EX1:

```
import pandas as pd
import numpy as np

data=pd.read_csv("EX1.csv")
print(data)

d = np.array(data)[:, :-1]
print("\nThe attributes are:", d)

target = np.array(data)[:, -1]
print("\nThe target is:", target)

def train(c, t):
   specific_hypothesis = None

   for i, val in enumerate(t):
      if val == "Yes":
         specific_hypothesis = c[i].copy()
         break

   if specific_hypothesis is None:
      raise ValueError("No positive instance found in the target list.")

   for i, val in enumerate(c):
      if t[i] == "Yes":
         for x in range(len(specific_hypothesis)):
            if val[x] != specific_hypothesis[x]:
               specific_hypothesis[x] = "?"

   return specific_hypothesis

print("\nThe specific hypothesis is:", train(d, target))
```

EX2:

```
import numpy as np
import pandas as pd

data = pd.read_csv('EX2.csv')
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("Initialization Of Specific_h \n",specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("Initialization Of general_h \n", general_h)

    for i, h in enumerate(concepts):
        if target[i] == "Yes":
            print("If Instance Is Positive ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    specific_h[x] ='?'
                    general_h[x][x] ='?'

        if target[i] == "No":
            print("If Instance Is Negative ")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Step {}".format(i+1))
        print(specific_h)
        print(general_h)
        print("\n")
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])
    return specific_h, general_h

s_final, g_final = learn(concepts, target)

print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

EX3:

```
import numpy as np
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

data = pd.read_csv('EX3.csv',names=['Equipments Used'],header=None)
data
data = list(data["Equipments Used"].apply(lambda x:x.split(',')))
te = TransactionEncoder()

te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data, columns=te.columns_)

frq_items = apriori(df, min_support=0.3, use_colnames=True)
print(frq_items)
rules = association_rules(frq_items, metric="confidence", min_threshold=0.5,
num_itemsets=len(frq_items))
rules = rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']]
print(rules)
```

EX4A:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load data
data = pd.read_csv('EX4A.csv')
data['days_per_week'] = data['days_per_week'].apply(lambda x: len(x.split(',')))

# Define features and target
X = data.drop('avg_time_in_gym', axis=1)
y = data['avg_time_in_gym']

# Feature selection
categorical_features = ['gender', 'abonoment_type', 'attend_group_lesson', 'fav_group_lesson',
'fav_drink', 'personal_training', 'name_personal_trainer', 'uses_sauna']
numerical_features = ['Age', 'visit_per_week', 'days_per_week']

# Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])
X_processed = preprocessor.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)

# Model training
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Metrics
train_mse = mean_squared_error(y_train, y_train_pred)
```

```
train_r2 = r2_score(y_train, y_train_pred)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Training Mean Squared Error: {train_mse:.2f}")
print(f"Training R-squared: {train_r2:.2f}")
print(f"Test Mean Squared Error: {test_mse:.2f}")
print(f"Test R-squared: {test_r2:.2f}")

# Plot training data
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(y_train, y_train_pred, alpha=0.5, color='blue')
plt.plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()], color='red', linestyle='--')
plt.xlabel('Actual Avg Time in Gym')
plt.ylabel('Predicted Avg Time in Gym')
plt.title('Training Set: Actual vs Predicted')

# Plot test data
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_test_pred, alpha=0.5, color='green')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
plt.xlabel('Actual Avg Time in Gym')
plt.ylabel('Predicted Avg Time in Gym')
plt.title('Test Set: Actual vs Predicted')

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

EX4B:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, roc_curve, auc, confusion_matrix,
classification_report
import matplotlib.pyplot as plt

data = pd.read_csv('EX4B.csv')
data['Journey Status'] = data['Journey Status'].apply(lambda x: 1 if x == 'On Time' else 0)

def time_to_seconds(time_str):
    hh, mm, ss = map(int, time_str.split(':'))
    return hh * 3600 + mm * 60 + ss

data['Departure Time'] = data['Departure Time'].apply(time_to_seconds)
data['Arrival Time'] = data['Arrival Time'].apply(time_to_seconds)

X = data[['Price', 'Ticket Class', 'Departure Time', 'Arrival Time']]  # Example features
y = data['Journey Status']  # Target variable

numerical_features = ['Price', 'Departure Time', 'Arrival Time']
categorical_features = ['Ticket Class']
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

EX5:

```
import pandas as pd
fromsklearn.model_selection import train_test_split
fromsklearn.tree import DecisionTreeClassifier, plot_tree, export_text
fromsklearn.metrics import accuracy_score
importmatplotlib.pyplot as plt

# Step 1: Load the dataset
file_path = 'EX5.csv'
df = pd.read_csv(file_path)

# Step 2: Encode categorical features
df_encoded = pd.get_dummies(df.drop('PlayTennis', axis=1))

# Step 3: Prepare the target variable
y = df['PlayTennis'].apply(lambda x: 1 if x == 'Yes' else 0)

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df_encoded, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the Decision Tree Classifier
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)
clf.fit(X_train, y_train)

# Step 6: Predict on the test set
y_pred = clf.predict(X_test)

# Step 7: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"\nDataset Accuracy: {accuracy:.2f}")

# Step 8: Extract and print decision tree rules
tree_rules = export_text(clf, feature_names=list(df_encoded.columns))
print("\nDecision Tree Rules:")
print(tree_rules)

# Step 9: Visualize the Decision Tree
plt.figure(figsize=(25, 15))
plot_tree(
clf,
filled=True,
feature_names=df_encoded.columns,
class_names=['No', 'Yes'],
rounded=True,
fontsize=12,
proportion=True,
precision=2,
)
plt.title("Decision Tree Visualization", fontsize=14)
plt.tight_layout()
plt.show()

# Step 10: Classify a new sample
new_sample = {
  'Outlook_Sunny': [1],
  'Outlook_Overcast': [0],
  'Outlook_Rain': [0],
  'Temperature_Cool': [0],
    'Temperature_Hot': [1],
    'Temperature_Mild': [0],
    'Humidity_High': [1],
    'Humidity_Normal': [0],
    'Wind_Strong': [0],
    'Wind_Weak': [1]
}

new_sample_df = pd.DataFrame(new_sample)
new_sample_df = new_sample_df[X_train.columns]

predicted_class = clf.predict(new_sample_df)
print(f"\nPredicted class for the new sample: {'Yes' if predicted_class[0] == 1 else 'No'}")
```

EX6:

```
import pandas as pd
from sklearn import tree
fromsklearn.preprocessing import LabelEncoder
fromsklearn.naive_bayes import GaussianNB
fromsklearn.model_selection import train_test_split
fromsklearn.metrics import accuracy_score

data = pd.read_csv('EX6.csv')
print("The first5 Values of data is :\n", data.head())
data=data.apply(LabelEncoder().fit_transform)
data.head()

X = data.iloc[:,:-1]
print("\nThe First 5 values of the train data is\n", X.head())

y = data.iloc[:, -1]
print("\nTheFirst 5 values of train output is\n", y.head())

le_Good_Job = LabelEncoder()
y = le_Good_Job.fit_transform(y)
print("\nNow the Train output is\n",y)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20)
classifier = GaussianNB()
classifier.fit(X_train,y_train)
print("Accuracy is:", accuracy_score(classifier.predict(X_test), y_test))
```

EX7:

```
importnumpy as np

class NeuralNetwork:

def __init__(self, input_size, hidden_size, output_size):

self.input_size = input_size

self.hidden_size = hidden_size

self.output_size = output_size


    # Initialize weights and biases

self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)

self.bias_hidden = np.zeros((1, self.hidden_size))

self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)

self.bias_output = np.zeros((1, self.output_size))


def sigmoid(self, x):

    return 1 / (1 + np.exp(-x))


defsigmoid_derivative(self, x):

    return x * (1 - x)


def forward(self, X):

    # Forward pass

self.hidden_layer_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden

self.hidden_layer_output = self.sigmoid(self.hidden_layer_input)


self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) +
self.bias_output

self.output = self.sigmoid(self.output_layer_input)
```

```
returnself.output

def backward(self, X, y, output, learning_rate):
    # Backpropagation
    error = y - output
output_delta = error * self.sigmoid_derivative(output)

hidden_error = output_delta.dot(self.weights_hidden_output.T)
hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_layer_output)

    # Update weights and biases
self.weights_hidden_output += self.hidden_layer_output.T.dot(output_delta) * learning_rate
self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate
self.weights_input_hidden += X.T.dot(hidden_delta) * learning_rate
self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * learning_rate

def train(self, X, y, epochs, learning_rate):
    for epoch in range(epochs):
      output = self.forward(X)
self.backward(X, y, output, learning_rate)
      if epoch % 100 == 0:
print(f'Epoch {epoch}: Error {np.mean(np.square(y - output))}')


# Example usage:
```

```
# Initialize neural network
input_size = 1
hidden_size = 1
output_size = 1
nn = NeuralNetwork(input_size, hidden_size, output_size)
# Example training data
X = np.array([[5]])  # Input
y = np.array([[1]])  # Target output

# Train the neural network
nn.train(X, y, epochs=1000, learning_rate=0.1)

# Make predictions
predictions = nn.forward(X)
print("Predictions:", predictions)

# Example usage:
# Initialize neural network
input_size = 2
hidden_size = 3
output_size = 1

nn = NeuralNetwork(input_size, hidden_size, output_size)
# Example training data
X = np.array([[0, 0],[1, 0],[0, 1], [1, 1]])
y = np.array([[1],[0],[1], [0]])
```

```
# Train the neural network
nn.train(X, y, epochs=987, learning_rate=0.1)

# Make predictions
predictions = nn.forward(X)
print("Predictions:", predictions)
```

EX8:

```
import pandas as pd
import numpy as np
importmatplotlib.pyplot as plt
%matplotlib inline

df = pd.DataFrame({
    'x': [13, 27, 35, 49, 58, 62, 74, 81, 93, 15, 38, 47, 55, 67, 78, 84, 92, 10, 31, 59],
    'y': [88, 76, 65, 54, 47, 39, 30, 22, 15, 91, 73, 59, 44, 37, 28, 20, 12, 95, 80, 50]
})
# Setting random seed for reproducibility
np.random.seed(200)

# Initializing centroids randomly
k = 3
centroids = {i+1: [np.random.randint(0, 80), np.random.randint(0, 80)] for i in range(k)}
print(centroids)

# Plotting initial points and centroids
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color='k')
colmap = {1: 'r', 2: 'g', 3: 'b'}
fori in centroids.keys():
plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

```
# Function to assign points to the nearest centroid
def assignment(df, centroids):
fori in centroids.keys():
df[f'distance_from_{i}'] = np.sqrt((df['x'] - centroids[i][0]) ** 2 + (df['y'] - centroids[i][1]) ** 2)
centroid_distance_cols = [f'distance_from_{i}' for i in centroids.keys()]
df['closest'] = df.loc[:, centroid_distance_cols].idxmin(axis=1)
df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance_from_')))
df['color'] = df['closest'].map(lambda x: colmap[x])
    return df

df = assignment(df, centroids)
print(df.head())

# Plotting points with colors corresponding to closest centroids
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
fori in centroids.keys():
plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

import copy

# Save the previous centroids to track movement
old_centroids = copy.deepcopy(centroids)
```

```
# Function to update centroids

def update(centroids):

fori in centroids.keys():

centroids[i][0] = np.mean(df[df['closest'] == i]['x'])

centroids[i][1] = np.mean(df[df['closest'] == i]['y'])

    return centroids


# Update centroids and plot

centroids = update(centroids)

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

ax = plt.axes()

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')

fori in centroids.keys():

plt.scatter(*centroids[i], color=colmap[i])

plt.xlim(0, 80)

plt.ylim(0, 80)


# Plot arrows showing centroid movement

fori in old_centroids.keys():

old_x = old_centroids[i][0]

old_y = old_centroids[i][1]

    dx = (centroids[i][0] - old_centroids[i][0]) * 0.75

dy = (centroids[i][1] - old_centroids[i][1]) * 0.75

ax.arrow(old_x, old_y, dx, dy, head_width=2, head_length=3, fc=colmap[i], ec=colmap[i])


plt.show()
```

```
# Assign points to closest centroids and update centroids again
df = assignment(df, centroids)
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
fori in centroids.keys():
plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()

# Iterative process of assignment and update until convergence
while True:
closest_centroids = df['closest'].copy(deep=True)
    centroids = update(centroids)
df = assignment(df, centroids)
ifclosest_centroids.equals(df['closest']):
        break

# Final plot after convergence
fig = plt.figure(figsize=(5, 5))
plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')
fori in centroids.keys():
plt.scatter(*centroids[i], color=colmap[i])
plt.xlim(0, 80)
plt.ylim(0, 80)
plt.show()
```

EX9:

```
import pandas as pd
import numpy as np
fromsklearn.model_selection import train_test_split
fromsklearn.ensemble import RandomForestClassifier
fromsklearn.metrics import accuracy_score

# Load dataset (Replace 'ipl_match.csv' with actual dataset)
data = pd.read_csv('EX9.csv')

# Selecting relevant features
features = ['team1', 'team2', 'venue', 'toss_winner', 'toss_decision']
X = data[features]
y = data['winner']

# Convert categorical data to numerical
X = pd.get_dummies(X)
y = pd.factorize(y)[0]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict outcomes
```

```
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Model Accuracy: {accuracy * 100:.2f}%')

# Function to predict match outcome based on user input
defpredict_winner():
   team1 = input("Enter Team 1: ")
   team2 = input("Enter Team 2: ")
venue = input("Enter Venue: ")
toss_winner = input("Enter Toss Winner: ")
toss_decision = input("Enter Toss Decision (bat/bowl): ")

input_data = pd.DataFrame([[team1, team2, venue, toss_winner, toss_decision]],
columns=features)
input_data = pd.get_dummies(input_data)
input_data = input_data.reindex(columns=X.columns, fill_value=0)
prediction = model.predict(input_data)
   # Getting the predicted team name
winner_labels = data['winner'].unique()
predicted_winner = winner_labels[prediction[0]]

print(f'Predicted Winner: {predicted_winner}')

# Get user input and predict winner
predict_winner()
```

EX10:

```
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

def load_data(file_path):
    df = pd.read_csv(file_path)
    df["Marital_Status"] = LabelEncoder().fit_transform(df["Marital_Status"])
    return df

def train_model(df):
    X = df.drop(columns=["Eligible"])
    y = df["Eligible"]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    joblib.dump(model, "loan_model.pkl")
    return model

def predict_eligibility(model, user_input):
    df_input = pd.DataFrame([user_input])
    return "Eligible for Loan" if model.predict(df_input)[0] == 1 else "Not Eligible for Loan"

if __name__ == "__main__":
    df = load_data("EX10.csv")
    model = train_model(df)
    print("Loan Elgibility Predictor")
    user_input = {
        "Age": int(input("Enter Age: ")),
        "Income": int(input("Enter Income: ")),
        "Credit_Score": int(input("Enter Credit Score(0-1000): ")),
        "Loan_Amount": int(input("Enter Loan Amount: ")),
        "Existing_Loans": int(input("Enter Number of Existing Loans: ")),
        "Employment_Years": int(input("Enter Years of Employment: ")),
        "Marital_Status": {"Single": 0, "Married": 1, "Divorced": 2}.get(input("Enter Marital Status (Single/Married/Divorced): "), 0),
        "Has_House": 1 if input("Do you own a house? (yes/no): ").strip().lower() == "yes" else 0,
        "Has_Car": 1 if input("Do you own a car? (yes/no): ").strip().lower() == "yes" else 0
    }

    print("Loan Prediction Result:", predict_eligibility(model, user_input))
```