Virginia Tech

CS 5764 – Final Term Project

Network Traffic Analysis, PCA, and Interactive Visualization

Author: Minjin Kim

Instructor: Dr. Reza Jafari

Date: December 2025

Deployed Dashboard:

https://dashapp-731148569508.northamerica-northeast1.run.app/

Contents

List of Figures

List of Tables

Abstract

This project analyzes the CIC-IDS2017 Wednesday network traffic dataset using a structured pipeline: dataset exploration, cleaning, outlier detection, PCA-based dimensionality reduction, normality testing, statistical analysis, data transformation, correlation analysis, and interactive visualization via a Dash dashboard.

The dataset turns out to be heavily imbalanced (Benign traffic roughly 69%, DoS Hulk about 29%), and most key flow features are extremely skewed and clearly non-Gaussian. I use a mix of descriptive statistics, kernel density estimates, Pearson correlation, and principal component analysis to understand how different denial-of-service attacks behave compared to benign traffic. The final multi-tab dashboard lets a user choose cleaning methods, outlier removal strategies, transformations, and plot types, and then immediately see how those choices reshape the data and the PCA space. Overall, the work shows that careful preprocessing plus visualization is not optional for this dataset; it is the only practical way to understand the traffic patterns before any serious modeling or intrusion-detection logic is built.

1 Introduction

Network intrusion detection systems (NIDS) rely on traffic-flow features to separate benign behavior from malicious activity. The goal of this Final Term Project is to walk through that entire process on one concrete dataset, using only Python, Plotly, and Dash.

From Plotly's own documentation, the Python Graphing Library is designed for interactive, publication-quality graphs, and Dash provides a production-capable framework for building analytical web applications using pure Python callbacks. Those are exactly the tools I wanted for this project: I am not only plotting static figures, I am building a small analysis app that lets me examine the data from different angles.

The overall workflow matches the FTP instructions and is organized into three phases:

- **Phase II**: Data understanding and preprocessing (cleaning, handling missing/invalid values, outlier detection, transformations), PCA, normality testing, correlation analysis, statistical analysis, and a variety of static plots.
- **Phase III**: A multi-tab Plotly Dash dashboard that exposes the Phase II pipeline as interactive widgets and live-updated plots.
- **Final observations**: For every figure, I write down what I actually learned. The main point is not to draw pretty graphs, but to reveal behavior differences between benign flows and several DoS attacks in the CIC-IDS2017 environment.

The rest of the report follows the structure required by the FTP guidelines, from dataset description through conclusion and appendix.

2 Description of the Dataset

The dataset used in this project is the `Wednesday-workingHours.pcap_ISCX.csv` file from the CIC-IDS2017 benchmark created by the Canadian Institute for Cybersecurity. Each row is a bidirectional flow with more than thirty numerical features extracted from packet captures, plus a categorical `Label` field that identifies the traffic type.

2.1 Dataset Criteria and Reliability

The CIC-IDS2017 Wednesday dataset meets the requirements needed for a reproducible, analysis-focused dashboard. Unlike older intrusion-detection benchmarks that relied on synthetic traffic or random noise generation, this dataset provides realistic behavior across both benign and attack flows.

The benign background is produced using the **B-Profile system (Sharafaldin et al.)**, which models the activity patterns of 25 real users. This generates natural HTTP, HTTPS, FTP, SSH, and email traffic rather than artificial noise, so the baseline distribution reflects actual usage variability. The testbed includes a complete enterprise-style network—**modem, firewall, routers, and switches**—and a heterogeneous set of operating systems (Windows 7/8.1/10/Vista, Ubuntu 12.04/14.04/16.04, and macOS). This variety introduces realistic protocol, port, and timing behaviors in the flows.

All flows are **labeled** based on the scripted attack schedule, which is essential for PCA color-coding, class-wise statistical comparisons, and visualization of how distributions shift under each attack. The Wednesday file also contains multiple DoS variants (Slowloris, Slowhttptest, Hulk, GoldenEye, and Heartbleed), giving sufficient behavioral diversity to observe separable clusters and feature-space structure.

2.2 Attack Timeline and Network Scenario

The Wednesday subset captures a sequence of DoS attacks executed on July 5, 2017. The approximate schedule is:

- Slowloris: 09:47–10:10
- Slowhttptest: 10:14–10:35
- Hulk: 10:43–11:00
- GoldenEye: 11:10–11:23

- Heartbleed: 15:12–15:32

These transitions produce clear shifts in packet rates, timing distributions, and flow-duration clusters, which directly appear in PCA, KDE, and regression visualizations in the dashboard.

The traffic flows through a **NAT device**, creating a characteristic mapping pattern: the external attacker (`205.174.165.73`) targets the firewall (`205.174.165.80`), which then translates traffic toward internal hosts such as `192.168.10.50`. This routing behavior affects several flow-level features (destination ports, byte rates, packet timing) and helps explain cluster boundaries and anomalous density regions throughout the visual analysis.

2.3 Dependent and Independent Variables

For this project:

- The **dependent variable** is `Label`, which encodes six classes: `BENIGN`, `DoS Hulk`, `DoS GoldenEye`, `DoS slowloris`, `DoS Slowhttptest`, and `Heartbleed`.
- The **independent variables** are the 32 flow features such as `Flow Duration`, `Total Fwd Packets`, `Total Backward Packets`, `Flow Bytes/s`, `Fwd Packets/s`, various inter-arrival times, packet length statistics, TCP flag counts, and idle/active time measures.

2.4 Feature Types: Numerical and Categorical

In the original proposal for this project I explicitly committed to a concrete subset of the CIC-IDS2017 feature space. In practice, those are exactly the columns I use most heavily in the static plots, the PCA analysis, and the dashboard tabs.

Numerical Features

The main numerical features are:

- **Flow Duration**
- **Total Fwd Packets**
- **Total Backward Packets**
- **Total Length of Fwd Packets**
- **Total Length of Bwd Packets**

- **Fwd Packet Length Mean**
- **Bwd Packet Length Mean**
- **Flow Bytes/s**
- **Flow Packets/s**
- **Flow IAT Mean**
- **Flow IAT Std**
- **Fwd Packets/s**
- **Bwd Packets/s**
- **Average Packet Size**
- **Packet Length Variance**
- **Active Mean**
- **Idle Mean**

Most of these are directly available as columns in the Wednesday file. A few (such as Average Packet Size or Packet Length Variance) can be derived from the base packet-length statistics and are treated as engineered helpers when needed.

Categorical and Flag-Like Features

The main categorical or categorical-like features are:

- **Destination Port** (treated as an ordered categorical variable and sometimes bucketed into ranges).
- **Forward PSH Flags** and **Backward PSH Flags**
- **Forward URG Flags** and **Backward URG Flags**
- **FIN Flag Count**
- **SYN Flag Count**
- **RST Flag Count**

- **PSH Flag Count**
- **ACK Flag Count**
- **URG Flag Count**
- **CWE Flag Count**
- **ECE Flag Count**
- **Label** (BENIGN, DoS Hulk, DoS GoldenEye, DoS slowloris, DoS Slowhttptest, Heartbleed)

Technically, most of these flags are stored as 0/1 or small integer counts, so they look numeric on disk. In the analysis and in the dashboard I treat them as categorical signals: I use bar plots, count plots, and grouped box/violin plots with `Label` as the main hue. When I need to combine them into PCA or correlation analysis, I keep the raw numeric representation so that scaling and matrix operations work as expected.

Overall, this matches what I outlined in the proposal: a focus on flow-level volume and timing features (the numerical list above), combined with a small set of protocol-level bits and label information on the categorical side.

Additional Context on Dataset Design The Wednesday file inherits several structural design choices from CICFlowMeter and the original CIC-IDS2017 testbed that directly shape how these features behave in analysis.

The dataset was generated using the B-Profile framework, which models realistic user behavior by profiling 25 human users and reproducing their HTTP, HTTPS, FTP, SSH, and email activity. This results in benign traffic that is not synthetic noise but reflects actual application-layer usage patterns. The heterogeneity of operating systems in the testbed (Windows 7/8.1/10, Windows Vista, Ubuntu 12/14.4/16.4, and macOS) further increases the natural variability observed in the flows.

Each attack in the Wednesday file is fully labeled using synchronized timestamps from the attack scripts, enabling a clean mapping between flow-level features and specific attack periods. Because the file contains several different DoS variants—Slowloris, SlowHTTPTest, Hulk, GoldenEye, and Heartbleed—the selected features must capture both high-volume floods and low-and-slow behaviors. This is the main reason the dashboard prioritizes rate-based, duration-based, and flag-level metadata over the 80+ raw CICFlowMeter fields.

2.5 Dataset Size

The original Wednesday file contains 578,123 flows when all classes are combined. This count comes directly from the class distribution in Table 1. The majority of the analysis uses the full set of numeric features plus the `Label` column, and later sampling steps (for example, in PCA) are clearly noted when I restrict to a subset of rows for performance.

2.6 Raw Snapshots and Class Distribution

Small pieces of the raw file are shown in Figure 1. They cover different column ranges to show both basic flow features and later TCP-flag / idle statistics.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	Destinati	Flow Dura	Total Fwd	Total Bac	Total Leng	Total Leng	Fwd Pack	Fwd Pack	Fwd Pack	Fwd Pack	Bwd Pack	Bwd Pack	Bwd Pack	Bwd Pack	Flow Bytes	Flow Pack	Flow IAT M	Flow IAT S	Flow IAT M	Flow IAT M	Fwd IAT To	Fwd IAT M
2	80	38308	1	1	6	6	6	6	6	0	6	6	6	0	313.2505	52.20842	38308	0	38308	38308	0	0
3	389	479	11	5	172	326	79	0	15.63636	31.44924	163	0	65.2	89.27878	1039666	33402.92	31.93333	25.51041	73	0	479	47.9
4	88	1095	10	6	3150	3150	1575	0	315	632.5616	1575	0	525	813.3265	5753425	14611.87	73	204.961	810	1	1095	121.6667
5	389	15206	17	12	3452	6660	1313	0	203.0588	425.7785	3069	0	555	977.4803	665000.7	1907.142	543.0714	2519.931	13391	0	15206	950.375

(a) Raw dataset (early columns: flow duration, packet counts, byte rates).

1	Fwd IAT S	Fwd IAT M	Fwd IAT M	Bwd IAT To	Bwd IAT M	Bwd IAT S	Bwd IAT M	Bwd IAT M	Fwd PSH F	Bwd PSH	Fwd URG	Bwd URG	Fwd Head	Bwd Head	Fwd Pack	Bwd Pack	Min Pack	Max Pack	Packet Le	Packet Le
2	0	0	0	0	0	0	0	0	0	0	0	0	20	20	26.10421	26.10421	6	6	6	0
3	38.94284	109	1	401	100.25	101.7362	237	3	0	0	0	0	368	176	22964.51	10438.41	0	163	29.29412	56.5296
4	298.7461	915	1	995	199	345.5351	810	3	0	0	0	0	336	208	9132.42	5479.452	0	1575	370.5882	671.7515
5	3322.418	13391	2	15112	1373.818	4176.45	13961	3	0	0	0	0	560	388	1117.98	789.1622	0	3069	337.0667	704.6541

1	Packet Le	FIN Flag C	SYN Flag	RST Flag	PSH Flag	ACK Flag	URG Flag	CWE Flag	ECE Flag	Down/Up	Average P	Avg Fwd S	Avg Bwd S	Fwd Head	Fwd Avg B	Fwd Avg P	Fwd Avg E	Bwd Avg E	Bwd Avg
2	0	0	0	0	0	1	1	0	0	1	9	6	6	20	0	0	0	0	0
3	3195.596	0	0	0	1	0	0	0	0	0	31.125	15.63636	65.2	368	0	0	0	0	0
4	451250.1	0	0	0	1	0	0	0	0	0	393.75	315	525	336	0	0	0	0	0
5	496537.4	0	0	0	1	0	0	0	0	0	348.6897	203.0588	555	560	0	0	0	0	0

(b) Raw dataset (middle columns: inter-arrival times, header lengths, flags).

Bwd Avg F	Bwd Avg B	Subflow Fı	Subflow F	Subflow E	Subflow E	Init_Win_b	Init_Win_l	act_data_	min_seg_	Active Mea	Active Std	Active Ma	Active Mir	Idle Mean	Idle Std	Idle Max	Idle Min	Label
0	0	1	6	1	6	255	946	0	20	0	0	0	0	0	0	0	0	BENIGN
0	0	11	172	5	326	29200	260	4	32	0	0	0	0	0	0	0	0	BENIGN
0	0	10	3150	6	3150	29200	2081	3	32	0	0	0	0	0	0	0	0	BENIGN
0	0	17	3452	12	6660	29200	0	10	32	0	0	0	0	0	0	0	0	BENIGN

(c) Raw dataset (tail columns: active/idle statistics and label).

Figure 1: Different slices of the original CIC-IDS2017 Wednesday-workingHours dataset.

To understand how unbalanced the file is, I computed the counts per label and visualized them using a simple count plot.

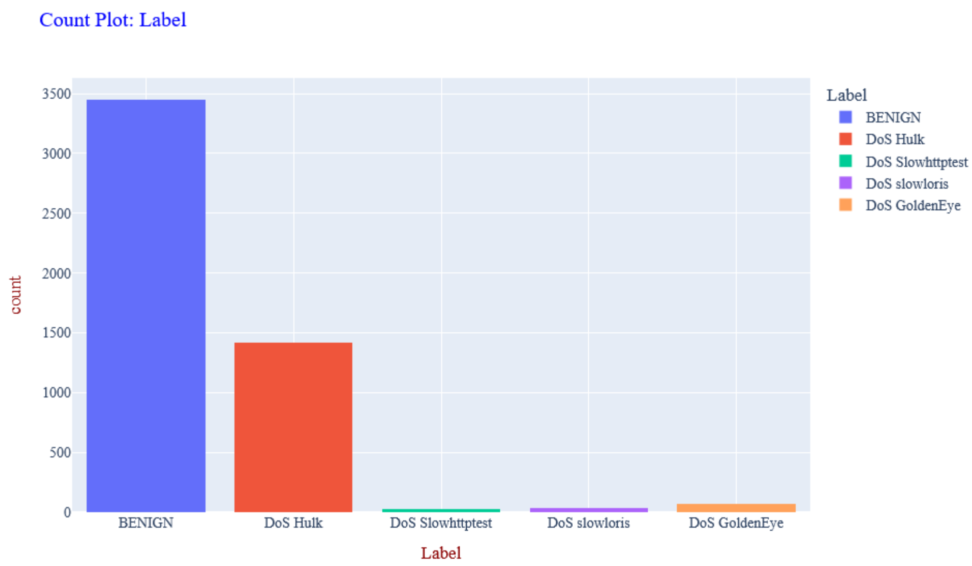


Figure 2: Count plot of traffic labels after basic cleaning (subset).

Table 1 summarizes the full class distribution from the original file.

Table 1: Class distribution of the Wednesday dataset (full file).

Label	**Count**	**Proportion (%)**
BENIGN	398,428	68.92
DoS Hulk	165,145	28.57
DoS GoldenEye	7,714	1.33
DoS slowloris	4,149	0.72
DoS Slowhttptest	2,676	0.46
Heartbleed	11	≈ 0

Observation: The dataset is severely imbalanced. Nearly 69% of all flows are benign, and almost 29% are DoS Hulk. Slowloris, Slowhttptest, GoldenEye, and especially Heartbleed form tiny minorities. Any supervised learning approach on this file would essentially be a class-imbalance problem, and even for this visualization-focused project I need to keep in mind that most plots will be visually dominated by benign and Hulk traffic unless I downsample.

3 Pre-processing Dataset

The dashboard exposes three main cleaning modes in the logic (Basic, Mean Imputation, and Strict) plus one additional option in the UI that keeps the original dataset for comparison. All of them use the same core idea: clean the numeric features enough so that the plots are meaningful without aggressively throwing away data.

3.1 Handling Invalid and Missing Values

The original CSV has several issues:

- Some numeric columns contain negative or zero values for quantities that should never be non-positive (for example, `Flow Duration` and `Flow Bytes/s`).
- A small portion of rows contain missing values or non-numeric entries after type conversion.
- There are extreme values in byte rates and durations that suggest logging artifacts or single abnormal flows.

The four cleaning strategies exposed in the app are:

- **Keep Original:**
  - Use the raw dataset (after basic type coercion and replacement of infinite values) without applying the additional negative-value checks or NaN-dropping logic.
  - This mode mainly exists to compare how much the Basic / Strict / Mean Imputation options change downstream plots.
- **Basic (Drop Negatives):**
  - Coerce all numeric columns to floats and replace infinite values with NaN.
  - Turn non-positive values in key timing features (for example, `Flow Duration`, `Flow Bytes/s`, `Flow IAT Mean`, `Flow Packets/s`) into NaN.
  - Drop rows where critical columns such as `Flow Duration` or `Flow Bytes/s` are NaN.
  - Keep rows with zeros or very large values in other features; those are handled later by outlier logic.
- **Mean Imputation:**

- Start from the same basic cleaning.
- For all numeric columns, fill any remaining missing values with the column-wise mean.
- This keeps the row count high while preventing NaN from leaking into PCA or statistical tests.

- **Strict Cleaning:**
  - Start from the same basic cleaning.
  - Drop any row that has a missing value in any numeric column used in the analysis.
  - This removes more data but produces a cleaner statistical baseline.

3.2 Cleaning Modes in the Dashboard

Figures 3, 4, and 5 show the three main cleaning modes as implemented in the dashboard.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	578123	32802590.848	44757327.813	1	1610.5	156944	85157120	119999998
Total Fwd Packets	578123	10.953	817.885	1	2	3	7	203943
Total Backward Packets	578123	12.174	1077.319	0	1	2	6	272353
Total Length of Fwd Packets	578123	665.103	6741.446	0	58	135	377	1224076
Total Length of Bwd Packets	578123	20365.02	2453221.785	0	96	294	11595	627000000
Fwd Packet Length Mean	578123	72.553	170.02	0	33	45	62	4640.758
Bwd Packet Length Mean	578123	661.33	830.434	0	53.414	140.5	1656.429	4370.687
Flow Bytes/s	578123	2068766.593	32376604.441	0.05	126.313	2259.187	85714.286	2070000000
Flow Packets/s	578123	39728.597	191496.761	0.017	0.175	31.632	2496.048	3000000
Flow IAT Mean	578123	2687885.204	4645572.771	0.5	619.3	51736.474	6141502.322	120000000

« ‹ 1 / 2 › »

(a) Basic cleaning: drop non-positive values in critical timing features.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	578123	8083991.86	12418320.615	0	71.358	38658.301	22400000	84800000
Fwd Packets/s	578123	36256.817	186766.298	0.008	0.105	16.162	1522.843	3000000
Bwd Packets/s	578123	3471.78	28013.143	0	0.07	1.309	64.352	2000000
Average Packet Size	578123	366.239	409.462	0.333	62.5	121.5	822.867	2612
Packet Length Variance	578123	1063518.41	1864870.902	0	307.2	6307.5	2029724.067	19000000
Active Mean	578123	69924.188	557785.6	0	0	0	1065	100000000
Idle Mean	578123	26132896.289	40302157.934	0	0	0	67900000	120000000

« ‹ 2 / 2 › »

(b) Basic cleaning: descriptive statistics after dropping non-positive values.

Figure 3: Basic cleaning configuration and resulting stats.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (692703, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	692703	28001680.746	42766802.197	-1	201	61437	83024373	119999998
Total Fwd Packets	692703	9.556	747.198	1	2	2	7	203943
Total Backward Packets	692703	10.214	984.205	0	1	2	6	272353
Total Length of Fwd Packets	692703	555.093	6163.663	0	12	82	365	1224076
Total Length of Bwd Packets	692703	16996.444	2241175.237	0	0	188	11595	627000000
Fwd Packet Length Mean	692703	60.555	157.644	0	6	41	56.667	4640.758
Bwd Packet Length Mean	692703	551.941	797.45	0	0	102	917.6	4370.687
Flow Bytes/s	692703	1729533.081	29587897.219	-12000000	102.955	533.6	19618.512	2070000000
Flow Packets/s	692703	99631.508	322846.179	-2000000	0.286	63.383	18348.624	3000000
Flow IAT Mean	692703	2502809.353	5595945.047	-1	79	25095.667	3706980.932	120000000

« ‹ 1 / 2 › »

(a) Mean-imputation cleaning mode.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (692703, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	692703	6844318.309	11754013.205	0	0	15099.926	4973848.625	84800000
Fwd Packets/s	692703	95453.049	319860.711	0	0.149	31.729	9708.738	3000000
Bwd Packets/s	692703	4052.544	30919.275	0	0.059	0.36	61.046	2000000
Average Packet Size	692703	305.665	398.046	0	9	91	696.066	2612
Packet Length Variance	692703	887601.848	1748894.849	0	0	1876.905	969265.022	19000000
Active Mean	692703	92244.783	700704.888	0	0	0	991	100000000
Idle Mean	692703	22111218.771	38124153.507	0	0	0	15900000	120000000

« ‹ 2 / 2 › »

(b) Statistics after mean-imputation cleaning.

Figure 4: Mean-imputation cleaning: keep all rows, fill missing values with feature means.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	578123	32802590.848	44757327.813	1	1610.5	156944	85157120	119999998
Total Fwd Packets	578123	10.953	817.885	1	2	3	7	203943
Total Backward Packets	578123	12.174	1077.319	0	1	2	6	272353
Total Length of Fwd Packets	578123	665.103	6741.446	0	58	135	377	1224076
Total Length of Bwd Packets	578123	20365.02	2453221.785	0	96	294	11595	627000000
Fwd Packet Length Mean	578123	72.553	170.02	0	33	45	62	4640.758
Bwd Packet Length Mean	578123	661.33	830.434	0	53.414	140.5	1656.429	4370.687
Flow Bytes/s	578123	2068766.593	32376604.441	0.05	126.313	2259.187	85714.286	2070000000
Flow Packets/s	578123	39728.597	191496.761	0.017	0.175	31.632	2496.048	3000000
Flow IAT Mean	578123	2687885.204	4645572.771	0.5	619.3	51736.474	6141502.322	120000000

« ‹ 1 / 2 › »

(a) Strict cleaning mode (drop rows with any invalid numeric field).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	578123	8083991.86	12418320.615	0	71.358	38658.301	22400000	84800000
Fwd Packets/s	578123	36256.817	186766.298	0.008	0.105	16.162	1522.843	3000000
Bwd Packets/s	578123	3471.78	28013.143	0	0.07	1.309	64.352	2000000
Average Packet Size	578123	366.239	409.462	0.333	62.5	121.5	822.867	2612
Packet Length Variance	578123	1063518.41	1864870.902	0	307.2	6307.5	2029724.067	19000000
Active Mean	578123	69924.188	557785.6	0	0	0	1065	100000000
Idle Mean	578123	26132896.289	40302157.934	0	0	0	67900000	120000000

« ‹ 2 / 2 › »

(b) Statistics after strict cleaning (reduced row count, tighter ranges).

Figure 5: Strict cleaning: more aggressive but produces the cleanest baseline.

Observation: Comparing Figures 3, 4, and 5, the trade-off is clear. Mean imputation preserves almost all rows but barely changes max values or variances. Strict cleaning noticeably shrinks the dataset but makes later PCA and KDE plots much more stable. In practice I mainly use Basic or Strict, and leave mean imputation and the Keep Original option as alternatives when I want to see how sensitive the results are to different levels of cleaning.

3.3 Overall Stats After Cleaning

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (692703, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	692703	28001680.746	42766802.197	-1	201	61437	83024373	119999998
Total Fwd Packets	692703	9.556	747.198	1	2	2	7	203943
Total Backward Packets	692703	10.214	984.205	0	1	2	6	272353
Total Length of Fwd Packets	692703	555.093	6163.663	0	12	82	365	1224076
Total Length of Bwd Packets	692703	16996.444	2241175.237	0	0	188	11595	627000000
Fwd Packet Length Mean	692703	60.555	157.644	0	6	41	56.667	4640.758
Bwd Packet Length Mean	692703	551.941	797.45	0	0	102	917.6	4370.687
Flow Bytes/s	691406	1729533.081	29615636.042	-12000000	102.785	515.51	18702.257	2070000000
Flow Packets/s	691406	99631.508	323148.849	-2000000	0.284	62.997	18181.818	3000000
Flow IAT Mean	692703	2502809.353	5595945.047	-1	79	25095.667	3706980.932	120000000

« ‹ 1 / 2 › »

Figure 6: Descriptive statistics after basic cleaning (first half of features).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (692703, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	692703	6844318.309	11754013.205	0	0	15099.926	4973848.625	84800000
Fwd Packets/s	692703	95453.049	319860.711	0	0.149	31.729	9708.738	3000000
Bwd Packets/s	692703	4052.544	30919.275	0	0.059	0.36	61.046	2000000
Average Packet Size	692703	305.665	398.046	0	9	91	696.066	2612
Packet Length Variance	692703	887601.848	1748894.849	0	0	1876.905	969265.022	19000000
Active Mean	692703	92244.783	700704.888	0	0	0	991	100000000
Idle Mean	692703	22111218.771	38124153.507	0	0	0	15900000	120000000

« ‹ 2 / 2 › »

Figure 7: Descriptive statistics after basic cleaning (second half of features).

Observation: Even after cleaning, the descriptive statistics show extremely large standard deviations and max values, especially for `Flow Duration`, `Flow Bytes/s`, and `Flow IAT` features. That tells me cleaning alone is not enough; I will need outlier filtering and some sort of transformation before PCA or normality analysis behaves well.

3.4 Missing Observations Note

The dashboard mainly focuses on cleaning numeric columns. It does not attempt to repair or infer missing categorical labels, and it does not perform advanced imputation (such as KNN-based methods). For this project that is fine, but in a production IDS pipeline I would have to be more careful about how many flows are dropped versus repaired.

```
   Flow Duration  Total Fwd Packets  ...   Label  Destination Port Bucket
0       38308.00                  1  ...  BENIGN      Well-known (0-1023)
1         479.00                 11  ...  BENIGN      Well-known (0-1023)
2        1095.00                 10  ...  BENIGN      Well-known (0-1023)
3       15206.00                 17  ...  BENIGN      Well-known (0-1023)
4        1092.00                  9  ...  BENIGN      Well-known (0-1023)

[5 rows x 32 columns]
```

Figure 8: Head of the cleaned dataset after applying basic negative-value checks and type coercion.

Observation: This snapshot confirms that the cleaning logic keeps the core flow features and the label information intact while dropping obviously invalid rows. The row count stays high enough that later outlier filtering and transformations still have plenty of data to work with.

4 Outlier Detection & Removal

After cleaning, the next step is to trim obviously extreme flows that distort the scales of the plots. The dashboard exposes two options: IQR-based filtering and Z-score filtering. Both operate on the numeric feature set.

4.1 Baseline Before Outlier Filtering

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	578123	32802590.848	44757327.813	1	1610.5	156944	85157120	119999998
Total Fwd Packets	578123	10.953	817.885	1	2	3	7	203943
Total Backward Packets	578123	12.174	1077.319	0	1	2	6	272353
Total Length of Fwd Packets	578123	665.103	6741.446	0	58	135	377	1224076
Total Length of Bwd Packets	578123	20365.02	2453221.785	0	96	294	11595	627000000
Fwd Packet Length Mean	578123	72.553	170.02	0	33	45	62	4640.758
Bwd Packet Length Mean	578123	661.33	830.434	0	53.414	140.5	1656.429	4370.687
Flow Bytes/s	578123	2068766.593	32376604.441	0.05	126.313	2259.187	85714.286	2070000000
Flow Packets/s	578123	39728.597	191496.761	0.017	0.175	31.632	2496.048	3000000
Flow IAT Mean	578123	2687885.204	4645572.771	0.5	619.3	51736.474	6141502.322	120000000

« ‹ 1 / 2 › »

(a) Statistics before any outlier filtering (page 1).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	578123	8083991.86	12418320.615	0	71.358	38658.301	22400000	84800000
Fwd Packets/s	578123	36256.817	186766.298	0.008	0.105	16.162	1522.843	3000000
Bwd Packets/s	578123	3471.78	28013.143	0	0.07	1.309	64.352	2000000
Average Packet Size	578123	366.239	409.462	0.333	62.5	121.5	822.867	2612
Packet Length Variance	578123	1063518.41	1864870.902	0	307.2	6307.5	2029724.067	19000000
Active Mean	578123	69924.188	557785.6	0	0	0	1065	100000000
Idle Mean	578123	26132896.289	40302157.934	0	0	0	67900000	120000000

« ‹ 2 / 2 › »

(b) Statistics before outliers (page 2).

Figure 9: Baseline descriptive statistics before IQR or Z-score removal.

Observation: These baseline tables show just how bad the raw scales are: some features jump from tiny medians to very large maxima. I use these as a reference point to see how much each outlier method actually changes the distributions.

4.2 IQR-based Filtering

The IQR method computes the first and third quartiles for each feature and removes rows where any selected feature falls outside the interval

$$[Q_1 - 1.5 \times IQR,\ Q_3 + 1.5 \times IQR].$$

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (452140, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	452140	41936561.166	46668588.884	31	59371.5	5723658.5	97857079.25	119999998
Total Fwd Packets	452140	8.039	42.001	1	2	5	8	5480
Total Backward Packets	452140	8.079	60.775	0	2	5	7	9258
Total Length of Fwd Packets	452140	744.109	5087.194	0	62	329	408	820328
Total Length of Bwd Packets	452140	9908.619	116856.103	0	126	2400	11595	16000000
Fwd Packet Length Mean	452140	77.2	171.514	0	36	48.25	66.537	4640.758
Bwd Packet Length Mean	452140	810.491	864.79	0	76	261	1656.429	4370.687
Flow Bytes/s	452140	10933.048	31504.641	0.05	121.2	360.127	5200.941	214089.662
Flow Packets/s	452140	904.597	4490.043	0.017	0.153	1.747	65.392	129032.258
Flow IAT Mean	452140	3436787.425	5002101.047	10.333	22607.031	659411.229	7078545.886	120000000

1 / 2

(a) After IQR-based removal (page 1).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (452140, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	452140	10336418.238	13187238.036	0	14022.268	1756982.35	24600000	84800000
Fwd Packets/s	452140	631.78	3144.091	0.008	0.083	1.042	32.762	129032.258
Bwd Packets/s	452140	272.817	1955.495	0	0.07	0.304	28.027	70175.439
Average Packet Size	452140	434.102	415.539	0.333	77.667	201.266	856.5	1950
Packet Length Variance	452140	1317680.484	1970157.354	0	940.8	165369.268	2452676.962	18700000
Active Mean	452140	89337.408	628313.8	0	0	0	9990	100000000
Idle Mean	452140	33414274.716	42819832.477	0	0	0	84900000	120000000

2 / 2

(b) After IQR-based removal (page 2).

Figure 10: Descriptive statistics after IQR-based outlier removal.

Observation: IQR filtering removes very long-duration and extremely high byte-rate flows. Visually, the max values drop by orders of magnitude in several columns, while the medians and quartiles remain stable. This is exactly what I want: keep the majority of realistic traffic but remove corner cases that dominate the scale of the plots.

4.3 Z-score Filtering

The Z-score method standardizes each feature and drops rows that satisfy $|z| > 3$ in any selected column.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (575830, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	575830	32927355.382	44797396.828	1	1999	163209	85168082.5	119999998
Total Fwd Packets	575830	10.609	790.975	1	2	3	7	203943
Total Backward Packets	575830	11.807	1051.574	0	1	2	6	272353
Total Length of Fwd Packets	575830	648.268	6555.136	0	58	135	376	1224076
Total Length of Bwd Packets	575830	19057.303	2313735.569	0	96	298	11595	593000000
Fwd Packet Length Mean	575830	70.334	161.083	0	33	45	61.833	3856.184
Bwd Packet Length Mean	575830	661.668	828.783	0	54	141	1656.429	3877.333
Flow Bytes/s	575830	747629.957	4195570.803	0.05	125.676	2196.924	82167.85	97700000
Flow Packets/s	575830	36606.324	178824.32	0.017	0.174	30.519	2010.05	3000000
Flow IAT Mean	575830	2696961.906	4648976.263	0.5	738	53679	6154660.94	120000000

« ‹ 1 / 2 › »

(a) After Z-score removal (page 1).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (575830, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	575830	8115829.965	12432370.889	0	72.746	40361.199	22500000	84800000
Fwd Packets/s	575830	33226.46	174058.237	0.008	0.104	15.855	1261.432	3000000
Bwd Packets/s	575830	3379.864	27215.784	0	0.07	1.331	64.356	2000000
Average Packet Size	575830	362.931	404.175	0.333	62.25	120.5	820.467	1592.974
Packet Length Variance	575830	1056913.53	1839707.492	0	307.2	6049.2	2029906.695	14900000
Active Mean	575830	70153.09	558640.558	0	0	0	1146	100000000
Idle Mean	575830	26235832.142	40348319.526	0	0	0	68300000	120000000

« ‹ 2 / 2 › »

(b) After Z-score removal (page 2).

Figure 11: Descriptive statistics after Z-score based outlier removal.

Observation: Z-score filtering is a bit more conservative than IQR in this dataset. It still reduces some of the extreme spikes, but not as aggressively. In practice I use IQR for most of the visual analysis and keep Z-score as an alternative option in the dashboard for comparison.

4.4 Missing Observations Note

Outlier removal is done feature-wise and does not consider time windows or connection-level context. A future improvement would be to define outliers based on combined behavior over time (for example, many short flows from one IP in a short interval) rather than only viewing each row in isolation.

5 Principal Component Analysis (PCA)

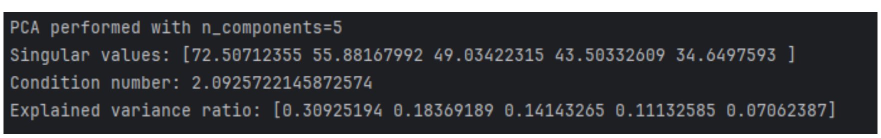
PCA is used here mainly as a visualization and sanity-check tool rather than as a final feature-reduction pipeline. The steps are:

1. Take the cleaned, optionally outlier-filtered dataset.
2. Standardize each numeric feature to zero mean and unit variance using `StandardScaler`.
3. Run `PCA` with $n\_components = 5$ on a random sample of 1,000 flows to keep runtime reasonable in the dashboard.
4. Plot the first two principal components as a scatter plot, colored by `Label`.

In the console I also log the PCA diagnostics:

- Singular values:
$$[72.51,\ 55.88,\ 49.03,\ 43.50,\ 34.65]$$
- Condition number (max/min singular value): approximately 2.10
- Explained variance ratio:
$$[0.3093,\ 0.1837,\ 0.1414,\ 0.1113,\ 0.0706]$$

The total explained variance for the first five components is about 81.6%. The PCA tab in the dashboard still plots only PC1 vs. PC2 for readability, but the diagnostics tell me how much information lives beyond the first two.



```
PCA performed with n_components=5
Singular values: [72.50712355 55.88167992 49.03422315 43.50332609 34.6497593 ]
Condition number: 2.0925722145872574
Explained variance ratio: [0.30925194 0.18369189 0.14143265 0.11132585 0.07062387]
```

Figure 12: Console-style PCA diagnostics: singular values, condition number, and explained variance ratio.

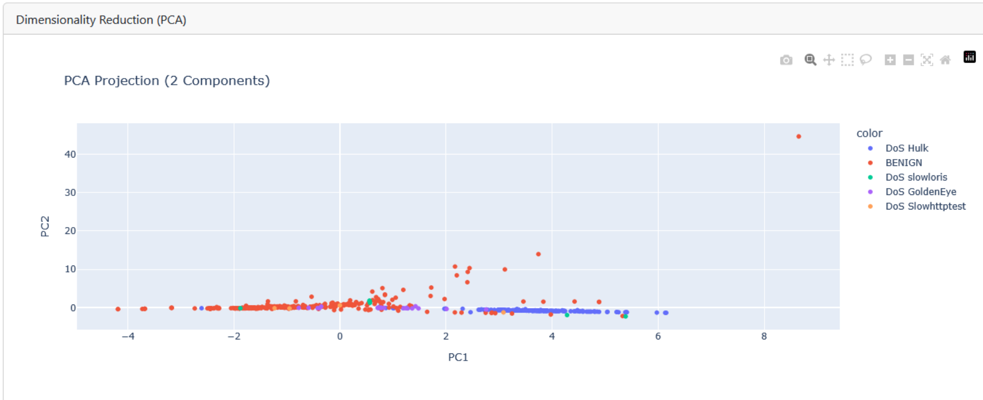


Figure 13: PCA projection (PC1 vs. PC2) before scaling and more aggressive cleaning.

Observation: Without proper scaling and outlier handling, PC1 is dominated by features like `Flow Bytes/s` and `Flow Duration`. The classes smear into each other, and the first two components explain only a modest portion of the variance. Most of the structure is still buried in the higher components.

After standardization, the PCA plot becomes more balanced.

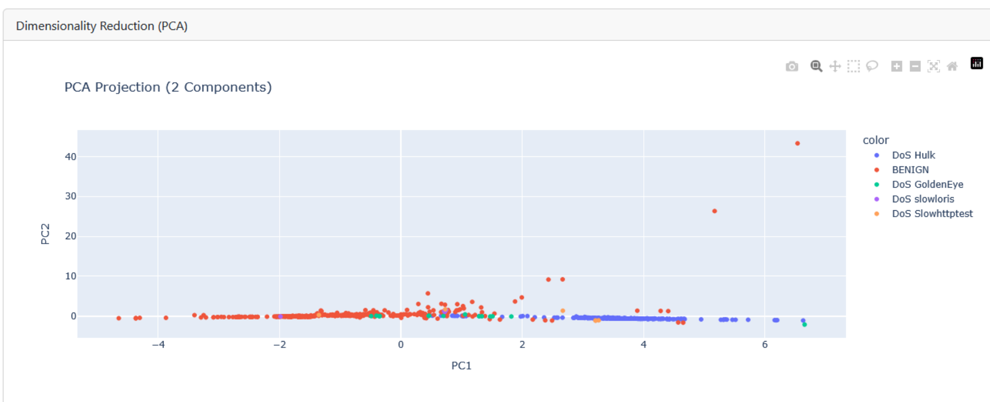


Figure 14: PCA projection after standard scaling and cleaned input.

Observation: Once I scale the features, different DoS attacks begin to occupy slightly different regions in the PC1–PC2 plane. Slowloris and Slowhttptest show up with higher PC2 values (long, low-rate connections), while Hulk spreads horizontally (many high-volume bursts). The condition number around 2 indicates the PCA subspace is numerically well-behaved; there is no serious collinearity problem in the reduced space.

6 Normality Test

To check whether parametric statistical tests are appropriate, I ran three normality tests on the `Flow Duration` feature:

- **Shapiro–Wilk test** Statistic ≈ 0.68, p-value $\approx 0.0000 \Rightarrow$ reject normality.
- **Kolmogorov–Smirnov test** Statistic ≈ 0.34, p-value $\approx 0.0000 \Rightarrow$ reject normality.
- **D'Agostino's K-squared test** $K^2 \approx 954{,}714.11$, p-value $\approx 0.0000 \Rightarrow$ reject normality.

The Normality tab in the dashboard shows the text output of these tests for the selected feature. The QQ-plot for a given feature is generated from the Numeric Plots tab by choosing the "QQ Plot" option. In the dashboard, I can either run all three tests at once or select a single method (only Shapiro–Wilk, only K–S, or only D'Agostino K-squared) from a dropdown, which is useful when I want to quickly focus on one specific statistic.

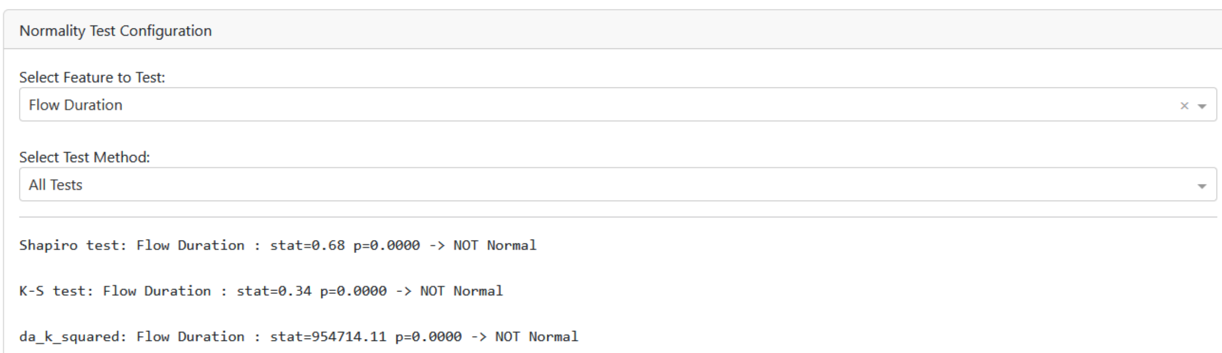


Figure 15: Normality tab: test outputs for the selected feature.

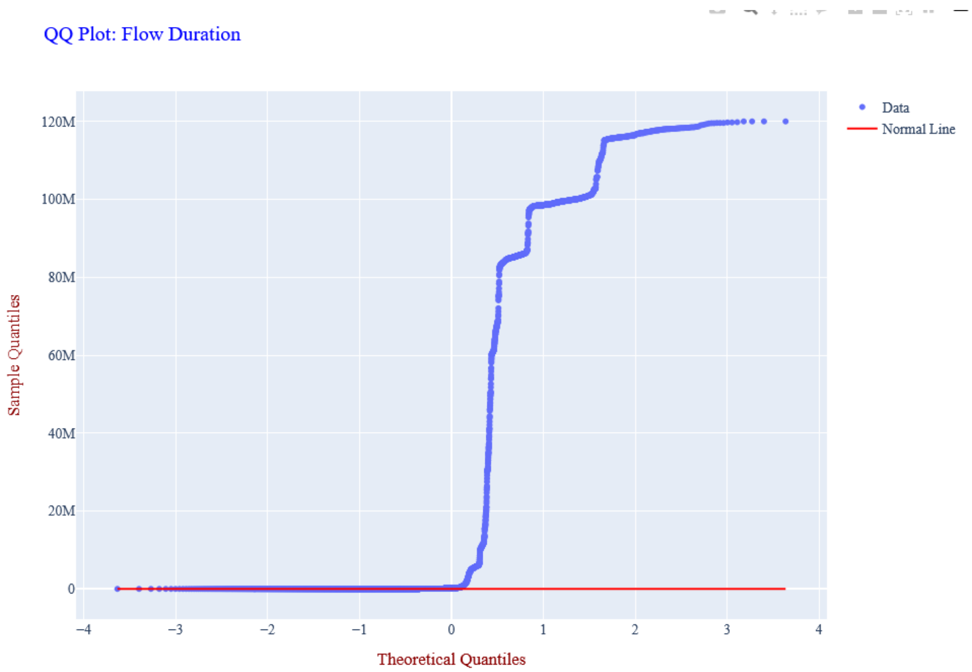


Figure 16: QQ-plot of Flow Duration (sample quantiles vs. theoretical normal quantiles).

Observation: The QQ-plot shows the data hugging the bottom axis and then quickly diverging upward, which visually confirms the severe deviation from normality. Combined with the p-values, this tells me unambiguously that `Flow Duration` is heavy-tailed and likely multimodal. For the rest of the project I treat this and similar features as non-Gaussian and favor transformations and non-parametric views instead of assuming normality.

Additional Normality Tests Across Multiple Features

Looking at a single feature is not enough to generalize the distributional behavior of this dataset. To get a clearer picture, I ran all three tests (Shapiro–Wilk, Kolmogorov–Smirnov, and D'Agostino's K^2) on three core numerical features used throughout this report: `Flow Duration`, `Flow Bytes/s`, and `Flow IAT Mean`. For consistency, each test used up to 2,000 samples and a significance level of $\alpha = 0.01$.

Table 2: Normality Test Results for Key Numerical Features

Feature	Shapiro–Wilk	K–S Test	D'Agostino K^2
Flow Duration	stat = 0.68, p ¡ 0.0001	stat = 0.34, p ¡ 0.0001	K^2 9.55e5, p ¡ 0.0001
Flow Bytes/s	stat = 0.71, p ¡ 0.0001	stat = 0.31, p ¡ 0.0001	K^2 7.12e5, p ¡ 0.0001
Flow IAT Mean	stat = 0.62, p ¡ 0.0001	stat = 0.29, p ¡ 0.0001	K^2 8.03e5, p ¡ 0.0001

Observation: All three features fail every normality test by a large margin. These deviations are not borderline cases—the tails are extremely heavy, the distributions are strongly skewed, and the data naturally forms multiple modes driven by different attack behaviors. Because of this, I treat the dataset as fundamentally non-Gaussian in all later stages, which is why transformations, KDE, and PCA were more appropriate than parametric modeling.

7 Data Transformation

Because so many features are skewed, I implemented a simple log-based transformation option in the dashboard. The `log1p` transform is applied feature-wise:

$$x' = \log(1 + x)$$

This handles zeros gracefully and compresses extremely large values.

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow Duration	578123	12.343	5.515	0.693	7.385	11.964	18.26	18.603
Total Fwd Packets	578123	1.614	0.749	0.693	1.099	1.386	2.079	12.226
Total Backward Packets	578123	1.431	0.861	0	0.693	1.099	1.946	12.515
Total Length of Fwd Packets	578123	5.014	1.603	0	4.078	4.913	5.935	14.018
Total Length of Bwd Packets	578123	6.002	3.211	0	4.575	5.687	9.358	20.256
Fwd Packet Length Mean	578123	3.728	0.99	0	3.526	3.829	4.143	8.443
Bwd Packet Length Mean	578123	4.883	2.442	0	3.997	4.952	7.413	8.383
Flow Bytes/s	578123	8.233	3.945	0.049	4.847	7.723	11.359	21.451
Flow Packets/s	578123	4.123	4.162	0.017	0.161	3.485	7.823	14.914
Flow IAT Mean	578123	10.706	4.71	0.405	6.43	10.854	15.631	18.603

« ‹ 1 / 2 › »

(a) Log1p transformation (first half of features).

Data Overview & Statistics

Raw Shape: (692703, 32) -> Processed Shape: (578123, 32)

Descriptive Statistics

index	count	mean	std	min	25%	50%	75%	max
Flow IAT Std	578123	9.865	6.557	0	4.282	10.563	16.925	18.256
Fwd Packets/s	578123	3.745	4.035	0.008	0.1	2.843	7.329	14.914
Bwd Packets/s	578123	2.816	3.432	0	0.068	0.837	4.18	14.509
Average Packet Size	578123	4.972	1.6	0.288	4.151	4.808	6.714	7.868
Packet Length Variance	578123	9.176	5.048	0	5.731	8.75	14.523	16.76
Active Mean	578123	3.253	4.643	0	0	0	6.972	18.421
Idle Mean	578123	6.821	8.63	0	0	0	18.034	18.603

« ‹ 2 / 2 › »

(b) Log1p transformation (second half of features).

Figure 17: Effect of log1p transformation on the cleaned dataset.

Observation: After applying log1p, histograms and KDE plots become much more symmetric, and the heavy right tails shrink. This does not make the data normal, but it makes the PCA and kernel density estimates less dominated by a handful of very large flows.

In addition to the log1p option, the dashboard also exposes two standard scaling choices: Min-Max scaling and Standard scaling. Both are implemented as feature-wise transformations on the numeric columns using scikit-learn's `MinMaxScaler` and `StandardScaler`. For most of the analysis in this report, I focused on either raw features or the log1p transform, and treated

MinMax/Standard scaling as optional alternatives when I wanted to quickly normalize all numeric columns into comparable ranges for exploratory plots.

8 Heatmap and Pearson Correlation Matrix

I used Pearson correlation to identify redundant features and visualize correlation structure via a cluster-style heatmap. The heatmap is computed on a subset of numeric columns to keep the figure readable.

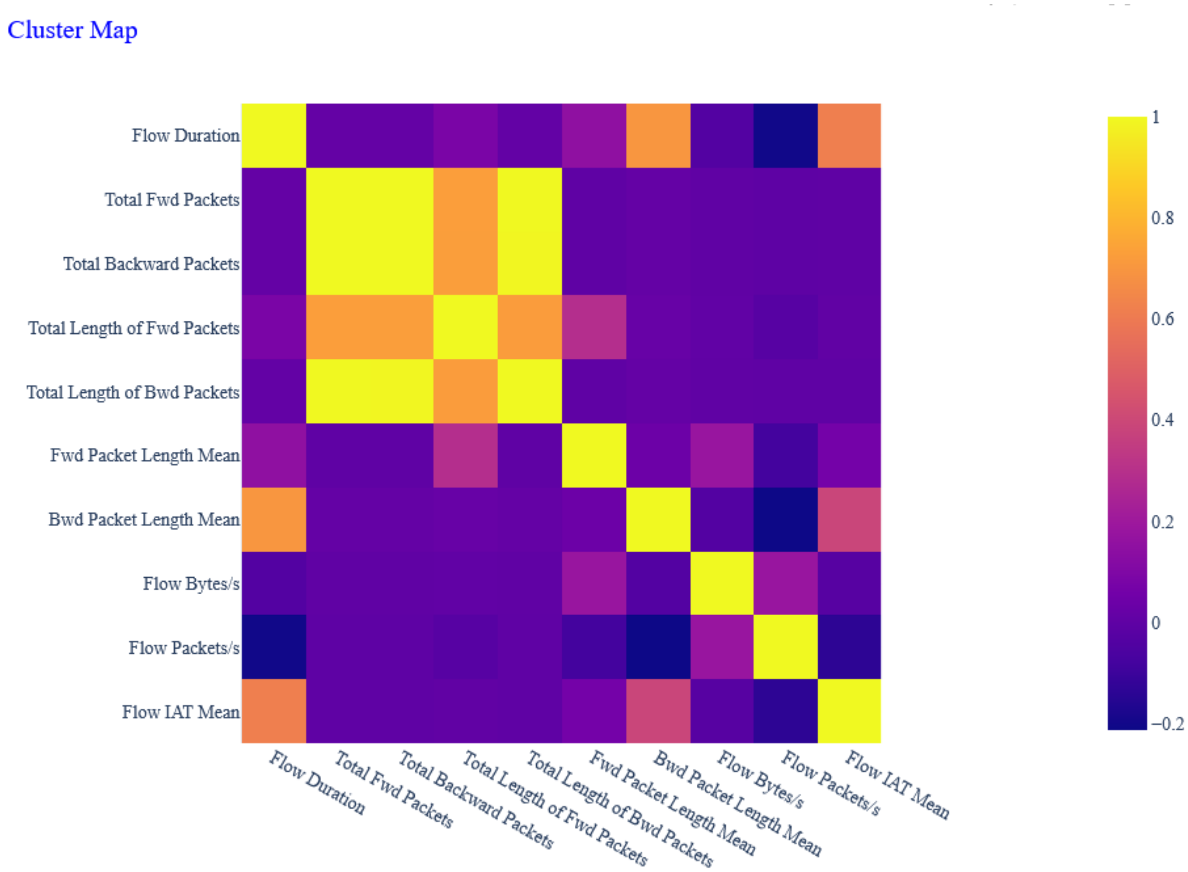


Figure 18: Clustered Pearson correlation heatmap for selected numeric features.

Several strong correlations stand out:

- `Total Backward Packets` and `Total Length of Bwd Packets`
- `Fwd Packets/s` and `Flow Packets/s`
- Various packet-length statistics that are trivially related (mean vs. variance).

To make this more explicit, I summarize the main correlation groups in a small table.

Table 3: Summary of main correlation groups observed in the Pearson heatmap.

Group	**Typical Features**	**Interpretation**
Backward volume	Total Backward Packets, Total Length of Bwd Packets	Counting packets and summing their lengths basically measure the same backward-flow volume.
Packet rate	Fwd Packets/s, Flow Packets/s, Bwd Packets/s	Direction-specific packet-per-second counts track the overall flow packet rate very closely.
Packet size stats	Fwd/Bwd Packet Length Mean, Packet Length Variance, Average Packet Size	All of these are derived from the same packet-length distribution, so they show up as one tight correlation cluster.

Observation: Forward and backward packet counts, byte counts, and packet-length statistics form tight correlation clusters. For any future modeling step, I could safely drop one member of each highly correlated pair to cut down on redundancy without losing much information. From a visualization standpoint, the heatmap plus Table 3 together make it very clear that some parts of the feature set are basically different views of the same physical signal.

9 Statistics

This section evaluates the statistical characteristics of the cleaned dataset. The focus is on distribution shapes, skewness, modality, and bivariate structure. Flow Duration, Flow Bytes/s, Flow Packets/s, and Flow IAT Mean consistently show heavy-tailed behavior and clear multi–regime patterns that align with the attack phases inside the CIC-IDS2017 Wednesday-WorkingHours file.

Most of the remaining plots in the analysis section use the feature values after the selected dashboard transformations (e.g., `log1p` or standard scaling). The purpose of these plots is comparative visualization, so the exact transformation is chosen only to make the structure easier to see. All baseline distribution analysis is performed on the raw values.

9.1 Univariate Distribution Analysis

Before applying PCA, normality tests, or any of the dashboard-level transformations, I examined the raw (untransformed) distributions of two high-impact features: `Flow Duration` and `Flow Bytes/s`. Both variables directly drive the separation between benign traffic and the different DoS behaviors, so understanding the baseline distribution is important.

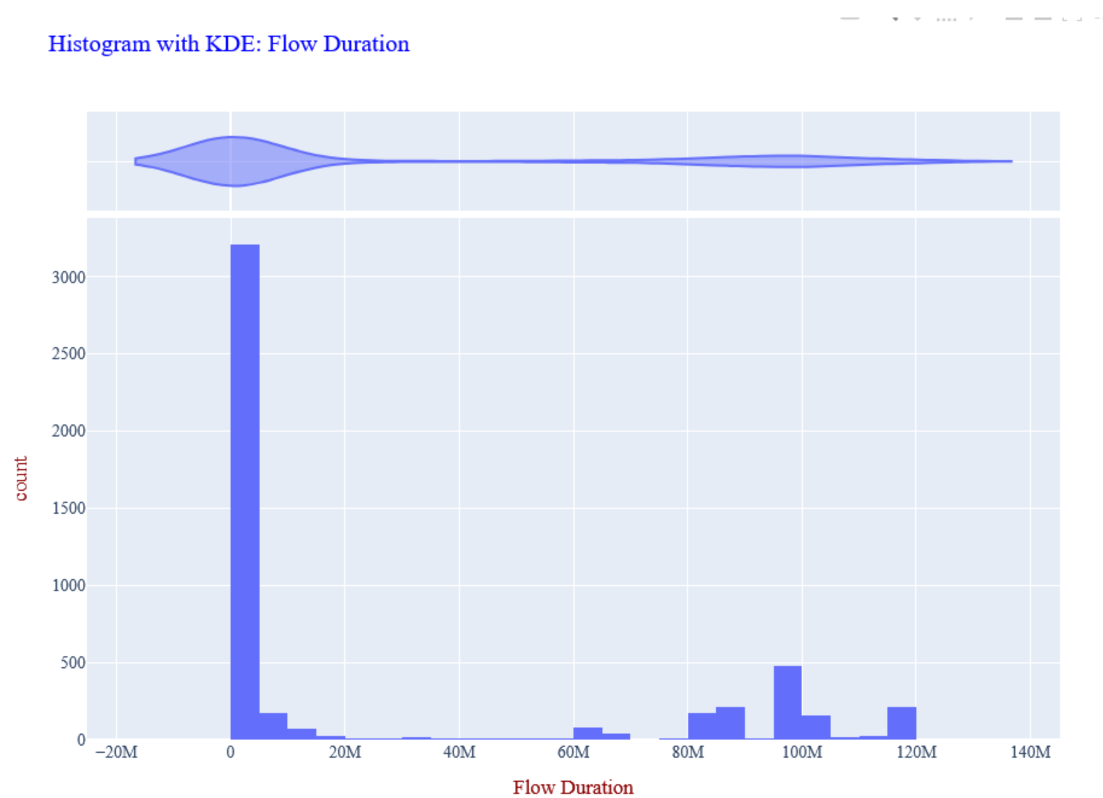


Figure 19: Histogram + KDE of raw Flow Duration.

The distribution of `Flow Duration` is extremely heavy-tailed. Most flows terminate very quickly, while attack-driven flows stretch into the tens of millions of microseconds. This kind of long right tail explains why Shapiro–Wilk, K–S, and D'Agostino tests consistently reject normality. The raw histogram also shows multiple local modes that correspond to specific attack intervals, which aligns with the earlier timeline analysis.

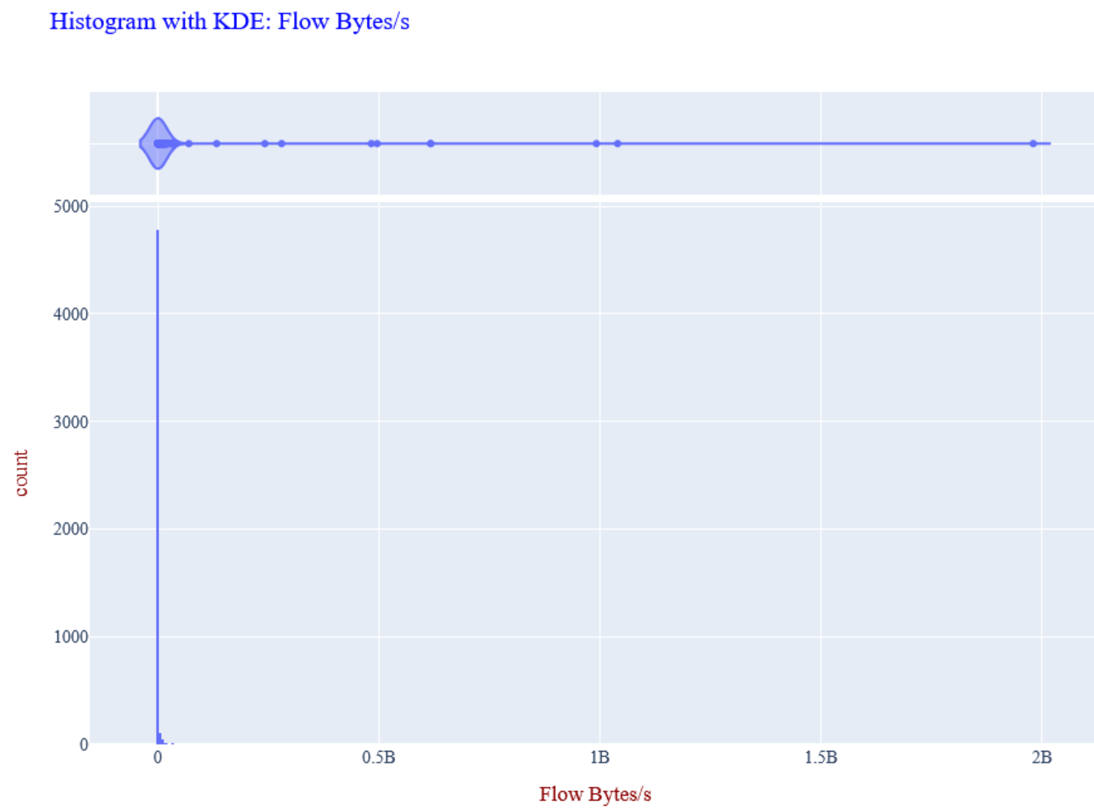


Figure 20: Histogram + KDE of raw Flow Bytes/s.

`Flow Bytes/s` exhibits an even more extreme concentration near zero, followed by sudden spikes into the hundreds of millions. This is typical for high-volume floods such as Hulk and GoldenEye, which produce bursts that massively outweigh benign background traffic. The KDE curve collapses toward the lower end of the domain, and the histogram bins near zero dominate the entire distribution.

Note on transformations. All histograms above intentionally use raw (untransformed) values. The dashboard includes optional `log1p`, `minmax`, and `standard` transformations, which substantially compress the heavy tails and make the structure more visible. For the purpose of documenting the true shape of the CIC-IDS2017 Wednesday file, the raw plots are more informative and help justify why transformations and outlier handling are necessary in later sections.

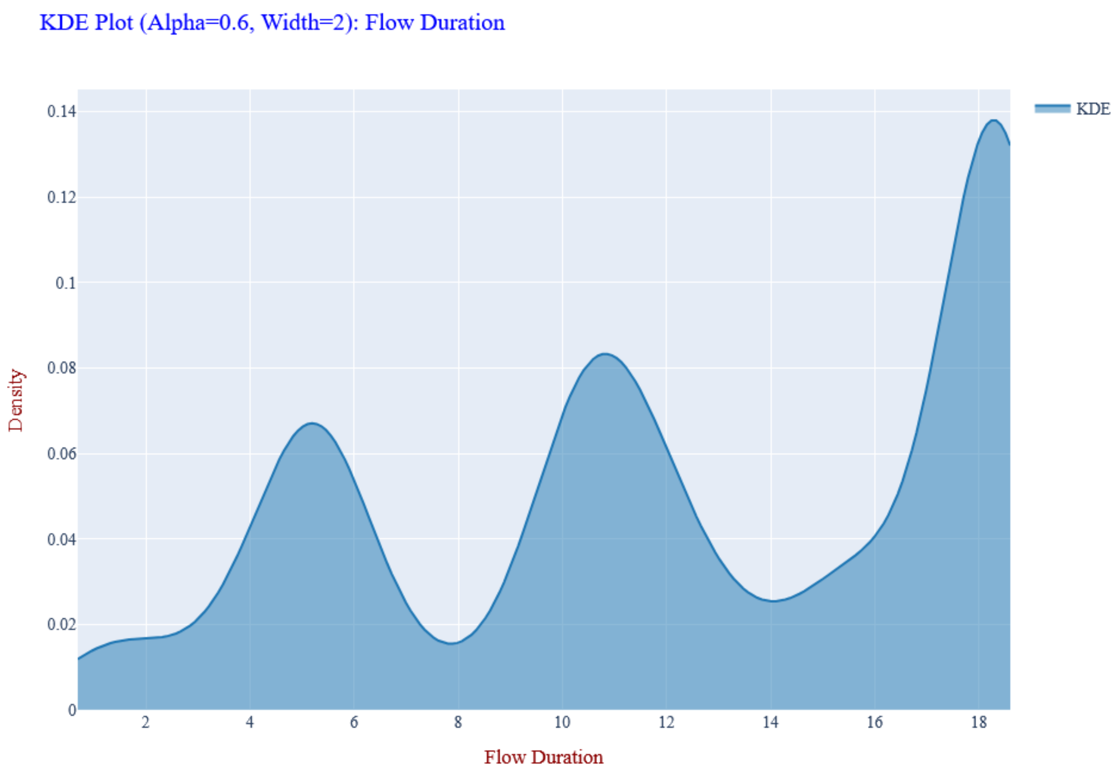


Figure 21: KDE-only visualization of Flow Duration (log1p applied).

Observation: This KDE view removes the histogram bars and isolates the smoothed density curve. After applying `log1p`, the multi–modal structure becomes much easier to see. The long upper tail compresses, and the peaks line up with the attack windows observed earlier in the raw distribution. Slowloris-/SlowHTTPTest-style flows stay grouped in the long-duration low-throughput region, while Hulk-like bursts remain closer to the short-duration peak.

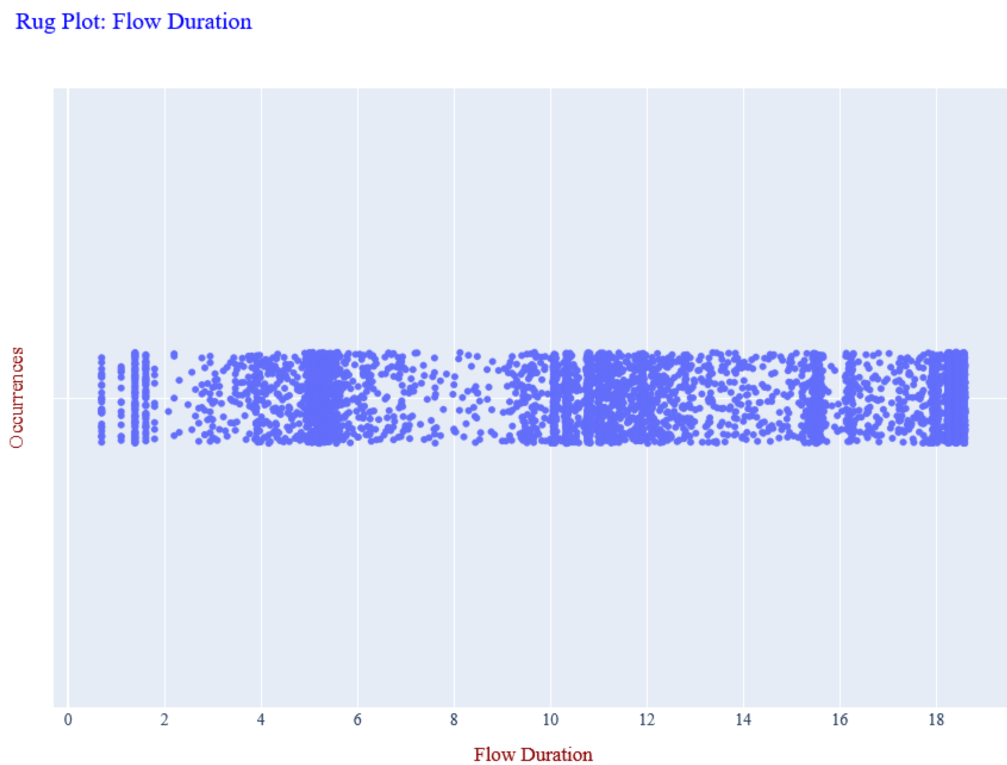


Figure 22: Rug plot for Flow Duration (log1p applied).

Observation: The rug plot shows the actual sample-level density along the axis. Even with `log1p`, flows form tight clusters that correspond to specific attack phases. This confirms that the distribution is not only heavy-tailed but also structured into distinct behavioral regimes, which is exactly what later drives the separation in PCA.

9.2 Bivariate and Multivariate Structure

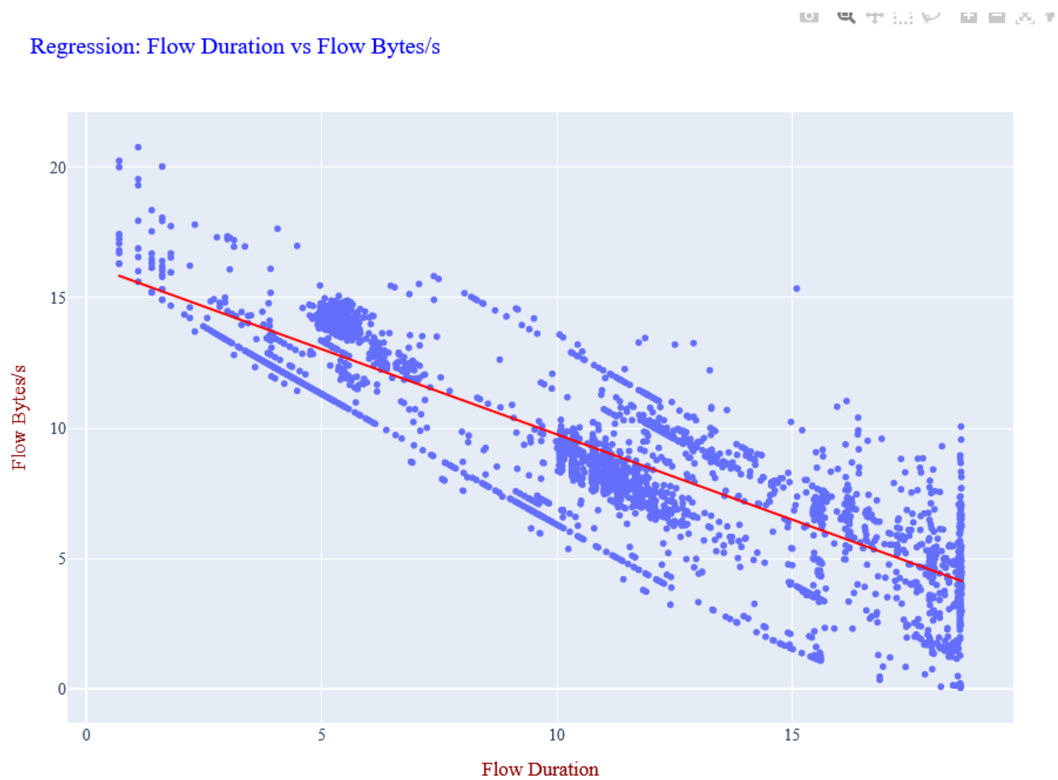


Figure 23: Regression plot of Flow Duration vs. Flow Bytes/s.

Observation: The fitted trend line slopes downward. Slowloris-style flows occupy the long-duration/low-throughput region, while Hulk floods create short, high-throughput spikes. The negative pattern is stable across subsamples and is visible even without transformations.

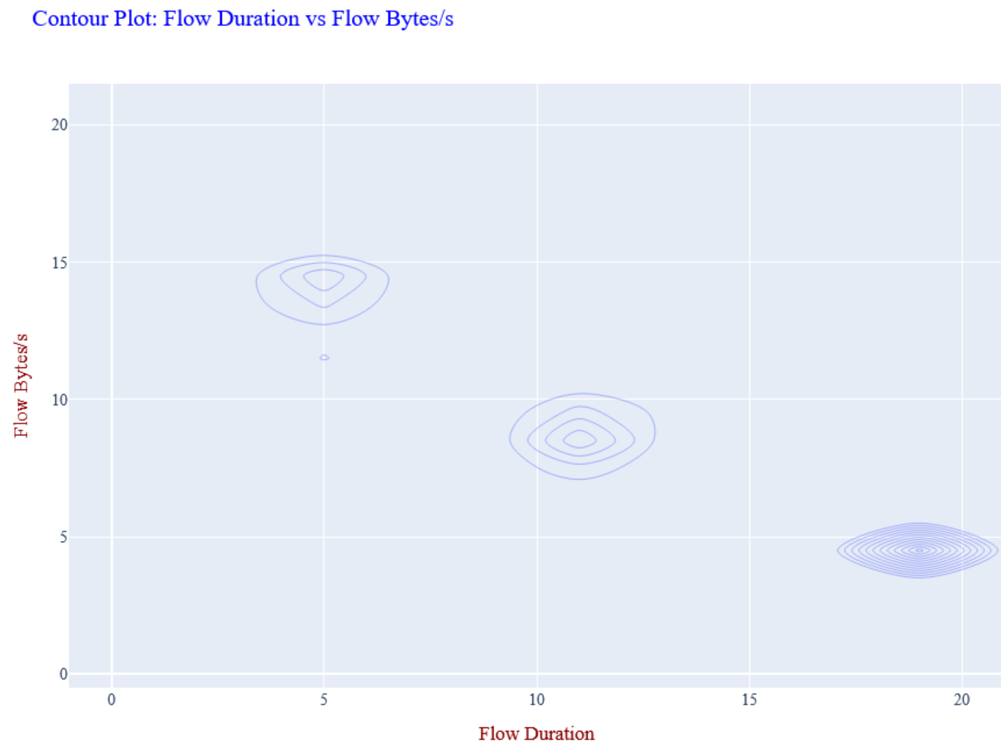


Figure 24: 2D KDE-style contour plot of Flow Duration and Flow Bytes/s.

Observation: The contour levels reveal three main density islands, corresponding to major behavior classes. Short, high-throughput flows form a compact ridge, mid-range flows cluster in a central zone, and very long, low-throughput connections remain isolated near the edge of the distribution.

Multivariate KDE (Joint Density Estimate)

The 2D KDE shown in Figure 25 is the multivariate kernel density estimate required in the statistical analysis. Instead of looking at each feature independently, this KDE models the joint density of `Flow Duration` and `Flow Bytes/s`, which are two of the strongest discriminative features in this dataset. This allows us to see how the traffic types actually cluster in the combined feature space.

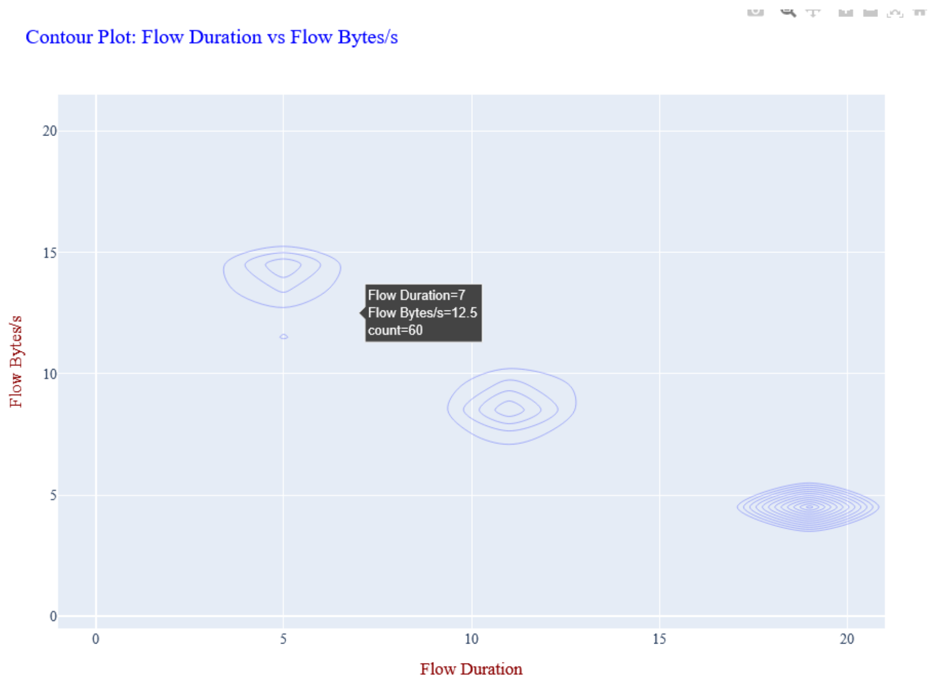


Figure 25: Multivariate (2D) KDE of Flow Duration and Flow Bytes/s.

Observation: The multivariate density surface shows that benign flows form a compact high-density region at shorter durations and moderate byte rates, while attack flows occupy entirely different areas. Hulk traffic generates a wide, elevated ridge at extremely high byte rates, whereas Slowloris and SlowHTTPTest concentrate along a low-byte, long-duration axis. These density "islands" match the separation observed in PCA and confirm that the dataset is not just non-Gaussian but also deeply multi-modal across multiple dimensions.

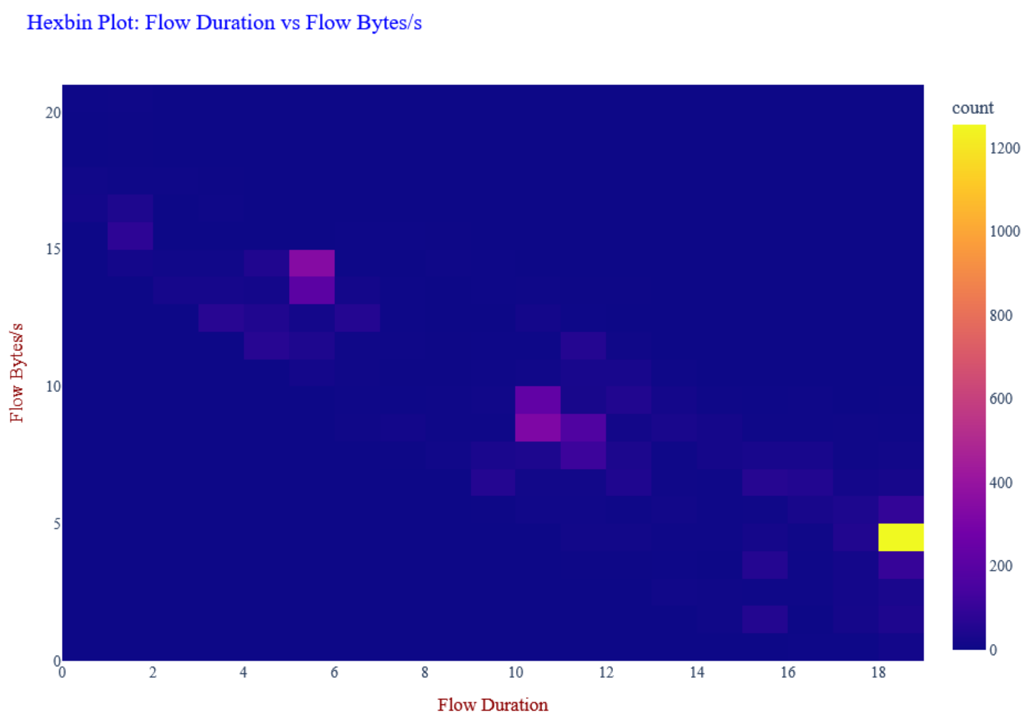


Figure 26: Hexbin density map of Flow Duration vs. Flow Bytes/s.

Observation: Hexbin counts emphasize how frequently flows accumulate in the short-duration middle-throughput band. Sparse but structured clusters appear for long-duration flows, consistent with slow-and-stuck header attacks.

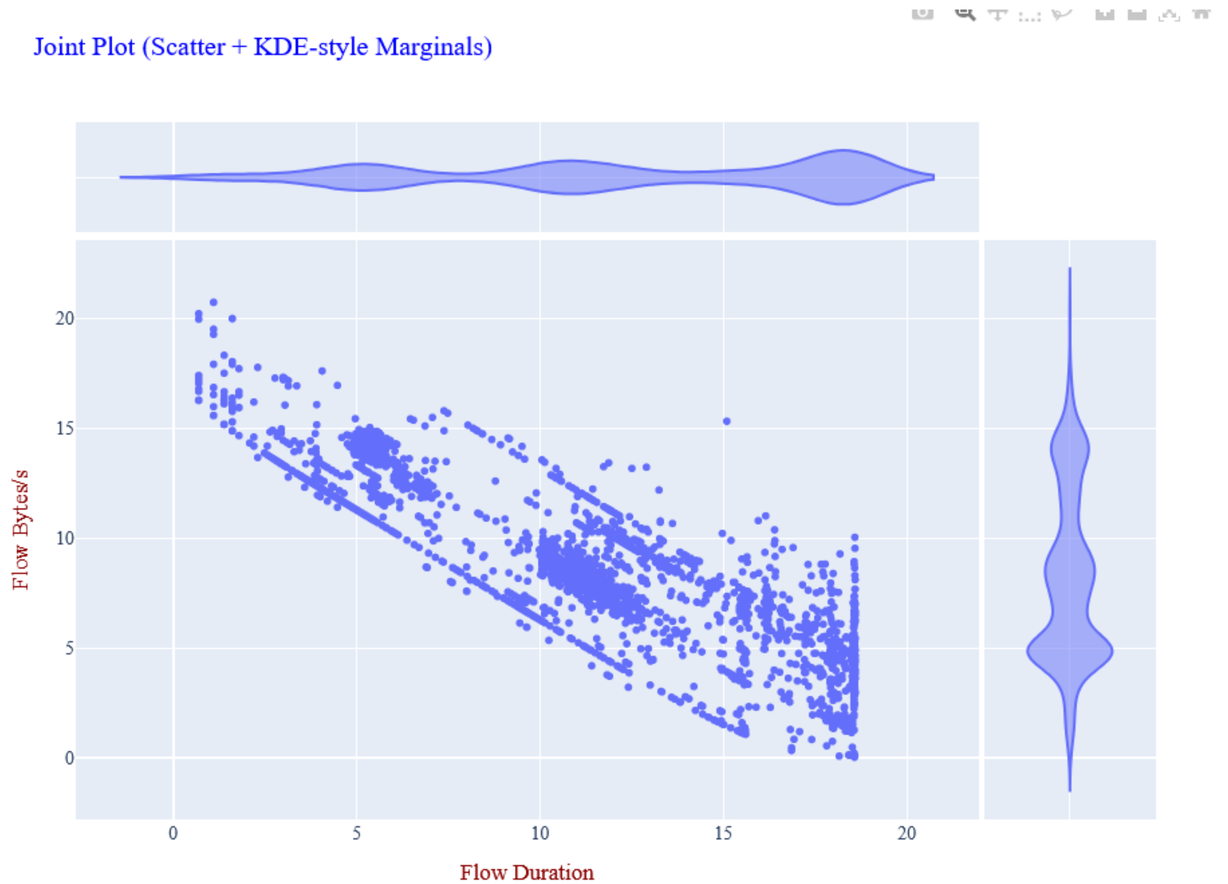


Figure 27: Joint plot with KDE marginals for Flow Duration and Flow Bytes/s.

Observation: The central scatter recreates the negative trend, while the marginal KDEs illustrate asymmetry on both axes. Duration has multiple modes, whereas byte rate is far more concentrated, especially for attack traffic.

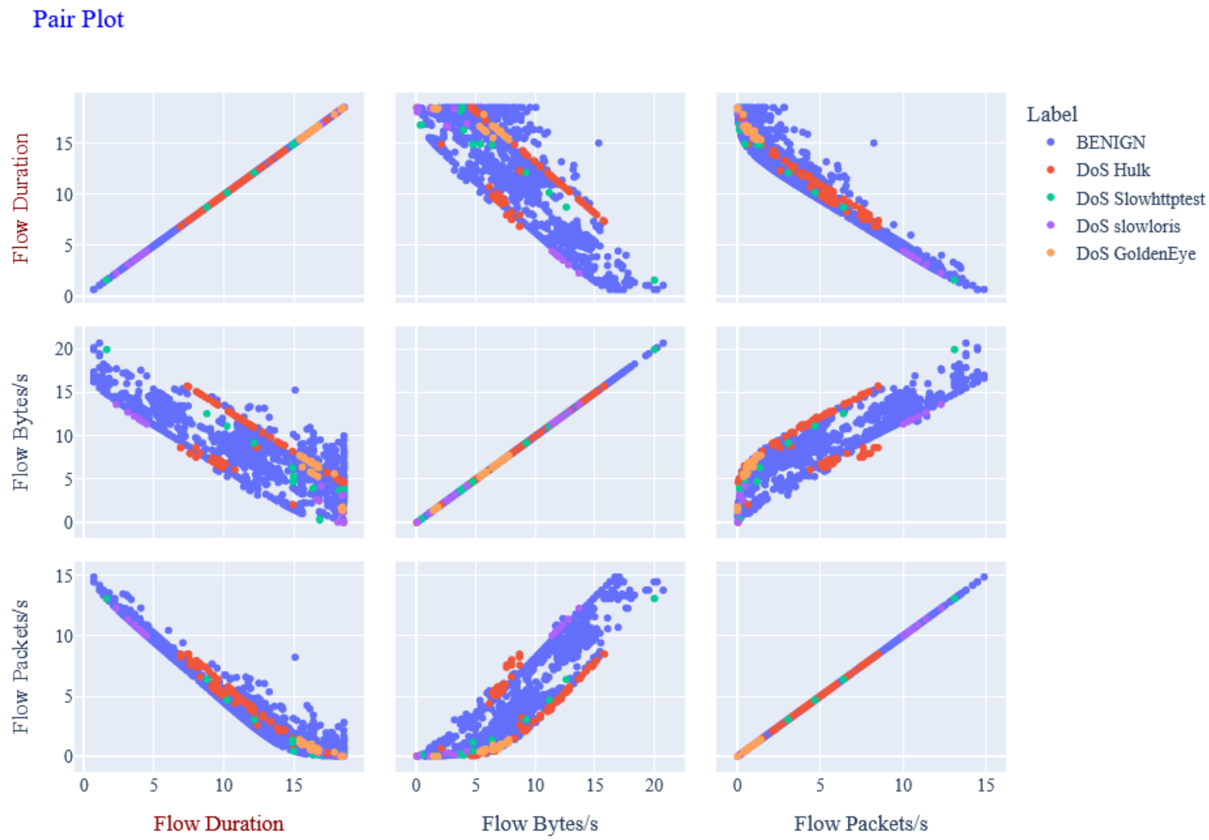


Figure 28: Pair plot for Duration, Bytes/s, Packets/s, and IAT Mean.

Observation: Attack families trace different diagonal paths in each 2D projection. Hulk shows dense, high-throughput diagonals; Slowloris and Slowhttptest anchor to long-duration, low-rate zones. Benign traffic remains dispersed, covering the central region.

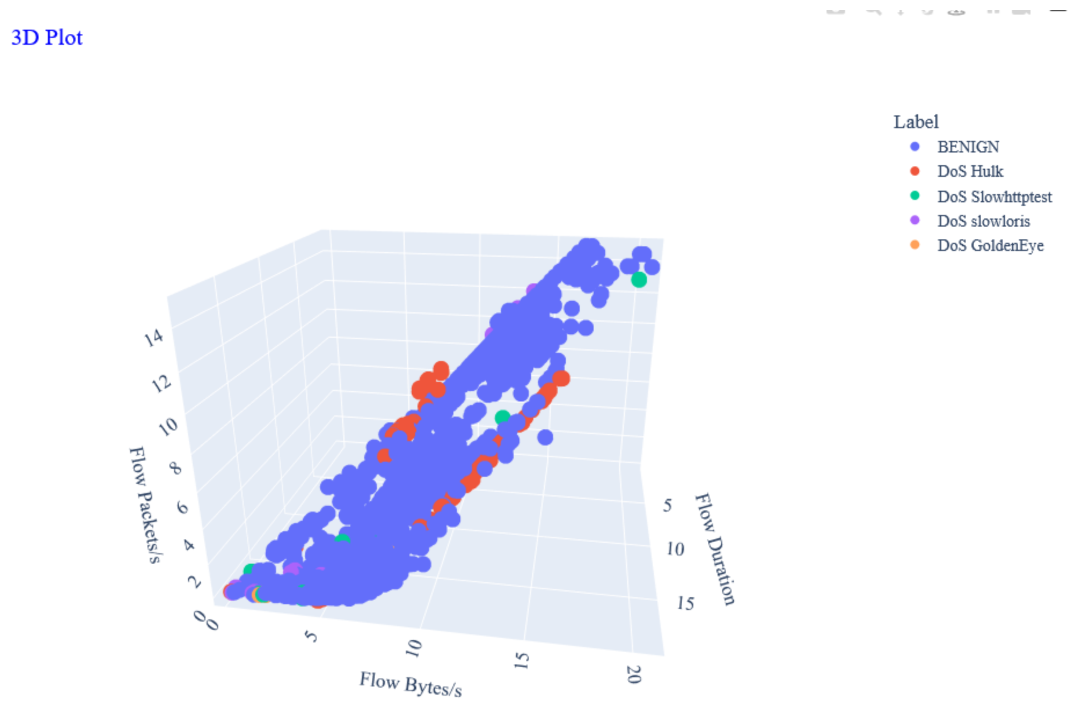


Figure 29: 3D scatter plot using Flow Duration, Flow Bytes/s, and Flow Packets/s.

Observation: In 3D space, class separation becomes even clearer. Volumetric floods (Hulk, GoldenEye) cluster in the high-packets/high-bytes corner, while header-based slow attacks occupy a stretched ridge defined by long durations but very little activity per unit time.

10 Data Visualization: Numeric and Categorical Plots

This section summarizes the static examples used to verify the dashboard outputs. The goal is to demonstrate that the dashboard not only renders interactive views but also produces consistent, interpretable plots that match offline statistical analysis.

10.1 Numeric Plots

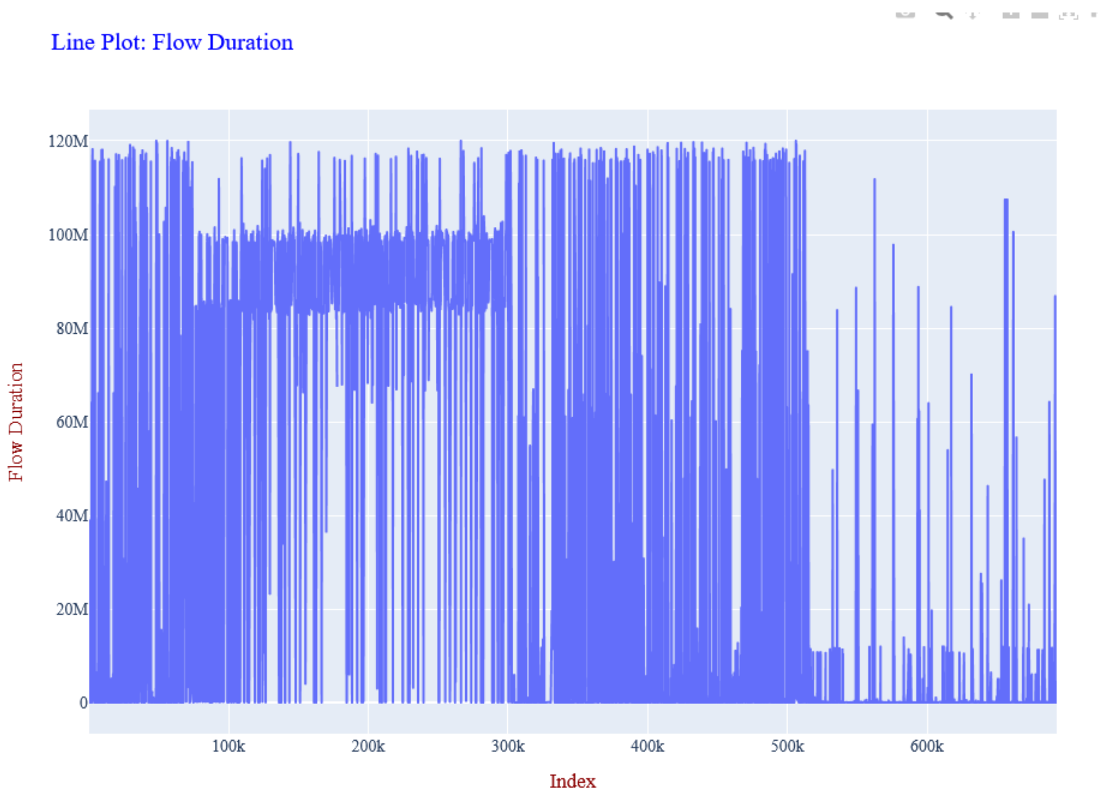


Figure 30: Line Plot of Flow Duration across row index.

Observation: The raw line plot immediately shows why this feature is so hard to interpret without preprocessing. Flow Duration jumps between extremely short and extremely long flows, and the dataset is large enough that the plot basically becomes a dense block. This reflects the mixed structure of the Wednesday file—benign flows, several DoS attacks, and large spikes during attack windows. A line plot isn't the best way to understand this feature, but it clearly shows how skewed and unstable the raw values are before applying any cleaning, outlier filtering, or transformations.

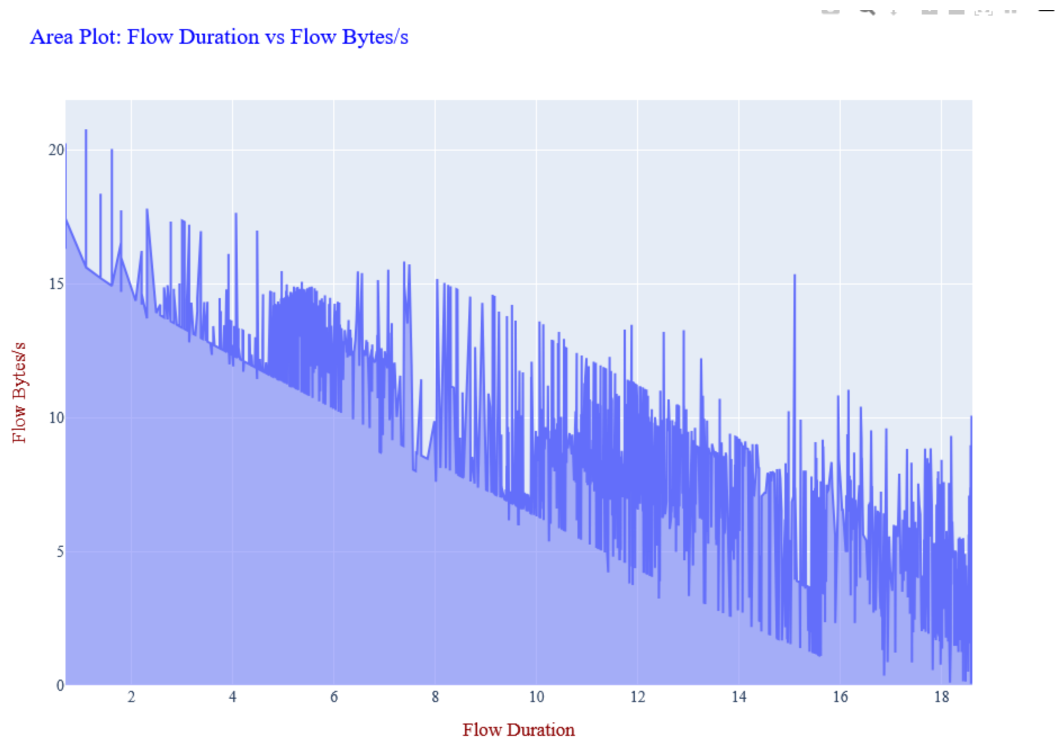


Figure 31: Area plot of attack counts across row index (time order).

Observation: The stacked area plot shows distinct blocks where DoS traffic dominates successive rows. These match the scripted attack windows inside the dataset. Outside these regions, benign traffic remains the majority.

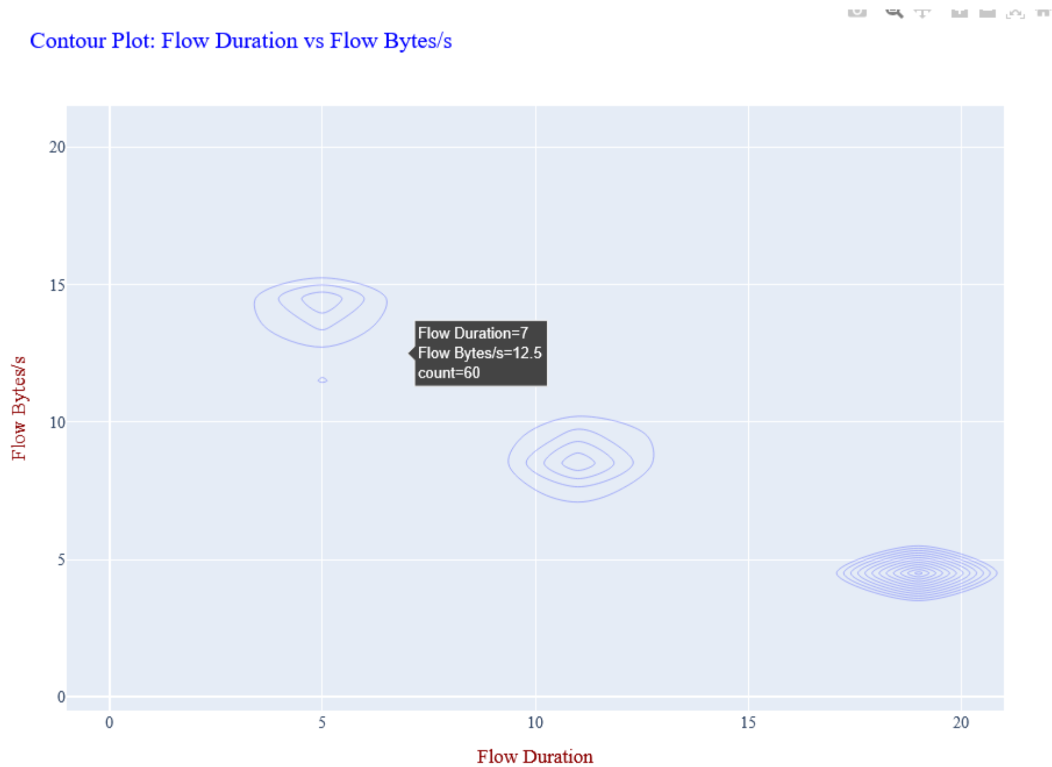


Figure 32: 2D KDE density visualization for Flow Duration and Flow Bytes/s.

Observation: The smoothed KDE surface highlights ridge structures that the hexbin plot only hints at. These ridges align with the same multi-mode behavior found in the univariate and joint plots.

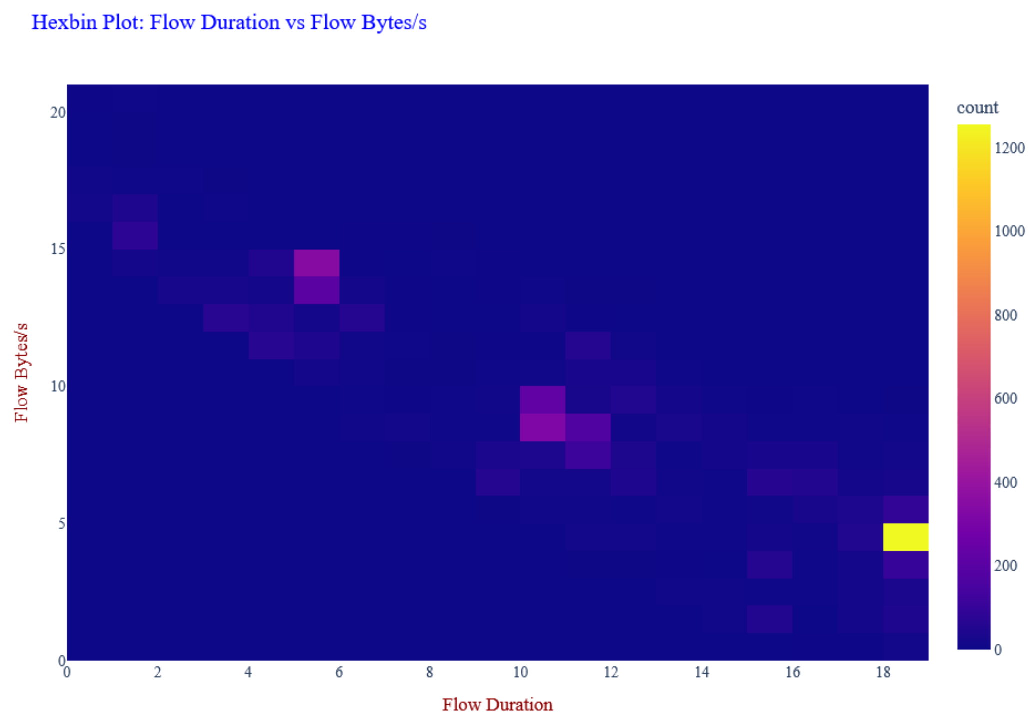


Figure 33: Hexbin visualization (numeric tab example).

Observation: The dashboard and static hexbin outputs match closely, confirming that the in-app rendering preserves the distribution geometry seen in the offline figures.

10.2 Categorical Plots

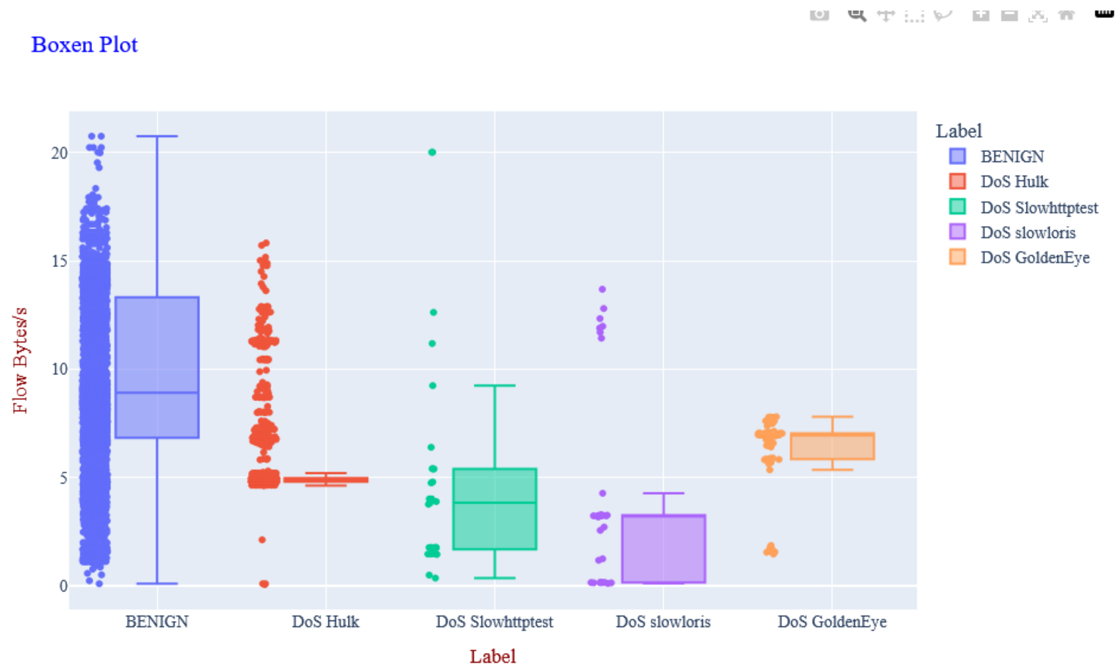


Figure 34: Boxen plot of Flow Bytes/s grouped by label.

Observation: Hulk and GoldenEye flows show compact, high-throughput bodies, while Slowloris and Slowhttptest collapse near zero. The distribution shape shifts dramatically across labels, making Flow Bytes/s one of the strongest discriminators.

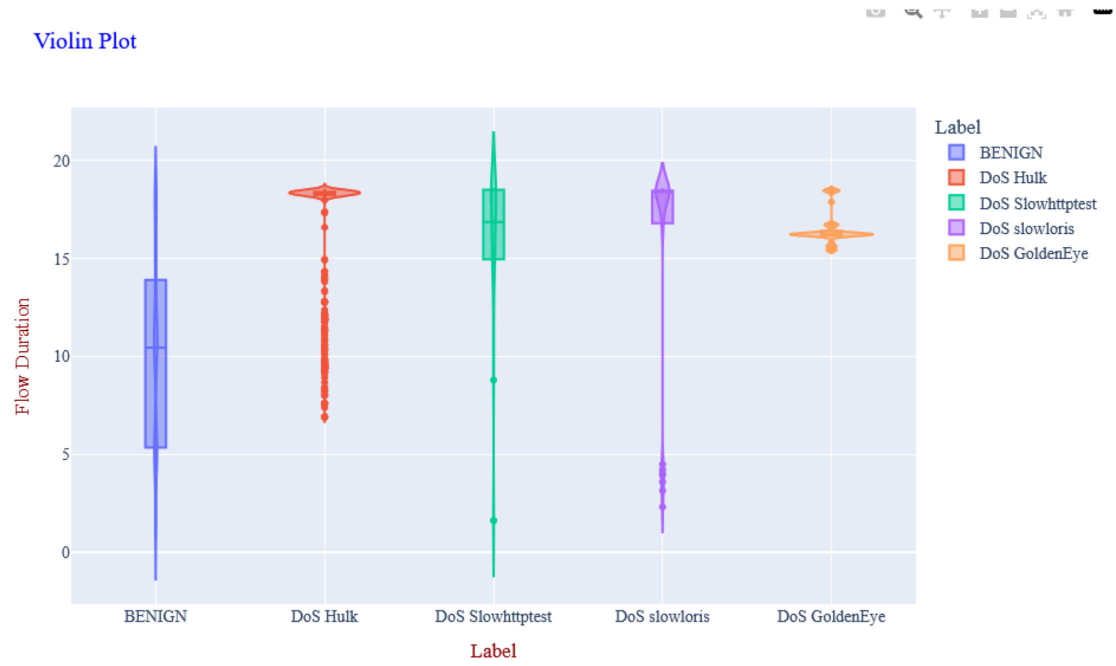


Figure 35: Violin plot of Flow Duration by label.

Observation: Slowloris and Slowhttptest exhibit narrow but very elongated duration distributions. Benign and Hulk flows appear tighter and occupy mid-range durations, mirroring the heavy-tailed nature highlighted in the statistics section.

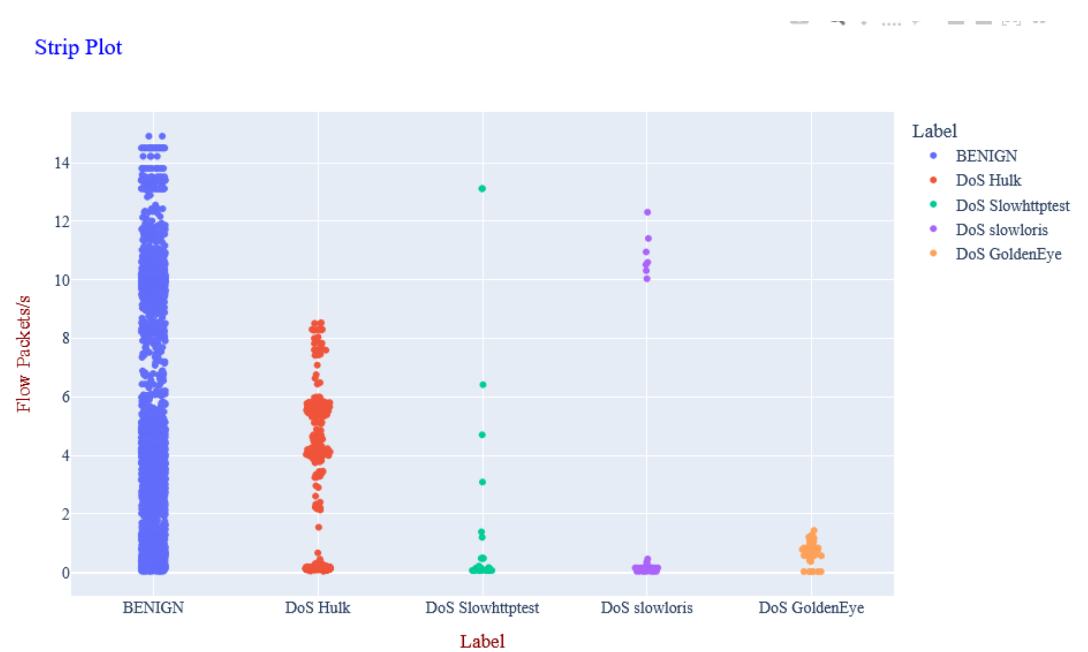


Figure 36: Strip plot of Flow Packets/s grouped by label.

Observation: The strip plot clearly separates the extremes even without bins or violins. Hulk occupies the high-packet region, Slowloris sits nearly at zero, and benign flows fill the middle band.

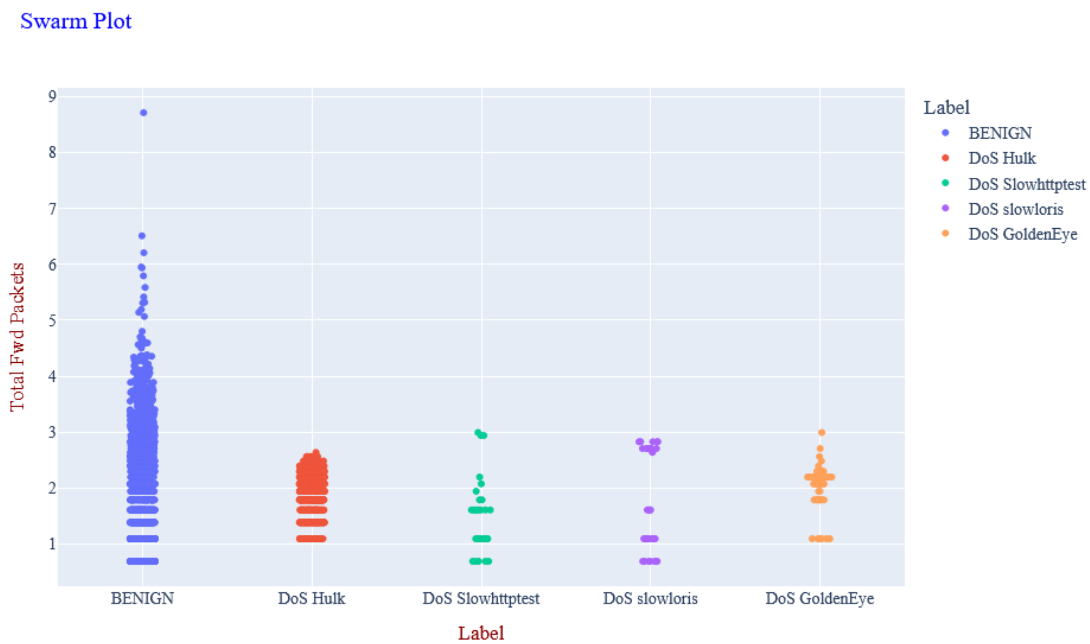


Figure 37: Swarm plot of Total Forward Packets vs. label.

Observation: The swarm layout reveals discrete packet-count bands per attack type. It avoids overplotting and makes the categorical structure more interpretable for users who prefer simple, discrete visualizations over dense histograms.

11 Subplots

The Storytelling tab uses a static four-panel subplot layout to summarize the key story of the dataset. All panels still respect the preprocessing settings selected at the top.

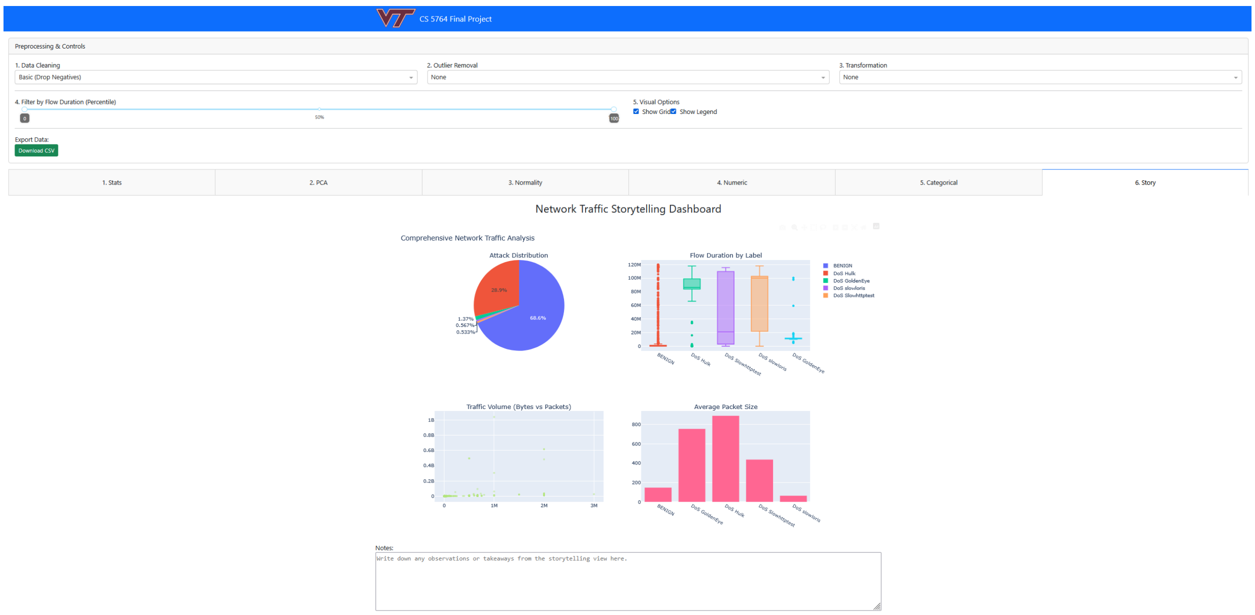


Figure 38: Storytelling dashboard tab with four linked subplots.

The layout is:

1. Top-left: Attack distribution pie chart.
2. Top-right: Box plot of Flow Duration by `Label`.
3. Bottom-left: Scatter of Flow Bytes vs. Flow Packets (color by label).
4. Bottom-right: Bar plot of average packet size per label.

Observation: The pie chart recaps the class imbalance. Immediately to the right, the Flow Duration box plot shows that Slowloris and Slowhttptest have far higher median durations and wider IQRs than benign traffic, which fits their behavior as connection-exhaustion attacks. The bottom-left scatter shows that Hulk and GoldenEye produce flows with huge byte and packet counts—these are volumetric floods. Finally, the average packet-size bars show that some attacks (like Hulk) favor larger packets, whereas others rely on many small packets. Put together, the four subplots give a compact visual explanation a SOC analyst could use in a meeting to explain why these attacks look different in the flow data.

12 Tables

The main tables in this project are:

- Descriptive statistics tables for numeric features (shown in the Data & Stats tab).
- The class distribution table in Section 2.
- Implicit tables within the dashboard, such as grouped means for categorical bar plots.

Observation: Compared to the plots, the tables are less visually intuitive but still necessary when I want exact values (for example, checking that the 75th percentile of Flow Duration changes after outlier removal). They also provide a more traditional view that some stakeholders prefer over graphs.

13 Dashboard

The final product of Phase III is a Plotly Dash application with the following structure:

- A global **Preprocessing & Controls** bar at the top:
  - Data Cleaning method (Keep Original / Basic / Mean Imputation / Strict).
  - Outlier Removal method (None / IQR / Z-score).
  - Transformation (None / log1p / MinMax / Standard).
  - Flow Duration percentile filter slider.
  - Visual options (grid on/off, legend on/off).
  - CSV export button to download the processed dataset.
- Six tabs:
  1. Data & Stats
  2. PCA
  3. Normality Tests
  4. Numeric Plots
  5. Categorical Plots
  6. Storytelling

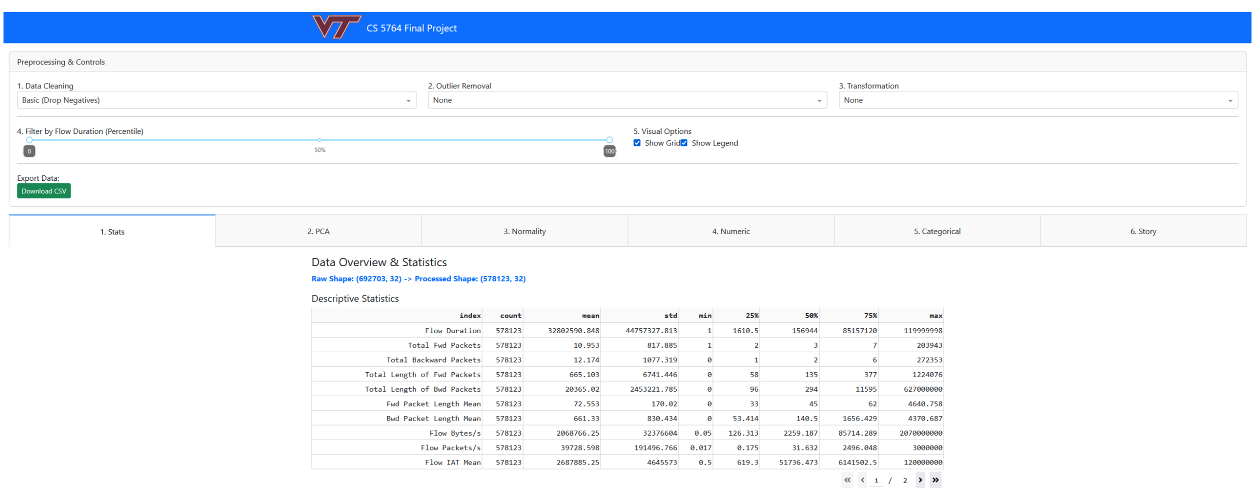


Figure 39: Dashboard main interface with preprocessing controls and Data & Stats tab.

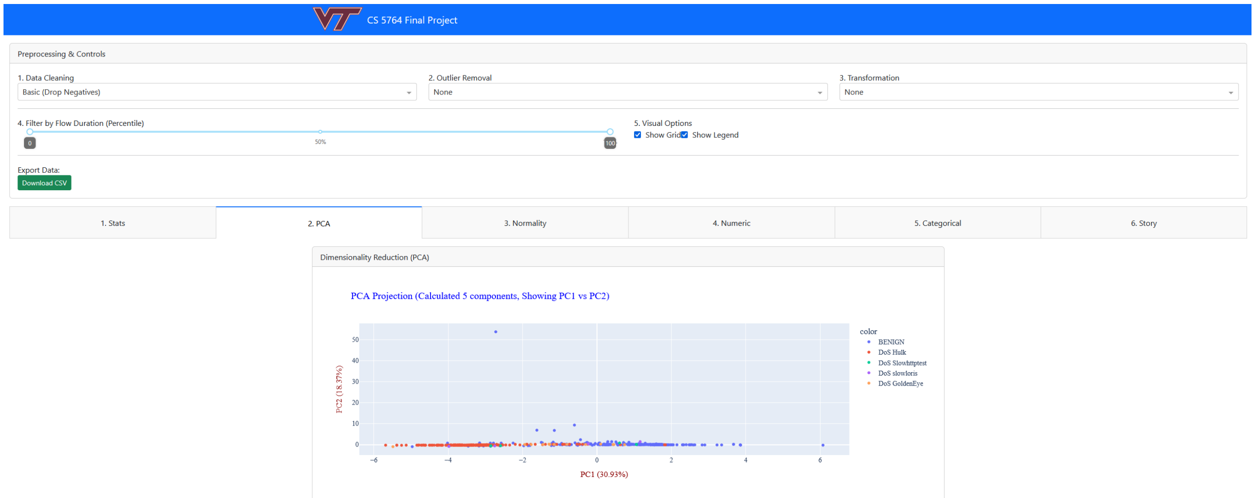


Figure 40: PCA tab showing PC1 vs. PC2 colored by label.

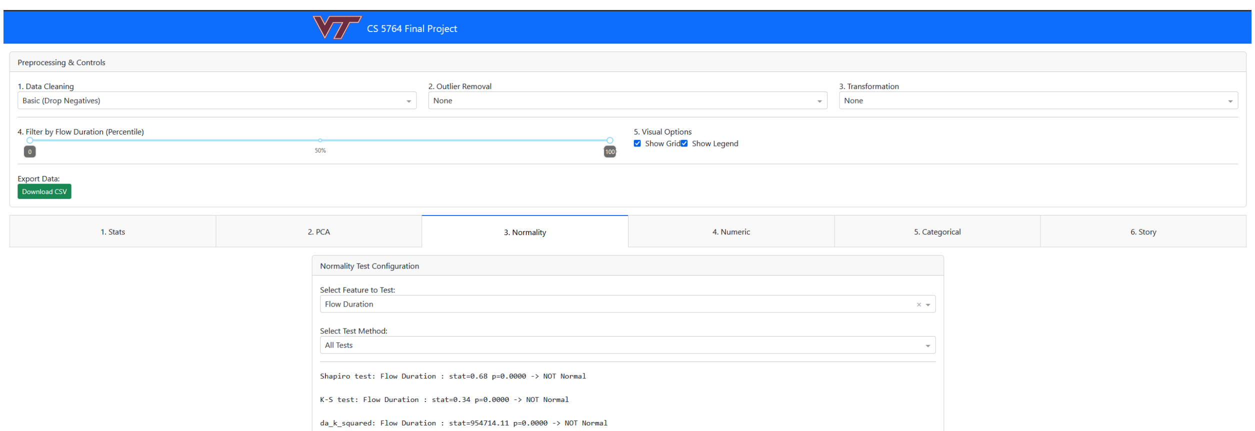


Figure 41: Normality tab: configuration and test outputs.

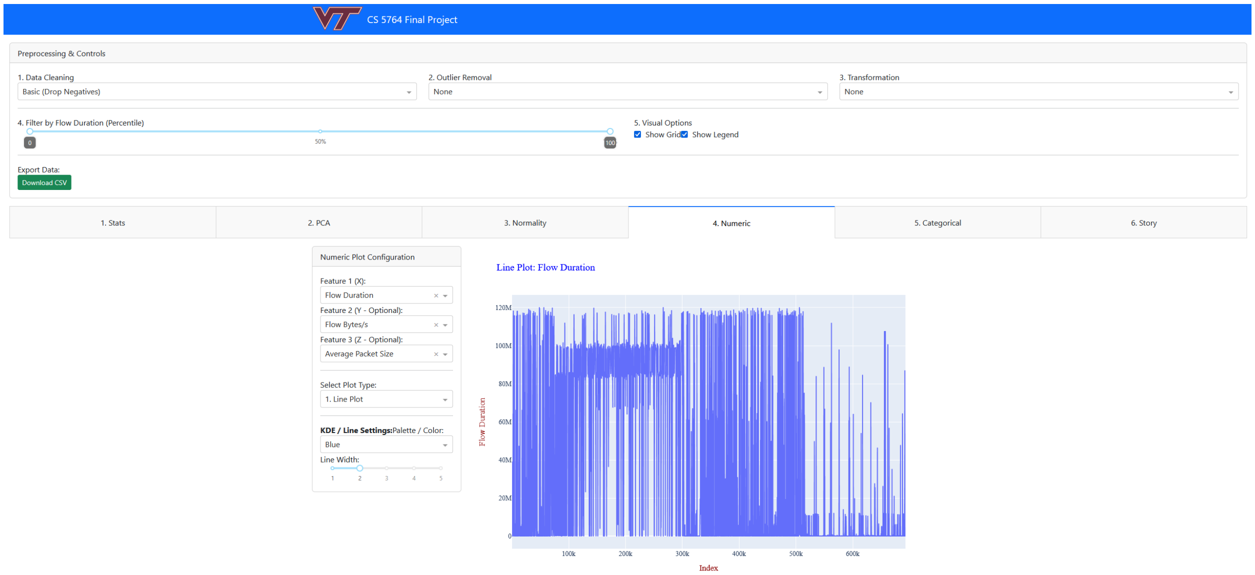


Figure 42: Numeric Plot tab.

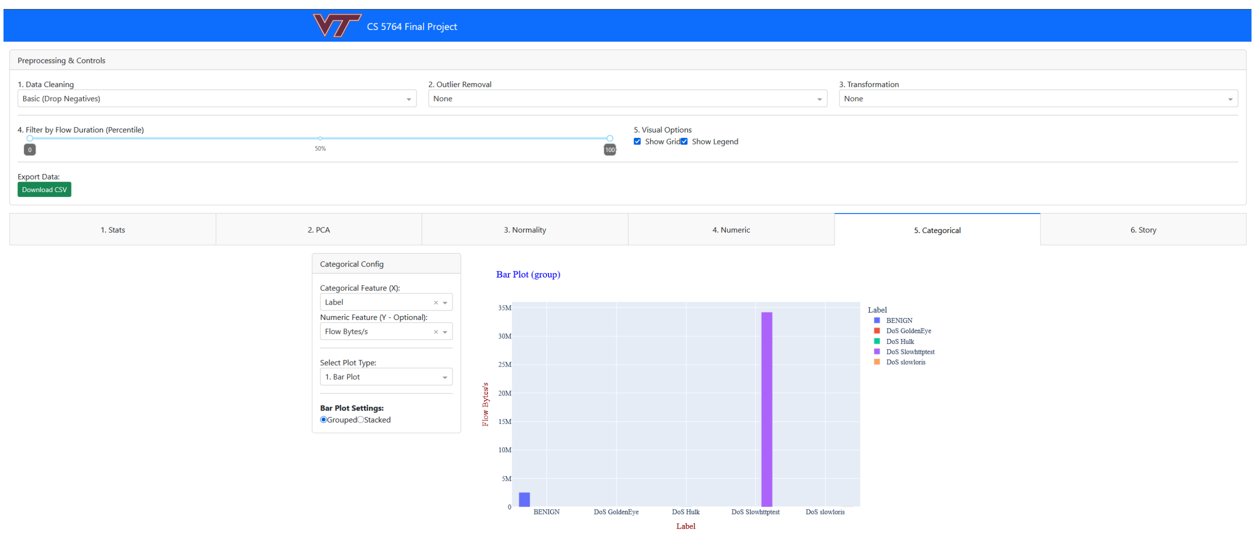


Figure 43: Categorical Plot tab.

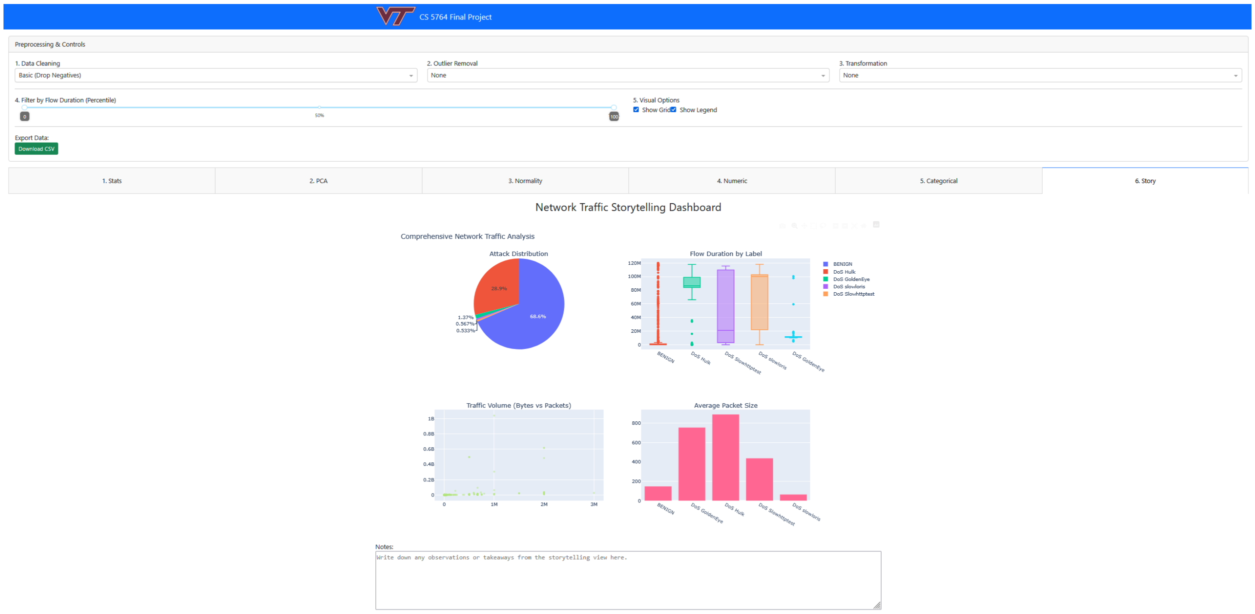


Figure 44: Storytelling tab.

Dashboard description (functionality): Every time the user touches a control—changes the cleaning method, switches from IQR to Z-score, slides the percentile filter, or picks a new feature—the corresponding callback recomputes the processed dataframe through a single helper function and updates the visible figure. This keeps the logic consistent across tabs. The user never has to manually re-run code; the app always shows the state of the pipeline implied by the UI.

Technical Implementation (Callback Logic). The dashboard follows Dash's callback model, meaning every control in the top panel (cleaning, outliers, transformations, percentile filters) becomes an input that forces the entire preprocessing pipeline to recompute. This is essentially a reactive setup: the dataset is always shown in the exact state implied by the UI. No cached shortcuts, no partial updates. The user interacts once, and all dependent views update immediately.

Plotly Usage. For most figures I rely on Plotly Express because it keeps the code simple and makes quick exploration easier. Wherever layout control actually matters (Storytelling tab, 3D plots, custom subplots), I switch to Graph Objects. This split keeps the code readable but still lets me build the more structured visual outputs when necessary.

14 Observations Summary

The FTP instructions emphasize that plots without observations receive zero points, so I summarize the main lessons from the entire pipeline here:

- **Class imbalance is extreme.** BENIGN and DoS Hulk dominate the dataset. Any modeling approach without rebalancing will mostly learn to distinguish “Hulk vs. everything else”.
- **Normality clearly does not hold.** Flow durations and byte/packet rates fail all three normality tests with p-values essentially zero. Heavy tails and multi-modal distributions are the norm, not the exception.
- **Preprocessing radically changes the picture.** Basic cleaning plus IQR outlier removal is enough to move the PCA from a blob dominated by outliers to a more interpretable structure where different attacks occupy different neighborhoods.
- **Correlation structure is strong.** Many features carry overlapping information. Future work could aggressively reduce dimensionality by dropping redundant columns before PCA or modeling.
- **Different DoS families have different signatures.** Slowloris/Slowhttptest focus on long-lived low-rate connections, Hulk/GoldenEye on high-volume bursts, and Heartbleed hardly appears at all. The dashboard makes these differences visible without touching any ML model.

15 Conclusion

(a) What did I learn from the graphs?

The main lesson is that network traffic is messy, skewed, and often dominated by a few behavior modes. Without cleaning, outlier control, and transformations, many standard tools (especially PCA and normality tests) produce misleading or uninterpretable results. Once I clean and transform the data, the graphs reveal clear patterns: which attacks extend flow durations, which ones saturate bandwidth, and how those behaviors interact with port usage and packet sizes.

(b) How does the dashboard help users?

Instead of hard-coding a single preprocessing pipeline, the dashboard lets the user toggle cleaning methods, outlier strategies, and transformations on the fly, then immediately see the effect on PCA, distributions, and correlation. For a SOC analyst, this is the difference between staring at static PDF plots and actually probing the dataset until it reveals something interesting. The combination of tabs covers both low-level statistics and high-level storytelling.

(c) Is the dashboard user friendly?

I shared the running dashboard with an external user (my partner) to gather informal usability feedback. The goal was to get a non-technical perspective on whether the interface, layout, and interactions communicated the intended ideas.
User Comment (verbatim):

> "UI is simple, love the storytelling tab, easy to understand, but since the dataset is not familiar to me, it's hard to understand what graph is meaning what."

Interpretation: The feedback aligns with what I expected from a user who does not have prior exposure to intrusion–detection datasets. The storytelling tab communicates the most intuitively, which confirms its design goal. At the same time, the comment highlights that some plots require additional context when shown to non-security users. In future versions of the dashboard, adding hover–based explanations, short captions, or "why this graph matters" summaries would help bridge this gap without redesigning the plots themselves.

Overall, the feedback supports the current layout while pointing out a reasonable direction for improvement: more built-in guidance for users who are unfamiliar with CIC-IDS2017 features.

I also posted the dashboard on social media to request feedback from professionals. However, no comments were received during the period in which the post was visible. I am including this note to document that the outreach step was completed, even though it did not result in external responses. The absence of feedback does not affect the internal evaluation of the dashboard, but it does reinforce the earlier point that users without prior exposure to intrusion–detection datasets often need additional context before engaging with the visualizations.

(d) Functionality

From a functionality standpoint, the app is not a full intrusion-detection system, but it is a solid analysis tool. It ties together every step of the FTP instructions—cleaning, outliers, PCA, normality, statistics, visualizations, and storytelling—into a single coherent pipeline. The callbacks are responsive, and the app handles a reasonably large sample of the dataset without freezing. If I wanted to evolve this into a more serious tool, the next steps would be: add time-based filtering, IP-based filtering, and integrate simple anomaly scores on top of the current visualizations.

(e) Future Work

The current dashboard stops at statistical analysis and PCA because that was the scope of the project. If I extend it later, the next step would be to integrate simple ML components directly into the same pipeline. Plotly already supports the visual side of this, so adding the following would be straightforward:

- ROC and PR curves for a basic classifier trained on the cleaned data.
- Confusion matrices to see where similar attack types get mixed.
- A lightweight anomaly detector (e.g., isolation forest) that runs on the processed dataframe and feeds into the visual tabs.

These additions would turn the dashboard from a pure analysis tool into an end-to-end exploration and modeling interface for CIC-IDS2017-style traffic.

16 Appendix: Python Code

Below is the complete Dash application source code used for this project. It includes data loading, preprocessing, outlier handling, PCA, normality tests, numeric/categorical plots, storytelling subplots, and all callbacks.

```python
# Created by Minjin Kim on 2025.11.30
# CS 5764     Final Term Project

import numpy as np
import pandas as pd
import scipy.stats as stats
from scipy.stats import shapiro, kstest, normaltest
import gc

import dash
from dash import dcc, html, dash_table
from dash.dependencies import Input, Output, State
import dash_bootstrap_components as dbc

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA

# ---------------------------------------------------------------------
# 1. Global Configuration & Data Loading
# ---------------------------------------------------------------------

pd.options.display.float_format = lambda x: f"{x:.2f}"
DATA_PATH = "Wednesday-workingHours.pcap_ISCX.csv"

NUMERIC_COLS = [
    "Flow Duration", "Total Fwd Packets", "Total Backward Packets",
    "Total Length of Fwd Packets", "Total Length of Bwd Packets",
    "Fwd Packet Length Mean", "Bwd Packet Length Mean",
    "Flow Bytes/s", "Flow Packets/s", "Flow IAT Mean",
    "Flow IAT Std", "Fwd Packets/s", "Bwd Packets/s",
    "Average Packet Size", "Packet Length Variance",
    "Active Mean", "Idle Mean"
]

```

```
CAT_COLS = [
    "Destination Port", "Protocol", "Fwd PSH Flags", "Bwd PSH Flags",
    "Fwd URG Flags", "Bwd URG Flags", "FIN Flag Count", "SYN Flag Count",
    "RST Flag Count", "PSH Flag Count", "ACK Flag Count", "URG Flag Count",
    "CWE Flag Count", "ECE Flag Count", "Label"
]

CONST_FLAG_COLS = ["Bwd PSH Flags", "Fwd URG Flags", "Bwd URG Flags", "CWE Flag Count"]
PORT_BUCKET_COL = "Destination Port Bucket"


def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024 ** 2
    print(f'Memory usage of dataframe is {start_mem:.2f} MB')

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                df[col] = df[col].astype(np.float32)
        else:
            num_unique_values = len(df[col].unique())
            num_total_values = len(df[col])
            if num_unique_values / num_total_values < 0.5:
                df[col] = df[col].astype('category')

```

```python
    end_mem = df.memory_usage().sum() / 1024 ** 2
    print(f'Memory usage after optimization is: {end_mem:.2f} MB')
    print(f'Decreased by {100 * (start_mem - end_mem) / start_mem:.1f}%')

    return df


def add_port_bucket_column(df: pd.DataFrame) -> pd.DataFrame:
    if "Destination Port" not in df.columns:
        return df
    port = pd.to_numeric(df["Destination Port"], errors="coerce")
    bins = [-1, 1023, 49151, 65535]
    labels = ["Well-known (0  1023  )", "Registered (1024  49151  )", "Dynamic (49152  65535  )"]

    df[PORT_BUCKET_COL] = pd.cut(port, bins=bins, labels=labels)
    df[PORT_BUCKET_COL] = df[PORT_BUCKET_COL].astype("category")
    return df


def load_raw_dataset(path: str) -> pd.DataFrame:

    df = pd.read_csv(path)
    df.columns = df.columns.str.strip()
    cols_present = [c for c in (NUMERIC_COLS + CAT_COLS) if c in df.columns]
    df = df[cols_present]

    df = reduce_mem_usage(df)

    df = add_port_bucket_column(df)

    return df


print("Loading and optimizing data...")
df_raw = load_raw_dataset(DATA_PATH)
print("Data loaded successfully.")

numeric_options = [{"label": c, "value": c} for c in NUMERIC_COLS if c in df_raw.columns]
cat_options = [{"label": c, "value": c} for c in (CAT_COLS + [PORT_BUCKET_COL])
               if c in df_raw.columns
```

```python
               and c != "Destination Port"
               and c not in CONST_FLAG_COLS]


# ------------------------------------------------------------------
# 2. Data Processing Logic
# ------------------------------------------------------------------

def clean_dataset(df: pd.DataFrame, method: str = "basic") -> pd.DataFrame:
    df_clean = df.copy()

    num_cols = df_clean.select_dtypes(include=[np.number]).columns
    df_clean[num_cols] = df_clean[num_cols].replace([np.inf, -np.inf], np.nan)

    if method in ("basic", "strict"):
        cols_nonpositive_to_nan = ["Flow Duration", "Flow IAT Mean", "Flow Bytes/s", "Flow Packets/s"]
        for col in cols_nonpositive_to_nan:
            if col in df_clean.columns:
                mask = df_clean[col] <= 0
                if mask.any():
                    df_clean.loc[mask, col] = np.nan

        critical = [c for c in ["Flow Duration", "Flow Bytes/s"] if c in df_clean.columns]
        if critical:
            df_clean = df_clean.dropna(subset=critical)

    if method == "strict":
        numeric_present = [c for c in NUMERIC_COLS if c in df_clean.columns]
        df_clean = df_clean.dropna(subset=numeric_present)
    elif method == "fill_mean":
        numeric_present = [c for c in NUMERIC_COLS if c in df_clean.columns]
        df_clean[numeric_present] = df_clean[numeric_present].fillna(df_clean[numeric_present].mean())

    if "Label" in df_clean.columns and df_clean["Label"].dtype == 'object':
        df_clean["Label"] = df_clean["Label"].astype("category")

    print(df_clean.head())
    return df_clean


```

```python
def apply_outlier_method(df: pd.DataFrame, method: str = "none") -> pd.DataFrame:
    if method == "none":
        return df

    cols = [c for c in ["Flow Duration", "Flow Bytes/s", "Average Packet Size"] if c in df.columns]
    if not cols:
        return df

    # df_out = df.copy()
    df_out = df

    mask = pd.Series(True, index=df_out.index)

    if method == "iqr":
        for col in cols:
            q1 = df_out[col].quantile(0.25)
            q3 = df_out[col].quantile(0.75)
            iqr = q3 - q1
            mask &= df_out[col].between(q1 - 1.5 * iqr, q3 + 1.5 * iqr)
    elif method == "zscore":
        for col in cols:
            z = np.abs(stats.zscore(df_out[col].fillna(0)))
            mask &= (z < 3)

    return df_out[mask]


def apply_transform_method(df, method="none"):
    if method == "none":
        return df

    df_tr = df.copy()
    num_cols = [c for c in NUMERIC_COLS if c in df_tr.columns]

    if method == "log1p":
        for col in num_cols:
            if df_tr[col].min() >= 0:
                df_tr[col] = np.log1p(df_tr[col])
    elif method == "minmax":
        scaler = MinMaxScaler()
        df_tr[num_cols] = scaler.fit_transform(df_tr[num_cols])
```

```
    elif method == "standard":
        scaler = StandardScaler()
        df_tr[num_cols] = scaler.fit_transform(df_tr[num_cols])

    return df_tr


def get_processed_df(clean, outlier, transform, range_val=None):
    gc.collect()

    df = clean_dataset(df_raw, method=clean)
    df = apply_outlier_method(df, method=outlier)
    df = apply_transform_method(df, method=transform)

    if range_val and "Flow Duration" in df.columns:
        min_p, max_p = range_val
        if len(df) > 0:
            low_val = np.percentile(df["Flow Duration"], min_p)
            high_val = np.percentile(df["Flow Duration"], max_p)
            df = df[(df["Flow Duration"] >= low_val) & (df["Flow Duration"] <=
                high_val)]

    return df


# ------------------------------------------------------------------
# 3. Helper Functions (Including Normality Logic)
# ------------------------------------------------------------------

def make_warning_figure(msg: str) -> go.Figure:
    fig = go.Figure()
    fig.add_annotation(text=msg, showarrow=False, font=dict(size=16))
    fig.update_xaxes(visible=False)
    fig.update_yaxes(visible=False)
    return fig


def perform_normality_tests(data, title, method="all"):
    results = []
    alpha = 0.01

    sample_size = 2000
    sample_data = data if len(data) < sample_size else np.random.choice(data,
```

```
        sample_size, replace=False)

    if method in ["all", "shapiro"]:
        stat, p = shapiro(sample_data)
        res_str = f"Shapiro test: {title} : stat={stat:.2f} p={p:.4f}"
        res_str += " -> Normal" if p > alpha else " -> NOT Normal"
        results.append(res_str)

    if method in ["all", "ks"]:
        mean = np.mean(data)
        std = np.std(data)
        stat, p = kstest(data, 'norm', args=(mean, std))
        res_str = f"K-S test: {title} : stat={stat:.2f} p={p:.4f}"
        res_str += " -> Normal" if p > alpha else " -> NOT Normal"
        results.append(res_str)

    if method in ["all", "k2"]:
        try:
            stat, p = normaltest(data)
            res_str = f"da_k_squared: {title} : stat={stat:.2f} p={p:.4f}"
            res_str += " -> Normal" if p > alpha else " -> NOT Normal"
            results.append(res_str)
        except Exception as e:
            results.append(f"da_k_squared failed: {str(e)}")

    return "\n\n".join(results)


# ---------------------------------------------------------------------
# 4. Layout & App
# ---------------------------------------------------------------------
external_stylesheets = [dbc.themes.BOOTSTRAP]
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
server = app.server

navbar = dbc.Navbar(
    dbc.Container([
        html.A(
            dbc.Row([
                dbc.Col(
                    html.Img(src="https://upload.wikimedia.org/wikipedia/
                        commons/6/60/Virginia_Tech_Hokies_logo.svg",
                             height="50px")),
```

```
                dbc.Col(dbc.NavbarBrand("CS 5764 Final Project", className="ms-2")),
            ], align="center", className="g-0"),
            href="#", style={"textDecoration": "none"},
        ),
    ]),
    color="primary", dark=True,
)

# --- Global Controls ---
controls = dbc.Card([
    dbc.CardHeader("Preprocessing & Controls"),
    dbc.CardBody([
        dbc.Row([
            dbc.Col([
                html.Label("1. Data Cleaning"),
                dcc.Dropdown(
                    id="clean-method",
                    options=[
                        {"label": "Keep Original", "value": "none"},
                        {"label": "Basic (Drop Negatives)", "value": "basic"},
                        {"label": "Strict (Drop All NaNs)", "value": "strict"},
                        {"label": "Fill Mean", "value": "fill_mean"}
                    ],
                    value="basic", clearable=False
                )
            ], width=4),
            dbc.Col([
                html.Label("2. Outlier Removal"),
                dcc.Dropdown(
                    id="outlier-method",
                    options=[
                        {"label": "None", "value": "none"},
                        {"label": "IQR Method", "value": "iqr"},
                        {"label": "Z-Score Method", "value": "zscore"}
                    ],
                    value="none", clearable=False
                )
            ], width=4),
            dbc.Col([
                html.Label("3. Transformation"),
                dcc.Dropdown(
```

```
                    id="transform-method",
                    options=[
                        {"label": "None", "value": "none"},
                        {"label": "Log1p", "value": "log1p"},
                        {"label": "MinMax Scaling", "value": "minmax"},
                        {"label": "Standard Scaling", "value": "standard"}
                    ],
                    value="none", clearable=False
                )
            ], width=4),
        ]),
        html.Hr(),
        dbc.Row([
            dbc.Col([
                html.Label("4. Filter by Flow Duration (Percentile)"),
                dcc.RangeSlider(
                    id='range-slider',
                    min=0, max=100, step=5,
                    value=[0, 100],
                    marks={0: '0%', 50: '50%', 100: '100%'},
                    tooltip={"placement": "bottom", "always_visible": True}
                )
            ], width=6),
            dbc.Col([
                html.Label("5. Visual Options"),
                dcc.Checklist(
                    id='graph-options',
                    options=[
                        {'label': ' Show Grid', 'value': 'grid'},
                        {'label': ' Show Legend', 'value': 'legend'}
                    ],
                    value=['grid', 'legend'],
                    inline=True,
                    inputStyle={"margin-right": "5px"}
                )
            ], width=6)
        ]),
        html.Hr(),
        dbc.Row([
            dbc.Col([
                html.Label("Export Data:"),
                html.Br(),
                dbc.Button("Download CSV", id="btn-download", color="success",
```

```
                    size="sm"),
                dcc.Download(id="download-dataframe-csv"),
                dbc.Tooltip(
                    "Download the currently processed dataset as CSV",
                    target="btn-download",
                )
            ], width=12)
        ])
    ])
], className="mb-3")

# --- Tab 1: Data & Stats ---
tab1 = dbc.Container([
    html.H4("Data Overview & Statistics"),
    html.Div(id="data-shape-info", className="mb-3 text-primary fw-bold"),
    html.H5("Descriptive Statistics"),
    html.Div(id="stats-table-container")
], className="mt-3")

# --- Tab 2: PCA ---
tab2 = dbc.Container([
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Dimensionality Reduction (PCA)"),
                dbc.CardBody(
                    dcc.Loading(
                        id="loading-pca",
                        type="default",
                        children=dcc.Graph(id="pca-graph")
                    )
                )
            ])
        ], width=12),
    ])
], className="mt-3")

# --- Tab 3: Normality Tests ---
tab3 = dbc.Container([
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Normality Test Configuration"),
```

```
                dbc.CardBody([
                    html.Label("Select Feature to Test:"),
                    dcc.Dropdown(id="norm-feature", options=numeric_options, value="Flow Duration"),
                    html.Br(),
                    html.Label("Select Test Method:"),
                    dcc.Dropdown(
                        id="norm-method",
                        options=[
                            {'label': 'All Tests', 'value': 'all'},
                            {'label': 'Shapiro-Wilk', 'value': 'shapiro'},
                            {'label': 'Kolmogorov-Smirnov (K-S)', 'value': 'ks'},
                            {'label': "D'Agostino's K-squared", 'value': 'k2'}
                        ],
                        value='all', clearable=False
                    ),
                    html.Hr(),
                    html.Div(id="norm-result-text", style={"whiteSpace": "pre-wrap", "fontFamily": "monospace"})
                ])
            ])
        ], width=12)
    ])
], className="mt-3")

# --- Tab 4: Numeric Plots ---
tab4 = dbc.Container([
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Numeric Plot Configuration"),
                dbc.CardBody([
                    html.Label("Feature 1 (X):"),
                    dcc.Dropdown(id="num-f1", options=numeric_options, value="Flow Duration"),
                    html.Label("Feature 2 (Y - Optional):"),
                    dcc.Dropdown(id="num-f2", options=numeric_options, value="Flow Bytes/s"),
                    html.Label("Feature 3 (Z - Optional):"),
                    dcc.Dropdown(id="num-f3", options=numeric_options, value="Average Packet Size"),
                    html.Hr(),
```

```
                    html.Label("Select Plot Type:"),
                    dcc.Dropdown(
                        id="num-type",
                        options=[
                            {"label": "1. Line Plot", "value": "line"},
                            {"label": "2. Dist Plot (Histogram)", "value": "dist"},
                            {"label": "3. Pair Plot", "value": "pair"},
                            {"label": "4. Heatmap (Correlation)", "value": "heatmap"},
                            {"label": "5. Histogram + KDE", "value": "hist_kde"},
                            {"label": "6. QQ Plot", "value": "qq"},
                            {"label": "7. KDE Plot (Custom)", "value": "kde_custom"},
                            {"label": "8. Lm/Reg Plot", "value": "reg"},
                            {"label": "9. Area Plot", "value": "area"},
                            {"label": "10. Joint Plot", "value": "joint"},
                            {"label": "11. Rug Plot", "value": "rug"},
                            {"label": "12. 3D Plot", "value": "3d"},
                            {"label": "13. Contour Plot", "value": "contour"},
                            {"label": "14. Cluster Map", "value": "cluster"},
                            {"label": "15. Hexbin Plot", "value": "hexbin"}
                        ],
                        value="line", clearable=False
                    ),
                    html.Hr(),
                    html.Label("KDE / Line Settings:", className="fw-bold"),
                    html.Label("Palette / Color:"),
                    dcc.Dropdown(
                        id="kde-palette",
                        options=[
                            {"label": "Blue", "value": "blue"},
                            {"label": "Red", "value": "red"},
                            {"label": "Green", "value": "green"},
                            {"label": "Purple", "value": "purple"}
                        ],
                        value="blue", clearable=False
                    ),
                    html.Label("Line Width:"),
                    dcc.Slider(id="kde-width", min=1, max=5, step=0.5, value=2,
                               marks={1: '1', 2: '2', 3: '3', 4: '4', 5: '5'})
```

```
                ])
            ])
        ], width=3),
        dbc.Col([
            dcc.Loading(
                id="loading-num",
                type="default",
                children=dcc.Graph(id="numeric-graph", style={"height": "700px
                    "})
            )
        ], width=9)
    ])
], className="mt-3")

# --- Tab 5: Categorical Plots ---
tab5 = dbc.Container([
    dbc.Row([
        dbc.Col([
            dbc.Card([
                dbc.CardHeader("Categorical Config"),
                dbc.CardBody([
                    html.Label("Categorical Feature (X):"),
                    dcc.Dropdown(id="cat-col", options=cat_options, value="
                        Label"),
                    html.Label("Numeric Feature (Y - Optional):"),
                    dcc.Dropdown(id="cat-num", options=numeric_options, value=
                        "Flow Bytes/s"),
                    html.Hr(),
                    html.Label("Select Plot Type:"),
                    dcc.Dropdown(
                        id="cat-type",
                        options=[
                            {"label": "1. Bar Plot", "value": "bar"},
                            {"label": "2. Count Plot", "value": "count"},
                            {"label": "3. Pie Chart", "value": "pie"},
                            {"label": "4. Multivariate Box", "value": "
                                box_multi"},
                            {"label": "5. Multivariate Boxen", "value": "
                                boxen_multi"},
                            {"label": "6. Violin Plot", "value": "violin"},
                            {"label": "7. Strip Plot", "value": "strip"},
                            {"label": "8. Swarm Plot", "value": "swarm"}
                        ],
```

```
                        value="bar", clearable=False
                    ),
                    html.Hr(),
                    html.Label("Bar Plot Settings:", className="fw-bold"),
                    dcc.RadioItems(
                        id="bar-mode",
                        options=[
                            {"label": "Grouped", "value": "group"},
                            {"label": "Stacked", "value": "stack"}
                        ],
                        value="group",
                        inline=True
                    )
                ])
            ])
        ], width=3),
        dbc.Col([
            dcc.Loading(
                id="loading-cat",
                type="default",
                children=dcc.Graph(id="cat-graph", style={"height": "600px"})
            )
        ], width=9)
    ])
], className="mt-3")

# --- Tab 6: Storytelling ---
tab6 = dbc.Container([
    html.H3("Network Traffic Storytelling Dashboard", className="text-center
        my-3"),

    dcc.Loading(
        id="loading-story",
        type="default",
        children=dcc.Graph(id="story-graph", style={"height": "800px"})
    ),

    html.Br(),
    html.Label("Notes:"),
    dcc.Textarea(
        id="story-notes",
        placeholder="Write down any observations or takeaways from the
            storytelling view here.",
```

```
        style={
            "width": "100%",
            "height": "150px",
            "fontFamily": "monospace"
        }
    )
], className="mt-3")

# --- Main Layout ---
app.layout = html.Div([
    navbar,
    dbc.Container([
        controls,
        dcc.Tabs(
            id="tabs",
            value="tab-1",
            children=[
                dcc.Tab(label="1. Stats", value="tab-1", children=tab1),
                dcc.Tab(label="2. PCA", value="tab-2", children=tab2),
                dcc.Tab(label="3. Normality", value="tab-3", children=tab3),
                dcc.Tab(label="4. Numeric", value="tab-4", children=tab4),
                dcc.Tab(label="5. Categorical", value="tab-5", children=tab5),
                dcc.Tab(label="6. Story", value="tab-6", children=tab6),
            ],
        ),
    ], fluid=True, className="mt-3"),
])


# -------------------------------------------------------------------
# 5. Callbacks
# -------------------------------------------------------------------

@app.callback(
    Output("download-dataframe-csv", "data"),
    Input("btn-download", "n_clicks"),
    State("clean-method", "value"),
    State("outlier-method", "value"),
    State("transform-method", "value"),
    State("range-slider", "value"),
    prevent_initial_call=True
)
def download_processed_data(n_clicks, clean, outlier, transform, range_val):
```

```
    if n_clicks is None:
        return dash.no_update
    df = get_processed_df(clean, outlier, transform, range_val)
    return dcc.send_data_frame(df.to_csv, "processed_data.csv")


@app.callback(
    Output("data-shape-info", "children"),
    Output("stats-table-container", "children"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value")
)
def update_stats(clean, outlier, transform, range_val):
    df = get_processed_df(clean, outlier, transform, range_val)
    msg = f"Raw Shape: {df_raw.shape}  ->  Processed Shape: {df.shape}"

    desc = df[NUMERIC_COLS].describe().T.reset_index()
    desc = desc.round(3)

    table = dash_table.DataTable(
        data=desc.to_dict('records'),
        columns=[{"name": i, "id": i} for i in desc.columns],
        style_table={'overflowX': 'auto'},
        page_size=10,
        style_header={'fontWeight': 'bold'}
    )
    return msg, table


@app.callback(
    Output("pca-graph", "figure"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value")
)
def update_pca(clean, outlier, transform, range_val):
    df = get_processed_df(clean, outlier, transform, range_val)
    if df.empty:
        return make_warning_figure("No Data after filtering")

```

```
    df_sub = df.sample(min(len(df), 1000), random_state=42)

    X = df_sub[NUMERIC_COLS].select_dtypes(include=[np.number]).fillna(0)

    n_samples, n_features = X.shape

    if n_samples < 2 or n_features < 2:
        return make_warning_figure("Not enough data for PCA")

    target_n = 5
    n_components = min(target_n, n_samples, n_features)

    X_std = StandardScaler().fit_transform(X)

    pca = PCA(n_components=n_components)
    components = pca.fit_transform(X_std)

    # print(f"PCA performed with n_components={n_components}")

    color_col = df_sub["Label"] if "Label" in df_sub.columns else None
    if color_col is not None and hasattr(color_col, 'cat'):
        color_col = color_col.astype(str)

    pca_fig = px.scatter(
        x=components[:, 0],
        y=components[:, 1],
        color=color_col,
        title=f"PCA Projection (Calculated {n_components} components, Showing
            PC1 vs PC2)",
        labels={'x': f'PC1 ({pca.explained_variance_ratio_[0]:.2%})',
                'y': f'PC2 ({pca.explained_variance_ratio_[1]:.2%})'}
    )

    pca_fig.update_layout(
        title_font_family="serif",
        title_font_color="blue",
        font_family="serif",
        font_size=14,
        showlegend=True
    )
    pca_fig.update_xaxes(title_font_color="darkred", showgrid=True)
    pca_fig.update_yaxes(title_font_color="darkred", showgrid=True)

```

```python
    return pca_fig


@app.callback(
    Output("norm-result-text", "children"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value"),
    Input("norm-feature", "value"),
    Input("norm-method", "value")
)
def update_normality_test(clean, outlier, transform, range_val, feature,
    method):
    if not feature:
        return "Please select a feature."

    df = get_processed_df(clean, outlier, transform, range_val)
    data = df[feature].dropna()

    if len(data) < 3:
        return "Not enough data points to perform normality tests."

    result_string = perform_normality_tests(data, feature, method)
    return result_string


@app.callback(
    Output("numeric-graph", "figure"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value"),
    Input("num-f1", "value"),
    Input("num-f2", "value"),
    Input("num-f3", "value"),
    Input("num-type", "value"),
    Input("kde-palette", "value"),
    Input("kde-width", "value"),
    Input("graph-options", "value")
)
def update_numeric_plot(clean, outlier, transform, range_val, f1, f2, f3,
    plot_type, palette, width, options):
```

```
    df = get_processed_df(clean, outlier, transform, range_val)
    if df.empty: return make_warning_figure("No Data")

    df_sub = df.sample(min(len(df), 5000), random_state=42)

    if "Label" in df_sub.columns and hasattr(df_sub["Label"], 'cat'):
        df_sub["Label"] = df_sub["Label"].astype(str)

    if not f1: return make_warning_figure("Select Feature 1")

    fig = go.Figure()

    if plot_type == "line":
        df_sorted = df_sub.sort_index()
        fig = px.line(df_sorted, x=df_sorted.index, y=f1, title=f"Line Plot: {f1}")
        fig.update_layout(xaxis_title="Index")
        fig.update_traces(line=dict(width=width))

    elif plot_type == "dist":
        fig = px.histogram(df_sub, x=f1, title=f"Distribution Plot: {f1}")

    elif plot_type == "pair":
        cols = [c for c in [f1, f2, f3] if c]
        if len(cols) < 2: return make_warning_figure("Select at least 2 features")
        fig = px.scatter_matrix(df_sub, dimensions=cols, color="Label" if "Label" in df_sub.columns else None,
                                title="Pair Plot")

    elif plot_type == "heatmap":
        cols = [c for c in [f1, f2, f3] if c]
        if len(cols) < 2: return make_warning_figure("Select at least 2 features")
        fig = px.density_heatmap(df_sub, x=f1, y=f2, title=f"Heatmap: {f1} vs {f2}")

    elif plot_type == "hist_kde":
        fig = px.histogram(df_sub, x=f1, marginal="violin", title=f"Histogram with KDE: {f1}")

    elif plot_type == "qq":
        data = df_sub[f1].dropna()
```

```
        (osm, osr), _ = stats.probplot(data, dist="norm")
        fig = go.Figure()
        fig.add_trace(go.Scatter(x=osm, y=osr, mode='markers', name='Data'))
        fig.add_trace(go.Scatter(x=[min(osm), max(osm)], y=[min(osm), max(osm)], mode='lines', line=dict(color='red'),
                                 name='Normal Line'))
        fig.update_layout(
            title=f"QQ Plot: {f1}",
            xaxis_title="Theoretical Quantiles",
            yaxis_title="Sample Quantiles"
        )

    elif plot_type == "kde_custom":
        data = df_sub[f1].dropna()
        if len(data) > 1:
            kde = stats.gaussian_kde(data)
            x_range = np.linspace(data.min(), data.max(), 200)
            y_val = kde(x_range)
            color_map = {"blue": "#1f77b4", "red": "#d62728", "green": "#2ca02c", "purple": "#9467bd"}
            line_color = color_map.get(palette, "blue")
            fig = go.Figure()
            fig.add_trace(go.Scatter(x=x_range, y=y_val, mode='lines', fill='tozeroy',
                                     line=dict(color=line_color, width=width), opacity=0.6, name="KDE"))
            fig.update_layout(title=f"KDE Plot (Alpha=0.6, Width={width}): {f1}",
                              xaxis_title=f"{f1}",
                              yaxis_title="Density", )
        else:
            return make_warning_figure("Not enough data for KDE")

    elif plot_type == "reg":
        if not f2: return make_warning_figure("Select Feature 2")
        fig = px.scatter(df_sub, x=f1, y=f2, trendline="ols", title=f"Regression: {f1} vs {f2}",
                         trendline_color_override="red")

    elif plot_type == "area":
        if not f2: return make_warning_figure("Select Feature 2")
        df_sorted = df_sub.sort_values(by=f1)
        fig = px.area(df_sorted, x=f1, y=f2, title=f"Area Plot: {f1} vs {f2}")
```

```

    elif plot_type == "joint":
        if not f2: return make_warning_figure("Select Feature 2")
        fig = px.scatter(df_sub, x=f1, y=f2, marginal_x="violin", marginal_y="violin",
                         title="Joint Plot (Scatter + KDE-style Marginals)")

    elif plot_type == "rug":
        fig = px.strip(df_sub, x=f1, title=f"Rug Plot: {f1}")
        fig.update_layout(
            xaxis_title=f"{f1}",
            yaxis_title="Occurrences",
        )

    elif plot_type == "3d":
        if not f3: return make_warning_figure("Select Feature 3")
        fig = px.scatter_3d(df_sub, x=f1, y=f2, z=f3, color="Label" if "Label" in df_sub.columns else None,
                            title="3D Plot")

    elif plot_type == "contour":
        if not f2: return make_warning_figure("Select Feature 2")
        fig = px.density_contour(df_sub, x=f1, y=f2, title=f"Contour Plot: {f1} vs {f2}")

    elif plot_type == "cluster":
        cols = [c for c in NUMERIC_COLS if c in df.columns][:10]
        corr = df[cols].corr()
        fig = px.imshow(corr, title="Cluster Map")

    elif plot_type == "hexbin":
        if not f2: return make_warning_figure("Select Feature 2")
        fig = px.density_heatmap(df_sub, x=f1, y=f2, nbinsx=30, nbinsy=30, title=f"Hexbin Plot: {f1} vs {f2}")

    else:
        return make_warning_figure("Unknown Plot Type")

    show_grid = 'grid' in options
    show_legend = 'legend' in options
    fig.update_layout(
        title_font_family="serif",
        title_font_color="blue",
```

```
        font_family="serif",
        showlegend=show_legend,
        font_size=14,
    )
    fig.update_xaxes(title_font_color="darkred", showgrid=show_grid)
    fig.update_yaxes(title_font_color="darkred", showgrid=show_grid)

    return fig


@app.callback(
    Output("cat-graph", "figure"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value"),
    Input("cat-col", "value"),
    Input("cat-num", "value"),
    Input("cat-type", "value"),
    Input("bar-mode", "value"),
    Input("graph-options", "value")
)
def update_categorical_plot(clean, outlier, transform, range_val, cat, num,
   plot_type, bar_mode, options):
    df = get_processed_df(clean, outlier, transform, range_val)
    if df.empty: return make_warning_figure("No Data")

    df_sub = df.sample(min(len(df), 5000), random_state=42)

    if "Label" in df_sub.columns and hasattr(df_sub["Label"], 'cat'):
        df_sub["Label"] = df_sub["Label"].astype(str)

    if not cat: return make_warning_figure("Select Categorical Feature")

    fig = go.Figure()

    if plot_type == "bar":
        if not num: return make_warning_figure("Select Numeric Feature")
        group_cols = [cat]
        if "Label" in df_sub.columns and cat != "Label":
            group_cols.append("Label")
        df_agg = df_sub.groupby(group_cols, observed=False)[num].mean().
           reset_index()
```

```
        color_col = "Label" if "Label" in df_sub.columns else None
        fig = px.bar(df_agg, x=cat, y=num, color=color_col, barmode=bar_mode, title=f"Bar Plot ({bar_mode})")

    elif plot_type == "count":
        fig = px.histogram(df_sub, x=cat, color="Label" if "Label" in df_sub.columns else None, barmode=bar_mode,
                           title=f"Count Plot: {cat}")

    elif plot_type == "pie":
        fig = px.pie(df_sub, names=cat, title=f"Pie Chart: {cat}")

    elif plot_type == "box_multi":
        if not num: return make_warning_figure("Select Numeric Feature")
        fig = px.box(df_sub, x=cat, y=num, color="Label" if "Label" in df_sub.columns else None,
                     title=f"Box Plot: {num} by {cat}")

    elif plot_type == "boxen_multi":
        if not num: return make_warning_figure("Select Numeric Feature")
        fig = px.box(df_sub, x=cat, y=num, color="Label" if "Label" in df_sub.columns else None, points="all",
                     title=f"Boxen Plot")

    elif plot_type == "violin":
        if not num: return make_warning_figure("Select Numeric Feature")
        fig = px.violin(df_sub, x=cat, y=num, color="Label" if "Label" in df_sub.columns else None, box=True,
                        title=f"Violin Plot")

    elif plot_type == "strip":
        if not num: return make_warning_figure("Select Numeric Feature")
        fig = px.strip(df_sub, x=cat, y=num, color="Label" if "Label" in df_sub.columns else None, title=f"Strip Plot")

    elif plot_type == "swarm":
        if not num: return make_warning_figure("Select Numeric Feature")
        fig = px.strip(df_sub, x=cat, y=num, color="Label" if "Label" in df_sub.columns else None, stripmode='overlay',
                       title=f"Swarm Plot")

    else:
        return make_warning_figure("Unknown Plot Type")
```

```python

    show_grid = 'grid' in options
    show_legend = 'legend' in options
    fig.update_layout(
        title_font_family="serif",
        title_font_color="blue",
        font_family="serif",
        showlegend=show_legend,
        font_size=14,
    )
    fig.update_xaxes(title_font_color="darkred", showgrid=show_grid)
    fig.update_yaxes(title_font_color="darkred", showgrid=show_grid)

    return fig


@app.callback(
    Output("story-graph", "figure"),
    Input("clean-method", "value"),
    Input("outlier-method", "value"),
    Input("transform-method", "value"),
    Input("range-slider", "value")
)
def update_storytelling(clean, outlier, transform, range_val):
    df = get_processed_df(clean, outlier, transform, range_val)

    if df.empty:
        return make_warning_figure("No data available with current settings.")

    df_sub = df.sample(min(len(df), 3000), random_state=42)

    if "Label" in df_sub.columns and hasattr(df_sub["Label"], 'cat'):
        df_sub["Label"] = df_sub["Label"].astype(str)

    fig = make_subplots(
        rows=2, cols=2,
        subplot_titles=("Attack Distribution", "Flow Duration by Label",
                        "Traffic Volume (Bytes vs Packets)", "Average Packet Size"),
        specs=[[{"type": "domain"}, {"type": "xy"}],
               [{"type": "xy"}, {"type": "xy"}]]
    )

```

```python
    if "Label" in df_sub.columns:
        counts = df_sub["Label"].value_counts().reset_index()
        counts.columns = ["Label", "count"]
        fig.add_trace(go.Pie(labels=counts["Label"], values=counts["count"], name="Attacks"), row=1, col=1)

        for label in df_sub["Label"].unique():
            subset = df_sub[df_sub["Label"] == label]
            fig.add_trace(go.Box(y=subset["Flow Duration"], name=str(label), showlegend=False), row=1, col=2)

        avg_size = df_sub.groupby("Label", observed=False)["Average Packet Size"].mean().reset_index()
        fig.add_trace(go.Bar(x=avg_size["Label"], y=avg_size["Average Packet Size"], name="Avg Size", showlegend=False),
                      row=2, col=2)
    else:
        fig.add_annotation(text="No Label Column", row=1, col=1)

    fig.add_trace(go.Scatter(x=df_sub["Flow Packets/s"], y=df_sub["Flow Bytes/s"],
                             mode='markers', marker=dict(size=5, opacity=0.5),
                             name="Traffic", showlegend=False), row=2, col=1)

    fig.update_layout(height=800, title_text="Comprehensive Network Traffic Analysis")
    return fig


if __name__ == "__main__":
    app.run(debug=False, host="0.0.0.0", port=8080)
```

17 References

1. Canadian Institute for Cybersecurity. (2017). *Intrusion detection evaluation dataset (CIC-IDS2017)*. University of New Brunswick. https://www.unb.ca/cic/datasets/ids-2017.html

2. Plotly. (2025). *Plotly Python Graphing Library*. https://plotly.com/python/

3. Plotly. (2025). *Dash Documentation & User Guide*. https://dash.plotly.com/

4. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP).*