**Assignment 1-Create a multiple linear regression model for house price dataset divide dataset into train and test data while giving it to model and predict prices of house.**

 # Import 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

from sklearn.impute import SimpleImputer

# Load the dataset

file\_path = 'Housing\_2.csv'

data = pd.read\_csv(file\_path)

# Explore the dataset

dataset\_info = data.info()

dataset\_head = data.head()

dataset\_description = data.describe()

# Handle missing values by imputing with the mean

imputer = SimpleImputer(strategy='mean')

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

# Preprocess the data

target\_variable = 'price'

X = data\_imputed.drop(columns=[target\_variable])

y = data\_imputed[target\_variable]

# Check for categorical variables and apply encoding if necessary

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

# 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)

# Train the Multiple Linear Regression model

model = LinearRegression()

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

# Evaluate the model

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

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

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

# Predictions

predictions = y\_pred[:5]

actual\_values = y\_test.values[:5]

print(dataset\_info)

print(dataset\_head)

print(dataset\_description)

print("Mean Squared Error ",mse)

print(r2)

print(predictions)

print(actual\_values)

**Assignment4-Write a python program to categorize the given news text into one of the available 20 categories of news groups, using multinomial Naïve Bayes machine learning model.**

 # Import necessary libraries

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

# Load the 20 Newsgroups dataset

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

# Display the available categories

print("Categories:")

print(newsgroups.target\_names)

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

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

# Convert the text data to TF-IDF features

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

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

# Initialize the Multinomial Naive Bayes classifier

model = MultinomialNB()

# Train the model

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

# Make predictions on the test data

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

# Evaluate the model's performance

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

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

# Print detailed classification report

print("\nClassification Report:")

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

**Assignment6-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)**

 pip install pandas scikit-learn ucimlrepo

import pandas as pd

from ucimlrepo import fetch\_ucirepo

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

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

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

# Step 1: Load the dataset from UCI repository

banknote\_authentication = fetch\_ucirepo(id=267)

data = banknote\_authentication.data.features

target = banknote\_authentication.data.targets

# Step 2: Create a DataFrame

df = pd.DataFrame(data)

df['class'] = target

# Step 3: Split the dataset into features and target variable

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

y = df['class']      # Target variable (last column)

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

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

# Step 5: Create and train the Decision Tree classifier

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

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

# Step 6: Make predictions on the test set

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

tree.plot\_tree(dt\_classifier)

# Step 7: Evaluate the model's performance

print("Confusion Matrix:")

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

print("\nClassification Report:")

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

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

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

**Assignment 13: Use Apriori algorithm on groceries dataset to find which items are brought together. Use minimum support =0.25**

import pandas as pd

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

# Step 1: Load the dataset

# Assume the dataset is in a transaction format where each row is a list of items bought together

# For simplicity, let's create a small example dataset in code

data = {

    'TransactionID': [1, 2, 3, 4, 5, 6],

    'Items': [

        ['milk', 'bread', 'butter'],

        ['bread', 'butter'],

        ['milk', 'bread'],

        ['butter', 'milk'],

        ['bread', 'butter', 'milk'],

        ['butter', 'milk']

    ]

}

# Converting the dataset into a DataFrame

df = pd.DataFrame(data)

# Step 2: Preprocess the data

# Convert the list of items per transaction into a one-hot encoded DataFrame

df\_expanded = df['Items'].str.join('|').str.get\_dummies()

# Step 3: Apply Apriori Algorithm

# Set minimum support to 0.25

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

# Step 4: Generate association rules

rules = association\_rules(frequent\_itemsets, metric="support", min\_threshold=0.25)

# Step 5: Print the frequent itemsets and association rules

print("Frequent Itemsets:")

print(frequent\_itemsets)

print("\nAssociation Rules:")

print(rules)

# Optional: Visualize the rules

import matplotlib.pyplot as plt

import seaborn as sns

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

sns.scatterplot(x='support', y='confidence', size='lift', data=rules, legend=False, sizes=(20, 200))

plt.title('Support vs Confidence')

plt.xlabel('Support')

plt.ylabel('Confidence')

plt.show()

**Assignment 14: 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, VotingClassifier, StackingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

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

from sklearn.preprocessing import StandardScaler

# Load the diabetes dataset

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

# Prepare the data (features and target)

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

y = data['Outcome']

# Standardize features for better performance

scaler = StandardScaler()

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

# Split the data into training and testing sets

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

# Define models for ensemble methods

# 1. Bagging using Random Forest

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

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

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

# 2. Boosting using AdaBoost

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

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

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

# 3. Voting Classifier (hard voting)

log\_clf = LogisticRegression(random\_state=42)

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

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

voting\_clf = VotingClassifier(estimators=[

    ('lr', log\_clf),

    ('svc', svc\_clf),

    ('dt', dt\_clf)

], voting='hard')

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

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

# 4. Stacking Classifier

stacking\_clf = StackingClassifier(

    estimators=[

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

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

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

    ],

    final\_estimator=LogisticRegression()

)

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

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

# Function to calculate and display metrics

def evaluate\_model(y\_true, y\_pred, model\_name):

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

    precision = precision\_score(y\_true, y\_pred)

    recall = recall\_score(y\_true, y\_pred)

    f1 = f1\_score(y\_true, y\_pred)

    print(f"{model\_name}:\n Accuracy: {accuracy:.2f}, Precision: {precision:.2f}, Recall: {recall:.2f}, F1 Score: {f1:.2f}\n")

# Display the evaluation metrics for each model

evaluate\_model(y\_test, rf\_pred, 'Random Forest (Bagging)')

evaluate\_model(y\_test, ada\_pred, 'AdaBoost (Boosting)')

evaluate\_model(y\_test, voting\_pred, 'Voting Classifier')

evaluate\_model(y\_test, stacking\_pred, 'Stacking Classifier')

**Assignment 15: Create a two layered neural network with relu and sigmoid activation function.**

 pip install tensorflow

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Define the neural network

model = Sequential()

# Input layer and the first hidden layer with ReLU activation

model.add(Dense(units=64, activation='relu', input\_shape=(2,)))

# Output layer with Sigmoid activation

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

# Compile the model with a loss function, optimizer, and evaluation metric

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

# Print the model summary to see the architecture

model.summary()

**Assignment 16: Create an ANN and train it on house price dataset classify the house price is above average or below average.**

 import pandas as pd

import numpy as np

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.preprocessing import StandardScaler

# Step 1: Load the dataset (replace with the path to your dataset)

# Replace with the actual file path

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

# Step 2: Preprocess the data

# Assume the target variable is named 'Price' and we want to classify it as 'Above\_Average' or 'Below\_Average'

# Calculate the average price

average\_price = data['price'].mean()

# Create a new target column 'Price\_Category'

# 1 if the price is above average, 0 if below average

data['Price\_Category'] = np.where(data['price'] > average\_price, 1, 0)

# Drop the original 'Price' column

data = data.drop('price', axis=1)

# Handle categorical variables if any using one-hot encoding

# data = pd.get\_dummies(data)  # Uncomment this line if there are categorical variables

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

X = data.drop('Price\_Category', axis=1)

y = data['Price\_Category']

# 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)

# Standardize the feature data

scaler = StandardScaler()

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

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

# Step 3: Create and compile the ANN model

model = Sequential()

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

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

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

# Compile the model

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

# Step 4: Train the model

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

# Step 5: Evaluate the model

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

print(f"Test Loss: {loss:.4f}")

print(f"Test Accuracy: {accuracy:.4f}")

# Optional: Plot training history

import matplotlib.pyplot as plt

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Test'], loc='upper left')

plt.show()

**Assignment 17: 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 numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# Step 1: Load the MNIST dataset

from tensorflow.keras.datasets import mnist

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

# Step 2: Preprocess the data

# Reshape the data to include the channel dimension (28x28 images with 1 channel for grayscale)

X\_train = X\_train.reshape((X\_train.shape[0], 28, 28, 1))

X\_test = X\_test.reshape((X\_test.shape[0], 28, 28, 1))

# Normalize the data to the range [0, 1]

X\_train = X\_train.astype('float32') / 255.0

X\_test = X\_test.astype('float32') / 255.0

# Convert labels to one-hot encoding

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

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

# Step 3: Create the CNN model

model = Sequential()

# Add convolutional and pooling layers

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

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

model.add(MaxPooling2D(pool\_size=(2, 2)))

# Flatten the output of the convolutional layers

model.add(Flatten())

# Add dense layers (fully connected)

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

model.add(Dense(10, activation='softmax'))  # 10 output neurons for 10 digit classes

# Step 4: Compile the model

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

# Step 5: Train the model

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

# Step 6: Evaluate the model

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

print(f"Test Loss: {loss:.4f}")

print(f"Test Accuracy: {accuracy:.4f}")

# Optional: Plot training history

plt.plot(history.history['accuracy'], label='train accuracy')

plt.plot(history.history['val\_accuracy'], label='validation accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.title('Model Accuracy')

plt.show()

# Step 7: Predict a digit from a given image

def predict\_digit(image):

    """

    Takes a preprocessed image and returns the predicted digit.

    """

    image = image.reshape(1, 28, 28, 1)  # Reshape to match input shape

    image = image.astype('float32') / 255.0  # Normalize the image

    prediction = model.predict(image)

    return np.argmax(prediction)  # Returns the class with the highest probability

# Test the model with an example from the test set

example\_index = 0  # Change this index to test with different images

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

plt.title(f"True Label: {np.argmax(y\_test[example\_index])}")

plt.show()

predicted\_digit = predict\_digit(X\_test[example\_index])

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

**Assignment 18: Create RNN model and analyze the Google stock price dataset. Find out increasing or decreasing trends of stock price for the next day.**

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from tensorflow.keras.models import Sequential

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

import matplotlib.pyplot as plt

# Load the Google stock price dataset

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

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

# Normalize the stock prices

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

stock\_prices\_normalized = scaler.fit\_transform(stock\_prices)

# Prepare the dataset

X, y = [], []

for i in range(len(stock\_prices\_normalized) - 1):

    X.append(stock\_prices\_normalized[i])

    y.append(stock\_prices\_normalized[i + 1])

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

# Create the RNN model

model = Sequential()

model.add(SimpleRNN(50, activation='relu', input\_shape=(1, 1)))

model.add(Dropout(0.2))

model.add(Dense(1))

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

# Train the model

model.fit(X.reshape(-1, 1, 1), y, epochs=100, batch\_size=32)

# Predict the next day's stock price

predicted\_price = model.predict(stock\_prices\_normalized[-1].reshape(-1, 1, 1))

predicted\_price = scaler.inverse\_transform(predicted\_price)

if predicted\_price > stock\_prices[-1]:

    print("The stock price is predicted to increase for the next day.")

else:

    print("The stock price is predicted to decrease for the next day.")