Intercept:", model.intercept\_)

print("Coefficient:", model.coef\_[0])

Q.2. The data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. Using data Wholesale

customer dataset compute agglomerative clustering to find out annual spending clients in the same region.

import pandas as pd

import numpy as np

from sklearn.cluster import AgglomerativeClustering

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset (replace 'Wholesale\_customers.csv' with your dataset path)

data = pd.read\_csv('Wholesale\_customers.csv')

# Check for any missing values

print("Null values in the dataset:")

print(data.isnull().sum())

# Select the numeric columns (annual spending categories)

X = data.drop(columns=['Channel', 'Region']) # Drop categorical columns (e.g., 'Channel', 'Region')

# Standardize the data

scaler = StandardScaler()

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

# Apply Agglomerative Clustering

model = AgglomerativeClustering(n\_clusters=4) # Choose number of clusters based on your analysis

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

# Visualize the clusters (For example, using 'Grocery' and 'Frozen' spending categories)

plt.figure(figsize=(10, 6))

sns.scatterplot(x=data['Grocery'], y=data['Frozen'], hue=data['Cluster'], palette='viridis', s=100)

plt.title('Agglomerative Clustering: Grocery vs Frozen Spending')

plt.xlabel('Grocery Spending')

plt.ylabel('Frozen Spending')

plt.legend(title='Cluster')

plt.show()

# Display the count of clients in each cluster

print("\nNumber of clients in each cluster:")

print(data['Cluster'].value\_counts())

Slip 3

Q.1. Write a python program to implement multiple Linear Regression for a house price dataset. Divide the dataset into training and testing data.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset

data = pd.read\_csv('house\_price\_dataset.csv') # Replace with the correct file path

# Select features and target variable

X = data[['feature1', 'feature2', 'feature3']] # Replace with relevant feature columns

y = data['price'] # Replace with the target column

# 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)

# Create the linear regression model

model = LinearRegression()

# Train the model

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

# Make predictions

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

Q.2. Use dataset crash.csv is an accident survivor’s dataset portal for USA hosted by

data.gov. The dataset contains passengers age and speed of vehicle (mph) at the time

of impact and fate of passengers (1 for survived and 0 for not survived) after a crash.

use logistic regression to decide if the age and speed can predict the survivability of the

passengers.

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

# Load the dataset

data = pd.read\_csv('crash.csv') # Replace with the correct file path

# Select features and target variable

X = data[['age', 'speed']] # Replace with relevant feature columns

y = data['survived'] # Replace with the target column, where 1 = survived, 0 = not survived

# 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)

# Create the logistic regression model

model = LogisticRegression()

# Train the model

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

# Make predictions

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

# Evaluate the model

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

print('Confusion Matrix:')

print(conf\_matrix)

Slip 4

Q.1. Write a python program to implement k-means algorithm on a mall\_customers dataset.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load the mall customers dataset

data = pd.read\_csv('mall\_customers.csv') # Adjust the path to your dataset

# Preprocessing (Standardizing data)

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(data[['Annual Income (k$)', 'Spending Score (1-100)']])

# Elbow Method to find the optimal number of clusters

inertia = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(scaled\_data)

inertia.append(kmeans.inertia\_)

# Plot Elbow Curve

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

plt.plot(range(1, 11), inertia, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

# Based on the elbow method, assume k=5 for optimal clusters

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(scaled\_data)

# Add cluster labels to the original dataset

data['Cluster'] = kmeans.labels\_

# Visualize the clusters

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

plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], c=data['Cluster'], cmap='viridis')

plt.title('Customer Segments')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()

Q.2. Write a python program to Implement Simple Linear Regression for predicting house price.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load the mall customers dataset

data = pd.read\_csv('mall\_customers.csv') # Adjust the path to your dataset

# Preprocessing (Standardizing data)

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(data[['Annual Income (k$)', 'Spending Score (1-100)']])

# Elbow Method to find the optimal number of clusters

inertia = []

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(scaled\_data)

inertia.append(kmeans.inertia\_)

# Plot Elbow Curve

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

plt.plot(range(1, 11), inertia, marker='o')

plt.title('Elbow Method for Optimal k')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

plt.show()

# Based on the elbow method, assume k=5 for optimal clusters

kmeans = KMeans(n\_clusters=5, random\_state=42)

kmeans.fit(scaled\_data)

# Add cluster labels to the original dataset

data['Cluster'] = kmeans.labels\_

# Visualize the clusters

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

plt.scatter(data['Annual Income (k$)'], data['Spending Score (1-100)'], c=data['Cluster'], cmap='viridis')

plt.title('Customer Segments')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.show()

Slip 5

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

# Load the dataset

data = pd.read\_csv('fuel\_consumption.csv') # Adjust the path to your dataset

# Preview the dataset

print(data.head())

# Assuming the dataset has the columns 'Cylinders', 'Engine Size', 'Fuel Consumption (L/100km)', 'CO2 Emissions (g/km)'

# Select the independent variables (features) and dependent variable (target)

X = data[['Cylinders', 'Engine Size', 'Fuel Consumption (L/100km)']] # Independent variables

y = data['CO2 Emissions (g/km)'] # Dependent variable

# 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)

# Standardizing the data

scaler = StandardScaler()

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

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

# Initialize the Linear Regression model

model = LinearRegression()

# Train the model

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

# Make predictions

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

# Plotting the actual vs predicted values

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

plt.scatter(y\_test, y\_pred, color='blue')

plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], color='red', lw=2)

plt.title('Actual vs Predicted CO2 Emissions')

plt.xlabel('Actual CO2 Emissions (g/km)')

plt.ylabel('Predicted CO2 Emissions (g/km)')

plt.show()

Q.2. Write a python program to implement k-nearest Neighbors ML algorithm to build

prediction model (Use iris Dataset)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features (sepal length, sepal width, petal length, petal width)

y = iris.target # Labels (species)

# Split the dataset 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)

# Standardizing the data (K-NN is distance-based, so scaling is important)

scaler = StandardScaler()

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

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

# Initialize the K-Nearest Neighbors classifier

k = 5 # Number of neighbors

knn = KNeighborsClassifier(n\_neighbors=k)

# Train the model

knn.fit(X\_train\_scaled, y\_train)

# Make predictions

y\_pred = knn.predict(X\_test\_scaled)

# Evaluate the model

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

print('Confusion Matrix:')

print(conf\_matrix)

# Plot the confusion matrix

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

plt.imshow(conf\_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title(f'Confusion Matrix for k={k}')

plt.colorbar()

plt.xticks(np.arange(3), iris.target\_names)

plt.yticks(np.arange(3), iris.target\_names)

plt.xlabel('Predicted label')

plt.ylabel('True label')

plt.show()

Slip 6

Q.1. Write a python program to implement Polynomial Linear Regression for

Boston Housing Dataset.

# Importing necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_boston

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

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

from sklearn.metrics import mean\_squared\_error

# Load Boston Housing dataset

boston = load\_boston()

X = boston.data

y = boston.target

# Splitting the dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Polynomial feature transformation

degree = 2 # Degree of the polynomial features

poly = PolynomialFeatures(degree=degree)

# Fit the polynomial features to the training data

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Create a linear regression model

model = LinearRegression()

# Train the model with the polynomial features

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

# Predict the target values

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

# Print results

print(f'Mean Squared Error: {mse}')

print(f'Root Mean Squared Error: {rmse}')

# Visualizing the actual vs predicted values for a simple feature (only for a univariate case)

plt.scatter(y\_test, y\_pred)

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted (Polynomial Regression)')

plt.show()

Q.2. Use K-means clustering model and classify the employees into various income groups

or clusters. Preprocess data if require (i.e. drop missing or null values).

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

import matplotlib.pyplot as plt

# Load dataset (assuming a CSV file with employee data)

# You can replace 'employee\_data.csv' with your actual dataset path

data = pd.read\_csv('employee\_data.csv')

# Check for missing values

print(data.isnull().sum())

# Preprocess the data: Drop or fill missing values (imputation)

imputer = SimpleImputer(strategy='mean') # You can also use 'median' or 'most\_frequent' for imputation

data\_imputed = pd.DataFrame(imputer.fit\_transform(data), columns=data.columns)

# You may want to focus on numerical columns (like income)

# If necessary, drop non-numerical columns (if they are not needed for clustering)

numerical\_data = data\_imputed.select\_dtypes(include=[np.number])

# Standardize the data (important for clustering algorithms)

scaler = StandardScaler()

scaled\_data = scaler.fit\_transform(numerical\_data)

# Apply K-means clustering

kmeans = KMeans(n\_clusters=4, random\_state=42) # Assuming 4 clusters for income groups

kmeans.fit(scaled\_data)

# Add the cluster labels to the original data

data\_imputed['Income\_Group'] = kmeans.labels\_

# Print the cluster centers and labels

print(kmeans.cluster\_centers\_)

print(data\_imputed[['Income\_Group']].head())

# Visualizing the clusters (if the data has 2 features for simplicity)

plt.scatter(scaled\_data[:, 0], scaled\_data[:, 1], c=kmeans.labels\_, cmap='viridis')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.title('K-means Clustering of Employees into Income Groups')

plt.show()

# If you want to save the data with clusters

data\_imputed.to\_csv('employees\_with\_clusters.csv', index=False)

Slip 7

Q.1. Fit the simple linear regression model to Salary\_positions.csv data. Predict the sa

of level 11 and level 12 employees.

# Import necessary libraries

import pandas as pd

from sklearn.linear\_model import LinearRegression

import numpy as np

# Load the dataset

df = pd.read\_csv('Salary\_positions.csv')

# Check the first few rows of the dataset

print(df.head())

# Step 1: Preprocess the data (if required, like dropping null values or converting columns)

# In this case, assume we have 'Position\_Level' and 'Salary' columns

# Drop rows with missing values (if any)

df = df.dropna()

# Step 2: Define the independent variable (X) and the dependent variable (y)

X = df[['Position\_Level']] # Feature: Position level

y = df['Salary'] # Target: Salary

# Step 3: Fit the simple linear regression model

model = LinearRegression()

model.fit(X, y)

# Step 4: Make predictions for level 11 and level 12 employees

levels = np.array([11, 12]).reshape(-1, 1) # Reshape for a 2D array input

salary\_predictions = model.predict(levels)

# Output the predictions

print(f"Predicted salary for level 11: {salary\_predictions[0]}")

print(f"Predicted salary for level 12: {salary\_predictions[1]}")

Q.2. Write a python program to implement Naive Bayes on weather forecast dataset

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# Example dataset (you can replace this with your own dataset)

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'High', 'Low', 'Low', 'Low', 'Low', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No']

}

# Creating a DataFrame

df = pd.DataFrame(data)

# Encoding categorical features

df\_encoded = pd.get\_dummies(df.drop('PlayTennis', axis=1))

df\_encoded['PlayTennis'] = df['PlayTennis'].map({'Yes': 1, 'No': 0})

# Splitting the dataset into features and target

X = df\_encoded

y = df['PlayTennis']

# Splitting 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)

# Initializing and training the Naive Bayes model

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

# Making predictions

y\_pred = nb\_model.predict(X\_test)

# Evaluating the model

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

print(f'Accuracy: {accuracy \* 100:.2f}%')

Slip 8

Q.1. Write a python program to categorize the given news text into one of the available 20

categories of news groups, using multinomial Naïve Bayes machine learning model

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import CountVectorizer

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import make\_pipeline

# Load the 20 newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all')

# Split the dataset into training and testing sets

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

# Create a pipeline with a CountVectorizer and MultinomialNB classifier

model = make\_pipeline(CountVectorizer(), MultinomialNB())

# Train the model

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

# Function to categorize a given news text

def categorize\_news(text):

prediction = model.predict([text])

category = newsgroups.target\_names[prediction[0]]

return category

# Example usage

news\_text = "NASA's new mission to Mars is groundbreaking and will help us understand the red planet's history."

category = categorize\_news(news\_text)

print(f"The news text belongs to the category: {category}")

Q.2. Write a python program to implement Decision Tree whether or not to play Tennis.

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

import pandas as pd

# Sample data for playing tennis (Outlook, Temperature, Humidity, Wind, PlayTennis)

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Mild', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Mild', 'Hot'],

'Humidity': ['High', 'High', 'High', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Strong', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

# Convert the data into a pandas DataFrame

df = pd.DataFrame(data)

# Convert categorical data to numeric values

df['Outlook'] = df['Outlook'].map({'Sunny': 0, 'Overcast': 1, 'Rain': 2})

df['Temperature'] = df['Temperature'].map({'Hot': 0, 'Mild': 1, 'Cool': 2})

df['Humidity'] = df['Humidity'].map({'High': 0, 'Low': 1})

df['Wind'] = df['Wind'].map({'Weak': 0, 'Strong': 1})

df['PlayTennis'] = df['PlayTennis'].map({'No': 0, 'Yes': 1})

# Features (X) and target (y)

X = df[['Outlook', 'Temperature', 'Humidity', 'Wind']]

y = df['PlayTennis']

# Initialize the Decision Tree classifier

clf = DecisionTreeClassifier()

# Train the classifier

clf.fit(X, y)

# Visualize the decision tree

tree.plot\_tree(clf, feature\_names=X.columns, class\_names=['No', 'Yes'], filled=True)

# Example usage: Predict if we should play tennis with a new input

new\_data = pd.DataFrame({'Outlook': [0], 'Temperature': [1], 'Humidity': [0], 'Wind': [1]})

prediction = clf.predict(new\_data)

print("Prediction (Play Tennis):", "Yes" if prediction[0] == 1 else "No")

Slip 9

Q.1. Implement Ridge Regression and Lasso regression model using boston\_houses.csv

and take only ‘RM’ and ‘Price’ of the houses. Divide the data as training and testing

data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms

and compare results

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import Ridge, Lasso

from sklearn.metrics import mean\_squared\_error

# Load the data from the CSV file

data = pd.read\_csv('boston\_houses.csv')

# Extract the relevant features: 'RM' and 'Price'

X = data[['RM']] # Feature (Number of Rooms)

y = data['Price'] # Target (House Price)

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize the Ridge and Lasso models

ridge\_model = Ridge(alpha=1.0)

lasso\_model = Lasso(alpha=0.1)

# Fit the models on the training data

ridge\_model.fit(X\_train, y\_train)

lasso\_model.fit(X\_train, y\_train)

# Predict the prices on the test set

ridge\_predictions = ridge\_model.predict(X\_test)

lasso\_predictions = lasso\_model.predict(X\_test)

# Calculate the Mean Squared Error (MSE) for both models

ridge\_mse = mean\_squared\_error(y\_test, ridge\_predictions)

lasso\_mse = mean\_squared\_error(y\_test, lasso\_predictions)

# Predict the price of a house with 5 rooms using both models

price\_ridge = ridge\_model.predict([[5]]) # Predict for 5 rooms

price\_lasso = lasso\_model.predict([[5]]) # Predict for 5 rooms

# Output the results

print(f"Ridge Regression Mean Squared Error: {ridge\_mse}")

print(f"Lasso Regression Mean Squared Error: {lasso\_mse}")

print(f"Predicted Price of a house with 5 rooms using Ridge Regression: ${price\_ridge[0]:,.2f}")

print(f"Predicted Price of a house with 5 rooms using Lasso Regression: ${price\_lasso[0]:,.2f}")

Q.2. Write a python program to implement Linear SVM using UniversalBank.csv

import pandas as pd

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

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the data from the CSV file

data = pd.read\_csv('UniversalBank.csv')

# Let's assume 'PersonalLoan' is the target variable, and we use all other columns as features

# Dropping columns that may not be useful like 'ID', 'ZIP Code', etc.

data = data.drop(['ID', 'ZIPCode'], axis=1)

# Encode the categorical variables (if any)

# For this example, assume 'SecuritiesAccount', 'CDAccount', and 'Online' are categorical

label\_encoder = LabelEncoder()

data['SecuritiesAccount'] = label\_encoder.fit\_transform(data['SecuritiesAccount'])

data['CDAccount'] = label\_encoder.fit\_transform(data['CDAccount'])

data['Online'] = label\_encoder.fit\_transform(data['Online'])

# Define the feature matrix (X) and the target variable (y)

X = data.drop('PersonalLoan', axis=1) # All columns except the target

y = data['PersonalLoan'] # Target variable (whether the person took a loan or not)

# Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Scale the features (SVMs perform better with standardized data)

scaler = StandardScaler()

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

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

# Initialize the Linear SVM model with a linear kernel

svm\_model = SVC(kernel='linear')

# Train the SVM model

svm\_model.fit(X\_train\_scaled, y\_train)

# Predict on the test set

y\_pred = svm\_model.predict(X\_test\_scaled)

# Evaluate the model

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

print(f"Accuracy of the Linear SVM model: {accuracy \* 100:.2f}%")

Slip 10

Q.1. Write a python program to transform data with Principal Component Analysis (PCA).

Use iris dataset.

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

# Load the iris dataset

data = load\_iris()

X = data.data

y = data.target

# Standardize the data

scaler = StandardScaler()

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

# Apply PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Create a DataFrame to view the result

df\_pca = pd.DataFrame(X\_pca, columns=['Principal Component 1', 'Principal Component 2'])

df\_pca['Target'] = y

# Display the transformed data

print(df\_pca.head())

Q.2. Write a Python program to prepare Scatter Plot for Iris Dataset. Convert Categorical

values in to numeric.

import seaborn as sns

import pandas as pd

import matplotlib.pyplot as plt

# Load the Iris dataset

iris = sns.load\_dataset('iris')

# Convert categorical 'species' column into numeric

iris['species'] = iris['species'].astype('category').cat.codes

# Create a scatter plot

sns.scatterplot(x='sepal\_length', y='sepal\_width', hue='species', data=iris)

# Show the plot

plt.title('Scatter Plot of Iris Dataset')

plt.show()

Slip 11

Q.1. Write a python program to implement Polynomial Regression for

Boston Housing Dataset.

import numpy as np

import pandas as pd

from sklearn.datasets import load\_boston

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

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Load the Boston housing dataset

boston = load\_boston()

X = boston.data

y = boston.target

# Split the dataset 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)

# Create a polynomial feature transformer (degree=2)

poly = PolynomialFeatures(degree=2)

# Transform the input features to higher degree features

X\_train\_poly = poly.fit\_transform(X\_train)

X\_test\_poly = poly.transform(X\_test)

# Train a linear regression model on the polynomial features

model = LinearRegression()

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

# Predict on the test data

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

# Calculate and print the mean squared error

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

# Visualize the first feature and its polynomial regression fit (for illustration)

plt.scatter(X\_test[:, 0], y\_test, color='blue', label='Actual')

plt.scatter(X\_test[:, 0], y\_pred, color='red', label='Predicted')

plt.title('Polynomial Regression: Boston Housing')

plt.xlabel('Feature 1 (CRIM)')

plt.ylabel('Target (Price)')

plt.legend()

plt.show()

Q.2. Write a python program to Implement Decision Tree classifier model on Data which

is extracted from images that were taken from genuine and forged banknote-like

specimens.

(refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

import urllib.request

# Step 1: Download the dataset from the UCI repository

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00267/data\_banknote\_authentication.csv"

file\_name = "banknote\_authentication.csv"

urllib.request.urlretrieve(url, file\_name)

# Step 2: Load the dataset into a pandas dataframe

data = pd.read\_csv(file\_name, header=None)

# Step 3: Assign the features and target variable

X = data.iloc[:, :-1] # Features (all columns except the last one)

y = data.iloc[:, -1] # Target (last column)

# Step 4: 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)

# Step 5: Initialize and train the Decision Tree Classifier

dt\_classifier = DecisionTreeClassifier(random\_state=42)

dt\_classifier.fit(X\_train, y\_train)

# Step 6: Make predictions on the test set

y\_pred = dt\_classifier.predict(X\_test)

# Step 7: Evaluate the model's performance

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

print(f"Accuracy of the Decision Tree Classifier: {accuracy \* 100:.2f}%")

Slip 12

Q.1. Write a python program to implement k-nearest Neighbors ML algorithm to build

prediction model (Use iris Dataset).

# Import necessary libraries

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.datasets import load\_iris

# Step 1: Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Target

# Step 2: Split the dataset 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: Initialize the KNN classifier

knn = KNeighborsClassifier(n\_neighbors=3)

# Step 4: Train the model using the training set

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

# Step 5: Make predictions on the test set

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

# Step 6: Evaluate the model's performance

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

print(f"Accuracy of the KNN classifier: {accuracy \* 100:.2f}%")

Q.2. Fit the simple linear regression and polynomial linear regression models to

Salary\_positions.csv data. Find which one is more accurately fitting to the given

data. Also predict the salaries of level 11 and level 12 employees.

# Importing the necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

# Step 1: Load the dataset

data = pd.read\_csv('Salary\_positions.csv')

# Assuming the dataset has columns 'Position Level' and 'Salary'

X = data.iloc[:, 1:2].values # Features (Position Level)

y = data.iloc[:, 2].values # Target (Salary)

# Step 2: Fit Simple Linear Regression model

linear\_regressor = LinearRegression()

linear\_regressor.fit(X, y)

# Step 3: Fit Polynomial Linear Regression model

poly = PolynomialFeatures(degree=4) # You can adjust the degree as needed

X\_poly = poly.fit\_transform(X)

poly\_regressor = LinearRegression()

poly\_regressor.fit(X\_poly, y)

# Step 4: Compare the models by calculating Mean Squared Error (MSE)

y\_pred\_linear = linear\_regressor.predict(X)

y\_pred\_poly = poly\_regressor.predict(X\_poly)

mse\_linear = mean\_squared\_error(y, y\_pred\_linear)

mse\_poly = mean\_squared\_error(y, y\_pred\_poly)

print(f'Mean Squared Error for Simple Linear Regression: {mse\_linear}')

print(f'Mean Squared Error for Polynomial Regression: {mse\_poly}')

# Step 5: Visualizing the results

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

# Plotting Simple Linear Regression

plt.subplot(1, 2, 1)

plt.scatter(X, y, color='red')

plt.plot(X, y\_pred\_linear, color='blue')

plt.title('Simple Linear Regression')

plt.xlabel('Position Level')

plt.ylabel('Salary')

# Plotting Polynomial Linear Regression

plt.subplot(1, 2, 2)

plt.scatter(X, y, color='red')

plt.plot(X, y\_pred\_poly, color='blue')

plt.title('Polynomial Linear Regression')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.tight\_layout()

plt.show()

# Step 6: Predict the salaries for level 11 and level 12 using both models

# For Polynomial Regression, need to transform the input before predicting

level\_11 = np.array([[11]])

level\_12 = np.array([[12]])

salary\_pred\_linear\_11 = linear\_regressor.predict(level\_11)

salary\_pred\_linear\_12 = linear\_regressor.predict(level\_12)

salary\_pred\_poly\_11 = poly\_regressor.predict(poly.transform(level\_11))

salary\_pred\_poly\_12 = poly\_regressor.predict(poly.transform(level\_12))

print(f'Predicted Salary for Level 11 (Linear Regression): {salary\_pred\_linear\_11[0]}')

print(f'Predicted Salary for Level 12 (Linear Regression): {salary\_pred\_linear\_12[0]}')

print(f'Predicted Salary for Level 11 (Polynomial Regression): {salary\_pred\_poly\_11[0]}')

print(f'Predicted Salary for Level 12 (Polynomial Regression): {salary\_pred\_poly\_12[0]}')

Slip 13

Q.1. Create RNN model and analyze the Google stock price dataset. Find out increasing or

decreasing trends of stock price for the next day.

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import yfinance as yf

from sklearn.preprocessing import MinMaxScaler

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

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, LSTM, Dropout

from tensorflow.keras.optimizers import Adam

# Step 1: Load the Google stock price dataset using yfinance

ticker = 'GOOGL'

data = yf.download(ticker, start='2010-01-01', end='2023-01-01')

# Step 2: Visualize the stock price (Closing price)

data['Close'].plot(figsize=(10,6))

plt.title(f'{ticker} Stock Price')

plt.xlabel('Date')

plt.ylabel('Price')

plt.show()

# Step 3: Preprocess the data (Scaling)

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_data = scaler.fit\_transform(data[['Close']])

# Step 4: Create a function to prepare data for RNN

def create\_dataset(data, time\_step=60):

X, y = [], []

for i in range(time\_step, len(data)):

X.append(data[i-time\_step:i, 0])

y.append(1 if data[i, 0] > data[i-1, 0] else 0) # 1 if price increased, 0 if decreased

return np.array(X), np.array(y)

# Step 5: Create dataset for training and testing

X, y = create\_dataset(scaled\_data)

# Reshaping X for LSTM (samples, time\_steps, features)

X = X.reshape(X.shape[0], X.shape[1], 1)

# 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)

# Step 6: Build the RNN model (LSTM)

model = Sequential()

# Adding LSTM layers

model.add(LSTM(units=100, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(units=100, return\_sequences=False))

model.add(Dropout(0.2))

# Adding output layer

model.add(Dense(units=1, activation='sigmoid')) # sigmoid for binary classification

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001), loss='binary\_crossentropy', metrics=['accuracy'])

# Step 7: Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Step 8: Predicting the next day's stock movement (increase or decrease)

predictions = model.predict(X\_test)

predictions = (predictions > 0.5).astype(int) # 1 if increase, 0 if decrease

# Step 9: Evaluate the model

accuracy = (predictions == y\_test).mean()

print(f'Accuracy: {accuracy \* 100:.2f}%')

# Step 10: Visualize the results

plt.figure(figsize=(10, 6))

plt.plot(y\_test[:50], color='red', label='True')

plt.plot(predictions[:50], color='blue', label='Predicted')

plt.title('Stock Price Movement Prediction')

plt.xlabel('Time')

plt.ylabel('Movement (1=Increase, 0=Decrease)')

plt.legend()

plt.show()

Q.2. Write a python program to implement simple Linear Regression for predicting house

price.

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Load the dataset (Here we create a simple synthetic dataset)

# Example dataset: Square footage and house prices

data = {

'Size (sqft)': [1000, 1500, 1800, 2400, 3000, 3500, 4000],

'Price ($)': [400000, 500000, 600000, 650000, 750000, 850000, 900000]

}

df = pd.DataFrame(data)

# Step 2: Preprocess the data

X = df[['Size (sqft)']].values # Features (Size of house)

y = df['Price ($)'].values # Target (Price of house)

# Step 3: 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)

# Step 4: Train the Linear Regression model

model = LinearRegression()

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')

# Step 7: Visualize the results (plotting the regression line)

plt.scatter(X, y, color='red') # Scatter plot of the actual data

plt.plot(X, model.predict(X), color='blue') # Regression line

plt.title('House Price Prediction')

plt.xlabel('Size (sqft)')

plt.ylabel('Price ($)')

plt.show()

Slip 14

Q.1. Create a CNN model and train it on mnist handwritten digit dataset. Using model find

out the digit written by a hand in a given image.

Import mnist dataset from tensorflow.keras.datasets.

# Import necessary libraries

import tensorflow as tf

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Step 1: Load the MNIST dataset

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

# Step 2: Preprocess the data

# Reshape the data to be 28x28x1 (for grayscale images) and normalize it

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1).astype('float32') / 255

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28, 1).astype('float32') / 255

# Convert labels to one-hot encoding

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Step 3: Build the CNN model

model = Sequential()

# Add convolutional layer with 32 filters and a 3x3 kernel, followed by max pooling

model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D((2, 2)))

# Add a second convolutional layer with 64 filters and a 3x3 kernel, followed by max pooling

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D((2, 2)))

# Add a third convolutional layer with 128 filters and a 3x3 kernel

model.add(Conv2D(128, (3, 3), activation='relu'))

# Flatten the output and add a dense layer

model.add(Flatten())

model.add(Dense(128, activation='relu'))

# Add dropout layer for regularization

model.add(Dropout(0.5))

# Add the output layer with 10 units (for 10 digits) and softmax activation

model.add(Dense(10, activation='softmax'))

# Step 4: Compile the model

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Step 5: Train the model

model.fit(x\_train, y\_train, epochs=5, batch\_size=64, validation\_data=(x\_test, y\_test))

# Step 6: Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test)

print(f'Test accuracy: {test\_acc \* 100:.2f}%')

# Step 7: Predicting the digit for a given image (for example, an image from the test set)

# Choose an image from the test set

image\_index = 0 # You can change this to test with different images

image = x\_test[image\_index].reshape(1, 28, 28, 1)

# Predict the class (digit)

predicted\_class = model.predict(image)

predicted\_digit = predicted\_class.argmax() # Get the index of the highest probability

# Display the image and the predicted digit

plt.imshow(x\_test[image\_index].reshape(28, 28), cmap='gray')

plt.title(f'Predicted Digit: {predicted\_digit}')

plt.show()

Q.2. Write a python program to find all null values in a given dataset and remove them.

Create your own dataset.

import pandas as pd

import numpy as np

# Step 1: Create a sample dataset

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David', np.nan],

'Age': [25, 30, np.nan, 35, 40],

'City': ['New York', np.nan, 'Los Angeles', 'Chicago', 'San Francisco'],

'Salary': [50000, 60000, 70000, np.nan, 80000]

}

# Create a DataFrame from the data

df = pd.DataFrame(data)

# Display the original dataset with null values

print("Original Dataset:")

print(df)

# Step 2: Find null values

print("\nNull values in the dataset:")

print(df.isnull().sum()) # Displays count of null values in each column

# Step 3: Remove rows with null values

df\_cleaned = df.dropna() # Drops rows with any null values

# Display the cleaned dataset

print("\nDataset after removing rows with null values:")

print(df\_cleaned)

# Alternatively, to remove columns with null values:

df\_cleaned\_columns = df.dropna(axis=1) # Drops columns with any null values

# Display the dataset after removing columns with null values

print("\nDataset after removing columns with null values:")

print(df\_cleaned\_columns)

Slip 15

Q.1. Create an ANN and train it on house price dataset classify the house price is above

average or below average.

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Step 1: Create a synthetic house price dataset

data = {

'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 5000, 6000, 7000, 8000],

'Bedrooms': [3, 4, 3, 5, 4, 6, 5, 6, 7, 8],

'Price ($)': [400000, 500000, 600000, 700000, 800000, 850000, 900000, 1000000, 1200000, 1400000]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Step 2: Preprocess the data

# Create the target variable (above average or below average price)

average\_price = df['Price ($)'].mean()

df['Price Category'] = np.where(df['Price ($)'] > average\_price, 1, 0) # 1 for above average, 0 for below average

# Features and target

X = df[['Size (sqft)', 'Bedrooms']] # Features

y = df['Price Category'] # Target (0: below average, 1: above average)

# Split the data 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: Standardize the features (ANNs benefit from normalized data)

scaler = StandardScaler()

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

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

# Step 4: Build the ANN model

model = Sequential()

# Input layer (2 features)

model.add(Dense(units=8, activation='relu', input\_dim=X\_train.shape[1]))

# Hidden layer

model.add(Dense(units=4, activation='relu'))

# Output layer (binary classification: 1 for above average, 0 for below average)

model.add(Dense(units=1, activation='sigmoid'))

# Step 5: Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Step 6: Train the model

model.fit(X\_train, y\_train, epochs=50, batch\_size=5, verbose=1)

# Step 7: Evaluate the model

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

y\_pred = (y\_pred > 0.5).astype(int) # Convert probabilities to binary class labels

# Step 8: Accuracy score

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

print(f'Accuracy: {accuracy \* 100:.2f}%')

# Example: Predicting a new house price category

new\_house = np.array([[2500, 4]]) # Example house with 2500 sqft and 4 bedrooms

new\_house = scaler.transform(new\_house) # Standardize the new input

predicted\_category = model.predict(new\_house)

predicted\_category = (predicted\_category > 0.5).astype(int)

print(f'Predicted Category for the new house: {"Above Average" if predicted\_category == 1 else "Below Average"}')

Q.2. Write a python program to implement multiple Linear Regression for a house price

dataset.

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Create a synthetic house price dataset

data = {

'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 5000, 6000, 7000, 8000],

'Bedrooms': [3, 4, 3, 5, 4, 6, 5, 6, 7, 8],

'Age (years)': [10, 15, 10, 20, 25, 30, 35, 40, 45, 50],

'Distance to City (miles)': [5, 6, 3, 10, 8, 15, 20, 30, 25, 10],

'Price ($)': [400000, 500000, 600000, 700000, 800000, 850000, 900000, 1000000, 1200000, 1400000]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Step 2: Preprocess the data

# Features (X) and Target (y)

X = df[['Size (sqft)', 'Bedrooms', 'Age (years)', 'Distance to City (miles)']] # Features

y = df['Price ($)'] # Target variable

# 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)

# Step 3: Standardize the features

scaler = StandardScaler()

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

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

# Step 4: Build the Multiple Linear Regression model

model = LinearRegression()

# Train the model

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

# Print the results

print(f'Mean Squared Error: {mse}')

print(f'Root Mean Squared Error: {rmse}')

print(f'R-squared: {r2}')

# Step 7: Predict the house price for a new house

new\_house = np.array([[2500, 4, 15, 10]]) # Example house: 2500 sqft, 4 bedrooms, 15 years old, 10 miles from the city

new\_house\_scaled = scaler.transform(new\_house) # Standardize the new input

predicted\_price = model.predict(new\_house\_scaled)

print(f'Predicted price for the new house: ${predicted\_price[0]:,.2f}')

Slip 16

Q.1. Create a two layered neural network with relu and sigmoid activation function

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

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

from sklearn.datasets import make\_classification

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

# Step 1: Create a synthetic dataset (binary classification)

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_classes=2, random\_state=42)

# Step 2: Split the data 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: Standardize the features

scaler = StandardScaler()

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

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

# Step 4: Create a Two-Layer Neural Network

model = Sequential()

# First layer with 64 neurons and ReLU activation function

model.add(Dense(units=64, activation='relu', input\_dim=X\_train.shape[1]))

# Output layer with 1 neuron and Sigmoid activation function (binary classification)

model.add(Dense(units=1, activation='sigmoid'))

# Step 5: Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Step 6: Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, verbose=1)

# Step 7: Evaluate the model

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

y\_pred = (y\_pred > 0.5).astype(int) # Convert probabilities to binary class labels

# Step 8: Evaluate accuracy

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

print(f'Accuracy: {accuracy \* 100:.2f}%')

Q.2. Write a python program to implement Simple Linear Regression for Boston housing dataset

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.datasets import load\_boston

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Step 1: Load the Boston Housing dataset

boston = load\_boston()

X = boston.data[:, 5].reshape(-1, 1) # We select only one feature (e.g., 'average number of rooms')

y = boston.target # The target variable (house prices)

# Step 2: 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)

# Step 3: Create a Simple Linear Regression model

model = LinearRegression()

# Step 4: Train the model

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

# Print evaluation results

print(f'Mean Squared Error: {mse}')

print(f'Root Mean Squared Error: {rmse}')

print(f'R-squared: {r2}')

# Step 7: Visualize the results

plt.scatter(X\_test, y\_test, color='blue', label='Actual Prices')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Regression Line')

plt.xlabel('Average Number of Rooms')

plt.ylabel('House Price ($1000s)')

plt.title('Simple Linear Regression on Boston Housing Dataset')

plt.legend()

plt.show()

# Step 8: Predict the price for a new value

new\_rooms = np.array([[6]]) # Example: House with 6 rooms

predicted\_price = model.predict(new\_rooms)

print(f'Predicted price for a house with 6 rooms: ${predicted\_price[0]:,.2f}K')

Slip 17

Q.1. Implement Ensemble ML algorithm on Pima Indians Diabetes Database with bagging

(random forest), boosting, voting and Stacking methods and display analysis

accordingly. Compare result.

import numpy as np

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier, StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

# Step 1: Load the Pima Indians Diabetes dataset (CSV format)

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

column\_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=column\_names)

# Step 2: Preprocess the data (split into features and target)

X = data.drop('Outcome', axis=1)

y = data['Outcome']

# Standardizing the features

scaler = StandardScaler()

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

# Step 3: Split the dataset into training and testing sets

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

# Step 4: Apply Bagging (Random Forest)

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

rf\_pred = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_pred)

# Step 5: Apply Boosting (AdaBoost)

ada\_model = AdaBoostClassifier(n\_estimators=100, random\_state=42)

ada\_model.fit(X\_train, y\_train)

ada\_pred = ada\_model.predict(X\_test)

ada\_accuracy = accuracy\_score(y\_test, ada\_pred)

# Step 6: Apply Voting Classifier

voting\_model = VotingClassifier(estimators=[

('rf', rf\_model),

('ada', ada\_model),

('svc', SVC(probability=True, random\_state=42))], voting='soft')

voting\_model.fit(X\_train, y\_train)

voting\_pred = voting\_model.predict(X\_test)

voting\_accuracy = accuracy\_score(y\_test, voting\_pred)

# Step 7: Apply Stacking Classifier

base\_learners = [

('rf', RandomForestClassifier(n\_estimators=50, random\_state=42)),

('dt', DecisionTreeClassifier(random\_state=42)),

('svc', SVC(probability=True, random\_state=42))

]

stacking\_model = StackingClassifier(estimators=base\_learners, final\_estimator=LogisticRegression())

stacking\_model.fit(X\_train, y\_train)

stacking\_pred = stacking\_model.predict(X\_test)

stacking\_accuracy = accuracy\_score(y\_test, stacking\_pred)

# Step 8: Compare Results

print(f"Random Forest (Bagging) Accuracy: {rf\_accuracy:.4f}")

print(f"AdaBoost (Boosting) Accuracy: {ada\_accuracy:.4f}")

print(f"Voting Classifier Accuracy: {voting\_accuracy:.4f}")

print(f"Stacking Classifier Accuracy: {stacking\_accuracy:.4f}")

.2. Write a python program to implement Multiple Linear Regression for a house price

dataset.

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

# Step 1: Load the dataset (Example synthetic dataset for house prices)

# You can replace this with your own dataset, e.g., `pd.read\_csv('your\_dataset.csv')`

data = {

'Square\_Feet': [1500, 1800, 2400, 3000, 3500, 4000, 4500],

'Bedrooms': [3, 4, 3, 4, 5, 4, 5],

'Age': [10, 15, 20, 5, 8, 10, 12],

'Price': [400000, 500000, 600000, 650000, 700000, 750000, 800000]

}

df = pd.DataFrame(data)

# Step 2: Split the data into features (X) and target (y)

X = df.drop('Price', axis=1) # Features (Square\_Feet, Bedrooms, Age)

y = df['Price'] # Target variable (Price)

# Step 3: 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)

# Step 4: Standardize the features (optional, especially if the features are on different scales)

scaler = StandardScaler()

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

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

# Step 5: Create and train the Multiple Linear Regression model

model = LinearRegression()

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

# Step 6: Make predictions on the test set

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

# Step 7: Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

# Step 8: Display results

print(f"Mean Squared Error (MSE): {mse}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R-squared (R2): {r2}")

# Step 9: Predicting the price for a new house (example)

new\_house = np.array([[3000, 4, 10]]) # Example: 3000 sqft, 4 bedrooms, 10 years old

new\_house\_scaled = scaler.transform(new\_house) # Standardize the new data

predicted\_price = model.predict(new\_house\_scaled)

print(f"Predicted price for the new house: ${predicted\_price[0]:,.2f}")

Slip 18

Q.1. Write a python program to implement k-means algorithm on a Diabetes dataset.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

# Step 1: Load the Diabetes dataset

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

column\_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',

'DiabetesPedigreeFunction', 'Age', 'Outcome']

data = pd.read\_csv(url, names=column\_names)

# Step 2: Preprocess the data (drop the 'Outcome' column as it is the target, and scale the features)

X = data.drop('Outcome', axis=1)

# Normalize the data using StandardScaler

scaler = StandardScaler()

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

# Step 3: Apply K-Means algorithm

kmeans = KMeans(n\_clusters=2, random\_state=42) # We are assuming 2 clusters for diabetes: 1 for patients, 0 for non-patients

kmeans.fit(X\_scaled)

# Step 4: Analyze the clustering results

labels = kmeans.labels\_ # Cluster labels for each data point

centroids = kmeans.cluster\_centers\_ # Cluster centroids

# Step 5: Evaluate the clustering using Silhouette Score

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

print(f"Silhouette Score: {sil\_score:.4f}")

# Step 6: Visualize the clusters (using the first two features for simplicity)

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

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

plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', color='red', s=200, label='Centroids')

plt.title('K-Means Clustering on Diabetes Dataset')

plt.xlabel('Pregnancies (scaled)')

plt.ylabel('Glucose (scaled)')

plt.legend()

plt.show()

# Step 7: Display cluster centers

print("Cluster Centers:")

print(centroids)

Q.2. Write a python program to implement Polynomial Linear Regression for

salary\_positions dataset.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

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

# Step 1: Load the Salary Positions dataset (example synthetic dataset)

# You can replace this with your own dataset

data = {

'Position Level': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'Salary': [40000, 45000, 50000, 60000, 65000, 70000, 80000, 85000, 90000, 95000]

}

df = pd.DataFrame(data)

# Step 2: Preprocess the data (separate features and target)

X = df[['Position Level']].values # Feature (Position Level)

y = df['Salary'].values # Target (Salary)

# Step 3: Transform the data into polynomial features (degree=4 for example)

poly = PolynomialFeatures(degree=4)

X\_poly = poly.fit\_transform(X)

# Step 4: Fit the polynomial regression model

poly\_reg = LinearRegression()

poly\_reg.fit(X\_poly, y)

# Step 5: Predict the results using the polynomial model

y\_pred = poly\_reg.predict(X\_poly)

# Step 6: Visualize the polynomial regression results

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

plt.scatter(X, y, color='red') # Actual data points

plt.plot(X, y\_pred, color='blue') # Polynomial regression curve

plt.title('Polynomial Regression (Salary vs. Position Level)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

# Step 7: Predict salary for new position levels (for example, Level 6.5 and Level 7.5)

new\_levels = np.array([[6.5], [7.5]])

new\_levels\_poly = poly.transform(new\_levels)

predicted\_salaries = poly\_reg.predict(new\_levels\_poly)

print(f"Predicted salary for Level 6.5: ${predicted\_salaries[0]:,.2f}")

print(f"Predicted salary for Level 7.5: ${predicted\_salaries[1]:,.2f}")

Slip 19

Q.1. Fit the simple linear regression and polynomial linear regression models to

Salary\_positions.csv data. Find which one is more accurately fitting to the given data.

Also predict the salaries of level 11 and level 12 employees.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

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

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Load the Salary Positions dataset (replace with actual path if needed)

# Assuming 'Salary\_positions.csv' has two columns: 'Position' and 'Salary'

data = pd.read\_csv('Salary\_positions.csv')

# Step 2: Extract features (X) and target (y)

X = data['Position'].values.reshape(-1, 1) # Feature: Position Level

y = data['Salary'].values # Target: Salary

# Step 3: Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Fit a Simple Linear Regression model

simple\_lr = LinearRegression()

simple\_lr.fit(X\_train, y\_train)

# Predict using Simple Linear Regression model

y\_pred\_simple = simple\_lr.predict(X\_test)

# Evaluate Simple Linear Regression

mse\_simple = mean\_squared\_error(y\_test, y\_pred\_simple)

r2\_simple = r2\_score(y\_test, y\_pred\_simple)

# Step 5: Fit a Polynomial Linear Regression model (degree=4 for example)

poly = PolynomialFeatures(degree=4)

X\_poly\_train = poly.fit\_transform(X\_train) # Transforming training features

X\_poly\_test = poly.transform(X\_test) # Transforming testing features

poly\_lr = LinearRegression()

poly\_lr.fit(X\_poly\_train, y\_train)

# Predict using Polynomial Linear Regression model

y\_pred\_poly = poly\_lr.predict(X\_poly\_test)

# Evaluate Polynomial Linear Regression

mse\_poly = mean\_squared\_error(y\_test, y\_pred\_poly)

r2\_poly = r2\_score(y\_test, y\_pred\_poly)

# Step 6: Compare both models' performance

print(f"Simple Linear Regression: MSE = {mse\_simple:.2f}, R2 = {r2\_simple:.2f}")

print(f"Polynomial Linear Regression: MSE = {mse\_poly:.2f}, R2 = {r2\_poly:.2f}")

# Step 7: Visualize both models

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

# Scatter plot of actual data points

plt.scatter(X, y, color='red', label='Actual data')

# Plot Simple Linear Regression result

plt.plot(X, simple\_lr.predict(X), color='blue', label='Simple Linear Regression')

# Plot Polynomial Linear Regression result

X\_grid = np.linspace(min(X), max(X), 100).reshape(-1, 1)

plt.plot(X\_grid, poly\_lr.predict(poly.transform(X\_grid)), color='green', label='Polynomial Regression')

plt.title('Simple vs Polynomial Linear Regression (Salary vs Position Level)')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 8: Predict salaries for Level 11 and Level 12 using both models

level\_11 = np.array([[11]])

level\_12 = np.array([[12]])

# Simple Linear Regression Predictions

salary\_11\_simple = simple\_lr.predict(level\_11)

salary\_12\_simple = simple\_lr.predict(level\_12)

# Polynomial Linear Regression Predictions

salary\_11\_poly = poly\_lr.predict(poly.transform(level\_11))

salary\_12\_poly = poly\_lr.predict(poly.transform(level\_12))

print(f"Predicted salary for Level 11 (Simple Linear): ${salary\_11\_simple[0]:,.2f}")

print(f"Predicted salary for Level 12 (Simple Linear): ${salary\_12\_simple[0]:,.2f}")

print(f"Predicted salary for Level 11 (Polynomial Linear): ${salary\_11\_poly[0]:,.2f}")

print(f"Predicted salary for Level 12 (Polynomial Linear): ${salary\_12\_poly[0]:,.2f}")

Q.2. Write a python program to implement Naive Bayes on weather forecast dataset.

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, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

# Step 1: Load the Weather Forecast dataset (replace with your own dataset if necessary)

# For this example, we create a synthetic dataset

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Mild', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Mild', 'Hot'],

'Humidity': ['High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'Low', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes']

}

df = pd.DataFrame(data)

# Step 2: Preprocess the data (Encode categorical variables)

label\_encoder = LabelEncoder()

# Encoding categorical columns

df['Outlook'] = label\_encoder.fit\_transform(df['Outlook'])

df['Temperature'] = label\_encoder.fit\_transform(df['Temperature'])

df['Humidity'] = label\_encoder.fit\_transform(df['Humidity'])

df['Wind'] = label\_encoder.fit\_transform(df['Wind'])

df['PlayTennis'] = label\_encoder.fit\_transform(df['PlayTennis'])

# Step 3: Split the dataset into training and testing sets

X = df.drop('PlayTennis', axis=1) # Features

y = df['PlayTennis'] # Target variable

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

# Step 4: Train the Naive Bayes model (GaussianNB for continuous data)

nb\_model = GaussianNB()

nb\_model.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred = nb\_model.predict(X\_test)

# Step 6: Evaluate the model

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Output the accuracy and confusion matrix

print(f"Accuracy of Naive Bayes model: {accuracy \* 100:.2f}%")

print("Confusion Matrix:")

print(conf\_matrix)

# Step 7: Visualize the confusion matrix using Seaborn

sns.heatmap(conf\_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'])

plt.title('Confusion Matrix - Naive Bayes')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

Slip 20

Q.1. Implement Ridge Regression, Lasso regression model using boston\_houses.csv and

take only ‘RM’ and ‘Price’ of the houses. divide the data as training and testing

data. Fit line using Ridge regression and to find price of a house if it contains 5

rooms. and compare results.

import pandas as pd

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

from sklearn.linear\_model import Ridge, Lasso

from sklearn.metrics import mean\_squared\_error

import numpy as np

# Step 1: Load the dataset

# Assuming the 'boston\_houses.csv' file has columns 'RM' and 'Price'.

df = pd.read\_csv('boston\_houses.csv')

# Step 2: Select the relevant features ('RM' and 'Price')

X = df[['RM']] # Feature: Number of rooms

y = df['Price'] # Target: House price

# Step 3: Split the data into training and testing sets (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train the Ridge Regression model

ridge\_model = Ridge(alpha=1.0) # Alpha is the regularization strength

ridge\_model.fit(X\_train, y\_train)

# Step 5: Train the Lasso Regression model

lasso\_model = Lasso(alpha=0.1) # Alpha is the regularization strength

lasso\_model.fit(X\_train, y\_train)

# Step 6: Predict the price of a house with 5 rooms using both models

rooms = np.array([[5]])

ridge\_prediction = ridge\_model.predict(rooms)

lasso\_prediction = lasso\_model.predict(rooms)

# Step 7: Evaluate the models on the test data

ridge\_y\_pred = ridge\_model.predict(X\_test)

lasso\_y\_pred = lasso\_model.predict(X\_test)

ridge\_mse = mean\_squared\_error(y\_test, ridge\_y\_pred)

lasso\_mse = mean\_squared\_error(y\_test, lasso\_y\_pred)

# Step 8: Compare the results

print(f"Ridge Regression predicted price for a house with 5 rooms: ${ridge\_prediction[0]:,.2f}")

print(f"Lasso Regression predicted price for a house with 5 rooms: ${lasso\_prediction[0]:,.2f}")

print(f"Ridge Regression Mean Squared Error: {ridge\_mse:.2f}")

print(f"Lasso Regression Mean Squared Error: {lasso\_mse:.2f}")

Q.2. Write a python program to implement Decision Tree whether or not to play Tennis.

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

# Sample dataset: Weather conditions and whether to play tennis

data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy', 'Sunny', 'Overcast', 'Overcast', 'Rainy'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'High', 'Normal', 'Normal', 'High', 'Normal', 'Normal', 'Normal', 'Normal', 'High', 'Normal'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

# Create a DataFrame

df = pd.DataFrame(data)

# Convert categorical data to numerical data

df\_encoded = pd.get\_dummies(df.drop('PlayTennis', axis=1))

# Encode the target variable (PlayTennis)

df\_encoded['PlayTennis'] = df['PlayTennis'].map({'Yes': 1, 'No': 0})

# Define features (X) and target variable (y)

X = df\_encoded

y = df\_encoded['PlayTennis']

# Train the Decision Tree classifier

clf = DecisionTreeClassifier()

clf.fit(X, y)

# Visualize the Decision Tree

tree.plot\_tree(clf, feature\_names=X.columns, class\_names=['No', 'Yes'], filled=True)

# Predict whether to play tennis based on new data

new\_data = pd.DataFrame({

'Outlook\_Sunny': [1],

'Outlook\_Overcast': [0],

'Outlook\_Rainy': [0],

'Temperature\_Hot': [0],

'Temperature\_Mild': [1],

'Temperature\_Cool': [0],

'Humidity\_High': [0],

'Humidity\_Normal': [1],

'Wind\_Weak': [1],

'Wind\_Strong': [0]

})

# Make a prediction

prediction = clf.predict(new\_data)

print("Prediction (1=Yes, 0=No):", prediction[0])

Slip 21

Q.1. Create a multiple linear regression model for house price dataset divide dataset into

train and test data while giving it to model and predict prices of house

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the dataset (replace 'house\_price.csv' with your actual file path)

# Sample data assumes columns 'SquareFeet', 'Bedrooms', 'Bathrooms', 'Location', 'Price'

df = pd.read\_csv('house\_price.csv')

# Preprocess the data (assuming 'Location' is categorical and needs encoding)

df\_encoded = pd.get\_dummies(df, drop\_first=True)

# Define features (X) and target variable (y)

X = df\_encoded.drop('Price', axis=1)

y = df\_encoded['Price']

# Split the dataset into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create the Linear Regression model

model = LinearRegression()

# Train the model using the training data

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

# Predict house prices on the test data

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Output the results

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

# Predict prices of new house data (Example: SquareFeet=2000, Bedrooms=3, Bathrooms=2)

new\_data = pd.DataFrame({

'SquareFeet': [2000],

'Bedrooms': [3],

'Bathrooms': [2],

'Location\_New York': [1], # Assuming one-hot encoding for location (New York)

# Add more columns for other locations as necessary

})

# Predict price of new data

predicted\_price = model.predict(new\_data)

print(f"Predicted House Price: {predicted\_price[0]}")

Q.2. Write a python program to implement Linear SVM using UniversalBank.csv.

import pandas as pd

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

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

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

# Load the UniversalBank dataset (replace with your actual file path)

df = pd.read\_csv('UniversalBank.csv')

# Preprocessing: Drop any irrelevant columns (like ID or Name)

df = df.drop(['ID', 'ZIP Code'], axis=1)

# Assuming the target variable is 'Personal Loan' (binary classification)

# and the rest are feature variables

X = df.drop('Personal Loan', axis=1)

y = df['Personal Loan']

# Encode categorical columns (if any)

X = pd.get\_dummies(X, drop\_first=True)

# Scale features (important for SVM to work well)

scaler = StandardScaler()

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

# Split the dataset into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

# Create the Linear SVM model

svm\_model = SVC(kernel='linear')

# Train the model

svm\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Evaluate the model

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

# Example: Predicting for new data (replace with actual data)

new\_data = pd.DataFrame({

'Age': [45],

'Experience': [20],

'Income': [100000],

'Family': [2],

'CCAvg': [1.5],

'Education\_2': [0], # Assuming Education has been encoded

'Education\_3': [1],

'Mortgage': [0],

'Securities Account': [0],

'CD Account': [1],

'Online': [1],

'CreditCard': [0]

})

# Scale the new data using the same scaler

new\_data\_scaled = scaler.transform(new\_data)

# Make a prediction for the new data

new\_prediction = svm\_model.predict(new\_data\_scaled)

print(f"\nPrediction for new data (1=Loan, 0=No Loan): {new\_prediction[0]}")

Slip 22

Q.1. Write a python program to implement simple Linear Regression for predicting house

price.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Sample dataset (replace with your actual data)

# Assume 'SquareFeet' and 'Price' are the features in your dataset

df = pd.read\_csv('house\_price.csv')

# Extract features and target variable

X = df[['SquareFeet']] # Feature (e.g., Square Feet)

y = df['Price'] # Target variable (Price)

# Split the dataset into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create and train the Linear Regression model

model = LinearRegression()

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

# Predict house prices on the test data

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

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

# Output the results

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

# Visualize the results (training data vs predicted values)

plt.scatter(X\_test, y\_test, color='blue', label='Actual Price')

plt.plot(X\_test, y\_pred, color='red', linewidth=2, label='Predicted Price')

plt.title('Simple Linear Regression: House Price Prediction')

plt.xlabel('Square Feet')

plt.ylabel('Price')

plt.legend()

plt.show()

# Example: Predicting for new data (SquareFeet = 1500)

new\_data = pd.DataFrame({'SquareFeet': [1500]})

predicted\_price = model.predict(new\_data)

print(f"Predicted House Price for 1500 square feet: ${predicted\_price[0]:,.2f}")

Q.2. Use Apriori algorithm on groceries dataset to find which items are brought together.

Use minimum support =0.25

To use the Apriori algorithm on a grocery dataset and find which items are frequently bought together with a minimum support of 0.25, you can follow the steps below. I'll walk you through the process with some explanations.

Ensure you have the necessary libraries installed before running the code:

pip install mlxtend pandas

Step-by-Step Guide to Implement Apriori on Grocery Dataset:

1. Load and preprocess the grocery dataset.

2. One-hot encode the data.

3. Apply the Apriori algorithm with a minimum support of 0.25.

4. Generate association rules.

5. Display frequent itemsets and association rules.

Here's the Python code for this process:

import pandas as pd

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

# Load the groceries dataset (replace with your actual file path)

# Assuming each row in the dataset represents a transaction and each column an item

df = pd.read\_csv('groceries.csv', header=None)

# Step 1: Convert the dataset into a one-hot encoded format

# We assume the dataset has one transaction per row, and each column represents an item purchased in that transaction

df\_onehot = pd.get\_dummies(df.stack()).sum(level=0)

# Step 2: Apply the Apriori algorithm with a minimum support of 0.25

frequent\_itemsets = apriori(df\_onehot, min\_support=0.25, use\_colnames=True)

# Step 3: Generate association rules with a minimum confidence of 0.7

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

# Step 4: Display frequent itemsets and association rules

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)

# Optional: You can filter and view specific rules, for example, rules with lift > 1

filtered\_rules = rules[rules['lift'] > 1]

print("\nFiltered Association Rules (Lift > 1):")

print(filtered\_rules)

Detailed Explanation:

1. Loading the Dataset:

The groceries dataset (groceries.csv) is loaded using pandas. This dataset should be structured such that each row represents a transaction, and each column represents an item. If an item was purchased, its value would be 1; if not, the value would be 0.

2. One-hot Encoding:

pd.get\_dummies(df.stack()) converts the dataset into a one-hot encoded format.

The stack() function reshapes the dataset by converting each item into a row per transaction, then pd.get\_dummies() creates binary columns for each item.

sum(level=0) combines the individual rows into a one-hot encoded dataframe where each column represents an item, and the value indicates whether the item was purchased in that transaction (1 for purchased, 0 for not purchased).

3. Apriori Algorithm:

The apriori() function from the mlxtend library is used to find frequent itemsets from the one-hot encoded dataset with a minimum support of 0.25. This means the algorithm will look for itemsets that appear in at least 25% of the transactions.

4. Association Rules:

The association\_rules() function generates the association rules based on the frequent itemsets. It uses the confidence metric, with a minimum threshold of 0.7 (i.e., only rules with 70% or higher confidence will be shown

Slip 23

Q.1. Fit the simple linear regression and polynomial linear regression models to

Salary\_positions.csv data. Find which one is more accurately fitting to the given

data. Also predict the salaries of level 11 and level 12 employees.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

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

# Step 1: Load the dataset

df = pd.read\_csv('Salary\_positions.csv') # Replace with your actual file path

# Assume the dataset has columns 'Position\_Level' and 'Salary'

X = df[['Position\_Level']].values # Feature: Position Level

y = df['Salary'].values # Target: Salary

# Step 2: Fit a Simple Linear Regression model

simple\_linear\_regressor = LinearRegression()

simple\_linear\_regressor.fit(X, y)

# Step 3: Fit a Polynomial Regression model (let's use degree=4 for this example)

poly = PolynomialFeatures(degree=4)

X\_poly = poly.fit\_transform(X)

polynomial\_regressor = LinearRegression()

polynomial\_regressor.fit(X\_poly, y)

# Step 4: Evaluate the models using R-squared and Mean Squared Error (MSE)

y\_pred\_simple = simple\_linear\_regressor.predict(X)

y\_pred\_poly = polynomial\_regressor.predict(X\_poly)

# Calculate R-squared for both models

r2\_simple = r2\_score(y, y\_pred\_simple)

r2\_poly = r2\_score(y, y\_pred\_poly)

# Calculate Mean Squared Error (MSE) for both models

mse\_simple = mean\_squared\_error(y, y\_pred\_simple)

mse\_poly = mean\_squared\_error(y, y\_pred\_poly)

print(f"Simple Linear Regression - R-squared: {r2\_simple}, MSE: {mse\_simple}")

print(f"Polynomial Regression - R-squared: {r2\_poly}, MSE: {mse\_poly}")

# Step 5: Visualize the models

plt.figure(figsize=(10, 6))

# Plot Simple Linear Regression

plt.scatter(X, y, color='red', label='Data Points')

plt.plot(X, y\_pred\_simple, color='blue', label='Simple Linear Regression')

# Plot Polynomial Regression (with a smooth curve)

X\_grid = np.arange(min(X), max(X), 0.1) # For smooth curve

X\_grid = X\_grid.reshape((len(X\_grid), 1))

y\_grid = polynomial\_regressor.predict(poly.transform(X\_grid))

plt.plot(X\_grid, y\_grid, color='green', label='Polynomial Regression')

plt.title('Simple Linear Regression vs Polynomial Regression')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 6: Predict salaries for level 11 and 12 employees using both models

level\_11 = np.array([[11]])

level\_12 = np.array([[12]])

salary\_11\_simple = simple\_linear\_regressor.predict(level\_11)

salary\_12\_simple = simple\_linear\_regressor.predict(level\_12)

salary\_11\_poly = polynomial\_regressor.predict(poly.transform(level\_11))

salary\_12\_poly = polynomial\_regressor.predict(poly.transform(level\_12))

print(f"Predicted Salary for Level 11 (Simple Linear): {salary\_11\_simple[0]}")

print(f"Predicted Salary for Level 12 (Simple Linear): {salary\_12\_simple[0]}")

print(f"Predicted Salary for Level 11 (Polynomial): {salary\_11\_poly[0]}")

print(f"Predicted Salary for Level 12 (Polynomial): {salary\_12\_poly[0]}")

Q.2. Write a python program to find all null values from a dataset and remove them.

To write a Python program that finds and removes all null values from a dataset, we can use pandas. The general steps are:

1. Load the dataset into a pandas DataFrame.

2. Check for null values in the dataset.

3. Remove rows or columns containing null values, depending on the desired behavior.

Here is the Python code to find and remove null values from a dataset:

import pandas as pd

# Step 1: Load the dataset (replace 'your\_dataset.csv' with the actual file path)

df = pd.read\_csv('your\_dataset.csv')

# Step 2: Find all null values

print("Null values in each column:")

print(df.isnull().sum()) # Shows the number of null values in each column

# Step 3: Remove rows with any null values

df\_cleaned = df.dropna()

# Alternatively, if you want to remove columns with any null values, use:

# df\_cleaned = df.dropna(axis=1)

# Step 4: Verify the removal of null values

print("\nNull values after removal:")

print(df\_cleaned.isnull().sum()) # Should show 0 for all columns

# Optionally, you can save the cleaned dataset to a new file

df\_cleaned.to\_csv('cleaned\_dataset.csv', index=False)

# Step 5: Display the cleaned DataFrame (if needed)

print("\nCleaned DataFrame:")

print(df\_cleaned.head())

Explanation:

1. Loading the dataset:

The dataset is loaded into a pandas DataFrame using pd.read\_csv(). Make sure to replace 'your\_dataset.csv' with the actual path to your dataset.

2. Finding null values:

The isnull() function is used to identify null values. df.isnull().sum() will display the number of null values in each column.

3. Removing null values:

The dropna() function removes rows containing any null values by default. If you want to remove columns containing null values, set the axis parameter to 1 (dropna(axis=1)).

You can also specify how to handle null values using the how parameter, e.g., dropna(how='all') to drop rows where all values are null.

4. Saving the cleaned dataset:

After cleaning the data, the to\_csv() function saves the cleaned DataFrame to a new CSV file (optional).

5. Verification:

After removal, we print the count of null values again to ensure that all null values have been removed.

Example Output:

Assuming the dataset had some null values, here's an example of what the output might look like:

Null values in each column:

Column1 2

Column2 0

Column3 5

dtype: int64

Null values after removal:

Column1 0

Column2 0

Column3 0

dtype: int64

Cleaned DataFrame:

Column1 Column2 Column3

0 1 5 9

1 2 6 10

...

This indicates that the null values were removed, and all columns no longer contain null values.

Let me know if you need further modifications or explanations!

Slip 24

Q.1. Write a python program to Implement Decision Tree classifier model on Data which is

extracted from images that were taken from genuine and forged banknote-like

specimens.

(refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)

To implement a Decision Tree Classifier model on the Banknote Authentication dataset from UCI, you'll need to follow the steps below. This dataset consists of features extracted from images of genuine and forged banknotes.

Steps to implement the Decision Tree classifier:

1. Download the dataset from the UCI repository.

2. Load the dataset using pandas.

3. Preprocess the data (e.g., handling missing values, splitting the data into training and testing sets).

4. Train the Decision Tree Classifier.

5. Evaluate the model using accuracy, confusion matrix, or other metrics.

6. Make predictions.

Full Python code:

First, ensure you have the necessary libraries installed:

pip install pandas scikit-learn matplotlib

Python Program for Decision Tree Classifier:

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

import matplotlib.pyplot as plt

from sklearn import tree

# Step 1: Load the Banknote Authentication dataset

# Download dataset from UCI repository: https://archive.ics.uci.edu/dataset/267/banknote+authentication

# Ensure the dataset is saved locally, then read it

df = pd.read\_csv('data\_banknote\_authentication.csv', header=None)

# Step 2: Preprocess the data

# Rename columns for easier understanding

df.columns = ['Variance', 'Skewness', 'Curtosis', 'Entropy', 'Class']

# Split the data into features (X) and target (y)

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

y = df['Class'] # Target variable

# Step 3: Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train a Decision Tree Classifier

dt\_classifier = DecisionTreeClassifier(random\_state=42)

dt\_classifier.fit(X\_train, y\_train)

# Step 5: Make predictions on the test set

y\_pred = dt\_classifier.predict(X\_test)

# Step 6: Evaluate the model

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy of Decision Tree Classifier: {accuracy \* 100:.2f}%")

print("\nConfusion Matrix:")

print(conf\_matrix)

# Step 7: Visualize the decision tree

plt.figure(figsize=(12, 8))

tree.plot\_tree(dt\_classifier, filled=True, feature\_names=X.columns, class\_names=['Genuine', 'Forged'], rounded=True)

plt.title("Decision Tree Classifier for Banknote Authentication")

plt.show()

Explanation:

1. Loading the Dataset:

The dataset is loaded from a CSV file using pandas.read\_csv(). Ensure that the dataset is saved locally (you can download it from UCI's Banknote Authentication dataset).

2. Preprocessing the Data:

The columns are renamed for clarity: Variance, Skewness, Curtosis, Entropy, and Class (where Class is the target variable indicating whether the banknote is genuine (0) or forged (1)).

The dataset is split into features X (all columns except Class) and the target variable y (the Class column).

3. Splitting the Data:

The dataset is split into a training set (80%) and a test set (20%) using train\_test\_split() from sklearn.model\_selection.

4. Training the Decision Tree Classifier:

A DecisionTreeClassifier() from sklearn.tree is used to train the model on the training data (X\_train, y\_train).

5. Prediction and Evaluation:

The model is used to predict the target values for the test set (X\_test).

The accuracy of the model is computed using accuracy\_score() from sklearn.metrics.

The confusion matrix is printed to assess the model's performance in terms of true positives, true negatives, false positives, and false negatives.

6. Visualizing the Decision Tree:

The plot\_tree() function from sklearn.tree is used to visualize the trained decision tree.

Example Output:

Accuracy: The model's performance is printed as a percentage.

Accuracy of Decision Tree Classifier: 99.00%

Confusion Matrix:

Confusion Matrix:

[[150 2]

[ 3 145]]

The confusion matrix shows how many predictions were correct (true positives and true negatives) and how many were incorrect (false positives and false negatives).

Visualization: The decision tree will be plotted showing the rules that the model has learned to classify the banknotes as genuine or forged.

Conclusion:

The Decision Tree Classifier is a simple but powerful model for classification tasks like banknote authentication.

By evaluating accuracy and inspecting the confusion matrix, you can assess the performance of the classifier.

The decision tree visualization provides insight into how the model makes its decisions based on the input features.

Let me know if you need further details or adjustments!

Q.2. Write a python program to implement linear SVM using UniversalBank.csv.

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

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

import matplotlib.pyplot as plt

# Step 1: Load the UniversalBank dataset

# Replace 'UniversalBank.csv' with your actual file path

df = pd.read\_csv('UniversalBank.csv')

# Step 2: Preprocess the data

# Assume 'Personal Loan' is the target variable and rest are features.

# Drop any unnecessary columns, for example, 'ID' and 'ZIP Code'.

df = df.drop(['ID', 'ZIP Code'], axis=1)

# Convert categorical variables into numeric values if needed (for example, 'Personal Loan')

# Assume that all columns are numeric except for the target variable 'Personal Loan'.

# If there are any non-numeric columns, you can use pd.get\_dummies() to encode them.

# Here, we proceed assuming all columns except 'Personal Loan' are numeric.

# Features (X) and target variable (y)

X = df.drop('Personal Loan', axis=1)

y = df['Personal Loan']

# Step 3: Split the data into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Standardize the features (SVMs are sensitive to feature scaling)

scaler = StandardScaler()

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

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

# Step 5: Train the Linear SVM classifier

svm\_classifier = SVC(kernel='linear', random\_state=42)

svm\_classifier.fit(X\_train\_scaled, y\_train)

# Step 6: Make predictions on the test set

y\_pred = svm\_classifier.predict(X\_test\_scaled)

# Step 7: Evaluate the model

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

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print(f"Accuracy of the Linear SVM model: {accuracy \* 100:.2f}%")

print("\nConfusion Matrix:")

print(conf\_matrix)

# Step 8: Visualize the confusion matrix

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

plt.imshow(conf\_matrix, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.colorbar()

tick\_marks = range(len(conf\_matrix))

plt.xticks(tick\_marks, ['Not Approved', 'Approved'])

plt.yticks(tick\_marks, ['Not Approved', 'Approved'])

plt.ylabel('True label')

plt.xlabel('Predicted label')

# Plotting the matrix

plt.show()

Slip 25

Q.1. Write a python program to implement Polynomial Regression for house price dataset.

import numpy as np

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Step 1: Load the dataset

# Assuming 'house\_price.csv' is a CSV file with columns 'Size' (independent variable) and 'Price' (dependent variable)

df = pd.read\_csv('house\_price.csv') # Replace with your actual dataset path

# Step 2: Preprocess the data

# Let's assume the dataset has two columns: 'Size' (feature) and 'Price' (target)

X = df[['Size']].values # Feature (Size of the house)

y = df['Price'].values # Target variable (Price of the house)

# Step 3: Split the data into training and testing sets (80% for training, 20% for testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Polynomial Transformation of Features (Degree = 3 for cubic polynomial)

degree = 3

poly\_reg = PolynomialFeatures(degree=degree)

X\_train\_poly = poly\_reg.fit\_transform(X\_train)

X\_test\_poly = poly\_reg.transform(X\_test)

# Step 5: Train the Polynomial Regression Model

poly\_reg\_model = LinearRegression()

poly\_reg\_model.fit(X\_train\_poly, y\_train)

# Step 6: Make predictions using the trained model

y\_pred = poly\_reg\_model.predict(X\_test\_poly)

# Step 7: Evaluate the Model

# Calculate the Mean Squared Error (MSE) and R^2 score

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

print(f"R^2 Score: {r2}")

# Step 8: Visualize the Polynomial Regression Model

# Plotting the training set results

plt.scatter(X, y, color='blue') # Plot original data

plt.plot(X, poly\_reg\_model.predict(poly\_reg.transform(X)), color='red') # Plot polynomial regression model

plt.title('Polynomial Regression (Degree = 3)')

plt.xlabel('Size of the House')

plt.ylabel('Price')

plt.show()

# Optional: Visualize predictions for the test set

plt.scatter(X\_test, y\_test, color='blue') # Actual test data points

plt.plot(X, poly\_reg\_model.predict(poly\_reg.transform(X)), color='red') # Polynomial regression curve

plt.title('Polynomial Regression Prediction')

plt.xlabel('Size of the House')

plt.ylabel('Price')

plt.show()

Q.2. Create a two layered neural network with relu and sigmoid activation function

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import BinaryCrossentropy

# Step 1: Create synthetic data for binary classification

# X will be input data, y will be the labels (0 or 1)

# Let's assume 1000 samples and 2 features (for simplicity).

X = np.random.rand(1000, 2) # 1000 samples, 2 features

y = (X[:, 0] + X[:, 1] > 1).astype(int) # Target: Sum of features > 1 (binary classification)

# Step 2: Define the neural network model

model = Sequential()

# Add the first layer (Dense layer) with ReLU activation

model.add(Dense(units=8, input\_dim=2, activation='relu'))

# Add the second layer (Dense layer) with Sigmoid activation

model.add(Dense(units=1, activation='sigmoid'))

# Step 3: Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001),

loss=BinaryCrossentropy(),

metrics=['accuracy'])

# Step 4: Train the model

model.fit(X, y, epochs=10, batch\_size=32, validation\_split=0.2)

# Step 5: Evaluate the model

loss, accuracy = model.evaluate(X, y)

print(f"Final model accuracy: {accuracy \* 100:.2f}%")

# Step 6: Make predictions

predictions = model.predict(X[:5])

print(f"Predictions for first 5 samples: {predictions}")

Slip 26

Q.1. Create KNN model on Indian diabetes patient’s database and predict whether a new

patient is diabetic (1) or not (0). Find optimal value of K.

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

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

import matplotlib.pyplot as plt

# Step 1: Load the Indian Diabetes dataset

# Replace 'diabetes.csv' with the actual path of the dataset

df = pd.read\_csv('diabetes.csv')

# Step 2: Preprocess the data

# Handling missing values - Replace 0 values in certain columns (like Glucose, BMI, etc.) with the mean of that column

df.replace(0, np.nan, inplace=True)

df.fillna(df.mean(), inplace=True)

# Step 3: Split the data into features (X) and target (y)

X = df.drop('Outcome', axis=1) # Features

y = df['Outcome'] # Target variable

# Step 4: Split the dataset into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 5: Feature scaling (KNN is sensitive to the scale of the data)

scaler = StandardScaler()

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

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

# Step 6: Find the optimal value of K

# We will try different values of K and select the one that gives the best accuracy

k\_range = range(1, 21)

k\_accuracies = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train\_scaled, y\_train)

y\_pred = knn.predict(X\_test\_scaled)

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

k\_accuracies.append(accuracy)

# Plot K values vs Accuracy

plt.plot(k\_range, k\_accuracies)

plt.xlabel('Number of Neighbors (K)')

plt.ylabel('Accuracy')

plt.title('KNN: Accuracy vs K value')

plt.show()

# Step 7: Choose the best K value (highest accuracy)

optimal\_k = k\_range[k\_accuracies.index(max(k\_accuracies))]

print(f"Optimal K value: {optimal\_k}")

# Step 8: Train the KNN model with the optimal K

knn = KNeighborsClassifier(n\_neighbors=optimal\_k)

knn.fit(X\_train\_scaled, y\_train)

# Step 9: Evaluate the model

y\_pred = knn.predict(X\_test\_scaled)

print("Classification Report:")

print(classification\_report(y\_test, y\_pred))

# Step 10: Predict whether a new patient is diabetic (1) or not (0)

# Example: New patient data

new\_patient = np.array([[5, 116, 74, 0, 0, 25.6, 0.201, 45]]) # Replace with new patient values

new\_patient\_scaled = scaler.transform(new\_patient)

prediction = knn.predict(new\_patient\_scaled)

print(f"The new patient is {'Diabetic' if prediction[0] == 1 else 'Not Diabetic'}")

Q.2. Use Apriori algorithm on groceries dataset to find which items are brought together.

Use minimum support =0.25

import pandas as pd

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

# Step 1: Load the groceries dataset (replace with the actual path)

# The dataset is assumed to be in a transactional format where each row represents a transaction

# and each column represents an item, with 1 indicating the item was bought and 0 if it wasn't.

# Example dataset (in real-world use, load your actual dataset)

# For example, the dataset might look like:

# Bread, Milk, Butter

# 1 1 1

# 1 0 1

# 1 1 0

# 0 1 1

data = {

'Bread': [1, 1, 1, 0],

'Milk': [1, 0, 1, 1],

'Butter': [1, 1, 0, 1],

'Cheese': [0, 1, 1, 1],

}

# Convert the dictionary to a DataFrame

df = pd.DataFrame(data)

# Step 2: Apply the Apriori algorithm with a minimum support of 0.25

# Find frequent itemsets with a minimum support of 0.25

frequent\_itemsets = apriori(df, min\_support=0.25, use\_colnames=True)

# Step 3: Generate association rules from the frequent itemsets

# We will generate rules with a minimum confidence of 0.7

rules = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=0.7)

# Step 4: Display the frequent itemsets and association rules

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)

# Optional: You can filter and view specific rules if you want, for example, rules with lift > 1

filtered\_rules = rules[rules['lift'] > 1]

print("\nFiltered Association Rules (Lift > 1):")

print(filtered\_rules)

Slip 27

Q.1. Create a multiple linear regression model for house price dataset divide dataset into

train and test data while giving it to model and predict prices of house.

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import StandardScaler

# Step 1: Load the dataset (replace 'house\_prices.csv' with the actual file path)

df = pd.read\_csv('house\_prices.csv')

# Step 2: Preprocess the data

# For example, let's assume the dataset has these columns:

# 'Size', 'Bedrooms', 'Bathrooms', 'Location', 'Price'

# We'll handle categorical features and fill any missing values (if any).

# Handling missing values (if any)

df.fillna(df.mean(), inplace=True)

# Convert categorical columns like 'Location' to numeric values (One-Hot Encoding)

df = pd.get\_dummies(df, drop\_first=True)

# Step 3: Define the feature set (X) and target variable (y)

X = df.drop('Price', axis=1) # Features (exclude 'Price')

y = df['Price'] # Target variable (Price)

# Step 4: Split the dataset into training and testing sets (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 5: Train the Multiple Linear Regression model

model = LinearRegression()

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

# Step 6: Predict house prices using the test set

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

# Step 7: Evaluate the model

# Calculate Mean Squared Error (MSE) and R-squared (R2) score

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse}")

print(f"R-squared: {r2}")

# Step 8: Display the predicted house prices along with actual prices for comparison

comparison = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print(comparison.head())

# Example: Predict the price of a new house (new feature values)

# Example: Size = 2000 sq ft, Bedrooms = 3, Bathrooms = 2, Location = 1 (encoded value)

new\_house = pd.DataFrame({'Size': [2000], 'Bedrooms': [3], 'Bathrooms': [2], 'Location\_2': [1], 'Location\_3': [0]})

predicted\_price = model.predict(new\_house)

print(f"Predicted Price for the new house: {predicted\_price[0]}")

Q.2. Fit the simple linear regression and polynomial linear regression models to

Salary\_positions.csv data. Find which one is more accurately fitting to the given data.

Also predict the salaries of level 11 and level 12 employees.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 1: Load the dataset (replace 'Salary\_positions.csv' with the actual file path)

df = pd.read\_csv('Salary\_positions.csv')

# Step 2: Preprocess the data

# Let's assume the dataset has two columns: 'Level' and 'Salary'.

X = df[['Level']].values # Feature: Employee Level

y = df['Salary'].values # Target: Salary

# Step 3: Fit a Simple Linear Regression model

simple\_lr = LinearRegression()

simple\_lr.fit(X, y)

# Step 4: Fit a Polynomial Linear Regression model

# Let's try polynomial degrees from 2 to 4

poly\_reg = PolynomialFeatures(degree=4)

X\_poly = poly\_reg.fit\_transform(X)

polynomial\_lr = LinearRegression()

polynomial\_lr.fit(X\_poly, y)

# Step 5: Evaluate both models

# Predicting with the Simple Linear Regression model

y\_pred\_simple = simple\_lr.predict(X)

# Predicting with the Polynomial Linear Regression model

y\_pred\_poly = polynomial\_lr.predict(poly\_reg.fit\_transform(X))

# Calculate R^2 (R-squared) score for both models to evaluate fit

r2\_simple = r2\_score(y, y\_pred\_simple)

r2\_poly = r2\_score(y, y\_pred\_poly)

# Display results

print(f"R-squared for Simple Linear Regression: {r2\_simple}")

print(f"R-squared for Polynomial Linear Regression: {r2\_poly}")

# Step 6: Predict Salaries for Level 11 and Level 12

level\_11 = np.array([[11]])

level\_12 = np.array([[12]])

# Simple linear regression predictions

salary\_level\_11\_simple = simple\_lr.predict(level\_11)

salary\_level\_12\_simple = simple\_lr.predict(level\_12)

# Polynomial regression predictions

salary\_level\_11\_poly = polynomial\_lr.predict(poly\_reg.transform(level\_11))

salary\_level\_12\_poly = polynomial\_lr.predict(poly\_reg.transform(level\_12))

print(f"Predicted Salary for Level 11 (Simple LR): {salary\_level\_11\_simple[0]}")

print(f"Predicted Salary for Level 12 (Simple LR): {salary\_level\_12\_simple[0]}")

print(f"Predicted Salary for Level 11 (Polynomial LR): {salary\_level\_11\_poly[0]}")

print(f"Predicted Salary for Level 12 (Polynomial LR): {salary\_level\_12\_poly[0]}")

# Step 7: Visualize the results (Optional)

# Visualizing Simple Linear Regression results

plt.scatter(X, y, color='red')

plt.plot(X, y\_pred\_simple, color='blue')

plt.title('Simple Linear Regression')

plt.xlabel('Employee Level')

plt.ylabel('Salary')

plt.show()

# Visualizing Polynomial Linear Regression results

plt.scatter(X, y, color='red')

plt.plot(X, y\_pred\_poly, color='blue')

plt.title('Polynomial Linear Regression')

plt.xlabel('Employee Level')

plt.ylabel('Salary')

plt.show()

Slip 28

Q.1. Write a python program to categorize the given news text into one of the available 20

categories of news groups, using multinomial Naïve Bayes machine learning model.

import pandas as pd

from sklearn.datasets import fetch\_20newsgroups

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

# Step 1: Load the 20 newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all')

# Step 2: Preprocess the data

# Split the data into features (X) and target labels (y)

X = newsgroups.data # Text data

y = newsgroups.target # Target labels (news categories)

# Step 3: Convert the text data into numerical data using TF-IDF Vectorization

tfidf\_vectorizer = TfidfVectorizer(stop\_words='english')

X\_tfidf = tfidf\_vectorizer.fit\_transform(X)

# Step 4: Split the dataset into training and testing sets

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

# Step 5: Train the Multinomial Naive Bayes model

naive\_bayes = MultinomialNB()

naive\_bayes.fit(X\_train, y\_train)

# Step 6: Predict the labels for the test set

y\_pred = naive\_bayes.predict(X\_test)

# Step 7: Evaluate the model

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

print(f"Accuracy: {accuracy \* 100:.2f}%")

# Display classification report for a more detailed evaluation

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names))

# Step 8: Predict the category for a new news text

new\_text = ["NASA is planning a new mission to Mars to study the planet's surface. The rover will be launched next year."]

new\_text\_tfidf = tfidf\_vectorizer.transform(new\_text)

predicted\_category = naive\_bayes.predict(new\_text\_tfidf)

# Output the predicted category

print("\nPredicted Category for the new text:")

print(newsgroups.target\_names[predicted\_category][0])

Q.2. Classify the iris flowers dataset using SVM and find out the flower type depending on

the given input data like sepal length, sepal width, petal length and petal width. Find

accuracy of all SVM kernels.

import pandas as pd

from sklearn import datasets

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Step 1: Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features: sepal length, sepal width, petal length, petal width

y = iris.target # Target: Flower species (setosa, versicolor, virginica)

# Step 2: Split the dataset 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 and evaluate SVM classifiers with different kernels

# Dictionary to store results

results = {}

# List of SVM kernels to evaluate

kernels = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel in kernels:

# Train the SVM model with the current kernel

svm = SVC(kernel=kernel)

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

# Predict on the test set

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

# Calculate accuracy

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

# Store the result

results[kernel] = accuracy

print(f"Accuracy for {kernel} kernel: {accuracy \* 100:.2f}%")

# Step 4: Predict the flower type for new input data

# Example input data: sepal length, sepal width, petal length, petal width

new\_data = [[5.1, 3.5, 1.4, 0.2]] # Example: Setosa flower

# Predict the flower type using the best performing kernel (you can choose the best one)

best\_kernel = max(results, key=results.get) # Select the kernel with the highest accuracy

best\_svm = SVC(kernel=best\_kernel)

best\_svm.fit(X\_train, y\_train)

predicted\_class = best\_svm.predict(new\_data)

# Output the predicted class

flower\_name = iris.target\_names[predicted\_class][0]

print(f"\nPredicted Flower Type for input data {new\_data}: {flower\_name}")

Slip 29

Q.1. Take iris flower dataset and reduce 4D data to 2D data using PCA. Then train the

model and predict new flower with given measurements.

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Step 1: Load the Iris dataset

iris = load\_iris()

X = iris.data # Features: sepal length, sepal width, petal length, petal width

y = iris.target # Target: Flower species (setosa, versicolor, virginica)

# Step 2: Apply PCA to reduce the data from 4D to 2D

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Step 3: Visualize the 2D PCA data

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

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis', edgecolor='k', s=50)

plt.title('Iris Dataset Reduced to 2D using PCA')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(label='Flower Species')

plt.show()

# Step 4: Split the dataset into training and testing sets

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

# Step 5: Train a classifier (Support Vector Machine)

svm = SVC(kernel='linear') # You can use other classifiers too

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

# Step 6: Predict the flower type on the test set

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

# Step 7: Calculate and print accuracy

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

print(f"Accuracy of the model: {accuracy \* 100:.2f}%")

# Step 8: Predict the flower type for a new flower with given measurements

new\_data = np.array([[5.1, 3.5, 1.4, 0.2]]) # Example: Setosa flower

new\_data\_pca = pca.transform(new\_data) # Apply PCA to the new data

predicted\_class = svm.predict(new\_data\_pca)

# Output the predicted class

flower\_name = iris.target\_names[predicted\_class][0]

print(f"\nPredicted Flower Type for input data {new\_data}: {flower\_name}")

Q.2. Use K-means clustering model and classify the employees into various income groups

or clusters. Preprocess data if require (i.e. drop missing or null values). Use elbow

method and Silhouette Score to find value of k.

To classify employees into various income groups or clusters using K-means clustering, we can follow the steps below. We'll also use the Elbow Method and Silhouette Score to determine the optimal value of k (the number of clusters).

Steps:

1. Load and preprocess the data: This includes handling missing or null values.

2. Use K-means clustering: Apply the K-means algorithm to the data.

3. Use the Elbow Method: Determine the optimal number of clusters (k).

4. Use Silhouette Score: Measure the quality of the clustering for different values of k.

5. Visualize the results: Display the clusters and interpret the data.

Python Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import StandardScaler

# Step 1: Load the employee data

# Replace 'employee\_data.csv' with your actual file path

df = pd.read\_csv('employee\_data.csv')

# Step 2: Preprocess the data

# Drop rows with missing values

df = df.dropna()

# Assume 'Income' is one of the columns you want to use for clustering.

# If the dataset has other features you want to use, select them accordingly.

# For example, selecting only relevant columns like 'Income', 'Age', 'YearsAtCompany', etc.

X = df[['Income', 'Age', 'YearsAtCompany']] # Modify columns based on your dataset

# Step 3: Standardize the features (important for K-means)

scaler = StandardScaler()

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

# Step 4: Use the Elbow Method to find the optimal value of k

inertia = [] # To store the sum of squared distances

k\_range = range(1, 11) # We will try values of k from 1 to 10

for k in k\_range:

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X\_scaled)

inertia.append(kmeans.inertia\_)

# Plot the Elbow Curve

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

plt.plot(k\_range, inertia, marker='o', linestyle='-', color='b')

plt.title('Elbow Method for Optimal k')

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

plt.ylabel('Inertia')

plt.show()

# Step 5: Use Silhouette Score to evaluate clustering quality for different k

sil\_scores = []

for k in k\_range[1:]: # Start from k=2 because silhouette score is undefined for k=1

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(X\_scaled)

score = silhouette\_score(X\_scaled, kmeans.labels\_)

sil\_scores.append(score)

# Plot Silhouette Scores

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

plt.plot(k\_range[1:], sil\_scores, marker='o', linestyle='-', color='g')

plt.title('Silhouette Score for Different Values of k')

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

plt.ylabel('Silhouette Score')

plt.show()

# Step 6: Fit KMeans with optimal k (let's say it is 3 based on the Elbow and Silhouette method)

optimal\_k = 3 # Update based on your analysis

kmeans = KMeans(n\_clusters=optimal\_k, random\_state=42)

kmeans.fit(X\_scaled)

# Add the cluster labels to the original dataset

df['Cluster'] = kmeans.labels\_

# Step 7: Visualize the clusters (2D)

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

plt.scatter(df['Income'], df['Age'], c=df['Cluster'], cmap='viridis')

plt.title(f'Employee Clusters (k={optimal\_k})')

plt.xlabel('Income')

plt.ylabel('Age')

plt.colorbar(label='Cluster')

plt.show()

# Step 8: Display the cluster centers

cluster\_centers = scaler.inverse\_transform(kmeans.cluster\_centers\_)

print("\nCluster Centers (in original scale):")

print(cluster\_centers)

# Step 9: Show some example employees in each cluster

for cluster\_num in range(optimal\_k):

print(f"\nCluster {cluster\_num} Employees:")

print(df[df['Cluster'] == cluster\_num].head())

Explanation of the Code:

1. Loading the Data:

The dataset is loaded using pd.read\_csv(). Replace 'employee\_data.csv' with the actual path to your dataset.

We assume that the dataset contains relevant columns like Income, Age, and YearsAtCompany (or others as per your dataset).

2. Preprocessing:

We drop any rows with missing values using dropna().

We then select only the relevant columns for clustering, in this case, Income, Age, and YearsAtCompany.

The data is scaled using StandardScaler to standardize the features before applying K-means.

3. Elbow Method:

We fit the K-means model for different values of k (from 1 to 10) and calculate the inertia (sum of squared distances from points to their cluster center) for each k.

The Elbow Curve is plotted, where the optimal k is typically where the curve shows an "elbow" — a point where the inertia starts to decrease more slowly.

4. Silhouette Score:

The Silhouette Score is computed for each k starting from 2, as it’s undefined for k=1. The Silhouette Score measures how well each point fits within its cluster, with higher values indicating better clustering.

5. K-means Clustering:

After selecting the optimal k (based on the Elbow Method and Silhouette Score), we fit the K-means model with that k and add the resulting cluster labels to the dataset.

6. Visualization:

We visualize the employee clusters using a scatter plot with Income and Age. The clusters are color-coded.

The cluster centers