```
# %%

# ================================================================================
# PRINCIPAL COMPONENT ANALYSIS
# ================================================================================


# %%


# importing the scikit-learn.dataset library

import pandas as pd

from sklearn import datasets

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV


from sklearn.preprocessing import StandardScaler


from sklearn.model_selection import train_test_split


# %%

# generating the dataset for 1000 samples with linear relation from make_regression() fucntion of skearn

x, y = datasets.make_classification(n_samples=1000,  # number of samples
                                    n_features=4)  # number of features


# %%

# visualizing each feature
```

```
fig, axs = plt.subplots(2, 2)
axs[0, 0].scatter(x[:, 0], y)
axs[0, 0].set_title('feature 1')
axs[0, 1].scatter(x[:, 1], y)
axs[0, 1].set_title('feature 2')
axs[1, 0].scatter(x[:, 2], y)
axs[1, 0].set_title('feature 3')
axs[1, 1].scatter(x[:, 3], y)
axs[1, 1].set_title('feature 4')

# Hide x labels and tick labels for top plots and y ticks for right plots.
for ax in axs.flat:
    ax.label_outer()


# %%

# creating training and test datasets with train_test_split func. of sklearn.model_selection

train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2)

print("train data size:", len(train_x))  # 800

print("test data size:", len(test_x))  # 200


# %%

# Logistic Regression


# fitting the the logistic regression model

LR = LogisticRegression()

LR.fit(train_x, train_y)


# model prediction

LR_predict = LR.predict(test_x)


# model accuracy


print("Linear Regression Model accuracy: ",
      accuracy_score(test_y, LR_predict)*100)
```

```
# %%

# Decision Tree


# fitting the the decision tree model

decision_tree = DecisionTreeClassifier()

decision_tree.fit(train_x, train_y)


# model prediction

decision_predict = decision_tree.predict(test_x)


# model accuracy


print("Decision Tree Model accuracy: ",
      accuracy_score(test_y, decision_predict)*100)
# %%

# Support Vector Machines (SVM)


# fitting the SVM model

svm = SVC()

svm.fit(train_x, train_y)


# model prediction

svm_predict = svm.predict(test_x)


# model accuracy


print("SVM Model accuracy: ",
      accuracy_score(test_y, svm_predict)*100)
```

```
# %%

# Random Forest Classifier


# fitting the Random Forest model

rf = RandomForestClassifier(n_estimators=800, max_depth=5)

rf.fit(train_x, train_y)


# model prediction

rf_predict = rf.predict(test_x)


# model accuracy


print("Random Forest Model accuracy: ",
      accuracy_score(test_y, rf_predict)*100)
# %%

# K-Nearest Neighbours (KNN)


# fitting the KNN model

knn = KNeighborsClassifier(n_neighbors=8)

knn.fit(train_x, train_y)


# model prediction

knn_predict = knn.predict(test_x)


# model accuracy


print("KNN Model accuracy: ",
      accuracy_score(test_y, knn_predict)*100)


# %%
```

```
# scaling our data

scaler = StandardScaler()
X_train, X_test, y_train, y_test = train_test_split(
    x, y, test_size=0.3, random_state=42)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# %%

# Principal Component Analysis (PCA)


pca = PCA(n_components=3)
scaler = StandardScaler()

X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


# %%

# plotting the data


plt.figure(figsize=(8, 6))
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='plasma')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')


# %%

# creating a function to find the moidel accuracy


def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(
            y_train, pred, output_dict=True))
        print("Train Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
```

```
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train == False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(
            classification_report(y_test, pred, output_dict=True))
        print("Test Result:\n================================================")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print("_____________________________________________")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print("_____________________________________________")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
        cm = confusion_matrix(y_test, pred)
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.imshow(cm)
        ax.grid(False)
        ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
        ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
        ax.set_ylim(1.5, -0.5)
        for i in range(2):
            for j in range(2):
                ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
                plt.show()


# %%

param_grid = {'C': [0.01, 0.1, 0.5, 1, 10, 100],
              'gamma': [1, 0.75, 0.5, 0.25, 0.1, 0.01, 0.001],
              'kernel': ['rbf', 'poly', 'linear']}

grid = GridSearchCV(SVC(), param_grid, verbose=1, cv=5)
grid.fit(X_train, y_train)
best_params = grid.best_params_
print(f"Best params: {best_params}")

svm_clf = SVC(**best_params)
svm_clf.fit(X_train, y_train)

print_score(svm_clf, X_train, y_train, X_test, y_test, train=True)
print_score(svm_clf, X_train, y_train, X_test, y_test, train=False)


# %%
```

==========
THE END
==========