ΕΞΟΡΥΞΗ ΔΕΔΟΜΕΝΩΝ και ΑΛΓΟΡΙΘΜΟΙ ΜΑΘΗΣΗΣ

ΤΣΑΜΟΠΟΥΛΟΣ ΑΓΓΕΛΟΣ, 1072591,
up1072591@upnet.gr

Ως γλώσσα υλοποίησης ορίστηκε η Python μαζί με τις βιβλιοθήκες pandas και notebook, numpy κτλ.

ΕΡΩΤΗΜΑ 1

Αρχικά, το αρχείο csv που προτείνεται για την εργασία ήταν πολύ μεγάλο (περίπου 5 GB) με σκοπό το διάβασμα από το pandas να μην είναι ρεαλιστικό. Για αυτό το λόγο, ακολουθήθηκε η διαδικασία κοψίματος του dataset σε chunks ώστε να μπορέσουμε να κάνουμε με αυτά τις κατάλληλες διαδικασίες. Πιο συγκεκριμένα, τα chunks που παράγονται είναι **parquet** αρχεία για ταχύτερη διαχείριση της μνήμης και συμπίεση (ειναι column-based και συνεπώς φορτώνονται πιο εύκολα ενώ αντιθέτως το csv φορτώνεται όλο μαζί). Επίσης γίνεται downcasting αριθμητικών τύπων (πχ int64 => int 32) για οικονομία μνήμης. Όλη αυτή η διαδικασία πραγματοποιείται στο αρχείο **main.py**. Συνολικά έχουμε 2 πακέτα με 4500000 γραμμες το ένα και τις υπόλοιπες το άλλο.

Αφού παραχθούν αυτά τα πακέτα, άμα ξανατρέξουμε το **main.py** θα μας βγάλει ένα json αρχείο που έχει παραχθεί μαζί με τα parquet το οποίο περιέχει αναλυτικά το πλήθος των διαφορετικών τιμών του Label, Traffic Type και Traffic Subtype. Επισυνάπτεται ένα screenshot παρακάτω το οποίο δείχνει το αποτέλεσμα:

```
Parquet files existed
{
  "rows_written": 8656767,
  "format": "parquet",
  "num_parts": 9,
  "label_counts": {
    "Benign": 1301,
    "Malicious": 8655466
  },
  "traffic_type_counts": {
    "Audio": 190,
    "Background": 32,
    "Bruteforce": 35172,
    "DoS": 7490929,
    "Information Gathering": 1038363,
    "Mirai": 91002,
    "Text": 209,
    "Video": 870
  },
  "traffic_subtype_counts": {
    "Audio": 190,
    "Background": 32,
    "Bruteforce FTP": 3485,
    "Bruteforce SSH": 3967,
    "Bruteforce Telnet": 4913,
    "Bruteforce HTTP": 628,
    "Bruteforce DNS": 22179,
    "DoS ICMP": 9,
    "DoS UDP": 257994,
    "DoS SYN": 856764,
    "DoS FIN": 725600,
    "DoS ACK": 936307,
    "DoS PSH": 909507,
    "DoS RST": 1072504,
    "DoS URG": 906190,
    "DoS ECN": 871150,
    "DoS CWR": 872523,
    "DoS HTTP": 82351,
    "DoS MAC": 30,
    "Information Gathering": 1038363,
    "Mirai Scan Bruteforce": 8731,
    "Mirai DDoS UDP": 71,
    "Mirai DDoS ACK": 3779,
    "Mirai DDoS SYN": 14210,
    "Mirai DDoS GREIP": 49,
    "Mirai DDoS GREETH": 43,
    "Mirai DDoS HTTP": 8923,
    "Mirai DDoS DNS": 55196,
    "Text": 209,
    "Video HTTP": 376,
    "Video RTP": 349,
    "Video UDP": 145
  }
}
```

Τα parquet αρχεία μετά την εκτέλεση βρίσκονται στο φάκελο **data** μαζί με το αρχείο **summary.json**.

Στο αρχείο **main.ipynb** υπάρχουν πληροφορίες για τις στήλες που μπορούμε να βρούμε με την εντολή **df.info()** (έχοντας διαβάσει ένα chunk), όπου μας παρουσιάζει τι τύπο είναι η κάθε στήλη (float32, int32 ή category) και το περιεχόμενο των αρχείων μπορούμε να το δούμε με την εντολή **df.head()**. Επίσης με την εντολή **df.['name']=value.counts()** μπορούμε να δούμε τις διαφορετικές τιμές που έχει η στήλη με όνομα **name** και πόσες είναι αυτές (βοηθάει στο να δούμε διάφορες πληροφορίες για την επεξεργασία).

```
df.head()
```

	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Duration	Total Fwd Packet	Total Bwd packets	...	Active Std	Active Max	Active Min	Idle Mean	Idl
0	192.168.1.90-192.168.1.3-53930-64738-6	192.168.1.90	53930.0	192.168.1.3	64738	6.0	01/01/1970 07:41:46 AM	52601172.0	1701.0	1793.0	...	0.0	0.0	0.0	0.0	0.0(
1	192.168.1.3-192.168.1.90-64738-37700-6	192.168.1.3	64738.0	192.168.1.90	37700	6.0	01/01/1970 07:41:46 AM	119106944.0	36.0	57.0	...	3416174.0	19996926.0	14078617.0	5001511.0	1737.4(
2	192.168.1.3-192.168.1.90-22-40854-6	192.168.1.3	22.0	192.168.1.90	40854	6.0	01/01/1970 07:41:46 AM	5589.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0(
3	192.168.1.70-192.168.1.3-55422-64738-6	192.168.1.70	55422.0	192.168.1.3	64738	6.0	01/01/1970 07:41:47 AM	118166560.0	3932.0	4196.0	...	0.0	0.0	0.0	0.0	0.0(
4	192.168.1.90-192.168.1.3-59658-64738-17	192.168.1.90	59658.0	192.168.1.3	64738	17.0	01/01/1970 07:41:50 AM	119988384.0	25.0	6795.0	...	0.0	0.0	0.0	0.0	0.0(

5 rows × 86 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4500000 entries, 0 to 4499999
Data columns (total 86 columns):
 #   Column                       Dtype
---  ------                       -----
 0   Flow ID                      category
 1   Src IP                       category
 2   Src Port                     float32
 3   Dst IP                       category
 4   Dst Port                     int32
 5   Protocol                     float32
 6   Timestamp                    category
 7   Flow Duration                float32
 8   Total Fwd Packet             float32
 9   Total Bwd packets            float32
 10  Total Length of Fwd Packet   float32
 11  Total Length of Bwd Packet   float32
 12  Fwd Packet Length Max        float32
 13  Fwd Packet Length Min        float32
 14  Fwd Packet Length Mean       float32
 15  Fwd Packet Length Std        float32
 16  Bwd Packet Length Max        float32
 17  Bwd Packet Length Min        float32
 18  Bwd Packet Length Mean       float32
 19  Bwd Packet Length Std        float32
 20  Flow Bytes/s                 float32
 21  Flow Packets/s               float32
 22  Flow IAT Mean                float32
 23  Flow IAT Std                 float32
 24  Flow IAT Max                 float32
 25  Flow IAT Min                 float32
 26  Fwd IAT Total                float32
 27  Fwd IAT Mean                 float32
 28  Fwd IAT Std                  float32
 29  Fwd IAT Max                  float32
 30  Fwd IAT Min                  float32
 31  Bwd IAT Total                float32
 32  Bwd IAT Mean                 float32
 33  Bwd IAT Std                  float32
 34  Bwd IAT Max                  float32
 35  Bwd IAT Min                  float32
 36  Fwd PSH Flags                float32
 37  Bwd PSH Flags                float32
 38  Fwd URG Flags                float32
 39  Bwd URG Flags                float32
 40  Fwd Header Length            float32
 41  Bwd Header Length            float32
 42  Fwd Packets/s                float32
 43  Bwd Packets/s                float32
 44  Packet Length Min            float32
 45  Packet Length Max            float32
 46  Packet Length Mean           float32
 47  Packet Length Std            float32
 48  Packet Length Variance       float32
 49  FIN Flag Count               float32
 50  SYN Flag Count               float32
 51  RST Flag Count               float32
 52  PSH Flag Count               float32
 53  ACK Flag Count               float32
 54  URG Flag Count               float32
 55  CWR Flag Count               float32
 56  ECE Flag Count               float32
 57  Down/Up Ratio                float32
 58  Average Packet Size          float32
 59  Fwd Segment Size Avg         float32
 60  Bwd Segment Size Avg         float32
```

```
df['Fwd Seg Size Min'].value_counts()
```

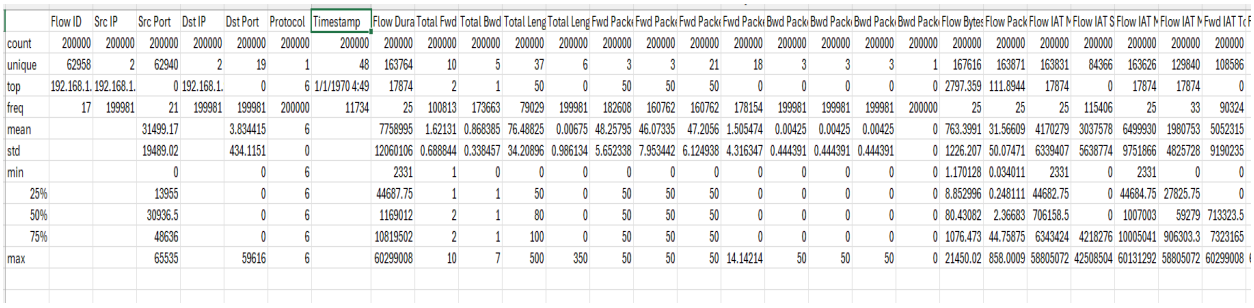
```
Fwd Seg Size Min
20.0    774320
8.0     133827
40.0     78934
32.0     12806
0.0         61
44.0        52
Name: count, dtype: int64
```

Κύριος στόχος είναι έχοντας τα πακέτα και τα κατάλληλα εργαλεία για επεξεργασία, να αφαιρέσουμε στήλες οι οποίες είναι ασήμαντες και μετά να τα κάνουμε aggregate σε ένα ενιαίο αρχείο για τα επόμενα ερωτήματα.

Σκοπός της επεξεργασίας για εμάς σε αυτό το σημείο είναι να δούμε τις διακυμάνσεις των τιμών σε κάθε στήλη. Για παράδειγμα, στην περίπτωση που μια στήλη έχει παρόμοιες τιμές σημαίνει για εμάς ότι δεν μας δίνει σημαντική πληροφορία. Μπορούμε πληροφορίες σαν και αυτές να βρούμε παρατηρώντας την μέγιστη και ελάχιστη τιμή κάθε στήλης, το std κτλ.

Τα αρχεία **main1.py** υπολογίζει τα στατιστικά μεγέθη (με την συνάρτηση **describe**) για το κάθε chunk και τα αποτελέσματα αποθηκεύονται στον φάκελο **eda_chunks** όπου βρίσκεται στον φάκελο **outputs**. Με αυτά μπορούμε να διακρίνουμε μέσους όρους, διασπορές, πόσες διακριτές τιμές υπάρχουν για την κάθε στήλη κτλ.

Παρακάτω επισυνάπτεται ένα screenshot που περιέχει αυτές τις πληροφορίες για το chunk 2.



	Flow ID	Src IP	Src Port	Dst IP	Dst Port	Protocol	Timestamp	Flow Dura	Total Fwd	Total Bwd	Total Leng	Total Leng	Fwd Packe	Fwd Packe	Fwd Packe	Fwd Packe	Bwd Packe	Bwd Packe	Bwd Packe	Bwd Packe	Flow Bytes	Flow Pack	Flow IAT M	Flow IAT S	Flow IAT M	Flow IAT M	Fwd IAT To
count	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000	200000
unique	62958	2	62940	2	19	1	48	163764	10	5	37	6	3	3	21	18	3	3	3	1	167616	163871	163831	84366	163626	129840	108586
top	192.168.1.	192.168.1.	0	192.168.1.	0	6	1/1/1970 4:49	17874	2	1	50	0	50	50	50	0	0	0	0	0	2797.359	111.8944	17874	0	17874	17874	0
freq	17	199981	21	199981	199981	200000	11734	25	100813	173663	79029	199981	182608	160762	160762	178154	199981	199981	199981	200000	25	25	25	115406	25	33	90324
mean			31499.17		3.834415	6		7758995	1.62131	0.868385	76.48825	0.00675	48.25795	46.07335	47.2056	1.505474	0.00425	0.00425	0.00425	0	763.3991	31.56609	4170279	3037578	6499930	1980753	5052315
std			19489.02		434.1151	0		12060106	0.688844	0.338457	34.20896	0.986134	5.652338	7.953442	6.124938	4.316347	0.444391	0.444391	0.444391	0	1226.207	50.07471	6339407	5638774	9751866	4825728	9190235
min			0		0	6		2331	1	0	0	0	0	0	0	0	0	0	0	0	1.170128	0.034011	2331	0	2331	0	0
25%			13955		0	6		44687.75	1	1	50	0	50	50	50	0	0	0	0	0	8.852996	0.248111	44682.75	0	44684.75	27825.75	0
50%			30936.5		0	6		1169012	2	1	80	0	50	50	50	0	0	0	0	0	80.43082	2.36683	706158.5	0	1007003	59279	713323.5
75%			48636		0	6		10819502	2	1	100	0	50	50	50	0	0	0	0	0	1076.473	44.75875	6343424	4218276	10005041	906303.3	7323165
max			65535		59616	6		60299008	10	7	500	350	50	50	50	14.14214	50	50	50	0	21450.02	858.0009	58805072	42508504	60131292	58805072	60299008

Το **main2.py** αρχείο υπολογίζει τα στατιστικά μεγέθη για όλα τα chunks (για να έχουμε μια πλήρη εικόνα όλου του αρχικού αρχείου) και τα αποθηκεύει στο αρχείο **global__minmax_mean_from_derscibe.csv**

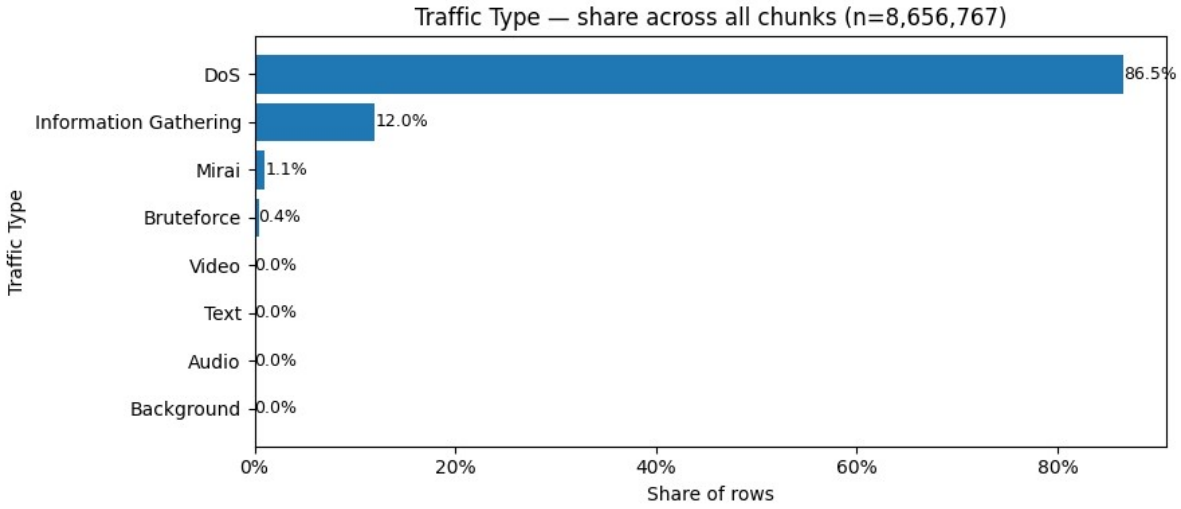
Η δομή του αρχείου διακρίνεται παρακάτω:

column	global_min	global_max	global_mean	global_std	total_count
ACK Flag Count	0	27079	1.145650347	43.07230678	8656767
Active Max	0	110239168	304639.3909	1102056.642	8656767
Active Mean	0	110239168	290594.2942	1028034.364	8656767
Active Min	0	110239168	278280.4434	990828.5629	8656767
Active Std	0	70699816	16318.35434	218156.705	8656767
Average Packet Size	0	2052	343.5453575	441.6241459	8656767
Bwd Bulk Rate Avg	0	8088000000	7565.117708	6123425.631	8656767
Bwd Bytes/Bulk Avg	0	27879460	299.2146538	50673.18444	8656767
Bwd Header Length	0	561288	19.61887396	782.9581124	8656767
Bwd IAT Max	0	115207168	50079.5261	899333.3152	8656767
Bwd IAT Mean	0	110919136	20320.74085	396279.4975	8656767
Bwd IAT Min	0	110919136	5808.570859	298426.337	8656767
Bwd IAT Std	0	78547408	24320.30078	436425.0508	8656767
Bwd IAT Total	0	119999112	62613.13583	1192730.314	8656767
Bwd Init Win Bytes	0	65280	24.04280652	1106.811057	8656767
Bwd PSH Flags	0	0	0	0	8656767
Bwd Packet Length Max	0	1448	4.076596573	55.20419001	8656767
Bwd Packet Length Mean	0	1448	1.381144806	22.79265163	8656767
Bwd Packet Length Min	0	1448	0.34384568	14.45402182	8656767
Bwd Packet Length Std	0	953.1799316	1.817069043	26.14955114	8656767
Bwd Packet/Bulk Avg	0	21185	0.25207921	38.73481521	8656767
Bwd Packets/s	0	83333.33594	14.99740095	119.6536664	8656767
Bwd Segment Size Avg	0	1448	1.381144806	22.79265163	8656767
Bwd URG Flags	0	0	0	0	8656767
CWR Flag Count	0	14	0.185295042	0.682903609	8656767
Down/Up Ratio	0	341	0.434887065	0.70678689	8656767
Dst Port	0	65535	4050.300695	12685.8323	8656767
ECE Flag Count	0	13	0.183680468	0.668433378	8656767
FIN Flag Count	0	3	0.174718576	0.512243753	8656767
FWD Init Win Bytes	0	65535	1345.015652	7325.898313	8656767
Flow Bytes/s	0	224666672	3734.520199	224729.8776	8656767
Flow Duration	1	119999984	8932502.229	13110651.24	8656767
Flow IAT Max	1	119824816	6646996.325	9213789.119	8656767
Flow IAT Mean	1	110940976	4698789.328	6940261.065	8656767

Επίσης στο αρχείο **main.ipynb** παρουσιάζεται ο παρακάτω κώδικας που δείχνει για την στήλη Traffic Type σε τι ποσοστό παρουσιάζονται οι διαφορετικές του τιμές:

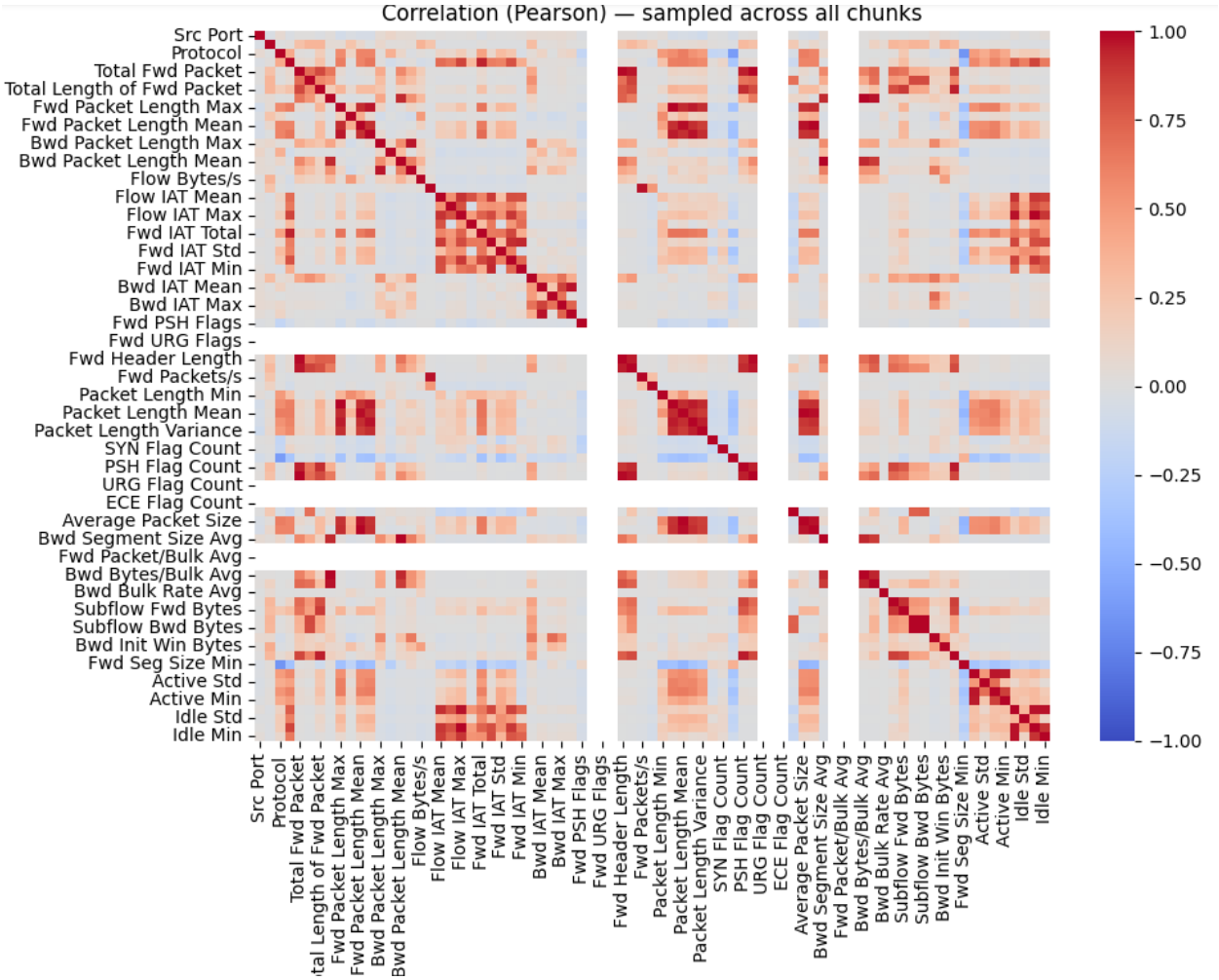
```
import matplotlib.pyplot as plt
import matplotlib.ticker as mtick

vc = (pd.Series(counts, dtype="int64")
      .sort_values(ascending=False)
      .rename_axis("Traffic Type")
      .to_frame("count"))
vc["share"] = vc["count"] / total

plt.figure(figsize=(9, 4))
bars = plt.barh(vc.index.astype(str), vc["share"].values)
plt.title(f"Traffic Type Column (n={total:,})")
plt.xlabel("Share of rows"); plt.ylabel("Traffic Type")
plt.gca().xaxis.set_major_formatter(mtick.PercentFormatter(1.0))
plt.gca().invert_yaxis()  # largest on top

for r, v in zip(bars, vc["share"].values):
   plt.text(v, r.get_y()+r.get_height()/2, f"{v:.1%}", va="center", ha="left", fontsize=9)

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

Επισυνάπτεται παρακάτω ένα screenshot με τα αποτελέσματα:



Βλέπουμε την διακύμανση των τιμών αυτού του γνωρίσματος καθώς με αυτό θα δουλέψουμε στο τρίτο ερώτημα (μαζί με το Label). Παρατηρείται ότι η τιμή DoS υπερισχύει κατά πολύ από τις υπόλοιπες τιμές με δεύτερη να είναι η information Gathering. Την παραπάνω διαδικασία μπορούμε να την εφαρμόσουμε για κάθε στήλη.

Επιπλέον έχουμε υπολογίσει και το **heatmap** μητρώο (correlation matrix) το οποίο φαίνεται παρακάτω και μας βοηθάει να διακρίνουμε συσχετίσεις μεταξύ των στηλών:



Μπορούμε να παρατηρήσουμε ότι οι στήλες Fwd Packet Length Mean και Fwd Packet Length Max είναι πολύ συσχετιζόμενες και αντίστοιχα οι τιμές Idle Min με τις τιμές Flow IAT Mean/Max και Fwd IAT Total. Επίσης, και οι στήλες Fwd Packet Length Max/Mean φαίνονται συσχετιζόμενες με τις Packet Length Min/Mean και Average Packet Size και ούτε καθεξής.

Μια άλλη στρατηγική που θεωρήθηκε χρήσιμη είναι να χρησιμοποιηθεί η συνάρτηση **pivot**. Πιο συγκεκριμένα επειδή θα γίνει train το μοντέλο με βάση το Label, μπορούμε να δούμε πως επηρεάζεται η κάθε στήλη με βάση το Label.

```
results
```

	mean		std	
	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Max	Fwd Packet Length Min
Label				
Benign	353.986481	6.202703	465.961060	15.703687
Malicious	333.857666	319.607910	395.093567	387.401672

Από τα αποτελέσματα παρατηρείται ότι η μέση ελάχιστη τιμή του μήκους πακέτων στην κατεύθυνση forward (Fwd Packet Length Min) είναι εξαιρετικά μικρή για την κανονική κυκλοφορία (περίπου bytes), ενω για την κακόβουλη κυκλοφορία εμφανίζεται σημαντικά αυξημένη (περίπου 320 bytes). Η διαφορά αυτή υποδηλώνει οτί οι κανονικές ροές περιλαμβάνουν συχνά πολύ μικρά πακέτα ελέγχου σε αντίθεση με τις κακόβουλες ροές που χαρακτηρίζονται από μεγαλύτερα πακέτα.

Παράλληλα, το χαρακτηριστικό Fwd Packet Length Max παρουσιάζει συγκρίσιμες μέσες τιμές μεταξύ των δυο κατηγοριών, ωστόσο εμφανίζει μεγάλη τυπική απόκλιση, γεγονός που υποδηλώνει υψηλή διασπορά στη συμπεριφορά των ροών και για τις δυο κλάσεις.

Αυτά τα αποτελέσματα δείχνουν ότι το χαρακτηριστικό Fwd Packet Length Min αποτελεί ιδιαίτερα διακριτικό γνώριμα για τον διαχωρισμό μεταξύ κανονικής και κακόβουλης μεταφοράς ενώ το Fwd Packet Length Max παρέχει συμπληρωματική πληροφορία.

Βλέπουμε και τι γίνεται με βάση το **Traffic Type** για τις 2 συγκεκριμένες στήλες:

```
results
```

	mean		std	
	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Max	Fwd Packet Length Min
Traffic Type				
Audio	413.605255	7.247368	478.635162	16.755537
Background	0.000000	0.000000	0.000000	0.000000
Bruteforce	94.370239	39.269989	225.611267	118.493080
DoS	335.744354	321.816406	395.565765	387.979065

Παρατηρούμε εδώ ότι οι διαφορετικοί τύποι κυκλοφορίας εμφανίζουν σαφώς διαφοροποιημένα πρότυπα ως προς τα μήκη των πακέτων. Η audio κυκλοφορία χαρακτηρίζεται από μικρές ελάχιστες τιμές και μεγάλη διασπορά (Fwd Packet Length Min). Αντίθετα, οι επιθέσεις Bruteforce εμφανίζουν μικρά μεγέθη ελάχιστων πακέτων , ενώ οι επιθέσεις DoS χαρακτηρίζονται από σημαντικά μεγαλύτερα μήκη ελάχιστων πακέτων και αυξημένη μεταβλητότητα. Η τιμή Background δεν προσφέρει πληροφορία για περαιτέρω ανάλυση.

Συνεπώς, οι στήλες Fwd Packet Length Max και Fwd Packet Length Min θεωρούνται απαραίτητες.

Αντίστοιχα μπορούμε να το κάνουμε και για άλλες στήλες.

Μια ακόμα σημαντική συνάρτηση του pandas είναι η **crosstab**. Στο παράδειγμά μας, μας δείχνει σε τι ποσοστό κάθε τιμή του Label πιάνει για κάθε συνδυασμό των τιμών του Protocol:

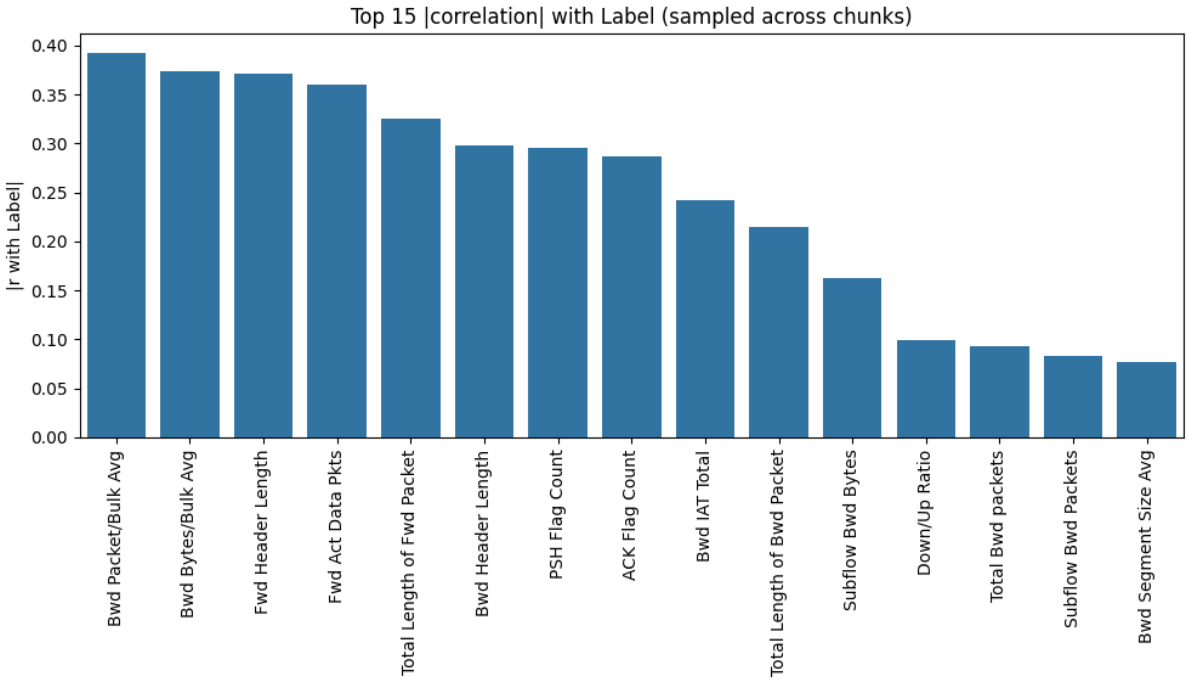
```
rate = pd.crosstab(df['Protocol'], df['Label'], normalize='index')
```

```
rate
```

Label	Benign	Malicious
Protocol		
0.0	0.517544	0.482456
6.0	0.000031	0.999969
17.0	0.000150	0.999850

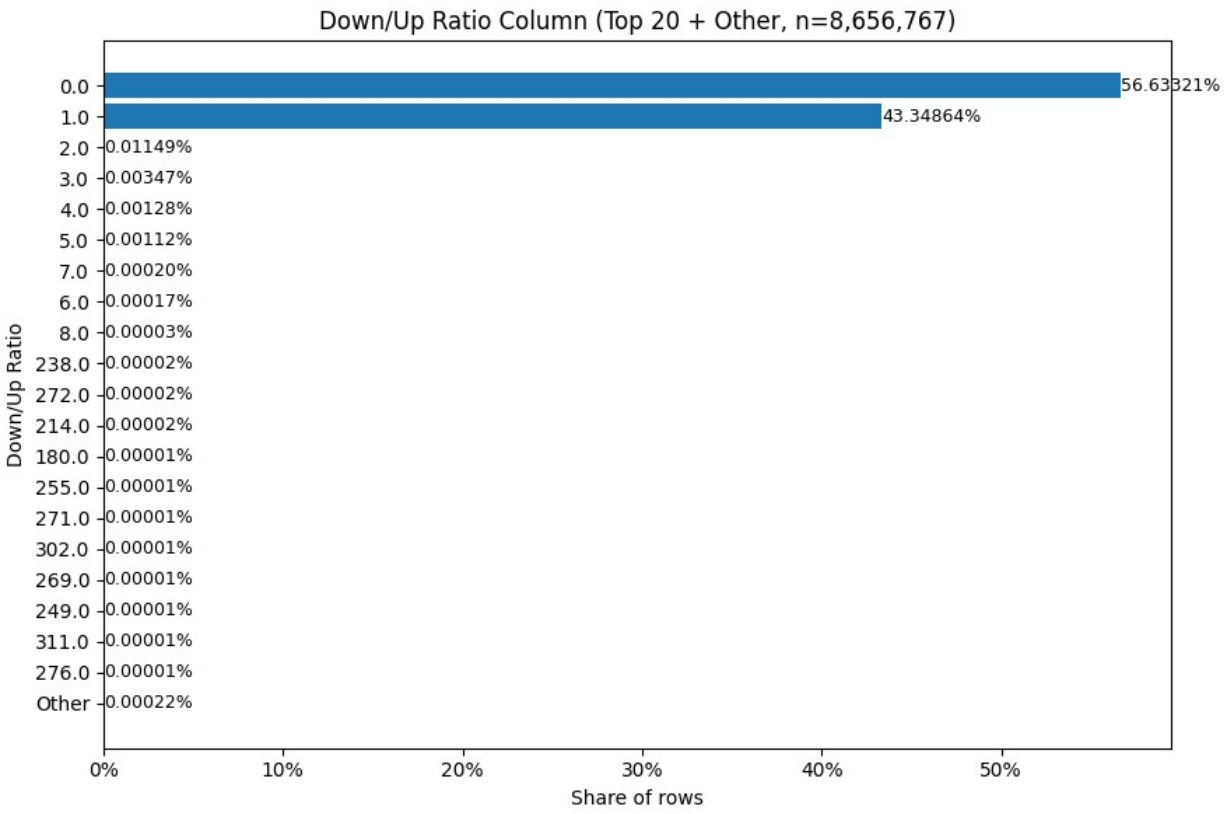
Παρατηρούμε ότι οι τιμές 6 και 17 (TCP και UDP) επιφέρουν κακόβουλο πρωτόκολλο σχεδόν πάντα ενώ η τιμή 0 (HOPOPT) δεν είναι διακριτική από μόνη του. Συμπεραίνουμε ότι είναι αυτή η στήλη είναι σημαντική για ταυτοποίηση επιθέσεων.

Τέλος, σημαντικό είναι να βρούμε την συσχέτιση του label με τα υπόλοιπα χαρακτηριστικά (ποια είναι πιο σχετικά και μπορούν να συνεισφέρουν στον διαχωρισμό των τιμών της) οπτικά, με βάση το παρακάτω διάγραμμα:



Επειδή τα στοιχεία που υπάρχουν ήταν παρα πολλά, έγινε μια τυχαία δειγματοληψία στα chunks που έχουμε (σύνολο 2) και κρατήθηκε το 0.8 ποσοστό αυτών. Στο διάγραμμα βλέπουμε αριστερά τις πιο σχετικές στήλες.

Θα μπορούσαμε να πούμε ότι η **Down/Up Ratio** στήλη είναι σχετική αλλά επίσης από την ανάλυση, η διακύμανση των τιμών της είναι εξαιρετικά μικρή όπως μπορούμε να διακρίνουμε παρακάτω:


Down/Up Ratio Column (Top 20 + Other, n=8,656,767)
Down/Up Ratio
0.0
1.0
2.0
3.0
4.0
5.0
7.0
6.0
8.0
238.0
272.0
214.0
180.0
255.0
271.0
302.0
269.0
249.0
311.0
276.0
Other
56.63321%
43.34864%
0.01149%
0.00347%
0.00128%
0.00112%
0.00020%
0.00017%
0.00003%
0.00002%
0.00002%
0.00002%
0.00001%
0.00001%
0.00001%
0.00001%
0.00001%
0.00001%
0.00001%
0.00001%
0.00022%
0%
10%
20%
30%
40%
50%
Share of rows

```
unique values: 39, top share: 0.566332
0.0 4902605
1.0 3752591
2.0 995
3.0 300
4.0 111
5.0 97
7.0 17
6.0 15
8.0 3
214.0 2
272.0 2
238.0 2
271.0 1
255.0 1
180.0 1
302.0 1
311.0 1
249.0 1
276.0 1
280.0 1
341.0 1
335.0 1
269.0 1
306.0 1
198.0 1
284.0 1
233.0 1
251.0 1
309.0 1
327.0 1
250.0 1
288.0 1
274.0 1
240.0 1
261.0 1
296.0 1
14.0 1
15.0 1
10.0 1
```

Συνεπώς, είναι προτιμητέο να την διαγράψουμε για την συνέχεια.

Σαν συμπέρασμα όλης αυτής της έρευνας, καταλήξαμε ότι οι στήλες οι οποίες θεωρούμε σημαντικές να κρατήσουμε για την συνέχεια είναι οι εξής:

Flow Duration, Total Fwd Packet, Total Bwd packets, Total Length of Fwd Packet, Total Length of Bwd Packet, Flow Bytes/s, Flow Packets/s, Fwd Packet Length Max, Fwd Packet Length Min, Bwd Packet Length Max, Bwd Packet Length Min, Packet Length Mean, Packet Length Std, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min, Fwd IAT Mean, Bwd IAT Mean, Protocol, SYN Flag Count, ACK Flag Count, RST Flag Count, PSH Flag Count, FIN Flag Count, Fwd PSH Flags, Bwd PSH Flags, Label, Traffic Type, Traffic Subtype.

Για να αφαιρέσουμε τις υπόλοιπες στήλες, τρέχουμε το αρχείο **reduce_chunk.py** το οποίο κόβει τις συγκεκριμένες στήλες από κάθε chunk. Τα νέα πακέτα υπάρχουν στον φάκελο data, στον φάκελο reduced.

Τέλος, τρέχοντας το αρχείο **aggregation.py** ενώνουμε τα πακέτα και σχηματίζεται το τελικό αρχείο **merged_reduced.parquet**, το οποίο χρησιμοποιείται στα επόμενα ερωτήματα.

ΕΡΩΤΗΜΑ 2

Οι στήλες που ήταν να αφαιρεθούν σε αυτό το ερώτημα έγινε προηγουμένως και τώρα έχουμε να διαχειριστούμε το αρχείο **merged_reduced.parquet**.

1) Στο παρών ερώτημα, σκοπός είναι η μείωση των γραμμών του συνόλου δεδομένων με τη μέθοδο δειγματοληψίας.

Επιλέχθηκε η μέθοδος του stratified sampling λαμβάνοντας υπόψιν τα γνωρίσματα **Label** και **Traffic Type** καθώς αυτά αποτελούν τις βασικές μεταβλητές στόχου για το τρίτο ερώτημα. Υπάρχει το αρχείο **row_sampl.py** το οποίο κάνει τα εξής πράγματα:

- Ορίζεται ότι το σύνολο δεδομένων που θα παραχθεί από την δειγματοληψία θα είναι 20% του αρχικού συνόλου που είχαμε στο πρώτο ερώτημα. Αυτός ο αριθμός περιέχεται στην μεταβλητή **target_total**.
- Διαβάζεται το αρχείο **merged_reduced.parquet** που περιέχει το αρχικό σύνολο δεδομένων και υπολογίζει για κάθε συνδυασμό (Label, Traffic Type) τον ιδανικό αριθμό (δεκαδικός) γραμμών που θα έπρεπε να περιλαμβάνει στο δείγμα για την διατήρηση

της αρχικής κατανομής των δεδομένων. Αυτό καθορίζεται ανάλογα με το ποσοστό της συμμετοχής του στο αρχικό σύνολο δεδομένων.

- Επειδή το παραπάνω αποτέλεσμα βγάζει κλασματικούς αριθμούς (βγαίνει ότι ένας συνδυασμός έχει 135,54 γραμμές), στρογγυλοποιούνται οι αριθμοί προς τα κάτω και το υπόλοιπο τους αποθηκεύεται στην μεταβλητή **remainder**. Αυτό όμως έχει σαν αποτέλεσμα γραμμές να λείπουν από κάποιους συνδυασμούς. Συνεπώς, γίνεται ταξινόμηση των υπολοίπων των συνδυασμών και έγινε διανομή των υπολειπόμενων γραμμών, μία ανά συνδυασμό ξεκινώντας από εκείνα με το μεγαλύτερο υπόλοιπο. Έτσι, ο κάθε συνδυασμός θα πάρει τις γραμμές που του αντιστοιχούνε.
- Αφού καθορίστηκε η ποσόστωση, πραγματοποιείται τυχαία δειγματοληψία για κάθε συνδυασμό, χρησιμοποιώντας γεννήτρια τυχαίων αριθμών με σταθερό seed (ίδιο seed => ίδιο δείγμα κάθε φορά που εκτελείται το πρόγραμμα).
- Τα επιμέρους δείγματα από κάθε συνδυασμό ενώνοναι σε ένα ενιαίο σύνολο δεδομένων, το οποίο αποθηκεύεται σε μορφή Parquet και λέγεται **stratified.parquet**.

Μια παρατήρηση που μπορεί να γίνει είναι ότι επιλέχθηκε το συγκεκριμένο ποσοστό δειγματοληψίας ώστε να επιτευχθεί ουσιαστική μείωση του μεγέθους του συνόλου δεδομένων χωρίς σημαντική απώλεια πληροφορίας.

2) Στο σημείο αυτό έπρεπε να επιλεχθούν δύο αλγόριθμοι για να πραγματοποιηθεί η συσταδοποίηση των δεδομένων. Οι δύο αυτοί αλγόριθμοι είναι ο Birch και ο DBScan.

2.1 **Birch**

Ο συγκεκριμένος αλγόριθμος επιλέχθηκε, διότι είναι κατάλληλος για μεγάλα σύνολα δεδομένων, καθώς αντί να αποθηκεύει όλα τα δεδομένα στη μνήμη, τα συνοψίζει σε ένα δέντρο συσταδοποίησης. Με αυτόν τον τρόπο δημιουργεί μια συμπύκνωση των δεδομένων, γεγονός που τον καθιστά ιδιαίτερα αποδοτικό τόσο σε χρόνο όσο και σε χρήση μνήμης.

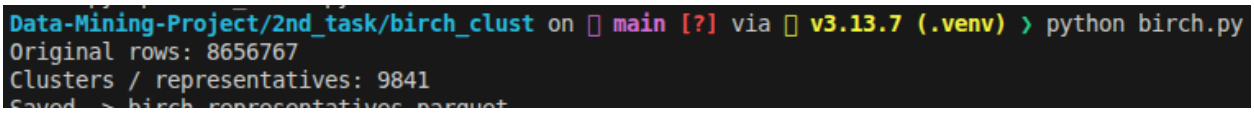
Αρχικά απομονώνουμε τις στήλες **Label**, **Traffic Type** και **Traffic Subtype** για να εξαιρεθούν από την διαδικασία συσταδοποίησης. Στη συνέχεια, όλα τα υπόλοιπα χαρακτηριστικά κανονικοποιούνται με τη χρήση του **StandardScaler** για να έχουν μέση τιμή 0 και διασπορά 1. Αυτό το βήμα είναι απαραίτητο καθώς ο Birch βασίζεται σε μετρικές απόστασης και διαφορετικές κλίμακες τιμών θα επηρέαζαν δυνανάλογα το αποτέλεσμα της συσταδοποίησης.

Έπειτα, εφαρμόζεται ο αλγόριθμος στο σύνολο δεδομένων που έχουμε και αναθέτει κάθε εγγραφή σε μια συστάδα. Για κάθε συστάδα, υπολογίζονται τα κεντροειδή ως ο μέσος όρος των αριθμητικών χαρακτηριστικών όλων των εγγραφών που ανήκουν στην συστάδα. Έτσι, κάθε συστάδα αντικαθίσταται από μια μόνο γραμμή που

συνοψίζει τη συμπεριφορά όλων των υπόλοιπων εγγραφών της ίδιας της συστάδας.

Τέλος, για κάθε συστάδα, βλέπουμε τις μεταβλητές στόχου (τις στήλες που αφαιρέσαμε προηγουμένως) και επιλέγουμε την τιμή αυτών που εμφανίζονται συχνότερα σε κάθε συστάδα ως την αντιπροσωπευτική τιμή. Αυτή την πληροφορία (τις τιμές των 3 στηλών στόχου) την προσθέτουμε στο αρχείο με τα κεντροειδή που πάμε να παράξουμε μαζί και με το πλήθος των αρχικών εγγραφών που αντιστοιχούν σε κάθε συστάδα (πόσα στοιχεία περιέχει κάθε συστάδα).

Το πλήθος των συστάδων όπως βλέπουμε παρακάτω, είναι 984 δηλαδή 9841 γραμμές και περιέχονται στο τελικό αρχείο (**birch_representatives.parquet**) που προκύπτει από τον birch.



Επισυνάπτουμε επίσης πως φαίνεται πως φαίνεται το τελικό αρχείο μέσω του αρχείου **main.ipynb**:

Birch

```
df_birch = pd.read_parquet("data/birch_representatives.parquet")
df_birch
```

cluster	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	Flow Bytes/s	Flow Packets/s	Fwd Packet Length Max	Fwd Packet Length Min	Bwd Packet Length Max	...	ACK Flag Count	RST Flag Count	PSH Flag Count	FIN Flag Count	Fwd PSH Flags	Bwd PSH Flags
0	52601172.0	1701.0	1793.0	149935.0	140566.0	5522.709473	66.424377	1318.0	0.0	1348.0	...	3493.0	0.0	2416.0	2.0	0.0	0.0
1	39624536.0	1319.0	1418.0	104208.0	115073.0	5533.970215	69.073364	1348.0	0.0	778.0	...	2737.0	1.0	1901.0	0.0	1.0	0.0
2	119996544.0	2879.0	3202.0	258125.0	266531.0	4372.259277	50.676456	1348.0	0.0	1348.0	...	6080.0	0.0	3570.0	0.0	0.0	0.0
3	64513256.0	1918.0	2012.0	145727.0	135911.0	4365.583496	60.917713	1304.0	0.0	1348.0	...	3929.0	0.0	2383.0	2.0	0.0	0.0
4	39434956.0	1281.0	1294.0	69415.0	109465.0	4536.077148	65.297394	523.0	0.0	1348.0	...	2574.0	0.0	1723.0	2.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9894	21410920.0	4459.0	7223.0	137.0	9626400.0	449608.750000	545.609436	137.0	0.0	1348.0	...	11681.0	0.0	2801.0	1.0	0.0	0.0
9895	32298440.0	5325.0	9451.0	137.0	12646391.0	391552.312500	457.483398	137.0	0.0	1348.0	...	14775.0	0.0	4199.0	1.0	0.0	0.0
9896	44296932.0	9161.0	14099.0	137.0	18523820.0	418176.968750	525.092834	137.0	0.0	1348.0	...	23259.0	0.0	6289.0	1.0	0.0	0.0
9897	53764816.0	5396.0	7740.0	137.0	10206203.0	189833.062500	244.323349	137.0	0.0	1348.0	...	13135.0	0.0	3749.0	1.0	0.0	0.0
9898	32189512.0	4554.0	7621.0	137.0	10174075.0	316072.250000	378.228760	137.0	0.0	1348.0	...	12174.0	0.0	3635.0	1.0	0.0	0.0

9841 rows × 30 columns

2.2 **Mini-Batch K-Means**

Ο συγκεκριμένος αλγόριθμος επιλέχθηκε ως ο δεύτερος αλγόριθμος ομαδοποίησης για τη μείωση του πλήθους των γραμμών του συνόλου δεδομένων λόγω της υψηλής κλιμακωσιμότητας για μεγάλα σύνολα δεδομένων. Επίσης, το αρχείο **merged_reduced.parquet** ήταν πολύ μεγάλο και δεν μπορούσε να αξιοποιηθεί κάποιος scalable αλγόριθμος όπως ο dbscan ενώ ο κλασσικός k-means παρόλο που ήταν η αρχική ιδέα είναι ιδιαίτερα ακριβός σε μνήμη και ταχύτητα. Αντιθέτως, ο mini-batch k-means αλγόριθμος χρησιμοποιεί τυχαία υποσύνολα των δεδομένων σε κάθε επανάληψη και παράγει κεντροειδή τα οποία

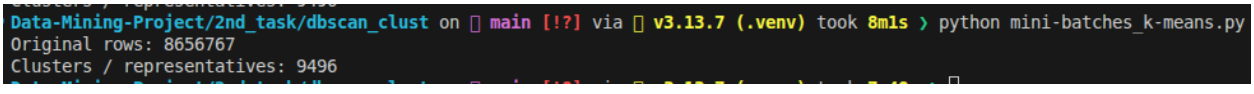
μπορούν να χρησιμοποιηθούν ως αντιπροσωπευτικές γραμμές του αρχικού συνόλου δεδομένων.

Επιλέχθηκε το πλήθος των συστάδων να ισούται αυθαίρετα με 10000 για να έχουμε ένα αρκετά καλό δείγμα των δεδομένων αλλά επίσης για να γίνει μια καλή συμπίεση αυτών. Παρόλα αυτά, στο τέλος εκτέλεσης του αλγορίθμου, οι συστάδες είναι πιο λίγες λόγω του ότι σε πολλές δεν τους έχει ανατεθεί ούτε ένα στοιχείο δίπλα τους και είναι κενές.

Ο αλγόριθμος δουλεύει σαν τον k-Means με την διαφορά ότι παίρνει δείγματα των 50000 στοιχείων με στοχαστικό τρόπο. Η διαδικασία αυτή επαναλαμβάνεται για συνολικά 500 επαναλήψεις. Επίσης, υπάρχει πιθανότητα να μην ανατίθενται σε κεντροειδή καθόλου στοιχεία και να παραμένουν κενά. Λόγω αυτού, έχουμε ορίσει παράμετρο **reassignement_ratio=0.01** το οποίο επιτρέπει την επανατοποθέτηση έως και 1% των κεντροειδών που θεωρούνται ανεπαρκώς χρησιμοποιούμενα σε σημεία του χώρου όπου υπάρχουν δεδομένα, βελτιώνοντας έτσι την ποιότητα του αλγορίθμου.

Τέλος, σημειώνεται ότι χρησιμοποιείται ο StandardScaler πριν την εφαρμογή του αλγορίθμου για την κανονικοποίηση των χαρακτηριστικών ενώ η διαδικασία προεπεξεργασίας είναι παρόμοια με αυτή που ακολουθήθηκε στο κομμάτι του Birch.

Το αρχείο που παράγεται λέγεται **mini-batches_k-means.parquet** και περιέχει 9496 γραμμές (συστάδες).



Επισυνάπτουμε επίσης πως φαίνεται πως φαίνεται το τελικό αρχείο μέσω του αρχείου **main.ipynb**:

```
df_batch_kmeans = pd.read_parquet("data/minibatch_kmeans_representatives.parquet")
df_batch_kmeans
```

cluster	Flow Duration	Total Fwd Packet	Total Bwd packets	Total Length of Fwd Packet	Total Length of Bwd Packet	Flow Bytes/s	Flow Packets/s	Fwd Packet Length Max	Fwd Packet Length Min	Bwd Packet Length Max	...	ACK Flag Count	RST Flag Count	PSH Flag Count	Fl Fla Cour
0	1.896218e+05	1.000000	1.0	1356.000000	0.0	7155.882812	10.554399	1356.000000	1356.000000	0.0	...	1.000000	1.000000	0.000000	0.
1	1.582129e+05	1.000000	1.0	50.000000	0.0	316.059021	12.642360	50.000000	50.000000	0.0	...	1.000000	1.000000	0.000000	0.
2	9.761597e+04	1.000000	1.0	50.000000	0.0	512.352295	20.494093	50.000000	50.000000	0.0	...	1.000000	1.000000	0.000000	0.
3	1.041331e+07	2.000000	1.0	1000.000000	0.0	96.039566	0.288119	500.000000	500.000000	0.0	...	1.090343	1.000000	0.000000	0.
4	1.012886e+07	2.000000	1.0	980.000000	0.0	96.754166	0.296186	500.000000	480.000000	0.0	...	2.000000	1.000000	0.000000	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
9994	3.993209e+07	2.000000	0.0	2712.000000	0.0	67.920715	0.050089	1356.000000	1356.000000	0.0	...	0.000000	2.000000	0.000000	0.
9995	1.868699e+04	1.000000	1.0	50.000000	0.0	2678.284424	107.131371	50.000000	50.000000	0.0	...	1.000000	1.000000	0.000000	1.
9996	2.119596e+07	2.000000	1.0	45.714287	0.0	2.206134	0.141683	28.571428	17.142857	0.0	...	1.049689	1.428571	0.006211	0.
9998	4.401307e+07	6.583688	0.0	3271.205566	0.0	74.315102	0.149571	500.000000	480.000000	0.0	...	0.000000	0.000000	6.583688	0.

Τα παραγόμενα σύνολα δεδομένων που είδαμε χρησιμοποιούνται στο επόμενο ερώτημα για την εκπαίδευση και σύγκριση μοντέλων ταξινόμησης.

Ο αλγόριθμος αυτός έχει σαν μειονέκτημα ότι η κατανομή του Label είναι πολύ κακιά και πιο συγκεκριμένα:

```
]: df_batch_kmeans["Label"].value_counts()

]: Label
   Malicious    9490
   Benign          6
   Name: count, dtype: int64
```

Αυτό θα δημιουργήσει πρόβλημα όταν θα προσπαθούμε να προβλέψουμε αν η κυκλοφορία είναι κανονική ή κακόβουλη.

ΕΡΩΤΗΜΑ 3

Χρησιμοποιούμε αρχικά για κατηγοριοποιητή βασισμένο σε Neural Networks έναν **MLP** (multi-layer percepton) για την πρόβλεψη της μεταβλητής **Label**, η οποία δηλώνει αν η δικτυακή κίνηση είναι Benign ή Malicious. Στόχος είναι η αξιολόγηση της απόδοσης του MLP στα σύνολα δεδομένων που προέκυψαν από το προηγούμενο ερώτημα.

Αφαιρούμε αρχικά τις στήλες Traffic Type και Traffic Subtype καθώς επηρεάζουν άμεσα την στήλη Label (και λόγω του ότι αποτελούν υψηλού επιπέδου σημασιολογικές ετικέτες) που είναι η μεταβλητή-στόχος και ταυτόχρονα κωδικοποιούμε την στήλη αυτή σε ακέραιες τιμές (0/1) με χρήση Label Encoding. Το αντίστροφο κάνουμε αν στόχος μας είναι η πρόβλεψη του Traffic Type που εκεί αφαιρούμε το Label και το Traffic Subtype. Χωρίζουμε το σύνολο δεδομένων σε σε training set (80%) και test set (20%), με την συνάρτηση train_test_split αλλά με τέτοιο τρόπο ώστε διατηρούμε τις αναλογίες των κλάσεων της μεταβλητής y (Label) και στα δυο σύνολα (stratify=y).

Πρέπει να σημειωθεί εδώ ότι έχουμε ορίσει την παράμετρο **early_stopping** με την τιμή true για να διαχωριστεί ένα 10% του training set ως validation set. Αυτό έγινε γιατί σταματάει την

εκπαίδευση του μοντέλου όταν αυτό μάθει απέξω το training set και έτσι αποφεύγεται το overfitting. Δηλαδή, όταν για n epochs (όπου n η τιμή της παραμέτρου n_iter_no_change) δεν βελτιώνεται η απόδοση του validation set, τότε σταματάει το training (έχουμε ορίσει το μέγιστο μέχρι 50 epochs να γίνονται).

Έπειτα, πραγματοποιούμε κανονικοποίηση των χαρακτηριστικών με χρήση του StandardScaler αλλά ξεχωριστά στο κάθε σύνολο (train και test).

Όσον αφορά το νευρωνικό δίκτυο, τέθηκαν τα εξής χαρακτηριστικά:

- Δυο κρυφά επίπεδα με 128 και 64 νευρώνες αντίστοιχα
- Συνάρτηση ενεργοποίησης ReLU
- Αλγόριθμο Βελτιστοποίησης Adam
- Παράμετρο μάθησης 10^{-3}
- Παράμετρος early_stopping σε τιμή true
- Παράμετρος n_iter_no_changes σε τιμή 5
- Παράμετρος max_iter (μέγιστος αριθμός epochs) ίση με 50.
- Παράμετρος Random_state=42 ώστε κάθε φορά που εκτελείται ο κώδικας να διαχωρίζονται τα ίδια δείγματα για validation set, να υπάρχει ίδια σειρά των batches κάθε φορά που ετοιμάζονται για εκπαίδευση και να έχουμε ίδια αρχικοποίηση των βαρών κάθε φορά.

Τέλος, η απόδοση του μοντέλου αξιολογήθηκε με τις ακόλουθες μετρικές:

- Accuracy που εκφράζει το συνολικό ποσοστό των σωστών προβλέψεων

- F1-score(macro), που υπολογίζει τον μέσο όρο του f1-score κάθε κλάσης δίνοντας ίσο βάρος σε όλες τις κλάσεις
- F1-score(weighted) το οποίο σταθμίζει το F1-score κάθε κλάσης με βάση το πλήθος των δειγμάτων της
- Support όπου δείχνει το πλήθος των δειγμάτων της κάθε κλάσης στο test set.

Αυτές οι μετρικές παρέχονται από την βιβλιοθήκη **classification_report** της scikit-learn.

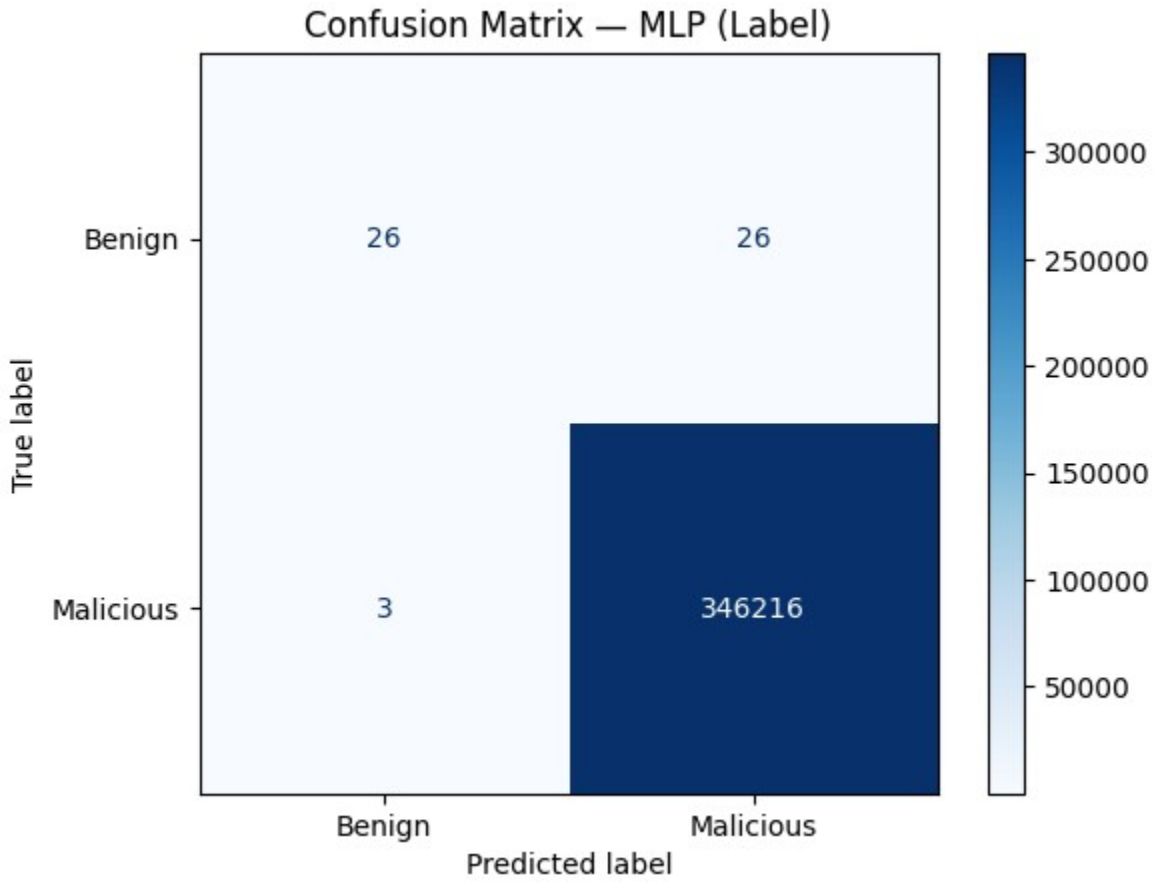
Τέλος, χρησιμοποιούμε και το confusion matrix για plotting.

Όλη η παραπάνω επεξεργασία χρησιμοποιείται με τον ίδιο τρόπο σε όλα τα datasets.

Για το stratified.parquet έχουμε τα εξής αποτελέσματα:

```
=== MLP on Label (stratified.parquet) ===
Accuracy:      0.9999
F1 (macro):    0.8210
F1 (weighted): 0.9999

Classification report:
              precision    recall  f1-score   support

      Benign     0.8966    0.5000    0.6420        52
   Malicious     0.9999    1.0000    1.0000    346219

    accuracy                         0.9999    346271
   macro avg     0.9482    0.7500    0.8210    346271
weighted avg     0.9999    0.9999    0.9999    346271
```

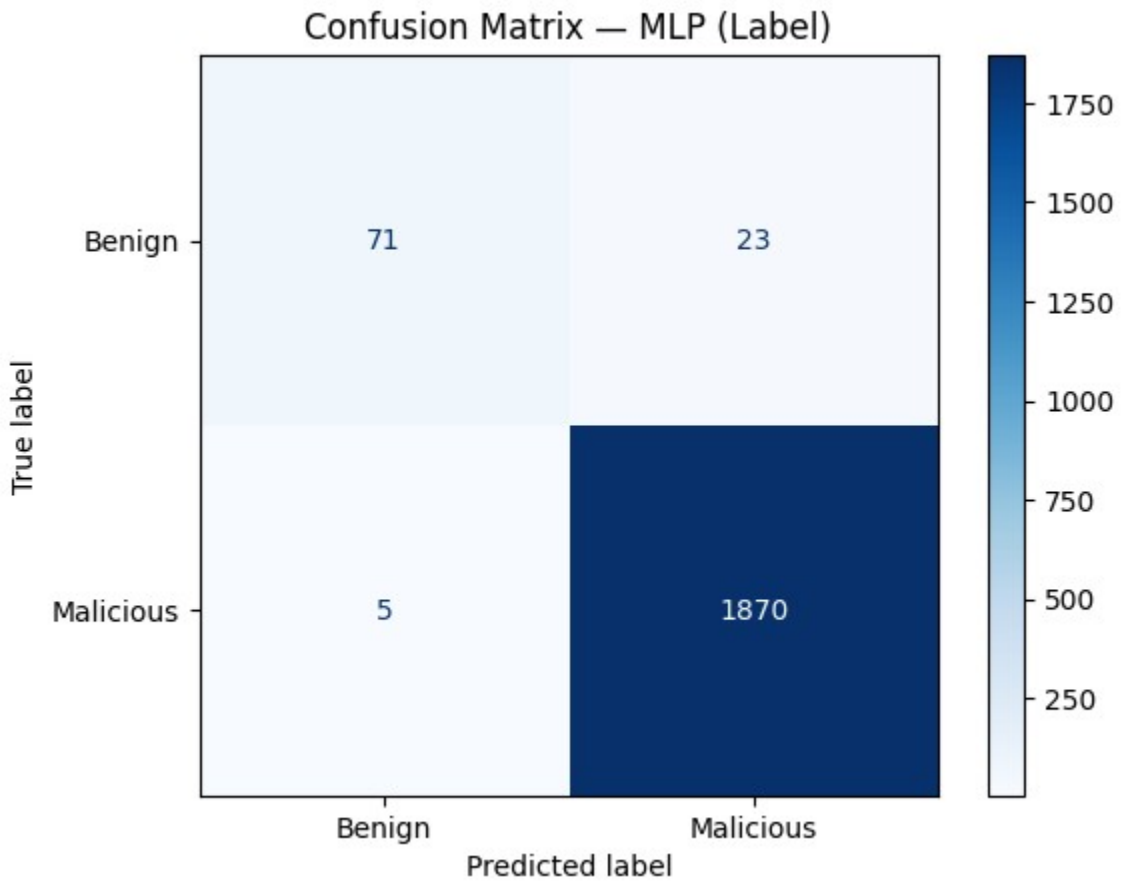
και το confusion matrix:



Για το **birch_representatives.parquet** επειδή έχουμε και σαν παράμετρο το cluster size, διαγράφουμε και αυτό σαν feature αλλά το χρησιμοποιούμε στην συνάρτηση train_test_split και όταν εκπαιδεύουμε το μοντέλο με το train dataset. Αυτό γίνεται για το λόγω του ότι το cluster_size το θέλουμε σαν βάρος δείγματος ώστε οι αντιπρόσωποι που προέρχονται από μεγαλύτερες συστάδες να επηρεάζουν αναλογικά περισσότερο την εκπαίδευση του ταξινομητή. Οι διαφορές της παραμέτρου αυτή σε πολλές συστάδες ήταν αρκετά μεγάλες και συνεπώς προστέθηκε και διπλό log για να εξομαλυνθεί.

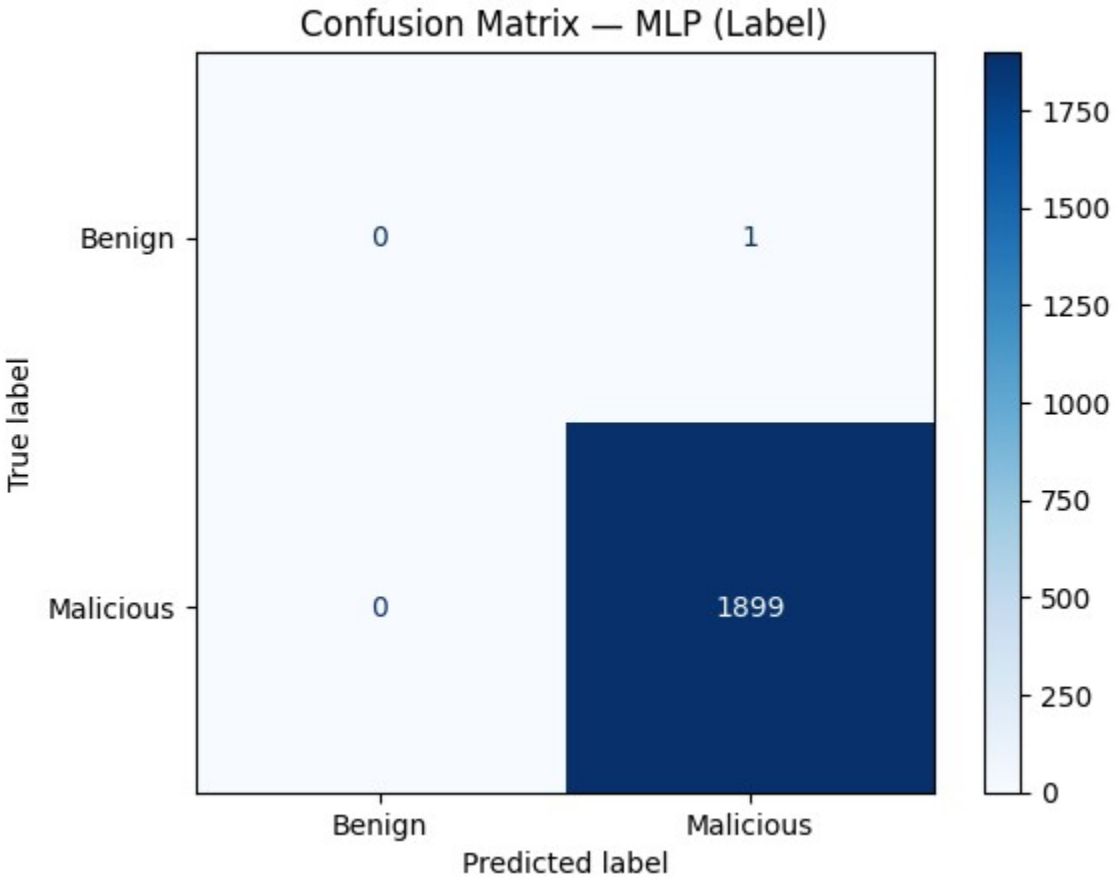
Παρακάτω παρουσιάζονται τα αποτελέσματα και το confusion matrix:

```
=== MLP on Label (birch_representatives.parquet) ===
Accuracy:      0.9858
F1 (macro):    0.9139
F1 (weighted): 0.9851

Classification report:
              precision    recall  f1-score   support

      Benign     0.9342    0.7553    0.8353        94
   Malicious     0.9878    0.9973    0.9926      1875

    accuracy                         0.9858      1969
   macro avg     0.9610    0.8763    0.9139      1969
weighted avg     0.9853    0.9858    0.9851      1969
```



Όσον αφορά το **minibatch_k_means_representatives.parquet**, το μοντέλο μαθαίνει απλά ότι θα είναι malicious η πρόβλεψη λόγω του ότι δεν υπάρχει πολύ μεγάλο support.

```
=== MLP on Label (minibatch_kmeans_representatives.parquet) ===
Accuracy:      0.9995
F1 (macro):    0.4999
F1 (weighted): 0.9992

Classification report:
              precision    recall  f1-score   support

      Benign     0.0000    0.0000    0.0000         1
   Malicious     0.9995    1.0000    0.9997      1899

    accuracy                         0.9995      1900
   macro avg     0.4997    0.5000    0.4999      1900
weighted avg     0.9989    0.9995    0.9992      1900
```



Αυτά δεν είναι καλά αποτελέσματα και αυτά οφείλονται λόγω του clustering.

Όσον αφορά το Traffic Label για το MLP:

Για το stratified.parquet, έχουμε τα παρακάτω αποτελέσματα:

```
=== MLP on Traffic Type (stratified.parquet) ===
Accuracy:      0.9968
F1 (macro):    0.7692
F1 (weighted): 0.9968

Classification report:
                      precision    recall  f1-score   support

               Audio     1.0000    0.7500    0.8571         8
          Background     0.0000    0.0000    0.0000         1
          Bruteforce     0.9801    0.9808    0.9805      1407
                 DoS     0.9994    0.9976    0.9985    299637
Information Gathering     0.9814    0.9987    0.9900     41535
               Mirai     0.9673    0.9179    0.9419      3640
                Text     1.0000    0.7500    0.8571         8
               Video     0.7778    0.4000    0.5283        35

            accuracy                         0.9968    346271
           macro avg     0.8382    0.7244    0.7692    346271
        weighted avg     0.9968    0.9968    0.9968    346271
```

Για το birch_representatives.parquet έχουμε τα παρακάτω αποτελέσματα

```
=== MLP on Traffic Type (birch_representatives.parquet) ===
Accuracy:      0.9345
F1 (macro):    0.7034
F1 (weighted): 0.9293

Classification report:
                      precision    recall  f1-score   support

               Audio     0.8750    0.9130    0.8936        23
          Bruteforce     0.8571    0.8308    0.8438        65
                 DoS     0.9563    0.9789    0.9674      1563
Information Gathering     1.0000    0.0526    0.1000        19
               Mirai     0.7956    0.7851    0.7903       228
                Text     0.7500    0.2727    0.4000        11
               Video     1.0000    0.8667    0.9286        60

            accuracy                         0.9345      1969
           macro avg     0.8906    0.6714    0.7034      1969
        weighted avg     0.9340    0.9345    0.9293      1969
```

Για το minibatch_kmeans_representatives.parquet, έχουμε τα εξής αποτελέσματα:

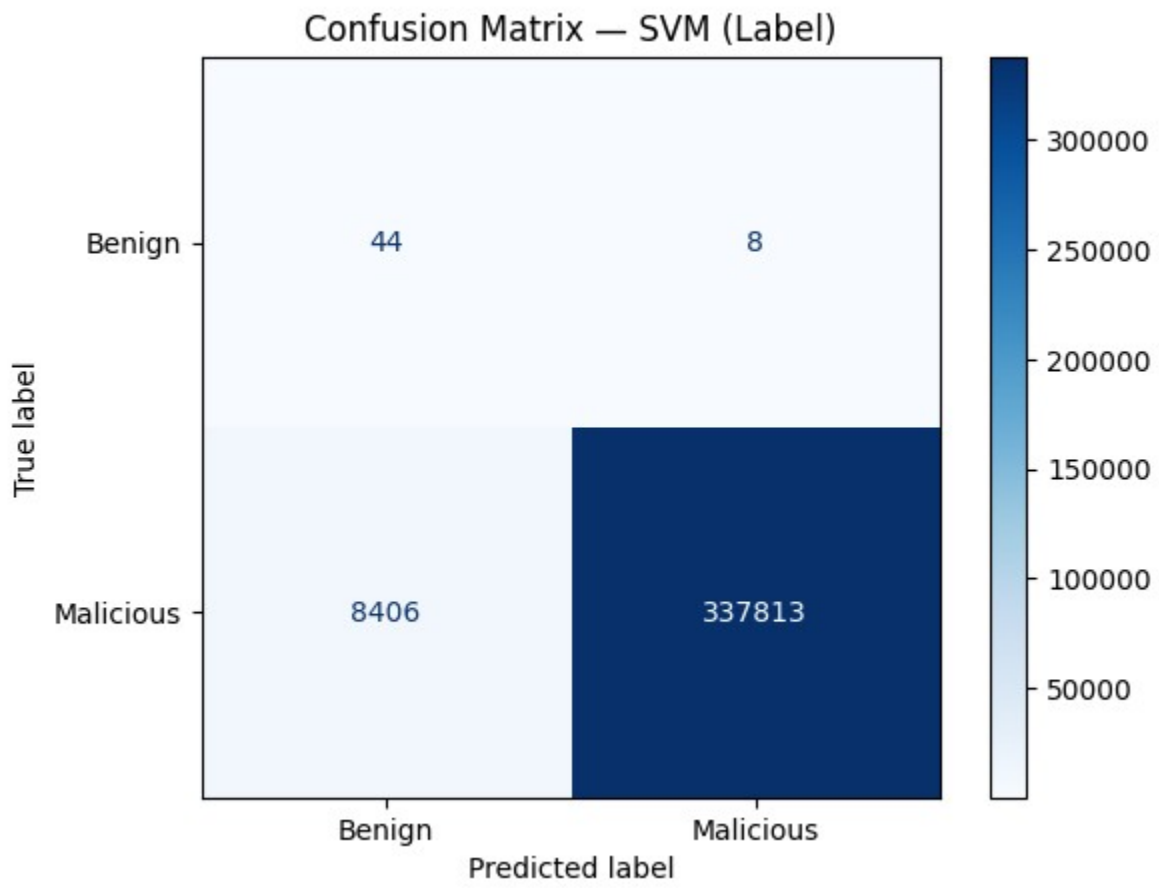
```
=== MLP on Traffic Type (minibatch_kmeans_representatives.parquet) ===
Accuracy:      0.8725
F1 (macro):    0.2330
F1 (weighted): 0.8131

Classification report:
                      precision    recall  f1-score   support

          Bruteforce     0.0000    0.0000    0.0000         9
                 DoS     0.8725    1.0000    0.9319      1656
Information Gathering     0.0000    0.0000    0.0000       212
               Mirai     0.0000    0.0000    0.0000        21

            accuracy                         0.8725      1898
           macro avg     0.2181    0.2500    0.2330      1898
        weighted avg     0.7613    0.8725    0.8131      1898
```

Ο κατηγοριοποιητής SVM έχει τα παραπάνω εξής χαρακτηριστικά:

- Class_weight = “balanced” ώστε το μοντέλο να λαμβάνει υπόψιν το πλήθος των δειγμάτων κάθε κλάσης
- Max_iter = 5000, που δηλώνει τον μέγιστο αριθμό επαναλήψεων ώστε να διασφαλιστεί η σύγκλιση του αλγορίθμου
- Random_state=42

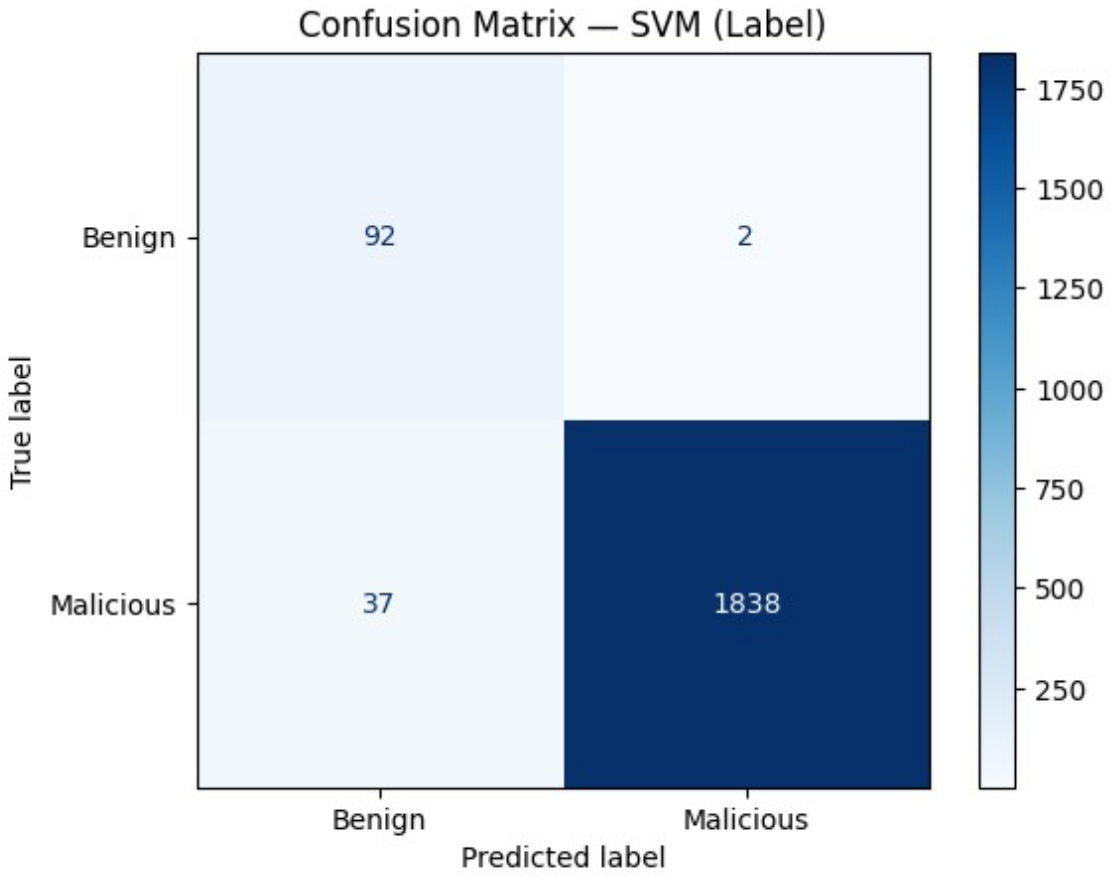
Για το χαρακτηριστικό Label, με το stratified.parquet παίρνουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Label (stratified.parquet) ===
Accuracy:      0.9757
F1 (macro):    0.4990
F1 (weighted): 0.9876

Classification report:
              precision    recall  f1-score   support

      Benign       0.01      0.85      0.01        52
   Malicious       1.00      0.98      0.99    346219

    accuracy                           0.98    346271
   macro avg       0.50      0.91      0.50    346271
weighted avg       1.00      0.98      0.99    346271
```



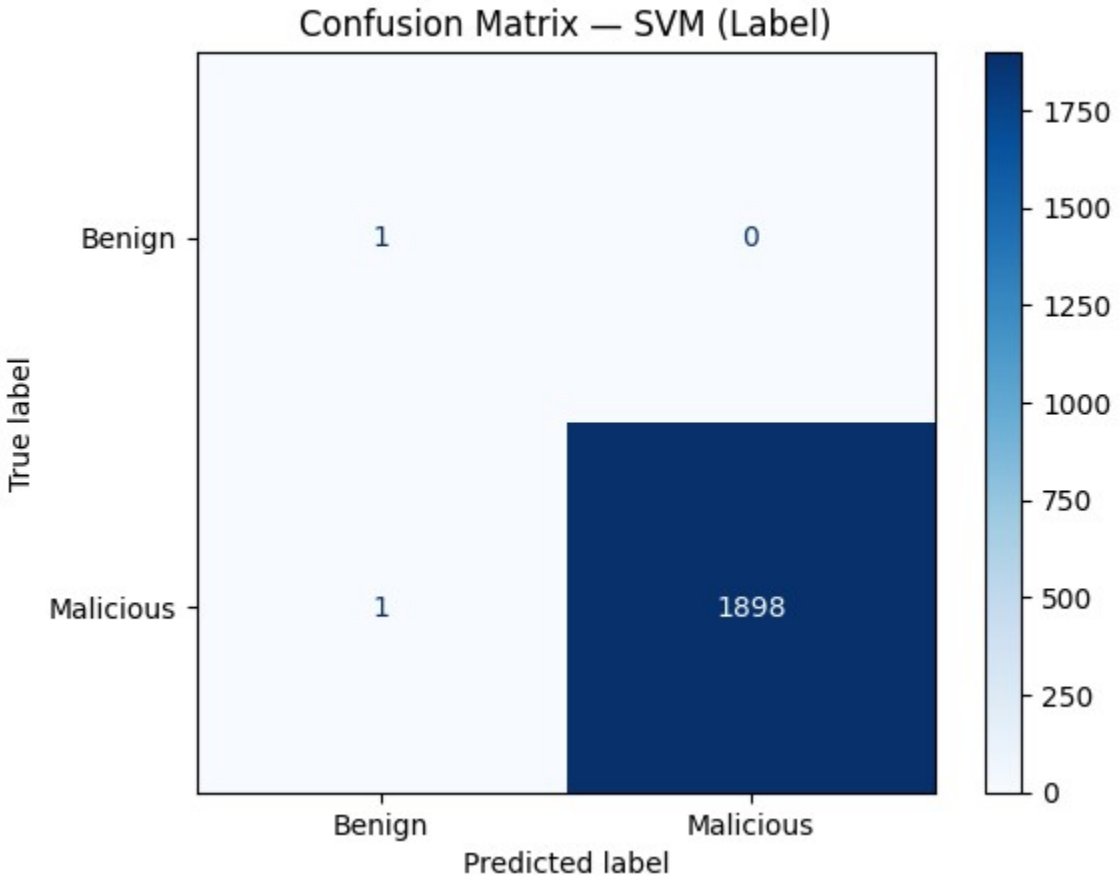
Για το χαρακτηριστικό Label, με το birch_representatives.parquet παίρνουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Label (birch_representatives.parquet) ===
Accuracy:      0.9802
F1 (macro):    0.9073
F1 (weighted): 0.9817

Classification report:
              precision    recall  f1-score   support

      Benign     0.7132    0.9787    0.8251        94
   Malicious     0.9989    0.9803    0.9895      1875

    accuracy                         0.9802      1969
   macro avg     0.8560    0.9795    0.9073      1969
weighted avg     0.9853    0.9802    0.9817      1969
```



Για το χαρακτηριστικό Label, με το minibatch_kmens_representatives.parquet παίρνουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Label (minibatch_kmeans_representatives.parquet) ===
Accuracy:      0.9995
F1 (macro):    0.8332
F1 (weighted): 0.9996

Classification report:
              precision    recall  f1-score   support

      Benign     0.5000    1.0000    0.6667         1
   Malicious     1.0000    0.9995    0.9997      1899

    accuracy                         0.9995      1900
   macro avg     0.7500    0.9997    0.8332      1900
weighted avg     0.9997    0.9995    0.9996      1900
```



Όσον αφορά το traffic type, για το stratified.parquet έχουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Traffic Type (stratified.parquet) ===
Accuracy:      0.9804
F1 (macro):    0.8254
F1 (weighted): 0.9821

Classification report:
                      precision    recall  f1-score   support

           Bruteforce     0.8586    0.9581    0.9056      1407
                  DoS     0.9974    0.9835    0.9904    299637
Information Gathering     0.9502    0.9965    0.9728     41535
               Mirai      0.3565    0.5505    0.4328      3640

             accuracy                         0.9804    346219
            macro avg     0.7907    0.8722    0.8254    346219
         weighted avg     0.9844    0.9804    0.9821    346219
```

Για το birch_represenatives.parquet έχουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Traffic Type (birch_representatives.parquet) ===
Accuracy:      0.7897
F1 (macro):    0.5969
F1 (weighted): 0.8049

Classification report:
                      precision    recall  f1-score   support

               Audio     0.8750    0.9130    0.8936        23
           Bruteforce     0.2531    0.6308    0.3612        65
                  DoS     0.9348    0.8624    0.8972      1563
Information Gathering     0.2037    0.5789    0.3014        19
               Mirai      0.3553    0.3070    0.3294       228
                Text      0.3913    0.8182    0.5294        11
               Video      0.8209    0.9167    0.8661        60

             accuracy                         0.7897      1969
            macro avg     0.5477    0.7182    0.5969      1969
         weighted avg     0.8309    0.7897    0.8049      1969
```

Για το minibatch_representatives.parquet έχουμε τα παρακάτω αποτελέσματα:

```
=== SVM on Traffic Type (minibatch_kmeans_representatives.parquet) ==
Accuracy:      0.9009
F1 (macro):    0.4319
F1 (weighted): 0.8923

Classification report:
                      precision    recall  f1-score   support

           Bruteforce     0.1538    0.2222    0.1818         9
                  DoS     0.9260    0.9825    0.9534      1656
Information Gathering     0.8977    0.3726    0.5267       212
               Mirai      0.0500    0.0952    0.0656        21

            accuracy                         0.9009      1898
           macro avg     0.5069    0.4181    0.4319      1898
        weighted avg     0.9095    0.9009    0.8923      1898
```

ΣΥΓΚΡΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ

Όσον αφορά το Label:

Για το stratified.parquet, το MLP φαίνεται καλύτερο καθώς έχει καλύτερο accuracy.

Όσον αφορά το birch_representatives.parquet, τα δυο μοντέλα λειτουργούν πολύ καλά και έχουμε σχεδόν τα ίδια αποτελέσματα.

Για το minibatch_kmeans_represnetatives.parquet, το dataset είναι ακραία μη ισορροπημένο. Το test set έχει μόνο ένα δείγμα για benign το οποίο το SVM το πιάνει ενώ το MLP όχι. Επίσης, το SVM εμφανίζει

μεγαλύτερο macro-F1 σε σχέση με τον MLP αλλά το αποτέλεσμα δεν θεωρείται αξιόπιστο, καθώς το μοντέλο περιλαμβάνει ελάχιστα δείγματα της μειοψηφικής κλάσης.

Σε σύγκριση των datasets, το stratified.parquet είναι το καλύτερο συνολικά για την πρόβλεψη του Label.

Όσον αφορά το Traffic Type:

Το MLP λειτουργεί καλύτερα πάλι με το stratified.parquet ενώ παρουσιάζει μια μικρή απώλεια απόδοσης με το birch_representatives.parquet.

Οσον αφορά το SVM, το stratified.parquet τρέχει καλύτερα με αυτό το μοντέλο ενώ αυτό δεν ισχύει με το birch_representatives.parquet. Το ίδιο ισχύει και για το minibatch_kmeans-representatives.parquet.

Συνολικά, παρατηρείται ότι η ποιότητα του dataset παίζει καθοριστικό ρόλο στην απόδοση των ταξινομητών, συχνά σημαντικότερο από την επιλογή του ίδιου του μοντέλου. Το stratified.parquet διατηρεί την αρχική κατανομή και οδηγεί στα πιο αξιόπιστα αποτελέσματα, ενώ το Birch representatives αποτελεί έναν αποδοτικό συμβιβασμό μεταξύ μείωσης δεδομένων και ακρίβειας. Αντίθετα, το MiniBatch K-Means, αν και ιδιαίτερα αποδοτικό υπολογιστικά, δεν διατηρεί επαρκώς τις μειοψηφικές κλάσεις, γεγονός που επηρεάζει αρνητικά την αξιολόγηση.