1a) pandas library funs

```
import pandas as pd
# Create a sample DataFrame
data = {
'Name': ['John', 'Jane', 'Bob', 'Alice'],
'Age': [25, 30, 22, 28],
'City': ['New York', 'San Francisco', 'Los Angeles', 'Chicago']
}
df = pd.DataFrame(data)
# Export data to a CSV file
csv_filename = 'sample_data.csv'
df.to_csv(csv_filename, index=False)
print(f'Data exported to {csv_filename}')
# Import data from the CSV file
imported_df = pd.read_csv(csv_filename)
# Display the imported DataFrame
print('\nImported DataFrame:')
print(imported_df)
# Export data to an Excel file
excel_filename = 'sample_data.xlsx'
df.to_excel(excel_filename, index=False, sheet_name='Sheet1')
print(f'\nData exported to {excel_filename}')
# Import data from the Excel file
imported_df_excel = pd.read_excel(excel_filename, sheet_name='Sheet1')
# Display the imported DataFrame from Excel
print('\nImported DataFrame from Excel:')
print(imported_df_excel)
```

1B.& 6A.) Logistic Regression model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Load the Iris dataset
from sklearn.datasets import load_iris
iris_data = load_iris()
iris_df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
iris_df['target'] = iris_data.target
# Consider a binary classification problem (0 or 1)
iris_df['binary_target'] = (iris_df['target'] == 0).astype(int)
# Split the dataset into features (X) and binary target variable (y)
X = iris_df.drop(['target', 'binary_target'], axis=1)
y = iris_df['binary_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 logistic regression model
log_reg = LogisticRegression(random_state=42)
# Train the model
log_reg.fit(X_train, y_train)
# Make predictions on the test set
y_pred = log_reg.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
# Plot the confusion matrix
plt.figure(figsize=(6, 4))
```

```
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Not Class 0', 'Class 0'], yticklabels=['Not Class 0', 'Class 0'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# ROC Curve
from sklearn.metrics import roc_curve, auc, confusion_matrix
import matplotlib.pyplot as plt

# Get the predicted probabilities for class 1 (positive class)
y_pred_proba = log_reg.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate the Area Under the ROC Curve (AUC)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

2A&3A&13B) Data pre-processing (EDA)

```
#import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# Load the dataset
used_cars_df = pd.read_CSV('used_cars.CSV')
# Display the first few rows of the dataset
print(used_cars_df.head())
#Get the basic information about the dataset
print(used_cars_df.info())
# Summary stastistics of the numerical columns
print(used_cars_df.describe())
#Check for missing values
print(used_cars_df.isnull().sum())
#Check the distribution of the categorical variables
for column in used_cars_df.select_dtypes(include='object').columns:
   print(used_cars_df.[column].value_counts())
#Visualize the distribution of numerical features
sns.pairplot(used_cars_df)
plt.show()
#Correlation Matrix
correlation_matrix = used_cars_df.corr()
sns.heatmap(correlation_matris, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
#Visualize the distribution of categorical variables
for column in used_cars_df.select_dtypes(include='object').columns:
   sns.countplot(x=column, data=used_cars_df)
   plt.title(fDistribution of {column}')
```

plt.xticks(rotation=45)

plt.show()

3B&11B) Naïve byes theorem

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Sample text data
data = {'text': ["I love programming", "Machine learning is fascinating", "Spam emails are annoying",
"Python is a great language", "Buy our new product now"]}
labels = [1, 1, 0, 1, 0] # 1 for positive, 0 for negative
# Create a DataFrame
df = pd.DataFrame({'text': data['text'], 'label': labels})
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2,
random_state=42)
# Vectorize the text data using CountVectorizer
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
# Initialize the Naive Bayes model (Multinomial Naive Bayes for text data)
naive_bayes = MultinomialNB()
# Train the model
naive_bayes.fit(X_train_vectorized, y_train)
# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_vectorized)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

3B&11B DATA VISUALIZATION

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
iris = sns.load_dataset('Iris')
plt.figure(figsize=(10,6))
sns.scaterplot(x='sepal_length', y='sepal_width', hue='species', data=iris)
plt.title('Scatter Plot of Sepal length Vs Sepal Width')
plt.show()
sns.pairplot(iris, hue='species', height=2.5)
plt.subtitle('Pairwise Relationships and Distributions')
plt.show()
plt.figure(figsize=(15,8))
for i, feature in enumerate(iris.columns[:-1]):
plt.subplot(2,2, i+1)
sns.boxplot(x='species', y=feature, data=iris)
plt.title(fBoxplot of {feature}')
plt.tight_layout()
plt.show()
plt.figure(figsize=(15,8))
for i, feature in enumerate(iris.columns[:-1]):
plt.subplot(2,2, i+1)
sns.violivplot(x='species', y=feature, data=iris)
plt.title(fViolinplot of {feature}')
plt.tight_layout()
plt.show()
correlation_matrix = iris.corr(()
```

sns.heatmap(correlation_matris, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

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

sns.histplot(iris['sepal_length'], kde=True)

plt.title('Distribution of Sepal Length')

plt.show()

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

sns.countplot(x='species', data=iris)

plt.title('Count Plot of Species')

plt.show()

4A) DECISION TREE CLASS

import pandas as pd

import numpy as np

df=pd.read_csv('PlayTennis.csv')

from sklearn.preprocessing import LabelEncoder

enc = LabelEncoder()

df_num_cat=pd.DataFrame()

df_num_cat['Outlook']= enc.fit_transform(df['Outlook'])

df_num_cat['Temperature']= enc.fit_transform(df['Temperature'])

df_num_cat['Humidity']= enc.fit_transform(df['Humidity'])

df_num_cat['Wind']= enc.fit_transform(df['Wind'])

df_num_cat['Play Tennis']= enc.fit_transform(df['Play Tennis'])

df_num_cat

x=df_num_cat.drop(['Play Tennis'],axis=1)

y=df_num_cat['Play Tennis']

Split dataset into training set and test set

from sklearn.model_selection import train_test_split # Import train_test_split function

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1) #
70% training

and 30% test

```
print(X_train)

print(X_test)

print(y_train)

print(y_test)

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

df_clf=DecisionTreeClassifier(criterion='entropy')

#df_clf=DecisionTreeClassifier()

df_clf.fit(X_train,y_train)

y_pred=df_clf.predict(X_test)

print(y_train)

print(y_test)

print(y_pred)

print("Accuracy")

print(metrics.accuracy_score(y_test,y_pred))

# Plot the decision tree

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

plot_tree(df_clf, filled=True, feature_names=X_train.columns, class_names=df['Play
Tennis'].unique())

plt.show()
```

4B) NUMPY PROG

```
import numpy as np

# Define two vectors

vector1 = np.array([1, 2, 3])

vector2 = np.array([4, 5, 6])

# Element-wise multiplication

result = vector1 * vector2

# Display the result

print("Vector 1:", vector1)

print("Vector 2:", vector2)

print("Element-wise Multiplication:", result)
```

5A)SIMPLE & MULTI LINEARS

```
import numpy as np
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
# Sample data: one feature (e.g., study hours) and one target (e.g., marks)
X = np.array([[1], [2], [3], [4], [5]])  # Feature
y = np.array([2, 4, 5, 4, 5])         # Target
# Create and train the model
model = LinearRegression()
model.fit(X, y)
# Predict
y_pred = model.predict(X)
# Print coefficients
print("Simple Linear Regression:")
print("Intercept:", model.intercept_)
print("Slope:", model.coef_)
# Plotting
plt.scatter(X, y, color='blue')
plt.plot(X, y_pred, color='red')
plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.show()
```

MULTIPLE :

```
import numpy as np
from sklearn.linear_model import LinearRegression
# Sample data: multiple features (e.g., study hours and sleep hours)
X = np.array([[1, 7], [2, 8], [3, 8], [4, 6], [5, 5]])  # Features
y = np.array([50, 60, 65, 70, 75])              # Target
# Create and train the model
model = LinearRegression()
```

```
model.fit(X, y)
# Predict
y_pred = model.predict(X)
# Print coefficients
print("\nMultiple Linear Regression:")
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
# Display predicted values
print("Predicted y:", y_pred)
```

5B) PANDAS PROG TO ADD,SUBTRACT,MULTIPLY AND DIVIDE 2 PANDAS SERIES

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 9]) ds = ds1 + ds2
print("Add two Series:") print(ds)
print("Subtract two Series:") ds = ds1 - ds2
print(ds)
print("Multiply two Series:") ds = ds1 * ds2
print(ds)
print("Divide Series1 by Series2:") ds = ds1 / ds2
print(ds)
```

2B&7A NAÏVE BAYES

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Sample text data
data = {'text': ["I love programming", "Machine learning is fascinating", "Spam emails are annoying",
"Python is a great language", "Buy our new product now"]}
labels = [1, 1, 0, 1, 0] # 1 for positive, 0 for negative
```

```
# Create a DataFrame
df = pd.DataFrame({'text': data['text'], 'label': labels})
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2,
random_state=42)
# Vectorize the text data using CountVectorizer
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
# Initialize the Naive Bayes model (Multinomial Naive Bayes for text data)
naive_bayes = MultinomialNB()
# Train the model
naive_bayes.fit(X_train_vectorized, y_train)
# Make predictions on the test set
y_pred = naive_bayes.predict(X_test_vectorized)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

7B) FRIDAY

```
# Given probabilities
P_Friday_and_Absent = 0.03
P_Friday = 0.20
# Bayes' Theorem: P(Absent | Friday) = P(Friday and Absent) / P(Friday)
P_Absent_given_Friday = P_Friday_and_Absent / P_Friday
# Print the result
print("Probability that a student is absent given that today is Friday:")
print(f"P(Absent | Friday) = {P_Absent_given_Friday:.2f}")
```

8A) K-NEAREST NEIGHBOURS

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load the Iris dataset
from sklearn.datasets import load_iris
iris_data = load_iris()
iris_df = pd.DataFrame(iris_data.data, columns=iris_data.feature_names)
iris_df['target'] = iris_data.target
# Split the dataset into features (X) and target variable (y)
X = iris_df.drop('target', axis=1)
y = iris_df['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 K-Nearest Neighbors classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
# Train the model
knn_classifier.fit(X_train, y_train)
# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

8B) RESHAPING

```
import numpy as np
import pandas as pd
```

```
# -------------------------------
# (i) Reshaping the Data
# -------------------------------
# Create a 1D NumPy array
data = np.array([1, 2, 3, 4, 5, 6])
# Reshape it into 2D (2 rows, 3 columns)
reshaped_data = data.reshape((2, 3))
print("Original Data (1D):")
print(data)
print("\nReshaped Data (2D):")
print(reshaped_data)
# -------------------------------
# (ii) Filtering the Data
# -------------------------------
# Create a Pandas DataFrame for filtering
df = pd.DataFrame({
   'Student': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'Marks': [85, 40, 95, 35, 75]
})
# Filter students who scored more than 50
filtered_df = df[df['Marks'] > 50]
print("\nOriginal DataFrame:")
print(df)
print("\nFiltered Data (Marks > 50):")
print(filtered_df)
```

9A&9B) K- MEANS

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
```

```
# Create a synthetic dataset
data, true_labels = make_blobs(n_samples=300, centers=4, random_state=42)
# Convert the data to a DataFrame
df = pd.DataFrame(data, columns=['Feature1', 'Feature2'])
# Visualize the original data
plt.scatter(df['Feature1'], df['Feature2'], s=50)
plt.title('Original Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
# Initialize the K-Means model
kmeans = KMeans(n_clusters=4, random_state=42)
# Fit the model to the data
kmeans.fit(df)
# Get the cluster labels and centroids
cluster_labels = kmeans.labels_
centroids = kmeans.cluster_centers_
# Add cluster labels to the DataFrame
df['Cluster'] = cluster_labels
# Visualize the clustered data
plt.scatter(df['Feature1'], df['Feature2'], c=df['Cluster'], cmap='viridis', s=50)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='X', s=200, color='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

10A&16B) PCA

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
```

```
import matplotlib.pyplot as plt
# Load the Iris dataset
iris_data = load_iris()
X = iris_data.data
y = iris_data.target
class_names = iris_data.target
# Convert the data to a DataFrame
df = pd.DataFrame(X, columns=iris_data.feature_names)
# Standardize the data (optional but recommended for PCA)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_standardized = scaler.fit_transform(X)
# Initialize PCA with the number of components to retain
n_components = 2 # Set the desired number of components
pca = PCA(n_components=n_components)
# Fit and transform the data
X_pca = pca.fit_transform(X_standardized)
# Create a DataFrame from the PCA-transformed data
df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(n_components)])
df_pca['Target'] = y
# Visualize the data before and after PCA
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
# Before PCA
plt.subplot(1, 2, 2)
for i in range(3):
plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1], c=[plt.cm.viridis(i / 2.0)], edgecolors='k', s=50,
label=class_names[i])
ax1.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolors='k', s=50)
ax1.set_title('Original Data')
ax1.set_xlabel('Feature 1')
ax1.set_ylabel('Feature 2')
```

```
plt.legend()
# After PCA
ax2.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis', edgecolors='k', s=50)
ax2.set_title('Data After PCA')
ax2.set_xlabel('Principal Component 1')
ax2.set_ylabel('Principal Component 2')
plt.legend()
plt.show()
```

10B) MEAN,MEDIAN,MODE AND VARIANCE

```
import pandas as pd
from scipy import stats
# Sample dataset (you can also load from CSV using pd.read_csv)
data = {
   'Scores': [88, 92, 79, 93, 85, 91, 76, 89, 95, 92]
}
# Create a DataFrame
df = pd.DataFrame(data)
# Compute statistical measures
mean_val = df['Scores'].mean()
median_val = df['Scores'].median()
mode_val = df['Scores'].mode()[0]     # mode() returns a Series
variance_val = df['Scores'].var()
std_dev_val = df['Scores'].std()
# Display the results
print("Descriptive Statistics:")
print(f"Mean: {mean_val}")
print(f"Median: {median_val}")
print(f"Mode: {mode_val}")
print(f"Variance: {variance_val}")
print(f"Standard Deviation: {std_dev_val}")
```

11A) LDA

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import matplotlib.pyplot as plt
# Load the iris dataset
iris = load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names
# Perform Linear Discriminant Analysis (LDA)
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)
# Plot the results
plt.figure(figsize=(12, 5))
# Before LDA
plt.subplot(1, 2, 1)
for i in range(3):
plt.scatter(X[y == i, 0], X[y == i, 1], c=[plt.cm.viridis(i / 2.0)], edgecolors='k', s=50,
label=class_names[i])
plt.title('Original Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
# After LDA
plt.subplot(1, 2, 2)
for i in range(3):
plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], c=[plt.cm.viridis(i / 2.0)], edgecolors='k', s=50,
label=class_names[i])
plt.title('Data After LDA')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
```

```
plt.legend()
plt.tight_layout()
plt.show()
```

12A&14B) AND LOGIC GATE

```
import numpy as np
# Define input features:
input_features = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
print("Input features shape:", input_features.shape)
# Define target output:
target_output = np.array([0, 0, 0, 1])
print("Target output shape:", target_output.shape)
print("Target output:", target_output)
# Define weights:
weights = np.array([0.1, 0.2])
print("Initial weights:", weights)
# Bias weight:
bias = 0.3
# Learning Rate:
lr = 0.05
# Step function:
def step_function(x):
return 1 if x >= 0 else 0
# Main logic for Perceptron:
# Running our code 10000 times:
for epoch in range(10000):
total_error = 0
for input_data, label in zip(input_features, target_output):
inputs = input_data
# Feedforward input:
in_o = np.dot(inputs, weights) + bias
# Feedforward output:
```

```
out_o = step_function(in_o)
# Calculating error
error = label - out_o
total_error += abs(error)
# Updating the weights values:
weights += lr * error * inputs
# Updating the bias weight value:
bias += lr * error
# Early stopping if total_error is 0
if total_error == 0:
break
# Check the final values for weight and bias
print("Final weights:", weights)
print("Final bias:", bias)
# Test the trained perceptron
print("Testing the trained perceptron:")
for inputs, label in zip(input_features, target_output):
prediction = step_function(np.dot(inputs, weights) + bias)
print(f"Input: {inputs}, Predicted Output: {prediction}, Target Output: {label}")
```

12B) Write a python program to compute (i) Feature Normalization: Min-max normalization (ii) Filtering the data

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
# Sample dataset
data = {
  'Student': ['A', 'B', 'C', 'D', 'E'],
  'Maths': [45, 89, 72, 60, 95],
  'Science': [38, 85, 66, 55, 90]
}
# Create DataFrame
df = pd.DataFrame(data)
```

```
print("Original Data:")
print(df)
# ------------------------------------
# (i) Feature Normalization (Min-Max)
# ------------------------------------
scaler = MinMaxScaler()
# Normalize only numeric columns
df[['Maths_normalized', 'Science_normalized']] = scaler.fit_transform(df[['Maths',
'Science']])
print("\nData after Min-Max Normalization:")
print(df)
# ------------------------------------
# (ii) Filtering the Data
# ------------------------------------
# Filter students who scored more than 80 in Maths
filtered_df = df[df['Maths'] > 80]
print("\nFiltered Data (Maths > 80):")
print(filtered_df)
```

6. **13 A) OR logic gate with 2-bit Binary input**

```
 import numpy as np
# Step activation function
def step_function(x):
   return 1 if x >= 0 else 0
# Perceptron training function
def train_perceptron(X, y, lr=0.1, epochs=10):
   weights = np.zeros(X.shape[1])  # 2 inputs
   bias = 0
 for epoch in range(epochs):
       print(f"Epoch {epoch + 1}")
       for i in range(len(X)):
          # Linear combination
```

```
            z = np.dot(X[i], weights) + bias
            # Activation
            y_pred = step_function(z)
            # Update weights and bias if prediction is wrong
            error = y[i] - y_pred
            weights += lr * error * X[i]
            bias += lr * error
            print(f"Input: {X[i]}, Target: {y[i]}, Predicted: {y_pred}, Error: {error}")
        print(f"Weights: {weights}, Bias: {bias}\n")
    return weights, bias
# Prediction function
def predict(X, weights, bias):
    return [step_function(np.dot(x, weights) + bias) for x in X]
# OR Gate dataset
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([0, 1, 1, 1])
# Train the perceptron
weights, bias = train_perceptron(X, y)
# Final predictions
print("Final predictions:")
for i in range(len(X)):
    print(f"Input: {X[i]}, Output: {predict([X[i]], weights, bias)[0]}")
```

14 A) ANN

```
Import numpy as np
#creating the input array
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])
Print('\n Input:')
Print(X)
Y=np.array([[1],[1],[0]])
Print('\n Actual Output:')
Print(y)
```

```
Print"\n shape of Input:", X.shape)
def.sigmoid(x):
  return 1/(1 + np.exp(-x))
derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 – x)
lr=0.1 #learning rate
inputlayer_neurons = X.shape[1] # no. of features in dataset
hiddenlayer_neurons = 3 # no. of hidden layers neurons
output_neurons = 1 # no. of neurons at output layer
#Initializing weight and bias
wh=np.random.uniform(size=(inputlayer_neurons, hiddenlayer_neurons))
bh= np.random.uniform(size=(1, hiddenlayer_neurons))
wout= np.random.unifor(size=(hiddenlayer_neurons, output_neurons))
bout= np.random.unifor(size=(1, output_neurons))
#training the model
For I in range(epoch):
#Forward propagation
hidden_layer_input1=np.dot(X,wh)
hidden_layer_input=hidden_layer_input 1 +bh
hidden_layer_activations=sigmoiud(hidden_layer_input)
output_layer_input1=np.dot(hiddenlayer_activations, wout))
output_layer_input= output_layer_input1bout
output = sigmoid(output_layer_input)
#Backpropagation
E = y-output
Slope_output_layer = derivatives_sigmoid(output)
Slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
d_output = E * slope_Output_layer
Error_at_hidden_layer = d_output.dot(wout.T)
d_hiddenlayer= Error_at_hidden_layer * slope_Hidden_layer
```

```
wout+ =hiddenlayer_activations.T.dot(d_output) * lr
bout+ =np.sum(d_output,axis=0, keepdims=True) * lr
wh+ =X.T.dot(d_hiddenlayer) * lr
bh+=np.sum(d_hiddenlayer,axis=0, keepdims=True) * lr
```

15 A) RFE

```
import pandas as pd
from sklearn.datasets import load_Iris
fromsklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
#Load the Iris dataset
Iris_data -= load_Iris()
X = Iris_data.data
Y = Iris_data.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)
#Intitalize the Random Forest Classifier
random_forest = RandomForestClassifier(n_estimators=100, random_state = 42)
#Train the Model
random_forest.fit(X_train, y_train)
#Make Predictions on the test set
y_pred = random_forest.predict(X_test)
#Evaluate the Model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```

15B) Decision Tree Classification

```
# Read the CSV Files
import pandas as pd
import numpy as np
df=pd.read_csv('PlayTennis.csv')
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
df_num_cat=pd.DataFrame()
df_num_cat['Outlook']= enc.fit_transform(df['Outlook'])
df_num_cat['Temperature']= enc.fit_transform(df['Temperature'])
df_num_cat['Humidity']= enc.fit_transform(df['Humidity'])
df_num_cat['Wind']= enc.fit_transform(df['Wind'])
df_num_cat['Play Tennis']= enc.fit_transform(df['Play Tennis'])
df_num_cat
x=df_num_cat.drop(['Play Tennis'],axis=1)
y=df_num_cat['Play Tennis']
# Split dataset into training set and test set
from sklearn.model_selection import train_test_split # Import train_test_split function
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=1) #
70% training
and 30% test
print(X_train)
print(X_test)
print(y_train)
print(y_test)

from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
df_clf=DecisionTreeClassifier(criterion='entropy')
#df_clf=DecisionTreeClassifier()
df_clf.fit(X_train,y_train)
```

```
y_pred=df_clf.predict(X_test)

print(y_train)

print(y_test)

print(y_pred)

print("Accuracy")

print(metrics.accuracy_score(y_test,y_pred))

# Plot the decision tree

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

plot_tree(df_clf, filled=True, feature_names=X_train.columns, class_names=df['Play Tennis'].unique())

plt.show(
```

16A) BOOSTING ENABLE METHOD

```
import pandas as pd

from sklearn.datasets import load_Iris

fromsklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#Load the Iris dataset

Iris_data -= load_Iris()

X = Iris_data.data

Y = Iris_data.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)

#Intitalize the AdaBoost Classifier with Decision Tree as base estimator

adaboost = AdaBoostClassifier(n_estimators=50, random_state = 42)

#Train the Model

adaboost.fit(X_train, y_train)

#Make Predictions on the test set

y_pred = adaboost.predict(X_test)

#Evaluate the Model

accuracy = accuracy_score(y_test, y_pred)
```

```
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", classification_rep)
print("\nConfusion Matrix:\n", conf_matrix)
```