CHRIST

(DEEMED TO BE UNIVERSITY)

PUNE LAVASA CAMPUS

The Hub of Analytics

LINEAR ALGEBRA

PRINCIPAL COMPONENT ANALYSIS – IRIS DATASET

Simran Adwani

(22112335)

Under the guidance of

Prof. Libin Chacko

Data Science

CHRIST (Deemed to be University), Pune, Lavasa Campus

April, 23, 2024

INDEX

CHRIST
(DEEMED TO BE UNIVERSITY)
PUNE LAVASA CAMPUS
The Hub of Analytics

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the Dean and Director, **Dr. Fr. Lijo Thomas**, Academic Coordinator, **Fr. Justin P Varghese**, Head of the Data Science Department, **Dr. Lija Jacob**, and to all the faculty members of the department of Data Science, for their constant support and suggestions in each phase of the project.

I would like to thank my university guide Dr. Libin Chacko for their continuous support and guidance throughout the project work.

I am indebted with gratitude to my parents who supported me throughout the way. Last but not least I would like to thank my peers for their help and support.

1. INTRODUCTION

1.1 What is PCA?

PCA was invented by Karl Pearson in 1901. Given a dataset with dimensions (where each dimension could be the measurement of a single gene, or cytokine, or SNP), we can project all observations into a set of k new dimensions, which end up being linear combinations of the original dimensions. Though some information is "lost" in this reduction, PCA preserves the maximal variation in those new dimensions.

PCA is useful for reducing the size of data, particularly data with many features while retaining the ability to model. This is a highly intelligible way, though watered down, of saying 'PCA uses singular value decomposition to find orthogonal, recombined principle components from the original features which can explain most of the variance in the original data with remarkably less memory'.

Steps Involved:

- Standardize the data. (with mean =0 and variance = 1)
- Compute the Covariance matrix of dimensions.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix (we can also use correlation matrix or even Single value decomposition.
- Sort eigenvalues in descending order and choose the top k Eigenvectors that correspond to the k largest eigenvalues (k will become the number of dimensions of the new feature subspace $k \leq d$, d is the number of original dimensions).
- Construct the projection matrix W from the selected k Eigenvectors.
- Transform the original data set X via W to obtain the new k-dimensional feature subspace Y.

1.2 IRIS Dataset

The Iris dataset is one of those datasets that one frequently encounters in the pursuit of acquiring or honing data science techniques. The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica, and Iris versicolor).

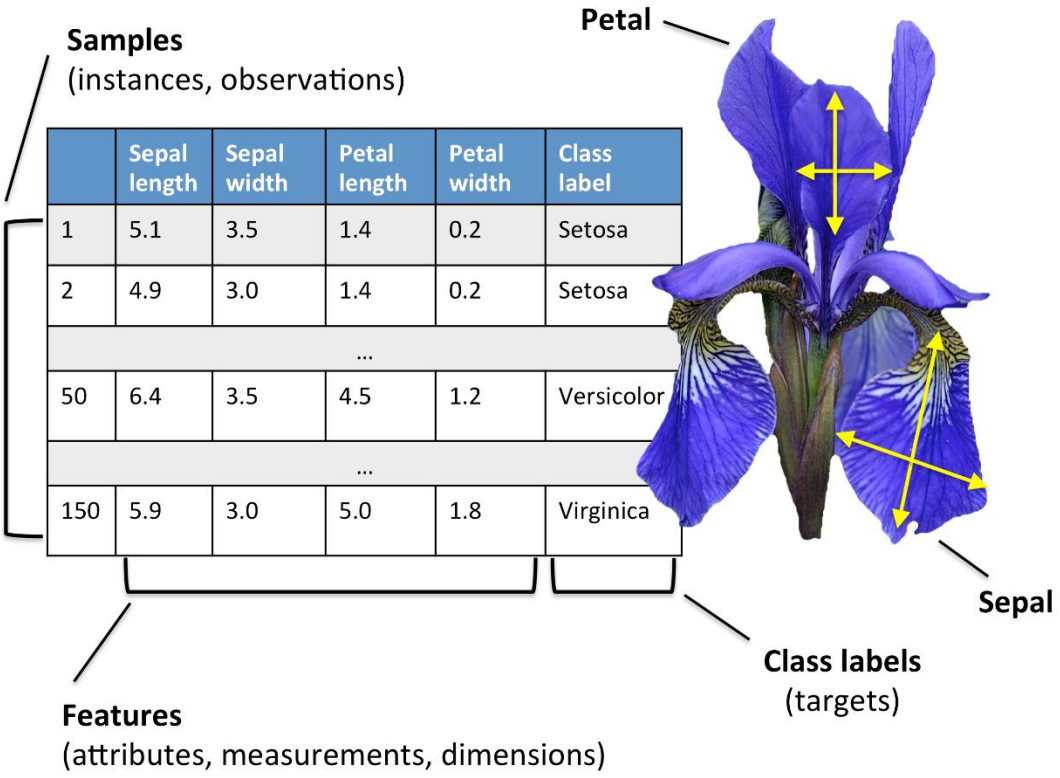
The iris dataset contains measurements for 150 iris flowers from three different species.

The three classes in the Iris dataset are:

- Iris-setosa (n=50)
- Iris-versicolor (n=50)
- Iris-virginica (n=50)

And the four features of in Iris dataset are:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm



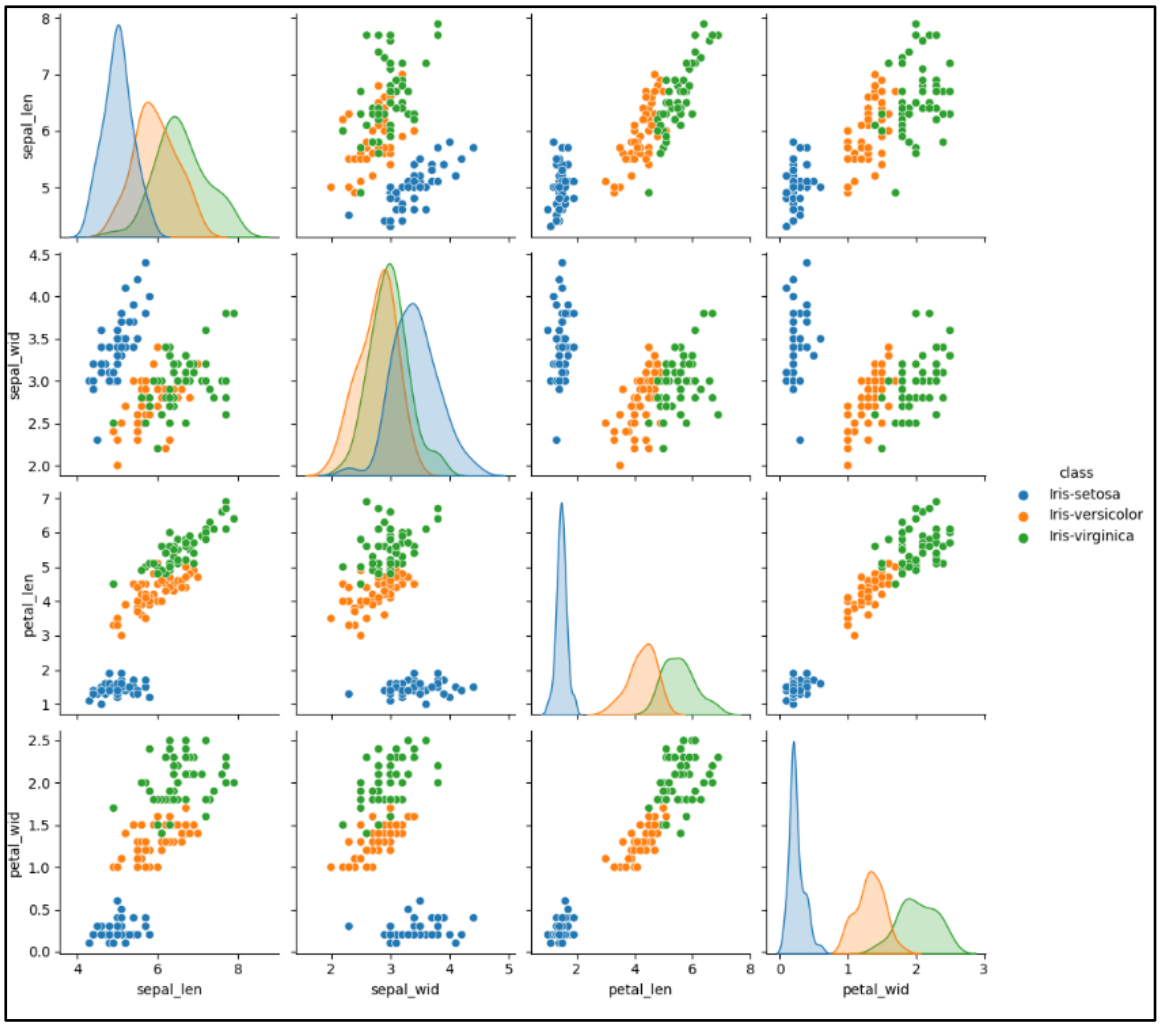
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

1.3 Understanding of the Dataset - Pairplot

Feature Distributions: The diagonal plots in the pair plot show the distribution of each feature. We can observe that sepal length and petal length have a roughly normal distribution, while sepal width and petal width have slightly skewed distributions.

Feature Correlations: The strength and direction of the correlations between pairs of features can be observed from the scatter plots. For example, we can see a strong positive correlation between petal length and petal width and a moderate positive correlation between sepal length and petal length.

Class Separability: The different colors and markers used in the scatter plots represent the three different classes (species) of iris flowers. By observing the scatter plots, we can assess how well the classes are separated in different feature spaces. For instance, the petal length vs. petal width plot shows a good separation between the Iris Setosa class and the other two classes, while the sepal length vs. sepal width plot does not provide a clear separation between the classes.



1.4 Should we use PCA?

I performed Bartlett's Test of Sphericity to identify whether to perform PCA on the given dataset or not to perform the same.

Bartlett's Test of Sphericity: It tests the hypothesis that the variables are uncorrelated within the population.

H0: Null Hypothesis: All variables in the data are uncorrelated.

Ha: Alternate Hypothesis: At least one pair of variables in the data are correlated if the null hypothesis cannot be rejected, then PCA is not advisable.

If the p-value is small, then we can reject the Null Hypothesis and agree that there is at least one pair of variables in the data that are correlated hence PCA is recommended.

For our dataset p_value is 0.0 which is less than 0.05. Hence we can reject the Null Hypothesis and agree that there is at least one pair of variables in the data that are correlated. Hence, PCA is recommended.

```
[61]: from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
      chi_square_value,p_value=calculate_bartlett_sphericity(X_std)
      p_value

[61]: 2.5882058732105877e-149
```

2. CODE SNIPPETS AND EXPLANATION

2.1 Loading the Dataset and importing necessary libraries

- pandas
  Pandas is a popular data manipulation and analysis library in Python. It was used to load the Iris dataset into a DataFrame and perform operations like checking for null values, dropping rows, and selecting columns.

- seaborn
  Seaborn is a data visualization library based on matplotlib. It was used to create a pair plot of the Iris dataset, which shows the pairwise relationships between the features and the target variable.

- numpy
  NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, and includes a large collection of high-level mathematical functions to operate on these arrays. In the code, NumPy was used for various mathematical operations, such as calculating the covariance matrix, performing eigen decomposition, and projecting the data onto the principal components.

- matplotlib.pyplot
  Matplotlib is a plotting library for creating static, animated, and interactive visualizations in Python. The pyplot module was likely used for creating the scatter plots and visualizations of the projected data onto the principal components.

- sklearn.preprocessing
  Scikit-learn is a machine-learning library in Python. The StandardScaler class from the preprocessing module was used to standardize the features (sepal length, sepal width, petal length, and petal width) before applying PCA. Standardization is necessary to ensure that features with different scales do not dominate the analysis.

- factor_analyzer
  The factor_analyzer library in Python provides a function called calculate_bartlett_sphericity that can be used to perform Bartlett's test of sphericity. This test checks if the correlation matrix is significantly different from an identity matrix, which would indicate that the variables are correlated and therefore suitable for factor analysis or PCA.

```
[1]: import numpy as np
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     from sklearn.datasets import load_wine
     %matplotlib inline
     from sklearn.preprocessing import StandardScaler

     import matplotlib.pyplot as plt
     import mpl_toolkits.mplot3d
     import seaborn as sns

     df = pd.read_csv( filepath_or_buffer='https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data', header=None, sep=',')
```

2.2 Data Wrangling

- Checked for null values and dropped,if any.
- Made sure that the values are float or int by looking at their data types.
- Explored the label/class/species columns, by using nunique function.
- Split the dataset.

```
[2]: df.columns=['sepal_len', 'sepal_wid', 'petal_len', 'petal_wid', 'class']

[3]: print(df.isnull().values.any())
     False

[4]: df.dropna(how="all", inplace=True)

[5]: df.tail()
```

[5]:

	sepal_len	sepal_wid	petal_len	petal_wid	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
[6]: df['class'].nunique()

[6]: 3
```

- Our iris dataset is now stored in form of a 150×4 matrix where the columns are the different features, and every row represents a separate flower sample. Each sample row x can be pictured as a 4-dimensional vector.

$$\mathbf{x^T} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \text{sepal length} \\ \text{sepal width} \\ \text{petal length} \\ \text{petal width} \end{pmatrix}$$

```
[7]: X = df.iloc[:,0:4].values
     y = df.iloc[:,4].values
```

2.3 Standardization

When there are different scales used for the measurement of the values of the features, then it is advisable to do the standardization to bring all the feature spaces with mean = 0 and variance = 1.

The reason why standardization is very much needed before performing PCA is that PCA is very sensitive to variances. Meaning, if there are large differences between the scales

(ranges) of the features, then those with larger scales will dominate over those with small scales. For example, a feature that ranges between 0 to 100 will dominate over a feature that ranges between 0 to 1 and it will lead to biased results. So, transforming the data to the same scales will prevent this problem. That is where we use standardization to bring the features with a mean value of 0 and a variance 1.

So here is the formula to calculate the standardized value of features:

$$\boldsymbol{Standardized\ value\ of\ x_i = \frac{x_i\ - mean\ of\ x}{std\ Deviation\ of\ x}}$$

In the output screenshot below you see that all X_std values are standardized in the range of -1 to +1.

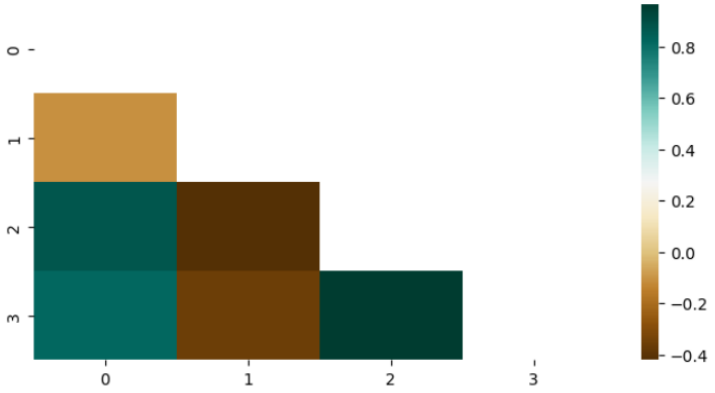
Since PCA yields a feature subspace that maximizes the variance along the axes, it makes sense to standardize the data, especially, if it was measured on different scales. Although, all features in the Iris dataset were measured in centimeters, let us continue with the transformation of the data onto unit scale (mean=0 and variance=1), which is a requirement for the optimal performance of many machine learning algorithms.

```
[8]: from sklearn.preprocessing import StandardScaler
     X_std = StandardScaler().fit_transform(X)
```

2.4 Covariance Matrix Calculation and Heatmap

```
[42]: X_std_df = pd.DataFrame(X_std)

      # Calculate the correlation matrix
      corr_matrix = X_std_df.corr()
      mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

      # Plot the heatmap with a valid color palette and annotations
      plt.figure(figsize=(8, 4))
      sns.heatmap(corr_matrix, cmap='BrBG', annot=True, mask=mask, fmt=".2f")
      plt.show()
```

```
C:\Users\simra\AppData\Local\Programs\Python\Python310\lib\site-packages\seaborn\matrix.py:260: FutureWarning: Format st
ut in future may error or produce different behavior
  annotation = ("{:" + self.fmt + "}").format(val)
```



- The darkest red off-diagonal cells correspond to the strongest positive correlations, which are between petal length and sepal length, and between petal length and petal width. This confirms the observations from the covariance matrix.
- The blue off-diagonal cells correspond to negative correlations, with the darkest blue representing the strongest negative correlations, which are between petal length and sepal width, and between petal width and sepal width. This also aligns with the covariance matrix findings.

The classic approach to PCA is to perform the Eigendecomposition on the covariance matrix Σ, which is a d×d matrix where each element represents the covariance between two features. Note, d is the number of original dimensions of the data set. In Iris data set we have 4 features hence covariance matrix will be of order 4×4.

[28]:
```
X_std_df.corr()
```

[28]:

	0	1	2	3
0	1.000000	-0.109369	0.871754	0.817954
1	-0.109369	1.000000	-0.420516	-0.356544
2	0.871754	-0.420516	1.000000	0.962757
3	0.817954	-0.356544	0.962757	1.000000

[29]:
```
mean_vec = np.mean(X_std, axis=0)
cov_mat = (X_std - mean_vec).T.dot((X_std - mean_vec)) / (X_std.shape[0]-1)
print('Covariance matrix \n%s' %cov_mat)
```

```
Covariance matrix
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

- The diagonal elements represent the variances of the individual features. For example, the variance of sepal length is 1.00671141.
- The off-diagonal elements represent the covariances between pairs of features. A positive covariance indicates that the two features tend to increase or decrease together, while a negative covariance indicates an inverse relationship.
- The largest positive covariances are between petal length and sepal length (0.87760486), and between petal length and petal width (0.96921855). This suggests a strong positive correlation between these pairs of features.
- The largest negative covariances are between petal length and sepal width (-0.42333835), and between petal width and sepal width (-0.358937). This suggests a moderate negative correlation between these pairs of features.
- The covariance between sepal length and sepal width (-0.11010327) is relatively small, indicating a weak negative correlation.

Overall, both the covariance matrix and the heatmap provide insights into the relationships and correlations between the features in the Iris dataset. The covariance matrix quantifies the degree of linear relationship, while the heatmap offers a visual representation of the correlation strengths and directions. These interpretations can inform further analysis, such as feature selection or dimensionality reduction techniques like Principal Component Analysis (PCA).

2.5 Eigendecomposition — Computing Eigenvectors and Eigenvalues

The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the "core" of a PCA.

The Eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude.

In other words, the eigenvalues explain the variance of the data along the new feature axes. It means the corresponding eigenvalue tells us how much variance is included in that new transformed feature.

To get eigenvalues and Eigenvectors we need to compute the covariance matrix, which we did in the previous step.

```
[ ]: cov_mat = np.cov(X_std.T)
     eig_vals, eig_vecs = np.linalg.eig(cov_mat)
     print('Eigenvectors \n%s' %eig_vecs)
     print('\nEigenvalues \n%s' %eig_vals)

     Eigenvectors
     [[ 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]]

     Eigenvalues
     [2.93035378 0.92740362 0.14834223 0.02074601]
```

Eigenvector:

- Each column in the eigenvector matrix represents an eigenvector, which corresponds to a principal component.
- The eigenvectors are orthogonal (perpendicular) to each other, ensuring that the principal components capture unique and independent sources of variation in the data.
- The elements (coefficients) of each eigenvector represent the contribution of the corresponding original features to that principal component.
- The first eigenvector (first column) has the highest coefficients for sepal length (0.52237162) and petal length (0.58125401), indicating that these features contribute the most to the first principal component.

- The second eigenvector (second column) has the highest coefficient for sepal width (-0.92555649), suggesting that this feature is the primary contributor to the second principal component.
- The third and fourth eigenvectors capture the remaining variation in the data, but their interpretations may be less straightforward due to their lower eigenvalues.

Eigenvalues:

- The eigenvalues represent the amount of variance captured by each corresponding principal component (eigenvector).
- The sum of all eigenvalues is equal to the total variance in the original dataset.
- The first eigenvalue (2.93035378) is the largest, indicating that the first principal component captures the maximum variance in the data.
- The second eigenvalue (0.92740362) is the second-largest, indicating that the second principal component captures the second-largest portion of the variance.
- The eigenvalues decrease in magnitude for subsequent principal components, with the fourth eigenvalue being relatively small (0.02074601).
- By examining the proportion of variance explained by each eigenvalue (eigenvalue divided by the sum of all eigenvalues), we can determine how many principal components to retain for dimensionality reduction while preserving most of the variance in the data.

Based on these interpretations, we can make the following observations:

- The first two principal components (eigenvectors) capture a significant portion of the total variance in the Iris dataset, as indicated by their relatively large eigenvalues.
- The first principal component is primarily influenced by sepal length and petal length, while the second principal component is primarily influenced by sepal width.
- By retaining only the first two principal components, we can potentially reduce the dimensionality of the data from 4 features to 2 principal components while preserving a substantial amount of the total variance.

2.6 Eigen Vectors Verification

As we know the sum of the square of each value in an Eigenvector is 1. So let's see if it holds which means we have computed Eigenvectors correctly.

```
[ ]: sq_eig=[]
     for i in eig_vecs:
         sq_eig.append(i**2)
         print(sq_eig)

[array([0.27287211, 0.13862096, 0.51986524, 0.06864169])]
[array([0.27287211, 0.13862096, 0.51986524, 0.06864169]), array([0.06935581, 0.85665482, 0.05857991, 0.01540945])]
[array([0.27287211, 0.13862096, 0.51986524, 0.06864169]), array([0.06935581, 0.85665482, 0.05857991, 0.01540945]), array([3.37856219e-01, 4.44989610e-04, 1.98506285e
-02, 6.41848163e-01])]
[array([0.27287211, 0.13862096, 0.51986524, 0.06864169]), array([0.06935581, 0.85665482, 0.05857991, 0.01540945]), array([3.37856219e-01, 4.44989610e-04, 1.98506285e
-02, 6.41848163e-01]), array([0.31991586, 0.00427922, 0.40170422, 0.2741007 ])]
```

```
[ ]: print("sum of squares of each values in an eigen vector is \n", 0.27287211+ 0.13862096+0.51986524+ 0.06864169)
     for ev in eig_vecs: np.testing.assert_array_almost_equal(1.0, np.linalg.norm(ev))

     sum of squares of each values in an eigen vector is
      1.0
```

2.7 Principal Component Selection

To decide which Eigenvector(s) can be dropped without losing too much information for the construction of lower-dimensional subspace, we need to inspect the corresponding

```
[ ]: eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i])
     for i in range(len(eig_vals))]
     print(type(eig_pairs))

     #Sort the (eigenvalue, eigenvector) tuples from high to low eig_pairs.sort()
     eig_pairs.sort(reverse=True)
     print("\n",eig_pairs)
     #Visually confirm that the list is correctly sorted by decreasing eigenvalues
     print('\n\n\nEigenvalues in descending order:')
     for i in eig_pairs:
         print(i[0])

     <class 'list'>

      [(2.930353775589318, array([ 0.52237162, -0.26335492,  0.58125401,  0.56561105])), (0.9274036215173425, array([-0.37231836, -0.92555649, -0.02109478, -0.0654157
     7])), (0.14834222648163978, array([-0.72101681,  0.24203288,  0.14089226,  0.6338014 ])), (0.02074601399559601, array([ 0.26199559, -0.12413481, -0.80115427,  0.523
     4627]))]



     Eigenvalues in descending order:
     2.930353775589318
     0.9274036215173425
     0.14834222648163978
     0.02074601399559601
```

Eigenvalues:

- The Eigenvectors with the lowest eigenvalues bear the least information about the distribution of the data; those are the ones can be dropped.
- To do so, the common approach is to rank the eigenvalues from highest to lowest to choose the top k Eigenvectors.

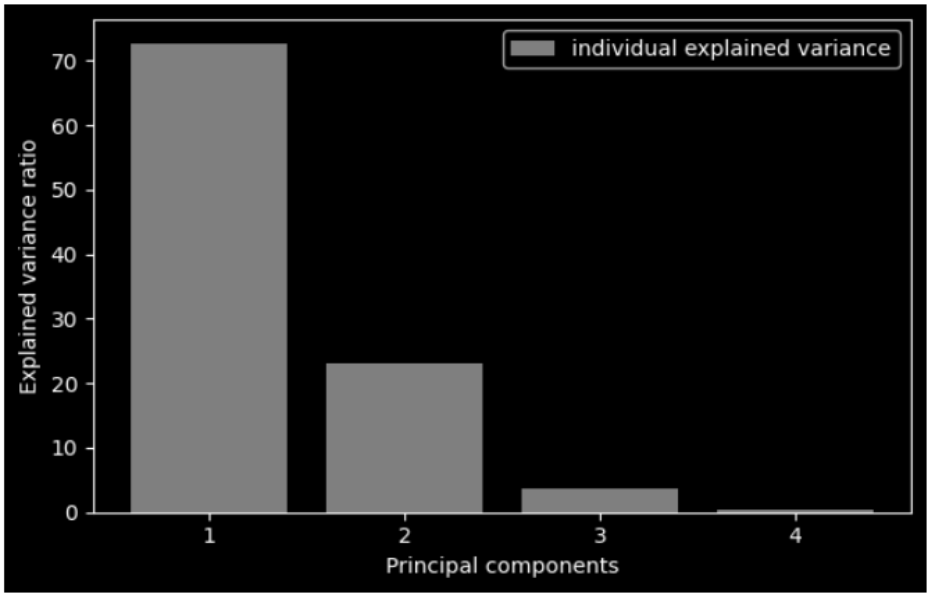
- After sorting the eigenpairs, the next question is “How many principal components are we going to choose for our new feature subspace?”
- A useful measure is the so-called “explained variance,” which can be calculated from the eigenvalues.
- The explained variance tells us how much information (variance) can be attributed to each of the principal components.

```
[ ]: tot = sum(eig_vals)
print("\n",tot)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
print("\n\n1. Variance Explained\n",var_exp)
cum_var_exp = np.cumsum(var_exp)
print("\n\n2. Cumulative Variance Explained\n",cum_var_exp)
print("\n\n3. Percentage of variance the first two principal components each contain\n ",var_exp[0:2])
print("\n\n4. Percentage of variance the first two principal components together contain\n",sum(var_exp[0:2]))
```

```
 4.026845637583896


1. Variance Explained
 [72.77045209380134, 23.030523267680653, 3.683831957627385, 0.5151926808906337]


2. Cumulative Variance Explained
 [ 72.77045209  95.80097536  99.48480732 100.        ]


3. Percentage of variance the first two principal components each contain
  [72.77045209380134, 23.030523267680653]


4. Percentage of variance the first two principal components together contain
 95.80097536148199
```

The below graph shows the explained variance in a visualized form. The plot above clearly shows that most of the variance (72.77% of the variance to be precise) can be explained by the first principal component alone. The second principal component still bears some information (23.03%) while the third and fourth principal components can safely be dropped without losing too much information. Together, the first two principal components contain 95.8% of the information.

```
[48]: tot = sum(eig_vals)
with plt.style.context('dark_background'):
    plt.figure(figsize=(6, 4))

    plt.bar(range(len(var_exp)), var_exp, alpha=0.5, align='center',
            label='individual explained variance', color='white')
    plt.ylabel('Explained variance ratio')
    plt.xlabel('Principal components')
    plt.xticks(range(len(var_exp)), range(1, len(var_exp) + 1))
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()
```



2.8 Data Projection Matrix

A projection matrix will be used to transform the Iris data onto the new feature subspace or we say newly transformed data set with reduced dimensions.

It is a matrix of our concatenated top k Eigenvectors.

Here, we are reducing the 4-dimensional feature space to a 2-dimensional feature subspace, by choosing the "top 2" Eigenvectors with the highest Eigenvalues to construct our d×k-dimensional Eigenvector matrix W.

```
[ ]: print(eig_pairs[0][1])
     print(eig_pairs[1][1])
     matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1), eig_pairs[1][1].reshape(4,1)))
     print('Matrix W:\n', matrix_w)
```

```
[ 0.52237162 -0.26335492  0.58125401  0.56561105]
[-0.37231836 -0.92555649 -0.02109478 -0.06541577]
Matrix W:
 [[ 0.52237162 -0.37231836]
 [-0.26335492 -0.92555649]
 [ 0.58125401 -0.02109478]
 [ 0.56561105 -0.06541577]]
```

2.9 Projection Onto the New Feature Space

In this last step, we will use the 4×2-dimensional projection matrix W to transform our samples onto the new subspace via the equation Y=X×W, where the output matrix Y will be a 150×2 matrix of our transformed samples.

```
[ ]: Y = X_std.dot(matrix_w)
     principalDf = pd.DataFrame(data = Y , columns = ['principal component 1', 'principal component 2'])
     principalDf.head()
```

[36]:

	principal component 1	principal component 2
0	-2.264542	-0.505704
1	-2.086426	0.655405
2	-2.367950	0.318477
3	-2.304197	0.575368
4	-2.388777	-0.674767

```
[ ]: finalDf = pd.concat([principalDf,pd.DataFrame(y,columns = ['species'])], axis = 1)
     finalDf.head()
```

[37]:

	principal component 1	principal component 2	species
0	-2.264542	-0.505704	Iris-setosa
1	-2.086426	0.655405	Iris-setosa
2	-2.367950	0.318477	Iris-setosa
3	-2.304197	0.575368	Iris-setosa
4	-2.388777	-0.674767	Iris-setosa

```
[79]: projected_data = X_std.dot(matrix_w)
      reconstructed_data = projected_data.dot(matrix_w.T)
      reconstruction_error = np.mean(np.square(X_std - reconstructed_data)) * 100

      print("Reconstruction Error: {:.2f}%".format(reconstruction_error))

      Reconstruction Error: 4.20%

[77]: projected_data = X_std.dot(matrix_w)

      # Calculate the projection error for each data point
      projection_errors = np.linalg.norm(X_std - projected_data.dot(matrix_w.T), axis=1)

      # Calculate the average projection error
      average_projection_error = np.mean(projection_errors) * 100

      print("Average Projection Error: {:.2f}%".format(average_projection_error))

      Average Projection Error: 34.32%
```

The principal components explain approximately 95.80% of the total variance in the original data (as calculated above), as the reconstruction error represents the percentage of variance that is not captured by the principal components. A lower reconstruction error indicates that the principal components are more effective at capturing the variability present in the original dataset. Therefore, in this case***, a reconstruction error of 4.20% suggests that the principal components are doing a relatively good job of representing the data.***

In this case, a projection error of 34.32% suggests that the principal components may not fully capture the variability present in the original dataset, and there may be significant information loss during the dimensionality reduction process.

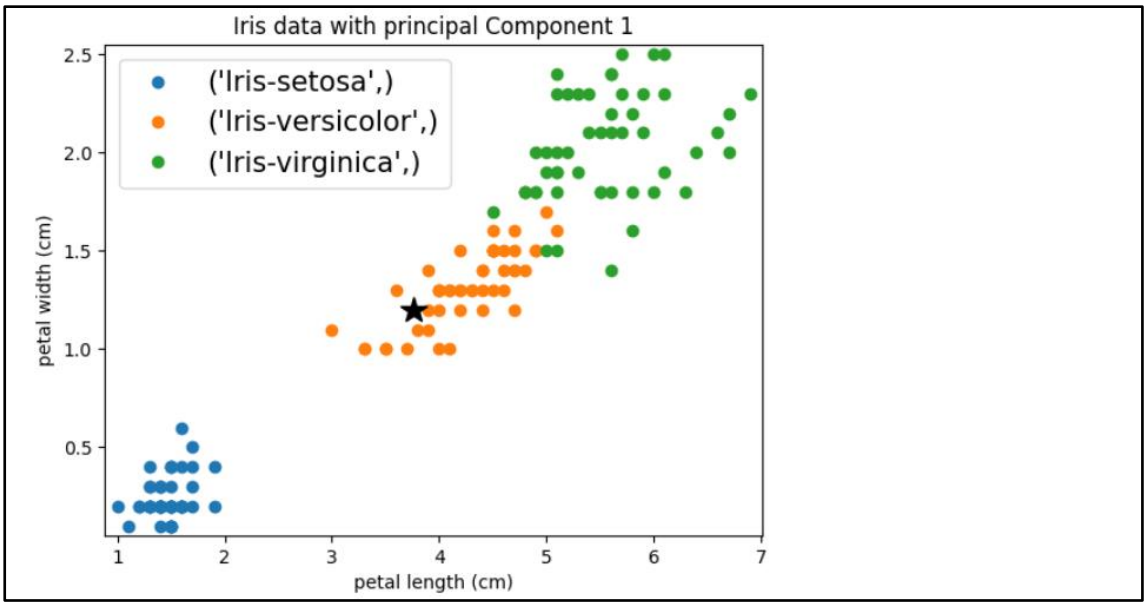
3. VISUALISATION

The result of this plot shows that the first principal component alone can separate the Iris setosa species (blue markers) from the other two species (Iris versicolor and Iris virginica) quite well. However, there is some overlap between Iris versicolor and Iris virginica along the first principal component axis, indicating that additional principal components might be needed to fully separate these two species.

Overall, the 2D scatter plot visualization allows you to assess the separability of the different classes (species) in the reduced principal component space and gain insights into the underlying patterns and relationships within the data.

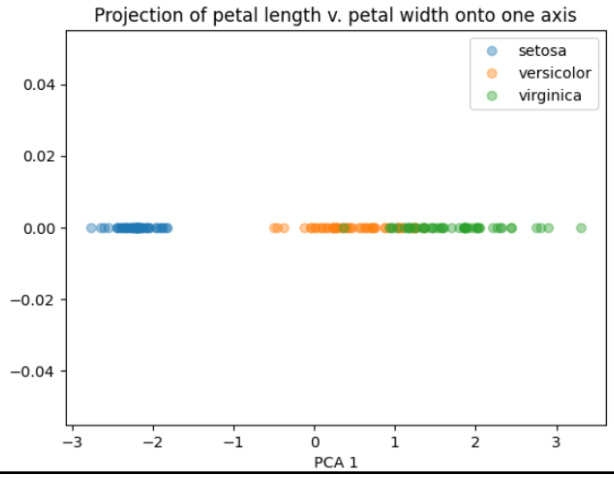
2D Graph-

```
[49]: # Plot Petal length vs petal width only
m = np.array([df['petal_len'].mean(), df['petal_wid'].mean()])

for key, group in df.groupby(['class']):
    plt.plot(group['petal_len'], group['petal_wid'], label=key, marker='o', linestyle='none')

# Add the mean value to the plot
plt.plot(m[0], m[1], marker='*', color='black', markersize=15)

# Tidy up plot
plt.legend(loc=0, fontsize=15)
plt.margins(0.02)
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
plt.title('Iris data with principal Component 1')
plt.show()
```



1D Graph:

```
[47]: principal_component_1 = finalDf['principal component 1']
      for key, group in finalDf.groupby('species'):
          plt.plot(group['principal component 1'], np.zeros_like(group['principal component 1']),
                   label=key, marker='o', linestyle='none', alpha=0.4)
      plt.margins(0.05)
      plt.xlabel('PCA 1')
      plt.legend(['setosa', 'versicolor', 'virginica'])
      plt.title('Projection of petal length v. petal width onto one axis')
      plt.show()
```



4. CONCLUSION

Principal Component Analysis, a powerful dimensionality reduction technique, was applied to the Iris dataset, comprising measurements of sepal length, sepal width, petal length, and petal width for three distinct species of iris flowers. Through a structured approach involving data preprocessing, covariance matrix calculation, eigen decomposition, and projection onto principal components, this analysis unveiled valuable insights and facilitated effective dimensionality reduction.

The covariance matrix and correlation heatmap revealed intricate relationships between the features, with petal length and petal width exhibiting the strongest positive correlation, while petal length and sepal width displayed a moderate negative correlation. Eigen decomposition identified the first two principal components as capturing a substantial portion of the total variance, with the first component primarily influenced by sepal length and petal length, and the second by sepal width.

Visualizations of the projected data onto the first two principal components demonstrated remarkable separability among the three iris species, enabling dimensionality reduction from four features to two principal components while preserving most of the variance. The pair plot further corroborated these findings, showcasing the discriminatory power of petal length and petal width in distinguishing the species, hinting at potential feature redundancy.

The interpretations derived from the eigenvalues, eigenvectors, and visualizations provide a comprehensive understanding of the underlying structure and patterns within the Iris dataset. These insights can guide informed decision-making processes, such as feature selection, data visualization, and the development of robust classification models.

In conclusion, this Principal Component Analysis exemplifies the utility of dimensionality reduction techniques in extracting meaningful information from high-dimensional datasets, paving the way for efficient data analysis and modeling in diverse domains.

5. REFERENCES

- Sheik, M. (2022). Capstone Project: California Housing Price Prediction [Jupyter Notebook]. GitHub. Link
- Doddamani, S. Navigating the Dimensions: PCA, Covariance, and Feature Extraction [Blog post]. LinkedIn. Link
- Shafizadegan, M. PCA Data Analysis [Jupyter Notebook]. GitHub. Link
- Analytics Vidhya. (2021, September). PCA and its Underlying Mathematical Principles [Blog post]. Link
- GeeksforGeeks. Principal Component Analysis (PCA) [Web page]. Link
- Visual Design. Linear Algebra for ML Part 2: Principal Component Analysis [Blog post]. Link