cytoautocluster-1

November 14, 2024

Load csv

This code imports the pandas library and loads a CSV file (Levine_32dim.fcs.csv) into a DataFrame called df.

[1]:
```
import pandas as pd
df = pd.read_csv('/content/drive/My Drive/Levine_32dim.fcs.csv')
df
```

[1]:

	Event	Time	Cell_length	DNA1	DNA2	CD45RA	\
0	1	2693.00	22	4.391057	4.617262	0.162691	
1	2	3736.00	35	4.340481	4.816692	0.701349	
2	3	7015.00	32	3.838727	4.386369	0.603568	
3	4	7099.00	29	4.255806	4.830048	0.433747	
4	5	7700.00	25	3.976909	4.506433	-0.008809	
…	…	…	…	…	…	…	
265622	265623	707951.44	41	6.826629	7.133022	1.474081	
265623	265624	708145.44	45	6.787791	7.154026	0.116755	
265624	265625	708398.44	41	6.889866	7.141219	0.684921	
265625	265626	708585.44	39	6.865218	7.144353	0.288761	
265626	265627	709122.44	41	6.887820	7.127359	0.360753	

	CD133	CD19	CD22	CD11b	…	CD117	CD49d	\
0	-0.029585	-0.006696	0.066388	-0.009184	…	0.053050	0.853505	
1	-0.038280	-0.016654	0.074409	0.808031	…	0.089660	0.197818	
2	-0.032216	0.073855	-0.042977	-0.001881	…	0.046222	2.586670	
3	-0.027611	-0.017661	-0.044072	0.733698	…	0.066470	1.338669	
4	-0.030297	0.080423	0.495791	1.107627	…	-0.006223	0.180924	
…	…	…	…	…	…	…	…	
265622	-0.019174	-0.055620	-0.007261	0.063395	…	-0.011105	0.533736	
265623	-0.056213	-0.008864	-0.035158	-0.041845	…	0.143869	1.269464	
265624	-0.006264	-0.026111	-0.030837	-0.034641	…	0.087102	-0.055912	
265625	-0.011310	-0.048786	0.073983	-0.031787	…	-0.047971	0.101955	
265626	0.128604	-0.006934	0.109846	3.864711	…	0.080195	0.037962	

	HLA-DR	CD64	CD41	Viability	file_number	event_number	\
0	1.664480	-0.005376	-0.001961	0.648429	3.627711	307	
1	0.491592	0.144814	0.868014	0.561384	3.627711	545	
2	1.308337	-0.010961	-0.010413	0.643337	3.627711	1726	

```
3       0.140523 -0.013449 -0.026039  -0.026523     3.627711          1766
4       0.197332  0.076167 -0.040488   0.283287     3.627711          2031
…            …        …        …         …            …              …
265622  0.123758 -0.042495 -0.027971   0.236957     3.669327        102686
265623  0.047215 -0.008000 -0.025811  -0.003500     3.669327        102690
265624  0.501536  0.053884 -0.042602   0.107206     3.669327        102701
265625  6.200001  0.296877  0.192786   0.620872     3.669327        102706
265626  3.675123 -0.000878 -0.052526   0.310466     3.669327        102720

        label  individual
0         1.0           1
1         1.0           1
2         1.0           1
3         1.0           1
4         1.0           1
…         …             …
265622    NaN           2
265623    NaN           2
265624    NaN           2
265625    NaN           2
265626    NaN           2

[265627 rows x 42 columns]
```

This line retrieves the column names of the DataFrame df.

[ ]:
```
df.columns
```

[ ]:
```
Index(['Event', 'Time', 'Cell_length', 'DNA1', 'DNA2', 'CD45RA', 'CD133',
       'CD19', 'CD22', 'CD11b', 'CD4', 'CD8', 'CD34', 'Flt3', 'CD20', 'CXCR4',
       'CD235ab', 'CD45', 'CD123', 'CD321', 'CD14', 'CD33', 'CD47', 'CD11c',
       'CD7', 'CD15', 'CD16', 'CD44', 'CD38', 'CD13', 'CD3', 'CD61', 'CD117',
       'CD49d', 'HLA-DR', 'CD64', 'CD41', 'Viability', 'file_number',
       'event_number', 'label', 'individual'],
      dtype='object')
```

This code accesses the 'Viability' column of the DataFrame.

[ ]:
```
df['Viability']
```

[ ]:
```
0         0.648429
1         0.561384
2         0.643337
3        -0.026523
4         0.283287
            …
265622    0.236957
265623   -0.003500
```

```
265624    0.107206
265625    0.620872
265626    0.310466
Name: Viability, Length: 265627, dtype: float64
```

Null and Non null values

The code calculates and prints the number of null and non-null values for each column in the DataFrame.

[ ]:
```
null_values = df.isnull().sum()
non_null_values = df.notnull().sum()
print("Null values in each column:")
print(null_values)
print("\nNon-null values in each column:")
print(non_null_values)
```

```
Null values in each column:
Event                0
Time                 0
Cell_length          0
DNA1                 0
DNA2                 0
CD45RA               0
CD133                0
CD19                 0
CD22                 0
CD11b                0
CD4                  0
CD8                  0
CD34                 0
Flt3                 0
CD20                 0
CXCR4                0
CD235ab              0
CD45                 0
CD123                0
CD321                0
CD14                 0
CD33                 0
CD47                 0
CD11c                0
CD7                  0
CD15                 0
CD16                 0
CD44                 0
CD38                 0
CD13                 0
```

```
CD3                 0
CD61                0
CD117               0
CD49d               0
HLA-DR              0
CD64                0
CD41                0
Viability           0
file_number         0
event_number        0
label          161443
individual          0
dtype: int64

Non-null values in each column:
Event          265627
Time           265627
Cell_length    265627
DNA1           265627
DNA2           265627
CD45RA         265627
CD133          265627
CD19           265627
CD22           265627
CD11b          265627
CD4            265627
CD8            265627
CD34           265627
Flt3           265627
CD20           265627
CXCR4          265627
CD235ab        265627
CD45           265627
CD123          265627
CD321          265627
CD14           265627
CD33           265627
CD47           265627
CD11c          265627
CD7            265627
CD15           265627
CD16           265627
CD44           265627
CD38           265627
CD13           265627
CD3            265627
CD61           265627
CD117          265627
```

```
CD49d           265627
HLA-DR          265627
CD64            265627
CD41            265627
Viability       265627
file_number     265627
event_number    265627
label           104184
individual      265627
dtype: int64
```

This block identifies columns and rows that contain null values.

[ ]:
```
columns_with_null = df.columns[df.isnull().any()]
rows_with_null_values = df[columns_with_null][df[columns_with_null].isnull().
  ↪any(axis=1)]
rows_with_null_values
```

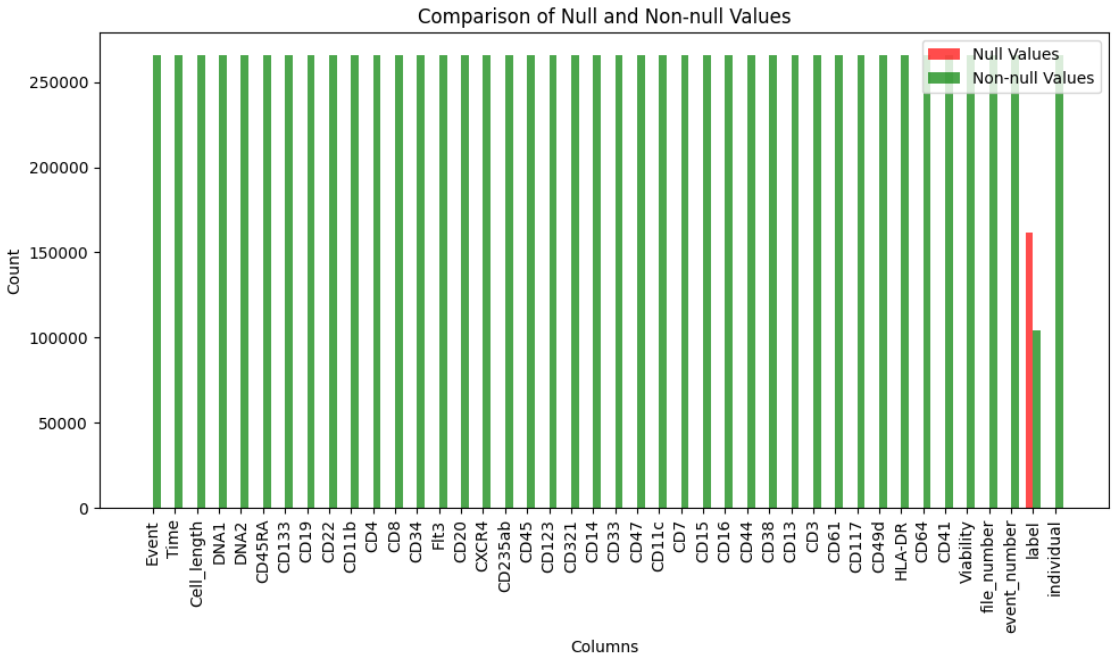
[ ]:
```
        label
104184    NaN
104185    NaN
104186    NaN
104187    NaN
104188    NaN
…         …
265622    NaN
265623    NaN
265624    NaN
265625    NaN
265626    NaN

[161443 rows x 1 columns]
```

Comparison between null and non null values

This code visualizes the comparison between null and non-null values in each column using a bar chart.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

null_counts = df.isnull().sum()
non_null_counts = df.notnull().sum()

bar_width = 0.35
index = np.arange(len(df.columns))

plt.figure(figsize=(10,6))
```

```
plt.bar(index, null_counts, bar_width, label='Null Values', color='red',␣
 ↪alpha=0.7)
plt.bar(index + bar_width, non_null_counts, bar_width, label='Non-null Values',␣
 ↪color='green', alpha=0.7)

plt.xlabel('Columns')
plt.ylabel('Count')
plt.title('Comparison of Null and Non-null Values')
plt.xticks(index + bar_width / 2, df.columns, rotation=90)

plt.legend()
plt.tight_layout()
plt.show()
```

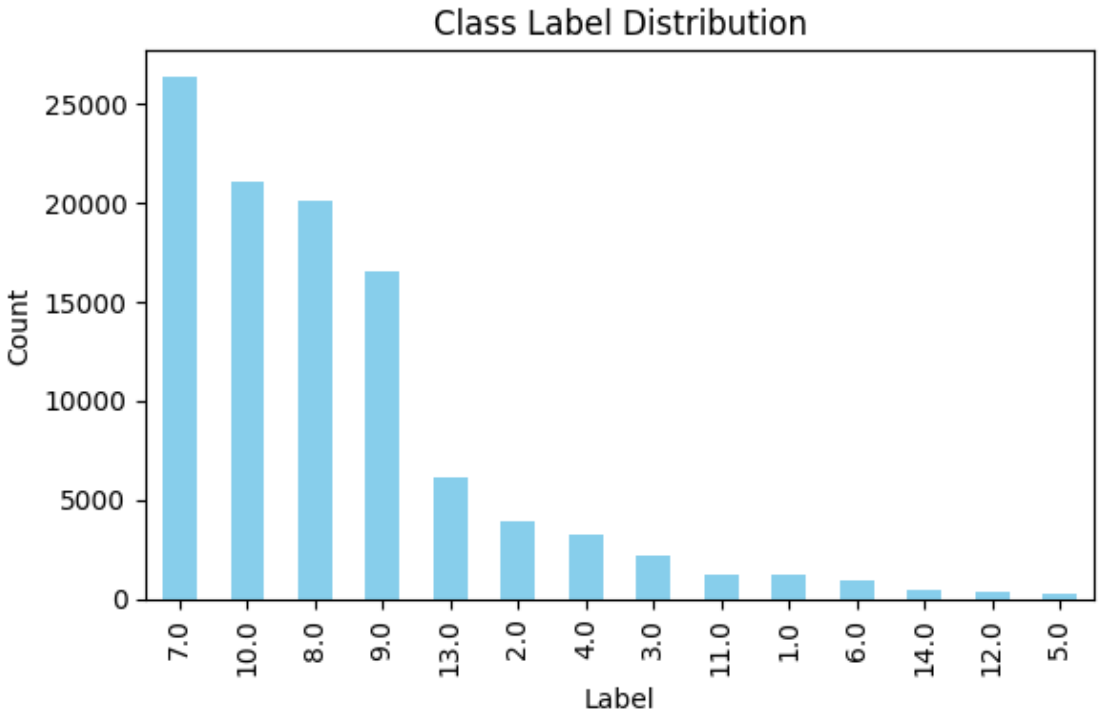


Class label distribution

This code plots the distribution of the 'label' column, showing how many instances of each class are present.

[ ]:
```
label_distribution = df['label'].value_counts()

plt.figure(figsize=(6,4))
label_distribution.plot(kind='bar', color='skyblue')
plt.title('Class Label Distribution')
plt.xlabel('Label')
plt.ylabel('Count')
```

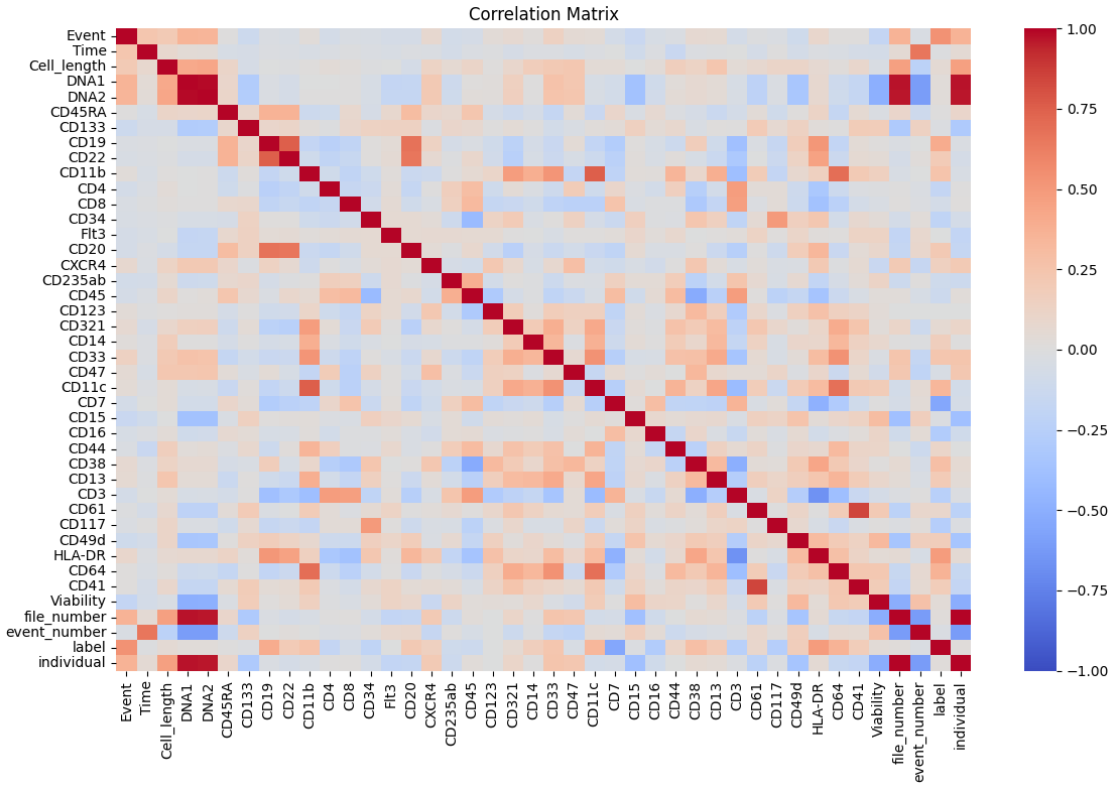
```
plt.tight_layout()
plt.show()
```



Correlation matrix

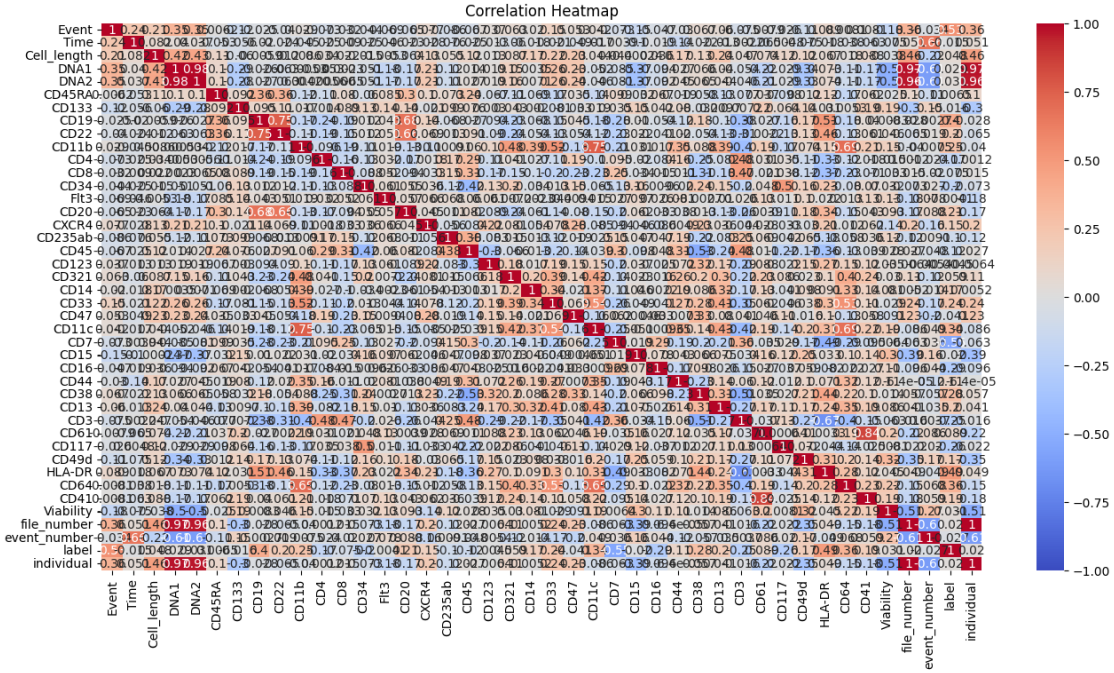
This code calculates and visualizes the correlation matrix for numeric columns using a heatmap.

[ ]:
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
corr_matrix = df[numeric_columns].corr()

plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix')
plt.tight_layout()
plt.show()
```



This block visualizes the correlation matrix again, but with annotations showing the exact correlation values.

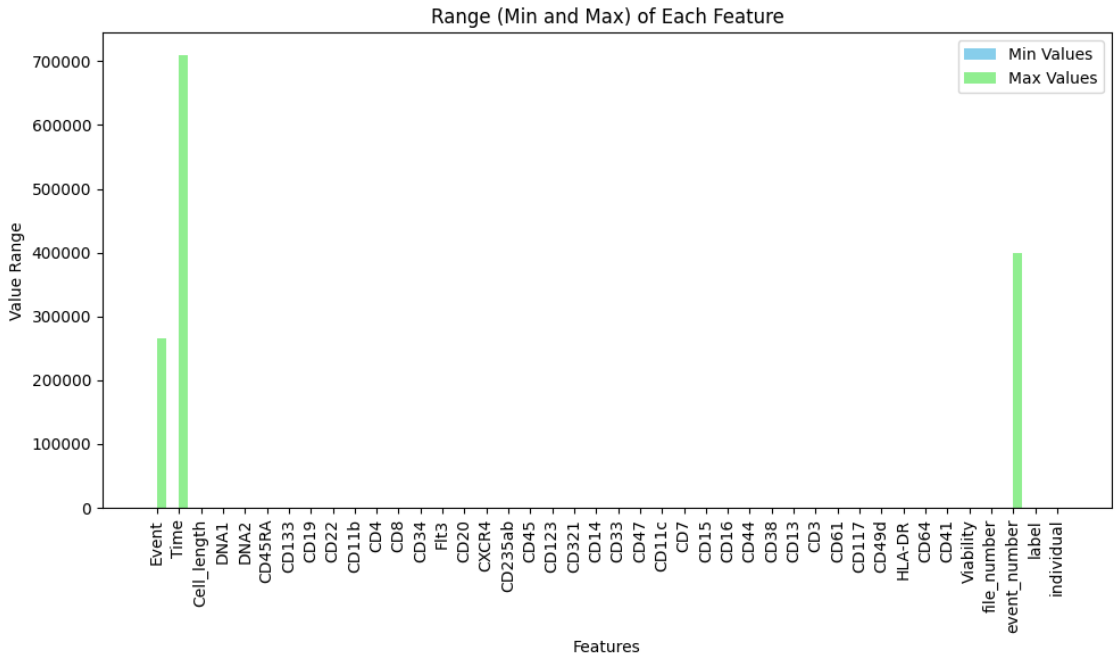
[ ]:
```
plt.figure(figsize=(16,8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap')
plt.show()
```



Range of each feature

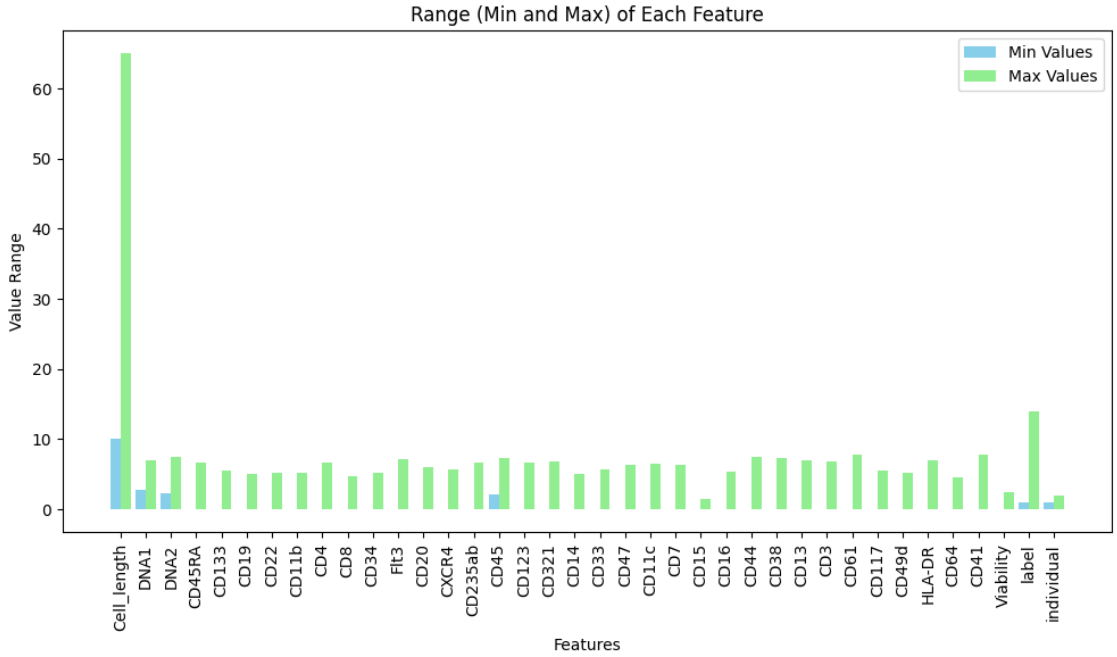
This code plots the minimum and maximum values of each numeric feature using a bar chart.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

min_values = df[numeric_columns].min()
max_values = df[numeric_columns].max()

index = np.arange(len(numeric_columns))

plt.figure(figsize=(10,6))

plt.bar(index - 0.2, min_values, 0.4, label='Min Values', color='skyblue')
plt.bar(index + 0.2, max_values, 0.4, label='Max Values', color='lightgreen')

plt.xlabel('Features')
plt.ylabel('Value Range')
plt.title('Range (Min and Max) of Each Feature')
plt.xticks(index, numeric_columns, rotation=90)

plt.legend()
plt.tight_layout()
plt.show()
```



Range after removing 'file_number', 'Event', 'Time', 'event_number'

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt

filtered_df = df.drop(columns=['file_number', 'Event', 'Time', 'event_number'])
numeric_columns = filtered_df.select_dtypes(include=['float64', 'int64']).
 ↪columns

min_values = filtered_df[numeric_columns].min()
max_values = filtered_df[numeric_columns].max()

index = np.arange(len(numeric_columns))

plt.figure(figsize=(10, 6))
plt.bar(index - 0.2, min_values, 0.4, label='Min Values', color='skyblue')
plt.bar(index + 0.2, max_values, 0.4, label='Max Values', color='lightgreen')

plt.xlabel('Features')
plt.ylabel('Value Range')
plt.title('Range (Min and Max) of Each Feature')
plt.xticks(index, numeric_columns, rotation=90)

plt.legend()
plt.tight_layout()
plt.show()
```



This code prints the minimum and maximum values for each numeric feature in the DataFrame. It gives a textual overview of the range of values

[ ]:
```
min_values = df[numeric_columns].min()
max_values = df[numeric_columns].max()

print("Range of Each Feature:")
for col in numeric_columns:
    print(f"{col}: Min = {min_values[col]}, Max = {max_values[col]}")
```

```
Range of Each Feature:
Cell_length: Min = 10.0, Max = 65.0
DNA1: Min = 2.7864876, Max = 7.001489
DNA2: Min = 2.2364502, Max = 7.472308
CD45RA: Min = -0.057305153, Max = 6.691197
CD133: Min = -0.05808065, Max = 5.5274944
CD19: Min = -0.058088884, Max = 4.990085
CD22: Min = -0.05734217, Max = 5.160477
CD11b: Min = -0.05823572, Max = 5.2607894
CD4: Min = -0.057751145, Max = 6.581762
CD8: Min = -0.05800326, Max = 4.693694
CD34: Min = -0.058008093, Max = 5.1479964
Flt3: Min = -0.057884354, Max = 7.117323
CD20: Min = -0.05813245, Max = 6.051411
CXCR4: Min = -0.057042465, Max = 5.6966743
CD235ab: Min = -0.057612002, Max = 6.646699
```

```
CD45: Min = 2.0402431, Max = 7.238076
CD123: Min = -0.058003303, Max = 6.6406264
CD321: Min = -0.053551525, Max = 6.8673882
CD14: Min = -0.057954386, Max = 5.0061207
CD33: Min = -0.058079027, Max = 5.6124687
CD47: Min = -0.055087358, Max = 6.402488
CD11c: Min = -0.05805277, Max = 6.520939
CD7: Min = -0.058161568, Max = 6.319219
CD15: Min = -0.058076803, Max = 1.5341506
CD16: Min = -0.057779938, Max = 5.338305
CD44: Min = 0.026061073, Max = 7.4045644
CD38: Min = -0.057193976, Max = 7.293085
CD13: Min = -0.057727765, Max = 6.9811873
CD3: Min = -0.058240842, Max = 6.7483625
CD61: Min = -0.05764178, Max = 7.7484975
CD117: Min = -0.05766792, Max = 5.502125
CD49d: Min = -0.058063813, Max = 5.153438
HLA-DR: Min = -0.05797395, Max = 7.052507
CD64: Min = -0.058199227, Max = 4.517843
CD41: Min = -0.058243938, Max = 7.7182884
Viability: Min = -0.0579794, Max = 2.4330306
label: Min = 1.0, Max = 14.0
individual: Min = 1.0, Max = 2.0
```

Box plot

This line generates and displays summary statistics (like mean, std, min, max) for the DataFrame's numeric columns.

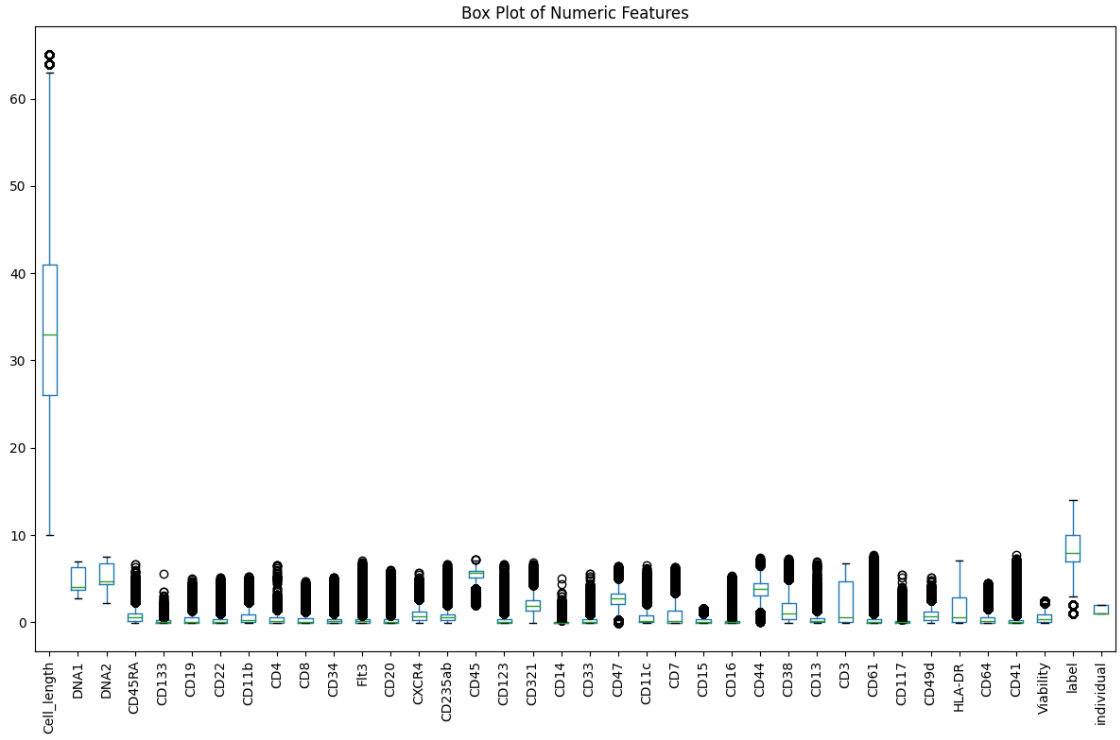
[ ]:
```
summary_stats = df.describe()
summary_stats
```

[ ]:
```
                Event           Time    Cell_length           DNA1  \
count  265627.000000  265627.000000  265627.000000  265627.000000
mean   132814.000000  272948.345014      34.450572       4.606956
std     76680.054314  171220.139430      11.446694       1.312831
min         1.000000       1.000000      10.000000       2.786488
25%     66407.500000  120196.000000      26.000000       3.700023
50%    132814.000000  253276.000000      33.000000       4.022127
75%    199220.500000  424502.500000      41.000000       6.353313
max    265627.000000  709122.440000      65.000000       7.001489

                DNA2         CD45RA          CD133           CD19  \
count  265627.000000  265627.000000  265627.000000  265627.000000
mean        5.198308       0.688127       0.145960       0.509301
std         1.150357       0.609105       0.259267       0.857462
min         2.236450      -0.057305      -0.058081      -0.058089
25%         4.407822       0.204625      -0.022935      -0.018838
50%         4.698415       0.549387       0.025353       0.075210
```

```
75%         6.766268       1.031198       0.224299       0.548386
max         7.472308       6.691197       5.527494       4.990085

                CD22           CD11b  …          CD117          CD49d  \
count   265627.000000  265627.000000  …  265627.000000  265627.000000
mean         0.397323       0.710319  …       0.131199       0.794938
std          0.762126       1.011434  …       0.313208       0.627619
min         -0.057342      -0.058236  …      -0.057668      -0.058064
25%         -0.020689      -0.000294  …      -0.023957       0.283013
50%          0.058790       0.257923  …      -0.000410       0.677212
75%          0.386481       0.923517  …       0.154736       1.190787
max          5.160477       5.260789  …       5.502125       5.153438

              HLA-DR           CD64           CD41      Viability  \
count   265627.000000  265627.000000  265627.000000  265627.000000
mean         1.521812       0.551512       0.261754       0.570037
std          1.694211       0.888739       0.617065       0.589738
min         -0.057974      -0.058199      -0.058244      -0.057979
25%          0.057709      -0.010582      -0.020166       0.065523
50%          0.611335       0.122493       0.052229       0.398230
75%          2.888240       0.604131       0.305591       0.931058
max          7.052507       4.517843       7.718288       2.433031

          file_number   event_number          label     individual
count   265627.000000  265627.000000  104184.000000  265627.000000
mean         3.639348  171288.314234       8.116102       1.279625
std          0.018678  123904.361456       2.457486       0.448816
min          3.627711       1.000000       1.000000       1.000000
25%          3.627711   58679.500000       7.000000       1.000000
50%          3.627711  152783.000000       8.000000       1.000000
75%          3.669327  282369.000000      10.000000       2.000000
max          3.669327  400112.000000      14.000000       2.000000

[8 rows x 42 columns]
```

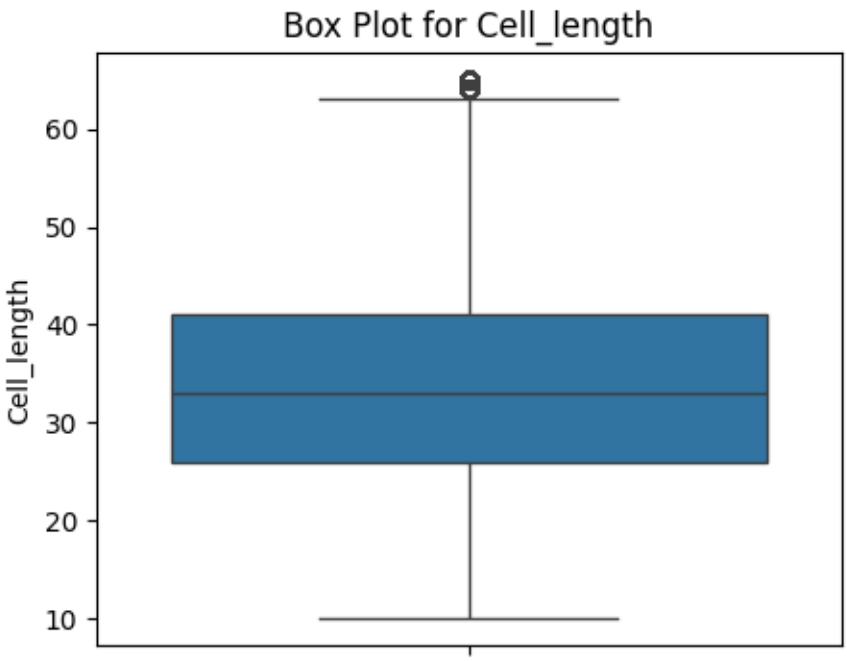
This code creates a box plot for the numeric features in the DataFrame, showing their spread, quartiles, and outliers.

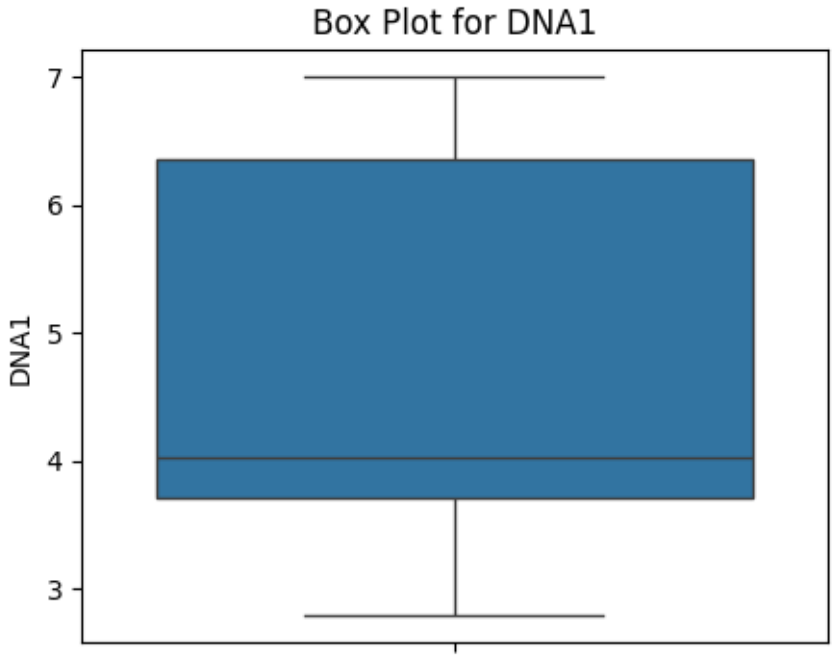
[ ]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
df[numeric_columns].boxplot(grid=False)
plt.xticks(rotation=90)
plt.title('Box Plot of Numeric Features')
plt.tight_layout()
plt.show()
```

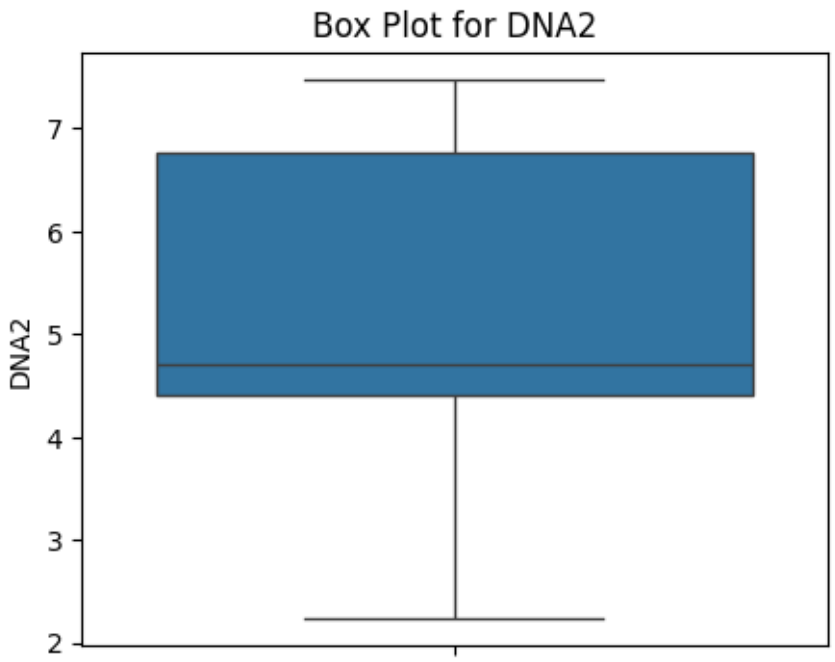


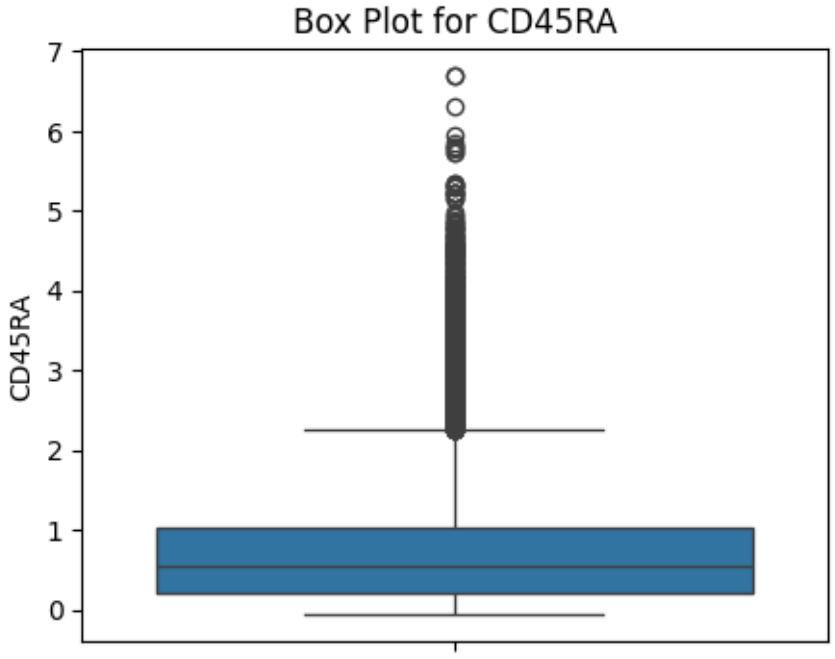
This code creates individual box plots for each column (after dropping certain irrelevant columns).

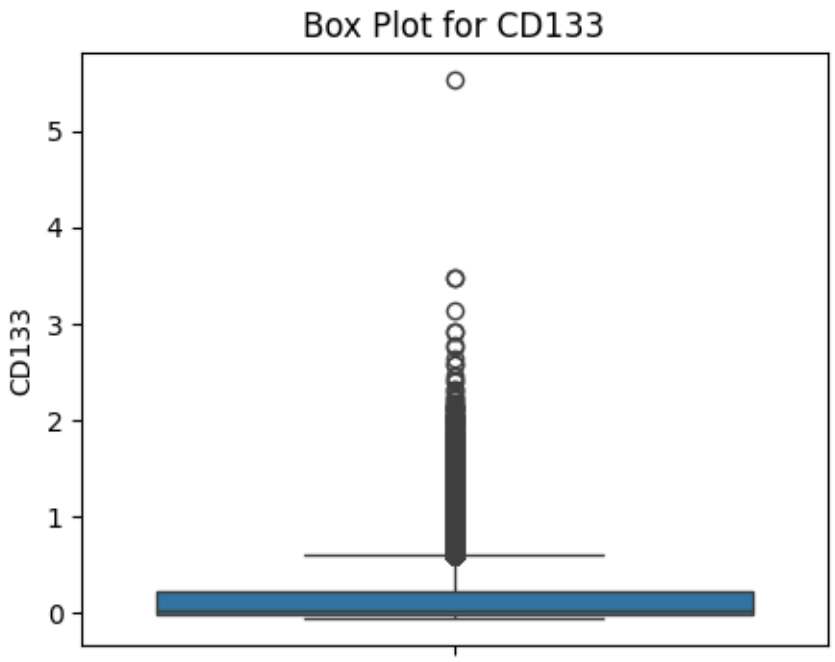
[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

filtered_df = df.drop(columns=['file_number', 'Event', 'Time', 'event_number'])

for column in filtered_df.columns:
    plt.figure(figsize=(5, 4))
    sns.boxplot(y=filtered_df[column])
    plt.title(f'Box Plot for {column}')
    plt.ylabel(column)
    plt.show()
```

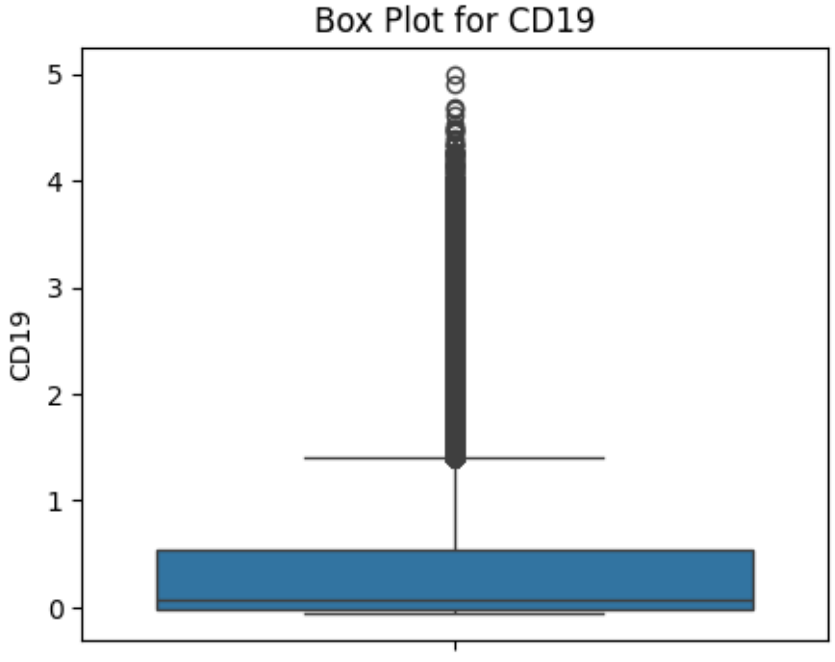

Box Plot for Cell_length
Cell_length
60
50
40
30
20
10

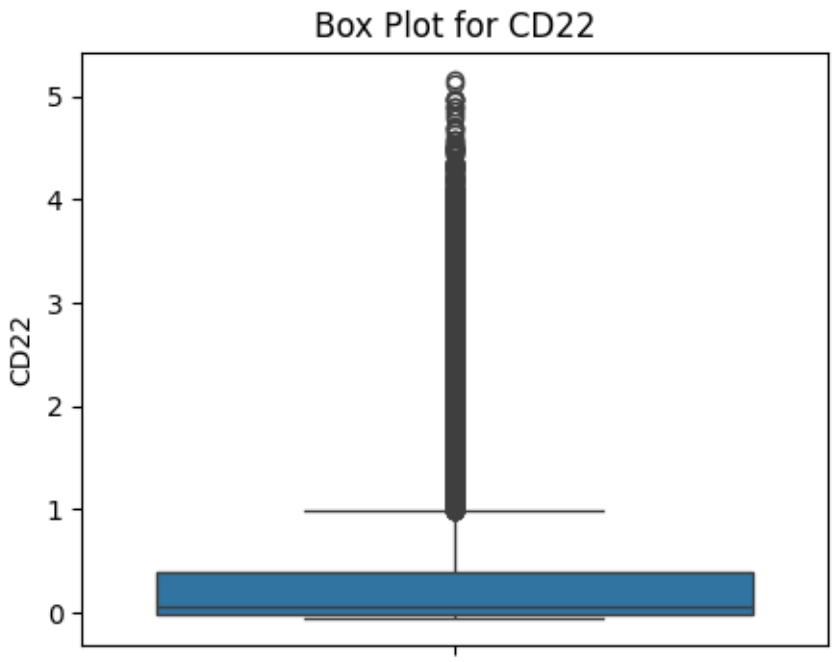

Box Plot for DNA1
DNA1
7
6
5
4
3

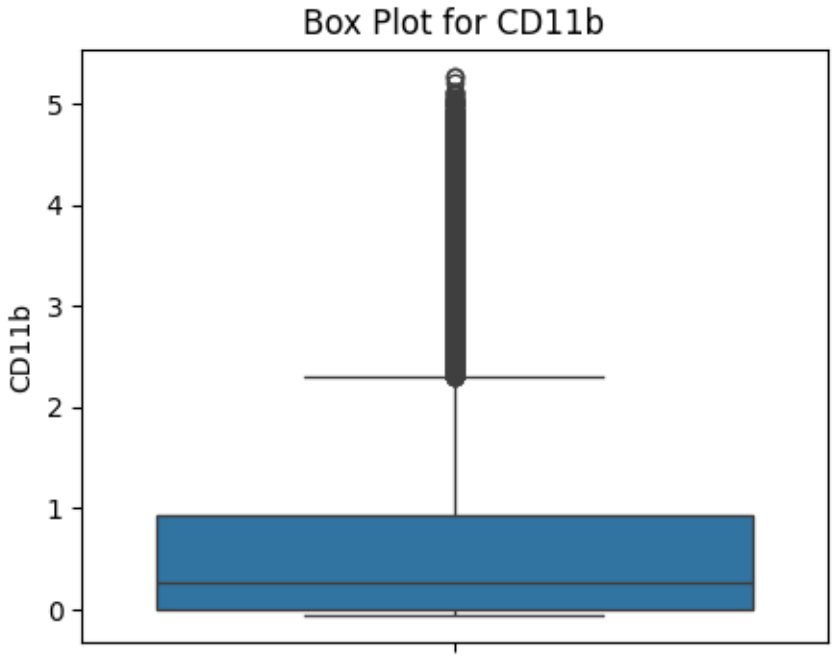

Box Plot for DNA2
DNA2
7
6
5
4
3
2

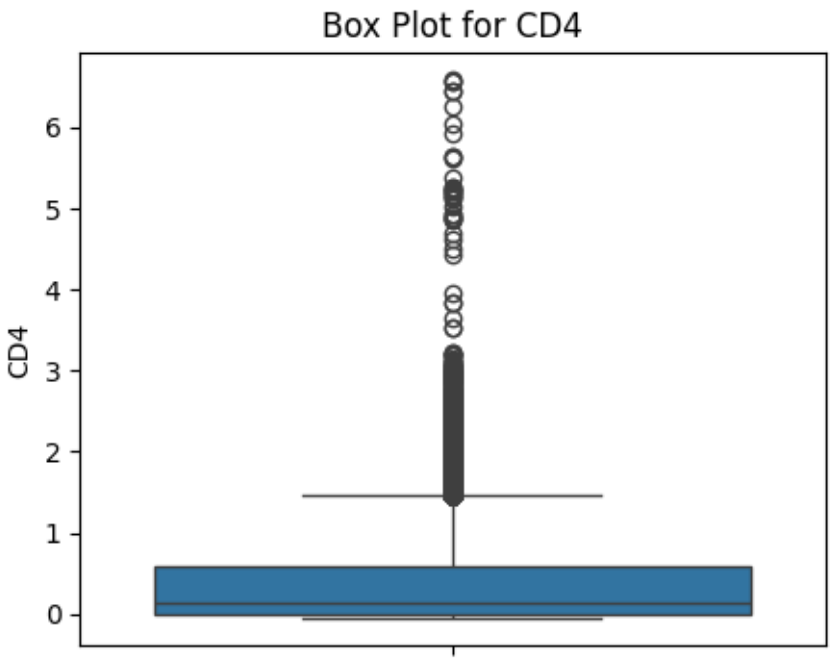

Box Plot for CD45RA
CD45RA
7
6
5
4
3
2
1
0

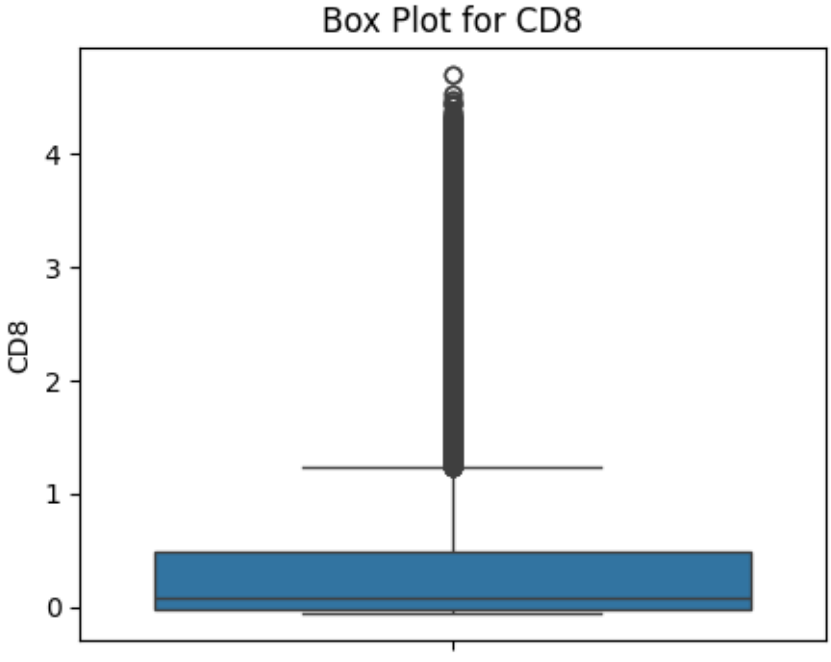

Box Plot for CD133
CD133
0
1
2
3
4
5

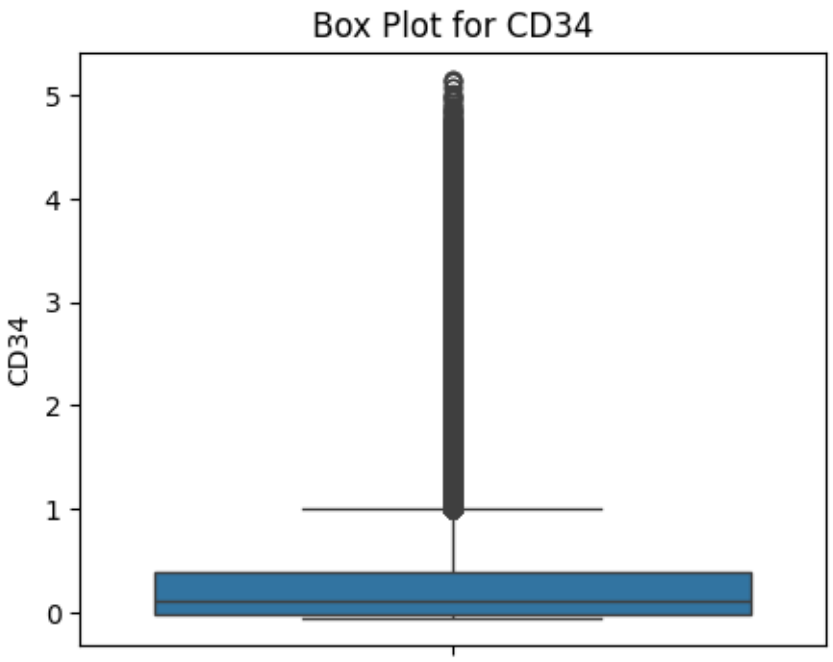

Box Plot for CD19
CD19
0
1
2
3
4
5

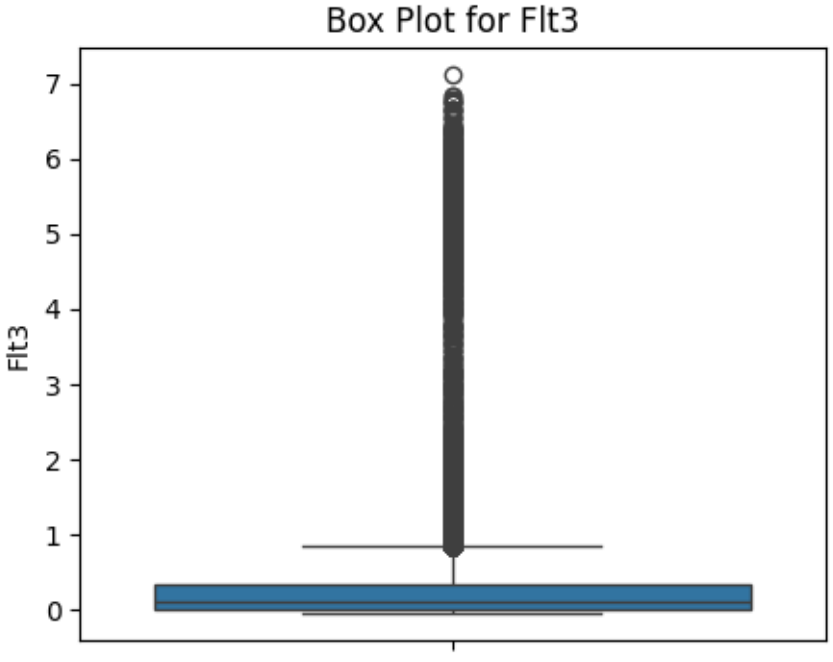

Box Plot for CD22
CD22
0
1
2
3
4
5

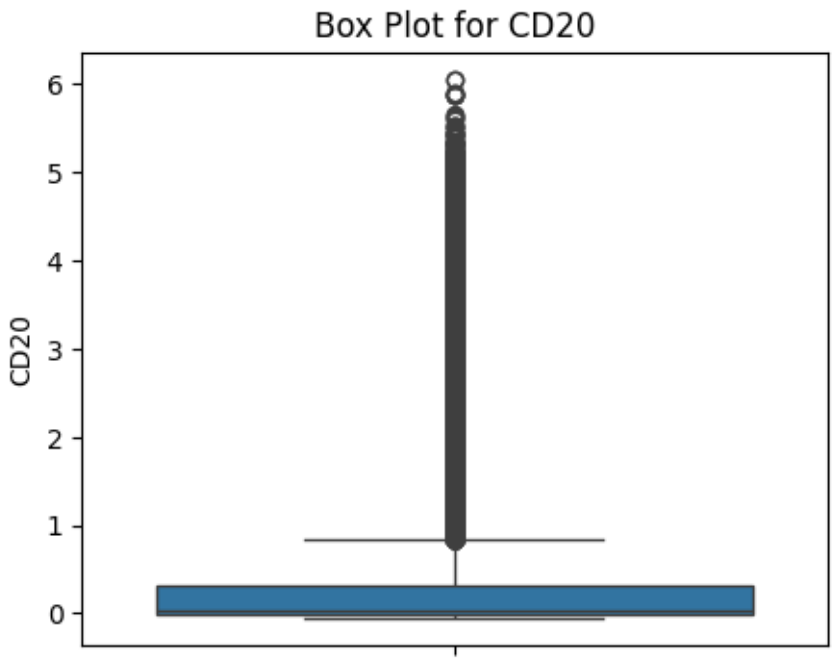

Box Plot for CD11b
CD11b
0
1
2
3
4
5

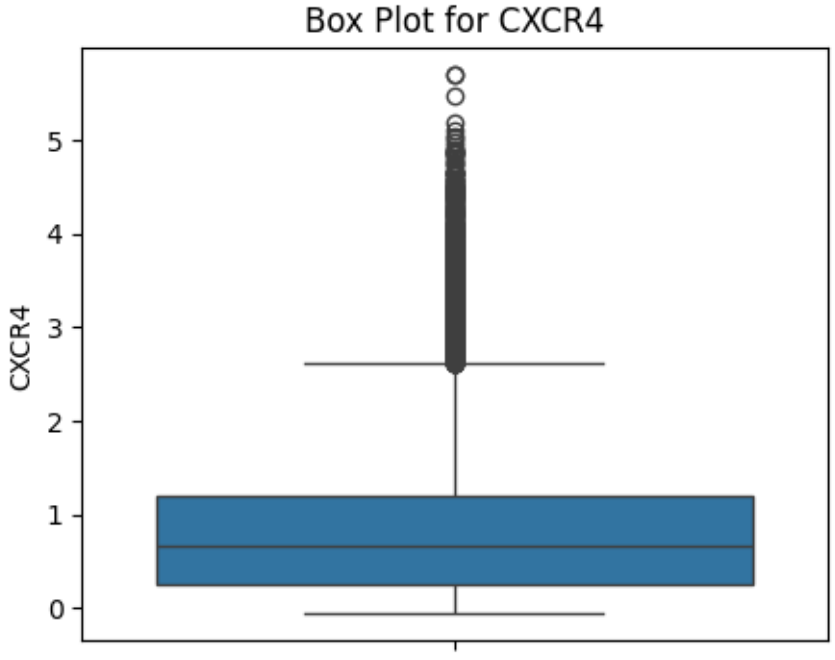

Box Plot for CD4
CD4
0
1
2
3
4
5
6

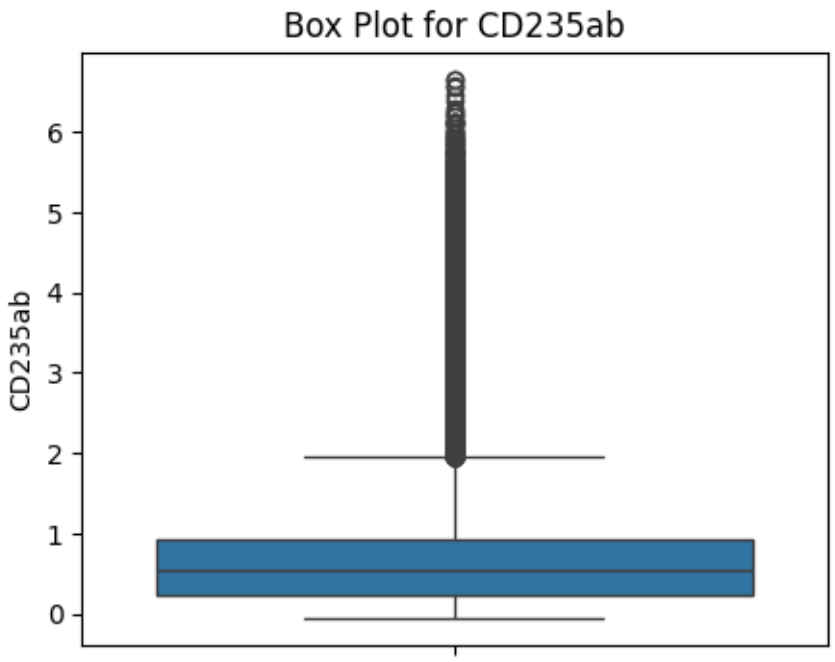

Box Plot for CD8
CD8
0
1
2
3
4

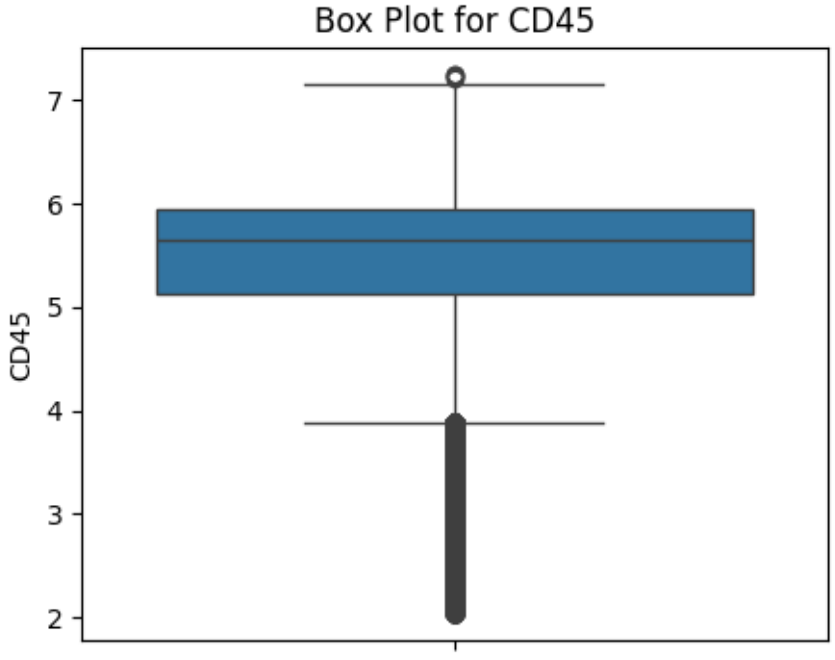

Box Plot for CD34
CD34
0
1
2
3
4
5

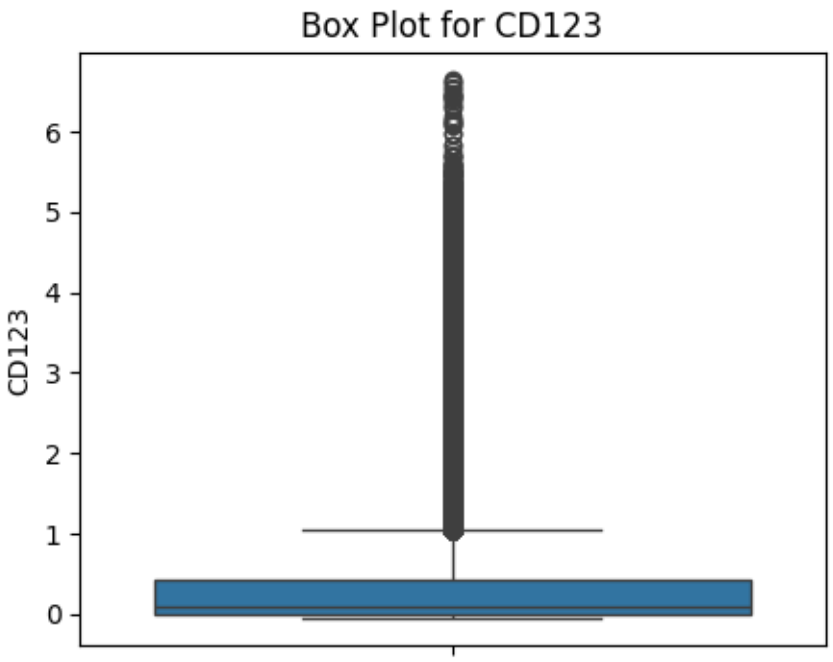

Box Plot for Flt3
Flt3
0
1
2
3
4
5
6
7

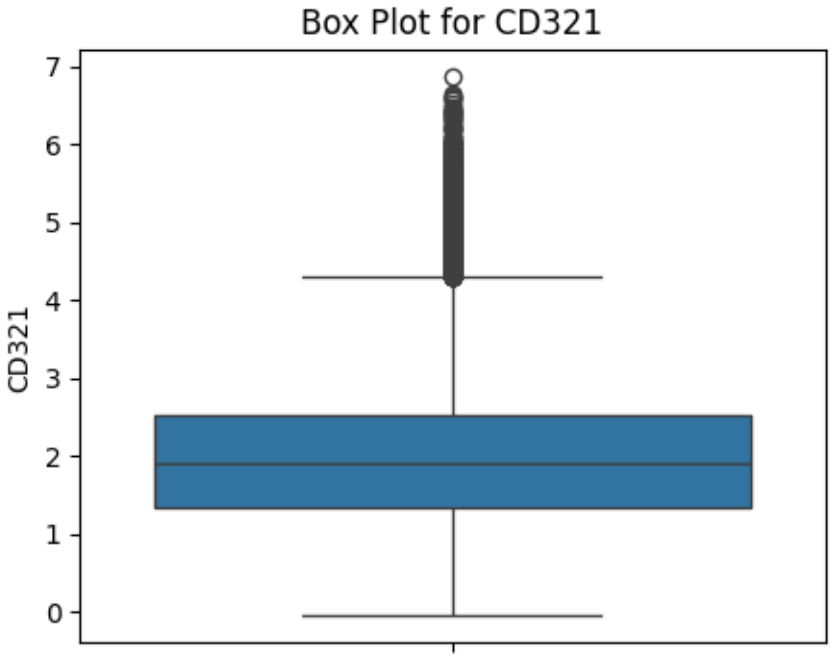

Box Plot for CD20
CD20
0
1
2
3
4
5
6

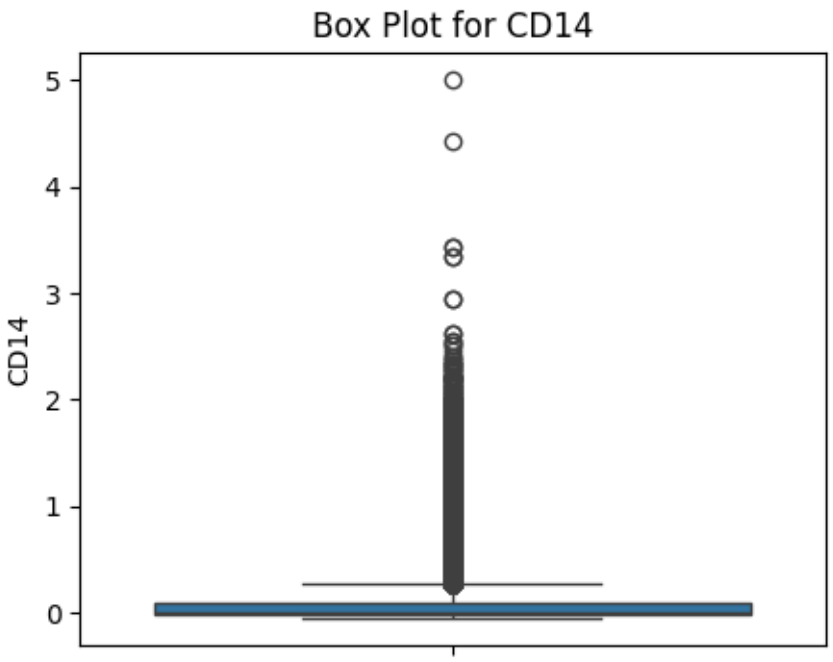

Box Plot for CXCR4
CXCR4
0
1
2
3
4
5

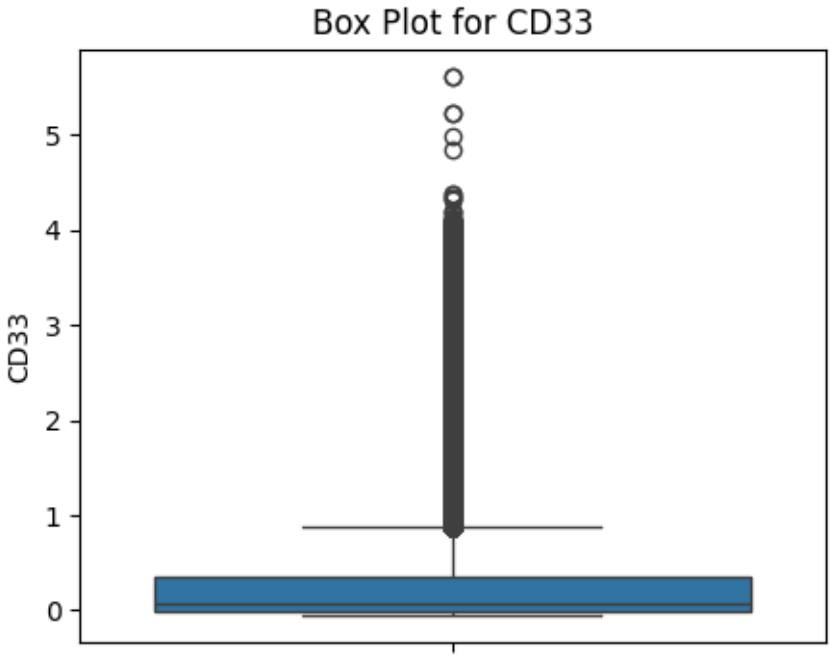

Box Plot for CD235ab
CD235ab
0
1
2
3
4
5
6

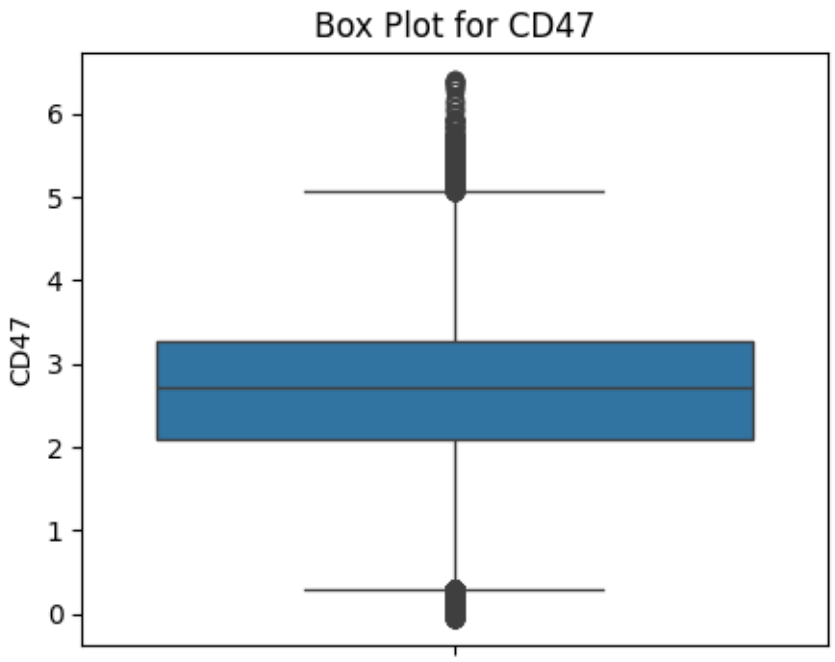

Box Plot for CD45
CD45
2
3
4
5
6
7

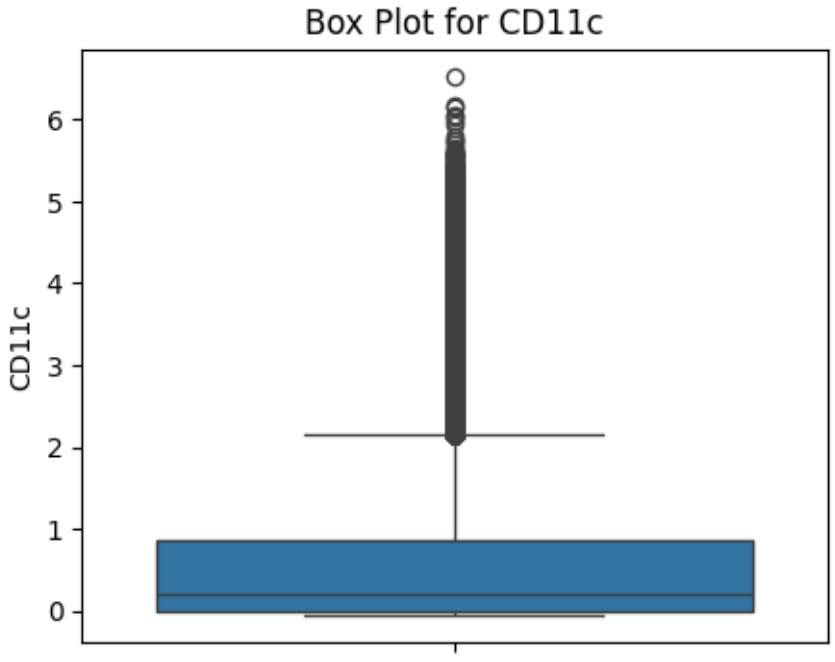

Box Plot for CD123
CD123
0
1
2
3
4
5
6

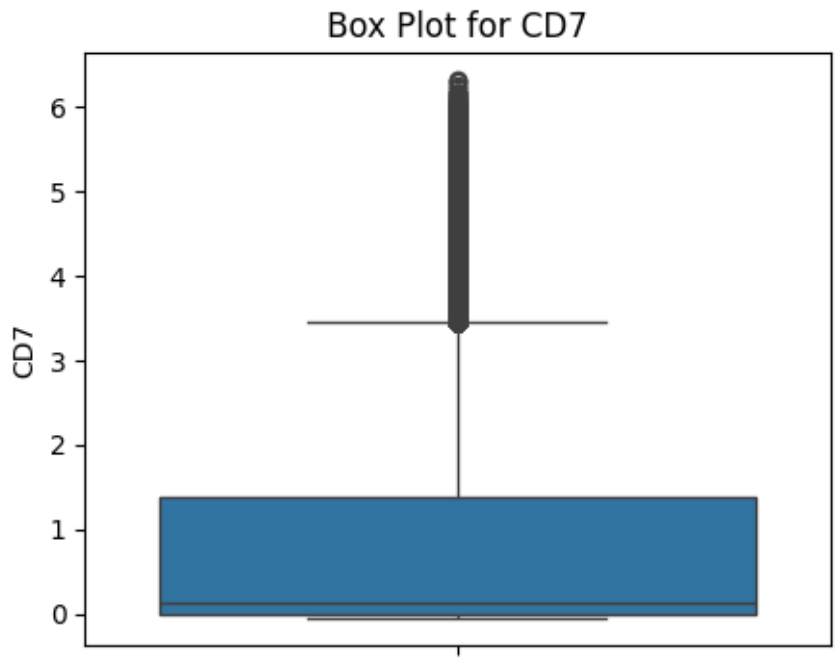

Box Plot for CD321
CD321
0
1
2
3
4
5
6
7

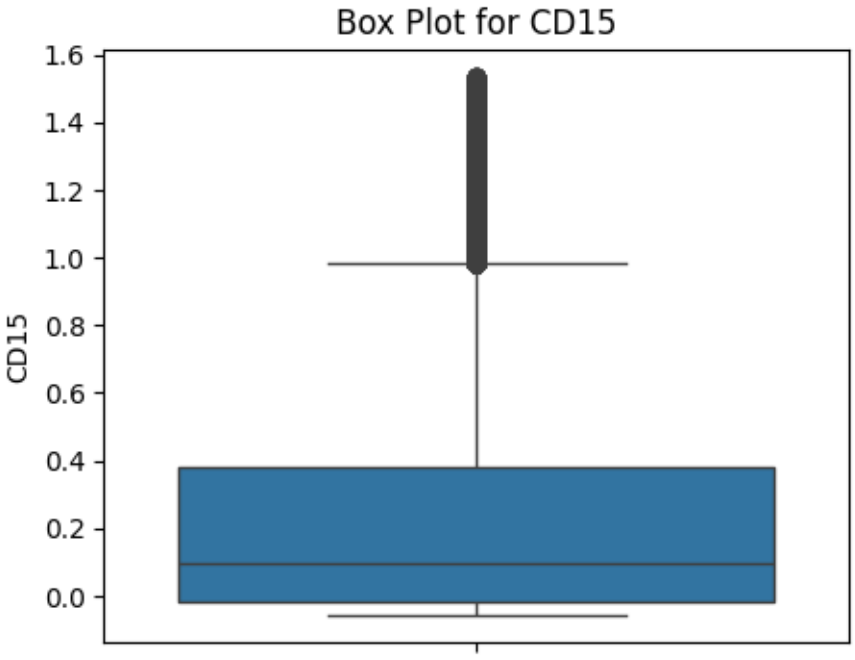

Box Plot for CD14
CD14
5
4
3
2
1
0

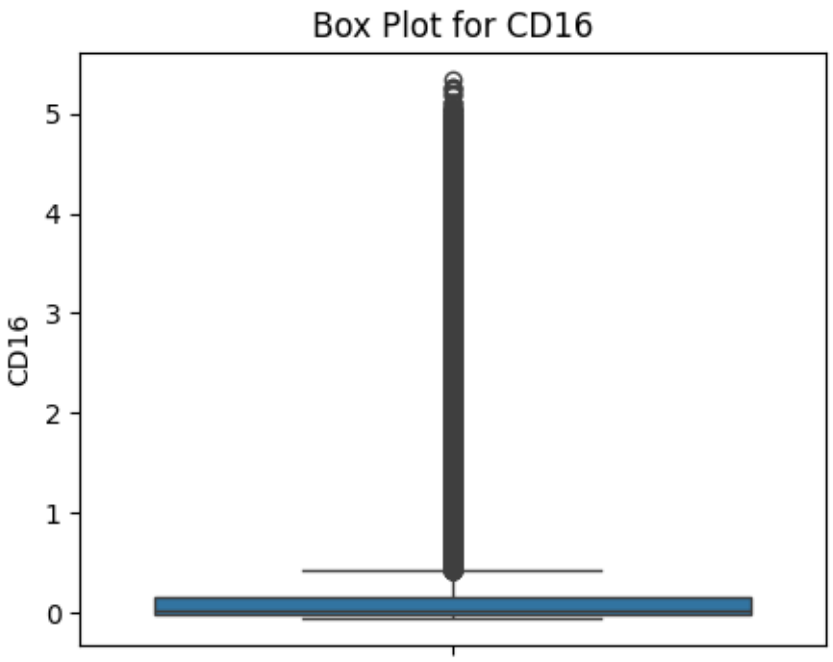

Box Plot for CD33
CD33
5
4
3
2
1
0

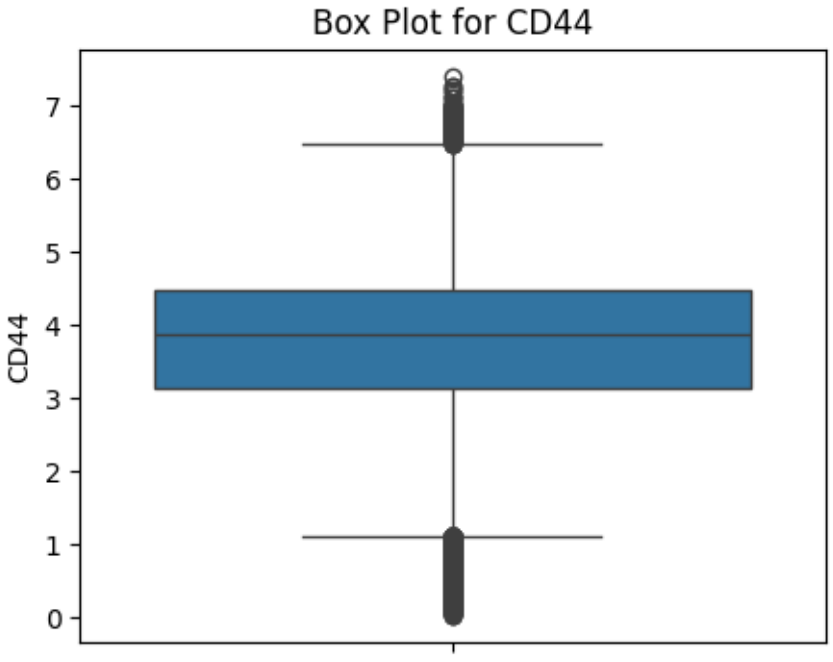

Box Plot for CD47
CD47
0
1
2
3
4
5
6

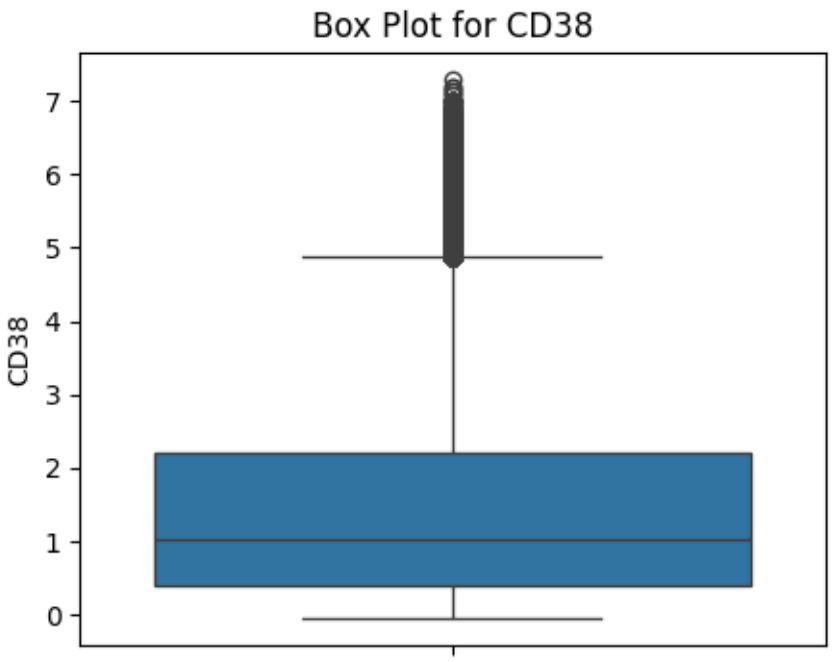

Box Plot for CD11c
CD11c
0
1
2
3
4
5
6

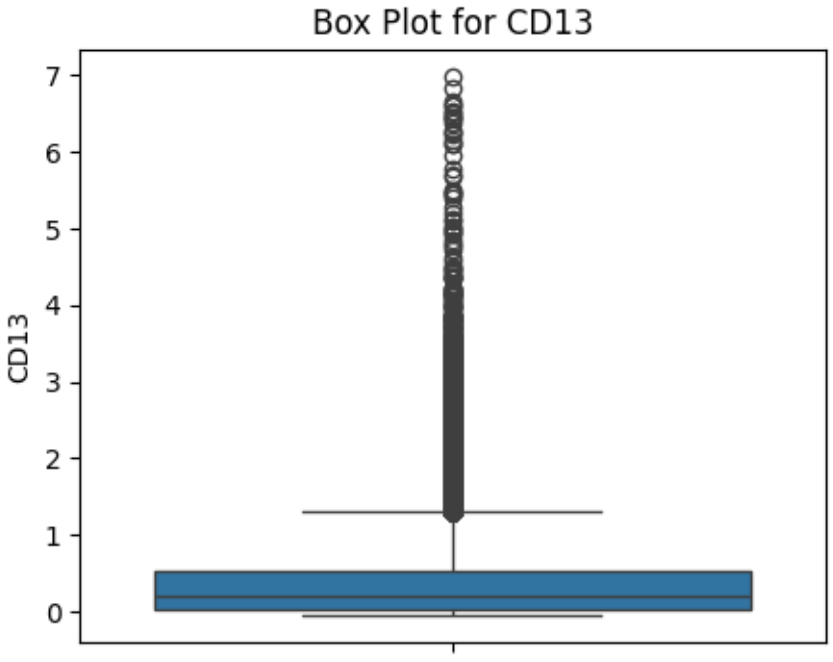

Box Plot for CD7
CD7
0
1
2
3
4
5
6

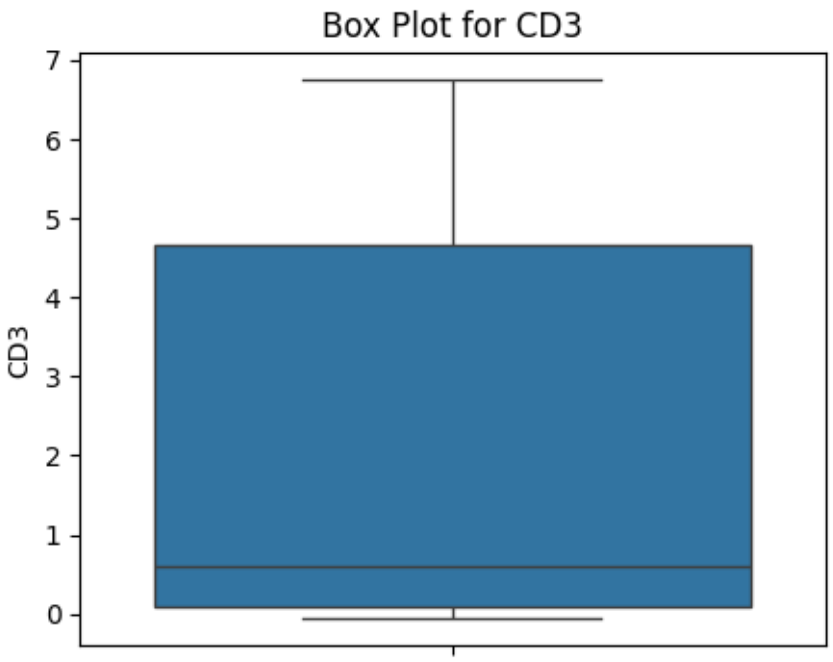

Box Plot for CD15
CD15
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

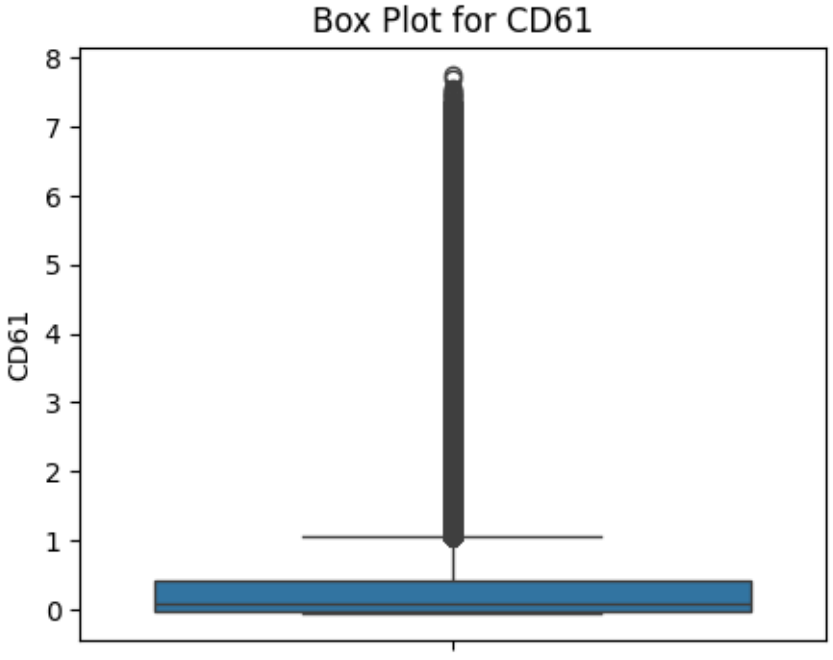

Box Plot for CD16
CD16
0
1
2
3
4
5

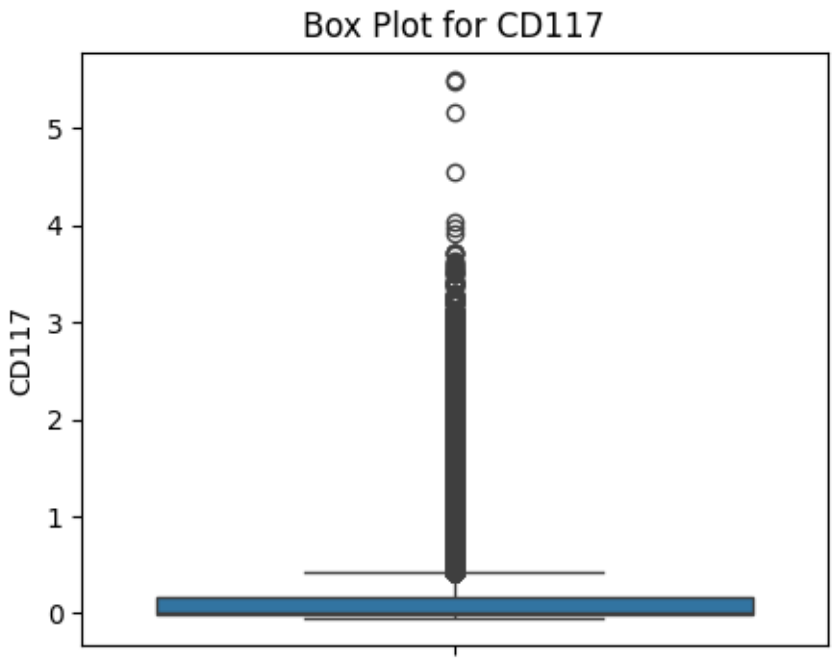

Box Plot for CD44
CD44
0
1
2
3
4
5
6
7

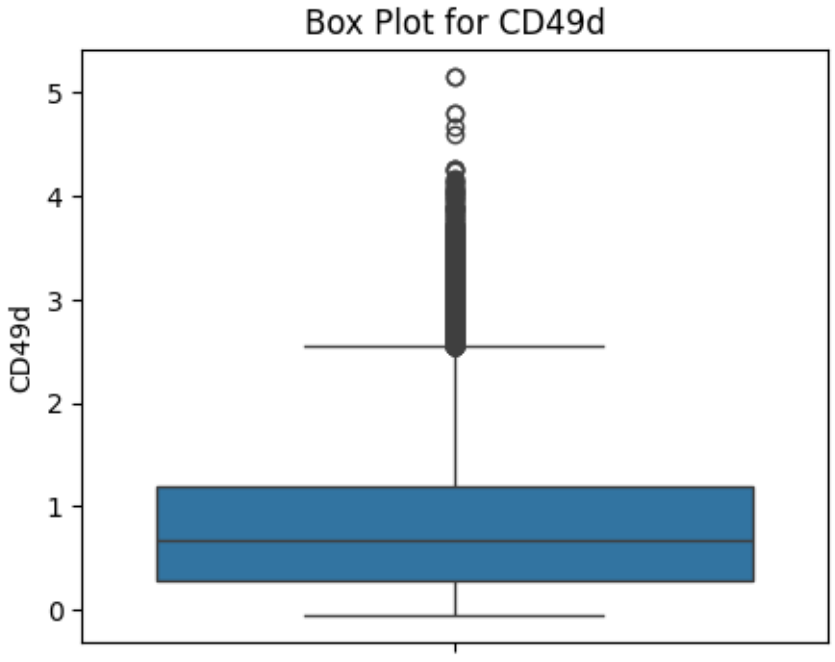

Box Plot for CD38
CD38
0
1
2
3
4
5
6
7

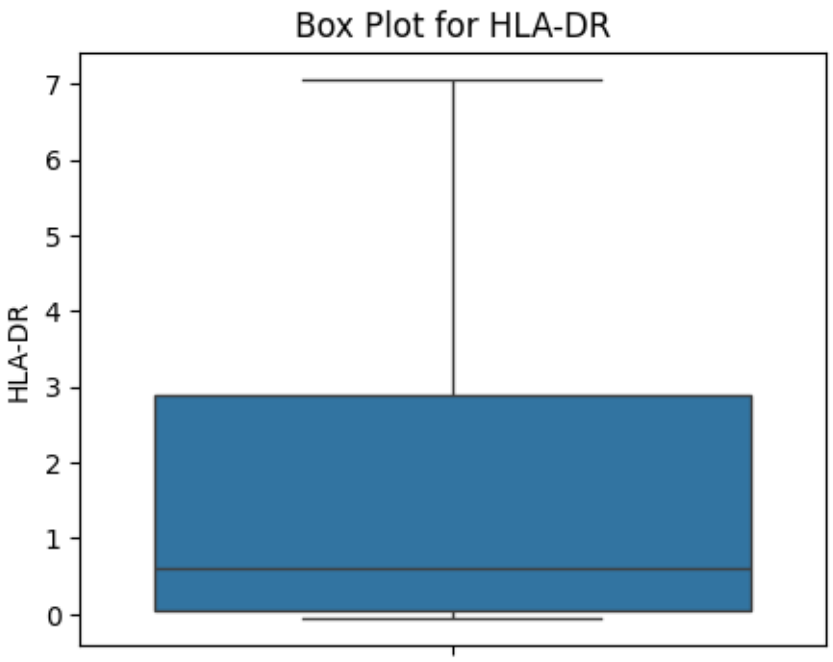

Box Plot for CD13
CD13
0
1
2
3
4
5
6
7

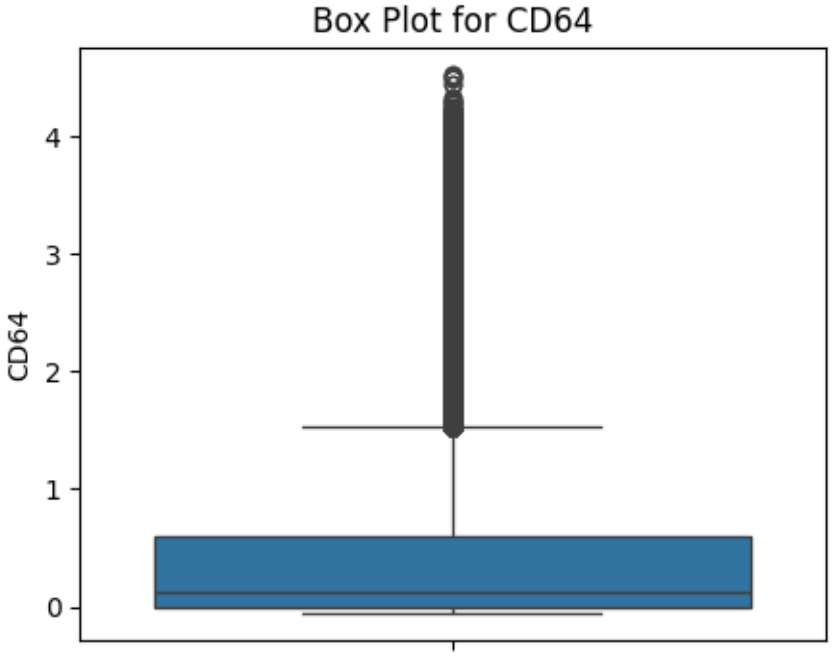

Box Plot for CD3
CD3
0
1
2
3
4
5
6
7

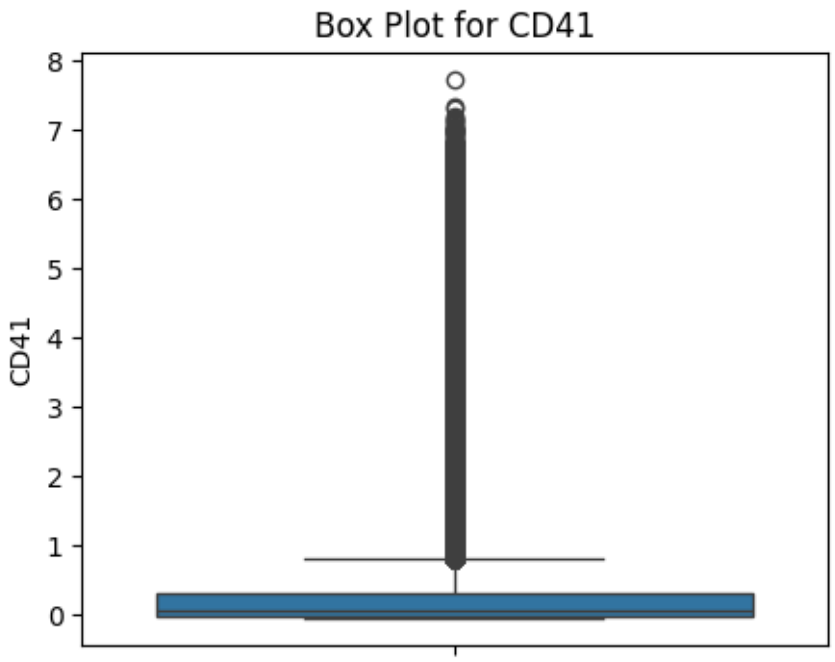

Box Plot for CD61
CD61
0
1
2
3
4
5
6
7
8

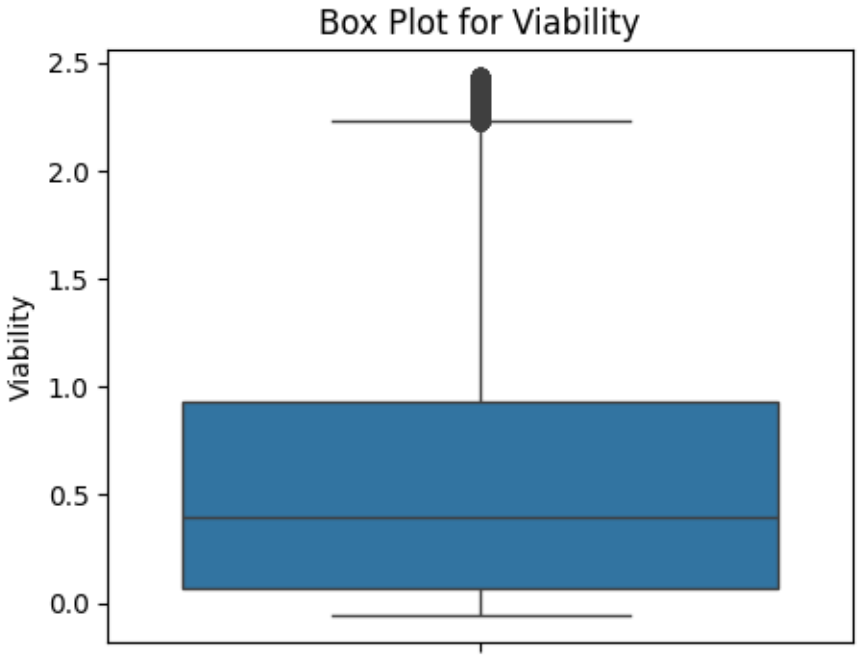

Box Plot for CD117
CD117
0
1
2
3
4
5

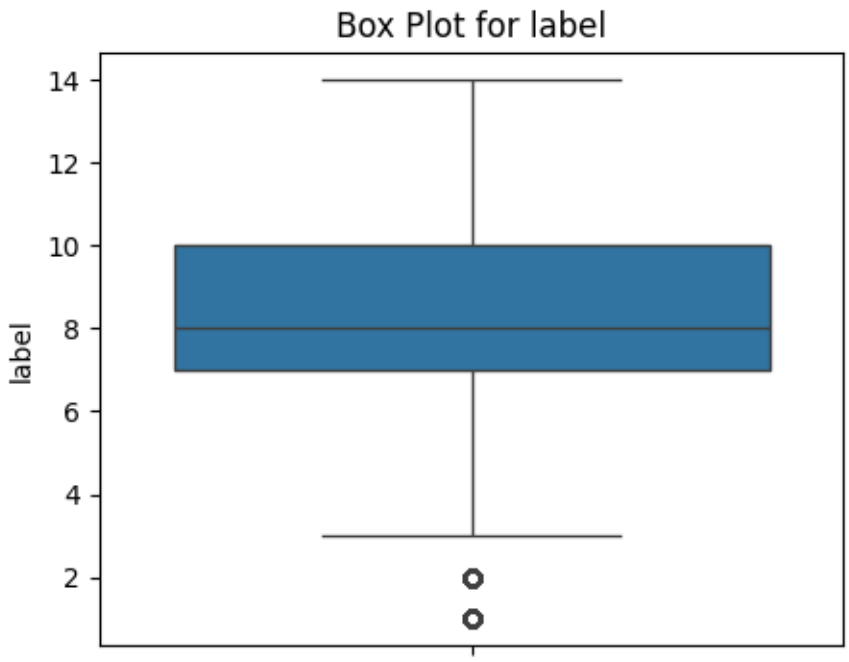

Box Plot for CD49d
CD49d
0
1
2
3
4
5

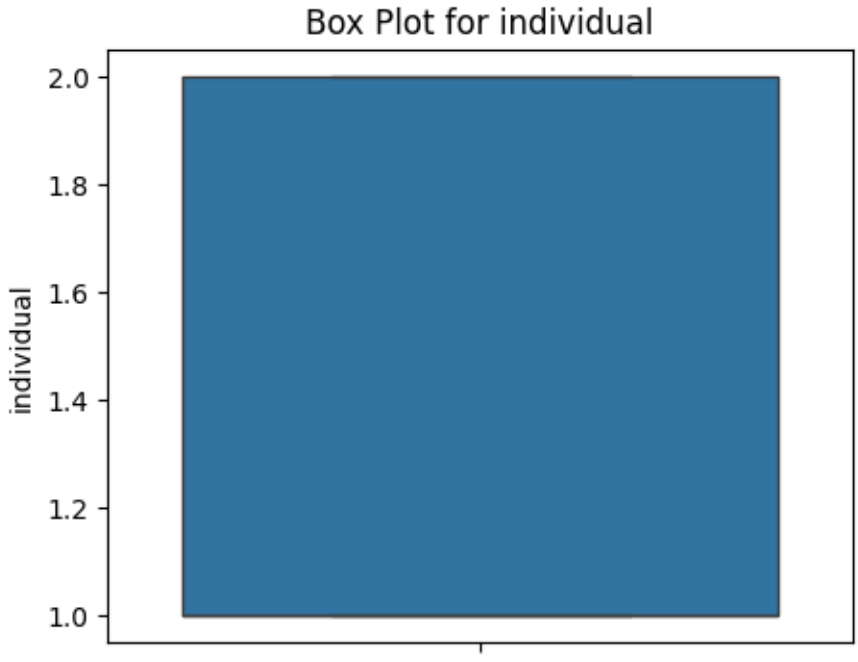

Box Plot for HLA-DR
HLA-DR
0
1
2
3
4
5
6
7


Box Plot for CD64
CD64
0
1
2
3
4


Box Plot for CD41
CD41
8
7
6
5
4
3
2
1
0


Box Plot for Viability
Viability
2.5
2.0
1.5
1.0
0.5
0.0





Skewness and kertosis

This code calculates and prints the skewness and kurtosis for each feature . Skewness measures asymmetry, while kurtosis indicates the tailedness of the distribution, giving insights into the shape

of the data.

[ ]:
```
from scipy.stats import skew, kurtosis

filtered_df = df.drop(columns=['file_number', 'Event', 'Time','event_number'])

for column in filtered_df.columns:
    skewness = skew(filtered_df[column].dropna())
    kurt = kurtosis(filtered_df[column].dropna())
    print(f'Feature: {column}')
    print(f'Skewness: {skewness:.4f}')
    print(f'Kurtosis: {kurt:.4f}\n')
```

```
Feature: Cell_length
Skewness: 0.5278
Kurtosis: -0.1660

Feature: DNA1
Skewness: 0.8450
Kurtosis: -1.0060

Feature: DNA2
Skewness: 0.7792
Kurtosis: -1.0250

Feature: CD45RA
Skewness: 1.1916
Kurtosis: 1.9643

Feature: CD133
Skewness: 2.1420
Kurtosis: 6.1901

Feature: CD19
Skewness: 1.6826
Kurtosis: 1.5909

Feature: CD22
Skewness: 2.2832
Kurtosis: 4.5002

Feature: CD11b
Skewness: 1.6791
Kurtosis: 1.9645

Feature: CD4
Skewness: 1.6220
Kurtosis: 2.8443
```

```
Feature: CD8
Skewness: 1.7757
Kurtosis: 1.7458

Feature: CD34
Skewness: 3.4924
Kurtosis: 13.5964

Feature: Flt3
Skewness: 7.0982
Kurtosis: 82.5835

Feature: CD20
Skewness: 2.7547
Kurtosis: 7.4354

Feature: CXCR4
Skewness: 0.9553
Kurtosis: 0.9363

Feature: CD235ab
Skewness: 2.0015
Kurtosis: 10.4406

Feature: CD45
Skewness: -1.4848
Kurtosis: 2.2468

Feature: CD123
Skewness: 3.6489
Kurtosis: 15.3612

Feature: CD321
Skewness: 0.2471
Kurtosis: -0.0854

Feature: CD14
Skewness: 3.6090
Kurtosis: 20.0625

Feature: CD33
Skewness: 2.7250
Kurtosis: 7.9675

Feature: CD47
Skewness: -0.2503
Kurtosis: -0.0562
```

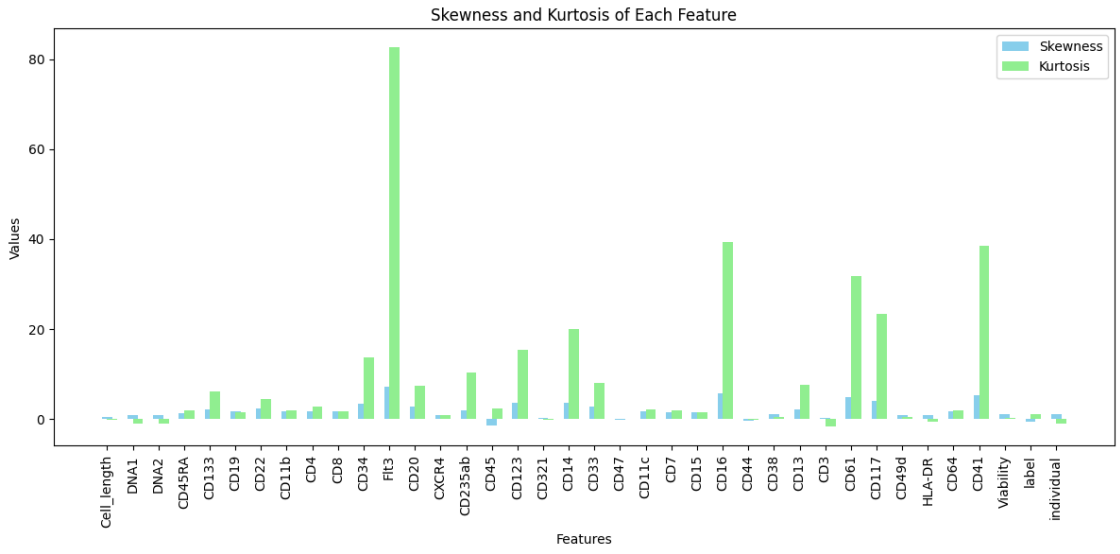
```
Feature: CD11c
Skewness: 1.7339
Kurtosis: 2.1172

Feature: CD7
Skewness: 1.6065
Kurtosis: 1.8851

Feature: CD15
Skewness: 1.4451
Kurtosis: 1.5044

Feature: CD16
Skewness: 5.7332
Kurtosis: 39.2877

Feature: CD44
Skewness: -0.4316
Kurtosis: -0.0812

Feature: CD38
Skewness: 1.1415
Kurtosis: 0.5212

Feature: CD13
Skewness: 2.2343
Kurtosis: 7.6376

Feature: CD3
Skewness: 0.3422
Kurtosis: -1.7354

Feature: CD61
Skewness: 4.8947
Kurtosis: 31.8780

Feature: CD117
Skewness: 4.0975
Kurtosis: 23.3751

Feature: CD49d
Skewness: 0.8568
Kurtosis: 0.4681

Feature: HLA-DR
Skewness: 0.7954
Kurtosis: -0.6901
```

```
Feature: CD64
Skewness: 1.7437
Kurtosis: 1.9106

Feature: CD41
Skewness: 5.3663
Kurtosis: 38.5211

Feature: Viability
Skewness: 0.9854
Kurtosis: 0.1569

Feature: label
Skewness: -0.5776
Kurtosis: 1.1252

Feature: individual
Skewness: 0.9820
Kurtosis: -1.0356
```

This code calculates the skewness and kurtosis for each feature in the dataset and stores the values. It then visualizes these values using a bar chart, showing the distribution characteristics (asymmetry and tailedness) of each feature.

[ ]:
```
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt

filtered_df = df.drop(columns=['file_number', 'Event', 'Time','event_number'])

skewness_values = []
kurtosis_values = []
columns = filtered_df.columns

for column in columns:
    skewness_values.append(skew(filtered_df[column].dropna()))
    kurtosis_values.append(kurtosis(filtered_df[column].dropna()))

index = range(len(columns))
plt.figure(figsize=(12, 6))

plt.bar(index, skewness_values, width=0.4, label='Skewness', align='center',␣
 ↪color='skyblue')
plt.bar(index, kurtosis_values, width=0.4, label='Kurtosis', align='edge',␣
 ↪color='lightgreen')

plt.xlabel('Features')
```

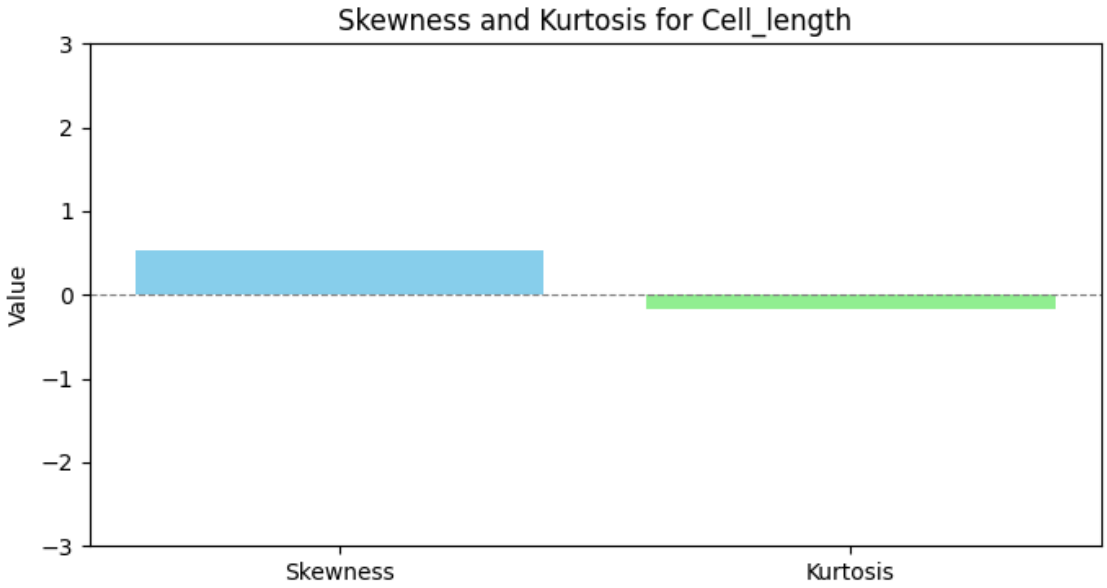
```
plt.ylabel('Values')
plt.title('Skewness and Kurtosis of Each Feature')
plt.xticks(index, columns, rotation=90)
plt.legend()

plt.tight_layout()
plt.show()
```

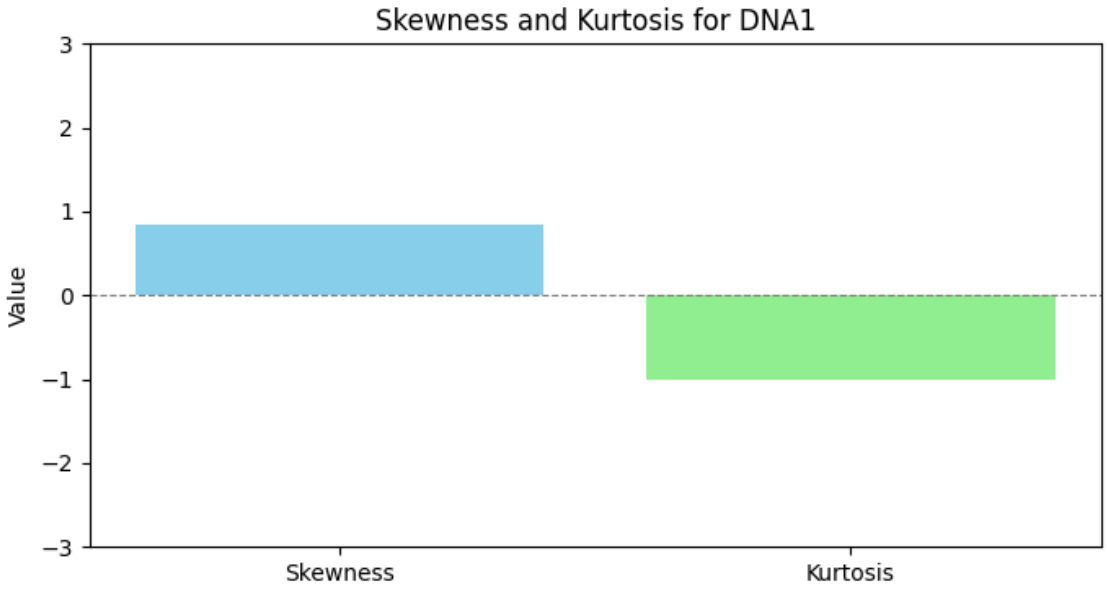


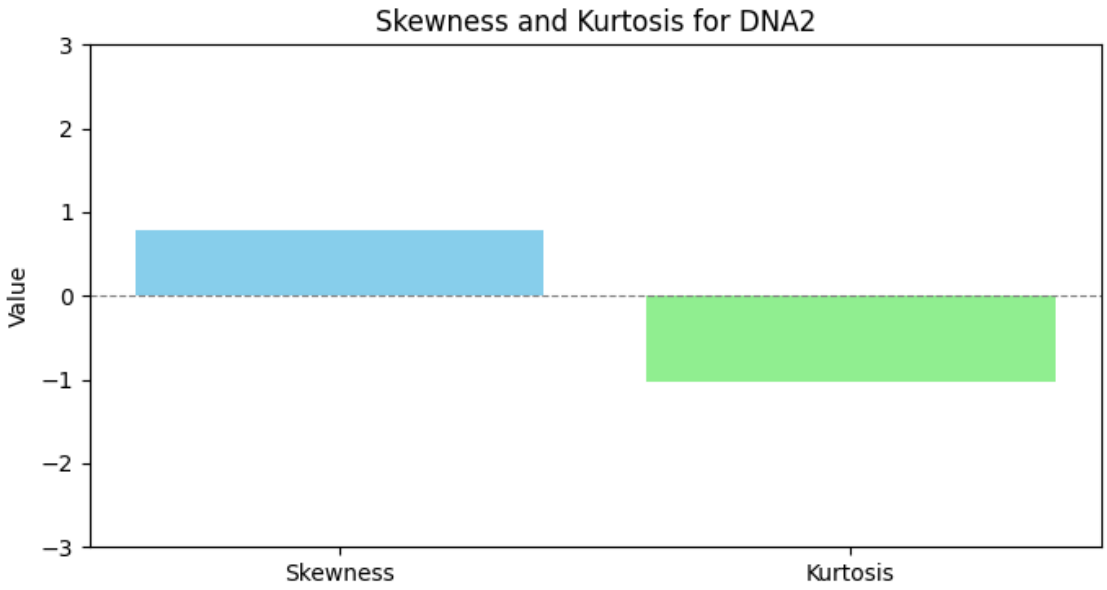
This code calculates the skewness and kurtosis for each feature in the dataset individually. It then generates a bar chart for each feature, comparing its skewness and kurtosis values, with a y-axis range of -3 to 3 for visual consistency.

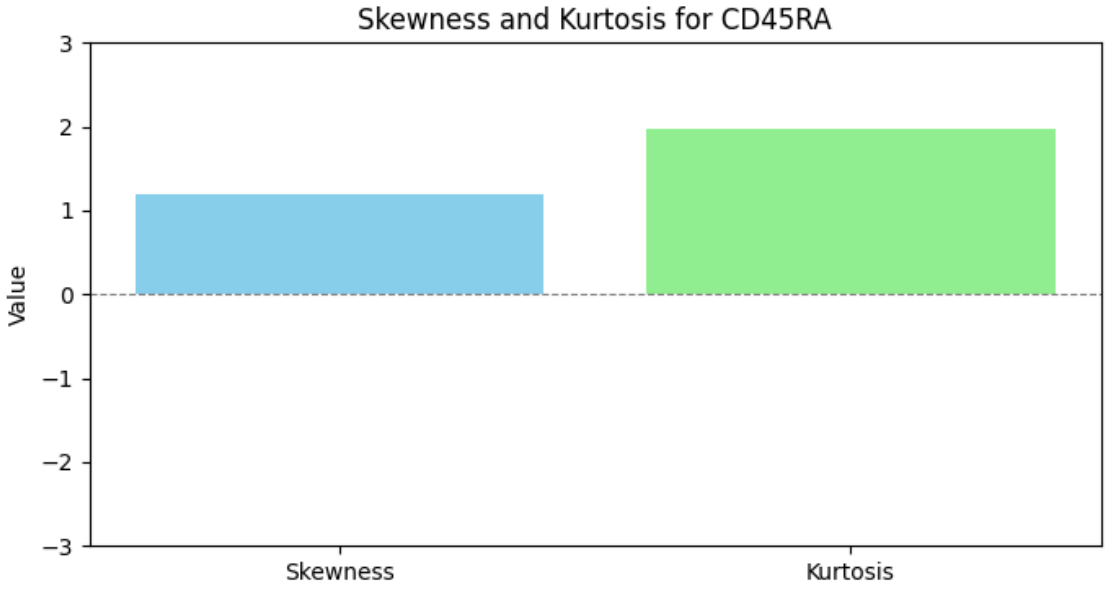
[ ]:
```
from scipy.stats import skew, kurtosis
import matplotlib.pyplot as plt

filtered_df = df.drop(columns=['file_number', 'Event', 'Time','event_number'])
columns = filtered_df.columns

for column in columns:
    skewness = skew(filtered_df[column].dropna())
    kurt = kurtosis(filtered_df[column].dropna())

    plt.figure(figsize=(8, 4))
    plt.bar(['Skewness', 'Kurtosis'], [skewness, kurt], color=['skyblue',␣
 ↪'lightgreen'])
    plt.ylabel('Value')
    plt.title(f'Skewness and Kurtosis for {column}')
    plt.ylim([-3, 3])
    plt.axhline(0, color='gray', linewidth=0.8, linestyle='--')
```

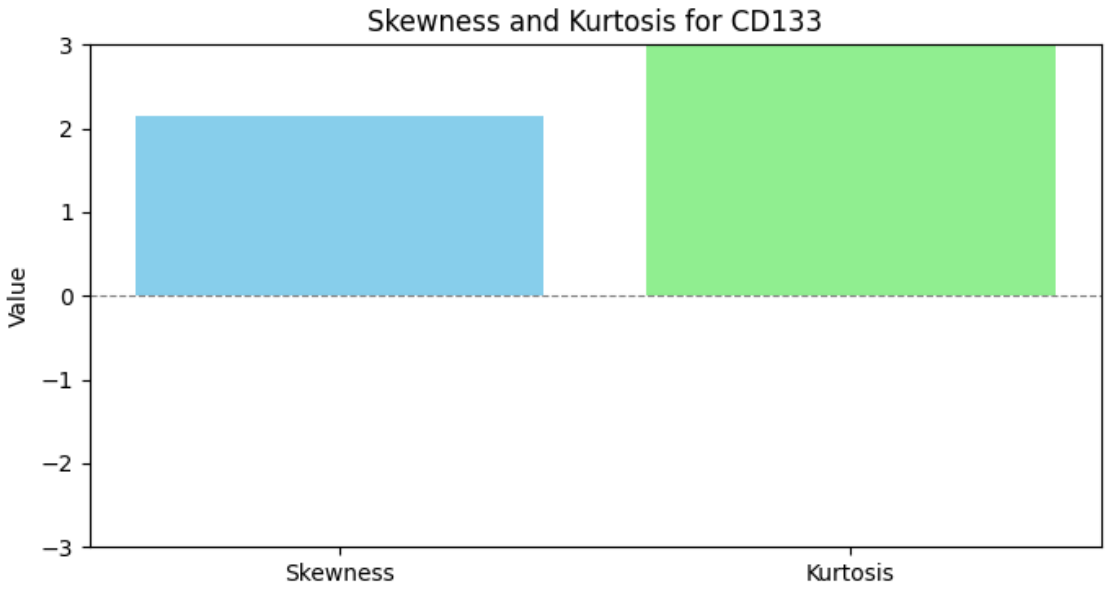
```
    plt.show()
```

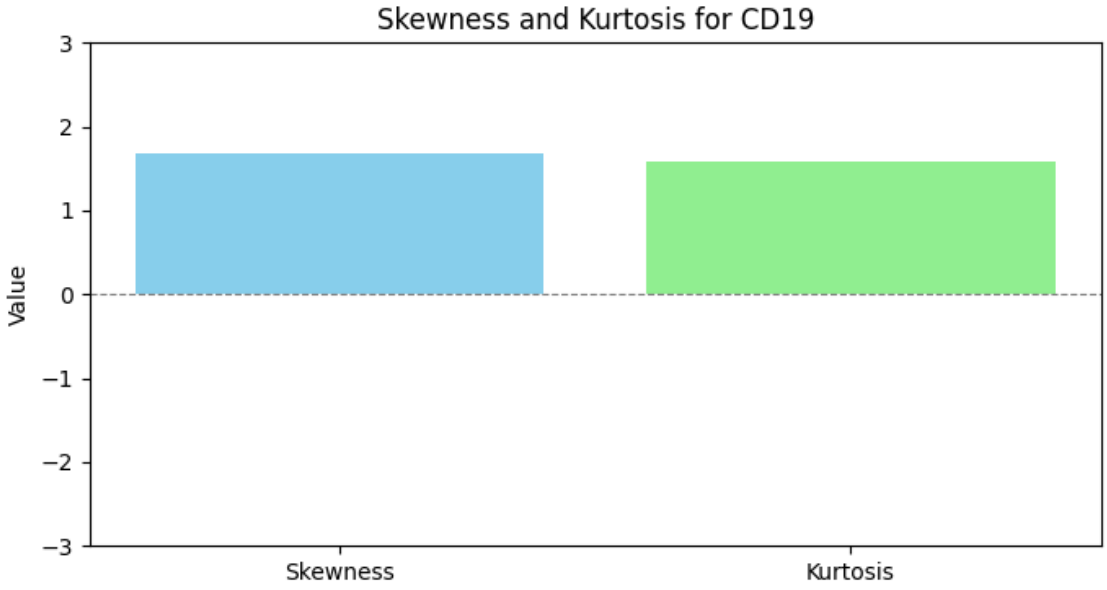

Skewness and Kurtosis for Cell_length
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

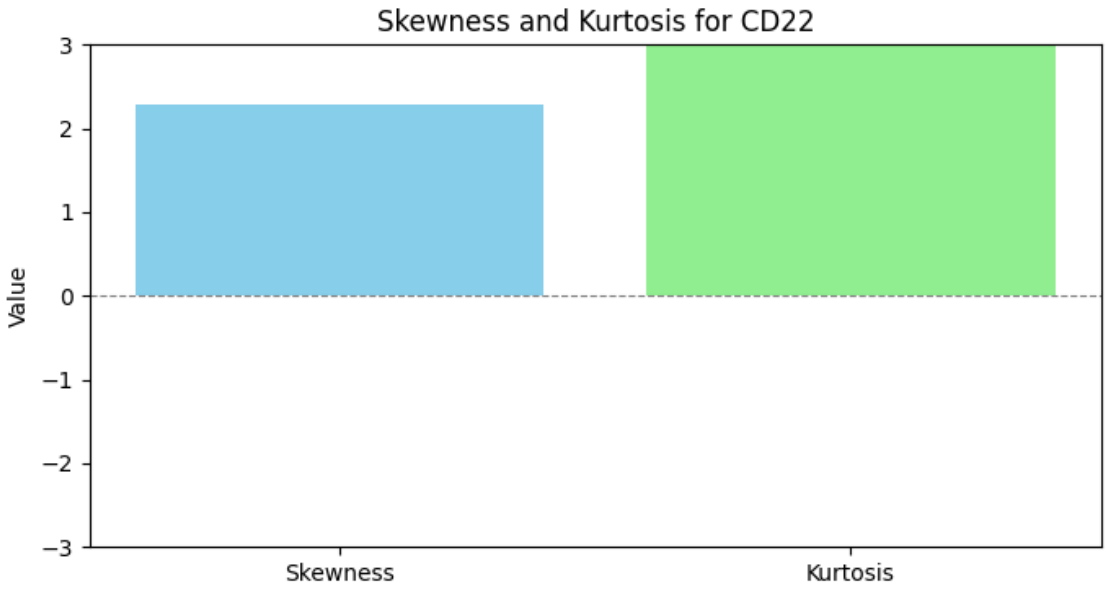

Skewness and Kurtosis for DNA1
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

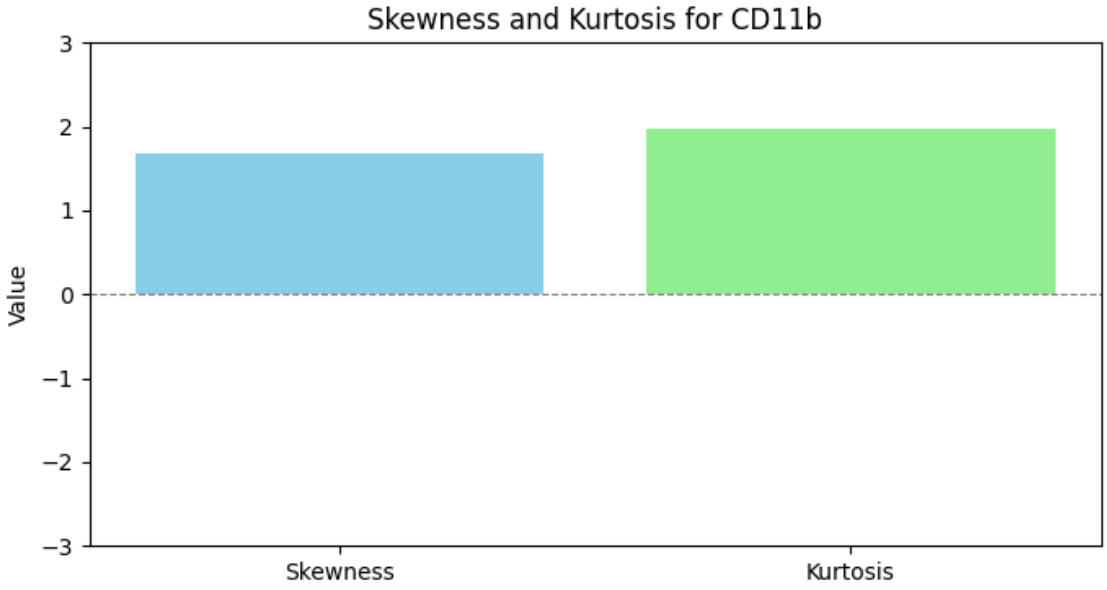

Skewness and Kurtosis for DNA2
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

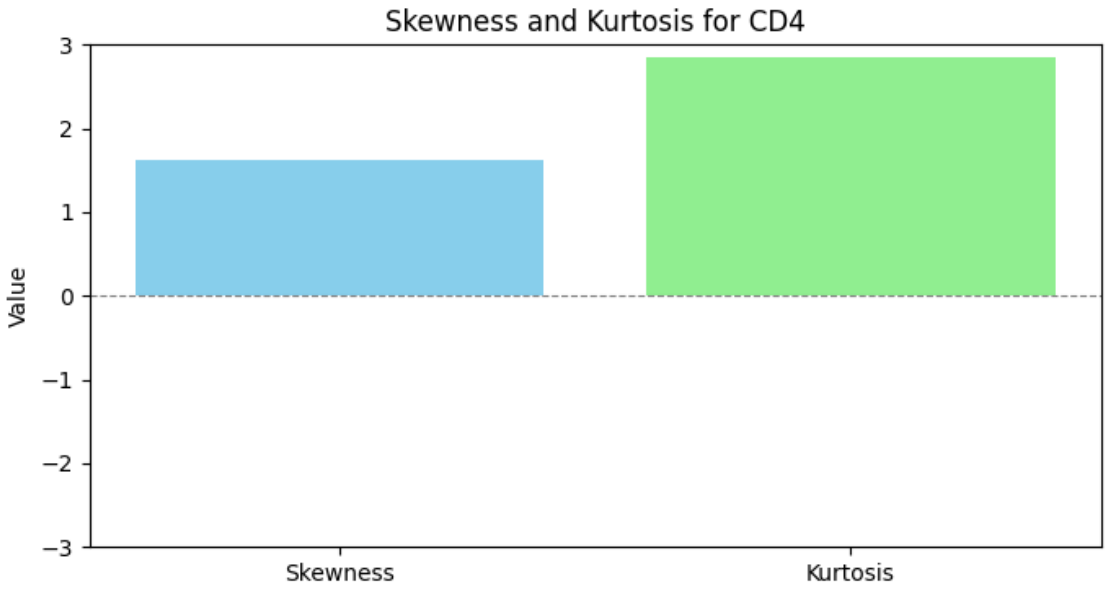

Skewness and Kurtosis for CD45RA
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

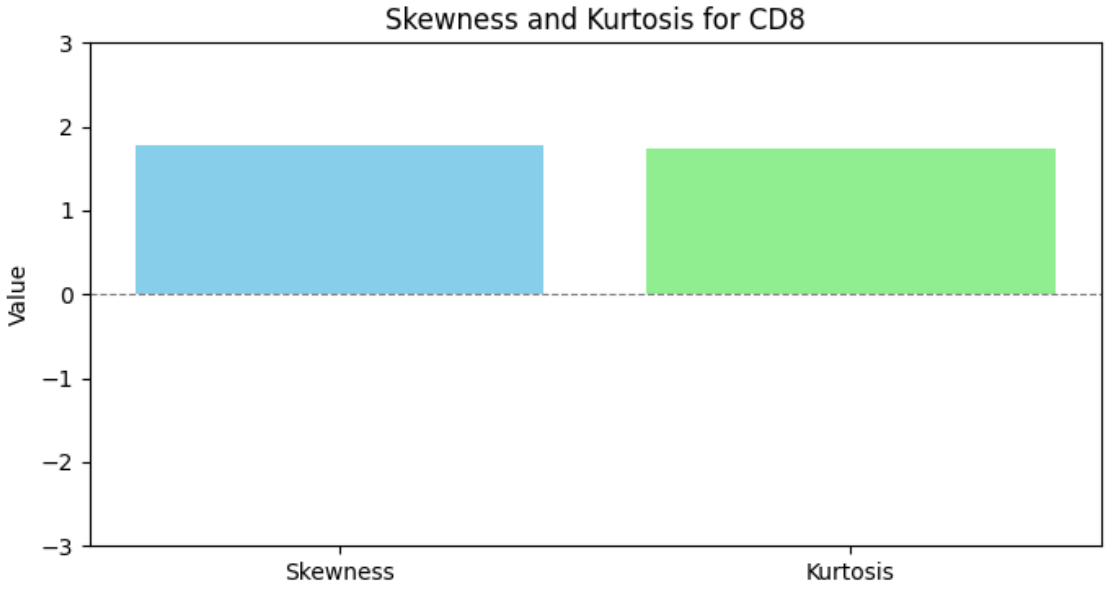

Skewness and Kurtosis for CD133
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

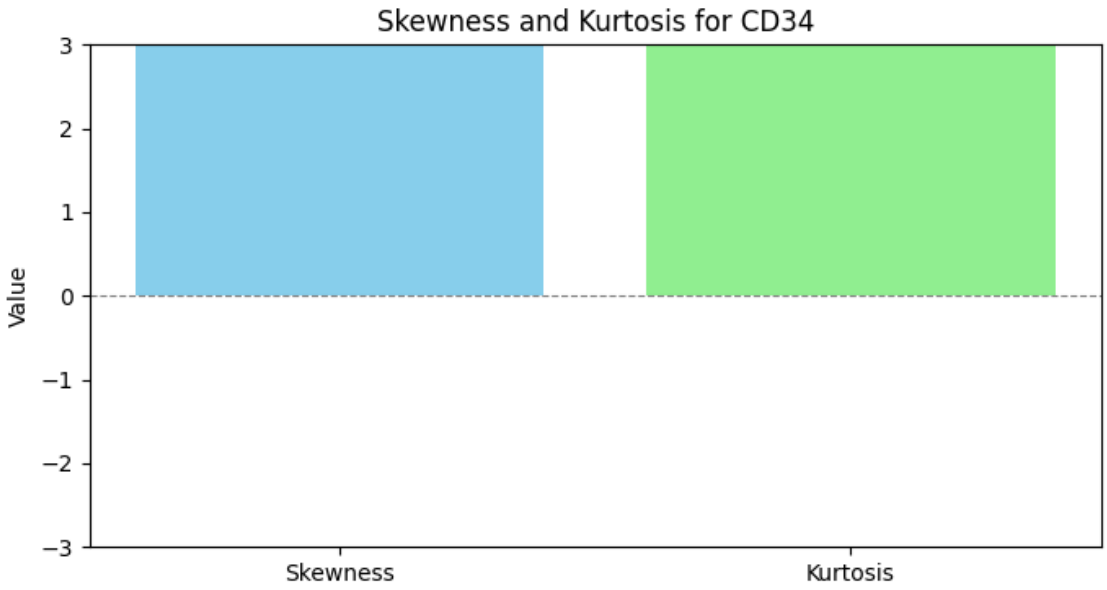

Skewness and Kurtosis for CD19
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

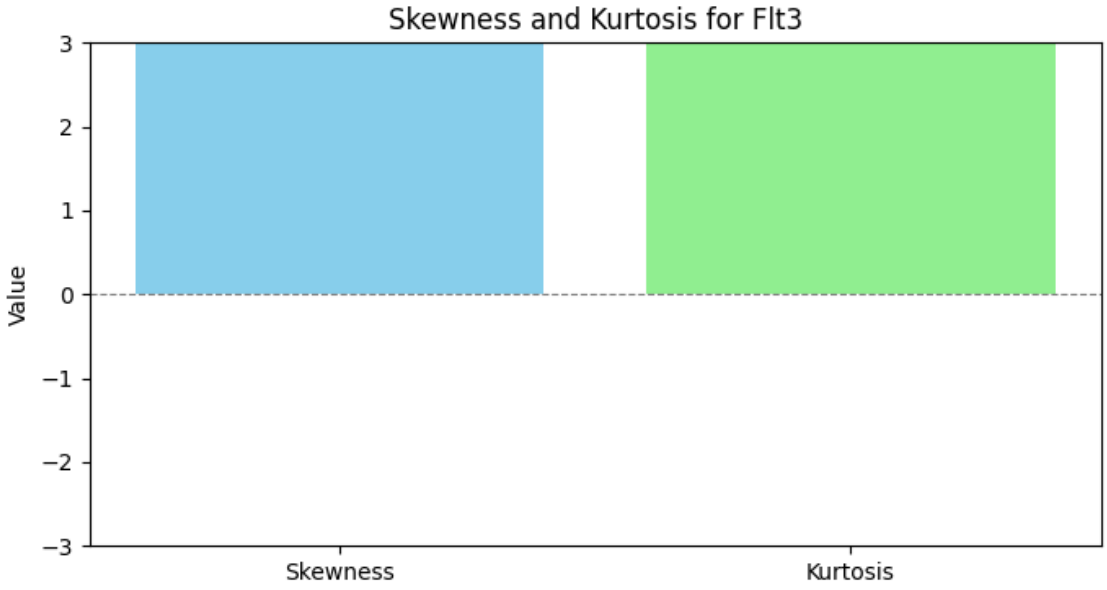

Skewness and Kurtosis for CD22
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

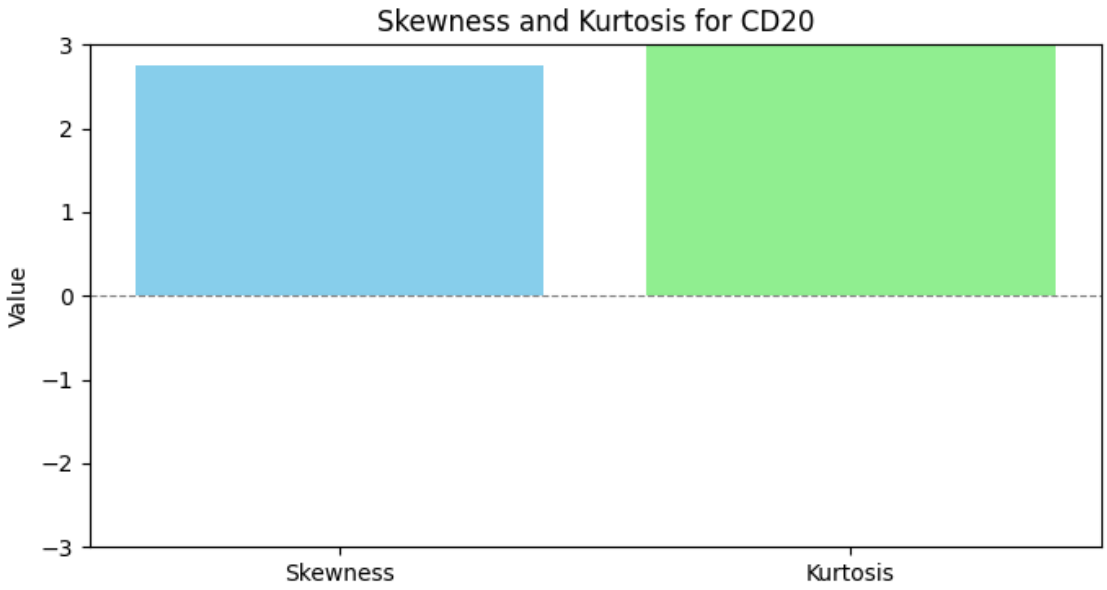

Skewness and Kurtosis for CD11b
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

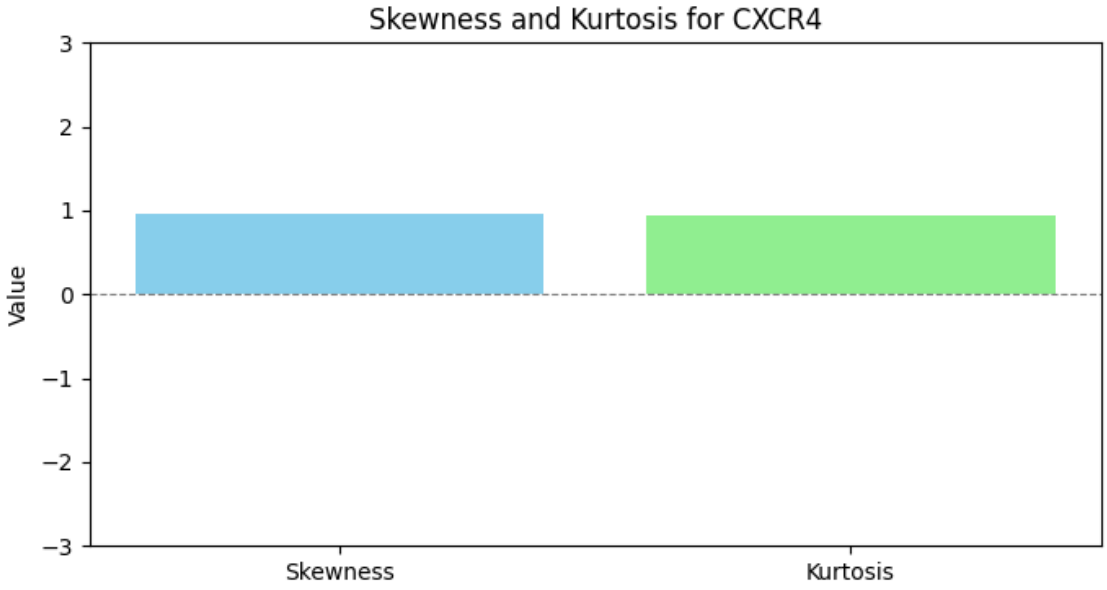

Skewness and Kurtosis for CD4
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

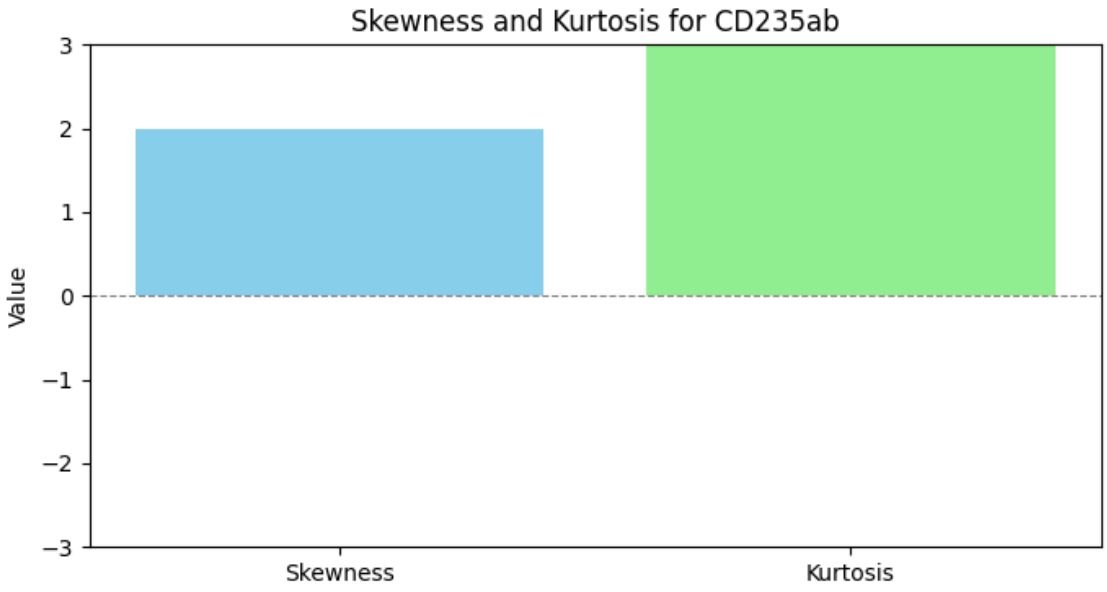

Skewness and Kurtosis for CD8
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

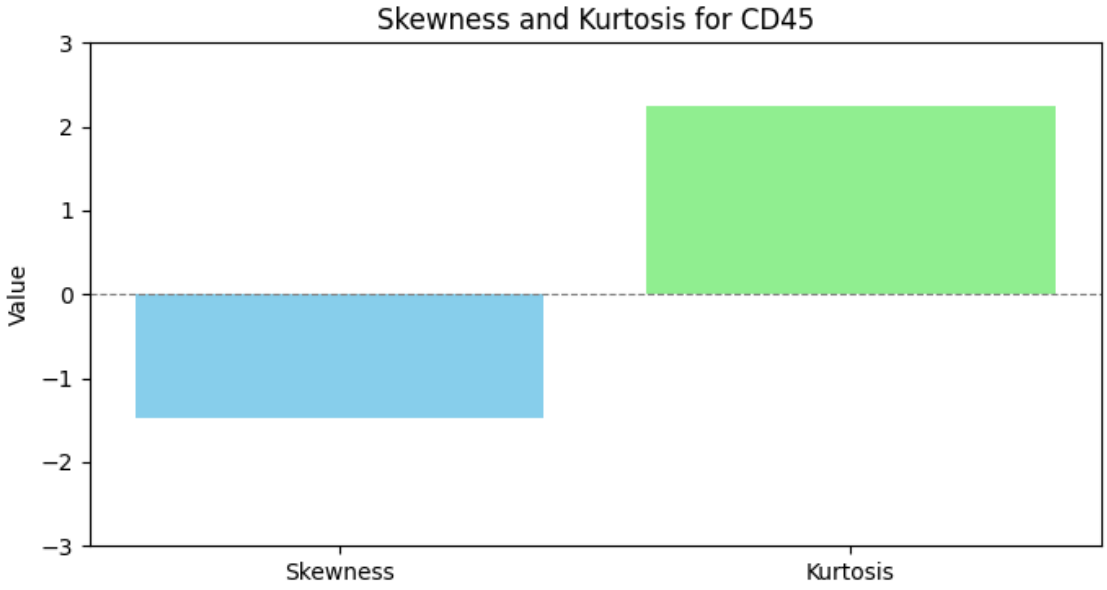

Skewness and Kurtosis for CD34
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

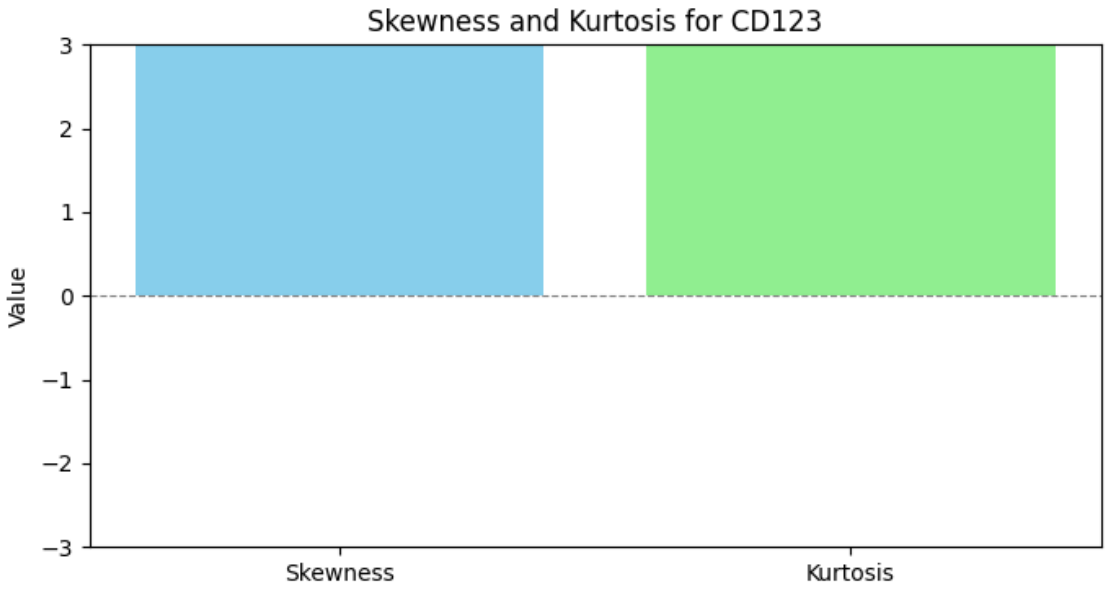

Skewness and Kurtosis for Flt3
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

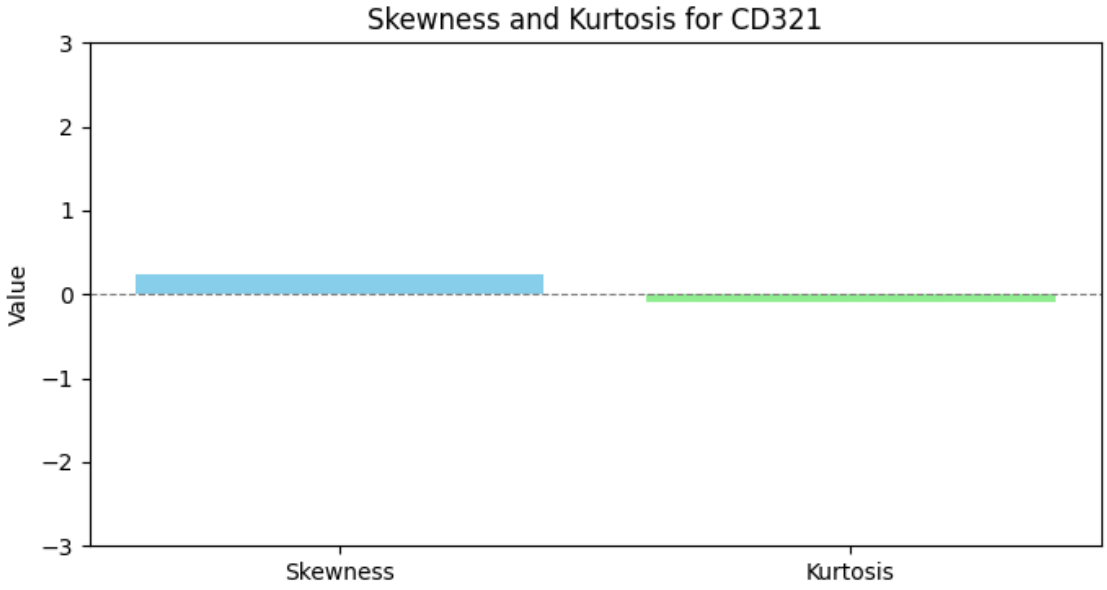

Skewness and Kurtosis for CD20
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

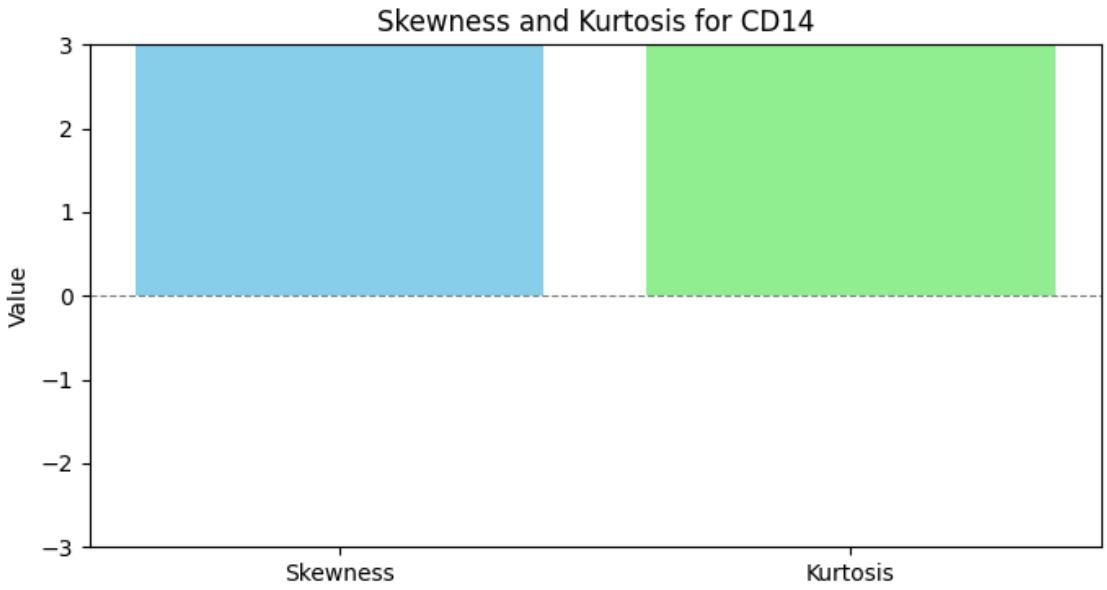

Skewness and Kurtosis for CXCR4
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

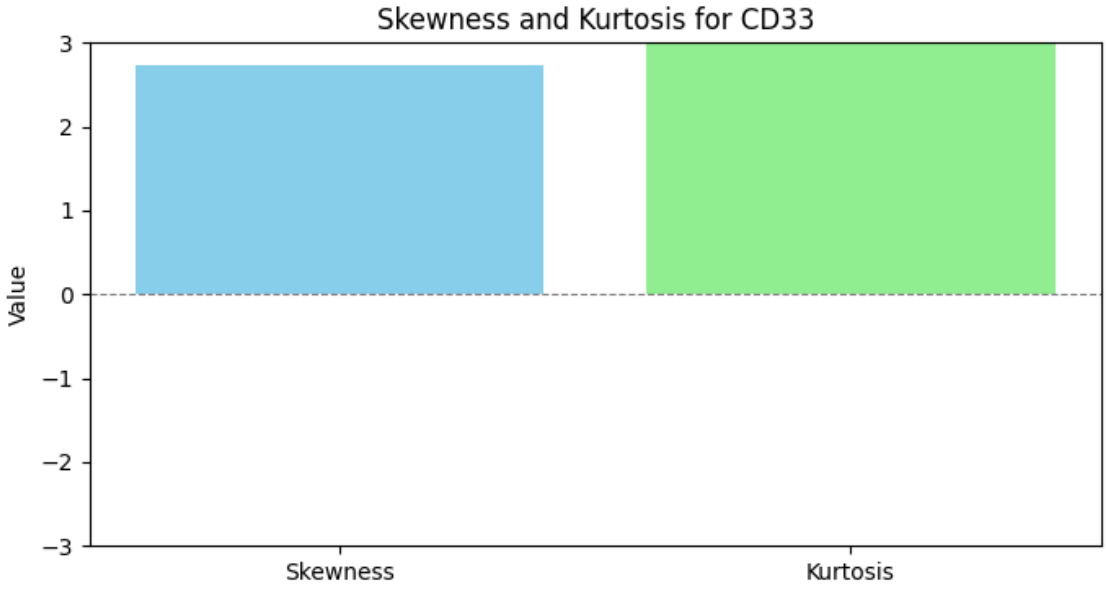

Skewness and Kurtosis for CD235ab
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

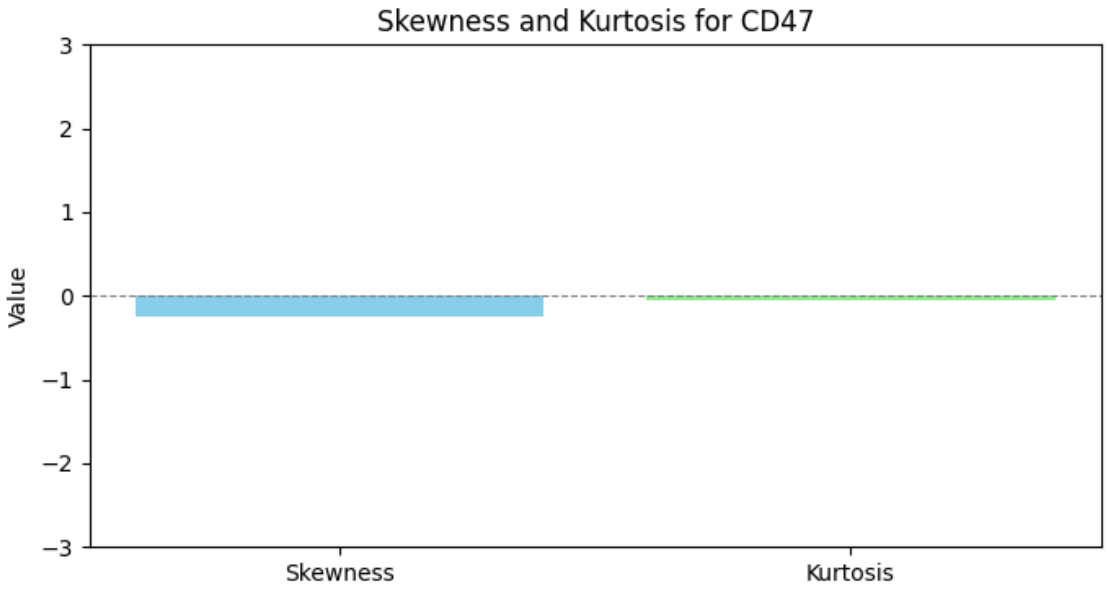

Skewness and Kurtosis for CD45
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

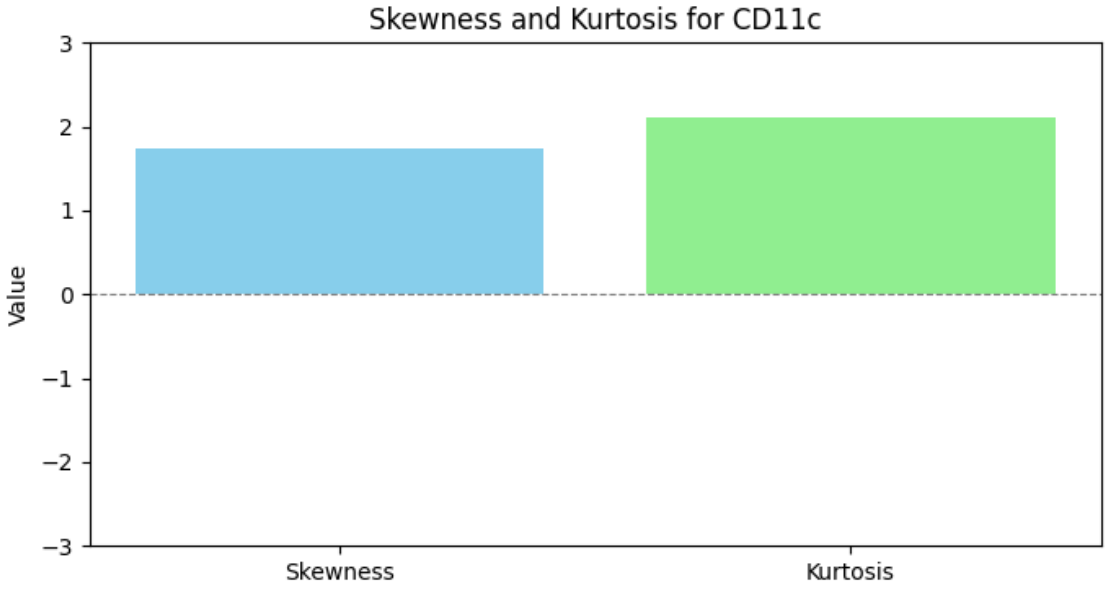

Skewness and Kurtosis for CD123
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

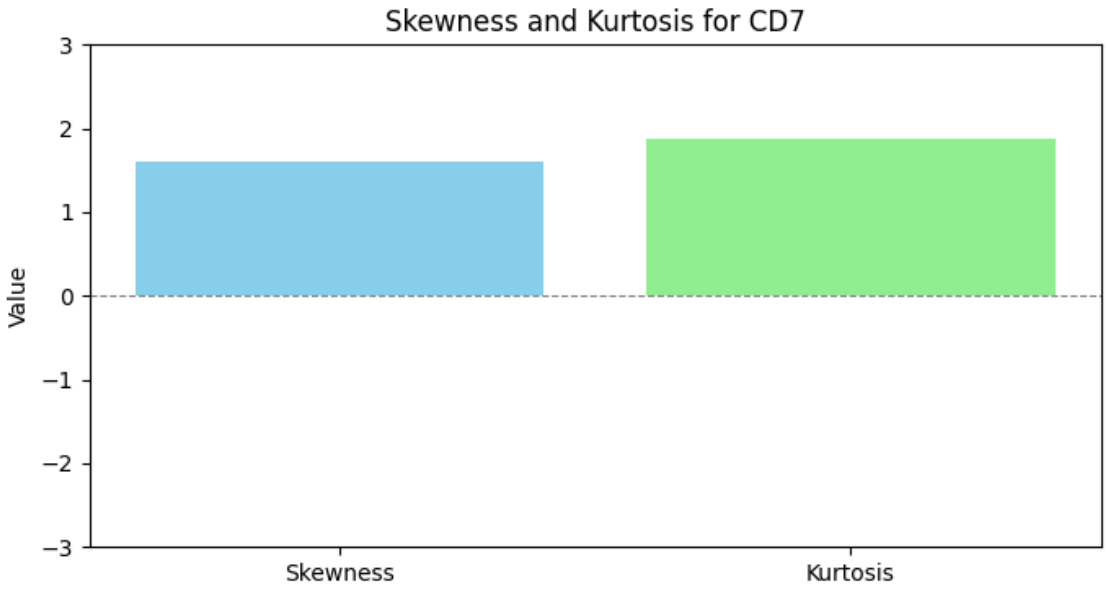

Skewness and Kurtosis for CD321
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

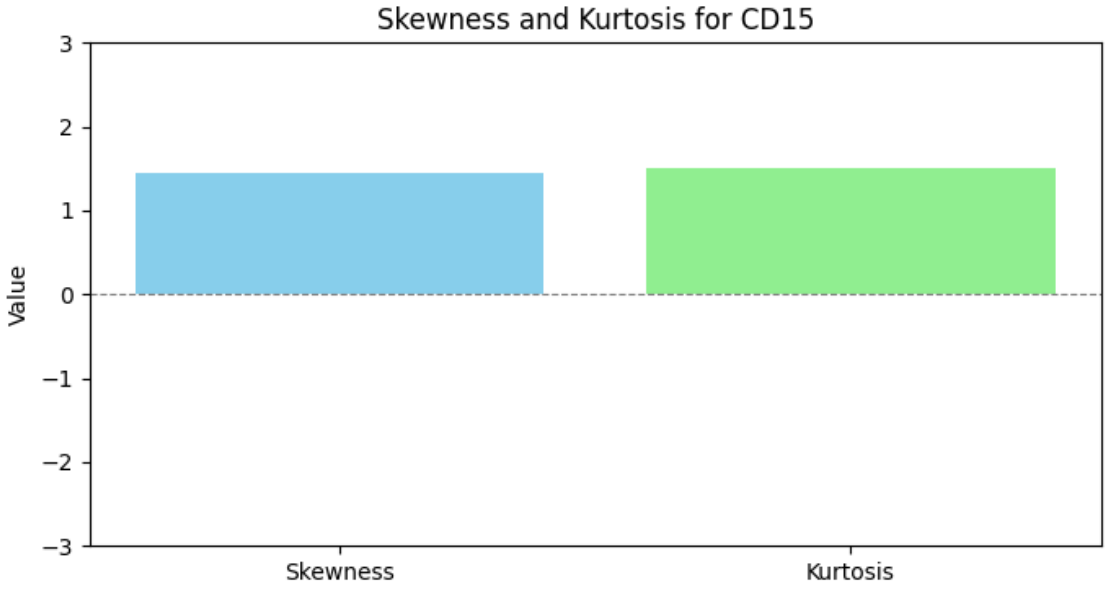

Skewness and Kurtosis for CD14
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

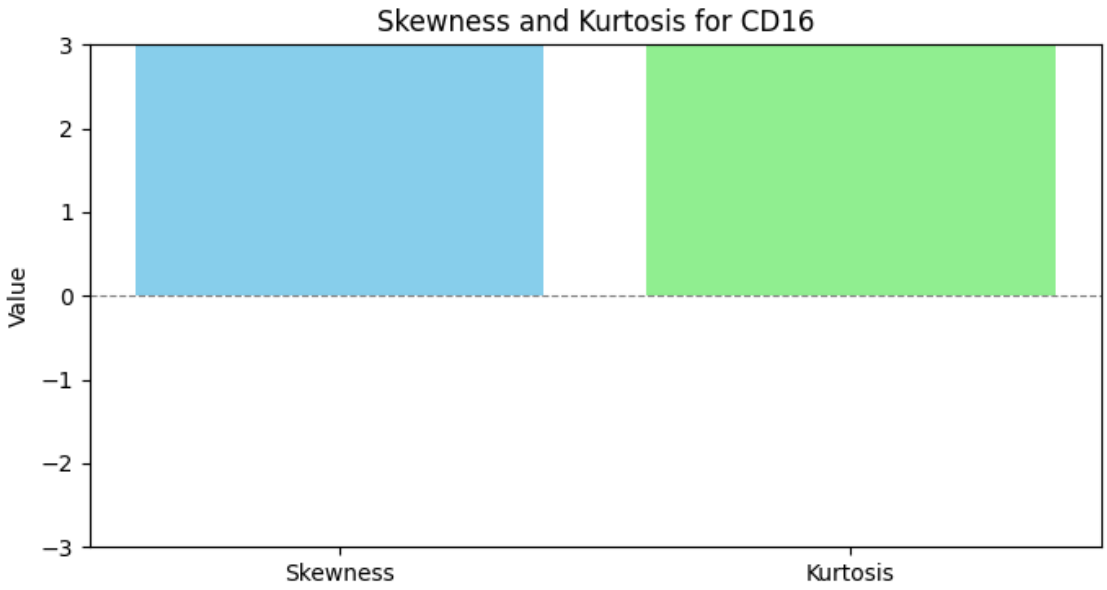

Skewness and Kurtosis for CD33
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

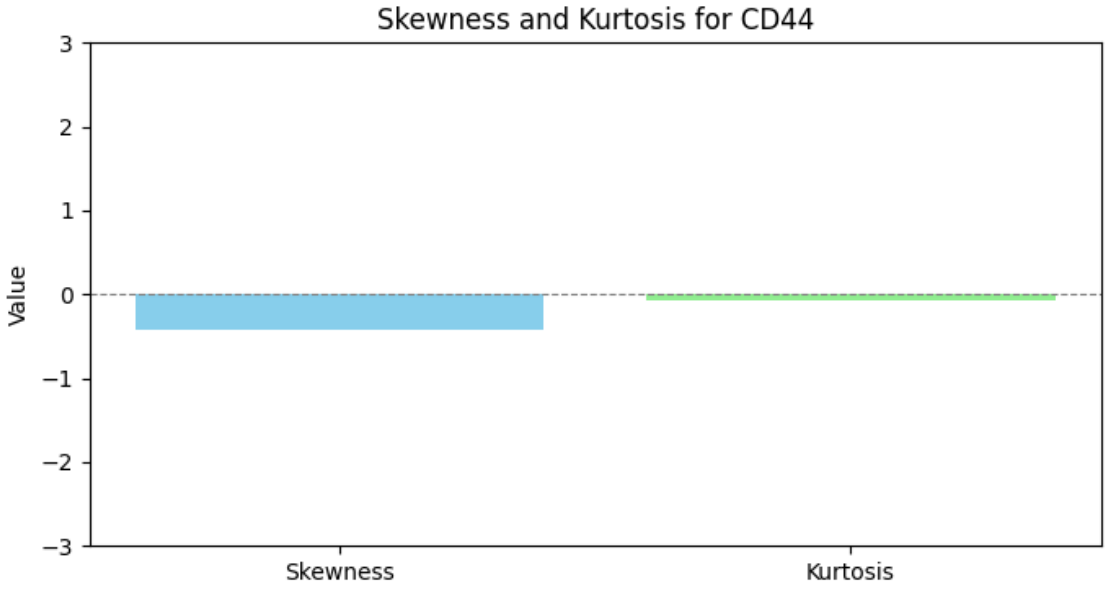

Skewness and Kurtosis for CD47
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

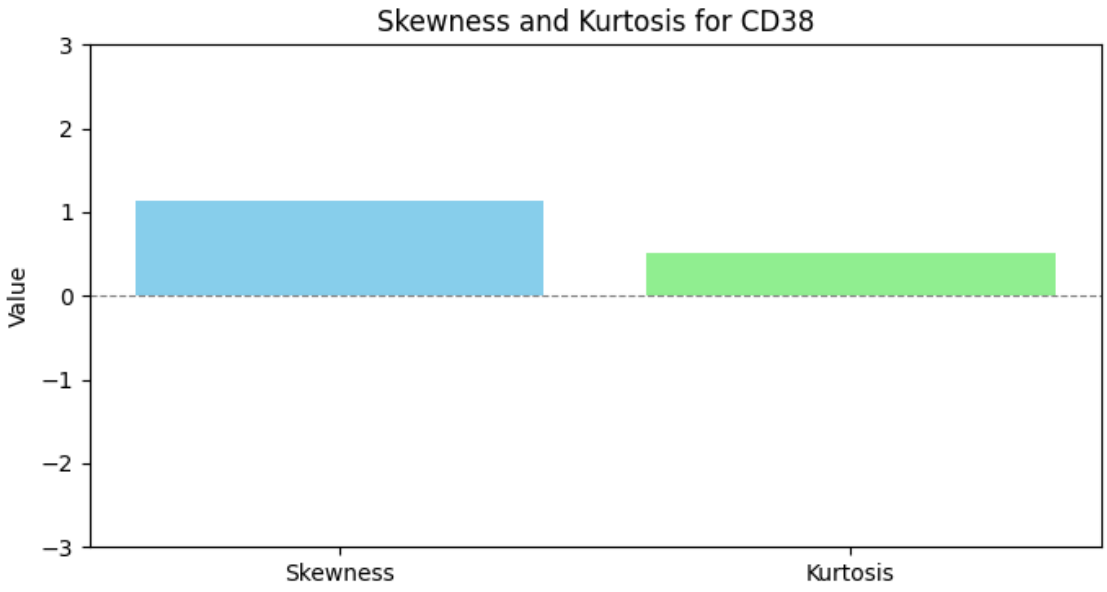

Skewness and Kurtosis for CD11c
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

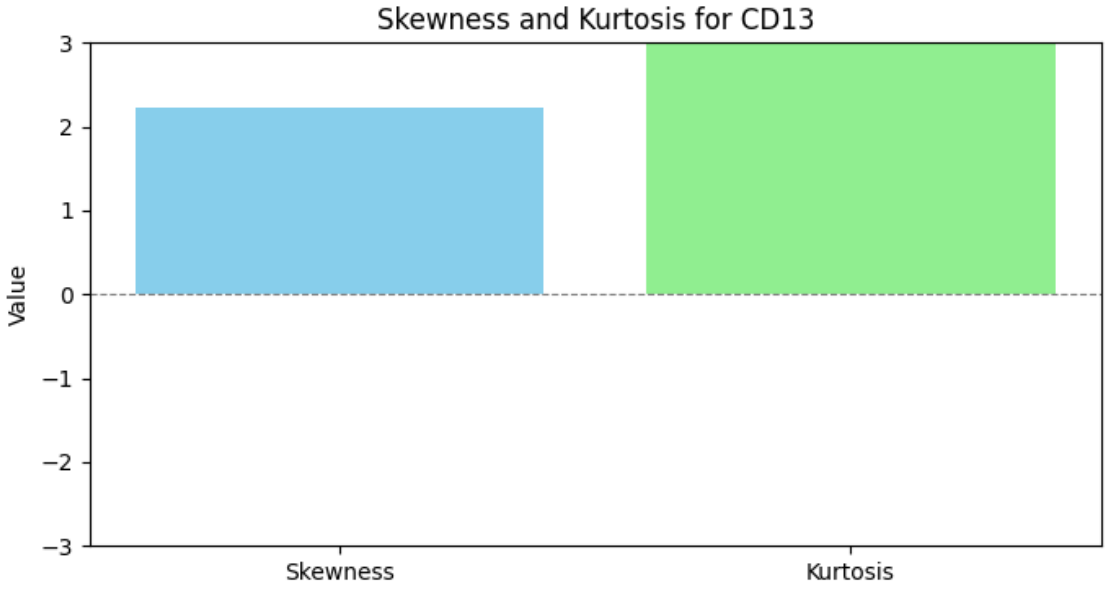

Skewness and Kurtosis for CD7
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

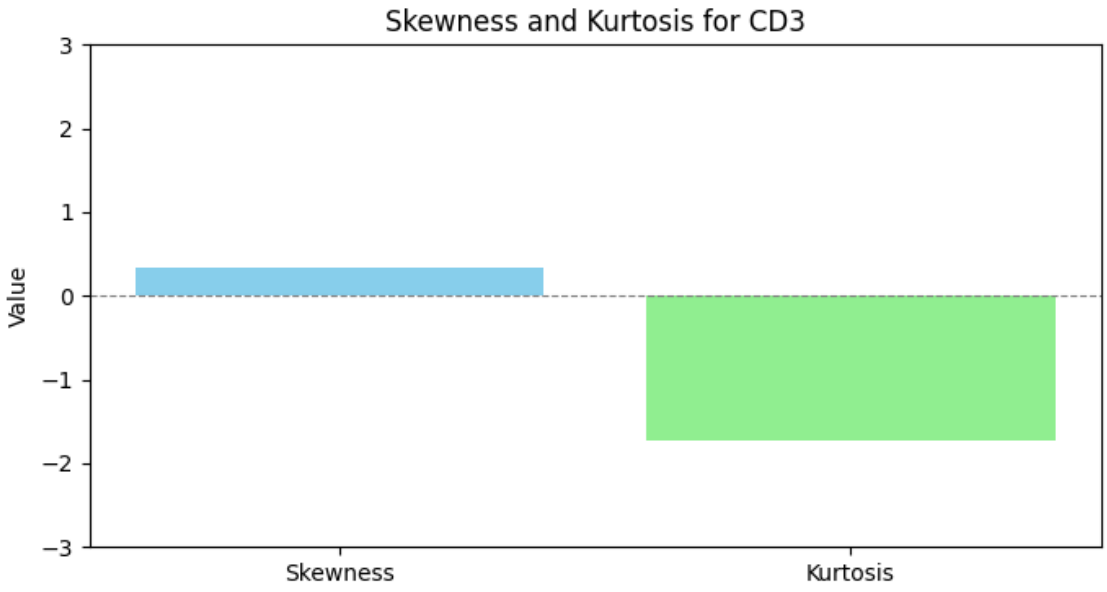

Skewness and Kurtosis for CD15
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

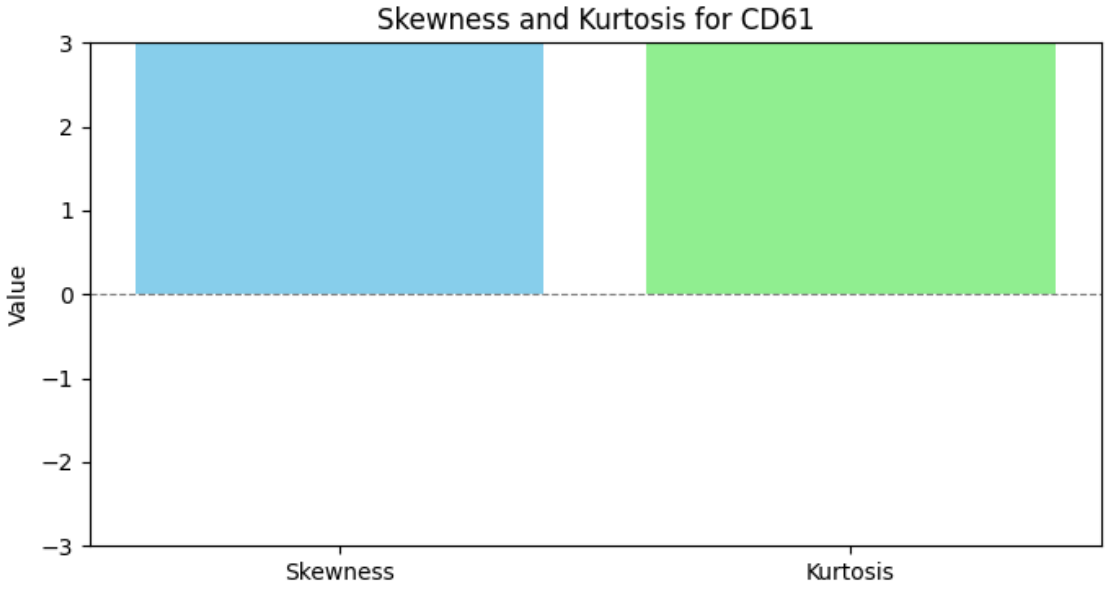

Skewness and Kurtosis for CD16
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

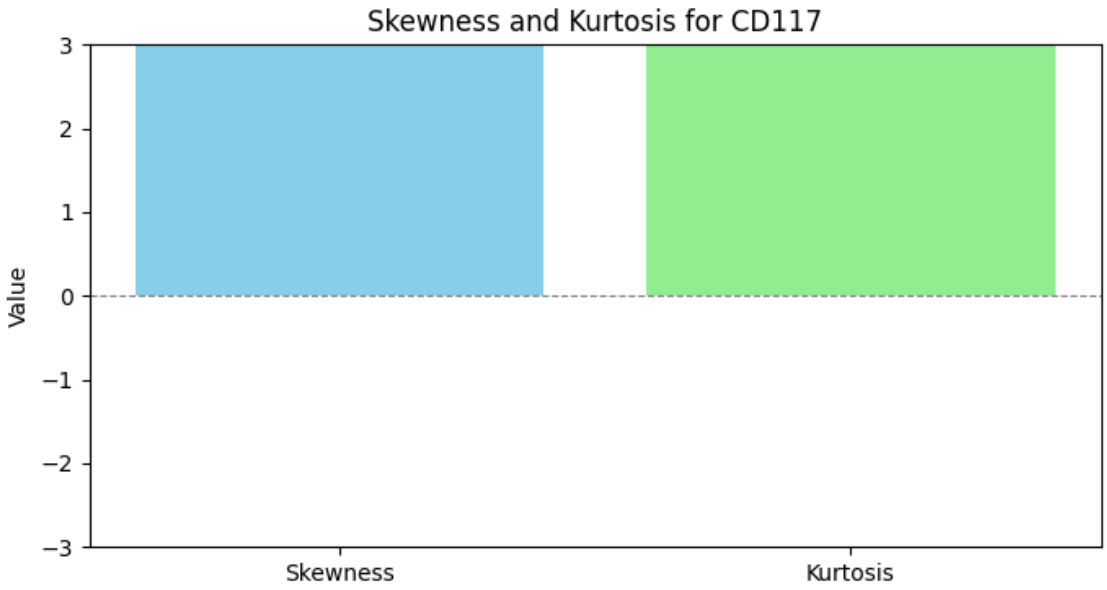

Skewness and Kurtosis for CD44
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

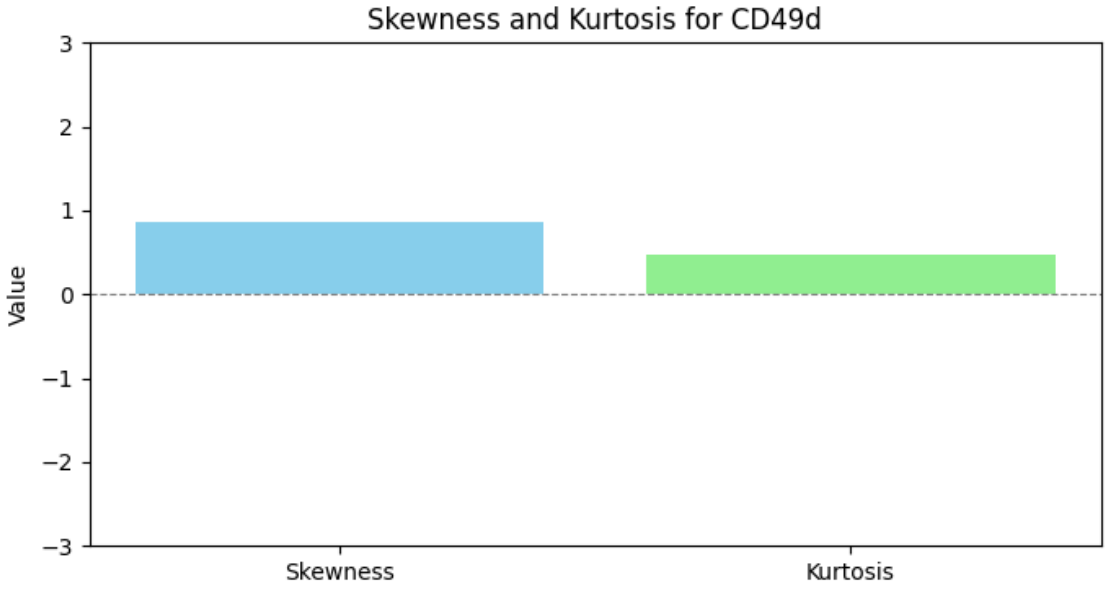

Skewness and Kurtosis for CD38
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

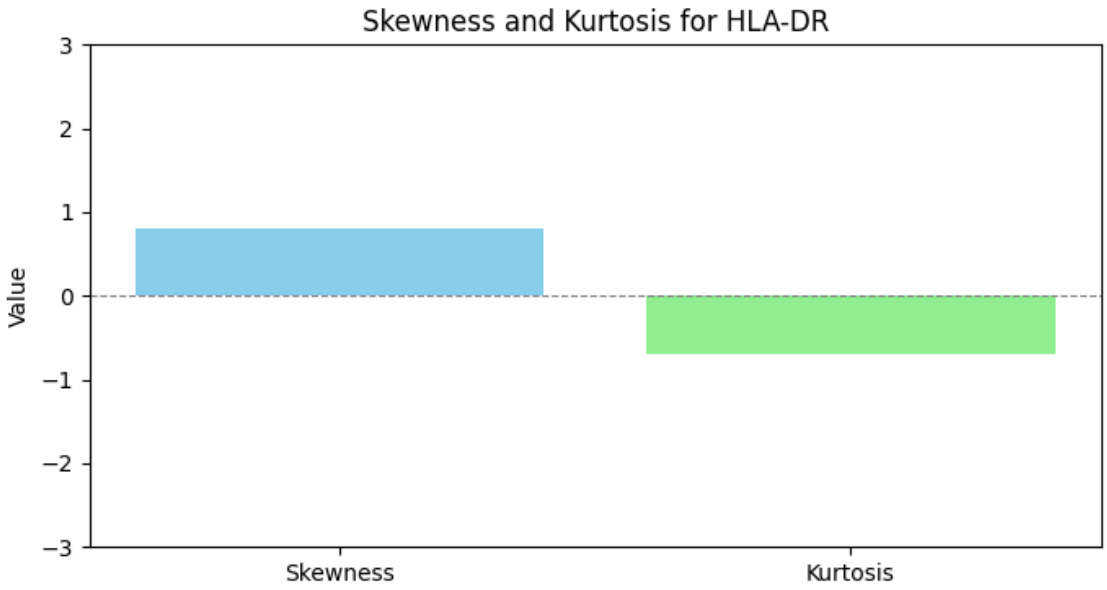

Skewness and Kurtosis for CD13
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

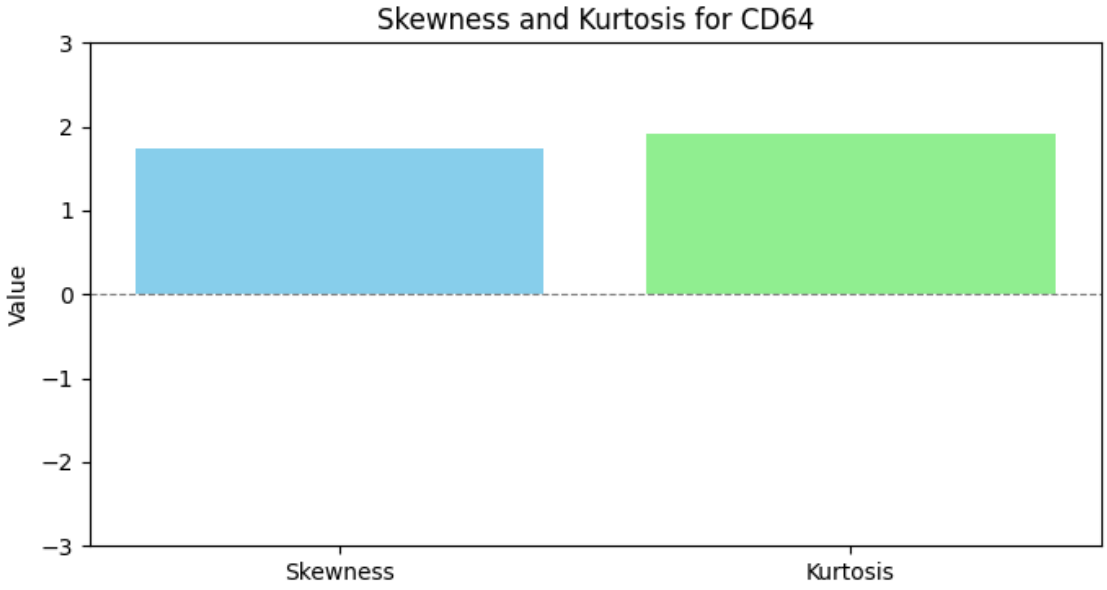

Skewness and Kurtosis for CD3
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

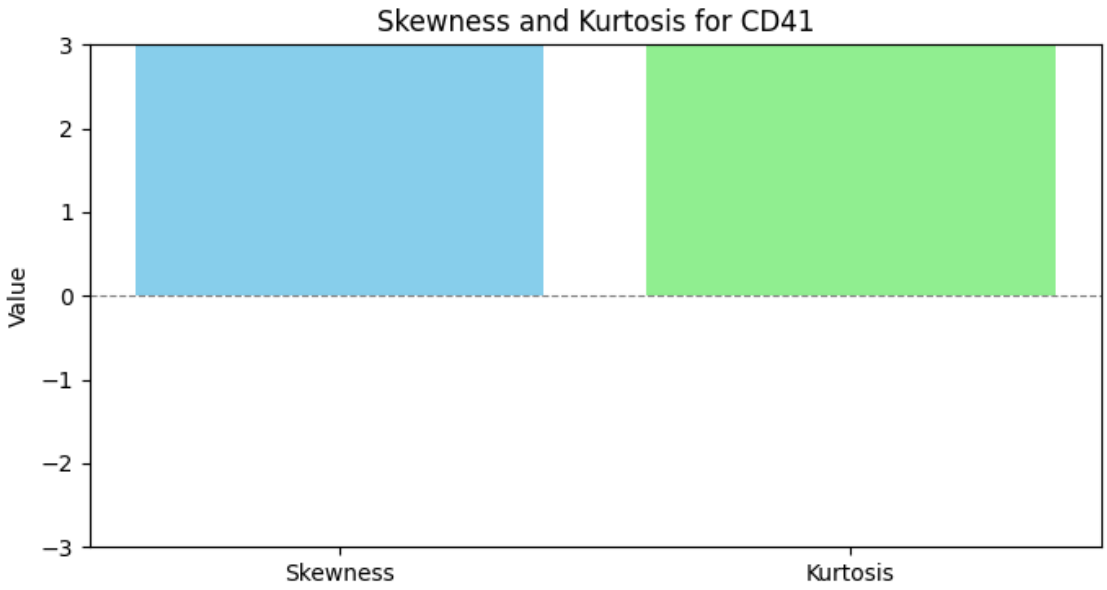

Skewness and Kurtosis for CD61
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

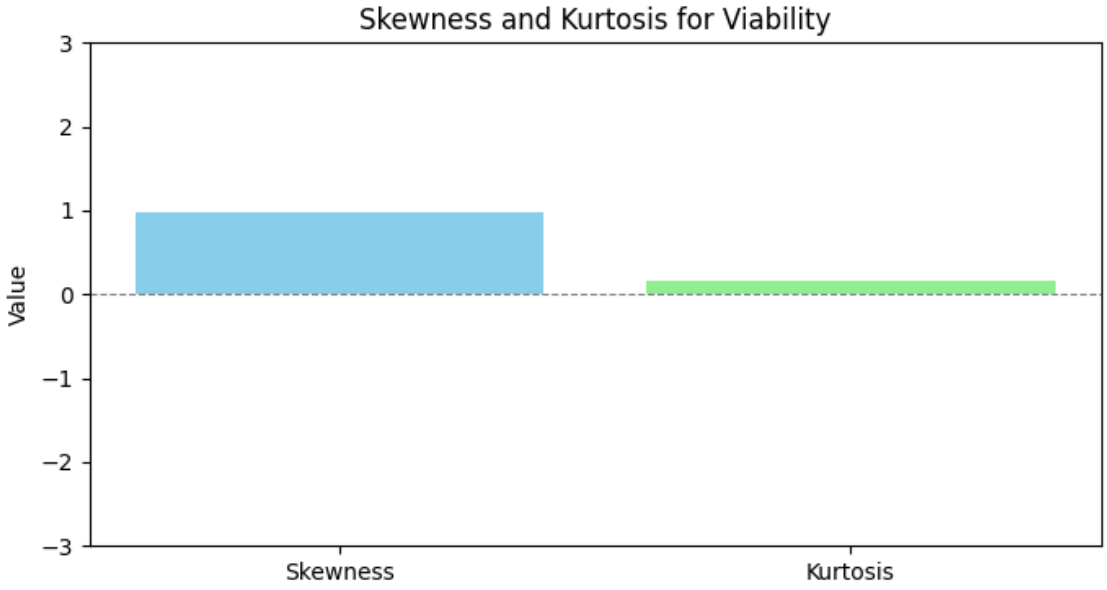

Skewness and Kurtosis for CD117
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

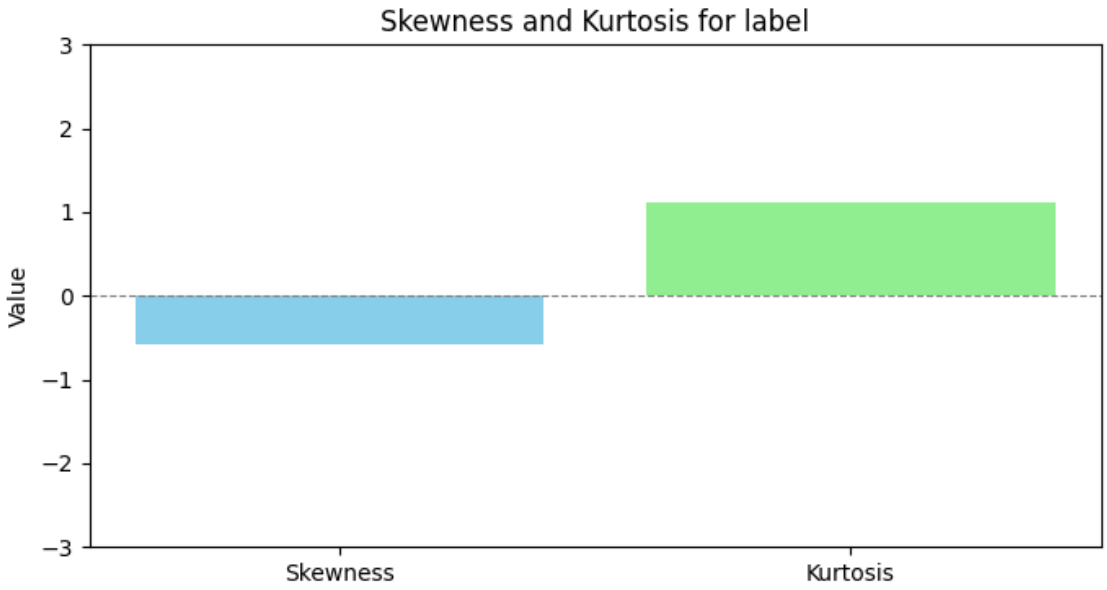

Skewness and Kurtosis for CD49d
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

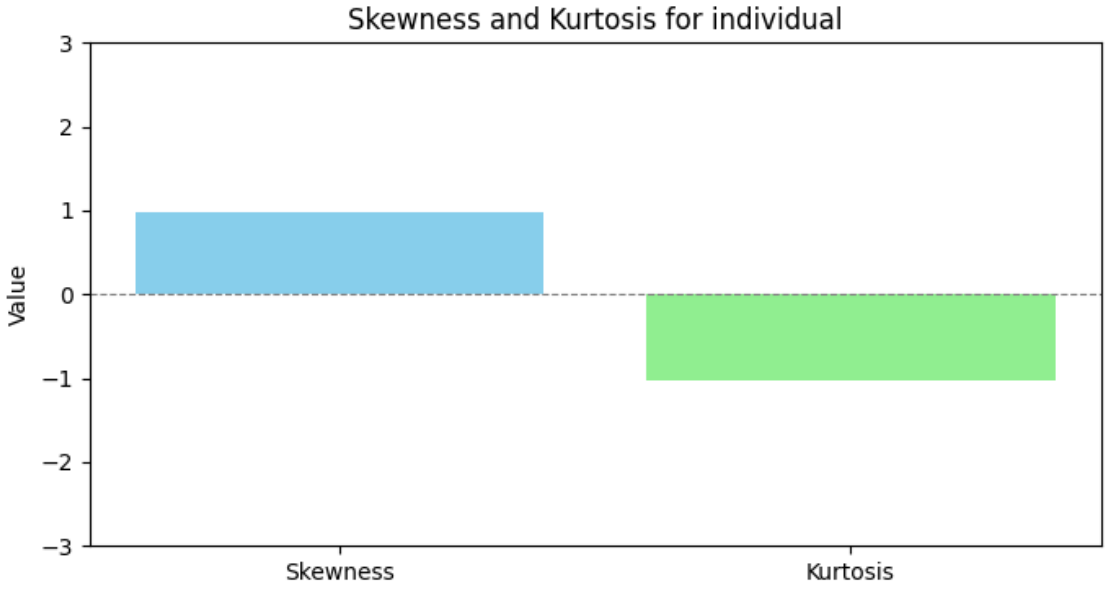

Skewness and Kurtosis for HLA-DR
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis


Skewness and Kurtosis for CD64
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis


Skewness and Kurtosis for CD41
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis


Skewness and Kurtosis for Viability
Value
3
2
1
0
−1
−2
−3
Skewness
Kurtosis

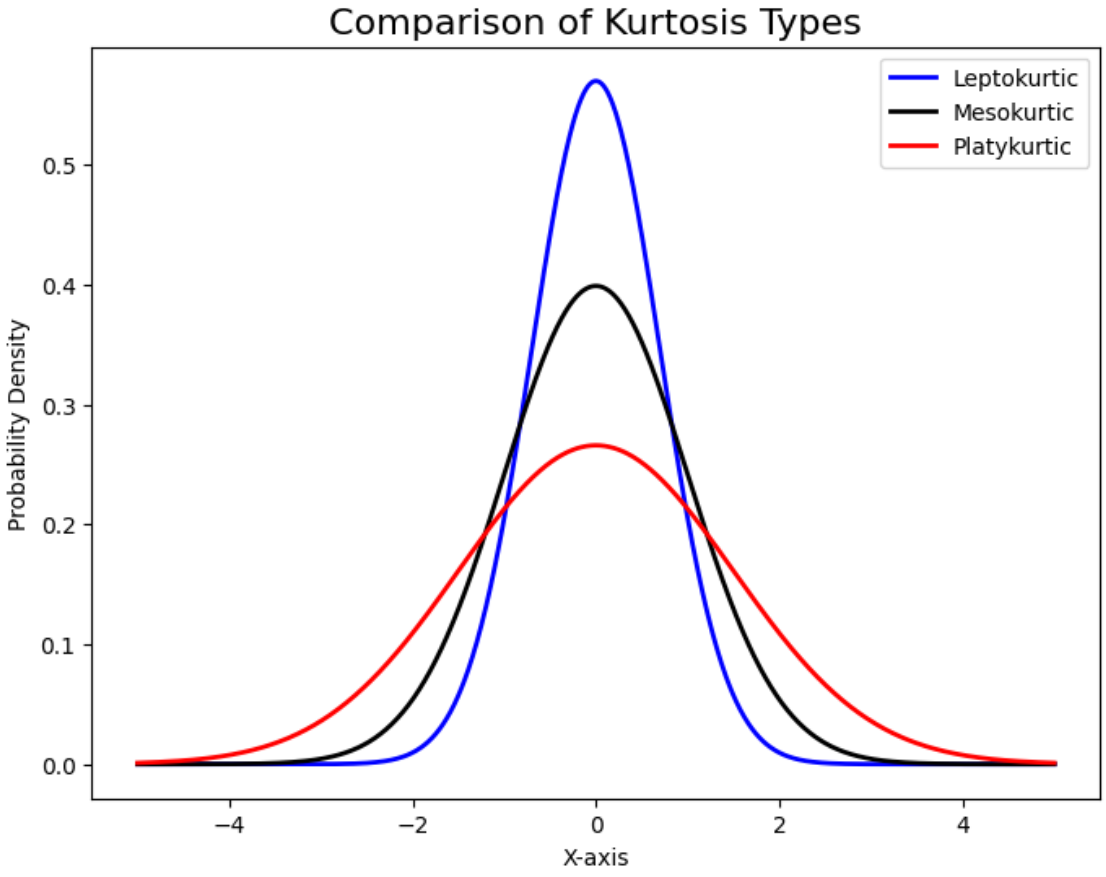




This code generates probability density curves for leptokurtic, mesokurtic, and platykurtic distributions using the normal distribution function. It then plots these curves on the same graph to compare different kurtosis types visually.

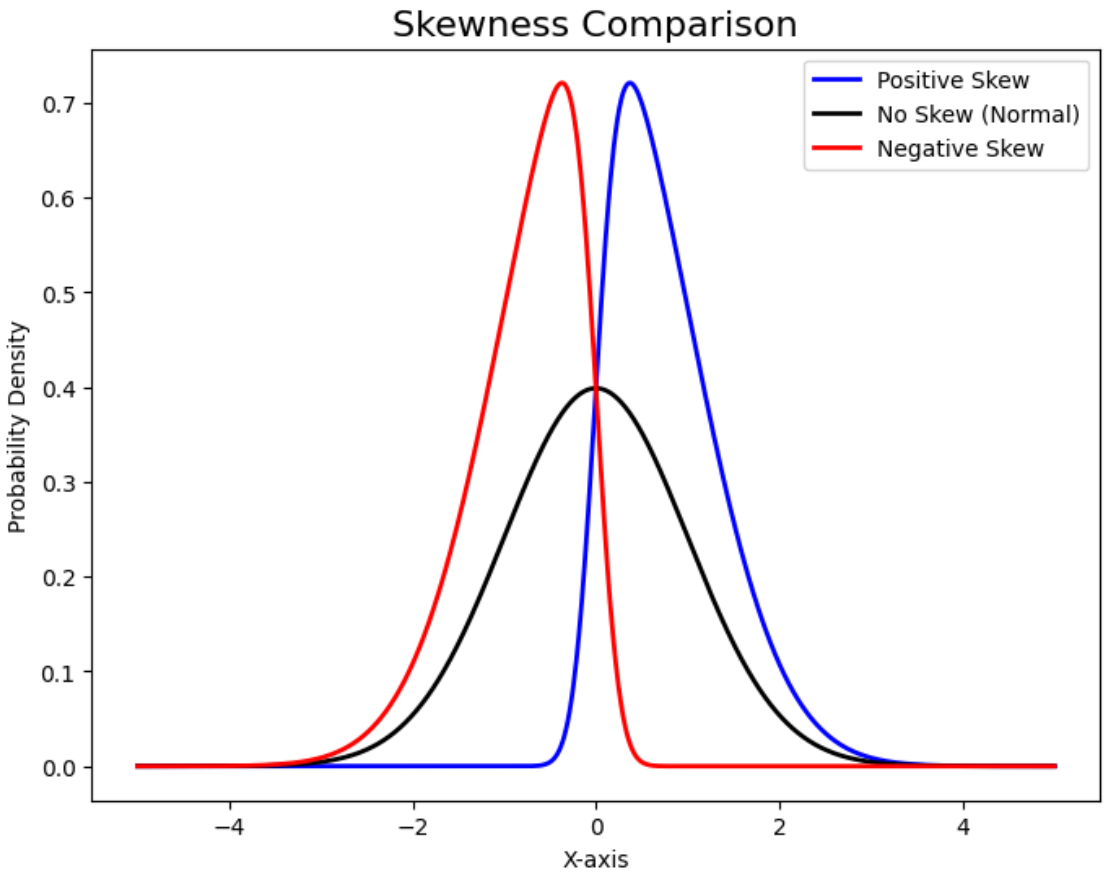
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
```

```
x = np.linspace(-5, 5, 1000)

mesokurtic = norm.pdf(x, 0, 1)
leptokurtic = norm.pdf(x, 0, 0.7)
platykurtic = norm.pdf(x, 0, 1.5)

plt.figure(figsize=(8, 6))
plt.plot(x, leptokurtic, label='Leptokurtic', color='blue', linewidth=2)
plt.plot(x, mesokurtic, label='Mesokurtic', color='black', linewidth=2)
plt.plot(x, platykurtic, label='Platykurtic', color='red', linewidth=2)

plt.title('Comparison of Kurtosis Types', fontsize=16)
plt.xlabel('X-axis')
plt.ylabel('Probability Density')

plt.legend()

plt.show()
```



This code generates and plots positive skew, negative skew, and no skew (normal distribution) using the skew normal distribution function. The graph visually compares how skewness affects the shape of the probability density curves.

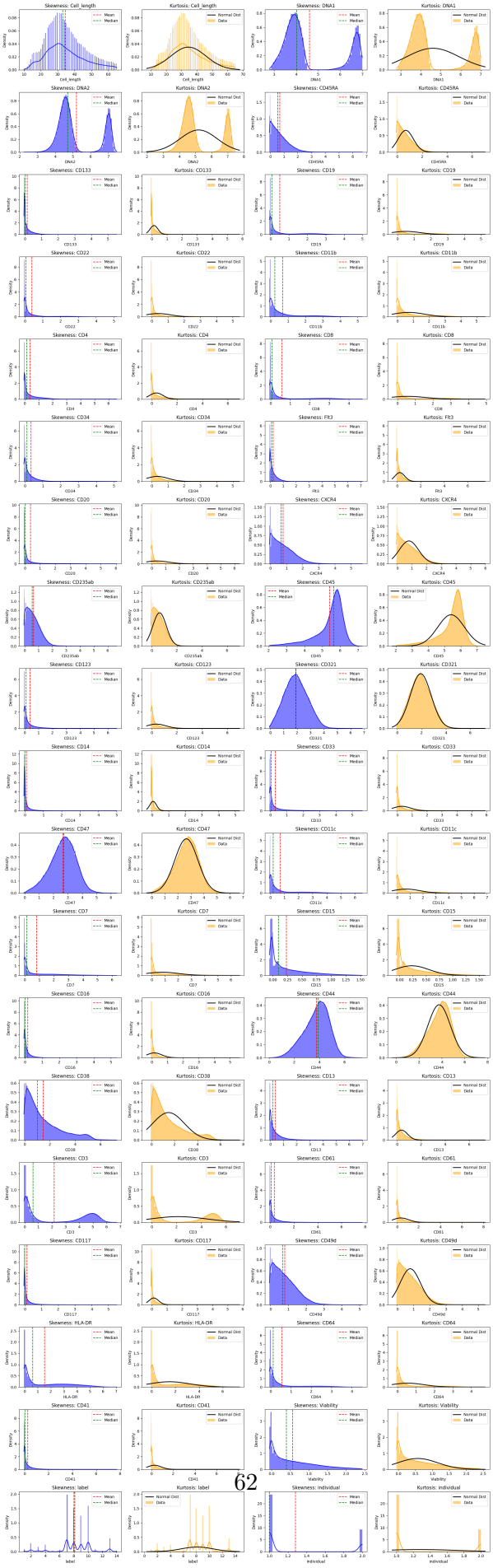
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import skewnorm, norm

x = np.linspace(-5, 5, 1000)

positive_skew = skewnorm.pdf(x, 5)
negative_skew = skewnorm.pdf(x, -5)
normal_dist = skewnorm.pdf(x, 0)

plt.figure(figsize=(8, 6))
plt.plot(x, positive_skew, label='Positive Skew', color='blue', linewidth=2)
plt.plot(x, normal_dist, label='No Skew (Normal)', color='black', linewidth=2)
plt.plot(x, negative_skew, label='Negative Skew', color='red', linewidth=2)
plt.title('Skewness Comparison', fontsize=16)
plt.xlabel('X-axis')
plt.ylabel('Probability Density')
plt.legend()
plt.show()
```



This code creates a grid of subplots, where it visualizes both skewness and kurtosis for each feature in the dataset. For skewness, histograms with kernel density estimation are plotted, while for kurtosis, histograms are compared to the normal distribution curve to show the tailedness.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm

filtered_df = df.drop(columns=['file_number', 'Event', 'Time','event_number'])
columns = filtered_df.columns
n_features = len(columns)

n_cols = 4
n_rows = int(np.ceil(n_features * 2 / n_cols))

fig, axes = plt.subplots(n_rows, n_cols, figsize=(18, n_rows * 3))
axes = axes.flatten()

def plot_skewness(column_data, column_name, ax):
```

```
    sns.histplot(column_data.dropna(), kde=True, stat="density", linewidth=0,␣
 ↪ax=ax, color='blue')
    ax.axvline(x=np.mean(column_data.dropna()), color='red', linestyle='--',␣
 ↪label='Mean')
    ax.axvline(x=np.median(column_data.dropna()), color='green',␣
 ↪linestyle='--', label='Median')
    ax.set_title(f'Skewness: {column_name}')
    ax.legend()

def plot_kurtosis(column_data, column_name, ax):
    sns.histplot(column_data.dropna(), kde=True, stat="density", linewidth=0,␣
 ↪ax=ax, color='orange', label='Data')
    xmin, xmax = ax.get_xlim()
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, np.mean(column_data.dropna()), np.std(column_data.dropna()))
    ax.plot(x, p, 'k', linewidth=2, label='Normal Dist')
    ax.set_title(f'Kurtosis: {column_name}')
    ax.legend()

for i, column in enumerate(columns):
    plot_skewness(filtered_df[column], column, axes[2*i])
    plot_kurtosis(filtered_df[column], column, axes[2*i + 1])

plt.tight_layout()
plt.show()
```


Skewness: Cell_length
Kurtosis: Cell_length
Skewness: DNA1
Kurtosis: DNA1
Skewness: DNA2
Kurtosis: DNA2
Skewness: CD45RA
Kurtosis: CD45RA
Skewness: CD133
Kurtosis: CD133
Skewness: CD19
Kurtosis: CD19
Skewness: CD22
Kurtosis: CD22
Skewness: CD11b
Kurtosis: CD11b
Skewness: CD4
Kurtosis: CD4
Skewness: CD8
Kurtosis: CD8
Skewness: CD34
Kurtosis: CD34
Skewness: Flt3
Kurtosis: Flt3
Skewness: CD20
Kurtosis: CD20
Skewness: CXCR4
Kurtosis: CXCR4
Skewness: CD235ab
Kurtosis: CD235ab
Skewness: CD45
Kurtosis: CD45
Skewness: CD123
Kurtosis: CD123
Skewness: CD321
Kurtosis: CD321
Skewness: CD14
Kurtosis: CD14
Skewness: CD33
Kurtosis: CD33
Skewness: CD47
Kurtosis: CD47
Skewness: CD11c
Kurtosis: CD11c
Skewness: CD7
Kurtosis: CD7
Skewness: CD15
Kurtosis: CD15
Skewness: CD16
Kurtosis: CD16
Skewness: CD44
Kurtosis: CD44
Skewness: CD38
Kurtosis: CD38
Skewness: CD13
Kurtosis: CD13
Skewness: CD3
Kurtosis: CD3
Skewness: CD61
Kurtosis: CD61
Skewness: CD117
Kurtosis: CD117
Skewness: CD49d
Kurtosis: CD49d
Skewness: HLA-DR
Kurtosis: HLA-DR
Skewness: CD64
Kurtosis: CD64
Skewness: CD41
Kurtosis: CD41
Skewness: Viability
Kurtosis: Viability
Skewness: label
Kurtosis: label
Skewness: individual
Kurtosis: individual
Mean
Median
Normal Dist
Data
Density

Dimensionality reduction

PCA

pca and tsne

[ ]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Standardize all columns in the dataset
df[:] = scaler.fit_transform(df)

# Check the standardized data
print(df.head())
```

```
      Event      Time  Cell_length      DNA1      DNA2    CD45RA     CD133  \
0  0.000000  0.003796     0.218182  0.380681  0.454713  0.032599  0.005102
1  0.000004  0.005267     0.454545  0.368682  0.492802  0.112418  0.003545
2  0.000008  0.009891     0.400000  0.249642  0.410614  0.097929  0.004631
3  0.000011  0.010010     0.345455  0.348593  0.495353  0.072765  0.005455
4  0.000015  0.010857     0.272727  0.282425  0.433546  0.007186  0.004974

       CD19      CD22     CD11b  …     CD117     CD49d    HLA-DR      CD64  \
0  0.010180  0.023713  0.009222  …  0.019914  0.174915  0.242242  0.011544
1  0.008208  0.025250  0.162862  …  0.026499  0.049100  0.077290  0.044364
2  0.026137  0.002753  0.010595  …  0.018686  0.507480  0.192154  0.010323
3  0.008008  0.002543  0.148887  …  0.022328  0.268010  0.027916  0.009779
4  0.027438  0.106009  0.219187  …  0.009253  0.045858  0.035906  0.029363

       CD41  Viability  file_number  event_number  label  individual
0  0.007238   0.283583          0.0      0.000765    0.0           0
1  0.119109   0.248639          0.0      0.001360    0.0           0
2  0.006151   0.281539          0.0      0.004311    0.0           0
3  0.004141   0.012628          0.0      0.004411    0.0           0
4  0.002283   0.136999          0.0      0.005074    0.0           0

[5 rows x 42 columns]
```

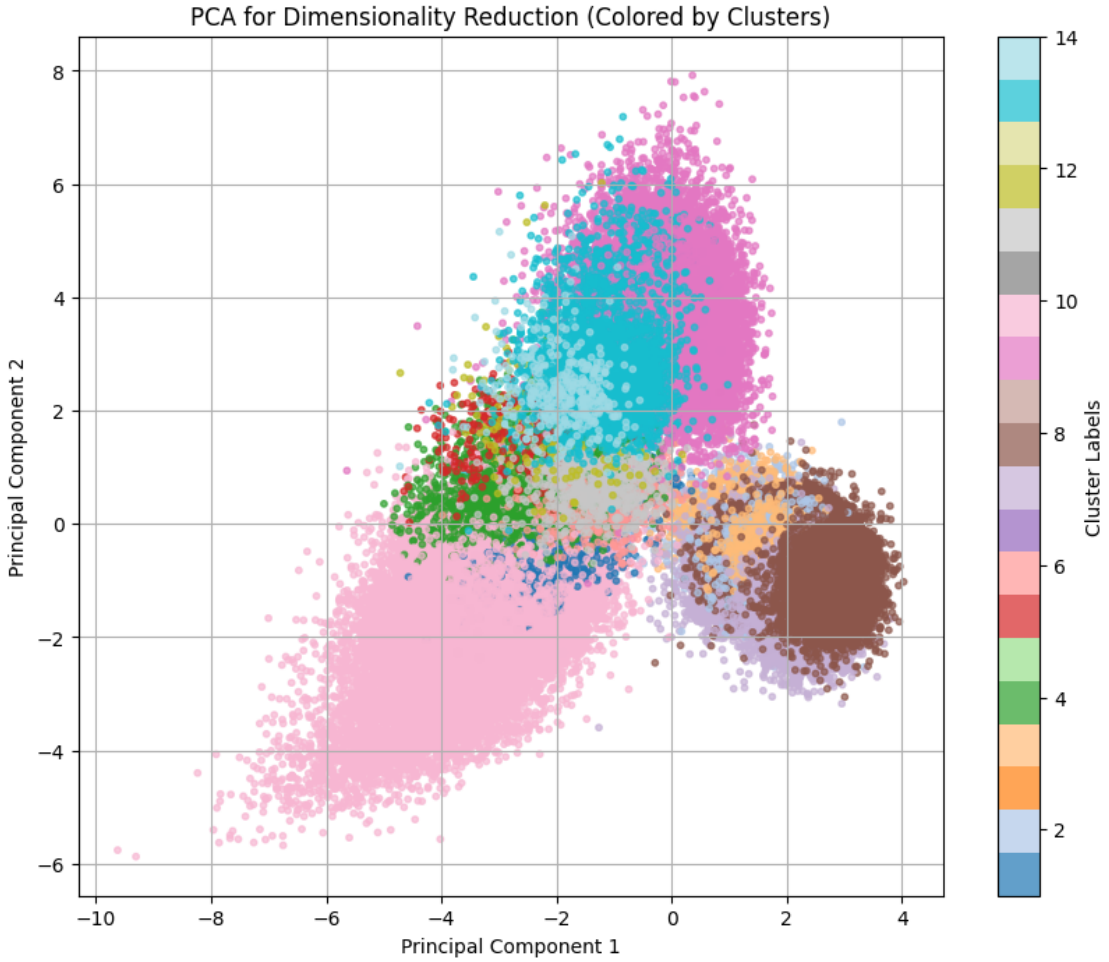
```
<ipython-input-2-06590f96526d>:8: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value
'[0.00000000e+00 3.76469171e-06 7.52938342e-06 … 9.99992471e-01
 9.99996235e-01 1.00000000e+00]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  df[:] = scaler.fit_transform(df)
```

```
<ipython-input-2-06590f96526d>:8: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value
'[0.21818182 0.45454545 0.4        … 0.56363636 0.52727273 0.56363636]' has
dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df[:] = scaler.fit_transform(df)
<ipython-input-2-06590f96526d>:8: FutureWarning: Setting an item of incompatible
dtype is deprecated and will raise in a future error of pandas. Value
'[0.00076479 0.00135962 0.0043113  … 0.25667877 0.25669127 0.25672626]' has
dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df[:] = scaler.fit_transform(df)
```

[ ]:
```python
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np

# Assuming df is already loaded and 'label' column exists for cluster labeling
filtered_df = df.drop(columns=['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual'])
labels = df['label']  # Assuming you have a 'label' column indicating clusters

imputer = SimpleImputer(strategy='mean')
filled_df = imputer.fit_transform(filtered_df)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(filled_df)

# Perform PCA
pca = PCA(n_components=4)  # Set to 4 components for demonstration
pca_transformed = pca.fit_transform(scaled_data)

explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
std_dev = np.sqrt(pca.explained_variance_)

# Create a summary table for PCA results
pca_summary = pd.DataFrame({
    'Standard deviation': std_dev,
    'Proportion of Variance': explained_variance,
    'Cumulative Proportion': cumulative_variance
}, index=[f'PC{i+1}' for i in range(len(std_dev))])

print(pca_summary)
```

```
# Scatter plot for the first two components, with different colors for␣
 ↪different clusters
plt.figure(figsize=(10, 8))
scatter = plt.scatter(pca_transformed[:, 0], pca_transformed[:, 1], c=labels,␣
 ↪cmap='tab20', s=10, alpha=0.7)
plt.colorbar(scatter, label='Cluster Labels')
plt.title('PCA for Dimensionality Reduction (Colored by Clusters)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

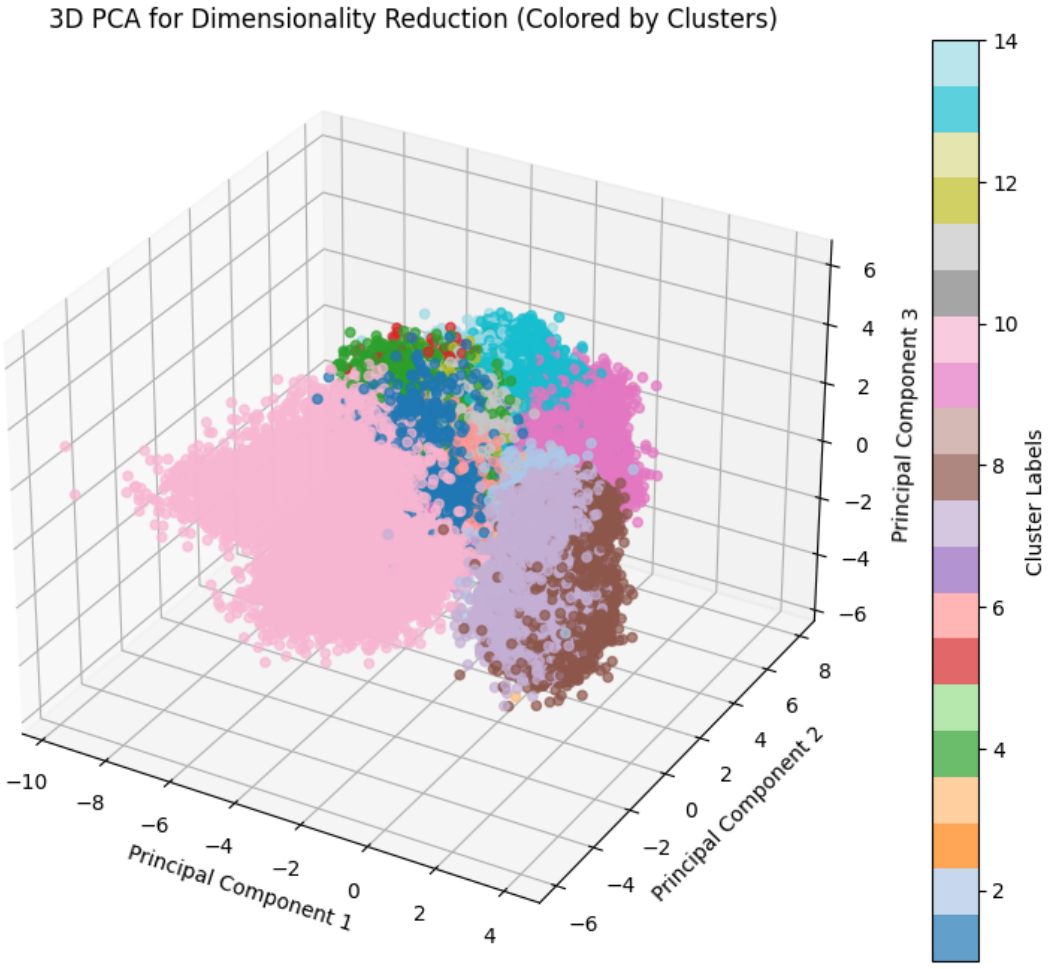
	Standard deviation	Proportion of Variance	Cumulative Proportion
PC1	2.327669	0.154801	0.154801
PC2	1.957437	0.109473	0.264273
PC3	1.877982	0.100766	0.365039
PC4	1.606712	0.073758	0.438797



[ ]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Assuming df is already loaded and 'label' column exists for cluster labeling
filtered_df = df.drop(columns=['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual'])
labels = df['label']  # Assuming you have a 'label' column indicating clusters

imputer = SimpleImputer(strategy='mean')
filled_df = imputer.fit_transform(filtered_df)

scaler = StandardScaler()
scaled_data = scaler.fit_transform(filled_df)

# Perform PCA for 4 components
pca = PCA(n_components=4)
pca_transformed = pca.fit_transform(scaled_data)

explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)
std_dev = np.sqrt(pca.explained_variance_)

# Create a summary table for PCA results
pca_summary = pd.DataFrame({
    'Standard deviation': std_dev,
    'Proportion of Variance': explained_variance,
    'Cumulative Proportion': cumulative_variance
}, index=[f'PC{i+1}' for i in range(len(std_dev))])

print(pca_summary)

# 3D scatter plot for the first three components, with different colors for␣
 ↪different clusters
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(pca_transformed[:, 0], pca_transformed[:, 1],␣
 ↪pca_transformed[:, 2],
                     c=labels, cmap='tab20', s=20, alpha=0.7)
```

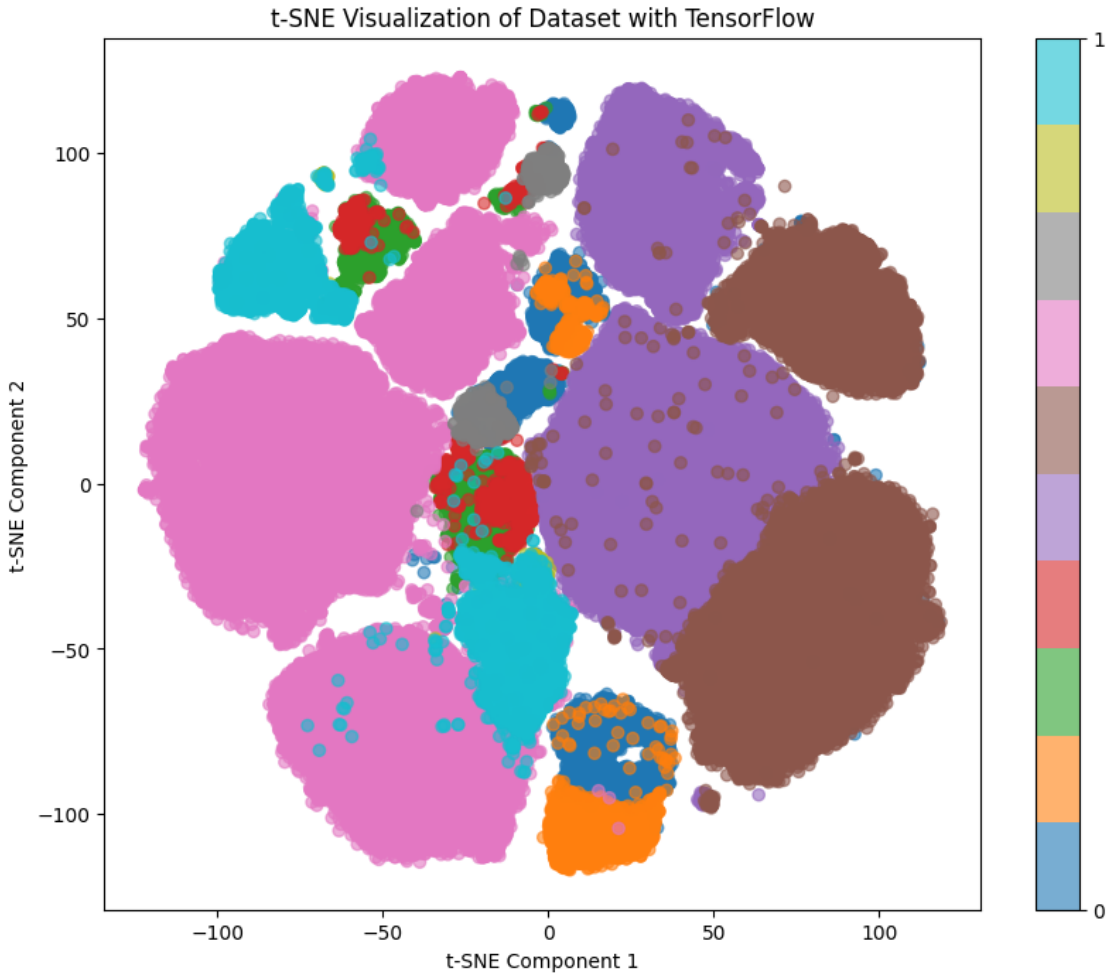
```
# Add labels and colorbar
ax.set_title('3D PCA for Dimensionality Reduction (Colored by Clusters)')
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
fig.colorbar(scatter, ax=ax, label='Cluster Labels')

plt.show()
```

	Standard deviation	Proportion of Variance	Cumulative Proportion
PC1	2.327669	0.154801	0.154801
PC2	1.957437	0.109473	0.264273
PC3	1.877982	0.100766	0.365039
PC4	1.606712	0.073758	0.438797



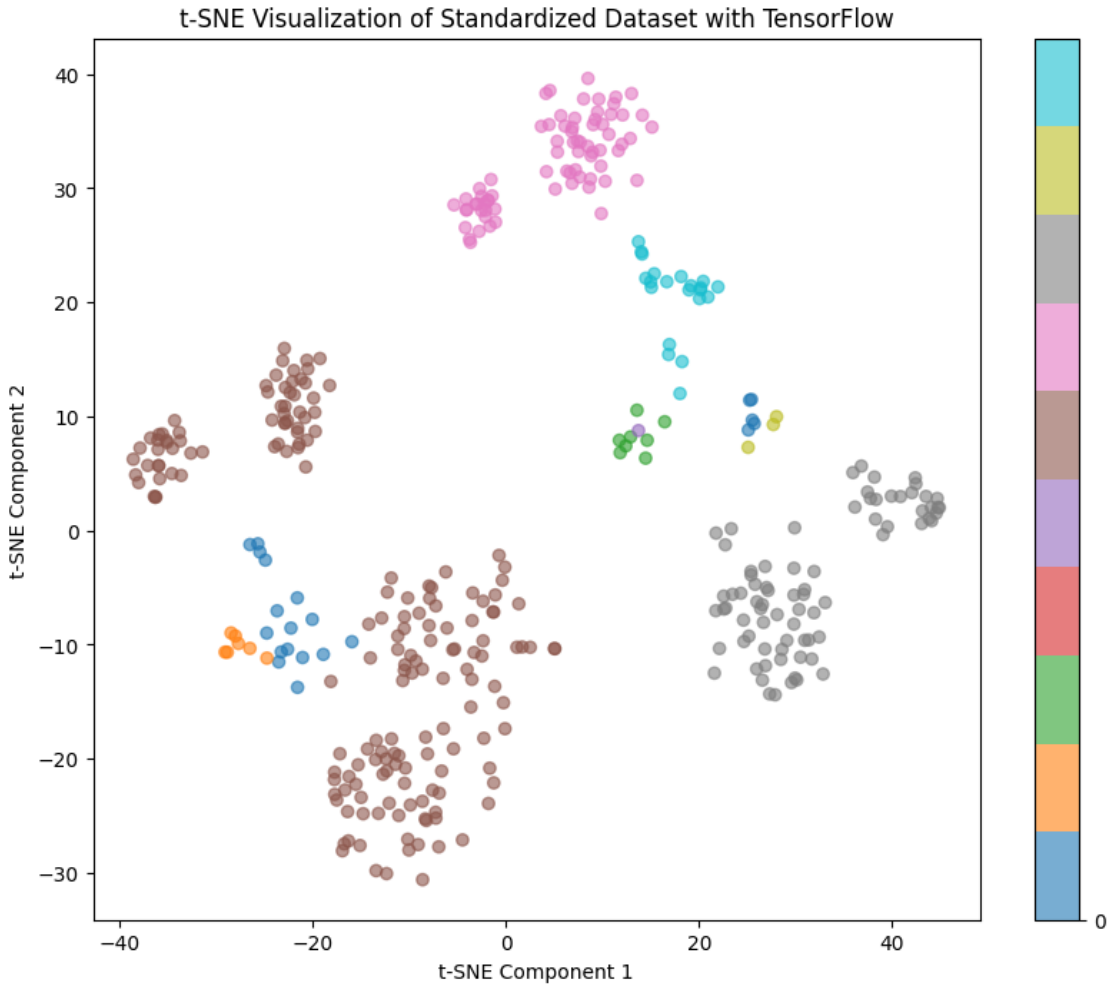
[ ]:
```
import pandas as pd
import tensorflow as tf
```

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual']
features = df.drop(columns=exclude_columns)
labels = df['label']

features = features.astype('float32') / features.max()
features_tf = tf.convert_to_tensor(features)
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_result = tsne.fit_transform(features_tf)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=labels,␣
 ↪cmap='tab10', alpha=0.6)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Dataset with TensorFlow')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



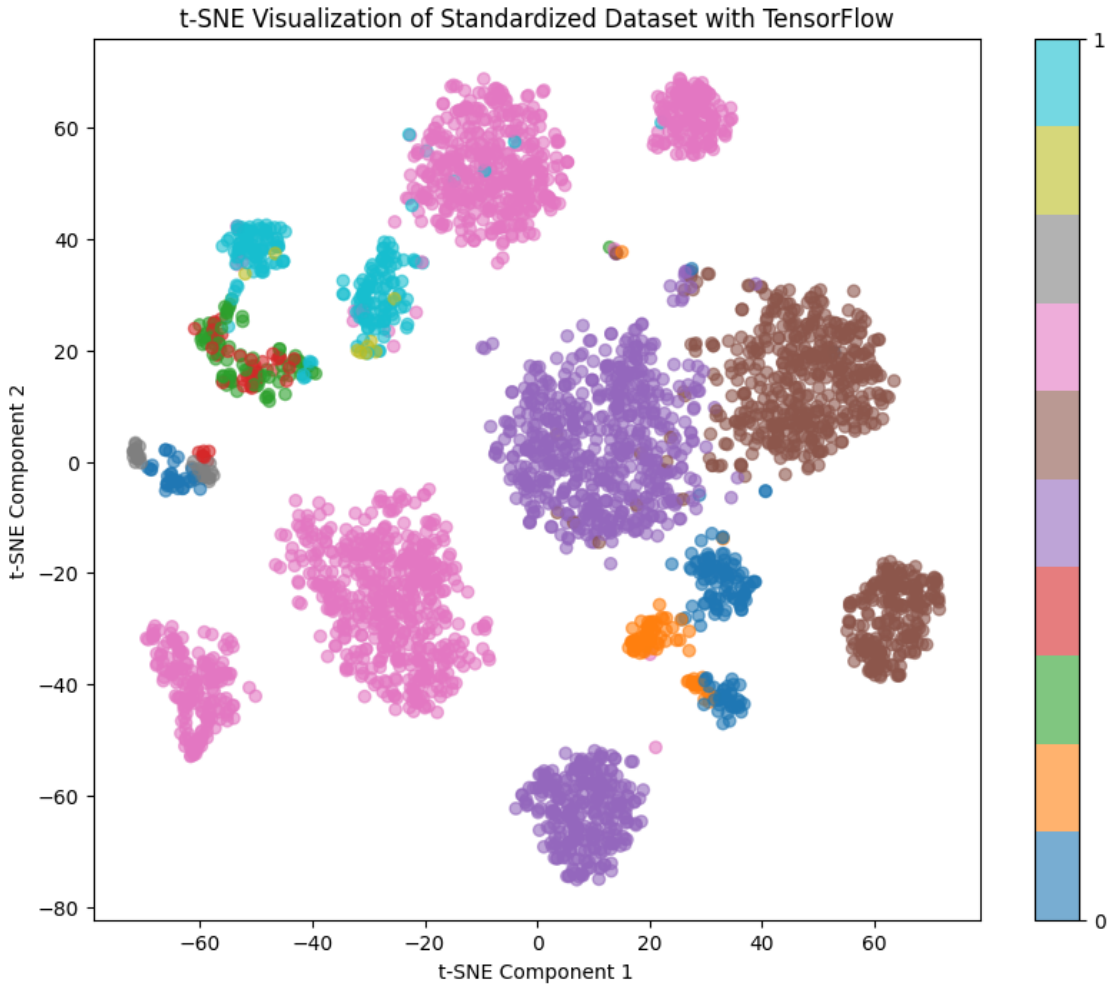
[ ]:
```
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

sample_df = df.sample(n=1000, random_state=42)  # Reduce the sample size to␣
 ↪1000 rows

# Exclude specific columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual']
features = sample_df.drop(columns=exclude_columns)
labels = sample_df['label']

# Standardize the features
```

```
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Convert to TensorFlow tensor
features_tf = tf.convert_to_tensor(features_scaled)

# Apply t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_result = tsne.fit_transform(features_tf)

# Plot the results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=labels,␣
 ↪cmap='tab10', alpha=0.6)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Standardized Dataset with TensorFlow')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



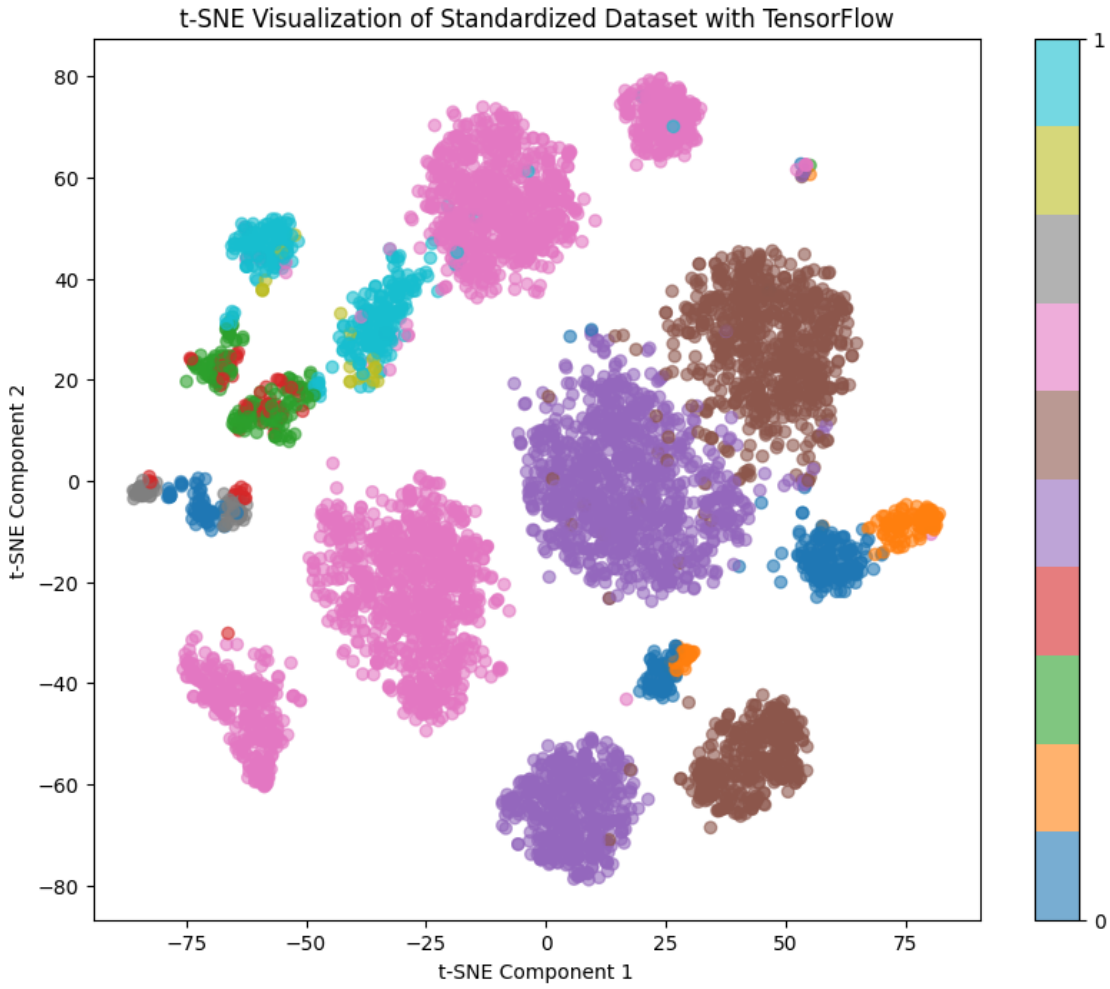
[ ]:
```
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

sample_df = df.sample(n=9000, random_state=42)  # Reduce the sample size to␣
 ↪1000 rows

# Exclude specific columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual']
features = sample_df.drop(columns=exclude_columns)
labels = sample_df['label']

# Standardize the features
```

```
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Convert to TensorFlow tensor
features_tf = tf.convert_to_tensor(features_scaled)

# Apply t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_result = tsne.fit_transform(features_tf)

# Plot the results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=labels,␣
 ↪cmap='tab10', alpha=0.6)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Standardized Dataset with TensorFlow')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



[ ]:
```
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

sample_df = df.sample(n=15000, random_state=42)  # Reduce the sample size to␣
 ↪1000 rows

# Exclude specific columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual']
features = sample_df.drop(columns=exclude_columns)
labels = sample_df['label']

# Standardize the features
```

```
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

# Convert to TensorFlow tensor
features_tf = tf.convert_to_tensor(features_scaled)

# Apply t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
tsne_result = tsne.fit_transform(features_tf)

# Plot the results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1], c=labels,␣
 ↪cmap='tab10', alpha=0.6)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Standardized Dataset with TensorFlow')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



Generating binary mask to a dataframe

[ ]:
```
import pandas as pd
import numpy as np

np.random.seed(42)
demo_data = pd.DataFrame({
    'A': [5, 11, 18, 8],
    'B': [10, 40, 15, 30],
    'C': [9, 25, 35, 20]
})

p_m = 0.5

data_array = demo_data.values
mask = np.random.binomial(1, p_m, data_array.shape)
```

```
print("Generated Mask (1 represents masked values):\n", mask)

masked_data = np.where(mask == 1, np.nan, data_array)
masked_demo_data = pd.DataFrame(masked_data, columns=demo_data.columns)

print("\nOriginal DataFrame:\n", demo_data)
print("\nMasked DataFrame:\n", masked_demo_data)
```

```
Generated Mask (1 represents masked values):
 [[0 1 1]
 [1 0 0]
 [0 1 1]
 [1 0 1]]

Original DataFrame:
     A   B   C
0   5  10   9
1  11  40  25
2  18  15  35
3   8  30  20

Masked DataFrame:
       A     B     C
0   5.0   NaN   NaN
1   NaN  40.0  25.0
2  18.0   NaN   NaN
3   NaN  30.0   NaN
```

Shuffling the values in the column of a dataframe

[ ]:
```
import pandas as pd
import numpy as np

data = {
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50],
    'C': [100, 200, 300, 400, 500],
    'D': [120,300,231,450,200],
    'E': [12,30,31,40,20]
}
df = pd.DataFrame(data)

shuffled_df = df.apply(np.random.permutation)

print("Original DataFrame:")
print(df)
print("\nDataFrame with shuffled column values:")
print(shuffled_df)
```

```
Original DataFrame:
   A   B    C    D   E
0  1  10  100  120  12
1  2  20  200  300  30
2  3  30  300  231  31
3  4  40  400  450  40
4  5  50  500  200  20

DataFrame with shuffled column values:
   A   B    C    D   E
0  1  40  500  300  30
1  3  50  100  231  12
2  5  30  200  450  20
3  2  20  300  120  40
4  4  10  400  200  31
```

Finding corrupted dataframe

[ ]:
```python
import pandas as pd
import numpy as np

data = {
    'A': [1, 2, 3, 4, 5],
    'B': [10, 20, 30, 40, 50],
    'C': [100, 200, 300, 400, 500],
    'D': [1000, 2000, 3000, 4000, 5000],
    'E': [10000, 20000, 30000, 40000, 50000]
}
x = pd.DataFrame(data)

m = pd.DataFrame(np.random.binomial(1, 0.5, x.shape), columns=x.columns)
x_shuffled = x.apply(np.random.permutation)
x_corrupted = x * (1 - m) + x_shuffled * m

print("Original DataFrame (x):")
print(x)
print("\nBinary Mask (m):")
print(m)
print("\nShuffled DataFrame (x_shuffled):")
print(x_shuffled)
print("\nCorrupted DataFrame (x_corrupted):")
print(x_corrupted)
```

```
Original DataFrame (x):
   A   B    C     D      E
0  1  10  100  1000  10000
1  2  20  200  2000  20000
2  3  30  300  3000  30000
```

```
3  4  40  400  4000  40000
4  5  50  500  5000  50000

Binary Mask (m):
   A  B  C  D  E
0  0  0  0  0  1
1  0  1  1  1  1
2  1  0  0  1  0
3  0  1  0  1  1
4  1  0  0  1  0

Shuffled DataFrame (x_shuffled):
   A   B    C     D      E
0  1  20  200  5000  10000
1  2  50  100  1000  40000
2  4  30  300  2000  50000
3  3  40  400  4000  30000
4  5  10  500  3000  20000

Corrupted DataFrame (x_corrupted):
   A   B    C     D      E
0  1  10  100  1000  10000
1  2  50  100  1000  40000
2  4  30  300  2000  30000
3  4  40  400  4000  30000
4  5  50  500  3000  50000
```

Applying shuffling and corrupted dataframe to our original data

[ ]:
```
import pandas as pd
import numpy as np

data = pd.read_csv('/content/drive/My Drive/Levine_32dim.fcs.csv')
x = pd.DataFrame(data)

m = pd.DataFrame(np.random.binomial(1, 0.5, x.shape), columns=x.columns)
x_shuffled = x.apply(np.random.permutation)
x_corrupted = x * (1 - m) + x_shuffled * m

print("Original DataFrame (x):")
x
```

```
Original DataFrame (x):
```

[ ]:

	Event	Time	Cell_length	DNA1	DNA2	CD45RA	\
0	1	2693.00	22	4.391057	4.617262	0.162691	
1	2	3736.00	35	4.340481	4.816692	0.701349	
2	3	7015.00	32	3.838727	4.386369	0.603568	

```
3            4    7099.00           29  4.255806  4.830048  0.433747
4            5    7700.00           25  3.976909  4.506433 -0.008809
…          …        …           …        …        …        …
265622  265623  707951.44           41  6.826629  7.133022  1.474081
265623  265624  708145.44           45  6.787791  7.154026  0.116755
265624  265625  708398.44           41  6.889866  7.141219  0.684921
265625  265626  708585.44           39  6.865218  7.144353  0.288761
265626  265627  709122.44           41  6.887820  7.127359  0.360753

           CD133      CD19      CD22     CD11b  …     CD117     CD49d  \
0      -0.029585 -0.006696  0.066388 -0.009184  …  0.053050  0.853505
1      -0.038280 -0.016654  0.074409  0.808031  …  0.089660  0.197818
2      -0.032216  0.073855 -0.042977 -0.001881  …  0.046222  2.586670
3      -0.027611 -0.017661 -0.044072  0.733698  …  0.066470  1.338669
4      -0.030297  0.080423  0.495791  1.107627  … -0.006223  0.180924
…            …       …        …        …  …        …        …
265622 -0.019174 -0.055620 -0.007261  0.063395  … -0.011105  0.533736
265623 -0.056213 -0.008864 -0.035158 -0.041845  …  0.143869  1.269464
265624 -0.006264 -0.026111 -0.030837 -0.034641  …  0.087102 -0.055912
265625 -0.011310 -0.048786  0.073983 -0.031787  … -0.047971  0.101955
265626  0.128604 -0.006934  0.109846  3.864711  …  0.080195  0.037962

          HLA-DR      CD64      CD41  Viability  file_number  event_number  \
0       1.664480 -0.005376 -0.001961   0.648429     3.627711           307
1       0.491592  0.144814  0.868014   0.561384     3.627711           545
2       1.308337 -0.010961 -0.010413   0.643337     3.627711          1726
3       0.140523 -0.013449 -0.026039  -0.026523     3.627711          1766
4       0.197332  0.076167 -0.040488   0.283287     3.627711          2031
…            …       …        …        …           …            …
265622  0.123758 -0.042495 -0.027971   0.236957     3.669327        102686
265623  0.047215 -0.008000 -0.025811  -0.003500     3.669327        102690
265624  0.501536  0.053884 -0.042602   0.107206     3.669327        102701
265625  6.200001  0.296877  0.192786   0.620872     3.669327        102706
265626  3.675123 -0.000878 -0.052526   0.310466     3.669327        102720

        label  individual
0         1.0           1
1         1.0           1
2         1.0           1
3         1.0           1
4         1.0           1
…       …          …
265622    NaN           2
265623    NaN           2
265624    NaN           2
265625    NaN           2
265626    NaN           2
```

```
[265627 rows x 42 columns]
```

Genarating binary mask

[ ]:
```
print("\nBinary Mask (m):")
m
```

```
Binary Mask (m):
```

[ ]:
```
        Event  Time  Cell_length  DNA1  DNA2  CD45RA  CD133  CD19  CD22  \
0           1     1            0     1     1       1      0     0     0
1           1     0            0     1     1       1      1     0     0
2           1     1            0     1     0       1      1     1     1
3           1     1            1     1     0       0      1     1     0
4           1     0            1     1     1       1      0     1     1
…       …   …          …   …   …     …     …   …   …
265622      1     1            0     1     0       1      1     1     1
265623      0     1            0     0     1       0      0     0     0
265624      0     0            0     0     0       0      0     1     1
265625      1     1            0     1     0       1      1     1     1
265626      0     1            1     0     0       1      0     1     1

        CD11b  …  CD117  CD49d  HLA-DR  CD64  CD41  Viability  file_number  \
0           1  …      0      0       0     1     1          0            0
1           1  …      1      1       1     0     1          0            1
2           1  …      1      0       1     0     0          0            0
3           1  …      0      0       1     1     0          1            1
4           0  …      1      1       0     0     0          1            1
…       …  …    …    …      …       …   …   …        …          …
265622      0  …      1      0       1     0     1          0            0
265623      0  …      1      1       1     1     1          0            1
265624      0  …      1      1       1     1     0          1            0
265625      1  …      0      0       0     1     1          1            1
265626      1  …      1      0       1     1     0          0            0

        event_number  label  individual
0                  0      0           0
1                  0      0           0
2                  0      1           0
3                  0      1           0
4                  1      0           1
…                …    …         …
265622             0      1           0
265623             1      1           0
265624             1      0           1
```

```
265625              1       1            0
265626              0       1            1

[265627 rows x 42 columns]
```

Shuffling the data in the columns

[ ]:
```
print("\nShuffled DataFrame (x_shuffled):")
x_shuffled
```

```
Shuffled DataFrame (x_shuffled):
```

[ ]:
```
          Event       Time  Cell_length      DNA1      DNA2    CD45RA  \
0        187202  388371.00           53  3.309411  4.626784  0.815742
1         23544  127010.00           44  6.829794  4.707252  0.641168
2        203873  292640.00           30  3.353900  4.237659  1.039439
3        248040  113354.00           20  3.948969  5.056162  1.748991
4        182130  509590.44           50  4.127899  4.454341  1.499796
…           …         …          …         …         …         …
265622   238993  471666.00           27  6.688271  4.302677  0.512701
265623   213134  147989.00           17  4.066928  7.261207  0.757576
265624    19250  154481.00           32  4.160570  4.789861  0.908849
265625   252279  364254.00           22  6.548409  4.992044 -0.019653
265626   176681  462371.00           26  6.890335  4.084628  0.107580

            CD133      CD19      CD22      CD11b  …     CD117     CD49d  \
0       -0.030859  2.786928  1.651811  0.090328  … -0.000258  0.463311
1       -0.016154 -0.012219 -0.022682 -0.011631  … -0.005055  0.811016
2       -0.041769 -0.046534 -0.033154  2.369852  … -0.038846  1.769642
3       -0.035696  0.083110  0.087136  0.071977  …  0.133068  0.132315
4       -0.005710 -0.006969 -0.001594 -0.021893  …  0.079691  1.076435
…             …         …         …         …  …         …         …
265622   0.316323  1.315536 -0.009590 -0.033987  …  0.159418 -0.031561
265623  -0.039402  0.054993  0.551100  0.785193  … -0.043184  1.310663
265624   0.103880  2.878767 -0.007618  0.349996  … -0.005842  1.176842
265625  -0.044238 -0.020715  0.063187  0.662827  …  0.365161  0.326083
265626  -0.033992 -0.029501 -0.039558  3.331731  … -0.004791  0.655700

           HLA-DR      CD64      CD41  Viability  file_number  event_number  \
0       -0.021865  0.655971 -0.038410   0.520253     3.669327        364662
1       -0.040806 -0.047163  0.051815   0.414680     3.669327        274558
2        3.582167 -0.001655  0.093777   1.061496     3.627711        339539
3        4.026518 -0.001935 -0.010961   0.638660     3.627711         40848
4        2.099337 -0.013524  0.946248   0.897206     3.627711         19315
…           …         …         …          …            …              …
265622   0.094093 -0.000305  0.100088   1.382112     3.627711        264823
```

```
265623  5.732980  0.279093  0.334612   0.755504     3.627711         42419
265624  0.249220 -0.036054  0.172954   0.127050     3.627711        260163
265625 -0.015527  0.314697  0.056913  -0.015051     3.669327         13710
265626  1.854876  2.989399 -0.051715   1.168277     3.627711        215650

        label  individual
0         NaN           1
1        13.0           1
2         8.0           1
3         8.0           2
4         7.0           2
…         …           …
265622    NaN           1
265623    NaN           1
265624    NaN           1
265625    NaN           1
265626    NaN           1

[265627 rows x 42 columns]
```

Corrupted dataframe

[ ]:
```
print("\nCorrupted DataFrame (x_corrupted):")
x_corrupted
```

```
Corrupted DataFrame (x_corrupted):
```

[ ]:
```
          Event       Time  Cell_length      DNA1      DNA2    CD45RA  \
0        187202  388371.00           22  3.309411  4.626784  0.815742
1         23544    3736.00           35  6.829794  4.707252  0.641168
2        203873  292640.00           32  3.353900  4.386369  1.039439
3        248040  113354.00           20  3.948969  4.830048  0.433747
4        182130    7700.00           50  4.127899  4.454341  1.499796
…         …         …          …        …         …         …
265622   238993  471666.00           41  6.688271  7.133022  0.512701
265623   265624  147989.00           45  6.787791  7.261207  0.116755
265624   265625  708398.44           41  6.889866  7.141219  0.684921
265625   252279  364254.00           39  6.548409  7.144353 -0.019653
265626   265627  462371.00           26  6.887820  7.127359  0.107580

           CD133      CD19      CD22     CD11b  …     CD117     CD49d  \
0      -0.029585 -0.006696  0.066388  0.090328  …  0.053050  0.853505
1      -0.016154 -0.016654  0.074409 -0.011631  … -0.005055  0.811016
2      -0.041769 -0.046534 -0.033154  2.369852  … -0.038846  2.586670
3      -0.035696  0.083110 -0.044072  0.071977  …  0.066470  1.338669
4      -0.030297 -0.006969 -0.001594  1.107627  …  0.079691  1.076435
```

```
…           …         …         …         …  …         …         …
265622  0.316323  1.315536 -0.009590  0.063395  …  0.159418  0.533736
265623 -0.056213 -0.008864 -0.035158 -0.041845  … -0.043184  1.310663
265624 -0.006264  2.878767 -0.007618 -0.034641  … -0.005842  1.176842
265625 -0.044238 -0.020715  0.063187  0.662827  … -0.047971  0.101955
265626  0.128604 -0.029501 -0.039558  3.331731  … -0.004791  0.037962

          HLA-DR      CD64      CD41  Viability  file_number  event_number  \
0       1.664480  0.655971 -0.038410   0.648429     3.627711           307
1      -0.040806  0.144814  0.051815   0.561384     3.669327           545
2       3.582167 -0.010961 -0.010413   0.643337     3.627711          1726
3       4.026518 -0.001935 -0.026039   0.638660     3.627711          1766
4       0.197332  0.076167 -0.040488   0.897206     3.627711         19315
…           …         …         …         …            …             …
265622  0.094093 -0.042495  0.100088   0.236957     3.669327        102686
265623  5.732980  0.279093  0.334612  -0.003500     3.627711         42419
265624  0.249220 -0.036054 -0.042602   0.127050     3.669327        260163
265625  6.200001  0.314697  0.056913  -0.015051     3.669327         13710
265626  1.854876  2.989399 -0.052526   0.310466     3.669327        102720

        label  individual
0         NaN           1
1         1.0           1
2         8.0           1
3         8.0           1
4         1.0           2
…         …           …
265622    NaN           2
265623    NaN           2
265624    NaN           1
265625    NaN           2
265626    NaN           1

[265627 rows x 42 columns]
```

Creating new mask

[ ]:
```
mask_new = 1 * (data != x_corrupted)
mask_new
```

[ ]:
```
        Event  Time  Cell_length  DNA1  DNA2  CD45RA  CD133  CD19  CD22  \
0           1     1            0     1     1       1      0     0     0
1           1     0            0     1     1       1      1     0     0
2           1     1            0     1     0       1      1     1     1
3           1     1            1     1     0       0      1     1     0
4           1     0            1     1     1       1      0     1     1
…       …   …           …   …   …     …    …   …   …
```

```
265622      1     1            0     1     0       1      1     1     1
265623      0     1            0     0     1       0      0     0     0
265624      0     0            0     0     0       0      0     1     1
265625      1     1            0     1     0       1      1     1     1
265626      0     1            1     0     0       1      0     1     1

        CD11b  …  CD117  CD49d  HLA-DR  CD64  CD41  Viability  file_number  \
0           1  …      0      0       0     1     1          0            0
1           1  …      1      1       1     0     1          0            1
2           1  …      1      0       1     0     0          0            0
3           1  …      0      0       1     1     0          1            0
4           0  …      1      1       0     0     0          1            0
…         …  …    …    …      …     …   …   …        …          …
265622      0  …      1      0       1     0     1          0            0
265623      0  …      1      1       1     1     1          0            1
265624      0  …      1      1       1     1     0          1            0
265625      1  …      0      0       0     1     1          1            0
265626      1  …      1      0       1     1     0          0            0

        event_number  label  individual
0                  0      1           0
1                  0      0           0
2                  0      1           0
3                  0      1           0
4                  1      0           1
…                …    …         …
265622             0      1           0
265623             1      1           0
265624             1      1           1
265625             1      1           0
265626             0      1           1

[265627 rows x 42 columns]
```

Generating binary mask, shuffling and corrupted dataframe by removing certain columns

[ ]:
```
import pandas as pd
import numpy as np

# Load the dataset
data = pd.read_csv('/content/drive/My Drive/Levine_32dim.fcs.csv')
x = data.drop(columns=['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual'])

# Generate the binary mask (m) and shuffled data (x_shuffled)
m = pd.DataFrame(np.random.binomial(1, 0.5, x.shape), columns=x.columns)
x_shuffled = x.apply(np.random.permutation)
```

```
# Apply the mask to create the corrupted DataFrame
x_corrupted = x * (1 - m) + x_shuffled * m

# Display outputs
print("Original DataFrame (x):")
x
```

```
Original DataFrame (x):
```

[ ]:

	DNA1	DNA2	CD45RA	CD133	CD19	CD22	CD11b	\
0	4.391057	4.617262	0.162691	-0.029585	-0.006696	0.066388	-0.009184	
1	4.340481	4.816692	0.701349	-0.038280	-0.016654	0.074409	0.808031	
2	3.838727	4.386369	0.603568	-0.032216	0.073855	-0.042977	-0.001881	
3	4.255806	4.830048	0.433747	-0.027611	-0.017661	-0.044072	0.733698	
4	3.976909	4.506433	-0.008809	-0.030297	0.080423	0.495791	1.107627	
…	…	…	…	…	…	…	…	
265622	6.826629	7.133022	1.474081	-0.019174	-0.055620	-0.007261	0.063395	
265623	6.787791	7.154026	0.116755	-0.056213	-0.008864	-0.035158	-0.041845	
265624	6.889866	7.141219	0.684921	-0.006264	-0.026111	-0.030837	-0.034641	
265625	6.865218	7.144353	0.288761	-0.011310	-0.048786	0.073983	-0.031787	
265626	6.887820	7.127359	0.360753	0.128604	-0.006934	0.109846	3.864711	

	CD4	CD8	CD34	…	CD38	CD13	CD3	\
0	0.363602	0.520195	-0.012805	…	1.395208	0.038552	-0.032596	
1	-0.035424	-0.010551	0.089467	…	3.448410	1.457326	-0.043466	
2	-0.008781	-0.005632	-0.028717	…	1.513209	0.213583	0.320792	
3	-0.019066	0.056109	-0.027419	…	4.147996	0.514349	0.060443	
4	0.552746	0.031310	-0.038895	…	3.711521	0.585712	0.137186	
…	…	…	…	…	…	…	…	
265622	0.145304	0.358648	-0.029219	…	3.351452	0.490487	4.984959	
265623	0.970120	-0.023903	-0.005332	…	1.469735	0.408006	5.112841	
265624	1.597189	0.257884	0.107905	…	1.621310	0.104754	5.098065	
265625	0.078800	-0.000954	1.678589	…	4.313742	0.275652	-0.014854	
265626	0.792307	0.113039	0.333462	…	0.057280	3.389432	0.171945	

	CD61	CD117	CD49d	HLA-DR	CD64	CD41	Viability
0	-0.002936	0.053050	0.853505	1.664480	-0.005376	-0.001961	0.648429
1	1.258437	0.089660	0.197818	0.491592	0.144814	0.868014	0.561384
2	0.257137	0.046222	2.586670	1.308337	-0.010961	-0.010413	0.643337
3	-0.041140	0.066470	1.338669	0.140523	-0.013449	-0.026039	-0.026523
4	0.168609	-0.006223	0.180924	0.197332	0.076167	-0.040488	0.283287
…	…	…	…	…	…	…	…
265622	0.861068	-0.011105	0.533736	0.123758	-0.042495	-0.027971	0.236957
265623	0.565170	0.143869	1.269464	0.047215	-0.008000	-0.025811	-0.003500
265624	-0.008680	0.087102	-0.055912	0.501536	0.053884	-0.042602	0.107206
265625	-0.029347	-0.047971	0.101955	6.200001	0.296877	0.192786	0.620872

```
265626 -0.023831  0.080195  0.037962  3.675123 -0.000878 -0.052526   0.310466

[265627 rows x 35 columns]
```

[ ]:
```
print("\nBinary Mask (m):")
m
```

```
Binary Mask (m):
```

[ ]:
```
        DNA1  DNA2  CD45RA  CD133  CD19  CD22  CD11b  CD4  CD8  CD34  …  \
0          1     1       1      1     1     0      0    1    1     1  …
1          0     0       0      1     1     1      1    0    0     1  …
2          1     0       1      0     0     1      0    1    1     0  …
3          1     0       1      1     0     0      1    1    1     0  …
4          0     0       0      0     0     1      1    1    0     0  …
…      …   …     …       …      …     …     …      …    …    …     …  …
265622     0     1       1      0     0     0      0    0    0     0  …
265623     1     0       1      1     0     0      0    0    1     0  …
265624     0     1       0      0     0     1      0    0    0     1  …
265625     0     1       0      1     1     0      0    0    0     0  …
265626     0     1       0      0     0     1      0    1    0     1  …

        CD38  CD13  CD3  CD61  CD117  CD49d  HLA-DR  CD64  CD41  Viability
0          1     1    0     1      0      0       0     1     1          0
1          1     1    0     1      0      1       1     0     0          1
2          0     1    1     0      0      0       0     1     0          0
3          0     0    0     1      1      0       0     1     0          0
4          0     1    1     0      1      0       1     0     0          1
…      …   …     …    …     …      …      …       …     …     …          …
265622     0     0    0     0      0      0       0     0     1          0
265623     1     0    0     1      1      0       0     0     0          1
265624     1     1    1     0      0      1       1     0     0          1
265625     0     1    0     1      1      0       1     1     0          1
265626     1     1    1     0      0      1       0     0     0          1

[265627 rows x 35 columns]
```

[ ]:
```
print("\nShuffled DataFrame (x_shuffled):")
x_shuffled
```

```
Shuffled DataFrame (x_shuffled):
```

[ ]:
```
           DNA1      DNA2    CD45RA     CD133      CD19      CD22     CD11b  \
0      4.025968  4.642405  0.291089 -0.028408  0.501712  0.029038  0.155267
1      3.650288  4.391123  0.206405 -0.031280 -0.006081  2.550728  0.734031
```

```
2       4.389097  4.811013  0.546464  0.220849  2.235526  3.219952  0.376342
3       6.670080  4.695067  1.065516 -0.044531 -0.038179  1.464128  0.050722
4       6.626521  4.489879  1.542452 -0.029122  0.111850 -0.002813  3.307385
…            …         …         …         …         …         …         …
265622  6.895804  4.121008  0.123823  0.202793  2.101189  0.220571  0.812162
265623  3.218730  4.390070 -0.033357 -0.046132  0.696602  0.336232  0.202770
265624  4.025221  3.976387  0.312992  0.551332 -0.011330  0.029634  0.147735
265625  3.154437  7.044741  0.594436 -0.019158  0.481110  0.124352  0.174737
265626  6.602685  5.093980  0.208504  0.199261 -0.009185 -0.024097  0.173277

              CD4       CD8       CD34  …      CD38      CD13       CD3  \
0       -0.025076 -0.003350  0.197969  …  3.613061  1.220800  0.704932
1        0.175474 -0.030586  0.338903  …  3.707935 -0.006825  3.191412
2       -0.029451  3.412749 -0.036557  …  0.937023  0.725284  0.293934
3       -0.012208 -0.028573  0.187456  … -0.035546  0.425974  4.806763
4       -0.042306  0.272895  0.330874  …  1.580200  0.380442  4.823356
…             …         …         …  …         …         …         …
265622 -0.026390 -0.000823  0.370550  …  0.072926  0.420778  0.014674
265623  1.090885  0.519821  0.272157  …  0.216765 -0.018585  0.246699
265624  0.649378 -0.044727 -0.037653  …  4.323329  0.611901  1.088604
265625  2.477728 -0.031607 -0.024753  …  2.786135  0.581389  1.140703
265626  0.050122 -0.010724  0.545426  …  4.390322 -0.039817  4.966832

             CD61      CD117      CD49d     HLA-DR       CD64       CD41  Viability
0       -0.011878 -0.025783  0.210362 -0.009498  1.028395 -0.001744   0.975227
1       -0.024439  0.241601 -0.029360 -0.020691 -0.006511  0.849638   0.979103
2       -0.018172 -0.017015  0.160140  0.159760 -0.041559  1.050223   0.973177
3        3.309842 -0.033197  0.975825  4.363635 -0.017906  0.480471  -0.027947
4        1.809730 -0.020327  0.321591 -0.001334  0.401389  1.095780   0.160037
…             …         …         …         …         …         …         …
265622   2.225585  0.025803 -0.039817  4.399343  0.063801  0.467069  -0.041553
265623  -0.008724 -0.048249  0.533720 -0.019124  0.050101  0.216266   0.285243
265624   0.579357  0.035410  0.067765  2.322792  0.043149  0.295939  -0.015056
265625   4.732519 -0.043821  1.501023  0.211464  0.733245 -0.045400  -0.039075
265626   0.584086 -0.031974  0.456630  3.315812 -0.034485  6.346675   0.369969

[265627 rows x 35 columns]
```

[ ]:
```
print("\nCorrupted DataFrame (x_corrupted):")
x_corrupted
```

```
Corrupted DataFrame (x_corrupted):
```

[ ]:
```
              DNA1      DNA2    CD45RA     CD133      CD19      CD22     CD11b  \
0        4.025968  4.642405  0.291089 -0.028408  0.501712  0.066388 -0.009184
1        4.340481  4.816692  0.701349 -0.031280 -0.006081  2.550728  0.734031
```

```
2       4.389097  4.386369  0.546464 -0.032216  0.073855  3.219952 -0.001881
3       6.670080  4.830048  1.065516 -0.044531 -0.017661 -0.044072  0.050722
4       3.976909  4.506433 -0.008809 -0.030297  0.080423 -0.002813  3.307385
…            …         …         …         …         …         …         …
265622  6.826629  4.121008  0.123823 -0.019174 -0.055620 -0.007261  0.063395
265623  3.218730  7.154026 -0.033357 -0.046132 -0.008864 -0.035158 -0.041845
265624  6.889866  3.976387  0.684921 -0.006264 -0.026111  0.029634 -0.034641
265625  6.865218  7.044741  0.288761 -0.019158  0.481110  0.073983 -0.031787
265626  6.887820  5.093980  0.360753  0.128604 -0.006934 -0.024097  3.864711

              CD4       CD8       CD34  …       CD38      CD13       CD3  \
0       -0.025076 -0.003350  0.197969  …   3.613061  1.220800 -0.032596
1       -0.035424 -0.010551  0.338903  …   3.707935 -0.006825 -0.043466
2       -0.029451  3.412749 -0.028717  …   1.513209  0.725284  0.293934
3       -0.012208 -0.028573 -0.027419  …   4.147996  0.514349  0.060443
4       -0.042306  0.031310 -0.038895  …   3.711521  0.380442  4.823356
…             …         …         …  …          …         …         …
265622  0.145304  0.358648 -0.029219  …   3.351452  0.490487  4.984959
265623  0.970120  0.519821 -0.005332  …   0.216765  0.408006  5.112841
265624  1.597189  0.257884 -0.037653  …   4.323329  0.611901  1.088604
265625  0.078800 -0.000954  1.678589  …   4.313742  0.581389 -0.014854
265626  0.050122  0.113039  0.545426  …   4.390322 -0.039817  4.966832

             CD61      CD117     CD49d    HLA-DR      CD64      CD41  Viability
0       -0.011878  0.053050  0.853505  1.664480  1.028395 -0.001744   0.648429
1       -0.024439  0.089660 -0.029360 -0.020691  0.144814  0.868014   0.979103
2        0.257137  0.046222  2.586670  1.308337 -0.041559 -0.010413   0.643337
3        3.309842 -0.033197  1.338669  0.140523 -0.017906 -0.026039  -0.026523
4        0.168609 -0.020327  0.180924 -0.001334  0.076167 -0.040488   0.160037
…             …         …         …         …         …         …         …
265622   0.861068 -0.011105  0.533736  0.123758 -0.042495  0.467069   0.236957
265623  -0.008724 -0.048249  1.269464  0.047215 -0.008000 -0.025811   0.285243
265624  -0.008680  0.087102  0.067765  2.322792  0.053884 -0.042602  -0.015056
265625   4.732519 -0.043821  0.101955  0.211464  0.733245  0.192786  -0.039075
265626  -0.023831  0.080195  0.456630  3.675123 -0.000878 -0.052526   0.369969

[265627 rows x 35 columns]
```

[ ]:
```
mask_new = 1 * (x != x_corrupted)
mask_new
```

[ ]:
```
        DNA1  DNA2  CD45RA  CD133  CD19  CD22  CD11b  CD4  CD8  CD34  …  \
0          1     1       1      1     1     0      0    1    1     1  …
1          0     0       0      1     1     1      1    0    0     1  …
2          1     0       1      0     0     1      0    1    1     0  …
3          1     0       1      1     0     0      1    1    1     0  …
4          0     0       0      0     0     1      1    1    0     0  …
```

```
…       …   …      …    …   …   …    …   …   …   …   …
265622      0      1       1      0      0      0      0      0      0      0  …
265623      1      0       1      1      0      0      0      0      1      0  …
265624      0      1       0      0      0      1      0      0      0      1  …
265625      0      1       0      1      1      0      0      0      0      0  …
265626      0      1       0      0      0      1      0      1      0      1  …

        CD38  CD13  CD3  CD61  CD117  CD49d  HLA-DR  CD64  CD41  Viability
0          1     1    0     1      0      0       0     1     1          0
1          1     1    0     1      0      1       1     0     0          1
2          0     1    1     0      0      0       0     1     0          0
3          0     0    0     1      1      0       0     1     0          0
4          0     1    1     0      1      0       1     0     0          1
…       …   …  …   …    …     …      …    …   …        …
265622     0     0    0     0      0      0       0     0     1          0
265623     1     0    0     1      1      0       0     0     0          1
265624     1     1    1     0      0      1       1     0     0          1
265625     0     1    0     1      1      0       1     1     0          1
265626     1     1    1     0      0      1       0     0     0          1

[265627 rows x 35 columns]
```

Classification of labelled and unlabelled data

[9]:
```
import pandas as pd

df = pd.read_csv('/content/drive/My Drive/Levine_32dim.fcs.csv')

df_labeled = df[df['label'].notnull()]
df_unlabeled = df[df['label'].isnull()]

x_labeled = df_labeled.drop(columns=['label'])
y_labeled = df_labeled['label']

x_unlabeled = df_unlabeled.drop(columns=['label'])
y_unlabeled = df_unlabeled['label']

print("Labeled Features (x_labeled):\n", x_labeled.head())
print("\nLabeled Label (y_labeled):\n", y_labeled.head())
print("\nUnlabeled Features (x_unlabeled):\n", x_unlabeled.head())
print("\nUnlabeled Label (y_unlabeled):\n", y_unlabeled.head())
```

```
Labeled Features (x_labeled):
    Event    Time  Cell_length      DNA1      DNA2    CD45RA     CD133  \
0      1  2693.0           22  4.391057  4.617262  0.162691 -0.029585
1      2  3736.0           35  4.340481  4.816692  0.701349 -0.038280
2      3  7015.0           32  3.838727  4.386369  0.603568 -0.032216
3      4  7099.0           29  4.255806  4.830048  0.433747 -0.027611
```

```
4      5  7700.0           25  3.976909  4.506433 -0.008809 -0.030297

       CD19      CD22     CD11b  …      CD61     CD117     CD49d    HLA-DR  \
0 -0.006696  0.066388 -0.009184  … -0.002936  0.053050  0.853505  1.664480
1 -0.016654  0.074409  0.808031  …  1.258437  0.089660  0.197818  0.491592
2  0.073855 -0.042977 -0.001881  …  0.257137  0.046222  2.586670  1.308337
3 -0.017661 -0.044072  0.733698  … -0.041140  0.066470  1.338669  0.140523
4  0.080423  0.495791  1.107627  …  0.168609 -0.006223  0.180924  0.197332

       CD64      CD41  Viability  file_number  event_number  individual
0 -0.005376 -0.001961   0.648429     3.627711           307           1
1  0.144814  0.868014   0.561384     3.627711           545           1
2 -0.010961 -0.010413   0.643337     3.627711          1726           1
3 -0.013449 -0.026039  -0.026523     3.627711          1766           1
4  0.076167 -0.040488   0.283287     3.627711          2031           1

[5 rows x 41 columns]

Labeled Label (y_labeled):
 0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
Name: label, dtype: float64

Unlabeled Features (x_unlabeled):
          Event   Time  Cell_length      DNA1      DNA2    CD45RA     CD133  \
104184  104185   40.0           25  4.203073  4.837565  0.095543 -0.027206
104185  104186  176.0           34  4.042991  4.808275  0.035310 -0.013869
104186  104187  189.0           37  4.233125  4.922201  0.415954  0.412757
104187  104188  193.0           26  3.997143  4.685426 -0.038565  0.125894
104188  104189  204.0           20  4.115830  4.893428  0.177246  0.171916

            CD19      CD22     CD11b  …      CD61     CD117     CD49d  \
104184  0.172384 -0.001950  0.505713  …  3.029787 -0.010093  0.387121
104185 -0.043922 -0.001871  0.180261  … -0.017628  0.346248  0.089940
104186  0.431715 -0.025619  0.491190  …  0.000544  0.691393  2.996583
104187  0.191383 -0.026497  0.342190  … -0.012887  0.033096 -0.029722
104188  0.028568 -0.029751  2.480689  … -0.015719 -0.043689  0.027586

          HLA-DR      CD64      CD41  Viability  file_number  event_number  \
104184  2.859639  2.709532  1.208795   0.102978     3.627711             1
104185 -0.017702  0.045091 -0.022009   0.092770     3.627711             6
104186  5.812406  1.713608  0.479122   1.888485     3.627711             7
104187 -0.031126 -0.020739 -0.014693   0.067437     3.627711             8
104188  2.543139  3.323810 -0.002918   0.109243     3.627711             9
```

```
        individual
104184           1
104185           1
104186           1
104187           1
104188           1

[5 rows x 41 columns]

Unlabeled Label (y_unlabeled):
 104184   NaN
104185   NaN
104186   NaN
104187   NaN
104188   NaN
Name: label, dtype: float64
```

[10]:
```
# Display the overall labeled data
print("Overall Labeled Features (x_labeled):")
print("Shape:", x_labeled.shape)

print("\nOverall Labeled Target (y_labeled):")
print("Shape:", y_labeled.shape)

print("Shape:",df.shape)
```

```
Overall Labeled Features (x_labeled):
Shape: (104184, 41)

Overall Labeled Target (y_labeled):
Shape: (104184,)
Shape: (265627, 42)
```

[11]:
```
from sklearn.model_selection import train_test_split

# Split labeled data into training and testing sets (70% train, 30% test)
x_train, x_test, y_train, y_test = train_test_split(x_labeled, y_labeled,␣
 ↪test_size=0.3, random_state=42)

# Display the shapes of each set
print("Training Features Shape (x_train):", x_train.shape)
print("Training Target Shape (y_train):", y_train.shape)
print("Testing Features Shape (x_test):", x_test.shape)
print("Testing Target Shape (y_test):", y_test.shape)

# Display the first few rows of each set
print("\nTraining Features (x_train):\n", x_train.head())
```

```
print("\nTraining Target (y_train):\n", y_train.head())
print("\nTesting Features (x_test):\n", x_test.head())
print("\nTesting Target (y_test):\n", y_test.head())
```

```
Training Features Shape (x_train): (72928, 41)
Training Target Shape (y_train): (72928,)
Testing Features Shape (x_test): (31256, 41)
Testing Target Shape (y_test): (31256,)

Training Features (x_train):
        Event       Time  Cell_length      DNA1      DNA2    CD45RA     CD133  \
64113   64114  401196.00           25  3.899656  4.594272  0.976652  0.302811
82744   82745  502826.44           31  6.592998  6.901888  0.431481 -0.052898
24294   24295  488377.00           41  3.543583  4.467671  0.377192  0.219081
7820     7821  225689.00           38  4.305227  4.881685  0.199351  0.100678
43295   43296  153333.00           26  4.159271  4.861015  0.831285  0.191518

           CD19      CD22     CD11b  …      CD61     CD117     CD49d  \
64113  0.154761 -0.011676  3.180236  …  0.051464 -0.003680  1.260410
82744 -0.037690 -0.029715 -0.040846  … -0.036430  0.021689  0.034946
24294  0.245478  0.193328  0.075123  …  1.003383  0.406137  1.928676
7820  -0.025812 -0.002898  1.437247  … -0.007282  1.421540  1.443145
43295  2.002712  3.387782  0.179219  … -0.040754  0.060944  1.294561

         HLA-DR      CD64      CD41  Viability  file_number  event_number  \
64113  0.700093  2.355886  0.125409   0.840205     3.627711        318320
82744 -0.055651 -0.023248 -0.054842  -0.009329     3.669327         80934
24294 -0.046849  0.229309  0.937020   1.231347     3.627711        366690
7820   2.461705  0.528679  0.072205   0.892480     3.627711        203131
43295  3.085858 -0.014128  0.479256   2.269233     3.627711        152117

       individual
64113           1
82744           2
24294           1
7820            1
43295           1

[5 rows x 41 columns]

Training Target (y_train):
 64113    10.0
82744     7.0
24294     7.0
7820      6.0
43295     9.0
Name: label, dtype: float64
```

```
Testing Features (x_test):
        Event      Time  Cell_length      DNA1      DNA2    CD45RA     CD133  \
60544  60545  278003.0           49  3.618797  4.144135  0.198186  0.000282
50673  50674  490341.0           27  3.660988  4.497041  1.272625  0.129642
50682  50683  490912.0           23  3.854865  4.663734  1.527763  0.151383
1761    1762  170466.0           17  3.716473  4.465312  0.375236 -0.037150
98760  98761  423490.0           32  6.826030  7.007709  0.223441 -0.048813

           CD19      CD22     CD11b  …      CD61     CD117     CD49d  \
60544  0.253703 -0.018972  2.665005  …  0.307357  0.208639  2.039954
50673  3.054480  2.493220  0.189975  …  0.084448  0.033192  0.004637
50682  2.361353  2.281009  0.528589  … -0.041903 -0.026017  0.109363
1761  -0.035385  0.127904  0.415204  … -0.001024 -0.017034  0.023385
98760 -0.018816 -0.045954  4.067125  … -0.029816 -0.046020  0.140410

         HLA-DR      CD64      CD41  Viability  file_number  event_number  \
60544  2.847283  2.798986  1.090235   1.005784     3.627711        237532
50673  4.488360  0.866820 -0.002174   0.917810     3.627711        367731
50682  2.328828 -0.008223 -0.018680   1.091297     3.627711        367970
1761   0.120367  0.472159 -0.014919   0.620643     3.627711        164637
98760  0.735830  1.011186 -0.044875   0.149759     3.669327         62492

       individual
60544           1
50673           1
50682           1
1761            1
98760           2

[5 rows x 41 columns]

Testing Target (y_test):
 60544    10.0
50673     9.0
50682     9.0
1761      2.0
98760    10.0
Name: label, dtype: float64
```

Logistic regression

[12]:
```
from sklearn.linear_model import LogisticRegression

def logistic_regression(x_train, y_train, x_test):

    model = LogisticRegression()
    model.fit(x_train, y_train)
```

```
    y_test_prob = model.predict_proba(x_test)

    return y_test_prob
```

XGBoost

[13]:
```
from xgboost import XGBClassifier

def xgboost_model(x_train, y_train, x_test):
    # Initialize the XGBoost model
    model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
    model.fit(x_train, y_train)
    y_test_prob = model.predict_proba(x_test)

    return y_test_prob
```

Logistic regression and xgb

[14]:
```
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification
import numpy as np

# Generate a synthetic dataset for demonstration
x, y = make_classification(n_samples=100, n_features=10, random_state=42)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)

def logistic_regression_with_outputs(x_train, y_train, x_test):
    """
    Implements Logistic Regression and returns output probabilities.

    Args:
        x_train, y_train: Training data.
        x_test: Testing data.

    Returns:
        y_pred_prob: Predicted probabilities for x_test.
    """
    # Create and train the Logistic Regression model
    model = LogisticRegression()
    model.fit(x_train, y_train)

    # Predict probabilities on the test data
    y_pred_prob = model.predict_proba(x_test)
```

```
    # Print the output probabilities
    print("Logistic Regression Output Probabilities:")
    print(y_pred_prob)

    return y_pred_prob

def xgboost_with_outputs(x_train, y_train, x_test):
    """
    Implements XGBoost and returns output probabilities.

    Args:
        x_train, y_train: Training data.
        x_test: Testing data.

    Returns:
        y_pred_prob: Predicted probabilities for x_test.
    """
    # Create and train the XGBoost model
    model = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
    model.fit(x_train, y_train)

    # Predict probabilities on the test data
    y_pred_prob = model.predict_proba(x_test)

    # Print the output probabilities
    print("\nXGBoost Output Probabilities:")
    print(y_pred_prob)

    return y_pred_prob

# Call the Logistic Regression function and print results
y_pred_prob_logistic = logistic_regression_with_outputs(x_train, y_train,␣
 ↪x_test)
print("\nFinal Logistic Regression Probabilities:")
print(y_pred_prob_logistic)

# Call the XGBoost function and print results
y_pred_prob_xgboost = xgboost_with_outputs(x_train, y_train, x_test)
print("\nFinal XGBoost Probabilities:")
print(y_pred_prob_xgboost)
```

```
Logistic Regression Output Probabilities:
[[9.04675251e-01 9.53247494e-02]
 [7.97965075e-02 9.20203492e-01]
 [1.33736145e-02 9.86626386e-01]
 [7.86373395e-02 9.21362661e-01]
```

```
 [2.07691091e-02 9.79230891e-01]
 [9.72546031e-01 2.74539693e-02]
 [9.78171609e-01 2.18283908e-02]
 [2.54443533e-01 7.45556467e-01]
 [2.14117582e-01 7.85882418e-01]
 [1.65396820e-01 8.34603180e-01]
 [9.88737978e-01 1.12620223e-02]
 [2.80587665e-03 9.97194123e-01]
 [7.85824805e-04 9.99214175e-01]
 [9.99130522e-01 8.69477913e-04]
 [5.13601693e-03 9.94863983e-01]
 [9.67299512e-01 3.27004880e-02]
 [3.91255210e-02 9.60874479e-01]
 [9.82741573e-01 1.72584274e-02]
 [9.69083770e-01 3.09162297e-02]
 [9.97698675e-01 2.30132477e-03]]

Final Logistic Regression Probabilities:
[[9.04675251e-01 9.53247494e-02]
 [7.97965075e-02 9.20203492e-01]
 [1.33736145e-02 9.86626386e-01]
 [7.86373395e-02 9.21362661e-01]
 [2.07691091e-02 9.79230891e-01]
 [9.72546031e-01 2.74539693e-02]
 [9.78171609e-01 2.18283908e-02]
 [2.54443533e-01 7.45556467e-01]
 [2.14117582e-01 7.85882418e-01]
 [1.65396820e-01 8.34603180e-01]
 [9.88737978e-01 1.12620223e-02]
 [2.80587665e-03 9.97194123e-01]
 [7.85824805e-04 9.99214175e-01]
 [9.99130522e-01 8.69477913e-04]
 [5.13601693e-03 9.94863983e-01]
 [9.67299512e-01 3.27004880e-02]
 [3.91255210e-02 9.60874479e-01]
 [9.82741573e-01 1.72584274e-02]
 [9.69083770e-01 3.09162297e-02]
 [9.97698675e-01 2.30132477e-03]]

XGBoost Output Probabilities:
[[0.973202   0.02679799]
 [0.05494148 0.9450585 ]
 [0.03825974 0.96174026]
 [0.0221805  0.9778195 ]
 [0.00801504 0.99198496]
 [0.98203075 0.01796927]
 [0.97422457 0.02577544]
 [0.03004247 0.96995753]
```

```
 [0.02741474 0.97258526]
 [0.02611899 0.973881  ]
 [0.99399126 0.00600873]
 [0.09624082 0.9037592 ]
 [0.04554039 0.9544596 ]
 [0.9983915  0.00160852]
 [0.00383204 0.99616796]
 [0.9877788  0.0122212 ]
 [0.15245807 0.8475419 ]
 [0.92679    0.07320997]
 [0.98324716 0.01675281]
 [0.9970123  0.00298767]]

Final XGBoost Probabilities:
[[0.973202   0.02679799]
 [0.05494148 0.9450585 ]
 [0.03825974 0.96174026]
 [0.0221805  0.9778195 ]
 [0.00801504 0.99198496]
 [0.98203075 0.01796927]
 [0.97422457 0.02577544]
 [0.03004247 0.96995753]
 [0.02741474 0.97258526]
 [0.02611899 0.973881  ]
 [0.99399126 0.00600873]
 [0.09624082 0.9037592 ]
 [0.04554039 0.9544596 ]
 [0.9983915  0.00160852]
 [0.00383204 0.99616796]
 [0.9877788  0.0122212 ]
 [0.15245807 0.8475419 ]
 [0.92679    0.07320997]
 [0.98324716 0.01675281]
 [0.9970123  0.00298767]]

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[13:52:13] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.


  warnings.warn(smsg, UserWarning)
```

Logistic regression and xgb log loss

[ ]:
```python
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, log_loss # Import log_loss
from sklearn.model_selection import train_test_split
```

```
from sklearn.datasets import make_classification
import numpy as np

# Generate a synthetic dataset for demonstration
x, y = make_classification(n_samples=100, n_features=10, random_state=42)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)

def logistic_regression_with_outputs(x_train, y_train, x_test, y_test): # Added␣
 ↪y_test
    """
    Implements Logistic Regression, calculates and prints log loss.

    Args:
        x_train, y_train: Training data.
        x_test, y_test: Testing data.

    Returns:
        y_pred_prob: Predicted probabilities for x_test.
    """
    model = LogisticRegression()
    model.fit(x_train, y_train)
    y_pred_prob = model.predict_proba(x_test)

    # Calculate and print log loss
    lr_log_loss = log_loss(y_test, y_pred_prob)
    print(f"Logistic Regression Log Loss: {lr_log_loss}")

    return y_pred_prob

def xgboost_with_outputs(x_train, y_train, x_test, y_test): # Added y_test
    """
    Implements XGBoost, calculates and prints log loss.

    Args:
        x_train, y_train: Training data.
        x_test, y_test: Testing data.

    Returns:
        y_pred_prob: Predicted probabilities for x_test.
    """
    model = XGBClassifier()
    model.fit(x_train, y_train)
    y_pred_prob = model.predict_proba(x_test)

    # Calculate and print log loss
    xgb_log_loss = log_loss(y_test, y_pred_prob)
```

```
    print(f"XGBoost Log Loss: {xgb_log_loss}")

    return y_pred_prob

# Example usage:
logistic_regression_probs = logistic_regression_with_outputs(x_train, y_train,␣
 ↪x_test, y_test) # Pass y_test
xgboost_probs = xgboost_with_outputs(x_train, y_train, x_test, y_test) # Pass␣
 ↪y_test
```

```
Logistic Regression Log Loss: 0.060509459503793404
XGBoost Log Loss: 0.03575391160039887
```

[ ]:
```
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier  # Import XGBClassifier
from sklearn.metrics import log_loss
import numpy as np

# Define the logistic regression function
def Logistic(x_train, y_train, x_test, y_test):
     # Check and reshape y_train if needed
    if len(y_train.shape) > 1:
        y_train = y_train.ravel()

    # Define and fit the logistic regression model
    model = LogisticRegression(random_state=42, max_iter=1000)
    model.fit(x_train, y_train)

    # Predict probabilities on x_test
    y_test_hat = model.predict_proba(x_test)

    # Calculate log loss using the true labels (y_test) and predicted␣
 ↪probabilities (y_test_hat)
    loss = log_loss(y_test, y_test_hat)

    return y_test_hat, loss

# Define the XGBoost function
def XGBoost(x_train, y_train, x_test, y_test):
    # Check and reshape y_train if needed
    if len(y_train.shape) > 1:
        y_train = y_train.ravel()

    # Define and fit the XGBoost model
    model = XGBClassifier(random_state=42, use_label_encoder=False,␣
 ↪eval_metric='logloss')  # Initialize XGBClassifier
    model.fit(x_train, y_train)
```

```
    # Predict probabilities on x_test
    y_test_hat = model.predict_proba(x_test)

    # Calculate log loss
    loss = log_loss(y_test, y_test_hat)

    return y_test_hat, loss

# Get predicted probabilities and log loss for Logistic Regression
y_test_probabilities_logistic, loss_value_logistic = Logistic(x_train, y_train,␣
 ↪x_test, y_test)

# Get predicted probabilities and log loss for XGBoost
y_test_probabilities_xgboost, loss_value_xgboost = XGBoost(x_train, y_train,␣
 ↪x_test, y_test)

# Print results
print("Logistic Regression:")
print("Predicted probabilities:\n", y_test_probabilities_logistic)
print("\nLog Loss:", loss_value_logistic)

print("\nXGBoost:")
print("Predicted probabilities:\n", y_test_probabilities_xgboost)
print("\nLog Loss:", loss_value_xgboost)
```

```
Logistic Regression:
Predicted probabilities:
 [[9.04675251e-01 9.53247494e-02]
 [7.97965075e-02 9.20203492e-01]
 [1.33736145e-02 9.86626386e-01]
 [7.86373395e-02 9.21362661e-01]
 [2.07691091e-02 9.79230891e-01]
 [9.72546031e-01 2.74539693e-02]
 [9.78171609e-01 2.18283908e-02]
 [2.54443533e-01 7.45556467e-01]
 [2.14117582e-01 7.85882418e-01]
 [1.65396820e-01 8.34603180e-01]
 [9.88737978e-01 1.12620223e-02]
 [2.80587665e-03 9.97194123e-01]
 [7.85824805e-04 9.99214175e-01]
 [9.99130522e-01 8.69477913e-04]
 [5.13601693e-03 9.94863983e-01]
 [9.67299512e-01 3.27004880e-02]
 [3.91255210e-02 9.60874479e-01]
 [9.82741573e-01 1.72584274e-02]
 [9.69083770e-01 3.09162297e-02]
```

```
 [9.97698675e-01 2.30132477e-03]]

Log Loss: 0.060509459503793404

XGBoost:
Predicted probabilities:
 [[0.973202   0.02679799]
 [0.05494148 0.9450585 ]
 [0.03825974 0.96174026]
 [0.0221805  0.9778195 ]
 [0.00801504 0.99198496]
 [0.98203075 0.01796927]
 [0.97422457 0.02577544]
 [0.03004247 0.96995753]
 [0.02741474 0.97258526]
 [0.02611899 0.973881  ]
 [0.99399126 0.00600873]
 [0.09624082 0.9037592 ]
 [0.04554039 0.9544596 ]
 [0.9983915  0.00160852]
 [0.00383204 0.99616796]
 [0.9877788  0.0122212 ]
 [0.15245807 0.8475419 ]
 [0.92679    0.07320997]
 [0.98324716 0.01675281]
 [0.9970123  0.00298767]]

Log Loss: 0.03575391160039887

/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning:
[12:51:07] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

Encoder model

[2]:
```
from keras.layers import Input, Dense
from keras.models import Model
import numpy as np

def binary_mask(p_m, data):
    """Generates a binary mask with probability p_m."""
    return np.random.binomial(1, 1 - p_m, data.shape)

def corruption(mask, data):
    num_samples, num_features = data.shape
    shuffled_data = np.zeros([num_samples, num_features])
```

```
    for feature_idx in range(num_features):
        shuffled_indices = np.random.permutation(num_samples)
        shuffled_data[:, feature_idx] = data[shuffled_indices, feature_idx]

    data_corrupted = data * (1 - mask) + shuffled_data * mask
    mask_new = (data != data_corrupted).astype(int)

    return mask_new, data_corrupted

def self_supervised(x_unlabeled, p_m, alpha, parameters):
    epochs = parameters['epochs']
    batch_size = parameters['batch_size']
    _, dimension = x_unlabeled.shape

    # Define model architecture
    input_layer = Input(shape=(dimension,))
    h = Dense(int(dimension), activation='relu')(input_layer)

    output1 = Dense(int(dimension), activation='sigmoid',␣
 ↪name='mask_estimation')(h)
    output2 = Dense(int(dimension), activation='sigmoid',␣
 ↪name='feature_estimation')(h)

    model = Model(inputs=input_layer, outputs=[output1, output2])

    # Compile model with appropriate loss functions and weights
    model.compile(
        optimizer="rmsprop",
        loss={'mask_estimation': 'binary_crossentropy', 'feature_estimation':␣
 ↪'mean_squared_error'},
        loss_weights={'mask_estimation': 1.0, 'feature_estimation':␣
 ↪float(alpha)}  # Corrected to use float
    )

    # Generate corrupted input and mask labels
    corruption_binary_mask = binary_mask(p_m, x_unlabeled)
    x_unlabeled_corrupted,mask_label = corruption(corruption_binary_mask,␣
 ↪x_unlabeled)

    assert x_unlabeled_corrupted.shape == mask_label.shape


    # Train model
    model.fit(x_unlabeled_corrupted, {'mask_estimation': mask_label,␣
 ↪'feature_estimation': x_unlabeled},
              epochs=epochs, batch_size=batch_size)
```

```
    # Display model summary (this will print the model's parameters)
    model.summary()

    # Define encoder
    name_of_layer = model.layers[1].name
    layer_output = model.get_layer(name_of_layer).output
    encoder = Model(inputs=model.input, outputs=layer_output)

    return encoder
```

[3]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

# Exclude specified columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number',␣
 ↪'event_number', 'label', 'individual']

# Check if all columns in exclude_columns are present in the DataFrame
missing_columns = [col for col in exclude_columns if col not in df.columns]

# Print the missing columns, if any
if missing_columns:
    print(f"Warning: The following columns are not found in the DataFrame:␣
 ↪{missing_columns}")

# Remove missing columns from exclude_columns
exclude_columns = [col for col in exclude_columns if col in df.columns]

data_filtered = df.drop(columns=exclude_columns)

# Convert all columns to numeric, coercing errors to NaN
for col in data_filtered.columns:
    data_filtered[col] = pd.to_numeric(data_filtered[col], errors='coerce')

# Impute or drop NaN values strategically
# Option 1: Impute with mean/median
# for col in data_filtered.columns:
#     data_filtered[col] = data_filtered[col].fillna(data_filtered[col].mean())

# Option 2: Drop only rows where all values are NaN
# data_filtered = data_filtered.dropna(how='all')

# Option 3: (If a specific column causes most NaNs, and you can drop it):
# data_filtered = data_filtered.drop(columns=['problematic_column']) #Replace␣
 ↪problematic_column
```

```
# data_filtered = data_filtered.dropna()


# Standardize the data
scaler = StandardScaler()
x_unlabeled_scaled = scaler.fit_transform(data_filtered)  # Now␣
 ↪x_unlabeled_scaled is defined

# Define other parameters
p_m = 0.3
alpha = 2.0
parameters = {
    'batch_size': 128,
    'epochs': 50,
}

# Run the self_supervised function with the scaled data
encoder_model = self_supervised(x_unlabeled_scaled, p_m, alpha, parameters)
```

```
Epoch 1/50
2076/2076             6s 2ms/step -
loss: 2.0981
Epoch 2/50
2076/2076             5s 2ms/step -
loss: 2.0044
Epoch 3/50
2076/2076             4s 2ms/step -
loss: 2.0115
Epoch 4/50
2076/2076             6s 2ms/step -
loss: 2.0005
Epoch 5/50
2076/2076             5s 2ms/step -
loss: 1.9965
Epoch 6/50
2076/2076             4s 2ms/step -
loss: 1.9969
Epoch 7/50
2076/2076             4s 2ms/step -
loss: 1.9988
Epoch 8/50
2076/2076             6s 3ms/step -
loss: 1.9655
Epoch 9/50
2076/2076             8s 2ms/step -
loss: 1.9949
Epoch 10/50
```

```
2076/2076            5s 3ms/step -
loss: 1.9554
Epoch 11/50
2076/2076            9s 2ms/step -
loss: 1.9515
Epoch 12/50
2076/2076            5s 2ms/step -
loss: 1.9801
Epoch 13/50
2076/2076            4s 2ms/step -
loss: 1.9438
Epoch 14/50
2076/2076            4s 2ms/step -
loss: 1.8821
Epoch 15/50
2076/2076            4s 2ms/step -
loss: 1.9974
Epoch 16/50
2076/2076            5s 3ms/step -
loss: 1.9161
Epoch 17/50
2076/2076            4s 2ms/step -
loss: 1.9393
Epoch 18/50
2076/2076            4s 2ms/step -
loss: 1.8220
Epoch 19/50
2076/2076            5s 3ms/step -
loss: 1.6759
Epoch 20/50
2076/2076            4s 2ms/step -
loss: 1.6625
Epoch 21/50
2076/2076            5s 2ms/step -
loss: 1.8091
Epoch 22/50
2076/2076            7s 3ms/step -
loss: 2.0014
Epoch 23/50
2076/2076            4s 2ms/step -
loss: 1.7987
Epoch 24/50
2076/2076            5s 2ms/step -
loss: 1.6211
Epoch 25/50
2076/2076            7s 3ms/step -
loss: 1.6691
Epoch 26/50
```

```
2076/2076            4s 2ms/step -
loss: 1.6696
Epoch 27/50
2076/2076            4s 2ms/step -
loss: 1.6703
Epoch 28/50
2076/2076            5s 2ms/step -
loss: 1.6415
Epoch 29/50
2076/2076            5s 2ms/step -
loss: 1.6590
Epoch 30/50
2076/2076            4s 2ms/step -
loss: 1.7814
Epoch 31/50
2076/2076            5s 2ms/step -
loss: 1.1253
Epoch 32/50
2076/2076            5s 2ms/step -
loss: 0.8085
Epoch 33/50
2076/2076            4s 2ms/step -
loss: 1.4767
Epoch 34/50
2076/2076            5s 2ms/step -
loss: 0.9526
Epoch 35/50
2076/2076            6s 2ms/step -
loss: 0.8387
Epoch 36/50
2076/2076            4s 2ms/step -
loss: 1.6512
Epoch 37/50
2076/2076            7s 3ms/step -
loss: 0.6641
Epoch 38/50
2076/2076            5s 2ms/step -
loss: 1.1370
Epoch 39/50
2076/2076            4s 2ms/step -
loss: 0.5306
Epoch 40/50
2076/2076            7s 3ms/step -
loss: 0.3233
Epoch 41/50
2076/2076            8s 2ms/step -
loss: 1.3047
Epoch 42/50
```

```
2076/2076            7s 3ms/step -
loss: 1.6888
Epoch 43/50
2076/2076            8s 2ms/step -
loss: 1.6454
Epoch 44/50
2076/2076            7s 3ms/step -
loss: 0.8684
Epoch 45/50
2076/2076            8s 2ms/step -
loss: 1.4379
Epoch 46/50
2076/2076            5s 2ms/step -
loss: 0.2145
Epoch 47/50
2076/2076            9s 2ms/step -
loss: 0.9052
Epoch 48/50
2076/2076            8s 3ms/step -
loss: 0.9297
Epoch 49/50
2076/2076            8s 2ms/step -
loss: 1.9027
Epoch 50/50
2076/2076            7s 3ms/step -
loss: -0.1530
```

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 35)	0	-
dense (Dense)	(None, 35)	1,260	input_layer[0][0]
mask_estimation (Dense)	(None, 35)	1,260	dense[0][0]
feature_estimation (Dense)	(None, 35)	1,260	dense[0][0]

```
Total params: 7,562 (29.54 KB)

Trainable params: 3,780 (14.77 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 3,782 (14.78 KB)
```

[17]:
```
encoder_path = "/content/encoder_model.keras"
encoder_model.save(encoder_path)
```

[18]:
```
from keras.models import load_model
encoder = load_model(encoder_path)
```

[19]:
```
# check how well logistic regression and xgboost works on the encoded data

X_train_scaled_encoded = encoder.predict(X_train_scaled)
X_test_scaled_encoded = encoder.predict(X_test_scaled)

y_encoded = logistic(X_train_scaled_encoded, y_train, X_test_scaled_encoded)

# compute log loss for y_encoded and y_test
print(log_loss(y_test, y_encoded))

# do the sam xgboost
y_encoded_xgb = xgboost(X_train_scaled_encoded, y_train, X_test_scaled_encoded)

# compute log loss for y_encoded_xgb and y_test
print(log_loss(y_test,y_encoded_xgb))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-d07b42155292> in <cell line: 3>()
      1 # check how well logistic regression and xgboost works on the encoded␣
 ↪data
      2
----> 3 X_train_scaled_encoded = encoder.predict(X_train_scaled)
      4 X_test_scaled_encoded = encoder.predict(X_test_scaled)
      5

/usr/local/lib/python3.10/dist-packages/keras/src/utils/traceback_utils.py in␣
 ↪error_handler(*args, **kwargs)
    120             # To get the full stack trace, call:
```

```
    121             # `keras.config.disable_traceback_filtering()`
--> 122             raise e.with_traceback(filtered_tb) from None
    123         finally:
    124             del filtered_tb

/usr/local/lib/python3.10/dist-packages/keras/src/layers/input_spec.py in␣
 ↪assert_input_compatibility(input_spec, inputs, layer_name)
    243                 if spec_dim is not None and dim is not None:
    244                     if spec_dim != dim:
--> 245                         raise ValueError(
    246                             f'Input {input_index} of layer␣
 ↪"{layer_name}" is '
    247                             "incompatible with the layer: "

ValueError: Input 0 of layer "functional_1" is incompatible with the layer:␣
 ↪expected shape=(None, 35), found shape=(32, 41)
```

[20]:
```
from keras.layers import Input,Dense
from keras.models import Model
from keras import models
import numpy as np

def self_supervised(x_unlabeled, p_m , alpha , parameters):


  #extract batch_size and epochs
  epochs = parameters['epochs']
  batch_size = parameters['batch_size']
  _,dimension = x_unlabeled.shape

  # model creation.
  # Defining an encoder.
  # Auto encoder sturcuter ---> corrupted input ---> encoder ---> latent space␣
 ↪---> decoder.
  # working on the encoder part and extracting th
x_unlab = x_unlabeled_scaled

p_m = 0.3

alpha = 2.0

parameters = {'batch_size':128 ,
              'epochs':50
              }

encoder = self_supervised(x_unlab,p_m,alpha,parameters)
```

[21]:
```
import tensorflow as tf
from tensorflow.keras import layers, models

def model(input_dimension, hidden_dimension, label_dimension, activation=tf.nn.
 ↪relu):
    inputs = tf.keras.Input(shape=input_dimension, name='model_input')
    x = layers.Dense(hidden_dimension, activation=activation,␣
 ↪name='model_dense_layer_1')(inputs)
    x = layers.Dense(hidden_dimension, activation=activation,␣
 ↪name='model_dense_layer_2')(x)
    y_logit = layers.Dense(label_dimension, activation=None,␣
 ↪name='model_logit_output')(x)
    y = layers.Activation('softmax', name='model_output')(y_logit)
    model = models.Model(inputs=inputs, outputs=[y_logit, y], name="model")
    return model

def train(feature_batch , label_batch, unlabeled_feature_batch , model , beta ,␣
 ↪supv_loss_fn, optimizer):

  with tf.GradientTape() as tape:
    y_logit, y = model(feature_batch, training = True) # getting outputs for␣
 ↪labeled data
    y_loss = supv_loss_fn(label_batch, y_logit) # calculating supervised loss␣
 ↪function for labeled data
    unlabeled_y_logit, unlabeled_y = model(unlabeled_feature_batch, training =␣
 ↪True) # getting outputs for unlabeled data
    unlabeled_y_logit, unlabeled_y = model(unlabeled_feature_batch, training =␣
 ↪True) # getting outputs for unlabeled data
    unlabeled_y_loss = tf.reduce.mean(tf.nn.moments(unlabeled_y_logit , axes =␣
 ↪0)[1]) # loss function for unlabeled data
    # unsupervised loss function calculates the mean and variance of the␣
 ↪outputs and will penalize if the variance is high i.e, it will try to
    # reduce the variance of the output
    total_loss = y_loss + beta * unlabeled_y_loss # loss formula. Beta is a␣
 ↪hyperparameter i.e, you have to enter your own value for this
    grads = tape.gradient(total_loss, model.trainable_weights) # calculating␣
 ↪gradiennts or by how much the weights need to be changed
    optimizer.apply_gradients(zip(grads, model.trainable_weights)) # making the␣
 ↪changes to the weights

    return total_loss
```

[ ]:
```

```