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.

Aim: To 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.

Program:

```
import pandas as pd
import numpy as np

# Load the CSV file
data = pd.read_csv('data.csv')

# Extract the attributes (concepts) and target values
concepts = np.array(data)[:,:-1]
target = np.array(data)[:,-1]

print("\nConcepts:")
print(concepts)

print("\nTarget:")
print(target)

# Function to implement FIND-S algorithm with hypothesis printing at each step
def train(con, tar):
    # Initialize the specific hypothesis with the first positive example
    for i, val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            print(f"\nInitial hypothesis from first positive example: {specific_h}")
            break
    # Update the hypothesis based on other positive examples
    for i, val in enumerate(con):
        if tar[i] == 'yes':
            for x in range(len(specific_h)):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
            print(f"\nHypothesis after example {i+1}: {specific_h}")
    return specific_h

# Apply the FIND-S algorithm to the data
specific_hypothesis = train(concepts, target)

# Display the most specific hypothesis
print("\nThe most specific hypothesis is:", specific_hypothesis)
```

Data Set:

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:

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.

Aim: 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.

Program:

```
import numpy as np
import pandas as pd

# Load the dataset from a CSV file
data = pd.read_csv('2.csv')

# Extract the concepts (features) and target (labels)
concepts = np.array(data.iloc[:, 0:-1])
target = np.array(data.iloc[:, -1])

# Function to learn the concept using the Candidate Elimination algorithm
def learn(concepts, target):
    # Initialize specific hypothesis as the first positive instance
    specific_h = concepts[0].copy()
    print("Initialization of specific_h \n", specific_h)

    # Initialize general hypothesis to the most general hypothesis
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("Initialization of general_h \n", general_h)

    # Iterate through all instances
    for i, h in enumerate(concepts):
        if target[i] == "yes":
            print("Instance is Positive")
            # Update specific hypothesis for positive instance
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
```

```
                specific_h[x] = '?'  # Generalize the hypothesis
                general_h[x][x] = '?'
        elif target[i] == "no":
            print("Instance is Negative")
            # Update general hypothesis for negative instance
            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 the intermediate steps
        print(f"Step {i + 1}")
        print(specific_h)
        print(general_h)
        print("\n")

    # Remove redundant hypotheses from the general hypothesis
    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h

# Run the learning algorithm
s_final, g_final = learn(concepts, target)

# Output the final specific and general hypotheses
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

Data Set:

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:

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.

Aim: To Implement 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.

Program:

```
import pandas as pd
from pprint import pprint
from sklearn.feature_selection import mutual_info_classif
from collections import Counter

# ID3 algorithm implementation
def id3(df, target_attribute, attribute_names, default_class=None):
    cnt = Counter(x for x in df[target_attribute])

    # If the target attribute has only one unique value, return that value
    if len(cnt) == 1:
        return next(iter(cnt))

    # If the dataset is empty or attribute_names list is empty, return the default class
    elif df.empty or (not attribute_names):
        return default_class

    else:
        # Calculate information gain for each attribute
        gainz=mutual_info_classif(df[attribute_names],df[target_attribute],
discrete_features=True)
        index_of_max = gainz.tolist().index(max(gainz))
        best_attr = attribute_names[index_of_max]

        # Create a new decision tree node with the best attribute
        tree = {best_attr: {}}
    # Remove the best attribute from the list of attributes
    remaining_attribute_names = [i for i in attribute_names if i != best_attr]

    # Recursively create subtrees for each value of the best attribute
    for attr_val, data_subset in df.groupby(best_attr):
        subtree = id3(data_subset, target_attribute, remaining_attribute_names, default_class)
        tree[best_attr][attr_val] = subtree
    return tree

# Function to classify a new sample
def classify(tree, sample):
    if not isinstance(tree, dict):
        return tree
    attr = next(iter(tree))
    if sample[attr] in tree[attr]:
        return classify(tree[attr][sample[attr]], sample)
    else:
        return None

# Create the dataset from the provided data
data = {
    "Outlook": ["Sunny", "Sunny", "Overcast", "Rain", "Rain", "Rain", "Overcast", "Sunny",
"Sunny", "Rain", "Sunny", "Overcast", "Overcast"],
    "Temperature": ["Hot", "Hot", "Hot", "Mild", "Mild", "Mild", "Mild", "Hot", "Mild",
"Mild", "Overcast", "Hot", "Hot"],
    "Humidity": ["High", "High", "High", "High", "Normal", "Normal", "Normal", "Normal",
"High", "High", "High", "Normal", "High"],
    "Windy": ["FALSE", "TRUE", "FALSE", "FALSE", "FALSE", "TRUE", "TRUE",
"FALSE", "FALSE", "TRUE", "TRUE", "FALSE", "TRUE"],
    "PlayTennis": ["No", "No", "Yes", "Yes", "Yes", "No", "Yes", "No", "Yes", "No", "Yes",
"Yes", "Yes"]
}

df = pd.DataFrame(data)
# Extract attribute names and remove the target attribute
```

```
attribute_names = df.columns.tolist()
attribute_names.remove("PlayTennis")
# Factorize categorical columns and store the mappings
factor_mappings = {}
for colname in df.select_dtypes("object"):
    df[colname], mapping = df[colname].factorize()
    factor_mappings[colname] = mapping
# Print the factorized dataset
print("Factorized dataset:")
print(df)
# Build the ID3 decision tree
tree = id3(df, "PlayTennis", attribute_names)

# Print the resulting tree structure
print("The tree structure:")
pprint(tree)

# Define a new sample to classify
new_sample = {
    "Outlook": "Sunny",
    "Temperature": "Hot",
    "Humidity": "High",
    "Windy": "FALSE"
}

# Factorize the new sample based on the existing factor mappings
for colname in new_sample:
    new_sample[colname] = factor_mappings[colname].tolist().index(new_sample[colname])
# Classify the new sample
classification = classify(tree, new_sample)
print(f"The classification for the new sample is: {'Yes' if classification == 1 else 'No' if
classification == 0 else 'Unknown'}")
```

Output:

Experiment-4

Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression

Aim: To solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression

Program:

Simple Linear Regression

```
# importing the dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataset = pd.read_csv('Salary_Data.csv')
dataset.head()

# data preprocessing
X = dataset.iloc[:, :-1].values  #independent variable array
y = dataset.iloc[:,1].values  #dependent variable vector

# splitting the dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=1/3,random_state=0)

# fitting the regression model
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train) #actually produces the linear eqn for the data

# predicting the test set results
y_pred = regressor.predict(X_test)
y_pred

y_test

# visualizing the results
#plot for the TRAIN

plt.scatter(X_train, y_train, color='red') # plotting the observation line
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Training set)") # stating the title of the graph

plt.xlabel("Years of experience") # adding the name of x-axis
plt.ylabel("Salaries") # adding the name of y-axis
plt.show() # specifies end of graph

#plot for the TEST

plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue') # plotting the regression line
plt.title("Salary vs Experience (Testing set)")

plt.xlabel("Years of experience")
plt.ylabel("Salaries")
plt.show()
```

Output:

Multiple Linear Regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load your dataset
data = pd.read_csv('mlr_dataset.csv')

# Prepare the input (X) and output (y) variables
X = data[['age', 'experience']]
y = data['salary']

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

# Create and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

Output:

b) Logistic Regression

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

data = pd.DataFrame({
    'X1': [1, 2, 3, 4, 5, 6],
    'X2': [2, 3, 4, 5, 6, 7],
    'Y': [0, 0, 0, 1, 1, 1]
})

X = data[['X1', 'X2']]
y = data['Y']

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

model = GaussianNB()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

Output:

Experiment-5

Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Aim: Develop a program for Bias, Variance, Remove duplicates, Cross Validation

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# 1. Importing the dataset
dataset = pd.read_csv('Salary_Data.csv')

# Remove duplicates from the dataset
dataset = dataset.drop_duplicates()

# 2. Data Preprocessing
X = dataset.iloc[:, :-1].values  # Independent variable array
y = dataset.iloc[:, 1].values    # Dependent variable vector

# 3. Splitting the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=1/3,random_state=0)

# 4. Fitting the regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)  # Fit the model

# 5. Predicting the test set results
y_pred = regressor.predict(X_test)

# 6. Calculate Bias
def calculate_bias(y_test, y_pred):
    return np.mean(y_pred - y_test)

# 7. Calculate Variance
def calculate_variance(y_pred):
    return np.var(y_pred)

# 8. Cross-Validation with Bias and Variance Calculation
def cross_validation(model, X, y, cv=5):
    kf = KFold(n_splits=cv, shuffle=True, random_state=0)
    biases, variances, mse_scores = [], [], []

    for train_idx, test_idx in kf.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        biases.append(calculate_bias(y_test, y_pred))
        variances.append(calculate_variance(y_pred))
        mse_scores.append(mean_squared_error(y_test, y_pred))

    return np.mean(biases), np.mean(variances), np.mean(mse_scores)

# Example: Cross-validation on the model
avg_bias, avg_variance, avg_mse = cross_validation(regressor, X, y)

# Output the results
print("Bias on Test Set:", calculate_bias(y_test, y_pred))
print("Variance on Test Set:", calculate_variance(y_pred))
print("Average Bias across CV:", avg_bias)
print("Average Variance across CV:", avg_variance)
print("Average MSE across CV:", avg_mse)

# 9. Visualizing the training set results (optional)
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()

# 10. Visualizing the test set results (optional)
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Test set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Output:

Experiment-6

Write a program to implement Categorical Encoding, One-hot Encoding

Aim: To Develop a program to implement Categorical Encoding, One-hot Encoding

Program:

Label Encoding:

```
from sklearn.preprocessing import LabelEncoder
# Create a sample dataset
fruits = ['Apple', 'Orange', 'Banana', 'Carrot', 'Tomato', 'Potato']
fruit_types = ['Fruit', 'Fruit', 'Fruit', 'Vegetable', 'Vegetable', 'Vegetable']
# Create a LabelEncoder object
le = LabelEncoder()
# Fit and transform the categorical data
encoded_types = le.fit_transform(fruit_types)
# Print the original and encoded data
print('Original Data:', fruit_types)
print('Encoded Data:', encoded_types)
```

Output:

OneHotEncoding:

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

# Sample data
data = np.array(['red', 'blue', 'green', 'blue', 'red']).reshape(-1, 1)

# Initialize OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=False)

# Fit and transform the data
encoded_data = onehot_encoder.fit_transform(data)

print(encoded_data)
```

Output:

Experiment-7

Write a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions

Aim: To implement a program to implement k-Nearest Neighbor algorithm to classify the iris data set. Print both correct and wrong predictions.

Program:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Define column names for the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read the dataset into a pandas DataFrame
dataset = pd.read_csv("/content/8-dataset.csv", names=names)

# Split the dataset into features (X) and target (y)
X = dataset.iloc[:, :-1]  # All columns except the last one as features
y = dataset.iloc[:, -1]   # The last column as the target

# Print the first few rows of the features to verify data
print(X.head())

# Split the dataset into training and testing sets
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.10, random_state=42)

# Create and train the K-Nearest Neighbors (KNN) classifier
classifier = KNeighborsClassifier(n_neighbors=5).fit(Xtrain, ytrain)

# Predict the labels for the test set
ypred = classifier.predict(Xtest)

# Initialize a counter for indexing
i = 0
print("\n-------------------------------------------------------------------------")
print('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))
print("-------------------------------------------------------------------------")

# Compare each original label with its prediction and print the results
for label in ytest:
    print('%-25s %-25s' % (label, ypred[i]), end="")
    if label == ypred[i]:
        print('%-25s' % ('Correct'))
    else:
        print('%-25s' % ('Wrong'))
    i = i + 1

print("-------------------------------------------------------------------------")
```

```
# Display the confusion matrix
print("\nConfusion Matrix:\n", metrics.confusion_matrix(ytest, ypred))
print("----------------------------------------------------------------------")

# Display the classification report with precision, recall, and F1-score
print("\nClassification Report:\n", metrics.classification_report(ytest, ypred))
print("----------------------------------------------------------------------")

# Display the accuracy of the classifier
print('Accuracy of the classifier is %0.2f' % metrics.accuracy_score(ytest, ypred))
print("----------------------------------------------------------------------")
```

Output:

EXPERIMENT-8

Write a program to implement k-Means algorithm

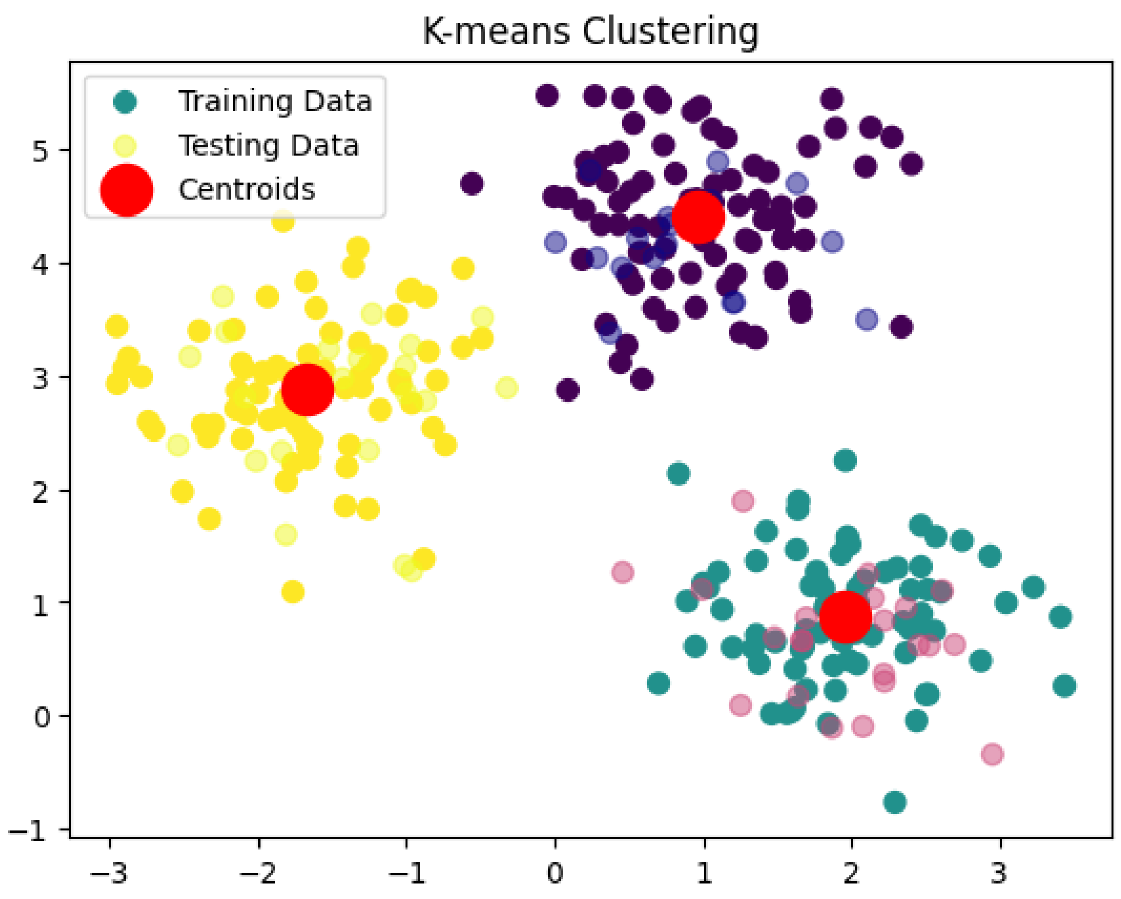
Aim: To implement a program to implement k-Means algorithm

Program:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
# Step 2: Import Dataset
# Load your dataset here. For demonstration, let's create a synthetic dataset.
from sklearn.datasets import make_blobs

# Generate sample data
data, labels = make_blobs(n_samples=300, n_features=2, centers=4, random_state=42)

# Convert the data into a Pandas DataFrame for easier viewing
df = pd.DataFrame(data)

# View the first 5 rows of the generated data
df.head()
# Create a dataset with 3 clusters
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.60, random_state=0)
# Step 3: Split Dataset into Training and Testing Sets
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
# Step 4: Fit the Model
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_train)
# Step 5: Predict the Model
y_train_pred = kmeans.predict(X_train)
y_test_pred = kmeans.predict(X_test)
# Step 6: Metrics
# Calculate silhouette score for both training and testing sets
train_silhouette = silhouette_score(X_train, y_train_pred)
```

```
test_silhouette = silhouette_score(X_test, y_test_pred)
# Print results
print(f"Training Silhouette Score: {train_silhouette:.2f}")
print(f"Testing Silhouette Score: {test_silhouette:.2f}")
Training Silhouette Score: 0.66
Testing Silhouette Score: 0.64
# Plotting the clusters
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train_pred, s=50, cmap='viridis', label='Training
Data')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test_pred, s=50, cmap='plasma', label='Testing
Data', alpha=0.5)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red',
label='Centroids')
plt.title('K-means Clustering')
plt.legend()
plt.show()
```

OUTPUT:



Experiment-9

Exploratory Data Analysis for Classification using Pandas or Matplotlib.
Aim: Exploratory Data Analysis for Classification using Pandas or Matplotlib.

EDA is a phenomenon under data analysis used for gaining a better understanding of data aspects like:

- main features of data
- variables and relationships that hold between them
- Identifying which variables are important for our problem

Step 1: Importing Required Libraries

```
# importting Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings as wr
wr.filterwarnings('ignore')
```

Understanding and experimenting with our data using libraries is the first step in utilizing Python for machine learning.

Step 2: Reading Dataset

```
# loading and reading dataset
df = pd.read_csv("winequality-red.csv")
print(df.head())
```

Output:

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076
   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56
   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

Step 3: Analyzing the Data

Gaining general knowledge about the data—including its values, kinds, number of rows and columns, and missing values—is the primary objective of data understanding.

shape: shape will show how many features (columns) and observations (rows) there are in the dataset.

```
# shape of the data
```

```
df.shape
```
Output:
```
(1599, 12)
```

info() facilitates comprehension of the data type and related information, such as the quantity of records in each column, whether the data is null or not, the type of data, and the dataset's memory use.

```
#data information
df.info()
```

Output:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
# describing the data
df.describe()
```

Output:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count  1599.000000    1599.000000       1599.000000  599.000000
mean   8.319637       0.527821          0.270976     2.538806
std    1.741096       0.179060          0.194801     1.409928
min    4.600000       0.120000          0.000000     0.900000
25%    7.100000       0.390000          0.090000     1.900000
50%    7.900000       0.520000          0.260000     2.200000
75%    9.200000       0.640000          0.420000     2.600000
max    15.900000      1.580000          1.000000     15.500000

       chlorides    free sulfur dioxide  total sulfur dioxide  density  \
count  1599.000000  1599.000000          1599.000000           1599.000000
mean    0.087467    15.874922            46.467792             0.996747
std    0.047065     10.460157            32.895324             0.001887
min    0.012000     1.000000             6.000000              0.990070
25%    0.070000     7.000000             22.000000             0.995600
```

```
50%     0.079000        14.000000        38.000000        0.996750
75%     0.090000        21.000000        62.000000        0.997835
max     0.611000        72.000000       289.000000        1.003690
                 pH       sulphates      alcohol
count       1599.000000  1599.000000  1599.000000
mean           3.311113     0.658149    10.422983
std            0.154386     0.169507     1.065668
min            2.740000     0.330000     8.400000
25%            3.210000     0.550000     9.500000
50%            3.310000     0.620000    10.200000
75%            3.400000     0.730000    11.100000
max            4.010000     2.000000    14.900000
```

The DataFrame "df" is statistically summarized by the code df.describe(), which gives the count, mean, standard deviation, minimum, and quartiles for each numerical column. The dataset's central tendencies and spread are briefly summarized.

Checking Columns

```
#column to list
df.columns.tolist()
```

Output:

```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

The code df.columns.tolist() converts the column names of the DataFrame 'df' into a Python list, providing a convenient way to access and manipulate column names.

Checking Missing Values

Python

```
# check for missing values:
df.isnull().sum()
```

Output:

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

The code df.isnull().sum() checks for missing values in each column of the DataFrame 'df' and returns the sum of null values for each column

Checking for the duplicate values

Python

```
#checking duplicate values
df.nunique()
```

Output:

```
fixed acidity          96
volatile acidity       143
citric acid            80
residual sugar         91
chlorides              153
free sulfur dioxide    60
total sulfur dioxide   144
density                436
pH                     89
sulphates              96
alcohol                65
quality                6
dtype: int64
```

The function df.nunique() determines how many unique values there are in each column of the DataFrame "df," offering information about the variety of data that makes up each feature.

Exploratory Data Analysis

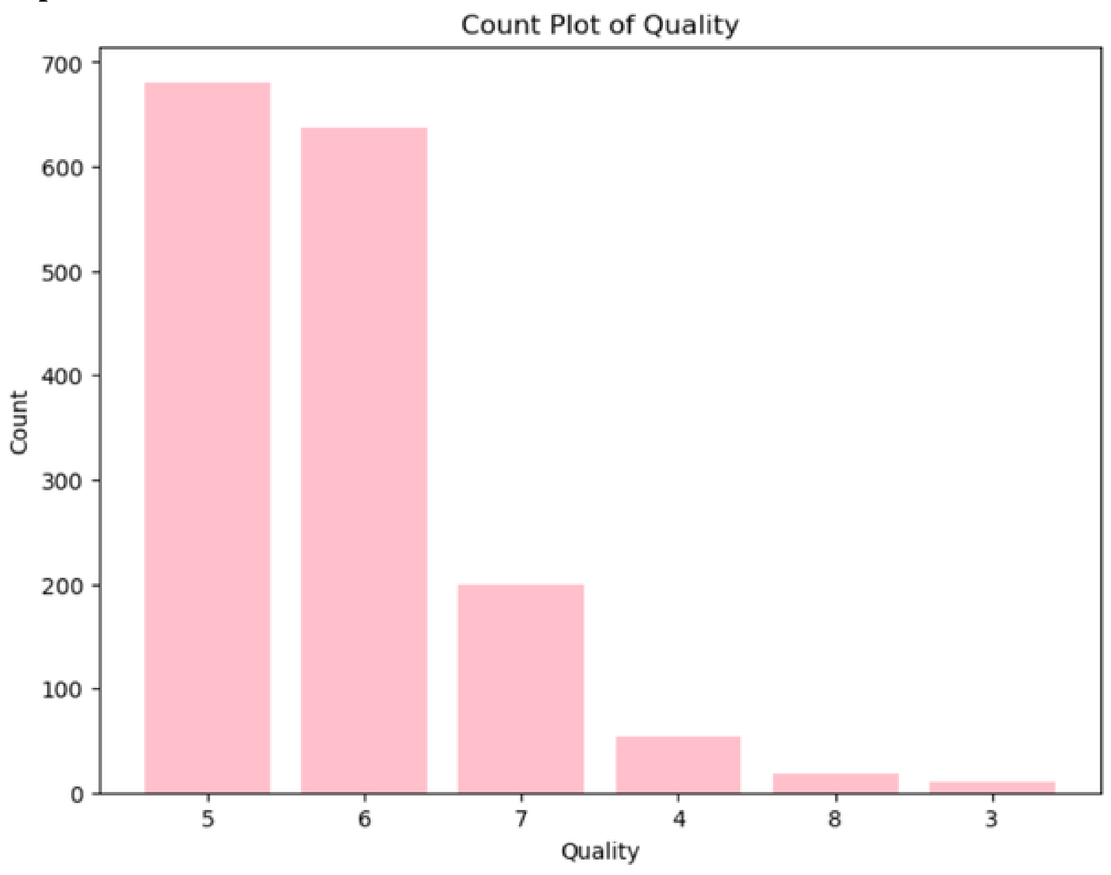
The principal goals of exploratory data analysis (EDA) are to detect anomalies in the dataset and develop recommendations for additional investigation, thereby guaranteeing a thorough comprehension of the subtleties of the data.

Step 4: Univariate Analysis

In Univariate analysis, plotting the right charts can help us better understand the data, which is why data visualization is so important. Matplotlib and Seaborn libraries are used in this post to visualize our data.

```
# Assuming 'df' is your DataFrame
quality_counts = df['quality'].value_counts()

# Using Matplotlib to create a count plot
plt.figure(figsize=(8, 6))
plt.bar(quality_counts.index, quality_counts, color='darpink')
plt.title('Count Plot of Quality')
plt.xlabel('Quality')
plt.ylabel('Count')
plt.show()
```

Output:

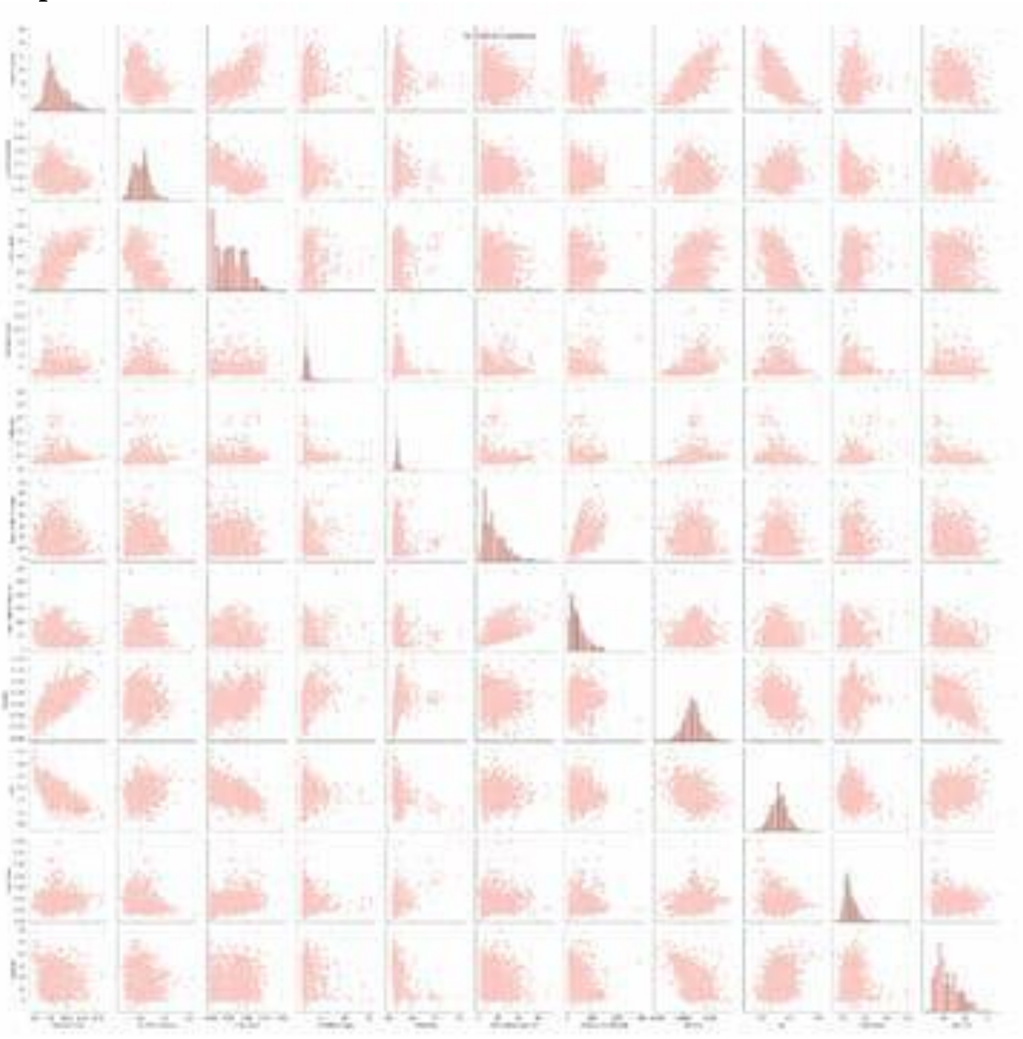


Step 5: Bivariate Analysis

When doing a bivariate analysis, two variables are examined simultaneously in order to look for patterns, dependencies, or interactions between them.

```
# Set the color palette
sns.set_palette("Pastel1")

# Assuming 'df' is your DataFrame
plt.figure(figsize=(10, 6))

# Using Seaborn to create a pair plot with the specified color palette
sns.pairplot(df)

plt.suptitle('Pair Plot for DataFrame')
plt.show()
```

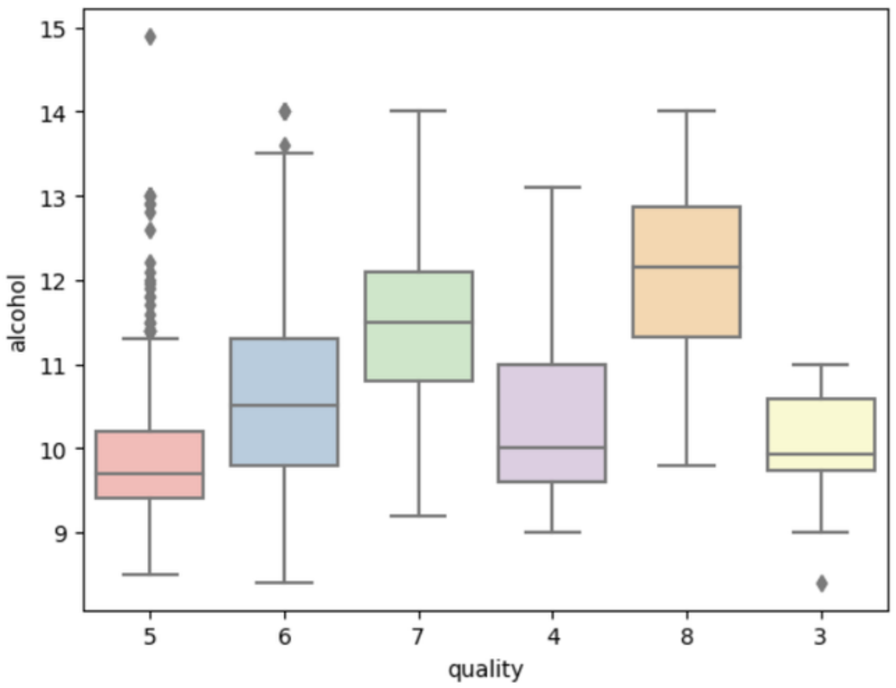
Output:



Box Plot

#plotting box plot between alcohol and quality

sns.boxplot(x='quality', y='alcohol', data=df)

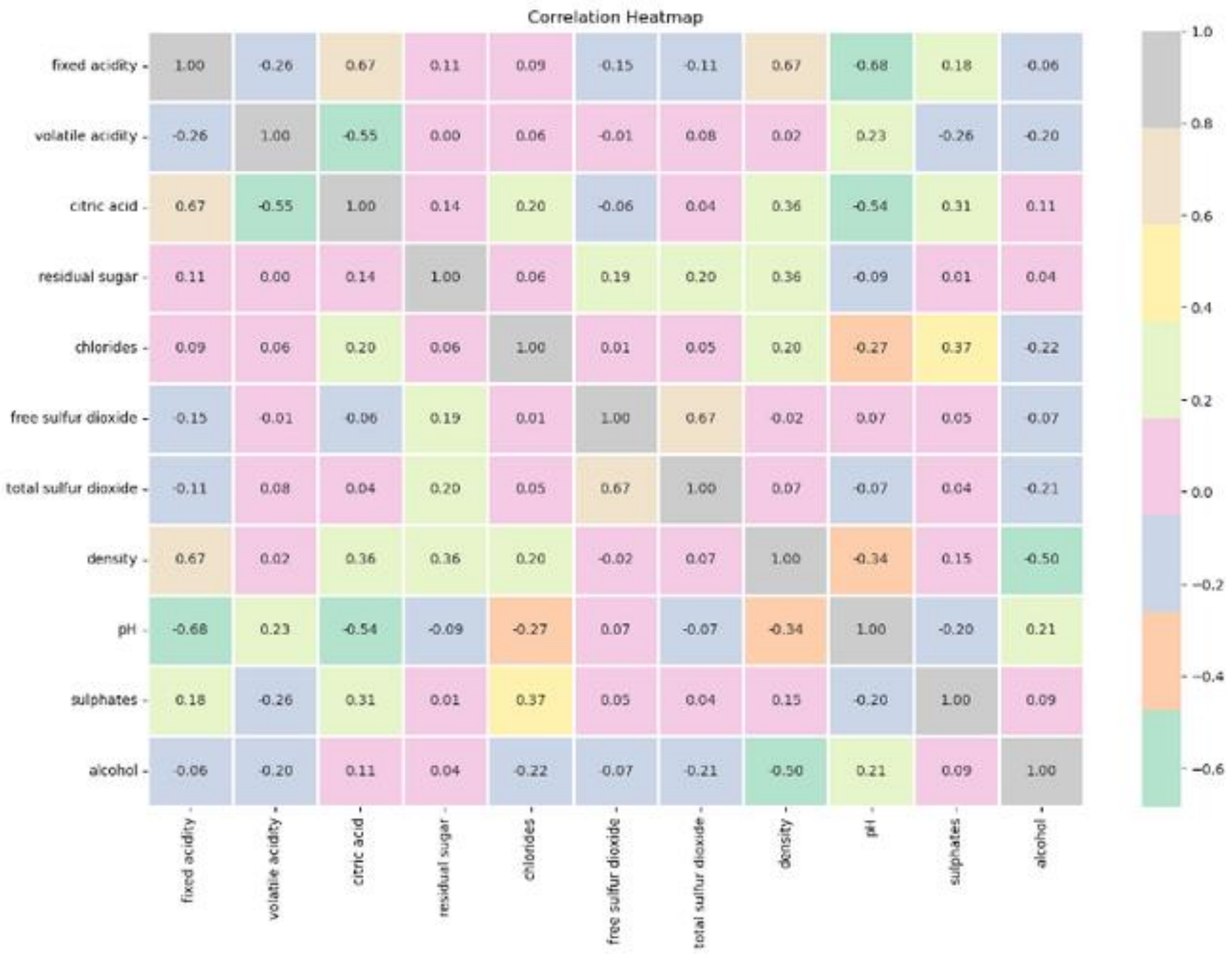


Step 6: Multivariate Analysis

Interactions between three or more variables in a dataset are simultaneously analyzed and interpreted in multivariate analysis.

```
# Assuming 'df' is your DataFrame
plt.figure(figsize=(15, 10))

# Using Seaborn to create a heatmap
sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='Pastel2', linewidths=2)

plt.title('Correlation Heatmap')
plt.show()
```

Output:



Experiment-10

Write a program to Implement Support Vector Machines

Aim: Write a program to Implement Support Vector Machines

Program:

```
# Step 1: Import necessary libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn import datasets

# Step 2: Load the Iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Labels

# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 4: Initialize and fit the SVM model
svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train, y_train)

# Step 5: Make predictions on the test data
y_pred = svm_model.predict(X_test)

# Step 6: Display results
# Calculate and print the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
# Print the classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Generate and print the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)
```

Output:

Experiment-11

Write a program to Implement Principle Component Analysis

Aim: Write a program to Implement Principle Component Analysis

Program:

PCA:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA

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

pca=PCA(n_components=2)
pca.fit(X)

print(pca.explained_variance_ratio_)
print(pca.singular_values_)
```

Output:

Randomized_PCA:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA

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

pca=PCA(n_components=2, svd_solver='randomized')
pca.fit(X)

print(pca.explained_variance_ratio_)
print(pca.singular_values_)
```

Output:

Kernal_PCA:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import KernalPCA

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

pca=KeranlPCA(n_components=2, kernal=”rbf”,gamma=0.04)
pca.fit(X)

X_reduced=rbf_pca.fit_transform(X)
print(X_reduced)
```

Output: