CytoAutoCluster

Importing Dataset

In [ ]:
```
import pandas as pd
```

In [ ]:
```
data= pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')
```

In [ ]:
```
data.head()
```

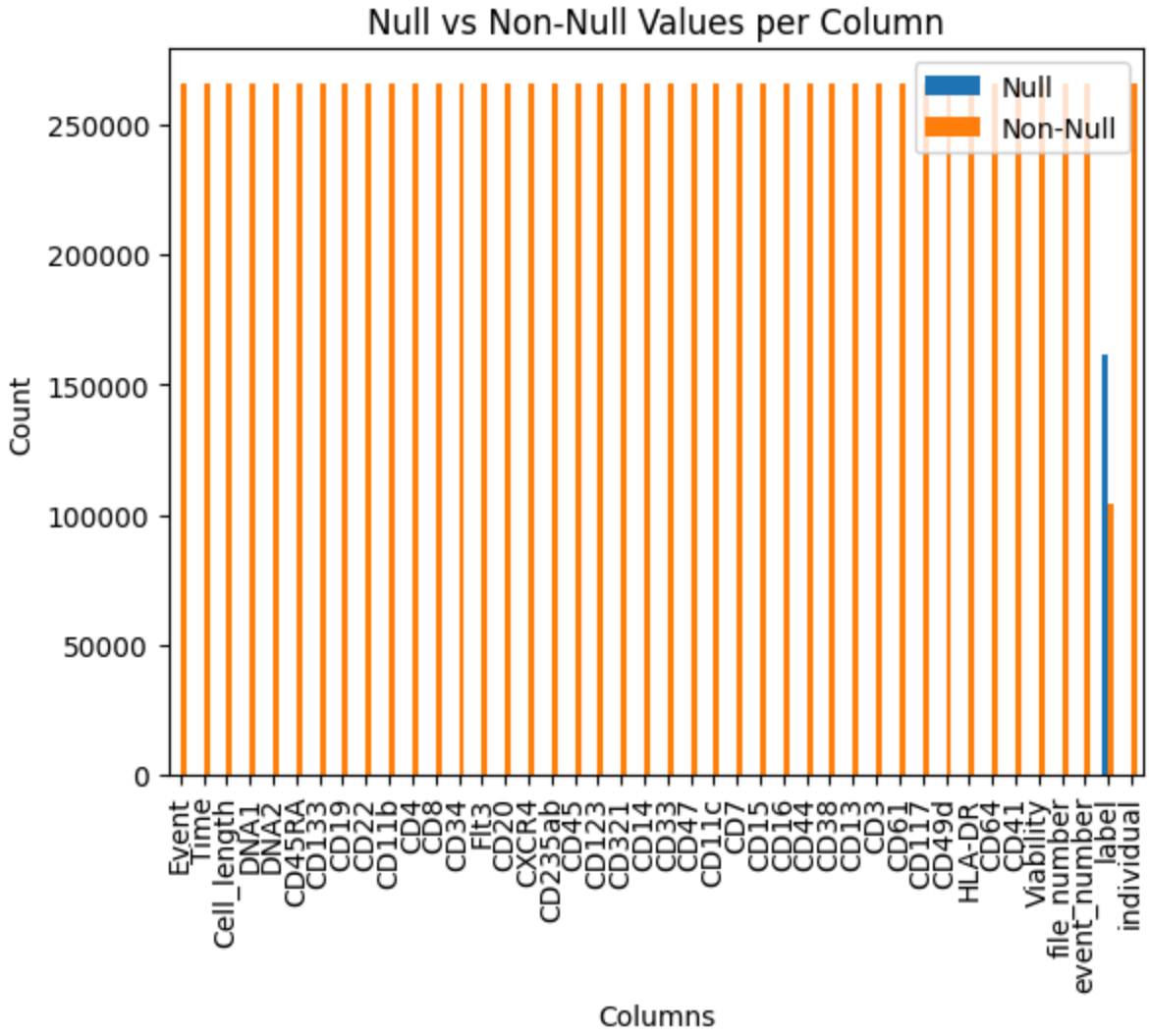
Out[ ]:

	Event	Time	Cell_length	DNA1	DNA2	CD45RA	CD133	CD1
0	1	2693.0	22	4.391057	4.617262	0.162691	-0.029585	-0.00669
1	2	3736.0	35	4.340481	4.816692	0.701349	-0.038280	-0.01665
2	3	7015.0	32	3.838727	4.386369	0.603568	-0.032216	0.07385
3	4	7099.0	29	4.255806	4.830048	0.433747	-0.027611	-0.01766
4	5	7700.0	25	3.976909	4.506433	-0.008809	-0.030297	0.08042

5 rows × 42 columns

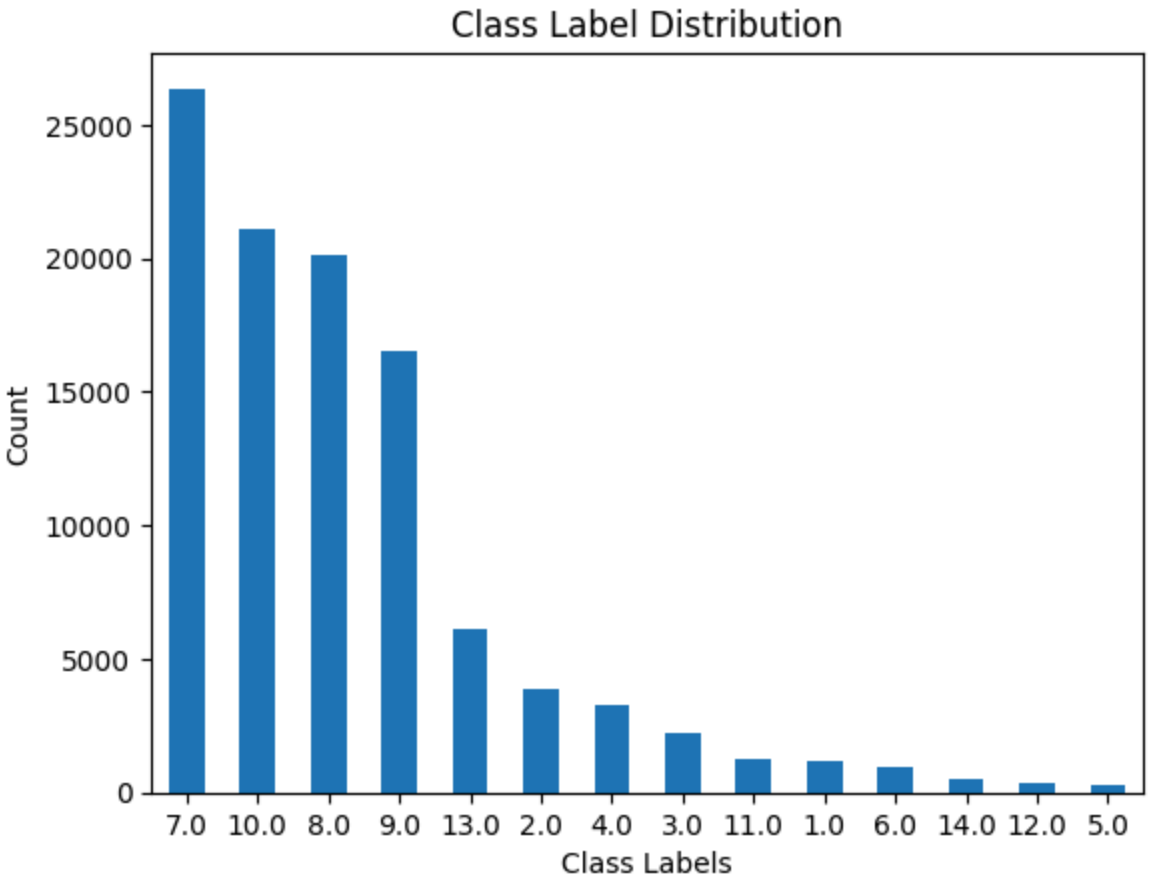
NULL VS NOT NULL

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

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

counts_df = pd.DataFrame({
    'Null': null_counts,
    'Non-Null': non_null_counts
})


counts_df.plot(kind='bar')

plt.title('Null vs Non-Null Values per Column')
plt.ylabel('Count')
plt.xlabel('Columns')
plt.xticks(rotation=90)
plt.show()
```



CLASS LABEL DISTRIBUTION

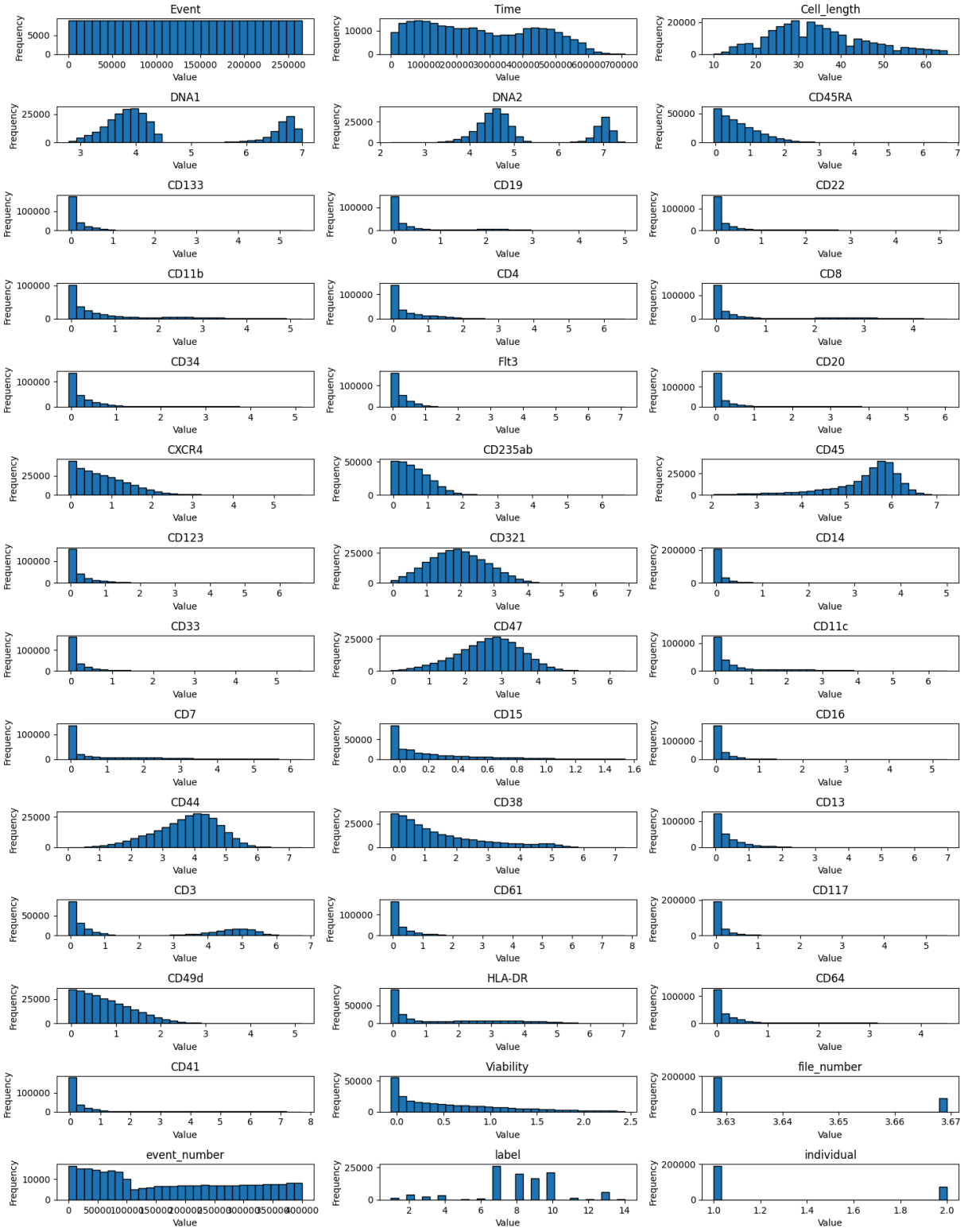
In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

class_counts = data['label'].value_counts()

class_counts.plot(kind='bar')

plt.title('Class Label Distribution')
plt.ylabel('Count')
plt.xlabel('Class Labels')
plt.xticks(rotation=0)
plt.show()
```



Histograms of each feature

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Select only numerical columns for histogram plotting
numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns

# Set up the figure for subplots
plt.figure(figsize=(15, 20))

# Iterate through numerical columns and create a histogram for each
for i, column in enumerate(numerical_columns, 1):
    plt.subplot(len(numerical_columns)//3 + 1, 3, i)
    plt.hist(data[column], bins=30, edgecolor='black')
    plt.title(column)
    plt.xlabel('Value')
    plt.ylabel('Frequency')

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

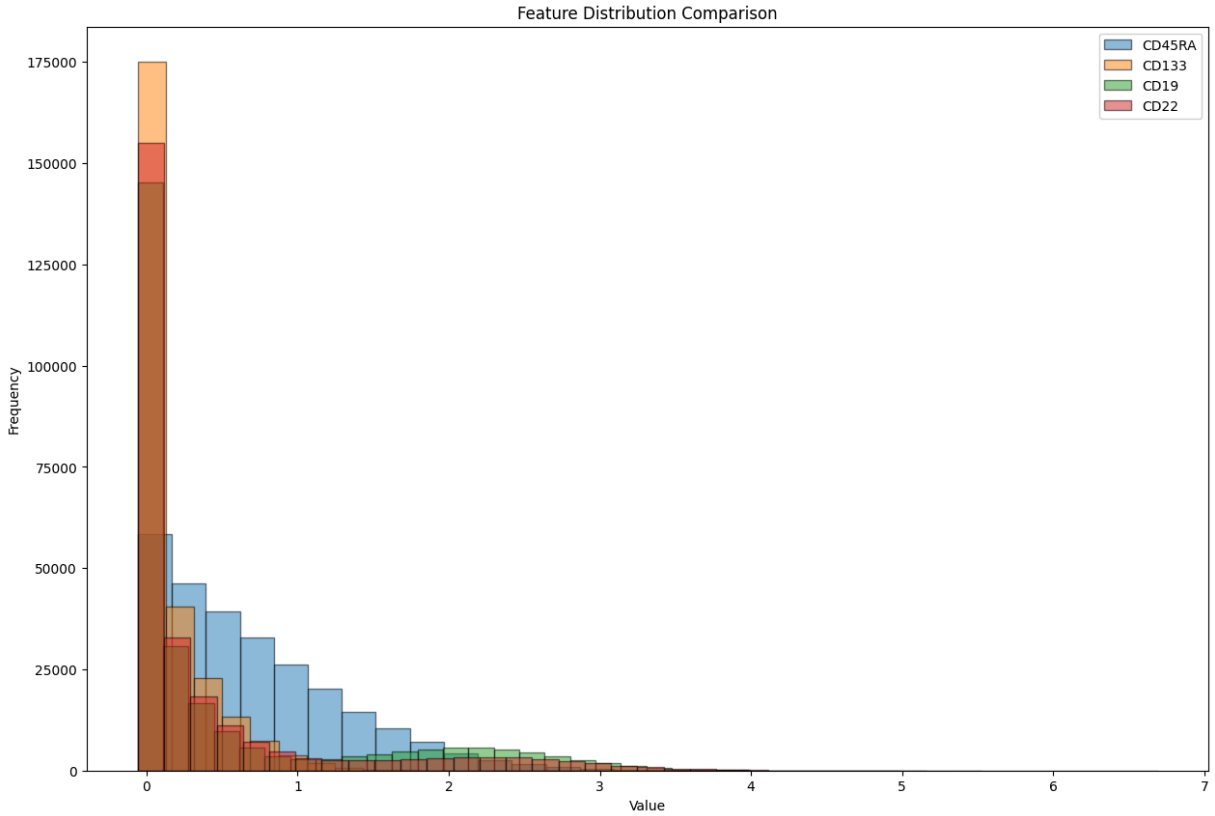


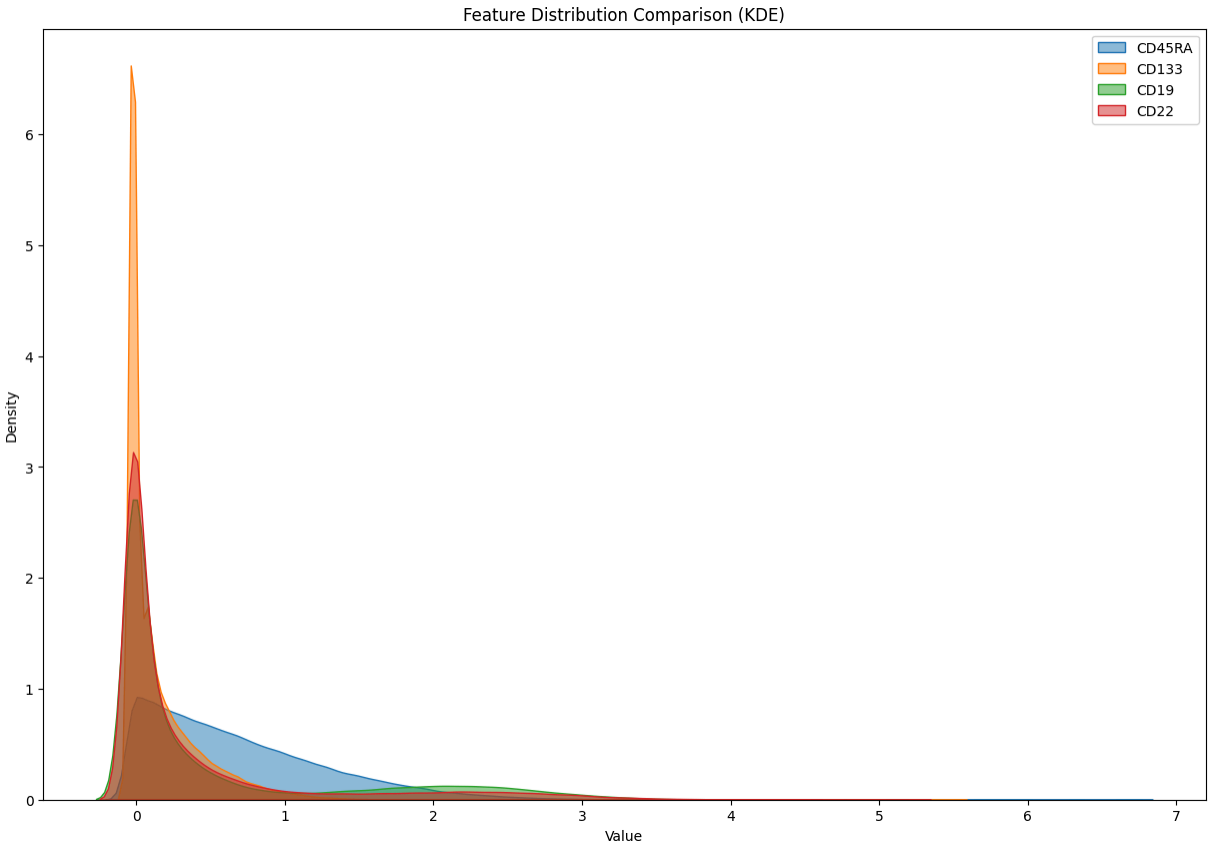
Feature Distribution Comparison Using Histograms and KDE

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Select features for comparison (adjust based on your dataset)
features_to_compare = ['CD45RA', 'CD133', 'CD19', 'CD22']  # Example feature

# Step 1: Histograms for feature distribution comparison
plt.figure(figsize=(15, 10))

for feature in features_to_compare:
    plt.hist(data[feature], bins=30, alpha=0.5, label=feature, edgecolor='bl

plt.title('Feature Distribution Comparison')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend()
plt.show()

# Step 2: Kernel Density Estimation (KDE) for smoother distribution comparis
plt.figure(figsize=(15, 10))

for feature in features_to_compare:
    sns.kdeplot(data[feature], label=feature, fill=True, alpha=0.5)

plt.title('Feature Distribution Comparison (KDE)')
plt.xlabel('Value')
plt.ylabel('Density')
plt.legend()
plt.show()
```



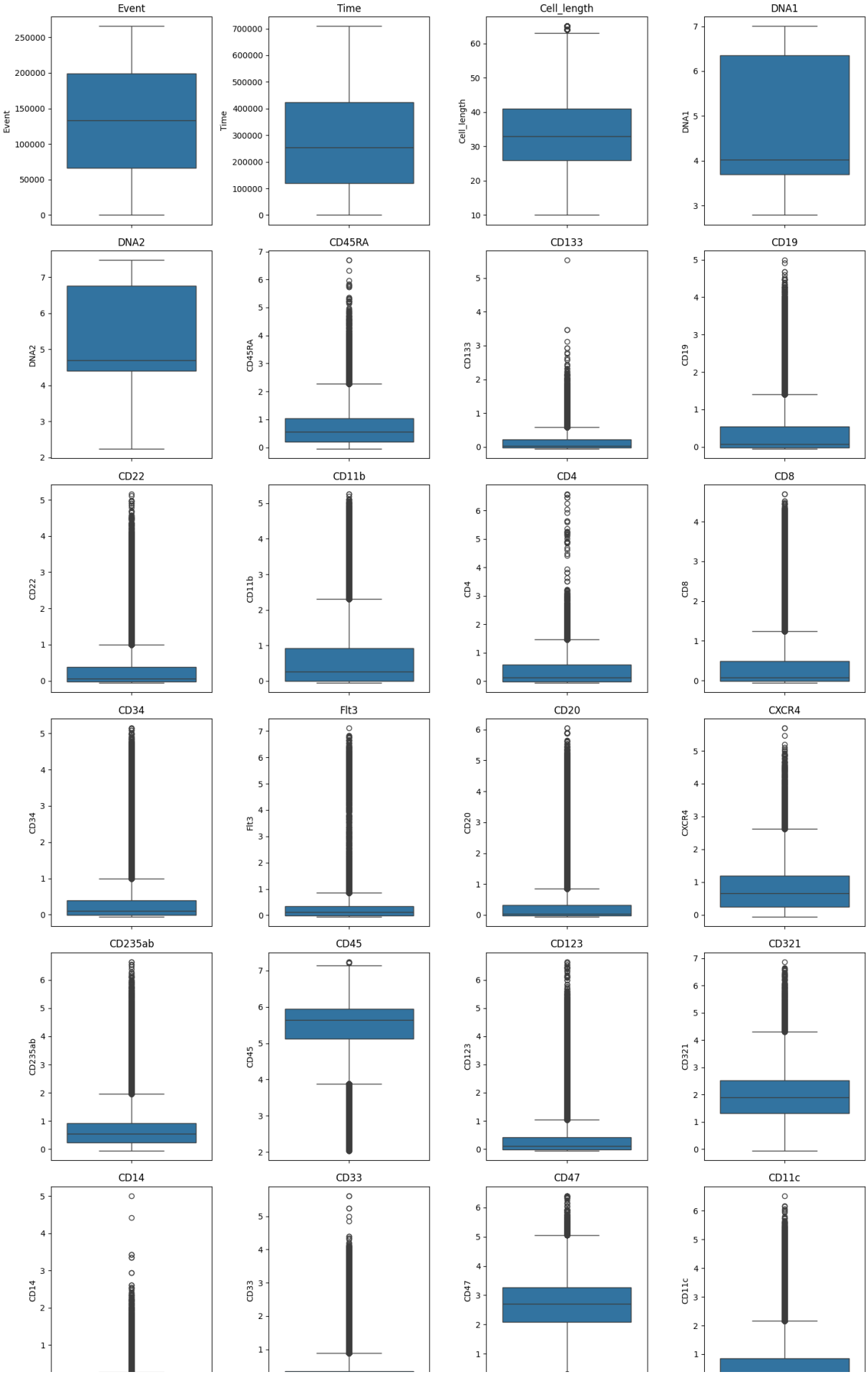


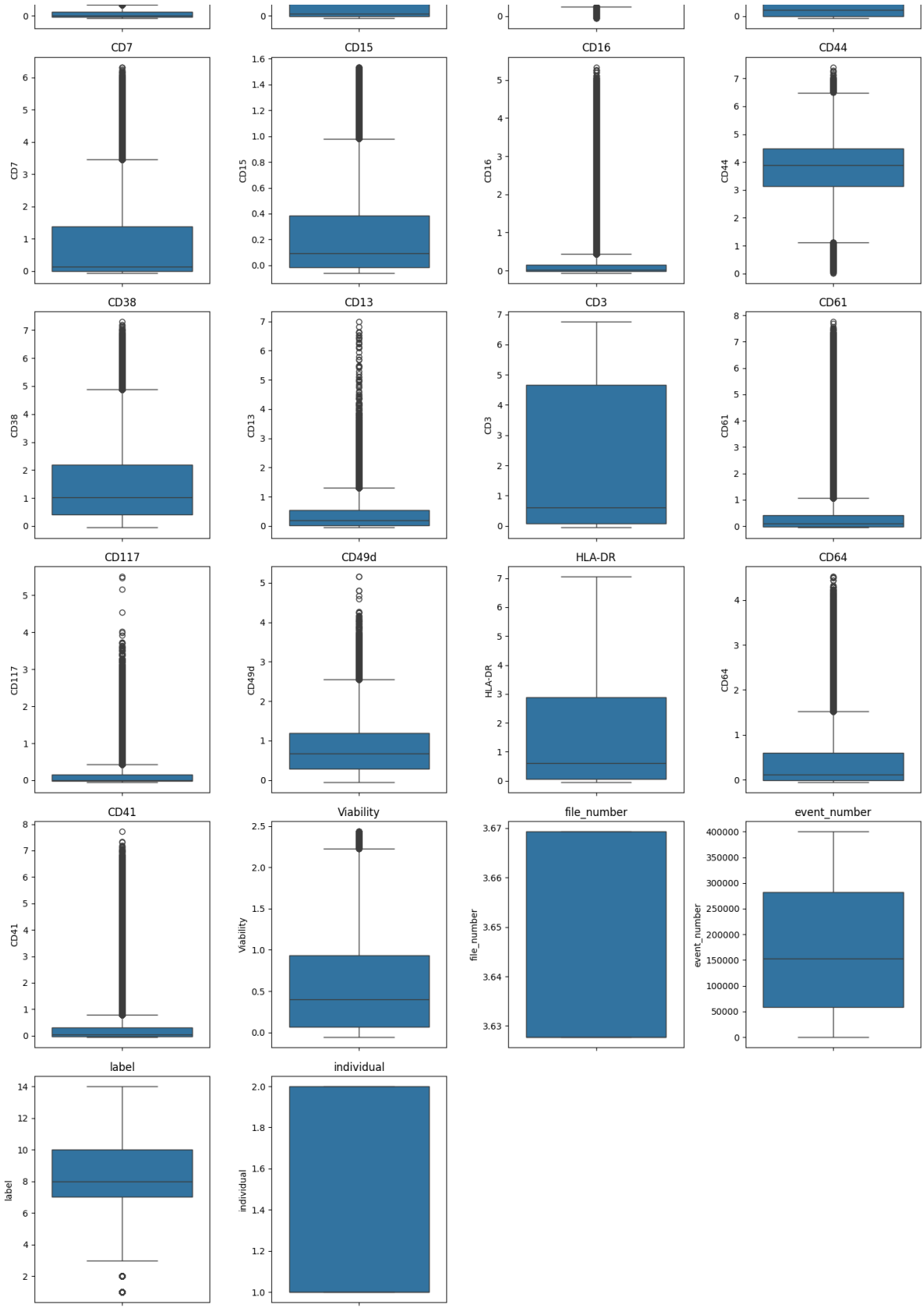
Box Plot

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Step 1: Box Plots for Numerical Features
numerical_features = data.select_dtypes(include=['float64', 'int64']).columr
rows = (len(numerical_features) // 4) + 1  # Calculate the number of rows ne

plt.figure(figsize=(15, rows * 4))
for i, feature in enumerate(numerical_features):
    plt.subplot(rows, 4, i + 1)
    sns.boxplot(data[feature])
    plt.title(feature)
plt.tight_layout()
plt.show()

# Step 2: Count Plots for Categorical Features
categorical_features = data.select_dtypes(include=['object']).columns  # Sel

plt.figure(figsize=(15, 10))
for i, feature in enumerate(categorical_features):
    plt.subplot(2, 2, i + 1)
    sns.countplot(x=data[feature], order=data[feature].value_counts().index)
```

```
    plt.title(feature)
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```


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

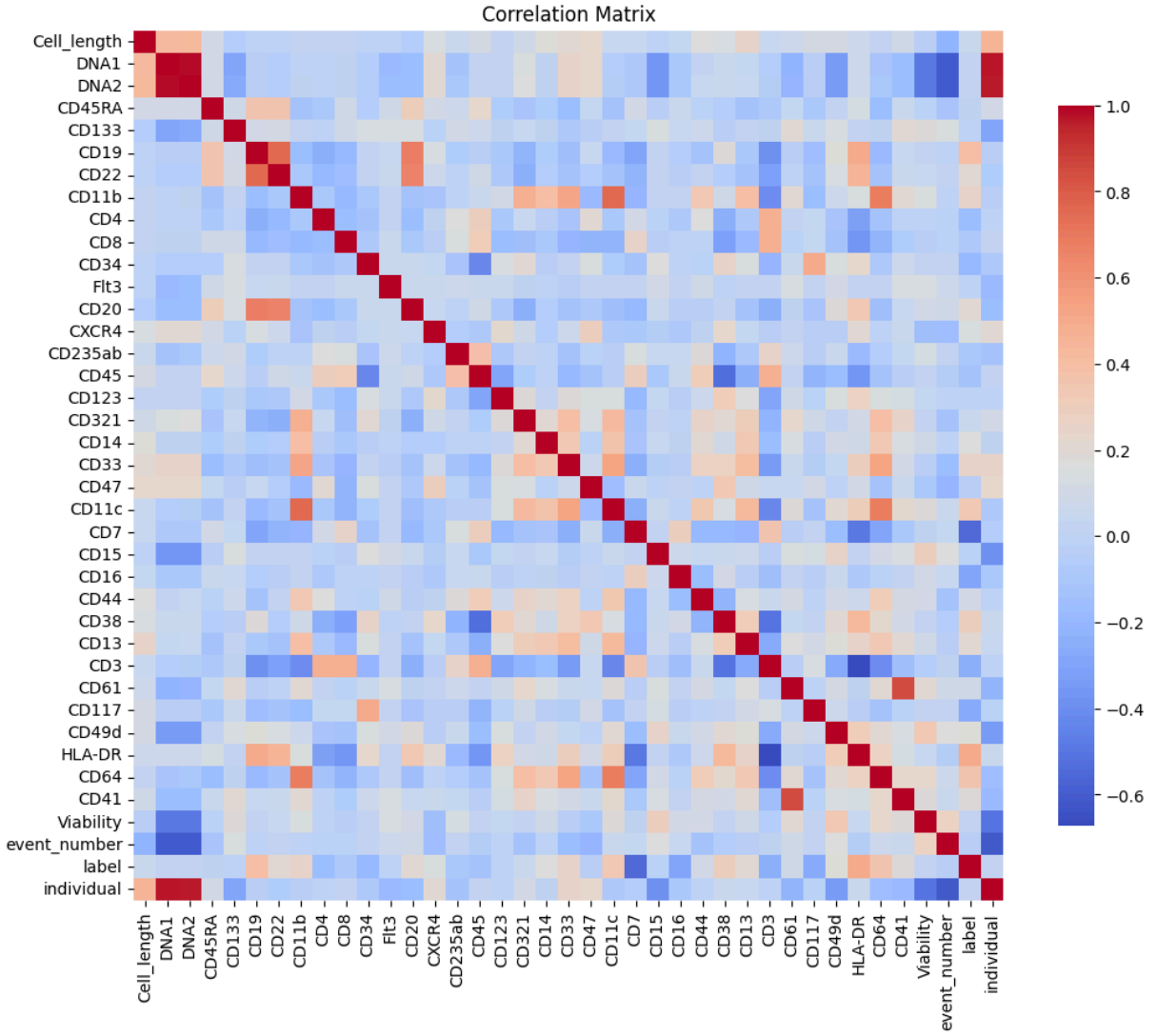


<Figure size 1500x1000 with 0 Axes>

Correlation Matrix

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the data
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Drop the specified columns
data = data.drop(columns=['file_number', 'Event', 'Time'])

# Calculate the correlation matrix
correlation_matrix = data.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Create a heatmap using Seaborn without annotations
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', square=True, c

plt.title('Correlation Matrix')
plt.show()
```



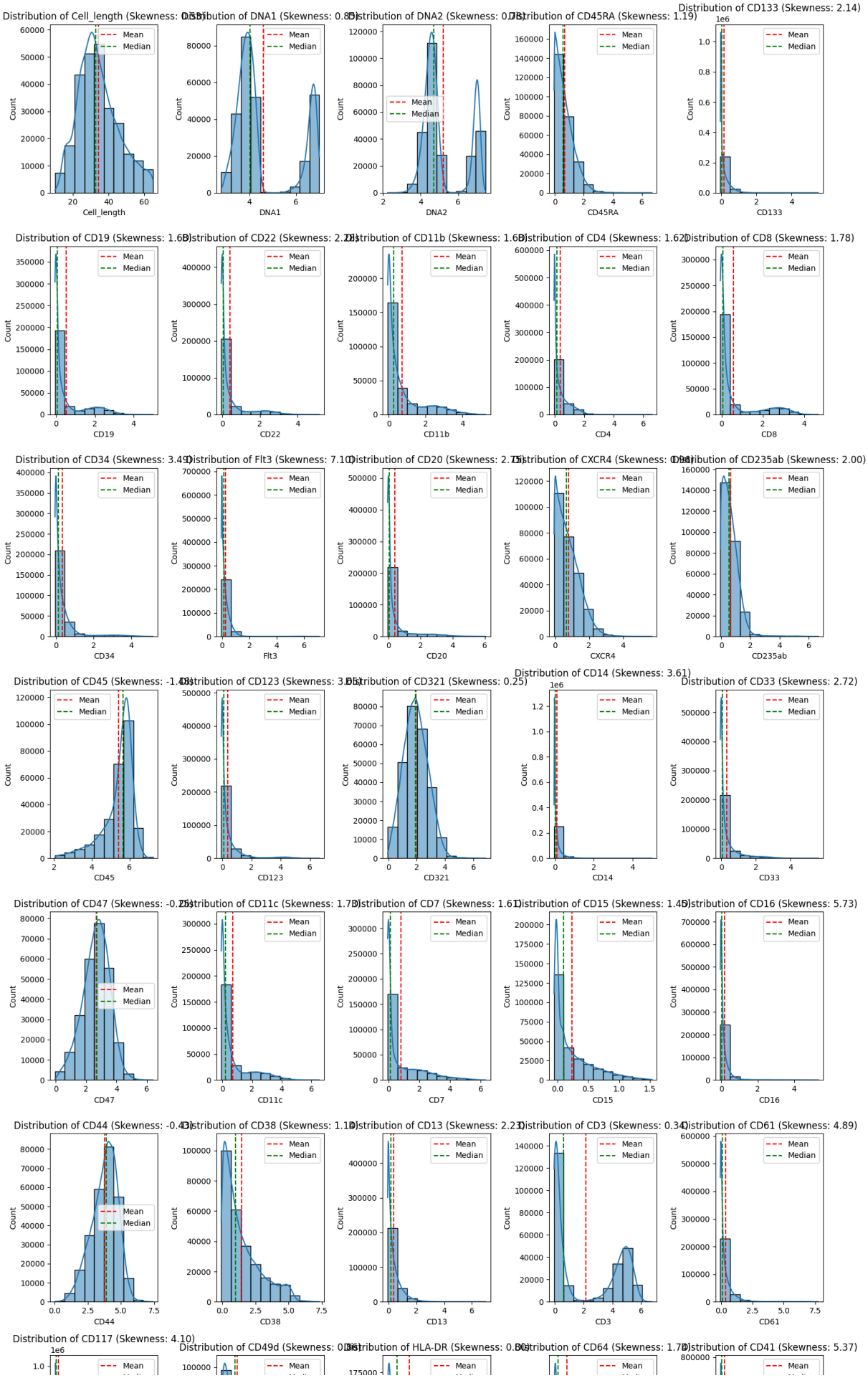
Skewness

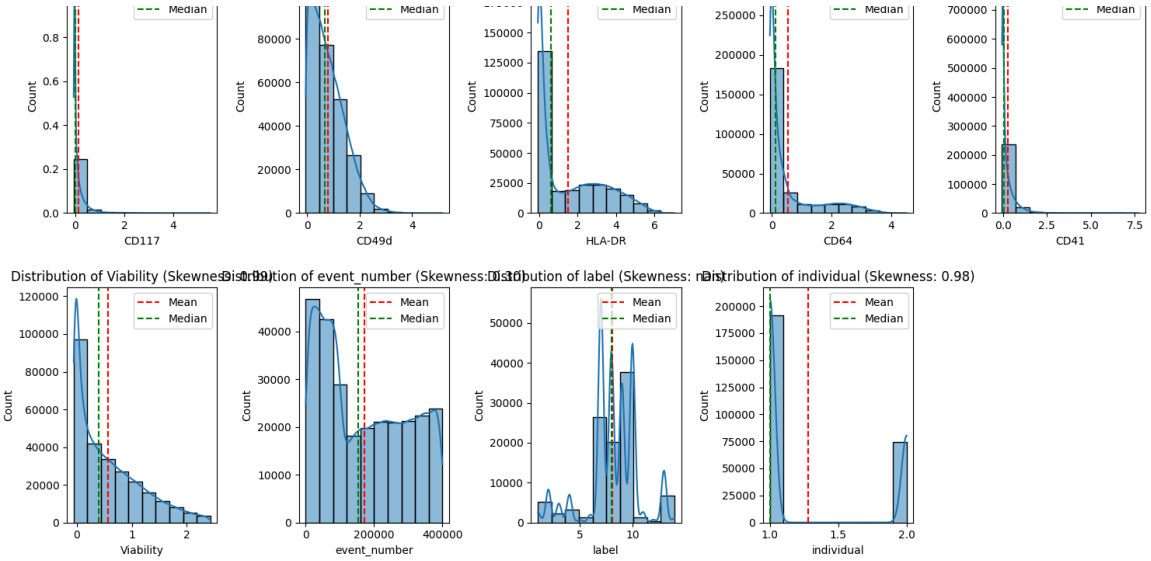
In [ ]:

```
import pandas as pd
from scipy.stats import skew
import matplotlib.pyplot as plt
import seaborn as sns
import math

# Load the data
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')
data = data.drop(columns=['file_number', 'Event', 'Time'])

# Calculate skewness
skewness = data.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
skewness_category = skewness.apply(categorize_skewness)

# Display skewness and its categorization
skewness_df = pd.DataFrame({'Skewness': skewness, 'Category': skewness_categ
print(skewness_df)

# Set the number of columns in the grid
n_cols = 5  # Adjust this value for number of plots per row
n_plots = len(data.columns)
n_rows = math.ceil(n_plots / n_cols)

# Create subplots grid
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))  # Adjust
axes = axes.flatten()  # Flatten axes array to make it easier to index

# Loop through columns and plot histograms on each subplot
for idx, col in enumerate(data.columns):
    sns.histplot(data[col], bins=10, kde=True, ax=axes[idx])
    axes[idx].set_title(f'Distribution of {col} (Skewness: {skewness[col]:.2
    axes[idx].axvline(data[col].mean(), color='red', linestyle='--', label='
    axes[idx].axvline(data[col].median(), color='green', linestyle='--', lab
    axes[idx].legend()

# Remove any unused subplots (if n_plots is not a perfect multiple of n_cols
for i in range(n_plots, len(axes)):
    fig.delaxes(axes[i])

# Ensure the layout is tight and the plot is shown properly
```

```
plt.tight_layout()
plt.show(block=True)  # Ensure plt.show() does not block rendering
```

	Skewness	Category
Cell_length	0.527832	Right-skewed
DNA1	0.845010	Right-skewed
DNA2	0.779167	Right-skewed
CD45RA	1.191595	Right-skewed
CD133	2.141953	Right-skewed
CD19	1.682609	Right-skewed
CD22	2.283181	Right-skewed
CD11b	1.679089	Right-skewed
CD4	1.622044	Right-skewed
CD8	1.775713	Right-skewed
CD34	3.492437	Right-skewed
Flt3	7.098151	Right-skewed
CD20	2.754699	Right-skewed
CXCR4	0.955342	Right-skewed
CD235ab	2.001479	Right-skewed
CD45	-1.484824	Left-skewed
CD123	3.648890	Right-skewed
CD321	0.247097	Approximately symmetrical
CD14	3.609006	Right-skewed
CD33	2.724977	Right-skewed
CD47	-0.250323	Approximately symmetrical
CD11c	1.733888	Right-skewed
CD7	1.606528	Right-skewed
CD15	1.445147	Right-skewed
CD16	5.733203	Right-skewed
CD44	-0.431589	Approximately symmetrical
CD38	1.141482	Right-skewed
CD13	2.234311	Right-skewed
CD3	0.342239	Approximately symmetrical
CD61	4.894707	Right-skewed
CD117	4.097508	Right-skewed
CD49d	0.856805	Right-skewed
HLA-DR	0.795359	Right-skewed
CD64	1.743733	Right-skewed
CD41	5.366314	Right-skewed
Viability	0.985417	Right-skewed
event_number	0.304116	Approximately symmetrical
label	NaN	Approximately symmetrical
individual	0.982030	Right-skewed


Distribution of Cell_length (Skewness: 0.53)
Distribution of DNA1 (Skewness: 0.85)
Distribution of DNA2 (Skewness: 0.78)
Distribution of CD45RA (Skewness: 1.19)
Distribution of CD133 (Skewness: 2.14)
Distribution of CD19 (Skewness: 1.68)
Distribution of CD22 (Skewness: 2.28)
Distribution of CD11b (Skewness: 1.68)
Distribution of CD4 (Skewness: 1.62)
Distribution of CD8 (Skewness: 1.78)
Distribution of CD34 (Skewness: 3.49)
Distribution of Flt3 (Skewness: 7.10)
Distribution of CD20 (Skewness: 2.75)
Distribution of CXCR4 (Skewness: 1.96)
Distribution of CD235ab (Skewness: 2.00)
Distribution of CD45 (Skewness: -1.48)
Distribution of CD123 (Skewness: 3.05)
Distribution of CD321 (Skewness: 0.25)
Distribution of CD14 (Skewness: 3.61)
Distribution of CD33 (Skewness: 2.72)
Distribution of CD47 (Skewness: -0.25)
Distribution of CD11c (Skewness: 1.78)
Distribution of CD7 (Skewness: 1.61)
Distribution of CD15 (Skewness: 1.45)
Distribution of CD16 (Skewness: 5.73)
Distribution of CD44 (Skewness: -0.43)
Distribution of CD38 (Skewness: 1.10)
Distribution of CD13 (Skewness: 2.21)
Distribution of CD3 (Skewness: 0.34)
Distribution of CD61 (Skewness: 4.89)
Distribution of CD117 (Skewness: 4.10)
Distribution of CD49d (Skewness: 0.86)
Distribution of HLA-DR (Skewness: 0.80)
Distribution of CD64 (Skewness: 1.76)
Distribution of CD41 (Skewness: 5.37)
Mean
Median
Count



In [ ]:

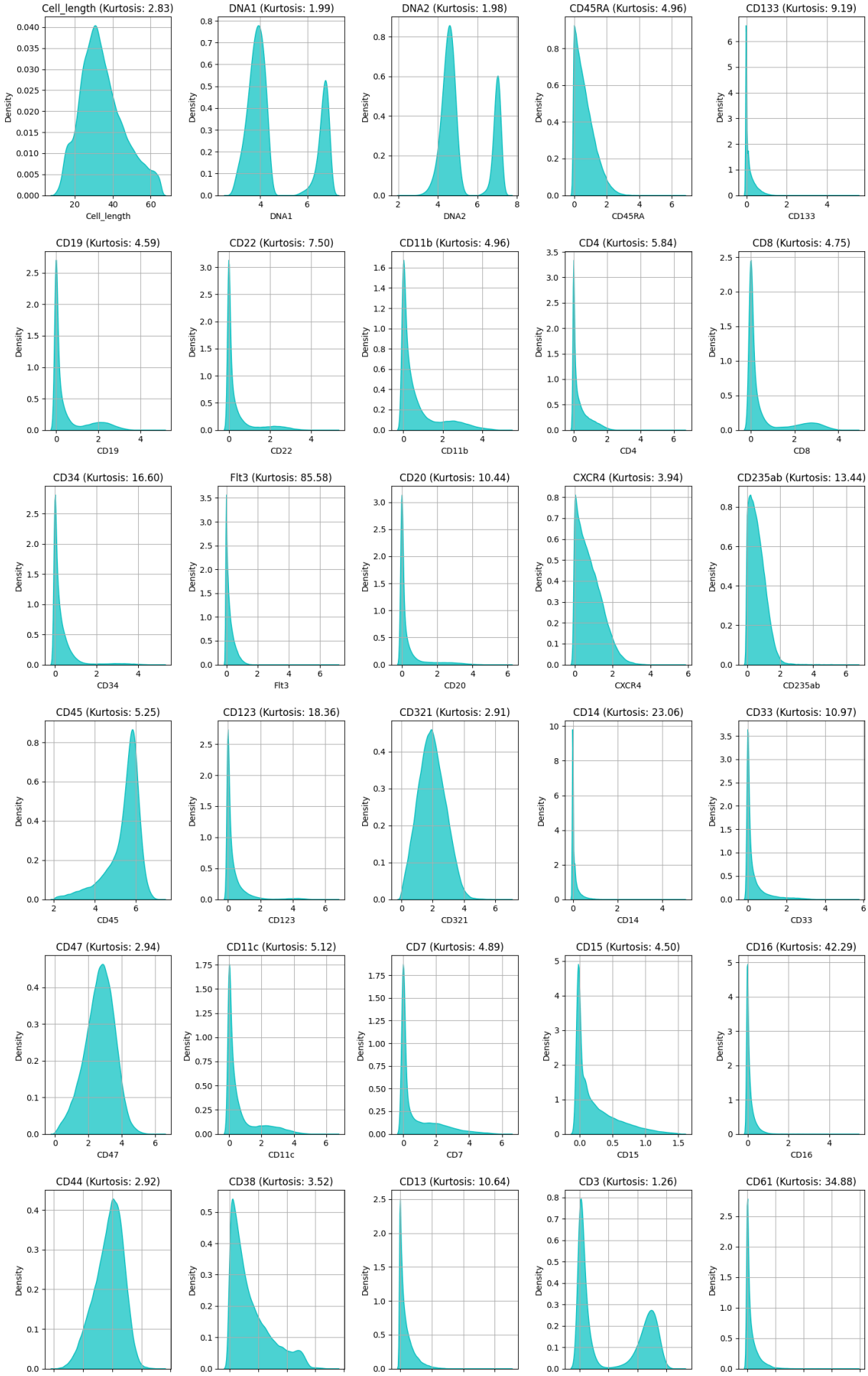
Kurtosis

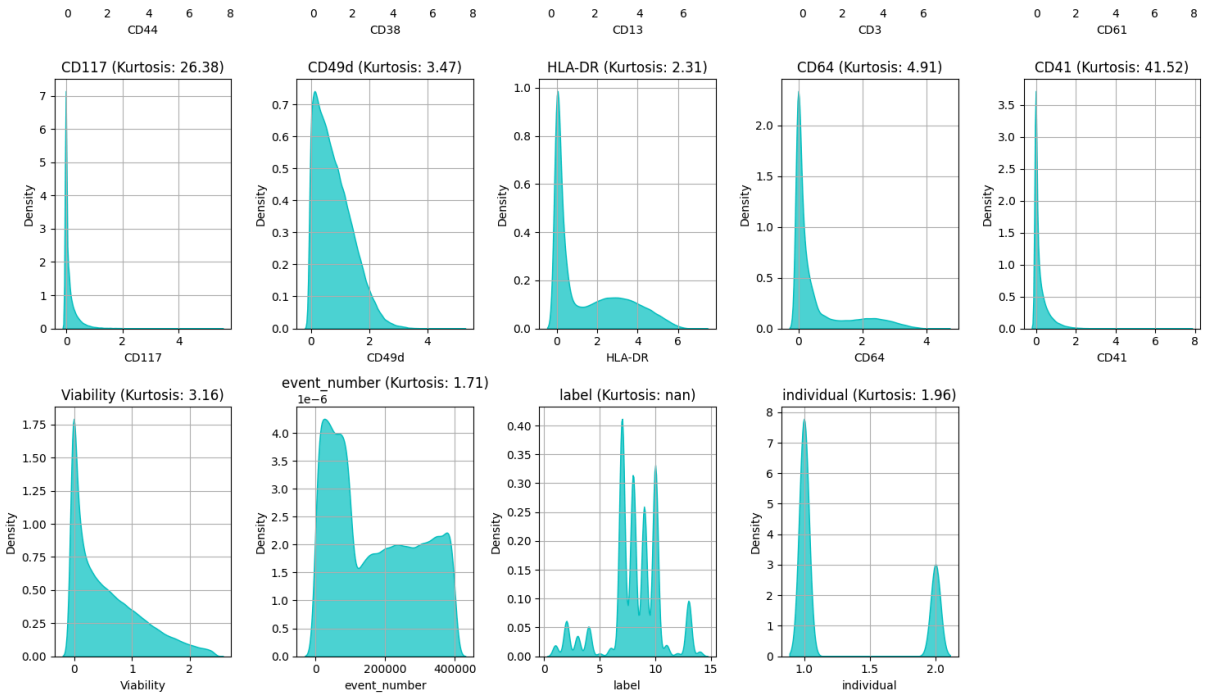
In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import kurtosis
import math

# Load the data
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Drop the specified columns
data = data.drop(columns=['file_number', 'Event', 'Time'])

# Calculate kurtosis for each column
kurtosis_values = data.apply(kurtosis, fisher=False)  # Fisher=False gives F

# Create a DataFrame with kurtosis values
kurtosis_df = pd.DataFrame({'Column': data.columns, 'Kurtosis': kurtosis_val

# Categorize the kurtosis values (Leptokurtic, Mesokurtic, Platykurtic)
def categorize_kurtosis(value):
    if value > 3:
        return 'Leptokurtic (heavy tails)'
    elif value < 3:
        return 'Platykurtic (light tails)'
    else:
        return 'Mesokurtic (normal tails)'

kurtosis_df['Category'] = kurtosis_df['Kurtosis'].apply(categorize_kurtosis)

# Print the kurtosis values and their categories
print(kurtosis_df)

# Set the number of columns in the grid
```

```
n_cols = 5  # You can adjust this to control how many plots per row
n_plots = len(data.columns)
n_rows = math.ceil(n_plots / n_cols)

# Create subplots grid
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 4 * n_rows))  # Adjust
axes = axes.flatten()  # Flatten axes array to make it easier to index

# Loop through columns and plot KDE on each subplot
for idx, column in enumerate(data.columns):
    sns.kdeplot(data[column].dropna(), color='c', fill=True, alpha=0.7, ax=a
    axes[idx].set_title(f'{column} (Kurtosis: {kurtosis_df.loc[kurtosis_df["
    axes[idx].set_xlabel(column)
    axes[idx].set_ylabel('Density')
    axes[idx].grid(True)

# Remove any unused subplots (if n_plots is not a perfect multiple of n_cols
for i in range(n_plots, len(axes)):
    fig.delaxes(axes[i])

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

	Column	Kurtosis	Category
Cell_length	Cell_length	2.834033	Platykurtic (light tails)
DNA1	DNA1	1.994037	Platykurtic (light tails)
DNA2	DNA2	1.975021	Platykurtic (light tails)
CD45RA	CD45RA	4.964272	Leptokurtic (heavy tails)
CD133	CD133	9.190066	Leptokurtic (heavy tails)
CD19	CD19	4.590887	Leptokurtic (heavy tails)
CD22	CD22	7.500223	Leptokurtic (heavy tails)
CD11b	CD11b	4.964495	Leptokurtic (heavy tails)
CD4	CD4	5.844261	Leptokurtic (heavy tails)
CD8	CD8	4.745776	Leptokurtic (heavy tails)
CD34	CD34	16.596416	Leptokurtic (heavy tails)
Flt3	Flt3	85.583534	Leptokurtic (heavy tails)
CD20	CD20	10.435449	Leptokurtic (heavy tails)
CXCR4	CXCR4	3.936307	Leptokurtic (heavy tails)
CD235ab	CD235ab	13.440586	Leptokurtic (heavy tails)
CD45	CD45	5.246770	Leptokurtic (heavy tails)
CD123	CD123	18.361217	Leptokurtic (heavy tails)
CD321	CD321	2.914593	Platykurtic (light tails)
CD14	CD14	23.062535	Leptokurtic (heavy tails)
CD33	CD33	10.967536	Leptokurtic (heavy tails)
CD47	CD47	2.943834	Platykurtic (light tails)
CD11c	CD11c	5.117156	Leptokurtic (heavy tails)
CD7	CD7	4.885115	Leptokurtic (heavy tails)
CD15	CD15	4.504387	Leptokurtic (heavy tails)
CD16	CD16	42.287749	Leptokurtic (heavy tails)
CD44	CD44	2.918792	Platykurtic (light tails)
CD38	CD38	3.521190	Leptokurtic (heavy tails)
CD13	CD13	10.637564	Leptokurtic (heavy tails)
CD3	CD3	1.264612	Platykurtic (light tails)
CD61	CD61	34.878020	Leptokurtic (heavy tails)
CD117	CD117	26.375108	Leptokurtic (heavy tails)
CD49d	CD49d	3.468119	Leptokurtic (heavy tails)
HLA-DR	HLA-DR	2.309924	Platykurtic (light tails)
CD64	CD64	4.910631	Leptokurtic (heavy tails)
CD41	CD41	41.521113	Leptokurtic (heavy tails)
Viability	Viability	3.156935	Leptokurtic (heavy tails)
event_number	event_number	1.706183	Platykurtic (light tails)
label	label	NaN	Mesokurtic (normal tails)
individual	individual	1.964382	Platykurtic (light tails)


Cell_length (Kurtosis: 2.83)
DNA1 (Kurtosis: 1.99)
DNA2 (Kurtosis: 1.98)
CD45RA (Kurtosis: 4.96)
CD133 (Kurtosis: 9.19)
CD19 (Kurtosis: 4.59)
CD22 (Kurtosis: 7.50)
CD11b (Kurtosis: 4.96)
CD4 (Kurtosis: 5.84)
CD8 (Kurtosis: 4.75)
CD34 (Kurtosis: 16.60)
Flt3 (Kurtosis: 85.58)
CD20 (Kurtosis: 10.44)
CXCR4 (Kurtosis: 3.94)
CD235ab (Kurtosis: 13.44)
CD45 (Kurtosis: 5.25)
CD123 (Kurtosis: 18.36)
CD321 (Kurtosis: 2.91)
CD14 (Kurtosis: 23.06)
CD33 (Kurtosis: 10.97)
CD47 (Kurtosis: 2.94)
CD11c (Kurtosis: 5.12)
CD7 (Kurtosis: 4.89)
CD15 (Kurtosis: 4.50)
CD16 (Kurtosis: 42.29)
CD44 (Kurtosis: 2.92)
CD38 (Kurtosis: 3.52)
CD13 (Kurtosis: 10.64)
CD3 (Kurtosis: 1.26)
CD61 (Kurtosis: 34.88)
Density

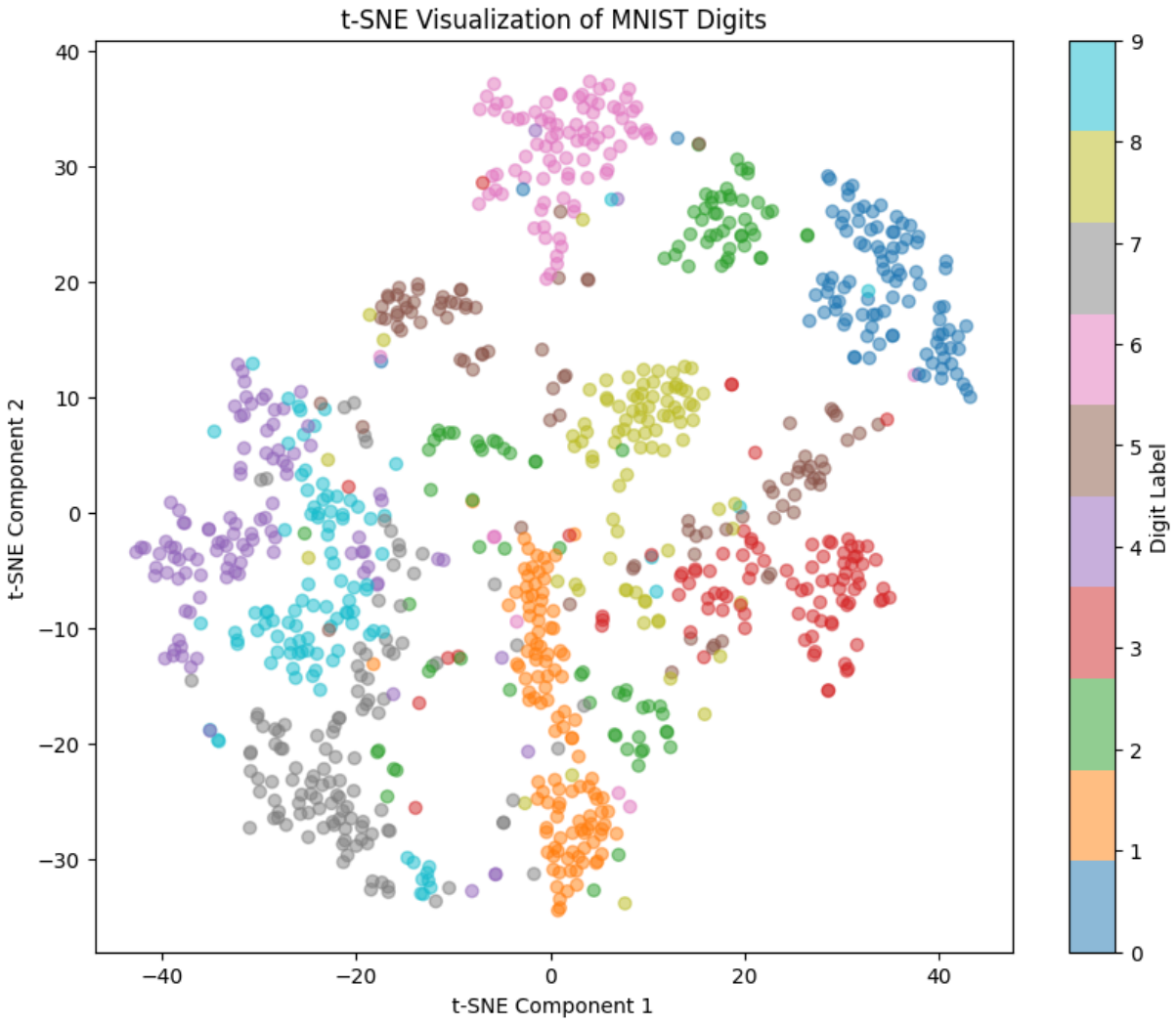


CLASS WORK CODE

In [1]:

```
import tensorflow as tf
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets
train_images = train_images.astype('float32') / 255.0
test_images = test_images.astype('float32') / 255.0

# Flatten the images and take a subset
n_samples = 1000
train_images_flat = train_images[:n_samples].reshape(n_samples, -1)
train_labels_subset = train_labels[:n_samples]

# Perform t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)
train_images_embedded = tsne.fit_transform(train_images_flat)

# Plot the t-SNE results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(train_images_embedded[:, 0], train_images_embedded[:,
plt.colorbar(scatter, label='Digit Label')
plt.title('t-SNE Visualization of MNIST Digits')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

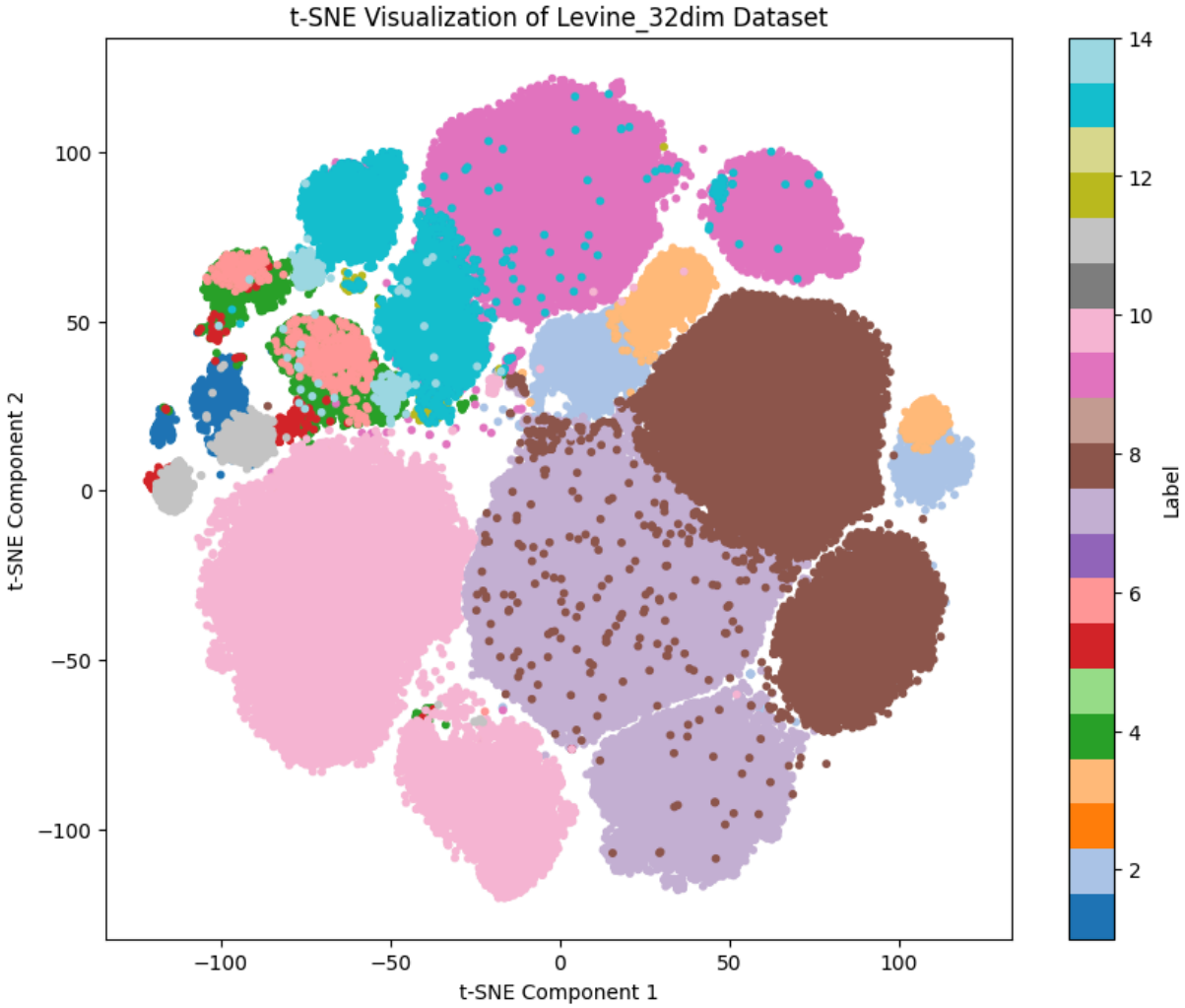


T-SNE

In [ ]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('/content/Levine_32dim.fcs.csv')

# Exclude the specified columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_num
data_filtered = data.drop(columns=exclude_columns)

# Standardize the data (z-score normalization)
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data_filtered)

# Perform t-SNE
tsne = TSNE(n_components=2, random_state=42, perplexity=30)  # You can adjus
tsne_results = tsne.fit_transform(data_standardized)
```

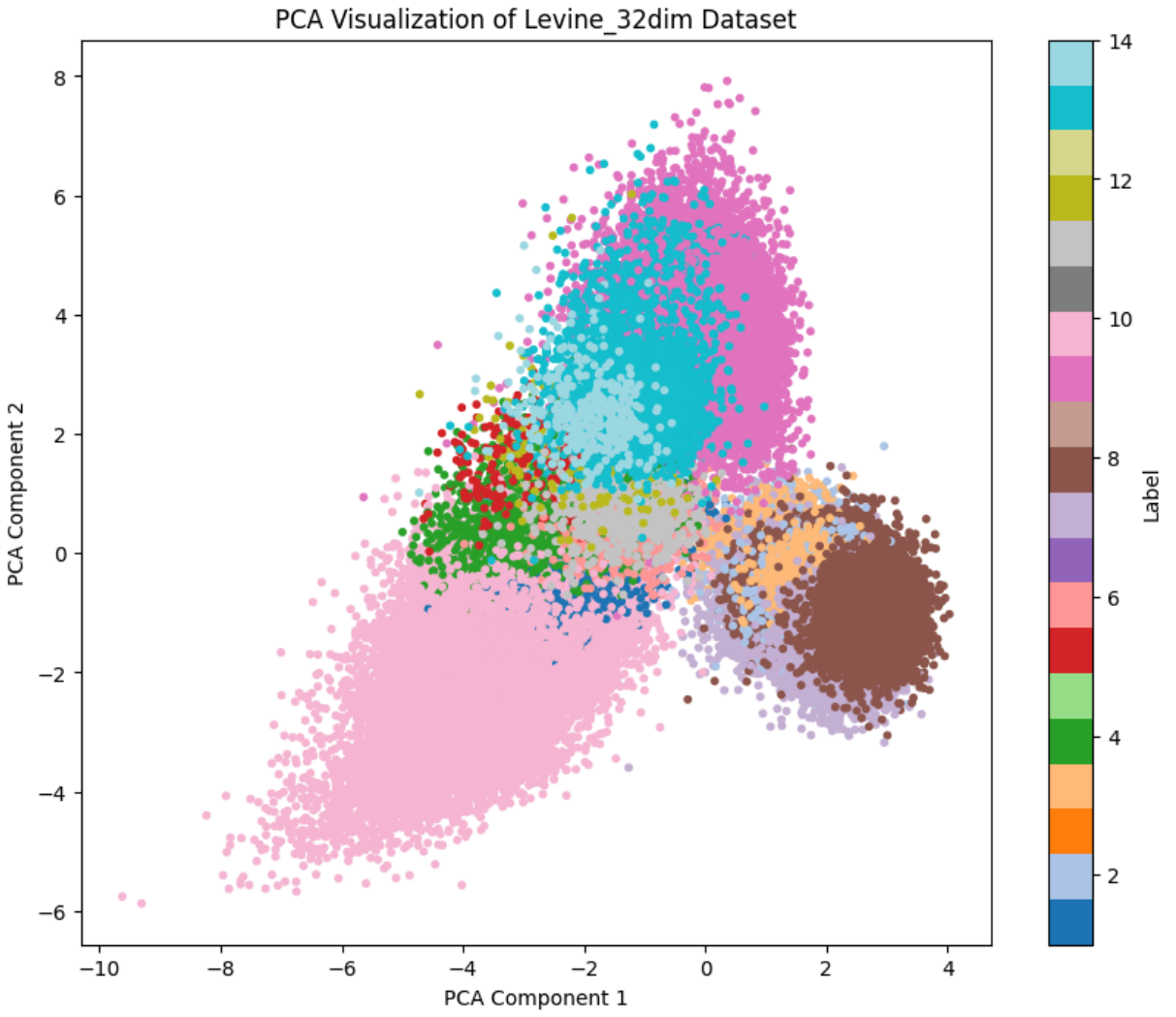
```
# Add the t-SNE results to the original data for visualization
data['t-SNE Component 1'] = tsne_results[:, 0]
data['t-SNE Component 2'] = tsne_results[:, 1]

# Plot the t-SNE visualization
plt.figure(figsize=(10, 8))
scatter = plt.scatter(data['t-SNE Component 1'], data['t-SNE Component 2'],
plt.colorbar(scatter, label='Label')
plt.title('t-SNE Visualization of Levine_32dim Dataset')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



PCA

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('/content/Levine_32dim.fcs.csv')
```

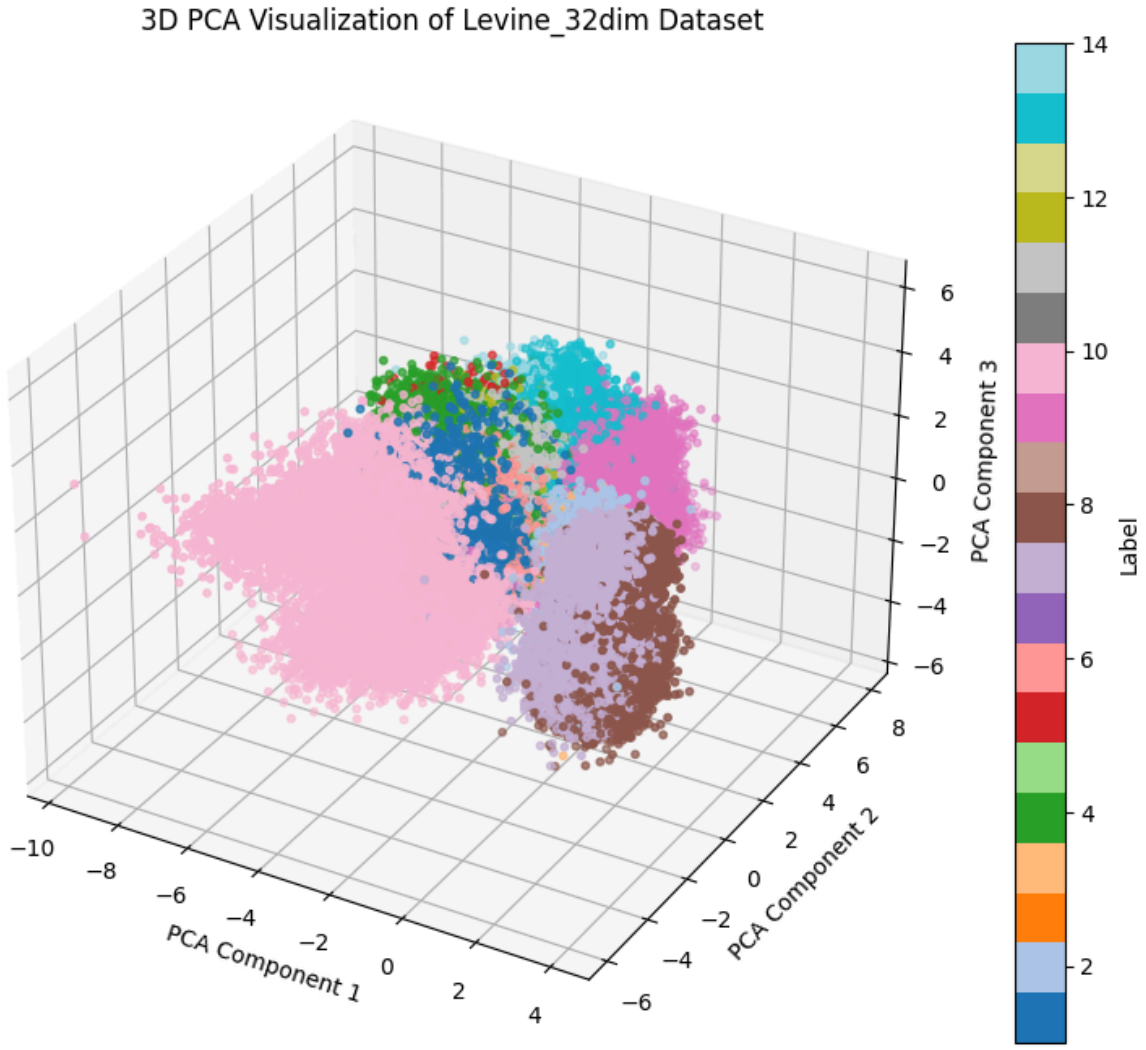
```python
# Exclude the specified columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_num
data_filtered = data.drop(columns=exclude_columns)

# Standardize the data (z-score normalization)
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data_filtered)

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

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

# Plot the PCA results
plt.figure(figsize=(10, 8))
scatter = plt.scatter(data['PCA Component 1'], data['PCA Component 2'], c=da
plt.colorbar(scatter, label='Label')
plt.title('PCA Visualization of Levine_32dim Dataset')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



3D PCA graph

In [ ]:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # Importing 3D plotting

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Exclude the specified columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_num
data_filtered = data.drop(columns=exclude_columns)

# Standardize the data (z-score normalization)
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data_filtered)

# Perform PCA
pca = PCA(n_components=3)  # Reduce to 3 dimensions for 3D visualization
pca_result = pca.fit_transform(data_standardized)

# Add the PCA results to the original data for visualization
data['PCA Component 1'] = pca_result[:, 0]
data['PCA Component 2'] = pca_result[:, 1]
data['PCA Component 3'] = pca_result[:, 2]

# Plot the PCA results in 3D
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Create a 3D scatter plot
scatter = ax.scatter(data['PCA Component 1'], data['PCA Component 2'], data[
                     c=data['label'], cmap='tab20', s=10)

# Add color bar and labels
plt.colorbar(scatter, label='Label')
ax.set_title('3D PCA Visualization of Levine_32dim Dataset')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')

# Show the plot
plt.show()
```



Variance, Cumulative Proportion and S.D.

In [ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the dataset
data = pd.read_csv('/content/drive/MyDrive/Datasets/Levine_32dim.fcs.csv')

# Exclude the specified columns
exclude_columns = ['Event', 'Time', 'Cell_length', 'file_number', 'event_num
data_filtered = data.drop(columns=exclude_columns)

# Standardize the data (z-score normalization)
scaler = StandardScaler()
data_standardized = scaler.fit_transform(data_filtered)

# Perform PCA
pca = PCA(n_components=4)  # Use 4 principal components
pca.fit(data_standardized)

# Extract the required information
```

```
explained_variance = pca.explained_variance_ratio_
cumulative_variance = explained_variance.cumsum()
standard_deviation = pca.singular_values_ / (len(data_standardized) - 1)**0.

# Create a DataFrame for the output
pca_summary = pd.DataFrame({
    'PC1': [standard_deviation[0], explained_variance[0], cumulative_varianc
    'PC2': [standard_deviation[1], explained_variance[1], cumulative_varianc
    'PC3': [standard_deviation[2], explained_variance[2], cumulative_varianc
    'PC4': [standard_deviation[3], explained_variance[3], cumulative_varianc
}, index=['Standard Deviation', 'Proportion of Variance', 'Cumulative Propor

# Round the numbers for better readability
pca_summary = pca_summary.map(lambda x: f'{x:.4f}')

# Apply styles to the DataFrame
styled_summary = (pca_summary.style
                  .set_caption("PCA Summary")
                  .set_table_styles(
                      [{'selector': 'caption', 'props': [('font-size', '16px
                  )
                  .background_gradient(cmap='coolwarm', axis=None)
                  .set_properties(**{'text-align': 'center'})
)

# Hiding the index column manually (workaround)
styled_summary.set_table_styles({
    'index': [{'selector': '', 'props': 'display:none;'}]  # Hides the index
})

# Display the styled DataFrame
styled_summary
```

Out[ ]:

PCA Summary

	PC1	PC2	PC3	PC4
Standard Deviation	2.3277	1.9574	1.8780	1.6067
Proportion of Variance	0.1548	0.1095	0.1008	0.0738
Cumulative Proportion	0.1548	0.2643	0.3650	0.4388