

EDA Lab 1

André Plancha

andre.plancha@hotmail.com

Yana Zlatanova

yana-zlatanova-gold@gmail.com

MALMÖ UNIVERTITY

MA661E - VT25

March 30, 2025

1) Generate $n = 150$, $p = 6$ normally distributed random variables that have high variance in some dimension and low variance in another dimension. Try PCA using both correlation and covariance matrices.

Following the example given, a dataset with 2 dimensions of high variance and a dataset with 4 dimensions of low variance were created and combined into one dataset with $n = 150$, $p = 6$.

a) Is the covariance matrix very informative?

The covariance matrix PCA is not very informative in our example because it reflects only the variance of the the first 2 dimensions with high-variance and it fails to show the structures in the remaining dimensions.

b) Which one would be better to use in this case?

In our case correlation matrix would be a better choice because it standardizes all variables to have equal variance before finding principal components.

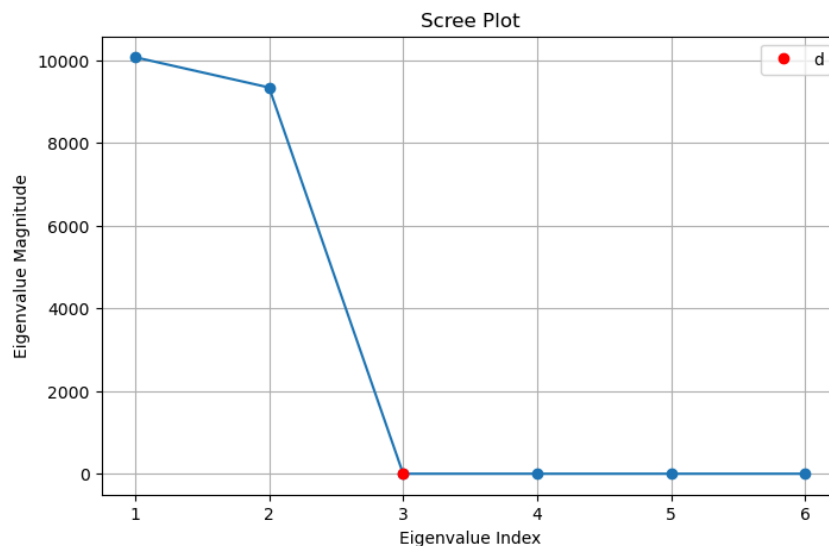


Figure 1: PCA using covariance matrix

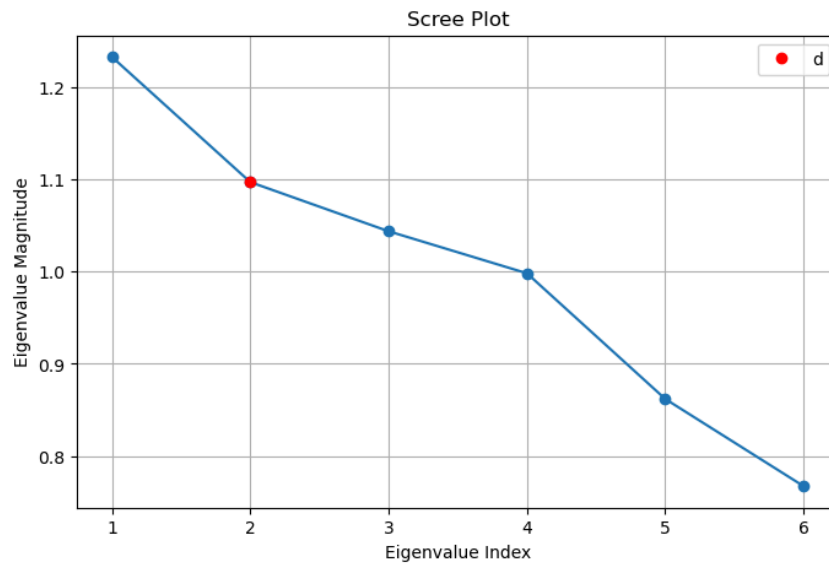


Figure 2: PCA using correlation matrix

2) For each case, apply Linear Discriminant Analysis (LDA) to the Iris dataset and visualize the data in one dimension, using the Kernel Density Estimate; and discuss how good the mapping are for each case.

case a): Apply LDA for only 2 classes at a time, i.e., [setosa, versicolor]; [versicolor, virginica]; [virginica, setosa].

The Iris dataset is composed of 150 samples of iris flowers with 4 numerical features that describe sepal length, sepal width, petal length and petal width. Each flower can be a setosa, a versicolor, or a virginica (target).

For this case, for each pair, we filtered out the other class and applied `sklearn.discriminant_analysis.LinearDiscriminantAnalysis(n_components=1)` to the remaining rows, using the class as the target. To calculate and plot Kernel Density Estimate, `plotnine.geom_density` was used, which by default uses a *Gaussian* kernel. The code for this process can be observed in Listing 3.

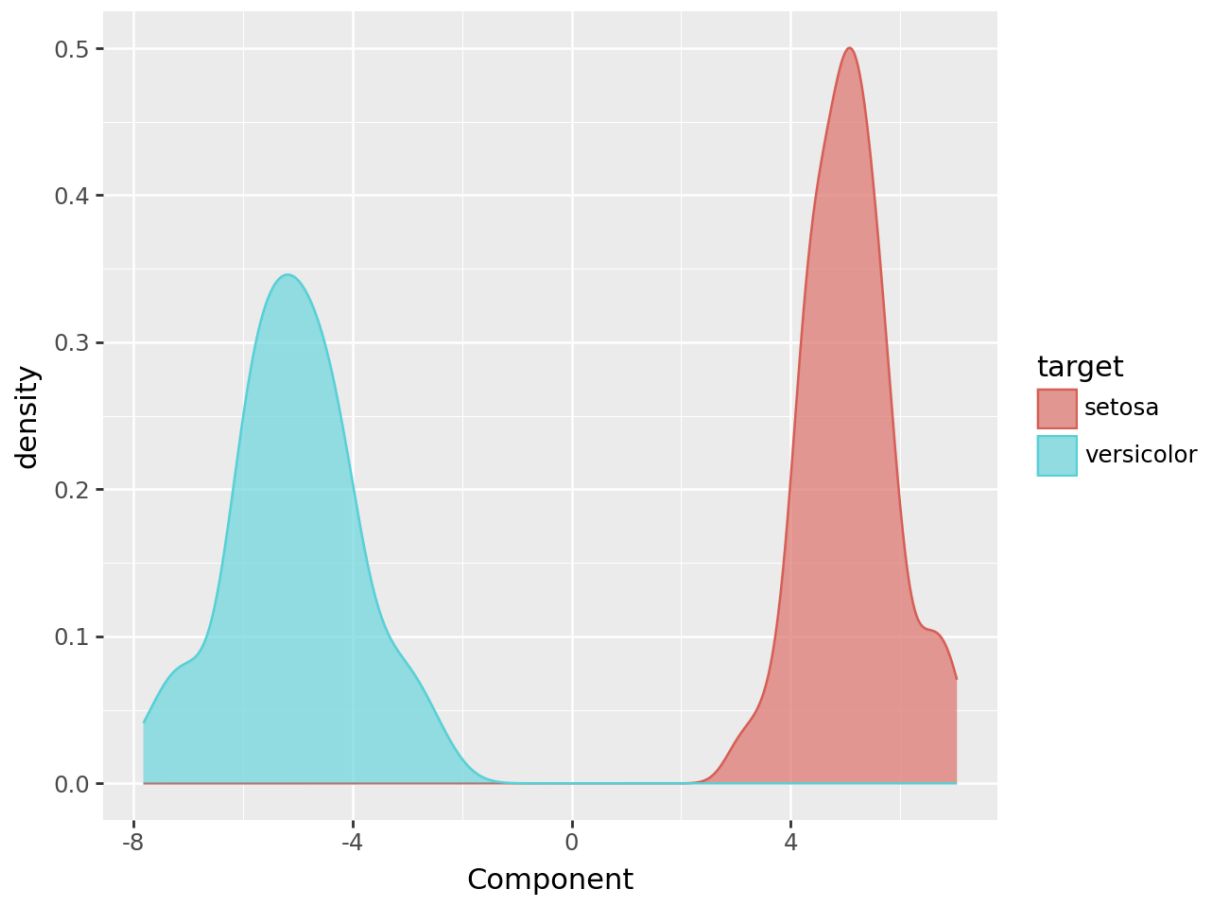


Figure 3: LDA without virginica

As observed in Figure 3, since there's no overlap between the mappings, the LDA transformation from the *setosa* and *versicolor* pair effectively found a direction where the two classes are linearly separable.

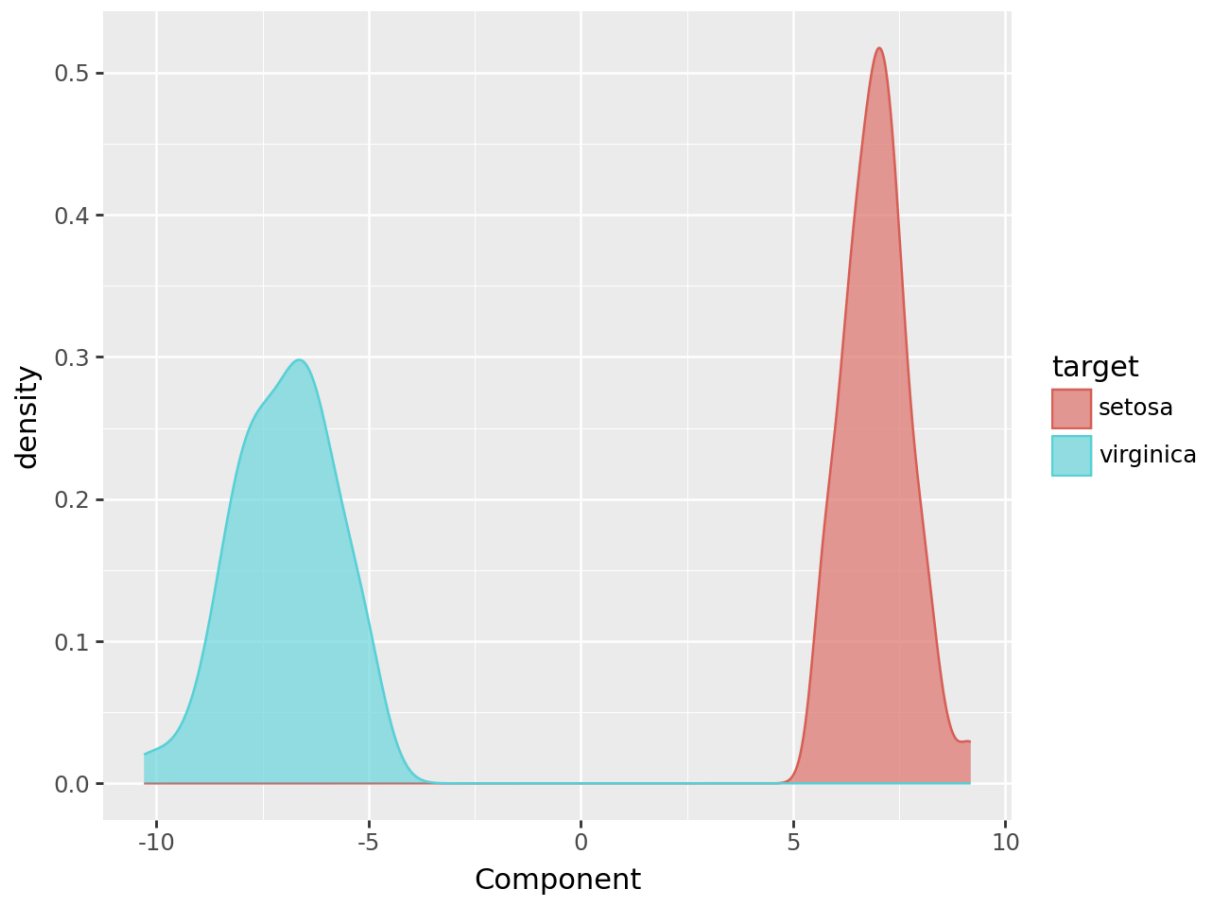


Figure 4: LDA without versicolor

Similar to the previous pair, the *setosa* and *virginica* pair also shows no overlap between the mappings, so the LDA transformation effectively found a direction where the two classes are linearly separable as well.

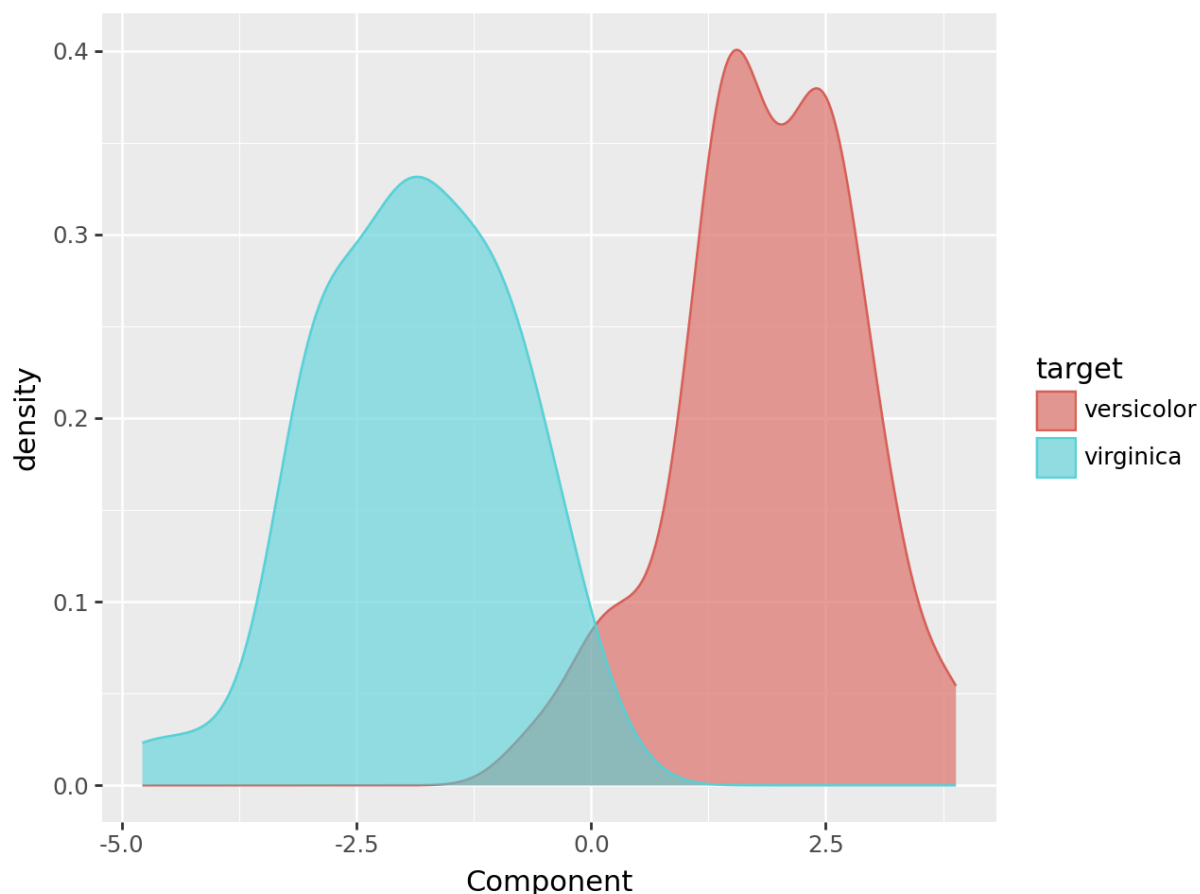


Figure 5: LDA without setosa

However, since there's some overlap in the transformation from the *versicolor* and *virginica* pair, as observed in Figure 5 we can conclude that the LDA couldn't find a direction as effective as the other ones; with that being said, the overlap is quite small, so it can be argued that it's still a good mapping.

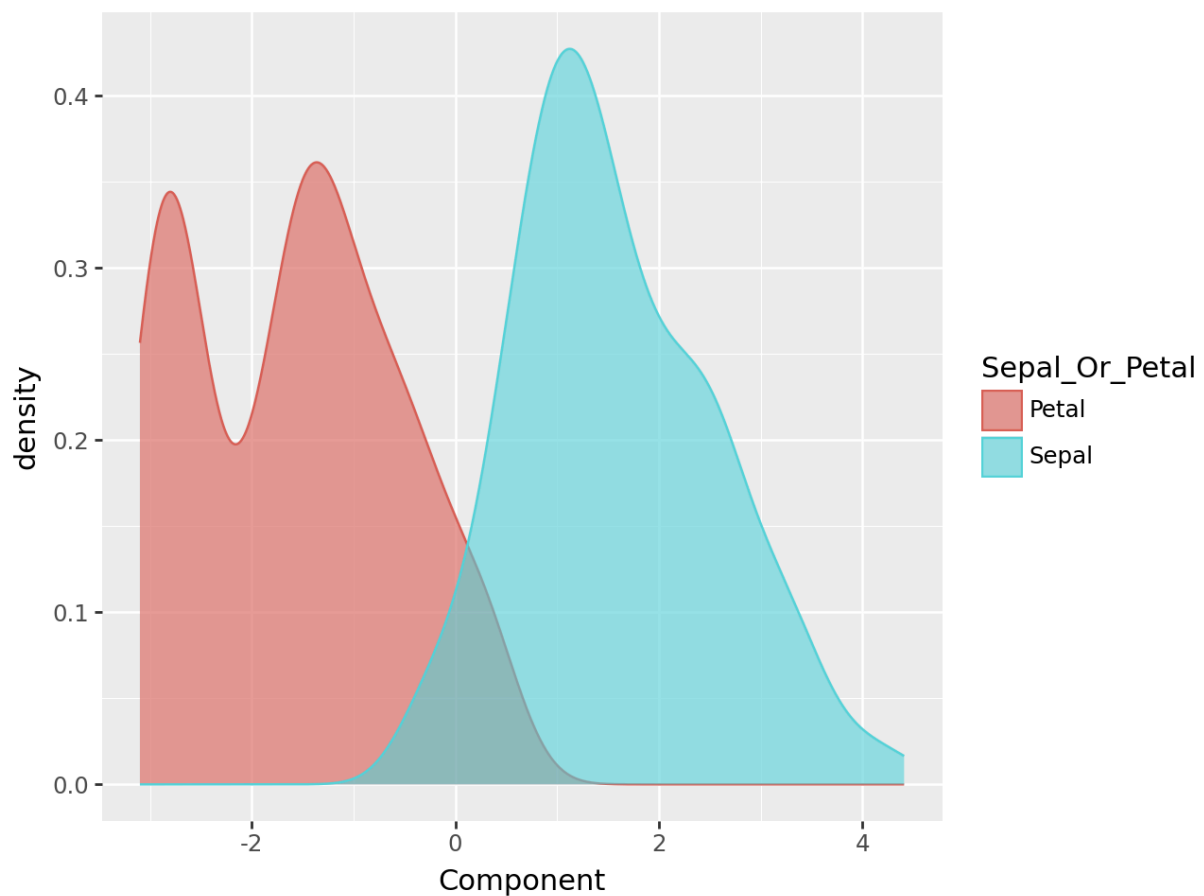
case b): Instead of classifying based on species, consider two broad classes—Sepals and Petals. Apply LDA to distinguish between sepal-based features ([sepal length, sepal width]) and petal-based features ([petal length, petal width]).

For this case, we decided on concatenating the sepal width/length and petal width/length to make a Width/Lenght column, and make the target Sepal_or_Petal describe if the row was produced from the Sepal or Petal part. A sample of this transformed dataset can be seen on Figure 6.

	# Width	# Length	△ Sepal_Or_Petal
146	2.5	6.3	Sepal
147	3.0	6.5	Sepal
148	3.4	6.2	Sepal
149	3.0	5.9	Sepal
150	0.2	1.4	Petal
151	0.2	1.4	Petal
152	0.2	1.3	Petal
153	0.2	1.5	Petal
154	0.2	1.4	Petal
155	0.4	1.7	Petal
156	0.3	1.4	Petal
157	0.2	1.5	Petal

Figure 6: Sepal or Petal dataset sample

After this, a similar process from 2) a) was done for the LDA transformation and Kernel Density Estimation. The code for this one can be observed in Listing 4.



As observed in Figure 7, there's some small overlap between the mappings, which suggests that the direction found by the LDA is somewhat effective.

3) Factory analysis

a) Repeat example 2.5 for dataset stockreturns.

For this analysis, we worked with the stockreturns dataset containing 10 columns (companies) and 100 rows (trading days, assumed). We first standardized the data and then applied

factor analysis to extract three factors. We implemented the analysis both with varimax rotation and without rotation for comparison.

The 2D factor loading plots below visualize how each company relates to the first two factors, revealing which companies tend to move together and which respond differently to the factors.

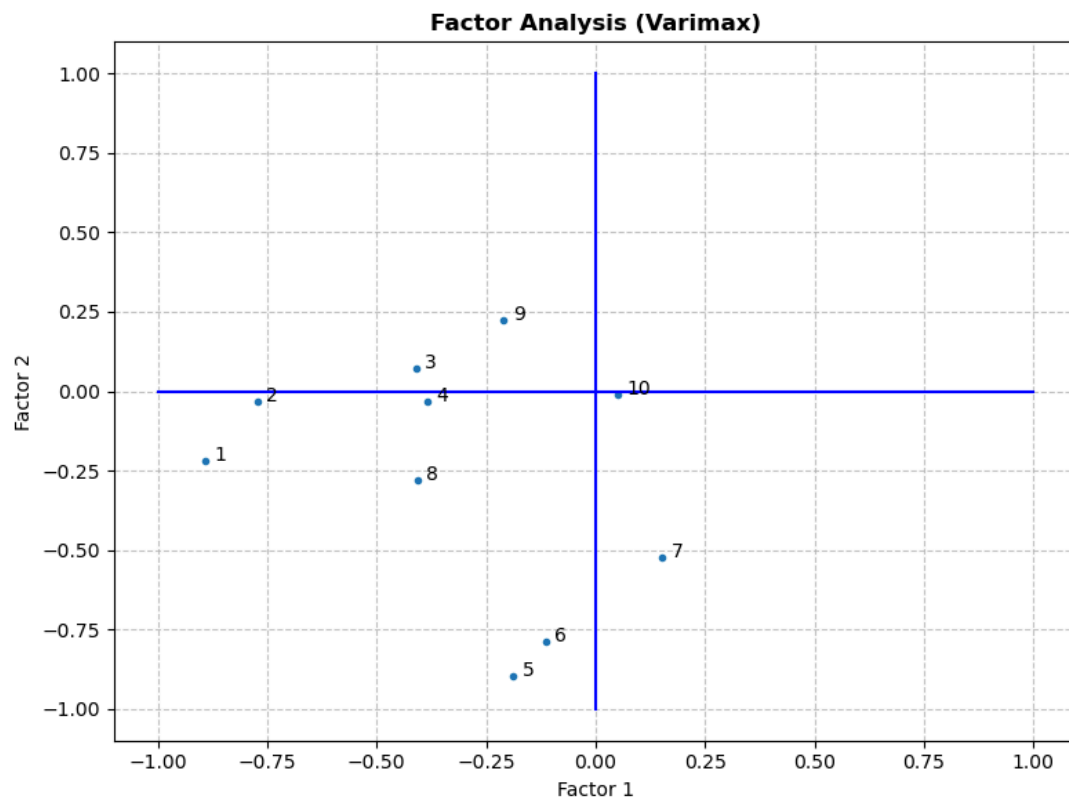


Figure 8: Factor loading plot with varimax rotation

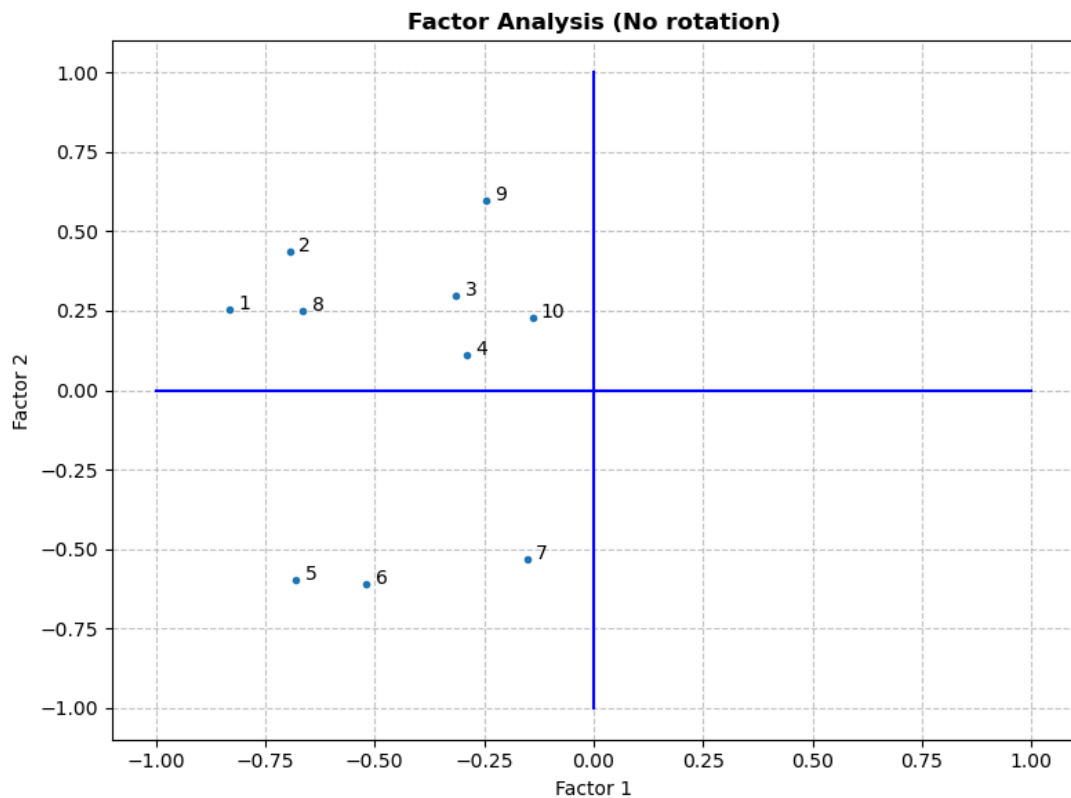


Figure 9: Factor loading plot with no rotation

If we compare the two figures, it's evident that rotation creates a more differentiated structure. The un-rotated solution primarily separates companies along Factor 2, while the rotated solution identifies more distinct groupings that better reflect underlying relationships.

The figure below shows a few plots that display the factor scores from the factor analysis with rotation, where each point represents a single trading day positioned according to its scores on each factor. These plots reveal varied market behavior across trading days with no distinct clustering patterns.

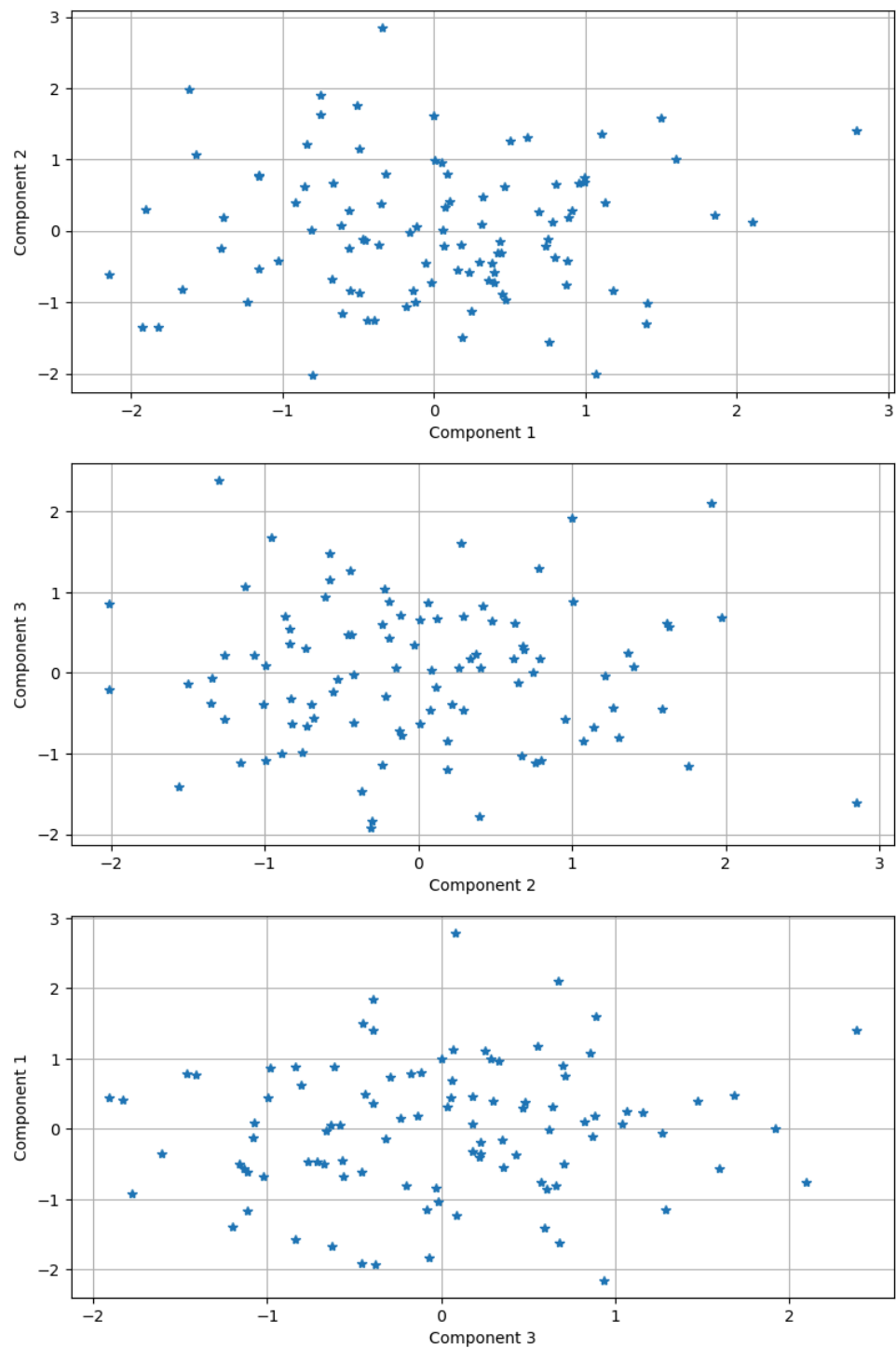


Figure 10: Factor scores from the factor analysis with rotation

b) Carry out a factor analysis for your data for 10 companies over 20 days.

Based on Figure 11, SEB and Nordea bank are positioned close together, indicating they have similar factor loadings and patterns of correlation with the underlying factors.

3D Factor Analysis

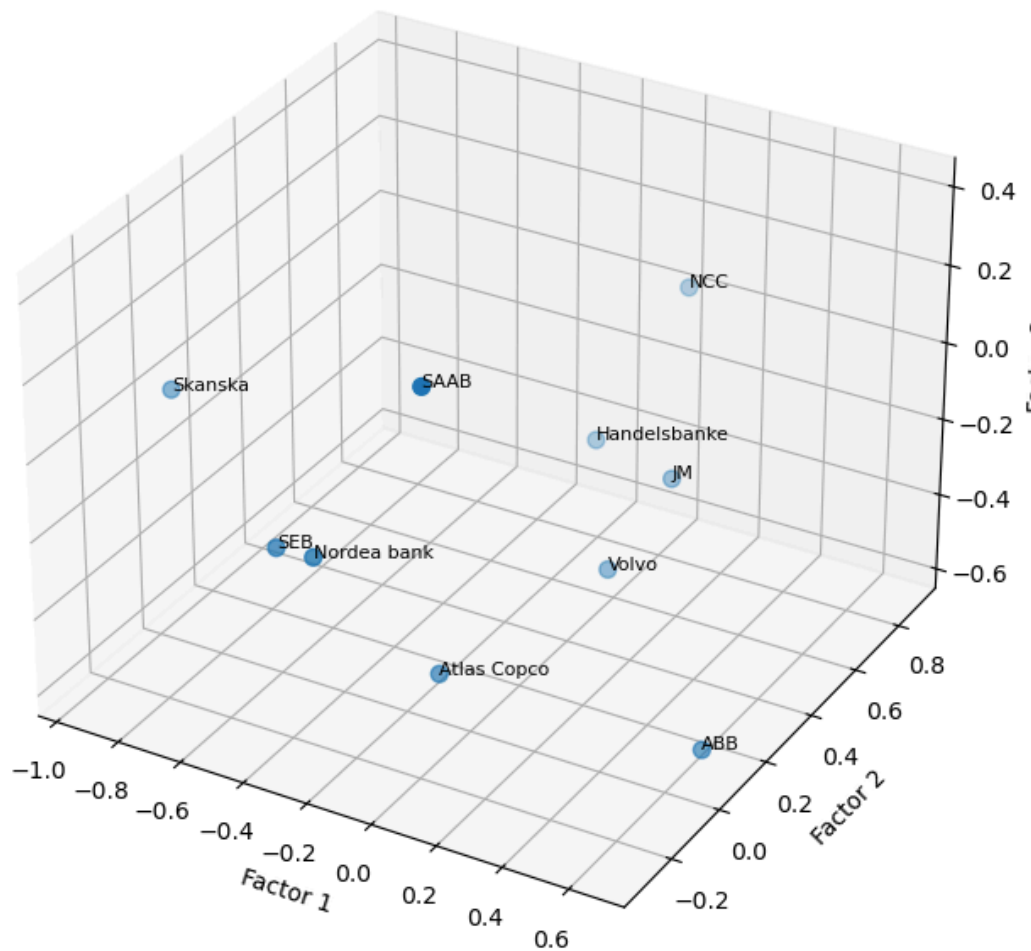


Figure 11: Factor loading plot with rotation

Figure 12 illustrates a broad distribution of points across all three plots, confirming that each component captures variation in the data. However, the absence of clear linear patterns between any pair of components suggests that the components are relatively independent.

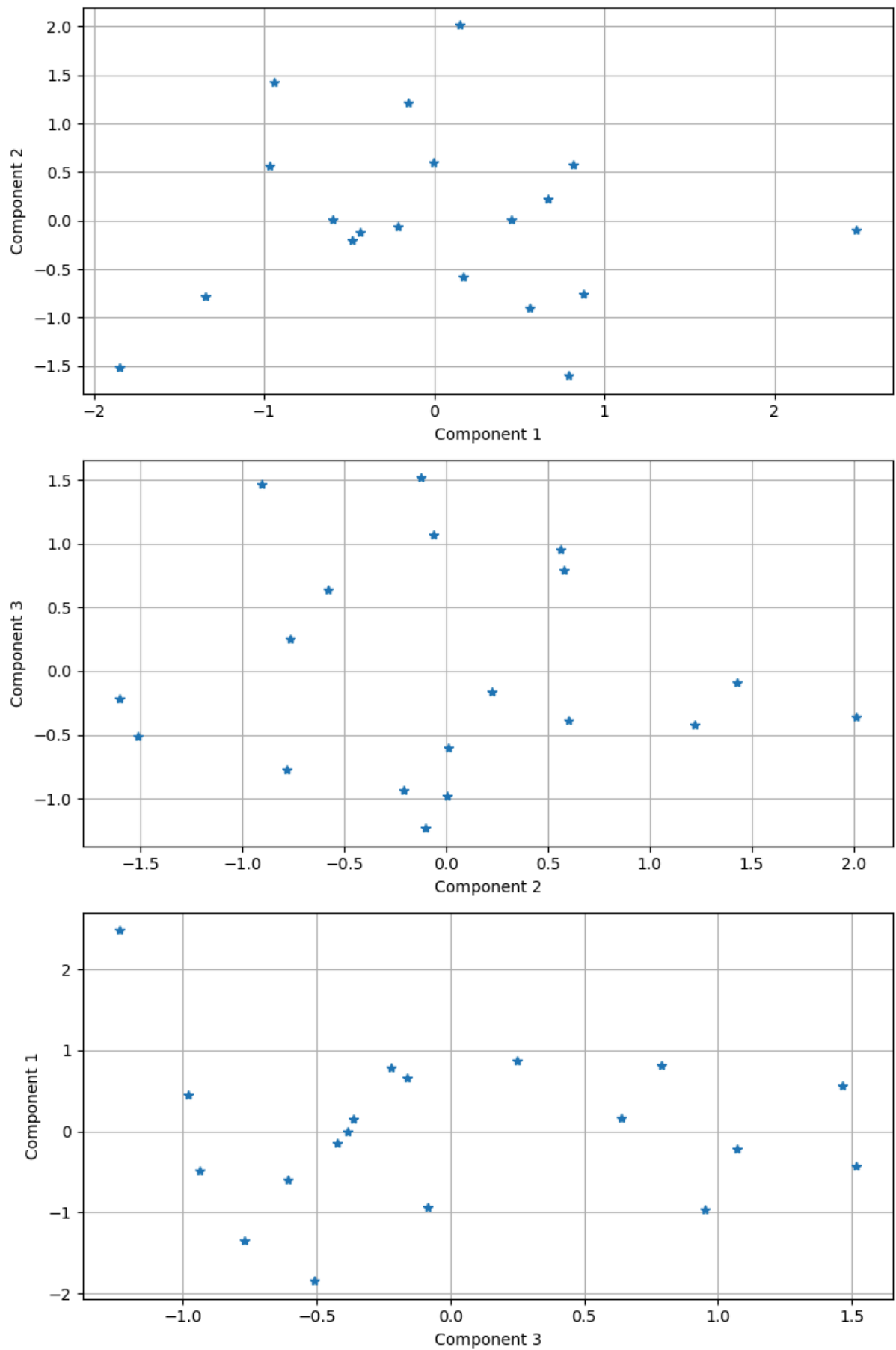


Figure 12: Factor scores from the factor analysis with rotation

4) Using the following curve:

$$x_1 = \frac{t \cos t}{1+t^2}, x_2 = \frac{t \sin t}{1+t^2}, x_3 = t, -2\pi \leq t \leq 2\pi,$$

a) Estimate the intrinsic dimensionality using the Pettis, Bailey, Jain, and Dubes algorithm (available in `idpettis.m`)

The curve was generated using the code listed in Listing 9 and the curve can be visualized in Figure 13.

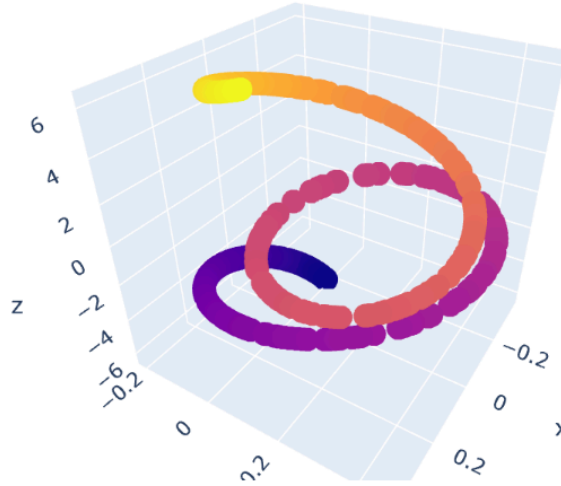


Figure 13: Curve generated

To calculate the intrinsic dimensionality estimate, `pyEDAKit.IntrinsicDimensionality.id_pettis` was used, resulting in

$$\text{IDE} \approx 1.119$$

b) Study the intrinsic dimensionality when introducing noise of various sizes to the curve.

For this exercise, to introduce noise of various sizes, we decided to generate and add random points to our plot, and record the intrinsic dimensionality every time a new random point was added. The random points were uniformly generated over the minimum and maximum of every dimension of our curve. The code for this process is listed in Listing 11, the curve with the final noise can be observed in Figure 14, and the Estimate Intrinsic Dimensionality plotted over the number of noise data points added can be observed in Figure 15

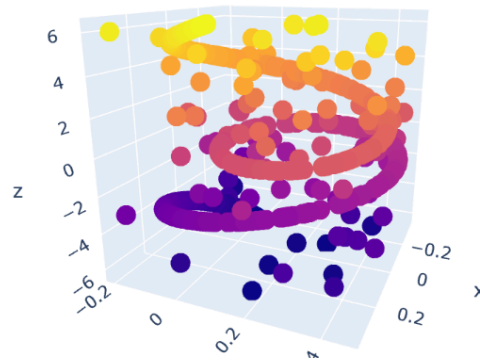


Figure 14: Curve with noise added

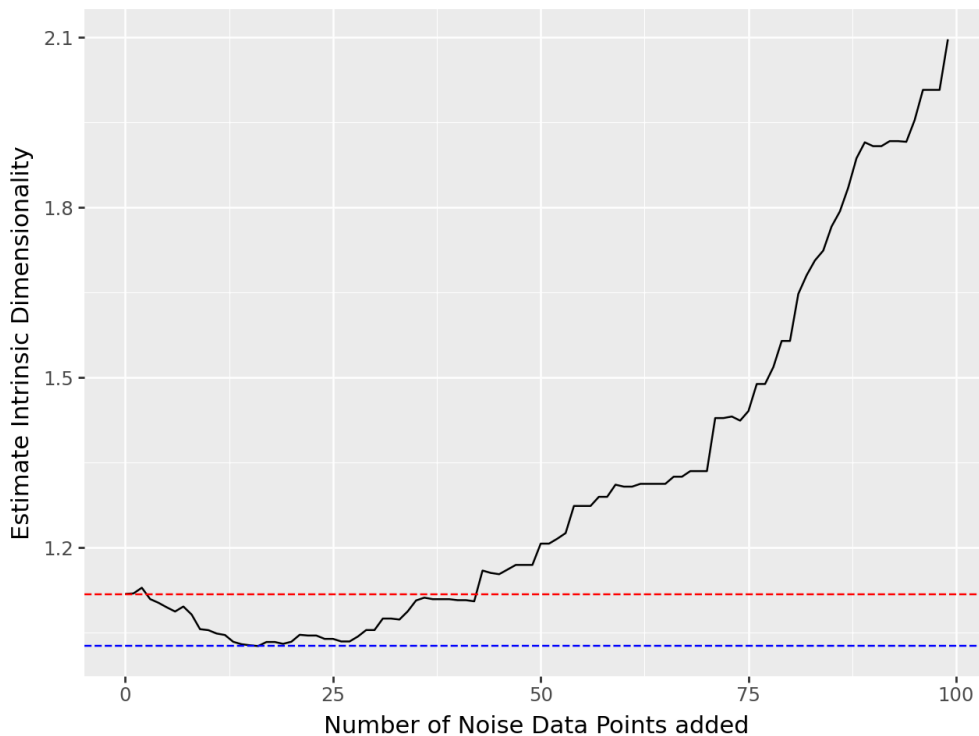


Figure 15: Intrinsic Dimensionality over noise size

For the first ≈ 12 noise data points, the intrinsic Dimensionality seems to be relatively low and going down, suggesting that the dataset has a well-defined structure; until it reaches a somewhat consistent value until ≈ 27 , where the values start increasing rapidly with the number of noise data points added. This suggests that beyond a certain point, as noise increases, the metric also increases, showing how noise disrupts rapidly disrupts the original structure.

c) Is there any threshold number of noise size for the intrinsic dimensionality estimate?

While it's difficult to define a strict threshold, for this particular curve, a noise level of approximately 40 data points can be considered a reasonable threshold.

5)

a) Apply the Singular value decomposition (SVD) to dataset Leukemia, choose a proper lower dim k via elbow in the plot of singular vaules, then plot the dimension reduced data in both 2-dim and 3-dim (in case your k is at least three).

With the elbow method we identified $k = 6$ as the optimal number of components for dimensionality reduction, as shown on Figure 16. Using this information, we then performed Singular Value Decomposition (SVD) and visualized the reduced-dimension data in both 2D and 3D, as shown in Figure 17 and 18.

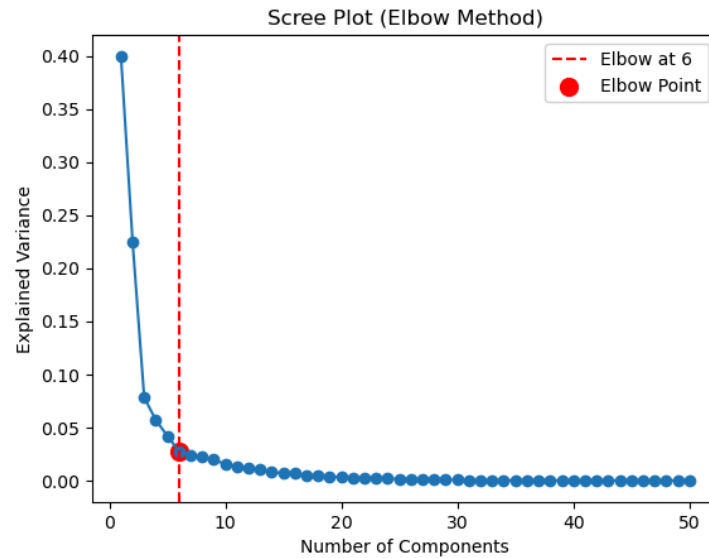
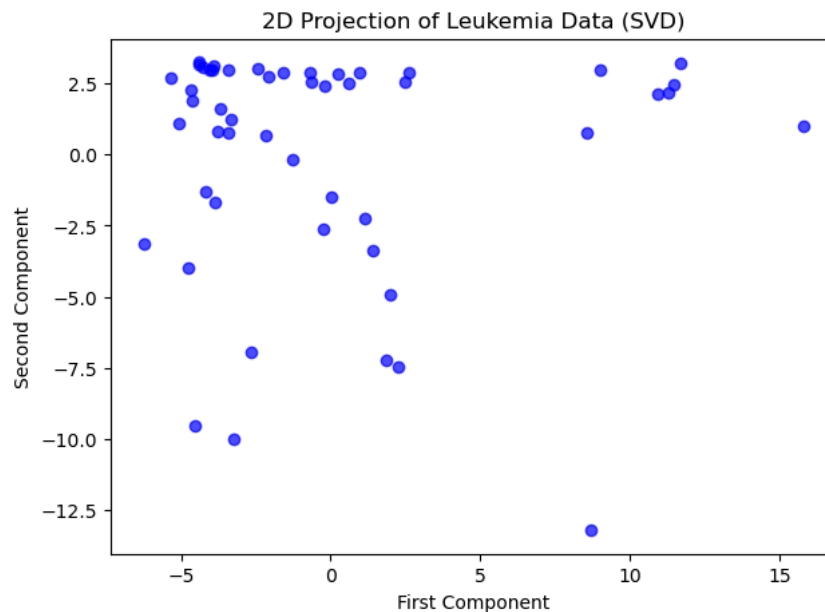


Figure 16: Elbow method.



3D Projection of Leukemia Data (SVD)

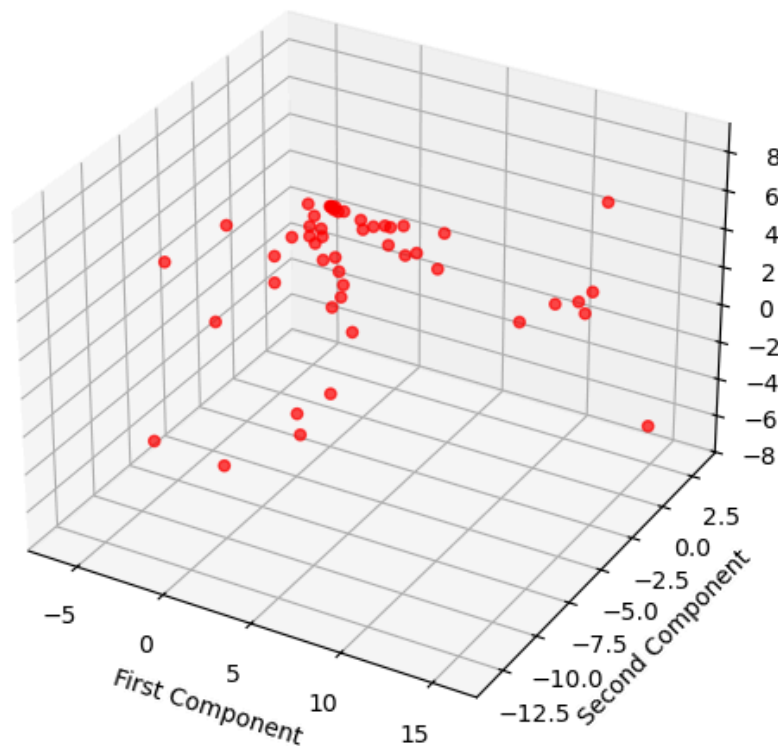


Figure 18: Singular Value Decomposition 3D visualization.

b) Try PCA (covariance) and compare the results from these two different methods.

From the figures below the results of Singular Value Decomposition (SVD) and Principal Component Analysis (PCA) applied to the same leukemia dataset can be compared. Both methods reveal similar distribution patterns with a significant difference in the range of the second component, where for SVD it spans from -13 to 3 , and for PCA from -3 to 13 . Additionally, the SVD results show a higher concentration of points in the upper half of the plot, while PCA points clustered mostly in the lower half. However, the overall shape and relative positioning of the points somehow appears consistent, indicating that both methods capture similar underlying patterns in the data.

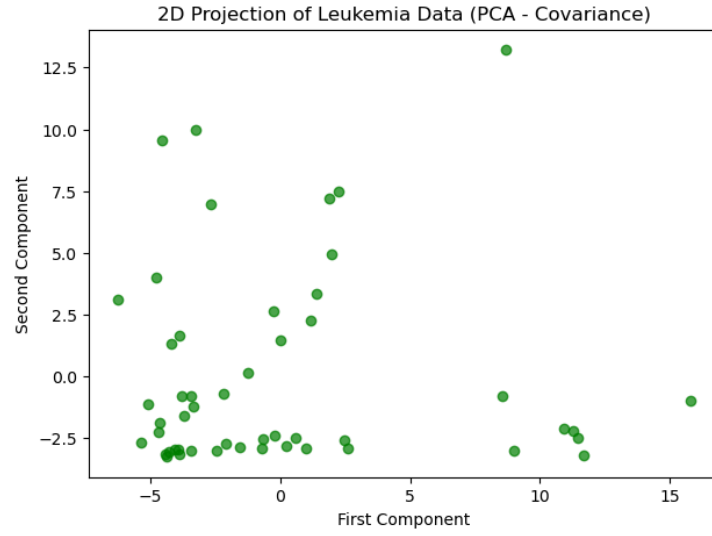


Figure 19: Singular Value Decomposition 3D visualization.

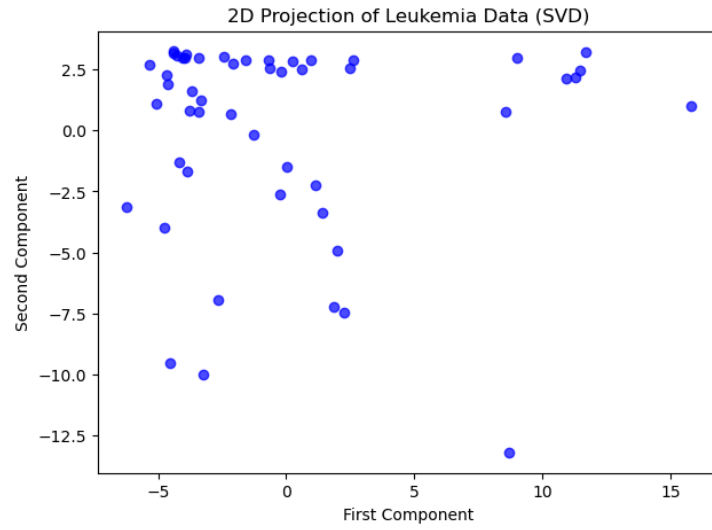


Figure 20: Singular Value Decomposition 2D visualization.

6) a) Using `genLDdata.m`, calculate the global and local intrinsic dimensionalities, using MLE and `CorrDim`, and compare the results. Use a neighborhood size of $k = 100$.

`genLDdata` generates random data points from the surface of a sphere, from a cube, and from 4 horizontal or vertical line segments coming out of the sphere. The global intrinsic dimensionality using MLE of the generated data was ≈ 1.52 , and using *CorrDim* was ≈ 1.00 . The local intrinsic dimensionality counts are presented in Table 1, and a scatter plot of these are available on Figure 21 and Figure 22.

Table 1: Local Intrinsic Dimensionality Counts

Dimension	Count (MLE)	Count (CorrDim)
1	2798 (46.63%)	2749 (45.82%)
2	1922 (32.03%)	2170 (36.17%)
3	1257 (20.95%)	1080 (18.00%)
4	23 (0.38%)	1 (0.02%)

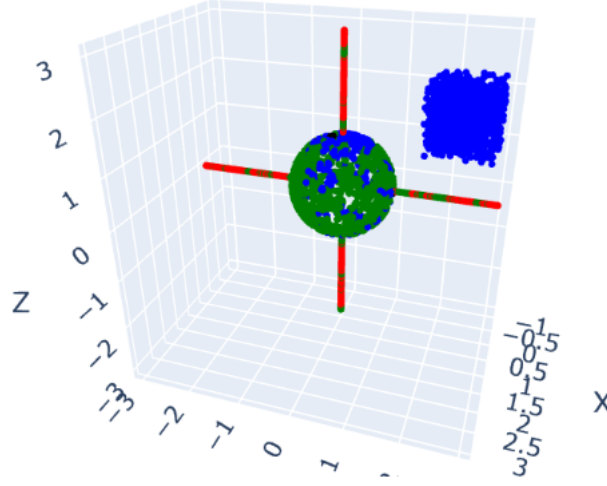


Figure 21: Local Intrinsic Dimensionality Scatter plot with MLE

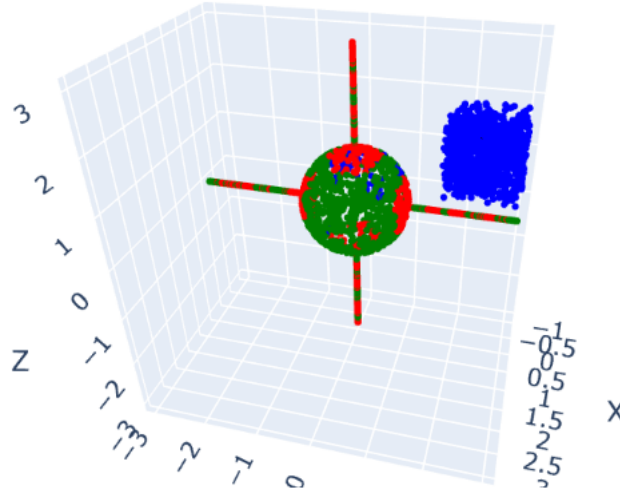


Figure 22: Local Intrinsic Dimensionality Scatter plot with CorrDim

From the figures, we can see that each shape, excluding the sphere surface, has a distinct local intrinsic dimensionality that matches our geometric intuition: the cube completely displays a local intrinsic dimensionality of 3 with both algorithms and the segments mostly displays a local intrinsic dimensionality of 1; however, since the sphere surface is a more complex surface, it seems the different algorithms classify it differently and not consistently: the output from the *MLE* presents the local intrinsic dimensionality of the sphere to be between 2 and 3, while the one with *CorrDim* presents it to be between 1 and

2 (the 1 points are closer to the connection between the segments and the sphere). Lastly, it seems the *MLE* could classify the segments more consistently than *CorrDim*.

b) Modify `genLDdata.m` so that:

- Instead of the sphere, we have the surface of an ellipsoid of $\frac{x_1^2}{4^2} + \frac{x_2^2}{5^2} + \frac{x_3^2}{6^2} = 1$,
- The line segments are replaced by the curve $x_1 = \frac{t \cos t^2}{1+t^2}, x_2 = \frac{t \sin t^2}{1+t^2}, x_3 = t, -2\pi \leq t \leq 2\pi$
- A line segment connecting a point from the ellipsoid and a point from the cube is added

Using this modified method, study the global and local intrinsic dimensionalities using *PackingNumbers*

The modified function can be found on Listing 17 and its output can be visualized on Figure 23, along with the output of the local intrinsic dimensionality with *PackingNumbers*. The global intrinsic dimensionality was ≈ 0.69

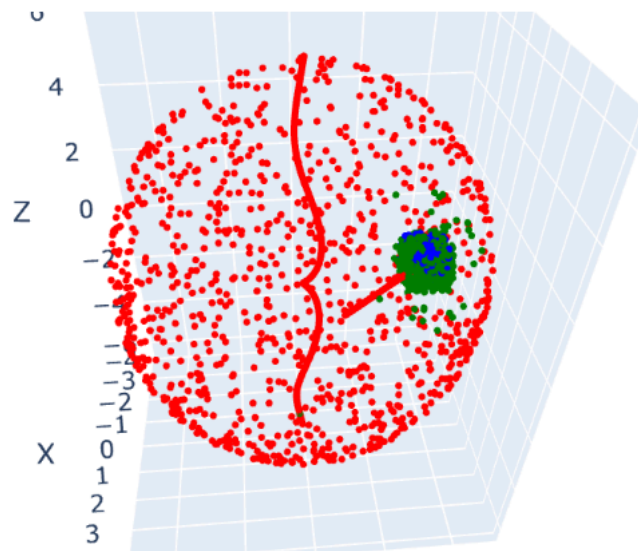



Figure 23: Local Intrinsic Dimensionality with *PackingNumbers*

We can see that the algorithm could not effectively find the expected local intrinsic dimensionality for any of our shapes, besides the connecting line between the cube and the sphere.

Appendix

1)

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pyEDAKit as eda_lin
4
5 np.random.seed(42)
6 x1 = np.random.randn(150, 2) * 100 # 2 dimensions with high variance
```

 Python

```

7  x2 = np.random.randn(150, 4) # 4 dimensions with low variance
8  X = np.hstack((x1, x2))
9
10 print("\nPCA using covariance matrix")
11 PCA_cov = eda_lin.PCA(X, d=3, covariance = True, plot=True)
12
13 print("\nPCA using correlation matrix")
14 PCA_corr = eda_lin.PCA(X, d=2, covariance = False, plot=True)


```

Listing 1: PCA using covariance vs correlation matrix for task 1

```

1  def PCA(X, d, covariance = True, plot = False):
2      X_mean = X.mean(axis=0)
3      X = X - X_mean
4      S = None
5      if covariance:
6          S = np.cov(X, rowvar=False)
7      else:
8          S = np.corrcoef(X, rowvar=False)
9      eigen_values, eigen_vectors = np.linalg.eig(S)
10     sorted_index = np.argsort(eigen_values)[::-1]
11     sorted_eigenvalue = eigen_values[sorted_index]
12     sorted_eigenvectors = eigen_vectors[:, sorted_index]
13
14     Z = (X @ sorted_eigenvectors)[:d, :d]
15
16     if plot:
17         plt.figure(figsize=(8, 5))
18         plt.plot(range(1, len(sorted_eigenvalue) + 1), sorted_eigenvalue,
19                 marker='o', linestyle='-')
20         plt.plot(d, sorted_eigenvalue[d - 1], 'ro', label = 'd')
21         plt.title('Scree Plot')
22         plt.xlabel('Eigenvalue Index')
23         plt.ylabel('Eigenvalue Magnitude')
24         plt.legend()
25         plt.grid(True)
26         # scatter matrix of Z
27         sns.pairplot(pd.DataFrame(Z, columns=[f"PC{i+1}" for i in
28         range(d)]), diag_kind='kde')
29         plt.legend()
30         plt.show()
31
32     return Z

```

 Python

Listing 2: PCA method for task 1

2)

a)

```
1  from plotnine import *
2  from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
3
4  plots = []
5  lda = LDA(n_components=1)
6  for target in iris_target_names:
7      # filter out target
8      iris_filtered = iris[iris["target"] != target]
9      Z = lda.fit_transform(iris_filtered.drop(columns="target"),
10                           iris_filtered["target"])
11      plot = (
12          ggplot(
13              pd.DataFrame({"Component": Z.flatten(), "target":
14                           iris_filtered["target"]}),
15              aes(x = 'Component', color = 'target', fill = "target"))
16      )
17      + geom_density(alpha = 0.6)
18      )
19      plots.append(plot)
20  plots[0]
21  plots[1]
22  plots[2]
```

Listing 3: LDA for case a

b)

```
1  import numpy as np
2
3  data_other_way = pd.DataFrame(
4      {
5          "Width": np.concatenate([iris["sepal width (cm)"], iris["petal width
6          (cm)"]]),
7          "Length": np.concatenate([iris["sepal length (cm)"], iris["petal
8          length (cm)"]]),
9          "Sepal_Or_Petal": np.concatenate([np.repeat("Sepal", len(iris)),
10                                             np.repeat("Petal", len(iris))])
11      }
12  )
```

```

10 Z = lda.fit_transform(data_other_way.drop(columns="Sepal_Or_Petal"),
11 data_other_way["Sepal_Or_Petal"])
12 (
13     ggplot(
14         pd.DataFrame({"Component": Z.flatten(), "Sepal_Or_Petal":
15 data_other_way["Sepal_Or_Petal"]}),
16         aes(x = 'Component', color = 'Sepal_Or_Petal', fill =
17 "Sepal_Or_Petal")
18     )
19     + geom_density(alpha = 0.6)
20 )

```

Listing 4: LDA for case *b*


3)

a)

```

1 stocks_df = pd.read_excel('../data/stockreturns.xlsx')
2 stocks = stocks_df.values # convert to numpy array
3
4 scaler = StandardScaler()
5 stocks_scaled = scaler.fit_transform(stocks)
6
7 # FA with rotation (default)
8 fa_default = FactorAnalysis(n_components=3, rotation='varimax')
9 fa_default.fit(stocks_scaled)
10 loadings_default = fa_default.components_.T
11
12 # FA NO rotation
13 fa_none = FactorAnalysis(n_components=3, rotation=None)
14 fa_none.fit(stocks_scaled)
15 loadings_none = fa_none.components_.T

```


 Python

Listing 5: Factor Analysis with and without rotation

```

1 lab = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10'] #
   labels
2 t = np.linspace(-1, 1, 20) # reference line vector
3
4 def plot_factor_loadings(loadings, title):
5     plt.figure(figsize=(8, 6))
6     plt.plot(loadings[:, 0], loadings[:, 1], '.')
7     plt.plot(t, np.zeros_like(t), 'b') # horizontal line
8     plt.plot(np.zeros_like(t), t, 'b') # vertical line
9

```

 Python

```

10     # add labels
11     for i, txt in enumerate(lab):
12         plt.annotate(txt, (loadings[i, 0] + 0.02, loadings[i, 1]))
13
14     plt.xlabel('Factor 1')
15     plt.ylabel('Factor 2')
16     plt.title(title, fontweight='bold')
17     plt.grid(True, linestyle='--', alpha=0.7)
18     plt.tight_layout()
19
20 plot_factor_loadings(loadings_default, 'Factor Analysis (Varimax)')
21 plot_factor_loadings(loadings_none, 'Factor Analysis (No rotation)')


```

Listing 6: Plot factor loadings method

```

1  fa_scores = fa_default.transform(stocks_scaled)
2  fig, axes = plt.subplots(3, 1, figsize=(8, 12))
3
4  axes[0].plot(fa_scores[:, 0], fa_scores[:, 1], '*')
5  axes[0].set_xlabel('Component 1')
6  axes[0].set_ylabel('Component 2')
7  axes[0].grid(True)
8
9  axes[1].plot(fa_scores[:, 1], fa_scores[:, 2], '*')
10 axes[1].set_xlabel('Component 2')
11 axes[1].set_ylabel('Component 3')
12 axes[1].grid(True)
13
14 axes[2].plot(fa_scores[:, 2], fa_scores[:, 0], '*')
15 axes[2].set_xlabel('Component 3')
16 axes[2].set_ylabel('Component 1')
17 axes[2].grid(True)
18
19 plt.tight_layout()
20 plt.show()

```

 Python


Listing 7: Factor scores for Factor Analysis with rotation

b)

```

1  data = pd.read_csv('../data/my_stock_returns.csv')
2
3  my_stocks_scaled = StandardScaler().fit_transform(data.values)
4  stock_names = data.columns.tolist()
5

```

 Python

```

6  # FA with varimax rotation
7  fa = FactorAnalysis(n_components=3, rotation='varimax')
8  fa.fit(my_stocks_scaled)
9  loadings = fa.components_.T
10
11 # Create 3D plot
12 fig = plt.figure(figsize=(10, 8))
13 ax = fig.add_subplot(111, projection='3d')
14
15 # Plot points
16 ax.scatter(loadings[:, 0], loadings[:, 1], loadings[:, 2], s=50)
17 for i in range(len(loadings)):
18     ax.text(loadings[i, 0], loadings[i, 1], loadings[i, 2],
19            stock_names[i], size=8)
19
20 for spine in ['xy', 'xz', 'yz']:
21     ax.grid(True, ls='--', alpha=0.3, which='major', zorder=0)
22
23 ax.set_xlabel('Factor 1')
24 ax.set_ylabel('Factor 2')
25 ax.set_zlabel('Factor 3')
26 ax.set_title('3D Factor Analysis loadings plot')
27
28 # subplots for component visualization
29 fa_scores = fa.transform(my_stocks_scaled)
30 fig, axes = plt.subplots(3, 1, figsize=(8, 12))
31
32 axes[0].plot(fa_scores[:, 0], fa_scores[:, 1], '*')
33 axes[0].set_xlabel('Component 1')
34 axes[0].set_ylabel('Component 2')
35 axes[0].grid(True)
36
37 axes[1].plot(fa_scores[:, 1], fa_scores[:, 2], '*')
38 axes[1].set_xlabel('Component 2')
39 axes[1].set_ylabel('Component 3')
40 axes[1].grid(True)
41
42 axes[2].plot(fa_scores[:, 2], fa_scores[:, 0], '*')
43 axes[2].set_xlabel('Component 3')
44 axes[2].set_ylabel('Component 1')
45 axes[2].grid(True)
46

```

```

47 plt.tight_layout()
48 plt.show()

```

Listing 8: Factor analysis of collected stock data.


4)

a)

```

1  import numpy as np
2  import pandas as pd
3  import plotly.express as px
4  np.random.seed(1)
5
6  n = 500
7  theta = np.random.uniform(-2*np.pi, 2*np.pi, n)
8  x = (theta*np.cos(theta))/(1+theta**2)
9  y = (theta*np.sin(theta))/(1+theta**2)
10 z = theta
11 df = pd.DataFrame({'x':x, 'y':y, 'z':z})
12
13 fig = px.scatter_3d(df, x='x', y='y', z='z', color='z')
14 fig.show()

```


 Python

Listing 9: generating and plotting curve

```

1  from pyEDAKit.IntrinsicDimensionality import id_pettis
2  id_pettis(df)

```

 Python


Listing 10: Estimate of intrinsic dimensionality

b)

```

1  iterations = 100
2  n_generated = 100
3
4  noise_list = [[0, id_pettis(df)]]
5  df_noised = df.copy()
6  np.random.seed(1)
7  for i in range(1, iterations):
8      x = np.random.uniform(min_max['x']['min'], min_max['x']['max'],
9                             n_generated//iterations)
9      y = np.random.uniform(min_max['y']['min'], min_max['y']['max'],
10                             n_generated//iterations)
10     z = np.random.uniform(min_max['z']['min'], min_max['z']['max'],
11                             n_generated//iterations)
11

```

 Python


```

12     df_noised = pd.concat([df_noised, pd.DataFrame({'x': x, 'y': y, 'z':
13         z})], ignore_index=True)
14     noise_list.append([(n_generated//iterations)*i, id_pettis(df_noised)])
15 noise_df = pd.DataFrame(noise_list, columns=['n', 'eid'])
16
17 fig = px.scatter_3d(df_noised, x='x', y='y', z='z', color='z')
18 fig.show()
19
20 from plotnine import *
21 (
22     ggplot(noise_df, aes(x='n', y='eid'))
23     + geom_line()
24     + geom_hline(yintercept=eid, linetype='dashed', color='red')
25     + geom_hline(yintercept=min(noise_df['eid']), linetype='dashed',
26         color='blue')
27     + labs(x='Number of Noise Data Points added', y='Estimate Intrinsic
28         Dimensionality')
29 )

```

Listing 11: Noise generated and plotted


5)

a)

```

1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.io import loadmat
4  from mpl_toolkits.mplot3d import Axes3D
5  from sklearn.preprocessing import StandardScaler
6  from kneed import KneeLocator
7
8  data = loadmat('../data/leukemia.mat')
9  X = data["leukemia"]
10 print("Dataset shape:", X.shape)
11
12 X_scaled = StandardScaler().fit_transform(X)
13
14 U, S, Vt = np.linalg.svd(X_scaled, full_matrices=False)
15
16 explained_variance = S**2 / np.sum(S**2) # S contains singular values
17
18 plt.plot(range(1, len(S) + 1), explained_variance, marker='o',
19     linestyle='-')
20 plt.xlabel("Number of Components")

```

 Python

```

20 plt.ylabel("Explained Variance")
21 plt.title("Scree Plot (Elbow Method)")
22
23 # find k
24 knee_locator = KneeLocator(range(1, len(S) + 1), explained_variance,
25                             curve="convex", direction="decreasing")
26 elbow_point = knee_locator.knee
27
28 # highlight elbow point
29 plt.axvline(x=elbow_point, color='r', linestyle='--', label=f'Elbow at
30 {elbow_point}')
31 plt.scatter(elbow_point, explained_variance[elbow_point - 1],
32             color='red', s=100, label='Elbow Point')
33 plt.legend()
34 plt.show()
35
36 # 2D plot
37 plt.figure(figsize=(7, 5))
38 plt.scatter(X_reduced_svd[:, 0], X_reduced_svd[:, 1], c='b', alpha=0.7)
39 plt.xlabel('First Component')
40 plt.ylabel('Second Component')
41 plt.title('2D Projection of Leukemia Data (SVD)')
42 plt.show()
43
44 # 3D plot
45 fig = plt.figure(figsize=(8, 6))
46 ax = fig.add_subplot(111, projection='3d')
47 ax.scatter(X_reduced_svd[:, 0], X_reduced_svd[:, 1], X_reduced_svd[:, 2],
48           c='r', alpha=0.7)
49 ax.set_xlabel('First Component')
50 ax.set_ylabel('Second Component')
51 ax.set_zlabel('Third Component')
52 ax.set_title('3D Projection of Leukemia Data (SVD)')
53 plt.show()

```


Listing 12: Singular value decomposition on leukemia dataset.

b)

```

1 import pyEDAKit as eda_lin
2

```

 Python

```

3 X_reduced_pca_cov = eda_lin.PCA(X_scaled, d=6, covariance=True,
  plot=False)
4
5 # 2D plot
6 plt.figure(figsize=(7, 5))
7 plt.scatter(X_reduced_pca_cov[:, 0], X_reduced_pca_cov[:, 1], c='g',
  alpha=0.7)
8 plt.xlabel('First Component')
9 plt.ylabel('Second Component')
10 plt.title('2D Projection of Leukemia Data (PCA - Covariance)')
11 plt.show()

```


Listing 13: Singular value decomposition on leukemia dataset.

6)

```

1 import numpy as np
2 import pyEDAKit as kit
3
4 def genLDdata(seed: int | None = 1):
5     if seed is not None:
6         np.random.seed(seed)
7         # Sample from the surface of a sphere
8         X1, X2, X3 = np.random.rand(3, 1000)
9         lambda_ = np.sqrt(X1**2 + X2**2 + X3**2)
10        X1, X2, X3 = X1 / lambda_, X2 / lambda_, X3 / lambda_
11        X = np.column_stack((X1, X2, X3))
12
13        # Sample from a cube
14        X1, X2, X3 = np.random.rand(3, 1000) + 2
15        XX = np.column_stack((X1, X2, X3))
16
17        # Sample from lines attached to a sphere
18        L1 = np.column_stack((np.zeros(1000), np.zeros(1000), 2 *
  np.random.rand(1000) + 1))
19        L2 = np.column_stack((np.zeros(1000), np.zeros(1000), -2 *
  np.random.rand(1000) - 1))
20        L3 = np.column_stack((np.zeros(1000), 2 * np.random.rand(1000) + 1,
  np.zeros(1000)))
21        L4 = np.column_stack((np.zeros(1000), -2 * np.random.rand(1000) - 1,
  np.zeros(1000)))
22
23        A = np.vstack((X, XX, L1, L2, L3, L4))
24        return A
25 A = genLDdata()

```

 Python

```
26 print(kit.IntrinsicDimensionality.MLE(A))
```


Listing 14: genLDdata function in python

```
1 import numpy as np
2 import pandas as pd
3 import scipy.spatial.distance as dist
4
5 def compute_local_intrinsic_dimensionality(A, method, k=100):
6     # Compute pairwise Euclidean distances
7     Ad = dist.squareform(dist.pdist(A))
8
9     # Get the dimensions of A
10    nr, nc = A.shape
11    Ldim = np.zeros(nr)
12
13    # Sort distances and get indices
14    Ads = np.sort(Ad, axis=1)
15    J = np.argsort(Ad, axis=1)
16
17    # Compute local intrinsic dimensionality
18    for m in range(nr):
19        Ldim[m] = method(A[J[m, :k], :])
20
21    # Adjust local dimensions
22    Ldim[Ldim > 3] = 4
23    Ldim = np.ceil(Ldim).astype(int)
24
25    # Tabulate results
26    unique, counts = np.unique(Ldim, return_counts=True)
27    percentages = (counts / nr) * 100
28    tabulation_df = pd.DataFrame({'Dimension': unique, 'Count': counts,
29    'Percentage': np.round(percentages, 3)})
30
31    return Ldim, tabulation_df
```

 Python

Listing 15: Local intrinsic calculation function (generated from *Example 2.10*)

```
1 import plotly.graph_objects as go
2 # Scatter plot with color map
3 colors = {1: 'red', 2: 'green', 3: 'blue', 4: 'black'}
4 fig = go.Figure()
5 labels = [1, 2, 3, 4]
6 for label in labels:
```

 Python

```

7     indices = np.where(Ldim == label)[0]
8     if len(indices) > 0:
9         fig.add_trace(go.Scatter3d(
10             x=A[indices, 0], y=A[indices, 1], z=A[indices, 2],
11             mode='markers',
12             marker=dict(color=colors[label], size=2),
13             name=f"Dim {label}"
14         ))
15
16 fig.update_layout(scene=dict(xaxis_title='X', yaxis_title='Y',
17                               zaxis_title='Z'), title="Intrinsic Dimensionality Scatterplot with [x]")
17 fig.show()


```

Listing 16: Local Intrinsic Dimensionality scatter plot

```

1  def genLDdata_mod(seed: int | None = 1):
2      if seed is not None:
3          np.random.seed(seed)
4
5      # Sample from the surface of an ellipsoid
6      X1, X2, X3 = np.random.randn(3, 1000)
7      lambda_ = np.sqrt((X1/4)**2 + (X2/5)**2 + (X3/6)**2)
8      X1, X2, X3 = X1 / lambda_, X2 / lambda_, X3 / lambda_
9      X = np.column_stack((X1, X2, X3))
10
11     # Sample from a cube
12     X1, X2, X3 = np.random.rand(3, 1000) + 2
13     XX = np.column_stack((X1, X2, X3))
14
15     # Sample of the curve
16     t = np.random.uniform(-2*np.pi, 2*np.pi, 1000)
17     X1 = (t * np.cos(t))/(1 + t**2)
18     X2 = (t * np.sin(t))/(1 + t**2)
19     X3 = t
20     X_curve = np.column_stack((X1, X2, X3))
21
22     # Sample from segment connecting ellipsoid and the cube
23     point1 = X[np.random.randint(len(X))]
24     point2 = XX[np.random.randint(len(XX))]
25     t = np.linspace(0, 1, 1000)
26     # linear interpolation
27     X1 = point1[0] + t * (point2[0] - point1[0])
28     X2 = point1[1] + t * (point2[1] - point1[1])

```

 Python

```

29 X3 = point1[2] + t * (point2[2] - point1[2])
30 X_segment = np.column_stack((X1, X2, X3))
31
32 A = np.vstack((X, XX, X_curve, X_segment))
33 return A


```

Listing 17: genLDdata modified

```

1 A = genLDdata()
2
3 print("global intrinsic dimensionality (MLE):",
4 kit.IntrinsicDimensionality.MLE(A))
5 Ldim, tabulation_df = compute_local_intrinsic_dimensionality(A,
6 kit.IntrinsicDimensionality.MLE)
7 print(tabulation_df) # Ldim used for the plotting
8
9 print("global intrinsic dimensionality (CorrDim):",
10 kit.IntrinsicDimensionality.corr_dim(A))
11 Ldim, tabulation_df = compute_local_intrinsic_dimensionality(A,
12 kit.IntrinsicDimensionality.corr_dim)
13 print(tabulation_df)
14
15 A = genLDdata_mod()
16 print("global intrinsic dimensionality (PackingNumbers):",
17 kit.IntrinsicDimensionality.packing_numbers(A))
18 Ldim, tabulation_df = compute_local_intrinsic_dimensionality(A,
19 kit.IntrinsicDimensionality.packing_numbers)

```

 Python

Listing 18: Function calls for 6)