Homework 4

Name: Hanumantha Rao Vakkalanka
Email: hvakkala@uncc.edu
Student ID: 801333188
Course: Intro to ML

PROBLEM -1

```
import numpy as np  # Importing NumPy for numerical computations
import pandas as pd  # Importing Pandas for data manipulation
import matplotlib.pyplot as plt  # Importing Matplotlib for data visualization

from sklearn.preprocessing import StandardScaler  # Importing StandardScaler for feature scaling
from sklearn.preprocessing import MinMaxScaler  # Importing MinMaxScaler for feature scaling
from sklearn.model_selection import KFold  # Importing KFold for cross-validation
from sklearn.model_selection import cross_val_score  # Importing cross_val_score for cross-validation
from sklearn.linear_model import LogisticRegression  # Importing LogisticRegression for classification
from sklearn import datasets  # Importing datasets from sklearn
from sklearn import metrics  # Importing metrics for performance evaluation
from sklearn.metrics import confusion_matrix  # Importing confusion_matrix for performance evaluation
from sklearn.metrics import classification_report  # Importing classification_report for performance evaluation

from sklearn.datasets import load_breast_cancer  # Importing breast cancer dataset from sklearn
from sklearn.naive_bayes import GaussianNB  # Importing GaussianNB for classification
from sklearn.metrics import precision_recall_curve  # Importing precision_recall_curve for performance evaluation

# Load the breast cancer dataset from sklearn
breast = load_breast_cancer()
```

```python
# Extract the features (input data) from the breast cancer dataset
X = breast.data

# Extract the target labels (output data) from the breast cancer dataset
Y = breast.target

# Reshape the feature matrix X to have dimensions (569, 30)
X = np.reshape(X, (569, 30))

# Reshape the target labels Y to have dimensions (569, 1)
Y = np.reshape(Y, (569, 1))

# Create a pandas DataFrame from the reshaped target labels Y
df = pd.DataFrame(Y)

#Doing the SVM for 2 imp components

from sklearn.decomposition import PCA  # Import PCA from
sklearn.decomposition

# Initialize PCA with 2 components
pca = PCA(n_components=2)

# Fit PCA to the feature matrix X and transform it to obtain the principal
components
principalComponents = pca.fit_transform(X)

# Create a pandas DataFrame from the principal components with column names
principalDf = pd.DataFrame(data=principalComponents, columns=['principal
component 1', 'principal component 2'])



# Concatenate the principal components DataFrame and the target labels
DataFrame along the columns axis
finalDf1 = pd.concat([principalDf, df], axis=1)

# Display the concatenated DataFrame
finalDf1
```

```
     principal component 1  principal component 2  0
0              1160.142574            -293.917544  0
1              1269.122443              15.630182  0
2               995.793889              39.156743  0
3              -407.180803             -67.380320  0
4               930.341180             189.340742  0
..                     ...                    ... ..
564            1414.126684             110.222492  0
565            1045.018854              77.057589  0
566             314.501756              47.553525  0
567            1124.858115              34.129225  0
```

```
568             -771.527622            -88.643106  1

[569 rows x 3 columns]
```

```python
# Split the concatenated DataFrame into training and test sets
train = finalDf1.sample(frac=0.8, random_state=0)  # Randomly sample 80% for training set
test = finalDf1.drop(train.index)  # Remaining data becomes the test set

# Extract the features (principal component 1) and target labels from the training and test sets
X_train = train.values[:, 0:1]  # Extract the first column (principal component 1) as X_train
X_test = test.values[:, 0:1]  # Extract the first column (principal component 1) as X_test

# Extract the target labels (column 2) from the training and test sets
Y_train = train.values[:, 2]  # Extract column 2 (target labels) as Y_train
Y_test = test.values[:, 2]  # Extract column 2 (target labels) as Y_test

# Display the extracted target labels from the test set
Y_test
```

```
array([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., 0., 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.])
```

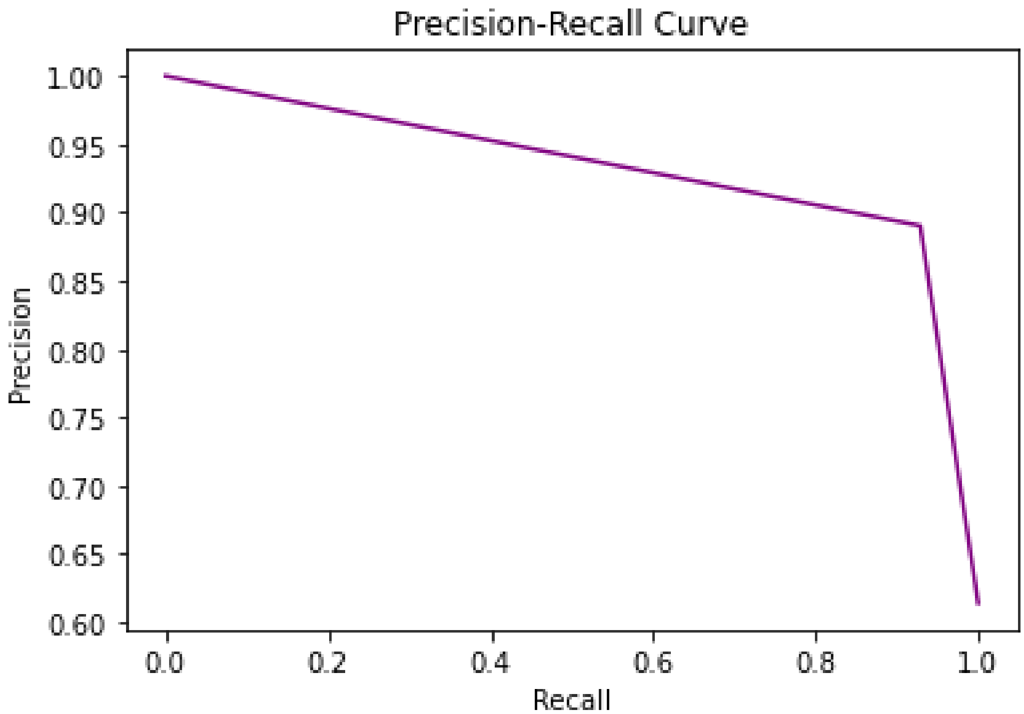
```python
# Import GridSearchCV and SVC from sklearn
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define the hyperparameter grid for GridSearchCV
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

# Create an instance of GridSearchCV with SVC as the estimator
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5)

# Fit the GridSearchCV to the training data
grid.fit(X_train, Y_train)
```

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})
```

```python
# Fit the GridSearchCV to the training data
grid.fit(X_train, Y_train)

# Predict the target labels on the test data
Y_predicted = grid.predict(X_test)

# Display the classification report
print(metrics.classification_report(Y_test, Y_predicted))

# Display the confusion matrix
print(metrics.confusion_matrix(Y_test, Y_predicted))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.82      0.85        44
         1.0       0.89      0.93      0.91        70

    accuracy                           0.89       114
   macro avg       0.88      0.87      0.88       114
weighted avg       0.89      0.89      0.89       114

[[36  8]
 [ 5 65]]
```

```python
# Calculate precision and recall
precision, recall, thresholds = precision_recall_curve(Y_test, Y_predicted)

# Create precision-recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

# Add axis labels to the plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

# Display the plot
plt.show()
```




```
# Doing SVM for 6 important components

from sklearn.decomposition import PCA

# Perform PCA with 6 components
pca = PCA(n_components=6)
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data=principalComponents,
                           columns=['principal component 1', 'principal
component 2', 'principal component 3',
                                    'principal component 4', 'principal
component 5', 'principal component 6'])

# Concatenate PCA components with target variable
finalDf2 = pd.concat([principalDf, df], axis=1)
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
3              -407.180803             -67.380320               8.672848
4               930.341180             189.340742               1.374801
..                     ...                    ...                    ...
564            1414.126684             110.222492              40.065944
565            1045.018854              77.057589               0.036669
566             314.501756              47.553525             -10.442407
567            1124.858115              34.129225             -19.742087
```

```
568             -771.527622            -88.643106            23.889032

     principal component 4  principal component 5  principal component 6  0
0                -8.711975              32.000486              1.265415  0
1                17.861283              -4.334874             -0.225872  0
2                 4.199340              -0.466529             -2.652811  0
3               -11.759867               7.115461              1.299436  0
4                 8.499183               7.613289              1.021160  0
..                     ...                    ...                   ... ..
564               6.562240              -5.102856             -0.395424  0
565              -4.753245             -12.417863             -0.059637  0
566              -9.771881              -6.156213             -0.870726  0
567             -23.660881               3.565133              4.086390  0
568               2.547249             -14.717566              4.418123  1

[569 rows x 7 columns]
```

```
# Splitting data into train and test sets

train2 = finalDf2.sample(frac=0.8, random_state=0)
test2 = finalDf2.drop(train2.index)

# Extracting features and target variables from train and test sets

X_train2 = train2.values[:, 0:5]
X_test2 = test2.values[:, 0:5]

Y_train2 = train2.values[:, 6]
Y_test2 = test2.values[:, 6]
```

```
array([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., 0., 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.])
```

```
# Performing Grid Search with SVM on the reduced-dimensionality data

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define the hyperparameter grid for Grid Search
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

# Create a GridSearchCV object with SVM classifier, hyperparameter grid, and
5-fold cross-validation
```

```
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5)

# Fit the grid search object to the reduced-dimensionality training data
grid.fit(X_train2, Y_train2)
```

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})
```

```
# Fitting the GridSearchCV object to the reduced-dimensionality training data

grid.fit(X_train2, Y_train2)
```

```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})
```

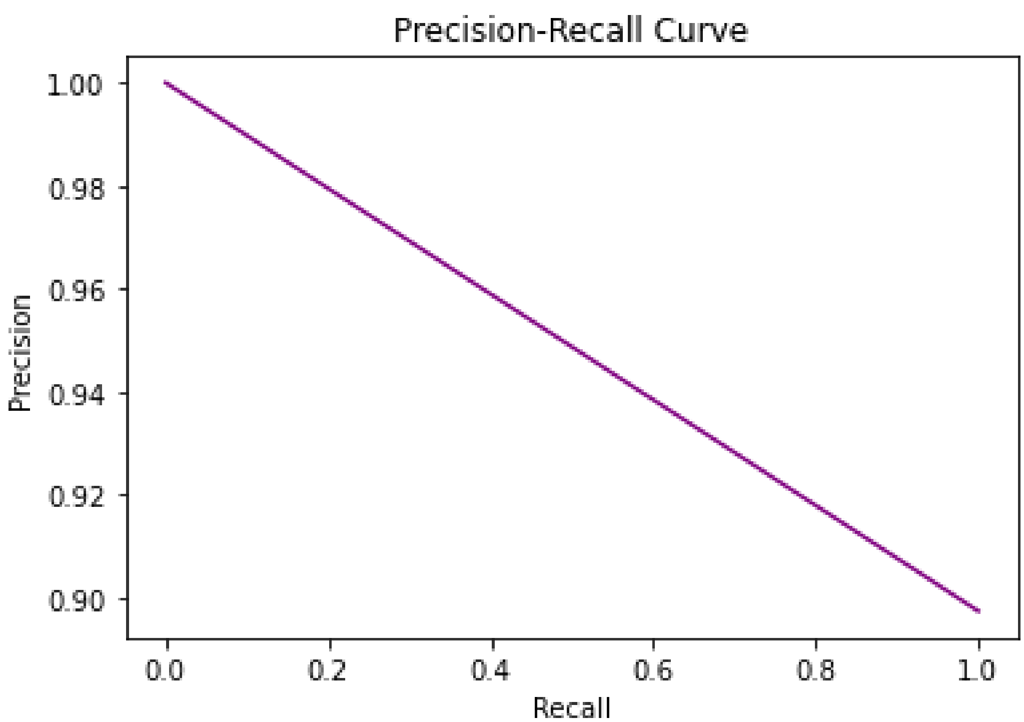
```
# Making predictions on the reduced-dimensionality test data

Y_predicted = grid.predict(X_test2)
Y_predicted
```

```
array([0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       0., 0., 0., 1., 1., 1., 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., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 0.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
# Printing classification report and confusion matrix

from sklearn import metrics

print(metrics.classification_report(Y_test2, Y_predicted))
print(metrics.confusion_matrix(Y_test2, Y_predicted))

# Calculating precision and recall

precision, recall, thresholds = precision_recall_curve(Y_test2, Y_predicted)

# Creating precision-recall curve plot

fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

# Adding axis labels to the plot
```

```
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')

# Displaying the plot

plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.82      0.90        44
         1.0       0.90      1.00      0.95        70

    accuracy                           0.93       114
   macro avg       0.95      0.91      0.92       114
weighted avg       0.94      0.93      0.93       114

[[36  8]
 [ 0 70]]
```




```
# Performing SVM for 18 important components

from sklearn.decomposition import PCA

# Applying PCA with n_components = 18
pca = PCA(n_components=18)
principalComponents = pca.fit_transform(X)
```

```
# Creating a DataFrame with the principal components as columns
principalDf = pd.DataFrame(data = principalComponents,
                           columns = ['principal component 1', 'principal
component 2','principal component 3',
                                      'principal component 4', 'principal
component 5','principal component 6',
                                      'principal component 7','principal
component 8', 'principal component 9',
                                      'principal component 10', 'principal
component 11','principal component 12',
                                      'principal component 13','principal
component 14', 'principal component 15',
                                      'principal component 16','principal
component 17', 'principal component 18'])

# Concatenating the principal components DataFrame with the original
DataFrame along the columns axis
finalDf3 = pd.concat([principalDf, df], axis = 1)
finalDf3
```

```
     principal component 1  principal component 2  principal component 3  \
0              1160.142574            -293.917544              48.578398
1              1269.122443              15.630182             -35.394534
2               995.793889              39.156743              -1.709753
3              -407.180803             -67.380320               8.672848
4               930.341180             189.340742               1.374801
..                     ...                    ...                    ...
564            1414.126684             110.222492              40.065944
565            1045.018854              77.057589               0.036669
566             314.501756              47.553525             -10.442407
567            1124.858115              34.129225             -19.742087
568            -771.527622             -88.643106              23.889032

     principal component 4  principal component 5  principal component 6  \
0                -8.711975              32.000486               1.265415
1                17.861283              -4.334874              -0.225872
2                 4.199340              -0.466529              -2.652811
3               -11.759867               7.115461               1.299436
4                 8.499183               7.613289               1.021160
..                     ...                    ...                    ...
564               6.562240              -5.102856              -0.395424
565              -4.753245             -12.417863              -0.059637
566              -9.771881              -6.156213              -0.870726
567             -23.660881               3.565133               4.086390
568               2.547249             -14.717566               4.418123

     principal component 7  principal component 8  principal component 9  \
0                 0.931337               0.148167               0.745463
1                -0.046037               0.200804              -0.485828
```

```
2               -0.779745               -0.274026               -0.173874
3               -1.267304               -0.060555               -0.330639
4               -0.335522                0.289109                0.036087
..                    ...                     ...                     ...
564             -0.786751                0.037082               -0.452530
565              0.449831                0.509154               -0.449986
566             -2.166493               -0.442279               -0.097398
567             -1.705401               -0.359964                0.385030
568             -2.815752                0.030039               -0.423451

     principal component 10  principal component 11  principal component 12
\
0                  0.589359               -0.307804                0.043452
1                 -0.084035                0.080642                0.033042
2                 -0.186994                0.279174               -0.020464
3                 -0.144155                0.927471               -0.174720
4                 -0.138502                0.042228               -0.062721
..                      ...                     ...                     ...
564               -0.235185                0.163649                0.052543
565                0.493247                0.007625                0.055832
566               -0.144667               -0.109147                0.076263
567                0.615467                0.307166               -0.028224
568               -0.301439                0.133353               -0.115105

     principal component 13  principal component 14  principal component 15
\
0                  0.034777                0.065069               -0.012934
1                  0.045485               -0.005534                0.021368
2                  0.083505                0.024824               -0.026887
3                  0.282556                0.080057                0.043201
4                 -0.114247                0.002274               -0.019548
..                      ...                     ...                     ...
564               -0.075032               -0.015211               -0.061390
565               -0.015163                0.009985                0.003312
566               -0.004448               -0.055285               -0.012459
567                0.060561               -0.037742               -0.031873
568               -0.019667                0.013734               -0.004134

     principal component 16  principal component 17  principal component 18
0
0                 -0.002670                0.018300                0.010263
0
1                 -0.028715                0.012371               -0.006009
0
2                 -0.041255                0.008218               -0.028044
0
3                 -0.034175                0.033742               -0.016965
0
4                  0.019932               -0.019201                0.004024
0
```

```
..                     ...                     ...                     ...
..
564               -0.054694               -0.004829               -0.011515
0
565               -0.020654                0.005197                0.002106
0
566               -0.005414                0.007866               -0.004484
0
567                0.020126                0.015243                0.043651
0
568                0.034264                0.009440               -0.028323
1

[569 rows x 19 columns]
```

```python
# Randomly sample 80% of rows from finalDf3 to create the training dataset
# Setting random_state to 0 for reproducibility
train3 = finalDf3.sample(frac=0.8, random_state=0)

# Drop the rows that were sampled for the training dataset to create the test
dataset
test3 = finalDf3.drop(train3.index)

# Extract the feature columns (principal components) from the training
dataset
X_train3 = train3.values[:, 0:17]

# Extract the feature columns (principal components) from the test dataset
X_test3 = test3.values[:, 0:17]

# Extract the target column from the training dataset
Y_train3 = train3.values[:, 18]

# Extract the target column from the test dataset
Y_test3 = test3.values[:, 18]
```

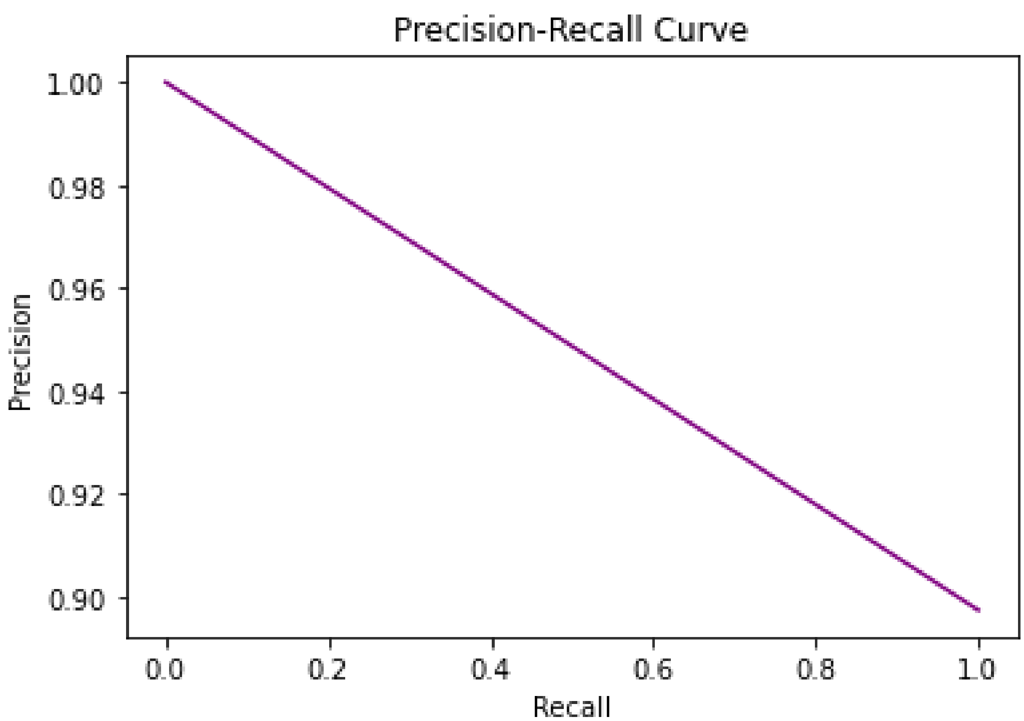
```
array([0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1.,
       1., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1., 1., 1., 0.,
       0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1.,
       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.])
```

```python
# Import necessary libraries for GridSearchCV and SVM
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define the hyperparameter grid for the SVM model
param_grid = {'C': [0.1, 1, 10, 100, 1000],
```

```
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

# Create an instance of GridSearchCV with SVC as the base estimator, the
defined parameter grid, refit=True for model re-fitting, and cv=5 for 5-fold
cross-validation
grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5)

# Fit the GridSearchCV object to the training data
grid.fit(X_train3, Y_train3)

GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})

# Fit the GridSearchCV object to the training data
grid.fit(X_train3, Y_train3)

GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']})

Y_predicted = grid.predict(X_test3)
Y_predicted

array([0., 0., 0., 0., 1., 0., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0.,
       1., 0., 1., 0., 0., 1., 1., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0.,
       0., 0., 0., 1., 1., 1., 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., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 0.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

print(metrics.classification_report(Y_test, Y_predicted))

print(metrics.confusion_matrix(Y_test,Y_predicted))


#calculate precision and recall
precision, recall, thresholds = precision_recall_curve(Y_test,Y_predicted)
#create precision recall curve
fig, ax = plt.subplots()
ax.plot(recall, precision, color='purple')

#add axis labels to plot
ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
```

```
#display plot
plt.show()
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.82      0.90        44
         1.0       0.90      1.00      0.95        70

    accuracy                           0.93       114
   macro avg       0.95      0.91      0.92       114
weighted avg       0.94      0.93      0.93       114

[[36  8]
 [ 0 70]]
```




```
# Trying different kernels

# SVM with linear kernel
# SVM with linear kernel and C=100.0
linear_svc=SVC(kernel='linear', C=100.0)
linear_svc.fit(X_train,Y_train)
# make predictions on test set
Y_pred_test=linear_svc.predict(X_test)
# compute and print accuracy score
print(metrics.classification_report(Y_test, Y_pred_test))
```

```
              precision    recall  f1-score   support
```

```
         0.0       0.95      0.84      0.89        44
         1.0       0.91      0.97      0.94        70

    accuracy                           0.92       114
   macro avg       0.93      0.91      0.91       114
weighted avg       0.92      0.92      0.92       114
```

```
#SVM with polynomial kernel and C=1000.0
poly_svc100=SVC(kernel='poly', C=1000.0)
# fit classifier to training set
poly_svc100.fit(X_train, Y_train)
# make predictions on test set
Y_pred=poly_svc100.predict(X_test)
# compute and print accuracy score
print(metrics.classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.66      0.79        44
         1.0       0.82      1.00      0.90        70

    accuracy                           0.87       114
   macro avg       0.91      0.83      0.85       114
weighted avg       0.89      0.87      0.86       114
```

```
# Run SVM with sigmoid kernel
# instantiate classifier with sigmoid kernel and C=100.0
sigmoid_svc100=SVC(kernel='sigmoid', C=100.0)
# fit classifier to training set
sigmoid_svc100.fit(X_train,Y_train)
# make predictions on test set
Y_pred=sigmoid_svc100.predict(X_test)
# compute and print accuracy score
print(metrics.classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.75      0.75      0.75        44
         1.0       0.84      0.84      0.84        70

    accuracy                           0.81       114
   macro avg       0.80      0.80      0.80       114
weighted avg       0.81      0.81      0.81       114
```

```
#accuracy graph
count = 30
while(count >= 2):
```

```python
    # Set the number of PCA components
    from sklearn.decomposition import PCA
    pca = PCA(n_components=count)
    # Perform PCA on the data
    principalComponents = pca.fit_transform(X)
    # Create a DataFrame to store the principal components
    principalDf = pd.DataFrame(data = principalComponents)
    # Concatenate the principal components with the original DataFrame
    finalDfi = pd.concat([principalDf, df], axis = 1)

    # Split the data into training and testing sets
    traini=finalDfi.sample(frac=0.8,random_state=0)
    testi=finalDfi.drop(traini.index)
    X_traini = traini.values[:,0:count-1]
    X_testi = testi.values[:,0:count-1]

    Y_traini = traini.values[:,count]
    Y_testi = testi.values[:,count]

    # Perform grid search with cross-validation to find the best SVM
hyperparameters
    from sklearn.model_selection import GridSearchCV
    from sklearn.svm import SVC
    param_grid = {'C': [0.1, 1, 10, 100, 1000],
          'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
          'kernel': ['rbf']}
    grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5)
    grid.fit(X_traini, Y_traini)

    grid.fit(X_traini,Y_traini)

    # Make predictions on the testing set
    Y_predicted = grid.predict(X_testi)

    # Print accuracy score
    print(metrics.accuracy_score(Y_testi, Y_predicted))

    # Decrement count for the next iteration
    count = count - 1
```

```
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
```

```
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9298245614035088
0.9385964912280702
0.9122807017543859
0.8859649122807017
```

PROBLEM-2

```
import numpy as np  # Import the NumPy library for numerical computing
import pandas as pd  # Import the Pandas library for data manipulation and
analysis
import matplotlib.pyplot as plt  # Import the Matplotlib library for data
visualization
from sklearn.preprocessing import StandardScaler  # Import the StandardScaler
class from the scikit-learn library for data normalization

data = pd.read_csv("/content/sample_data/Housing.csv")

dft = data.drop(columns=['furnishingstatus'])  # Create a new DataFrame "dft"
by dropping the 'furnishingstatus' column from the "data" DataFrame
col = dft.columns  # Get the column names of the "dft" DataFrame and store
them in the "col" variable

from google.colab import drive  # Import the 'drive' module from the
'google.colab' library for accessing Google Drive functionalities in Google
Colab
drive.mount('/content/drive')  # Mount Google Drive to the '/content/drive'
directory in the Google Colab environment
```

```
Mounted at /content/drive
```

```python
scaler = StandardScaler()  # Create an instance of the StandardScaler class
from the scikit-learn library for data normalization

#
def bin_map(var):
    # Create a copy of the input variable to avoid modifying the original
data
    var_copy = var.copy()

    # Loop through each element in the input variable
    for i in range(len(var_copy)):
        # Convert 'yes' to 1 and 'no' to 0
        if var_copy[i] == 'yes':
            var_copy[i] = 1
        elif var_copy[i] == 'no':
            var_copy[i] = 0

    # Return the modified variable
    return var_copy


# Define a mapping function to convert 'yes' to 1 and 'no' to 0
def bin_map(var):
    # Create a copy of the input variable to avoid modifying the original
data
    var_copy = var.copy()

    # Loop through each element in the input variable
    for i in range(len(var_copy)):
        # Convert 'yes' to 1 and 'no' to 0
        if var_copy[i] == 'yes':
            var_copy[i] = 1
        elif var_copy[i] == 'no':
            var_copy[i] = 0

    # Return the modified variable
    return var_copy

# Call the bin_map function on each column of the dft DataFrame
dft['mainroad'] = bin_map(dft['mainroad'])
dft['guestroom'] = bin_map(dft['guestroom'])
dft['basement'] = bin_map(dft['basement'])
dft['hotwaterheating'] = bin_map(dft['hotwaterheating'])
dft['airconditioning'] = bin_map(dft['airconditioning'])
dft['prefarea'] = bin_map(dft['prefarea'])

dft = scaler.fit_transform(dft)  # Use the `fit_transform` method of the
`StandardScaler` object to normalize the data in `dft`

Y = scaler.fit_transform(np.array(data.price).reshape(545,1))  # Use the
`fit_transform` method of the `StandardScaler` object to normalize the
```

```
'price' column of the `data` DataFrame, reshape it to a column vector with 545 rows, and store the normalized data in the variable `Y`

Wye = pd.DataFrame(Y)  # Create a new DataFrame `Wye` from the normalized data `Y` using the `pd.DataFrame` constructor

data = pd.DataFrame(dft, columns=col)  # Create a new DataFrame `data` from the normalized data `dft` using the `pd.DataFrame` constructor, specifying the column names as `col`

data  # Display the `data` DataFrame
```

```
        price      area  bedrooms  bathrooms   stories  mainroad  guestroom  \
0    4.566365  1.046726  1.403419   1.421812  1.378217  0.405623  -0.465315
1    4.004484  1.757010  1.403419   5.405809  2.532024  0.405623  -0.465315
2    4.004484  2.218232  0.047278   1.421812  0.224410  0.405623  -0.465315
3    3.985755  1.083624  1.403419   1.421812  0.224410  0.405623  -0.465315
4    3.554979  1.046726  1.403419  -0.570187  0.224410  0.405623   2.149083
..        ...       ...       ...        ...       ...       ...        ...
540 -1.576868 -0.991879 -1.308863  -0.570187 -0.929397  0.405623  -0.465315
541 -1.605149 -1.268613  0.047278  -0.570187 -0.929397 -2.465344  -0.465315
542 -1.614327 -0.705921 -1.308863  -0.570187 -0.929397  0.405623  -0.465315
543 -1.614327 -1.033389  0.047278  -0.570187 -0.929397 -2.465344  -0.465315
544 -1.614327 -0.599839  0.047278  -0.570187  0.224410  0.405623  -0.465315

     basement  hotwaterheating  airconditioning   parking  prefarea
0   -0.734539        -0.219265         1.472618  1.517692  1.804941
1   -0.734539        -0.219265         1.472618  2.679409 -0.554035
2    1.361397        -0.219265        -0.679063  1.517692  1.804941
3    1.361397        -0.219265         1.472618  2.679409  1.804941
4    1.361397        -0.219265         1.472618  1.517692 -0.554035
..        ...              ...              ...       ...       ...
540  1.361397        -0.219265        -0.679063  1.517692 -0.554035
541 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
542 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
543 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
544 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035

[545 rows x 12 columns]
```

```
Wye = pd.DataFrame(np.array(data.price))  # Create a new DataFrame `Wye` from the 'price' column of the `data` DataFrame using the `pd.DataFrame` constructor
```



```
Wye = pd.DataFrame(np.array(data.price))  # Create a new DataFrame `Wye` from the 'price' column of the `data` DataFrame using the `pd.DataFrame` constructor
```

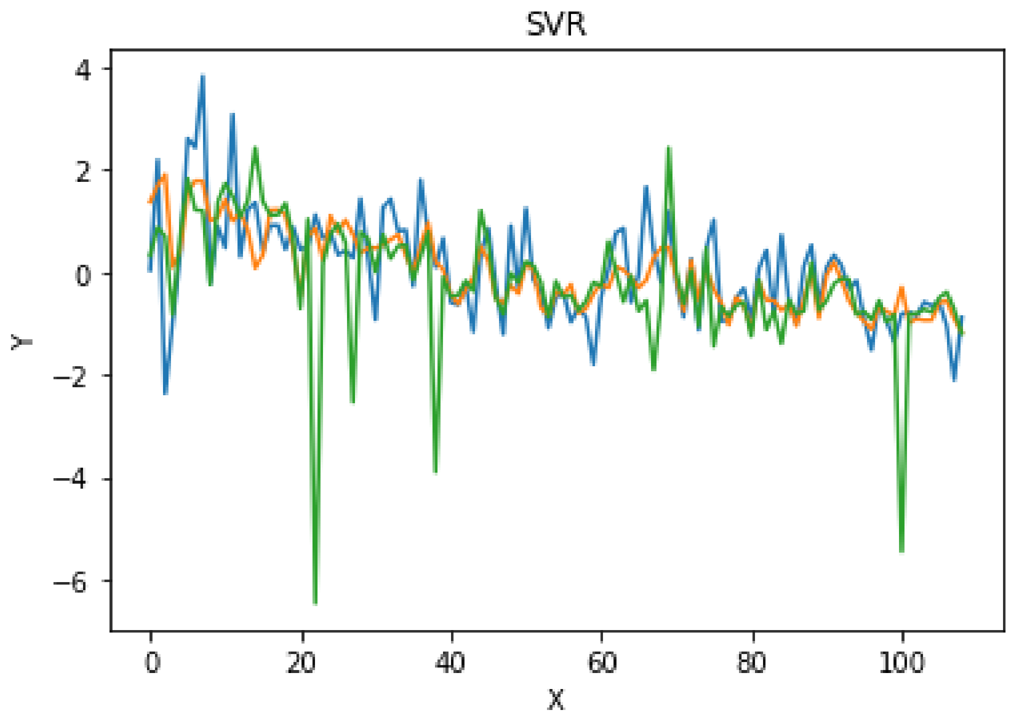
```python
train = data.sample(frac=0.8, random_state=1)  # Create a new DataFrame `train` by randomly sampling 80% of the rows from the `data` DataFrame using the `sample` method, with a random seed of 1

test = data.drop(train.index)  # Create a new DataFrame `test` by dropping the rows from the `data` DataFrame that are present in the `train` DataFrame using the `drop` method and passing the indices of `train` DataFrame as argument

y_train = pd.DataFrame(np.array(train.price))  # Create a new DataFrame `y_train` from the 'price' column of the `train` DataFrame, converting it to a NumPy array and then to a DataFrame

x_train = train.drop(columns=['price'])  # Create a new DataFrame `x_train` by dropping the 'price' column from the `train` DataFrame using the `drop` method

y_test = pd.DataFrame(np.array(test.price))  # Create a new DataFrame `y_test` from the 'price' column of the `test` DataFrame, converting it to a NumPy array and then to a DataFrame

x_test = np.array(test.drop(columns=['price']))  # Create a NumPy array `x_test` by dropping the 'price' column from the `test` DataFrame using the `drop` method, without converting it to a DataFrame


from sklearn.svm import SVR  # Import the Support Vector Regression (SVR) class from the scikit-learn library

svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)  # Create an instance of SVR with RBF kernel, and set the hyperparameters C and gamma to 1e3 and 0.1, respectively

svr_lin = SVR(kernel='linear', C=1e3)  # Create an instance of SVR with linear kernel, and set the hyperparameter C to 1e3

svr_poly = SVR(kernel='poly', C=1e3, degree=2)  # Create an instance of SVR with polynomial kernel of degree 2, and set the hyperparameters C and degree to 1e3 and 2, respectively

y_rbf = svr_rbf.fit(x_train, y_train).predict(x_test)  # Fit the SVR model with RBF kernel using the training data (x_train, y_train), and predict the target values for the test data (x_test). Store the predicted values in y_rbf.

y_lin = svr_lin.fit(x_train, y_train).predict(x_test)  # Fit the SVR model with linear kernel using the training data (x_train, y_train), and predict the target values for the test data (x_test). Store the predicted values in y_lin.
```

```
y_poly = svr_poly.fit(x_train, y_train).predict(x_test)  # Fit the SVR model with polynomial kernel using the training data (x_train, y_train), and predict the target values for the
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVR was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVR was fitted with feature names
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVR was fitted with feature names
  warnings.warn(
```

```
plt.plot(y_rbf)  # Plot the predicted target values obtained from SVR model with RBF kernel (y_rbf) as a line plot.

plt.plot(y_lin)  # Plot the predicted target values obtained from SVR model with linear kernel (y_lin) as a line plot.

plt.plot(y_poly)  # Plot the predicted target values obtained from SVR model with polynomial kernel (y_poly) as a line plot.

plt.xlabel('X')  # Set the label for x-axis as 'X'.

plt.ylabel('Y')  # Set the label for y-axis as 'Y'.

plt.title('SVR')  # Set the title of the plot as 'SVR', indicating that it represents the results of Support Vector Regression (SVR).
```

```
Text(0.5, 1.0, 'SVR')
```




```
import numpy as np  # Import the NumPy library for numerical computing
import pandas as pd  # Import the Pandas library for data manipulation and
analysis
import matplotlib.pyplot as plt  # Import the Matplotlib library for data
visualization
from sklearn.preprocessing import StandardScaler  # Import the StandardScaler
class from the scikit-learn library for data normalization

data = pd.read_csv("/content/sample_data/Housing.csv")

dft = data.drop(columns=['furnishingstatus'])  # Create a new DataFrame "dft"
by dropping the 'furnishingstatus' column from the "data" DataFrame
col = dft.columns  # Get the column names of the "dft" DataFrame and store
them in the "col" variable

scaler = StandardScaler()  # Create an instance of the StandardScaler class
from the scikit-learn library for data normalization

#
def bin_map(var):
    # Create a copy of the input variable to avoid modifying the original
data
    var_copy = var.copy()

    # Loop through each element in the input variable
    for i in range(len(var_copy)):
```

```python
        # Convert 'yes' to 1 and 'no' to 0
        if var_copy[i] == 'yes':
            var_copy[i] = 1
        elif var_copy[i] == 'no':
            var_copy[i] = 0

    # Return the modified variable
    return var_copy


# Define a mapping function to convert 'yes' to 1 and 'no' to 0
def bin_map(var):
    # Create a copy of the input variable to avoid modifying the original
data
    var_copy = var.copy()

    # Loop through each element in the input variable
    for i in range(len(var_copy)):
        # Convert 'yes' to 1 and 'no' to 0
        if var_copy[i] == 'yes':
            var_copy[i] = 1
        elif var_copy[i] == 'no':
            var_copy[i] = 0

    # Return the modified variable
    return var_copy

# Call the bin_map function on each column of the dft DataFrame
dft['mainroad'] = bin_map(dft['mainroad'])
dft['guestroom'] = bin_map(dft['guestroom'])
dft['basement'] = bin_map(dft['basement'])
dft['hotwaterheating'] = bin_map(dft['hotwaterheating'])
dft['airconditioning'] = bin_map(dft['airconditioning'])
dft['prefarea'] = bin_map(dft['prefarea'])

dft = scaler.fit_transform(dft)  # Use the `fit_transform` method of the
`StandardScaler` object to normalize the data in `dft`

Y = scaler.fit_transform(np.array(data.price).reshape(545,1))  # Use the
`fit_transform` method of the `StandardScaler` object to normalize the
'price' column of the `data` DataFrame, reshape it to a column vector with
545 rows, and store the normalized data in the variable `Y`

Wye = pd.DataFrame(Y)  # Create a new DataFrame `Wye` from the normalized
data `Y` using the `pd.DataFrame` constructor

data = pd.DataFrame(dft, columns=col)  # Create a new DataFrame `data` from
the normalized data `dft` using the `pd.DataFrame` constructor, specifying
the column names as `col`
```

```
data  # Display the `data` DataFrame
```

```
        price      area  bedrooms  bathrooms   stories  mainroad  guestroom  \
0    4.566365  1.046726  1.403419   1.421812  1.378217  0.405623  -0.465315
1    4.004484  1.757010  1.403419   5.405809  2.532024  0.405623  -0.465315
2    4.004484  2.218232  0.047278   1.421812  0.224410  0.405623  -0.465315
3    3.985755  1.083624  1.403419   1.421812  0.224410  0.405623  -0.465315
4    3.554979  1.046726  1.403419  -0.570187  0.224410  0.405623   2.149083
..        ...       ...       ...        ...       ...       ...        ...
540 -1.576868 -0.991879 -1.308863  -0.570187 -0.929397  0.405623  -0.465315
541 -1.605149 -1.268613  0.047278  -0.570187 -0.929397 -2.465344  -0.465315
542 -1.614327 -0.705921 -1.308863  -0.570187 -0.929397  0.405623  -0.465315
543 -1.614327 -1.033389  0.047278  -0.570187 -0.929397 -2.465344  -0.465315
544 -1.614327 -0.599839  0.047278  -0.570187  0.224410  0.405623  -0.465315

     basement  hotwaterheating  airconditioning   parking  prefarea
0   -0.734539        -0.219265         1.472618  1.517692  1.804941
1   -0.734539        -0.219265         1.472618  2.679409 -0.554035
2    1.361397        -0.219265        -0.679063  1.517692  1.804941
3    1.361397        -0.219265         1.472618  2.679409  1.804941
4    1.361397        -0.219265         1.472618  1.517692 -0.554035
..        ...              ...              ...       ...       ...
540  1.361397        -0.219265        -0.679063  1.517692 -0.554035
541 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
542 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
543 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035
544 -0.734539        -0.219265        -0.679063 -0.805741 -0.554035

[545 rows x 12 columns]
```

```
Wye = pd.DataFrame(np.array(data.price))  # Create a new DataFrame `Wye` from the 'price' column of the `data` DataFrame using the `pd.DataFrame` constructor
```



```
Wye = pd.DataFrame(np.array(data.price))  # Create a new DataFrame `Wye` from the 'price' column of the `data` DataFrame using the `pd.DataFrame` constructor
```



```
#d = pd.DataFrame(np.hstack([Ex,Wye]))
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(Ex)
principalDf = pd.DataFrame(data = principalComponents
 , columns = ['principal component 1', 'principal component 2'])
```

```
df1 = pd.DataFrame(np.hstack([principalDf, Wye])) # Creates a new DataFrame by horizontally stacking two arrays: principalDf and Wye.

df1

            0         1         2
0    3.264248 -1.129485  4.566365
1    5.194952 -3.347516  4.004484
2    2.460935  1.278579  4.004484
3    3.625400  0.538743  3.985755
4    2.502535  1.070341  3.554979
..        ...       ...       ...
540 -1.078671  1.444086 -1.576868
541 -2.533313 -0.848245 -1.605149
542 -2.063004  0.305558 -1.614327
543 -2.441185 -0.810540 -1.614327
544 -1.140103 -0.688090 -1.614327

[545 rows x 3 columns]

train = data.sample(frac=0.8, random_state=1)  # Create a new DataFrame `train` by randomly sampling 80% of the rows from the `data` DataFrame using the `sample` method, with a random seed of 1

test = data.drop(train.index)  # Create a new DataFrame `test` by dropping the rows from the `data` DataFrame that are present in the `train` DataFrame using the `drop` method and passing the indices of `train` DataFrame as argument

from sklearn.svm import SVR    # Importing Support Vector Regression (SVR) from scikit-learn library
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)   # Creating an instance of SVR with 'rbf' kernel, regularization parameter (C) set to 1e3, and gamma parameter set to 0.1

train1ex = np.array(train1.drop(columns=[2]))   # Extracting the data from 'train1' DataFrame by dropping the column with index 2 and converting it to a numpy array
train1ex = train1ex.reshape(436, 2)   # Reshaping the extracted data into a 2-dimensional array with 436 rows and 2 columns

train1wye = np.array(train1.drop(columns=[0,1]))   # Extracting the data from 'train1' DataFrame by dropping the columns with index 0 and 1, and converting it to a numpy array
train1wye = train1wye.reshape(436, 1)   # Reshaping the extracted data into a 2-dimensional array with 436 rows and 1 column
train1wye = train1wye.ravel()   # Flattening the 2-dimensional array into a 1-dimensional array

test1ex = np.array(test1.drop(columns=[2]))   # Extracting the data from 'test1' DataFrame by dropping the column with index 2, and converting it to a
```

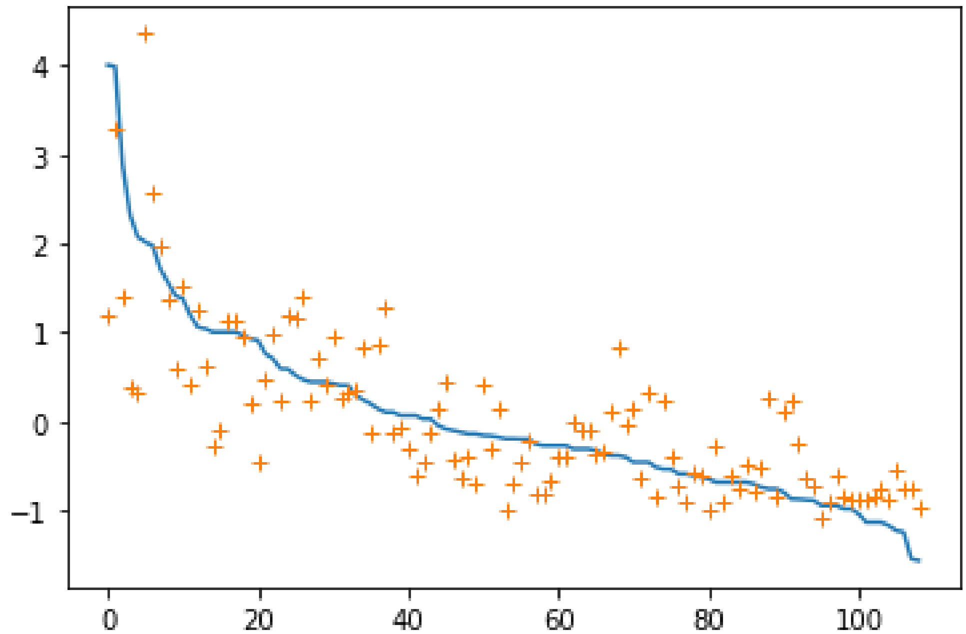
```
numpy array
test1ex = test1ex.reshape(109, 2)   # Reshaping the extracted data into a
2-dimensional array with 109 rows and 2 columns

test1wye = np.array(test1.drop(columns=[0,1]))   # Extracting the data from
'test1' DataFrame by dropping the columns with index 0 and 1, and converting
it to a numpy array
test1wye = test1wye.reshape(109, 1)   # Reshaping the extracted data into a
2-dimensional array with 109 rows and 1 column

model = svr_rbf.fit(train1ex, train1wye)   # Fitting the Support Vector
Regression (SVR) model with the training data 'train1ex' and 'train1wye'
using RBF kernel

prediction1 = model.predict(test1ex)    # Predicting the target values using
the trained SVR model and test data 'test1ex'
prediction1.reshape(109, 1)             # Reshaping the predicted values to
have dimensions (109, 1)

array([[ 1.19151924e+00],
       [ 3.30102310e+00],
       [ 1.39622186e+00],
       [ 3.93459752e-01],
       [ 3.36186921e-01],
       [ 4.36029068e+00],
       [ 2.58189216e+00],
       [ 1.96121431e+00],
       [ 1.38799113e+00],
       [ 5.98075163e-01],
       [ 1.53558191e+00],
       [ 4.23350039e-01],
       [ 1.23999285e+00],
       [ 6.28406224e-01],
       [-2.84284115e-01],
       [-9.25794346e-02],
       [ 1.13813357e+00],
       [ 1.14881210e+00],
       [ 9.57050121e-01],
       [ 2.11094127e-01],
       [-4.59462076e-01],
       [ 4.89055118e-01],
       [ 9.88438638e-01],
       [ 2.26987599e-01],
       [ 1.18487895e+00],
       [ 1.17244703e+00],
       [ 1.39474671e+00],
       [ 2.28369243e-01],
       [ 7.05032576e-01],
       [ 4.05263991e-01],
       [ 9.53412617e-01],
       [ 2.67012193e-01],
```

```
[ 3.18465224e-01],
[ 3.63766240e-01],
[ 8.22664454e-01],
[-1.23963110e-01],
[ 8.78462158e-01],
[ 1.28957501e+00],
[-1.34580596e-01],
[-5.37043563e-02],
[-3.04081703e-01],
[-6.07783029e-01],
[-4.67106739e-01],
[-1.08885329e-01],
[ 1.47711129e-01],
[ 4.50823183e-01],
[-4.29157181e-01],
[-6.36425238e-01],
[-3.80772185e-01],
[-6.98067381e-01],
[ 4.16601081e-01],
[-2.96608270e-01],
[ 1.54450890e-01],
[-9.82117842e-01],
[-6.92335414e-01],
[-4.62576470e-01],
[-2.20182045e-01],
[-8.08757955e-01],
[-8.08917695e-01],
[-6.51788685e-01],
[-3.80032844e-01],
[-3.78707046e-01],
[-3.61730447e-03],
[-9.55285148e-02],
[-1.08031964e-01],
[-3.66645425e-01],
[-3.44994357e-01],
[ 1.10838364e-01],
[ 8.24190922e-01],
[-3.82576758e-02],
[ 1.45870343e-01],
[-6.46586193e-01],
[ 3.23557673e-01],
[-8.33166551e-01],
[ 2.31803642e-01],
[-3.97594414e-01],
[-7.21799292e-01],
[-8.90278452e-01],
[-5.73060951e-01],
[-5.96010997e-01],
[-1.00565473e+00],
[-2.64599980e-01],
```

```
       [-9.03087824e-01],
       [-6.07433305e-01],
       [-7.46155877e-01],
       [-4.95912815e-01],
       [-7.74922359e-01],
       [-5.00362464e-01],
       [ 2.75375395e-01],
       [-8.56080255e-01],
       [ 1.19702367e-01],
       [ 2.26249858e-01],
       [-2.41687655e-01],
       [-6.22696587e-01],
       [-7.24296237e-01],
       [-1.08497870e+00],
       [-9.15844398e-01],
       [-5.94345980e-01],
       [-8.31366891e-01],
       [-8.78676322e-01],
       [-8.61433149e-01],
       [-8.79978297e-01],
       [-8.35936086e-01],
       [-7.39650680e-01],
       [-8.66176984e-01],
       [-5.48547119e-01],
       [-7.63062077e-01],
       [-7.38190753e-01],
       [-9.74750391e-01]])
```

```
plt.plot(test1wye)           # Plotting the actual target values from test data
plt.plot(prediction1, '+')   # Plotting the predicted target values with '+' marker
```

```
[<matplotlib.lines.Line2D at 0x231b82396a0>]
```




```
import sklearn.metrics as sm

#Print Mean Absolute Error
print("Mean absolute error =", round(sm.mean_absolute_error(test1wye,
prediction1), 2))

#Print Mean Squared Error
print("Mean squared error =", round(sm.mean_squared_error(test1wye,
prediction1), 2))

#Print Median Absolute Error
print("Median absolute error =", round(sm.median_absolute_error(test1wye,
prediction1), 2))

#Print Explained Variance Score
print("Explain variance score =", round(sm.explained_variance_score(test1wye,
prediction1), 2))

#Print R2 Score
print("R2 score =", round(sm.r2_score(test1wye, prediction1), 2))
```

```
Mean absolute error = 0.47
Mean squared error = 0.45
Median absolute error = 0.33
Explain variance score = 0.56
R2 score = 0.56
```

```
from sklearn.decomposition import PCA
```

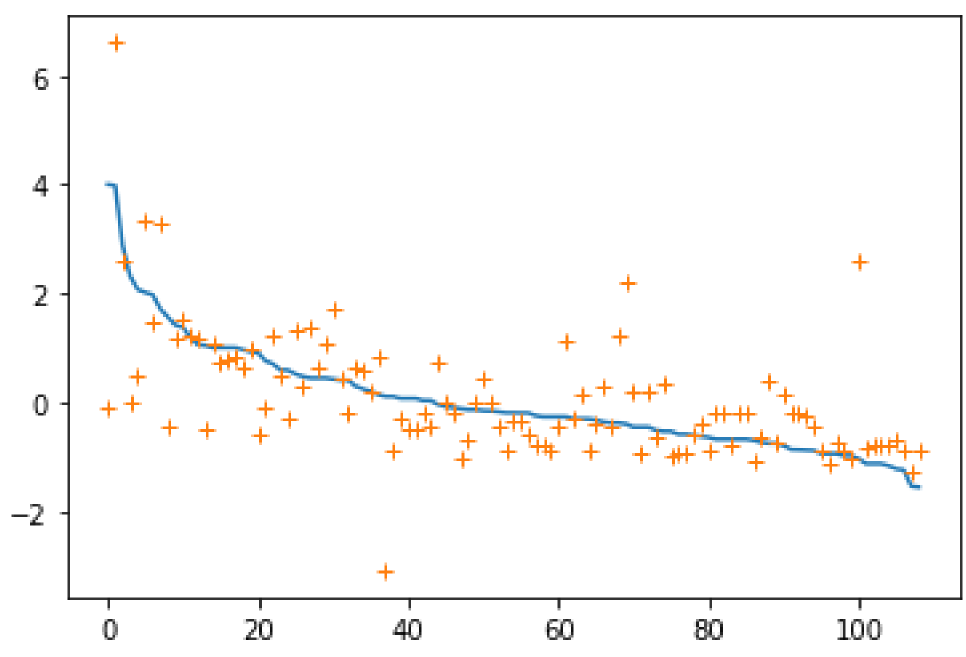
```
#Instantiate PCA with 6 components
pca = PCA(n_components=6)

#Fit and transform the data using PCA
principalComponents = pca.fit_transform(Ex)

#Create a DataFrame to store the principal components
principalDf = pd.DataFrame(data=principalComponents, columns=['principal
component 1', 'principal component 2', 'principal component 3', 'principal
component 4', 'principal component 5', 'principal component 6'])

#Concatenate the principal components with Wye
df2 = pd.DataFrame(np.hstack([principalDf, Wye]))

train2 = df2.sample(frac=0.8, random_state=1) # Create a training set by
randomly sampling 80% of the data from df2
test2 = df2.drop(train2.index) # Create a test set by removing the samples in
the training set from df2
from sklearn.svm import SVR # Import the SVR class from scikit-learn
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1) # Create an SVR model with RBF
kernel, C=1e3, and gamma=0.1
train2 # Display the training set, which is a subset of df2 after random
sampling
```

```
            0         1         2         3         4         5         6
62   1.668434 -1.577144 -0.350531 -0.284510 -0.738116 -0.047705  1.232537
247  2.043753 -1.676954 -1.799614  0.760626 -0.463055  0.400701 -0.115977
142  1.691797 -1.073583 -0.967244  0.813824 -0.525831  0.919381  0.445904
107  0.706713  0.801670  0.287002 -0.536019  1.818314  1.209268  0.726844
483 -0.640626 -0.483671 -0.736872 -0.194905  0.549685 -0.005113 -0.977528
..        ...       ...       ...       ...       ...       ...       ...
359 -1.203946 -0.065272 -0.719406  0.272859 -0.190551  0.152505 -0.565482
36   1.669630 -1.038315 -0.498018  3.720000  2.610809  0.056067  1.753214
30   2.528118 -2.072387 -1.250339 -0.374623 -0.620325 -0.739087  1.944253
20   0.044159  0.059643  0.115610  4.398738  0.810404 -1.162575  2.131547
527 -2.767009  0.582098  1.861415 -0.099819 -1.104261  0.482675 -1.333386

[436 rows x 7 columns]
```

```
train2ex = np.array(train2.drop(columns=[6])).reshape(436,6) # Extract the
features from the training set by dropping the column with index '6' and
reshaping the data into a 2D array with shape (436,6)

train2wye =
np.array(train2.drop(columns=[0,1,2,3,4,5])).reshape(436,1).ravel() # Extract
the target variable from the training set by dropping columns with indices
'0', '1', '2', '3', '4', '5', reshaping the data into a 1D array with shape
(436,) and flattening it
```

```
test2ex = np.array(test2.drop(columns=[6])).reshape(109,6) # Extract the features from the test set by dropping column with index '6', reshaping the data into a 2D array with shape (109, 6)

test2wye = np.array(test2.drop(columns=[0,1,2,3,4,5])).reshape(109,1) # Extract the target variable from the test set by dropping columns with indexes 0 to 5, reshaping the data into a 2D array with shape (109, 1)

model = svr_rbf.fit(train2ex,train2wye) # Fit the Support Vector Regression (SVR) model using the training data
prediction2 = model.predict(test2ex) # Make predictions on the test data using the trained model
prediction2.reshape(109, 1) # Reshape the prediction array to have shape (109, 1) for plotting

plt.plot(test2wye) # Plot the actual values of the target variable from the test set
plt.plot(prediction2,'+') # Plot the predicted values of the target variable using the SVR model
```

```
[<matplotlib.lines.Line2D at 0x231b816ec40>]
```




```
import sklearn.metrics as sm

#Print Mean Absolute Error
print("Mean absolute error =", round(sm.mean_absolute_error(test1wye, prediction1), 2))

#Print Mean Squared Error
print("Mean squared error =", round(sm.mean_squared_error(test1wye,
```

```
prediction1), 2))

#Print Median Absolute Error
print("Median absolute error =", round(sm.median_absolute_error(test1wye,
prediction1), 2))

#Print Explained Variance Score
print("Explain variance score =", round(sm.explained_variance_score(test1wye,
prediction1), 2))

#Print R2 Score
print("R2 score =", round(sm.r2_score(test1wye, prediction1), 2))
```

```
Mean absolute error = 0.63
Mean squared error = 0.94
Median absolute error = 0.44
Explain variance score = 0.1
R2 score = 0.09
```

```
from sklearn.decomposition import PCA # Import the Principal Component
Analysis (PCA) module
pca = PCA(n_components=11) # Initialize the PCA model with 11 principal
components
principalComponents = pca.fit_transform(Ex) # Fit the PCA model on the input
data
principalDf = pd.DataFrame(data = principalComponents,
columns = ['principal component 1', 'principal component 2', 'principal
component 3',
'principal component 4', 'principal component 5', 'principal component 6',
'principal component 7', 'principal component 8', 'principal component 9',
'principal component 10', 'principal component 11']) # Create a dataframe to
store the principal components

,'principal component 9','principal component 10','principal component 11'
df3 = pd.DataFrame(np.hstack([principalDf,Wye])) # Combine the principal
components dataframe with the target variable dataframe to create a new
dataframe for further processing or analysis.

train3 = df3.sample(frac=0.8, random_state=1) # Create a training set by
randomly sampling 80% of the data from df3 with a random state of 1 for
reproducibility
test3 = df3.drop(train3.index) # Create a test set by removing the samples in
the training set from df3

from sklearn.svm import SVR # Import the Support Vector Regression (SVR)
module
svr_rbf = SVR(kernel='linear', C=1e3, gamma=0.1) # Initialize the SVR model
```

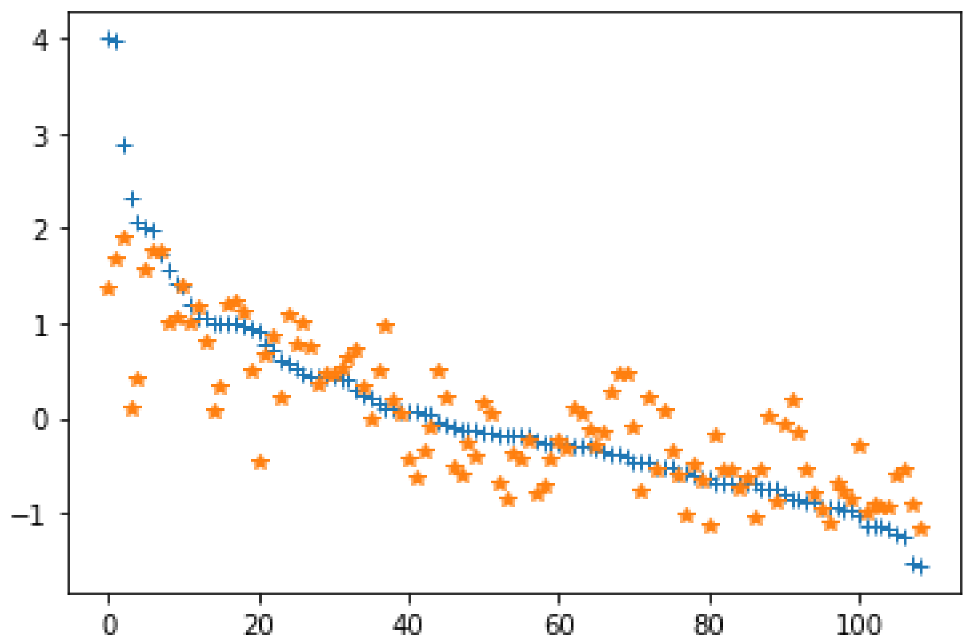
```
with a linear kernel, C value of 1e3, and gamma value of 0.1
train3 # Display the training set for further examination or analysis
```

```
            0         1         2         3         4         5         6  \
62   1.668434 -1.577144 -0.350531 -0.284510 -0.738116 -0.047705 -0.121506
247  2.043753 -1.676954 -1.799614  0.760626 -0.463055  0.400701 -0.852142
142  1.691797 -1.073583 -0.967244  0.813824 -0.525831  0.919381 -1.288370
107  0.706713  0.801670  0.287002 -0.536019  1.818314  1.209268  0.212905
483 -0.640626 -0.483671 -0.736872 -0.194905  0.549685 -0.005113 -0.635587
..        ...       ...       ...       ...       ...       ...       ...
359 -1.203946 -0.065272 -0.719406  0.272859 -0.190551  0.152505 -0.658521
36   1.669630 -1.038315 -0.498018  3.720000  2.610809  0.056067  1.965541
30   2.528118 -2.072387 -1.250339 -0.374623 -0.620325 -0.739087  0.054959
20   0.044159  0.059643  0.115610  4.398738  0.810404 -1.162575  1.328682
527 -2.767009  0.582098  1.861415 -0.099819 -1.104261  0.482675  0.698249

            7         8         9        10        11
62  -0.160084  0.522901  0.678605 -1.033756  1.232537
247 -0.194329 -2.276169 -0.701348  1.500006 -0.115977
142  1.552603  0.030389  1.088125 -0.051945  0.445904
107 -0.145298 -0.293530  0.394505  1.622250  0.726844
483  0.720711 -0.405571  0.746588  0.217113 -0.977528
..        ...       ...       ...       ...       ...
359 -0.470938 -0.347374 -0.178396 -0.807956 -0.565482
36   0.918476  0.903971 -0.430595  0.405776  1.753214
30  -0.117463  0.502399 -0.585040  1.183018  1.944253
20  -1.589544 -0.499869  0.278410  0.796768  2.131547
527 -0.361066  0.474880 -0.202054  1.022571 -1.333386

[436 rows x 12 columns]
```

```
train3ex = np.array(train3.drop(columns=[11])).reshape(436,11)

#Convert the training set to a numpy array and drop the column with index 11,
then reshape it into a 2-dimensional array with 436 rows and 11 columns.

train3wye =
np.array(train3.drop(columns=[0,1,2,3,4,5,6,7,8,9,10])).reshape(436,1).ravel(
)train3wye =
np.array(train3.drop(columns=[0,1,2,3,4,5,6,7,8,9,10])).reshape(436,1).ravel(
)

#Convert the target variable of the training set to a numpy array and drop
the columns with indices 0 to 10, then reshape it into a 1-dimensional array
with 436 elements using the ravel() function.

test3ex = np.array(test3.drop(columns=[11])).reshape(109,11)

#Convert the features of the test set to a numpy array and drop the column
```

```
with index 11, then reshape it into a 2-dimensional array with 109 rows and
11 columns.

test3wye =
np.array(test3.drop(columns=[0,1,2,3,4,5,6,7,8,9,10])).reshape(109,1)

#Convert the target variable of the test set to a numpy array and drop the
columns with indices 0 to 10, then reshape it into a 2-dimensional array with
109 rows and 1 column.

#Import the SVR class from sklearn.svm module
from sklearn.svm import SVR

#Create an instance of SVR with linear kernel and specified hyperparameters
svr_rbf = SVR(kernel='linear', C=1e3, gamma=0.1)

#Fit the SVR model to the training data using train3ex as input features and
train3wye as target variable
model = svr_rbf.fit(train3ex,train3wye)

#Make predictions on the test data using the trained SVR model
prediction3 = model.predict(test3ex)

#Reshape the predicted values to a 2D array with shape (109, 1)
prediction3 = prediction3.reshape(109, 1)

#Plot the actual and predicted values
plt.plot(test3wye, label='Actual')
plt.plot(prediction3, label='Predicted')

#Add a legend to the plot for identifying actual and predicted lines
plt.legend(loc='best')
```

```
[<matplotlib.lines.Line2D at 0x231b7e30e20>]
```




```
import sklearn.metrics as sm

#Print Mean Absolute Error
print("Mean absolute error =", round(sm.mean_absolute_error(test1wye,
prediction1), 2))

#Print Mean Squared Error
print("Mean squared error =", round(sm.mean_squared_error(test1wye,
prediction1), 2))

#Print Median Absolute Error
print("Median absolute error =", round(sm.median_absolute_error(test1wye,
prediction1), 2))

#Print Explained Variance Score
print("Explain variance score =", round(sm.explained_variance_score(test1wye,
prediction1), 2))

#Print R2 Score
print("R2 score =", round(sm.r2_score(test1wye, prediction1), 2))
```

```
Mean absolute error = 0.41
Mean squared error = 0.37
Median absolute error = 0.28
Explain variance score = 0.65
R2 score = 0.64
```

```
#Import the LinearRegression class from sklearn.linear_model module
from sklearn.linear_model import LinearRegression
```

```python
#Create an instance of LinearRegression
model = LinearRegression()

#Create a dataframe "datalin" by horizontally stacking the "Ex" features and
"Wye" target variable using np.hstack
datalin = pd.DataFrame(np.hstack([Ex, Wye]))
```

```
            0         1         2         3         4         5         6  \
0    1.046726  1.403419  1.421812  1.378217  0.405623 -0.465315 -0.734539
1    1.757010  1.403419  5.405809  2.532024  0.405623 -0.465315 -0.734539
2    2.218232  0.047278  1.421812  0.224410  0.405623 -0.465315  1.361397
3    1.083624  1.403419  1.421812  0.224410  0.405623 -0.465315  1.361397
4    1.046726  1.403419 -0.570187  0.224410  0.405623  2.149083  1.361397
..        ...       ...       ...       ...       ...       ...       ...
540 -0.991879 -1.308863 -0.570187 -0.929397  0.405623 -0.465315  1.361397
541 -1.268613  0.047278 -0.570187 -0.929397 -2.465344 -0.465315 -0.734539
542 -0.705921 -1.308863 -0.570187 -0.929397  0.405623 -0.465315 -0.734539
543 -1.033389  0.047278 -0.570187 -0.929397 -2.465344 -0.465315 -0.734539
544 -0.599839  0.047278 -0.570187  0.224410  0.405623 -0.465315 -0.734539

            7         8         9        10        11
0   -0.219265  1.472618  1.517692  1.804941  4.566365
1   -0.219265  1.472618  2.679409 -0.554035  4.004484
2   -0.219265 -0.679063  1.517692  1.804941  4.004484
3   -0.219265  1.472618  2.679409  1.804941  3.985755
4   -0.219265  1.472618  1.517692 -0.554035  3.554979
..        ...       ...       ...       ...       ...
540 -0.219265 -0.679063  1.517692 -0.554035 -1.576868
541 -0.219265 -0.679063 -0.805741 -0.554035 -1.605149
542 -0.219265 -0.679063 -0.805741 -0.554035 -1.614327
543 -0.219265 -0.679063 -0.805741 -0.554035 -1.614327
544 -0.219265 -0.679063 -0.805741 -0.554035 -1.614327

[545 rows x 12 columns]
```

```python
#Randomly sample 80% of the data for training
train_lin = datalin.sample(frac=0.8, random_state=1)

#Use the remaining data as testing set
test_lin = datalin.drop(train_lin.index)

#Get the shape of the testing set
test_lin_shape = test_lin.shape
```

```
(109, 12)
```

```python
testlinY = test_lin.drop(columns=[0,1,2,3,4,5,6,7,8,9,10]) #Drop columns 0 to
10 (inclusive) from the testing set to get the target variable
```

```
#Train the linear regression model using the training data, drop column 11
from the training set as input features (X), drop columns 0 to 10 (inclusive)
from the training set as target variable (y)

model.fit(train_lin.drop(columns=[11]),train_lin.drop(columns=[0,1,2,3,4,5,6,
7,8,9,10]))

LinearRegression()

#Predict using the linear regression model
pred = model.predict(test_lin.drop(columns=[11]))

import sklearn.metrics as sm

#Print Mean Absolute Error
print("Mean absolute error =", round(sm.mean_absolute_error(test1wye,
prediction1), 2))

#Print Mean Squared Error
print("Mean squared error =", round(sm.mean_squared_error(test1wye,
prediction1), 2))

#Print Median Absolute Error
print("Median absolute error =", round(sm.median_absolute_error(test1wye,
prediction1), 2))

#Print Explained Variance Score
print("Explain variance score =", round(sm.explained_variance_score(test1wye,
prediction1), 2))

#Print R2 Score
print("R2 score =", round(sm.r2_score(test1wye, prediction1), 2))

Mean absolute error = 0.44
Mean squared error = 0.38
Median absolute error = 0.33
Explain variance score = 0.63
R2 score = 0.63
```