11/12/2024

MACHINE LEARNING LAB BR20-CAD- III YEAR I SEM SYLLABUS

Prepared by:
MANIKANTA DARAPUREDDY-22221A4517
PRANAI BODAPATI-22221A4509
SURYA TEJA-22221A4552

BONAM VENKATA CHALAMAYYA ENGINERING COLLEGE
ODALAREVU – 533 210, Andhra Pradesh, India

III Year - I Semester	Code : 20AD5L03	L	T	P	C
		0	0	3	1.5

MACHINE LEARNING LAB

Course Objectives:
This course will enable students to learn and understand different Data sets in implementing the machine learning algorithms.

Course Outcomes:
At the end of the course, student will be able to

- Implement procedures for the machine learning algorithms
- Design and Develop Python programs for various Learning algorithms
- Apply appropriate data sets to the Machine Learning algorithms
- Develop Machine Learning algorithms to solve real world problems

Experiments:
Experiment-1:
Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
Experiment-2:
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.
Experiment-3:
Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
Experiment-4:
Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier
Experiment-5: Develop a program for Bias, Variance, Remove duplicates , Cross Validation
Experiment-6: Write a program to implement Categorical Encoding, One-hot Encoding Experiment-7:
Build an Artificial Neural Network by implementing the Back propagation algorithm and test the same using appropriate data sets.
Experiment-8:
Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.
Experiment-9: Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.
Experiment-10:
Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.
Experiment-11: Apply EM algorithm to cluster a Heart Disease Data Set. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.
Experiment-12: Exploratory Data Analysis for Classification using Pandas or Matplotlib.
Experiment-13:
Write a Python program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set Experiment-14:
Write a program to Implement Support Vector Machines and Principle Component Analysis
Experiment-15:
Write a program to Implement Principle Component Analysis

Text Books: 1. Machine Learning – Tom M. Mitchell, MGH
2. Fundamentals of Speech Recognition By Lawrence Rabiner and Biing – Hwang Juang.

Reference Books: 1. Machine Learning: An Algorithmic Perspective, Stephen Marsland, Taylor & Francis

Experiment 1: FIND-S Algorithm

Aim:

To implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

Description:-

The FIND-S algorithm finds the most specific hypothesis that fits all positive training examples in a dataset. The algorithm starts with the most specific hypothesis and generalizes it by comparing it with each positive instance from the dataset.

Code:-

```
import pandas as pd

data_set = pd.read_csv('ws.csv')
print(data_set)

hypothesis = ['Ø'] * (len(data_set.columns) - 1)
print(hypothesis)

for index, row in data_set.iterrows():
    if row['Enjoy sport'] == 'Yes':
        for i in range(len(hypothesis)):
            attribute = list(data_set.columns)[i]
            if hypothesis[i] == 'Ø':
                hypothesis[i] = row[attribute]
            elif hypothesis[i] != row[attribute]:
                hypothesis[i] = '?'
        print(f"Updated Hypothesis after row {index + 1}: {hypothesis}")

print("Final Hypothesis:", hypothesis)
```

Data set(ws.csv):-

```
SKY,AIR TEMP,HUMIDITY,WIND,WATER,FORECAST,Enjoy sport
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

Output:-

```
['Ø', 'Ø', 'Ø', 'Ø', 'Ø', 'Ø']

Updated Hypothesis after row 1: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm',
'Same']
Updated Hypothesis after row 2: ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
Updated Hypothesis after row 4: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

Final Hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

Experiment 2: Candidate-Elimination Algorithm

Aim:-

To implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Description:-

The Candidate-Elimination algorithm computes the version space, which consists of all the hypotheses consistent with the training examples. It maintains a boundary for the most general and most specific hypotheses and updates them based on each training example.

Code:-

```
import pandas as pd

data_set = pd.read_csv('sport.csv')

X = data_set.iloc[:, :-1].values
y = data_set.iloc[:, -1].values

S = X[0].copy()

G = [['?' for _ in range(len(S))]]

print(f"Initial Specific Hypothesis (S): {S}")
print(f"Initial General Hypothesis (G): {G}\n")

for i in range(len(X)):
    print(f"Instance {i+1}: {X[i]}, Outcome: {y[i]}")

    if y[i] == 'Yes':
        for j in range(len(S)):
            if S[j] != X[i][j]:
                S[j] = '?'
        G = [g for g in G if all(g[k] == '?' or g[k] == X[i][k] for k in
range(len(g)))]
    else:
        new_G = []
        for g in G:
            for j in range(len(g)):
                if g[j] == '?' and S[j] != X[i][j]:
                    new_g = g.copy()
                    new_g[j] = S[j]
                    if new_g != ['?' for _ in range(len(S))]:
                        new_G.append(new_g)
        G = new_G
    print(f"Specific Hypothesis after instance {i+1}: {S}")
    print(f"General Hypothesis after instance {i+1}: {G}\n")

print("Final Specific Hypothesis (S):", S)
print("Final General Hypothesis (G):", G)
```

Dataset:-

```
SKY,AIR TEMP,HUMIDITY,WIND,WATER,FORECAST,Enjoy sport
Sunny,Warm,Normal,Strong,Warm,Same,Yes
Sunny,Warm,High,Strong,Warm,Same,Yes
Rainy,Cold,High,Strong,Warm,Change,No
Sunny,Warm,High,Strong,Cool,Change,Yes
```

Output:-

```
Initial Specific Hypothesis (S): ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'
'Same']
Initial General Hypothesis (G): [['?', '?', '?', '?', '?', '?']]

Instance 1: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same'], Outcome: Yes
Specific Hypothesis after instance 1: ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'
'Same']
General Hypothesis after instance 1: [['?', '?', '?', '?', '?', '?']]

Instance 2: ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same'], Outcome: Yes
Specific Hypothesis after instance 2: ['Sunny' 'Warm' '?' 'Strong' 'Warm'
'Same']
General Hypothesis after instance 2: [['?', '?', '?', '?', '?', '?']]

Instance 3: ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change'], Outcome: No
Specific Hypothesis after instance 3: ['Sunny' 'Warm' '?' 'Strong' 'Warm'
'Same']
General Hypothesis after instance 3: [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

Instance 4: ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change'], Outcome: Yes
Specific Hypothesis after instance 4: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
General Hypothesis after instance 4: [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?']]

Final Specific Hypothesis (S): ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General Hypothesis (G): [['Sunny', '?', '?', '?', '?', '?'], ['?',
'Warm', '?', '?', '?', '?']]
```

Experiment 3: ID3 Decision Tree Algorithm

Aim:-

To demonstrate the working of the ID3 decision tree algorithm and classify a new sample.

Description:-

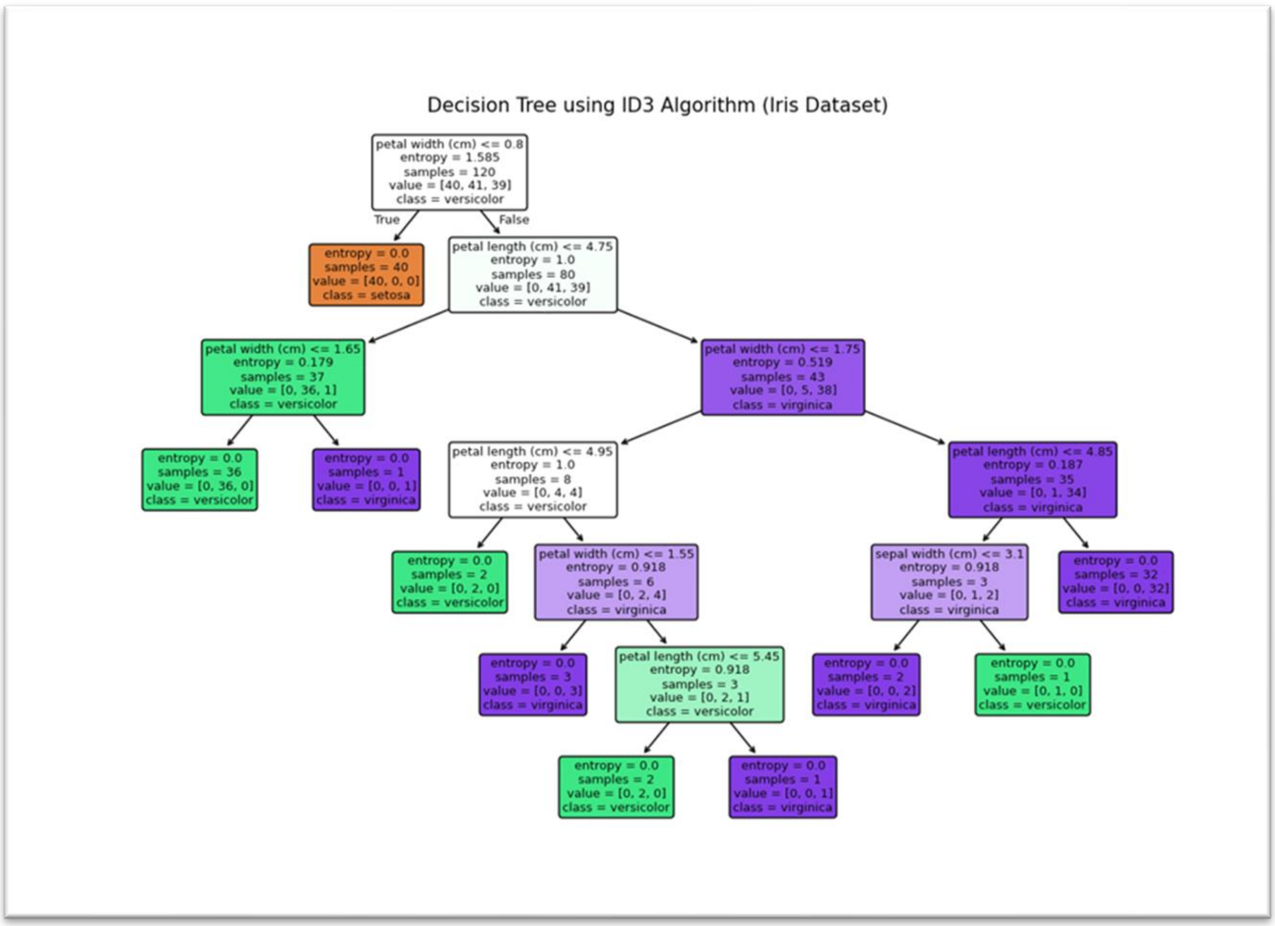
The ID3 algorithm is used to build a decision tree for classification problems. It selects the attribute that maximizes information gain and recursively partitions the dataset. The resulting tree can be used to classify new samples.

Code:-

```
# Importing the necessary libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = pd.Series(iris.target)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize the Decision Tree classifier using entropy (ID3)
clf = DecisionTreeClassifier(criterion='entropy')
# Train the model
clf.fit(X_train, y_train)
# Test the model on the testing set
y_pred = clf.predict(X_test)
# Check accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Draw the decision tree
plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=iris.feature_names,
class_names=iris.target_names, filled=True,
rounded=True)
plt.title("Decision Tree using ID3 Algorithm (Iris Dataset)")
plt.show()
```

Output:-

Accuracy: 100.00%



Experiment 4: Linear and Logistic Regression, Binary Classifier

Aim:-

To solve real-world problems using linear regression, logistic regression, and a binary classifier.

Description:-

Linear regression predicts a continuous target variable by finding the best-fit line. Logistic regression is used for binary classification problems where the output is categorical. A binary classifier assigns one of two possible classes to the input data.

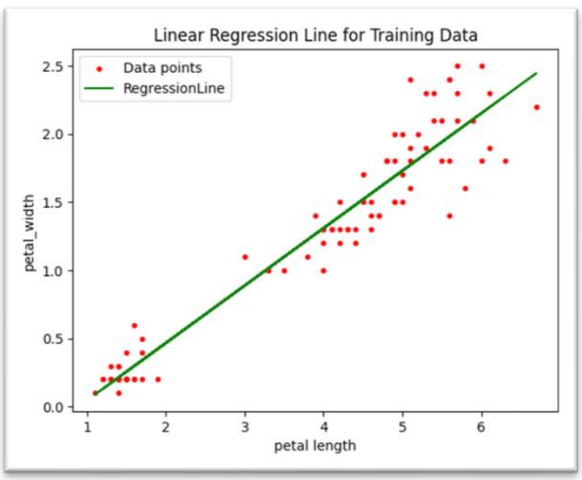
Code:-

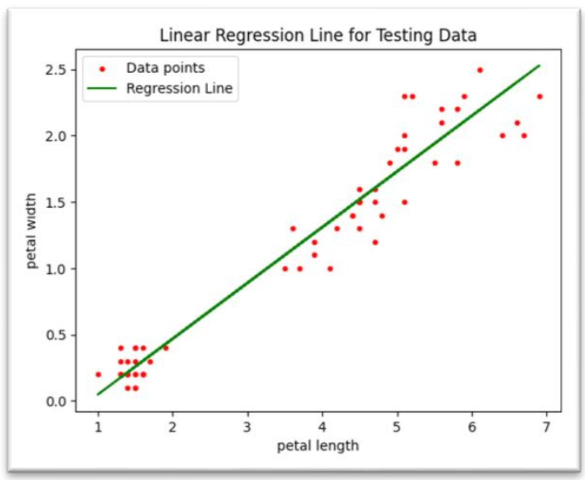
1.Linear Regression

```
# Import necessary libraries
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
data_set = sns.load_dataset("iris")

# Select features for training
data_set = data_set[["petal_length", "petal_width"]]

# Separate input and output
x = data_set['petal_length']
y = data_set['petal_width']

# Plot scatter plot of the data
plt.scatter(x, y, color='red', marker='.')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4,
random_state=42)

# Convert to numpy arrays and reshape
x_train = np.array(x_train).reshape(-1, 1)
y_train = np.array(y_train).reshape(-1, 1)
x_test = np.array(x_test).reshape(-1, 1)
y_test = np.array(y_test).reshape(-1, 1)

# Create and train the linear regression model
model = LinearRegression()
model.fit(x_train, y_train)

# Get model parameters
m = model.coef_
c = model.intercept_

# Predict on training data
y_train_pred=m*x_train+c
y_train_pred

# Predict on test data and calculate performance metrics
y_test_pred=m*x_test+c
y_test_pred
```

```python
# Plot regression line with training data
plt.scatter(x_train, y_train, color='blue', label='Training data')
plt.plot(x_train, y_train_pred, color='green', label='Regression Line')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.legend()
plt.show()

# Plot regression line with testing data
plt.title('Linear Regression Line for Testing Data')
plt.scatter(x_test, y_test, label='Data points', marker='.', color='r')
plt.plot(x_test, y_test_pred, label='Regression Line', color='g')
plt.xlabel("petal length")
plt.ylabel("petal width")
plt.legend()
plt.show()
```

Output:-



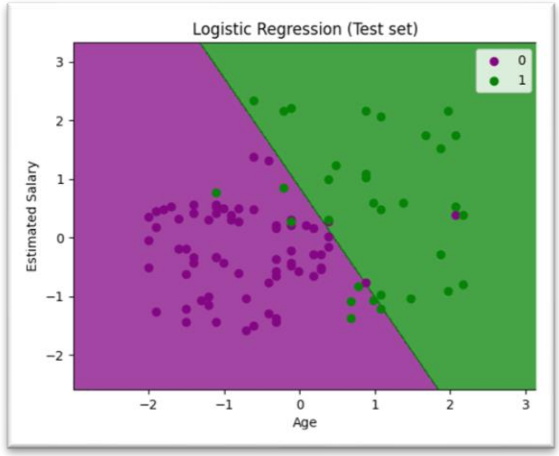


Logistic regression:-

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from matplotlib.colors import ListedColormap

# Load dataset
data_set = pd.read_csv('car_data.csv')
data_set = data_set[['Age', 'EstimatedSalary', 'Purchased']]

# Extract independent and dependent variables
x = data_set[['Age', 'EstimatedSalary']]
y = data_set['Purchased']

# Splitting the dataset into training and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25,
random_state=0)
```

```
# Feature scaling
st_x = StandardScaler()
x_train = st_x.fit_transform(x_train)
x_test = st_x.transform(x_test)

# Fitting Logistic Regression to the Training set
classifier = LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)

LogisticRegression()

# Predicting the Test Result
y_pred = classifier.predict(x_test)
y_pred

# Confusion matrix to evaluate the results
cm = confusion_matrix(y_test, y_pred)
cm
# Visualizing the training set results
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:,
0].max() + 1, step=0.01),
                     np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:,
1].max() + 1, step=0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
             alpha=0.75, cmap=ListedColormap(('purple', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c=ListedColormap(('purple', 'green'))(i), label=j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualizing the test set results
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start=x_set[:, 0].min() - 1, stop=x_set[:,
0].max() + 1, step=0.01),
                     np.arange(start=x_set[:, 1].min() - 1, stop=x_set[:,
1].max() + 1, step=0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
             alpha=0.75, cmap=ListedColormap(('purple', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c=ListedColormap(('purple', 'green'))(i), label=j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

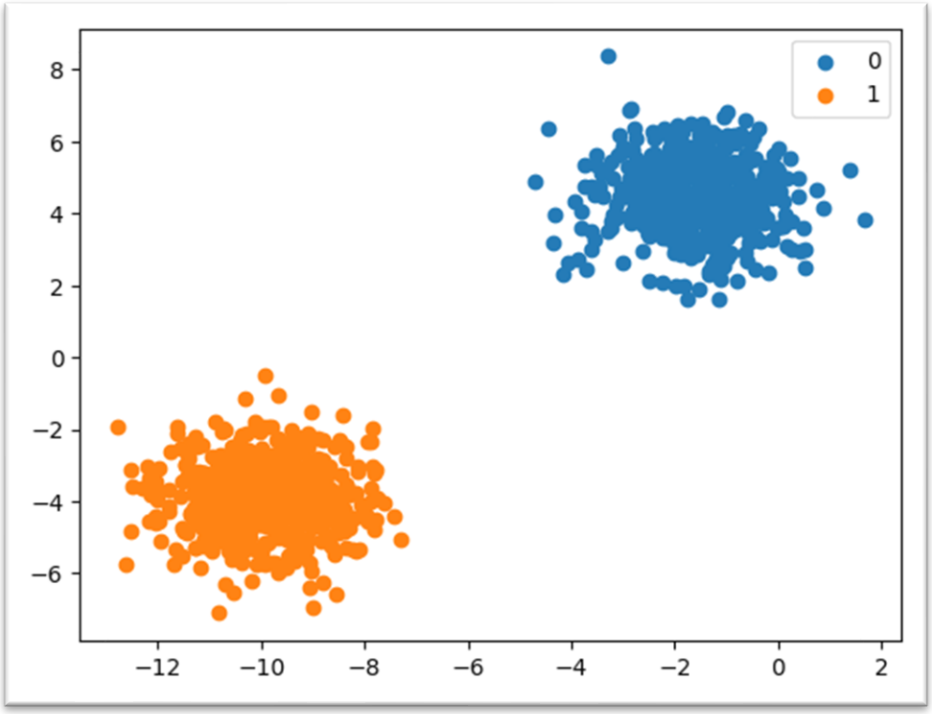
Output:-



Binary classifier:-

```
# Step 1: Import the libraries
from numpy import where
from collections import Counter
from sklearn.datasets import make_blobs
from matplotlib import pyplot as plt

# Step 2: Define the dataset
X, y = make_blobs(n_samples=1000, centers=2, random_state=1)

# Step 3: Shape of the independent and dependent variable
print(X.shape, y.shape)

# Step 4: For class label summarize 1000 samples of inbuilt dataset
counter = Counter(y)
print(counter)

# Step 5: Summarize the first few samples and plot the diagram based on
class label
for i in range(10):
    print(X[i], y[i])

# Plot the dataset and color it by class label
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    plt.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))

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

Output:-



Experiment 5: Bias, Variance, Remove Duplicates, Cross-Validation

Aim:-

To develop a program for bias-variance tradeoff, removing duplicates, and performing cross-validation.

Description:-

Bias-variance tradeoff is key in model generalization. Low bias and variance help to prevent overfitting and underfitting. Cross-validation ensures the model's performance is robust by splitting the data into training and test sets multiple times.

Code:-

```
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

# Generate sample data
np.random.seed(42)
X = np.linspace(0, 10, 100).reshape(-1, 1)
y = 3 * X + np.random.randn(100).reshape(-1, 1)

# Add duplicate samples
X = np.vstack((X, X[:10]))
y = np.vstack((y, y[:10]))

# 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)

# Fit a linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Calculate the training and testing errors (bias and variance)
y_train_pred = model.predict(X_train)
train_error = mean_squared_error(y_train, y_train_pred)
y_test_pred = model.predict(X_test)
test_error = mean_squared_error(y_test, y_test_pred)

print("Training error (bias):", train_error)
print("Testing error (variance):", test_error)

# Remove duplicate samples
X_unique, indices = np.unique(X, axis=0, return_index=True)
y_unique = y[indices]

# Perform cross-validation
cross_val_errors = []
for i in range(5):
```

```
    X_train, X_val, y_train, y_val = train_test_split(X_unique, y_unique,
test_size=0.2, random_state=i)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_val_pred = model.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_pred)
    cross_val_errors.append(val_error)

print("Cross-validation errors:", cross_val_errors)
```

Output:-

```
Training error (bias): 0.9087396281299075
Testing error (variance): 0.4189781106088322
Cross-validation errors: [0.8943963399542353, 0.711838267559308,
0.9664698336127481, 0.9156270854451775, 1.1754593805313762]
Average cross-validation error: 0.932758181420569
```

Experiment 6: Categorical Encoding, One-Hot Encoding

Aim:-

To implement categorical encoding and one-hot encoding techniques for preprocessing categorical data.

Description:-

Categorical encoding converts categorical variables into numerical formats for machine learning models. One-hot encoding represents each category as a binary vector, which helps in handling categorical data for algorithms that require numerical inputs.

Code:-

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Sample dataset
data = {'Color': ['Red', 'Blue', 'Green', 'Red', 'Blue']}
df = pd.DataFrame(data)

# Categorical Encoding
label_encoder = LabelEncoder()
df['Color_Encoded'] = label_encoder.fit_transform(df['Color'])

# One-Hot Encoding
onehot_encoder = OneHotEncoder(sparse_output=False)
onehot_encoded = onehot_encoder.fit_transform(df[['Color_Encoded']])
onehot_df = pd.DataFrame(onehot_encoded, columns=label_encoder.classes_)

# Print the original and encoded dataframes
print("Original DataFrame:")
print(df)
print("\nCategorical Encoded DataFrame:")
print(df[['Color', 'Color_Encoded']])
print("\nOne-Hot Encoded DataFrame:")
print(onehot_df)
```

Output:-

```
Original DataFrame:
   Color  Color_Encoded
0    Red              2
1   Blue              0
2  Green              1
3    Red              2
4   Blue              0

Categorical Encoded DataFrame:
   Color  Color_Encoded
0    Red              2
1   Blue              0
2  Green              1
3    Red              2
4   Blue              0

One-Hot Encoded DataFrame:
   Blue  Green  Red
0   0.0    0.0  1.0
1   1.0    0.0  0.0
2   0.0    1.0  0.0
3   0.0    0.0  1.0
4   1.0    0.0  0.0
```

Experiment 7: Artificial Neural Network using Backpropagation

Aim:-

To build an artificial neural network by implementing the backpropagation algorithm and test it using an appropriate dataset.

Description:-

Backpropagation is a supervised learning algorithm for training artificial neural networks. It computes the gradient of the loss function and adjusts the weights of the network to minimize the error.

Code:-(ANN)

```
import tensorflow as tf
from tensorflow import keras

dataframe = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = dataframe.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)

model.evaluate(x_test, y_test)
```

Output:-

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8555
- loss: 0.4875
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9558
- loss: 0.1491
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9674
- loss: 0.1107
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9735
- loss: 0.0861
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9778
- loss: 0.0724


313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9715 -
loss: 0.0896
[0.07459776848554611, 0.9764000177383423]
```

BACK ANN-

```
import tensorflow as tf
import numpy as np

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y = np.array([[0], [1], [1], [0]], dtype=np.float32)

n_input = 2
n_hidden = 2
n_output = 1

weights = {
    'hidden': tf.Variable(tf.random.normal([n_input, n_hidden])),
    'output': tf.Variable(tf.random.normal([n_hidden, n_output]))
}

biases = {
    'hidden': tf.Variable(tf.random.normal([n_hidden])),
    'output': tf.Variable(tf.random.normal([n_output]))
}

def forward_propagation(x):
    hidden_layer = tf.sigmoid(tf.add(tf.matmul(x, weights['hidden']),
biases['hidden']))
    output_layer = tf.sigmoid(tf.add(tf.matmul(hidden_layer,
weights['output']), biases['output']))
    return output_layer

def backpropagation(x, y):
    with tf.GradientTape() as tape:
        output_layer = forward_propagation(x)
        loss = tf.reduce_mean(0.5 * (y - output_layer) ** 2)

    gradients = tape.gradient(loss, [weights['hidden'], weights['output'],
biases['hidden'], biases['output']])
    optimizer.apply_gradients(zip(gradients, [weights['hidden'],
weights['output'], biases['hidden'], biases['output']]))

optimizer = tf.optimizers.SGD(learning_rate=0.1)
epochs = 10000

for epoch in range(epochs):
    backpropagation(X, y)
    if epoch % 1000 == 0:
        output = forward_propagation(X)
        loss = tf.reduce_mean(0.5 * (y - output) ** 2)
        print(f"Epoch: {epoch}, Loss: {loss}")

predictions = forward_propagation(X)
print("Predictions:")
print(predictions.numpy().round())
```

Output:-

```
Epoch: 0, Loss: 0.16701394319534302
Epoch: 1000, Loss: 0.11679938435554504
Epoch: 2000, Loss: 0.10932266712188721
Epoch: 3000, Loss: 0.09658908098936081
Epoch: 4000, Loss: 0.0756923258304596
Epoch: 5000, Loss: 0.05234337970614433
Epoch: 6000, Loss: 0.03388703614473343
Epoch: 7000, Loss: 0.022483371198177338
Epoch: 8000, Loss: 0.01583809219300747
Epoch: 9000, Loss: 0.011820018291473389

Predictions:
[[0.]
 [1.]
 [1.]
 [0.]]
```

Experiment 8: k-Nearest Neighbor Algorithm

Aim:-

To implement the k-Nearest Neighbor algorithm to classify the Iris dataset and print both correct and wrong predictions.

Description:-

k-Nearest Neighbor (k-NN) is a simple, non-parametric algorithm used for classification. It assigns the class based on the majority class of the k nearest neighbors in the feature space.

Code:-

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Load Iris dataset
data_set = sns.load_dataset('iris')

# Splitting data into features (X) and target variable (y)
X = data_set.iloc[:, :-1]
y = data_set.iloc[:, -1]   # Target (species)

# Splitting dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=0)

# Initialize and train the KNN classifier
classifier = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)

# Predict the target variable on the test set
y_pred = classifier.predict(X_test)

# Displaying comparison of actual vs predicted values
i = 0
print('-'*80)
print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label',
'Correct/Wrong'))
print('-'*80)
for label in y_test:
    print ('%-25s %-25s' % (label, y_pred[i]), end="")
    if label == y_pred[i]:
        print ('%-25s' % ('Correct'))
    else:
        print ('%-25s' % ('Wrong'))
    i += 1
print('-'*80)


# Confusion Matrix
print("\nConfusion Matrix:\n", metrics.confusion_matrix(y_test, y_pred))
```

```
# Classification Report
print('-'*80)
print("\nClassification Report:\n", metrics.classification_report(y_test,
y_pred))

# Accuracy of the classifier
print('-'*80)
print('Accuracy of the classifier is %0.2f' % metrics.accuracy_score(y_test,
y_pred))
print('-'*80)
```

Output:-

```
--------------------------------------------------------------------------------
Original Label            Predicted Label           Correct/Wrong
--------------------------------------------------------------------------------
virginica                 virginica                 Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
virginica                 virginica                 Correct
setosa                    setosa                    Correct
virginica                 virginica                 Correct
setosa                    setosa                    Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
virginica                 virginica                 Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
setosa                    setosa                    Correct
virginica                 virginica                 Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
setosa                    setosa                    Correct
virginica                 virginica                 Correct
setosa                    setosa                    Correct
setosa                    setosa                    Correct
versicolor                versicolor                Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
virginica                 virginica                 Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
```

```
virginica                 virginica                 Correct
virginica                 virginica                 Correct
versicolor                versicolor                Correct
setosa                    setosa                    Correct
versicolor                virginica                 Wrong
-------------------------------------------------------------------------------


Confusion Matrix:
 [[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]
-------------------------------------------------------------------------------


Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.94      0.97        16
   virginica       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38


-------------------------------------------------------------------------------
Accuracy of the classifer is 0.97
-------------------------------------------------------------------------------
```

Experiment 9: Locally Weighted Regression

Aim:

To implement the non-parametric Locally Weighted Regression algorithm to fit data points and visualize the results.

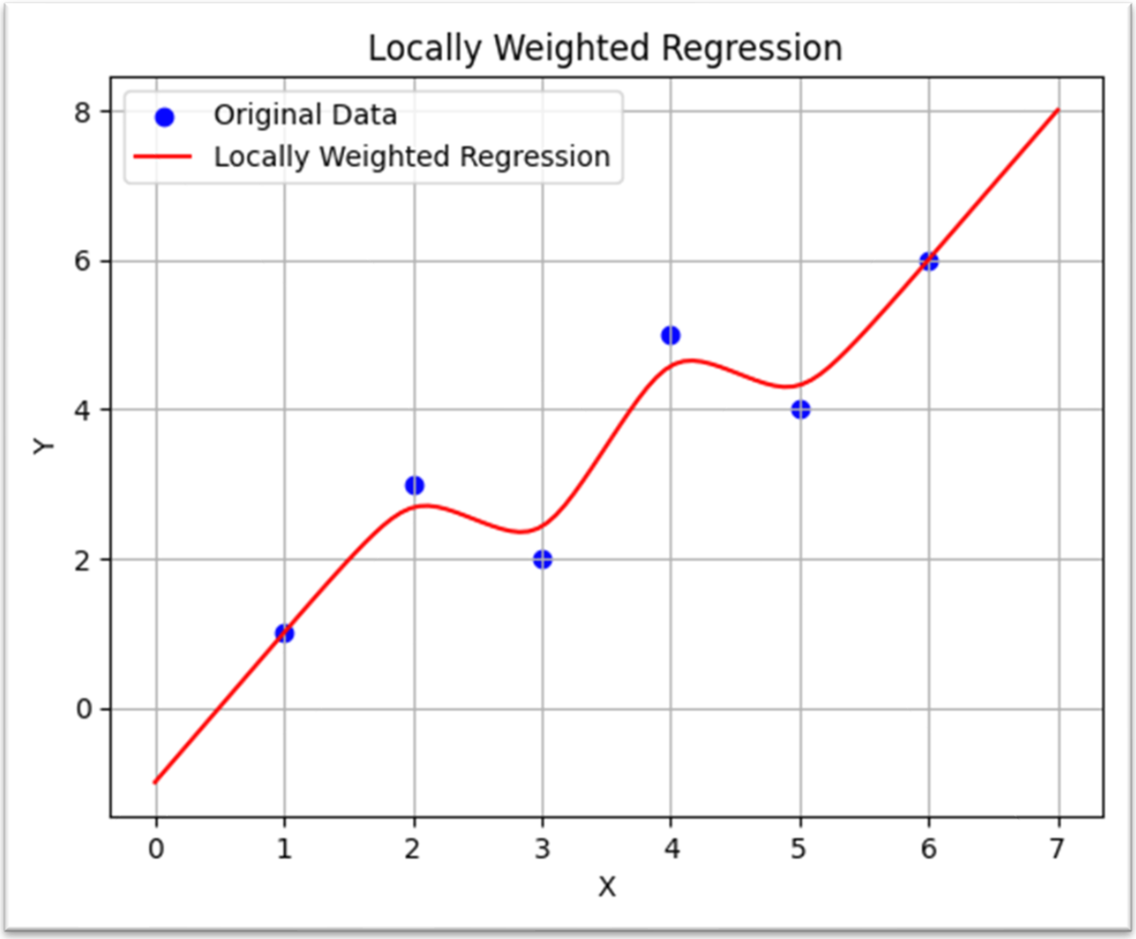
Description:-

Locally Weighted Regression is a type of regression that assigns weights to nearby points and fits a regression line only to those points. It is useful for handling nonlinear relationships in the data.

Code:-

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_kernel(x, xi, tau):
    return np.exp((x - xi)**2 / (-2 * tau**2))

def locally_weighted_regression(X_train, y_train, x_query, tau):
    m = len(X_train)
    X = np.column_stack((np.ones(m), X_train))
    W = np.diag([gaussian_kernel(x_query, xi, tau) for xi in X_train])
    theta = np.linalg.inv(X.T @ W @ X) @ X.T @ W @ y_train
    return theta

# Prepare the dataset
X_train = np.array([1, 2, 3, 4, 5, 6])
y_train = np.array([1, 3, 2, 5, 4, 6])

# Choose query points for prediction
x_query = np.linspace(0, 7, 100)

# Choose tau
tau = 0.5

# Perform locally weighted regression for each query point
y_pred = []
for x in x_query:
    theta = locally_weighted_regression(X_train, y_train, x, tau)
    y_pred.append(theta[0] + theta[1] * x)

# Plot the original data points and the fitted curve
plt.scatter(X_train, y_train, color='blue', label='Original Data')
plt.plot(x_query, y_pred, color='red', label='Locally Weighted Regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Locally Weighted Regression')
plt.legend()
plt.grid(True)
plt.show()
```

Output:-


Locally Weighted Regression
Original Data
Locally Weighted Regression
8
6
4
2
0
Y
0
1
2
3
4
5
6
7
X

Experiment 10: Naïve Bayesian Classifier

Aim:-

To use the Naïve Bayesian Classifier to classify documents and calculate accuracy, precision, and recall.

Description:-

Naïve Bayes is a probabilistic classifier based on Bayes' theorem. It assumes the features are conditionally independent given the class and is widely used for text classification tasks.

Code:-

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Load data from CSV
data = pd.read_csv('tennis.csv')
data

# Obtain Train data and Train output
X = data.iloc[:, :-1]  # Features
y = data.iloc[:, -1]   # Target

# Encode categorical variables into numbers
le_outlook = LabelEncoder()
X['Outlook'] = le_outlook.fit_transform(X['Outlook'])

le_Temperature = LabelEncoder()
X['Temperature'] = le_Temperature.fit_transform(X['Temperature'])

le_Humidity = LabelEncoder()
X['Humidity'] = le_Humidity.fit_transform(X['Humidity'])

le_Windy = LabelEncoder()
X['Windy'] = le_Windy.fit_transform(X['Windy'])

le_PlayTennis = LabelEncoder()
y = le_PlayTennis.fit_transform(y)

# 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 Naive Bayesian Classifier
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = classifier.predict(X_test)
y_pred

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate precision, recall, f1-score, and support
report = classification_report(y_test, y_pred)
```

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

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

Dataset:-

```
Outlook Temperature Humidity    Windy   PlayTennis
0   Sunny   Hot High    False   No
1   Sunny   Hot High    True    No
2   Overcast    Hot High    False   Yes
3   Rainy   Mild    High    False   Yes
4   Rainy   Cool    Normal  False   Yes
5   Rainy   Cool    Normal  True    No
6   Overcast    Cool    Normal  True    Yes
7   Sunny   Mild    High    False   No
8   Sunny   Cool    Normal  False   Yes
9   Rainy   Mild    Normal  False   Yes
10  Sunny   Mild    Normal  True    Yes
11  Overcast    Mild    High    True    Yes
12  Overcast    Hot Normal  False   Yes
13  Rainy   Mild    High    True    No
```

Output:-

```
    Outlook Temperature Humidity    Windy   PlayTennis
0   Sunny   Hot High    False   No
1   Sunny   Hot High    True    No
2   Overcast    Hot High    False   Yes
3   Rainy   Mild    High    False   Yes
4   Rainy   Cool    Normal  False   Yes
5   Rainy   Cool    Normal  True    No
6   Overcast    Cool    Normal  True    Yes
7   Sunny   Mild    High    False   No
8   Sunny   Cool    Normal  False   Yes
9   Rainy   Mild    Normal  False   Yes
10  Sunny   Mild    Normal  True    Yes
11  Overcast    Mild    High    True    Yes
12  Overcast    Hot Normal  False   Yes
13  Rainy   Mild    High    True    No

array([1, 1, 0])

Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Confusion Matrix:
 [[1 0]
 [0 2]]
```

Experiment 11: EM Algorithm and k-Means Clustering

Aim:-

To apply the EM algorithm and k-Means clustering to the Heart Disease dataset and compare the results.

Description:-

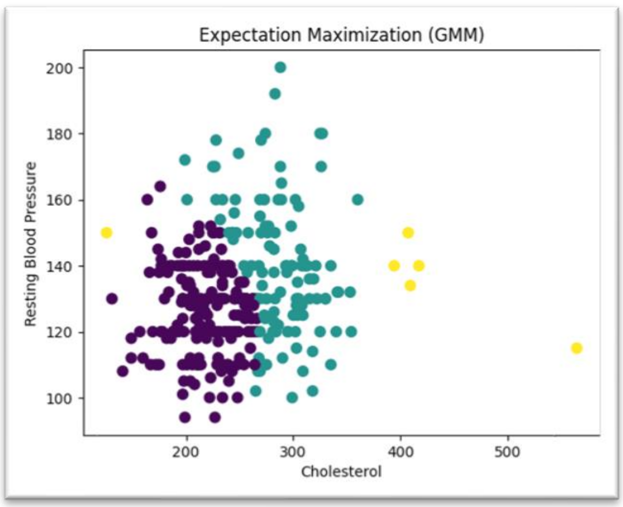
The Expectation-Maximization (EM) algorithm is used for finding the maximum likelihood in the presence of latent variables, while k-Means is a popular clustering method. The two methods are compared to evaluate the quality of clustering.

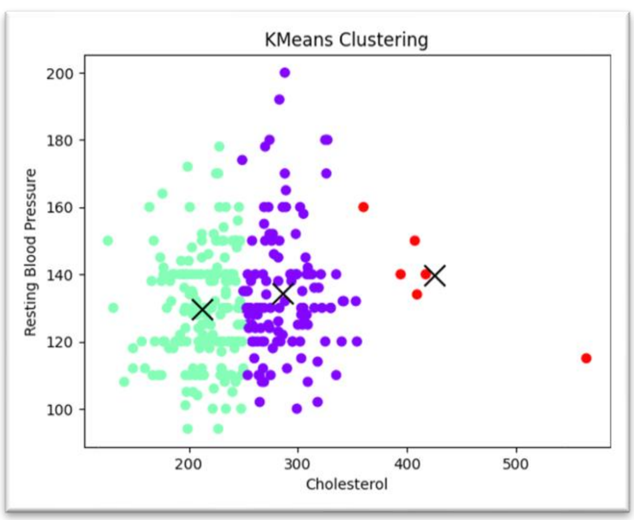
Code:-

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
import pandas as pd

# Load the dataset
X = pd.read_csv("Heart.csv")

# Extract features 'chol' (cholesterol) and 'trestbps' (resting blood
pressure)
x1 = X['chol'].values
x2 = X['trestbps'].values

# Reshape the data into a 2D array for clustering
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)

# ------------------------- Expectation Maximization (GMM) ------------------
gmm = GaussianMixture(n_components=3)
gmm.fit(X)

# Get EM predictions and print results
em_predictions = gmm.predict(X)
print("\nEM Predictions:")
print(em_predictions)
print("Mean:\n", gmm.means_)
print("\nCovariances:\n", gmm.covariances_)

# Plot the EM clustering results
plt.title('Expectation Maximization (GMM)')
plt.scatter(X[:, 0], X[:, 1], c=em_predictions, s=50)
plt.xlabel('Cholesterol')
plt.ylabel('Resting Blood Pressure')
plt.show()

# ------------------------- KMeans Clustering -------------------------------
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
# Print KMeans results
print("KMeans Cluster Centers:\n", kmeans.cluster_centers_)
```

```
print("KMeans Cluster Labels:\n", kmeans.labels_)

# Plot the KMeans clustering results
plt.title('KMeans Clustering')
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
color='black', marker='x', s=200)
plt.xlabel('Cholesterol')
plt.ylabel('Resting Blood Pressure')
plt.show()
```

Output:-





Experiment 12: Exploratory Data Analysis using Pandas or Matplotlib

Aim:-

To perform exploratory data analysis (EDA) for classification using Pandas or Matplotlib.

Description:-

EDA is the process of analyzing data sets to summarize their main characteristics using visualization tools like Pandas and Matplotlib. It is used to detect patterns, spot anomalies, and test hypotheses.

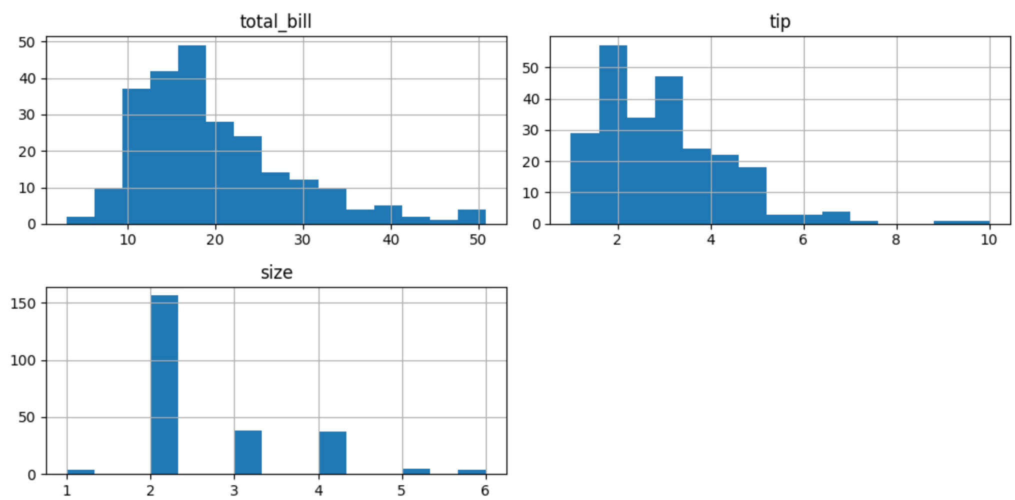
Code:-

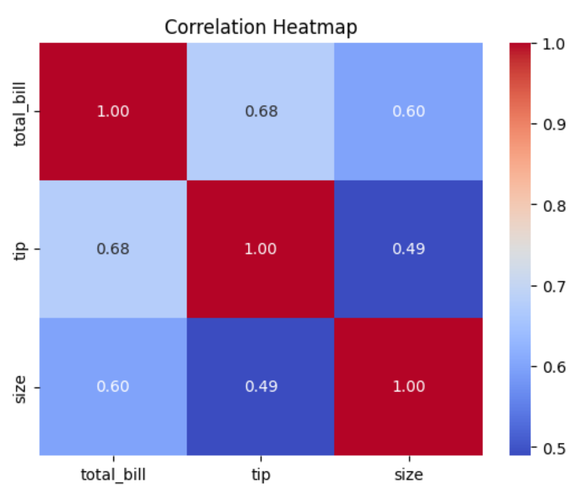
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Load the Tips dataset
tips = sns.load_dataset('tips')
# 1. Basic information
print(tips.info(), tips.describe())
# 2. Check for missing values
print("Missing values:\n", tips.isnull().sum())
# 3. Visualize distributions of numerical features
tips[['total_bill', 'tip', 'size']].hist(bins=15, figsize=(10, 5))
plt.tight_layout()
plt.show()
# 4. Correlation heatmap for numeric columns
sns.heatmap(tips[['total_bill', 'tip', 'size']].corr(), annot=True,
fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
# 5. Boxplot for tips by day
sns.boxplot(x='day', y='tip', data=tips)
plt.title('Boxplot of Tips by Day')
plt.show()
# 6. Scatter plot for total bill vs. tip
sns.scatterplot(x='total_bill', y='tip', hue='day', style='time', data=tips)
plt.title('Total Bill vs. Tip')
plt.xlabel('Total Bill')
plt.ylabel('Tip')
plt.show()
```

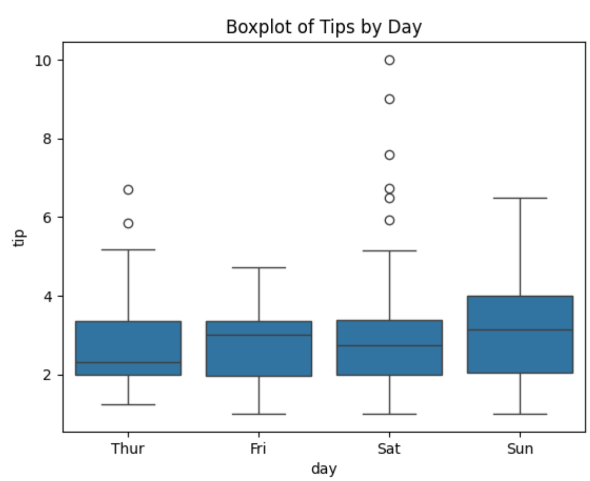
Output:-

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
None         total_bill         tip         size
```

```
count  244.000000  244.000000  244.000000
mean    19.785943    2.998279    2.569672
std      8.902412    1.383638    0.951100
min      3.070000    1.000000    1.000000
25%     13.347500    2.000000    2.000000
50%     17.795000    2.900000    2.000000
75%     24.127500    3.562500    3.000000
max     50.810000   10.000000    6.000000
Missing values:
 total_bill    0
tip           0
sex           0
smoker        0
day           0
time          0
size          0
dtype: int64
```







Experiment 13: Bayesian Network for Heart Disease Diagnosis

Aim:-

To construct a Bayesian network considering medical data and demonstrate the diagnosis of heart patients using the Heart Disease dataset.

Description:-

A Bayesian network is a probabilistic graphical model representing a set of variables and their conditional dependencies. It is used here to model the relationships between symptoms and diseases for diagnosing heart patients.

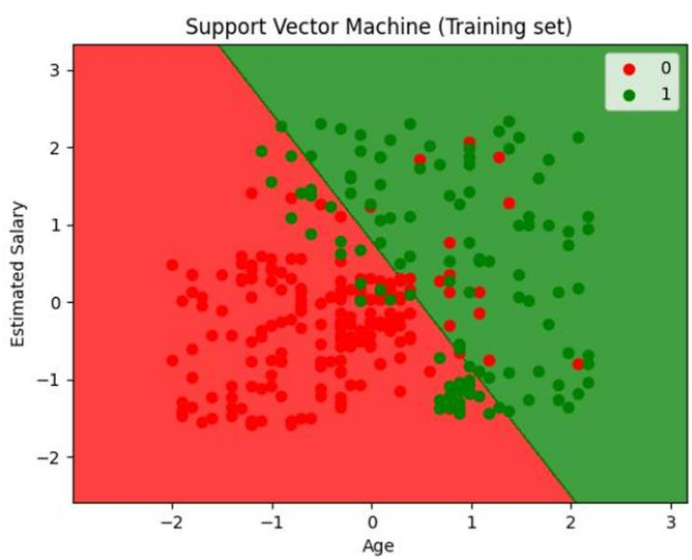
Code:-

```
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

Data = pd.read_csv("Heart.csv")
Heart_disease = pd.DataFrame(Data)

print(Heart_disease)

Model = BayesianNetwork([
    ('Age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholesterol'),
    ('Lifestyle', 'diet'),
    ('cholesterol', 'heartdisease'),
])

Model.fit(Heart_disease, estimator=MaximumLikelihoodEstimator)

HeartDisease_infer = VariableElimination(Model)

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, 
Youth:3, Teen:4 }')
print('For Gender Enter { Male:0, Female:1 }')
print('For Family History Enter { yes:1, No:0 }')
print('For diet Enter { High:0, Medium:1 }')
print('For lifestyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 
}')
print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

age = int(input('Enter age: '))
gender = int(input('Enter Gender: '))
family = int(input('Enter Family history: '))
diet = int(input('Enter diet: '))
lifestyle = int(input('Enter Lifestyle: '))
cholesterol = int(input('Enter cholesterol: '))

Q = HeartDisease_infer.query(variables=['heartdisease'], evidence={
    'Age': age,
    'Gender': gender,
    'Family': family,
    'diet': diet,
    'Lifestyle': lifestyle,
```

```
    'cholesterol': cholesterol
})

print(Q['heartdisease'])
```

Output:-

```
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2,
Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
Enter age :1
Enter Gender :1
Enter Family history :0
Enter diet :1
Enter Lifestyle :0
Enter cholestrol :1
+----------------+---------------------+
| heartdisease   |   phi(heartdisease) |
+================+=====================+
| heartdisease_0 |              0.0000 |
+----------------+---------------------+
| heartdisease_1 |              1.0000 |
+----------------+---------------------+
```



Experiment 14: Support Vector Machines and Principal Component Analysis

Aim:-

To implement Support Vector Machines (SVM) and Principal Component Analysis (PCA) for dimensionality reduction.

Description:-

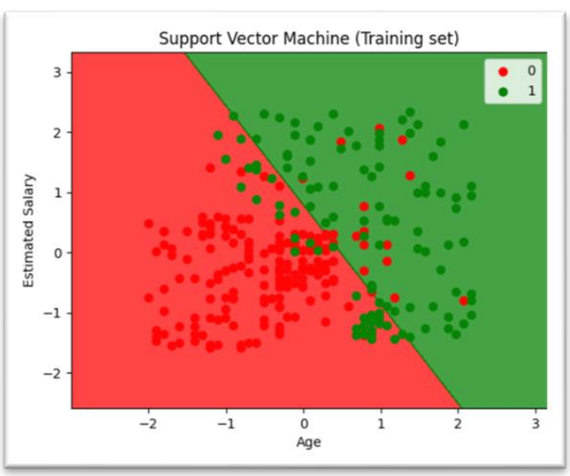
SVM is a supervised learning algorithm for classification tasks, and PCA is a technique used for reducing the dimensionality of data by projecting it onto a lower-dimensional space while retaining the most important information.

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data_set= pd.read_csv('car_data.csv')
data_set

x = data_set.iloc[:, [2,3]].values
y = data_set.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x, y, test_size = 0.25,
random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)


# Fitting the classifier into the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_Train, Y_Train)

# Predicting the test set results
Y_Pred = classifier.predict(X_Test)
Y_Pred

# Making the Confusion Matrix

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_Test, Y_Pred)
cm

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Train, Y_Train
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop =
X_Set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop =
X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
```

```
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Support Vector Machine (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap
X_Set, Y_Set = X_Test, Y_Test
X1, X2 = np.meshgrid(np.arange(start = X_Set[:, 0].min() - 1, stop =
X_Set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_Set[:, 1].min() - 1, stop =
X_Set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(Y_Set)):
    plt.scatter(X_Set[Y_Set == j, 0], X_Set[Y_Set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Support Vector Machine (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

Output:-



Experiment 15: Principal Component Analysis

Aim:-

To implement Principal Component Analysis (PCA) to reduce the dimensionality of the dataset.

Description:-

PCA is a statistical technique that transforms the dataset into a set of linearly uncorrelated variables, called principal components. It is widely used for feature reduction in machine learning.

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from mpl_toolkits.mplot3d import Axes3D

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4D)
y = iris.target  # Labels (3 classes)

# Step 1: Standardize the data (mean center it)
X_centered = X - np.mean(X, axis=0)

# Step 2: Compute the covariance matrix
cov_matrix = np.cov(X_centered, rowvar=False)

# Step 3: Perform eigen decomposition of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

# Step 4: Sort the eigenvalues and eigenvectors
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvectors = eigenvectors[:, sorted_indices]
top_eigenvectors = eigenvectors[:, :2]  # Select top 2 eigenvectors for 2D
projection

# Step 5: Project the data onto the 2D principal components
X_2d = np.dot(X_centered, top_eigenvectors)

# Step 6: Plot the original 3D data (using first 3 features as proxies for
3D space)
fig = plt.figure(figsize=(14, 6))
ax = fig.add_subplot(121, projection='3d')

# Colors and markers for each class
colors = ['red', 'green', 'blue']
markers = ['o', 's', '^']
labels = iris.target_names

# Scatter plot for each class with different colors and markers
for i in range(3):
    ax.scatter(X[y == i, 0], X[y == i, 1], X[y == i, 2],   color=colors[i],
marker=markers[i], label=labels[i], alpha=0.6)

# Set labels and title for 3D plot
```

```
ax.set_title('Original 3D Data (First 3 Features)')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.legend()

# Step 7: Plot the reduced 2D data with different colors and markers
ax2 = fig.add_subplot(122)
for i in range(3):
    ax2.scatter(X_2d[y == i, 0], X_2d[y == i, 1],
                color=colors[i], marker=markers[i], label=labels[i],
alpha=0.6)

# Set labels and title for 2D plot
ax2.set_title('Projected 2D Data after PCA')
ax2.set_xlabel('Principal Component 1')
ax2.set_ylabel('Principal Component 2')
ax2.legend()
plt.show()
```

Output:-