Homework 2

Name:	Hanumantha Rao Vakkalanka
Email:	hvakkala@uncc.edu
Student ID:	801333188
Course:	ECGR 5105 Intro to ML
Lab Number:	Spring 2023


```
import numpy as np   # Import the numpy library and give it the alias 'np'
import pandas as pd  # Import the pandas library and give it the alias 'pd'
import matplotlib.pyplot as plt  # Import the matplotlib.pyplot module and give it the alias 'plt'

from sklearn.preprocessing import StandardScaler  # Import the StandardScaler class from the preprocessing module of the scikit-learn library
from sklearn.preprocessing import MinMaxScaler  # Import the MinMaxScaler class from the preprocessing module of the scikit-learn library
from sklearn.model_selection import kfold  # Import the kfold class from the model_selection module of the scikit-learn library
from sklearn.model_selection import cross_val_score  # Import the cross_val_score function from the model_selection module of the scikit-learn library
from sklearn.linear_model import LogisticRegression  # Import the LogisticRegression class from the linear_model module of the scikit-learn library
from sklearn import datasets  # Import the datasets module from the scikit-learn library
from sklearn import metrics  # Import the metrics module from the scikit-learn library
from sklearn.metrics import confusion_matrix  # Import the confusion_matrix function from the metrics module of the scikit-learn library
from sklearn.metrics import classification_report  # Import the classification_report function from the metrics module of the scikit-learn library

df = pd.read_csv('/content/sample_data/diabetes.csv')  # Load a CSV file into a pandas DataFrame object
df  # Display the contents of the DataFrame object
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
```

```
1              1      85           66            29        0  26.6
2              8     183           64             0        0  23.3
3              1      89           66            23       94  28.1
4              0     137           40            35      168  43.1
..           ...     ...          ...           ...      ...   ...
763           10     101           76            48      180  32.9
764            2     122           70            27        0  36.8
765            5     121           72            23      112  26.2
766            1     126           60             0        0  30.1
767            1      93           70            31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
764                     0.340   27        0
765                     0.245   30        0
766                     0.349   47        1
767                     0.315   23        0

[768 rows x 9 columns]
```

```
train = df.sample(frac=0.8, random_state=0)  # Randomly select 80% of the rows from the DataFrame `df` and assign them to the `train` DataFrame
test = df.drop(train.index)  # Remove the rows that were selected for the `train` DataFrame and assign the remaining rows to the `test` DataFrame

train  #train the model
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
661            1      199             76             43        0  42.9
122            2      107             74             30      100  33.6
113            4       76             62              0        0  34.0
14             5      166             72             19      175  25.8
529            0      111             65              0        0  24.6
..           ...      ...            ...            ...      ...   ...
25            10      125             70             26      115  31.1
110            3      171             72             33      135  33.3
149            2       90             70             17        0  27.3
152            9      156             86             28      155  34.3
528            0      117             66             31      188  30.8

     DiabetesPedigreeFunction  Age  Outcome
661                     1.394   22        1
122                     0.404   23        0
113                     0.391   25        0
```

```
14                         0.587   51         1
529                        0.660   31         0
..                           ...  ...       ...
25                         0.205   41         1
110                        0.199   24         1
149                        0.085   22         0
152                        1.189   42         1
528                        0.493   22         0

[614 rows x 9 columns]
```

```
test #test the model
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
9              8      125             96              0        0   0.0
11            10      168             74              0        0  38.0
19             1      115             70             30       96  34.6
23             9      119             80             35        0  29.0
28            13      145             82             19      110  22.2
..           ...      ...            ...            ...      ...   ...
746            1      147             94             41        0  49.3
753            0      181             88             44      510  43.3
754            8      154             78             32        0  32.4
759            6      190             92              0        0  35.5
763           10      101             76             48      180  32.9

     DiabetesPedigreeFunction  Age  Outcome
9                       0.232   54        1
11                      0.537   34        1
19                      0.529   32        1
23                      0.263   29        1
28                      0.245   57        0
..                        ...  ...      ...
746                     0.358   27        1
753                     0.222   26        1
754                     0.443   45        1
759                     0.278   66        1
763                     0.171   63        0

[154 rows x 9 columns]
```

```
# Extract the predictor variables from the training set and assign them to
the `x_tr` DataFrame
x_tr = train[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]

# Extract the predictor variables from the testing set and assign them to the
`x_tst` DataFrame
x_tst = test[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]
```

```python
# Extract the outcome variable from the training set and assign it to a numpy array called `y_tr`
y_tr = np.array(train.Outcome)

# Extract the outcome variable from the testing set and assign it to a numpy array called `y_tst`
y_tst = np.array(test.Outcome)

# Create a MinMaxScaler object to perform min-max scaling on the `x_tr` DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tr` DataFrame and assign the result to a new object called `X`
X = min_max_Scaling.fit_transform(x_tr)

# Create a StandardScaler object to perform standardization on the `X` numpy array
sc = StandardScaler()

# Apply standardization to the `X` numpy array and assign the result to a new numpy array called `x_tring`
x_tring = sc.fit_transform(X)
x_tring
```

```
array([[-0.84710271,  2.41830371,  0.37313317, ...,  1.40057412,
         2.78993639, -0.98111368],
       [-0.55124318, -0.45753343,  0.27260497, ...,  0.19377803,
        -0.21742064, -0.89478468],
       [ 0.0404759 , -1.42656551, -0.33056422, ...,  0.24568324,
        -0.25691119, -0.72212667],
       ...,
       [-0.55124318, -0.98893812,  0.07154857, ..., -0.62372899,
        -1.18645791, -0.98111368],
       [ 1.51977358,  1.07416244,  0.87577416, ...,  0.28461215,
         2.16720085,  0.74546643],
       [-1.14296225, -0.14494244, -0.12950783, ..., -0.16955842,
         0.05293772, -0.98111368]])
```

```python
 # Create a MinMaxScaler object to perform min-max scaling on the `x_tst` DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tst` DataFrame and assign the result to a new object called `X`
X = min_max_Scaling.fit_transform(x_tst)

# Create a StandardScaler object to perform standardization on the `X` numpy array
sc = StandardScaler()
```

```
# Apply standardization to the `X` numpy array and assign the result to a new numpy array called `x_tst_std`
x_tst_std = sc.fit_transform(X)
x_tst_std
```

```
array([[ 1.27492994,  0.22340753,  1.47522188, ..., -3.70264042,
        -0.66557461,  1.71520033],
       [ 1.8781226 ,  1.58315447,  0.16580304, ...,  0.7587028 ,
         0.23610228,  0.10114075],
       [-0.83624437, -0.09281269, -0.07227312, ...,  0.35952998,
         0.21245174, -0.06026521],
       ...,
       [ 1.27492994,  1.14044617,  0.40387919, ...,  0.10124169,
        -0.04179158,  0.98887352],
       [ 0.67173728,  2.27883896,  1.23714573, ...,  0.46519338,
        -0.52958399,  2.68363609],
       [ 1.8781226 , -0.53552099,  0.28484111, ...,  0.15994358,
        -0.84590998,  2.44152715]])
```

```
# Create a LogisticRegression object with the 'liblinear' solver
model = LogisticRegression(solver='liblinear')

# Fit the logistic regression model on the standardized predictor variables in the training set (`x_tring`) and the corresponding response variable (`y_tr`)
model.fit(x_tring, y_tr)
```

```
LogisticRegression(solver='liblinear')
```

```
# Use the trained logistic regression model to make predictions on the standardized predictor variables in the test set (`x_tst_std`) and assign the predictions to a new numpy array called `y_pred`
y_pred = model.predict(x_tst_std)

# Print the predicted values of the response variable for the test set
print(y_pred)
```

```
[0 1 0 0 1 0 1 1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 1 0 0 0 1 1 0 0 0 1 0
 0 0 1 0 0 0 0 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 0 1 0 1 0 0
 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 1 1 1 0]
```

```
# Compute the accuracy of the predictions made by the logistic regression model on the test set

metrics.accuracy_score(y_tst,y_pred)
```

```
0.7662337662337663
```

```
# Compute the precision of the predictions made by the logistic regression
model on the test set
metrics.precision_score(y_tst,y_pred)
```

```
0.6444444444444445
```

```
# Compute the recall of the predictions made by the logistic regression model
on the test set
metrics.recall_score(y_tst,y_pred)
```

```
0.5918367346938775
```

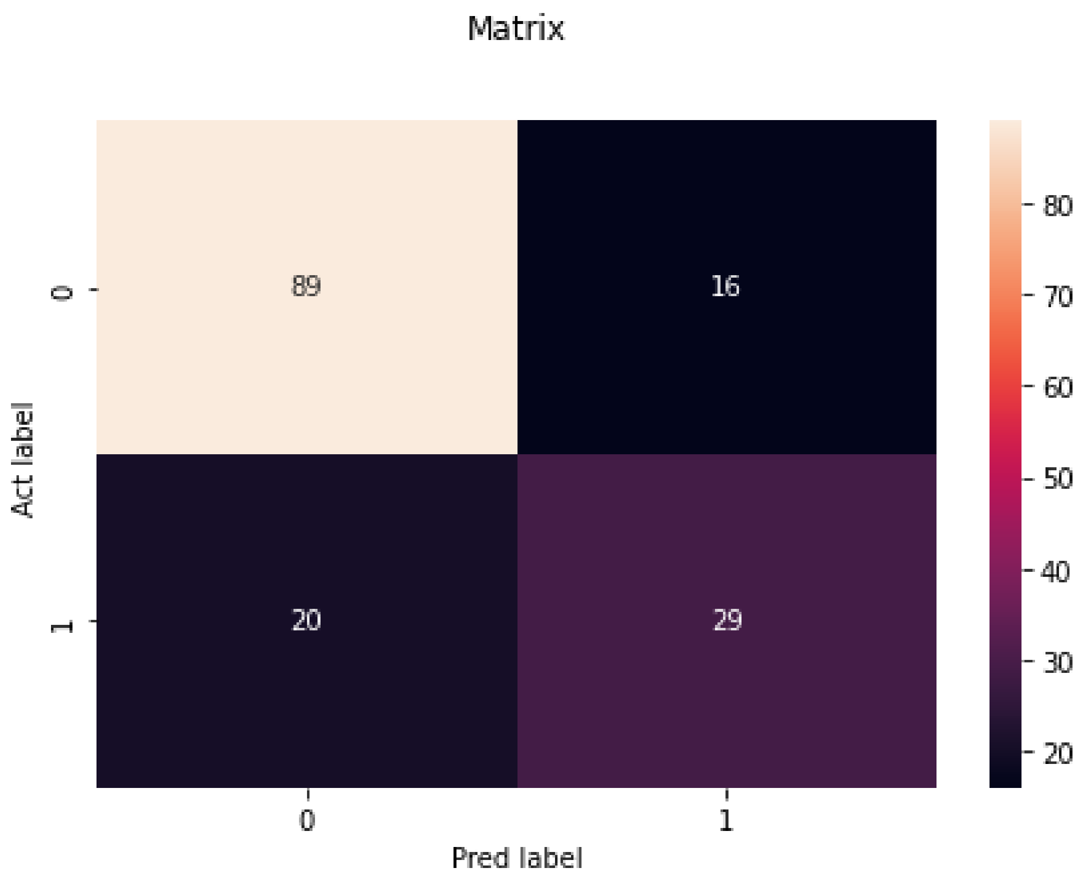
```
# Compute the confusion matrix for the predictions made by the logistic
regression model on the test set by comparing the predicted values (`y_pred`)
with the actual values of the response variable (`y_tst`) using the
`confusion_matrix` function from the `metrics` module
cnf_matrix = metrics.confusion_matrix(y_tst,y_pred)

cnf_matrix
```

```
array([[89, 16],
       [20, 29]])
```

```
# Import the Seaborn library for data visualization and plotting
import seaborn as sns

# Define the labels for the two classes in the classification problem
class_names = [0,1]

# Create a new figure and axis object for the heatmap plot
fig, ax = plt.subplots()

 # Create an array of tick locations for the heatmap axis labels
tick_marks = np.arange(len(class_names))

# Set the tick labels for the x-axis of the heatmap
plt.xticks(tick_marks, class_names)

# Set the tick labels for the y-axis of the heatmap
plt.yticks(tick_marks, class_names)



# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True,fmt='g')

#sets the position of the x-axis label to be at the top of the plot.
ax.xaxis.set_label_position("top")

# adjusts the spacing of the plot elements to avoid overlapping
plt.tight_layout()
```

```python
#sets the title of the plot to "Confusion matrix" and adjusts the position of the title along the y-axis.
plt.title('Matrix', y=1.1)

#sets the y-axis label to "Actual label".
plt.ylabel('Act label')

#sets the x-axis label to "Predicted label".
plt.xlabel('Pred label')
```

```
Text(0.5, 15.0, 'Pred label')
```



Problem 2

```python
import numpy as np   # Import the numpy library and give it the alias 'np'
import pandas as pd  # Import the pandas library and give it the alias 'pd'
```

```
import matplotlib.pyplot as plt  # Import the matplotlib.pyplot module and give it the alias 'plt'


from sklearn.preprocessing import StandardScaler  # Import the StandardScaler class from the preprocessing module of the scikit-learn library
from sklearn.preprocessing import MinMaxScaler  # Import the MinMaxScaler class from the preprocessing module of the scikit-learn library
from sklearn.model_selection import kfold  # Import the kfold class from the model_selection module of the scikit-learn library
from sklearn.model_selection import cross_val_score  # Import the cross_val_score function from the model_selection module of the scikit-learn library
from sklearn.linear_model import LogisticRegression  # Import the LogisticRegression class from the linear_model module of the scikit-learn library
from sklearn import datasets  # Import the datasets module from the scikit-learn library
from sklearn import metrics  # Import the metrics module from the scikit-learn library
from sklearn.metrics import confusion_matrix  # Import the confusion_matrix function from the metrics module of the scikit-learn library
from sklearn import model_selection #imports the model_selection module from the scikit-learn library

df = pd.read_csv('/content/sample_data/diabetes.csv')  # Load a CSV file into a pandas DataFrame object
df  # Display the contents of the DataFrame object
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           ...      ...            ...            ...      ...   ...
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age  Outcome
0                       0.627   50        1
1                       0.351   31        0
2                       0.672   32        1
3                       0.167   21        0
4                       2.288   33        1
..                        ...  ...      ...
763                     0.171   63        0
```

```
764                       0.340   27         0
765                       0.245   30         0
766                       0.349   47         1
767                       0.315   23         0

[768 rows x 9 columns]
```

```
#Selecting the features for X from the DataFrame 'df' and assigning to variable 'X'
X = df[['Pregnancies','Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age']]

#Extracting the 'Outcome' column from the DataFrame 'df' and converting to a NumPy array
Y = np.array(df.Outcome)

# Create a MinMaxScaler object to perform min-max scaling on the `x_tr` DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tr` DataFrame and assign the result to a new object called `X`
X = min_max_Scaling.fit_transform(X)

# Create a StandardScaler object to perform standardization on the `X` numpy array
sc = StandardScaler()

# Apply standardization to the `X` numpy array and assign the result to a new numpy array called `x_tring`
x_tring = sc.fit_transform(X)
x_tring
```

```
array([[ 0.63994726,  0.84832379,  0.14964075, ...,  0.20401277,
         0.46849198,  1.4259954 ],
       [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,
        -0.36506078, -0.19067191],
       [ 1.23388019,  1.94372388, -0.26394125, ..., -1.10325546,
         0.60439732, -0.10558415],
       ...,
       [ 0.3429808 ,  0.00330087,  0.14964075, ..., -0.73518964,
        -0.68519336, -0.27575966],
       [-0.84488505,  0.1597866 , -0.47073225, ..., -0.24020459,
        -0.37110101,  1.17073215],
       [-0.84488505, -0.8730192 ,  0.04624525, ..., -0.20212881,
        -0.47378505, -0.87137393]])
```

```
# initializes the kfold cross-validation method with 5 splits, a random state of 0, and shuffling the data before splitting
#kfold = kfold(n_splits=5, random_state=0, shuffle=True)
```

```python
# initializes the kfold cross-validation method with 10 splits, a random
state of 0, and shuffling the data before splitting
kfold = kfold(n_splits=10, random_state=0, shuffle=True)

#instantiates a logistic regression model with the 'liblinear' solver, which
is a solver for small datasets
model = LogisticRegression(solver='liblinear')

#Performing K-fold cross-validation
results = cross_val_score(model, X, Y, cv=kfold)

#Printing mean and standard deviation of the results
print("Accuracy: %0.3f%% (%0.3f%%)" % (results.mean()*100,
results.std()*100))
```

```
Accuracy: 76.946% (4.731%)
```

```python
from sklearn.metrics import make_scorer, accuracy_score, precision_score,
recall_score, f1_score

#Defining the scores to evaluate the model
scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score)}

#Using cross_validate to perform K-fold cross-validation on our logistic
regression model
results = model_selection.cross_validate(model,X,Y,cv=kfold,scoring=scoring)

#Storing the results of the cross-validation
a = results
a
```

```
{'fit_time': array([0.00344586, 0.00567126, 0.00232387, 0.0077908 ,
0.00215554]),
 'score_time': array([0.0063622 , 0.0090332 , 0.00456548, 0.00452542,
0.00657463]),
 'test_accuracy': array([0.80519481, 0.74675325, 0.76623377, 0.76470588,
0.74509804]),
 'test_precision': array([0.74285714, 0.81481481, 0.74468085, 0.8       ,
0.64705882]),
 'test_recall': array([0.55319149, 0.39285714, 0.59322034, 0.49122807,
0.44897959])}
```

Problem 3

```
import numpy as np   # Import the numpy library and give it the alias 'np'
import pandas as pd  # Import the pandas library and give it the alias 'pd'
import matplotlib.pyplot as plt  # Import the matplotlib.pyplot module and
give it the alias 'plt'


from sklearn.preprocessing import StandardScaler  # Import the StandardScaler
class from the preprocessing module of the scikit-learn library
from sklearn.preprocessing import MinMaxScaler  # Import the MinMaxScaler
class from the preprocessing module of the scikit-learn library
from sklearn.model_selection import kfold  # Import the kfold class from the
model_selection module of the scikit-learn library
from sklearn.model_selection import cross_val_score  # Import the
cross_val_score function from the model_selection module of the scikit-learn
library
from sklearn.linear_model import LogisticRegression  # Import the
LogisticRegression class from the linear_model module of the scikit-learn
library
from sklearn import datasets  # Import the datasets module from the
scikit-learn library
from sklearn import metrics  # Import the metrics module from the
scikit-learn library
from sklearn.metrics import confusion_matrix  # Import the confusion_matrix
function from the metrics module of the scikit-learn library
from sklearn.metrics import classification_report  # Import the
classification_report function from the metrics module of the scikit-learn
library
from sklearn.datasets import load_breast_cancer #Importing the required
library

#loading the dataset
breast = load_breast_cancer()

#Accessing the feature data of the breast cancer dataset
breast_data = breast.data

#Getting the shape of the feature data array
breast_data.shape
```

```
(569, 30)
```

```
#Convert the breast data into a pandas dataframe
breast_input = pd.DataFrame(breast_data)

#Display the first few rows of the dataframe
breast_input.head()
```

```
       0      1       2       3        4        5       6        7       8   \
0  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419   
1  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812   
2  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069   
3  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597   
4  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809   

         9   ...     20     21      22      23      24      25      26      27  
\
0  0.07871  ...  25.38  17.33  184.60  2019.0  0.1622  0.6656  0.7119  0.2654   
1  0.05667  ...  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860   
2  0.05999  ...  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430   
3  0.09744  ...  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575   
4  0.05883  ...  22.54  16.67  152.20  1575.0  0.1374  0.2050  0.4000  0.1625   

       28       29  
0  0.4601  0.11890  
1  0.2750  0.08902  
2  0.3613  0.08758  
3  0.6638  0.17300  
4  0.2364  0.07678  

[5 rows x 30 columns]
```

```python
#creates an array containing the target labels for the breast cancer dataset
breast_labels = breast.target

#returns the shape of the breast_labels array, which represents the target variable for the breast cancer dataset
breast_labels.shape

#Reshaping labels to (569,1)
labels = np.reshape(breast_labels,(569,1))

#create a new array that has the input data and the labels concatenated along the axis=1 (columns)
final_breast_data = np.concatenate([breast_data,labels],axis=1)

# returns the dimensions of final_breast_data array (number of rows, number of columns)
final_breast_data.shape

#create a pandas dataframe from final_breast_data
breast_dataset = pd.DataFrame(final_breast_data)

#set the feature names as columns of the dataframe
features = breast.feature_names

features
```

```
#add 'label' to features array
features_labels = np.append(features,'label')

#Renaming columns of the breast cancer dataset
breast_dataset.columns = features_labels

#returns the first 5 rows of the breast_dataset dataframe which consists of
the breast cancer data
breast_dataset.head()

   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points
\
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  label
0          0.4601                  0.11890    0.0
1          0.2750                  0.08902    0.0
2          0.3613                  0.08758    0.0
3          0.6638                  0.17300    0.0
4          0.2364                  0.07678    0.0

[5 rows x 31 columns]

# displays the last 5 rows of the breast cancer dataset
breast_dataset.tail()
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst texture  worst perimeter  worst area
\
564                 0.05623  ...          26.40           166.10      2027.0
565                 0.05533  ...          38.25           155.00      1731.0
566                 0.05648  ...          34.12           126.70      1124.0
567                 0.07016  ...          39.42           184.60      1821.0
568                 0.05884  ...          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
564           0.14100            0.21130           0.4107
565           0.11660            0.19220           0.3215
566           0.11390            0.30940           0.3403
567           0.16500            0.86810           0.9387
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  label
564                0.2216          0.2060                  0.07115    0.0
565                0.1628          0.2572                  0.06637    0.0
566                0.1418          0.2218                  0.07820    0.0
567                0.2650          0.4087                  0.12400    0.0
568                0.0000          0.2871                  0.07039    1.0

[5 rows x 31 columns]
```

```python
#Selecting a random sample of 80% of the dataset for training
train=breast_dataset.sample(frac=0.8,random_state=0)

#Dropping the selected training data from the dataset to create the test dataset
test=breast_dataset.drop(train.index)

train  #train the model
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
512        13.40         20.52           88.64      556.7          0.11060
457        13.21         25.25           84.10      537.9          0.08791
```

```
439         14.02         15.66            89.59      606.5            0.07966
298         14.26         18.17            91.22      633.1            0.06576
37          13.03         18.42            82.61      523.8            0.08983
..            ...           ...              ...        ...                ...
86          14.48         21.46            94.25      648.2            0.09444
266         10.60         18.95            69.28      346.4            0.09688
36          14.25         21.72            93.63      633.0            0.09823
193         12.34         26.86            81.15      477.4            0.10340
58          13.05         19.31            82.61      527.2            0.08060

     mean compactness  mean concavity  mean concave points  mean symmetry  \
512           0.14690        0.144500             0.081720         0.2116
457           0.05205        0.027720             0.020680         0.1619
439           0.05581        0.020870             0.026520         0.1589
298           0.05220        0.024750             0.013740         0.1635
37            0.03766        0.025620             0.029230         0.1467
..                ...             ...                  ...            ...
86            0.09947        0.120400             0.049380         0.2075
266           0.11470        0.063870             0.026420         0.1922
36            0.10980        0.131900             0.055980         0.1885
193           0.13530        0.108500             0.045620         0.1943
58            0.03789        0.000692             0.004167         0.1819

     mean fractal dimension  ...  worst texture  worst perimeter  worst area
\
512                 0.07325  ...          29.66           113.30       844.4
457                 0.05584  ...          34.23            91.29       632.9
439                 0.05586  ...          19.31            96.53       688.9
298                 0.05586  ...          25.26           105.80       819.7
37                  0.05863  ...          22.81            84.46       545.9
..                      ...  ...            ...              ...         ...
86                  0.05636  ...          29.25           108.40       808.9
266                 0.06491  ...          22.94            78.28       424.8
36                  0.06125  ...          30.36           116.20       799.6
193                 0.06937  ...          39.34           101.70       768.9
58                  0.05501  ...          22.25            90.24       624.1

     worst smoothness  worst compactness  worst concavity  \
512           0.15740            0.38560         0.510600
457           0.12890            0.10630         0.139000
439           0.10340            0.10170         0.062600
298           0.09445            0.21670         0.156500
37            0.09701            0.04619         0.048330
..                ...                ...              ...
86            0.13060            0.19760         0.334900
266           0.12130            0.25150         0.191600
36            0.14460            0.42380         0.518600
193           0.17850            0.47060         0.442500
58            0.10210            0.06191         0.001845
```

```
     worst concave points  worst symmetry  worst fractal dimension  label
512               0.20510          0.3585                  0.11090    0.0
457               0.06005          0.2444                  0.06788    1.0
439               0.08216          0.2136                  0.06710    1.0
298               0.07530          0.2636                  0.07676    1.0
37                0.05013          0.1987                  0.06169    1.0
..                    ...             ...                      ...    ...
86                0.12250          0.3020                  0.06846    0.0
266               0.07926          0.2940                  0.07587    1.0
36                0.14470          0.3591                  0.10140    0.0
193               0.14590          0.3215                  0.12050    0.0
58                0.01111          0.2439                  0.06289    1.0

[455 rows x 31 columns]
```

```
test #test the model
```

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
9          12.46         24.04           83.97      475.9          0.11860
23         21.16         23.04          137.20     1404.0          0.09428
28         15.30         25.27          102.40      732.4          0.10820
41         10.95         21.35           71.90      371.1          0.12270
..           ...           ...             ...        ...              ...
544        13.87         20.70           89.77      584.8          0.09578
551        11.13         22.44           71.49      378.4          0.09566
558        14.59         22.68           96.39      657.1          0.08473
559        11.51         23.93           74.52      403.5          0.09261
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
9             0.23960         0.22730              0.08543         0.2030
23            0.10220         0.10970              0.08632         0.1769
28            0.16970         0.16830              0.08751         0.1926
41            0.12180         0.10440              0.05669         0.1895
..                ...             ...                  ...            ...
544           0.10180         0.03688              0.02369         0.1620
551           0.08194         0.04824              0.02257         0.2030
558           0.13300         0.10290              0.03736         0.1454
559           0.10210         0.11120              0.04105         0.1388
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst texture  worst perimeter  worst area
\
0                   0.07871  ...          17.33           184.60      2019.0
9                   0.08243  ...          40.68            97.65       711.4
23                  0.05278  ...          35.59           188.00      2615.0
28                  0.06540  ...          36.71           149.30      1269.0
```

```
41                     0.06870  ...          35.34            87.22        514.0
..                         ...  ...            ...              ...          ...
544                    0.06688  ...          24.75            99.17        688.6
551                    0.06552  ...          28.26            77.80        436.6
558                    0.06147  ...          27.27           105.90        733.5
559                    0.06570  ...          37.16            82.28        474.2
568                    0.05884  ...          30.37            59.16        268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
9             0.18530            1.05800           1.1050
23            0.14010            0.26000           0.3155
28            0.16410            0.61100           0.6335
41            0.19090            0.26980           0.4023
..                ...                ...              ...
544           0.12640            0.20370           0.1377
551           0.10870            0.17820           0.1564
558           0.10260            0.31710           0.3662
559           0.12980            0.25170           0.3630
568           0.08996            0.06444           0.0000

     worst concave points  worst symmetry  worst fractal dimension  label
0                 0.26540          0.4601                  0.11890    0.0
9                 0.22100          0.4366                  0.20750    0.0
23                0.20090          0.2822                  0.07526    0.0
28                0.20240          0.4027                  0.09876    0.0
41                0.14240          0.2964                  0.09606    0.0
..                    ...             ...                      ...    ...
544               0.06845          0.2249                  0.08492    1.0
551               0.06413          0.3169                  0.08032    1.0
558               0.11050          0.2258                  0.08004    1.0
559               0.09653          0.2112                  0.08732    1.0
568               0.00000          0.2871                  0.07039    1.0

[114 rows x 31 columns]
```

```python
#selecting the first 30 columns of the train and test dataframes and storing them as x_tr and x_tst, respectively
x_tr = train.values[:,:30]
x_tst = test.values[:,:30]

#Getting the 'Y' (labels) of the training set
y_tr = np.array(train.label)

#Getting the 'Y' (labels) of the test set
y_tst = np.array(test.label)

#prints the shape (number of samples) of y_tr and y_tst.
y_tr.shape, y_tst.shape
```

```
((455,), (114,))
```

```
# Create a MinMaxScaler object to perform min-max scaling on the `x_tr`
DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tr` DataFrame and assign the result to a
new object called `X`
X = min_max_Scaling.fit_transform(x_tr)

# Create a StandardScaler object to perform standardization on the `X` numpy
array
sc = StandardScaler()

# Apply standardization to the `X` numpy array and assign the result to a new
numpy array called `x_tring`
x_tring = sc.fit_transform(X)
x_tring
```

```
(455, 30)
```

```
# Create a MinMaxScaler object to perform min-max scaling on the `x_tst`
DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tst` DataFrame and assign the result to a
new object called `X`
X = min_max_Scaling.fit_transform(x_tst)

# Create a StandardScaler object to perform standardization on the `X` numpy
array
sc = StandardScaler()

# Apply standardization to the `X` numpy array and assign the result to a new
numpy array called `x_tst_std`
x_tst_std = sc.fit_transform(X)
x_tst_std
```

```
(114, 30)
```

```
#creates a logistic regression model using the solver 'liblinear'
model = LogisticRegression(solver='liblinear')

#fitting the model to the training data, which has been standardized
model.fit(x_tring, y_tr)
```

```
LogisticRegression(solver='liblinear')
```

```
#Using the trained model to predict the class labels of the test set
y_pred = model.predict(x_tst_std)
```

```
#prints the predicted class labels of the test set
print(y_pred)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0.
 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0. 0. 0. 1.
 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 0.
 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1.
 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.]
```

```
# Compute the accuracy of the predictions made by the logistic regression
model on the test set
metrics.accuracy_score(y_tst,y_pred)
```

```
0.9912280701754386
```

```
# Compute the precision of the predictions made by the logistic regression
model on the test set
metrics.precision_score(y_tst,y_pred)
```

```
0.9859154929577465
```

```
# Compute the recall of the predictions made by the logistic regression model
on the test set
metrics.recall_score(y_tst,y_pred)
```

```
1.0
```

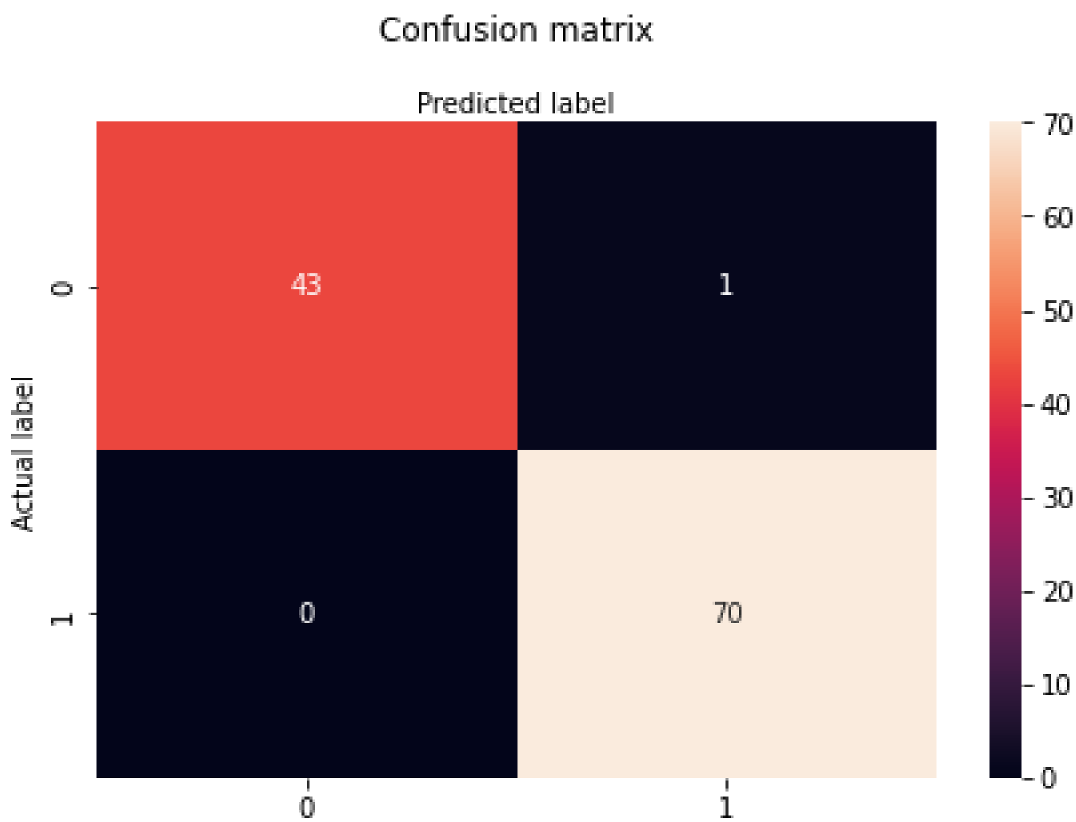
```
# Compute the confusion matrix for the predictions made by the logistic
regression model on the test set by comparing the predicted values (`y_pred`)
with the actual values of the response variable (`y_tst`) using the
`confusion_matrix` function from the `metrics` module

cnf_matrix = metrics.confusion_matrix(y_tst,y_pred)
cnf_matrix
```

```
array([[43,  1],
       [ 0, 70]], dtype=int64)
```

```
# Import the Seaborn library for data visualization and plotting
import seaborn as sns

# Define the labels for the two classes in the classification problem
class_names = [0,1]

# Create a new figure and axis object for the heatmap plot
fig, ax = plt.subplots()

 # Create an array of tick locations for the heatmap axis labels
tick_marks = np.arange(len(class_names))

# Set the tick labels for the x-axis of the heatmap
plt.xticks(tick_marks, class_names)
```

```python
# Set the tick labels for the y-axis of the heatmap
plt.yticks(tick_marks, class_names)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True,fmt='g')

#sets the position of the x-axis label to be at the top of the plot.
ax.xaxis.set_label_position("top")

# adjusts the spacing of the plot elements to avoid overlapping
plt.tight_layout()

#sets the title of the plot to "Confusion matrix" and adjusts the position of
the title along the y-axis.
plt.title('Matrix', y=1.1)

#sets the y-axis label to "Actual label".
plt.ylabel('Act label')

#sets the x-axis label to "Predicted label".
plt.xlabel('Pred label')
```

```
Text(0.5, 257.44, 'Predicted label')
```



```
#Defining a list of C values for weight penalties
C = [1, 0.5,0.1,  0.01,0.001]

#Looping through each value of C and fitting the model
for c in C:
    wp_model = LogisticRegression(penalty='l1', C = c, solver='liblinear') #
Creating a logistic regression model with L1 regularization and specified C
value
    wp_model.fit(x_tring, y_tr) # Fitting the model on standardized training
data
    print("C =", c) # Printing the C value for this run
    print("train accuracy: ", wp_model.score(x_tring, y_tr)) # Printing the
training accuracy score for this run
    print("testing accuracy: ", wp_model.score(x_tst_std, y_tst)) # Printing
the test accuracy score for this run
    print(' ') # Printing a space to separate results of each run
```

```
C = 1
training accuracy:  0.9868131868131869
test accuracy:  0.9736842105263158

C = 0.5
training accuracy:  0.9868131868131869
test accuracy:  0.9824561403508771

C = 0.1
training accuracy:  0.978021978021978
test accuracy:  0.9736842105263158

C = 0.01
training accuracy:  0.9406593406593406
test accuracy:  0.9473684210526315

C = 0.001
training accuracy:  0.36923076923076925
test accuracy:  0.38596491228070173
```

Problem 4

```
import numpy as np   # Import the numpy library and give it the alias 'np'
import pandas as pd  # Import the pandas library and give it the alias 'pd'
import matplotlib.pyplot as plt  # Import the matplotlib.pyplot module and
give it the alias 'plt'


from sklearn.preprocessing import StandardScaler  # Import the StandardScaler
class from the preprocessing module of the scikit-learn library
```

```
from sklearn.preprocessing import MinMaxScaler  # Import the MinMaxScaler
class from the preprocessing module of the scikit-learn library
from sklearn.model_selection import kfold  # Import the kfold class from the
model_selection module of the scikit-learn library
from sklearn.model_selection import cross_val_score  # Import the
cross_val_score function from the model_selection module of the scikit-learn
library
from sklearn.linear_model import LogisticRegression  # Import the
LogisticRegression class from the linear_model module of the scikit-learn
library
from sklearn import datasets  # Import the datasets module from the
scikit-learn library
from sklearn import metrics  # Import the metrics module from the
scikit-learn library
from sklearn.metrics import confusion_matrix  # Import the confusion_matrix
function from the metrics module of the scikit-learn library
from sklearn.metrics import classification_report  # Import the
classification_report function from the metrics module of the scikit-learn
library
from sklearn.datasets import load_breast_cancer #Importing the required
library

#loading the dataset
breast = load_breast_cancer()

##Accessing the feature data of the breast cancer dataset
breast_data = breast.data

#Getting the shape of the feature data array
breast_data.shape

(569, 30)

#Display the first few rows of the dataframe
breast_input = pd.DataFrame(breast_data)

#Display the first few rows of the dataframe
breast_input.head()

      0      1       2       3        4        5       6        7       8   \
0  17.99  10.38  122.80  1001.0  0.11840  0.27760  0.3001  0.14710  0.2419
1  20.57  17.77  132.90  1326.0  0.08474  0.07864  0.0869  0.07017  0.1812
2  19.69  21.25  130.00  1203.0  0.10960  0.15990  0.1974  0.12790  0.2069
3  11.42  20.38   77.58   386.1  0.14250  0.28390  0.2414  0.10520  0.2597
4  20.29  14.34  135.10  1297.0  0.10030  0.13280  0.1980  0.10430  0.1809

        9   ...     20     21      22      23      24      25      26      27
\
0  0.07871  ...  25.38  17.33  184.60  2019.0  0.1622  0.6656  0.7119  0.2654
1  0.05667  ...  24.99  23.41  158.80  1956.0  0.1238  0.1866  0.2416  0.1860
2  0.05999  ...  23.57  25.53  152.50  1709.0  0.1444  0.4245  0.4504  0.2430
```

```
3  0.09744  ...  14.91  26.50   98.87   567.7  0.2098  0.8663  0.6869  0.2575
4  0.05883  ...  22.54  16.67  152.20  1575.0  0.1374  0.2050  0.4000  0.1625

       28       29
0  0.4601  0.11890
1  0.2750  0.08902
2  0.3613  0.08758
3  0.6638  0.17300
4  0.2364  0.07678

[5 rows x 30 columns]
```

```
#reates an array containing the target labels for the breast cancer dataset
breast_labels = breast.target

#returns the shape of the breast_labels array, which represents the target
variable for the breast cancer dataset
breast_labels.shape
```

```
(569,)
```

```
#Reshaping labels to (569,1)
labels = np.reshape(breast_labels,(569,1))

#create a new array that has the input data and the labels concatenated along
the axis=1 (columns)
final_breast_data = np.concatenate([breast_data,labels],axis=1)

# returns the dimensions of final_breast_data array (number of rows, number
of columns)
final_breast_data.shape
```

```
(569, 31)
```

```
#create a pandas dataframe from final_breast_data
breast_dataset = pd.DataFrame(final_breast_data)

#set the feature names as columns of the dataframe
features = breast.feature_names

features
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
```

```
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')

#add 'label' to features array
features_labels = np.append(features,'label')

#Renaming columns of the breast cancer dataset
breast_dataset.columns = features_labels

#returns the first 5 rows of the breast_dataset dataframe which consists of
the breast cancer data
breast_dataset.head()

   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points
\
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  label
0          0.4601                  0.11890    0.0
1          0.2750                  0.08902    0.0
2          0.3613                  0.08758    0.0
3          0.6638                  0.17300    0.0
4          0.2364                  0.07678    0.0

[5 rows x 31 columns]
```

```python
#assigns the breast_input variable
x_tr = breast_input

#Getting the target labels
y_tr = np.array(breast_dataset.label)

# Create a MinMaxScaler object to perform min-max scaling on the `x_tr`
DataFrame
min_max_Scaling = MinMaxScaler()

# Apply min-max scaling to the `x_tr` DataFrame and assign the result to a
new object called `X`
X = min_max_Scaling.fit_transform(x_tr)

# Create a StandardScaler object to perform standardization on the `X` numpy
array
sc = StandardScaler()

# Apply standardization to the `X` numpy array and assign the result to a new
numpy array called `x_tring`
x_tring = sc.fit_transform(X)
x_tring
```

```
array([[ 1.09706398, -2.07333501,  1.26993369, ...,  2.29607613,
         2.75062224,  1.93701461],
       [ 1.82982061, -0.35363241,  1.68595471, ...,  1.0870843 ,
        -0.24388967,  0.28118999],
       [ 1.57988811,  0.45618695,  1.56650313, ...,  1.95500035,
         1.152255  ,  0.20139121],
       ...,
       [ 0.70228425,  2.0455738 ,  0.67267578, ...,  0.41406869,
        -1.10454895, -0.31840916],
       [ 1.83834103,  2.33645719,  1.98252415, ...,  2.28998549,
         1.91908301,  2.21963528],
       [-1.80840125,  1.22179204, -1.81438851, ..., -1.74506282,
        -0.04813821, -0.75120669]])
```

```python
# initializes the kfold cross-validation method with 5 splits, a random state
of 0, and shuffling the data before splitting
#kfold = kfold(n_splits=5, random_state=0, shuffle=True)

# initializes the kfold cross-validation method with 10 splits, a random
state of 0, and shuffling the data before splitting
kfold = kfold(n_splits=10, random_state=0, shuffle=True)

#instantiates a logistic regression model with the 'liblinear' solver, which
is a solver for small datasets
model = LogisticRegression(solver='liblinear')

#Performing K-fold cross-validation
results = cross_val_score(model, X, y_tr, cv=kfold)
```

```
# Compute the accuracy of the predictions made by the logistic regression model on the test set
#metrics.accuracy_score(y_tst,y_pred)

#Defining a list of C values for weight penalties
C = [1, 0.5,0.1,  0.01,0.001]

#Looping through each value of C and fitting the model
for c in C:
    wp_model = LogisticRegression(penalty='l1', C = c, solver='liblinear') # Creating a logistic regression model with L1 regularization and specified C value
    wp_model.fit(x_tring, y_tr) # Fitting the model on standardized training data
    print("C =", c) # Printing the C value for this run
    print("train accuracy: ", wp_model.score(x_tring, y_tr)) # Printing the training accuracy score for this run
    #print("testing accuracy: ", wp_model.score(x_tst__std, y_tst)) # Printing the test accuracy score for this run
    print(' ') # Printing a space to separate results of each run
```

```
C = 1
train accuracy:  0.9894551845342706

C = 0.5
train accuracy:  0.9894551845342706

C = 0.1
train accuracy:  0.9771528998242531

C = 0.01
train accuracy:  0.9402460456942003

C = 0.001
train accuracy:  0.37258347978910367
```