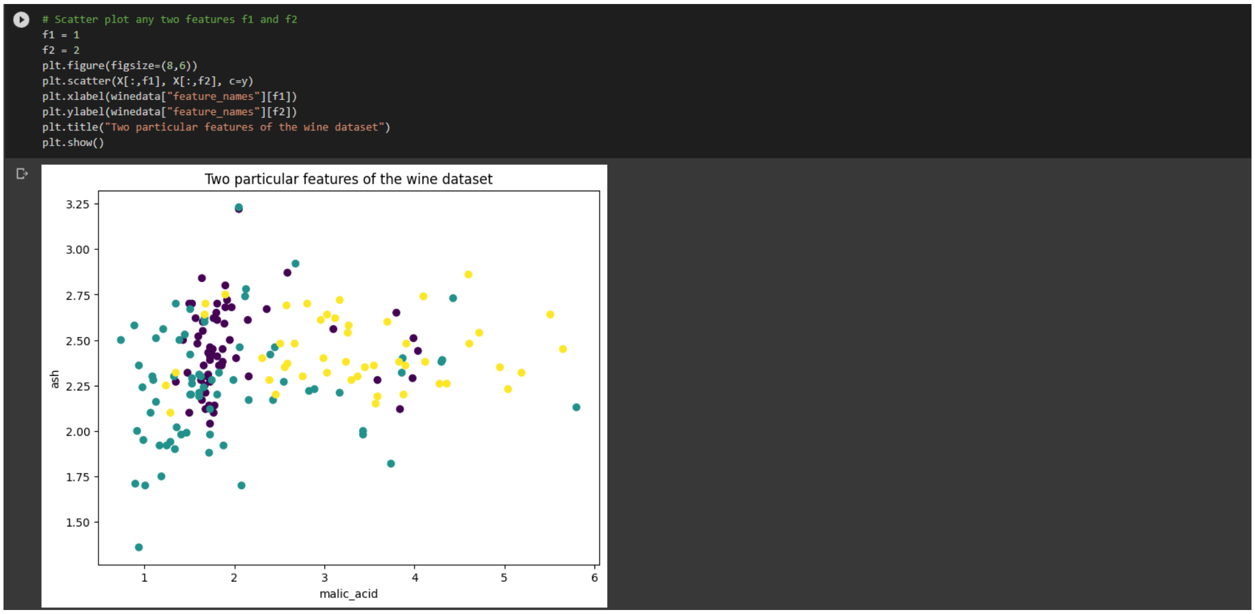
Kunnithi Treecharoensomboon ID 63070503461

**CPE383 Machine Learning: Quiz8**

Use the “PCA Examples.ipynb” for the following:

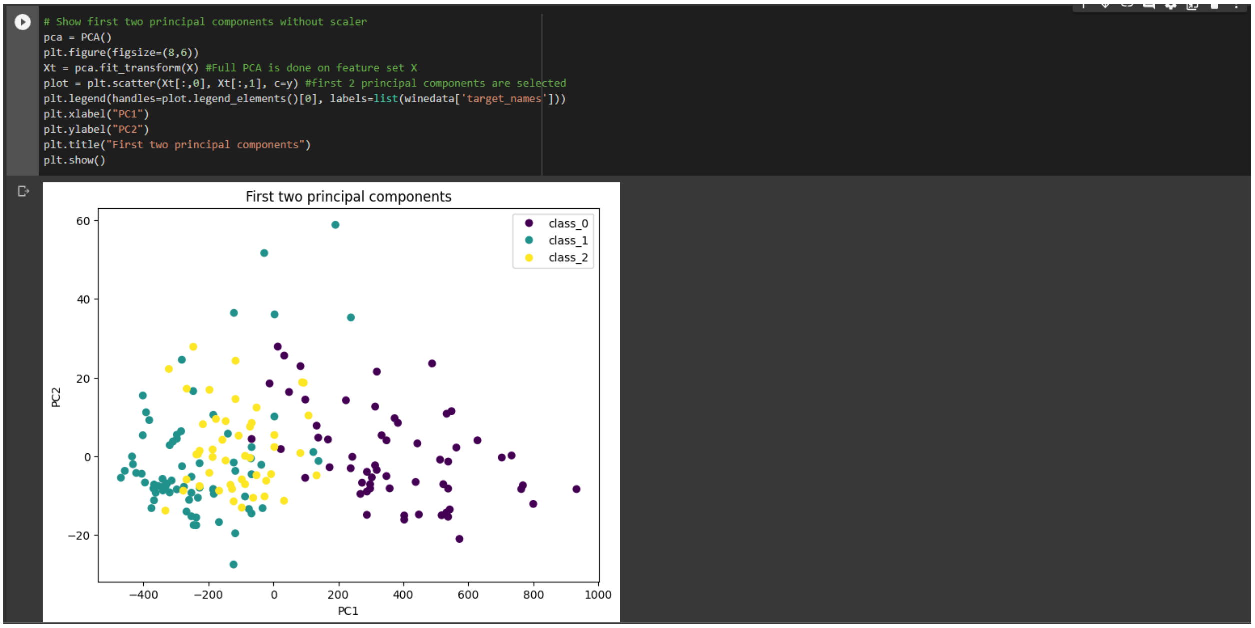
1. 15 points. 1 hour. Using the Wine dataset, use PCA to reduce it to 2 PCA features.

1.1. Visualize the first 2 columns as a scatter plot using all (100%) of the data.



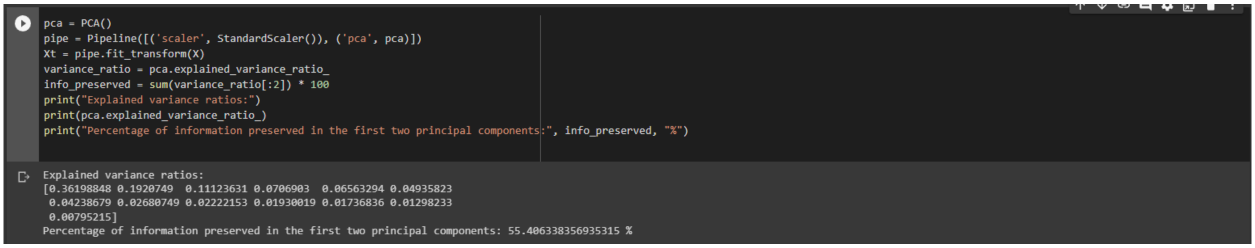
```
# Scatter plot any two features f1 and f2
f1 = 1
f2 = 2
plt.figure(figsize=(8,6))
plt.scatter(X[:,f1], X[:,f2], c=y)
plt.xlabel(winedata["feature_names"][f1])
plt.ylabel(winedata["feature_names"][f2])
plt.title("Two particular features of the wine dataset")
plt.show()
```

1.2. In another graph visualize the scatter plot for the 2 PCA features also using all the data.



```
# Show first two principal components without scaler
pca = PCA()
plt.figure(figsize=(8,6))
Xt = pca.fit_transform(X) #Full PCA is done on feature set X
plot = plt.scatter(Xt[:,0], Xt[:,1], c=y) #first 2 principal components are selected
plt.legend(handles=plot.legend_elements()[0], labels=list(winedata['target_names']))
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("First two principal components")
plt.show()
```

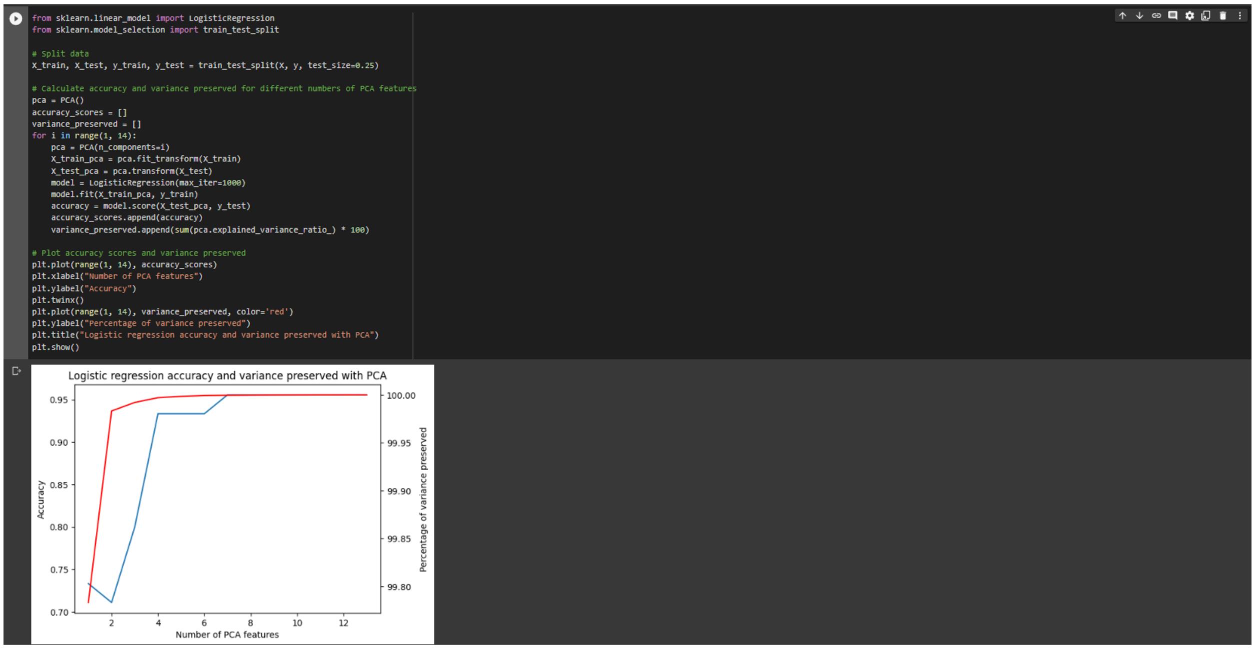
1.3. What % of information (“explained variance”) is preserved in the 2 PCA features?



```
pca = PCA()
pipe = Pipeline([('scaler', StandardScaler()), ('pca', pca)])
Xt = pipe.fit_transform(X)
variance_ratio = pca.explained_variance_ratio_
info_preserved = sum(variance_ratio[:2]) * 100
print("Explained variance ratios:")
print(pca.explained_variance_ratio_)
print("Percentage of information preserved in the first two principal components:", info_preserved, "%")
```

```
Explained variance ratios:
[0.36198848 0.1920749  0.11123631 0.0706903  0.06563294 0.04935823
 0.04238679 0.02680749 0.02222153 0.01930019 0.01736836 0.01298233
 0.00795215]
Percentage of information preserved in the first two principal components: 55.40633835693531 5 %
```

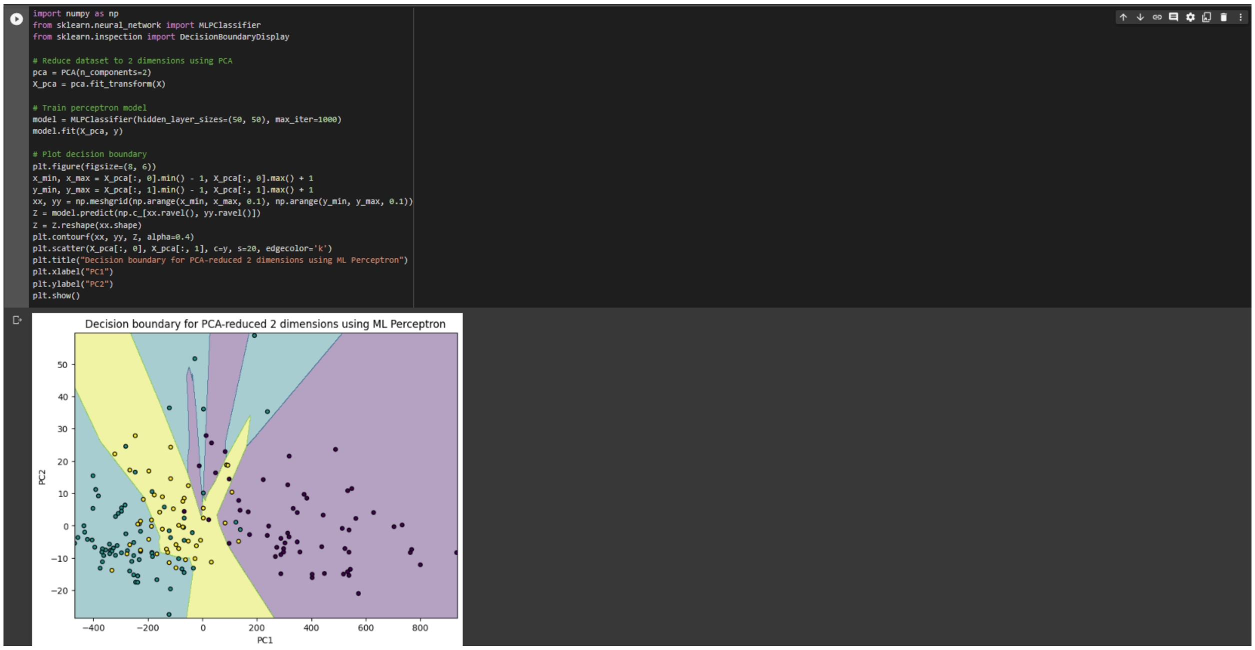
2. 10 points. 1 hour. Use 25% for testing for the Wine dataset. Plot a graph of the accuracy of the samples when using 1, 2, 3, ..., 13 PCA features along with showing how much % variance is preserved (p%) for each? Use logistic regression.



```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

# Calculate accuracy and variance preserved for different numbers of PCA features
pca = PCA()
accuracy_scores = []
variance_preserved = []
for i in range(1, 14):
    pca = PCA(n_components=i)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_pca, y_train)
    accuracy = model.score(X_test_pca, y_test)
    accuracy_scores.append(accuracy)
    variance_preserved.append(sum(pca.explained_variance_ratio_) * 100)

# Plot accuracy scores and variance preserved
plt.plot(range(1, 14), accuracy_scores)
plt.xlabel("Number of PCA features")
plt.ylabel("Accuracy")
plt.twinx()
plt.plot(range(1, 14), variance_preserved, color='red')
plt.ylabel("Percentage of variance preserved")
plt.title("Logistic regression accuracy and variance preserved with PCA")
plt.show()
```

Logistic regression accuracy and variance preserved with PCA
Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
Percentage of variance preserved
100.00
99.95
99.90
99.85
99.80
2
4
6
8
10
12
Number of PCA features

3. 15 points. 2 hours. Decision boundary. Plot the decision boundary for PCA-reduced 2 dimensions of the wine dataset. Use ML Perceptron in https://python-course.eu/machine-learning/neural-networks-with-scikit.php with 2 hidden layers.

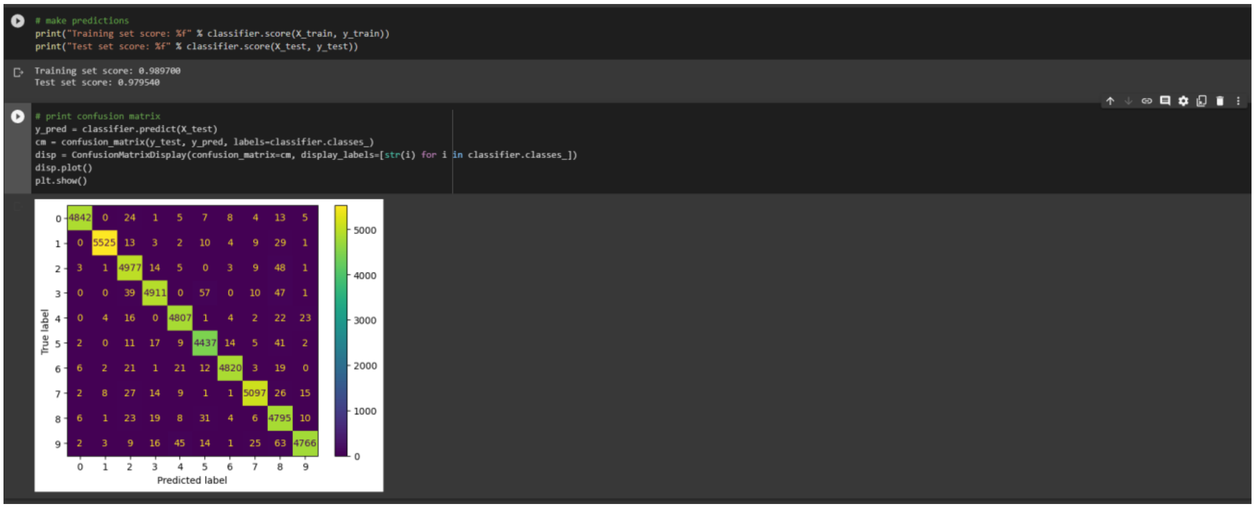


```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.inspection import DecisionBoundaryDisplay

# Reduce dataset to 2 dimensions using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Train perceptron model
model = MLPClassifier(hidden_layer_sizes=(50, 50), max_iter=1000)
model.fit(X_pca, y)

# Plot decision boundary
plt.figure(figsize=(8, 6))
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, s=20, edgecolor='k')
plt.title("Decision boundary for PCA-reduced 2 dimensions using ML Perceptron")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

Decision boundary for PCA-reduced 2 dimensions using ML Perceptron
PC2
50
40
30
20
10
0
−10
−20
−400
−200
0
200
400
600
800
PC1

4. Using the MNIST data of 784 dimensions (28 x 28 pixels).

4.1. 5 points. 0.5 hours. Show Quiz 7 problem #3 here again as a benchmark. Using 50,000 data samples for training and 20,000 for testing, apply standard scalar transform from the training dataset to both your training and testing datasets. Report the accuracy and print out the confusion matrix.



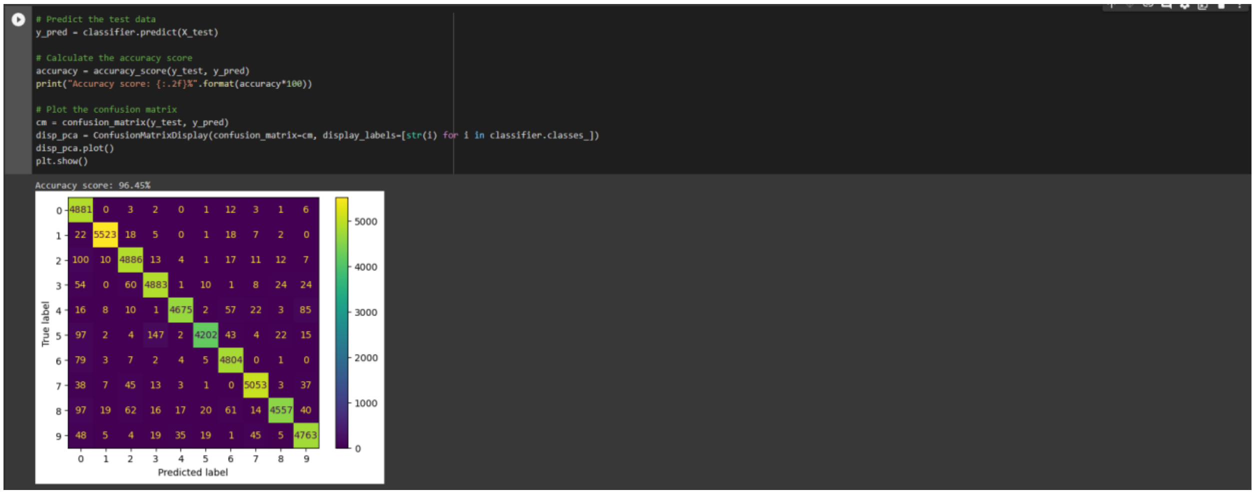
```
# make predictions
print("Training set score: %f" % classifier.score(X_train, y_train))
print("Test set score: %f" % classifier.score(X_test, y_test))
```

```
Training set score: 0.989700
Test set score: 0.979540
```

```
# print confusion matrix
y_pred = classifier.predict(X_test)
cm - confusion_matrix(y_test, y_pred, labels-classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[str(i) for i in classifier.classes_])
disp.plot()
plt.show()
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4842 | 0 | 24 | 1 | 5 | 7 | 8 | 4 | 13 | 5 |
| 1 | 0 | 5525 | 13 | 3 | 2 | 10 | 4 | 9 | 29 | 1 |
| 2 | 3 | 1 | 4977 | 14 | 5 | 0 | 3 | 9 | 48 | 1 |
| 3 | 0 | 0 | 39 | 4911 | 0 | 57 | 0 | 10 | 47 | 1 |
| 4 | 0 | 4 | 16 | 0 | 4807 | 1 | 4 | 2 | 22 | 23 |
| 5 | 2 | 0 | 11 | 17 | 9 | 4437 | 14 | 5 | 41 | 2 |
| 6 | 6 | 2 | 21 | 1 | 21 | 12 | 4820 | 3 | 19 | 0 |
| 7 | 2 | 8 | 27 | 14 | 9 | 1 | 1 | 5097 | 26 | 15 |
| 8 | 6 | 1 | 23 | 19 | 8 | 31 | 4 | 6 | 4795 | 10 |
| 9 | 2 | 3 | 9 | 16 | 45 | 14 | 1 | 25 | 63 | 4766 |

4.2. 20 points. 2 hours. Using the same neural network model as in 4.1 above, but using the PCA features that preserve up to 95% of the information content, find a new training and testing dataset using PCA transformation based on the training dataset. Apply standard scalar transform from the training dataset to both your training and testing datasets. Report the accuracy and print out the confusion matrix for PCA with reduced features.



```
# Predict the test data
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy score: {:.2f}%".format(accuracy*100))

# Plot the confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp_pca = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[str(i) for i in classifier.classes_])
disp_pca.plot()
plt.show()
```

```
Accuracy score: 96.45%
```

| True label \ Predicted label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 4881 | 0 | 3 | 2 | 0 | 1 | 12 | 3 | 1 | 6 |
| 1 | 22 | 5523 | 18 | 5 | 0 | 1 | 18 | 7 | 2 | 0 |
| 2 | 100 | 10 | 4886 | 13 | 4 | 1 | 17 | 11 | 12 | 7 |
| 3 | 54 | 0 | 60 | 4883 | 1 | 10 | 1 | 8 | 24 | 24 |
| 4 | 16 | 8 | 10 | 1 | 4675 | 2 | 57 | 22 | 3 | 85 |
| 5 | 97 | 2 | 4 | 147 | 2 | 4202 | 43 | 4 | 22 | 15 |
| 6 | 79 | 3 | 7 | 2 | 4 | 5 | 4804 | 0 | 1 | 0 |
| 7 | 38 | 7 | 45 | 13 | 3 | 1 | 0 | 5053 | 3 | 37 |
| 8 | 97 | 19 | 62 | 16 | 17 | 20 | 61 | 14 | 4557 | 40 |
| 9 | 48 | 5 | 4 | 19 | 35 | 19 | 1 | 45 | 5 | 4763 |