**SURYADATTA COLLEGE OF MANAGEMENT**

**INFORMATION RESEARCH & TECHNOLOGY**

**BAVDHAN, PUNE – 411021**

**CS-605-MJP :Lab Course on CS-602-MJ**

**(Machine Learning )**

**Submitted by**

* **NAME OF STUDENTS ROLL NO**
* **SIDDHI JAYSING SAWANT 2023100306**

**Under the Guidance of**

**Miss. Mrunal Gaikwad**

**SUBMITTED IN PARTIAL FULLFILLMENT OF MASTER OF SCIENCE (COMPUTER SCIENCE)**

**SEM-III**

**SAVITRIBAI PHULE PUNE UNIVERSITY**

**For Academic Year 2023-2024**

****

**CERTIFICATE**

**This is to certify that Mr./Ms.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­\_\_**

**student of MSC(CS) Semester\_\_\_\_\_\_\_\_\_ having Seat No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_at Suryadatta College of Management Information Research & Technology (SCMIRT), Pune, has successfully completed the assigned practical in \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ prescribed by the Savitribai Phule Pune University during the academic year\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.**

**Internal Examiner External Examiner**

**Principal**

**Place: Pune**

**Date:**

Slip-1

Q1) Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

# Install the mlxtend library

!pip install mlxtend

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

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

from mlxtend.preprocessing import TransactionEncoder

# Example groceries dataset (replace this with your actual dataset if needed)

groceries = [['milk', 'bread', 'eggs'],

['milk', 'bread'],

['bread', 'butter'],

['milk', 'bread', 'butter'],

['milk', 'eggs'],

['bread', 'butter', 'eggs'],

['milk', 'bread', 'eggs', 'butter']]

# Convert the dataset into a format suitable for Apriori

te = TransactionEncoder()

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

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

# Apply Apriori to find frequent itemsets with a minimum support of 0.25

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

# Display the frequent itemsets

print(frequent\_itemsets)

# Plot frequent itemsets by support

frequent\_itemsets.plot(kind='bar', x='itemsets', y='support', figsize=(10,6), legend=False)

plt.title('Frequent Itemsets by Support')

plt.ylabel('Support')

plt.xlabel('Itemsets')

plt.xticks(rotation=90)

plt.show()

# Generate association rules with minimum confidence of 0.6

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

# Display the association rules

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

# Scatter plot for Association Rules (Lift vs Confidence)

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

plt.scatter(rules['confidence'], rules['lift'], alpha=0.7, edgecolors='r')

plt.title('Association Rules (Lift vs Confidence)')

plt.xlabel('Confidence')

plt.ylabel('Lift')

plt.grid(True)

plt.show()

Q2) 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

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset

iris = load\_iris()

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

iris\_df['species'] = iris.target

# Convert categorical values (species) to numeric format using LabelEncoder

le = LabelEncoder()

iris\_df['species\_numeric'] = le.fit\_transform(iris\_df['species'])

# Display first few rows to check dataset structure

print(iris\_df.head())

# Create scatter plot for two features (e.g., sepal length and sepal width)

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

sns.scatterplot(x=iris\_df['sepal length (cm)'],

y=iris\_df['sepal width (cm)'],

hue=iris\_df['species\_numeric'],

palette='viridis',

s=100)

# Add plot title and labels

plt.title('Scatter Plot of Iris Dataset (Sepal Length vs Sepal Width)')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.legend(title='Species')

plt.show()

Slip-2

Q1) Write 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, r2\_score

import matplotlib.pyplot as plt

# Load the uploaded dataset

data = pd.read\_csv('/content/train.csv')

# Display the first few rows to understand the structure

print("First few rows of the dataset:")

print(data.head())

# Check for null values and handle them (e.g., imputation or removing specific columns)

print("\nChecking for null values...")

# Instead of dropping all rows with nulls, consider imputation or removing specific columns

# For example, to impute missing values with the mean:

# data\_cleaned = data.fillna(data.mean())

# Alternatively, to remove columns with more than a certain percentage of nulls:

# threshold = 0.5 # Example threshold: 50%

# data\_cleaned = data.dropna(axis=1, thresh=int(threshold \* len(data)))

# Or, to drop rows with nulls in specific columns:

data\_cleaned = data.dropna(subset=['GrLivArea', 'SalePrice']) # Only drop rows where 'GrLivArea' or 'SalePrice' are null

# Selecting a single feature ('GrLivArea') and target ('SalePrice') for simple linear regression

X = data\_cleaned[['GrLivArea']] # Predictor: living area size

y = data\_cleaned['SalePrice'] # Target: house price

# Splitting the data

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

# Creating and training the model

model = LinearRegression()

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

# Predictions

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

# Evaluating the model

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

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

print(f"\nMean Squared Error: {mse}")

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

# Visualizing the results

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

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

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

plt.xlabel('Living Area (GrLivArea)')

plt.ylabel('House Price (SalePrice)')

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

plt.legend()

plt.show()

Q2) 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. <https://archive.ics.uci.edu/dataset/292/wholesale+customers>

# Importing required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from sklearn.preprocessing import StandardScaler

import seaborn as sns

# Load the Wholesale Customers dataset

url = '/content/Wholesale customers data.csv'

data = pd.read\_csv(url)

# Display the first few rows of the dataset

print(data.head())

# Check for missing values

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

# Dropping non-numerical columns (if required)

data = data.drop(['Channel', 'Region'], axis=1)

# Standardizing the data

scaler = StandardScaler()

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

# Convert the scaled data back to a DataFrame

scaled\_df = pd.DataFrame(scaled\_data, columns=data.columns)

# Applying Agglomerative Clustering

# The 'affinity' argument is not needed when using 'ward' linkage.

# It defaults to 'euclidean' for 'ward', which is what you intended.

agglo = AgglomerativeClustering(n\_clusters=4, linkage='ward')

clusters = agglo.fit\_predict(scaled\_df)

# Add cluster labels to the original data

data['Cluster'] = clusters

# Visualizing the clusters using a pairplot

plt.figure(figsize=(12, 8))

sns.scatterplot(x='Fresh', y='Milk', hue='Cluster', data=data, palette='Set1', s=100)

plt.title('Agglomerative Clustering of Wholesale Customers')

plt.xlabel('Annual Spending on Fresh Products')

plt.ylabel('Annual Spending on Milk Products')

plt.legend(title='Cluster')

plt.grid()

plt.show()

Slip – 3

Q1) 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

import numpy as np

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

# Step 1: Generate synthetic dataset

np.random.seed(0)

# Generating random data

size = np.random.randint(500, 5000, 100) # Size in sq ft

bedrooms = np.random.randint(1, 6, 100) # 1 to 5 bedrooms

bathrooms = np.random.randint(1, 4, 100) # 1 to 3 bathrooms

age = np.random.randint(0, 30, 100) # Age of the house

distance = np.random.uniform(1, 20, 100) # Distance to city center in miles

# Creating a DataFrame

data = pd.DataFrame({

'Size': size,

'Bedrooms': bedrooms,

'Bathrooms': bathrooms,

'Age': age,

'Distance': distance

})

# Define price based on some arbitrary linear function with noise

data['Price'] = (data['Size'] \* 150 +

data['Bedrooms'] \* 10000 +

data['Bathrooms'] \* 5000 -

data['Age'] \* 2000 -

data['Distance'] \* 3000 +

np.random.normal(0, 20000, 100)) # Adding noise

# Display the first few rows of the dataset

print(data.head())

# Step 2: Split the dataset into training and testing sets

X = data[['Size', 'Bedrooms', 'Bathrooms', 'Age', 'Distance']]

y = data['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Step 3: Create the multiple 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)

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

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

print(f'R^2 Score: {r2}')

# Optional: Plotting predicted vs actual prices

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

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

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

plt.show()

Q2) 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

import numpy as np

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

from sklearn.linear\_model import LogisticRegression

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

import matplotlib.pyplot as plt

import seaborn as sns

# Step 1: Generate synthetic dataset

np.random.seed(0)

# Create a synthetic dataset

n\_samples = 1000

age = np.random.randint(1, 100, n\_samples) # Age of passengers

speed = np.random.randint(0, 100, n\_samples) # Speed of vehicle at impact

# Define survivability based on some arbitrary rules

# - Younger passengers and lower speeds tend to survive more

survivability = (age < 50) & (speed < 40)

survivability = survivability.astype(int) # Convert boolean to int

# Creating DataFrame

data = pd.DataFrame({

'Age': age,

'Speed': speed,

'Survivability': survivability

})

# Display the first few rows of the dataset

print(data.head())

# Step 2: Preprocess the data

# Check for missing values (just in case)

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

# Features and target variable

X = data[['Age', 'Speed']]

y = data['Survivability']

# Step 3: 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=0)

# Step 4: Create and train the logistic regression model

model = LogisticRegression()

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

# Step 5: Make predictions

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

# Step 6: Evaluate the model

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

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

class\_report = classification\_report(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

print('Confusion Matrix:\n', conf\_matrix)

print('Classification Report:\n', class\_report)

# Optional: Plotting confusion matrix

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Survived', 'Survived'], yticklabels=['Not Survived', 'Survived'])

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.title('Confusion Matrix')

plt.show()

Slip-4

Q1) write a python program to implement k-mean 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 dataset (after uploading it to Colab or your local environment)

data = pd.read\_csv('Mall\_Customers.csv')

# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())

# Select features for clustering (e.g., 'Annual Income (k$)' and 'Spending Score (1-100)')

X = data[['Annual Income (k$)', 'Spending Score (1-100)']]

# Standardize the features for better clustering

scaler = StandardScaler()

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

# Determine the optimal number of clusters using the Elbow method

wcss = []

for i in range(1, 11):

    kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state=42)

    kmeans.fit(X\_scaled)

    wcss.append(kmeans.inertia\_)

# Plot the Elbow method result

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

plt.plot(range(1, 11), wcss, marker='o', linestyle='-', color='b')

plt.xlabel('Number of Clusters')

plt.ylabel('WCSS (Within-Cluster Sum of Squares)')

plt.title('Elbow Method to Determine Optimal Number of Clusters')

plt.show()

# Choose the optimal number of clusters (e.g., 5 based on Elbow plot)

optimal\_clusters = 5

kmeans = KMeans(n\_clusters=optimal\_clusters, init='k-means++', random\_state=42)

y\_kmeans = kmeans.fit\_predict(X\_scaled)

# Add the cluster labels to the original dataset for analysis

data['Cluster'] = y\_kmeans

# Visualize the clusters

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

for i in range(optimal\_clusters):

    plt.scatter(X\_scaled[y\_kmeans == i, 0], X\_scaled[y\_kmeans == i, 1],

                s=100, label=f'Cluster {i+1}')

# Plot the centroids

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1],

            s=300, c='red', marker='X', label='Centroids')

plt.xlabel('Annual Income (k$) [scaled]')

plt.ylabel('Spending Score (1-100) [scaled]')

plt.title('Customer Segmentation using K-Means Clustering')

plt.legend()

plt.show()

Q2) write 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

# Load the dataset (update the path if needed)

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

# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())

# Select feature and target columns ('GrLivArea' and 'SalePrice')

X = data[['GrLivArea']] # Living area as predictor

y = data['SalePrice'] # House price as target

# Split the data into training and testing sets

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

# Create and train the Linear Regression model

model = LinearRegression()

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

# Make predictions on the test set

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

# Evaluate the model using Mean Squared Error and R-squared

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

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

print(f"\nMean Squared Error: {mse}")

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

# Display the model's coefficient and intercept

print(f"\nCoefficient (Slope): {model.coef\_[0]}")

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

# Plotting the results

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

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

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

plt.xlabel('Above Ground Living Area (GrLivArea)')

plt.ylabel('House Price (SalePrice)')

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

plt.legend()

plt.show()

Slip-5

Q1) write a python program to implement multiple linear regression for fuel consumption dataset.

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

# Load the dataset (update the path if needed)

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

# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())

# Select features and target columns

# In this example, we use 'ENGINESIZE', 'CYLINDERS', and 'FUELCONSUMPTION\_COMB' as predictors

X = data[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION\_COMB']]

y = data['CO2EMISSIONS'] # Target variable: CO2 emissions

# 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 and train the Multiple Linear Regression model

model = LinearRegression()

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

# Make predictions on the test set

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"\nMean Squared Error: {mse}")

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

# Display the model's coefficients and intercept

print("\nCoefficients (Slopes):", model.coef\_)

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

# Optional: Visualize the actual vs. predicted CO2 emissions

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

plt.scatter(y\_test, y\_pred, color='blue', edgecolor='k', alpha=0.6)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], 'r--', lw=3)

plt.xlabel('Actual CO2 Emissions')

plt.ylabel('Predicted CO2 Emissions')

plt.title('Multiple Linear Regression - CO2 Emissions Prediction')

plt.show()

Q2) Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use Iris Dataset).

import pandas as pd

from sklearn import datasets

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load the Iris dataset from sklearn

iris = datasets.load\_iris()

# Create a DataFrame from the Iris dataset

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

# Add the target labels to the DataFrame

data['species'] = iris.target

# Display the first few rows of the dataset

print("First few rows of the Iris dataset:")

print(data.head())

# Features (X) and Target labels (y)

X = data[iris.feature\_names] # Features: sepal length, sepal width, petal length, petal width

y = data['species'] # Target: species

# 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 a KNN classifier model with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the model using the training data

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

# Make predictions using the test data

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

# Evaluate the model's accuracy

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

print(f"\nAccuracy of the K-NN model: {accuracy \* 100:.2f}%")

# Optional: Visualize the results (only 2 features for simplicity)

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

plt.scatter(X\_test.iloc[:, 0], X\_test.iloc[:, 1], c=y\_pred, cmap='viridis', label="Predicted Species")

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.title('K-NN Prediction on Iris Dataset (Sepal Length vs. Sepal Width)')

plt.colorbar(label='Species')

plt.show()

Slip-6

Q1) write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

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

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

# Instead of using load\_boston, fetch the data directly or use an alternative

# Fetching directly (as suggested in the error message):

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

# Create a DataFrame

data = pd.DataFrame(data, columns=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'])

data['PRICE'] = target

# ... (rest of your code remains the same, using 'data' DataFrame) ...

# Save the dataset to a CSV file (optional)

data.to\_csv('HousingData.csv', index=False)

print("\nDataset saved to HousingData.csv")

# Display the first few rows of the dataset

print("\nFirst few rows of the dataset:")

print(data.head())

# Select the feature(s) and target variable

X = data[['RM']] # For simplicity, using 'RM' (average number of rooms per dwelling)

y = data['PRICE'] # Target: House prices

# 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)

# Polynomial feature transformation (degree 2 for example)

poly = PolynomialFeatures(degree=2)

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

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

# Train the Polynomial Linear Regression model

model = LinearRegression()

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

# Make predictions on the test data

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

# Evaluate the model

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

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

print(f"\nMean Squared Error: {mse}")

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

# Plot the results for the training and test set (Visualizing the Polynomial fit)

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

# Plot actual data points and regression curve

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

plt.plot(np.sort(X\_test.values), model.predict(poly.transform(np.sort(X\_test.values).reshape(-1, 1))),

color='red', label='Polynomial Regression Line')

plt.xlabel('Average Number of Rooms (RM)')

plt.ylabel('House Price')

plt.title('Polynomial Linear Regression - Boston Housing Prices')

plt.legend()

plt.show()

Q2) 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.

Sol:

# Importing required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

# Creating a sample employee dataset

data = {

    'EmployeeID': range(1, 21),

    'Age': np.random.randint(22, 60, size=20),

    'Income': np.random.randint(30000, 120000, size=20),

    'YearsAtCompany': np.random.randint(1, 30, size=20)

}

df = pd.DataFrame(data)

# Display the first few rows of the dataset

print(df.head())

# Check for missing values

print(df.isnull().sum())

# Drop any rows with missing values

df = df.dropna()

# Selecting features for clustering

features = df[['Age', 'Income', 'YearsAtCompany']]

# Finding the optimal number of clusters using the Elbow Method

wcss = []

for i in range(1, 11):

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

    kmeans.fit(features)

    wcss.append(kmeans.inertia\_)  # Within-cluster sum of squares

# Plotting the elbow curve

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

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

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

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

plt.ylabel('WCSS')

plt.grid()

plt.show()

# Calculate silhouette scores for each k

silhouette\_scores = []

for i in range(2, 11):

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

    kmeans.fit(features)

    score = silhouette\_score(features, kmeans.labels\_)

    silhouette\_scores.append(score)

# Plotting the Silhouette Scores

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

plt.plot(range(2, 11), silhouette\_scores, marker='o')

plt.title('Silhouette Scores for Different k Values')

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

plt.ylabel('Silhouette Score')

plt.grid()

plt.show()

# Fit the K-means model with the chosen number of clusters

optimal\_k = 3  # Replace this with the optimal k found from the plots

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

df['Cluster'] = kmeans.fit\_predict(features)

# Display the cluster assignment

print(df[['EmployeeID', 'Income', 'Cluster']])

Slip-7

Q1) 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.

Sol:

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

# Step 1: Load the dataset

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

# Step 2: Explore the data

print(data.head())  # Print the first few rows of the dataset

print(data.describe())  # Get summary statistics

# Extracting features and target variable

X = data.iloc[:, 1:2].values  # Position levels (assuming they are in the second column)

y = data.iloc[:, 2].values     # Salaries (assuming they are in the third column)

# Step 3: Fit the Simple Linear Regression Model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

# Step 4: Fit the Polynomial Linear Regression Model

poly\_features = PolynomialFeatures(degree=4)  # You can adjust the degree based on your data

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

polynomial\_model = LinearRegression()

polynomial\_model.fit(X\_poly, y)

# Step 5: Evaluate the models

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

y\_pred\_poly = polynomial\_model.predict(X\_poly)

# Calculate Mean Squared Error and R^2 Score

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

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

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

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

print(f"Linear Regression - MSE: {mse\_linear}, R^2: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, R^2: {r2\_poly}")

# Step 6: Plot the results

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

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

plt.scatter(X, y\_pred\_poly, color='green', label='Polynomial Regression')

plt.title('Salary vs Position Level')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 7: Predict Salaries for Level 11 and Level 12

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

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

salary\_level\_11\_linear = linear\_model.predict(level\_11)

salary\_level\_11\_poly = polynomial\_model.predict(poly\_features.transform(level\_11))

salary\_level\_12\_linear = linear\_model.predict(level\_12)

salary\_level\_12\_poly = polynomial\_model.predict(poly\_features.transform(level\_12))

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

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

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

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

Q2) Write a python program to implement Naïve 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

from sklearn.preprocessing import LabelEncoder

# Sample Weather Forecast Dataset (can be saved as CSV or used as is)

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', 'Hot', 'Mild'],

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

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

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

}

# Convert the data to a pandas DataFrame

df = pd.DataFrame(data)

# Display the first few rows

print("Dataset:")

print(df.head())

# Encode categorical variables using LabelEncoder

le = LabelEncoder()

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

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

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

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

df['PlayTennis'] = le.fit\_transform(df['PlayTennis']) # Target variable

# Features (X) and Target (y)

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

y = df['PlayTennis']

# 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 and train the Naïve Bayes model

nb = GaussianNB()

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

# Make predictions on the test data

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

# Evaluate the model

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

print(f"\nAccuracy of the Naïve Bayes model: {accuracy \* 100:.2f}%")

# Display the predictions

print("\nPredictions on Test Data:")

print(y\_pred)

Slip-8

Q1) 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.

Sol:

# Import necessary libraries

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 import metrics

# Step 1: Load the dataset

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

# Step 2: Preprocess the text data

X = newsgroups.data  # Features (news articles)

y = newsgroups.target  # Target labels (categories)

# Step 3: Convert text to numerical vectors using CountVectorizer

vectorizer = CountVectorizer(stop\_words='english')

X\_vectorized = vectorizer.fit\_transform(X)

# Step 4: Split the data into training and testing sets

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

# Step 5: Train the Multinomial Naïve Bayes model

model = MultinomialNB()

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

# Step 6: Evaluate the model

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

# Print accuracy and classification report

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

print(f'Accuracy: {accuracy:.4f}')

print('Classification Report:')

print(metrics.classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names))

# Step 7: Make predictions for new articles

def predict\_category(text):

    text\_vectorized = vectorizer.transform([text])  # Vectorize the new text

    predicted\_category = model.predict(text\_vectorized)  # Predict the category

    return newsgroups.target\_names[predicted\_category[0]]  # Return the category name

# Example usage of the prediction function

new\_article = """NASA's Mars rover Curiosity has discovered evidence that liquid water once flowed on the surface of the Red Planet."""

predicted\_category = predict\_category(new\_article)

print(f'Predicted Category: {predicted\_category}')

Q2) Write python program to implement Decision Tree whether or not to play Tennis.

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

# Sample Weather Forecast Dataset (can be saved as CSV or used as is)

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', 'Hot', 'Mild'],

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

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

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

}

# Convert the data to a pandas DataFrame

df = pd.DataFrame(data)

# Display the first few rows

print("Dataset:")

print(df.head())

# Encode categorical variables using LabelEncoder

le = LabelEncoder()

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

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

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

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

df['PlayTennis'] = le.fit\_transform(df['PlayTennis']) # Target variable

# Features (X) and Target (y)

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

y = df['PlayTennis']

# 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 and train the Decision Tree model

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

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

# Make predictions on the test data

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

# Evaluate the model

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

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

# Visualize the Decision Tree

plt.figure(figsize=(12, 8))

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

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

plt.show()

# Display predictions on test data

print("\nPredictions on Test Data:")

print(y\_pred)

Slip-9

Q1) Implement Ridge Regression, Lasso regression, ElasticNet 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.

Sol: 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 Ridge, Lasso, ElasticNet

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

# Step 1: Load the dataset

# Make sure to adjust the path to where your boston\_houses.csv is located

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

# Step 2: Extract relevant features

data = df[['RM', 'Price']]  # Select RM and Price columns

# Display the first few rows of the dataset

print(data.head())

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

X = data[['RM']]

y = data['Price']

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

# Step 4: Fit Ridge Regression

ridge\_model = Ridge(alpha=1.0)  # You can adjust alpha

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

# Step 5: Fit Lasso Regression

lasso\_model = Lasso(alpha=1.0)  # You can adjust alpha

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

# Step 6: Fit ElasticNet Regression

elastic\_net\_model = ElasticNet(alpha=1.0, l1\_ratio=0.5)  # You can adjust alpha and l1\_ratio

elastic\_net\_model.fit(X\_train, y\_train)

# Step 7: Make predictions for test data

y\_pred\_ridge = ridge\_model.predict(X\_test)

y\_pred\_lasso = lasso\_model.predict(X\_test)

y\_pred\_elastic\_net = elastic\_net\_model.predict(X\_test)

# Step 8: Evaluate Models

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

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

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

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

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

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

print(f'Ridge Regression MSE: {mse\_ridge}, R²: {r2\_ridge}')

print(f'Lasso Regression MSE: {mse\_lasso}, R²: {r2\_lasso}')

print(f'ElasticNet Regression MSE: {mse\_elastic\_net}, R²: {r2\_elastic\_net}')

# Step 9: Predict the price for a house with 5 rooms

rooms = np.array([[5]])  # 5 rooms

pred\_price\_ridge = ridge\_model.predict(rooms)

pred\_price\_lasso = lasso\_model.predict(rooms)

pred\_price\_elastic\_net = elastic\_net\_model.predict(rooms)

print(f'Predicted Price for a house with 5 rooms (Ridge): ${pred\_price\_ridge[0]:,.2f}')

print(f'Predicted Price for a house with 5 rooms (Lasso): ${pred\_price\_lasso[0]:,.2f}')

print(f'Predicted Price for a house with 5 rooms (ElasticNet): ${pred\_price\_elastic\_net[0]:,.2f}')

# Optional: Plotting the results

plt.scatter(data['RM'], data['Price'], color='blue', label='Data')

plt.scatter(X\_test, y\_test, color='black', label='Test Data', alpha=0.5)

plt.plot(X\_test, y\_pred\_ridge, color='red', label='Ridge Prediction')

plt.plot(X\_test, y\_pred\_lasso, color='green', label='Lasso Prediction')

plt.plot(X\_test, y\_pred\_elastic\_net, color='purple', label='ElasticNet Prediction')

plt.xlabel('Number of Rooms (RM)')

plt.ylabel('Price')

plt.title('House Price Predictions')

plt.legend()

plt.show()

Q2) 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, classification\_report

from sklearn.preprocessing import LabelEncoder

# Load the UniversalBank dataset

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

# Display the first few rows of the dataset

print("Dataset:")

print(df.head())

# Check for missing values

print("\nMissing values:")

print(df.isnull().sum())

# Preprocessing: Drop the 'ID' and 'ZIP Code' columns as they are not useful for prediction

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

# Encode categorical variables if any (e.g., 'Personal Loan', 'Education', 'Securities Account', etc.)

label\_encoder = LabelEncoder()

df['Personal Loan'] = label\_encoder.fit\_transform(df['Personal Loan']) # Binary encoding

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

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

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

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

# Features (X) and Target (y)

X = df.drop(columns=['Personal Loan']) # Drop target variable

y = df['Personal Loan'] # Target variable: whether the customer accepted a personal loan or not

# 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)

# Standardize the features using StandardScaler (important for SVM)

scaler = StandardScaler()

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

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

# Create and train the Linear SVM model

svm = SVC(kernel='linear', random\_state=42)

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

# Make predictions on the test data

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

# Evaluate the model

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

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

# Display detailed classification report

print("\nClassification Report:")

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

Slip-10

Q1) Write 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

import matplotlib.pyplot as plt

import seaborn as sns

# Load the Iris dataset from sklearn

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels (target)

# Convert to pandas DataFrame for better visualization

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

# Standardize the data (important for PCA)

scaler = StandardScaler()

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

# Apply PCA to reduce the dimensionality to 2 components

pca = PCA(n\_components=2)

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

# Create a DataFrame with the transformed data

df\_pca = pd.DataFrame(X\_pca, columns=['PC1', 'PC2'])

# Visualize the PCA results

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

sns.scatterplot(x='PC1', y='PC2', hue=y, palette='viridis', data=df\_pca, s=100, marker='o')

# Add labels and title

plt.title('PCA of Iris Dataset', fontsize=16)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(title='Species', labels=iris.target\_names)

# Show the plot

plt.show()

# Explained Variance

print(f'Explained Variance Ratio by each principal component: {pca.explained\_variance\_ratio\_}')

print(f'Total explained variance: {sum(pca.explained\_variance\_ratio\_)}')

Q2) 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

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset

iris = load\_iris()

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

iris\_df['species'] = iris.target

# Convert categorical values (species) to numeric format using LabelEncoder

le = LabelEncoder()

iris\_df['species\_numeric'] = le.fit\_transform(iris\_df['species'])

# Display first few rows to check dataset structure

print(iris\_df.head())

# Create scatter plot for two features (e.g., sepal length and sepal width)

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

sns.scatterplot(x=iris\_df['sepal length (cm)'],

y=iris\_df['sepal width (cm)'],

hue=iris\_df['species\_numeric'],

palette='viridis',

s=100)

# Add plot title and labels

plt.title('Scatter Plot of Iris Dataset (Sepal Length vs Sepal Width)')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

plt.legend(title='Species')

plt.show()

Slip-11

Q1) write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

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

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

# Instead of using load\_boston, fetch the data directly or use an alternative

# Fetching directly (as suggested in the error message):

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

# Create a DataFrame

data = pd.DataFrame(data, columns=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'])

data['PRICE'] = target

# ... (rest of your code remains the same, using 'data' DataFrame) ...

# Save the dataset to a CSV file (optional)

data.to\_csv('HousingData.csv', index=False)

print("\nDataset saved to HousingData.csv")

# Display the first few rows of the dataset

print("\nFirst few rows of the dataset:")

print(data.head())

# Select the feature(s) and target variable

X = data[['RM']] # For simplicity, using 'RM' (average number of rooms per dwelling)

y = data['PRICE'] # Target: House prices

# 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)

# Polynomial feature transformation (degree 2 for example)

poly = PolynomialFeatures(degree=2)

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

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

# Train the Polynomial Linear Regression model

model = LinearRegression()

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

# Make predictions on the test data

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

# Evaluate the model

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

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

print(f"\nMean Squared Error: {mse}")

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

# Plot the results for the training and test set (Visualizing the Polynomial fit)

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

# Plot actual data points and regression curve

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

plt.plot(np.sort(X\_test.values), model.predict(poly.transform(np.sort(X\_test.values).reshape(-1, 1))),

color='red', label='Polynomial Regression Line')

plt.xlabel('Average Number of Rooms (RM)')

plt.ylabel('House Price')

plt.title('Polynomial Linear Regression - Boston Housing Prices')

plt.legend()

plt.show()

Q2) Write a python program to Implement Decision Tree classifier model onData 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>)

Sol:

# Step 1: Import necessary libraries

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

import matplotlib.pyplot as plt

from sklearn import tree

# Step 2: Upload dataset to Colab (if using direct upload)

from google.colab import files

uploaded = files.upload()

# Step 3: Load the dataset (assuming the file is named 'banknote\_data.csv')

# If using Google Drive, replace '/content/...' with your file path

#df = pd.read\_csv('/content/banknote\_data.csv')

import io

df = pd.read\_csv(io.BytesIO(uploaded['data\_banknote\_authentication.txt'])) # This line is added to read data from uploaded file.

# Step 4: Explore the dataset (optional)

print(df.head())

# Step 5: Split data into features (X) and label (y)

X = df.drop('label', axis=1)  # Features: variance, skewness, curtosis, entropy

y = df['label']  # Labels: genuine (1), forged (0)

# Step 6: 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 7: Create and train the Decision Tree classifier

clf = DecisionTreeClassifier(random\_state=42)

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

# Step 8: Make predictions on the test set

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

# Step 9: Evaluate the model

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

print("Accuracy:", accuracy)

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

# Step 10: Visualize the Decision Tree (optional)

plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=['forged', 'genuine'], rounded=True)

plt.show()

Slip-12

Q1) Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use Iris Dataset).

import pandas as pd

from sklearn import datasets

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

import matplotlib.pyplot as plt

# Load the Iris dataset from sklearn

iris = datasets.load\_iris()

# Create a DataFrame from the Iris dataset

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

# Add the target labels to the DataFrame

data['species'] = iris.target

# Display the first few rows of the dataset

print("First few rows of the Iris dataset:")

print(data.head())

# Features (X) and Target labels (y)

X = data[iris.feature\_names] # Features: sepal length, sepal width, petal length, petal width

y = data['species'] # Target: species

# 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 a KNN classifier model with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the model using the training data

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

# Make predictions using the test data

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

# Evaluate the model's accuracy

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

print(f"\nAccuracy of the K-NN model: {accuracy \* 100:.2f}%")

# Optional: Visualize the results (only 2 features for simplicity)

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

plt.scatter(X\_test.iloc[:, 0], X\_test.iloc[:, 1], c=y\_pred, cmap='viridis', label="Predicted Species")

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.title('K-NN Prediction on Iris Dataset (Sepal Length vs. Sepal Width)')

plt.colorbar(label='Species')

plt.show()

Q2) 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.

Sol:

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

# Step 1: Load the dataset

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

# Step 2: Explore the data

print(data.head())  # Print the first few rows of the dataset

print(data.describe())  # Get summary statistics

# Extracting features and target variable

X = data.iloc[:, 1:2].values  # Position levels (assuming they are in the second column)

y = data.iloc[:, 2].values     # Salaries (assuming they are in the third column)

# Step 3: Fit the Simple Linear Regression Model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

# Step 4: Fit the Polynomial Linear Regression Model

poly\_features = PolynomialFeatures(degree=4)  # You can adjust the degree based on your data

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

polynomial\_model = LinearRegression()

polynomial\_model.fit(X\_poly, y)

# Step 5: Evaluate the models

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

y\_pred\_poly = polynomial\_model.predict(X\_poly)

# Calculate Mean Squared Error and R^2 Score

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

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

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

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

print(f"Linear Regression - MSE: {mse\_linear}, R^2: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, R^2: {r2\_poly}")

# Step 6: Plot the results

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

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

plt.scatter(X, y\_pred\_poly, color='green', label='Polynomial Regression')

plt.title('Salary vs Position Level')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 7: Predict Salaries for Level 11 and Level 12

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

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

salary\_level\_11\_linear = linear\_model.predict(level\_11)

salary\_level\_11\_poly = polynomial\_model.predict(poly\_features.transform(level\_11))

salary\_level\_12\_linear = linear\_model.predict(level\_12)

salary\_level\_12\_poly = polynomial\_model.predict(poly\_features.transform(level\_12))

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

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

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

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

Slip-13

Q1) Create RNN model and analyze the Google stock price dataset. Find out increasing or decreasing trends of stock price for the next day.

Sol:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.preprocessing import MinMaxScaler

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

from tensorflow.keras.models import Sequential

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

# Install the library if you haven't already

!pip install yfinance

import yfinance as yf

# Download Google stock data

data = yf.download('GOOGL', start='2010-01-01', end='2024-10-19')

# Save to CSV

data.to\_csv('GOOGL.csv')

# Step 1: Load the Google stock price dataset

# For this example, you can download the data from Yahoo Finance and save it as 'GOOGL.csv'

# Alternatively, you can use any other method to get the dataset.

# Load dataset

data = pd.read\_csv('GOOGL.csv')  # Make sure to have a file named 'GOOGL.csv' in the same directory

# Step 2: Preprocess the Data

# Use the 'Close' price for analysis

data = data[['Date', 'Close']]

data['Date'] = pd.to\_datetime(data['Date'])

data.set\_index('Date', inplace=True)

# Visualize the closing prices

plt.figure(figsize=(14, 5))

plt.plot(data['Close'], label='Google Stock Price')

plt.title('Google Stock Price History')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.show()

# Normalize the dataset

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

scaled\_data = scaler.fit\_transform(data['Close'].values.reshape(-1, 1))

# Create training and test sets

train\_size = int(len(scaled\_data) \* 0.8)

train\_data = scaled\_data[:train\_size]

test\_data = scaled\_data[train\_size:]

# Function to create sequences

def create\_dataset(data, time\_step=1):

    X, y = [], []

    for i in range(len(data) - time\_step - 1):

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

        y.append(data[i + time\_step, 0])

    return np.array(X), np.array(y)

# Create sequences

time\_step = 60  # Number of previous days to consider

X\_train, y\_train = create\_dataset(train\_data, time\_step)

X\_test, y\_test = create\_dataset(test\_data, time\_step)

# Reshape input to be [samples, time steps, features]

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

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

# Step 3: Build the RNN Model

model = Sequential()

model.add(LSTM(50, return\_sequences=True, input\_shape=(X\_train.shape[1], 1)))

model.add(Dropout(0.2))

model.add(LSTM(50, return\_sequences=False))

model.add(Dropout(0.2))

model.add(Dense(1))  # Output layer for the prediction

# Compile the model

model.compile(optimizer='adam', loss='mean\_squared\_error')

# Step 4: Train the Model

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

# Step 5: Make Predictions

train\_predictions = model.predict(X\_train)

test\_predictions = model.predict(X\_test)

# Inverse transform predictions to get actual prices

train\_predictions = scaler.inverse\_transform(train\_predictions)

test\_predictions = scaler.inverse\_transform(test\_predictions)

# Analyze trends

last\_test\_price = test\_predictions[-1][0]

next\_day\_price = scaler.inverse\_transform([[last\_test\_price]])

print(f"Predicted price for next day: {next\_day\_price[0][0]}")

# Plot the results

train\_plot = np.empty\_like(scaled\_data)

train\_plot[:, :] = np.nan

train\_plot[time\_step:len(train\_predictions) + time\_step, :] = train\_predictions

test\_plot = np.empty\_like(scaled\_data)

test\_plot[:, :] = np.nan

test\_plot[len(train\_predictions) + (time\_step \* 2) + 1:len(scaled\_data) - 1, :] = test\_predictions

plt.figure(figsize=(14, 5))

plt.plot(scaler.inverse\_transform(scaled\_data), label='Actual Stock Price', color='blue')

plt.plot(train\_plot, label='Train Predictions', color='orange')

plt.plot(test\_plot, label='Test Predictions', color='red')

plt.title('Google Stock Price Predictions')

plt.xlabel('Date')

plt.ylabel('Price (USD)')

plt.legend()

plt.show()

Q2) write 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

# Load the dataset (update the path if needed)

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

# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())

# Select feature and target columns ('GrLivArea' and 'SalePrice')

X = data[['GrLivArea']] # Living area as predictor

y = data['SalePrice'] # House price as target

# Split the data into training and testing sets

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

# Create and train the Linear Regression model

model = LinearRegression()

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

# Make predictions on the test set

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

# Evaluate the model using Mean Squared Error and R-squared

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

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

print(f"\nMean Squared Error: {mse}")

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

# Display the model's coefficient and intercept

print(f"\nCoefficient (Slope): {model.coef\_[0]}")

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

# Plotting the results

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

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

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

plt.xlabel('Above Ground Living Area (GrLivArea)')

plt.ylabel('House Price (SalePrice)')

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

plt.legend()

plt.show()

Slip-14

Q1) 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

Sol:

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

# Step 1: Import and Load the MNIST dataset

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

# Step 2: Preprocess the Data

# Reshape the data to add a channel dimension (28, 28, 1) for CNN

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 = models.Sequential()

# Convolutional Layer 1

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

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

# Convolutional Layer 2

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

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

# Flatten the output and add fully connected layers

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(10, activation='softmax'))  # Output layer for 10 classes (digits 0-9)

# Compile the model

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

# Step 4: Train the Model

history = model.fit(X\_train, y\_train, epochs=5, batch\_size=64, validation\_split=0.2)

# Step 5: Evaluate the Model

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {test\_loss}, Test Accuracy: {test\_accuracy}")

# Step 6: Predict on a Given Image

# Example: Predict the first image in the test set

image\_index = 0

test\_image = X\_test[image\_index].reshape(1, 28, 28, 1)  # Reshape for prediction

predicted\_class = model.predict(test\_image)

predicted\_digit = np.argmax(predicted\_class)

print(f"Predicted digit: {predicted\_digit}")

# Optional: Display the test image

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

plt.title(f"Predicted digit: {predicted\_digit}")

plt.axis('off')

plt.show()

Q2) 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

# Create a custom dataset

data = {

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

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

'City': ['New York', 'Los Angeles', 'Chicago', np.nan, 'Houston', 'Phoenix'],

'Salary': [50000, 60000, np.nan, 70000, 80000, 90000]

}

# Create a DataFrame from the dataset

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Check for null values in the dataset

null\_values = df.isnull().sum()

print("\nNull Values in Each Column:")

print(null\_values)

# Remove rows with any null values

df\_cleaned = df.dropna()

# Display the cleaned dataset

print("\nCleaned Dataset (Null values removed):")

print(df\_cleaned)

Slip-15

Q1) Create an ANN and train it on house price dataset classify the house price is above average or below average.

Sol:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

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

from sklearn.datasets import fetch\_california\_housing

from sklearn.preprocessing import StandardScaler

import numpy as np

# Step 1: Load the California Housing Dataset

data = fetch\_california\_housing()

X = data.data

y = data.target

# Step 2: Convert it into a classification task (Above or Below Average Price)

average\_price = np.mean(y)

y\_class = np.where(y > average\_price, 1, 0) # 1 if above average, 0 if below average

# Step 3: Train-Test Split

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

# Step 4: Normalize the features (scale the input data)

scaler = StandardScaler()

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

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

# Step 5: Create the ANN Model

model = Sequential()

# Input Layer and Hidden Layer (Dense Layer with ReLU activation)

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

# Output Layer (Binary Classification - Above/Below Average)

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

# Compile the Model

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

# Step 6: Train the Model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2)

# Step 7: Evaluate the Model on Test Data

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

# Predictions (if needed)

predictions = (model.predict(X\_test) > 0.5).astype("int32")

Q2) Write a python Program to implement multiple linear Regression for house price data set.

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: Create a sample dataset

data = {

'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 4500, 5000, 5500, 6000],

'Num Rooms': [3, 4, 3, 5, 4, 5, 6, 6, 7, 8],

'Age (years)': [10, 15, 20, 5, 30, 12, 25, 8, 18, 14],

'Price': [400000, 500000, 600000, 650000, 700000, 800000, 850000, 900000, 950000, 1000000]

}

# Step 2: Convert the dataset into a DataFrame

df = pd.DataFrame(data)

# Step 3: Split the data into features (X) and target (y)

X = df[['Size (sqft)', 'Num Rooms', 'Age (years)']] # Features

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

# Step 4: 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)

# Step 5: Standardize the features (important for regression models)

scaler = StandardScaler()

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

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

# Step 6: Create and train the Multiple Linear Regression model

model = LinearRegression()

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

# Step 7: Make predictions on the test data

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

# Step 8: Evaluate the model

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

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

# Display the results

print(f"Predicted Prices: {y\_pred}")

print(f"Actual Prices: {y\_test.values}")

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

print(f"R² Score: {r2}")

Slip-16

Q1) Create a two layered neural network with relu and sigmoid activation function.

Sol:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the neural network model

model = Sequential()

# Input Layer and Hidden Layer (Dense Layer with ReLU activation)

# Changed input\_dim to 20 to match the actual input data shape

model.add(Dense(units=64, activation='relu', input\_dim=20))

# Output Layer (Dense Layer with Sigmoid activation for binary classification)

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

# Compile the model

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

# Summary of the model

model.summary()

# Fit the model to your dataset

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

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

Q2) Write a python program to implement Simple linear Regression for Boston Houseing Dataset.

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

# Load the dataset (update the path if needed)

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

# Display the first few rows of the dataset to understand its structure

print("First few rows of the dataset:")

print(data.head())

# Select feature and target columns ('GrLivArea' and 'SalePrice')

X = data[['GrLivArea']] # Living area as predictor

y = data['SalePrice'] # House price as target

# Split the data into training and testing sets

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

# Create and train the Linear Regression model

model = LinearRegression()

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

# Make predictions on the test set

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

# Evaluate the model using Mean Squared Error and R-squared

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

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

print(f"\nMean Squared Error: {mse}")

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

# Display the model's coefficient and intercept

print(f"\nCoefficient (Slope): {model.coef\_[0]}")

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

# Plotting the results

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

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

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

plt.xlabel('Above Ground Living Area (GrLivArea)')

plt.ylabel('House Price (SalePrice)')

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

plt.legend()

plt.show()

Slip-17

Q1) 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, GradientBoostingClassifier, VotingClassifier, StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.preprocessing import StandardScaler

# Load dataset

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

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

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

# Split data into features and target

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

y = data['Outcome']

# Train-test split

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

# Standardize the data

scaler = StandardScaler()

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

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

# Bagging with Random Forest

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

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

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

# Random Forest performance

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

rf\_precision = precision\_score(y\_test, rf\_predictions)

rf\_recall = recall\_score(y\_test, rf\_predictions)

rf\_f1 = f1\_score(y\_test, rf\_predictions)

# AdaBoost

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

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

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

# AdaBoost performance

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

ada\_precision = precision\_score(y\_test, ada\_predictions)

ada\_recall = recall\_score(y\_test, ada\_predictions)

ada\_f1 = f1\_score(y\_test, ada\_predictions)

# Gradient Boosting

gb\_model = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

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

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

# Gradient Boosting performance

gb\_accuracy = accuracy\_score(y\_test, gb\_predictions)

gb\_precision = precision\_score(y\_test, gb\_predictions)

gb\_recall = recall\_score(y\_test, gb\_predictions)

gb\_f1 = f1\_score(y\_test, gb\_predictions)

# Voting Classifier

lr\_model = LogisticRegression(random\_state=42)

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

knn\_model = KNeighborsClassifier()

voting\_model = VotingClassifier(estimators=[('lr', lr\_model), ('rf', rf\_model), ('svc', svc\_model)], voting='soft')

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

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

# Voting Classifier performance

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

voting\_precision = precision\_score(y\_test, voting\_predictions)

voting\_recall = recall\_score(y\_test, voting\_predictions)

voting\_f1 = f1\_score(y\_test, voting\_predictions)

# Stacking Classifier

stacking\_model = StackingClassifier(estimators=[('lr', lr\_model), ('svc', svc\_model), ('rf', rf\_model)], final\_estimator=LogisticRegression())

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

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

# Stacking Classifier performance

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

stacking\_precision = precision\_score(y\_test, stacking\_predictions)

stacking\_recall = recall\_score(y\_test, stacking\_predictions)

stacking\_f1 = f1\_score(y\_test, stacking\_predictions)

# Creating a DataFrame to compare the results

results = pd.DataFrame({

    'Model': ['Random Forest (Bagging)', 'AdaBoost (Boosting)', 'Gradient Boosting', 'Voting Classifier', 'Stacking Classifier'],

    'Accuracy': [rf\_accuracy, ada\_accuracy, gb\_accuracy, voting\_accuracy, stacking\_accuracy],

    'Precision': [rf\_precision, ada\_precision, gb\_precision, voting\_precision, stacking\_precision],

    'Recall': [rf\_recall, ada\_recall, gb\_recall, voting\_recall, stacking\_recall],

    'F1-Score': [rf\_f1, ada\_f1, gb\_f1, voting\_f1, stacking\_f1]

})

print(results)

Q2) Write a python Program to implement multiple linear Regression for house price data set.

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: Create a sample dataset

data = {

'Size (sqft)': [1500, 1800, 2400, 3000, 3500, 4000, 4500, 5000, 5500, 6000],

'Num Rooms': [3, 4, 3, 5, 4, 5, 6, 6, 7, 8],

'Age (years)': [10, 15, 20, 5, 30, 12, 25, 8, 18, 14],

'Price': [400000, 500000, 600000, 650000, 700000, 800000, 850000, 900000, 950000, 1000000]

}

# Step 2: Convert the dataset into a DataFrame

df = pd.DataFrame(data)

# Step 3: Split the data into features (X) and target (y)

X = df[['Size (sqft)', 'Num Rooms', 'Age (years)']] # Features

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

# Step 4: 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)

# Step 5: Standardize the features (important for regression models)

scaler = StandardScaler()

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

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

# Step 6: Create and train the Multiple Linear Regression model

model = LinearRegression()

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

# Step 7: Make predictions on the test data

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

# Step 8: Evaluate the model

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

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

# Display the results

print(f"Predicted Prices: {y\_pred}")

print(f"Actual Prices: {y\_test.values}")

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

print(f"R² Score: {r2}")

Slip-16

Q1) Create a two layered neural network with relu and sigmoid activation function.

Sol:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the neural network model

model = Sequential()

# Input Layer and Hidden Layer (Dense Layer with ReLU activation)

# Changed input\_dim to 20 to match the actual input data shape

model.add(Dense(units=64, activation='relu', input\_dim=20))

# Output Layer (Dense Layer with Sigmoid activation for binary classification)

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

# Compile the model

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

# Summary of the model

model.summary()

# Fit the model to your dataset

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

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

Slip-18

Q1) Write a python program to implement K-means Algorithm on a Diabetes Dataset.

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load\_diabetes

# Step 1: Load the Diabetes Dataset (using sklearn)

diabetes = load\_diabetes()

# Convert the dataset to a pandas DataFrame

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

# Step 2: Standardize the features (important for K-means)

scaler = StandardScaler()

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

# Step 3: Apply K-means clustering

# Choosing the number of clusters, here we choose 2 clusters for simplicity

kmeans = KMeans(n\_clusters=2, random\_state=42)

df['Cluster'] = kmeans.fit\_predict(df\_scaled)

# Step 4: Visualize the clustering result (optional, works for 2D or 3D data)

# Since the dataset has more than 2 dimensions, we will reduce it to 2D using PCA for visualization

from sklearn.decomposition import PCA

pca = PCA(n\_components=2)

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

# Create a DataFrame for the 2D PCA components

df\_pca\_df = pd.DataFrame(df\_pca, columns=['PC1', 'PC2'])

df\_pca\_df['Cluster'] = df['Cluster']

# Plot the PCA components with cluster labels

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

sns.scatterplot(x='PC1', y='PC2', hue='Cluster', palette='viridis', data=df\_pca\_df, s=100, marker='o')

plt.title('K-means Clustering of Diabetes Dataset (PCA-reduced)', fontsize=16)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend(title='Cluster')

plt.show()

# Step 5: Display the cluster centers

print(f"Cluster Centers (in original scale):")

print(kmeans.cluster\_centers\_)

# Step 6: Evaluate clustering (in terms of inertia and silhouette score)

print(f"Inertia (Sum of squared distances to the nearest cluster center): {kmeans.inertia\_}")

from sklearn.metrics import silhouette\_score

sil\_score = silhouette\_score(df\_scaled, kmeans.labels\_)

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

Q2) Write Python program to implement Polynomial Linear Regression for salary\_positions dataset.

import pandas as pd

import numpy as np

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: Create a sample salary\_positions dataset

# Let's assume the dataset has 'Position Level' and 'Salary' columns

data = {

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

'Salary': [45000, 50000, 60000, 80000, 100000, 120000, 150000, 180000, 200000, 250000]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Step 2: Prepare the features and target variable

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

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

# Step 3: Split the data into training and testing sets (optional)

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

# Step 4: Apply Polynomial Feature Transformation

poly\_reg = PolynomialFeatures(degree=4) # You can change the degree based on your data

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

# Step 5: Train the Polynomial Linear Regression Model

poly\_model = LinearRegression()

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

# Step 6: Predict using the trained model

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

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

# Step 7: Visualize the original data and the Polynomial Regression result

# Plotting for training data

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

plt.scatter(X, y, color='blue') # Original data points

plt.plot(X, poly\_model.predict(poly\_reg.fit\_transform(X)), color='red') # Polynomial regression line

plt.title('Polynomial Linear Regression for Salary Prediction')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.show()

# Step 8: Display model's performance on the test set

print("Predicted salaries:", y\_pred)

print("Actual salaries:", y\_test)

Slip-19

Q1) 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.

Sol:

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

# Step 1: Load the dataset

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

# Step 2: Explore the data

print(data.head())  # Print the first few rows of the dataset

print(data.describe())  # Get summary statistics

# Extracting features and target variable

X = data.iloc[:, 1:2].values  # Position levels (assuming they are in the second column)

y = data.iloc[:, 2].values     # Salaries (assuming they are in the third column)

# Step 3: Fit the Simple Linear Regression Model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

# Step 4: Fit the Polynomial Linear Regression Model

poly\_features = PolynomialFeatures(degree=4)  # You can adjust the degree based on your data

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

polynomial\_model = LinearRegression()

polynomial\_model.fit(X\_poly, y)

# Step 5: Evaluate the models

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

y\_pred\_poly = polynomial\_model.predict(X\_poly)

# Calculate Mean Squared Error and R^2 Score

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

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

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

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

print(f"Linear Regression - MSE: {mse\_linear}, R^2: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, R^2: {r2\_poly}")

# Step 6: Plot the results

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

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

plt.scatter(X, y\_pred\_poly, color='green', label='Polynomial Regression')

plt.title('Salary vs Position Level')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 7: Predict Salaries for Level 11 and Level 12

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

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

salary\_level\_11\_linear = linear\_model.predict(level\_11)

salary\_level\_11\_poly = polynomial\_model.predict(poly\_features.transform(level\_11))

salary\_level\_12\_linear = linear\_model.predict(level\_12)

salary\_level\_12\_poly = polynomial\_model.predict(poly\_features.transform(level\_12))

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

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

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

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

Q2) Write a python program to implement Naïve 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

from sklearn.preprocessing import LabelEncoder

# Sample Weather Forecast Dataset (can be saved as CSV or used as is)

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', 'Hot', 'Mild'],

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

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

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

}

# Convert the data to a pandas DataFrame

df = pd.DataFrame(data)

# Display the first few rows

print("Dataset:")

print(df.head())

# Encode categorical variables using LabelEncoder

le = LabelEncoder()

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

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

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

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

df['PlayTennis'] = le.fit\_transform(df['PlayTennis']) # Target variable

# Features (X) and Target (y)

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

y = df['PlayTennis']

# 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 and train the Naïve Bayes model

nb = GaussianNB()

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

# Make predictions on the test data

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

# Evaluate the model

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

print(f"\nAccuracy of the Naïve Bayes model: {accuracy \* 100:.2f}%")

# Display the predictions

print("\nPredictions on Test Data:")

print(y\_pred)

Slip-20

Q1) Implement Ridge Regression, Lasso regression, ElasticNet 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.

Sol: 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 Ridge, Lasso, ElasticNet

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

# Step 1: Load the dataset

# Make sure to adjust the path to where your boston\_houses.csv is located

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

# Step 2: Extract relevant features

data = df[['RM', 'Price']]  # Select RM and Price columns

# Display the first few rows of the dataset

print(data.head())

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

X = data[['RM']]

y = data['Price']

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

# Step 4: Fit Ridge Regression

ridge\_model = Ridge(alpha=1.0)  # You can adjust alpha

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

# Step 5: Fit Lasso Regression

lasso\_model = Lasso(alpha=1.0)  # You can adjust alpha

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

# Step 6: Fit ElasticNet Regression

elastic\_net\_model = ElasticNet(alpha=1.0, l1\_ratio=0.5)  # You can adjust alpha and l1\_ratio

elastic\_net\_model.fit(X\_train, y\_train)

# Step 7: Make predictions for test data

y\_pred\_ridge = ridge\_model.predict(X\_test)

y\_pred\_lasso = lasso\_model.predict(X\_test)

y\_pred\_elastic\_net = elastic\_net\_model.predict(X\_test)

# Step 8: Evaluate Models

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

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

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

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

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

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

print(f'Ridge Regression MSE: {mse\_ridge}, R²: {r2\_ridge}')

print(f'Lasso Regression MSE: {mse\_lasso}, R²: {r2\_lasso}')

print(f'ElasticNet Regression MSE: {mse\_elastic\_net}, R²: {r2\_elastic\_net}')

# Step 9: Predict the price for a house with 5 rooms

rooms = np.array([[5]])  # 5 rooms

pred\_price\_ridge = ridge\_model.predict(rooms)

pred\_price\_lasso = lasso\_model.predict(rooms)

pred\_price\_elastic\_net = elastic\_net\_model.predict(rooms)

print(f'Predicted Price for a house with 5 rooms (Ridge): ${pred\_price\_ridge[0]:,.2f}')

print(f'Predicted Price for a house with 5 rooms (Lasso): ${pred\_price\_lasso[0]:,.2f}')

print(f'Predicted Price for a house with 5 rooms (ElasticNet): ${pred\_price\_elastic\_net[0]:,.2f}')

# Optional: Plotting the results

plt.scatter(data['RM'], data['Price'], color='blue', label='Data')

plt.scatter(X\_test, y\_test, color='black', label='Test Data', alpha=0.5)

plt.plot(X\_test, y\_pred\_ridge, color='red', label='Ridge Prediction')

plt.plot(X\_test, y\_pred\_lasso, color='green', label='Lasso Prediction')

plt.plot(X\_test, y\_pred\_elastic\_net, color='purple', label='ElasticNet Prediction')

plt.xlabel('Number of Rooms (RM)')

plt.ylabel('Price')

plt.title('House Price Predictions')

plt.legend()

plt.show()

Q2) Write python program to implement Decision Tree whether or not to play Tennis.

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt

# Sample Weather Forecast Dataset (can be saved as CSV or used as is)

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', 'Hot', 'Mild'],

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

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

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

}

# Convert the data to a pandas DataFrame

df = pd.DataFrame(data)

# Display the first few rows

print("Dataset:")

print(df.head())

# Encode categorical variables using LabelEncoder

le = LabelEncoder()

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

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

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

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

df['PlayTennis'] = le.fit\_transform(df['PlayTennis']) # Target variable

# Features (X) and Target (y)

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

y = df['PlayTennis']

# 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 and train the Decision Tree model

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

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

# Make predictions on the test data

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

# Evaluate the model

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

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

# Visualize the Decision Tree

plt.figure(figsize=(12, 8))

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

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

plt.show()

# Display predictions on test data

print("\nPredictions on Test Data:")

print(y\_pred)

Slip-21

Q1) 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

import numpy as np

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

# Step 1: Generate synthetic dataset

np.random.seed(0)

# Generating random data

size = np.random.randint(500, 5000, 100) # Size in sq ft

bedrooms = np.random.randint(1, 6, 100) # 1 to 5 bedrooms

bathrooms = np.random.randint(1, 4, 100) # 1 to 3 bathrooms

age = np.random.randint(0, 30, 100) # Age of the house

distance = np.random.uniform(1, 20, 100) # Distance to city center in miles

# Creating a DataFrame

data = pd.DataFrame({

'Size': size,

'Bedrooms': bedrooms,

'Bathrooms': bathrooms,

'Age': age,

'Distance': distance

})

# Define price based on some arbitrary linear function with noise

data['Price'] = (data['Size'] \* 150 +

data['Bedrooms'] \* 10000 +

data['Bathrooms'] \* 5000 -

data['Age'] \* 2000 -

data['Distance'] \* 3000 +

np.random.normal(0, 20000, 100)) # Adding noise

# Display the first few rows of the dataset

print(data.head())

# Step 2: Split the dataset into training and testing sets

X = data[['Size', 'Bedrooms', 'Bathrooms', 'Age', 'Distance']]

y = data['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Step 3: Create the multiple 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)

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

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

print(f'R^2 Score: {r2}')

# Optional: Plotting predicted vs actual prices

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

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

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

plt.show()

Q2) 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, classification\_report

from sklearn.preprocessing import LabelEncoder

# Load the UniversalBank dataset

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

# Display the first few rows of the dataset

print("Dataset:")

print(df.head())

# Check for missing values

print("\nMissing values:")

print(df.isnull().sum())

# Preprocessing: Drop the 'ID' and 'ZIP Code' columns as they are not useful for prediction

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

# Encode categorical variables if any (e.g., 'Personal Loan', 'Education', 'Securities Account', etc.)

label\_encoder = LabelEncoder()

df['Personal Loan'] = label\_encoder.fit\_transform(df['Personal Loan']) # Binary encoding

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

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

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

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

# Features (X) and Target (y)

X = df.drop(columns=['Personal Loan']) # Drop target variable

y = df['Personal Loan'] # Target variable: whether the customer accepted a personal loan or not

# 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)

# Standardize the features using StandardScaler (important for SVM)

scaler = StandardScaler()

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

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

# Create and train the Linear SVM model

svm = SVC(kernel='linear', random\_state=42)

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

# Make predictions on the test data

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

# Evaluate the model

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

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

# Display detailed classification report

print("\nClassification Report:")

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

Slip-22

Q1) write a python program to implement simple linear regression for predicting house price.

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

# Step 1: Create a simple dataset with square footage and house price

data = {

'SquareFootage': [1500, 1800, 2400, 3000, 3500, 4000, 4500, 5000],

'Price': [400000, 500000, 600000, 650000, 700000, 750000, 800000, 850000]

}

# Convert to DataFrame

df = pd.DataFrame(data)

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

X = df[['SquareFootage']] # Feature

y = df['Price'] # Target

# 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: Create and train the Simple Linear Regression model

model = LinearRegression()

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

# Step 5: Make predictions on the test set

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

# Step 6: Evaluate the model

mae = mean\_absolute\_error(y\_test, y\_pred)

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

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

# Output the evaluation metrics

print(f"Mean Absolute Error: {mae}")

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

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

# Step 7: Visualize the results

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

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

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

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

plt.xlabel('Square Footage')

plt.ylabel('Price')

plt.legend()

plt.show()

Q2) Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

# Install the mlxtend library

!pip install mlxtend

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

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

from mlxtend.preprocessing import TransactionEncoder

# Example groceries dataset (replace this with your actual dataset if needed)

groceries = [['milk', 'bread', 'eggs'],

['milk', 'bread'],

['bread', 'butter'],

['milk', 'bread', 'butter'],

['milk', 'eggs'],

['bread', 'butter', 'eggs'],

['milk', 'bread', 'eggs', 'butter']]

# Convert the dataset into a format suitable for Apriori

te = TransactionEncoder()

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

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

# Apply Apriori to find frequent itemsets with a minimum support of 0.25

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

# Display the frequent itemsets

print(frequent\_itemsets)

# Plot frequent itemsets by support

frequent\_itemsets.plot(kind='bar', x='itemsets', y='support', figsize=(10,6), legend=False)

plt.title('Frequent Itemsets by Support')

plt.ylabel('Support')

plt.xlabel('Itemsets')

plt.xticks(rotation=90)

plt.show()

# Generate association rules with minimum confidence of 0.6

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

# Display the association rules

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

# Scatter plot for Association Rules (Lift vs Confidence)

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

plt.scatter(rules['confidence'], rules['lift'], alpha=0.7, edgecolors='r')

plt.title('Association Rules (Lift vs Confidence)')

plt.xlabel('Confidence')

plt.ylabel('Lift')

plt.grid(True)

plt.show()

Slip-23

Q1) 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.

Sol:

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

# Step 1: Load the dataset

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

# Step 2: Explore the data

print(data.head())  # Print the first few rows of the dataset

print(data.describe())  # Get summary statistics

# Extracting features and target variable

X = data.iloc[:, 1:2].values  # Position levels (assuming they are in the second column)

y = data.iloc[:, 2].values     # Salaries (assuming they are in the third column)

# Step 3: Fit the Simple Linear Regression Model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

# Step 4: Fit the Polynomial Linear Regression Model

poly\_features = PolynomialFeatures(degree=4)  # You can adjust the degree based on your data

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

polynomial\_model = LinearRegression()

polynomial\_model.fit(X\_poly, y)

# Step 5: Evaluate the models

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

y\_pred\_poly = polynomial\_model.predict(X\_poly)

# Calculate Mean Squared Error and R^2 Score

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

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

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

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

print(f"Linear Regression - MSE: {mse\_linear}, R^2: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, R^2: {r2\_poly}")

# Step 6: Plot the results

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

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

plt.scatter(X, y\_pred\_poly, color='green', label='Polynomial Regression')

plt.title('Salary vs Position Level')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 7: Predict Salaries for Level 11 and Level 12

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

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

salary\_level\_11\_linear = linear\_model.predict(level\_11)

salary\_level\_11\_poly = polynomial\_model.predict(poly\_features.transform(level\_11))

salary\_level\_12\_linear = linear\_model.predict(level\_12)

salary\_level\_12\_poly = polynomial\_model.predict(poly\_features.transform(level\_12))

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

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

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

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

Q2) 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

# Create a custom dataset

data = {

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

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

'City': ['New York', 'Los Angeles', 'Chicago', np.nan, 'Houston', 'Phoenix'],

'Salary': [50000, 60000, np.nan, 70000, 80000, 90000]

}

# Create a DataFrame from the dataset

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Check for null values in the dataset

null\_values = df.isnull().sum()

print("\nNull Values in Each Column:")

print(null\_values)

# Remove rows with any null values

df\_cleaned = df.dropna()

# Display the cleaned dataset

print("\nCleaned Dataset (Null values removed):")

print(df\_cleaned)

Slip-24

Q1) . Write a python program to Implement Decision Tree classifier model onData 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>)

Sol:

# Step 1: Import necessary libraries

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

import matplotlib.pyplot as plt

from sklearn import tree

# Step 2: Upload dataset to Colab (if using direct upload)

from google.colab import files

uploaded = files.upload()

# Step 3: Load the dataset (assuming the file is named 'banknote\_data.csv')

# If using Google Drive, replace '/content/...' with your file path

#df = pd.read\_csv('/content/banknote\_data.csv')

import io

df = pd.read\_csv(io.BytesIO(uploaded['data\_banknote\_authentication.txt'])) # This line is added to read data from uploaded file.

# Step 4: Explore the dataset (optional)

print(df.head())

# Step 5: Split data into features (X) and label (y)

X = df.drop('label', axis=1)  # Features: variance, skewness, curtosis, entropy

y = df['label']  # Labels: genuine (1), forged (0)

# Step 6: 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 7: Create and train the Decision Tree classifier

clf = DecisionTreeClassifier(random\_state=42)

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

# Step 8: Make predictions on the test set

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

# Step 9: Evaluate the model

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

print("Accuracy:", accuracy)

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

# Step 10: Visualize the Decision Tree (optional)

plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, filled=True, feature\_names=X.columns, class\_names=['forged', 'genuine'], rounded=True)

plt.show()

Q2) 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, classification\_report

from sklearn.preprocessing import LabelEncoder

# Load the UniversalBank dataset

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

# Display the first few rows of the dataset

print("Dataset:")

print(df.head())

# Check for missing values

print("\nMissing values:")

print(df.isnull().sum())

# Preprocessing: Drop the 'ID' and 'ZIP Code' columns as they are not useful for prediction

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

# Encode categorical variables if any (e.g., 'Personal Loan', 'Education', 'Securities Account', etc.)

label\_encoder = LabelEncoder()

df['Personal Loan'] = label\_encoder.fit\_transform(df['Personal Loan']) # Binary encoding

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

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

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

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

# Features (X) and Target (y)

X = df.drop(columns=['Personal Loan']) # Drop target variable

y = df['Personal Loan'] # Target variable: whether the customer accepted a personal loan or not

# 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)

# Standardize the features using StandardScaler (important for SVM)

scaler = StandardScaler()

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

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

# Create and train the Linear SVM model

svm = SVC(kernel='linear', random\_state=42)

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

# Make predictions on the test data

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

# Evaluate the model

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

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

# Display detailed classification report

print("\nClassification Report:")

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

Slip-25

Q1) write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

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

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

# Instead of using load\_boston, fetch the data directly or use an alternative

# Fetching directly (as suggested in the error message):

data\_url = "http://lib.stat.cmu.edu/datasets/boston"

raw\_df = pd.read\_csv(data\_url, sep="\s+", skiprows=22, header=None)

data = np.hstack([raw\_df.values[::2, :], raw\_df.values[1::2, :2]])

target = raw\_df.values[1::2, 2]

# Create a DataFrame

data = pd.DataFrame(data, columns=['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'])

data['PRICE'] = target

# ... (rest of your code remains the same, using 'data' DataFrame) ...

# Save the dataset to a CSV file (optional)

data.to\_csv('HousingData.csv', index=False)

print("\nDataset saved to HousingData.csv")

# Display the first few rows of the dataset

print("\nFirst few rows of the dataset:")

print(data.head())

# Select the feature(s) and target variable

X = data[['RM']] # For simplicity, using 'RM' (average number of rooms per dwelling)

y = data['PRICE'] # Target: House prices

# 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)

# Polynomial feature transformation (degree 2 for example)

poly = PolynomialFeatures(degree=2)

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

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

# Train the Polynomial Linear Regression model

model = LinearRegression()

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

# Make predictions on the test data

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

# Evaluate the model

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

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

print(f"\nMean Squared Error: {mse}")

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

# Plot the results for the training and test set (Visualizing the Polynomial fit)

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

# Plot actual data points and regression curve

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

plt.plot(np.sort(X\_test.values), model.predict(poly.transform(np.sort(X\_test.values).reshape(-1, 1))),

color='red', label='Polynomial Regression Line')

plt.xlabel('Average Number of Rooms (RM)')

plt.ylabel('House Price')

plt.title('Polynomial Linear Regression - Boston Housing Prices')

plt.legend()

plt.show()

Q2) Create a two layered neural network with relu and sigmoid activation function.

Sol:

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the neural network model

model = Sequential()

# Input Layer and Hidden Layer (Dense Layer with ReLU activation)

# Changed input\_dim to 20 to match the actual input data shape

model.add(Dense(units=64, activation='relu', input\_dim=20))

# Output Layer (Dense Layer with Sigmoid activation for binary classification)

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

# Compile the model

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

# Summary of the model

model.summary()

# Fit the model to your dataset

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

# Evaluate the model

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

Slip-26

Q1) 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.

Sol:

# Step 1: Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

!pip install numpy pandas scikit-learn matplotlib

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score

# Step 2: Load the dataset

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

columns = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',

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

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

# Step 3: Data preprocessing

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

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

# Normalize the data (standardization)

scaler = StandardScaler()

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

# Step 4: Split the data into train and test 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)

# Step 5: Finding the optimal value of K

error\_rates = []

# Test K values from 1 to 20

for k in range(1, 21):

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train, y\_train)

    pred\_k = knn.predict(X\_test)

    error = np.mean(pred\_k != y\_test)

    error\_rates.append(error)

# Plot error rates to find optimal K

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

plt.plot(range(1,21), error\_rates, color='blue', linestyle='dashed', marker='o',

         markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Mean Error')

plt.show()

# Step 6: Train the model with the optimal K

optimal\_k = error\_rates.index(min(error\_rates)) + 1

print(f"Optimal value of K: {optimal\_k}")

# Train the KNN classifier with the optimal K

knn = KNeighborsClassifier(n\_neighbors=optimal\_k)

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

# Step 7: Predict and evaluate

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

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

print(f"Accuracy with K={optimal\_k}: {accuracy \* 100:.2f}%")

# Test with a new patient data (replace with actual values)

new\_patient = [[6, 148, 72, 35, 0, 33.6, 0.627, 50]]

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

prediction = knn.predict(new\_patient\_scaled)

print(f"The new patient is {'diabetic' if prediction == 1 else 'not diabetic'}")

Q2) Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25

# Install the mlxtend library

!pip install mlxtend

# Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

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

from mlxtend.preprocessing import TransactionEncoder

# Example groceries dataset (replace this with your actual dataset if needed)

groceries = [['milk', 'bread', 'eggs'],

['milk', 'bread'],

['bread', 'butter'],

['milk', 'bread', 'butter'],

['milk', 'eggs'],

['bread', 'butter', 'eggs'],

['milk', 'bread', 'eggs', 'butter']]

# Convert the dataset into a format suitable for Apriori

te = TransactionEncoder()

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

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

# Apply Apriori to find frequent itemsets with a minimum support of 0.25

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

# Display the frequent itemsets

print(frequent\_itemsets)

# Plot frequent itemsets by support

frequent\_itemsets.plot(kind='bar', x='itemsets', y='support', figsize=(10,6), legend=False)

plt.title('Frequent Itemsets by Support')

plt.ylabel('Support')

plt.xlabel('Itemsets')

plt.xticks(rotation=90)

plt.show()

# Generate association rules with minimum confidence of 0.6

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

# Display the association rules

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

# Scatter plot for Association Rules (Lift vs Confidence)

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

plt.scatter(rules['confidence'], rules['lift'], alpha=0.7, edgecolors='r')

plt.title('Association Rules (Lift vs Confidence)')

plt.xlabel('Confidence')

plt.ylabel('Lift')

plt.grid(True)

plt.show()

Slip-27

Q1) 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.

Sol:

import pandas as pd

import numpy as np

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

# Step 1: Generate synthetic dataset

np.random.seed(0)

# Generating random data

size = np.random.randint(500, 5000, 100) # Size in sq ft

bedrooms = np.random.randint(1, 6, 100) # 1 to 5 bedrooms

bathrooms = np.random.randint(1, 4, 100) # 1 to 3 bathrooms

age = np.random.randint(0, 30, 100) # Age of the house

distance = np.random.uniform(1, 20, 100) # Distance to city center in miles

# Creating a DataFrame

data = pd.DataFrame({

'Size': size,

'Bedrooms': bedrooms,

'Bathrooms': bathrooms,

'Age': age,

'Distance': distance

})

# Define price based on some arbitrary linear function with noise

data['Price'] = (data['Size'] \* 150 +

data['Bedrooms'] \* 10000 +

data['Bathrooms'] \* 5000 -

data['Age'] \* 2000 -

data['Distance'] \* 3000 +

np.random.normal(0, 20000, 100)) # Adding noise

# Display the first few rows of the dataset

print(data.head())

# Step 2: Split the dataset into training and testing sets

X = data[['Size', 'Bedrooms', 'Bathrooms', 'Age', 'Distance']]

y = data['Price']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Step 3: Create the multiple 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)

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

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

print(f'R^2 Score: {r2}')

# Optional: Plotting predicted vs actual prices

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

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

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

plt.show()

Q2) 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.

Sol:

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

# Step 1: Load the dataset

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

# Step 2: Explore the data

print(data.head())  # Print the first few rows of the dataset

print(data.describe())  # Get summary statistics

# Extracting features and target variable

X = data.iloc[:, 1:2].values  # Position levels (assuming they are in the second column)

y = data.iloc[:, 2].values     # Salaries (assuming they are in the third column)

# Step 3: Fit the Simple Linear Regression Model

linear\_model = LinearRegression()

linear\_model.fit(X, y)

# Step 4: Fit the Polynomial Linear Regression Model

poly\_features = PolynomialFeatures(degree=4)  # You can adjust the degree based on your data

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

polynomial\_model = LinearRegression()

polynomial\_model.fit(X\_poly, y)

# Step 5: Evaluate the models

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

y\_pred\_poly = polynomial\_model.predict(X\_poly)

# Calculate Mean Squared Error and R^2 Score

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

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

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

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

print(f"Linear Regression - MSE: {mse\_linear}, R^2: {r2\_linear}")

print(f"Polynomial Regression - MSE: {mse\_poly}, R^2: {r2\_poly}")

# Step 6: Plot the results

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

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

plt.scatter(X, y\_pred\_poly, color='green', label='Polynomial Regression')

plt.title('Salary vs Position Level')

plt.xlabel('Position Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Step 7: Predict Salaries for Level 11 and Level 12

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

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

salary\_level\_11\_linear = linear\_model.predict(level\_11)

salary\_level\_11\_poly = polynomial\_model.predict(poly\_features.transform(level\_11))

salary\_level\_12\_linear = linear\_model.predict(level\_12)

salary\_level\_12\_poly = polynomial\_model.predict(poly\_features.transform(level\_12))

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

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

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

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

Slip-28

Q1) 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.

Sol:

# Import necessary libraries

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 import metrics

# Step 1: Load the dataset

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

# Step 2: Preprocess the text data

X = newsgroups.data  # Features (news articles)

y = newsgroups.target  # Target labels (categories)

# Step 3: Convert text to numerical vectors using CountVectorizer

vectorizer = CountVectorizer(stop\_words='english')

X\_vectorized = vectorizer.fit\_transform(X)

# Step 4: Split the data into training and testing sets

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

# Step 5: Train the Multinomial Naïve Bayes model

model = MultinomialNB()

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

# Step 6: Evaluate the model

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

# Print accuracy and classification report

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

print(f'Accuracy: {accuracy:.4f}')

print('Classification Report:')

print(metrics.classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names))

# Step 7: Make predictions for new articles

def predict\_category(text):

    text\_vectorized = vectorizer.transform([text])  # Vectorize the new text

    predicted\_category = model.predict(text\_vectorized)  # Predict the category

    return newsgroups.target\_names[predicted\_category[0]]  # Return the category name

# Example usage of the prediction function

new\_article = """NASA's Mars rover Curiosity has discovered evidence that liquid water once flowed on the surface of the Red Planet."""

predicted\_category = predict\_category(new\_article)

print(f'Predicted Category: {predicted\_category}')

Q2) 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.

Sol:

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn import datasets

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

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

# Load the iris dataset

iris = datasets.load\_iris()

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

y = iris.target # Labels (flower types)

# Step 3: 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)

# List of SVM kernels

kernels = ['linear', 'poly', 'rbf', 'sigmoid']

accuracy\_results = {}

# Step 4: Train SVM models and evaluate accuracy

for kernel in kernels:

model = SVC(kernel=kernel)

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

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

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

accuracy\_results[kernel] = accuracy

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

# Step 5: Make predictions with user input

print("\nEnter flower measurements to classify:")

sepal\_length = float(input("Sepal Length (cm): "))

sepal\_width = float(input("Sepal Width (cm): "))

petal\_length = float(input("Petal Length (cm): "))

petal\_width = float(input("Petal Width (cm): "))

# Using the best kernel (RBF) for prediction

best\_model = SVC(kernel='rbf')

best\_model.fit(X\_train, y\_train) # Train again on the whole training data

predicted\_class = best\_model.predict([[sepal\_length, sepal\_width, petal\_length, petal\_width]])

# Display the predicted class

flower\_types = iris.target\_names

print(f'The predicted flower type is: {flower\_types[predicted\_class[0]]}')

Slip-29

Q1) Take iris flower dataset and reduce 4D data to 2D data using PCA. Then train the model and predict new flower with given measurements.

Sol:

# Importing required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.decomposition import PCA

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features (4D data)

y = iris.target # Target (species)

# Create a DataFrame for better visualization

iris\_df = pd.DataFrame(data=X, columns=iris.feature\_names)

iris\_df['species'] = y

print(iris\_df.head())

# Apply PCA to reduce dimensions from 4D to 2D

pca = PCA(n\_components=2)

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

# Create a DataFrame for PCA results

pca\_df = pd.DataFrame(data=X\_pca, columns=['PC1', 'PC2'])

pca\_df['species'] = y

# Plot the PCA result

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

scatter = plt.scatter(pca\_df['PC1'], pca\_df['PC2'], c=pca\_df['species'], cmap='viridis')

plt.title('PCA of Iris Dataset')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.colorbar(scatter, ticks=[0, 1, 2], label='Species')

plt.grid()

plt.show()

# Split the data into training and testing sets

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

# Train a Random Forest Classifier

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

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

# Make predictions

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

# Evaluate the model

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

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

# New flower measurements (example)

new\_flower = np.array([[5.1, 3.5, 1.4, 0.2]]) # Example measurements

# Transform the new flower measurements using PCA

new\_flower\_pca = pca.transform(new\_flower)

# Predict the species of the new flower

predicted\_species = model.predict(new\_flower\_pca)

predicted\_species\_name = iris.target\_names[predicted\_species][0]

print(f"Predicted Species: {predicted\_species\_name}")

Q2) 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.

Sol:

# Importing required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

# Creating a sample employee dataset

data = {

    'EmployeeID': range(1, 21),

    'Age': np.random.randint(22, 60, size=20),

    'Income': np.random.randint(30000, 120000, size=20),

    'YearsAtCompany': np.random.randint(1, 30, size=20)

}

df = pd.DataFrame(data)

# Display the first few rows of the dataset

print(df.head())

# Check for missing values

print(df.isnull().sum())

# Drop any rows with missing values

df = df.dropna()

# Selecting features for clustering

features = df[['Age', 'Income', 'YearsAtCompany']]

# Finding the optimal number of clusters using the Elbow Method

wcss = []

for i in range(1, 11):

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

    kmeans.fit(features)

    wcss.append(kmeans.inertia\_)  # Within-cluster sum of squares

# Plotting the elbow curve

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

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

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

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

plt.ylabel('WCSS')

plt.grid()

plt.show()

# Calculate silhouette scores for each k

silhouette\_scores = []

for i in range(2, 11):

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

    kmeans.fit(features)

    score = silhouette\_score(features, kmeans.labels\_)

    silhouette\_scores.append(score)

# Plotting the Silhouette Scores

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

plt.plot(range(2, 11), silhouette\_scores, marker='o')

plt.title('Silhouette Scores for Different k Values')

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

plt.ylabel('Silhouette Score')

plt.grid()

plt.show()

# Fit the K-means model with the chosen number of clusters

optimal\_k = 3  # Replace this with the optimal k found from the plots

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

df['Cluster'] = kmeans.fit\_predict(features)

# Display the cluster assignment

print(df[['EmployeeID', 'Income', 'Cluster']])