MIT World Peace University

Data Science for Cybersecurity and Forensics
Third Year B. Tech, Semester 6

Implementation of K-Means Clustering in Python

Assignment 7

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 10

April 17, 2024

Contents

1 Aim **1**

2 Objectives **1**

3 Theory **1**
3.1 K-Means Clustering 1
3.2 K-Means Clustering Algorithm 1

4 Procedure **2**

5 Platform **2**

6 Requirements **2**

7 Code **3**

8 FAQs **35**

9 Conclusion **35**

1 Aim

Implement K-Means Clustering in Python using IOT Based Attacks dataset.

2 Objectives

1. To understand the concept of K-Means Clustering.
2. To implement K-Means Clustering in Python.

3 Theory

3.1 K-Means Clustering

K-Means clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct clusters. The algorithm aims to minimize the variance within each cluster and maximize the variance between clusters.

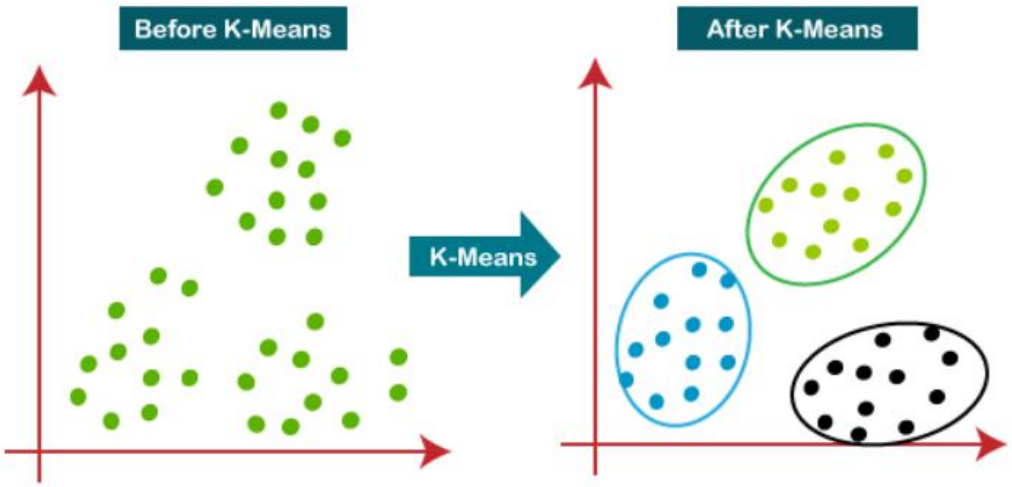


Figure 1: K Means Clustering Example

3.2 K-Means Clustering Algorithm

The K-Means clustering algorithm can be summarized in the following steps:

1. **Initialization**: Randomly initialize K cluster centroids.
2. **Assign Data Points to Nearest Centroid**: Assign each data point to the cluster with the nearest centroid.
3. **Update Centroids**: Recalculate the centroids of each cluster based on the mean of the data points assigned to that cluster.
4. **Repeat Steps 2 and 3**: Iterate the process of assigning data points to clusters and updating centroids until convergence.

5. **Convergence Criteria**: The algorithm converges when the centroids no longer change significantly between iterations or when a specified number of iterations is reached.

The objective function of K-Means clustering can be defined as minimizing the within-cluster sum of squared distances:

$$\underset{C}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Where: - C represents the set of clusters. - C_i represents the data points assigned to cluster i. - μ_i represents the centroid of cluster i. - $\|\cdot\|$ denotes the Euclidean distance.

The algorithm converges to a locally optimal solution, and the quality of the clustering depends on the initial placement of centroids and the choice of K. K-Means is efficient and scalable for large datasets but may converge to suboptimal solutions depending on the initial centroids and data distribution.

4 Procedure

1. Import the required python packages.
2. Load the dataset.
3. Data analysis.
4. Split the dataset into dependent/independent variables.
5. Split data into Train/Test sets.
6. Train the regression model.
7. Predict the result.

5 Platform

Operating System: Windows 11
IDEs or Text Editors Used: Visual Studio Code
Compilers or Interpreters: Python 3.10.1

6 Requirements

```
python==3.10.1
matplotlib==3.8.3
numpy==1.26.4
pandas==2.2.2
seaborn==0.13.2
```

7 Code

[95]:
```
# implementing k means clustering on iot based attacks dataset
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[96]:
```
# importing the dataset
df = pd.read_csv("part-00000-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.csv")
df.head()
```

[96]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	\
0	0.000000	54.00	6.00	64.00	0.329807	
1	0.000000	57.04	6.33	64.00	4.290556	
2	0.000000	0.00	1.00	64.00	33.396799	
3	0.328175	76175.00	17.00	64.00	4642.133010	
4	0.117320	101.73	6.11	65.91	6.202211	

	Srate	Drate	fin_flag_number	syn_flag_number	rst_flag_number	...	\
0	0.329807	0.0	1.0	0.0	1.0	...	
1	4.290556	0.0	0.0	0.0	0.0	...	
2	33.396799	0.0	0.0	0.0	0.0	...	
3	4642.133010	0.0	0.0	0.0	0.0	...	
4	6.202211	0.0	0.0	1.0	0.0	...	

	Std	Tot size	IAT	Number	Magnitue	Radius	\
0	0.000000	54.00	8.334383e+07	9.5	10.392305	0.000000	
1	2.822973	57.04	8.292607e+07	9.5	10.464666	4.010353	
2	0.000000	42.00	8.312799e+07	9.5	9.165151	0.000000	
3	0.000000	50.00	8.301570e+07	9.5	10.000000	0.000000	
4	23.113111	57.88	8.297300e+07	9.5	11.346876	32.716243	

	Covariance	Variance	Weight	label
0	0.000000	0.00	141.55	DDoS-RSTFINFlood
1	160.987842	0.05	141.55	DoS-TCP_Flood
2	0.000000	0.00	141.55	DDoS-ICMP_Flood
3	0.000000	0.00	141.55	DoS-UDP_Flood
4	3016.808286	0.19	141.55	DoS-SYN_Flood

[5 rows x 47 columns]

[97]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 238687 entries, 0 to 238686
```

```
Data columns (total 47 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   flow_duration    238687 non-null  float64
 1   Header_Length    238687 non-null  float64
 2   Protocol Type    238687 non-null  float64
 3   Duration         238687 non-null  float64
 4   Rate             238687 non-null  float64
 5   Srate            238687 non-null  float64
 6   Drate            238687 non-null  float64
 7   fin_flag_number  238687 non-null  float64
 8   syn_flag_number  238687 non-null  float64
 9   rst_flag_number  238687 non-null  float64
 10  psh_flag_number  238687 non-null  float64
 11  ack_flag_number  238687 non-null  float64
 12  ece_flag_number  238687 non-null  float64
 13  cwr_flag_number  238687 non-null  float64
 14  ack_count        238687 non-null  float64
 15  syn_count        238687 non-null  float64
 16  fin_count        238687 non-null  float64
 17  urg_count        238687 non-null  float64
 18  rst_count        238687 non-null  float64
 19  HTTP             238687 non-null  float64
 20  HTTPS            238687 non-null  float64
 21  DNS              238687 non-null  float64
 22  Telnet           238687 non-null  float64
 23  SMTP             238687 non-null  float64
 24  SSH              238687 non-null  float64
 25  IRC              238687 non-null  float64
 26  TCP              238687 non-null  float64
 27  UDP              238687 non-null  float64
 28  DHCP             238687 non-null  float64
 29  ARP              238687 non-null  float64
 30  ICMP             238687 non-null  float64
 31  IPv              238687 non-null  float64
 32  LLC              238687 non-null  float64
 33  Tot sum          238687 non-null  float64
 34  Min              238687 non-null  float64
 35  Max              238687 non-null  float64
 36  AVG              238687 non-null  float64
 37  Std              238687 non-null  float64
 38  Tot size         238687 non-null  float64
 39  IAT              238687 non-null  float64
 40  Number           238687 non-null  float64
 41  Magnitue         238687 non-null  float64
 42  Radius           238687 non-null  float64
 43  Covariance       238687 non-null  float64
 44  Variance         238687 non-null  float64
```

```
 45  Weight            238687 non-null  float64
 46  label             238687 non-null  object
dtypes: float64(46), object(1)
memory usage: 85.6+ MB
```

[98]:
```
# lets remove columns that we dont need.
df.columns
```

[98]:
```
Index(['flow_duration', 'Header_Length', 'Protocol Type', 'Duration', 'Rate',
       'Srate', 'Drate', 'fin_flag_number', 'syn_flag_number',
       'rst_flag_number', 'psh_flag_number', 'ack_flag_number',
       'ece_flag_number', 'cwr_flag_number', 'ack_count', 'syn_count',
       'fin_count', 'urg_count', 'rst_count', 'HTTP', 'HTTPS', 'DNS', 'Telnet',
       'SMTP', 'SSH', 'IRC', 'TCP', 'UDP', 'DHCP', 'ARP', 'ICMP', 'IPv', 'LLC',
       'Tot sum', 'Min', 'Max', 'AVG', 'Std', 'Tot size', 'IAT', 'Number',
       'Magnitue', 'Radius', 'Covariance', 'Variance', 'Weight', 'label'],
      dtype='object')
```

[ ]:
```
```

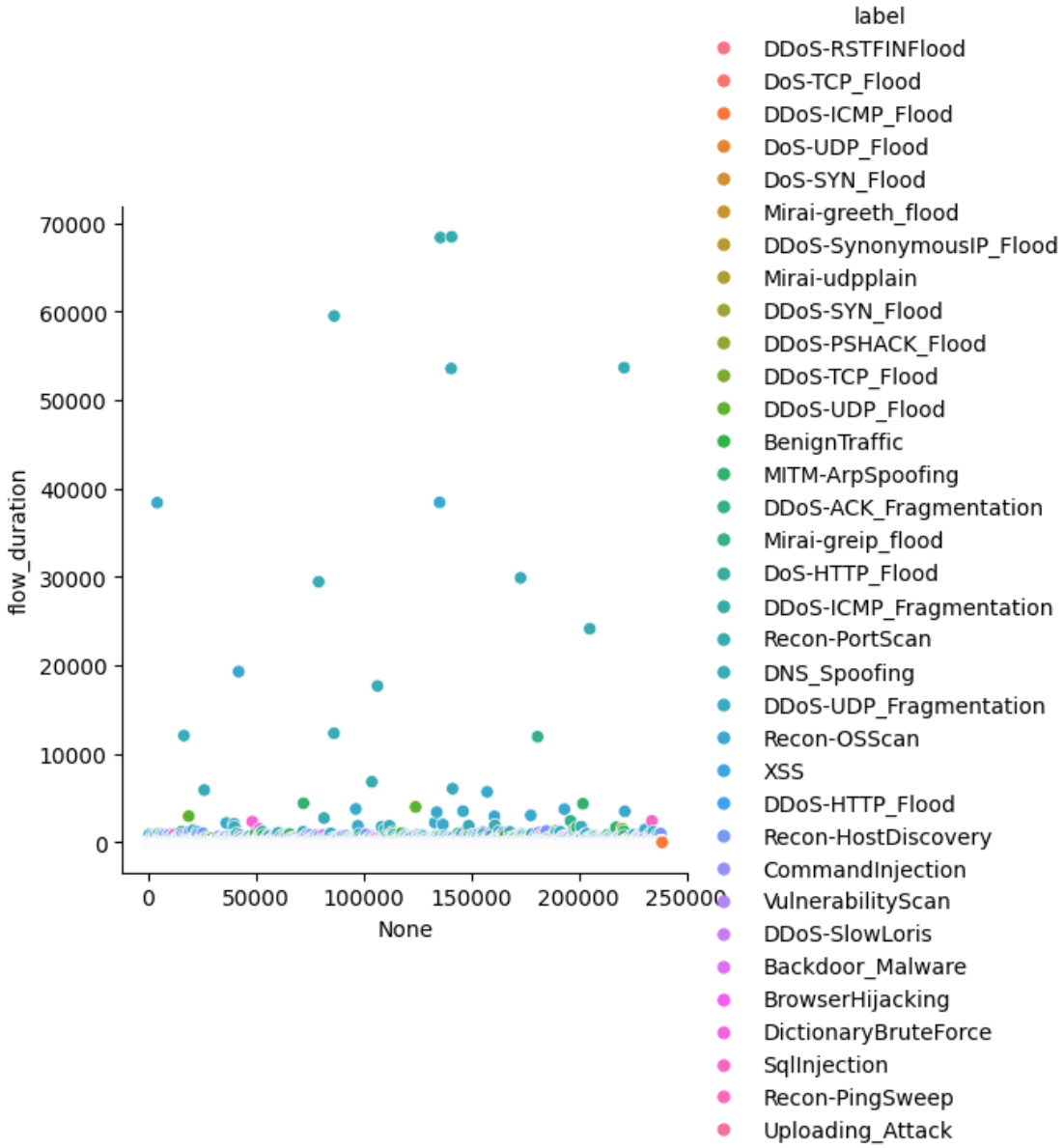
[99]:
```
df.drop(
    columns=[
        "Drate",
        "fin_flag_number",
        "syn_flag_number",
        "rst_flag_number",
        "psh_flag_number",
        "ack_flag_number",
        "ece_flag_number",
        "cwr_flag_number",
        "ack_count",
        "syn_count",
        "fin_count",
        "urg_count",
        "rst_count",
        "HTTP",
        "HTTPS",
        "DNS",
        "Telnet",
        "SMTP",
        "SSH",
        "IRC",
        "TCP",
        "UDP",
        "DHCP",
        "ARP",
        "ICMP",
        "IPv",
```

```
        "LLC",
        "Tot sum",
        "Min",
        "Max",
        "AVG",
        "Std",
        "Tot size",
        "IAT",
        "Number",
        "Magnitue",
        "Radius",
        "Covariance",
        "Variance",
        "Weight",
    ],
    inplace=True,
)
```

[100]:
```
# let us now visualize scatter plots of the data
import seaborn as sns

# before that lets scale the data down, just take the first 1000 rows
scaled_df = df[:1000]
```

[101]:
```
sns.relplot(df, y="flow_duration", x=df.index, kind="scatter", hue="label")
```

[101]:
```
<seaborn.axisgrid.FacetGrid at 0x2d6006d7520>
```



[102]:
```
# lets go through the dataset, and remove those rows with labels that occur very␣
↪less times to remove noise

df["label"].value_counts()
```

[102]:
```
label
DDoS-ICMP_Flood            36554
DDoS-UDP_Flood             27626
DDoS-TCP_Flood             23149
DDoS-PSHACK_Flood          21210
```

```
DDoS-SYN_Flood             20739
DDoS-RSTFINFlood           20669
DDoS-SynonymousIP_Flood    18189
DoS-UDP_Flood              16957
DoS-TCP_Flood              13630
DoS-SYN_Flood              10275
BenignTraffic               5600
Mirai-greeth_flood          5016
Mirai-udpplain              4661
Mirai-greip_flood           3758
DDoS-ICMP_Fragmentation     2377
MITM-ArpSpoofing            1614
DDoS-ACK_Fragmentation      1505
DDoS-UDP_Fragmentation      1484
DNS_Spoofing                 925
Recon-HostDiscovery          697
Recon-OSScan                 517
Recon-PortScan               430
DoS-HTTP_Flood               414
VulnerabilityScan            210
DDoS-HTTP_Flood              169
DDoS-SlowLoris               106
DictionaryBruteForce          63
SqlInjection                  31
BrowserHijacking              30
CommandInjection              28
Backdoor_Malware              22
XSS                           18
Uploading_Attack               8
Recon-PingSweep                6
Name: count, dtype: int64
```

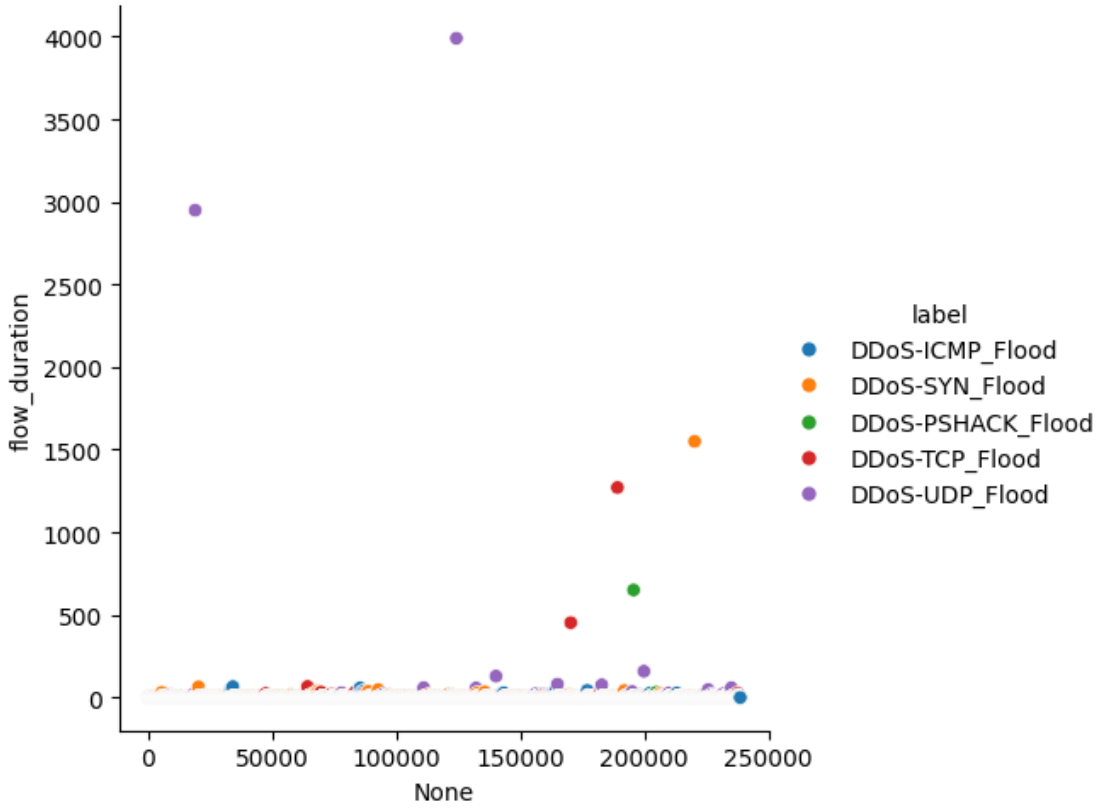
[103]:
```
# lets keep the top 5 labels and remove the rest
top_labels = df["label"].value_counts().head(5).index
df = df[df["label"].isin(top_labels)]
```

[104]:
```
df.shape
```

[104]:
```
(129278, 7)
```

[105]:
```
# lets try plotting again
sns.relplot(df, y="flow_duration", x=df.index, kind="scatter", hue="label")
```

[105]:
```
<seaborn.axisgrid.FacetGrid at 0x2d651613af0>
```

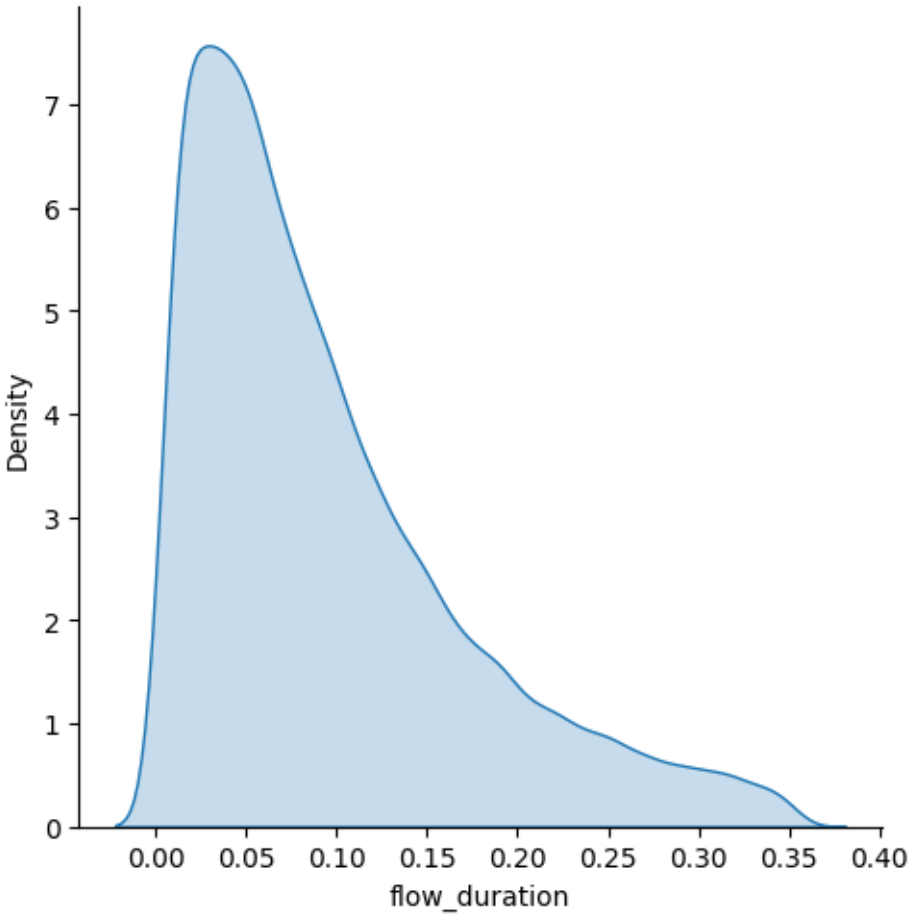


[127]:
```
# lets remove flow_duration rows with z scores more than 3
from scipy import stats

z = np.abs(stats.zscore(df["flow_duration"]))
non_outlier_df = df[(z < 0.0115)]
non_outlier_df.shape
```

[127]: (34489, 7)

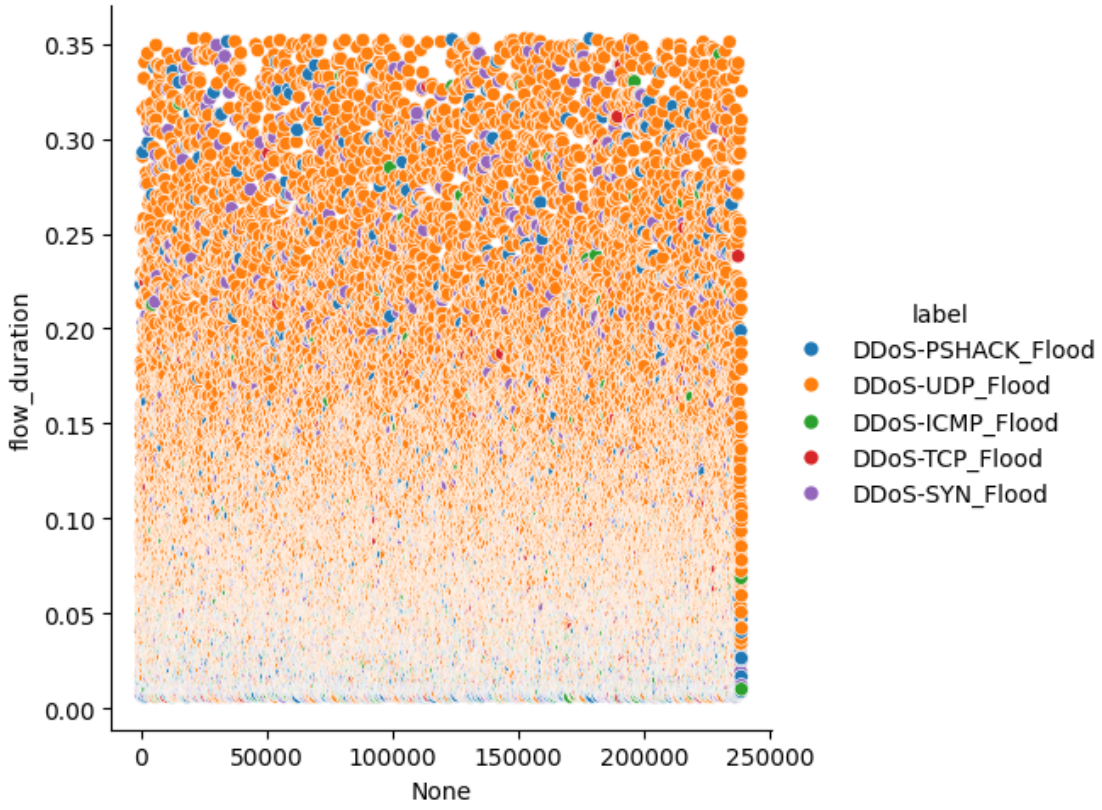
[128]:
```
# lets plot a distribution
sns.displot(non_outlier_df["flow_duration"], kind="kde", fill=True)
```

[128]: <seaborn.axisgrid.FacetGrid at 0x2d62991a350>



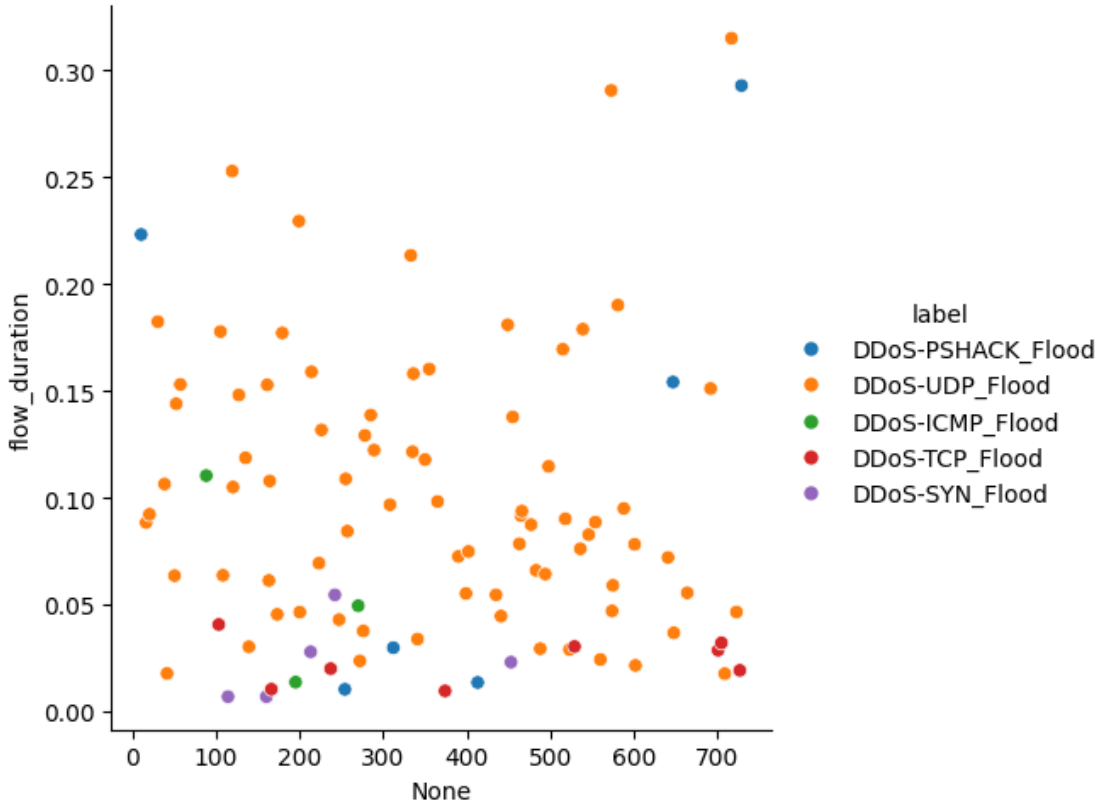
[129]:
```
# lets try plotting again
sns.relplot(
    non_outlier_df,
    y="flow_duration",
    x=non_outlier_df.index,
    kind="scatter",
    hue="label",
)
```

[129]: <seaborn.axisgrid.FacetGrid at 0x2d644e29030>



[131]:
```
# lets now try with only 1000 rows
scaled_df = non_outlier_df[:100]

# lets plot
sns.relplot(
    scaled_df, y="flow_duration", x=scaled_df.index, kind="scatter", hue="label"
)
```

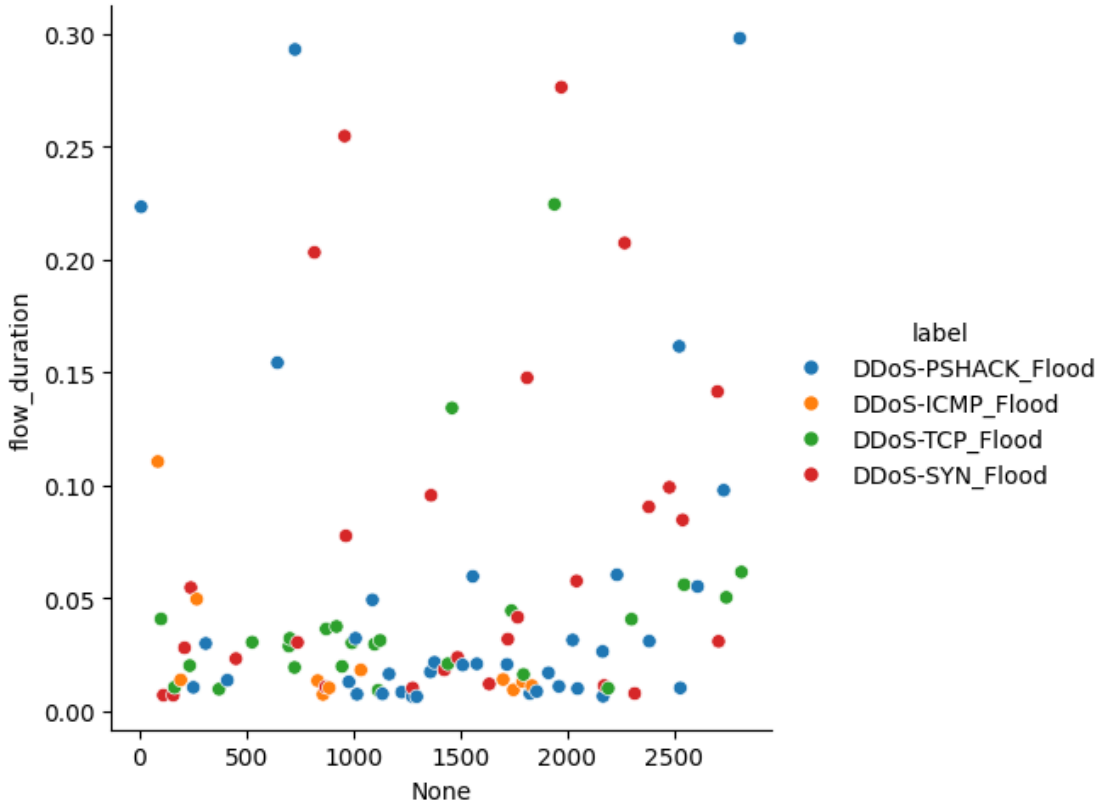
[131]: <seaborn.axisgrid.FacetGrid at 0x2d64f866d40>



[135]:
```
# we still cant see a nice cluster here, lets try and remove ddos udp flood
scaled_df = non_outlier_df[non_outlier_df["label"] != "DDoS-UDP_Flood"]
```

[138]:
```
# lets plot again
scaled_df = scaled_df[:100]
sns.relplot(
    scaled_df, y="flow_duration", x=scaled_df.index, kind="scatter", hue="label"
)
```

[138]:
```
<seaborn.axisgrid.FacetGrid at 0x2d651567850>
```



[139]:
```
# lets try other features
df.head()
```

[139]:

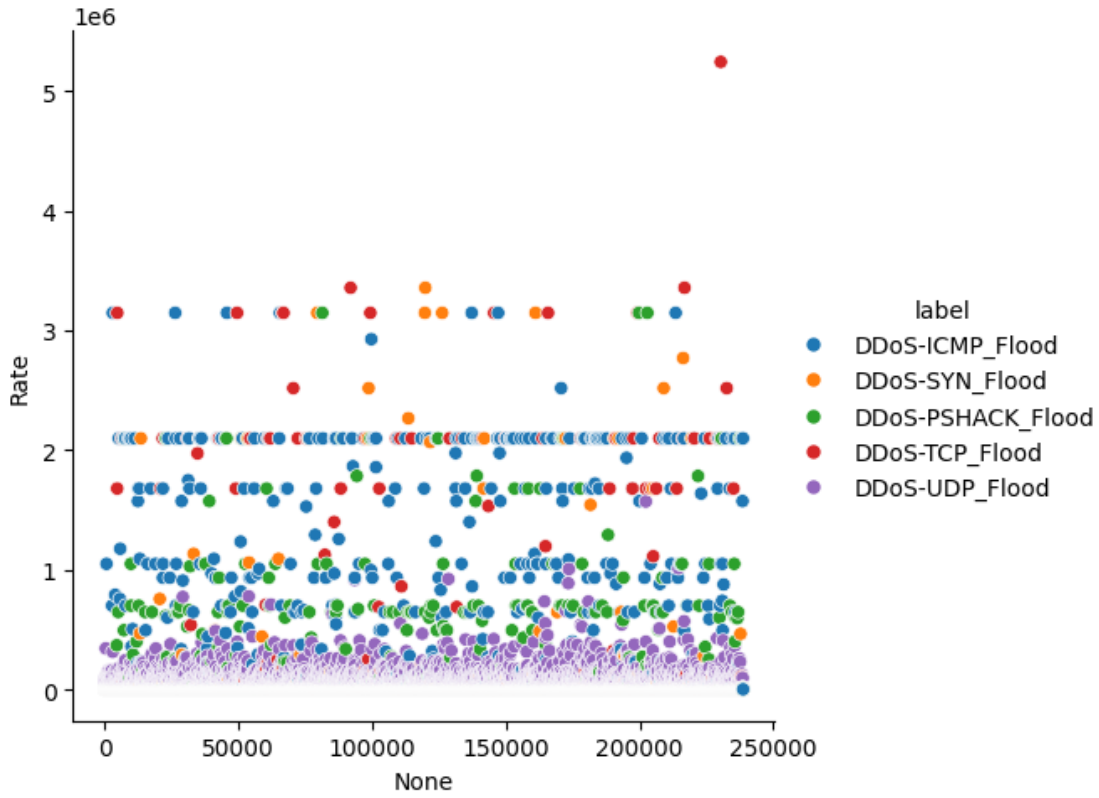
	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	label
2	0.000000	0.00	1.00	64.00	33.396799	33.396799	DDoS-ICMP_Flood
9	0.000000	54.20	6.00	64.00	11.243547	11.243547	DDoS-SYN_Flood
10	0.223192	61.54	6.11	64.64	9.087882	9.087882	DDoS-PSHACK_Flood
11	0.000000	54.00	6.00	64.00	17.333181	17.333181	DDoS-TCP_Flood
12	0.000000	0.00	1.00	75.46	0.000000	0.000000	DDoS-ICMP_Flood

[141]:
```
# lets plot rate just like flow_duration
df.shape
```

[141]: (129278, 7)

[143]:
```
sns.relplot(df, y="Rate", x=df.index, kind="scatter", hue="label")
```

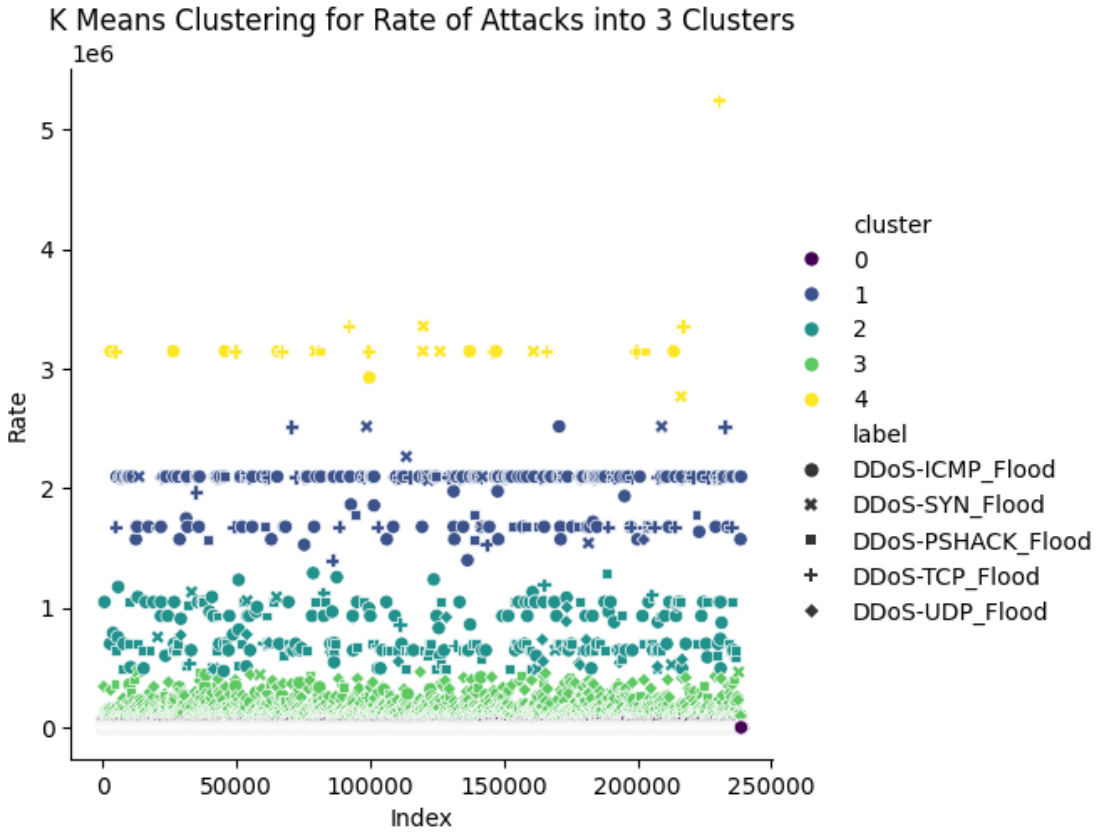
[143]:
```
<seaborn.axisgrid.FacetGrid at 0x2d61a8e65c0>
```



[154]:
```
# its not great but there are some clusters, we can try here.
# lets try implementing k means

# lets try with 5 clusters
kmeans = KMeans(n_clusters=5)
kmeans.fit(df[["Rate"]])

# lets get the labels
df["cluster"] = kmeans.labels_

# lets plot
sns.relplot(
    df,
    y="Rate",
    x=df.index,
    kind="scatter",
    hue="cluster",
```

```
    palette="viridis",
    legend="full",
    style="label",
)

# titles
plt.title("K Means Clustering for Rate of Attacks into 3 Clusters")
plt.xlabel("Index")
plt.ylabel("Rate")
```

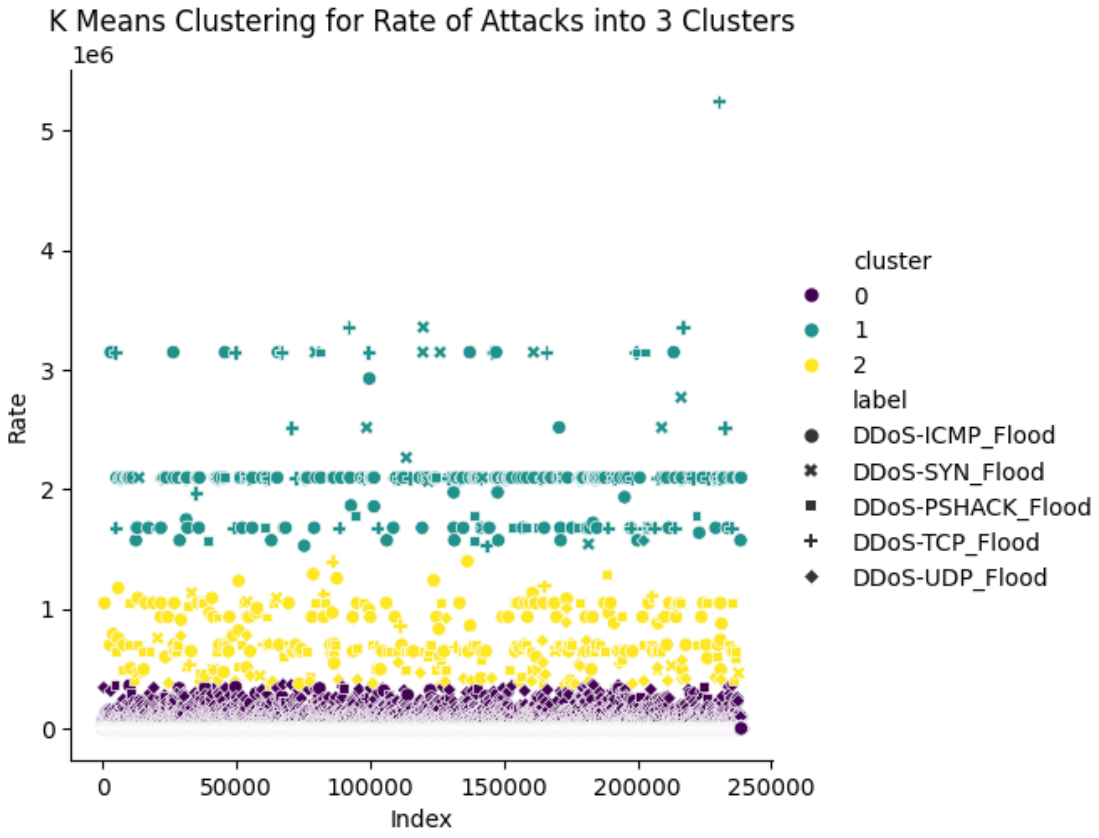
[154]: `Text(49.60998958333333, 0.5, 'Rate')`



[153]:
```
# lets try with 3 clusters
kmeans = KMeans(n_clusters=3)
kmeans.fit(df[["Rate"]])

# lets get the labels
df["cluster"] = kmeans.labels_

# lets plot
```

```
sns.relplot(
    df,
    y="Rate",
    x=df.index,
    kind="scatter",
    hue="cluster",
    palette="viridis",
    legend="full",
    style="label",
)

# titles
plt.title("K Means Clustering for Rate of Attacks into 3 Clusters")
plt.xlabel("Index")
plt.ylabel("Rate")
```

[153]: Text(49.60998958333333, 0.5, 'Rate')

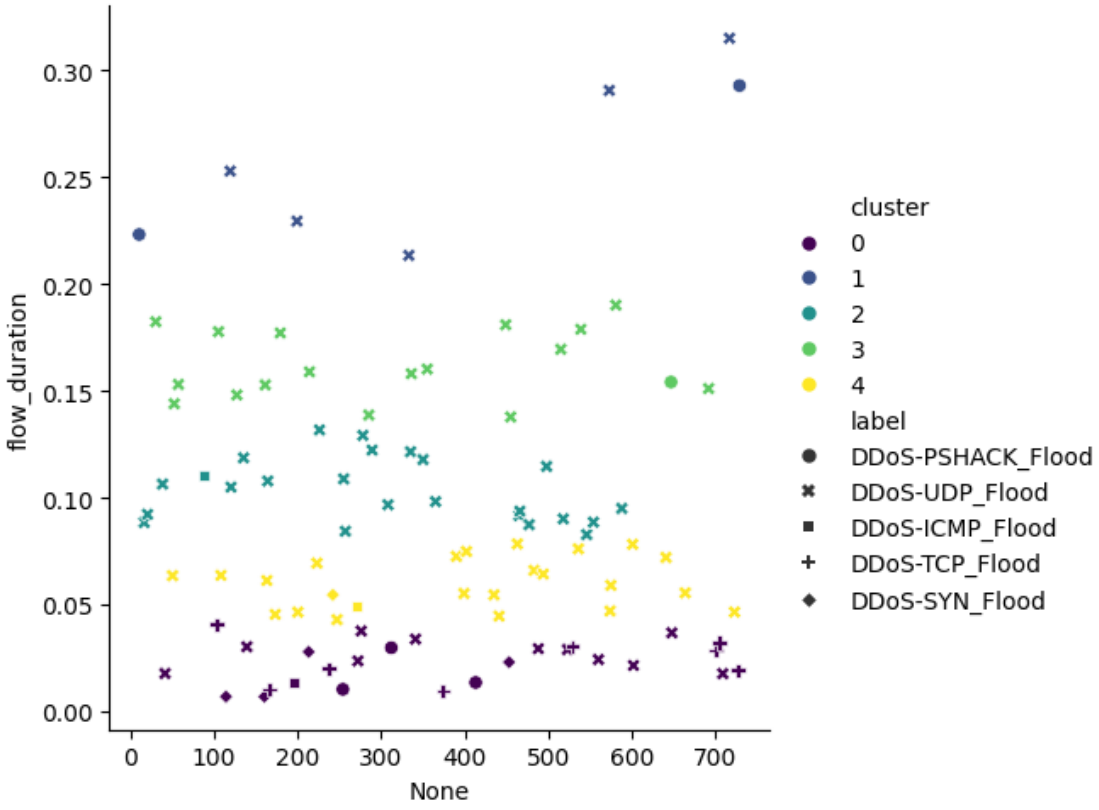


[156]:
```
scaled_df = non_outlier_df[:100]
```

```
# we could also try it on flow_duration
kmeans = KMeans(n_clusters=5)
kmeans.fit(scaled_df[["flow_duration"]])

# lets get the labels
scaled_df["cluster"] = kmeans.labels_

# lets plot
sns.relplot(
    scaled_df,
    y="flow_duration",
    x=scaled_df.index,
    kind="scatter",
    hue="cluster",
    palette="viridis",
    legend="full",
    style="label",
)
```

```
C:\Users\Krishnaraj\AppData\Local\Temp\ipykernel_43912\3087714870.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  scaled_df["cluster"] = kmeans.labels_
```

[156]: <seaborn.axisgrid.FacetGrid at 0x2d6821e67d0>



[157]:
```
# so that concludes our k means clustering analysis on features of an IOT dataset.
```

[159]:
```
# credits to https://www.unb.ca/cic/datasets/iotdataset-2022.html for the dataset
# Citation: Sajjad Dadkhah, Hassan Mahdikhani, Priscilla Kyei Danso, Alireza Zohourian, Kevin Anh Truong, Ali A. Ghorbani, "Towards the development of a realistic multidimensional IoT profiling dataset", Submitted to: The 19th Annual International Conference on Privacy, Security & Trust (PST2022) August 22-24, 2022, Fredericton, Canada.

# analysis by: Krishnaraj T
```

[ ]:
```
```

[95]:
```
        # implementing k means clustering on iot based attacks dataset
        # importing required libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.preprocessing import StandardScaler
```

[96]:
```
# importing the dataset
df = pd.read_csv("part-00000-363d1ba3-8ab5-4f96-bc25-4d5862db7cb9-c000.
↪csv")
df.head()
```

[96]:

	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	Drate	fin_flag_number	syn_flag_number	rst_flag_number	...	Std	Tot size	IAT	Number	Magnitue	Radius	Covariance	Variance	Weight	label
0	0.000000	54.00	6.00	64.00	0.329807	0.329807	0.0	1.0	0.0	1.0	...	0.000000	54.00	8.334383e+07	9.5	10.392305	0.000000	0.000000	0.00	141.55	DDoS-RSTFINFlood
1	0.000000	57.04	6.33	64.00	4.290556	4.290556	0.0	0.0	0.0	0.0	...	2.822973	57.04	8.292607e+07	9.5	10.464666	4.010353	160.987842	0.05	141.55	DoS-TCP_Flood
2	0.000000	0.00	1.00	64.00	33.396799	33.396799	0.0	0.0	0.0	0.0	...	0.000000	42.00	8.312799e+07	9.5	9.165151	0.000000	0.000000	0.00	141.55	DDoS-ICMP_Flood
3	0.328175	76175.00	17.00	64.00	4642.133010	4642.133010	0.0	0.0	0.0	0.0	...	0.000000	50.00	8.301570e+07	9.5	10.000000	0.000000	0.000000	0.00	141.55	DoS-UDP_Flood
4	0.117320	101.73	6.11	65.91	6.202211	6.202211	0.0	0.0	1.0	0.0	...	23.113111	57.88	8.297300e+07	9.5	11.346876	32.716243	3016.808286	0.19	141.55	DoS-SYN_Flood

[5 rows x 47 columns]

[97]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 238687 entries, 0 to 238686
Data columns (total 47 columns):
 #   Column           Non-Null Count   Dtype
```

```
---  ------           --------------   -----
 0   flow_duration    238687 non-null  float64
 1   Header_Length    238687 non-null  float64
 2   Protocol Type    238687 non-null  float64
 3   Duration         238687 non-null  float64
 4   Rate             238687 non-null  float64
 5   Srate            238687 non-null  float64
 6   Drate            238687 non-null  float64
 7   fin_flag_number  238687 non-null  float64
 8   syn_flag_number  238687 non-null  float64
 9   rst_flag_number  238687 non-null  float64
 10  psh_flag_number  238687 non-null  float64
 11  ack_flag_number  238687 non-null  float64
 12  ece_flag_number  238687 non-null  float64
 13  cwr_flag_number  238687 non-null  float64
 14  ack_count        238687 non-null  float64
 15  syn_count        238687 non-null  float64
 16  fin_count        238687 non-null  float64
 17  urg_count        238687 non-null  float64
 18  rst_count        238687 non-null  float64
 19  HTTP             238687 non-null  float64
 20  HTTPS            238687 non-null  float64
 21  DNS              238687 non-null  float64
 22  Telnet           238687 non-null  float64
 23  SMTP             238687 non-null  float64
 24  SSH              238687 non-null  float64
 25  IRC              238687 non-null  float64
 26  TCP              238687 non-null  float64
 27  UDP              238687 non-null  float64
 28  DHCP             238687 non-null  float64
 29  ARP              238687 non-null  float64
 30  ICMP             238687 non-null  float64
 31  IPv              238687 non-null  float64
 32  LLC              238687 non-null  float64
 33  Tot sum          238687 non-null  float64
 34  Min              238687 non-null  float64
 35  Max              238687 non-null  float64
 36  AVG              238687 non-null  float64
 37  Std              238687 non-null  float64
 38  Tot size         238687 non-null  float64
 39  IAT              238687 non-null  float64
 40  Number           238687 non-null  float64
 41  Magnitue         238687 non-null  float64
 42  Radius           238687 non-null  float64
 43  Covariance       238687 non-null  float64
 44  Variance         238687 non-null  float64
 45  Weight           238687 non-null  float64
 46  label            238687 non-null  object
```

```
dtypes: float64(46), object(1)
memory usage: 85.6+ MB
```

[98]:
```
# lets remove columns that we dont need.
df.columns
```

[98]:
```
Index(['flow_duration', 'Header_Length', 'Protocol Type', 'Duration',␣
↪'Rate',
       'Srate', 'Drate', 'fin_flag_number', 'syn_flag_number',
       'rst_flag_number', 'psh_flag_number', 'ack_flag_number',
       'ece_flag_number', 'cwr_flag_number', 'ack_count', 'syn_count',
       'fin_count', 'urg_count', 'rst_count', 'HTTP', 'HTTPS', 'DNS',␣
↪'Telnet',
       'SMTP', 'SSH', 'IRC', 'TCP', 'UDP', 'DHCP', 'ARP', 'ICMP', 'IPv',␣
↪'LLC',
       'Tot sum', 'Min', 'Max', 'AVG', 'Std', 'Tot size', 'IAT',␣
↪'Number',
       'Magnitue', 'Radius', 'Covariance', 'Variance', 'Weight',␣
↪'label'],
      dtype='object')
```

[ ]:
```
```

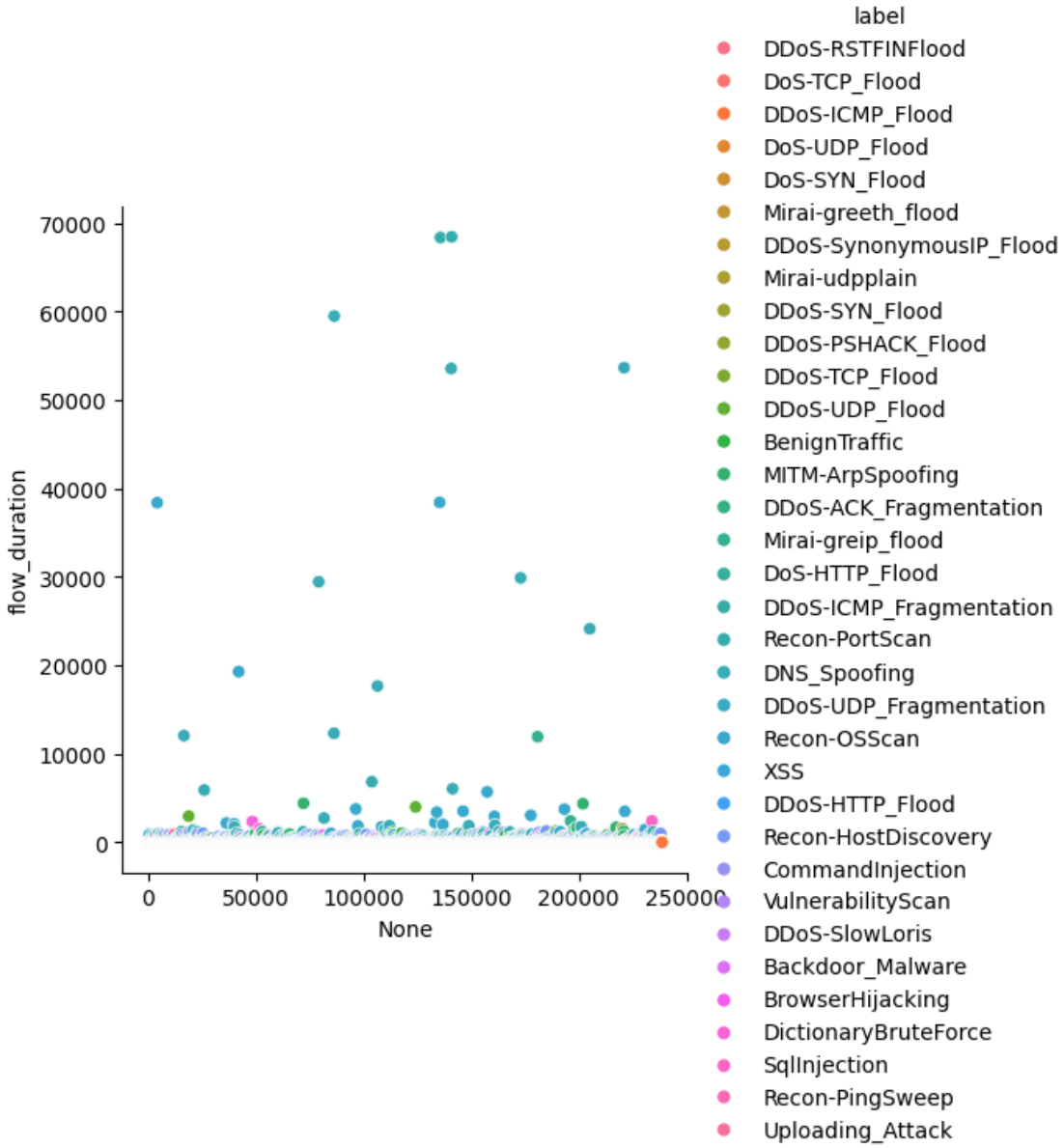
[99]:
```
df.drop(
    columns=[
        "Drate",
        "fin_flag_number",
        "syn_flag_number",
        "rst_flag_number",
        "psh_flag_number",
        "ack_flag_number",
        "ece_flag_number",
        "cwr_flag_number",
        "ack_count",
        "syn_count",
        "fin_count",
        "urg_count",
        "rst_count",
        "HTTP",
        "HTTPS",
        "DNS",
        "Telnet",
        "SMTP",
        "SSH",
        "IRC",
        "TCP",
        "UDP",
```

```
                "DHCP",
                "ARP",
                "ICMP",
                "IPv",
                "LLC",
                "Tot sum",
                "Min",
                "Max",
                "AVG",
                "Std",
                "Tot size",
                "IAT",
                "Number",
                "Magnitue",
                "Radius",
                "Covariance",
                "Variance",
                "Weight",
            ],
            inplace=True,
        )
```

[100]:
```
        # let us now visualize scatter plots of the data
        import seaborn as sns

        # before that lets scale the data down, just take the first 1000 rows
        scaled_df = df[:1000]
```

[101]:
```
        sns.relplot(df, y="flow_duration", x=df.index, kind="scatter",␣
↪hue="label")
```

[101]:
```
        <seaborn.axisgrid.FacetGrid at 0x2d6006d7520>
```



[102]:
```
        # lets go through the dataset, and remove those rows with labels that␣
↪occur very less times to remove noise

        df["label"].value_counts()
```

[102]:
```
        label
        DDoS-ICMP_Flood            36554
        DDoS-UDP_Flood             27626
        DDoS-TCP_Flood             23149
        DDoS-PSHACK_Flood          21210
```

```
DDoS-SYN_Flood             20739
DDoS-RSTFINFlood           20669
DDoS-SynonymousIP_Flood    18189
DoS-UDP_Flood              16957
DoS-TCP_Flood              13630
DoS-SYN_Flood              10275
BenignTraffic               5600
Mirai-greeth_flood          5016
Mirai-udpplain              4661
Mirai-greip_flood           3758
DDoS-ICMP_Fragmentation     2377
MITM-ArpSpoofing            1614
DDoS-ACK_Fragmentation      1505
DDoS-UDP_Fragmentation      1484
DNS_Spoofing                 925
Recon-HostDiscovery          697
Recon-OSScan                 517
Recon-PortScan               430
DoS-HTTP_Flood               414
VulnerabilityScan            210
DDoS-HTTP_Flood              169
DDoS-SlowLoris               106
DictionaryBruteForce          63
SqlInjection                  31
BrowserHijacking              30
CommandInjection              28
Backdoor_Malware              22
XSS                           18
Uploading_Attack               8
Recon-PingSweep                6
Name: count, dtype: int64
```

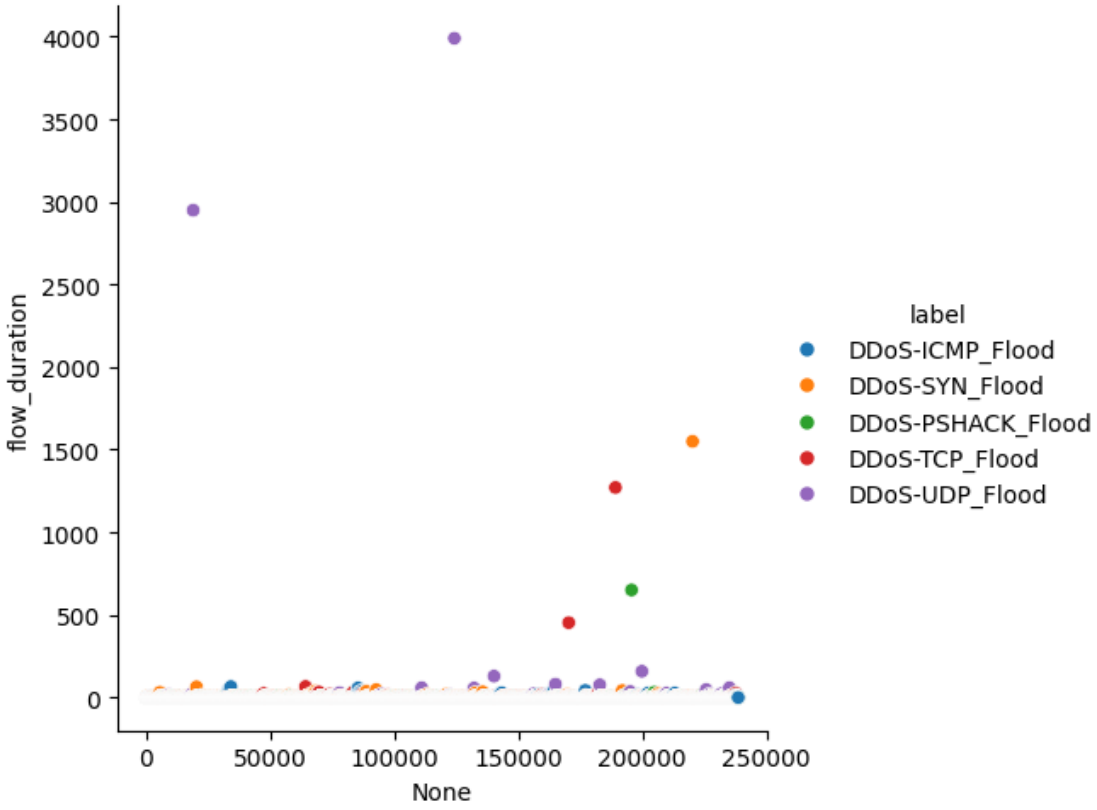
[103]:
```
# lets keep the top 5 labels and remove the rest
top_labels = df["label"].value_counts().head(5).index
df = df[df["label"].isin(top_labels)]
```

[104]:
```
df.shape
```

[104]:
```
(129278, 7)
```

[105]:
```
# lets try plotting again
sns.relplot(df, y="flow_duration", x=df.index, kind="scatter",␣
↪hue="label")
```

[105]:
```
<seaborn.axisgrid.FacetGrid at 0x2d651613af0>
```

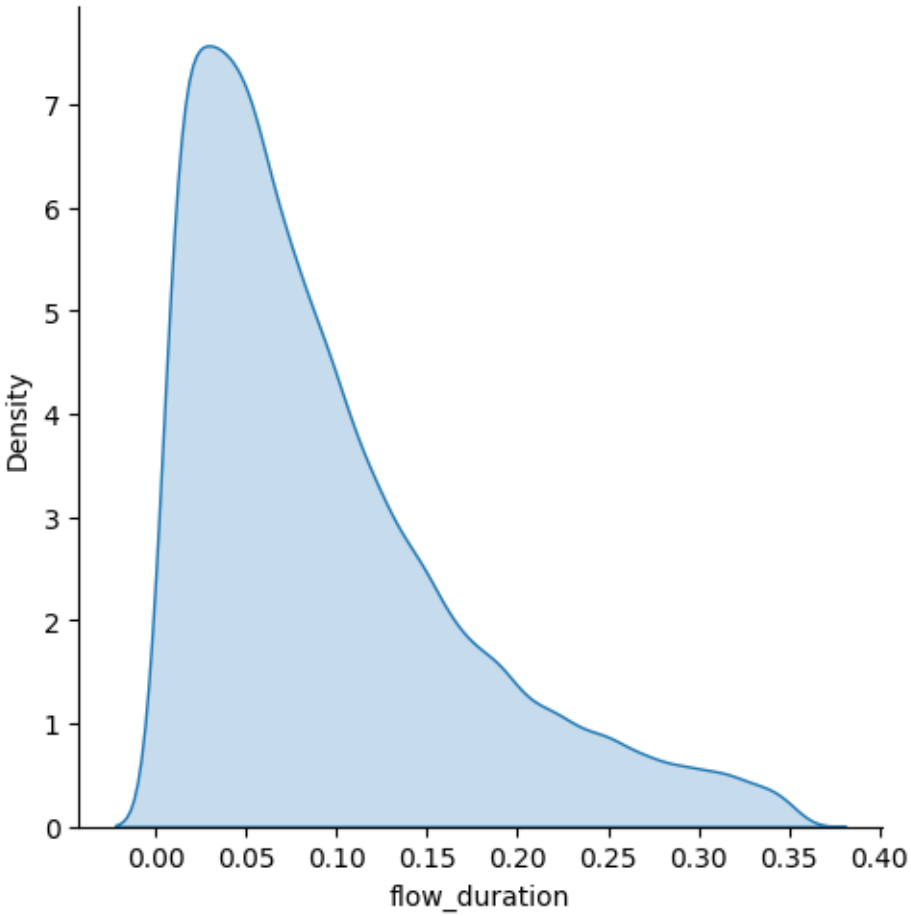


[127]:
```
# lets remove flow_duration rows with z scores more than 3
from scipy import stats

z = np.abs(stats.zscore(df["flow_duration"]))
non_outlier_df = df[(z < 0.0115)]
non_outlier_df.shape
```

[127]:
```
(34489, 7)
```

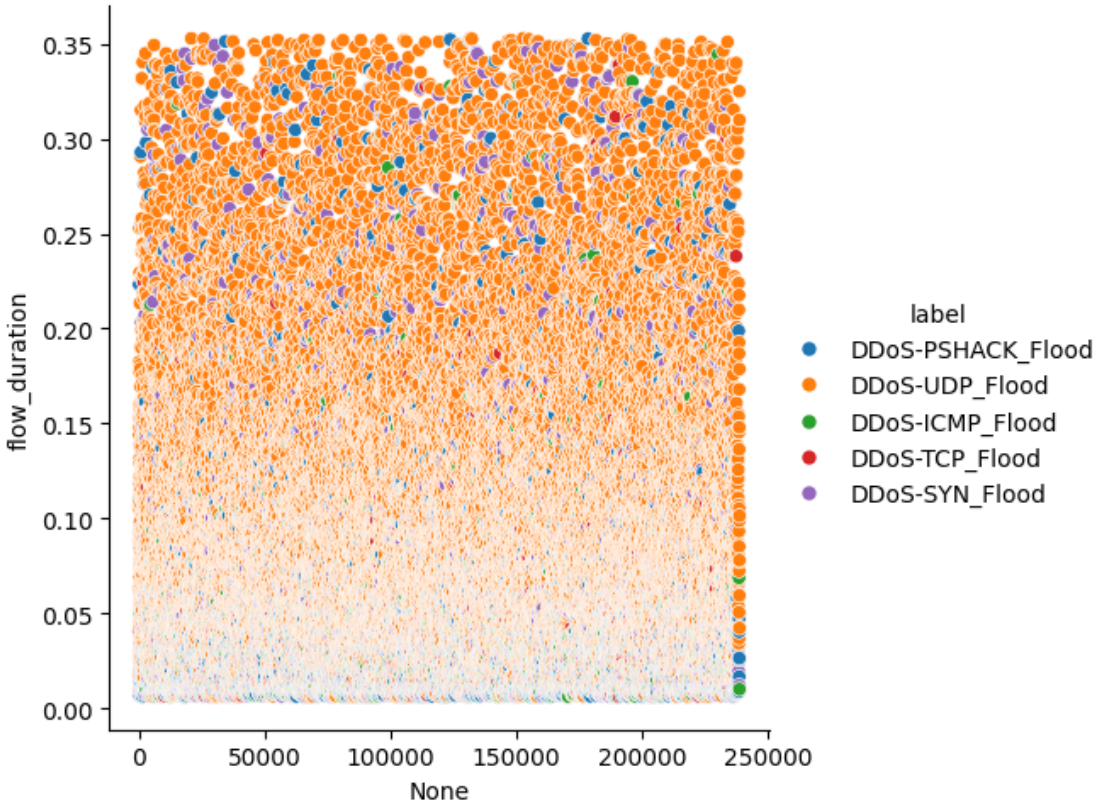
[128]:
```
# lets plot a distribution
sns.displot(non_outlier_df["flow_duration"], kind="kde", fill=True)
```

[128]:
```
<seaborn.axisgrid.FacetGrid at 0x2d62991a350>
```



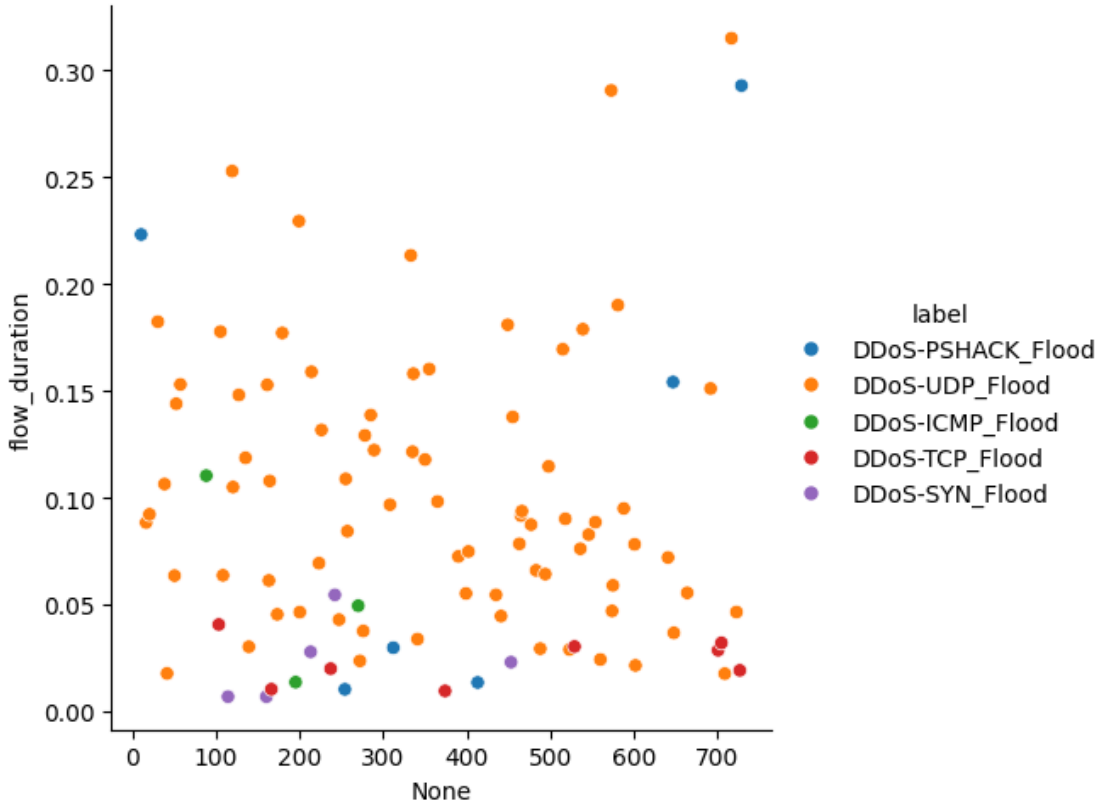
[129]:
```
# lets try plotting again
sns.relplot(
    non_outlier_df,
    y="flow_duration",
    x=non_outlier_df.index,
    kind="scatter",
    hue="label",
)
```

[129]:
```
<seaborn.axisgrid.FacetGrid at 0x2d644e29030>
```



[131]:
```
        # lets now try with only 1000 rows
        scaled_df = non_outlier_df[:100]

        # lets plot
        sns.relplot(
            scaled_df, y="flow_duration", x=scaled_df.index, kind="scatter",␣
↪hue="label"
        )
```

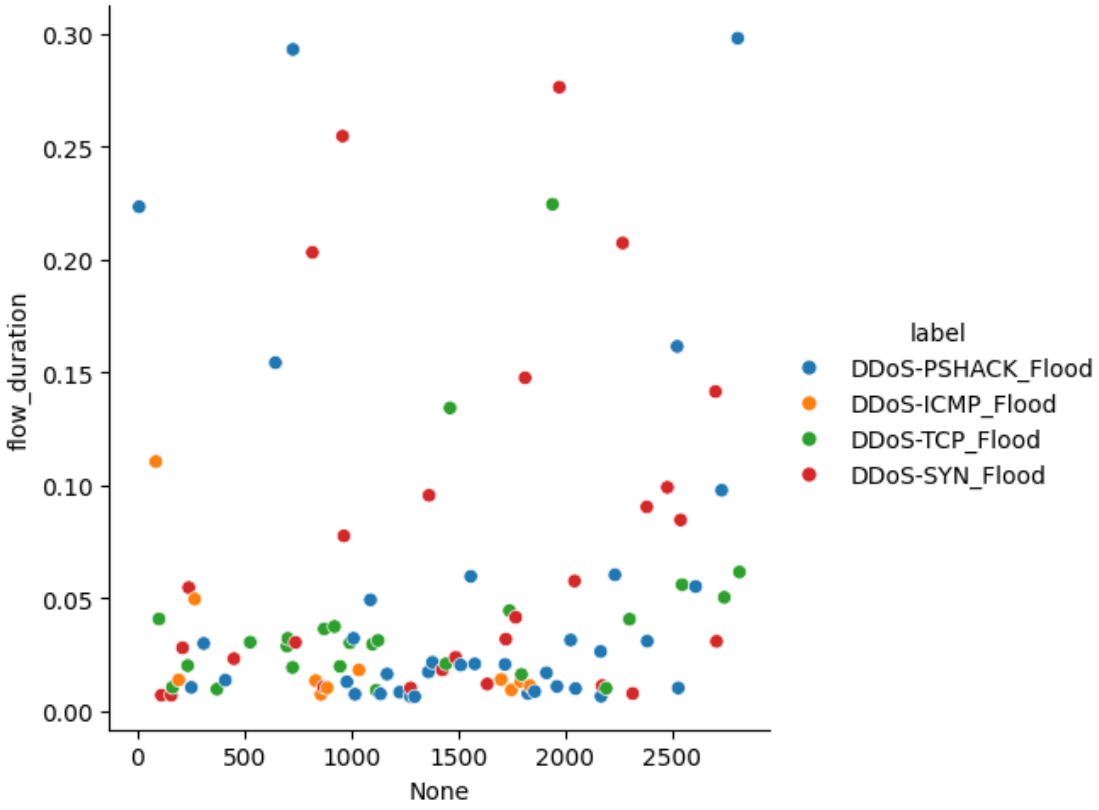
[131]:
```
        <seaborn.axisgrid.FacetGrid at 0x2d64f866d40>
```



[135]:
```
        # we still cant see a nice cluster here, lets try and remove ddos udp␣
↪flood
        scaled_df = non_outlier_df[non_outlier_df["label"] != "DDoS-UDP_Flood"]
```

[138]:
```
        # lets plot again
        scaled_df = scaled_df[:100]
        sns.relplot(
            scaled_df, y="flow_duration", x=scaled_df.index, kind="scatter",␣
↪hue="label"
        )
```

[138]:
```
        <seaborn.axisgrid.FacetGrid at 0x2d651567850>
```



[139]:
```
# lets try other features
df.head()
```

[139]:

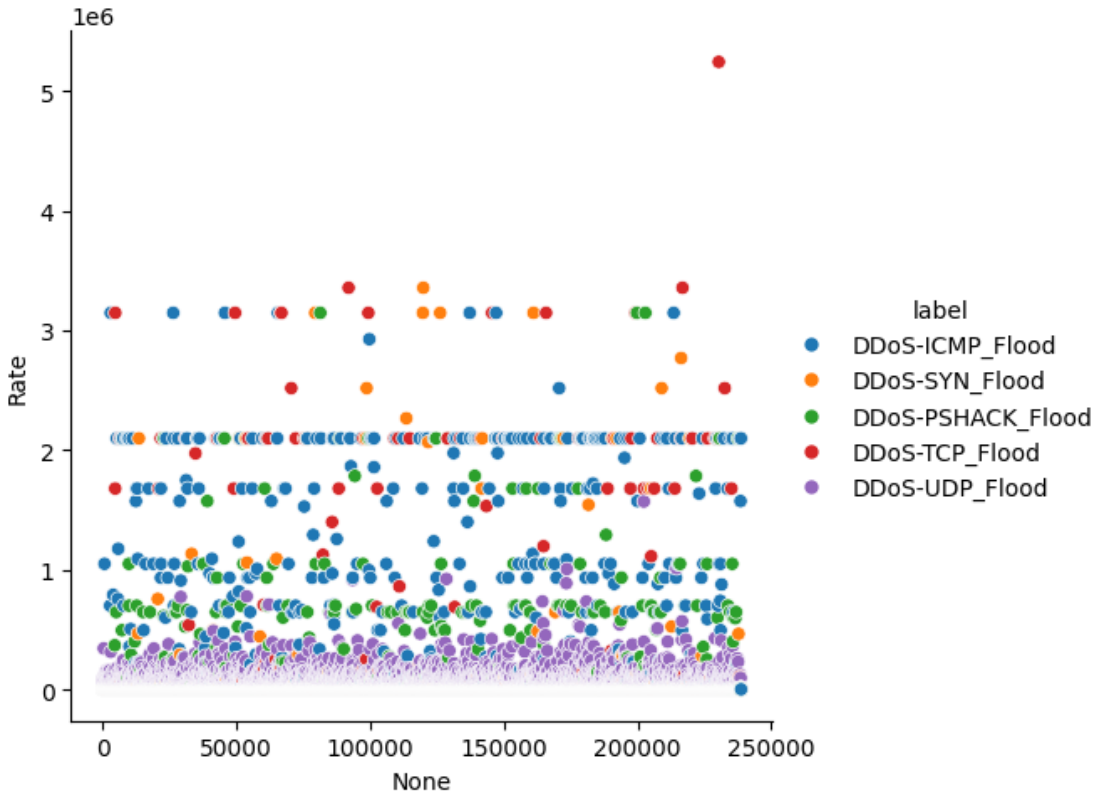
	flow_duration	Header_Length	Protocol Type	Duration	Rate	Srate	label
2	0.000000	0.00	1.00	64.00	33.396799	33.396799	DDoS-ICMP_Flood
9	0.000000	54.20	6.00	64.00	11.243547	11.243547	DDoS-SYN_Flood
10	0.223192	61.54	6.11	64.64	9.087882	9.087882	DDoS-PSHACK_Flood
11	0.000000	54.00	6.00	64.00	17.333181	17.333181	DDoS-TCP_Flood
12	0.000000	0.00	1.00	75.46	0.000000	0.000000	DDoS-ICMP_Flood

[141]:
```
# lets plot rate just like flow_duration
df.shape
```

[141]:
```
(129278, 7)
```

[143]:
```
sns.relplot(df, y="Rate", x=df.index, kind="scatter", hue="label")
```

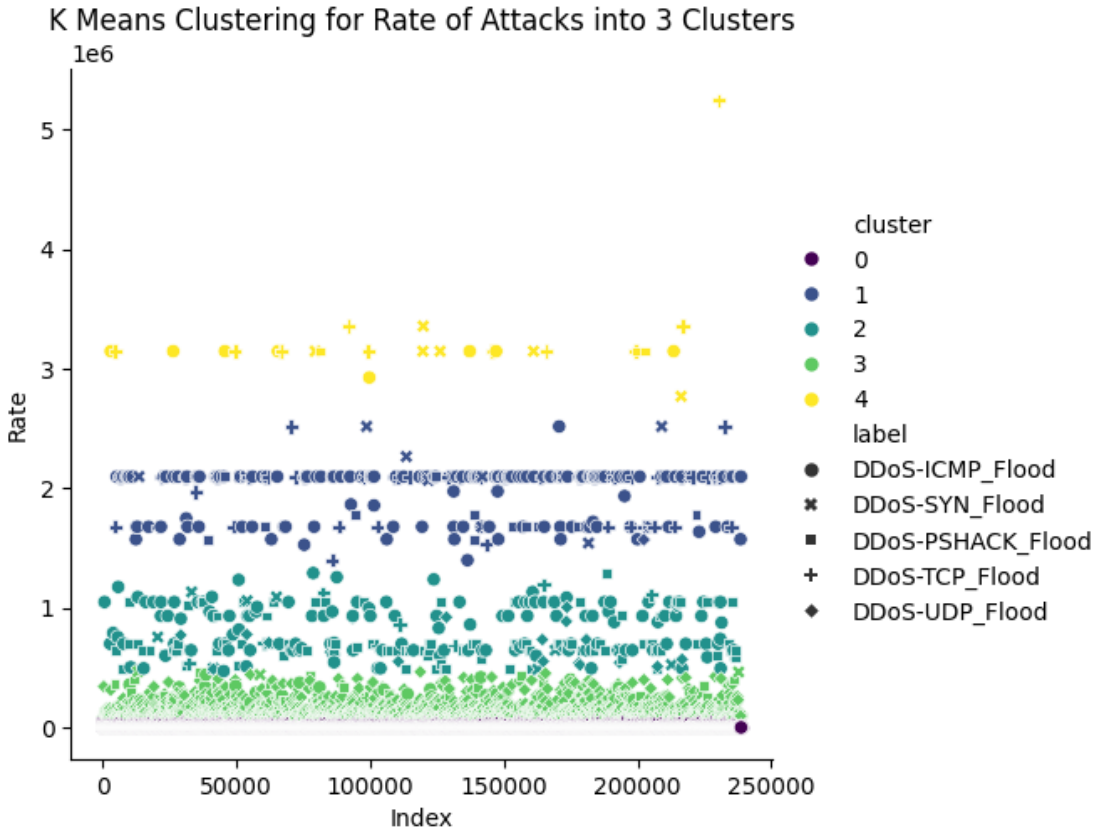
[143]:
```
<seaborn.axisgrid.FacetGrid at 0x2d61a8e65c0>
```



[154]:
```
# its not great but there are some clusters, we can try here.
# lets try implementing k means

# lets try with 5 clusters
kmeans = KMeans(n_clusters=5)
kmeans.fit(df[["Rate"]])

# lets get the labels
df["cluster"] = kmeans.labels_

# lets plot
sns.relplot(
    df,
    y="Rate",
    x=df.index,
    kind="scatter",
    hue="cluster",
```

```
        palette="viridis",
        legend="full",
        style="label",
    )

    # titles
    plt.title("K Means Clustering for Rate of Attacks into 3 Clusters")
    plt.xlabel("Index")
    plt.ylabel("Rate")
```

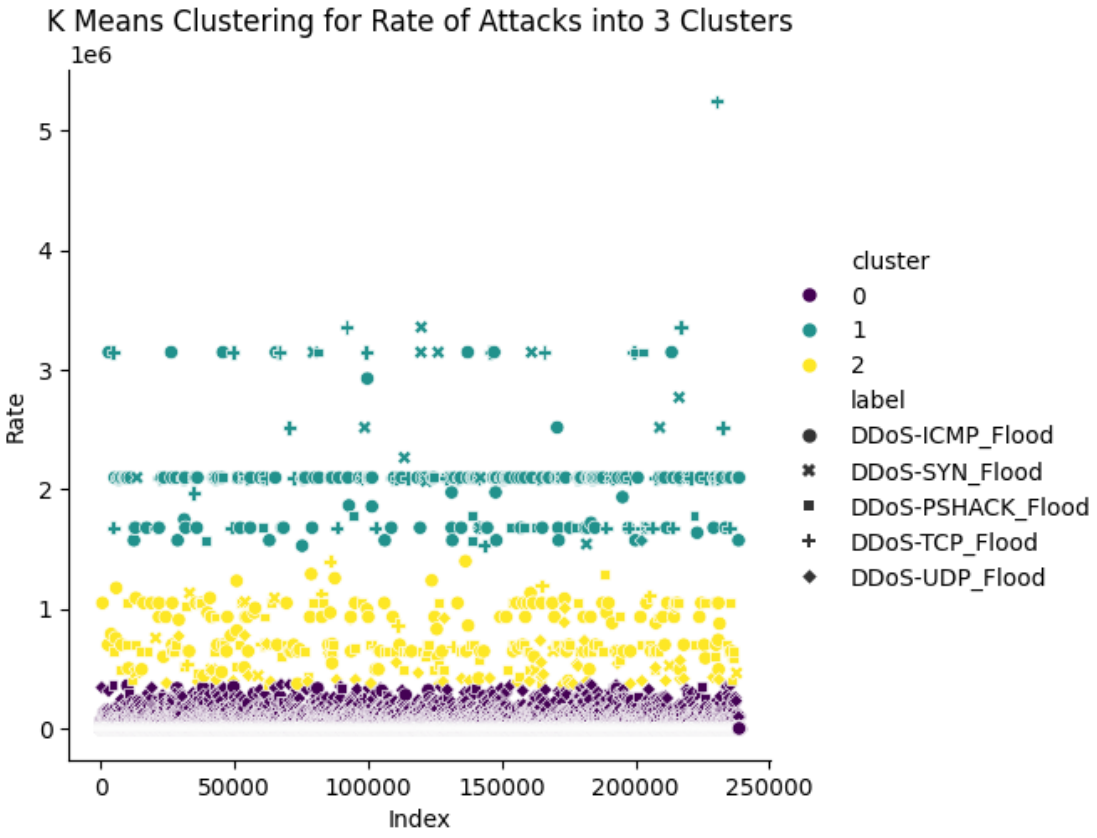
[154]:
```
Text(49.60998958333333, 0.5, 'Rate')
```



[153]:
```
    # lets try with 3 clusters
    kmeans = KMeans(n_clusters=3)
    kmeans.fit(df[["Rate"]])

    # lets get the labels
    df["cluster"] = kmeans.labels_

    # lets plot
```

```
sns.relplot(
    df,
    y="Rate",
    x=df.index,
    kind="scatter",
    hue="cluster",
    palette="viridis",
    legend="full",
    style="label",
)

# titles
plt.title("K Means Clustering for Rate of Attacks into 3 Clusters")
plt.xlabel("Index")
plt.ylabel("Rate")
```

[153]:
```
Text(49.60998958333333, 0.5, 'Rate')
```

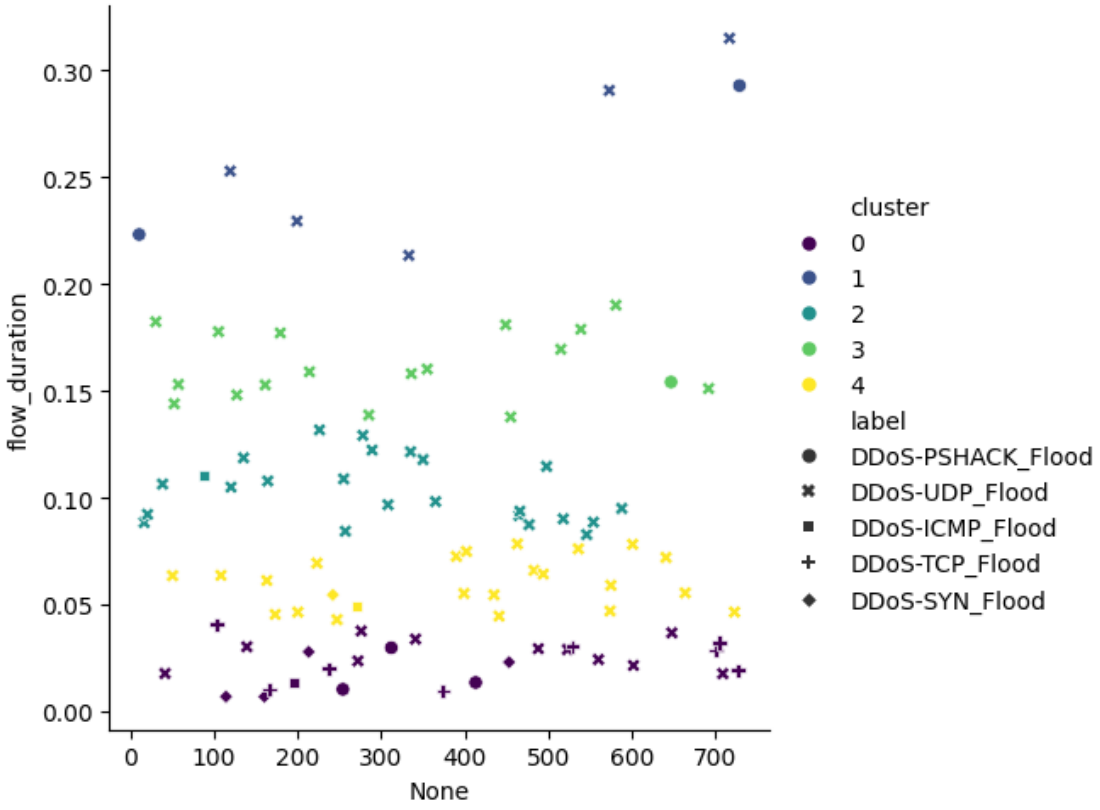


[156]:
```
scaled_df = non_outlier_df[:100]
```

```
# we could also try it on flow_duration
kmeans = KMeans(n_clusters=5)
kmeans.fit(scaled_df[["flow_duration"]])

# lets get the labels
scaled_df["cluster"] = kmeans.labels_

# lets plot
sns.relplot(
    scaled_df,
    y="flow_duration",
    x=scaled_df.index,
    kind="scatter",
    hue="cluster",
    palette="viridis",
    legend="full",
    style="label",
)
```

```
C:\Users\Krishnaraj\AppData\Local\Temp\ipykernel_43912\3087714870.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  scaled_df["cluster"] = kmeans.labels_
```

[156]:
```
<seaborn.axisgrid.FacetGrid at 0x2d6821e67d0>
```



[157]:
```
        # so that concludes our k means clustering analysis on features of an␣
↪IOT dataset.
```

[159]:
```
        # credits to https://www.unb.ca/cic/datasets/iotdataset-2022.html for␣
↪the dataset
        # Citation: Sajjad Dadkhah, Hassan Mahdikhani, Priscilla Kyei Danso,␣
↪Alireza Zohourian, Kevin Anh Truong, Ali A. Ghorbani, "Towards the development␣
↪of a realistic multidimensional IoT profiling dataset", Submitted to: The 19th␣
↪Annual International Conference on Privacy, Security & Trust (PST2022) August␣
↪22-24, 2022, Fredericton, Canada.

        # analysis by: Krishnaraj T
```

[ ]:

8 FAQs

1. **What do you understand by K-Means method?**
    - K-Means is a popular clustering algorithm used in machine learning and data mining.
    - It aims to partition a dataset into K distinct, non-overlapping clusters, where each data point belongs to the cluster with the nearest mean.
    - The algorithm iteratively assigns each data point to the nearest cluster centroid and recalculates the centroids until convergence.
    - K-Means is commonly used for data exploration, pattern recognition, and image segmentation tasks.
2. **Discuss on how can we get data from IoT devices for Cyber Security?**
    - IoT (Internet of Things) devices generate vast amounts of data that can be leveraged for cybersecurity purposes.
    - Data from IoT devices can be obtained through various means, including direct data collection from sensors, network traffic monitoring, and device logs.
    - Security protocols and standards such as MQTT (Message Queuing Telemetry Transport) and HTTPS (Hypertext Transfer Protocol Secure) can be used to securely transmit data from IoT devices to centralized servers or cloud platforms.
    - Data preprocessing techniques, such as data cleaning, normalization, and feature extraction, may be applied to IoT data to prepare it for cybersecurity analysis.
3. **Can K-Means method be used for Anomaly Detection? Explain how?**
    - While K-Means is primarily a clustering algorithm, it can be adapted for anomaly detection in certain scenarios.
    - One approach is to use K-Means to cluster normal data points and identify clusters with fewer data points, which may indicate anomalies.
    - Another approach is to calculate the distance of each data point to its nearest cluster centroid and flag data points with distances above a certain threshold as anomalies.
    - However, K-Means may not be suitable for detecting complex or nonlinear anomalies, and other anomaly detection techniques such as Isolation Forest or One-Class SVM may be more appropriate in such cases.

9 Conclusion

In this Assignment, we implemented the K-Means Clustering algorithm in Python using the IOT Based Attacks dataset. We loaded the dataset, performed data analysis, split the data into dependent and independent variables, and trained the K-Means model. We visualized the clusters and analyzed the results. K-Means clustering is a powerful technique for partitioning data into distinct clusters based on similarity and can be applied to various domains, including cybersecurity and forensics.