PCA Lab-03 Report

Index Number: - 190604N

Task 1

The data in iris.data file was first taken into a Pandas data frame. The 4 predictor columns in the dataset had to be preprocessed in this task. There are different ways available to standardize the data. Three of these methods, given below, were examined.

1. Subtracting mean and dividing by standard deviation using the NumPy library
2. Standardizing using StandardScaler of sklearn.preprocessing
3. Using zscore of Scipy Stats library

After standardizing the columns should have zero for mean and one for variance. This property was used to compare the results of the 3 methods stated above.

		Method 1	Method 2	Method 3
0	Sepal Length Mean	-1.457168e-15	-2.775558e-16	-2.775558e-16
1	Sepal Width Mean	-1.722511e-15	-5.140333e-16	-5.140333e-16
2	Petal Length Mean	-2.043551e-15	1.154632e-16	1.154632e-16
3	Petal Width Mean	-9.843977e-17	9.251859e-16	9.251859e-16

		Method 1	Method 2	Method 3
0	Sepal Length Variance	1.0	1.006711	1.006711
1	Sepal Width Variance	1.0	1.006711	1.006711
2	Petal Length Variance	1.0	1.006711	1.006711
3	Petal Width Variance	1.0	1.006711	1.006711

Mean of each column obtained via all 3 methods is very close to 0. However, variance of each column is closer to 1 when using method 1 rather than the other 2 methods. Thus, the columns were standardized using method 1, as shown below.

```
for col in df.columns:
  df[col] = (df[col] - df[col].mean()) / df[col].std()

df.head()
```

	sepal length	sepal width	petal length	petal width
0	-0.897674	1.028611	-1.336794	-1.308593
1	-1.139200	-0.124540	-1.336794	-1.308593
2	-1.380727	0.336720	-1.393470	-1.308593
3	-1.501490	0.106090	-1.280118	-1.308593
4	-1.018437	1.259242	-1.336794	-1.308593

Task 2

Next the covariance matrix was computed using the cov() function of NumPy library. The covariance matrix is as follows.

```
covmatrix = df.cov()
covmatrix
```

	sepal length	sepal width	petal length	petal width
sepal length	1.000000	-0.109369	0.871754	0.817954
sepal width	-0.109369	1.000000	-0.420516	-0.356544
petal length	0.871754	-0.420516	1.000000	0.962757
petal width	0.817954	-0.356544	0.962757	1.000000

Task 3

The covariance matrix obtained above was factorized into 3 matrices, U, S and V, using Singular Value Decomposition. This was performed using the svd() function of NumPy (numpy.linalg) library. The 3 factor matrices are shown below.

```
U, S, V = svd(covmatrix)

U: -

[[-0.52237162 -0.37231836  0.72101681  0.26199559]
 [ 0.26335492 -0.92555649 -0.24203288 -0.12413481]
 [-0.58125401 -0.02109478 -0.14089226 -0.80115427]
 [-0.56561105 -0.06541577 -0.6338014   0.52354627]]

S: -

[2.91081808 0.92122093 0.14735328 0.02060771]

V: -

[[-0.52237162  0.26335492 -0.58125401 -0.56561105]
 [-0.37231836 -0.92555649 -0.02109478 -0.06541577]
 [ 0.72101681 -0.24203288 -0.14089226 -0.6338014 ]
 [ 0.26199559 -0.12413481 -0.80115427  0.52354627]]
```

Since covariance matrix is a symmetric positive definite matrix, one of its factors, U (which is also equal to transpose of V) contains eigen vectors. That is, the columns of U are the 4 eigen vectors of the covariance matrix.

Task 4

It was required to project the data into first 2 principal components. To perform this, only the first 2 columns of matrix U, as given below, are required. That is, only the first 2 eigen vectors should be used.

```
array([[-0.52237162, -0.37231836],
       [ 0.26335492, -0.92555649],
       [-0.58125401, -0.02109478],
       [-0.56561105, -0.06541577]])
```

These 2 eigen vectors make up the Feature Vector. Next, for dimensionality reduction, the standardized data is recast along the axes of the 2 chosen principal components. For this, the transpose of the feature vector is multiplied by the transpose of the original dataset.

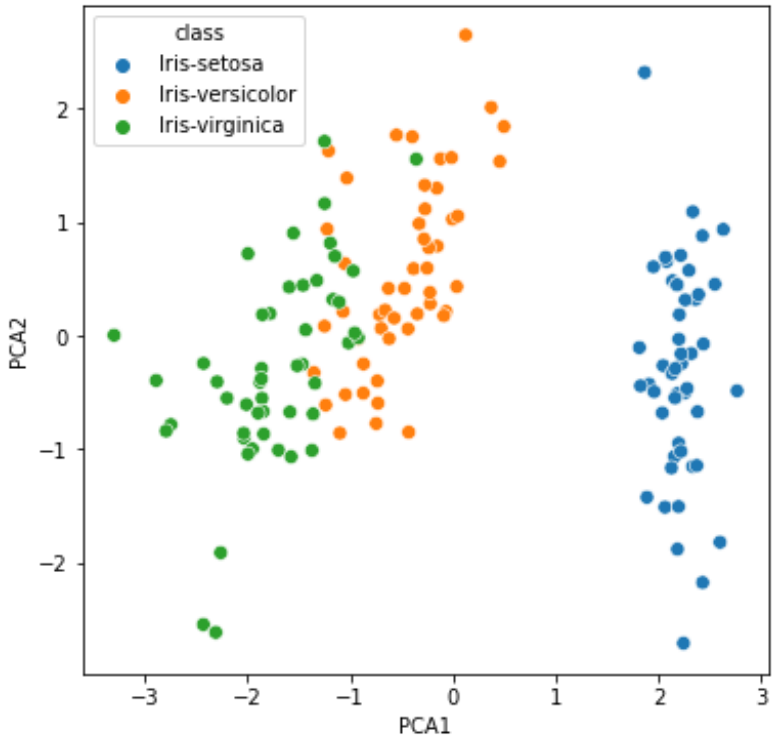
Final Dataset = $(\text{Feature Vector})^T$ x $(\text{Standardized Dataset})^T$

For this task, matrix multiplication functions in NumPy library were used.

```
reduced = feature_vector.transpose().dot(df.transpose())
reduceddf = pd.DataFrame(reduced.transpose())
reduceddf.columns = ['PCA1','PCA2']
reduceddf.head()
```

	PCA1	PCA2
0	2.256981	-0.504015
1	2.079459	0.653216
2	2.360044	0.317414
3	2.296504	0.573447
4	2.380802	-0.672514

Then, a scatter plot was drawn for this data that was projected onto the first 2 principal components. For this purpose, Matplotlib and Seaborn libraries were used.



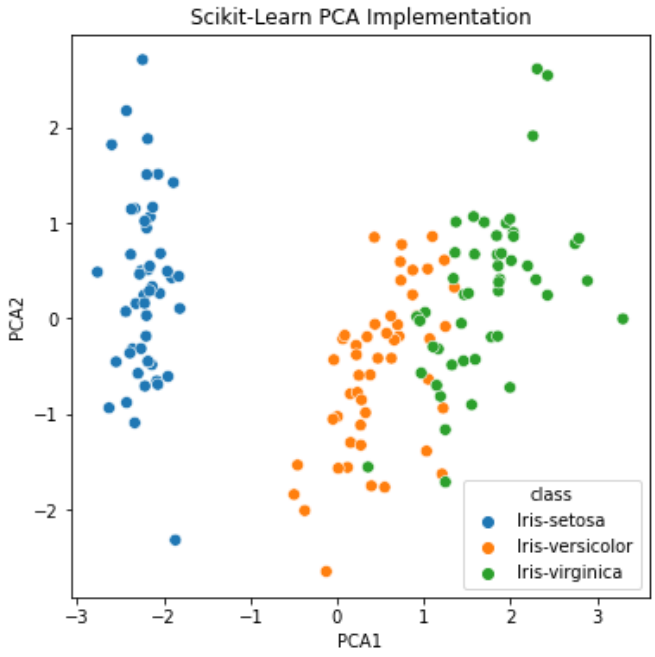
Task 5

In this step, the PCA implementation at Scikit-learn (sklearn.decomposition.PCA) was used to perform the same dimensionality reduction as given below.

```
pca_2 = PCA(n_components=2,random_state=3)
pca_2.fit(df)
reduceddf2 = pca_2.transform(df)
reduceddf2 = pd.DataFrame(reduceddf2)
reduceddf2.columns = ['PCA1','PCA2']
reduceddf2.head()
```

	PCA1	PCA2
0	-2.256981	0.504015
1	-2.079459	-0.653216
2	-2.360044	-0.317414
3	-2.296504	-0.573447
4	-2.380802	0.672514

The scatter plot for the data obtained using Scikit-learn PCA implementation is as follows.



However, there's a difference in the results obtained via the 2 methods. The result from Scikit-learn PCA implementation has opposite signs to the result obtained the other way.

Reason for difference in the results

The results of Singular Vector Decomposition are not unique in terms of singular vectors. However, it is unique up to a change in sign in pairs of left and right singular vectors. Therefore, to ensure a deterministic output, Scikit-learn PCA implementation flips the signs (if required) to impose that the largest coefficient in rows of matrix U (obtained from SVD) is positive. This might be the reason for the difference in signs of the results.

Anyhow, sign of the principal components doesn't matter. This is because, ultimately the components are vectors that identify the axes along which the data was projected for dimensionality reduction. No matter what direction each component points to, the new axes on which the data are reoriented will be the same.