Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe23 - Dimension Reduction Exercise

Name: Chong Mun Chen

IC Number: 960327-07-5097

Date : 24/7/2023

Introduction : Practising with Standard Scaler and PCA on this exercise.

Conclusion : Succeeded in reducing the complexity of the training model with Standard Scaler and PCA.

Dimension Reduction Exercise

Principal Component Analysis

Let's begin by importing the various library

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_breast_cancer
```

In [3]:
```
breast_cancer = load_breast_cancer()
df = pd.DataFrame(data = breast_cancer.data, columns = breast_cancer.featur
breast_cancer.keys()
```

Out[3]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_n
ames', 'filename', 'data_module'])
```

In [4]:
```
df.head()
```

Out[4]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	m symm
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1

5 rows × 30 columns

In [5]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [6]:
```
from sklearn.decomposition import PCA
pca = PCA(copy=True, iterated_power='auto', n_components=2, random_state=No
```

In [7]:
```
x = sc.fit_transform(df)
```

In [8]:
```
x.shape
```

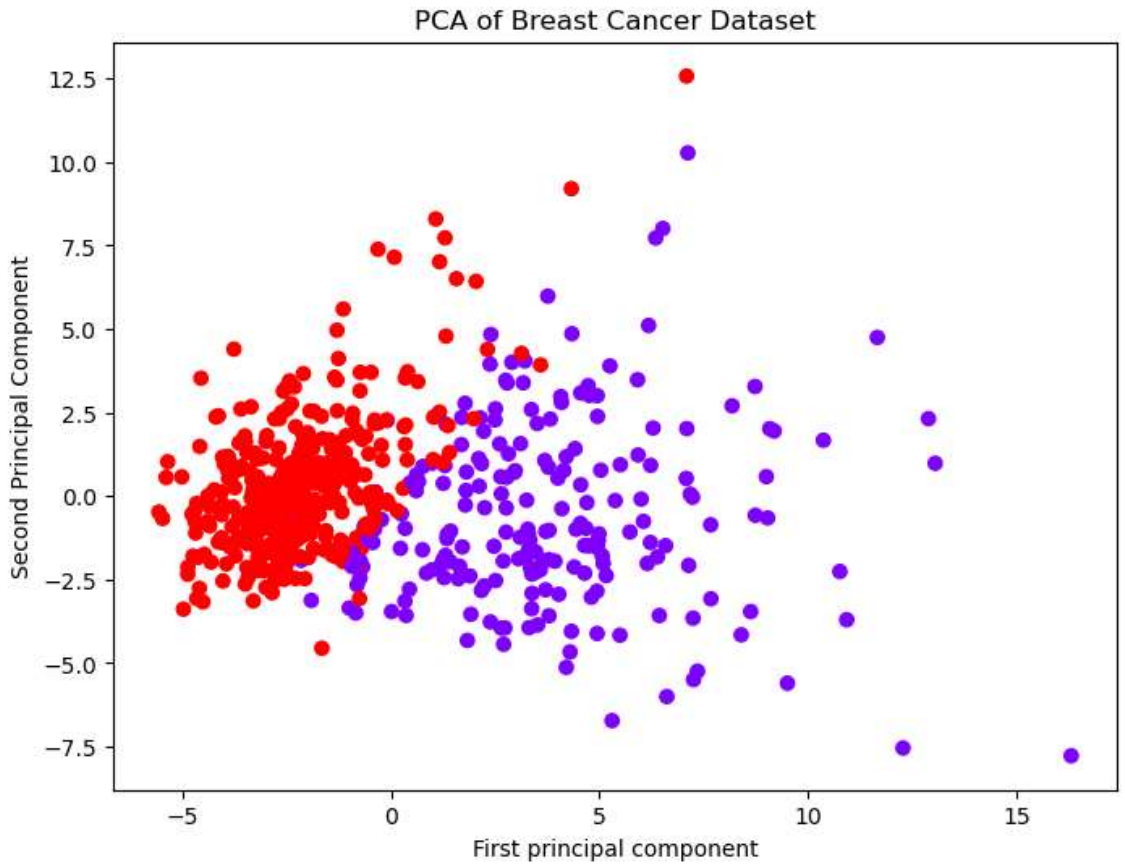
Out[8]:
```
(569, 30)
```

In [9]:
```
x_pca = pca.fit_transform(x)
x_pca.shape
```

Out[9]:
```
(569, 2)
```

In [10]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# pca_df = pd.DataFrame(X_pca,
#                       columns=['First Pricipal Component',
#                                'Second Principal Component'])

# pca_df['target'] = breast_cancer.target

# sns.scatterplot(x='First Pricipal Component',
#                 y='Second Principal Component',
#                 hue='target',
#                 data=pca_df,)

# plt.title('PCA of Breast Cancer Dataset')
# plt.show()

plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=breast_cancer['target'],cmap='rainbow')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
plt.title('PCA of Breast Cancer Dataset')
plt.show()
```

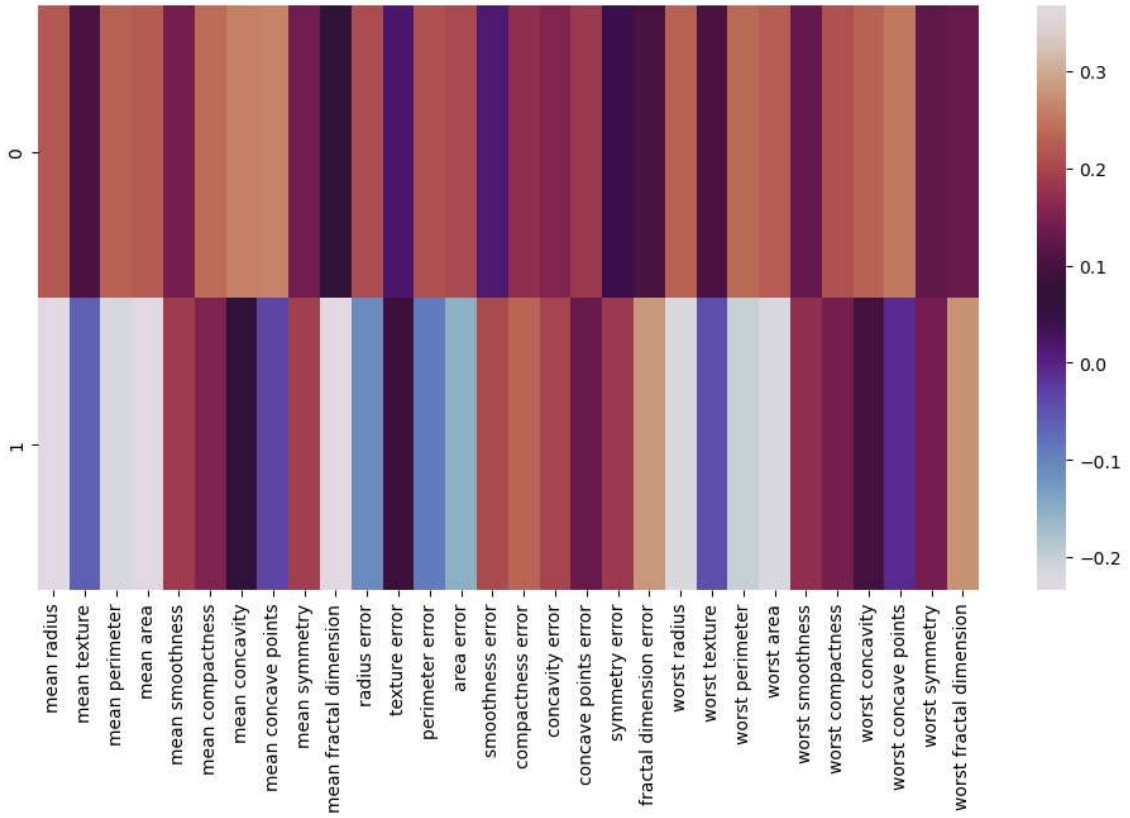


In [11]:
```
pca.components_
```

Out[11]:
```
array([[ 0.21890244,  0.10372458,  0.22753729,  0.22099499,  0.14258969,
         0.23928535,  0.25840048,  0.26085376,  0.13816696,  0.06436335,
         0.20597878,  0.01742803,  0.21132592,  0.20286964,  0.01453145,
         0.17039345,  0.15358979,  0.1834174 ,  0.04249842,  0.10256832,
         0.22799663,  0.10446933,  0.23663968,  0.22487053,  0.12795256,
         0.21009588,  0.22876753,  0.25088597,  0.12290456,  0.13178394],
       [-0.23385713, -0.05970609, -0.21518136, -0.23107671,  0.18611302,
         0.15189161,  0.06016536, -0.0347675 ,  0.19034877,  0.36657547,
        -0.10555215,  0.08997968, -0.08945723, -0.15229263,  0.20443045,
         0.2327159 ,  0.19720728,  0.13032156,  0.183848  ,  0.28009203,
        -0.21986638, -0.0454673 , -0.19987843, -0.21935186,  0.17230435,
         0.14359317,  0.09796411, -0.00825724,  0.14188335,  0.2753394
7]])
```

In [12]:
```
map_df = pd.DataFrame(pca.components_, columns=breast_cancer['feature_names
plt.figure(figsize=(12,6))
sns.heatmap(map_df, cmap='twilight')
```

Out[12]: <Axes: >



PCA Exercise

In [13]:
```
#1. Import the wine dataset and assign it to a variable. Split the data int
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
data = pd.read_csv(r'../data_samples2/Wine.csv')

# X = data.drop('Customer_Segment', axis=1)
# y = data['Customer_Segment']

X = data.iloc[:, 0:13].values
y = data.iloc[:, 13].values

data.head()
```

Out[13]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflava
0	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	

In [14]:
```
data['Customer_Segment'].values
```

Out[14]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3], dtype=int64)
```

In [15]:
```
#2. Split the dataset into the Training and the Test set. Set the test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,
```

In [16]:
```
#3. Scale the train and test set using the StandardScaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [17]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[17]:
```
array([0.36196226, 0.18763862])
```

In [18]:
```
#5. Create a Logistic Regression based on the training set
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
classifier.fit(X_train, y_train)
```

Out[18]:
```
LogisticRegression

LogisticRegression(random_state=0)
```

In [19]:
```
#6. Apply the created LR model onto the test data
# Predicting the test set result using
# predict function under LogisticRegression
y_pred = classifier.predict(X_test)
```

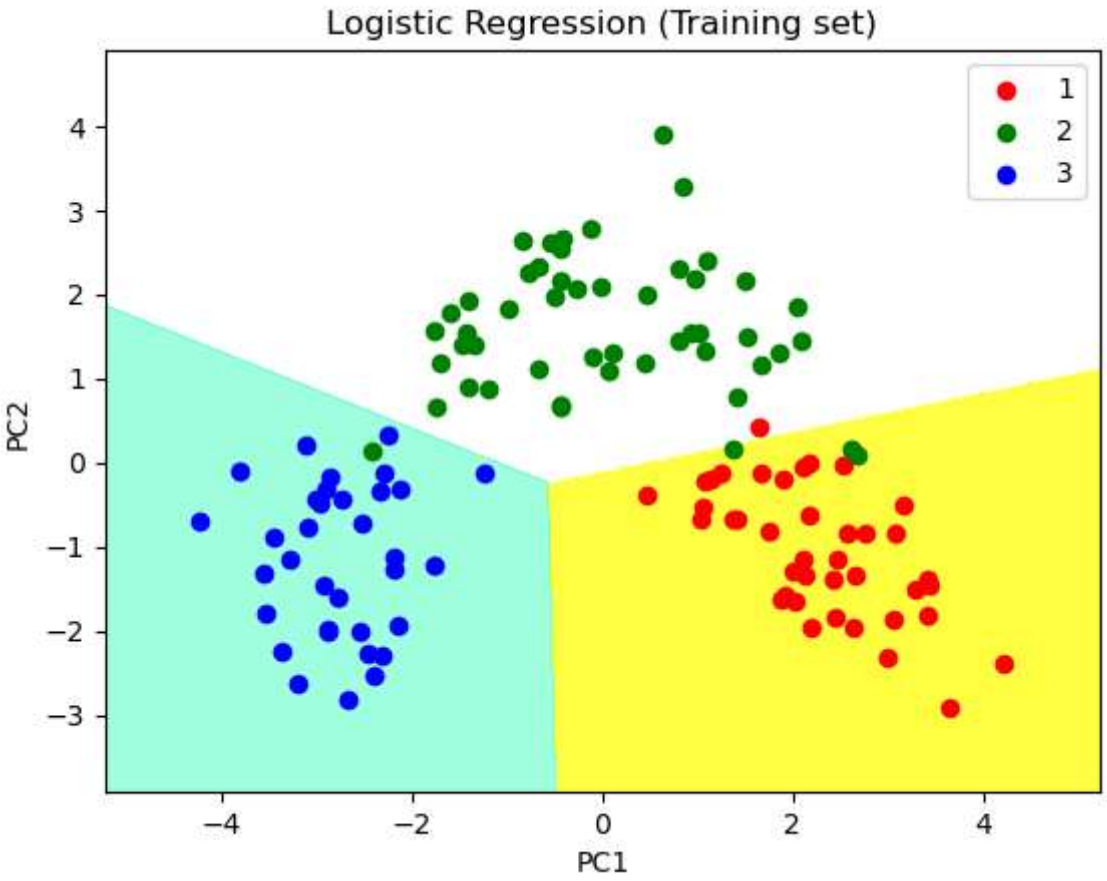
In [20]:
```
#7. Create a confusion matrix to score the prediction performed
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[20]:
```
array([[17,  2,  0],
       [ 0, 21,  0],
       [ 0,  0, 14]], dtype=int64)
```

In [21]:

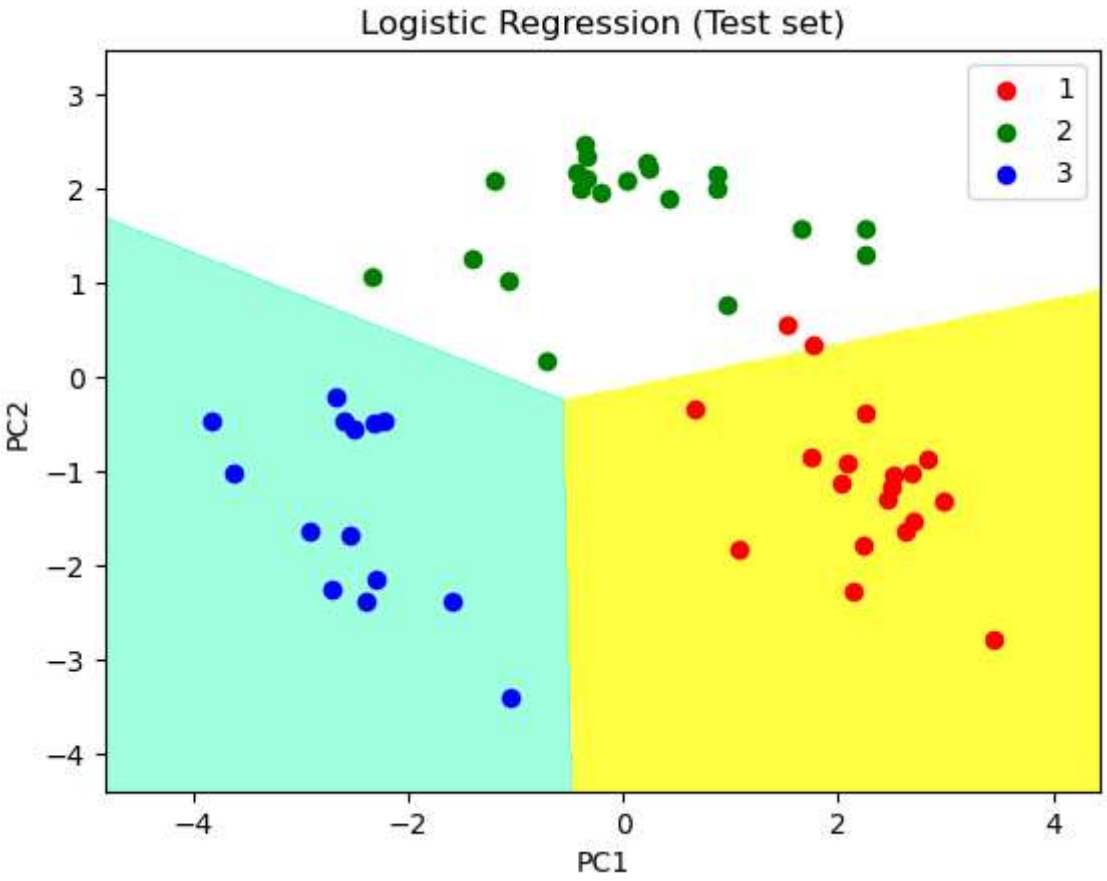
```
from matplotlib.colors import ListedColormap

X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                     stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
             X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
             cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

plt.title('Logistic Regression (Training set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend() # to show legend

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

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_9560\2623889199.py:17: UserWar
ning: *c* argument looks like a single numeric RGB or RGBA sequence, whic
h should be avoided as value-mapping will have precedence in case its len
gth matches with *x* & *y*.  Please use the *color* keyword-argument or p
rovide a 2D array with a single row if you intend to specify the same RGB
or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```


Logistic Regression (Training set)
1
2
3
PC2
PC1
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4

In [22]:
```
# Visualising the Test set results through scatter plot
from matplotlib.colors import ListedColormap

X_set, y_set = X_test, y_test

X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1,
                     stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1,
                     stop = X_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
             X2.ravel()]).T).reshape(X1.shape), alpha = 0.75,
             cmap = ListedColormap(('yellow', 'white', 'aquamarine')))

plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green', 'blue'))(i), label = j)

# title for scatter plot
plt.title('Logistic Regression (Test set)')
plt.xlabel('PC1') # for Xlabel
plt.ylabel('PC2') # for Ylabel
plt.legend()

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

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_9560\2404998179.py:19: UserWar
ning: *c* argument looks like a single numeric RGB or RGBA sequence, whic
h should be avoided as value-mapping will have precedence in case its len
gth matches with *x* & *y*.  Please use the *color* keyword-argument or p
rovide a 2D array with a single row if you intend to specify the same RGB
or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



Exercise 1. Take home test / challenge test

Repeat this exercise but this time do it without dimension reduction. What do you get?

In [24]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv(r'../data_samples2/Wine.csv')

X = data.iloc[:, 0:13].values
y = data.iloc[:, 13].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[24]: 0.9444444444444444

Random Forest Feature Selection

Let's do a guided walkthrough for a RF feature selection)
(https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_for
(https://chrisalbon.com/machine_learning/trees_and_forests/feature_selection_using_random_for

In [25]:
```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
```

In [26]:
```
# Load the iris dataset
iris = datasets.load_iris()

# Create a list of feature names
feat_labels = ['Sepal Length','Sepal Width','Petal Length','Petal Width']

# Create X from the features
X = iris.data

# Create y from output
y = iris.target
```

In [27]:
```
# View the features
X[:5]
```

Out[27]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [28]:
```
# View the target data
y
```

Out[28]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [ ]:
```
```

In [41]:

```
Out[41]: array([[1.371e+01, 1.860e+00, 2.360e+00, ..., 1.110e+00, 4.000e+00,
                 1.035e+03],
                [1.222e+01, 1.290e+00, 1.940e+00, ..., 8.600e-01, 3.020e+00,
                 3.120e+02],
                [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
                 8.350e+02],
                ...,
                [1.242e+01, 1.610e+00, 2.190e+00, ..., 1.060e+00, 2.960e+00,
                 3.450e+02],
                [1.390e+01, 1.680e+00, 2.120e+00, ..., 9.100e-01, 3.330e+00,
                 9.850e+02],
                [1.416e+01, 2.510e+00, 2.480e+00, ..., 6.200e-01, 1.710e+00,
                 6.600e+02]])
```

In [31]:

```
# Split the data into 40% test and 60% training
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,
```

In [32]:

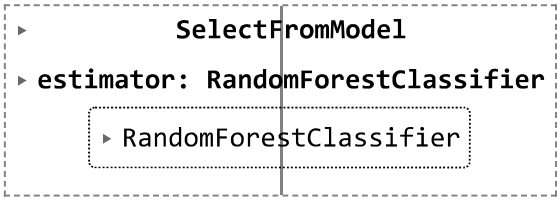
```
# Create a random forest classifier
classifier = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_we
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False)

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

# Print the name and gini importance of each feature
for feature in zip(feat_labels, classifier.feature_importances_):
    print(feature)
```

```
('Sepal Length', 0.11024282328064558)
('Sepal Width', 0.016255033655398383)
('Petal Length', 0.45028123999239505)
('Petal Width', 0.42322090307156096)
```

In [35]:
```
# Create a selector object that will use the random forest classifier to id
# features that have an importance of more than 0.15
feature = SelectFromModel(RandomForestClassifier(bootstrap=True, ccp_alpha=
                                                 max_depth=None, max_featur
                                                 min_samples_leaf=1, min_sa
                                                 n_estimators=10000, n_jobs
                                                 warm_start=False),
                          max_features=None, norm_order=1, prefit=False, th

# Train the selector
feature.fit(X_train, y_train)
```

```
C:\Users\ACER\anaconda3\envs\python-dscourse\lib\site-packages\sklearn\en
semble\_forest.py:424: FutureWarning: `max_features='auto'` has been depr
ecated in 1.1 and will be removed in 1.3. To keep the past behaviour, exp
licitly set `max_features='sqrt'` or remove this parameter as it is also
the default value for RandomForestClassifiers and ExtraTreesClassifiers.
  warn(
```

Out[35]:



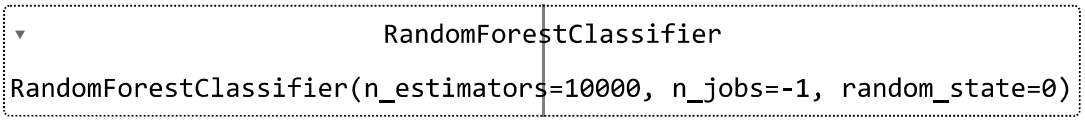
In [36]:
```
# Print the names of the most important features
for index in feature.get_support(indices=True):
    print(feat_labels[index])
```

```
Petal Length
Petal Width
```

In [37]:
```
# Transform the data to create a new dataset containing only the most impor
# Note: We have to apply the transform to both the training X and test X da
X_train_transformed = feature.transform(X_train)
X_test_transformed = feature.transform(X_test)
```

In [38]:
```
# Create a new random forest classifier for the most important features
classifier_important = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0
                       criterion='gini', max_depth=None,
                       max_leaf_nodes=None, max_samples=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10000,
                       n_jobs=-1, oob_score=False, random_state=0, verbose=
                       warm_start=False)

# Train the new classifier on the new dataset containing the most important
classifier_important.fit(X_train_transformed, y_train)
```

Out[38]:



In [39]:
```
# Apply The Full Featured Classifier To The Test Data
y_pred = classifier.predict(X_test)

# View The Accuracy Of Our Full Feature (4 Features) Model
accuracy_score(y_test, y_pred)
```

Out[39]: 0.9333333333333333

In [40]:
```
# Apply The Full Featured Classifier To The Test Data
y_pred = classifier_important.predict(X_test_transformed)

# View The Accuracy Of Our Limited Feature (2 Features) Model
accuracy_score(y_test, y_pred)
```

Out[40]: 0.9

Exercise 2. Now repeat RF feature selection but now use the wine dataset

In [41]:
```
df = pd.read_csv('../data_samples2/Wine.csv')
X = df.drop('Customer_Segment', axis=1)
y = df['Customer_Segment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,
```

In [42]:
```
# View The Accuracy Of Our Full Feature Model

classifier = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_we
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[42]: 0.9722222222222222

In [43]:
```
# View The Accuracy Of Our Limited Feature Model

feature = SelectFromModel(RandomForestClassifier(bootstrap=True, ccp_alpha=
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False), max_features=None, norm_order=1,

feature.fit(X_train, y_train)

X_train_transformed = feature.transform(X_train)
X_test_transformed = feature.transform(X_test)

classifier_important = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False)

classifier_important.fit(X_train_transformed, y_train)

y_pred = classifier_important.predict(X_test_transformed)

accuracy_score(y_test, y_pred)
```

Out[43]: 0.9722222222222222

RFE Walkthrough

In [45]:
```
# Automatically select the features
# automatically select the number of features for RFE
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFECV
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,

# create pipeline
rfe = RFECV(estimator=DecisionTreeClassifier())
model = DecisionTreeClassifier()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])

# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jo

# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.892 (0.027)
```

In [46]:
```
# report which features were selected by RFE
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier

# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5,

# define RFE
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)

# fit RFE
rfe.fit(X, y)

# summarize all features
for i in range(X.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i], rfe.
```

```
Column: 0, Selected False, Rank: 3.000
Column: 1, Selected False, Rank: 5.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected True, Rank: 1.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected False, Rank: 6.000
Column: 6, Selected True, Rank: 1.000
Column: 7, Selected False, Rank: 4.000
Column: 8, Selected True, Rank: 1.000
Column: 9, Selected False, Rank: 2.000
```

Exercise 3. Now repeat RF feature selection but now use the iris dataset

In [47]:
```
#Starter code for Exercise 3
# Load libraries
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
from sklearn.metrics import accuracy_score
from sklearn import datasets

# Load iris dataset
iris = datasets.load_iris()

# Create feature matrix
X = iris.data

# Create target vector
y = iris.target

# View the first observation's feature values
X[0]
```

Out[47]:
```
array([5.1, 3.5, 1.4, 0.2])
```

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.4,
```

In [49]:
```
# View The Accuracy Of Our Full Feature Model

classifier = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_we
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False)

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

accuracy_score(y_test, y_pred)
```

Out[49]:
```
0.9333333333333333
```

In [51]:
```
# View The Accuracy Of Our Limited Feature Model

feature = SelectFromModel(RandomForestClassifier(bootstrap=True, ccp_alpha=
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False), max_features=None, norm_order=1,

feature.fit(X_train, y_train)

X_train_transformed = feature.transform(X_train)
X_test_transformed = feature.transform(X_test)

classifier_important = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0
                        criterion='gini', max_depth=None,
                        max_leaf_nodes=None, max_samples=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10000,
                        n_jobs=-1, oob_score=False, random_state=0, verbose=
                        warm_start=False)

classifier_important.fit(X_train_transformed, y_train)

y_pred = classifier_important.predict(X_test_transformed)

accuracy_score(y_test, y_pred)
```

Out[51]: 0.9

t-SNE Walkthrough (https://cmdlinetips.com/2019/07/dimensionality-reduction-with-tsne/ (https://cmdlinetips.com/2019/07/dimensionality-reduction-with-tsne/))

In [52]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import pandas as pd
```

In [53]:
```
from sklearn.datasets import load_digits
digits = load_digits()
```

In [54]:
```
digits.data.shape
```

Out[54]: (1797, 64)

In [55]:
```
X = digits.data[:600]
y = digits.target[:600]
```

In [58]:
```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
```

In [59]:
```
tsne_obj= tsne.fit_transform(X)
```

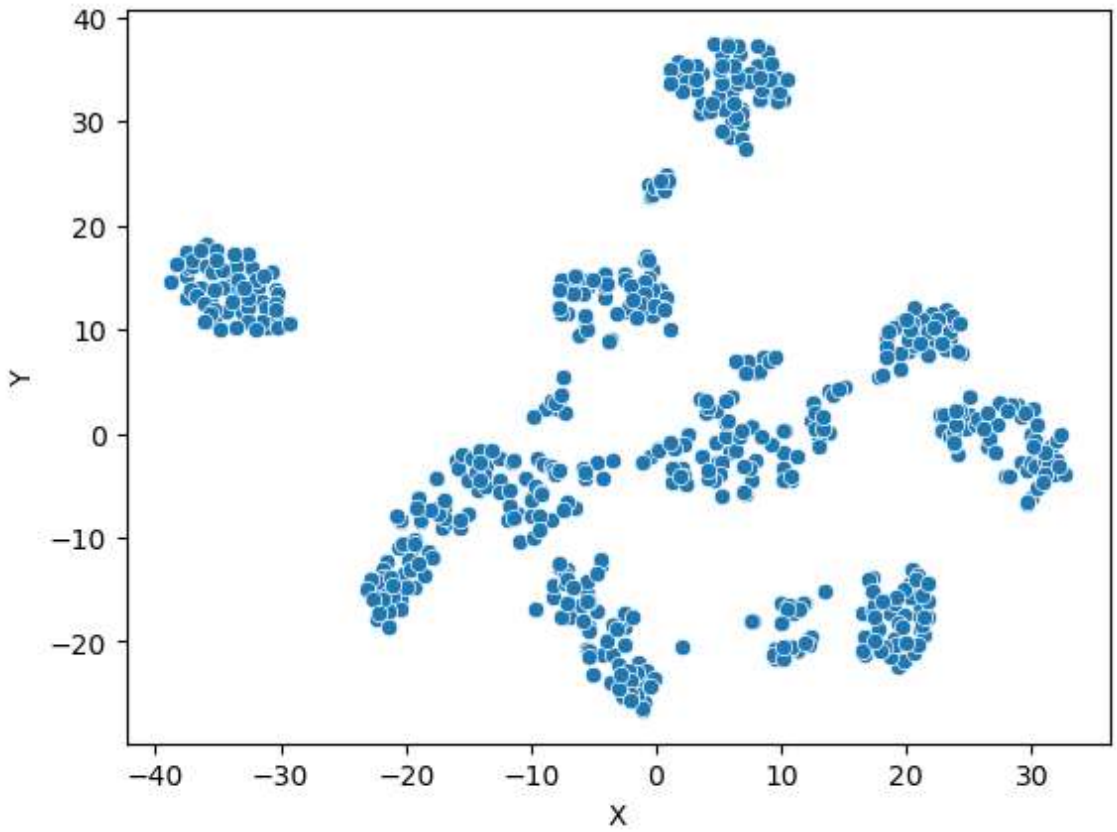
In [60]:
```
tsne_df = pd.DataFrame({'X':tsne_obj[:,0], 'Y':tsne_obj[:,1], 'digit':y})
tsne_df.head()
```

Out[60]:

	X	Y	digit
0	-33.064045	11.775319	0
1	17.766239	5.479626	1
2	8.055402	5.875648	2
3	-18.861454	-8.514892	3
4	30.009453	2.323586	4

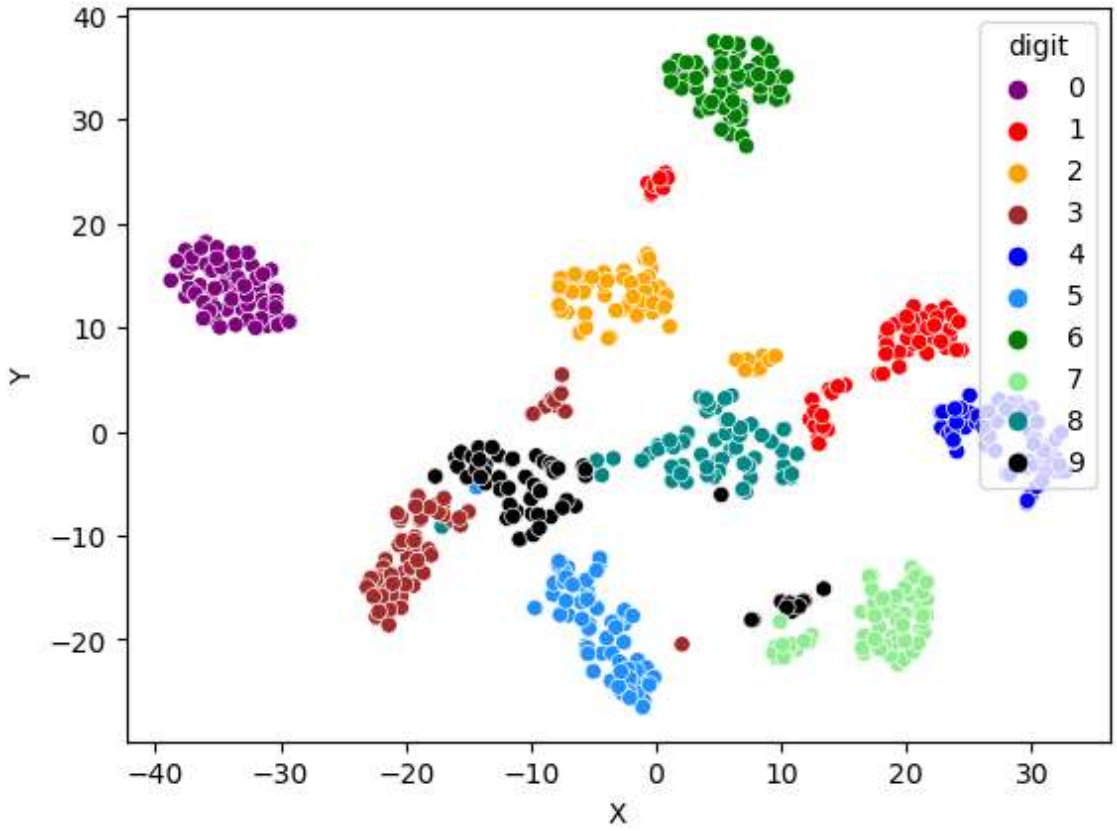
In [61]:
```
sns.scatterplot(x="X", y="Y", data=tsne_df)
```

Out[61]: <Axes: xlabel='X', ylabel='Y'>



In [62]:
```
sns.scatterplot(x="X", y="Y", hue="digit",
                palette=['purple','red','orange','brown','blue',
                         'dodgerblue','green','lightgreen','darkcyan', 'black
                legend='full', data=tsne_df)
```

Out[62]: <Axes: xlabel='X', ylabel='Y'>



Exercise 4. Now repeat t-SNE feature selection but now use the iris dataset

In [63]:
```
from sklearn.datasets import load_iris
from sklearn.manifold import TSNE

iris = load_iris()
X = iris.data
y = iris.target

tsne = TSNE(n_components=2, random_state=0)
tsne_obj= tsne.fit_transform(X)

tsne_df = pd.DataFrame({'X':tsne_obj[:,0], 'Y':tsne_obj[:,1], 'iris_classes
tsne_df.head()
```

Out[63]:

	X	Y	iris_classes
0	-23.875437	-1.430589	0
1	-21.332832	-2.522965	0
2	-21.183956	-1.316485	0
3	-20.842545	-1.611920	0
4	-23.845785	-0.945733	0

In [64]:
```
sns.scatterplot(x="X", y="Y", hue="iris_classes", legend='full', data=tsne_
plt.legend(title="Classes", labels=["Setosa", "Versicolor", "Virginica"])
plt.show()
```