CYTOAUTOCLUSTER

DAY-1 | 07-10-2024 :

- Sourced and downloaded a relevant semi-supervised cytometry mass dataset from resources such as Kaggle, Google Scholar, and Papers with Code.

DAY-2 | 08-10-2024 :

- The dataset was rejected as it contained fully labeled data, whereas semi-supervised learning requires a portion of unlabeled data.
- Presented the dataset to my mentor and received feedback to ensure the dataset includes more than 60% unlabeled data.
- The mentor advised that the final dataset should consist of over 40 columns and be completely in tabular format.

DAY-3 | 09-10-2024 :

- Acquired a foundational understanding of the components of cytometry using a sample dataset (Levine_32dim_notransform.csv).
- Learned basic Git commands for working with the project's GitHub repository.
- Understood collaborative repository management processes, including pushing, merging, and committing code changes.

DAY-4 | 10-10-2024 :

- Finalized the dataset: Levine_32dim_notransform.csv.
- Set up the Python environment, uploaded the dataset, and created a DataFrame for further analysis.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.semi_supervised import LabelPropagation
from sklearn.metrics import silhouette_score
from sklearn.manifold import TSNE

# Load the cytometry data (assumed to be in CSV format)
# Replace 'data.csv' with the actual data file path
df = pd.read_csv('/content/drive/MyDrive/data.csv')
```

- Imported the necessary Python packages for data analysis.
- Uploaded the dataset into the environment.

```
print(data.columns)
```

- Displayed the dataset’s columns using print(data.columns) to verify proper loading and understand the dataset structure.

DAY-5 | 11-10-2024 :

- Conducted Exploratory Data Analysis (EDA) using techniques such as info(), histograms, and calculating label and unlabel percentages.

```
# Display the first few rows of the dataset
print(data.head())
# Display the structure and data types of the dataset
print(data.info())
# Get summary statistics of numerical columns
print(data.describe())
```

- Displayed the first few rows of the dataset using print(data.head()) to inspect the initial entries and confirm successful data import.
- Displayed the dataset's structure and data types using print(data.info()) to understand the overall data schema and check for missing values or incorrect data types.
- Generated summary statistics for numerical columns using print(data.describe()) to analyse key metrics like mean, median, standard deviation, and range.

```
#calculate label and unlabel percentage
label_count = data['label'].count()
unlabel_count = data['label'].isna().sum()

label_percentage = (label_count / len(data)) * 100
unlabel_percentage = (unlabel_count / len(data)) * 100

print(f"Label percentage: {label_percentage:.2f}%")
print(f"Unlabel percentage: {unlabel_percentage:.2f}%")
```

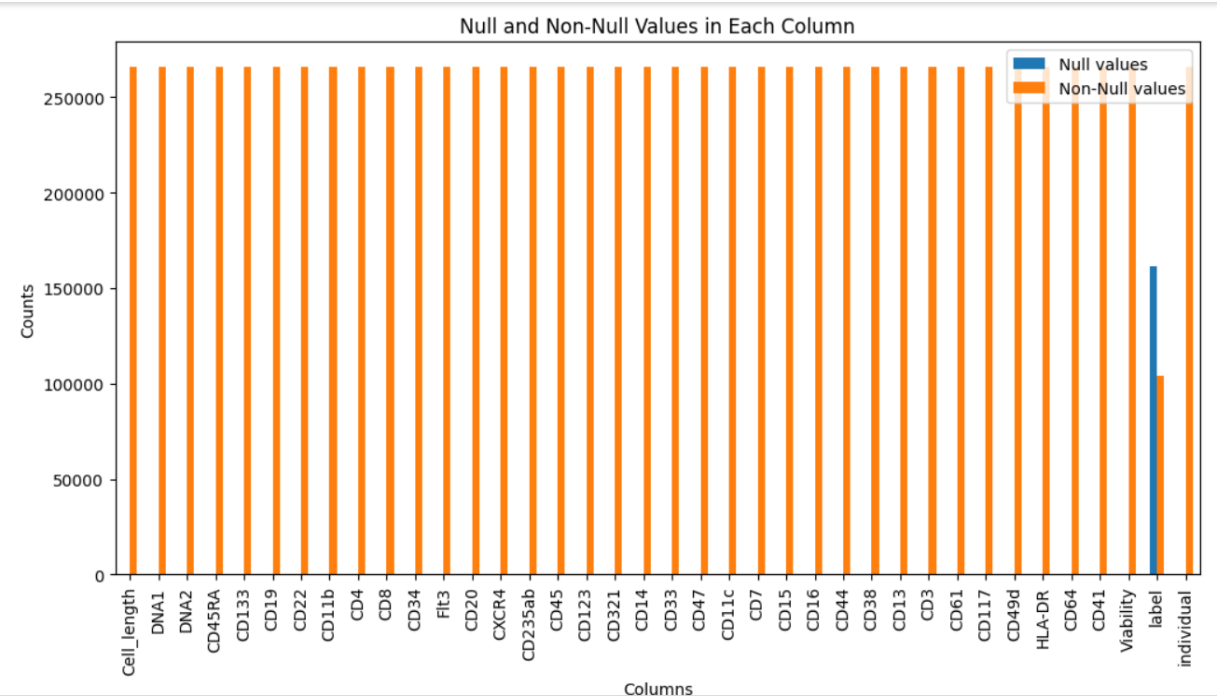
- Calculated the percentage of labeled and unlabeled data in the label column to prepare for semi-supervised learning.

DAY-6 | 14-10-2024 :

- Removed unnecessary columns (Cell_length, file_number, and event_number) from the dataset to focus on relevant features.

```
data=data.drop(columns=['Time', 'file_number', 'event_number'])
```

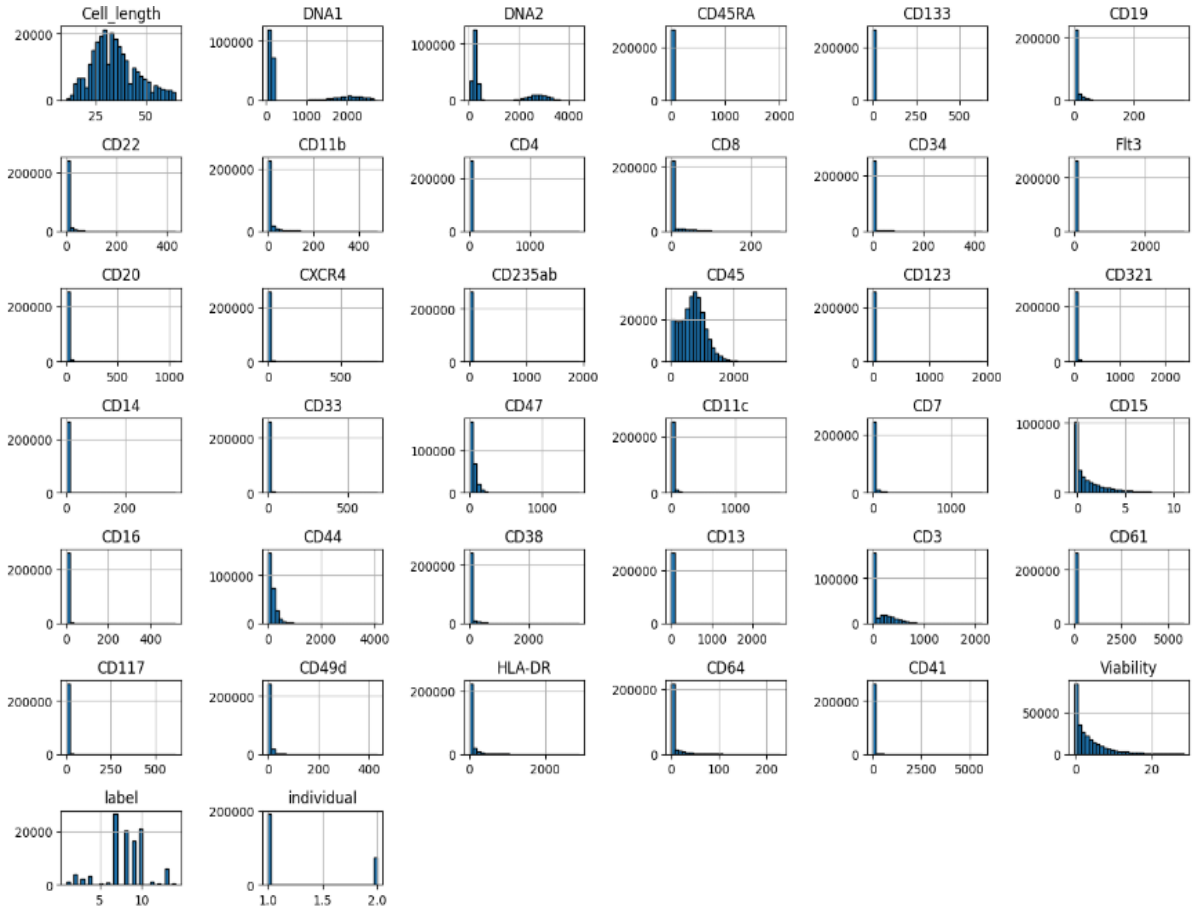
```
# Plot comparision of null values vs non-null values
null_counts=data.isnull().sum()
non_null_counts=data.notnull().sum()
plot_data=pd.DataFrame({"Null values": null_counts,
                        "Non-Null values": non_null_counts
                        })
plot_data.plot(kind="bar", figsize=(12, 6))
plt.title("Null and Non-Null Values in Each Column")
plt.xlabel("Columns")
plt.ylabel("Counts")
plt.xticks(rotation=90)
plt.legend(loc="upper right")
plt.show()
```



- Calculated the count of null and non-null values in each column using data.isnull().sum() and data.notnull().sum().
- Created a bar plot to compare the number of null and non-null values in each column, using a DataFrame to organize the counts.
- Visualized the plot to gain insights into the presence of missing data, with labeled axes and a legend for clarity.

```
# Plot histograms for all numerical columns
data.hist(figsize=(14, 10), bins=30, edgecolor='black')
plt.tight_layout()
plt.show()
```


- Calculated the count of null and non-null values in each column using data.isnull().sum() and data.notnull().sum().
- Created a bar plot to compare the number of null and non-null values in each column, using a DataFrame to organize the counts.
- Visualized the plot to gain insights into the presence of missing data, with labeled axes and a legend for clarity.

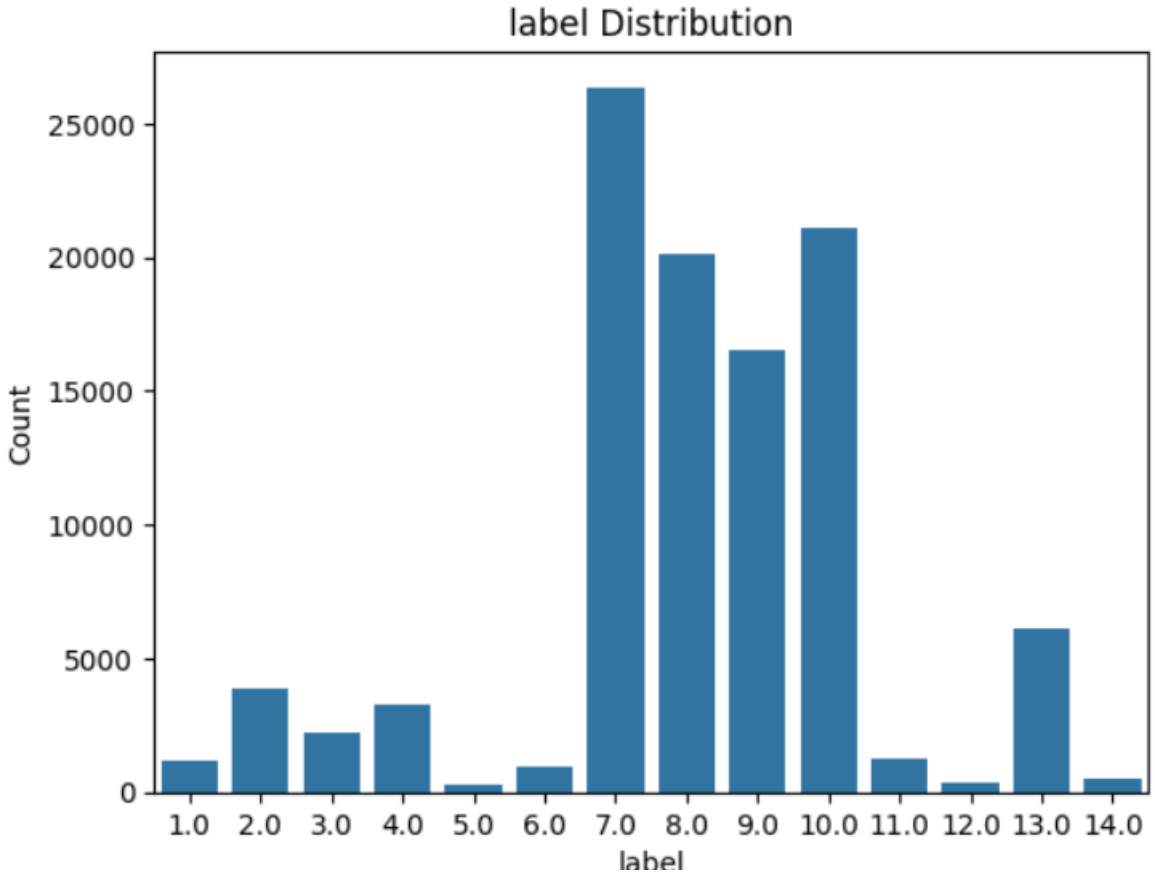


DAY-6 | 14-10-2024 :
Performed additional EDA techniques, including:

- ❖ Analyzing the range of each feature to understand their spread and distribution.
- ❖ Generating a correlation matrix to identify relationships between different features.
- ❖ Examining the class label distribution to gain insights into the balance of labeled data.
- ❖ Creating box plots to visualize feature variability and detect potential outliers.

```
# Countplot for another column in the dataset
sns.countplot(data=data, x='label')
plt.title('label Distribution')
plt.xlabel('label')
plt.ylabel('Count')
plt.show()
```

- ❖ Created a count plot for the label column in the dataset using Seaborn to visualize the distribution of class labels.
- ❖ The plot was labeled with a title and axes descriptions to clearly convey the number of occurrences for each label.




```
#range of each feature in data set
for column in data.columns:
  feature_range = data[column].max() - data[column].min()
  print(f'Range of {column}: {feature_range}')
```

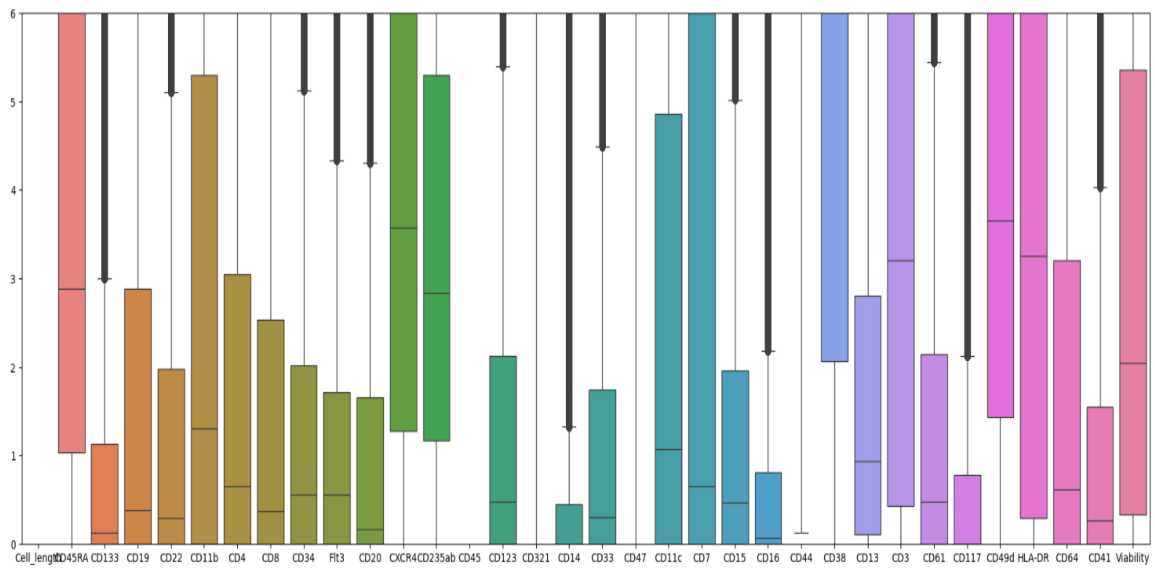
- Calculated the range of each feature in the dataset using the formula:
  Range = Maximum Value − Minimum Value

```
Range of Cell_length: 55
Range of DNA1: 2705.260757446289
Range of DNA2: 4373.519323348999
Range of CD45RA: 2013.4970097541834
Range of CD133: 629.0630275011063
Range of CD19: 367.64589342474915
Range of CD22: 435.8913907711069
Range of CD11b: 481.8620219230657
Range of CD4: 1804.8100344240064
Range of CD8: 273.4073664546013
Range of CD34: 430.49149709939906
Range of Flt3: 3083.149202555413
Range of CD20: 1062.0647517740726
Range of CXCR4: 744.964627265 93
Range of CD235ab: 1925.8784293532397
Range of CD45: 3459.6277294158986
Range of CD123: 1914.2225522100975
Range of CD321: 2401.3572410941124
Range of CD14: 373.58409309387 22
Range of CD33: 684.8300481736662
Range of CD47: 1508.6325097084045
Range of CD11c: 1698.3260714411736
Range of CD7: 1388.134843885896
Range of CD15: 11.344912439584778
Range of CD16: 520.675962328911
Range of CD44: 4108.539601758122
Range of CD38: 3675.532707840 2042
Range of CD13: 2690.77463901042 94
Range of CD3: 2131.94249191880 73
Range of CD61: 5795.503212273121
Range of CD117: 613.3092513978481
Range of CD49d: 432.8393713831899
Range of HLA-DR: 2889.668938398361
Range of CD64: 229.35811600089048
Range of CD41: 5623.056521087885
Range of Viability: 28.554032683372 51
Range of label: 13.0
Range of individual: 1
```

- This analysis helps in understanding the spread of values for each feature.

```
#box plot
fig, ax = plt.subplots(figsize=(24, 8))
sns.boxplot(data=data[['Cell_length', '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']])
ax.set_ylim(0, 6)
```

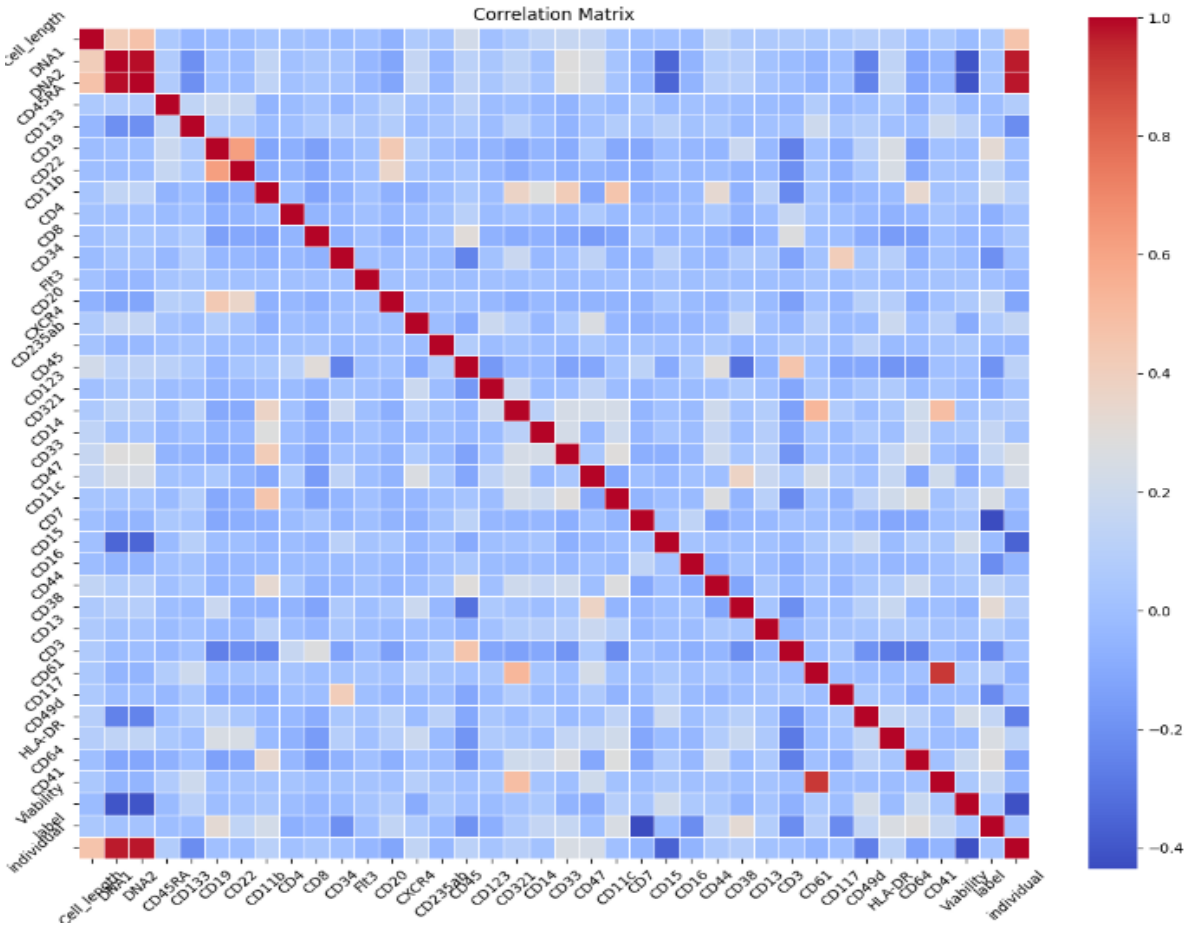
```
(0.0, 6.0)
```



- Created box plots for all protein features, which represented their values within a range of 0 to 6. This visualization helps to identify the distribution, central tendency, and potential outliers for each feature.

```
# Calculate the correlation matrix
correlation_matrix = data.corr()
# Create a heatmap for the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=False, fmt='.2f', cmap='coolwarm', square=True,
linewidths=0.5)
plt.title('Correlation Matrix')
plt.xticks(rotation=45)
plt.yticks(rotation=45)
plt.tight_layout()
plt.show()
```

- Calculated the correlation values for each feature in the dataset and visualized the results as a correlation matrix using a heatmap. The heatmap utilizes color gradients to indicate the strength of correlation between features, helping to identify highly correlated, low, or neutral relationships.



DAY-7 | 15-10-2024 :

Performed EDA Techniques more like Kurtosis, Skewness, and Pair Plot

```
from scipy.stats import skew
import pandas as pd

# Calculate skewness of each feature in the dataset
skewness = df.apply(skew)

# Function to categorize skewness
def categorize_skewness(value):
    if value > 0.5:
        return "Right-skewed"
    elif value < -0.5:
        return "Left-skewed"
    else:
        return "Approximately symmetrical"

# Apply the categorization function to the skewness values
skewness_category = skewness.apply(categorize_skewness)

# Display skewness and its categorization
skewness_df = pd.DataFrame({'Skewness': skewness, 'Category': skewness_category})
```

```
print(skewness_df)
```

- Here’s a refined version of your description for the skewness analysis:
- Calculated the skewness for each feature in the dataset using the scipy.stats library to measure asymmetry in data distribution.
- Defined a function to categorize skewness as:
  - "Right-skewed" for values greater than 0.5,
  - "Left-skewed" for values less than -0.5, and
  - "Approximately symmetrical" for values between -0.5 and 0.5.
- Applied the categorization function to the calculated skewness values and displayed the results in a DataFrame showing each feature's skewness and corresponding category.

	Skewness	Category
Cell_length	0.527832	Right-skewed
DNA1	1.155424	Right-skewed
DNA2	1.108669	Right-skewed
CD45RA	65.251655	Right-skewed
CD133	126.096395	Right-skewed
CD19	4.007221	Right-skewed
CD22	6.131244	Right-skewed
CD11b	5.264678	Right-skewed
CD4	114.022325	Right-skewed
CD8	3.313920	Right-skewed
CD34	8.397363	Right-skewed
Flt3	26.230625	Right-skewed
CD20	10.655454	Right-skewed
CXCR4	14.332247	Right-skewed
CD235ab	35.288190	Right-skewed
CD45	0.514492	Right-skewed
CD123	13.956222	Right-skewed
CD321	15.415273	Right-skewed
CD14	74.327532	Right-skewed
CD33	11.659128	Right-skewed
CD47	4.327074	Right-skewed
CD11c	7.679567	Right-skewed
CD7	7.405451	Right-skewed
CD15	1.860022	Right-skewed
CD16	14.520519	Right-skewed
CD44	3.436531	Right-skewed
CD38	7.733425	Right-skewed
CD13	99.104480	Right-skewed
CD3	1.479010	Right-skewed
CD61	17.909078	Right-skewed
CD117	54.242959	Right-skewed
CD49d	6.622882	Right-skewed
HLA-DR	4.612054	Right-skewed
CD64	3.752616	Right-skewed
CD41	22.140511	Right-skewed
Viability	2.013592	Right-skewed
label	NaN	Approximately symmetrical
individual	0.982030	Right-skewed

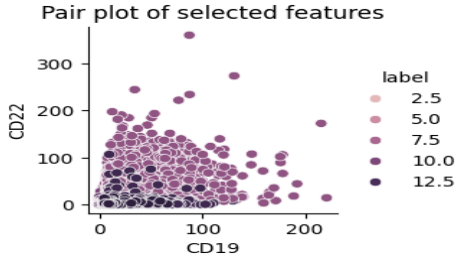
```
from scipy.stats import kurtosis
import pandas as pd

# Calculate kurtosis of each feature in the dataset
kurtosis_values = df.apply(kurtosis)

# Function to categorize kurtosis
def categorize_kurtosis(value):
    if value > 3:
        return "Leptokurtic (Heavy Tails)"
    elif value < 3:
        return "Platykurtic (Light Tails)"
    else:
        return "Mesokurtic (Normal Tails)"
```

```
# Apply the categorization function to the kurtosis values
kurtosis_category = kurtosis_values.apply(categorize_kurtosis)

# Display kurtosis and its categorization
kurtosis_df = pd.DataFrame({'Kurtosis': kurtosis_values, 'Category': kurtosis_category})
print(kurtosis_df)
```

	Kurtosis	Category
Cell_length	-0.165967	Platykurtic (Light Tails)
DNA1	-0.465002	Platykurtic (Light Tails)
DNA2	-0.572940	Platykurtic (Light Tails)
CD45RA	8911.445042	Leptokurtic (Heavy Tails)
CD133	39759.277510	Leptokurtic (Heavy Tails)
CD19	30.044641	Leptokurtic (Heavy Tails)
CD22	73.673855	Leptokurtic (Heavy Tails)
CD11b	38.254044	Leptokurtic (Heavy Tails)
CD4	17440.057242	Leptokurtic (Heavy Tails)
CD8	13.299477	Leptokurtic (Heavy Tails)
CD34	92.275706	Leptokurtic (Heavy Tails)
Flt3	859.869549	Leptokurtic (Heavy Tails)
CD20	203.215616	Leptokurtic (Heavy Tails)
CXCR4	574.158590	Leptokurtic (Heavy Tails)
CD235ab	1782.466381	Leptokurtic (Heavy Tails)
CD45	0.652132	Platykurtic (Light Tails)
CD123	395.071278	Leptokurtic (Heavy Tails)
CD321	530.082177	Leptokurtic (Heavy Tails)
CD14	16352.281414	Leptokurtic (Heavy Tails)
CD33	465.243060	Leptokurtic (Heavy Tails)
CD47	51.322868	Leptokurtic (Heavy Tails)
CD11c	124.718580	Leptokurtic (Heavy Tails)
CD7	73.440845	Leptokurtic (Heavy Tails)
CD15	3.609792	Leptokurtic (Heavy Tails)
CD16	268.364229	Leptokurtic (Heavy Tails)
CD44	28.102720	Leptokurtic (Heavy Tails)
CD38	106.335486	Leptokurtic (Heavy Tails)
CD13	12439.953303	Leptokurtic (Heavy Tails)
CD3	1.910874	Platykurtic (Light Tails)
CD61	391.035613	Leptokurtic (Heavy Tails)
CD117	7827.374159	Leptokurtic (Heavy Tails)
CD49d	170.700138	Leptokurtic (Heavy Tails)
HLA-DR	30.940301	Leptokurtic (Heavy Tails)
CD64	18.793802	Leptokurtic (Heavy Tails)
CD41	677.890565	Leptokurtic (Heavy Tails)
Viability	4.500375	Leptokurtic (Heavy Tails)
label	NaN	Mesokurtic (Normal Tails)
individual	-1.035618	Platykurtic (Light Tails)

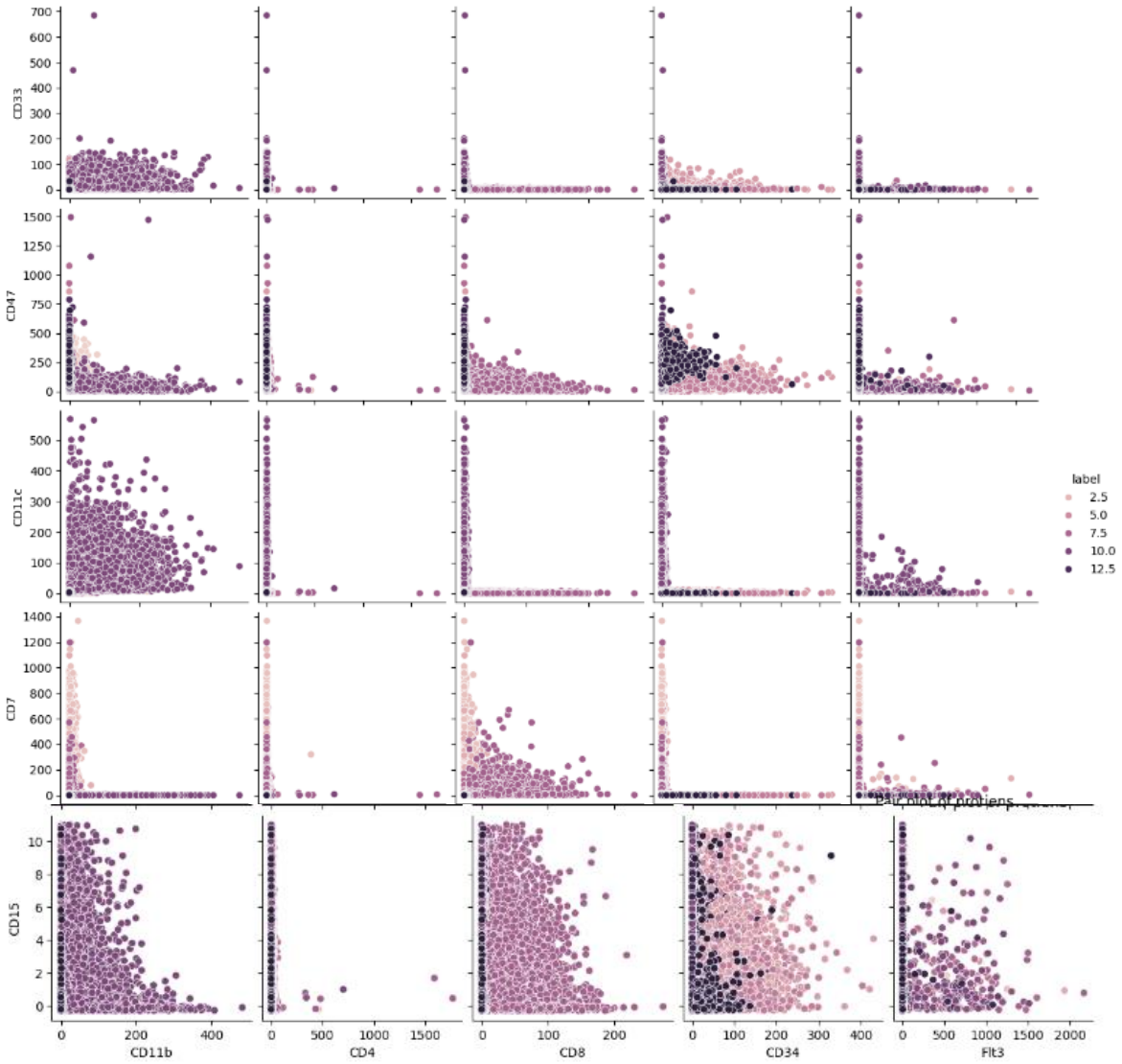
- Here's a refined version of your description for the kurtosis analysis:
- Calculated the kurtosis for each feature in the dataset using the scipy.stats library to measure the "tailedness" of data distribution.
- Defined a function to categorize kurtosis as:
  - **Leptokurtic (Heavy Tails)** for values greater than 3,
  - **Platykurtic (Light Tails)** for values less than 3, and
  - **Mesokurtic (Normal Tails)** for values equal to 3.
- Applied the categorization function to the calculated kurtosis values and displayed the results in a DataFrame showing each feature's kurtosis and corresponding category.

```
#pairplot
sns.pairplot(df, hue='label',x_vars=['CD19'],y_vars=['CD22'])
plt.title('Pair plot of selected features')
plt.show()
```



- Generated a pair plot to visualize the relationship between two selected protein features (CD19 and CD22), with different colors representing the label values.
- In the plot, darker shades indicate higher label values, while lighter shades indicate lower label values, highlighting the distribution of data points based on their labels.

```
#Pair plot for selected features colored by label
sns.pairplot(df, hue='label', x_vars=['CD11b', 'CD4', 'CD8', 'CD34', 'Flt3'], y_vars=['CD33', 'CD47', 'CD11c', 'CD7', 'CD15'])
plt.title('Pair plot of protiens')
plt.show()
```



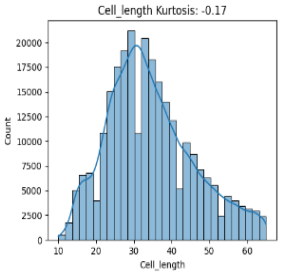
- Here's a refined version of your description for the pair plot involving 10 selected proteins:
- Generated a pair plot to visualize the relationships among 10 selected protein features, with colors representing the label values.
- In the plot, darker shades correspond to higher label values, while lighter shades indicate lower label values, allowing for a clear comparison of data point distributions based on their labels.

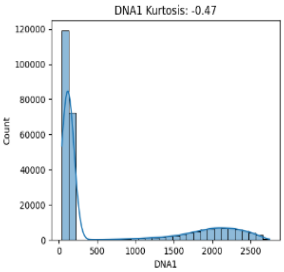
DAY-8 | 16-10-2024 :

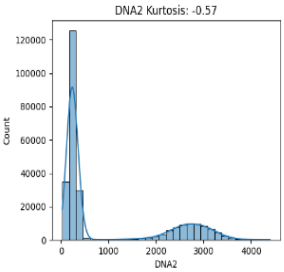
Performed individual plots for each feature to visualize kurtosis and skewness, allowing for a detailed examination of the distribution and "tailedness" of each feature in the dataset.

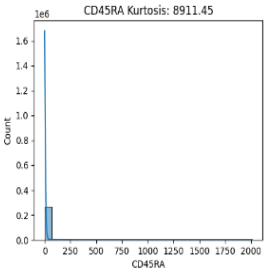
```
from scipy.stats import kurtosis
# Define the columns to exclude
excluded_columns = ['Event', 'event_time', 'file_number', 'event_number']

# Select relevant columns
relevant_columns = [col for col in df.columns if col not in excluded_columns]

# Set the number of columns and rows for the subplot grid
num_cols = 4
num_rows = (len(relevant_columns) + num_cols - 1) // num_cols

# Create subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, num_rows * 4))
axes = axes.flatten()

# Plot histograms with KDE for each relevant column
for i, col in enumerate(relevant_columns):
    sns.histplot(df[col].dropna(), kde=True, ax=axes[i], bins=30)
    axes[i].set_title(f'{col} Kurtosis: {kurtosis(df[col].dropna()):.2f}')

# Adjust layout and remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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

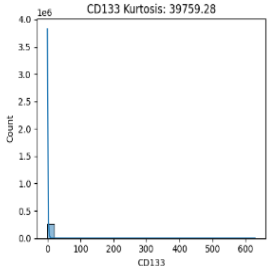
- Created histograms with Kernel Density Estimates (KDE) for each relevant feature in the dataset, excluding specific columns (Event, event_time, file_number, and event_number).
- Arranged the plots in a grid layout with a specified number of columns and rows to accommodate all relevant features.
- Each histogram was titled with the feature name and its calculated kurtosis value, providing insights into the distribution and tailedness of the data.
- Adjusted the layout to remove any unused subplots for a cleaner presentation.

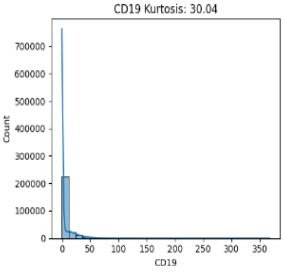

Cell_length Kurtosis: -0.17
Count
Cell_length

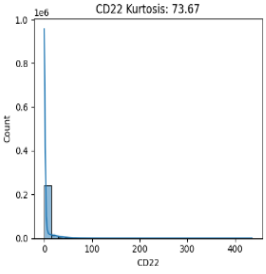

DNA1 Kurtosis: -0.47
Count
DNA1

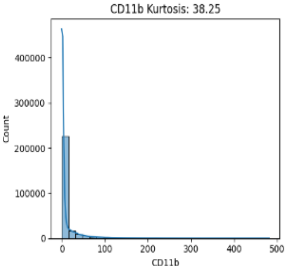

DNA2 Kurtosis: -0.57
Count
DNA2

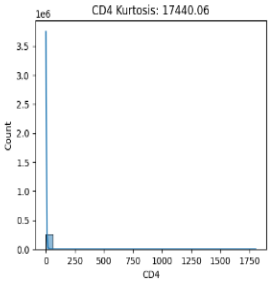

CD45RA Kurtosis: 8911.45
1e6
Count
CD45RA

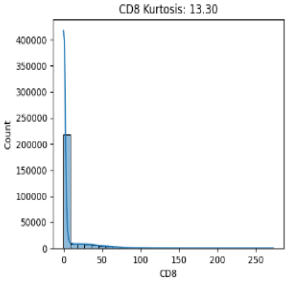

CD133 Kurtosis: 39759.28
1e6
Count
CD133

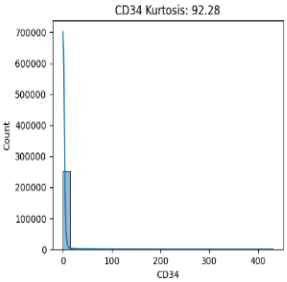

CD19 Kurtosis: 30.04
Count
CD19

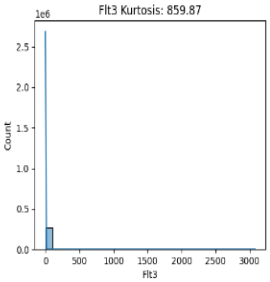

CD22 Kurtosis: 73.67
1e6
Count
CD22

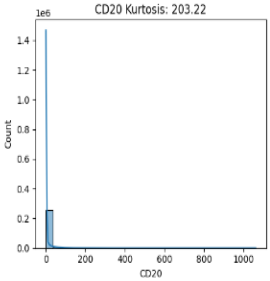

CD11b Kurtosis: 38.25
Count
CD11b

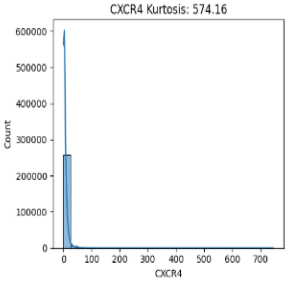

CD4 Kurtosis: 17440.06
1e6
Count
CD4

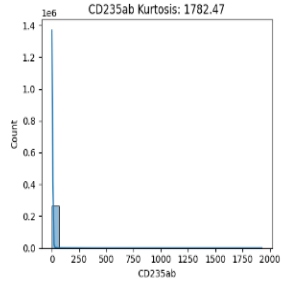

CD8 Kurtosis: 13.30
Count
CD8

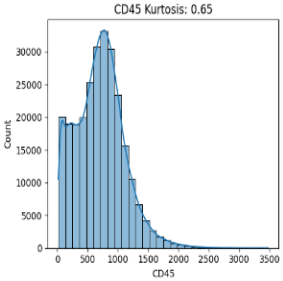

CD34 Kurtosis: 92.28
Count
CD34

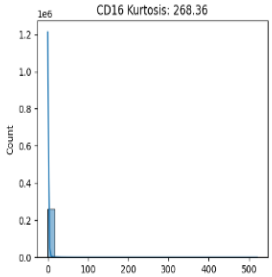

Flt3 Kurtosis: 859.87
1e6
Count
Flt3

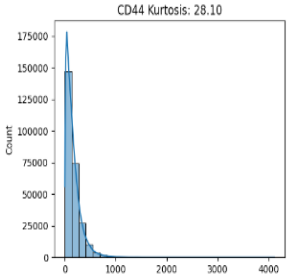

CD20 Kurtosis: 203.22
1e6
Count
CD20

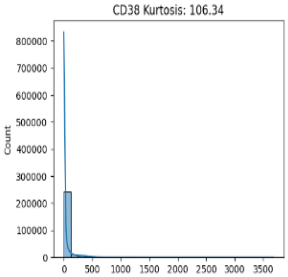

CXCR4 Kurtosis: 574.16
Count
CXCR4

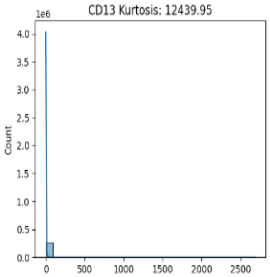

CD235ab Kurtosis: 1782.47
1e6
Count
CD235ab

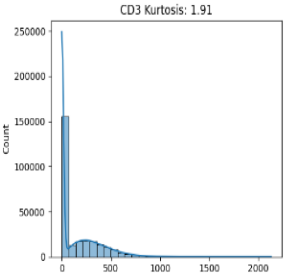

CD45 Kurtosis: 0.65
Count
CD45

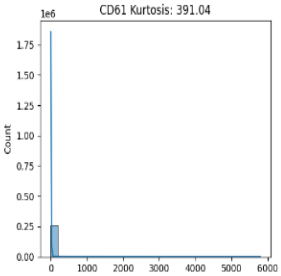

CD16 Kurtosis: 268.36
1e6
Count
CD16

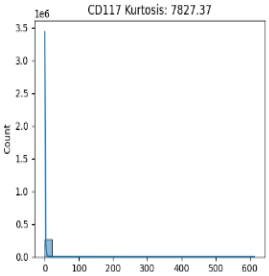

CD44 Kurtosis: 28.10
Count
CD44

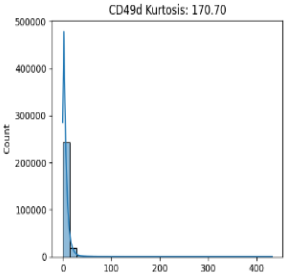

CD38 Kurtosis: 106.34
Count
CD38

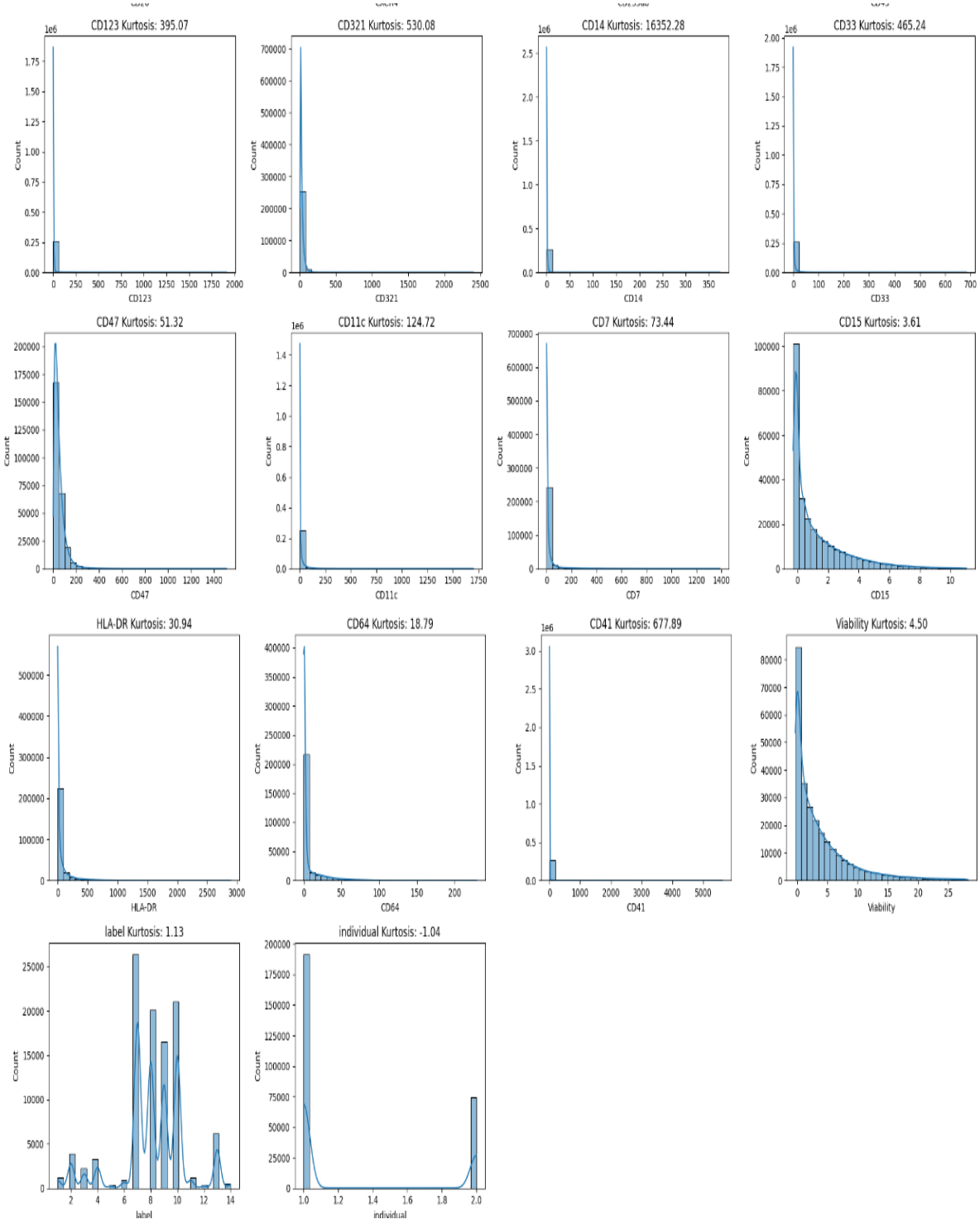

CD13 Kurtosis: 12439.95
1e6
Count
CD13


CD3 Kurtosis: 1.91
Count
CD3


CD61 Kurtosis: 391.04
1e6
Count
CD61

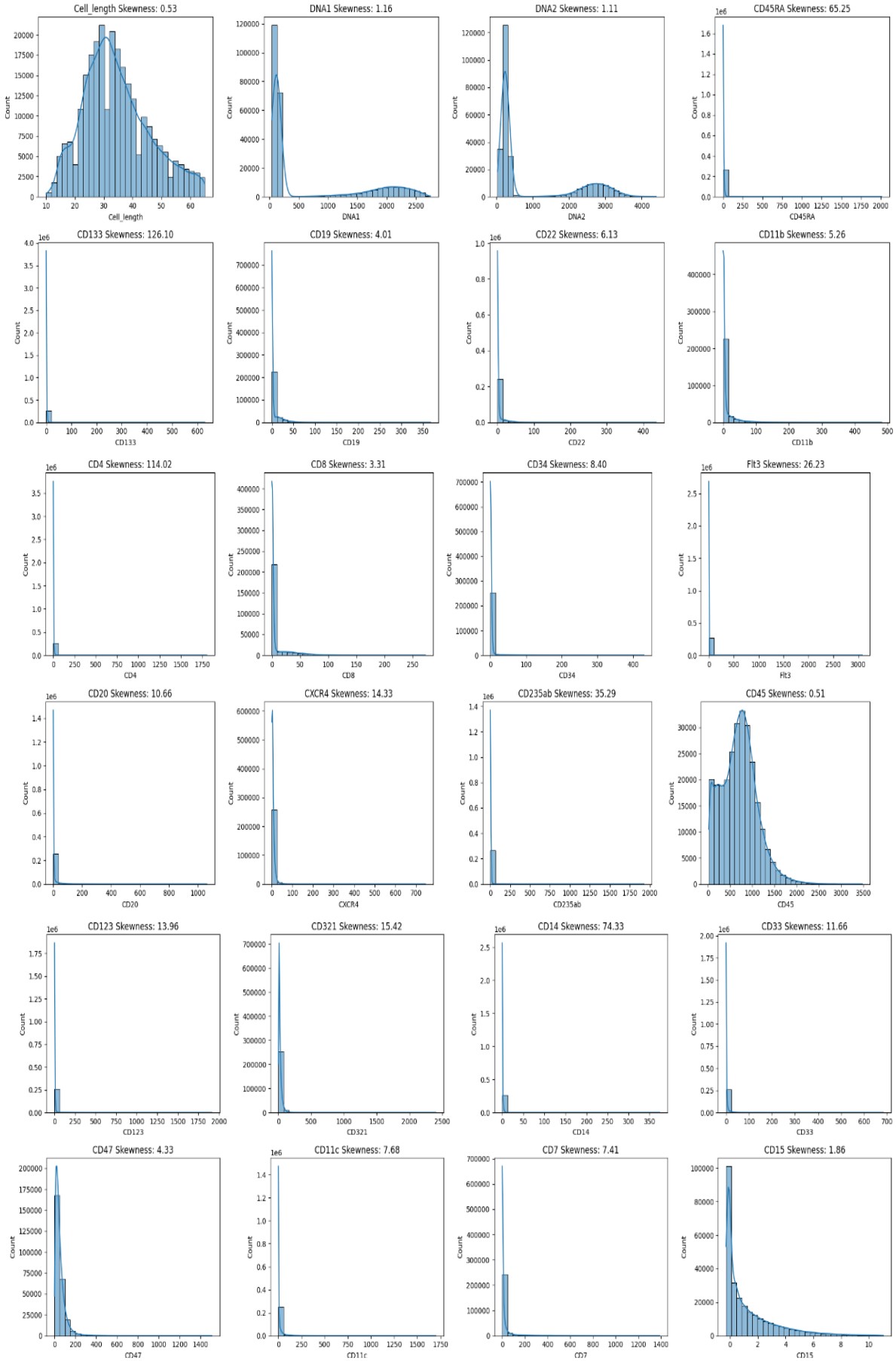

CD117 Kurtosis: 7827.37
1e6
Count
CD117

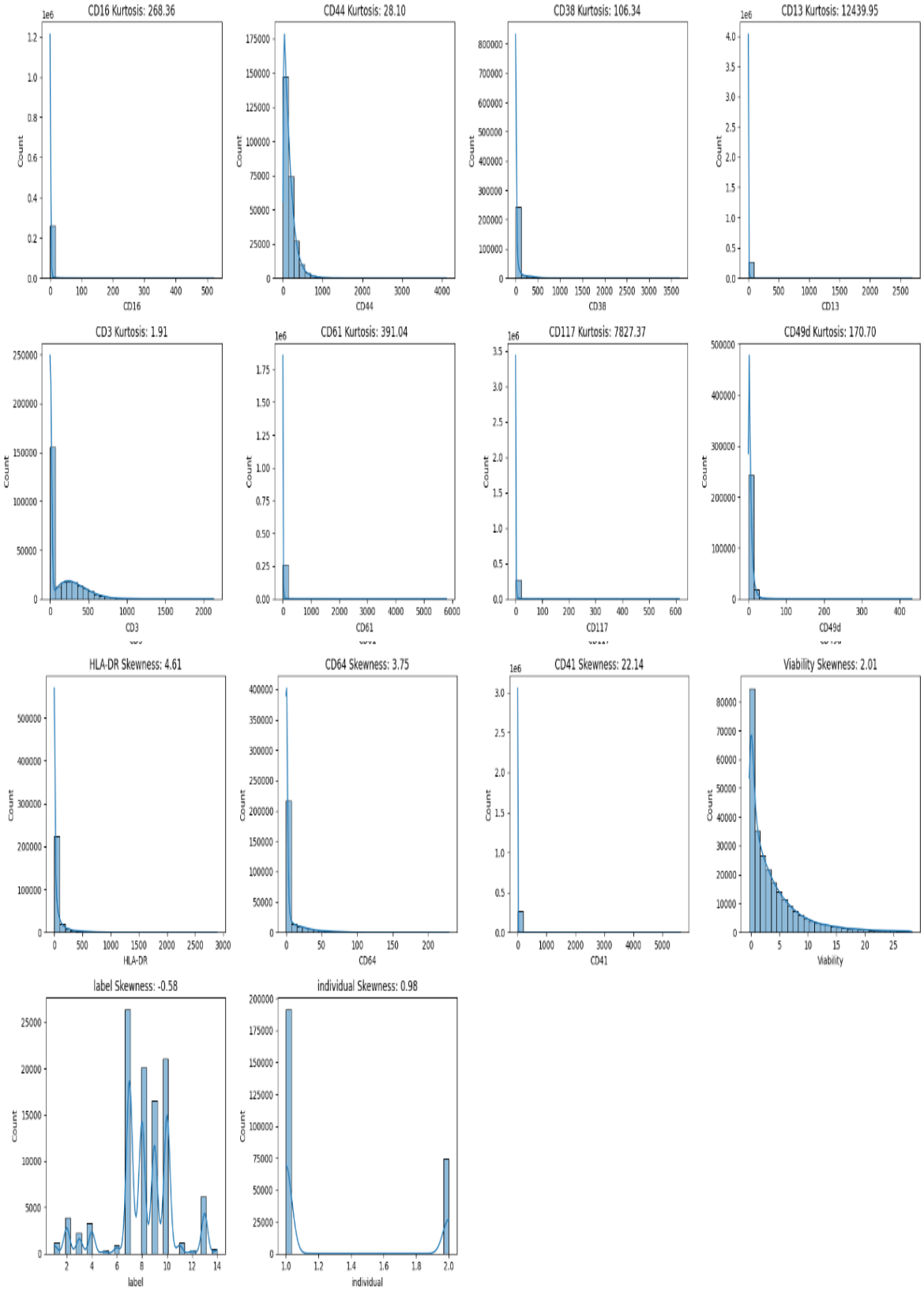

CD49d Kurtosis: 170.70
Count
CD49d


CD123 Kurtosis: 395.07
CD321 Kurtosis: 530.08
CD14 Kurtosis: 16352.28
CD33 Kurtosis: 465.24
CD47 Kurtosis: 51.32
CD11c Kurtosis: 124.72
CD7 Kurtosis: 73.44
CD15 Kurtosis: 3.61
HLA-DR Kurtosis: 30.94
CD64 Kurtosis: 18.79
CD41 Kurtosis: 677.89
Viability Kurtosis: 4.50
label Kurtosis: 1.13
individual Kurtosis: -1.04
Count
CD123
CD321
CD14
CD33
CD47
CD11c
CD7
CD15
HLA-DR
CD64
CD41
Viability
label
individual

```
from scipy.stats import skew

# Define the columns to exclude
excluded_columns = ['Event', 'event_time', 'file_number', 'event_number']

# Select relevant columns
relevant_columns = [col for col in df.columns if col not in excluded_columns]

# Set the number of columns and rows for the subplot grid
num_cols = 4
num_rows = (len(relevant_columns) + num_cols - 1) // num_cols

# Create subplots
fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(20, num_rows * 4))
axes = axes.flatten()

# Plot histograms with KDE for each relevant column
for i, col in enumerate(relevant_columns):
    sns.histplot(df[col].dropna(), kde=True, ax=axes[i], bins=30)
    # Calculate skewness
    column_skewness = skew(df[col].dropna())
    axes[i].set_title(f'{col} Skewness: {column_skewness:.2f}')

# Adjust layout and remove any unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

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

- Generated histograms with Kernel Density Estimates (KDE) for each relevant feature in the dataset, excluding certain columns.
- Plots were organized in a grid layout, with each title indicating the skewness of the respective feature, highlighting the degree of asymmetry in the data distribution.
- Unused subplots were removed for a cleaner visual presentation.


Cell_length Skewness: 0.53
DNA1 Skewness: 1.16
DNA2 Skewness: 1.11
CD45RA Skewness: 65.25
CD133 Skewness: 126.10
CD19 Skewness: 4.01
CD22 Skewness: 6.13
CD11b Skewness: 5.26
CD4 Skewness: 114.02
CD8 Skewness: 3.31
CD34 Skewness: 8.40
Flt3 Skewness: 26.23
CD20 Skewness: 10.66
CXCR4 Skewness: 14.33
CD235ab Skewness: 35.29
CD45 Skewness: 0.51
CD123 Skewness: 13.96
CD321 Skewness: 15.42
CD14 Skewness: 74.33
CD33 Skewness: 11.66
CD47 Skewness: 4.33
CD11c Skewness: 7.68
CD7 Skewness: 7.41
CD15 Skewness: 1.86
Count
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 Kurtosis: 268.36
CD44 Kurtosis: 28.10
CD38 Kurtosis: 106.34
CD13 Kurtosis: 12439.95
CD3 Kurtosis: 1.91
CD61 Kurtosis: 391.04
CD117 Kurtosis: 7827.37
CD49d Kurtosis: 170.70
HLA-DR Skewness: 4.61
CD64 Skewness: 3.75
CD41 Skewness: 22.14
Viability Skewness: 2.01
label Skewness: -0.58
individual Skewness: 0.98
Count
CD16
CD44
CD38
CD13
CD3
CD61
CD117
CD49d
HLA-DR
CD64
CD41
Viability
label
individual

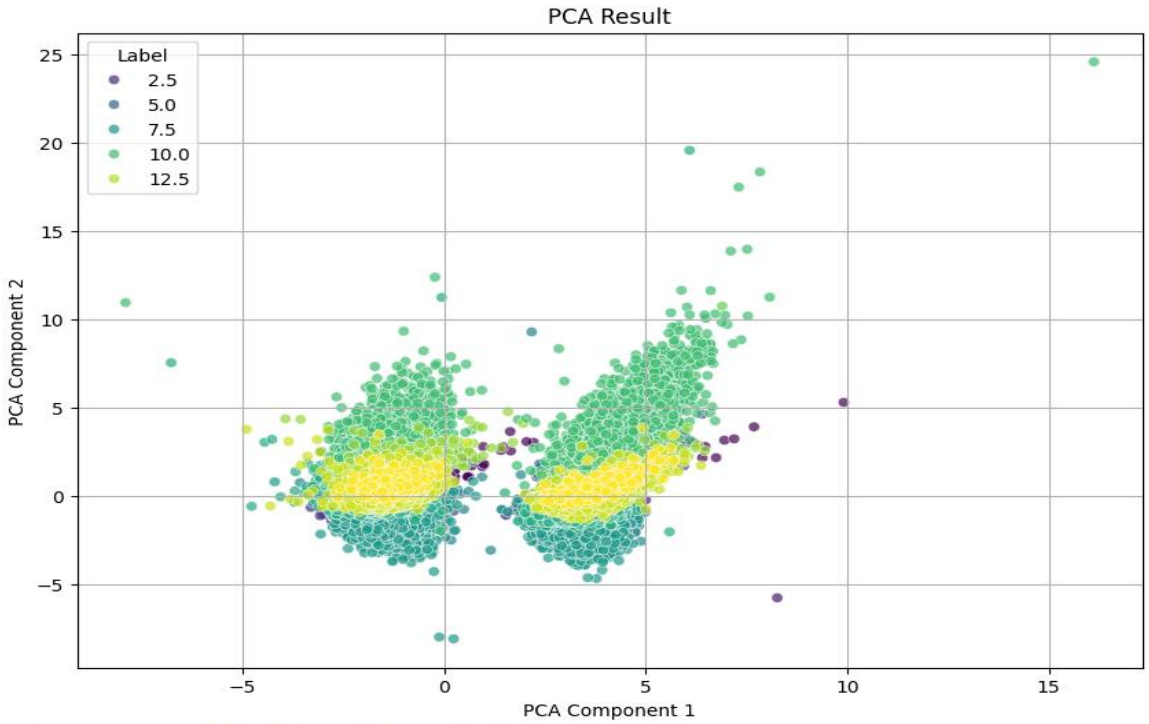
DAY-9 | 17-10-2024 :

Implementation of PCA and T-SNE-

PCA (Principal Component Analysis): PCA helps reduce the number of features in a dataset while preserving as much variance as possible. This is crucial in high-dimensional datasets like cytometry data, where having too many features can lead to overfitting and make it difficult to visualize or interpret the data.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the cytometry data (assumed to be in CSV format)
df = pd.read_csv('/content/drive/MyDrive/data.csv')

# Preprocessing: Standardizing the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop(['label'], axis=1).dropna())  # Exclude the label column

# Applying PCA
pca = PCA(n_components=2)  # Reduce to 2 components for visualization
pca_result = pca.fit_transform(scaled_data)

# Create a DataFrame with the PCA results
pca_df = pd.DataFrame(data=pca_result, columns=['PCA1', 'PCA2'])
pca_df['label'] = df['label'].dropna().reset_index(drop=True)

# Plot PCA results
plt.figure(figsize=(10, 7))
sns.scatterplot(data=pca_df, x='PCA1', y='PCA2', hue='label', palette='viridis', alpha=0.7)
plt.title('PCA Result')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Label')
plt.grid()
plt.show()

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_
print(f'Explained Variance Ratio: {explained_variance}')
```

- Applied Principal Component Analysis (PCA) to reduce the dataset to two principal components for visualization.
- Created a DataFrame containing the PCA results and the corresponding label values.
- Visualized the PCA results with a scatter plot, where each point represents a data sample, colored by its label, to reveal patterns in the data distribution.
- Displayed the explained variance ratio of the PCA components, indicating the proportion of variance captured by each component.



Explained Variance Ratio: [0.13483133 0.07788081]

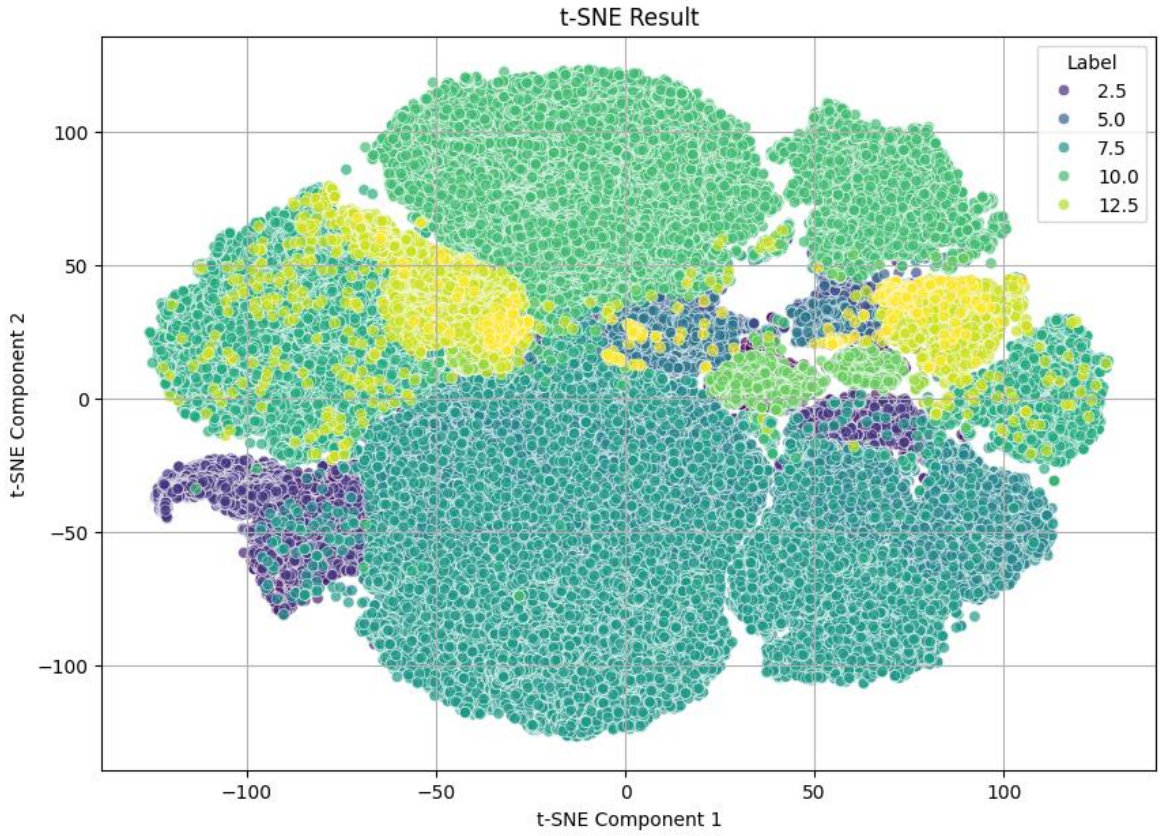
DAY-10 | 18-10-2024 :

- **t-SNE (t-Distributed Stochastic Neighbour Embedding)**: t-SNE is another powerful dimensionality reduction technique, particularly useful for visualizing high-dimensional data in two or three dimensions. It focuses on preserving local structures, making it excellent for revealing clusters and patterns in the data.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE

# Load the cytometry data (assumed to be in CSV format)
df = pd.read_csv('/content/drive/MyDrive/data.csv')

# Preprocessing: Standardizing the data
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df.drop(['label'], axis=1).dropna())  # Exclude the label column

# Applying t-SNE
tsne = TSNE(n_components=2, random_state=42)  # Reduce to 2 components for visualization
tsne_result = tsne.fit_transform(scaled_data)

# Create a DataFrame with the t-SNE results
tsne_df = pd.DataFrame(data=tsne_result, columns=['TSNE1', 'TSNE2'])
tsne_df['label'] = df['label'].dropna().reset_index(drop=True)
```

```
# Plot t-SNE results
plt.figure(figsize=(10, 7))
sns.scatterplot(data=tsne_df, x='TSNE1', y='TSNE2', hue='label', palette='viridis', alpha=0.7)
plt.title('t-SNE Result')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend(title='Label')
plt.grid()
plt.show()
```

- Loaded the cytometry dataset from a CSV file and preprocessed the data by standardizing it using StandardScaler, while excluding the label column to normalize the features.
- Applied t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the dataset to two components for visualization, setting a random seed for reproducibility.
- Created a DataFrame to hold the t-SNE results along with the corresponding label values.
- Visualized the t-SNE results with a scatter plot, where each point represents a sample colored by its label, revealing the clustering and distribution of data in the reduced-dimensional space.

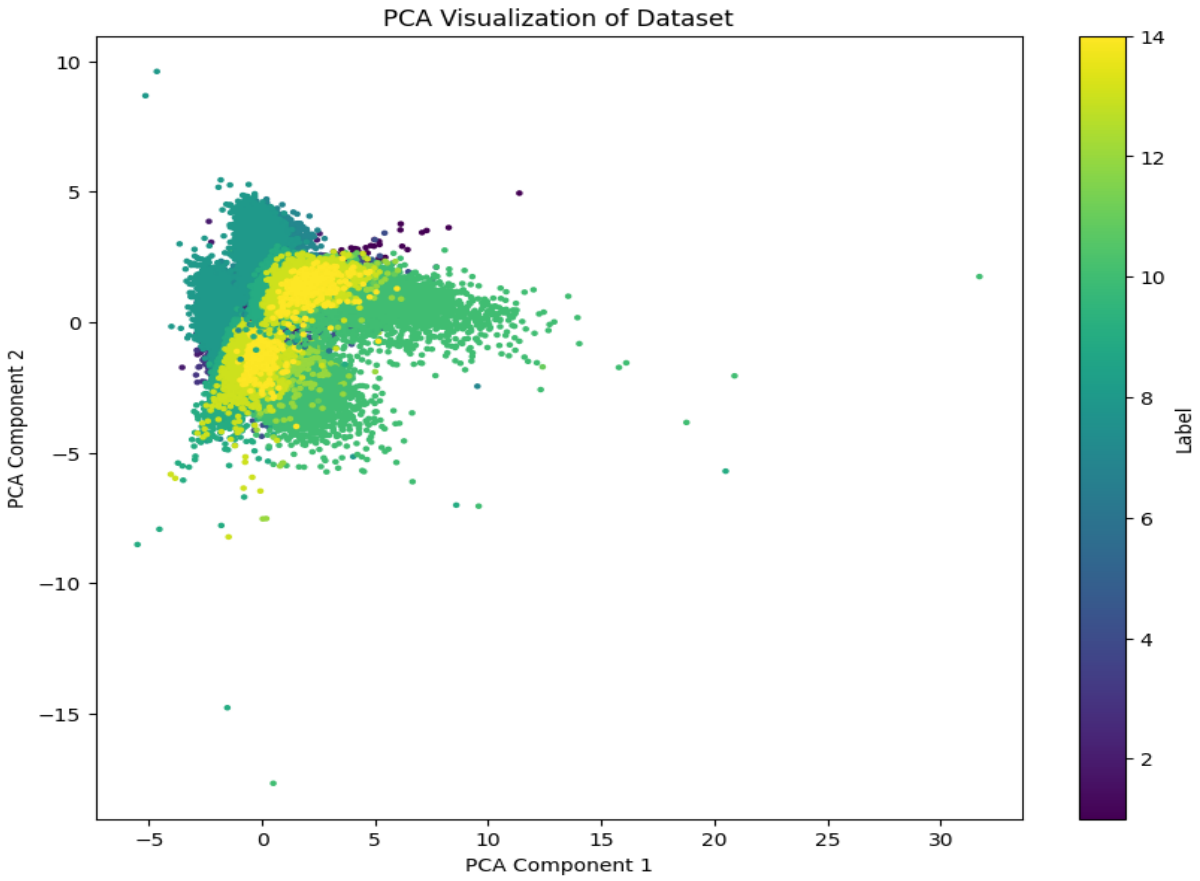


DAY-11 | 21-10-2024 :

As per my mentor's guidance, the following columns were excluded from the analysis to ensure only relevant features are considered:

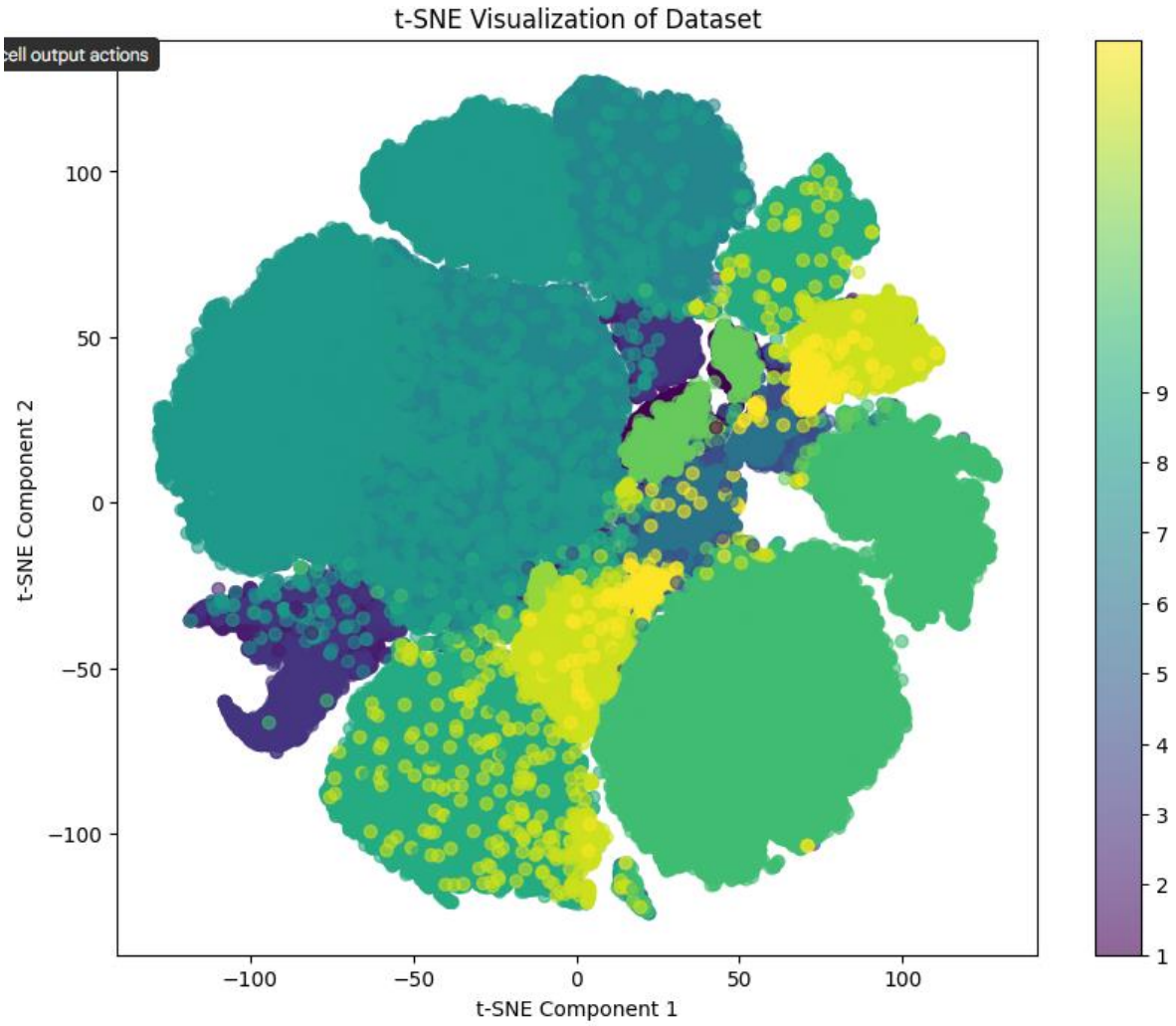
```
# Define the columns to exclude
excluded_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_number', 'label', 'individual']
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

excluded_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_number', 'label', 'individual']

# Filter out only the columns that exist in the DataFrame
existing_excluded_columns = [col for col in excluded_columns if col in df.columns]

# Exclude the specified columns
df_filtered = df.drop(existing_excluded_columns, axis=1)

# Standardize the data (Z-score normalization)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_filtered)

# Perform PCA
pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
pca_results = pca.fit_transform(scaled_data)

# Add the PCA results to the original data for visualization
df['PCA Component 1'] = pca_results[:, 0]
df['PCA Component 2'] = pca_results[:, 1]

# Plot the PCA results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(df['PCA Component 1'], df['PCA Component 2'], c=df['label'], cmap='viridis', s=5)
plt.colorbar(scatter, label='Label')
plt.title('PCA Visualization of Dataset')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



Same goes to t-sne method

```
from sklearn.manifold import TSNE

#apply t-sne
tsne = TSNE(n_components=2, random_state=42)
tsne_results = tsne.fit_transform(scaled_data)
#adding t-sne results to the original dataframe
df['TSNE1'] = tsne_results[:, 0]
df['TSNE2'] = tsne_results[:, 1]
#plotting the t-sne results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(df['TSNE1'], df['TSNE2'], c=df['label'], cmap='viridis', alpha=0.6)
plt.colorbar(scatter, ticks=range(10))
plt.title('t-SNE Visualization of Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



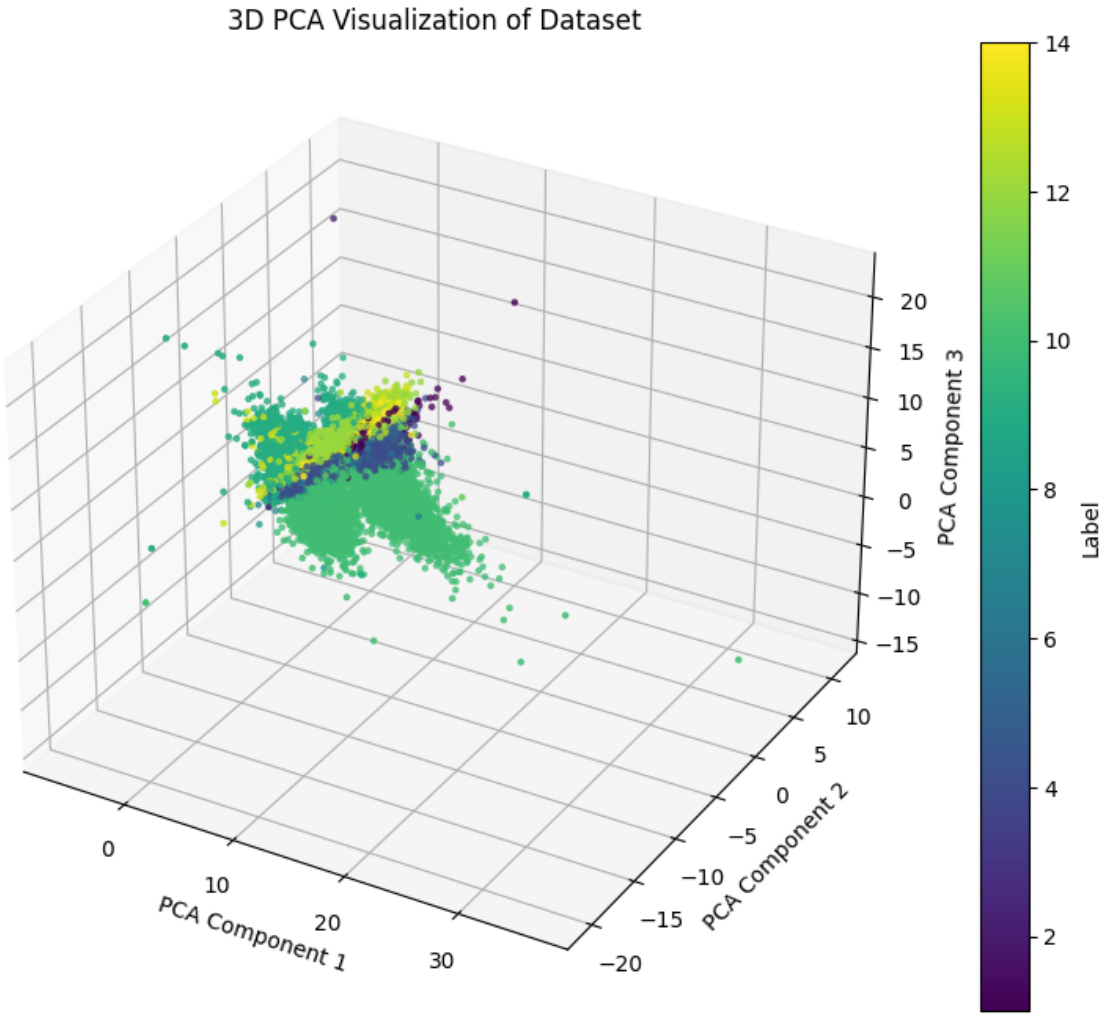
DAY-12 | 22-10-2024 :

- Loaded the cytometry dataset and preprocessed the data by excluding specific columns and standardizing the features using StandardScaler to ensure each feature contributes equally to the PCA analysis.
- Performed Principal Component Analysis (PCA) to reduce the dataset to four components, capturing the most variance while allowing for dimensionality reduction.
- Added the PCA results as new columns to the original DataFrame for further analysis and visualization.
- Calculated the standard deviations, proportion of variance, and cumulative proportion for each principal component to assess their contribution to the overall variance in the dataset.
- Visualized the PCA results in a 3D scatter plot, where each point represents a sample colored by its label, highlighting the clustering and distribution of data in the reduced-dimensional space.

```
excluded_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_number', 'label', 'individual']

# Filter out only the columns that exist in the DataFrame
existing_excluded_columns = [col for col in excluded_columns if col in df.columns]

# Exclude the specified columns
```

```
df_filtered = df.drop(existing_excluded_columns, axis=1)

# Handle missing data (optional)
df_filtered = df_filtered.dropna()  # Or use imputation

# Standardize the data (Z-score normalization)
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df_filtered)

# Perform PCA
pca = PCA(n_components=4)  # Reduce to 4 dimensions
pca_results = pca.fit_transform(scaled_data)

# Add the PCA results to the original data for visualization
df['PCA Component 1'] = pca_results[:, 0]
df['PCA Component 2'] = pca_results[:, 1]
df['PCA Component 3'] = pca_results[:, 2]
df['PCA Component 4'] = pca_results[:, 3]  # Add the 4th component

# Calculate standard deviation of each principal component
std_devs = np.sqrt(pca.explained_variance_)

# Calculate proportion of variance and cumulative proportion
proportion_variance = pca.explained_variance_ratio_
cumulative_proportion = np.cumsum(proportion_variance)

# Print results for all 4 components
print("Standard Deviations of PCA Components:", std_devs)
print("Proportion of Variance:", proportion_variance)
print("Cumulative Proportion of Variance:", cumulative_proportion)

# 3D Plot the PCA results
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df['PCA Component 1'], df['PCA Component 2'], df['PCA Component 3'],
                     c=df['label'], cmap='viridis', s=5)
plt.colorbar(scatter, label='Label')
ax.set_title('3D PCA Visualization of Dataset')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
plt.show()
```

```
Standard Deviations of PCA Components: [2.05032064 1.978851   1.62466973 1.54900803]
Proportion of Variance: [0.11361619 0.10583342 0.07133897 0.0648491 ]
Cumulative Proportion of Variance: [0.11361619 0.21944961 0.29078857 0.35563768]
```



Learnt a brief on Auto Encoders like on what domain encoders are used and some model names that are implemented using auto encoders**.**

DAY-13 | 23-10-2024 :

Studied the research paper on "Deep Semi-Supervised Learning" (https://arxiv.org/pdf/2006.05278).

DAY-14 | 25-10-2024 :

Gained a basic understanding of semi-supervised learning concepts from the research paper.