Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25. give me this code for jupyter anaconda.provide me a dataset related to it in a excel download form file

**Slips**

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

# Import necessary libraries

import pandas as pd

from mlxtend.frequent\_patterns import apriori

# Load dataset

df = pd.read\_excel("C:/Users/HRITIK/Downloads/groceries\_dataset.xlsx")

# Drop the Transaction column

df = df.drop(columns=["Transaction"])

# Convert dataset to boolean type

df = df.astype(bool)

# Apply the Apriori algorithm with minimum support of 0.25

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

# Function to calculate support

def calculate\_support(itemset, df):

return df[itemset].all(axis=1).mean()

# Function to calculate confidence

def calculate\_confidence(itemset, antecedent, df):

return calculate\_support(itemset, df) / calculate\_support(antecedent, df)

# Function to calculate lift

def calculate\_lift(confidence, consequent, df):

return confidence / calculate\_support(consequent, df)

# Calculate rules manually

rules = []

for \_, row in frequent\_itemsets.iterrows():

itemset = row['itemsets']

if len(itemset) > 1:

for antecedent in itemset:

consequent = itemset - {antecedent}

antecedent = frozenset([antecedent])

confidence = calculate\_confidence(itemset, antecedent, df)

lift = calculate\_lift(confidence, consequent, df)

rules.append({

'antecedent': antecedent,

'consequent': consequent,

'support': row['support'],

'confidence': confidence,

'lift': lift

})

# Convert rules to DataFrame for easy display

rules\_df = pd.DataFrame(rules)

# Display the frequent itemsets and association rules

print("Frequent Itemsets:\n", frequent\_itemsets)

print("\nAssociation Rules:\n", rules\_df)

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

# Import libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load Iris dataset from Excel file

iris\_df = pd.read\_excel("C:/Users/HRITIK/Downloads/file/iris\_dataset.xlsx")

# Plot scatter plot

sns.pairplot(iris\_df, hue="species", palette="Set1")

plt.show()

**Slips 2**

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

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Load the dataset

df = pd.read\_excel("C:/Users/HRITIK/Downloads/file/house\_prices\_dataset.xlsx")

# Display the first few rows of the dataset

print("Dataset preview:")

print(df.head())

# Check for and remove null values

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

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

df = df.dropna()

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

print(df)

# Define features (Square\_Feet, Bedrooms) and target (Price)

X = df[['Square\_Feet', 'Bedrooms']]

y = df['Price']

# 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)

# Initialize 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

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

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

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

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

# Predicting a new house price (example)

new\_data = pd.DataFrame({'Square\_Feet': [3200], 'Bedrooms': [4]})

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

print(f"\nPredicted price for 3200 sq.ft. with 4 bedrooms: ${predicted\_price[0]:,.2f}")

**Q.2. The data set refers to clients of a wholesale distributor. It includes the annual spending in monetary units on diverse product categories. Using data Wholesale customer dataset compute agglomerative clustering to find out annual spending clients in the same region**

# Importing required libraries

import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import AgglomerativeClustering

import matplotlib.pyplot as plt

import seaborn as sns

from scipy.cluster.hierarchy import dendrogram, linkage

# Loading the dataset

data = pd.read\_excel('C:/Users/HRITIK/Downloads/file/Wholesale\_customers\_dataset.xlsx')

# Selecting relevant features for clustering

X = data[['Fresh', 'Milk', 'Grocery', 'Frozen', 'Detergents\_Paper', 'Delicassen']]

# Standardizing the data

scaler = StandardScaler()

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

# Performing Agglomerative Clustering

clustering = AgglomerativeClustering(n\_clusters=3, metric='euclidean', linkage='ward')

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

# Visualizing the clusters using a pairplot

sns.pairplot(data, hue='Cluster', vars=['Fresh', 'Milk', 'Grocery', 'Frozen'])

plt.show()

# Plotting dendrogram to visualize the hierarchy

linked = linkage(X\_scaled, method='ward')

plt.figure(figsize=(10, 7))

dendrogram(linked, orientation='top', labels=data.index, distance\_sort='descending', show\_leaf\_counts=True)

plt.title('Dendrogram')

plt.show()

**Slips 3**

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

# Import necessary libraries

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

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

# Load the dataset

data = pd.read\_excel("C:/Users/HRITIK/Downloads/file/house\_price\_data3.xlsx")

# Separate features and target variable

X = data[['Square\_Feet', 'Bedrooms', 'Age']]

y = data['Price']

# Split the data into training and testing sets

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

# Initialize the Linear Regression model

model = LinearRegression()

# Train the model

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("Mean Squared Error:", mse)

print("R-squared Score:", r2)

# Display model coefficients

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

print("Coefficients:", model.coef\_)

**Q.2. Use dataset crash.csv is an accident survivor’s dataset portal for USA hosted by data.gov. The dataset contains passengers age and speed of vehicle (mph) at the time of impact and fate of passengers (1 for survived and 0 for not survived) after a crash. use logistic regression to decide if the age and speed can predict the survivability of the passengers**

# Import necessary libraries

import pandas as pd

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

from sklearn.linear\_model import LogisticRegression

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

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

# Load the Excel dataset

data = pd.read\_excel("C:/Users/HRITIK/Downloads/file/crash\_dataset.xlsx")

# Replace with the path if file location is different

# Display the first few rows of the dataset

print(data.head())

# Data preprocessing: Checking for null values and data types

print(data.info())

print(data.isnull().sum()) # Check for any missing values

# Splitting dataset into features (X) and target variable (y)

X = data[['age', 'speed\_mph']]

y = data['survived']

# 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)

# Create a Logistic Regression model

model = LogisticRegression()

# Train the model on the training data

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

# Make predictions on the test data

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

# 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("Accuracy:", accuracy)

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

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

# Plotting the confusion matrix

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

**Slips 4**

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

import os

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

import threading

# Dummy function to initiate threading before sklearn import

\_ = threading.current\_thread()

# Now, import sklearn and proceed with your code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

**1st run this**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

# Load the dataset

data = pd.read\_excel('C:/Users/HRITIK/Downloads/file/Mall\_Customers.xlsx') # Make sure this file is in the same directory

# Preview the dataset

print(data.head())

# Select the relevant features for clustering (e.g., Annual Income and Spending Score)

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

# Scale the features

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 Curve

plt.figure(figsize=(10, 5))

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

plt.title('Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()

# Train the KMeans model with the optimal number of clusters (e.g., k=5 based on the elbow curve)

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state=42)

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

# Plot the clusters

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

for i in range(5):

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', label='Centroids')

plt.title('Customer Clusters')

plt.xlabel('Annual Income (scaled)')

plt.ylabel('Spending Score (scaled)')

plt.legend()

plt.show()

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

# Import necessary libraries

import pandas as pd

import numpy as np

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

data = pd.read\_excel('C:/Users/HRITIK/Downloads/file/house\_prices4.xlsx')

# Inspect the dataset

print(data.head())

# Select the feature and target variable

X = data[['Square\_Feet']] # Feature

y = data['Price'] # Target

# Split the data into training and testing sets

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

# Initialize the Linear Regression model

model = LinearRegression()

# Train the model

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

# Make predictions

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

# Evaluate the model

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

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

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

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

# Plotting the results

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

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

plt.xlabel('Square Feet')

plt.ylabel('Price')

plt.title('Actual vs Predicted House Prices')

plt.legend()

plt.show()

**Slips 5**

**Q.1. 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

# Load dataset

data = pd.read\_excel("C:/Users/HRITIK/Downloads/file/FuelConsumptionDataset.xlsx") # Make sure to have the dataset in the same directory

# Display the first few rows of the dataset

print(data.head())

# Selecting independent and dependent variables

X = data[['Engine\_Size', 'Cylinders', 'Fuel\_Consumption\_City', 'Fuel\_Consumption\_Hwy']]

y = data['CO2\_Emissions']

# Splitting 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)

# Training the Multiple Linear Regression model

model = LinearRegression()

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

# Making predictions

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

# Model evaluation

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

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

print("Mean Squared Error:", mse)

print("R-squared:", r2)

**Q.2. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset).**

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import datasets

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = datasets.load\_iris()

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

data['target'] = iris.target

# Split the dataset into training and testing sets

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

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

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

# Initialize the k-NN model

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the model on the training data

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

# Make predictions on the testing set

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

# Evaluate the model

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

print(f"Accuracy of k-NN model: {accuracy \* 100:.2f}%")

**Slips 6**

**Q.1. Write a python program to implement Polynomial Linear Regression for Boston Housing Dataset.**

from sklearn.datasets import fetch\_california\_housing

import pandas as pd

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

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

# Fetch the California Housing dataset

california\_housing = fetch\_california\_housing()

# Create a DataFrame

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

df['Price'] = california\_housing.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('Price', axis=1), df['Price'], test\_size=0.2, random\_state=42)

# Apply Polynomial Features

poly = PolynomialFeatures(degree=2)

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

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

# Train the Linear Regression Model

model = LinearRegression()

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

# Make predictions

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

# Evaluate the model

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

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

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

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

# Visualize the predictions vs actual values (for the first feature, just as an example)

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

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Actual vs Predicted Prices")

plt.show()

**Q.2. Use K-means clustering model and classify the employees into various income groups or clusters. Preprocess data if require (i.e. drop missing or null values).**

# Import necessary libraries

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

import seaborn as sns

# Load the dataset

file\_path = 'employee\_income\_data.xlsx' # Update this if needed

df = pd.read\_excel("C:/Users/HRITIK/Downloads/file/employee\_income\_data.xlsx")

# Display first few rows

print("Initial Data Preview:")

display(df.head())

# Step 1: Data Preprocessing

# Drop rows with missing values

df.dropna(subset=['Income'], inplace=True)

# Select only the income column for clustering

income\_data = df[['Income']]

# Standardize the data (K-means performs better with standardized data)

scaler = StandardScaler()

income\_data\_scaled = scaler.fit\_transform(income\_data)

# Step 2: Determine Optimal Number of Clusters using Elbow Method

sse = []

k\_range = range(1, 11)

for k in k\_range:

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

kmeans.fit(income\_data\_scaled)

sse.append(kmeans.inertia\_)

# Plot the SSE for each k to find the elbow point

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

plt.plot(k\_range, sse, marker='o')

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

plt.ylabel('Sum of Squared Errors (SSE)')

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

plt.show()

# Step 3: Apply K-means Clustering

# Set number of clusters (choose based on elbow plot, e.g., k=3)

kmeans = KMeans(n\_clusters=3, random\_state=42)

df['IncomeCluster'] = kmeans.fit\_predict(income\_data\_scaled)

# Step 4: Analyze Clusters

print("Income Cluster Analysis:")

print(df.groupby('IncomeCluster')['Income'].describe())

# Visualize the clusters

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

sns.histplot(df, x="Income", hue="IncomeCluster", multiple="stack", palette="viridis")

plt.xlabel('Income')

plt.title('Income Distribution Across Clusters')

plt.show()

# Save clustered data if needed

df.to\_excel("employee\_income\_clusters.xlsx", index=False)

print("Clustered data saved to 'employee\_income\_clusters.xlsx'")

**Slips 7**

**Q.1. Fit the simple linear regression model to Salary\_positions.csv data. Predict the sa of level 11 and level 12 employees.**

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Load your dataset

data = pd.read\_csv('C:/Users/HRITIK/Downloads/file/Position\_Salaries.csv') # Ensure the file path is correct

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

print(data.head())

# Assume your dataset has columns 'Level' and 'Salary'

X = data[['Level']] # Independent variable (Position level)

y = data['Salary'] # Dependent variable (Salary)

# Initialize the linear regression model

model = LinearRegression()

# Fit the model

model.fit(X, y)

# Predict salary for level 11 and 12 using a DataFrame to avoid the warning

predictions = pd.DataFrame({'Level': [11, 12]})

predictions['Predicted\_Salary'] = model.predict(predictions[['Level']])

# Print predictions

print(f"Predicted salary for level 11: {predictions.iloc[0, 1]}")

print(f"Predicted salary for level 12: {predictions.iloc[1, 1]}")

# Append the predictions to the original dataset

data\_with\_predictions = pd.concat([data, predictions], ignore\_index=True)

# Save the data with predictions into an Excel file

data\_with\_predictions.to\_excel('Salary\_predictions.xlsx', index=False)

print("Predictions saved to 'C:/Users/HRITIK/Downloads/file/Position\_Salaries.csv'")

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

import pandas as pd

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

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

# Create a synthetic Weather Forecast Dataset

# Create the dataset

data = {

'Temperature': [30, 25, 20, 22, 28, 32, 35, 18, 24, 27], # Temperature in Celsius

'Humidity': [70, 85, 90, 80, 75, 65, 60, 95, 88, 76], # Humidity percentage

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

'Rain': ['Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No'] # Rain forecast (Target variable)

}

# Convert the dataset into a pandas DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to an Excel file

file\_path = 'C:/Users/HRITIK/Downloads/file/Weather\_Forecast\_Dataset.xlsx'

df.to\_excel(file\_path, index=False)

print(f"Dataset saved to '{file\_path}'")

# Convert categorical 'Wind' into numeric using Label Encoding

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

# Convert 'Rain' into binary labels (0 = No, 1 = Yes)

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

# Features (X) and Target (y)

X = df[['Temperature', 'Humidity', 'Wind']] # Features

y = df['Rain'] # Target

# Split the dataset into training and testing sets

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

# Initialize the Naive Bayes model

nb = GaussianNB()

# Train the Naive Bayes model

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

# Make predictions on the test set

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

# Evaluate the model

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

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

# Display the model's predictions

print("\nPredictions on the test set:")

print(pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred}))

**Slips 8**

**Q.1. Write a python program to categorize the given news text into one of the available 20 categories of news groups, using multinomial Naïve Bayes machine learning model. [15 M]**

import pandas as pd

from sklearn.datasets import fetch\_20newsgroups

from sklearn.feature\_extraction.text import TfidfVectorizer

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

from sklearn.naive\_bayes import MultinomialNB

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

# Step 1: Load the 20 newsgroups dataset

newsgroups = fetch\_20newsgroups(subset='all', remove=('headers', 'footers', 'quotes'))

# Step 2: Create a DataFrame from the dataset

df = pd.DataFrame({

'text': newsgroups.data,

'category': [newsgroups.target\_names[i] for i in newsgroups.target]

})

# Step 3: Preprocess the text data using TF-IDF vectorization

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

X = vectorizer.fit\_transform(df['text'])

y = df['category']

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

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

# Step 5: Train a Multinomial Naive Bayes model

model = MultinomialNB()

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

# Step 6: Make predictions on the test set

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

# Step 7: Evaluate the model's accuracy

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

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

# Step 8: Show detailed classification report

print("\nClassification Report:")

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

# Step 9: Make predictions on a new news text (example)

new\_text = [

"NASA's new rover is set to launch to explore Mars and gather valuable scientific data.",

"The stock market saw a major increase today as investors remain optimistic about the economy."

]

# Transform new text using the same vectorizer

new\_text\_transformed = vectorizer.transform(new\_text)

# Predict the categories of the new text

predictions = model.predict(new\_text\_transformed)

print("\nPredictions for new text:")

for text, category in zip(new\_text, predictions):

print(f"Text: {text[:50]}... -> Predicted Category: {category}")

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

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

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

from sklearn import metrics

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

# Step 1: Create a dataset

data = {

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

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

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

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

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

}

# Step 2: Create a DataFrame

df = pd.DataFrame(data)

# Step 3: Convert categorical data into numeric using label encoding

df\_encoded = df.copy()

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

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

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

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

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

# Step 4: Define features (X) and target (y)

X = df\_encoded[['Outlook', 'Temperature', 'Humidity', 'Wind']] # Features

y = df\_encoded['PlayTennis'] # Target

# Step 5: Split the data into training and testing sets

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

# Step 6: Create and train the Decision Tree model

clf = DecisionTreeClassifier(criterion='entropy') # Using entropy to create the decision tree

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

# Step 7: Make predictions on the test set

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

# Step 8: Evaluate the model's performance

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

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

# Step 9: Show the classification report

print("\nClassification Report:")

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

# Step 10: Visualize the Decision Tree

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

plot\_tree(clf, feature\_names=X.columns, class\_names=['No', 'Yes'], filled=True, rounded=True, fontsize=12)

plt.show()

**Slip No 9**

**Q.1. Implement Ridge Regression and Lasso regression model using boston\_houses.csv and take only ‘RM’ and ‘Price’ of the houses. Divide the data as training and testing data. Fit line using Ridge regression and to find price of a house if it contains 5 rooms and compare results. [15 M]**

import pandas as pd

import numpy as np

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

from sklearn.linear\_model import Ridge, Lasso

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Step 1: Generate a sample dataset resembling Boston housing data with 'RM' and 'Price' columns

np.random.seed(0)

rooms = np.random.normal(6, 1, 100) # Average number of rooms is around 6

price = 20000 \* rooms + np.random.normal(0, 10000, 100) # Price is correlated with the number of rooms

# Create DataFrame

data = pd.DataFrame({'RM': rooms, 'Price': price})

# Save dataset to Excel file

data.to\_excel("C:/Users/HRITIK/Downloads/file/boston\_houses\_sample.xlsx", index=False)

# Step 2: Split the data 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=0)

# Step 3: Implement Ridge and Lasso regression

ridge\_model = Ridge(alpha=1.0)

lasso\_model = Lasso(alpha=0.1)

# Fit the models

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

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

# Predict the price for a house with 5 rooms using both models (use DataFrame for input)

num\_rooms = pd.DataFrame([[5]], columns=['RM'])

ridge\_price = ridge\_model.predict(num\_rooms)

lasso\_price = lasso\_model.predict(num\_rooms)

# Print results

print(f"Predicted Price (Ridge) for a house with 5 rooms: {ridge\_price[0]}")

print(f"Predicted Price (Lasso) for a house with 5 rooms: {lasso\_price[0]}")

# Print results

print(f"Predicted Price (Ridge) for a house with 5 rooms: {ridge\_price[0]}")

print(f"Predicted Price (Lasso) for a house with 5 rooms: {lasso\_price[0]}")

# Step 5: Evaluate model performance on the test data

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

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

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

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

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

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

# Plotting the results

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

plt.plot(X\_test, ridge\_test\_pred, color='red', label="Ridge Regression Fit")

plt.plot(X\_test, lasso\_test\_pred, color='green', label="Lasso Regression Fit")

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

plt.ylabel("House Price")

plt.legend()

plt.show()

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

# Import required libraries

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

# Load dataset

data = pd.read\_csv("C:/Users/HRITIK/Downloads/file/UniversalBank.csv")

# Define feature variables (X) and target variable (y)

X = data.drop(columns=["Personal\_Loan"])

y = data["Personal\_Loan"]

# 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)

# Standardize the features for SVM

scaler = StandardScaler()

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

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

# Train the Linear SVM model

model = SVC(kernel="linear", 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)

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

print("Accuracy:", accuracy)

print("Classification Report:\n", report)

**Slips 10**

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

# Importing necessary libraries

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.decomposition import PCA

# Load the Iris dataset

iris = load\_iris()

data = iris.data

target = iris.target

# Convert the dataset into a DataFrame for easier manipulation

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

df['target'] = target

# Save the dataset to an Excel file

df.to\_excel('/mnt/data/iris\_dataset.xlsx', index=False)

print("Iris dataset saved as Excel file.")

# Apply PCA to reduce dimensions

pca = PCA(n\_components=2) # Reducing to 2 components for visualization

principal\_components = pca.fit\_transform(data)

# Creating a DataFrame for the PCA results

pca\_df = pd.DataFrame(data=principal\_components, columns=['Principal Component 1', 'Principal Component 2'])

pca\_df['target'] = target

# Save the PCA-transformed data to an Excel file

pca\_df.to\_excel('/mnt/data/iris\_pca\_transformed.xlsx', index=False)

print("PCA-transformed Iris dataset saved as Excel file.")

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

[**https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data**](https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data)

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset from a URL

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

iris = pd.read\_csv(url, header=None, names=column\_names)

# Convert the categorical 'species' column to numeric values

label\_encoder = LabelEncoder()

iris['species'] = label\_encoder.fit\_transform(iris['species'])

# Display the first few rows of the dataset

print(iris.head())

# Plotting a scatter plot for Sepal Length vs Sepal Width with color based on species

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

scatter = plt.scatter(iris['sepal\_length'], iris['sepal\_width'], c=iris['species'], cmap='viridis')

plt.colorbar(scatter, label='Species')

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

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

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

plt.show()

**slips 11**

**Q.1. Write a python program to implement Polynomial Regression for Boston Housing Dataset.**

import pandas as pd

from sklearn.datasets import fetch\_california\_housing

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

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Load the California Housing Dataset

california\_housing = fetch\_california\_housing()

X = pd.DataFrame(california\_housing.data, columns=california\_housing.feature\_names)

y = pd.Series(california\_housing.target, name='Target')

# Save the dataset to an Excel file

dataset = pd.concat([X, y], axis=1)

dataset.to\_excel('C:/Users/HRITIK/Downloads/file/Boston\_Housing.xlsx', index=False)

# 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 Regression

poly = PolynomialFeatures(degree=2)

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

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

# Train the model

model = LinearRegression()

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

# Predict and evaluate

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

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

print("Mean Squared Error:", mse)

# Plot predictions vs actual values for visual assessment

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

plt.xlabel("Actual Prices")

plt.ylabel("Predicted Prices")

plt.title("Polynomial Regression: Predicted vs Actual Prices")

plt.show()

**Q.2. Write a python program to Implement Decision Tree classifier model on Data which is extracted from images that were taken from genuine and forged banknote-like specimens. (refer UCI dataset https://archive.ics.uci.edu/dataset/267/banknote+authentication)**

**Slips 12**

**Q.1. Write a python program to implement k-nearest Neighbors ML algorithm to build prediction model (Use iris Dataset). # Import necessary libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Load the Iris dataset from UCI repository

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

iris = pd.read\_csv(url, header=None, names=column\_names)

# Display the first few rows of the dataset

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

print(iris.head())

# Split the dataset into features (X) and target (y)

X = iris.drop('species', axis=1) # Features

y = iris['species'] # Target

# Encode target labels (species) to numeric values

le = LabelEncoder()

y\_encoded = le.fit\_transform(y)

# Split the data into training and testing sets

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

# Initialize the KNN classifier

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the model on the training data

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

# Predict on the test data

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

# Calculate the accuracy of the model

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

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

# Visualize the results

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

# Use the encoded predictions for color mapping

plt.scatter(X\_test.iloc[:, 0], X\_test.iloc[:, 1], c=y\_pred, cmap='viridis', edgecolor='k', s=100)

plt.title("KNN Prediction on Iris Dataset")

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.colorbar(label='Predicted Species')

plt.show()

**Q.2. Fit the simple linear regression and polynomial linear regression models to Salary\_positions.csv data. Find which one is more accurately fitting to the given data. Also predict the salaries of level 11 and level 12 employees.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures

from sklearn.metrics import mean\_squared\_error

# Load the dataset

data = pd.read\_csv('C:/Users/HRITIK/Downloads/file/Salary\_positions.csv')

# Simple Linear Regression

X = data['Level'].values.reshape(-1, 1)

y = data['Salary'].values

# Fit Simple Linear Regression model

linear\_regressor = LinearRegression()

linear\_regressor.fit(X, y)

# Predict using the simple linear regression model

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

# Polynomial Linear Regression (degree 2)

poly = PolynomialFeatures(degree=2)

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

# Fit Polynomial Linear Regression model

poly\_regressor = LinearRegression()

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

# Predict using the polynomial regression model

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

# Plotting the results

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

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

plt.plot(X, y\_pred\_poly, color='green', label='Polynomial Regression (Degree 2)')

plt.title('Salary vs Job Level')

plt.xlabel('Job Level')

plt.ylabel('Salary')

plt.legend()

plt.show()

# Evaluate the models

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

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

print(f"Mean Squared Error of Linear Regression: {mse\_linear}")

print(f"Mean Squared Error of Polynomial Regression: {mse\_poly}")

# Predict the salary for levels 11 and 12 using both models

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

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

# Linear prediction

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

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

# Polynomial prediction

level\_11\_poly = poly.transform(level\_11)

level\_12\_poly = poly.transform(level\_12)

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

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

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

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

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

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

**Slips 13**

**Q.1. Create RNN model and analyze the Google stock price dataset. Find out increasing or decreasing trends of stock price for the next day.**

!pip install yfinance

2.

import yfinance as yf

import pandas as pd

# Download Google stock price data (Alphabet Inc.)

stock\_data = yf.download("GOOGL", start="2015-01-01", end="2024-01-01")

# Save the dataset as CSV

stock\_data.to\_csv('C:/Users/HRITIK/Downloads/file/google\_stock\_data.csv')

# Show the first few rows

stock\_data.head()

3.

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

import numpy as np

# Load the dataset

data = pd.read\_csv('C:/Users/HRITIK/Downloads/file/google\_stock\_data.csv')

# Check the first few rows to make sure the data is correct

print(data.head())

# Remove any rows with non-numeric data, such as the column labels

# We need to ensure that the 'Close' column contains only numeric values

# Convert the 'Close' column to numeric, forcing errors to NaN (this will help identify invalid data)

data['Close'] = pd.to\_numeric(data['Close'], errors='coerce')

# Drop any rows with NaN values (which may have come from invalid data)

data = data.dropna(subset=['Close'])

# Now that we have cleaned the data, let's work with the 'Close' column

close\_prices = data['Close'].values.reshape(-1, 1)

# Initialize the scaler

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

# Normalize the data (ensure we're only normalizing numerical data)

scaled\_data = scaler.fit\_transform(close\_prices)

# Create a function to prepare the dataset for the RNN model

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

x\_data, y\_data = [], []

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

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

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

return np.array(x\_data), np.array(y\_data)

# Create datasets for training

time\_step = 60

x\_data, y\_data = create\_dataset(scaled\_data, time\_step)

# Reshape the input data for RNN

x\_data = np.reshape(x\_data, (x\_data.shape[0], x\_data.shape[1], 1))

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

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

x\_train, x\_test = x\_data[:train\_size], x\_data[train\_size:]

y\_train, y\_test = y\_data[:train\_size], y\_data[train\_size:]

# Output the shapes to verify everything is correct

print(f"Training data shape: {x\_train.shape}")

print(f"Test data shape: {x\_test.shape}")

!pip install tensorflow

from tensorflow.keras.models import Sequential

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

from keras.models import Sequential

from keras.layers import SimpleRNN, Dense, Dropout

# Build the RNN model

model = Sequential()

# Add RNN layers

model.add(SimpleRNN(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1)))

model.add(Dropout(0.2))

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

model.add(Dropout(0.2))

# Add output layer

model.add(Dense(units=1))

# Compile the model

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

# Train the model

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

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

# Importing necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Load the dataset

url = 'C:/Users/HRITIK/Downloads/file/Housing.csv'

data = pd.read\_csv(url)

# Inspect the first few rows

print(data.head())

# Assuming the dataset has 'Size' (square feet) and 'Price' columns

# If not, modify accordingly

X = data[['Size']] # Feature: Size of the house

y = data['Price'] # Target: House Price

# 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)

# Calculate the mean squared error

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

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

# Plotting the regression line

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

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

plt.xlabel('Size of the house (Square feet)')

plt.ylabel('Price of the house')

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

plt.legend()

plt.show()

# Output the coefficients

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

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

**Slips 14**

**Q.1. Create a CNN model and train it on mnist handwritten digit dataset. Using model find out the digit written by a hand in a given image.**

**Import mnist dataset from tensorflow.keras.datasets.**

import tensorflow as tf

import pandas as pd

import numpy as np

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

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

import matplotlib.pyplot as plt

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

import pandas as pd

# Create a sample dataset with null values

data = {

'Name': ['Alice', 'Bob', 'Charlie', None, 'Eve'],

'Age': [24, 27, None, 22, 29],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', None]

}

# Convert to a DataFrame

df = pd.DataFrame(data)

# Display the original dataset

print("Original Dataset:")

print(df)

# Find and remove null values

cleaned\_df = df.dropna()

# Display the cleaned dataset

print("\nCleaned Dataset (without null values):")

print(cleaned\_df)

# Save the cleaned dataset to a CSV file

cleaned\_df.to\_csv('cleaned\_data.csv', index=False)

**Slip 15**

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

# Import necessary libraries

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Step 1: Load dataset

url = "C:/Users/HRITIK/Downloads/file/housing.csv"

columns = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']

data = pd.read\_csv(url, header=None, names=columns)

# Step 2: Preprocessing the data

# Create target variable 'Above\_Avg' where price above average is 1 and below average is 0

mean\_price = data['MEDV'].mean()

data['Above\_Avg'] = (data['MEDV'] > mean\_price).astype(int)

# Select features (exclude the target and other non-features)

features = data.drop(columns=['MEDV', 'Above\_Avg'])

target = data['Above\_Avg']

# Step 3: Split the data into training and test sets

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

# Step 4: Feature scaling

scaler = StandardScaler()

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

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

# Step 5: Build the ANN model

model = Sequential()

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

model.add(Dense(units=32, activation='relu'))

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

# Step 6: Compile the model

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

# Step 7: Train the model

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

# Step 8: Evaluate the model

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

print(f"Test accuracy: {accuracy \* 100:.2f}%")

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

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import LabelEncoder

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

# Load the dataset

data = pd.read\_csv('C:/Users/HRITIK/Downloads/file/house\_price\_data.csv')

# Check the column names

print("Columns in the dataset:", data.columns)

# If 'Size' or 'Square\_Feet' isn't found, adjust to the correct column name

if 'Size' not in data.columns:

if 'Square\_Feet' in data.columns:

print("Using 'Size (sq ft)' as the size column.")

size\_column = 'Size (sq ft)' # Correct column name

else:

raise KeyError("Neither 'Size' nor 'Square\_Feet' columns found. Check the dataset.")

else:

size\_column = 'Size (sq ft)'

# Encoding the categorical feature 'Location'

label\_encoder = LabelEncoder()

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

# Features (independent variables) and target (dependent variable)

X = data[['Bedrooms', 'Size (sq ft)', 'Location']] # Adjusted column names

y = data['Price'] # Target

# Split the data into training and test sets

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

# Initialize and train the Multiple Linear Regression model

model = LinearRegression()

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

# Make predictions using the test data

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

# Evaluate the model

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

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

# Display the results

print("Mean Squared Error:", mse)

print("R-squared:", r2)

# Visualizing actual vs predicted prices

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

plt.xlabel('Actual Prices')

plt.ylabel('Predicted Prices')

plt.title('Actual vs Predicted Prices')

plt.show()

# Show coefficients of the model

print("Coefficients:", model.coef\_)

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

Slip 16

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

pip install numpy pandas scikit-learn tensorflow

# Importing necessary libraries

import numpy as np

import pandas as pd

from sklearn.datasets import make\_classification

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

# Step 1: Create a synthetic classification dataset

# This will create a dataset with 1000 samples, 20 features, and a binary target (0 or 1)

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=10, n\_redundant=5, n\_classes=2, random\_state=42)

# Convert the dataset into a pandas DataFrame and save it to a CSV file

data = pd.DataFrame(X, columns=[f'feature\_{i}' for i in range(1, 21)])

data['target'] = y

# Save to a CSV file

data.to\_csv('C:/Users/HRITIK/Downloads/file/syntheticdata1000-privateleaderboard-2024-11-13T15\_35\_24.csv', index=False)

# Step 2: Build a Two-Layer Neural Network

# Initialize a Sequential model

model = Sequential()

# Add the first hidden layer with 64 neurons, ReLU activation function

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

# Add the second layer (output layer) with 1 neuron and Sigmoid activation

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

# Compile the model using Adam optimizer and binary cross-entropy loss

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

# Step 3: Train the model (optional, can be skipped for just the architecture)

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

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

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

# Train the model on the dataset

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

# Save the trained model (optional)

model.save('binary\_classification\_model.h5')

# Output message indicating that the dataset is ready

print("Dataset has been saved as 'syntheticdata1000-privateleaderboard-2024-11-13T15\_35\_24.csv'.")

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

# Importing necessary libraries

import pandas as pd

from sklearn.datasets import fetch\_california\_housing

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

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

# Load the California housing dataset

california\_housing = fetch\_california\_housing()

# Convert it into a pandas DataFrame

housing\_df = pd.DataFrame(california\_housing.data, columns=california\_housing.feature\_names)

housing\_df['PRICE'] = california\_housing.target # Add the target variable (house prices)

# Save the dataset as a CSV file

housing\_df.to\_csv('C:/Users/HRITIK/Downloads/file/Boston\_Housing.csv', index=False)

# Display the first few rows of the dataset

print(housing\_df.head())

# Simple Linear Regression: Using 'AveRooms' (average number of rooms per household) as the independent variable

X = housing\_df[['AveRooms']] # Independent variable

y = housing\_df['PRICE'] # Target variable

# 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)

# Initialize the model

model = LinearRegression()

# Train the model

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

# Predict using the test set

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

# Print the results

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

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

# Evaluate the model

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

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

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

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

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

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

# Plotting the results

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

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

plt.xlabel('Average number of rooms per household (AveRooms)')

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

plt.legend()

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

plt.show()

Slips 17

**Q.1. Implement Ensemble ML algorithm on Pima Indians Diabetes Database with bagging (random forest), boosting, voting and Stacking methods and display analysis accordingly. Compare result.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

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

from sklearn.linear\_model import LogisticRegression

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

# Load the dataset directly from URL

url = 'C:/Users/HRITIK/Downloads/file/pima-indians-diabetes.data.csv'

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

data = pd.read\_csv(url, header=None, names=columns)

# Check for missing values or incorrect data types

print(data.info())

# Check for any non-numeric values in the dataset

print(data.isna().sum()) # Check for missing values

print(data.dtypes) # Check the data types

# If there are missing values, handle them (e.g., imputation or removal)

data = data.dropna() # Remove rows with missing values (or use imputation if needed)

# Check that 'Outcome' column is binary

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

# Split the dataset into features and target variable

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

y = data['Outcome']

# Ensure that the data is numeric

X = X.apply(pd.to\_numeric, errors='coerce')

# Check if there are any NaNs after coercion

print(X.isna().sum())

# Drop rows with NaN values after coercion

X = X.dropna()

y = y[X.index] # Make sure that the target variable matches the feature set after dropping NaNs

# 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)

# Function to evaluate models

def evaluate\_model(model):

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

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

print(f'Accuracy: {accuracy\_score(y\_test, y\_pred)}')

print('Confusion Matrix:')

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

print('Classification Report:')

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

# 1. Bagging (Random Forest)

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

print("Random Forest (Bagging) Performance:")

evaluate\_model(rf\_model)

# 2. Boosting (Gradient Boosting)

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

print("\nGradient Boosting Performance:")

evaluate\_model(gb\_model)

# 3. Voting Classifier

voting\_model = VotingClassifier(estimators=[

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

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

('lr', LogisticRegression())

], voting='hard')

print("\nVoting Classifier Performance:")

evaluate\_model(voting\_model)

# 4. Stacking Classifier

stacking\_model = StackingClassifier(estimators=[

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

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

], final\_estimator=LogisticRegression())

print("\nStacking Classifier Performance:")

evaluate\_model(stacking\_model)

# Plotting the accuracy for comparison

models = ['Random Forest', 'Gradient Boosting', 'Voting Classifier', 'Stacking Classifier']

accuracies = [

accuracy\_score(y\_test, rf\_model.predict(X\_test)),

accuracy\_score(y\_test, gb\_model.predict(X\_test)),

accuracy\_score(y\_test, voting\_model.predict(X\_test)),

accuracy\_score(y\_test, stacking\_model.predict(X\_test))

]

plt.bar(models, accuracies)

plt.xlabel('Model')

plt.ylabel('Accuracy')

plt.title('Model Comparison')

plt.show()

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

# Load the dataset

df = pd.read\_csv('C:/Users/HRITIK/Downloads/file/dataa.csv') # Replace 'your\_dataset.csv' with the actual file path

# Selecting relevant columns for regression

# Assume 'SalePrice' is the target variable, and 'OverallQual', 'GrLivArea', etc., are predictors

X = df[['CRIM', 'INDUS', 'TAX', 'LSTAT']]

y = df['MEDV']

# Splitting the dataset

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

# Training the model

model = LinearRegression()

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

# Making predictions

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

# Evaluating the model

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

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

Slips 18

**Q.1. Implement Ensemble ML algorithm on Pima Indians Diabetes Database with bagging (random forest), boosting, voting and Stacking methods and display analysis accordingly. Compare result.**

import 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.tree import DecisionTreeClassifier

from sklearn.svm import SVC

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

from sklearn.datasets import load\_diabetes

# Load Dataset

url = 'C:/Users/HRITIK/Downloads/file/diabetes.csv'

data = pd.read\_csv(url)

# Split dataset into features and target variable

X = data.drop(columns=['Outcome'])

y = data['Outcome']

# Split 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)

# Random Forest Classifier

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

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

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

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

# AdaBoost Classifier with SAMME algorithm

ada = AdaBoostClassifier(n\_estimators=100, random\_state=42, algorithm='SAMME')

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

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

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

# Gradient Boosting Classifier

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

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

gb\_pred = gb.predict(X\_test)

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

# Voting Classifier

voting\_clf = VotingClassifier(

estimators=[('rf', rf), ('ada', ada), ('gb', gb)], voting='soft')

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

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

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

# Stacking Classifier

estimators = [

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

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

]

stacking\_clf = StackingClassifier(

estimators=estimators, final\_estimator=LogisticRegression()

)

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

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

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

from sklearn.metrics import classification\_report

# Display detailed results for each model

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

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

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

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

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