Descriptive Analytics

In [1]:
```
import warnings

import numpy as np
import pandas as pd
from scipy.stats import uniform, randint
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.figure_factory as ff
import plotly.offline as pyo
import plotly.graph_objs as go
from plotly.subplots import make_subplots

pyo.init_notebook_mode()

import missingno as msno

from scipy.stats.contingency import association

from sklearn.preprocessing import LabelEncoder

%matplotlib inline
```

In [2]:
```
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 500)
plt.style.use('ggplot')
pd.options.plotting.backend = 'plotly'
```

In [3]:
```
data = (pd
        .read_csv("../../dataset/ddos_sdn/dataset_sdn.csv")
       )
```

1. Structure Investigation

In [4]:
```
data.head()
```

Out[4]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	p
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428310	
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	14424046	
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	

In [5]:
```
(data
 .shape
)
```

Out[5]:
```
(104345, 23)
```

In [6]:
```
(data
 .info()
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   dt           104345 non-null  int64  
 1   switch       104345 non-null  int64  
 2   src          104345 non-null  object 
 3   dst          104345 non-null  object 
 4   pktcount     104345 non-null  int64  
 5   bytecount    104345 non-null  int64  
 6   dur          104345 non-null  int64  
 7   dur_nsec     104345 non-null  int64  
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64  
 10  packetins    104345 non-null  int64  
 11  pktperflow   104345 non-null  int64  
 12  byteperflow  104345 non-null  int64  
 13  pktrate      104345 non-null  int64  
 14  Pairflow     104345 non-null  int64  
 15  Protocol     104345 non-null  object 
 16  port_no      104345 non-null  int64  
 17  tx_bytes     104345 non-null  int64  
 18  rx_bytes     104345 non-null  int64  
 19  tx_kbps      104345 non-null  int64  
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64  
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

In [7]:
```
(data
 .describe()
)
```

Out[7]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000	1
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234	
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036	
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000	
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000	
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000	
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000	
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000	

In [8]:
```
(data
 .isna()
 .sum()
)
```

Out[8]:
```
dt               0
switch           0
src              0
dst              0
pktcount         0
bytecount        0
dur              0
dur_nsec         0
tot_dur          0
flows            0
packetins        0
pktperflow       0
byteperflow      0
pktrate          0
Pairflow         0
Protocol         0
port_no          0
tx_bytes         0
rx_bytes         0
tx_kbps          0
rx_kbps        506
tot_kbps       506
label            0
dtype: int64
```

In [9]:
```
pd.value_counts(data.dtypes)
```
Out[9]:
```
int64      17
object      3
float64     3
dtype: int64
```

1.1. Structure of non-numerical features

In [10]:
```
(data
 .loc[:, data.columns!='label']
 .select_dtypes(exclude="number")
 .head()
)
```
Out[10]:

	src	dst	Protocol
0	10.0.0.1	10.0.0.8	UDP
1	10.0.0.1	10.0.0.8	UDP
2	10.0.0.2	10.0.0.8	UDP
3	10.0.0.2	10.0.0.8	UDP
4	10.0.0.2	10.0.0.8	UDP

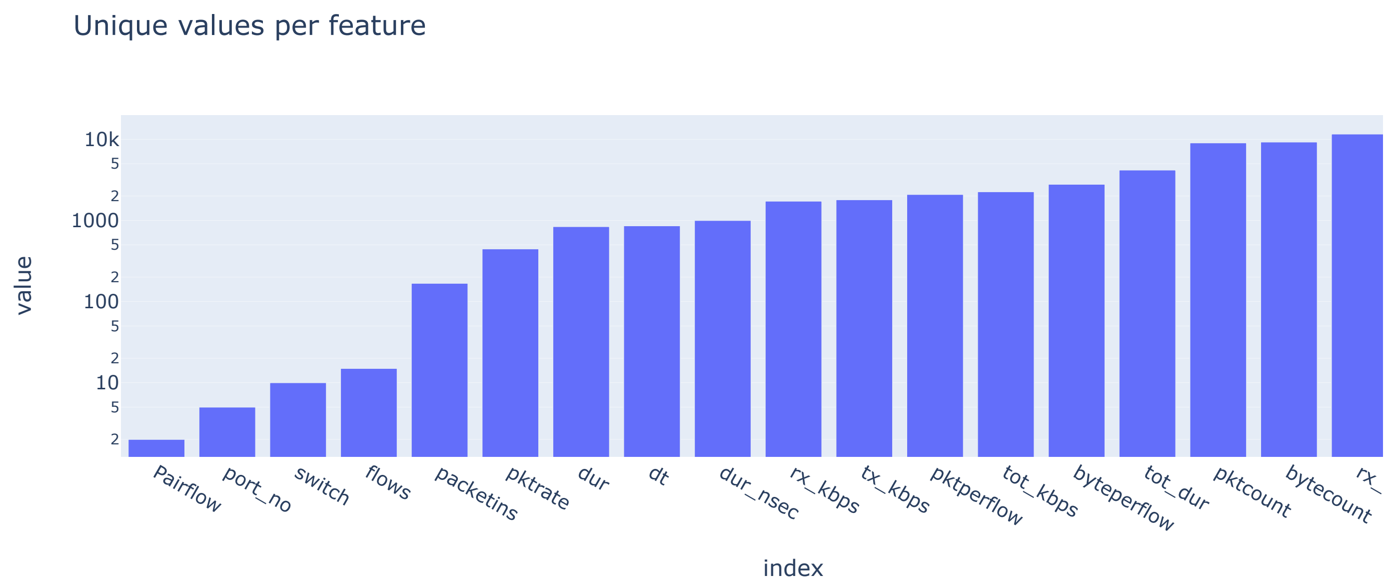
In [11]:
```
(data
 .loc[:, data.columns!='label']
 .describe(exclude="number")
)
```
Out[11]:

	src	dst	Protocol
count	104345	104345	104345
unique	19	18	3
top	10.0.0.3	10.0.0.7	ICMP
freq	11491	18020	41321

1.2. Structure of numerical features

In [12]:
```
unique_values = (data
                 .loc[:, data.columns!='label']
                 .select_dtypes(include="number")
                 .nunique()
                 .sort_values())
unique_values
```
Out[12]:
```
Pairflow            2
port_no             5
switch             10
flows              15
packetins         168
pktrate           446
dur               840
dt                859
dur_nsec         1000
rx_kbps          1730
tx_kbps          1800
pktperflow       2092
tot_kbps         2259
byteperflow      2793
tot_dur          4183
pktcount         9045
bytecount        9271
rx_bytes        11625
tx_bytes        12257
dtype: int64
```

In [13]:
```
#figsize = (15,4)
(unique_values
 .plot
 .bar(log_y=True, title="Unique values per feature")
 .update_layout(showlegend=False, width=1000, height=400)
)
```



2. Quality Investigation

In [14]:
```
n_duplicates = (data
                .duplicated()
                .sum())
print(f"You seem to have {n_duplicates} duplicates in your database.")
```

```
You seem to have 5091 duplicates in your database.
```

In [15]:
```
(data
 .loc[data.duplicated()]
 .head()
)
```

Out[15]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflow	r
13	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428310	
15	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428310	
30	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	
34	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	
40	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427244	

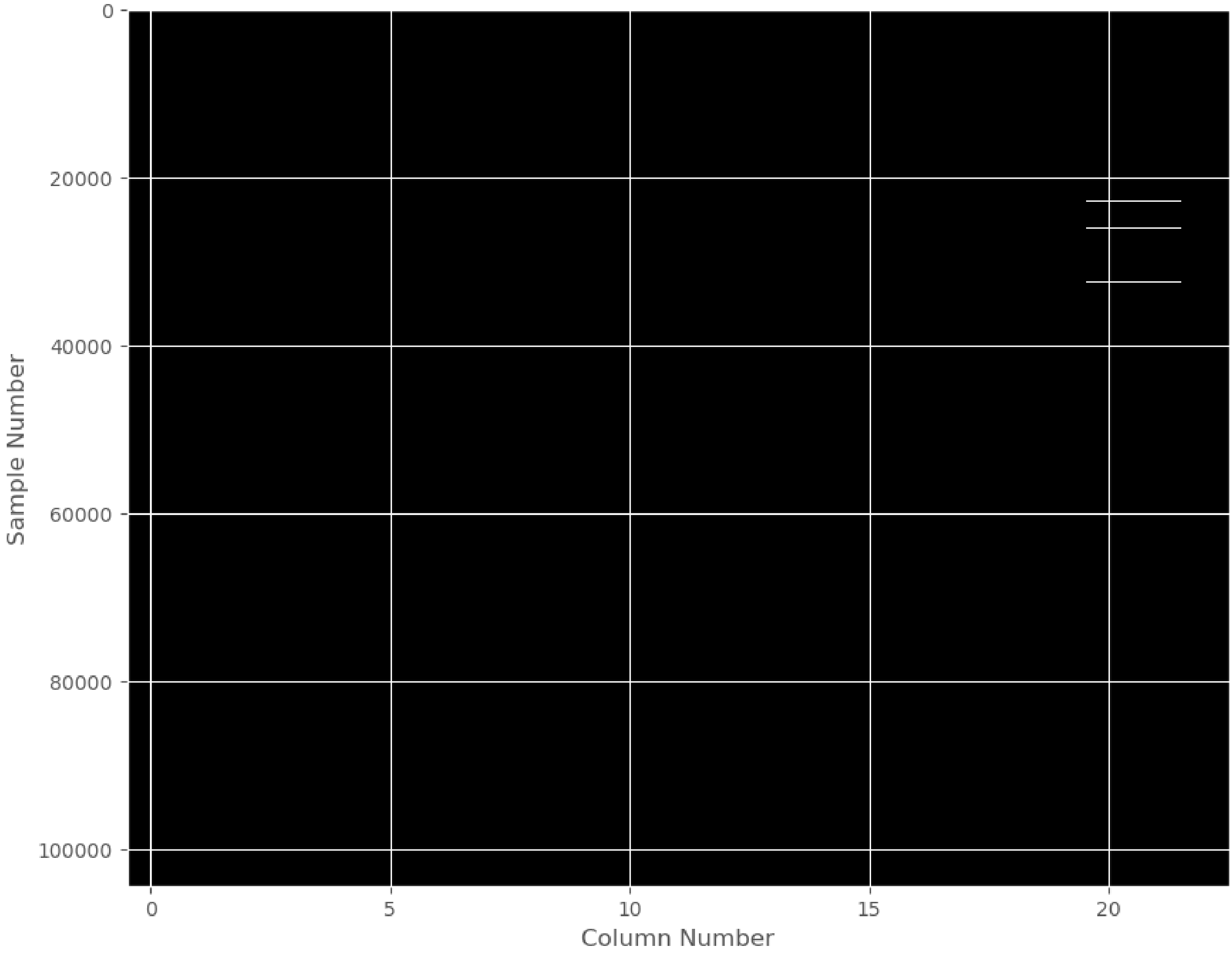
Insights:

1. Drop duplicates as duplicated rows can introduce biases and inaccuracies in the model training process.
2. Duplicated rows can artificially inflate the importance of certain observations, leading to overfitting.
3. Dropping 4.88% of total rows

2.2. Missing values

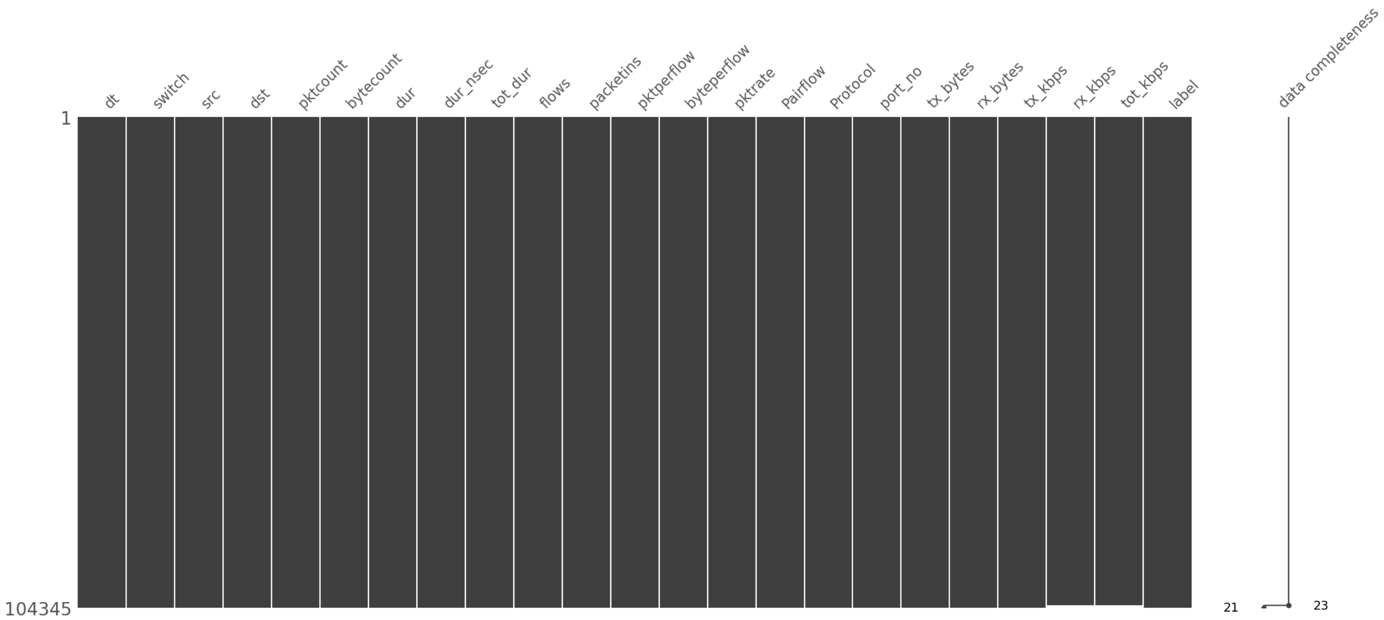
In [16]:
```
#Per sample
plt.figure(figsize=(10,8))
plt.imshow(data.isna(), aspect="auto", interpolation="nearest", cmap="gray")
plt.xlabel("Column Number")
plt.ylabel("Sample Number")
```

Out[16]:
```
Text(0, 0.5, 'Sample Number')
```



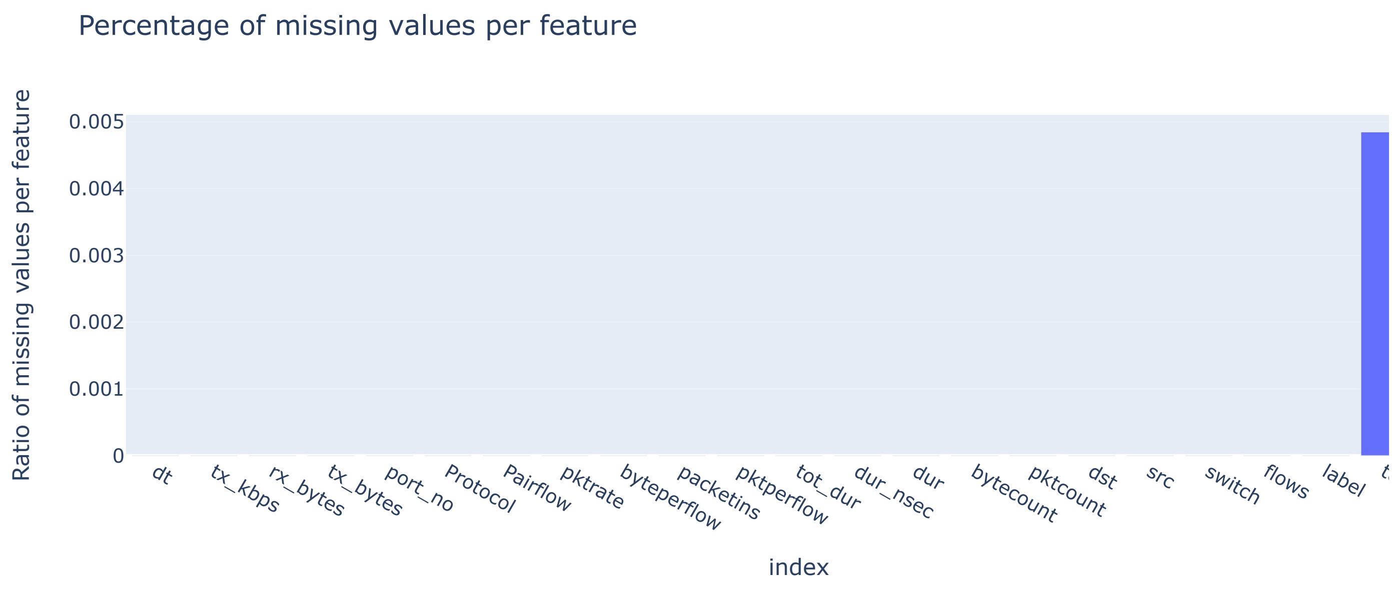
In [17]:

```
msno.matrix(data, labels=True, sort="descending");
```



In [18]:

```
#Per Feature
(data
 .isna()
 .mean()
 .sort_values()
 .plot(kind='bar', title='Percentage of missing values per feature')
 .update_layout(showlegend=False, width=1000, height=400)
 .update_yaxes(title="Ratio of missing values per feature")
)
```



Insights:

1. Only `tot_kbps` and `rx_kbps` contain missing value
2. Missing value less than 0.5% of total rows

3. Content Investigation

3.1. General Overview of Histogram

In [19]:
```
pd.options.plotting.backend = 'plotly'

plot_rows=4
plot_cols=5
fig = make_subplots(rows=plot_rows, cols=plot_cols, subplot_titles=(data.columns.tolist()))

# add traces
x = 0
for i in range(1, plot_rows + 1):
    for j in range(1, plot_cols + 1):
        fig.add_trace(go.Histogram(x = data[data.columns[x]].values,
                                   name = data.columns[x],),
                      row=i,
                      col=j)
        x=x+1

# Format and show fig
fig.update_layout(height=1200, width=1000, title_text='Quick Overview of Variables Distribution', showlegend=Fa
fig.show()
```

3.2. Feature patterns

In [ ]:
```
cols_continuous = (data
                   .loc[:, data.columns != 'label']
                   .select_dtypes(include="number")
                   .nunique()
                   .ge(25)
                  )
cols_continuous
```

Out[ ]:

```
dt              True
switch         False
pktcount        True
bytecount       True
dur             True
dur_nsec        True
tot_dur         True
flows          False
packetins       True
pktperflow      True
byteperflow     True
pktrate         True
Pairflow       False
port_no        False
tx_bytes        True
rx_bytes        True
tx_kbps         True
rx_kbps         True
tot_kbps        True
dtype: bool
```

Insights:

1. Possible continuous values (unique values >= 25):

`dt`, `pktcount`, `bytecount`, `dur`, `dur_nsec`, `tot_dur`, `packetins`, `pktperflow`, `byteperflow`, `pktrate`, `tx_bytes`, `rx_bytes`, `tx_kbps`, `rx_kbps`, `tot_kbps`

1. Possible categorical values (unique values < 25, on top of `src`, `dst`, `Protocol`):

`switch`, `flows`, `Pairflow`, `port_no`

However, based on external reference, it seems that `flows` and `Pairflow` might not be category!

3.2.1. Continuous features

In [21]:

```
(data
 [list(cols_continuous.index[cols_continuous == True])]
 .corr(method='spearman')
 .style
 .background_gradient(cmap="viridis", axis=None)
)
```

Out[21]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_b
dt	1.000000	-0.196057	-0.333296	0.247739	-0.133751	0.247493	0.057449	-0.260257	-0.332987	-0.276766	0.26
pktcount	-0.196057	1.000000	0.872567	0.137107	0.036113	0.137305	0.202976	0.595541	0.598707	0.602500	-0.11
bytecount	-0.333296	0.872567	1.000000	-0.028295	0.041895	-0.028061	-0.022006	0.605611	0.752251	0.618310	-0.25
dur	0.247739	0.137107	-0.028295	1.000000	-0.017575	0.999994	0.044583	-0.328619	-0.299349	-0.377494	0.26
dur_nsec	-0.133751	0.036113	0.041895	-0.017575	1.000000	-0.015166	-0.043570	0.048388	0.055893	0.049624	-0.05
tot_dur	0.247493	0.137305	-0.028061	0.999994	-0.015166	1.000000	0.044486	-0.328457	-0.299188	-0.377328	0.26
packetins	0.057449	0.202976	-0.022006	0.044583	-0.043570	0.044486	1.000000	0.111686	-0.009969	0.112389	0.31
pktperflow	-0.260257	0.595541	0.605611	-0.328619	0.048388	-0.328457	0.111686	1.000000	0.912005	0.994212	-0.21
byteperflow	-0.332987	0.598707	0.752251	-0.299349	0.055893	-0.299188	-0.009969	0.912005	1.000000	0.905798	-0.27
pktrate	-0.276766	0.602500	0.618310	-0.377494	0.049624	-0.377328	0.112389	0.994212	0.905798	1.000000	-0.22
tx_bytes	0.262838	-0.115539	-0.252420	0.262301	-0.057304	0.262175	0.313367	-0.213125	-0.275900	-0.229467	1.00
rx_bytes	0.161447	-0.031831	-0.125555	0.207248	-0.017563	0.207262	0.267976	-0.114844	-0.157462	-0.126731	0.44
tx_kbps	0.120652	-0.015063	-0.090290	-0.039285	-0.064939	-0.039506	0.154028	0.016757	-0.037484	0.015453	0.69
rx_kbps	0.081517	0.024581	-0.042715	-0.074940	-0.060700	-0.075142	0.157652	0.074449	0.018146	0.074995	0.24
tot_kbps	-0.016597	0.043244	0.041104	-0.140704	-0.065463	-0.140916	0.113200	0.139388	0.115193	0.142774	0.46

Insights:

1. Examine features with high correlation in section 3.4

In [22]:

```
df_continuous = (data
                 [cols_continuous[cols_continuous].index]
```

```
                )
df_continuous.shape
```

Out[22]:

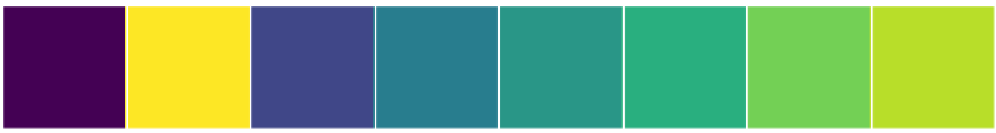
```
(104345, 15)
```

In [23]:

```
index_vals = data['label'].astype('category').cat.codes

fig = go.Figure(data=go.Splom(
                dimensions=[dict(label=f'{col}',
                                 values=data[f'{col}']) for col in df_continuous.columns],
                showupperhalf=False, # remove plots on diagonal
                text=data['label'],
                marker=dict(color=index_vals,
                            showscale=False, # colors encode categorical variables
                            line_color='white', line_width=0.5)
                ))

fig.update_layout(
    title='DDOS attack',
    width=1000,
    height=1000,
    font=dict(
        size=6,
    ))

fig.update_xaxes(tickfont=dict(
                        size=6,))
fig.update_yaxes(title_standoff=10,
               tickfont=dict(
                        size=6,))
fig.show()
```

In [ ]:

```
pd.options.plotting.backend = 'matplotlib'
color_palette = ["#440154", "#fde725", "#404788", "#287d8e", "#299687", "#29af7f", "#73d055", "#b8de29", ]
fp = matplotlib.font_manager.FontProperties(
    fname='/Fonts/roboto/Roboto-Condensed.ttf')
sns.set_palette(color_palette)
sns.set_style("darkgrid")

sns.set_palette(color_palette)
sns.palplot(sns.color_palette())
```



In [ ]:

```
def continuous_plot(df: pd.DataFrame,
                    col: str,
                    title: str,
                    symb: str):

    with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":
        fig, ax = plt.subplots(2, 1, sharex=True, figsize=(9,5),gridspec_kw={"height_ratios": (.2, .8)})

        ax[0].set_title(title,fontsize=18)
        (df
         [[col]]
         .boxplot(ax=ax[0], vert=False))

        ax[0].set(yticks=[])
        (df
         [[col]]
         .plot
         .hist(ax=ax[1], bins=50, edgecolor="#1D1EA2"))
        ax[1].set_xlabel(col, fontsize=16)

        plt.axvline(df[col].mean(), color='darkgreen', linestyle='--',linewidth=2.2, label='mean=' + str(np.rou
        plt.axvline(df[col].median(), color='red', linestyle='--',linewidth=2.2, label='median='+ str(np.round
        plt.axvline(df[col].mode()[0], color='purple', linestyle='--',linewidth=2.2, label='mode='+ str(np.rou
        plt.legend(bbox_to_anchor=(1, 1.03), ncol=1, fontsize=12, fancybox=True, shadow=True, frameon=True)

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

In [ ]:

```
def outlier_thresholds(df: pd.DataFrame,
                       col: str,
                       q1: float = 0.05,
                       q3: float = 0.95):
    #1.5 as multiplier is a rule of thumb. Generally, the higher the multiplier,
```

```python
    #the outlier threshold is set farther from the third quartile, allowing fewer data points to be classified

    return (df[col].quantile(q1) - 1.5 * (df[col].quantile(q3) - df[col].quantile(q1)),
            df[col].quantile(q3) + 1.5 * (df[col].quantile(q3) - df[col].quantile(q1)))
```

In [ ]:
```python
def loc_potential_outliers(df: pd.DataFrame,
                           col: str):

    low, high = outlier_thresholds(df, col)
    res = df.loc[(df[col] < low) | (df[col] > high)]
    print(f'Detected total of {len(res)} potential outliers')
    return res
```

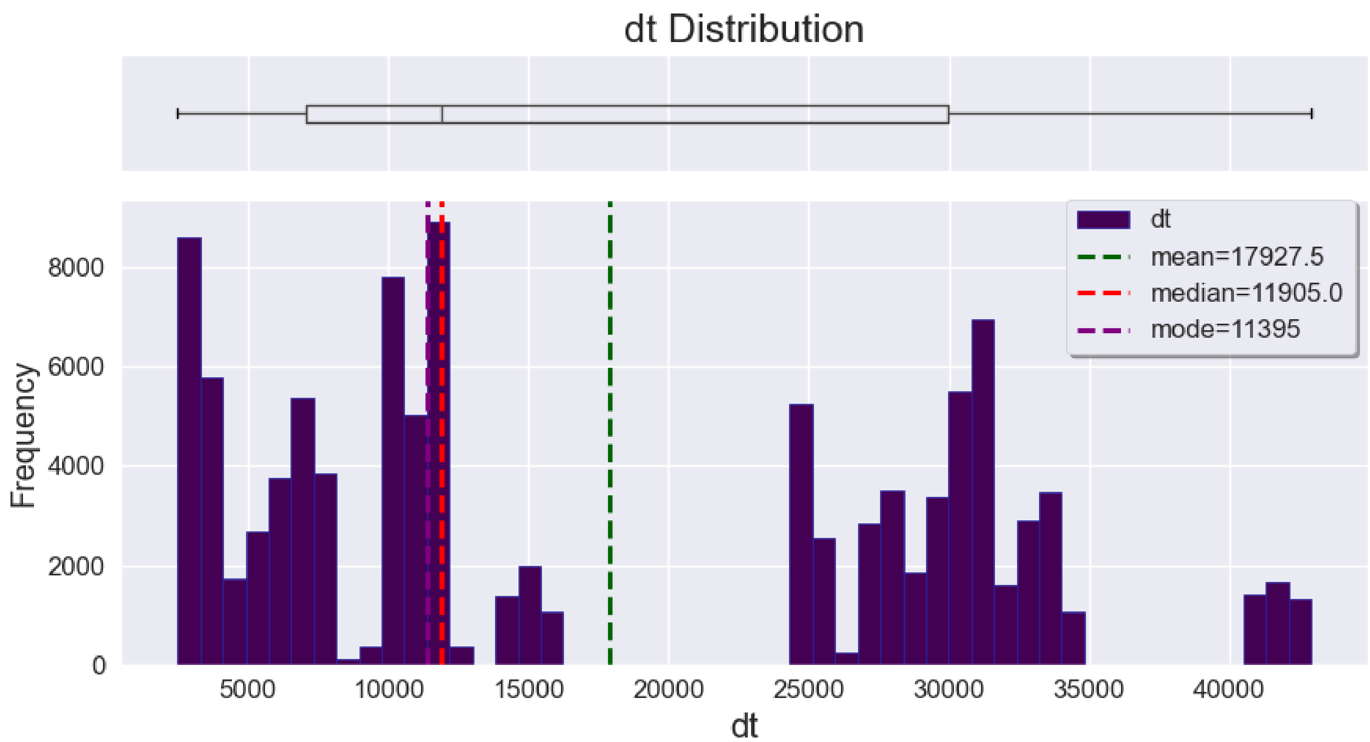
In [ ]:
```python
def any_potential_outlier(df: pd.DataFrame,
                          col: str) -> int:

    low, high = outlier_thresholds(df, col)
    if (df
        .loc[(df[col] > high) | (df[col] < low)]
        .any(axis=None)):
        return df.loc[(df[col] > high) | (df[col] < low)].shape[0]
    else:
        return 0
```

In [ ]:
```python
def delete_potential_outlier(df: pd.DataFrame,
                             col: str) -> pd.DataFrame:

    low, high = outlier_thresholds(df, col)
    df.loc[(df[col]>high) | (df[col]<low),col] = np.nan
    return df
```

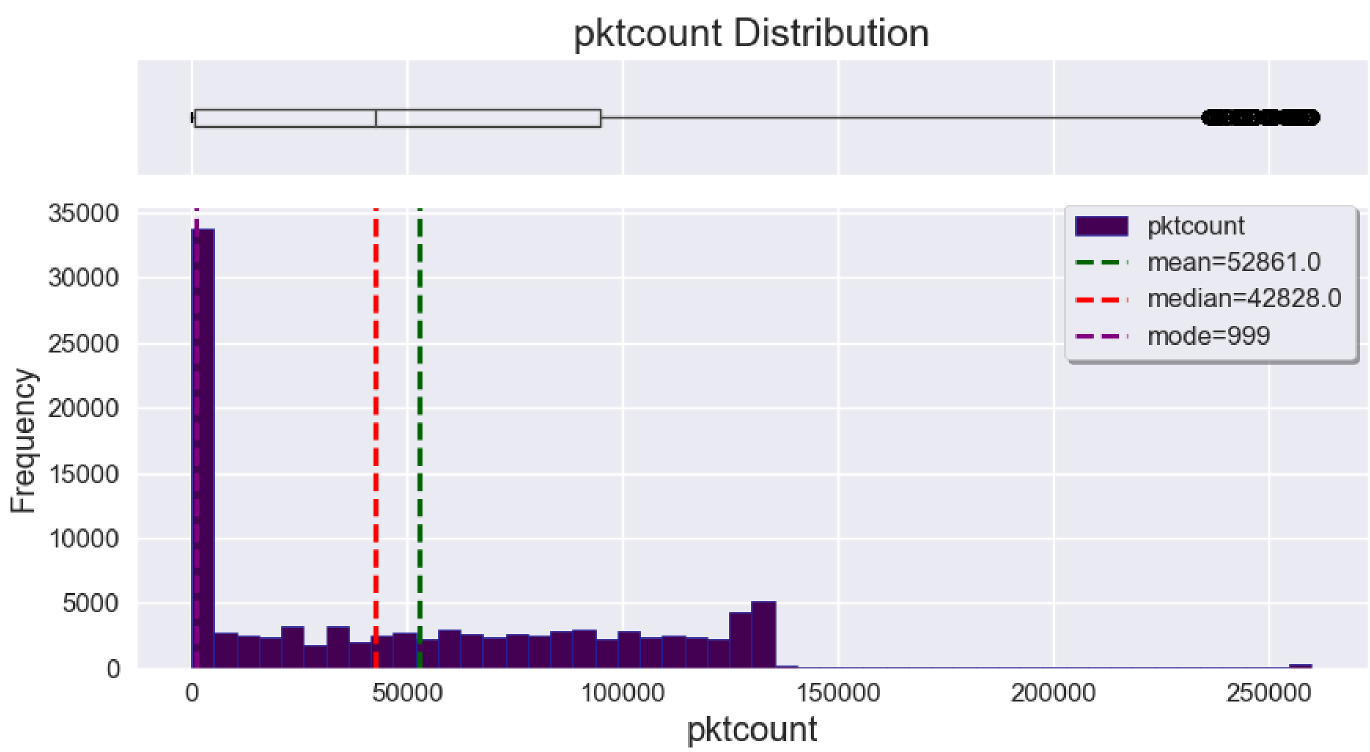
In [ ]:
```python
for col in df_continuous.columns:
    print(f'Column {col}: Detected a total of {any_potential_outlier(df_continuous, col)} potential outliers')
```

```
Column dt: Detected a total of 0 potential outliers
Column pktcount: Detected a total of 0 potential outliers
Column bytecount: Detected a total of 0 potential outliers
Column dur: Detected a total of 0 potential outliers
Column dur_nsec: Detected a total of 0 potential outliers
Column tot_dur: Detected a total of 0 potential outliers
Column packetins: Detected a total of 0 potential outliers
Column pktperflow: Detected a total of 188 potential outliers
Column byteperflow: Detected a total of 140 potential outliers
Column pktrate: Detected a total of 188 potential outliers
Column tx_bytes: Detected a total of 39 potential outliers
Column rx_bytes: Detected a total of 39 potential outliers
Column tx_kbps: Detected a total of 302 potential outliers
Column rx_kbps: Detected a total of 99 potential outliers
Column tot_kbps: Detected a total of 0 potential outliers
```

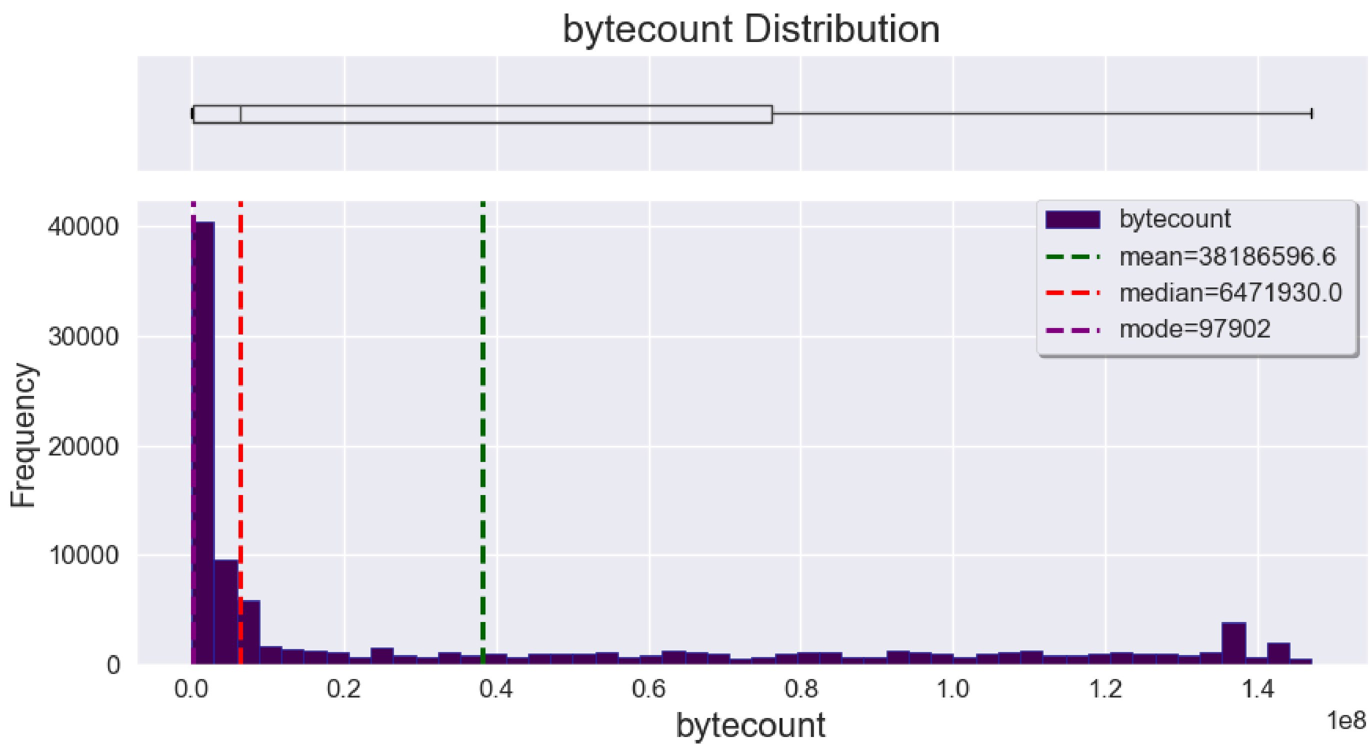
In [31]:
```python
pd.options.plotting.backend = 'matplotlib'
continuous_plot(df_continuous.join(data.label), 'dt', "dt Distribution", "")
```



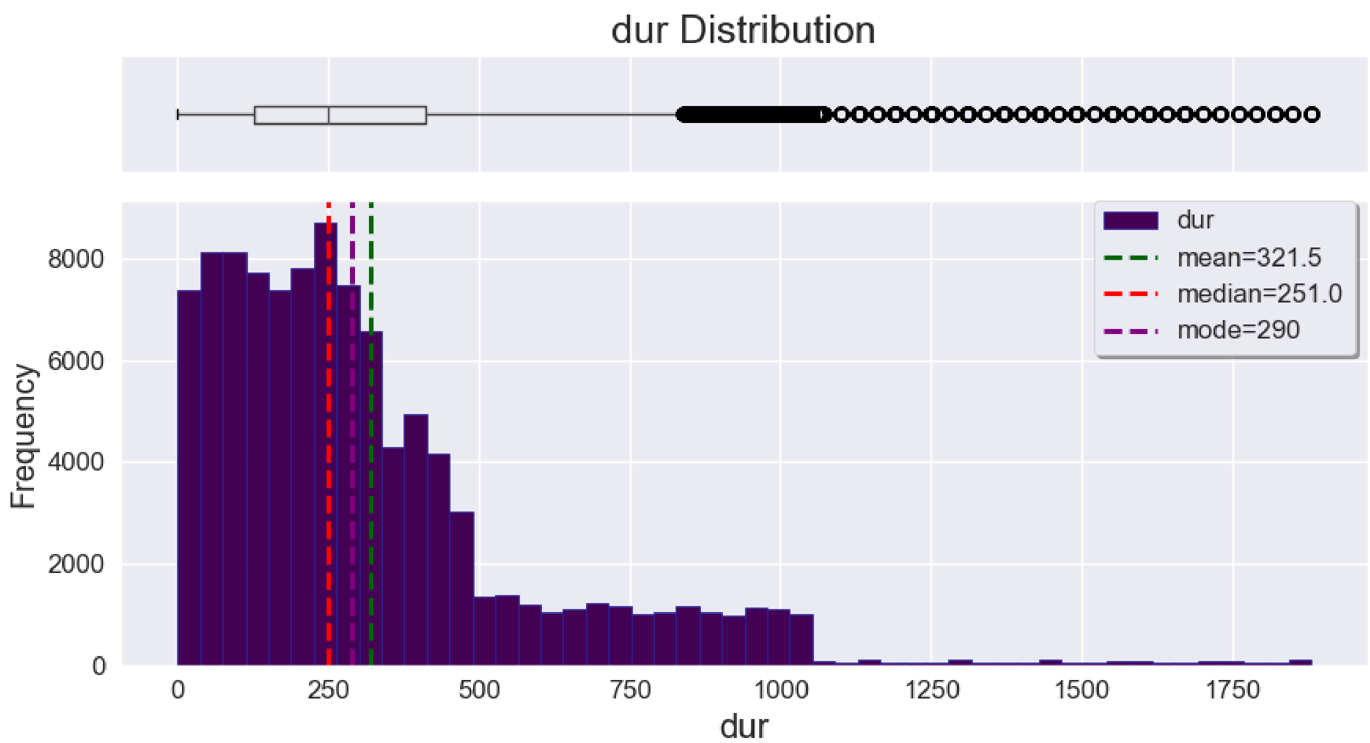
In [32]:
```
continuous_plot(df_continuous.join(data.label), 'pktcount', "pktcount Distribution", "")
```



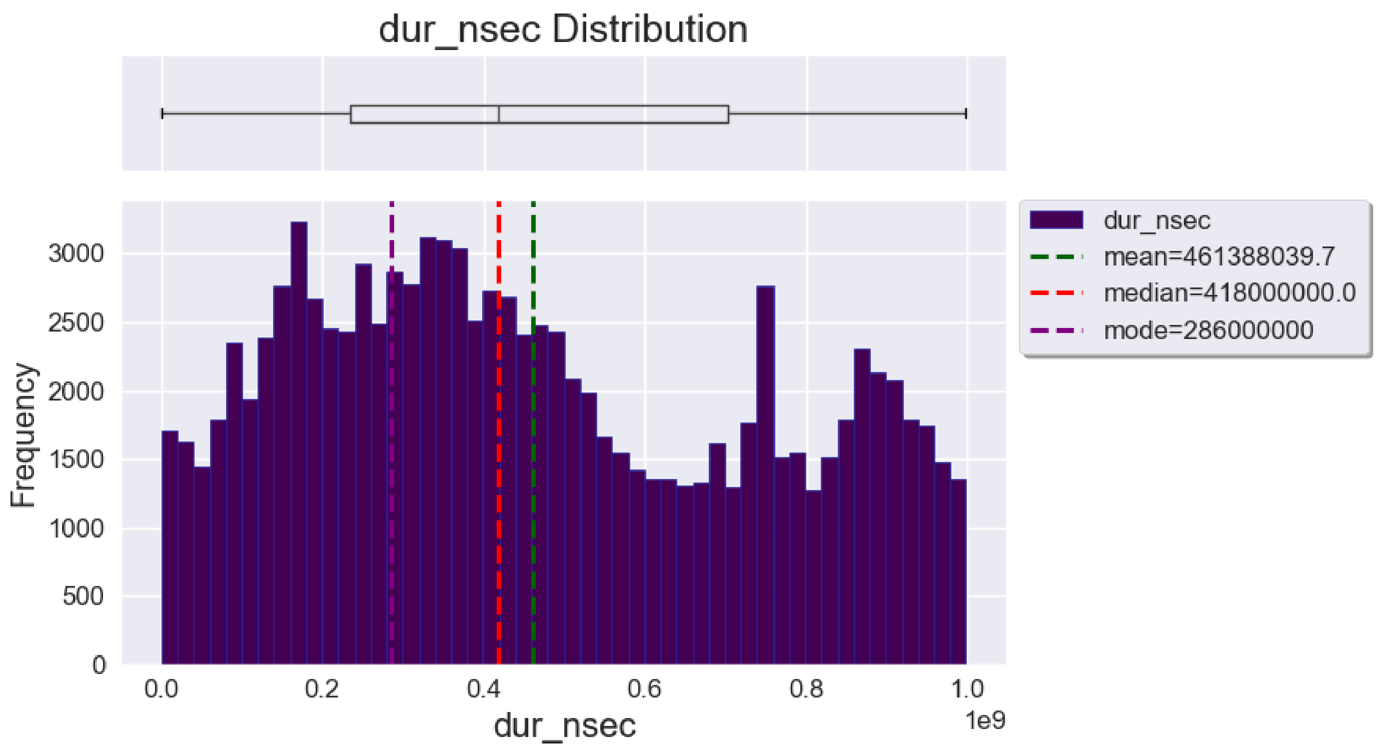
In [33]:
```
continuous_plot(df_continuous.join(data.label), 'bytecount', "bytecount Distribution", "")
```



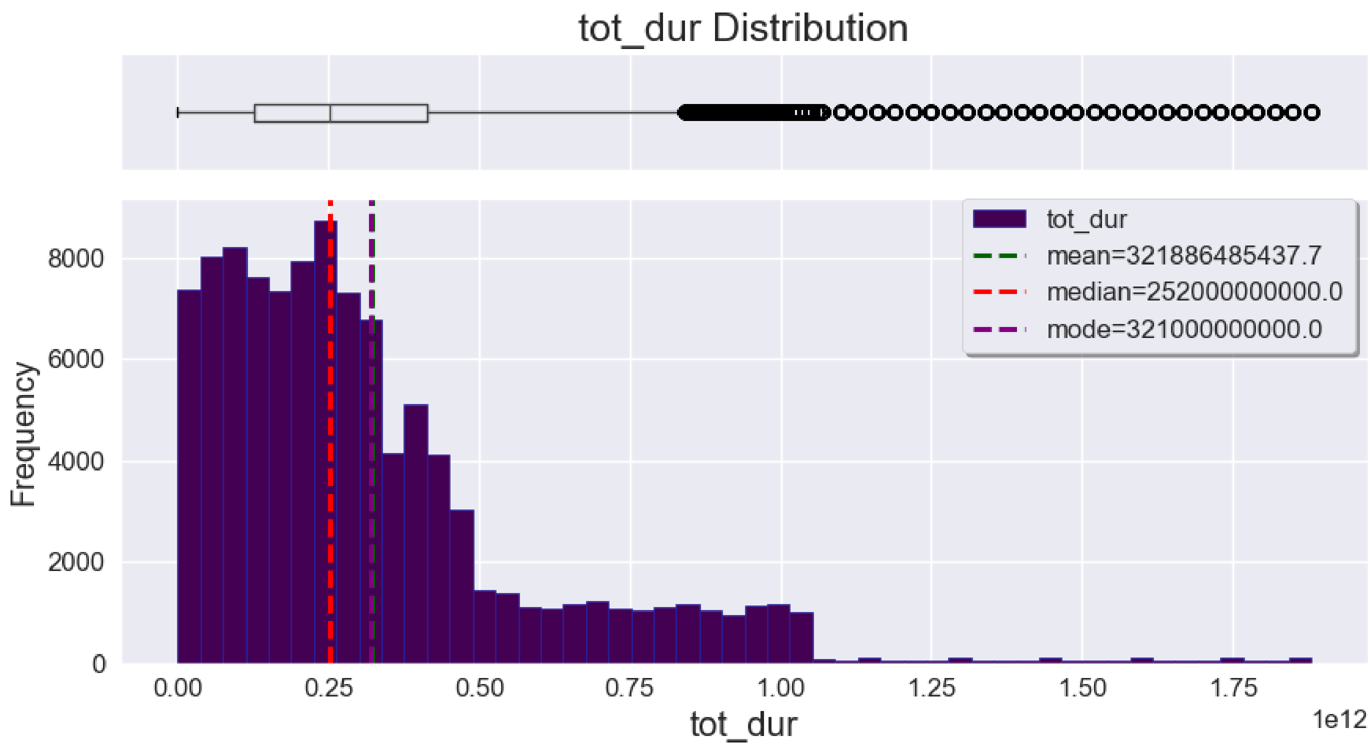
In [34]:
```
continuous_plot(df_continuous.join(data.label), 'dur', "dur Distribution", "")
```



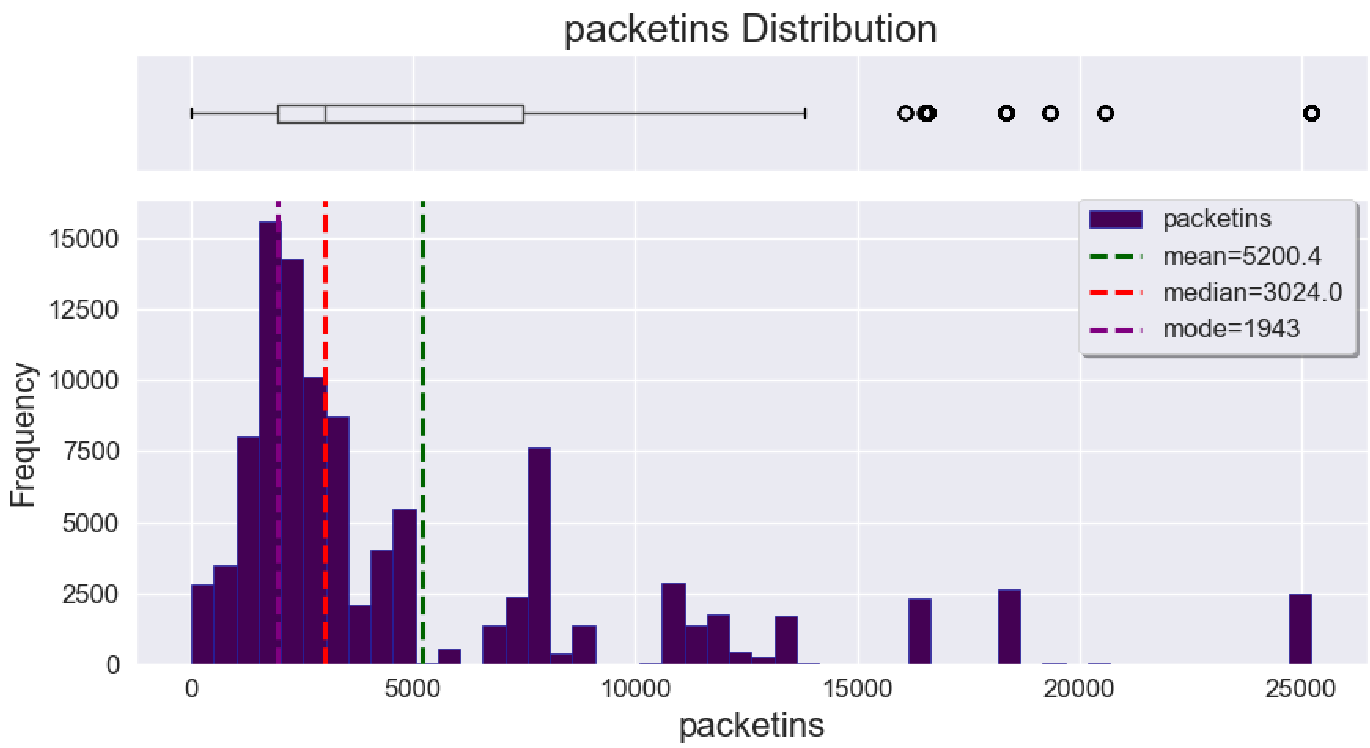
In [35]:
```
continuous_plot(df_continuous.join(data.label), 'dur_nsec', "dur_nsec Distribution", "")
```



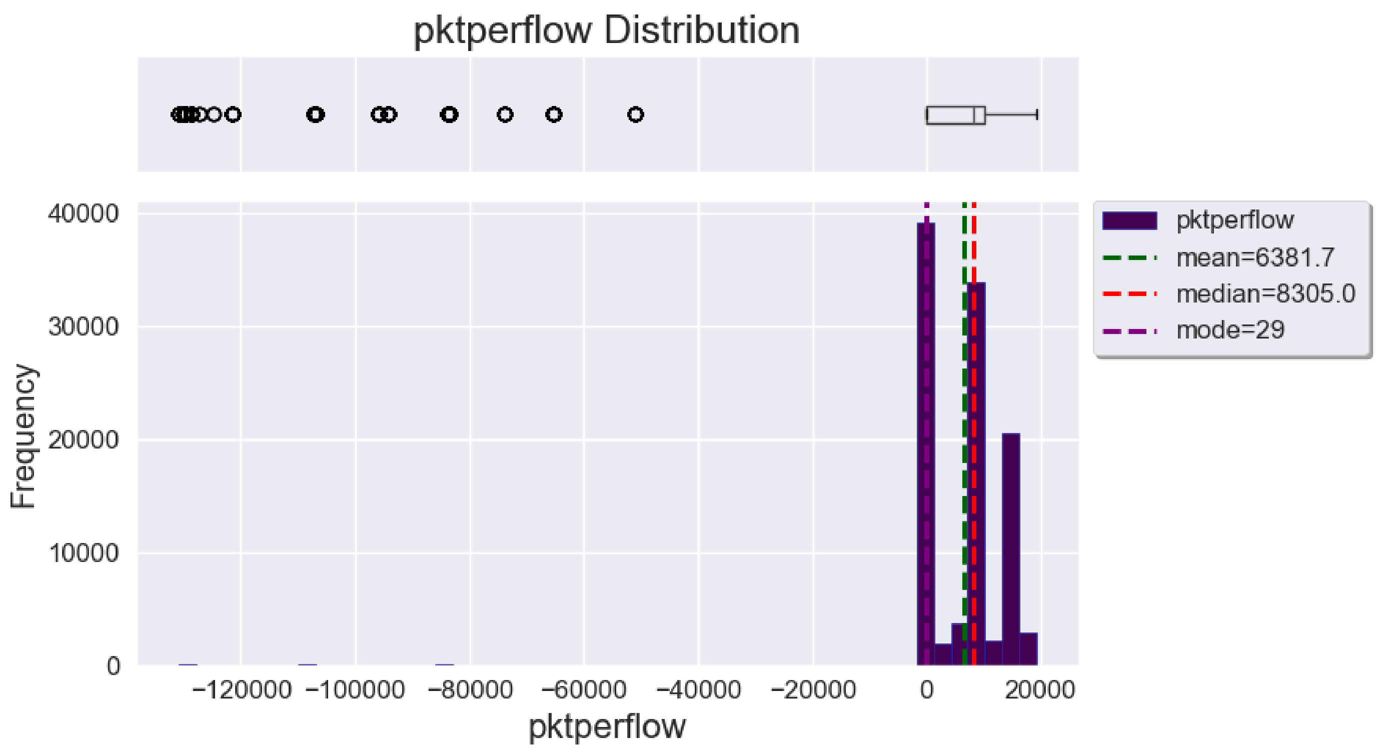
In [36]:
```
continuous_plot(df_continuous.join(data.label), 'tot_dur', "tot_dur Distribution", "")
```



In [37]:
```
continuous_plot(df_continuous.join(data.label), 'packetins', "packetins Distribution", "")
```



In [38]:
```
continuous_plot(df_continuous.join(data.label), 'pktperflow', "pktperflow Distribution", "")
```



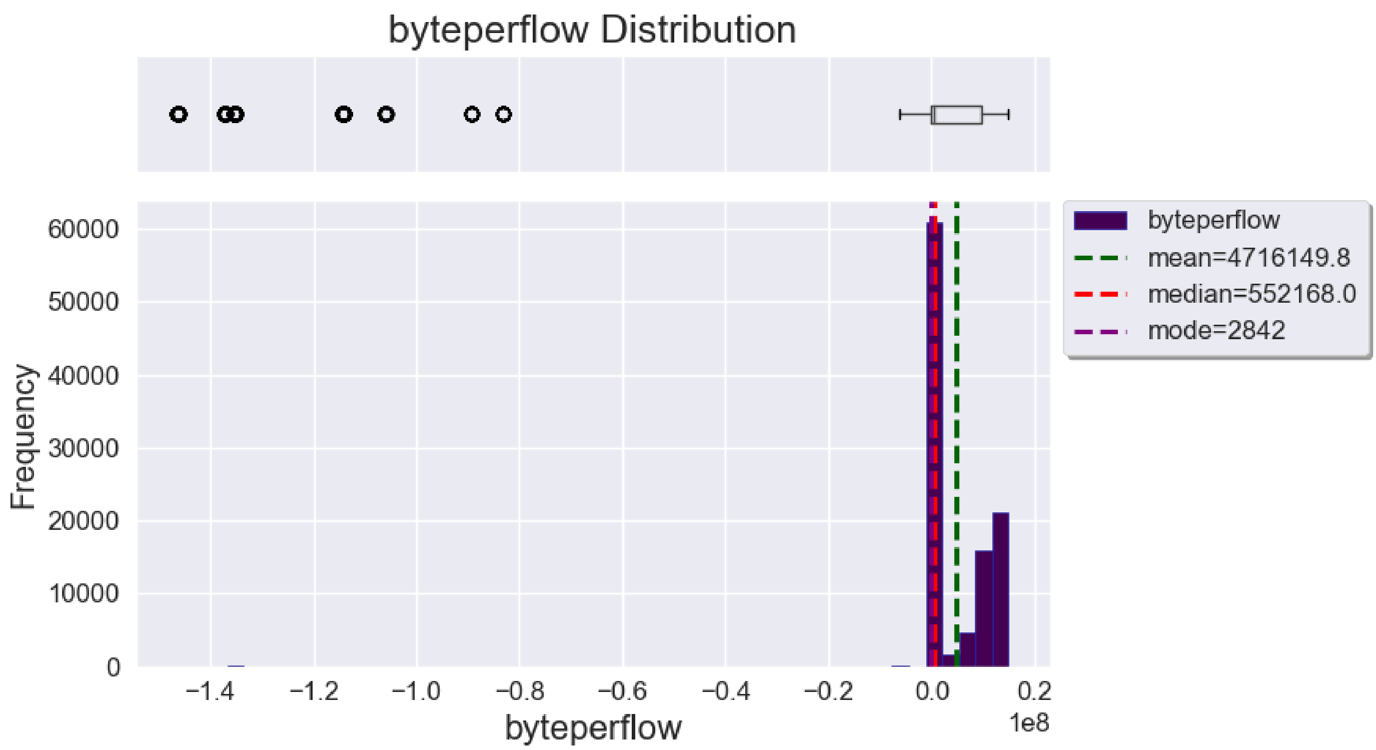
In [39]:
```
loc_potential_outliers(df_continuous, "pktperflow")
```

```
Detected total of 188 potential outliers
```

Out[39]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_by
20463	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	192570655	3
20465	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3711	12
20470	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	489182151	2
20472	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	4253	296773
20489	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3801	12
...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	5703	1
82400	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	20833777	299799
82404	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	299799473	20833
82405	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	7508889	6822
82406	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	13330713	292975

In [40]:
```
continuous_plot(df_continuous.join(data.label), 'byteperflow', "byteperflow Distribution", "")
```



In [41]:
```
loc_potential_outliers(df_continuous, "byteperflow")
```

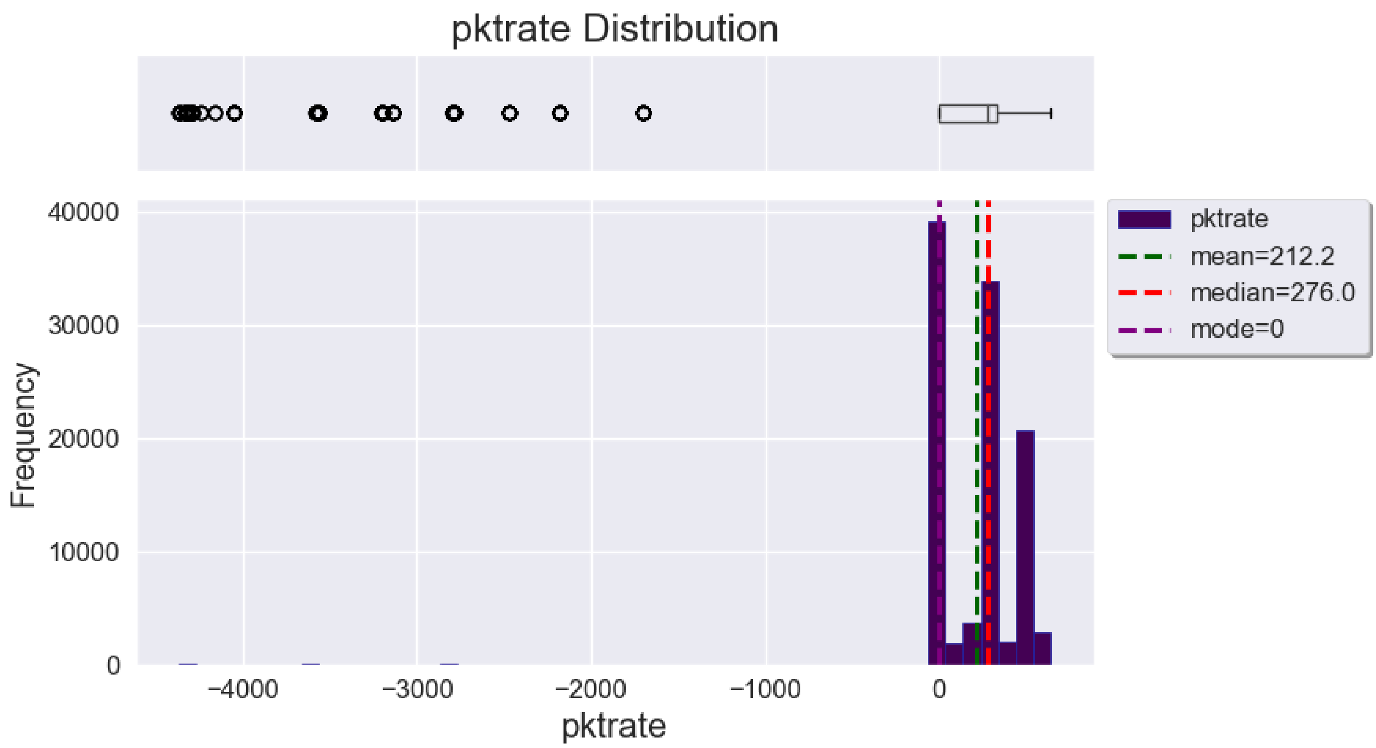
```
Detected total of 140 potential outliers
```

Out[41]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_by
20463	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	192570655	3
20465	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3711	1
20470	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	489182151	2
20472	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	4253	296773
20489	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3801	1
...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	5703	1
82400	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	20833777	299799
82404	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	299799473	20833
82405	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	7508889	6822
82406	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	13330713	292975

140 rows × 15 columns

In [42]:
```
continuous_plot(df_continuous.join(data.label), 'pktrate', "pktrate Distribution", "")
```



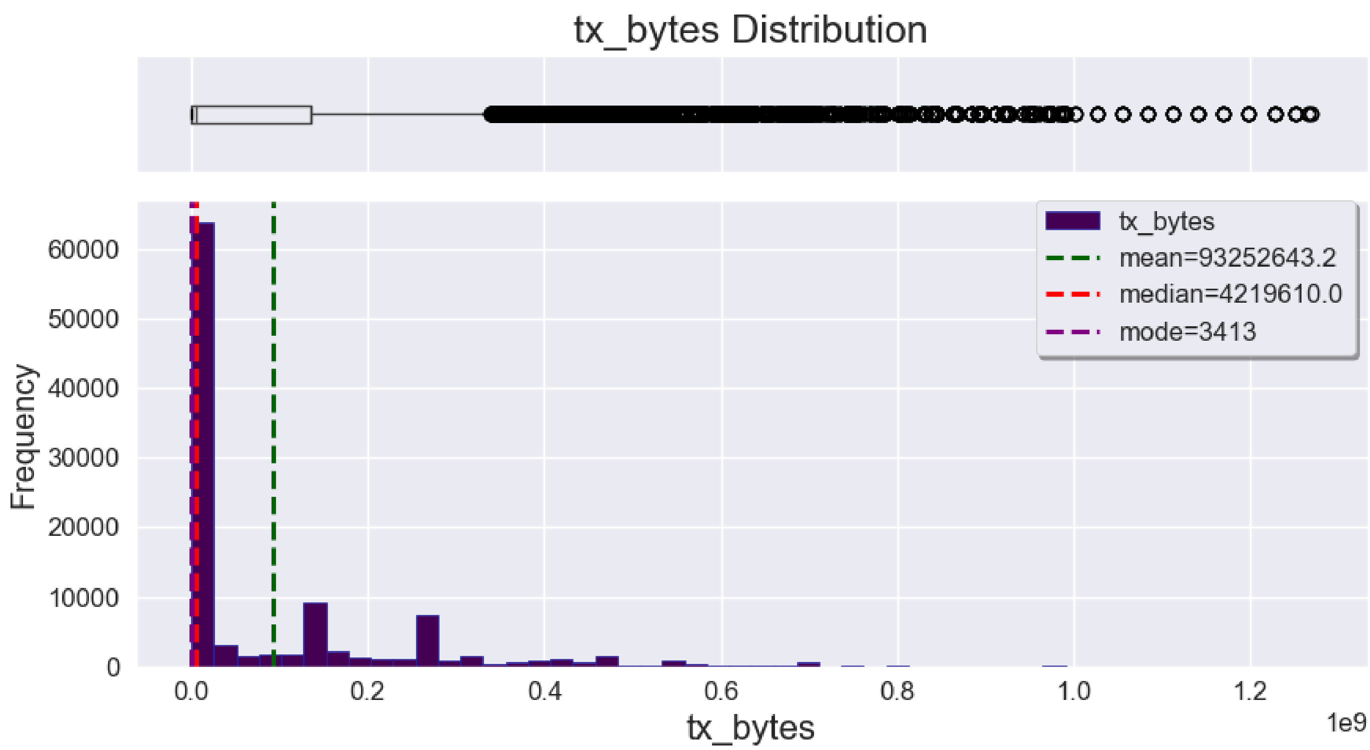
In [43]:
```
loc_potential_outliers(df_continuous, "pktrate")
```

Detected total of 188 potential outliers

Out[43]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_by
20463	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	192570655	3
20465	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3711	1
20470	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	489182151	2
20472	2740	2670	2846220	5	651000000	5.651000e+09	4073	-128767	-137265622	-4293	4253	296773
20489	2740	2671	2847286	5	651000000	5.651000e+09	4073	-83850	-89384100	-2795	3801	1
...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	5703	1
82400	15695	8746	472284	28	199000000	2.819900e+10	16540	-124723	-146107254	-4158	20833777	299799
82404	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	299799473	20833
82405	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	7508889	6822
82406	15695	6171	333234	21	89000000	2.108900e+10	16540	-127298	-146246304	-4244	13330713	292975

In [44]:
```
continuous_plot(df_continuous.join(data.label), 'tx_bytes', "tx_bytes Distribution", "")
```



In [45]:
```
loc_potential_outliers(df_continuous, "tx_bytes")
```

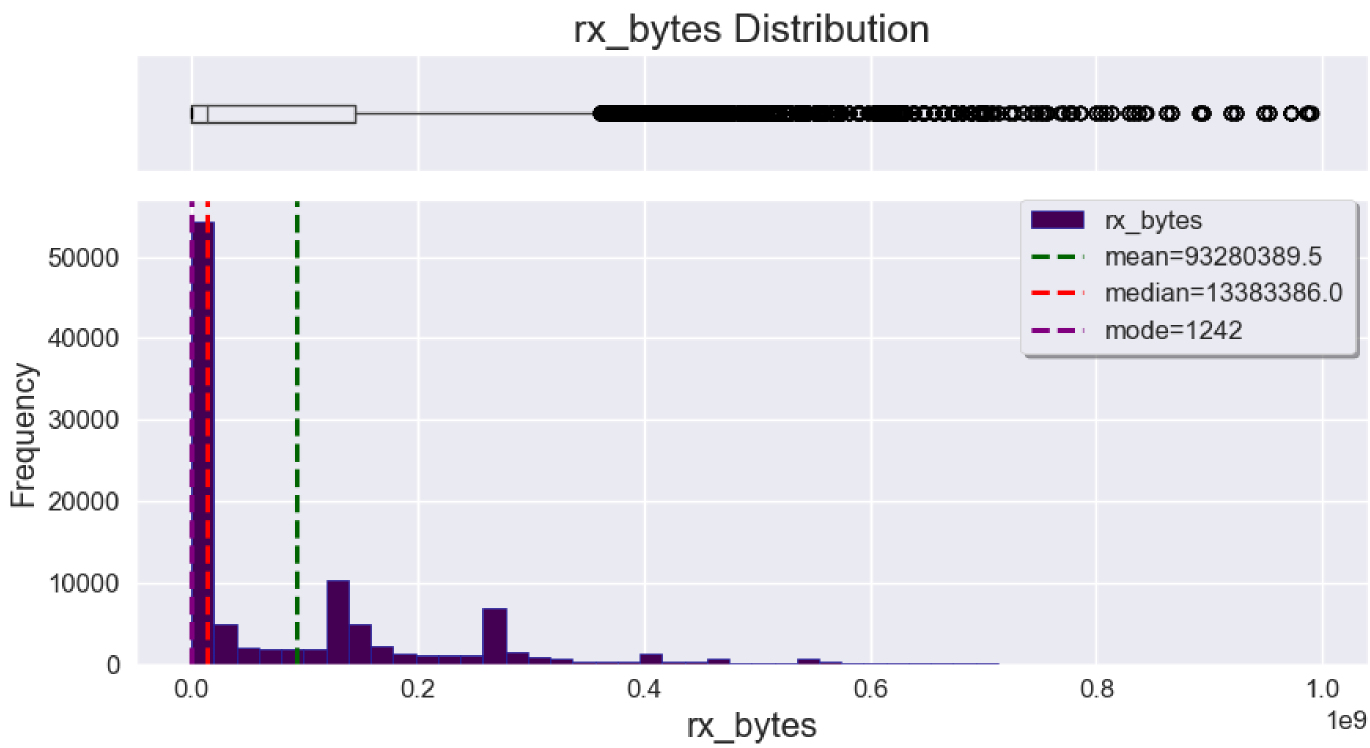
```
Detected total of 39 potential outliers
```

Out[45]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_by
1293	12085	85848	91513968	190	666000000	1.910000e+11	2242	13494	14384604	449	1141268426	48
1338	12175	126152	134478032	280	693000000	2.810000e+11	2242	13299	14176734	443	1227633931	50
1659	12145	112853	120301298	250	691000000	2.510000e+11	2242	13530	14422980	451	1198851889	49
1715	12025	58691	62564606	130	661000000	1.310000e+11	2242	13478	14367548	449	1083670062	45
1739	12055	72354	77129364	160	663000000	1.610000e+11	2242	13663	14564758	455	1112458626	47
2092	12205	134997	143906802	310	738000000	3.110000e+11	2242	8845	9428770	294	1251211803	51
2169	12115	99323	105878318	220	666000000	2.210000e+11	2242	13475	14364350	449	1170058079	49
4966	12025	36183	38571078	80	676000000	8.067600e+10	2242	13478	14367548	449	1083670062	45
5181	12025	58647	62517702	130	524000000	1.310000e+11	2242	13478	14367548	449	1083670062	45
7401	12055	49846	53135836	110	678000000	1.110000e+11	2242	13663	14564758	455	1112458626	47
7448	12175	126108	134431128	280	556000000	2.810000e+11	2242	13299	14176734	443	1227633931	50
7486	12085	85804	91467064	190	528000000	1.910000e+11	2242	13494	14384604	449	1141268426	48
7539	12175	103646	110486636	230	708000000	2.310000e+11	2242	13301	14178866	443	1227633931	50
7583	12055	72310	77082460	160	526000000	1.610000e+11	2242	13663	14564758	455	1112458626	47
7657	12115	99278	105830348	220	529000000	2.210000e+11	2242	13474	14363284	449	1170058079	49
7659	12115	76818	81887988	170	681000000	1.710000e+11	2242	13478	14367548	449	1170058079	49
7737	12085	63340	67520440	140	680000000	1.410000e+11	2242	13494	14384604	449	1141268426	48
7803	12145	112809	120254394	250	554000000	2.510000e+11	2242	13531	14424046	451	1198851889	49
7836	12145	90345	96307770	200	706000000	2.010000e+11	2242	13527	14419782	450	1198851889	49
7932	12235	130915	139555390	290	800000000	2.910000e+11	2242	13684	14587144	456	1265626297	51
7971	12205	117231	124968246	260	752000000	2.610000e+11	2242	13585	14481610	452	1251211803	51
8065	12205	134953	143859898	310	600000000	3.110000e+11	2242	8845	9428770	294	1251211803	51
8126	12265	135003	143913198	320	801000000	3.210000e+11	2242	4088	4357808	136	1269981973	51
11555	12235	130900	139539400	290	749000000	2.910000e+11	2242	13685	14588210	456	1265626297	51
11656	12265	134988	143897208	320	750000000	3.210000e+11	2242	4088	4357808	136	1269981973	51
13824	12085	63325	67504450	140	629000000	1.410000e+11	2242	13494	14384604	449	1141268426	48
13845	12085	85739	91397774	190	244000000	1.900000e+11	2242	13494	14384604	449	1141268426	48
13886	12115	76803	81871998	170	629000000	1.710000e+11	2242	13478	14367548	449	1170058079	49
13907	12115	99213	105761058	220	244000000	2.200000e+11	2242	13474	14363284	449	1170058079	49
13936	12055	72245	77013170	160	241000000	1.600000e+11	2242	13663	14564758	455	1112458626	47
13967	12025	36168	38555088	80	625000000	8.062500e+10	2242	13478	14367548	449	1083670062	45
14048	12055	49831	53119846	110	626000000	1.110000e+11	2242	13663	14564758	455	1112458626	47
14099	12175	103631	110470646	230	656000000	2.310000e+11	2242	13301	14178866	443	1227633931	50
14133	12205	117215	124951190	260	702000000	2.610000e+11	2242	13584	14480544	452	1251211803	51
14154	12205	134888	143790608	310	317000000	3.100000e+11	2242	8845	9428770	294	1251211803	51
14199	12145	90330	96291780	200	654000000	2.010000e+11	2242	13527	14419782	450	1198851889	49
14231	12145	112744	120185104	250	269000000	2.500000e+11	2242	13531	14424046	451	1198851889	49
14251	12175	126043	134361838	280	271000000	2.800000e+11	2242	13299	14176734	443	1227633931	50

In [46]:

```
continuous_plot(df_continuous.join(data.label), 'rx_bytes', "rx_bytes Distribution", "")
```



In [47]:
```
loc_potential_outliers(df_continuous, "rx_bytes")
```

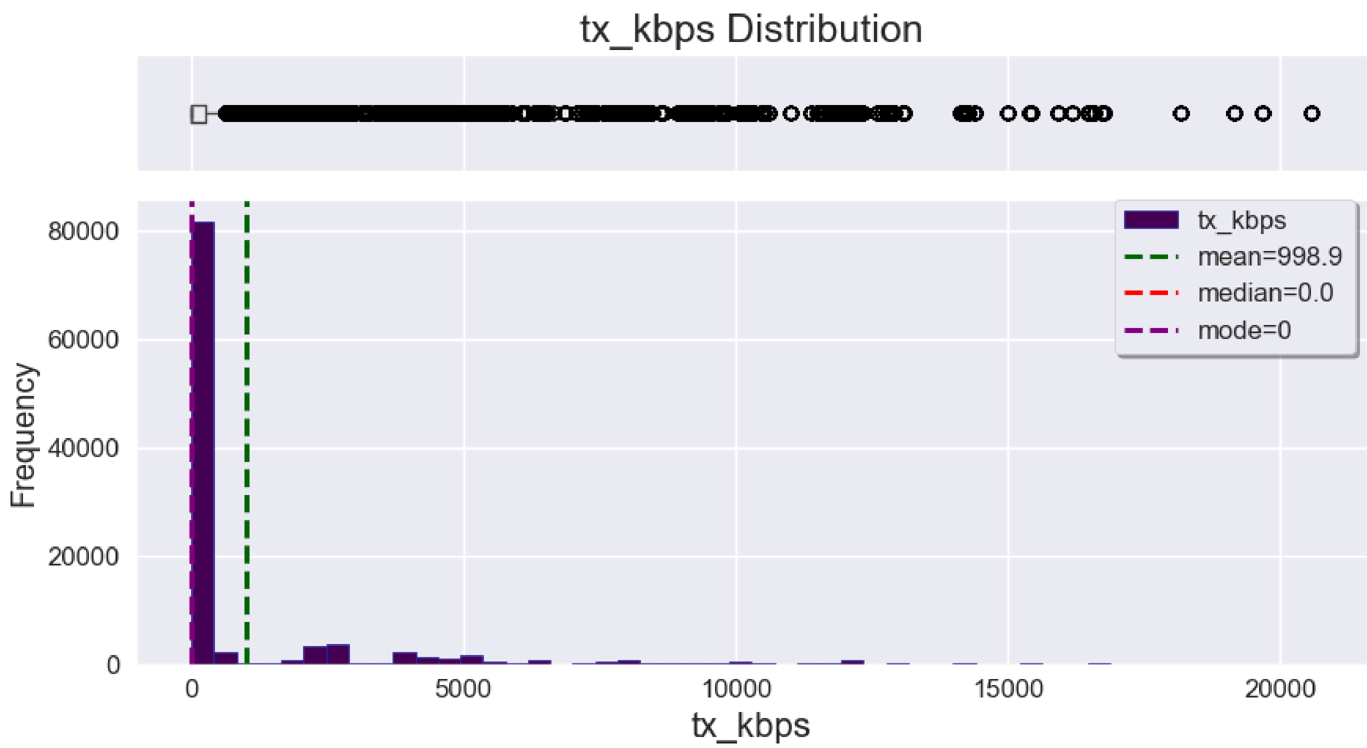
```
Detected total of 39 potential outliers
```

Out[47]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_byt
1649	12145	112853	120301298	250	691000000	2.510000e+11	2242	13530	14422980	451	6707	9194660
1796	12175	126152	134478032	280	693000000	2.810000e+11	2242	13299	14176734	443	6749	9482480
2089	12205	134997	143906802	310	738000000	3.110000e+11	2242	8845	9428770	294	6833	9718259
7407	12175	126108	134431128	280	556000000	2.810000e+11	2242	13299	14176734	443	6749	9482480
7613	12175	103646	110486636	230	708000000	2.310000e+11	2242	13301	14178866	443	6749	9482480
7819	12145	112809	120254394	250	554000000	2.510000e+11	2242	13531	14424046	451	6707	9194660
7837	12145	90345	96307770	200	706000000	2.010000e+11	2242	13527	14419782	450	6707	9194660
7933	12235	130915	139555390	290	800000000	2.910000e+11	2242	13684	14587144	456	6875	9862404
8014	12205	117231	124968246	260	752000000	2.610000e+11	2242	13585	14481610	452	6833	9718259
8054	12205	134953	143859898	310	600000000	3.110000e+11	2242	8845	9428770	294	6833	9718259
8108	12265	135003	143913198	320	801000000	3.210000e+11	2242	4088	4357808	136	6875	9905961
11556	12235	130900	139539400	290	749000000	2.910000e+11	2242	13685	14588210	456	6875	9862404
11568	12265	134988	143897208	320	750000000	3.210000e+11	2242	4088	4357808	136	6875	9905961
11581	12205	117215	124951190	260	702000000	2.610000e+11	2242	13584	14480544	452	6833	9718259
14092	12175	103631	110470646	230	656000000	2.310000e+11	2242	13301	14178866	443	6749	9482480
14164	12205	134888	143790608	310	317000000	3.100000e+11	2242	8845	9428770	294	6833	9718259
14198	12145	90330	96291780	200	654000000	2.010000e+11	2242	13527	14419782	450	6707	9194660
14203	12145	112744	120185104	250	269000000	2.500000e+11	2242	13531	14424046	451	6707	9194660
14235	12175	126043	134361838	280	271000000	2.800000e+11	2242	13299	14176734	443	6749	9482480
22408	4029	115625	123256250	255	888000000	2.560000e+11	8803	13457	14345162	448	6665	9232824
22558	4029	92968	99103888	205	281000000	2.050000e+11	8803	13457	14345162	448	6665	9232824
23464	4089	119983	127901878	265	314000000	2.650000e+11	8803	13640	14540240	454	6791	9723665
23485	4089	134759	143653094	315	921000000	3.160000e+11	8803	5741	6119906	191	6791	9723665
23598	4059	129018	137533188	285	922000000	2.860000e+11	8803	13393	14276938	446	6749	9520720
23604	4059	106343	113361638	235	315000000	2.350000e+11	8803	13375	14257750	445	6749	9520720
23723	4149	134585	143467610	325	362000000	3.250000e+11	8803	1234	1315444	41	6833	9879110
23765	4119	133351	142152166	295	344000000	2.950000e+11	8803	13368	14250288	445	6833	9867661
26528	4059	106597	113632402	236	753000000	2.370000e+11	8803	13375	14257750	445	6749	9520720
26546	4089	134971	143879086	316	743000000	3.170000e+11	8803	5741	6119906	191	6791	9723665
26576	4029	93222	99374652	206	720000000	2.070000e+11	8803	13457	14345162	448	6665	9232824
26626	4059	129230	137759180	286	745000000	2.870000e+11	8803	13393	14276938	446	6749	9520720
26698	4119	133605	142422930	296	782000000	2.970000e+11	8803	13368	14250288	445	6833	9867661
26703	4089	120237	128172642	266	751000000	2.670000e+11	8803	13640	14540240	454	6791	9723665
26727	4029	115837	123482242	256	712000000	2.570000e+11	8803	13457	14345162	448	6665	9232824
26729	4149	134839	143738374	326	800000000	3.270000e+11	8803	1234	1315444	41	6833	9879110
29111	4029	93323	99482318	207	247000000	2.070000e+11	8803	13457	14345162	448	6665	9232824
29394	4059	106698	113740068	237	281000000	2.370000e+11	8803	13375	14257750	445	6749	9520720
29431	4119	133706	142530596	297	309000000	2.970000e+11	8803	13368	14250288	445	6833	9867661

In [48]:

```
continuous_plot(df_continuous.join(data.label), 'tx_kbps', "tx_kbps Distribution", "")
```



In [49]:
```
loc_potential_outliers(df_continuous, "tx_kbps")
```

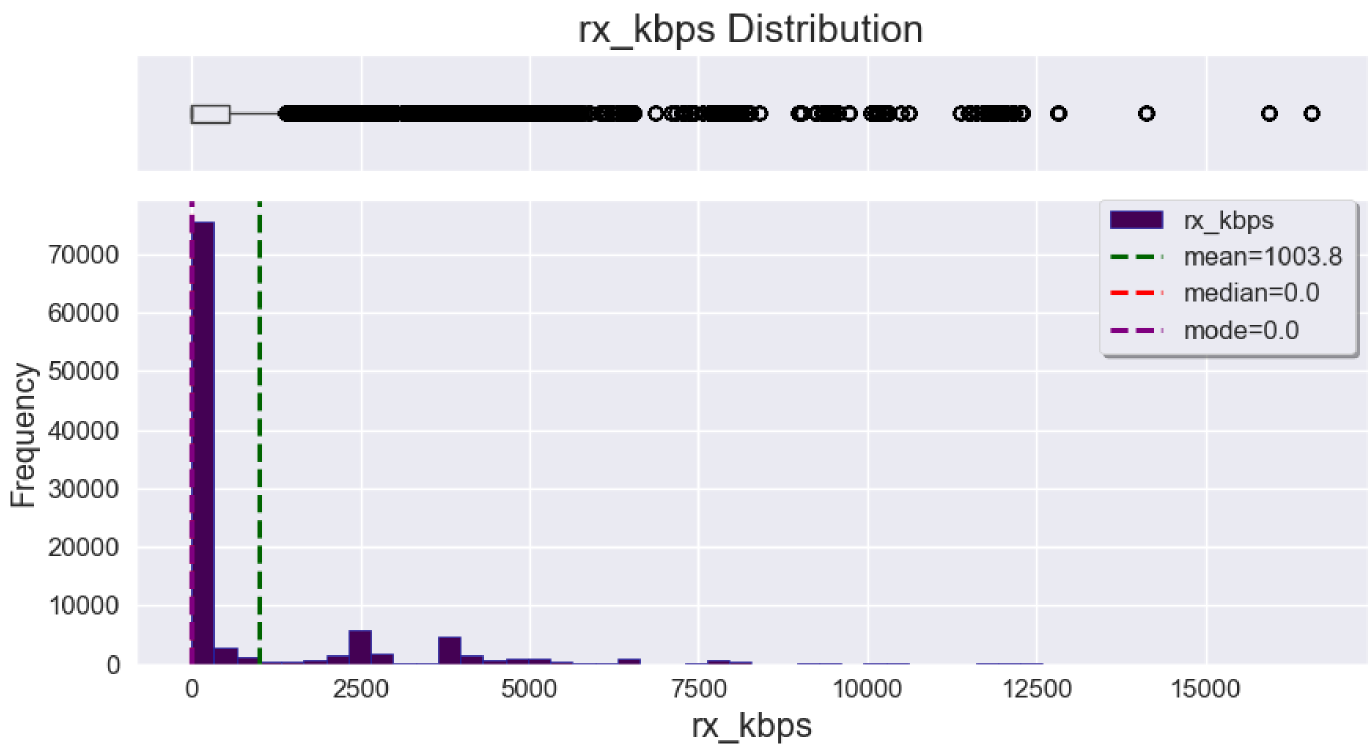
Detected total of 302 potential outliers

Out[49]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_byte
9	11425	90333	96294978	200	744000000	2.010000e+11	1943	13534	14427244	451	354583059	429
14	11425	45304	48294064	100	716000000	1.010000e+11	1943	13535	14428310	451	580813093	258
48	9906	32914	35086324	73	246000000	7.324600e+10	1931	13385	14268410	446	273821796	184
84	11425	45304	48294064	100	716000000	1.010000e+11	1943	13535	14428310	451	354583059	429
108	11425	90333	96294978	200	744000000	2.010000e+11	1943	13534	14427244	451	580813093	258
...	...	...	...	...	...	...	...	...	...	...	...	
32917	3279	24184	25199728	77	455000000	7.745500e+10	7916	9211	9597862	307	290077280	190
33074	3309	33564	34973688	107	458000000	1.070000e+11	7916	9380	9773960	312	352795458	206
33107	3309	115799	123441734	257	341000000	2.570000e+11	7916	13526	14418716	450	352795458	206
33416	3339	42940	44743480	137	462000000	1.370000e+11	7916	9376	9769792	312	415531170	215
33453	3339	129331	137866846	287	345000000	2.870000e+11	7916	13532	14425112	451	415531170	215

302 rows × 15 columns

In [50]:
```
continuous_plot(df_continuous.join(data.label), 'rx_kbps', "rx_kbps Distribution", "")
```



In [51]:
```
loc_potential_outliers(df_continuous, "rx_kbps")
```

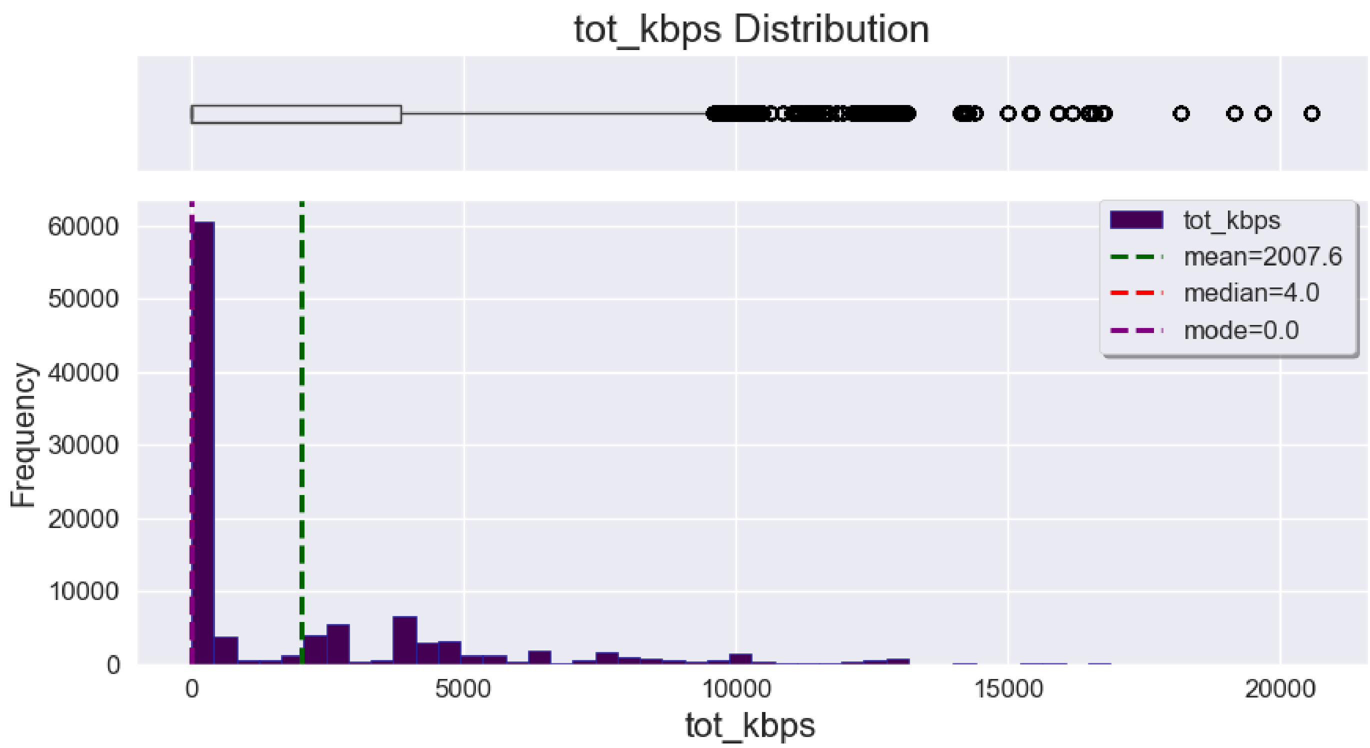
```
Detected total of 99 potential outliers
```

Out[51]:

	dt	pktcount	bytecount	dur	dur_nsec	tot_dur	packetins	pktperflow	byteperflow	pktrate	tx_bytes	rx_byte
39	11425	90333	96294978	200	744000000	2.010000e+11	1943	13534	14427244	451	4295	35458305
79	11425	45304	48294064	100	716000000	1.010000e+11	1943	13535	14428310	451	4295	35458305
257	11455	103866	110721156	230	747000000	2.310000e+11	1943	13533	14426178	451	4463	40265184
331	11485	117399	125147334	260	750000000	2.610000e+11	1943	13533	14426178	451	4589	4507183
336	11485	72370	77146420	160	722000000	1.610000e+11	1943	13533	14426178	451	4589	4507183
...	...	...	...	...	...	...	...	...	...	...	...	
17350	11485	97496	101590832	310	801000000	3.110000e+11	1943	9306	9696852	310	4589	4507183
17519	11395	134886	143788476	320	273000000	3.200000e+11	1943	4124	4396184	137	4127	2924183
17563	11515	106652	111131384	340	804000000	3.410000e+11	1943	9156	9540552	305	4715	4988380
18077	11395	134981	143889746	320	656000000	3.210000e+11	1943	4124	4396184	137	4127	2924183
18336	11365	130857	139493562	290	654000000	2.910000e+11	1790	13529	14421914	450	4043	2326761

99 rows × 15 columns

In [52]:
```
continuous_plot(df_continuous.join(data.label), 'tot_kbps', "tot_kbps Distribution", "")
```



Negative values

In [53]:
```
#pktperflow
(data
 .pktperflow
 .describe()
)
```

Out[53]:
```
count     104345.000000
mean        6381.715291
std         7404.777808
min      -130933.000000
25%           29.000000
50%         8305.000000
75%        10017.000000
max        19190.000000
Name: pktperflow, dtype: float64
```

In [54]:
```
(data
 .loc[data.pktperflow < 0]
)
```

Out[54]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo
20463	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20465	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20470	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20472	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20489	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
...	...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82400	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82404	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82405	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82406	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463

188 rows × 23 columns

In [55]:
```
#byteperflow
(data
 .byteperflow
 .describe()
)
```

Out[55]:
```
count    1.043450e+05
mean     4.716150e+06
std      7.560116e+06
min     -1.464426e+08
25%      2.842000e+03
50%      5.521680e+05
75%      9.728112e+06
max      1.495387e+07
Name: byteperflow, dtype: float64
```

In [56]:
```
(data
 .loc[data.byteperflow < 0]
)
```

Out[56]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo
20463	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20465	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20470	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20472	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20489	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
...	...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82400	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82404	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82405	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82406	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463

188 rows × 23 columns

In [57]:
```
#pktrate
(data
 .pktrate
 .describe()
)
```

Out[57]:
```
count    104345.000000
mean        212.210676
std         246.855123
min       -4365.000000
25%           0.000000
50%         276.000000
75%         333.000000
max         639.000000
Name: pktrate, dtype: float64
```

In [58]:
```
(data
 .loc[data.pktrate < 0]
)
```

Out[58]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperflo
20463	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20465	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
20470	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20472	2740	1	10.0.0.1	10.0.0.3	2670	2846220	5	651000000	5.651000e+09	4	4073	-128767	-1372656
20489	2740	1	10.0.0.2	10.0.0.3	2671	2847286	5	651000000	5.651000e+09	4	4073	-83850	-893841
...	...	...	...	...	...	...	...	...	...	...	...	...	
82399	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82400	15695	2	10.0.0.2	10.0.0.8	8746	472284	28	199000000	2.819900e+10	3	16540	-124723	-1461072
82404	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82405	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463
82406	15695	1	10.0.0.2	10.0.0.8	6171	333234	21	89000000	2.108900e+10	3	16540	-127298	-1462463

188 rows × 23 columns

Insights:

1. `pktperflow`, `byteeperflow`, `pktrate` all contain 188 instances with negative values.

3.2.2. Discrete and ordinal features

In [59]:
```
def cat_value_count(df: pd.DataFrame,
                    col: str,
                    reindex: list = None) -> pd.DataFrame:

    return (pd
            .DataFrame((df[col].value_counts(normalize=i) for i in [False, True]), index=['abs_count', 'norm_co
            .T
            .reindex(reindex)
            .assign(cumsum=lambda df_: df_.norm_count.cumsum(),
                    mean_target=df.groupby(col)["label"].mean())
#             .sort_index(ascending=True)
            .pipe(lambda df_: print(f'This categorical predictor has {len(df_)} unique values\n\n', df_))
           )
```

In [60]:
```
categorical_data_with_target = (data
                                .loc[:, cols_continuous.index[cols_continuous == False].to_list()]
                                .join(data.select_dtypes(exclude="number"))
                                .join(data.label)
                               )
categorical_data_with_target
```

Out[60]:

	switch	flows	Pairflow	port_no	src	dst	Protocol	label
0	1	3	0	3	10.0.0.1	10.0.0.8	UDP	0
1	1	2	0	4	10.0.0.1	10.0.0.8	UDP	0
2	1	3	0	1	10.0.0.2	10.0.0.8	UDP	0
3	1	3	0	2	10.0.0.2	10.0.0.8	UDP	0
4	1	3	0	3	10.0.0.2	10.0.0.8	UDP	0
...	...	...	...	...	...	...	...	...
104340	3	5	0	1	10.0.0.5	10.0.0.7	ICMP	0
104341	3	5	0	3	10.0.0.5	10.0.0.7	ICMP	0
104342	3	5	0	2	10.0.0.11	10.0.0.5	ICMP	0
104343	3	5	0	1	10.0.0.11	10.0.0.5	ICMP	0
104344	3	5	0	3	10.0.0.11	10.0.0.5	ICMP	0

104345 rows × 8 columns

In [61]:
```
df_discrete = (data
               [cols_continuous[~cols_continuous].index]
              )
df_discrete.shape
```

Out[61]: (104345, 4)

In [62]:
```
cat_value_count(categorical_data_with_target, "switch")
```

This categorical predictor has 10 unique values

	abs_count	norm_count	cumsum	mean_target
4	22077.0	0.211577	0.211577	0.403814
3	20965.0	0.200920	0.412497	0.393275
5	15442.0	0.147990	0.560487	0.390947
2	14135.0	0.135464	0.695951	0.401486
6	10058.0	0.096392	0.792343	0.400875
7	8368.0	0.080196	0.872538	0.380497
1	6464.0	0.061948	0.934487	0.275371
8	4504.0	0.043165	0.977651	0.433837
9	1686.0	0.016158	0.993809	0.460261
10	646.0	0.006191	1.000000	0.287926

In [63]:
```
cat_value_count(categorical_data_with_target, "flows")
```

This categorical predictor has 15 unique values

	abs_count	norm_count	cumsum	mean_target
5	24046.0	0.230447	0.230447	0.390585
3	20628.0	0.197690	0.428137	0.419721
7	11093.0	0.106311	0.534448	0.299288
4	9907.0	0.094945	0.629393	0.459574
2	9830.0	0.094207	0.723600	0.639268
11	8514.0	0.081595	0.805194	0.234202
9	8030.0	0.076956	0.882151	0.209963
6	6722.0	0.064421	0.946571	0.459387
13	1956.0	0.018746	0.965317	0.307771
8	1533.0	0.014692	0.980009	0.390085
10	810.0	0.007763	0.987771	0.295062
17	384.0	0.003680	0.991451	0.375000
15	336.0	0.003220	0.994672	0.357143
14	325.0	0.003115	0.997786	0.200000
12	231.0	0.002214	1.000000	0.177489

In [64]:
```
cat_value_count(categorical_data_with_target, "Pairflow")
```

This categorical predictor has 2 unique values

	abs_count	norm_count	cumsum	mean_target
1	62710.0	0.600987	0.600987	0.371312
0	41635.0	0.399013	1.000000	0.420295

In [65]:
```
cat_value_count(categorical_data_with_target, "port_no")
```

This categorical predictor has 5 unique values

	abs_count	norm_count	cumsum	mean_target
1	29645.0	0.284106	0.284106	0.392815
2	29148.0	0.279343	0.563448	0.389804
3	28413.0	0.272299	0.835747	0.389294
4	15637.0	0.149859	0.985605	0.404297
5	1502.0	0.014395	1.000000	0.262317

In [66]:
```
cat_value_count(categorical_data_with_target, "src")
```

This categorical predictor has 19 unique values

	abs_count	norm_count	cumsum	mean_target
10.0.0.3	11491.0	0.110125	0.110125	0.432338
10.0.0.7	10313.0	0.098836	0.208961	0.348395
10.0.0.10	9671.0	0.092683	0.301644	0.735291
10.0.0.1	8645.0	0.082850	0.384494	0.309543
10.0.0.12	8147.0	0.078078	0.462571	0.195409
10.0.0.2	8063.0	0.077273	0.539844	0.179958
10.0.0.5	7291.0	0.069874	0.609718	0.243177
10.0.0.9	7209.0	0.069088	0.678806	0.207796
10.0.0.11	6455.0	0.061862	0.740668	0.233153
10.0.0.4	5999.0	0.057492	0.798160	0.516419
10.0.0.8	5241.0	0.050228	0.848388	0.289067
10.0.0.6	2740.0	0.026259	0.874647	0.389416
10.0.0.18	2590.0	0.024822	0.899468	0.694981
10.0.0.13	2484.0	0.023806	0.923274	0.840982
10.0.0.14	2265.0	0.021707	0.944981	0.781015
10.0.0.15	1858.0	0.017806	0.962787	0.616792
10.0.0.16	1789.0	0.017145	0.979932	0.362214
10.0.0.20	1114.0	0.010676	0.990608	0.933573
10.0.0.17	980.0	0.009392	1.000000	0.454082

In [67]:
```
src_value = (data
             .src
             .str
             .split('.')
             .apply(lambda x: x[3])
             .value_counts(normalize=True)
            )
src_value.index = pd.to_numeric(src_value.index)
```

In [68]:
```
cat_value_count(categorical_data_with_target, "dst")
```

```
This categorical predictor has 18 unique values

            abs_count  norm_count    cumsum  mean_target
10.0.0.7      18020.0    0.172696  0.172696     0.434184
10.0.0.8      15587.0    0.149379  0.322076     0.422018
10.0.0.5      15184.0    0.145517  0.467593     0.372366
10.0.0.3      13051.0    0.125075  0.592669     0.457973
10.0.0.9       6318.0    0.060549  0.653218     0.237733
10.0.0.12      5635.0    0.054004  0.707221     0.297604
10.0.0.2       4990.0    0.047822  0.755043     0.192585
10.0.0.1       4645.0    0.044516  0.799559     0.485899
10.0.0.4       3963.0    0.037980  0.837539     0.530406
10.0.0.10      3926.0    0.037625  0.875164     0.367295
10.0.0.11      3370.0    0.032297  0.907461     0.056083
10.0.0.14      2007.0    0.019234  0.926695     0.724963
10.0.0.15      1765.0    0.016915  0.943610     0.616997
10.0.0.16      1684.0    0.016139  0.959749     0.322447
10.0.0.6       1590.0    0.015238  0.974987     0.378616
10.0.0.13      1076.0    0.010312  0.985299     0.672862
10.0.0.18       790.0    0.007571  0.992870     0.000000
10.0.0.17       744.0    0.007130  1.000000     0.279570
```

In [69]:
```
dst_value = (data
             .dst
             .str
             .split('.')
             .apply(lambda x: x[3])
             .value_counts(normalize=True)
            )
dst_value.index = pd.to_numeric(dst_value.index)
```

In [70]:
```
cat_value_count(categorical_data_with_target, "Protocol")
```

```
This categorical predictor has 3 unique values

       abs_count  norm_count    cumsum  mean_target
ICMP     41321.0    0.396004  0.396004     0.227947
UDP      33588.0    0.321894  0.717897     0.520990
TCP      29436.0    0.282103  1.000000     0.471056
```

In [71]:
```
df_discrete.columns
```

Out[71]:
```
Index(['switch', 'flows', 'Pairflow', 'port_no'], dtype='object')
```

In [72]:
```
pd.options.plotting.backend='plotly'
plot_rows=4
plot_cols=2
fig = make_subplots(rows=plot_rows, cols=plot_cols, subplot_titles=("switch", "flows", "Pairflow", "port_no", '

# add traces
x = 0
for i in range(1, plot_rows + 1):
    for j in range(1, plot_cols + 1):
        if x==4:
            break
        unique, counts = np.unique(df_discrete[df_discrete.columns[x]].values, return_counts=True)
        bar_plot = pd.DataFrame(np.asarray((unique, counts)).T,
                                columns =['index', 'count'])

        fig.add_trace(go.Bar(x = bar_plot["count"],
                             y = bar_plot["index"],
                             orientation='h'),
                      row=i,
                      col=j)
        x=x+1

fig.add_trace(go.Bar(x = src_value.sort_index(ascending=True).values,
                     y = src_value.sort_index(ascending=True).index,
                     orientation='h'),
              row=3,
              col=1)
```

```
fig.add_trace(go.Bar(x = dst_value.sort_index(ascending=True).values,
                     y = dst_value.sort_index(ascending=True).index,
                     orientation='h'),
              row=3,
              col=2)

fig.add_trace(go.Bar(x = data.Protocol.value_counts().values,
                     y = data.Protocol.value_counts().index,
                     orientation='h'),
              row=4,
              col=1)

# Format and show fig
fig.update_layout(height=1200,
                  width=1000,
                  title_text='Categorical Variables',
                  showlegend=False
                 )
fig.show()
```

In [73]:

```
label = LabelEncoder()
df_encoded = pd.DataFrame()

for i in categorical_data_with_target.columns[:-1]:
    df_encoded[i] = label.fit_transform(categorical_data_with_target[i])
```

```python
def Cramers_V(var1, var2):
    crosstab = np.array(pd.crosstab(index=var1, columns=var2)) # Cross Tab
    return (association(crosstab, method='cramer'))            # Return Cramer's V

# Create the dataFrame matrix with the returned Cramer's V
rows = []

for var1 in df_encoded:
    col = []

    for var2 in df_encoded:
        V = Cramers_V(df_encoded[var1], df_encoded[var2]) # Return Cramer's V
        col.append(V)                                      # Store values to subsequent columns

    rows.append(col)                                       # Store values to subsequent rows

CramersV_results = np.array(rows)
CramersV_df = (pd
               .DataFrame(CramersV_results, columns = df_encoded.columns, index = df_encoded.columns)
               .style
               .background_gradient(cmap="viridis", axis=None))
CramersV_df
```

Out[73]:

	switch	flows	Pairflow	port_no	src	dst	Protocol
switch	1.000000	0.162042	0.151066	0.111376	0.217362	0.211616	0.166522
flows	0.162042	1.000000	0.528886	0.074812	0.127964	0.143813	0.495846
Pairflow	0.151066	0.528886	1.000000	0.039299	0.425112	0.547936	0.860833
port_no	0.111376	0.074812	0.039299	1.000000	0.060824	0.067697	0.064152
src	0.217362	0.127964	0.425112	0.060824	1.000000	0.281506	0.410049
dst	0.211616	0.143813	0.547936	0.067697	0.281506	1.000000	0.514345
Protocol	0.166522	0.495846	0.860833	0.064152	0.410049	0.514345	1.000000

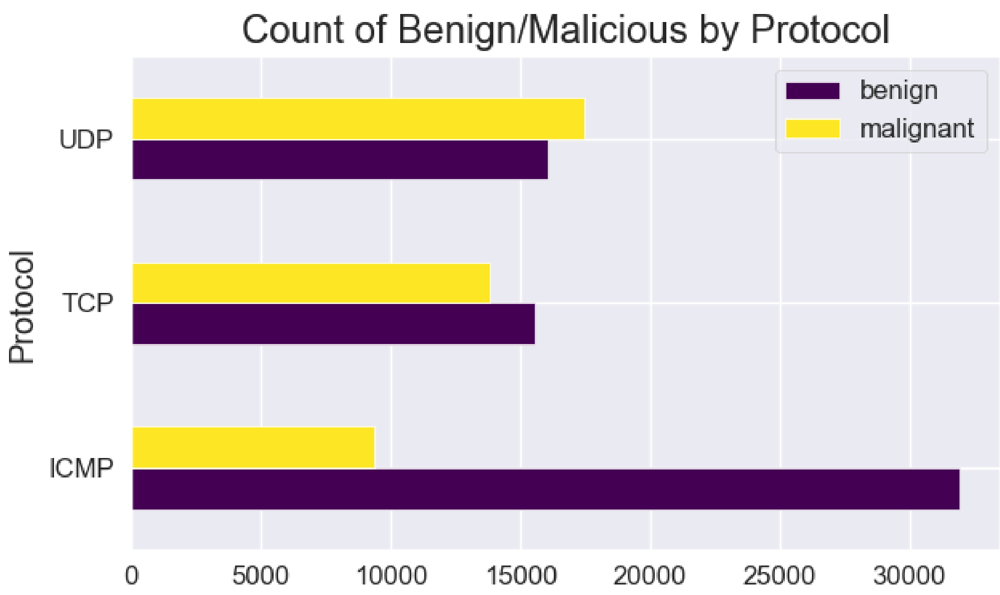
Insights:

1. Strong positive correlation between Protocol and Pairflow
2. Moderate positive correlation between Pairflow and flows, Pairflow and dst, flows and Protocol, dst and Pairflow, dst and Protocol

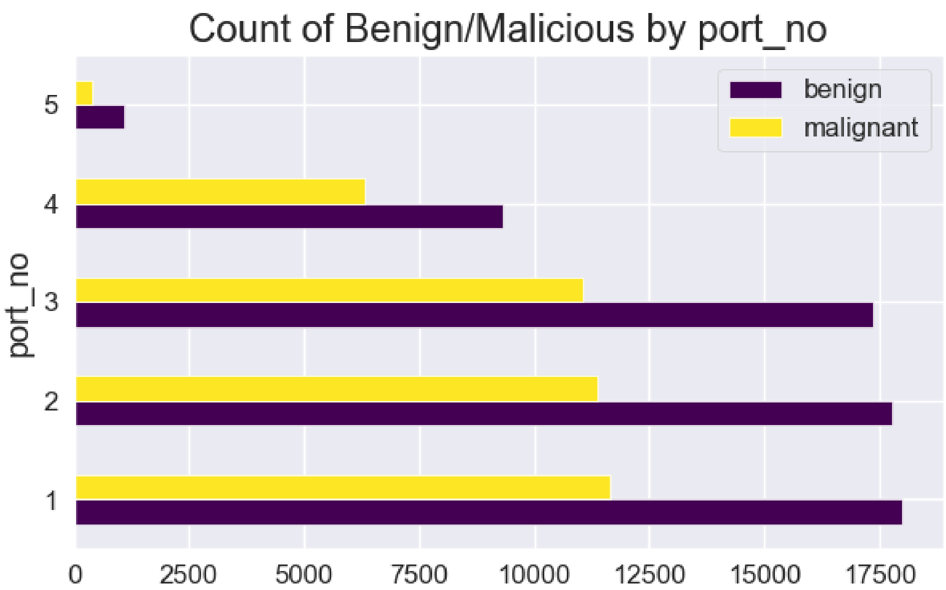
3.3. Relationship between features and target

In [74]:

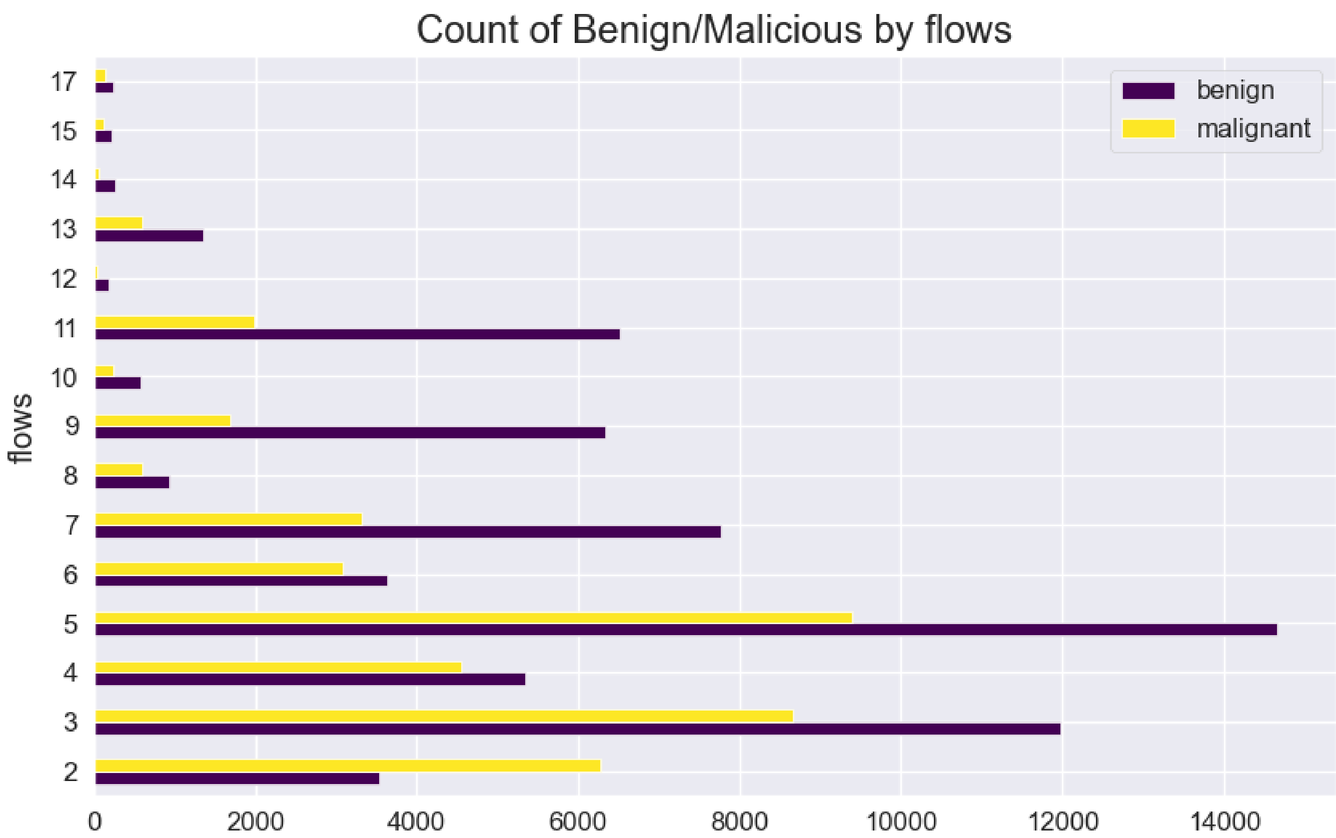
```python
pd.options.plotting.backend = 'matplotlib'

with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(figsize=(7,4))
    (data
     .groupby(['label', 'Protocol'])
     ['Protocol']
     .describe()
     ["count"]
     .unstack(level=0)
     .plot
     .barh(title="Count of Benign/Malicious by Protocol", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```



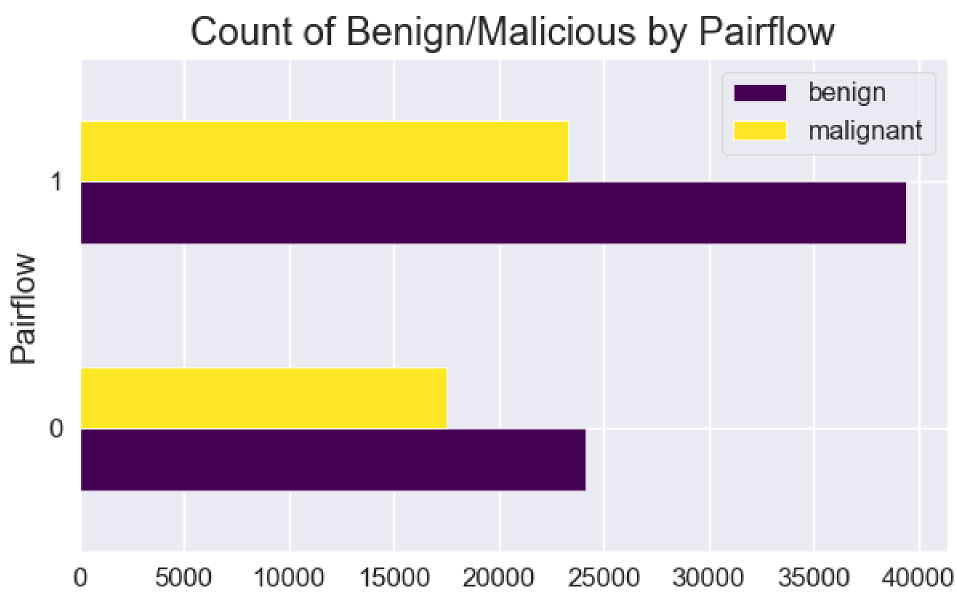
In [75]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(figsize=(7,4))
    (data
     .groupby(['label', 'port_no'])
     ['port_no']
     .describe()
     ["count"]
     .unstack(level=0)
     .plot
     .barh(rot="horizontal", title="Count of Benign/Malicious by port_no", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```



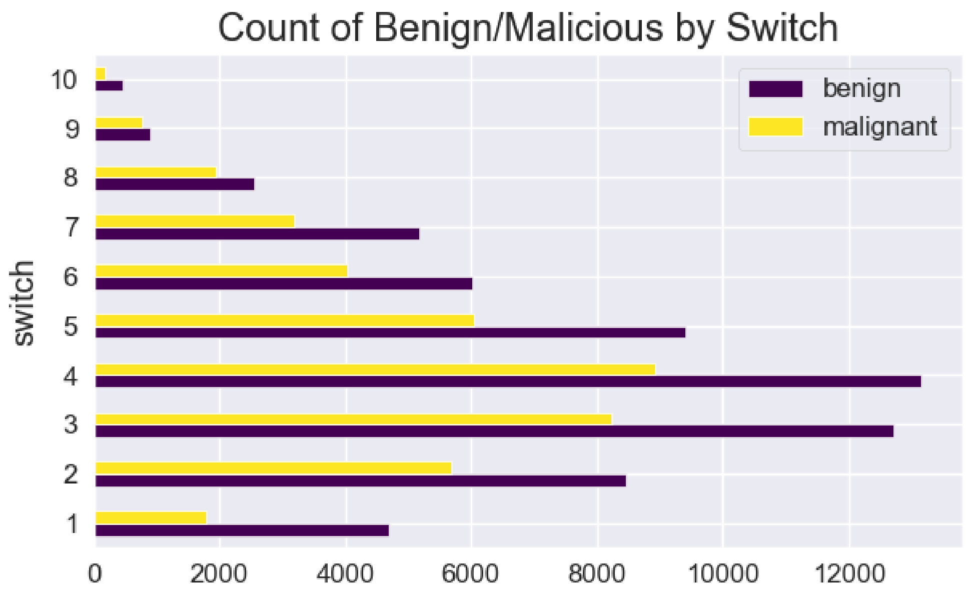
In [76]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(figsize=(10,6))
    (data
     .groupby(['label', 'flows'])
     ['flows']
     .describe()
     ["count"]
     .unstack(level=0)
     .plot
     .barh(rot="horizontal", title="Count of Benign/Malicious by flows", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```



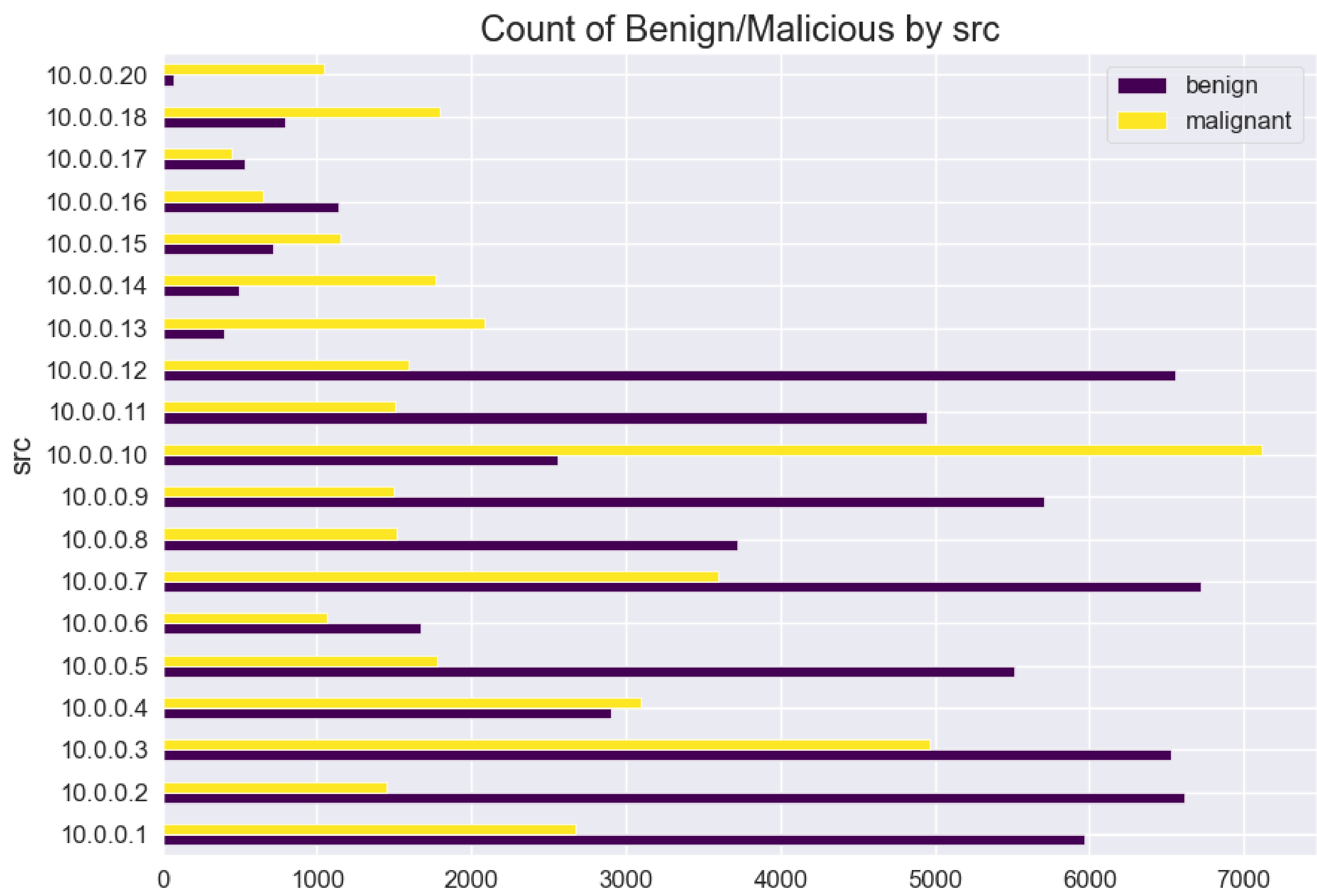
In [77]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(figsize=(7,4))
    (data
     .groupby(['label', 'Pairflow'])
     ['Pairflow']
     .describe()
     ["count"]
     .unstack(level=0)
     .plot
     .barh(rot="horizontal", title="Count of Benign/Malicious by Pairflow", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```



In [78]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(figsize=(7,4))
    (data
     .groupby(['label', 'switch'])
     ['switch']
     .describe()
     ["count"]
     .unstack(level=0)
     .plot
     .barh(rot="horizontal", title="Count of Benign/Malicious by Switch", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```

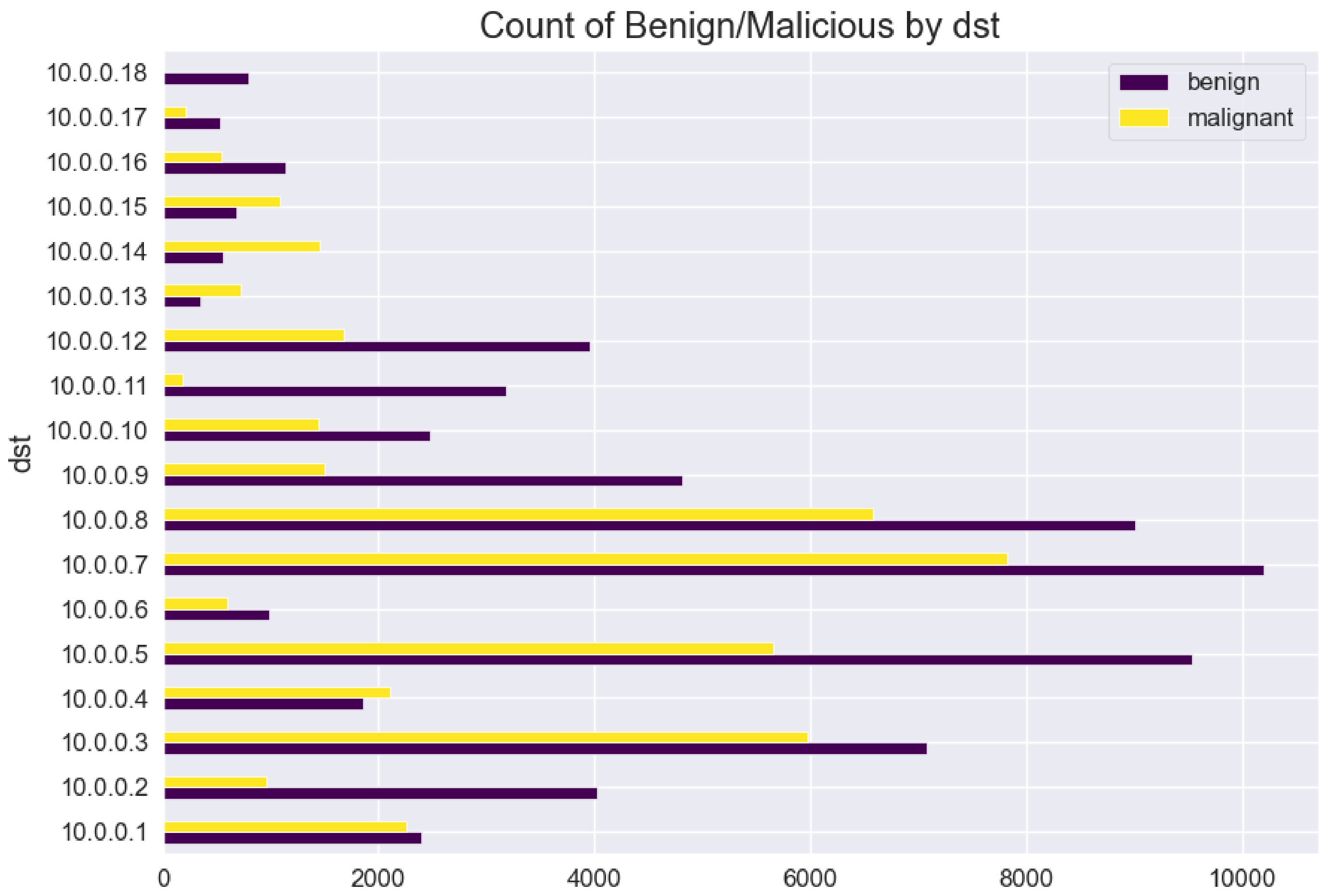


In [79]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0]
    fig, ax = plt.subplots(figsize=(10,7))
    (data
     .groupby(['label', 'src'])
     ['src']
     .describe()
     ["count"]
     .unstack(level=0)
     .reindex(['10.0.0.1', '10.0.0.2', '10.0.0.3', '10.0.0.4', '10.0.0.5', '10.0.0.6', '10.0.0.7', '10.0.0.8',
               '10.0.0.11', '10.0.0.12', '10.0.0.13', '10.0.0.14', '10.0.0.15', '10.0.0.16', '10.0.0.17', '10.0
     .plot
     .barh(rot="horizontal", title="Count of Benign/Malicious by src", ax=ax)
     .legend(labels=['benign', 'malignant'])
    );
```



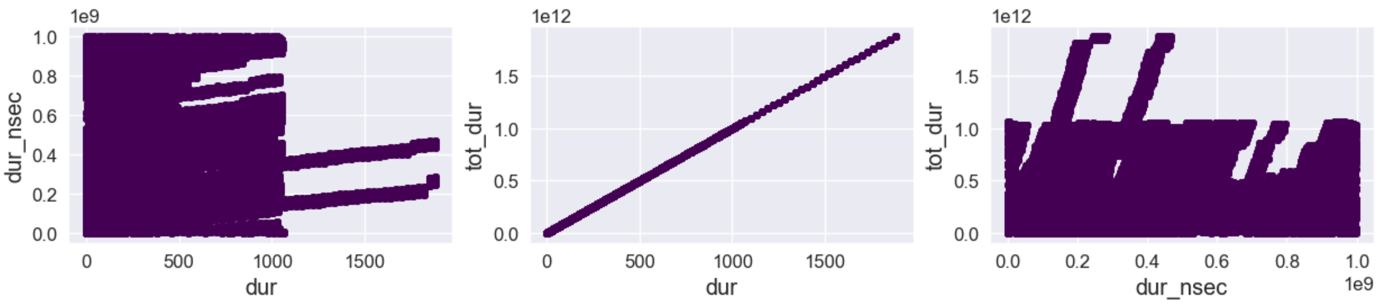
In [80]:
```
with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0]
    fig, ax = plt.subplots(figsize=(10,7))
    (data
     .groupby(['label', 'dst'])
     ['dst']
```

```
    .describe()
    ["count"]
    .unstack(level=0)
    .reindex(['10.0.0.1', '10.0.0.2', '10.0.0.3', '10.0.0.4', '10.0.0.5', '10.0.0.6', '10.0.0.7', '10.0.0.8',
              '10.0.0.11', '10.0.0.12', '10.0.0.13', '10.0.0.14', '10.0.0.15', '10.0.0.16', '10.0.0.17', '10.0
    .plot
    .barh(rot="horizontal", title="Count of Benign/Malicious by dst", ax=ax)
    .legend(labels=['benign', 'malignant'])
   );
```



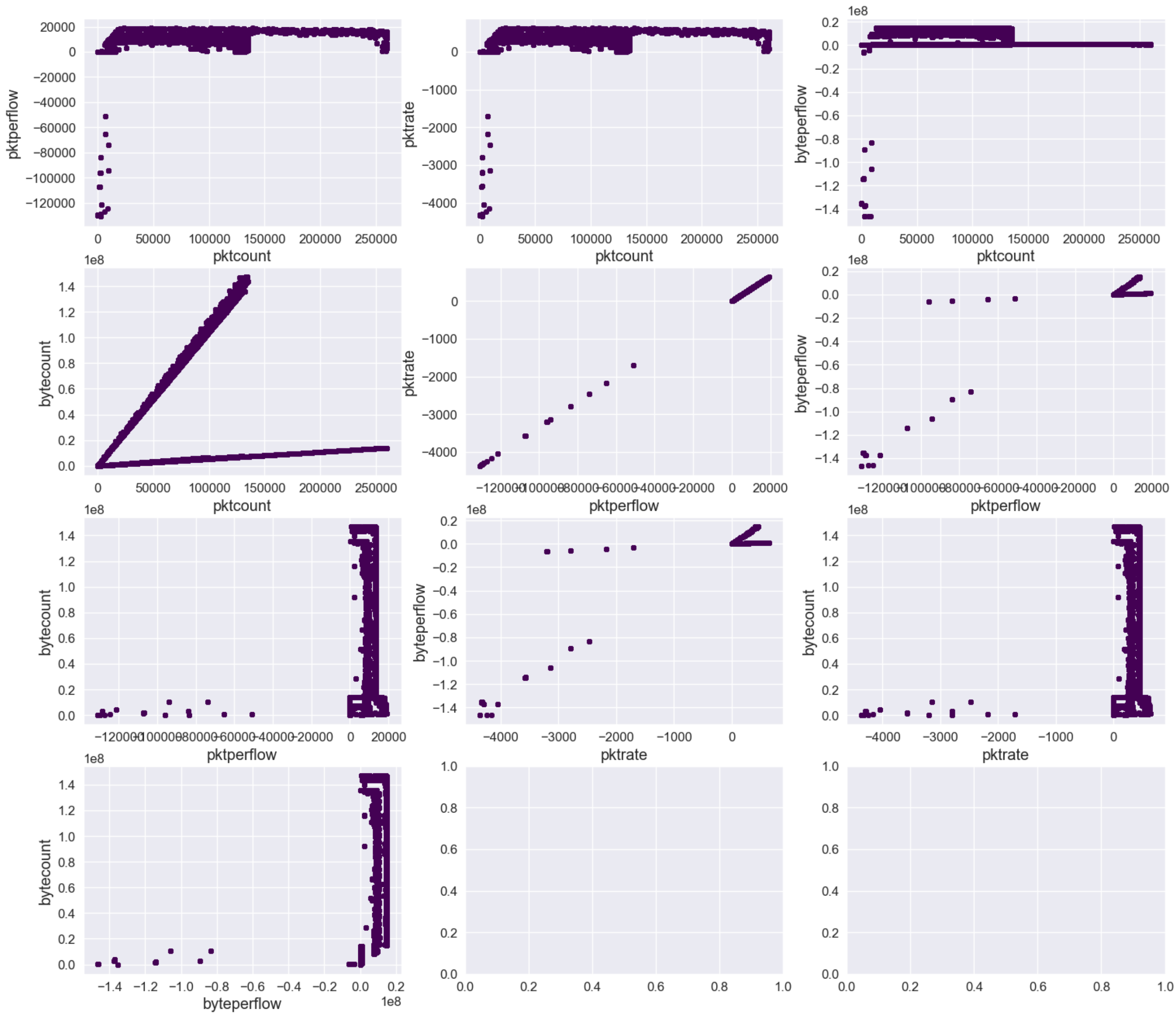
3.4. Relationship between features

In [81]:
```
n_cols = 3
n_elements = len(df_continuous[['dur', 'dur_nsec', 'tot_dur']].columns)
n_rows = np.ceil(n_elements / n_cols).astype("int")

with sns.plotting_context(rc={"font":"Roboto", "palette": color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, ax = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(15, n_rows * 2.5))

    (df_continuous
     .plot
     .scatter(x='dur', y='dur_nsec', ax=ax[0], color='#440154')
    )

    (df_continuous
     .plot
     .scatter(x='dur', y='tot_dur', ax=ax[1], color='#440154')
    )

    (df_continuous
     .plot
     .scatter(x='dur_nsec', y='tot_dur', ax=ax[2], color='#440154')
    )
```



In [82]:

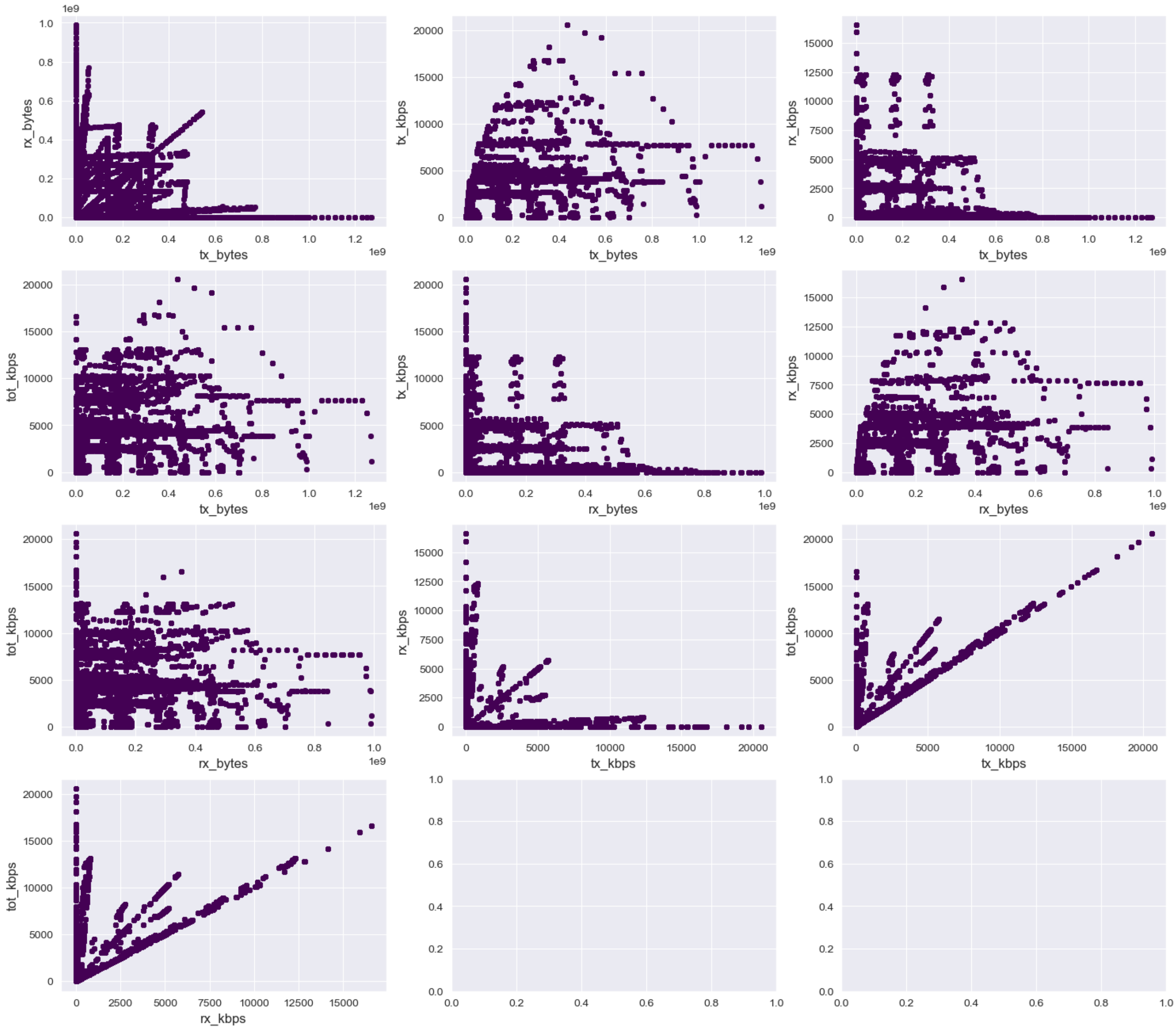
```
n_cols = 3
n_elements = len(df_continuous[["pktcount", "pktperflow", "pktrate", "byteperflow", "bytecount"]].columns)
n_rows = np.ceil(n_elements / n_cols).astype("int")
n_rows = 4

with sns.plotting_context(rc={"font":"Roboto", "palette":color_palette, "grid.linewidth":1.0, "font.size":12.0
    fig, axes = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(18, n_rows * 4.0))

    (df_continuous
     .plot
     .scatter(x='pktcount', y='pktperflow', ax=axes[0][0], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktcount', y='pktrate', ax=axes[0][1], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktcount', y='byteperflow', ax=axes[0][2], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktcount', y='bytecount', ax=axes[1][0], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktperflow', y='pktrate', ax=axes[1][1], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktperflow', y='byteperflow', ax=axes[1][2], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktperflow', y='bytecount', ax=axes[2][0], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktrate', y='byteperflow', ax=axes[2][1], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='pktrate', y='bytecount', ax=axes[2][2], color='#440154')
    );

    (df_continuous
     .plot
     .scatter(x='byteperflow', y='bytecount', ax=axes[3][0], color='#440154')
    );
```



In [83]:

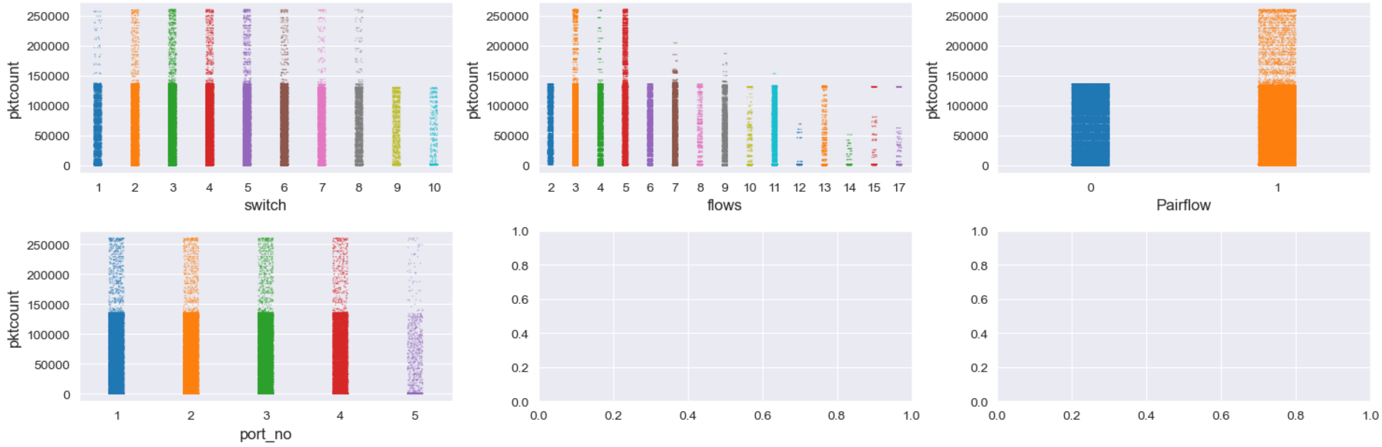
```
n_cols = 3
n_elements = len(df_continuous[["tx_bytes", "rx_bytes", "tx_kbps", "rx_kbps", "tot_kbps"]].columns)
# n_rows = np.ceil(n_elements / n_cols).astype("int")
n_rows = 4

fig, axes = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(18, n_rows * 4.0))


(df_continuous
 .plot
 .scatter(x='tx_bytes', y='rx_bytes', ax=axes[0][0], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='tx_bytes', y='tx_kbps', ax=axes[0][1], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='tx_bytes', y='rx_kbps', ax=axes[0][2], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='tx_bytes', y='tot_kbps', ax=axes[1][0], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='rx_bytes', y='tx_kbps', ax=axes[1][1], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='rx_bytes', y='rx_kbps', ax=axes[1][2], color='#440154')
```

```python
);

(df_continuous
 .plot
 .scatter(x='rx_bytes', y='tot_kbps', ax=axes[2][0], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='tx_kbps', y='rx_kbps', ax=axes[2][1], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='tx_kbps', y='tot_kbps', ax=axes[2][2], color='#440154')
);

(df_continuous
 .plot
 .scatter(x='rx_kbps', y='tot_kbps', ax=axes[3][0], color='#440154')
);
```

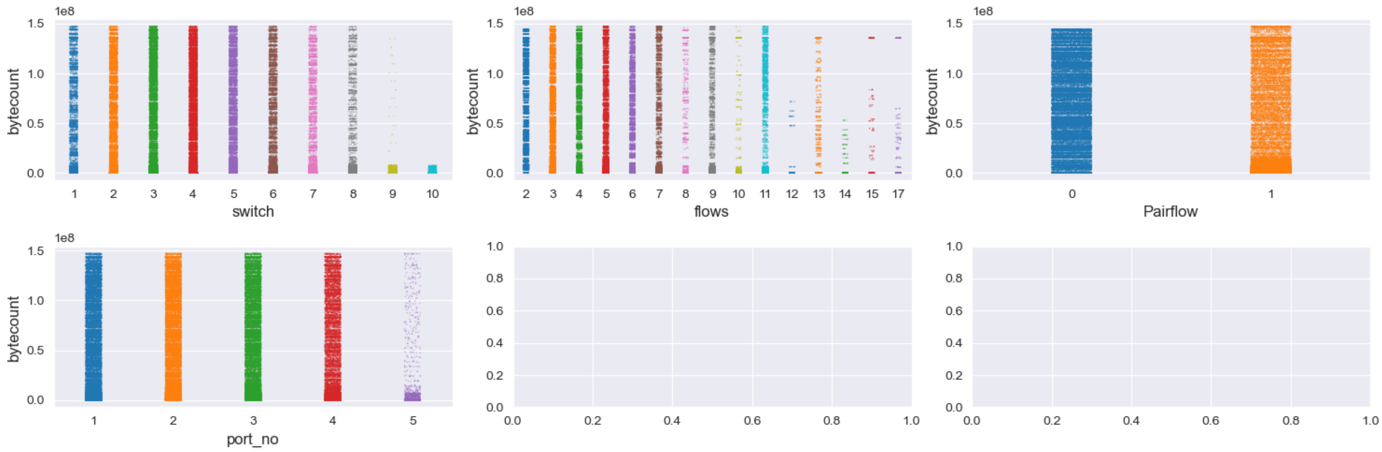


In [84]:

```python
n_cols = 3
n_elements = len(df_discrete.columns)
n_rows = np.ceil(n_elements / n_cols).astype("int")

y_value = data["pktcount"]

fig, axes = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(15, n_rows * 2.5))

for col, ax in zip(df_discrete.columns, axes.ravel()):
    sns.stripplot(data=data, x=col, y=y_value, ax=ax, palette="tab10", size=1, alpha=0.5)

plt.tight_layout();
```



In [85]:
```python
n_cols = 3
n_elements = len(df_discrete.columns)
n_rows = np.ceil(n_elements / n_cols).astype("int")

y_value = data["bytecount"]

fig, axes = plt.subplots(ncols=n_cols, nrows=n_rows, figsize=(15, n_rows * 2.5))

for col, ax in zip(df_discrete.columns, axes.ravel()):
    sns.stripplot(data=data, x=col, y=y_value, ax=ax, palette="tab10", size=1, alpha=0.5)

plt.tight_layout();
```



Insights:

1. Delete some variables with high multicollinearity

Predictive Analytics

In [1]:
```
import os
import warnings
import time
from IPython.display import display

import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import boxcox
from scipy.stats.contingency import association

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, FeatureUnion

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OrdinalEncoder, LabelEncoder, OneHotEncoder, Fu
from sklearn.impute import SimpleImputer, KNNImputer

from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import RFE

from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

import sklearn.metrics as skmet
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_auc_score, confusion_matrix, cla
```

In [2]:
```
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', 500)
```

In [3]:
```
def get_var(df, var_name):
    globals()[var_name] = df
    return df
```

In [4]:
```
data = (pd
        .read_csv("../../dataset/ddos_sdn/dataset_sdn.csv")
       )
```

In [5]:
```
(data
 .info()
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104345 entries, 0 to 104344
Data columns (total 23 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   dt           104345 non-null  int64
 1   switch       104345 non-null  int64
 2   src          104345 non-null  object
 3   dst          104345 non-null  object
 4   pktcount     104345 non-null  int64
 5   bytecount    104345 non-null  int64
 6   dur          104345 non-null  int64
 7   dur_nsec     104345 non-null  int64
 8   tot_dur      104345 non-null  float64
 9   flows        104345 non-null  int64
 10  packetins    104345 non-null  int64
 11  pktperflow   104345 non-null  int64
 12  byteperflow  104345 non-null  int64
 13  pktrate      104345 non-null  int64
 14  Pairflow     104345 non-null  int64
 15  Protocol     104345 non-null  object
 16  port_no      104345 non-null  int64
 17  tx_bytes     104345 non-null  int64
 18  rx_bytes     104345 non-null  int64
 19  tx_kbps      104345 non-null  int64
 20  rx_kbps      103839 non-null  float64
 21  tot_kbps     103839 non-null  float64
 22  label        104345 non-null  int64
dtypes: float64(3), int64(17), object(3)
memory usage: 18.3+ MB
```

In [6]:

```
(data
 .describe()
)
```

Out[6]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000

In [7]:

```
(data
 .loc[data.duplicated()]
)
```

Out[7]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	byteperf
13	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428
15	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	14428
30	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427
34	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427
40	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	14427
...	...	...	...	...	...	...	...	...	...	...	...	...	
33474	3249	8	10.0.0.12	10.0.0.5	88782	94641612	197	336000000	1.970000e+11	3	7894	13531	14424
33501	3609	8	10.0.0.3	10.0.0.5	119500	124519000	407	592000000	4.080000e+11	2	7916	7070	7366
33512	3609	8	10.0.0.3	10.0.0.5	119500	124519000	407	592000000	4.080000e+11	2	7916	7070	7366
33559	3159	8	10.0.0.12	10.0.0.5	48292	51479272	107	327000000	1.070000e+11	2	7503	13548	14442
33584	3249	8	10.0.0.3	10.0.0.5	14973	15601866	47	453000000	4.745300e+10	3	7894	9516	9915

5091 rows × 23 columns

In [8]:
```
class ClfSwitcher(BaseEstimator):

    def __init__(self, estimator = RandomForestClassifier()):
        self.estimator = estimator

    def fit(self, X, y=None, **kwargs):
        self.estimator.fit(X, y)
        return self

    def predict(self, X, y=None):
        return self.estimator.predict(X)

    def predict_proba(self, X):
        return self.estimator.predict_proba(X)

    def score(self, X, y):
        return self.estimator.score(X, y)
```

In [9]:
```
def outlier_thresholds(df: pd.DataFrame,
                       col: str,
                       q1: float = 0.05,
                       q3: float = 0.95):
    #1.5 as multiplier is a rule of thumb. Generally, the higher the multiplier,
    #the outlier threshold is set farther from the third quartile, allowing fewer data points to be classified

    return (df[col].quantile(q1) - 1.5 * (df[col].quantile(q3) - df[col].quantile(q1)),
            df[col].quantile(q3) + 1.5 * (df[col].quantile(q3) - df[col].quantile(q1)))
```

In [10]:
```
def loc_potential_outliers(df: pd.DataFrame,
                           col: str):

    low, high = outlier_thresholds(df, col)
    res = df.loc[(df[col] < low) | (df[col] > high)]
    print(f'Detected total of {len(res)} potential outliers')
    return res
```

In [11]:
```
def any_potential_outlier(df: pd.DataFrame,
                          col: str) -> int:

    low, high = outlier_thresholds(df, col)
    if (df
        .loc[(df[col] > high) | (df[col] < low)]
        .any(axis=None)):
        return df.loc[(df[col] > high) | (df[col] < low)].shape[0]
    else:
        return 0
```

In [12]:
```
def delete_potential_outlier(df: pd.DataFrame,
                             col: str) -> pd.DataFrame:

    low, high = outlier_thresholds(df, col)
    df.loc[(df[col]>high) | (df[col]<low),col] = np.nan
    return df
```

In [13]:
```
def delete_potential_outlier_list(df: pd.DataFrame,
                                  cols: list) -> pd.DataFrame:

    for item in cols:
        df = delete_potential_outlier(df, item)
    return df
```

In [14]:
```
#Drop duplicates
data = (data
        .drop_duplicates()
        .pipe(lambda df_: delete_potential_outlier_list(df_, ['pktcount', 'bytecount', 'dur', 'dur_nsec', 'pacl
       )
```

In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(data.drop(columns=['label']),
                                                    data[['label']].values.ravel(),
                                                    test_size=0.2,
                                                    random_state=42)
```

In [16]:
```
class TweakDDOS(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        return (X
```

```python
                .drop(columns=["dt", "tot_dur", "pktrate", "tot_kbps",])
                .assign(pktperflow= lambda df_: np.where(df_.pktperflow < 0, np.nan, df_.pktperflow),
                        byteperflow= lambda df_: np.where(df_.byteperflow < 0, np.nan, df_.byteperflow),)
                .astype({'switch': 'category',
                         'src': 'category',
                         'dst': 'category',
                         'flows': 'category',
                         'Pairflow': 'category',
                         'Protocol': 'category',
                         'port_no': 'category',})
                .pipe(get_var, 'pipeline_checkpoint1') #check intermediary result
               )
```

In [17]:
```python
data.columns.to_list()
```

Out[17]:
```
['dt',
 'switch',
 'src',
 'dst',
 'pktcount',
 'bytecount',
 'dur',
 'dur_nsec',
 'tot_dur',
 'flows',
 'packetins',
 'pktperflow',
 'byteperflow',
 'pktrate',
 'Pairflow',
 'Protocol',
 'port_no',
 'tx_bytes',
 'rx_bytes',
 'tx_kbps',
 'rx_kbps',
 'tot_kbps',
 'label']
```

In [18]:
```python
standard_numerical_features = ['pktcount', 'bytecount', 'dur', 'dur_nsec', 'packetins', 'pktperflow', 'byteper
standard_numerical_transformer = Pipeline(steps=[
    ('scale', StandardScaler())
])

ohe_categorical_features = ['switch', 'src', 'dst', 'Protocol', 'port_no']
ohe_categorical_transformer = Pipeline(steps=[
    ('ohe', OneHotEncoder(handle_unknown='ignore', sparse_output=False, drop='first'))
])

orde_categorical_features = ['flows', 'Pairflow']
orde_categorical_transformer = Pipeline(steps=[
    ('orde', OrdinalEncoder(dtype='float'))
])

col_trans = ColumnTransformer(
    transformers=[
        ('standard_numerical_features', standard_numerical_transformer, standard_numerical_features),
        ('ohe_categorical_features', ohe_categorical_transformer, ohe_categorical_features),
        ('orde_categorical_features', orde_categorical_transformer, orde_categorical_features),
    ],
    remainder='passthrough',
    verbose=0,
    verbose_feature_names_out=False,
    n_jobs=-1,)
```

In [19]:
```python
params_grid = [

    {'clf__estimator': [RandomForestClassifier()],
     'clf__estimator__n_estimators': [100],
     'clf__estimator__max_features': [3, 4, 5],
     'clf__estimator__max_depth': [2, 3],
    },

    {'clf__estimator': [SVC()],
     'clf__estimator__C': [0.1, 1],
     'clf__estimator__kernel': ['sigmoid'],
     'clf__estimator__gamma': ['auto']
    },

    {'clf__estimator': [KNeighborsClassifier()],
     'clf__estimator__n_neighbors': [5, 8],
     'clf__estimator__n_jobs': [-1],
```

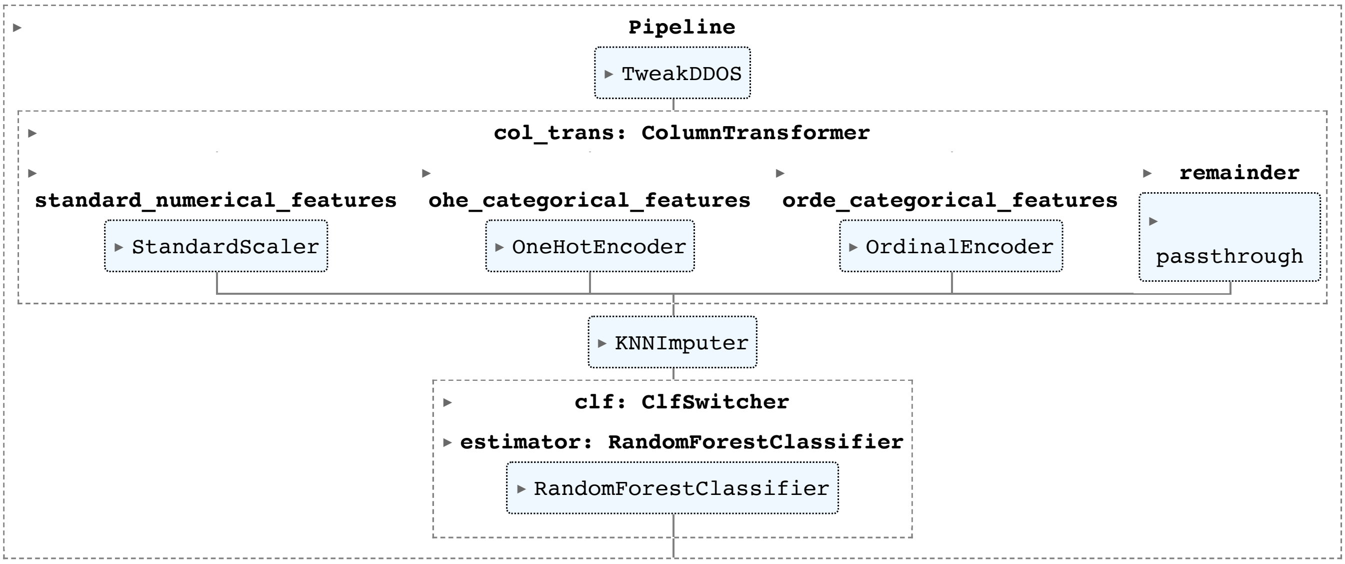
```
    }
]
```

Models with all Features

In [20]:
```python
pipeline = Pipeline(steps = [
    ('tweak_ddos', TweakDDOS()),
    ('col_trans', col_trans),
    ('imputer', KNNImputer(n_neighbors=5)),
    ('clf', ClfSwitcher()),
])
```

In [21]:
```python
pipeline
```

Out[21]:



In [22]:
```python
start = time.time()
grid = RandomizedSearchCV(pipeline, params_grid, cv=5, n_jobs=-1, return_train_score=False, scoring= ['accuracy
grid.fit(X_train, y_train)
end = time.time()
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

In [23]:
```python
total_time = end - start
print(f'Total time taken to for all model combinations: {total_time}')
```

```
Total time taken to for all model combinations: 389.9027647972107
```

RandomizedSearchCV Summary Table

In [24]:
```python
pd.DataFrame(grid.cv_results_)
```

Out[24]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf__estimator__n_estimators	param_clf__estimator__max_fe
0	9.911682	0.155951	1.264010	0.071503	100	
1	9.937751	0.208413	1.309055	0.044273	100	
2	9.457320	0.340877	1.381980	0.085891	100	
3	9.217384	0.188704	1.425150	0.034780	100	
4	9.443858	0.935446	1.904988	0.184854	100	
5	9.960010	0.085438	2.217296	0.121973	100	
6	316.188313	19.322583	21.876175	1.076428	NaN	
7	305.182136	16.123959	21.970616	0.892117	NaN	
8	4.003314	0.086331	6.622456	0.247147	NaN	
9	4.024444	0.057864	6.100951	0.433512	NaN	

Best Estimator

In [25]:
```
print("Best estimator:\n{}".format(grid.best_estimator_))
```

```
Best estimator:
Pipeline(steps=[('tweak_ddos', TweakDDOS()),
                ('col_trans',
                 ColumnTransformer(n_jobs=-1, remainder='passthrough',
                                   transformers=[('standard_numerical_features',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['pktcount', 'bytecount',
                                                   'dur', 'dur_nsec',
                                                   'packetins', 'pktperflow',
                                                   'byteperflow', 'tx_bytes',
                                                   'rx_bytes', 'tx_kbps',
                                                   'rx_kbps']),
                                                 ('ohe_categorical_features',...
                                                                   OneHotEncoder(drop='first',
                                                                                 handle_unknown='ignore',
                                                                                 sparse_output=False))]),
                                                  ['switch', 'src', 'dst',
                                                   'Protocol', 'port_no']),
                                                 ('orde_categorical_features',
                                                  Pipeline(steps=[('orde',
                                                                   OrdinalEncoder(dtype='float'))]),
                                                  ['flows', 'Pairflow'])],
                                   verbose=0,
                                   verbose_feature_names_out=False)),
                ('imputer', KNNImputer()),
                ('clf',
                 ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1)))])
```

In [26]:
```
print("Best estimator:\n{}".format(grid.best_estimator_.named_steps['clf']))
```

```
Best estimator:
ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1))
```

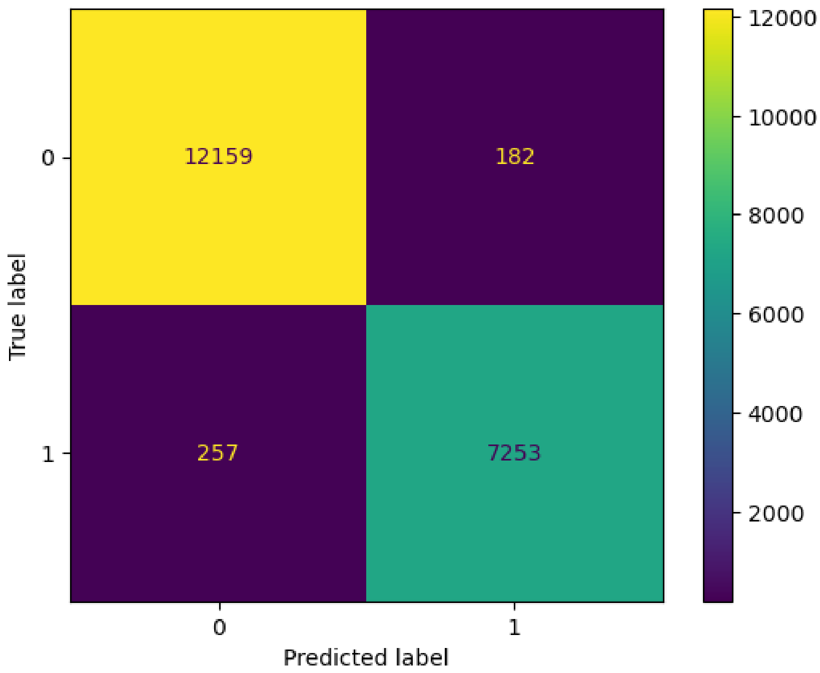
In [27]:
```
print(f'Best params: {grid.best_params_}')
print(f'Best CV score: {grid.best_score_}')
print(f'Validation-set score: {grid.score(X_test, y_test)}')
```

```
Best params: {'clf__estimator__n_neighbors': 5, 'clf__estimator__n_jobs': -1, 'clf__estimator': KNeighborsClas
sifier(n_jobs=-1)}
Best CV score: 0.975857350251564
Validation-set score: 0.9778852450758149
```

In [28]:
```
grid.predict(X_test)
```

Out[28]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

In [29]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test, grid.predict(X_test))).plot();
```



In [30]:
```python
print(f'Accuracy score: {accuracy_score(y_test, grid.predict(X_test))}')
print(f'Precision score: {precision_score(y_test, grid.predict(X_test))}')
print(f'Recall score: {recall_score(y_test, grid.predict(X_test))}')
print(f'ROC-AUC score: {roc_auc_score(y_test, grid.predict(X_test))}')
```

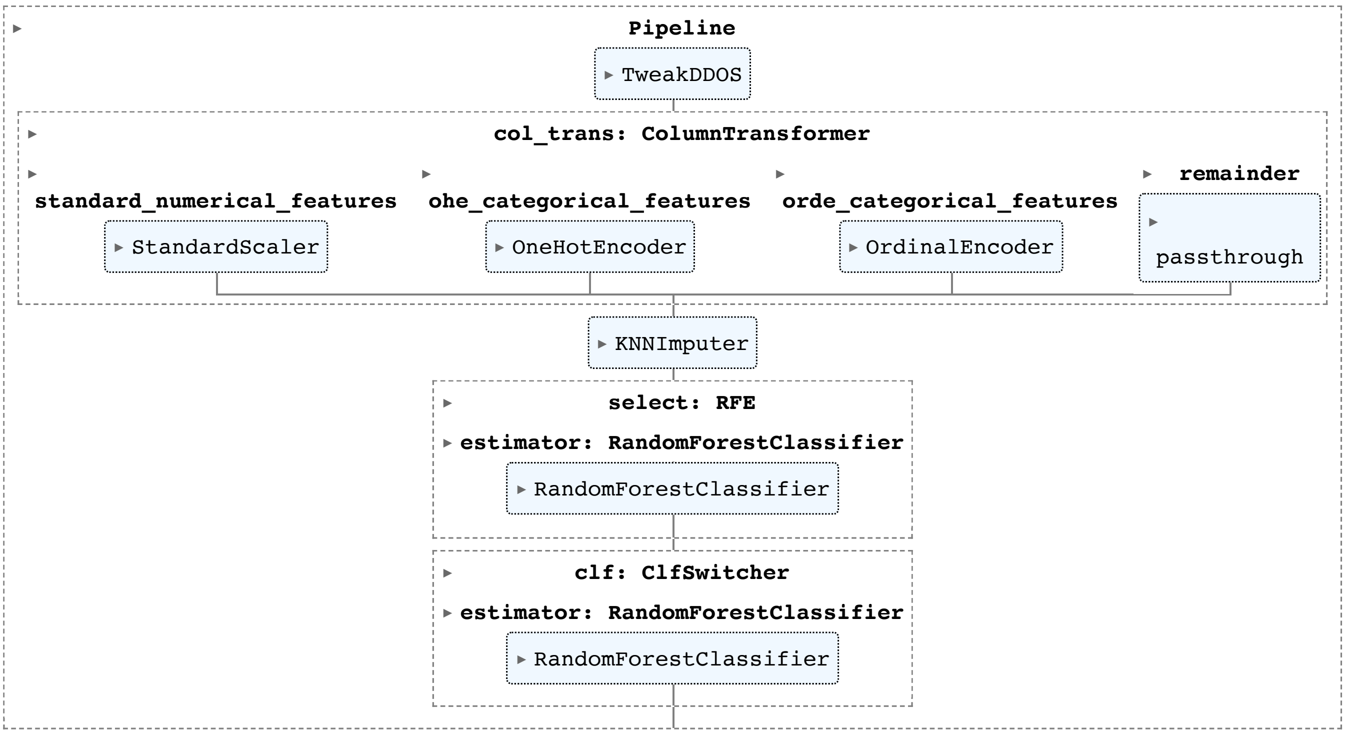
```
Accuracy score: 0.9778852450758149
Precision score: 0.975521183591123
Recall score: 0.9657789613848202
ROC-AUC score: 0.9755156860242309
```

Models with RFE

In [31]:
```python
pipeline = Pipeline(steps = [
    ('tweak_ddos', TweakDDOS()),
    ('col_trans', col_trans),
    ('imputer', KNNImputer(n_neighbors=5)),
    ('select', RFE(estimator=RandomForestClassifier(n_estimators=100, max_features=5, max_depth=3),
                   n_features_to_select=5)),
    ('clf', ClfSwitcher()),
])
```

In [32]:
```python
pipeline
```

Out[32]:



In [33]:
```python
start = time.time()
grid = RandomizedSearchCV(pipeline, params_grid, cv=5, n_jobs=-1, return_train_score=False, scoring= ['accuracy
grid.fit(X_train, y_train)
end = time.time()
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

In [34]:
```python
total_time = end - start
print(f'Total time taken to for all model combinations: {total_time}')
```

```
Total time taken to for all model combinations: 850.5236768722534
```

RandomizedSearchCV Summary Table

In [35]:
```python
pd.DataFrame(grid.cv_results_)
```

Out[35]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf__estimator__n_estimators	param_clf__estimator__max_fe
0	127.840107	1.389782	1.102601	0.104146	100	
1	127.466722	0.986384	1.174974	0.074041	100	
2	127.640082	1.982510	1.048479	0.121998	100	
3	125.518639	0.685731	1.121923	0.129123	100	
4	125.847267	1.728732	1.081889	0.078363	100	
5	126.361964	0.892071	1.327079	0.177044	100	
6	197.733585	2.700151	10.925061	0.628801	NaN	
7	197.310020	2.699516	11.270596	1.124497	NaN	
8	123.132504	1.522191	0.990048	0.101899	NaN	
9	124.449963	2.080926	0.865089	0.060362	NaN	

Best Estimator

In [36]:
```python
print("Best estimator:\n{}".format(grid.best_estimator_))
```

```
Best estimator:
Pipeline(steps=[('tweak_ddos', TweakDDOS()),
                ('col_trans',
                 ColumnTransformer(n_jobs=-1, remainder='passthrough',
                                   transformers=[('standard_numerical_features',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['pktcount', 'bytecount',
                                                   'dur', 'dur_nsec',
                                                   'packetins', 'pktperflow',
                                                   'byteperflow', 'tx_bytes',
                                                   'rx_bytes', 'tx_kbps',
                                                   'rx_kbps']),
                                                 ('ohe_categorical_features',...
                                                   'Protocol', 'port_no']),
                                                 ('orde_categorical_features',
                                                  Pipeline(steps=[('orde',
                                                                   OrdinalEncoder(dtype='float'))]),
                                                  ['flows', 'Pairflow'])],
                                   verbose=0,
                                   verbose_feature_names_out=False)),
                ('imputer', KNNImputer()),
                ('select',
                 RFE(estimator=RandomForestClassifier(max_depth=3,
                                                      max_features=5),
                     n_features_to_select=5)),
                ('clf',
                 ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1)))])
```

In [37]:
```python
print("Best estimator:\n{}".format(grid.best_estimator_.named_steps['clf']))
```

```
Best estimator:
ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1))
```

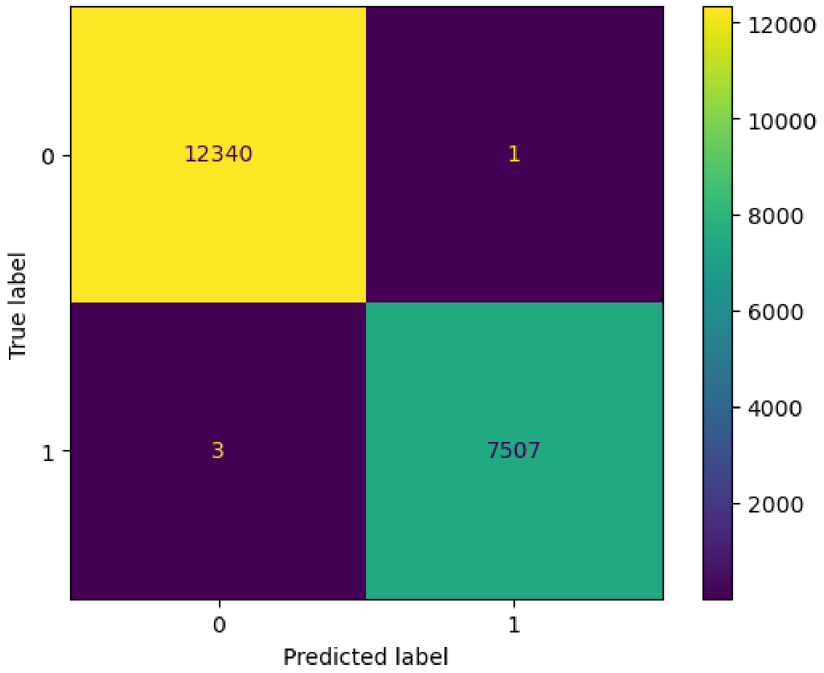
In [38]:
```python
print(f'Best params: {grid.best_params_}')
print(f'Best CV score: {grid.best_score_}')
print(f'Validation-set score: {grid.score(X_test, y_test)}')
```

```
Best params: {'clf__estimator__n_neighbors': 5, 'clf__estimator__n_jobs': -1, 'clf__estimator': KNeighborsClas
sifier(n_jobs=-1)}
Best CV score: 0.9992695444090867
Validation-set score: 0.9997984988161805
```

In [39]:
```python
grid.predict(X_test)
```

Out[39]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

In [40]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test, grid.predict(X_test))).plot();
```



In [41]:
```python
print(f'Accuracy score: {accuracy_score(y_test, grid.predict(X_test))}')
print(f'Precision score: {precision_score(y_test, grid.predict(X_test))}')
print(f'Recall score: {recall_score(y_test, grid.predict(X_test))}')
print(f'ROC-AUC score: {roc_auc_score(y_test, grid.predict(X_test))}')
```

```
Accuracy score: 0.9997984988161805
Precision score: 0.9998668087373468
Recall score: 0.9996005326231691
ROC-AUC score: 0.9997597509562649
```

Insights:

1. Best Estimator is **KNN** with `n_neighbors=5`

In [42]:
```python
support = grid.best_estimator_.named_steps['select'].get_support()
```

In [43]:
```python
grid.best_estimator_.named_steps['select'].ranking_
```

Out[43]:
```
array([ 1,  1,  3, 14,  1,  1,  1,  8, 19, 12, 11, 36, 42, 33, 27, 43, 41,
       35, 46, 55,  4, 26,  7, 13, 15, 23, 54, 58, 18,  9, 21, 40, 34, 24,
       49, 16, 20, 30, 50, 17, 53, 47, 45, 22, 57, 44, 56, 28, 39, 29, 31,
       32, 37, 25, 38,  2,  5, 48, 51, 52, 59, 10,  6])
```

In [44]:
```python
grid.best_estimator_.named_steps['select'].get_feature_names_out()
```

Out[44]:
```
array(['x0', 'x1', 'x4', 'x5', 'x6'], dtype=object)
```

In [45]:
```python
feature_names = grid.best_estimator_.named_steps['col_trans'].get_feature_names_out()
```

In [46]:
```python
np.array(feature_names)[support]
```

Out[46]:
```
array(['pktcount', 'bytecount', 'packetins', 'pktperflow', 'byteperflow'],
      dtype=object)
```

Models with PCA

In [47]:
```
pipeline = Pipeline(steps = [
    ('tweak_ddos', TweakDDOS()),
    ('col_trans', col_trans),
    ('imputer', KNNImputer(n_neighbors=5)),
    ('pca', PCA(n_components=5)),
    ('clf', ClfSwitcher()),
])
```

In [48]:
```
start = time.time()
grid = RandomizedSearchCV(pipeline, params_grid, cv=5, n_jobs=-1, return_train_score=False, scoring= ['accuracy
grid.fit(X_train, y_train)
end = time.time()
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

In [49]:
```
total_time = end - start
print(f'Total time taken to for all model combinations: {total_time}')
```

```
Total time taken to for all model combinations: 185.2110641002655
```

RandomizedSearchCV Summary Table

In [50]:
```
pd.DataFrame(grid.cv_results_)
```

Out[50]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_clf__estimator__n_estimators	param_clf__estimator__max_fe
0	11.550738	0.189608	1.242019	0.054139	100	
1	12.431185	0.925364	1.267442	0.221396	100	
2	12.402073	0.126310	1.294169	0.170338	100	
3	11.707410	0.472535	1.812607	0.344990	100	
4	14.286285	0.201206	1.131432	0.131050	100	
5	16.119315	0.672735	1.574014	0.078231	100	
6	109.827759	17.704490	12.790444	2.953247	NaN	
7	108.391840	17.146400	11.262120	0.987596	NaN	
8	4.272716	0.370159	1.231297	0.077853	NaN	
9	4.032245	0.272842	1.237535	0.087692	NaN	

Best Estimator

In [51]:
```
print("Best estimator:\n{}".format(grid.best_estimator_))
```

```
Best estimator:
Pipeline(steps=[('tweak_ddos', TweakDDOS()),
                ('col_trans',
                 ColumnTransformer(n_jobs=-1, remainder='passthrough',
                                   transformers=[('standard_numerical_features',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['pktcount', 'bytecount',
                                                   'dur', 'dur_nsec',
                                                   'packetins', 'pktperflow',
                                                   'byteperflow', 'tx_bytes',
                                                   'rx_bytes', 'tx_kbps',
                                                   'rx_kbps']),
                                                 ('ohe_categorical_features',...
                                                                                handle_unknown='ignore',
                                                                                sparse_output=False))]),
                                                  ['switch', 'src', 'dst',
                                                   'Protocol', 'port_no']),
                                                 ('orde_categorical_features',
                                                  Pipeline(steps=[('orde',
                                                                   OrdinalEncoder(dtype='float'))]),
                                                  ['flows', 'Pairflow'])],
                                   verbose=0,
                                   verbose_feature_names_out=False)),
                ('imputer', KNNImputer()), ('pca', PCA(n_components=5)),
                ('clf',
                 ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1)))])
```

In [52]:
```python
print("Best estimator:\n{}".format(grid.best_estimator_.named_steps['clf']))
```

```
Best estimator:
ClfSwitcher(estimator=KNeighborsClassifier(n_jobs=-1))
```

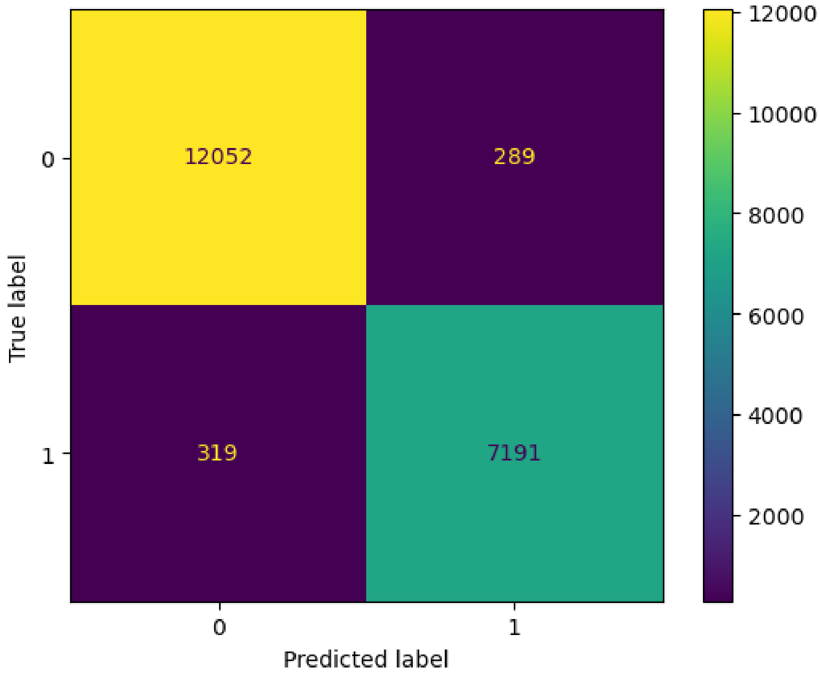
In [53]:
```python
print(f'Best params: {grid.best_params_}')
print(f'Best CV score: {grid.best_score_}')
print(f'Validation-set score: {grid.score(X_test, y_test)}')
```

```
Best params: {'clf__estimator__n_neighbors': 5, 'clf__estimator__n_jobs': -1, 'clf__estimator': KNeighborsClas
sifier(n_jobs=-1)}
Best CV score: 0.9659081166807857
Validation-set score: 0.9693718200594429
```

In [54]:
```python
grid.predict(X_test)
```

Out[54]:
```
array([1, 0, 0, ..., 0, 0, 0])
```

In [55]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test, grid.predict(X_test))).plot();
```



In [56]:
```python
print(f'Accuracy score: {accuracy_score(y_test, grid.predict(X_test))}')
print(f'Precision score: {precision_score(y_test, grid.predict(X_test))}')
print(f'Recall score: {recall_score(y_test, grid.predict(X_test))}')
print(f'ROC-AUC score: {roc_auc_score(y_test, grid.predict(X_test))}')
```

```
Accuracy score: 0.9693718200594429
Precision score: 0.9613636363636363
Recall score: 0.9575233022636485
ROC-AUC score: 0.9670527134444407
```

Models with t-SNE

In [57]:

```
pipeline_checkpoint1
```

Out[57]:

	switch	src	dst	pktcount	bytecount	dur	dur_nsec	flows	packetins	pktperflow	byteperflow	Pairflow	P
38301	6	10.0.0.4	10.0.0.15	68240.0	3957920.0	227.0	991000000.0	5	10817.0	7907.0	458606.0	1	
1781	2	10.0.0.4	10.0.0.8	18301.0	19508866.0	40.0	655000000.0	2	2175.0	13640.0	14540240.0	0	
26208	4	10.0.0.8	10.0.0.5	12170.0	12973220.0	26.0	680000000.0	3	8803.0	0.0	0.0	0	
96438	4	10.0.0.5	10.0.0.8	509.0	49882.0	521.0	844000000.0	9	1264.0	29.0	2842.0	0	
89381	4	10.0.0.10	10.0.0.4	207.0	20286.0	212.0	741000000.0	9	4942.0	29.0	2842.0	1	
...	...	...	...	...	...	...	...	...	...	...	...	...	
96860	1	10.0.0.9	10.0.0.2	30.0	2940.0	31.0	429000000.0	3	1278.0	29.0	2842.0	0	
65880	4	10.0.0.7	10.0.0.6	459.0	44982.0	469.0	964000000.0	13	2053.0	30.0	2940.0	1	
98597	5	10.0.0.5	10.0.0.11	999.0	97902.0	1051.0	915000000.0	13	3421.0	1.0	98.0	0	
101763	3	10.0.0.2	10.0.0.9	939.0	92022.0	961.0	673000000.0	5	3443.0	29.0	2842.0	0	
58089	6	10.0.0.12	10.0.0.5	398.0	39004.0	408.0	364000000.0	7	3024.0	29.0	2842.0	1	

19851 rows × 18 columns

In [58]:

```
grid.best_estimator_.named_steps['col_trans'].fit_transform(pipeline_checkpoint1)
```

Out[58]:

```
array([[ 0.3115381 , -0.66526378, -0.35067898, ...,  0.        ,
         3.        ,  1.        ],
       [-0.64484209, -0.34283938, -1.00650517, ...,  0.        ,
         0.        ,  0.        ],
       [-0.76225668, -0.47834572, -1.05560446, ...,  0.        ,
         1.        ,  0.        ],
       ...,
       [-0.97619214, -0.74529517,  2.53916477, ...,  0.        ,
        11.        ,  0.        ],
       [-0.9773412 , -0.74541709,  2.2235265 , ...,  0.        ,
         3.        ,  0.        ],
       [-0.98770187, -0.74651633,  0.28410466, ...,  0.        ,
         5.        ,  1.        ]])
```

In [59]:

```
n_components = 2
tsne = TSNE(n_components)
tsne_result = tsne.fit_transform(KNNImputer(n_neighbors=5).fit_transform(grid.best_estimator_.named_steps['col_
```

In [60]:

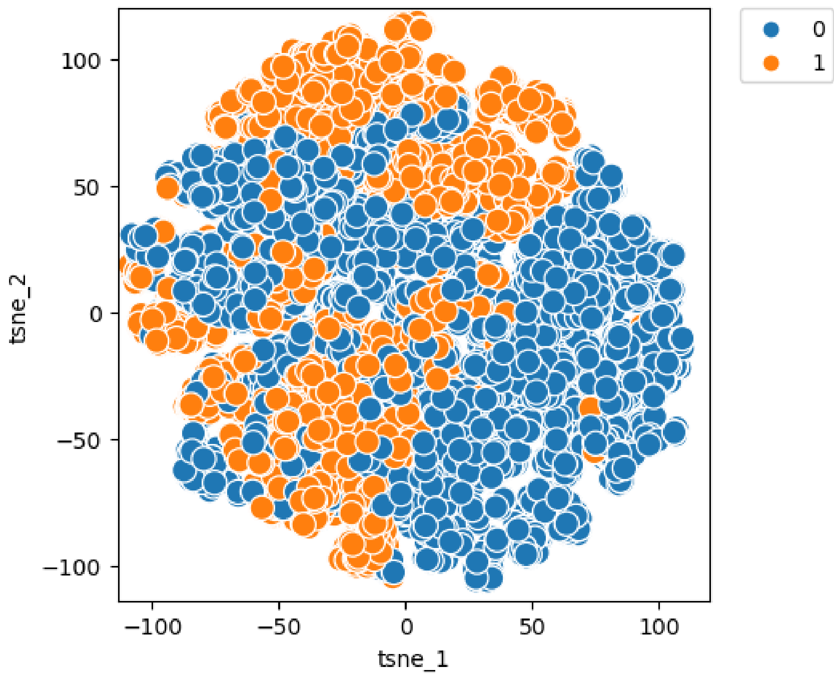
```
tsne_result
```

Out[60]:

```
array([[-41.06658 , -33.567894],
       [-83.46477 ,  60.11285 ],
       [-19.255608, -94.466576],
       ...,
       [105.77756 ,  22.610973],
       [ 62.99284 , -59.80178 ],
       [ 48.513172, -19.531235]], dtype=float32)
```

In [61]:

```
tsne_result_df = pd.DataFrame({'tsne_1': tsne_result[:,0], 'tsne_2': tsne_result[:,1], 'label': y_test})
fig, ax = plt.subplots(1)
sns.scatterplot(x='tsne_1', y='tsne_2', hue='label', data=tsne_result_df, ax=ax, s=120)
lim = (tsne_result.min()-5, tsne_result.max()+5)
ax.set_xlim(lim)
ax.set_ylim(lim)
ax.set_aspect('equal')
ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0);
```



0
1
100
50
0
−50
−100
tsne_2
−100
−50
0
50
100
tsne_1