# MAHARAJA SURAJMAL INSTITUTE

## DEPARTMENT OF COMPUTER APPLICATION

## BCA I SHIFT

# Subject Code: BCAP 311

# Subject Name:

# Machine Learning with Python Lab

**Submitted by:**


**Name: Rahul Bera**

**Enrolment no: 09914902022**

**BCA 5A (Morning Shift)**


**Submitted to:**

**Dr. Neetu Anand**

**Associate Professor**

**Signature:**

# INDEX

## 1. Write a program to implement Simple linear regression on one variable using Placement .csv dataset.

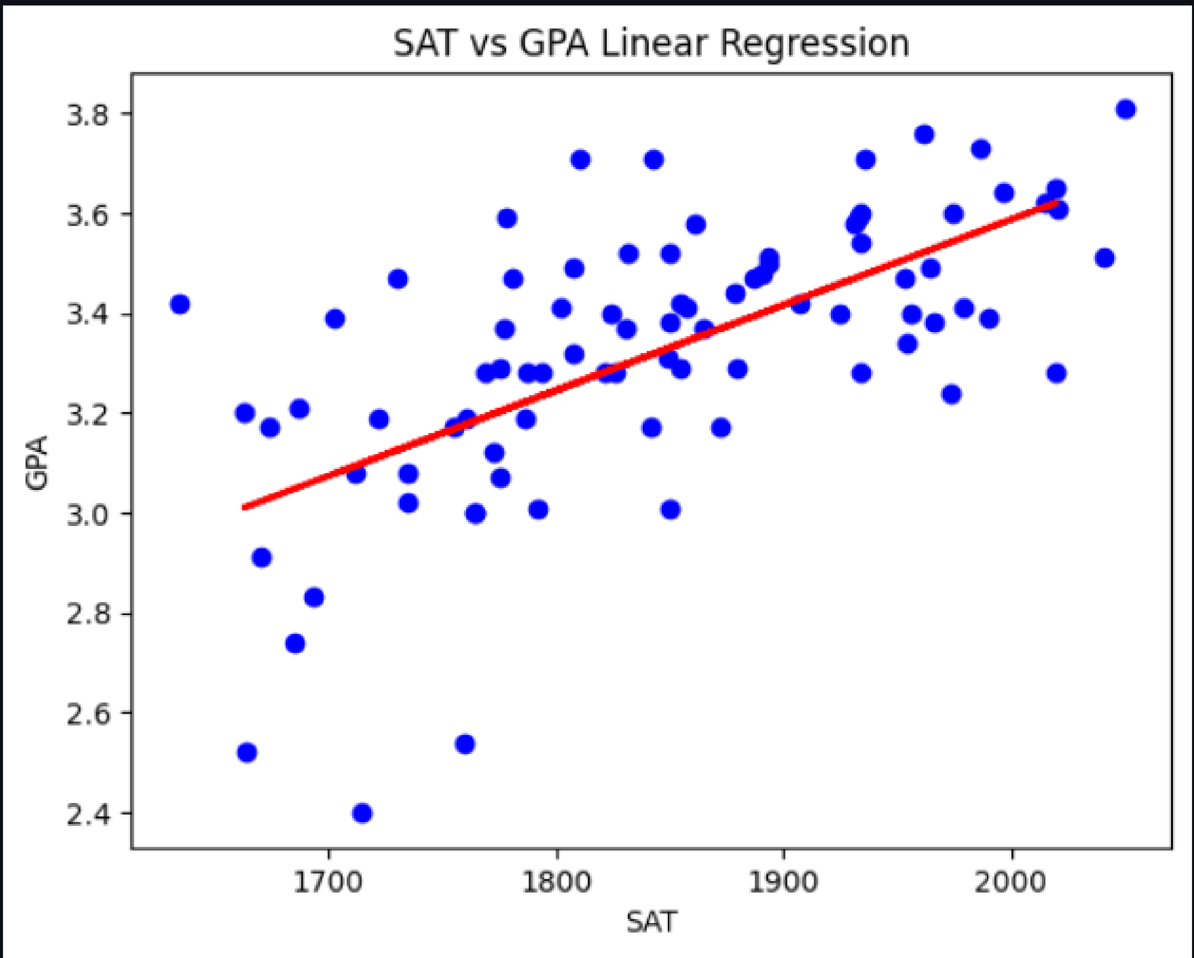
Code:

```
# Import necessary libraries
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
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("slr.csv")
print(df.head())

# Prepare the data
X = df[['SAT']]
y = df['GPA']

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create a Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

# Output
print('Model Coefficients:', model.coef_)
print('Model Intercept:', model.intercept_)
print('Mean Squared Error:', mse)
print('Predictions:', y_pred)

# Plot the linear regression results
plt.scatter(X, y, color='blue')  # scatter plot of the data points
plt.plot(X_test, y_pred, color='red', linewidth=2)  # linear regression line
plt.xlabel('SAT')
```

```
plt.ylabel('GPA')
plt.title('SAT vs GPA Linear Regression')
plt.show()
```

Output:

```
     SAT   GPA
0  1714  2.40
1  1664  2.52
2  1760  2.54
3  1685  2.74
4  1693  2.83
Model Coefficients: [0.00171033]
Model Intercept: 0.165864780191043
Mean Squared Error: 0.04691776855623536
Predictions: [3.19143271 3.35562407 3.07684082 3.24787349 3.01013808 3.37956865
 3.17603977 3.12472997 3.22221859 3.23077023 3.46850564 3.09394409
 3.32996917 3.62072471 3.16748814 3.2803697  3.61217308]
```



## 2. WAP to implement Multiple Linear Regression to predict prices of new homes based on area, bedrooms and age by using homeprices.csv.

Code:

```
# Import necessary libraries
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
import matplotlib.pyplot as plt
import numpy as np

# Load the dataset
df = pd.read_csv("housing.csv")

# Check out the first few rows of the DataFrame
print(df.head())

# Prepare the data (use relevant columns for prediction)
X = df[['Avg. Area House Age', 'Avg. Area Number of Rooms', 'Avg. Area Number of Bedrooms']]
y = df['Price']

# 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=0)

# Create a Multiple Linear Regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate the mean squared error
mse = mean_squared_error(y_test, y_pred)

# Output the results
print('Model Coefficients:', model.coef_)
print('Model Intercept:', model.intercept_)
print('Mean Squared Error:', mse)
print('Predictions:', y_pred)

# Plotting the predicted vs actual prices with a regression line
plt.figure(figsize=(10, 6))
```

```
plt.scatter(y_test, y_pred, color='blue', edgecolor='w', alpha=0.6)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs Predicted Prices")

# Create a line of best fit for predictions
m, b = np.polyfit(y_test, y_pred, 1)  # Linear fit
plt.plot(y_test, m * y_test + b, color='red', linewidth=2, label='Line of Best Fit')  # line indicating trend of predictions
plt.legend()

# Set limits for x and y axes
plt.xlim([min(y_test) - 100000, max(y_test) + 100000])
plt.ylim([min(y_pred) - 100000, max(y_pred) + 100000])

plt.grid()
plt.show()
```

Output:

```
   Avg. Area Income  Avg. Area House Age  Avg. Area Number of Rooms  \
0       79545.45857             5.682861                   7.009188
1       79248.64245             6.002900                   6.730821
2       61287.06718             5.865890                   8.512727
3       63345.24005             7.188236                   5.586729
4       59982.19723             5.040555                   7.839388

   Avg. Area Number of Bedrooms  Area Population         Price  \
0                          4.09      23086.80050  1.059034e+06
1                          3.09      40173.07217  1.505891e+06
2                          5.13      36882.15940  1.058988e+06
3                          3.26      34310.24283  1.260617e+06
4                          4.23      26354.10947  6.309435e+05

                                             Address
0  208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1  188 Johnson Views Suite 079\nLake Kathleen, CA...
2  9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3                          USS Barnett\nFPO AP 44820
4                         USNS Raymond\nFPO AE 09386
Model Coefficients: [162618.3070195   113829.3279345    5248.86918563]
Model Intercept: -555560.9258429043
Mean Squared Error: 82410045818.38522
Predictions: [1038151.05121551 1009594.19388622 1019270.47602753 1034834.89014662
 1108022.19391607  845067.13513864 1321543.48265709 1243328.84769941
...
 1227317.60029984 1636523.23899262 1243987.44068642 1248022.24574312
 1008000.5014977  1396045.55628315 1251634.08730898 1012278.59433029
 1374043.1622093  1365202.46835124 1324732.09073639 1012618.75050921
 1284107.62028866 1185154.64552303 1246285.520876   1377509.9732504 ]
```

*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*


Actual Prices vs Predicted Prices
1e6
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
Predicted Prices
Line of Best Fit
0.5
1.0
1.5
2.0
Actual Prices
1e6

## 3. Write a program to implement Binary logistic regression to predict if a person will buy life insurance based on his age using file insurance_data.csv.

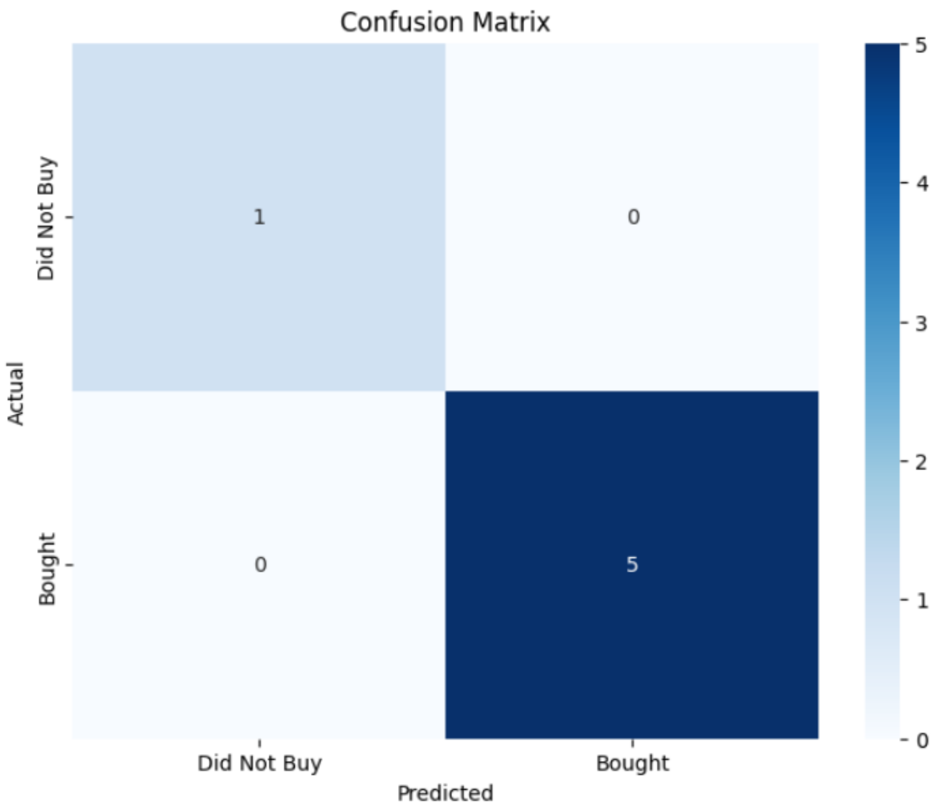
Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("insurance_data.csv")

# Check out the first few rows of the DataFrame
print(df.head())

# Prepare the data
X = df[['age']]  # Features
y = df['bought_insurance']  # Target variable

# 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=0)

# Create a Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Output the results
print('Accuracy:', accuracy)
print('Confusion Matrix:\n', conf_matrix)
print('Classification Report:\n', class_report)

# Visualizing the confusion matrix
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Did Not Buy', 'Bought'],
            yticklabels=['Did Not Buy', 'Bought'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Output:

```
   age  bought_insurance
0   22                 0
1   25                 0
2   47                 1
3   52                 0
4   46                 1
Accuracy: 1.0
Confusion Matrix:
 [[1 0]
 [0 5]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         5

    accuracy                           1.00         6
   macro avg       1.00      1.00      1.00         6
weighted avg       1.00      1.00      1.00         6
```



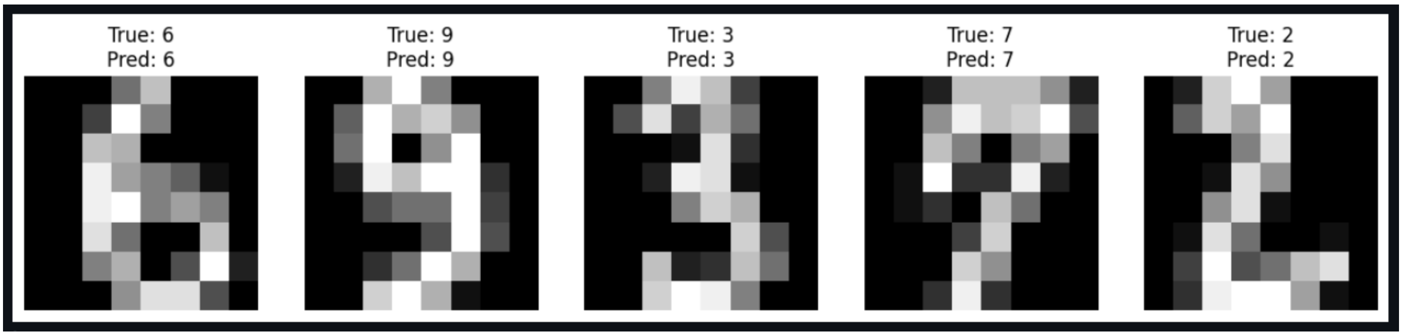
## 4. WAP to implement Multiclass Logistic Regression on digits dataset

Code:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Load the digits dataset
digits = datasets.load_digits()

# Features and target variable
X = digits.data  # The pixel values
y = digits.target  # The corresponding digit labels

# 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)

# Create a Multiclass Logistic Regression model
model = LogisticRegression(max_iter=10000, solver='lbfgs', multi_class='multinomial')

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualizing some predictions
n_images = 5
plt.figure(figsize=(15, 6))
for i in range(n_images):
   plt.subplot(1, n_images, i + 1)
   plt.imshow(X_test[i].reshape(8, 8), cmap='gray')
   plt.title(f'True: {y_test[i]}\nPred: {y_pred[i]}')
   plt.axis('off')
plt.show()
```

Output:

```
Confusion Matrix:
 [[53  0  0  0  0  0  0  0  0  0]
 [ 0 47  1  0  0  0  0  0  2  0]
 [ 0  0 47  0  0  0  0  0  0  0]
 [ 0  0  1 52  0  1  0  0  0  0]
 [ 0  1  0  0 58  0  1  0  0  0]
 [ 0  1  0  0  0 63  1  0  0  1]
 [ 0  0  0  0  0  1 52  0  0  0]
 [ 0  0  0  0  0  1  0 53  0  1]
 [ 0  0  0  0  0  1  0  0 42  0]
 [ 0  0  0  1  0  0  0  0  2 56]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        53
           1       0.96      0.94      0.95        50
           2       0.96      1.00      0.98        47
           3       0.98      0.96      0.97        54
           4       1.00      0.97      0.98        60
           5       0.94      0.95      0.95        66
           6       0.96      0.98      0.97        53
           7       1.00      0.96      0.98        55
           8       0.91      0.98      0.94        43
           9       0.97      0.95      0.96        59
...
    accuracy                           0.97       540
   macro avg       0.97      0.97      0.97       540
weighted avg       0.97      0.97      0.97       540
```



## 5. WAP to implement Decision Tree Classifier on drug.csv dataset

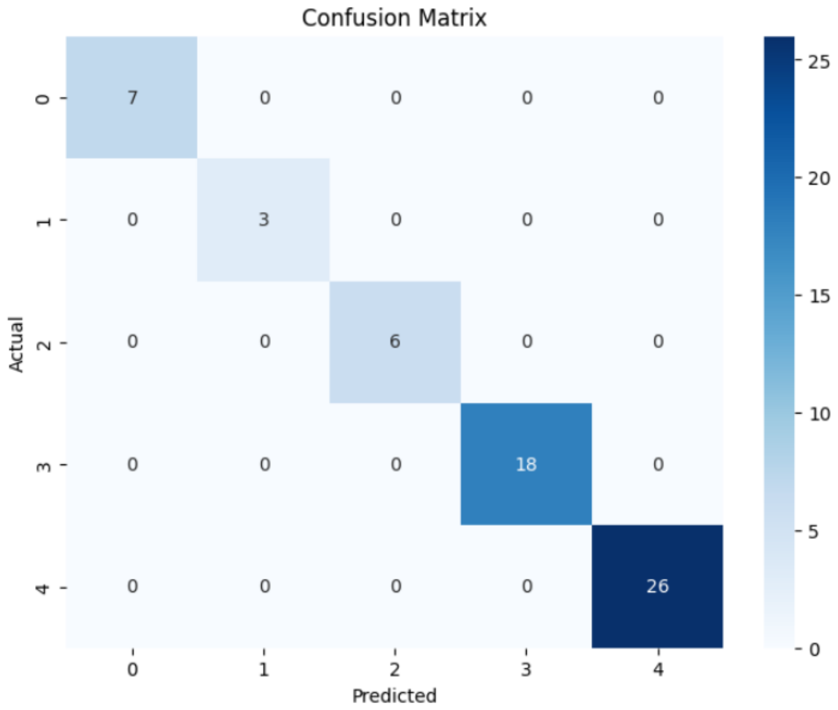
Code:

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv('drug.csv')

# Separate features (x) and target (y)
x = df.drop('Drug', axis=1)
y = df[['Drug']]

# Encode categorical features using LabelEncoder
le = LabelEncoder()
x['Sex'] = le.fit_transform(x['Sex'])
x['BP'] = le.fit_transform(x['BP'])
x['Cholesterol'] = le.fit_transform(x['Cholesterol'])
y['Drug'] = le.fit_transform(y['Drug'])

# Split data 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)

# Create and train the Decision Tree Classifier
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(x_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

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

class_report = classification_report(y_test, y_pred)
print("Classification Report:\n", class_report)
```

```
# Visualize the confusion matrix
confusion = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Visualize the decision tree
plt.figure(figsize=(10, 10))
tree.plot_tree(clf.fit(x_train, y_train), filled=True)
plt.show()
```

Output:

```
Accuracy: 1.0
Confusion Matrix:
 [[ 7  0  0  0  0]
 [ 0  3  0  0  0]
 [ 0  0  6  0  0]
 [ 0  0  0 18  0]
 [ 0  0  0  0 26]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00         3
           2       1.00      1.00      1.00         6
           3       1.00      1.00      1.00        18
           4       1.00      1.00      1.00        26

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```


Confusion Matrix
Actual
Predicted
0
1
2
3
4
7
3
6
18
26
25
20
15
10
5
0


x[4] <= 14.829
gini = 0.692
samples = 140
value = [16, 13, 10, 36, 65]
True
False
x[2] <= 0.5
gini = 0.676
samples = 75
value = [16, 13, 10, 36, 0]
gini = 0.0
samples = 65
value = [0, 0, 0, 0, 65]
x[0] <= 50.5
gini = 0.495
samples = 29
value = [16, 13, 0, 0, 0]
x[2] <= 1.5
gini = 0.34
samples = 46
value = [0, 0, 10, 36, 0]
gini = 0.0
samples = 16
value = [16, 0, 0, 0, 0]
gini = 0.0
samples = 13
value = [0, 13, 0, 0, 0]
x[3] <= 0.5
gini = 0.499
samples = 21
value = [0, 0, 10, 11, 0]
gini = 0.0
samples = 25
value = [0, 0, 0, 25, 0]
gini = 0.0
samples = 10
value = [0, 0, 10, 0, 0]
gini = 0.0
samples = 11
value = [0, 0, 0, 11, 0]

## 6. Write a program to implement the naive Bayesian classifier on titanic .CSV file.

Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the Titanic data
titanic = pd.read_csv("titanic.csv")

print(titanic.head())

# Drop irrelevant columns for prediction
titanic.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

# Define the target variable
target = titanic['Survived']
input_data = titanic.drop(['Survived'], axis=1)

# Convert 'Sex' column to numerical using get_dummies
sex_dummies = pd.get_dummies(input_data['Sex'], prefix='Sex', drop_first=True)

# Convert 'Embarked' column to numerical using get_dummies
embarked_dummies = pd.get_dummies(input_data['Embarked'], prefix='Embarked',
drop_first=True)

# Append the new columns to input_data
input_data = pd.concat([input_data, sex_dummies, embarked_dummies], axis=1)

# Drop the original 'Sex' and 'Embarked' columns
input_data.drop(['Sex', 'Embarked'], axis=1, inplace=True)

# Check for NaN values and fill them in the 'Age' column with the mean
input_data['Age'].fillna(input_data['Age'].mean(), inplace=True)

# Train-test split with an 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(input_data, target, test_size=0.2,
random_state=42)

# Check the lengths of the datasets
print(f"X_train length: {len(X_train)}")
print(f"X_test length: {len(X_test)}")
```

```
print(f"Input data length: {len(input_data)}")

# Train the model using Gaussian Naive Bayes
model = GaussianNB()
model.fit(X_train, y_train)

# Measure the accuracy score
accuracy = model.score(X_test, y_test)
print(f"Model accuracy: {accuracy:.2f}")

# Make predictions on X_test samples and compare with y_test
y_pred = model.predict(X_test[:10])
comparison = pd.DataFrame({'Predicted': y_pred, 'Actual': y_test[:10].values})
print("Comparison of predicted and actual values for 10 samples:\n", comparison)


# Run predict_proba on X_test to check survival probability
probabilities = model.predict_proba(X_test[:10])
print("Predicted probabilities for 10 samples:\n", probabilities)
```

Output:

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
X_train length: 712
X_test length: 179
Input data length: 891
Model accuracy: 0.77
Comparison of predicted and actual values for 10 samples:
...
 [0.21978697 0.78021303]
 [0.9873942  0.0126058 ]
 [0.17701381 0.82298619]
 [0.05689315 0.94310685]]
```

## 7. Write a program to implement the naive Bayesian classifier on email dataset spam.csv file.

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

# Load the spam data with specified encoding
df = pd.read_csv("spam.csv", encoding="ISO-8859-1")

# Keep only the first two columns
df = df.iloc[:, :2]
print(df.head())

# Describe data by category
print(df.groupby('Category').describe())

# Add a binary column 'spam'
df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)
print(df.head())

# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df['Message'], df['spam'], test_size=0.2)

# Vectorize the text data
v = CountVectorizer()
X_train_count = v.fit_transform(X_train)
print(X_train_count.toarray()[:2])

# Build and train the model
model = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('classifier', MultinomialNB())
])

model.fit(X_train, y_train)

# Make predictions and evaluate the model
y_pred = model.predict(X_test)
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
```

Output:

```
Execution Order                                                    Message
0      ham  Go until jurong point, crazy.. Available only ...
1      ham                       Ok lar... Joking wif u oni...
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...
3      ham  U dun say so early hor... U c already then say...
4      ham  Nah I don't think he goes to usf, he lives aro...
         Message                                                              \
           count unique                                                  top
Category
ham         4825   4516                             Sorry, I'll call later
spam         747    653  Please call our customer service representativ...


         freq
Category
ham        30
spam        4
  Category                                            Message  spam
0      ham  Go until jurong point, crazy.. Available only ...     0
1      ham                       Ok lar... Joking wif u oni...     0
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...     1
3      ham  U dun say so early hor... U c already then say...     0
4      ham  Nah I don't think he goes to usf, he lives aro...     0
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Accuracy: 0.9856502242152466
```

## 8. Write a program to implement k-nearest neighbors (KNN) on iris.csv dataset.

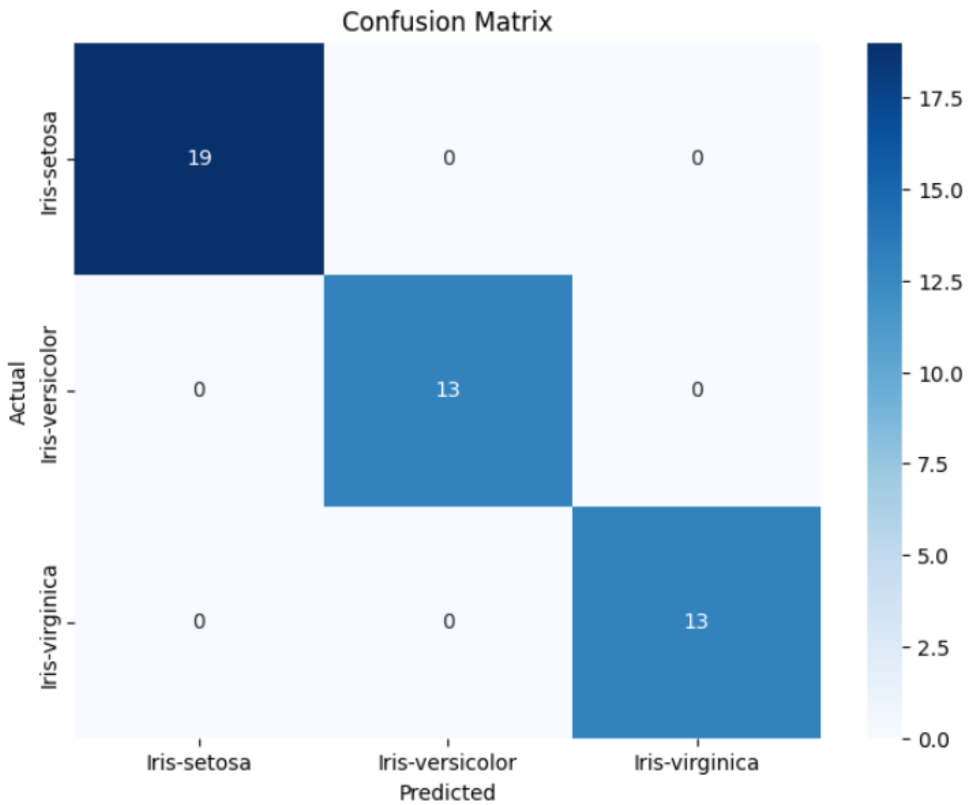
Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("iris.csv")

# Display the first few rows of the DataFrame
print(df.head())

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Prepare features and target variable
X = df.iloc[:, :-1]  # Features (all columns except the last)
y = df['iris_name']  # Target variable (last column)

# 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)

# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust n_neighbors as needed

# Train the model
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate the model
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Visualize the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues',
        xticklabels=knn.classes_, yticklabels=knn.classes_)
```

```
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
```

Output:

```
   sepal_length(cm)  sepal_width(cm)  petal_length(cm)  petal_width(cm)  \
0               5.1              3.5               1.4              0.2
1               4.9              3.0               1.4              0.2
2               4.7              3.2               1.3              0.2
3               4.6              3.1               1.5              0.2
4               5.0              3.6               1.4              0.2

     iris_name
0  Iris-setosa
1  Iris-setosa
2  Iris-setosa
3  Iris-setosa
4  Iris-setosa

Missing values in each column:
sepal_length(cm)    0
sepal_width(cm)     0
petal_length(cm)    0
petal_width(cm)     0
iris_name           0
dtype: int64
Confusion Matrix:
 [[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
...
       accuracy                           1.00        45
      macro avg       1.00      1.00      1.00        45
   weighted avg       1.00      1.00      1.00        45
```



## 9. Write a program to implement Support Vector Machine (SVM) Algorithm on iris.csv file.

Code:

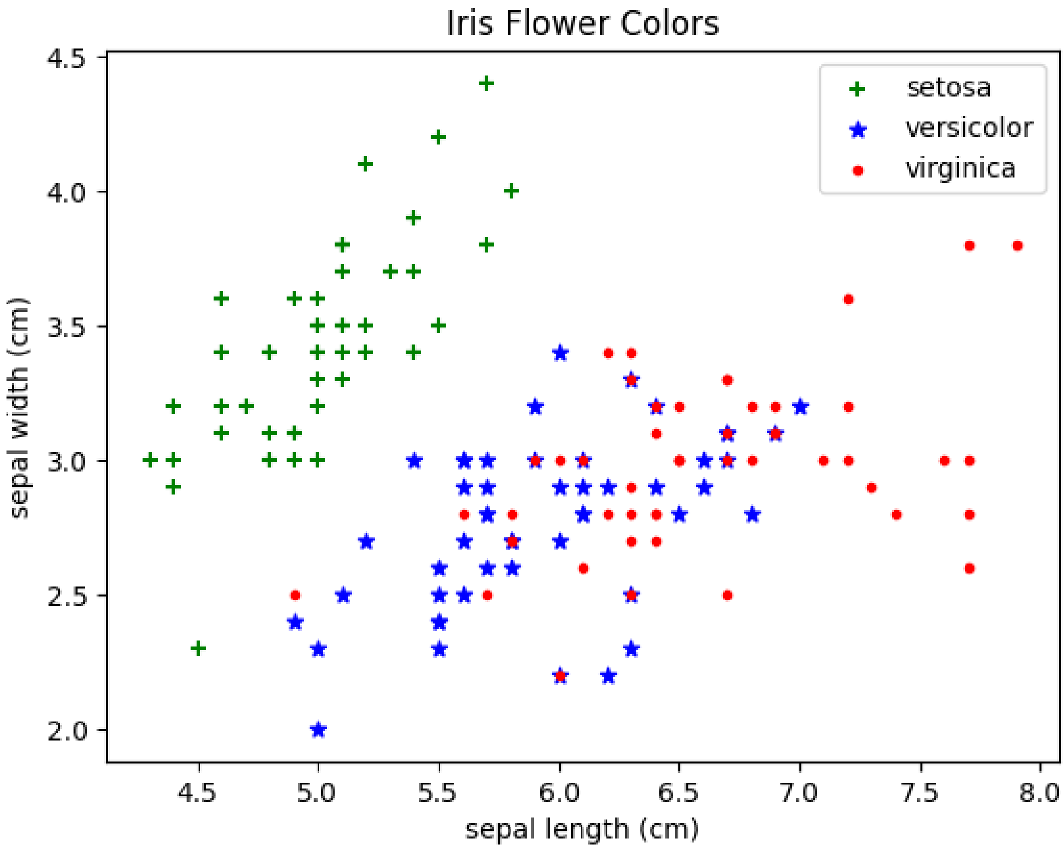
```
        # Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load dataset and add targets
iris=load_iris()
dir(iris)
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df['target']=iris.target
df['flower_name']=iris.target_names[df['target']]
df.head()

# Plot scatterplot
setosa=df[df.target==0]
versicolor=df[df.target==1]
virginica=df[df.target==2]

plt.title('Iris Flower Colors')
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
plt.scatter(setosa['sepal length (cm)'],setosa['sepal width (cm)'],color='green',marker='+')
plt.scatter(versicolor['sepal length (cm)'],versicolor['sepal width (cm)'],color='blue',marker='*')
plt.scatter(virginica['sepal length (cm)'],virginica['sepal width (cm)'],color='red',marker='.')
plt.legend(iris.target_names)
plt.show()

# Split data into training and test sets
x=df.drop(['target','flower_name'],axis='columns')
y=df.target
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)

# Build and train the model
model=SVC()
model.fit(x_train,y_train)
model.score(x_test,y_test)

# Predict values using model
```

```
model.predict([[5.1,3.5,1.4,0.2]])

#Tuning the model with different parameters
model_C=SVC(C=1)
model_C.fit(x_train,y_train)
model_C.score(x_test,y_test)

model_C=SVC(C=10)
model_C.fit(x_train,y_train)
model_C.score(x_test,y_test)

model_g=SVC(gamma=10)
model_g.fit(x_train,y_train)
model_g.score(x_test,y_test)

model_linear_kernel=SVC(kernel='linear')
model_linear_kernel.fit(x_train,y_train)
model_linear_kernel.score(x_test,y_test)
```

Output:

```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target | flower_name |
|---|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 | setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 | setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 | setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 | setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 | setosa |




0.9

array([0])

0.9

0.9666666666666667

0.9666666666666667

1.0

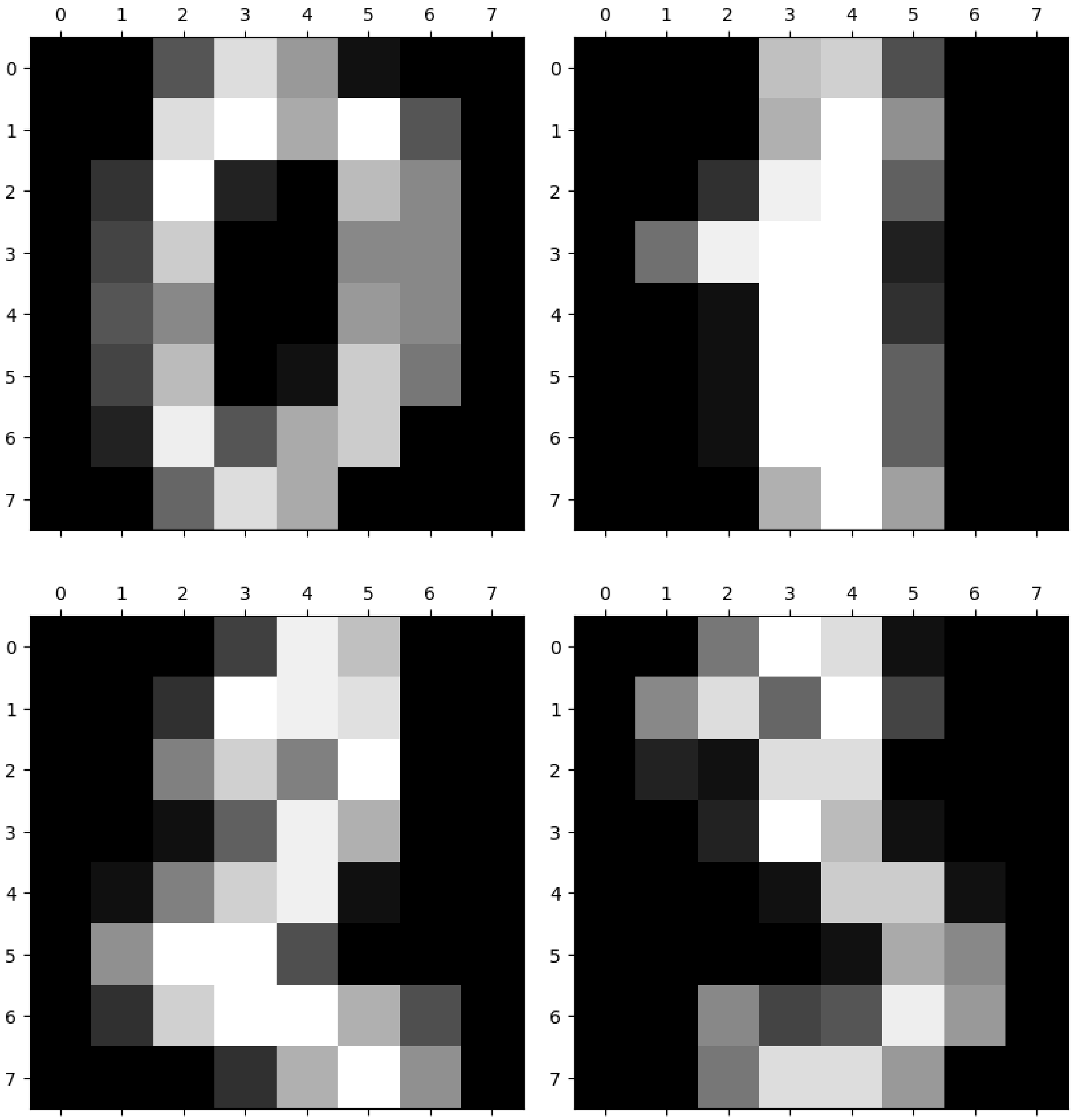
## 10. Implement classification on a digits.csv dataset using random forest classifier.

Code:

```
        # Import Libraries
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
digits = load_digits()
dir(digits)

# Create the dataframe
df = pd.DataFrame(digits.data)
df['target'] = digits.target
df.head()

plt.gray()
for i in range(4):
   plt.matshow(digits.images[i])

# Split the data into training and testing sets
X=df.drop('target',axis='columns')
y=df.target
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)

# Build and train the model
model = RandomForestClassifier(n_estimators=20)
model.fit(X_train, y_train)

# check model score
model.score(X_test,y_test)

# Confusion Matrix
y_predicted = model.predict(X_test)
cm = confusion_matrix(y_test,y_predicted)
cm

# Plot the heatmap
plt.figure(figsize=(10,7))
sns.heatmap(cm, annot=True)
```

```
        plt.xlabel('Predicted')
        plt.ylabel('Truth')
```

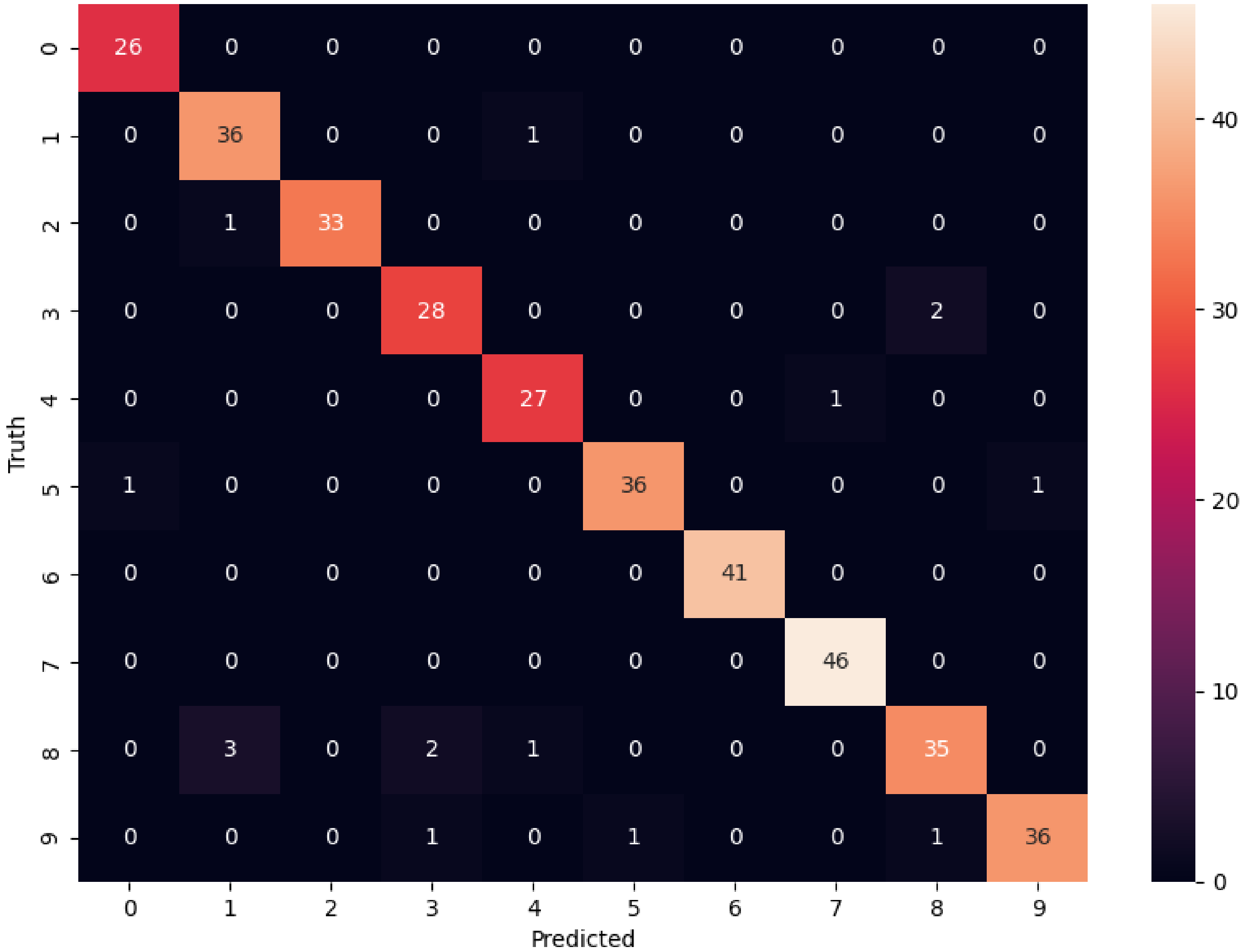
Output:




```
0.9555555555555556


array([[26,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 36,  0,  0,  1,  0,  0,  0,  0,  0],
       [ 0,  1, 33,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 28,  0,  0,  0,  0,  2,  0],
```

```
 [ 0,  0,  0,  0, 27,  0,  0,  1,  0,  0],
 [ 1,  0,  0,  0,  0, 36,  0,  0,  0,  1],
 [ 0,  0,  0,  0,  0,  0, 41,  0,  0,  0],
 [ 0,  0,  0,  0,  0,  0,  0, 46,  0,  0],
 [ 0,  3,  0,  2,  1,  0,  0,  0, 35,  0],
 [ 0,  0,  0,  1,  0,  1,  0,  0,  1, 36]])
```

Text(95.72222222222221, 0.5, 'Truth')



## 11. Build an Artificial Neural Network (ANN) on digits.csv file.

Code:

```
# Import Libraries
import pandas as pd
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

# Read the CSV file into a DataFrame
df = pd.read_csv("drug200.csv")

# Display the first 5 rows
df.head()

# Print the column names and their data types
print(df.info())

# Create a OneHotEncoder object
enc = OneHotEncoder(handle_unknown='ignore')

# Fit the encoder on the categorical columns
enc.fit(df[['Sex', 'BP', 'Cholesterol']])

# Transform the categorical features into numerical representations
num_cat_features = enc.transform(df[['Sex', 'BP', 'Cholesterol']]).toarray()

# Create a new DataFrame with the encoded categorical features
df_catnum = pd.DataFrame(num_cat_features)

# Concatenate the numerical and encoded categorical features
df_num = pd.concat([df[['Age','Na_to_K']], df_catnum], axis=1)

# Add the target variable to the new DataFrame
df_num['Drug'] = df['Drug']

# Split into features (X) and target (y)
X = df_num.drop('Drug', axis=1)
y = df_num['Drug']

# Split 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)

# Convert column names to strings
```

```
X_train.columns = X_train.columns.astype(str)
X_test.columns = X_test.columns.astype(str)

# Initialize the  MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam',
max_iter=200, random_state=42)

# Train the model
mlp.fit(X_train, y_train)

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

# Evaluate the model's performance
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(classification_report(y_test, y_pred))
```

Output:

| | Age | Sex | BP | Cholesterol | Na_to_K | Drug |
|---|---|---|---|---|---|---|
| **0** | 23 | F | HIGH | HIGH | 25.355 | DrugY |
| **1** | 47 | M | LOW | HIGH | 13.093 | drugC |
| **2** | 47 | M | LOW | HIGH | 10.114 | drugC |
| **3** | 28 | F | NORMAL | HIGH | 7.798 | drugX |
| **4** | 61 | F | LOW | HIGH | 18.043 | DrugY |

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

None

Accuracy: 0.85

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| DrugY | 0.88 | 1.00 | 0.94 | 15 |
| drugA | 1.00 | 0.67 | 0.80 | 6 |
| drugB | 0.75 | 1.00 | 0.86 | 3 |
| drugC | 1.00 | 0.20 | 0.33 | 5 |
| drugX | 0.79 | 1.00 | 0.88 | 11 |
| accuracy | | | 0.85 | 40 |
| macro avg | 0.88 | 0.77 | 0.76 | 40 |
| weighted avg | 0.88 | 0.85 | 0.82 | 40 |

## 12. Apply k-Means algorithm to cluster a set of data stored in an income.CSV file.

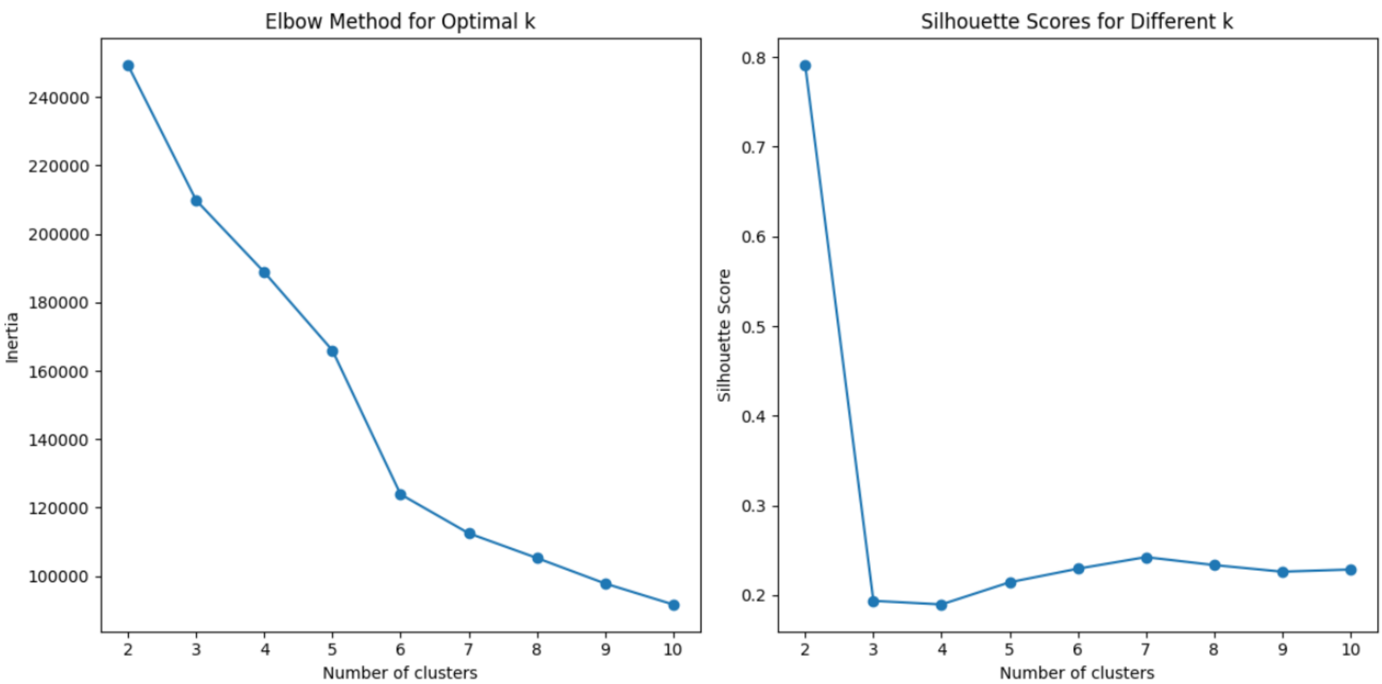
Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Load the dataset
df = pd.read_csv("income.csv")

# Display the first few rows of the DataFrame
print("Dataset Head:\n", df.head())

# Check for missing values
print("\nMissing values in each column:")
print(df.isnull().sum())

# Prepare the features for clustering
features = df.iloc[:, :-1]  # Select all columns except 'income_level'
print("\nFeatures for clustering:\n", features.head())

# Scale the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Determine the optimal number of clusters using the elbow method
inertia = []
silhouette_scores = []
k_range = range(2, 11)  # Trying different numbers of clusters from 2 to 10

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(features_scaled)
    inertia.append(kmeans.inertia_)
    score = silhouette_score(features_scaled, kmeans.labels_)
    silhouette_scores.append(score)

# Plot the elbow curve
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(k_range, inertia, marker='o')
```

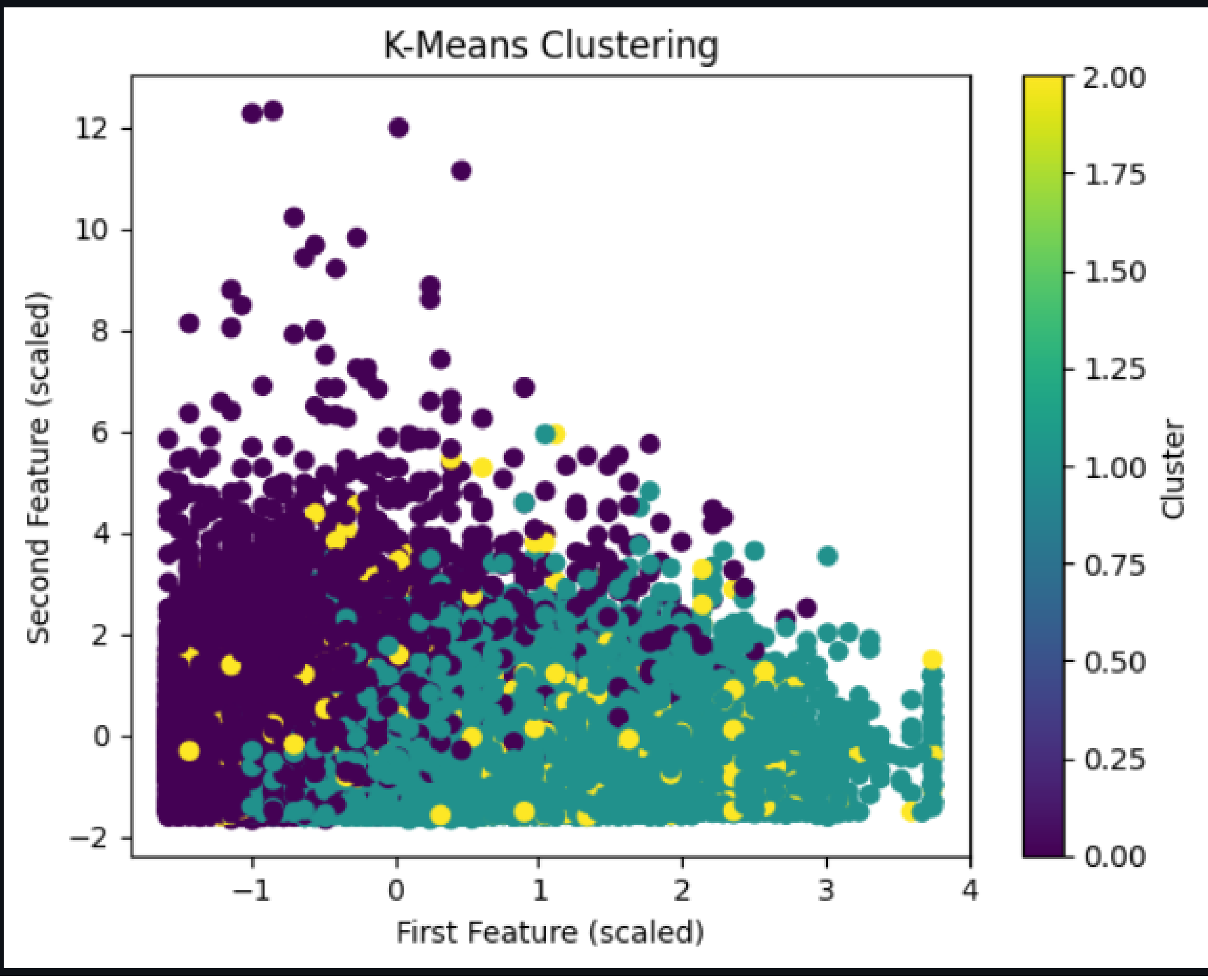
```
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')

# Plot the silhouette scores
plt.subplot(1, 2, 2)
plt.plot(k_range, silhouette_scores, marker='o')
plt.title('Silhouette Scores for Different k')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')

plt.tight_layout()
plt.show()

# Fit the K-Means model with the optimal number of clusters (you can choose based on the
plots)
optimal_k = 3  # Change this based on the elbow plot or silhouette score
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans.fit(features_scaled)

# Add the cluster labels to the original dataframe
df['Cluster'] = kmeans.labels_

# Display the clustered data
print("\nClustered Data:\n", df.head())

# Visualizing the clusters (if you have only two features, you can use a scatter plot)
plt.scatter(features_scaled[:, 0], features_scaled[:, 1], c=df['Cluster'], cmap='viridis',
marker='o')
plt.title('K-Means Clustering')
plt.xlabel('First Feature (scaled)')
plt.ylabel('Second Feature (scaled)')
plt.colorbar(label='Cluster')
plt.show()
```

Output:

```
Dataset Head:
   age  fnlwgt  education_num  capital_gain  capital_loss  hours_per_week  \
0   39   77516             13          2174             0              40
1   50   83311             13             0             0              13
2   38  215646              9             0             0              40
3   53  234721              7             0             0              40
4   28  338409             13             0             0              40

   income_level
0             0
1             0
2             0
3             0
4             0

Missing values in each column:
age               0
fnlwgt            0
education_num     0
capital_gain      0
capital_loss      0
hours_per_week    0
income_level      0
dtype: int64
...
1   50   83311             13             0             0              13
2   38  215646              9             0             0              40
3   53  234721              7             0             0              40
4   28  338409             13             0             0              40
```



```
Clustered Data:
   age  fnlwgt  education_num  capital_gain  capital_loss  hours_per_week  \
0   39   77516             13          2174             0              40
1   50   83311             13             0             0              13
2   38  215646              9             0             0              40
3   53  234721              7             0             0              40
4   28  338409             13             0             0              40

   income_level  Cluster
0             0        1
1             0        1
2             0        0
3             0        1
4             0        0
```



## 13. Write a program to implement Self -Organizing Map (SOM) on Credit_card_applications.csv file

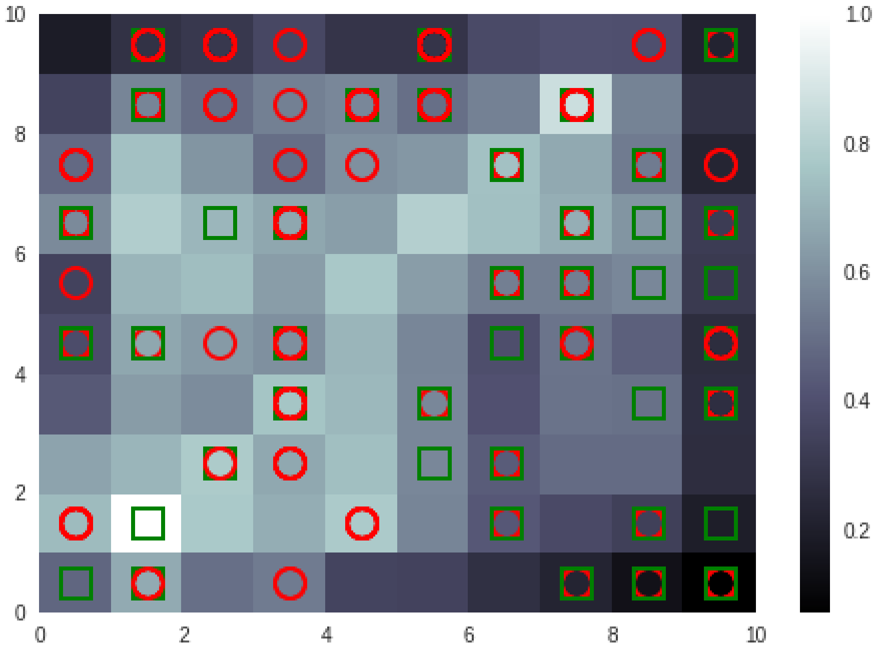
Code:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
data = pd.read_csv('drive/My Drive/super/Self Organizing Maps/Credit_Card_Applications.csv')
```

```
# splitting the data into dependent and independent variables

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

print(X.shape)
print(y.shape)
from sklearn.preprocessing import MinMaxScaler

# creating a scaler function
mm = MinMaxScaler(feature_range = (0, 1))

# feeding the independent variable into the scaler function
X = mm.fit_transform(X)
# training the SOM

from minisom import MiniSom

som = MiniSom(x = 10, y = 10, input_len = 15, sigma = 1.0, learning_rate = 0.5)
som.random_weights_init(X)
som.train_random(data = X, num_iteration = 100)
# visualizing the results

from pylab import bone, pcolor, colorbar, plot, show

bone()
pcolor(som.distance_map().T)
colorbar()
markers = ['o', 's']
colors = ['r', 'g']

for i, x in enumerate(X):
w = som.winner(x)
plot(w[0]+0.5, w[1]+0.5,
markers[y[i]],
markeredgecolor = colors[y[i]],
markerfacecolor = 'None',
markersize = 15,
```

```
markeredgewidth  = 2)

show()
```

Output:




```
# finding the frauds

mappings  = som.win_map(X)

frauds = np.concatenate((mappings[(2,  4)], mappings[(7,  4)]), axis = 0)
frauds = mm.inverse_transform(frauds)
# creating the matrix of features
customers  = data.iloc[:,1:].values

# creating the dependent  variable
is_fraud = np.zeros(len(data))

for i in range(len(data)):
if data.iloc[i,0]  in frauds :
is_fraud[i]  = 1
# feature  scaling

from sklearn.preprocessing  import StandardScaler

sc = StandardScaler()

customers  = sc.fit_transform(customers)
import keras
from keras.layers  import Dense
from keras.models  import Sequential
model  = Sequential()
model.add(Dense(units  = 8, init = 'uniform',  activation  = 'relu', input_dim  = 15))
model.add(Dense(units  = 1, init = 'uniform',  activation  = 'sigmoid',))
```

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

model.fit(customers, is_fraud, batch_size = 10, epochs = 2)
# predicting the test set results

y_pred = model.predict(customers)
y_pred = np.concatenate((data.iloc[:, 0:1].values, y_pred), axis = 1)
y_pred = y_pred[y_pred[:,1].argsort()]
```

## 14. Write a program to implement PCA on digits dataset.

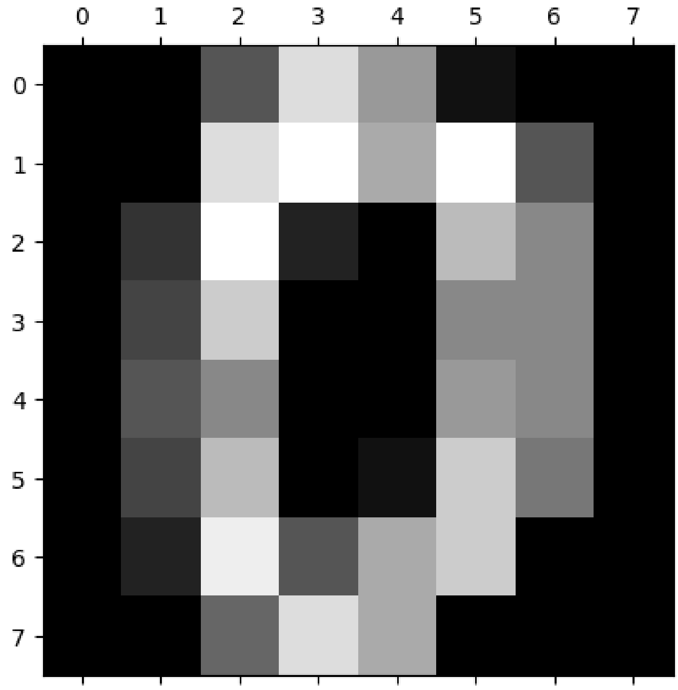
Code:

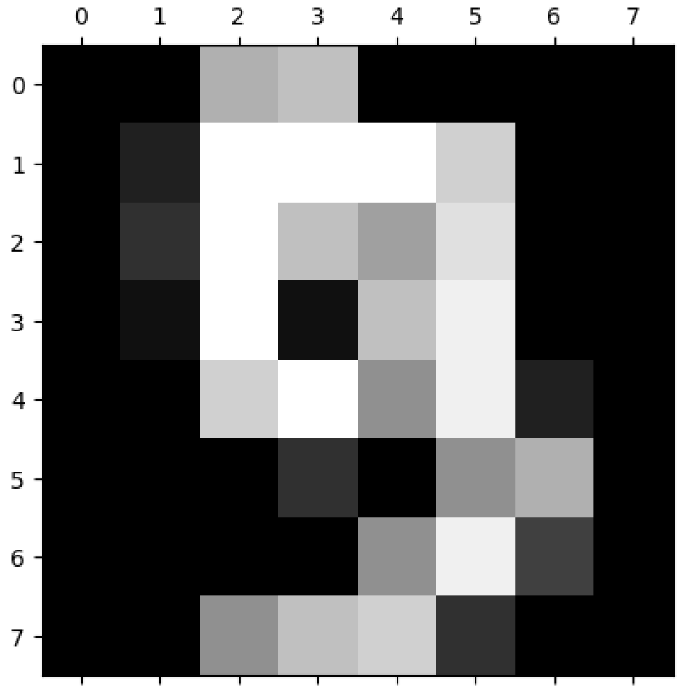
```
# Import Libraries
import pandas as pd
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

digits = load_digits()
digits.data.shape

digits.data[0].reshape(8, 8)

plt.gray()
plt.matshow(digits.data[0].reshape(8, 8))
plt.matshow(digits.data[9].reshape(8, 8))

df=pd.DataFrame(digits.data, columns=digits.feature_names)
df.head()

df.describe()

X = df
y = digits.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=30)

model = LogisticRegression()
model.fit(X_train, y_train)
accuracy_score(y_test, model.predict(X_test))

pca = PCA(0.95)
X_pca = pca.fit_transform(X)
X_pca.shape

pca.explained_variance_ratio_
```

```
pca.n_components_

X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=30)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca, y_train)
accuracy_score(y_test, model.predict(X_test_pca))

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
X_pca.shape

pca.explained_variance_ratio_

X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=30)

model = LogisticRegression(max_iter=1000)
model.fit(X_train_pca, y_train)
accuracy_score(y_test, model.predict(X_test_pca))
```

Output:

```
(1797, 64)

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```






0.9722222222222222

(1797, 29)

array([0.14890594, 0.13618771, 0.11794594, 0.08409979, 0.05782415,
 0.0491691 , 0.04315987, 0.03661373, 0.03353248, 0.03078806,
 0.02372341, 0.02272697, 0.01821863, 0.01773855, 0.01467101,
 0.01409716, 0.01318589, 0.01248138, 0.01017718, 0.00905617,
 0.00889538, 0.00797123, 0.00767493, 0.00722904, 0.00695889,
 0.00596081, 0.00575615, 0.00515158, 0.0048954 ])

29

0.9694444444444444

(1797, 2)

array([0.14890594, 0.13618771])

0.6083333333333333