Import Essential Modules

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense
from sklearn import metrics
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import roc_curve
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.naive_bayes import CategoricalNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import AdaBoostClassifier
import xgboost as xgb
from sklearn.decomposition import PCA
from mlxtend.plotting import plot_decision_regions
```

```
from mlxtend.plotting import plot_decision_regions
from sklearn.tree import export_graphviz
from six import StringIO
from IPython.display import Image
import graphviz
%matplotlib inline
```

In [6]:

```
df = pd.read_csv(r'downloads/dataset_sdn.csv')
df.head(10)
```

Out[6]:

	dt	switch	src	dst	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	...	pktrate	Pairflow	Protocol	port_no	tx
0	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	3	1439
1	11605	1	10.0.0.1	10.0.0.8	126395	134737070	280	734000000	2.810000e+11	2	...	451	0	UDP	4	
2	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	
3	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	2	
4	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	3	
5	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	1	
6	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	4	
7	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	1	
8	11425	1	10.0.0.1	10.0.0.8	45304	48294064	100	716000000	1.010000e+11	3	...	451	0	UDP	2	
9	11425	1	10.0.0.2	10.0.0.8	90333	96294978	200	744000000	2.010000e+11	3	...	451	0	UDP	4	3545

10 rows × 23 columns

Data Preprocessing

Dataset Dimensions

In [3]:
```
print("This Dataset has {} rows and {} columns".format(df.shape[0], df.shape[1]))
```

```
This Dataset has 104345 rows and 23 columns
```

Concise summary of dataset

In [4]:
```
df.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
```

Descriptive statistics of dataset

In [5]:
```
df.describe()
```

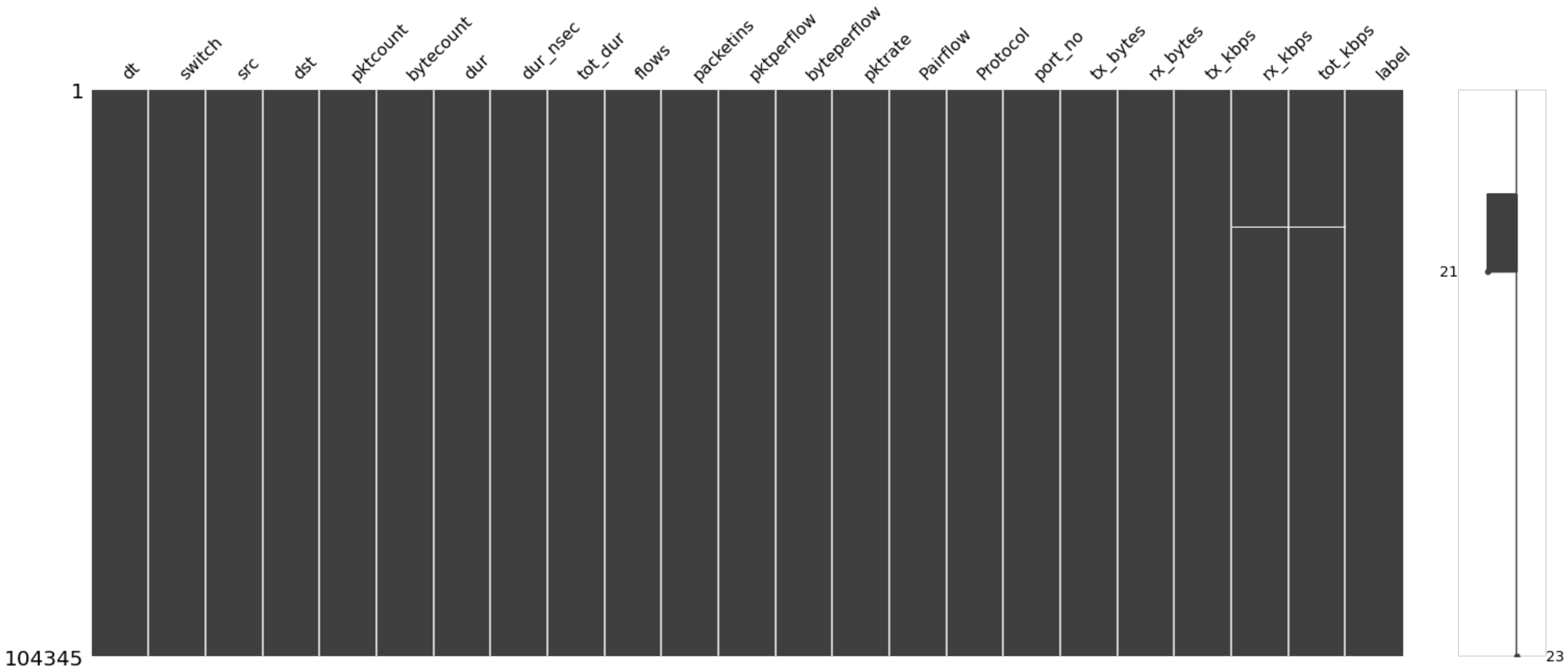
Out[5]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins
count	104345.000000	104345.000000	104345.000000	1.043450e+05	104345.000000	1.043450e+05	1.043450e+05	104345.000000	104345.00000(
mean	17927.514169	4.214260	52860.954746	3.818660e+07	321.497398	4.613880e+08	3.218865e+11	5.654234	5200.38346ε
std	11977.642655	1.956327	52023.241460	4.877748e+07	283.518232	2.770019e+08	2.834029e+11	2.950036	5257.00145(
min	2488.000000	1.000000	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	2.000000	4.00000(
25%	7098.000000	3.000000	808.000000	7.957600e+04	127.000000	2.340000e+08	1.270000e+11	3.000000	1943.00000(
50%	11905.000000	4.000000	42828.000000	6.471930e+06	251.000000	4.180000e+08	2.520000e+11	5.000000	3024.00000(
75%	29952.000000	5.000000	94796.000000	7.620354e+07	412.000000	7.030000e+08	4.130000e+11	7.000000	7462.00000(
max	42935.000000	10.000000	260006.000000	1.471280e+08	1881.000000	9.990000e+08	1.880000e+12	17.000000	25224.00000(

heatmap of missing values

In [6]:
```
msno.matrix(df)
```

Out[6]: <AxesSubplot:>



Count of null values in each feature

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
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 [8]:
```
(df.isnull().sum()/df.isnull().count())*100
```

Out[8]:
```
dt             0.00000
switch         0.00000
src            0.00000
dst            0.00000
pktcount       0.00000
bytecount      0.00000
dur            0.00000
dur_nsec       0.00000
tot_dur        0.00000
flows          0.00000
packetins      0.00000
pktperflow     0.00000
byteperflow    0.00000
pktrate        0.00000
Pairflow       0.00000
Protocol       0.00000
port_no        0.00000
tx_bytes       0.00000
rx_bytes       0.00000
tx_kbps        0.00000
rx_kbps        0.48493
tot_kbps       0.48493
label          0.00000
dtype: float64
```

Drop rows with null values

In [9]:
```
df.dropna(inplace=True)
```

Info after handling Null Values

In [10]:
```
print(df.isnull().sum())
print("This Dataframe has {} rows and {} columns after removing null values".format(df.shape[0], df.shap
```

```
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        0
tot_kbps       0
label          0
dtype: int64
This Dataframe has 103839 rows and 23 columns after removing null values
```

Distribution of Target Class

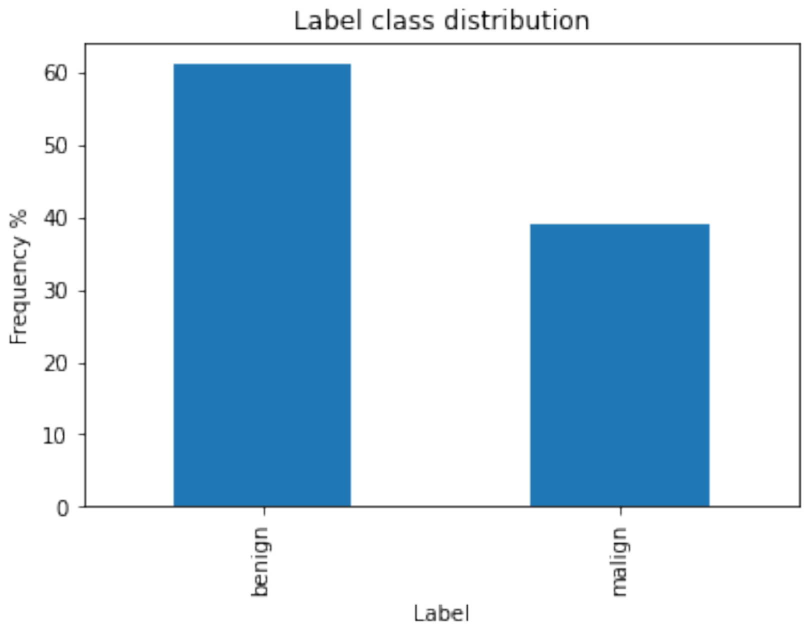
In [11]:
```
malign = df[df['label'] == 1]
benign = df[df['label'] == 0]

print('Number of DDOS attacks that has occured :',round((len(malign)/df.shape[0])*100,2),'%')
print('Number of DDOS attacks that has not occured :',round((len(benign)/df.shape[0])*100,2),'%')
```

```
Number of DDOS attacks that has occured : 39.01 %
Number of DDOS attacks that has not occured : 60.99 %
```

Barplot of Target Class

In [12]:
```
# Let's plot the Label class against the Frequency
labels = ['benign','malign']
classes = pd.value_counts(df['label'], sort = True) / df['label'].count() *100
classes.plot(kind = 'bar')
plt.title("Label class distribution")
plt.xticks(range(2), labels)
plt.xlabel("Label")
plt.ylabel("Frequency %")
```

Out[12]:
```
Text(0, 0.5, 'Frequency %')
```

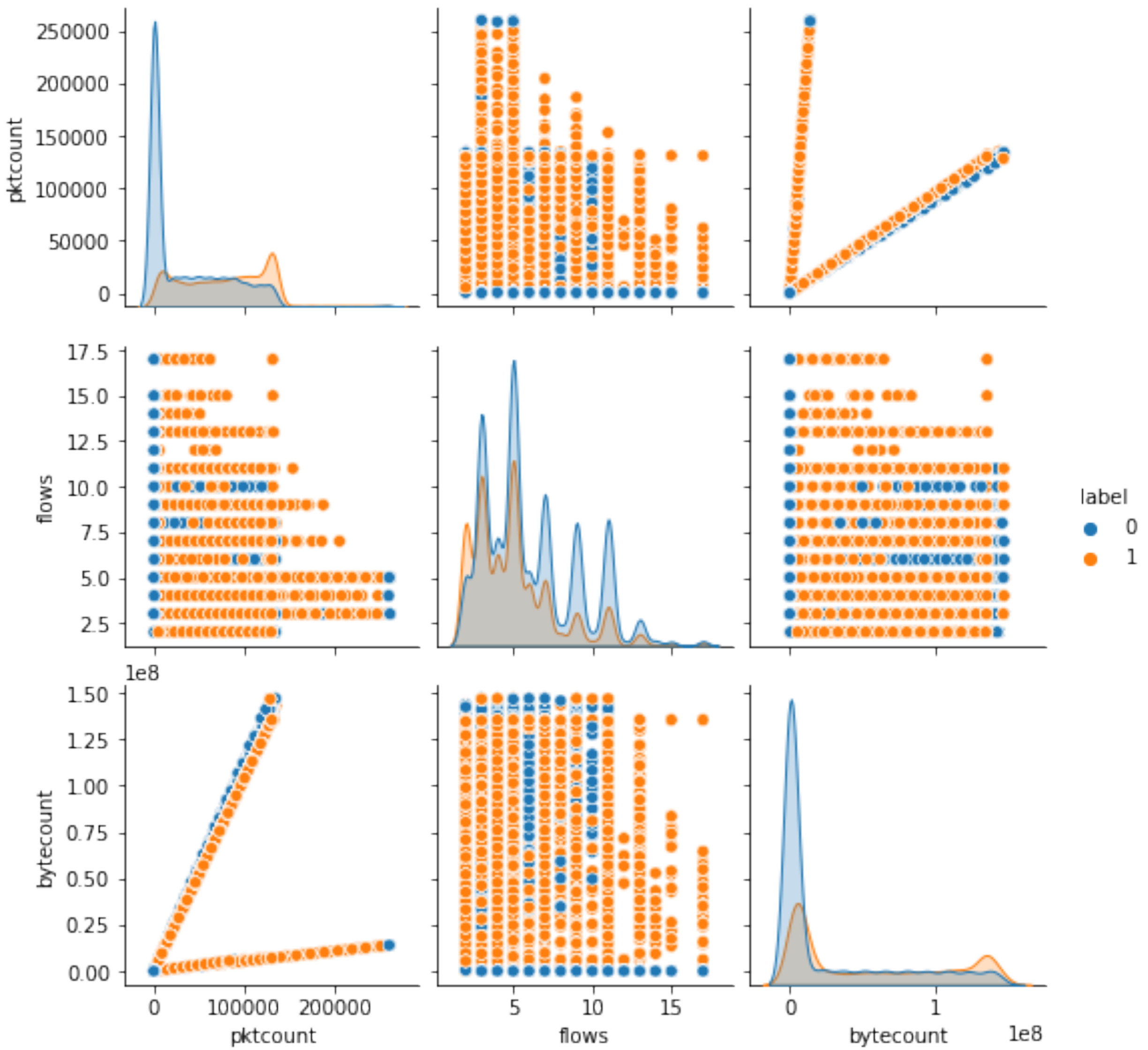


Pairplot of select features

In [13]:

```
sns.pairplot(df,hue="label",vars=['pktcount','flows','bytecount'])
```

Out[13]: <seaborn.axisgrid.PairGrid at 0x2356a24b610>



Columns in the dataset

In [14]:
```
df.columns
```

Out[14]:
```
Index(['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'],
      dtype='object')
```

Unique values in each column

In [15]:
```
print(df.apply(lambda col: col.unique()))
```

```
dt              [11425, 11605, 11455, 11515, 9906, 11335, 1157...
switch                                 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
src             [10.0.0.1, 10.0.0.2, 10.0.0.4, 10.0.0.10, 10.0...
dst             [10.0.0.8, 10.0.0.7, 10.0.0.3, 10.0.0.5, 10.0....
pktcount        [45304, 126395, 90333, 103866, 85676, 32914, 4...
bytecount       [48294064, 134737070, 96294978, 110721156, 913...
dur             [100, 280, 200, 230, 190, 73, 10, 250, 80, 260...
dur_nsec        [716000000, 734000000, 744000000, 747000000, 7...
tot_dur         [101000000000.0, 281000000000.0, 201000000000....
flows           [3, 2, 4, 5, 6, 7, 8, 11, 9, 10, 13, 15, 17, 1...
packetins       [1943, 1931, 1790, 1306, 1910, 2242, 2175, 110...
pktperflow      [13535, 13531, 13534, 13533, 13306, 13385, 0, ...
byteperflow     [14428310, 14424046, 14427244, 14426178, 14184...
pktrate         [451, 443, 446, 0, 288, 450, 448, 449, 455, 14...
Pairflow                                                  [0, 1]
Protocol                                       [UDP, TCP, ICMP]
port_no                                         [3, 4, 1, 2, 5]
tx_bytes        [143928631, 3842, 3795, 3688, 3413, 3665, 3775...
rx_bytes        [3917, 3520, 1242, 1492, 3665, 1402, 3413, 429...
tx_kbps         [0, 16578, 19164, 12831, 7676, 10271, 2587, 16...
rx_kbps         [0.0, 6307.0, 3838.0, 6400.0, 7676.0, 10271.0,...
tot_kbps        [0.0, 16578.0, 19164.0, 6307.0, 3838.0, 6400.0...
label                                                     [0, 1]
dtype: object
```

Numerical Features

In [16]:
```
numerical_features = [feature for feature in df.columns if df[feature].dtypes != 'O']
print("The number of numerical features is",len(numerical_features),"and they are : \n",numerical_featur
```

```
The number of numerical features is 20 and they are :
 ['dt', 'switch', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'flows', 'packetins', 'pktperf
low', 'byteperflow', 'pktrate', 'Pairflow', 'port_no', 'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 't
ot_kbps', 'label']
```

Categorical Features

In [17]:
```
categorical_features = [feature for feature in df.columns if df[feature].dtypes == 'O']
print("The number of categorical features is",len(categorical_features),"and they are : \n",categorical_
```

```
The number of categorical features is 3 and they are :
 ['src', 'dst', 'Protocol']
```

Number of Unique values in the numerical features

In [18]:
```
# number of unique values in each numerical variable
df[numerical_features].nunique(axis=0)
```

Out[18]:
```
dt              858
switch           10
pktcount       9044
bytecount      9270
dur             840
dur_nsec       1000
tot_dur        4183
flows            15
packetins       168
pktperflow     2092
byteperflow    2793
pktrate         446
Pairflow          2
port_no           5
tx_bytes      12257
rx_bytes      11623
tx_kbps        1800
rx_kbps        1730
tot_kbps       2259
label             2
dtype: int64
```

Discrete numerical features

In [19]:
```
#discrete numerical features
discrete_feature = [feature for feature in numerical_features if df[feature].nunique()<=15 and feature !
print("The number of discrete features is",len(discrete_feature),"and they are : \n",discrete_feature)
```

```
The number of discrete features is 4 and they are :
 ['switch', 'flows', 'Pairflow', 'port_no']
```

Continuous features

In [21]:
```
continuous_feature=[feature for feature in numerical_features if feature not in discrete_feature + ['lab
print("The number of continuous_feature features is",len(continuous_feature),"and they are : \n",continu
```

```
The number of continuous_feature features is 15 and they are :
 ['dt', 'pktcount', 'bytecount', 'dur', 'dur_nsec', 'tot_dur', 'packetins', 'pktperflow', 'byteperflow'
, 'pktrate', 'tx_bytes', 'rx_bytes', 'tx_kbps', 'rx_kbps', 'tot_kbps']
```

Exploratory Data Analysis

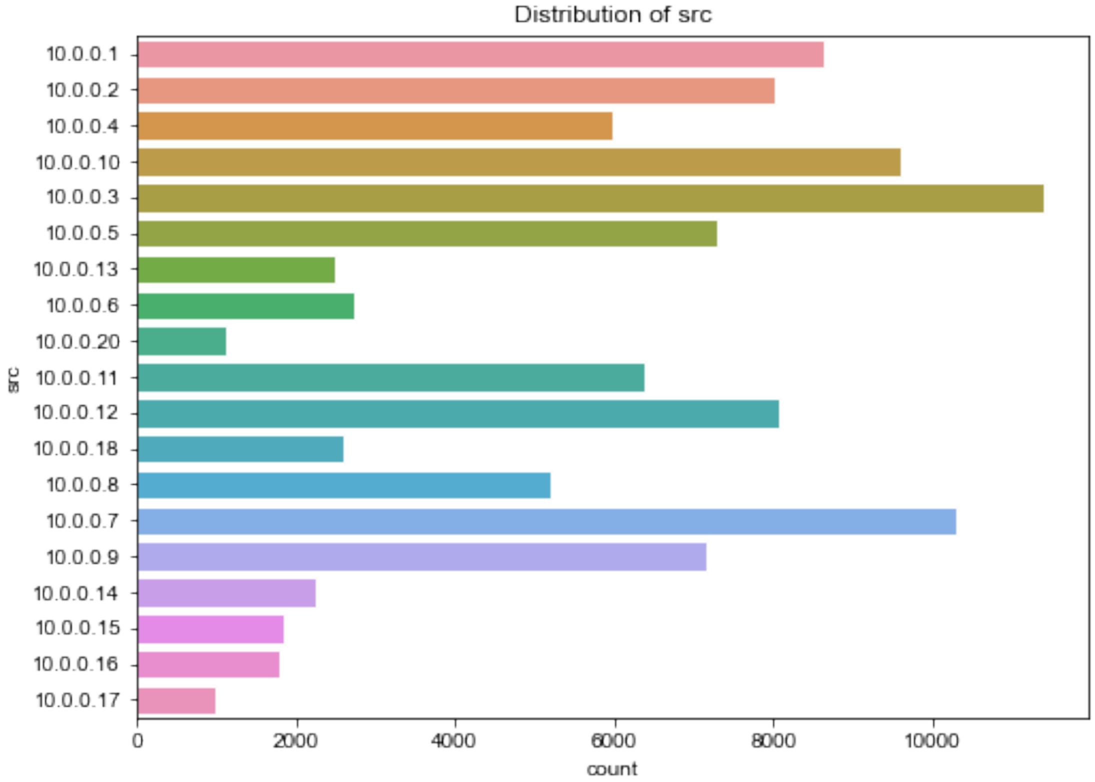
Plotting function definition

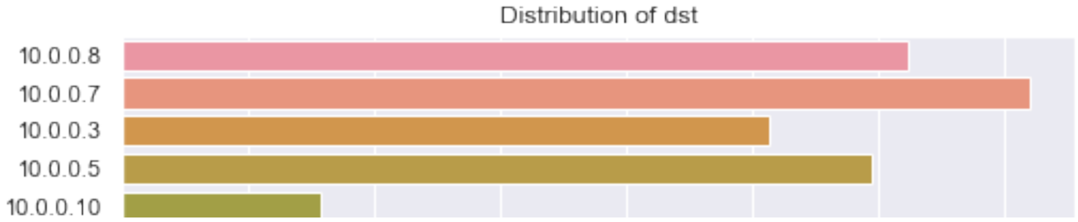
In [22]:
```
def countplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.countplot(y=col, data=df).set(title = 'Distribution of ' + col)

def histplot_distribution(col):
    sns.set_theme(style="darkgrid")
    sns.histplot(data=df,x=col, kde=True,color="red").set(title = 'Distribution of ' + col)
```

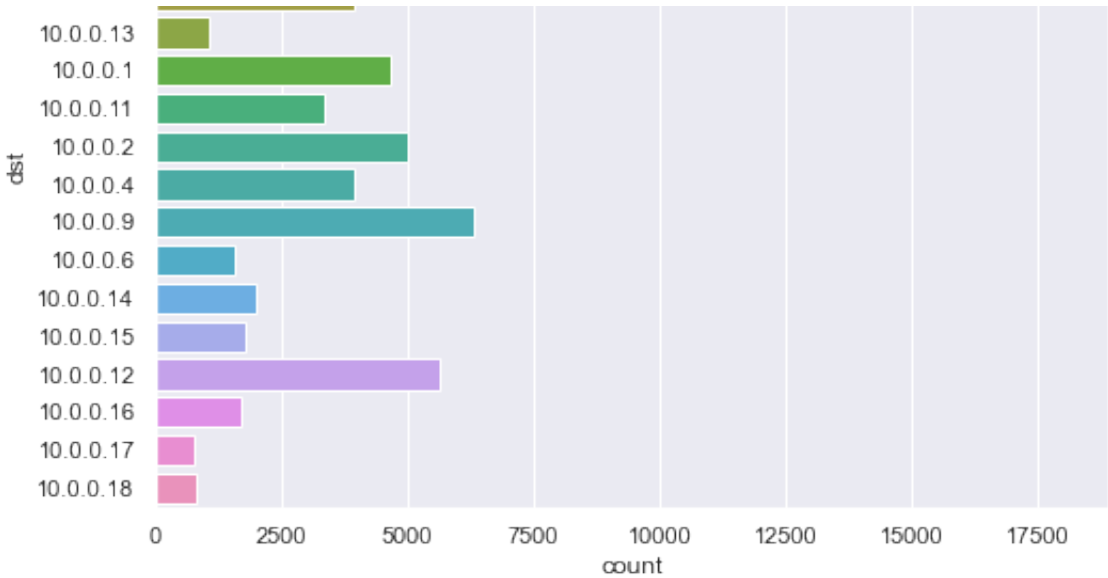
Visualize the distribution of Categorical features

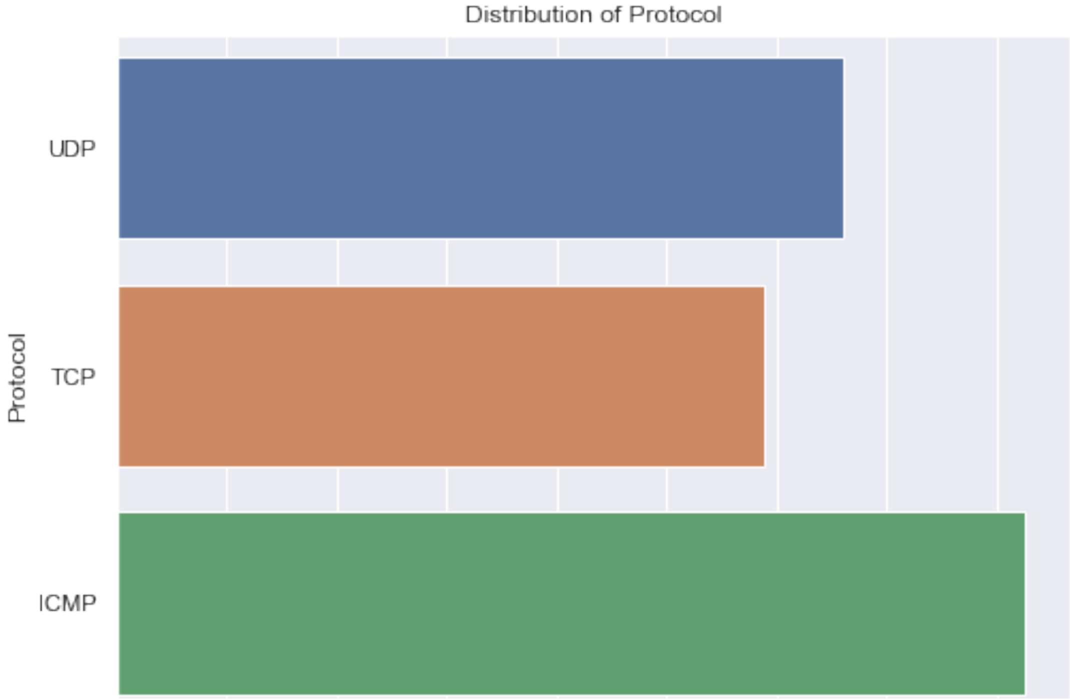
In [23]:

```
## Lets analyse the categorical values by creating histograms to understand the distribution
f = plt.figure(figsize=(8,20))
for i in range(len(categorical_features)):
    f.add_subplot(len(categorical_features), 1, i+1)
    countplot_distribution(categorical_features[i])
plt.show()
```



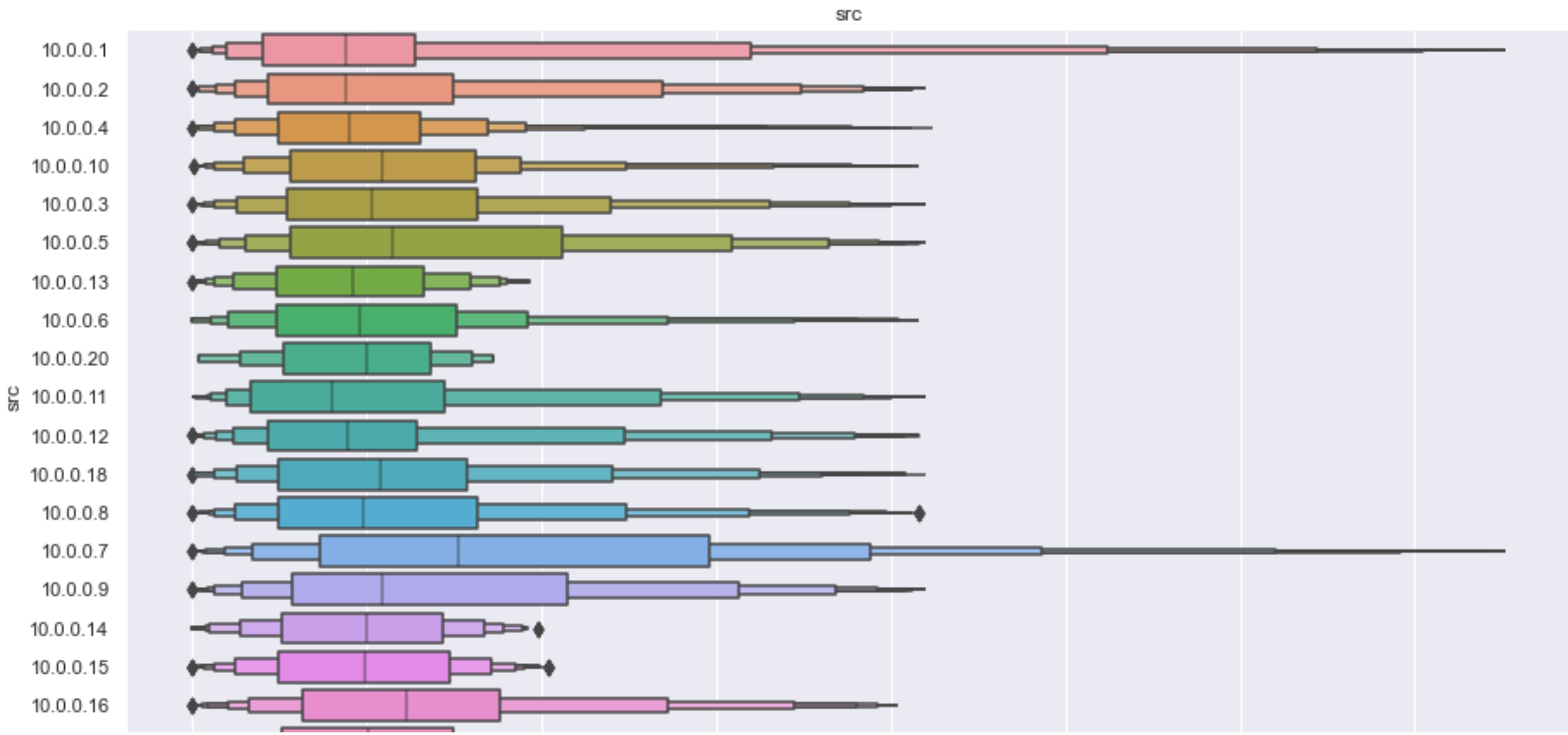



10.0.0.13
10.0.0.1
10.0.0.11
10.0.0.2
10.0.0.4
10.0.0.9
10.0.0.6
10.0.0.14
10.0.0.15
10.0.0.12
10.0.0.16
10.0.0.17
10.0.0.18
dst
0
2500
5000
7500
10000
12500
15000
17500
count

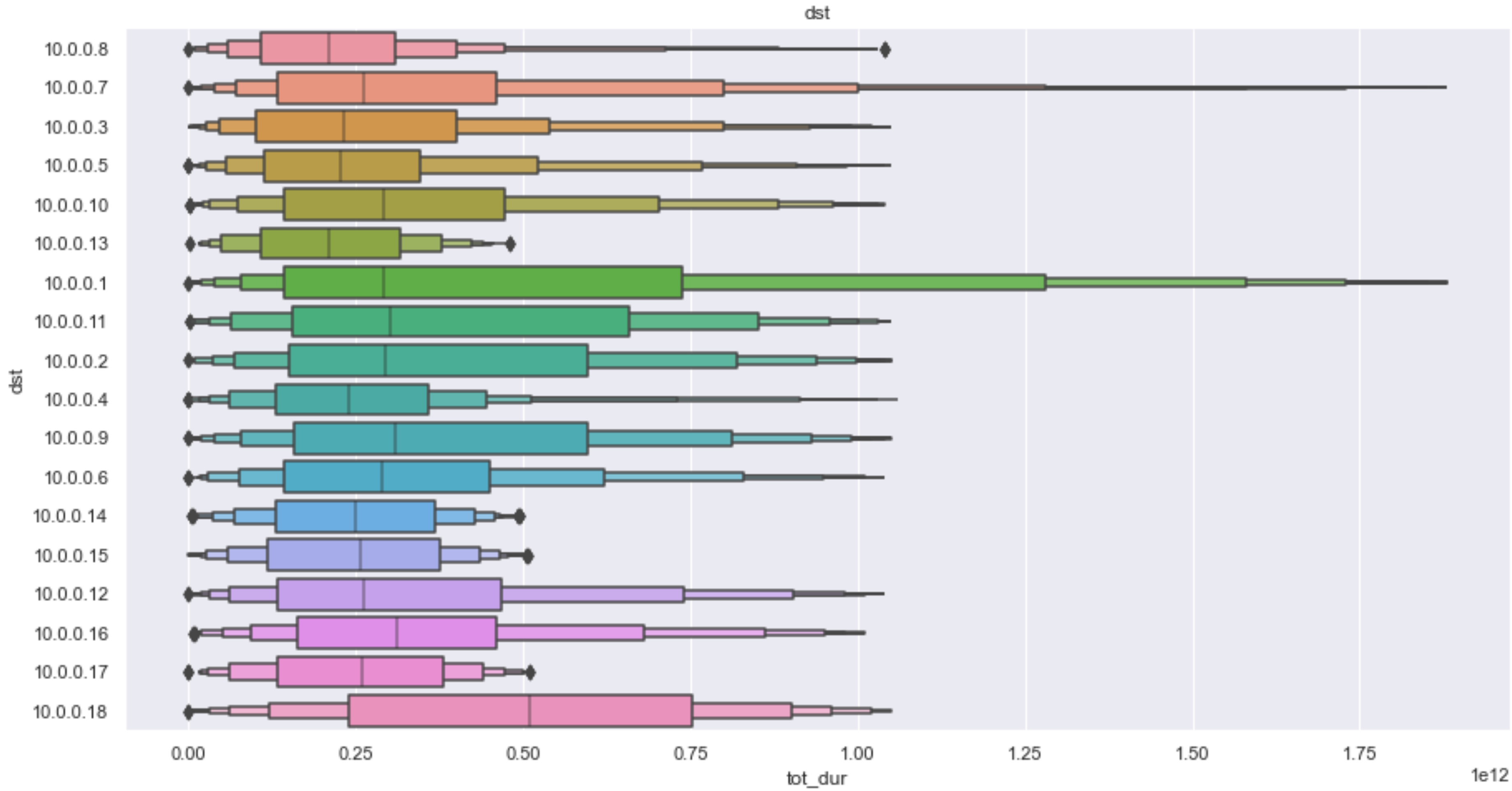

Distribution of Protocol
UDP
TCP
ICMP
Protocol

Visualize the quartiles of categorical features wrt total duration

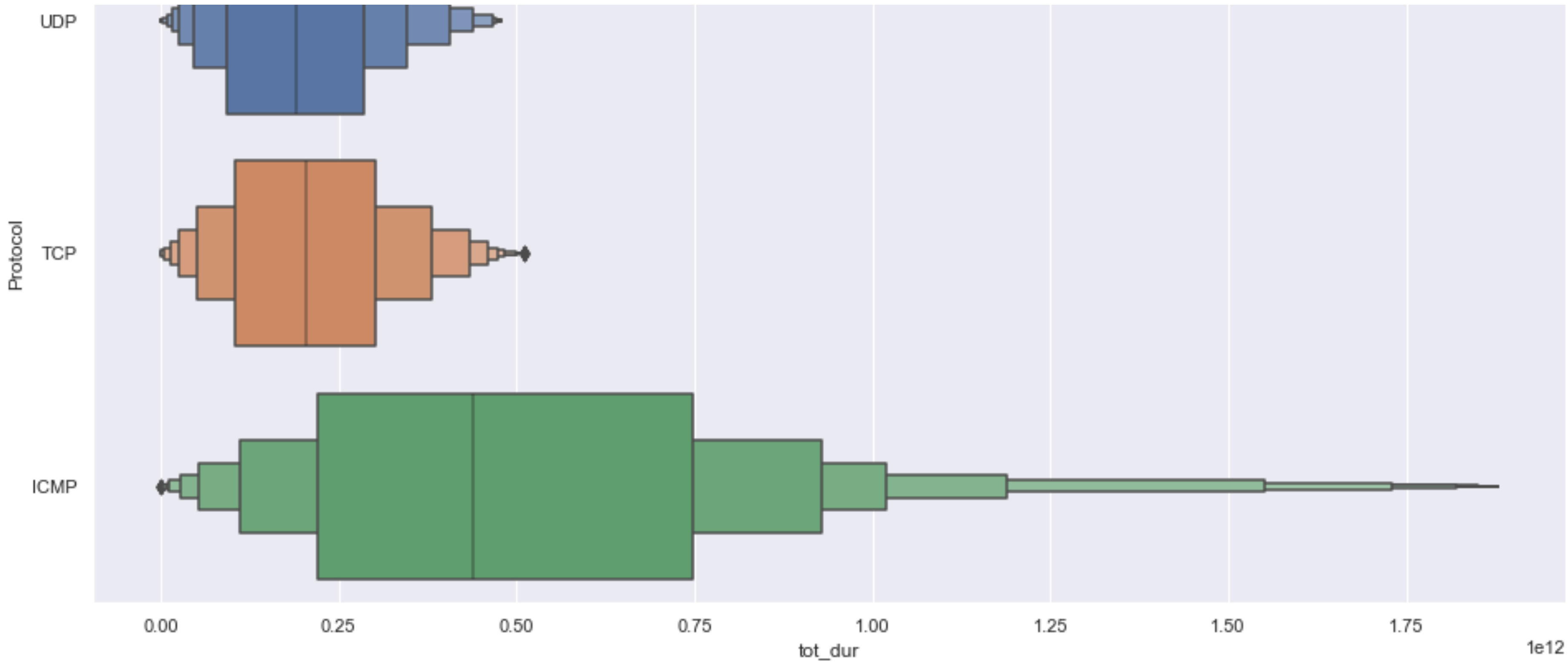
In [24]:
```python
for i in range(len(categorical_features)):
    g = sns.catplot(data=df,x="tot_dur",y=categorical_features[i],kind="boxen").set(title = categorical_
    g.fig.set_figheight(7)
    g.fig.set_figwidth(15)
```




10.0.0.17
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
tot_dur
1e12

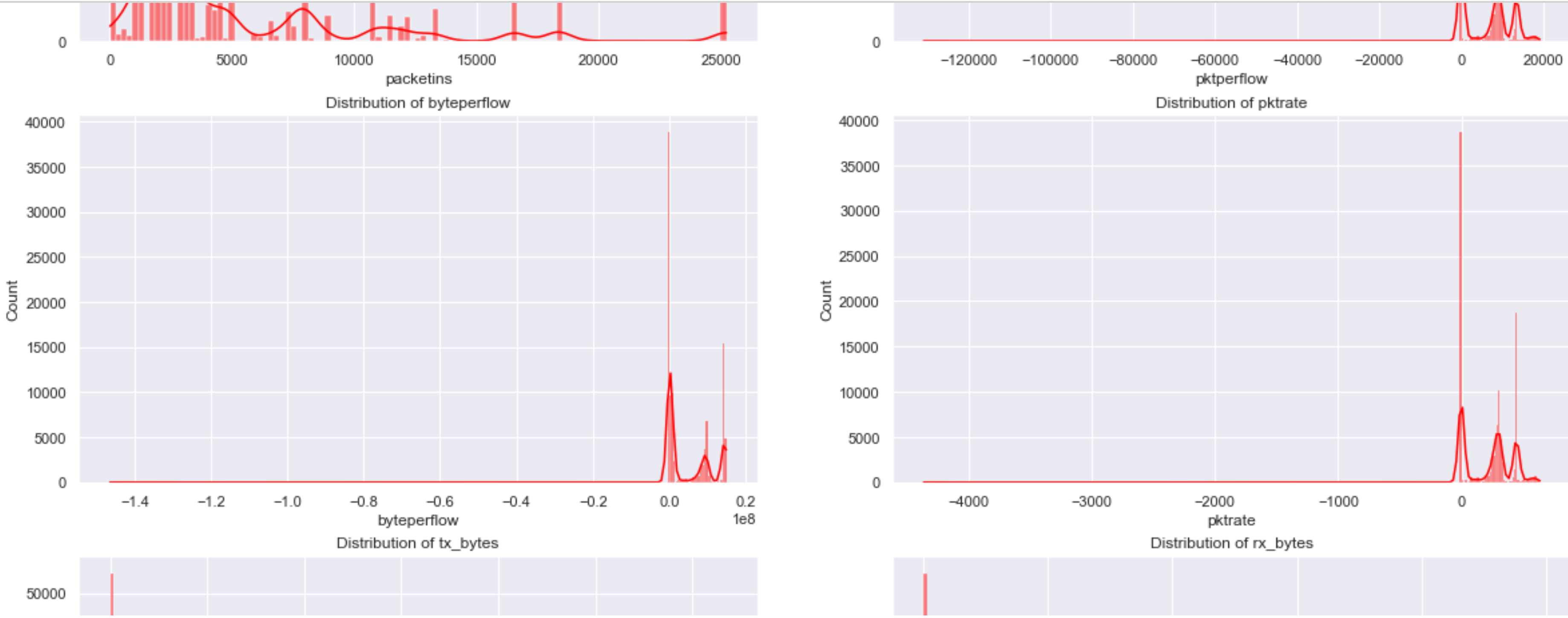

dst
10.0.0.8
10.0.0.7
10.0.0.3
10.0.0.5
10.0.0.10
10.0.0.13
10.0.0.1
10.0.0.11
10.0.0.2
10.0.0.4
10.0.0.9
10.0.0.6
10.0.0.14
10.0.0.15
10.0.0.12
10.0.0.16
10.0.0.17
10.0.0.18
dst
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
tot_dur
1e12


Protocol



Visualize the distribution of continuous features

In [25]:
```python
## Lets analyse the continuous values by creating histograms to understand the distribution
f = plt.figure(figsize=(20,90))
for i in range(len(continuous_feature)):
    f.add_subplot(len(continuous_feature), 2, i+1)
    histplot_distribution(continuous_feature[i])
plt.show()
```

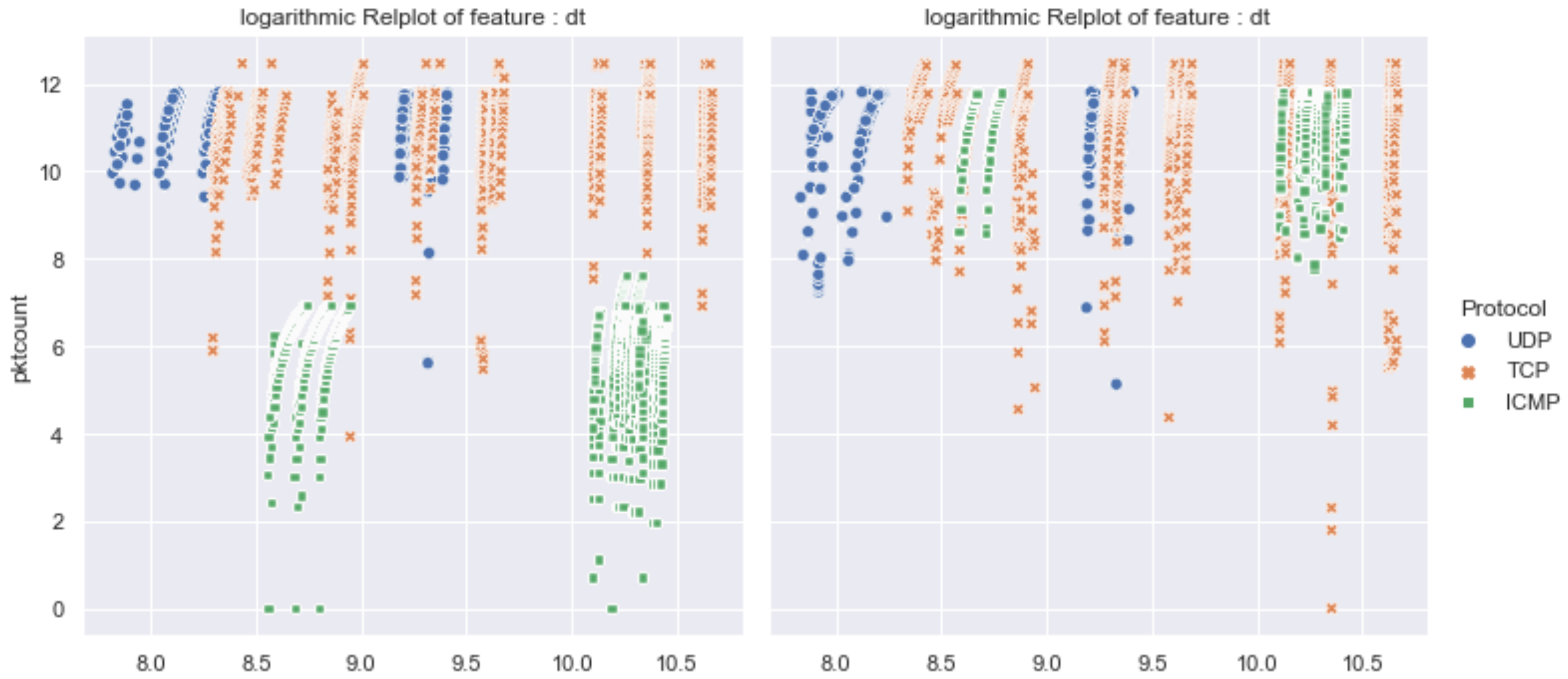


Visualize the distribution of continuous features wrt packet count, protocol and type of attack

In [26]:

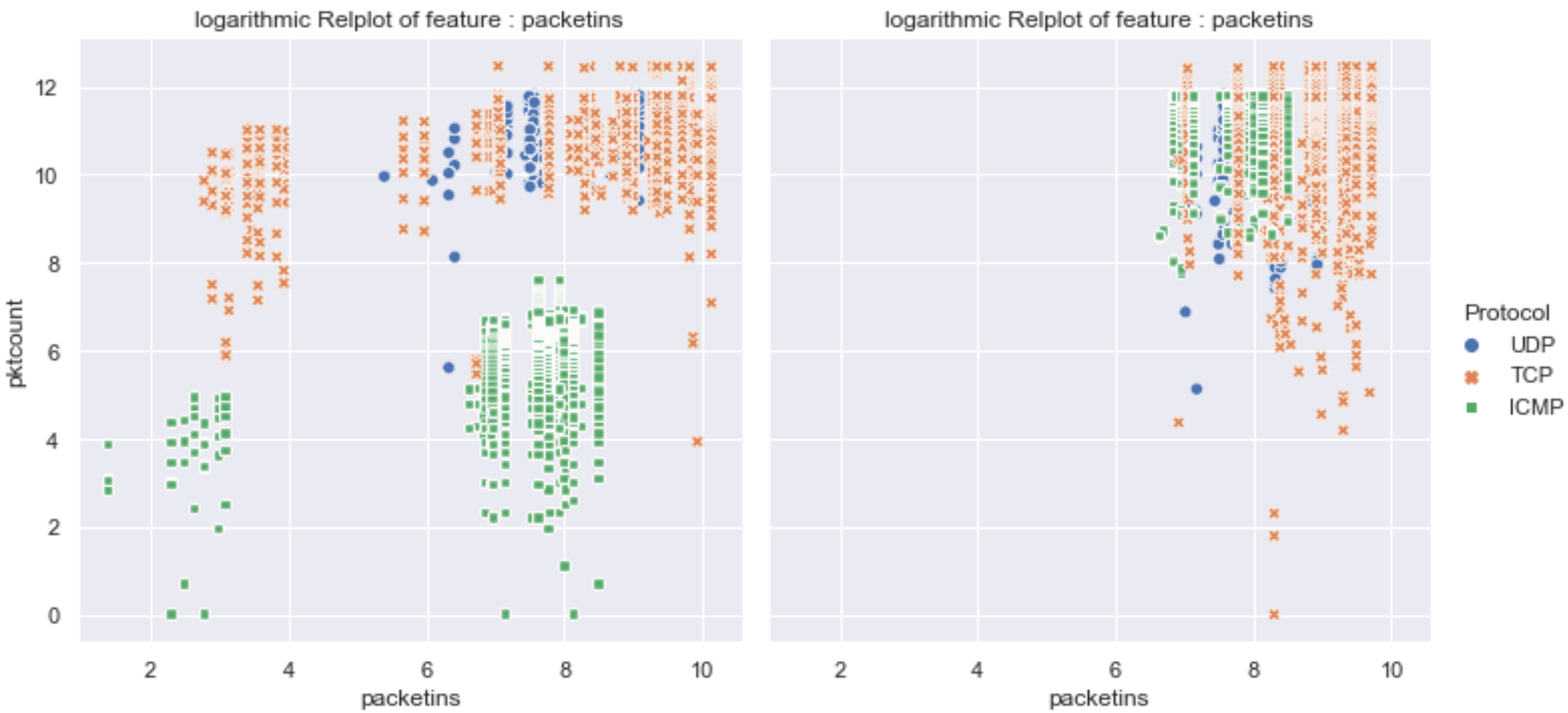
```
## Relplot of log(variable)
import warnings
warnings.filterwarnings("ignore")
for feature in continuous_feature:
    data=df.copy()
    if 0 in data[feature].unique():
        pass
    else:
        data[feature]=np.log(data[feature])
        data['pktcount']=np.log(data['pktcount'])
        plt.figure(figsize=(20,20))
        sns.relplot(data=data, x=data[feature],y=data['pktcount'],hue="Protocol",style="Protocol",
                    col="label",kind="scatter").set(title="logarithmic Relplot of feature : " + feature)
```

```
<Figure size 1440x1440 with 0 Axes>
```

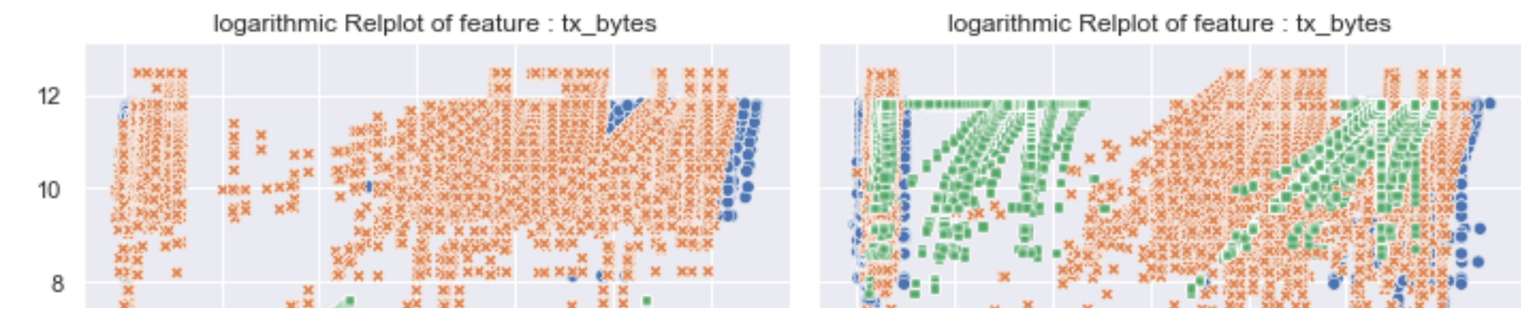


dt dt

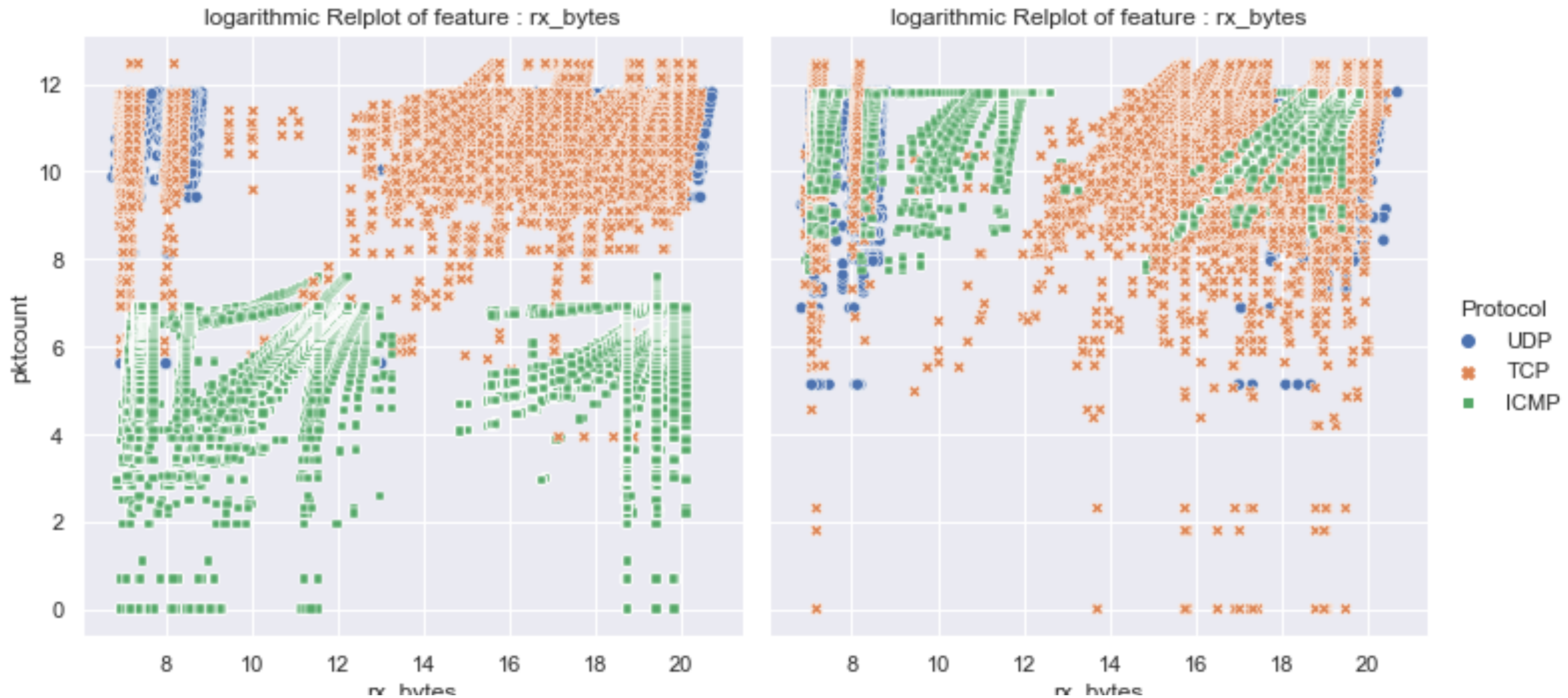
```
<Figure size 1440x1440 with 0 Axes>
```




```
<Figure size 1440x1440 with 0 Axes>
```

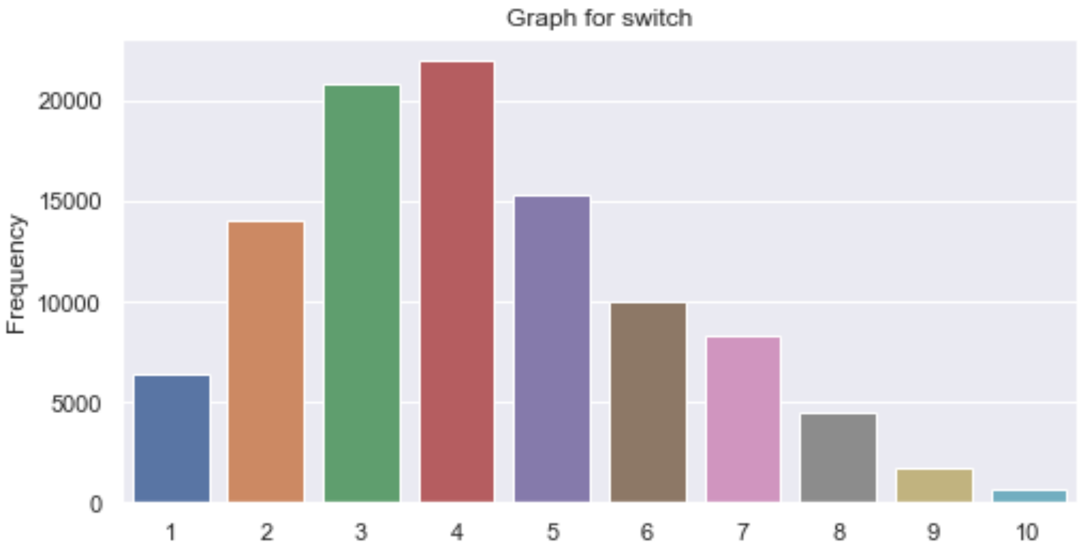


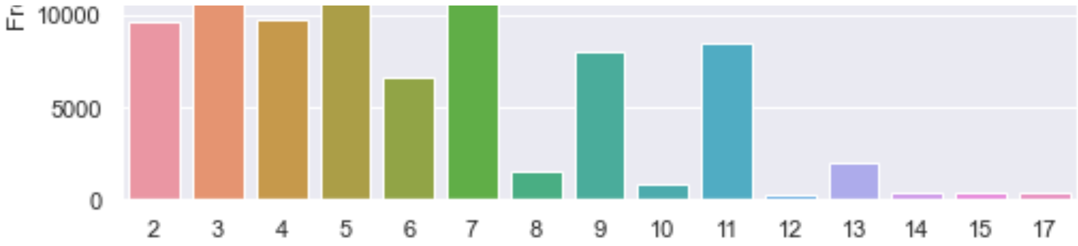

```
<Figure size 1440x1440 with 0 Axes>
```

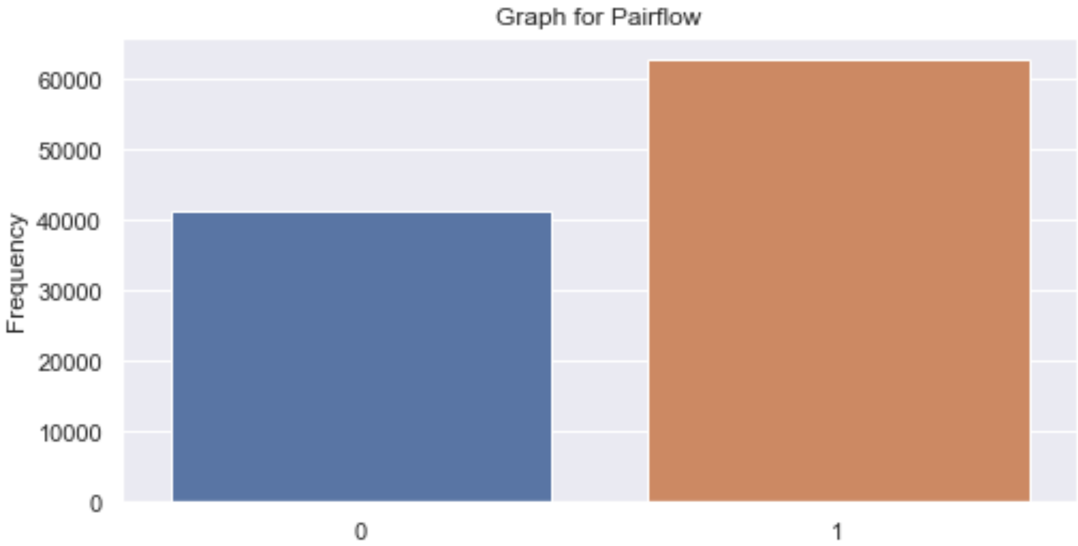


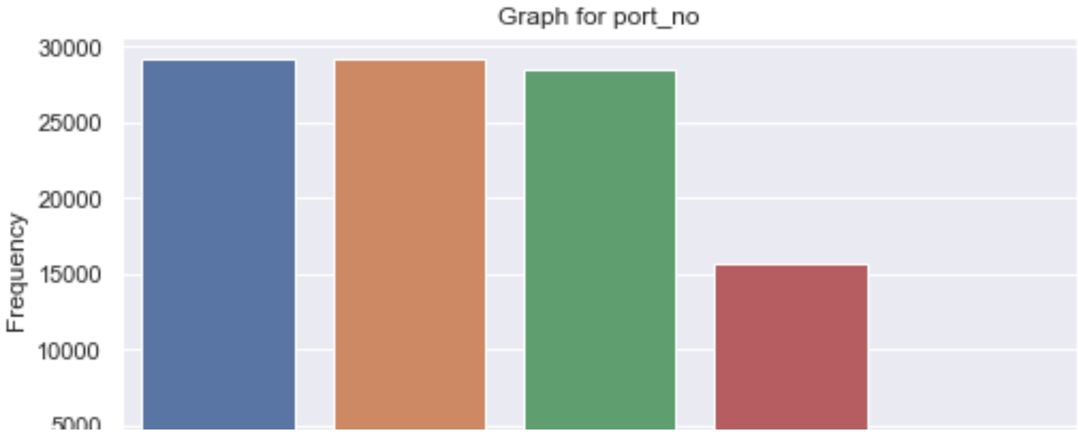
Visualize the distribution of numerical discrete features

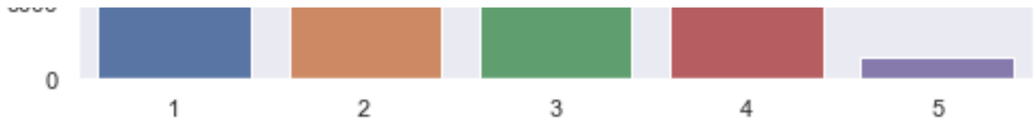
In [27]:
```
for feature in discrete_feature:
    plt.figure(figsize=(8,4))
    cat_num = df[feature].value_counts()
    sns.barplot(x=cat_num.index, y = cat_num).set(title = "Graph for "+feature, ylabel="Frequency")
    plt.show()
```




Fr
10000
5000
0
2
3
4
5
6
7
8
9
10
11
12
13
14
15
17


Graph for Pairflow
Frequency
60000
50000
40000
30000
20000
10000
0
0
1


Graph for port_no
Frequency
30000
25000
20000
15000
10000
5000

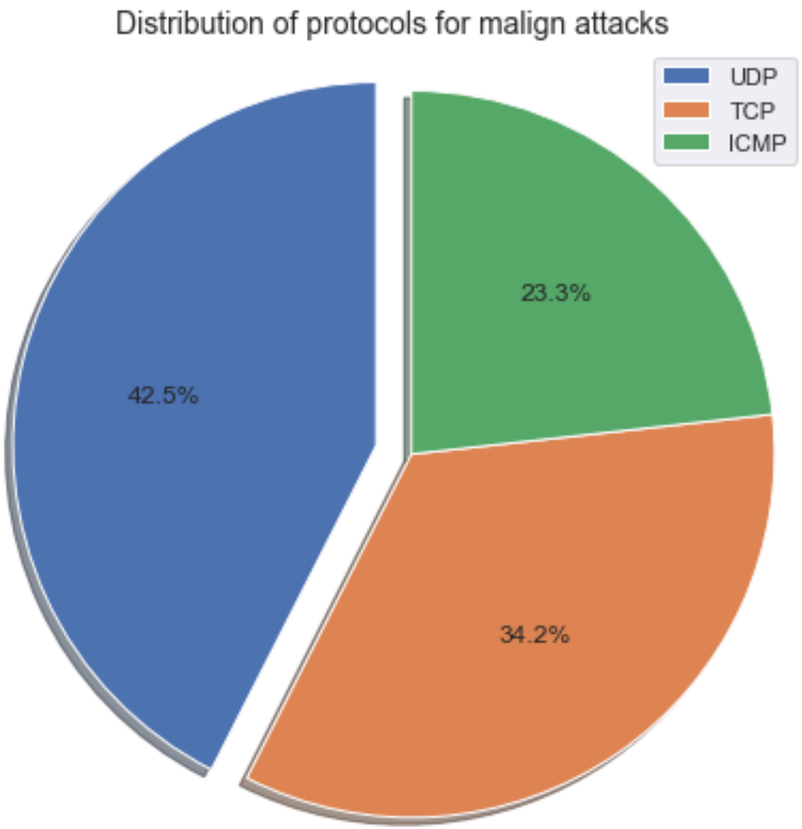


In [28]:

```
def get_percentage_malign_protocols():
    arr = [x for x, y in zip(df['Protocol'], df['label']) if y == 1]
    perc_arr = []
    for i in ['UDP','TCP','ICMP']:
        perc_arr.append(arr.count(i)/len(arr) *100)
    return perc_arr
```

Distribution of protocols for malign attacks

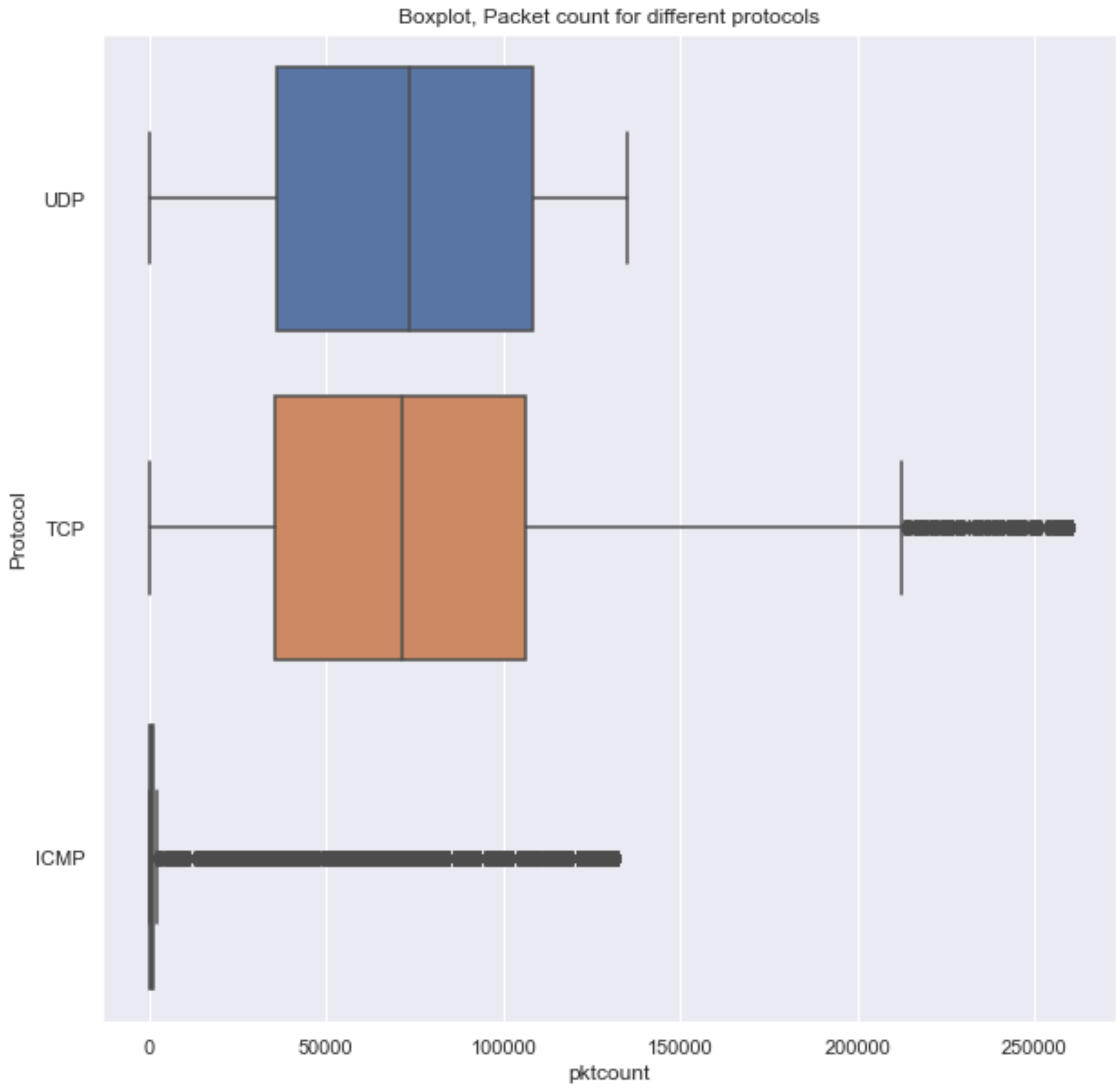
In [29]:
```
fig1, ax1 = plt.subplots(figsize=[7,7])
ax1.pie(get_percentage_malign_protocols(), explode=(0.1, 0, 0), autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
ax1.legend(['UDP', 'TCP', 'ICMP'],loc="best")
plt.title('Distribution of protocols for malign attacks',fontsize = 14)
plt.show()
```



Checking for outliers in Packet count feature

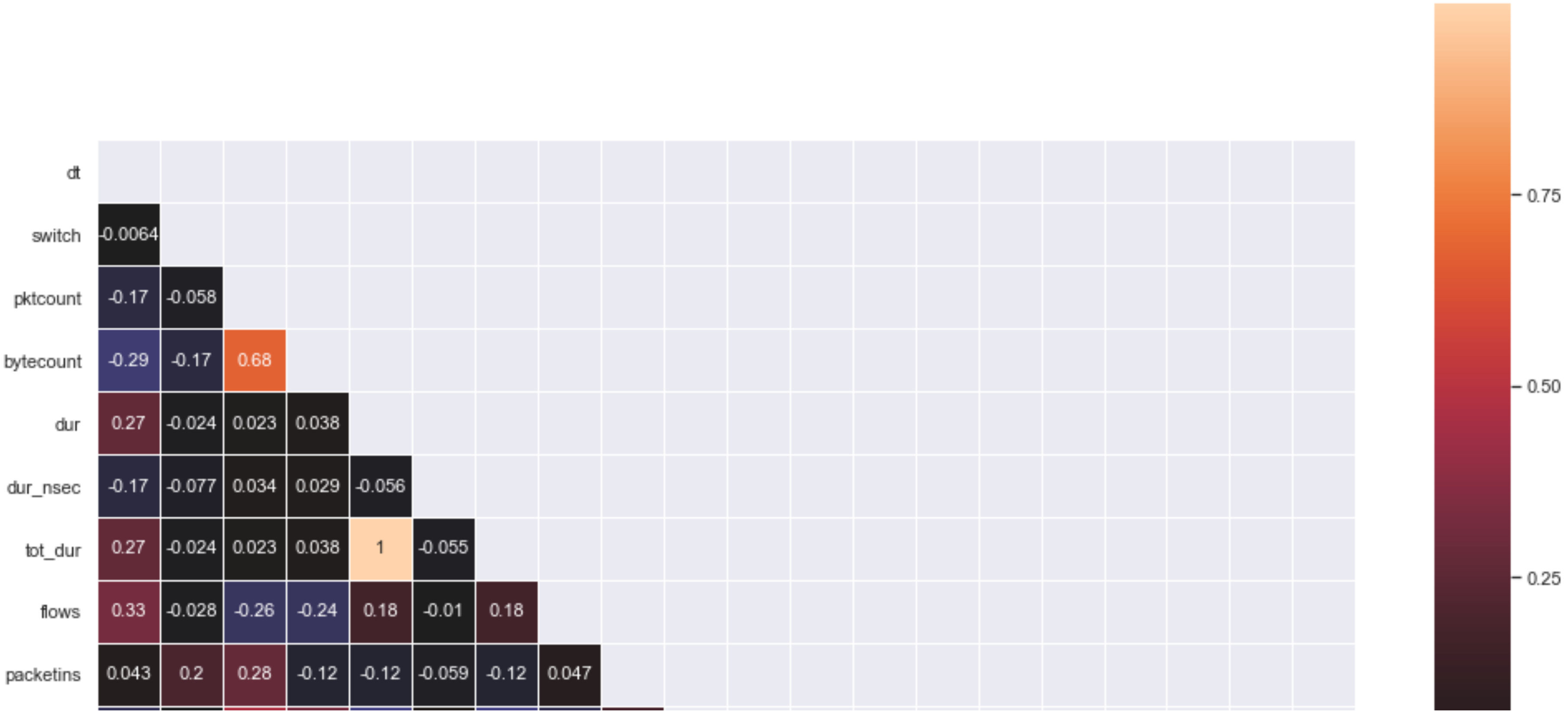
In [30]:
```
fig, ax = plt.subplots(figsize=[10, 10])
sns.boxplot(
    data=df,
    x='pktcount',
    y='Protocol'
)
ax.set_title('Boxplot, Packet count for different protocols')
```

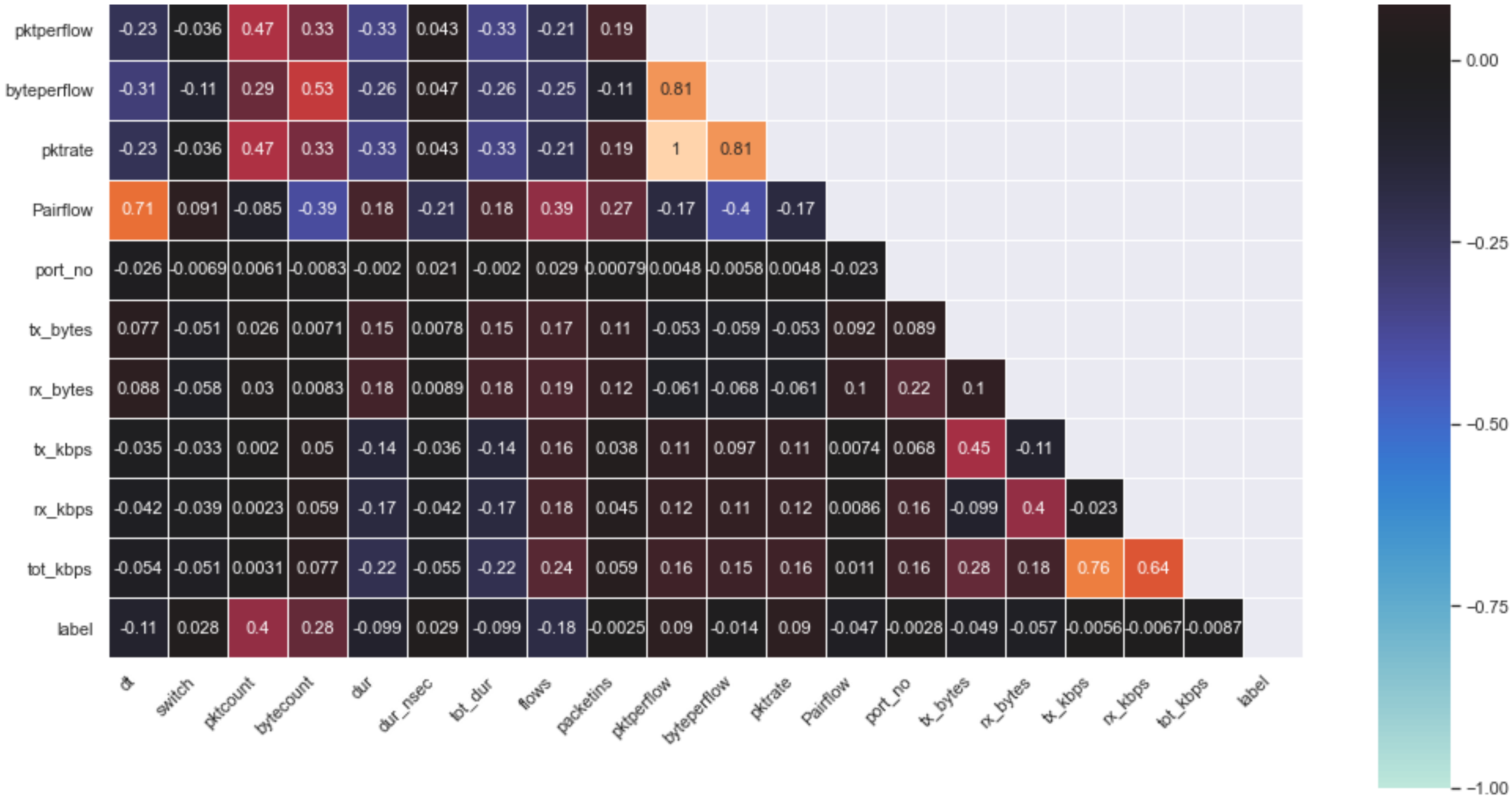
Out[30]:
```
Text(0.5, 1.0, 'Boxplot, Packet count for different protocols')
```


Boxplot, Packet count for different protocols
Protocol
UDP
TCP
ICMP
0
50000
100000
150000
200000
250000
pktcount

Heat map of correlation of features

In [31]:
```
correlation_matrix = df.corr()
fig = plt.figure(figsize=(17,17))
mask = np.zeros_like(correlation_matrix, dtype=np.bool)
mask[np.triu_indices_from(mask)]= True
sns.set_theme(style="darkgrid")
ax = sns.heatmap(correlation_matrix,square = True,annot=True,center=0,vmin=-1,linewidths = .5,annot_kws
ax.set_xticklabels(ax.get_xticklabels(),rotation=45, horizontalalignment='right');
plt.show()
```





In [32]:
```
print("Features which need to be encoded are : \n" ,categorical_features)
```

```
Features which need to be encoded are :
 ['src', 'dst', 'Protocol']
```

Encoding categorical features

In [33]:
```
df = pd.get_dummies(df, columns=categorical_features,drop_first=True)
print("This Dataframe has {} rows and {} columns after encoding".format(df.shape[0], df.shape[1]))
```

```
This Dataframe has 103839 rows and 57 columns after encoding
```

In [34]:
```
#dataframe after encoding
df.head(10)
```

Out[34]:

	dt	switch	pktcount	bytecount	dur	dur_nsec	tot_dur	flows	packetins	pktperflow	...	dst_10.0.0.2	dst_10.0.0.3	dst_10.0.0.4
0	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
1	11605	1	126395	134737070	280	734000000	2.810000e+11	2	1943	13531	...	0	0	
2	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
3	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
4	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
5	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	
6	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
7	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
8	11425	1	45304	48294064	100	716000000	1.010000e+11	3	1943	13535	...	0	0	
9	11425	1	90333	96294978	200	744000000	2.010000e+11	3	1943	13534	...	0	0	

10 rows × 57 columns

In [35]:

```
df.dtypes
```

Out[35]:
```
dt                   int64
switch               int64
pktcount             int64
bytecount            int64
dur                  int64
dur_nsec             int64
tot_dur            float64
flows                int64
packetins            int64
pktperflow           int64
byteperflow          int64
pktrate              int64
Pairflow             int64
port_no              int64
tx_bytes             int64
rx_bytes             int64
tx_kbps              int64
rx_kbps            float64
tot_kbps           float64
label                int64
src_10.0.0.10        uint8
src_10.0.0.11        uint8
src_10.0.0.12        uint8
src_10.0.0.13        uint8
src_10.0.0.14        uint8
src_10.0.0.15        uint8
src_10.0.0.16        uint8
src_10.0.0.17        uint8
src_10.0.0.18        uint8
src_10.0.0.2         uint8
src_10.0.0.20        uint8
src_10.0.0.3         uint8
src_10.0.0.4         uint8
src_10.0.0.5         uint8
```

```
src_10.0.0.5        uint8
src_10.0.0.6        uint8
src_10.0.0.7        uint8
src_10.0.0.8        uint8
src_10.0.0.9        uint8

dst_10.0.0.10       uint8
dst_10.0.0.11       uint8
dst_10.0.0.12       uint8
dst_10.0.0.13       uint8
dst_10.0.0.14       uint8
dst_10.0.0.15       uint8
dst_10.0.0.16       uint8
dst_10.0.0.17       uint8
dst_10.0.0.18       uint8
dst_10.0.0.2        uint8
dst_10.0.0.3        uint8
dst_10.0.0.4        uint8
dst_10.0.0.5        uint8
dst_10.0.0.6        uint8
dst_10.0.0.7        uint8
dst_10.0.0.8        uint8
dst_10.0.0.9        uint8
Protocol_TCP        uint8
Protocol_UDP        uint8
dtype: object
```

Split into Independent and dependent variables

In [36]:
```
#separating input and output attributes
x = df.drop(['label'], axis=1)
y = df['label']
```

Normalizing features

In [37]:
```
ms = MinMaxScaler()
x = ms.fit_transform(x)
```

Train-Test-Split [75-25]

In [38]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y,test_size=0.3)
print(X_train.shape, X_test.shape)
```

```
(72687, 56) (31152, 56)
```

BASELINE CLASSIFIERS

1. DNN
2. KNN
3. SVM
4. Decision tree
5. Naive Bayes
6. Quadratic Discriminant Analaysis
7. SGD
8. Logistic Regression
9. XGBoost

Deep Neural Network

In [39]:
```
Classifier_accuracy = []
```

Defining the Deep Neural Network

In [40]:
```
# Define and compile model
model = keras.Sequential()
model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
opt = keras.optimizers.Adam(learning_rate=0.01)
model.compile( optimizer=opt, loss="binary_crossentropy", metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Hidden_Layer_1 (Dense)       (None, 28)                1596
_________________________________________________________________
Hidden_Layer_2 (Dense)       (None, 10)                290
_________________________________________________________________
Output_Layer (Dense)         (None, 1)                 11
=================================================================
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
_________________________________________________________________
```

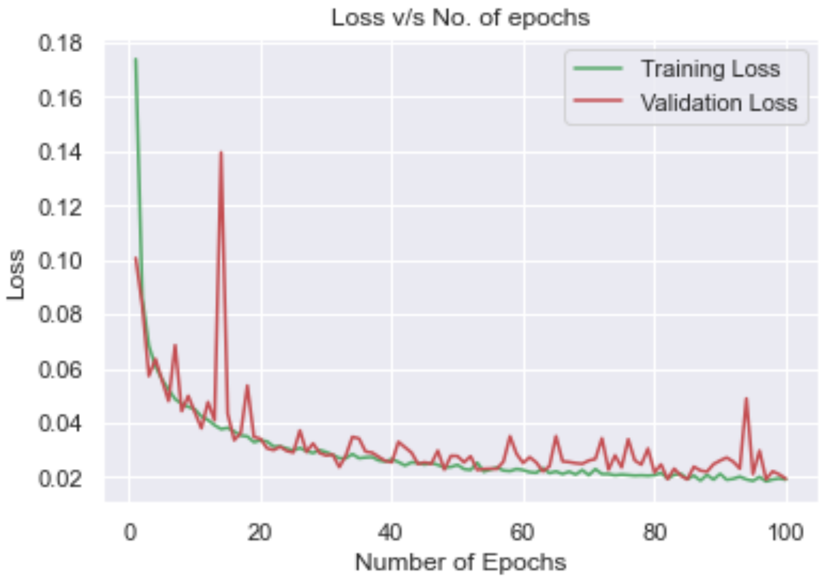
Model fitting

In [41]:
```
# fit model
history_org = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=100, verbose=2,
    callbacks=None,
    validation_data=(X_test,y_test),
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0)
```

```
Epoch 18/100
2272/2272 - 2s - loss: 0.0350 - accuracy: 0.9856 - val_loss: 0.0537 - val_accuracy: 0.9772
Epoch 19/100
2272/2272 - 2s - loss: 0.0329 - accuracy: 0.9870 - val_loss: 0.0348 - val_accuracy: 0.9861
Epoch 20/100
2272/2272 - 2s - loss: 0.0336 - accuracy: 0.9864 - val_loss: 0.0341 - val_accuracy: 0.9855
Epoch 21/100
2272/2272 - 2s - loss: 0.0331 - accuracy: 0.9865 - val_loss: 0.0306 - val_accuracy: 0.9871
Epoch 22/100
2272/2272 - 2s - loss: 0.0313 - accuracy: 0.9869 - val_loss: 0.0300 - val_accuracy: 0.9878
Epoch 23/100
2272/2272 - 2s - loss: 0.0313 - accuracy: 0.9866 - val_loss: 0.0312 - val_accuracy: 0.9867
Epoch 24/100
2272/2272 - 2s - loss: 0.0306 - accuracy: 0.9874 - val_loss: 0.0297 - val_accuracy: 0.9875
Epoch 25/100
2272/2272 - 2s - loss: 0.0299 - accuracy: 0.9875 - val_loss: 0.0291 - val_accuracy: 0.9871
Epoch 26/100
2272/2272 - 2s - loss: 0.0308 - accuracy: 0.9880 - val_loss: 0.0372 - val_accuracy: 0.9842
Epoch 27/100
```

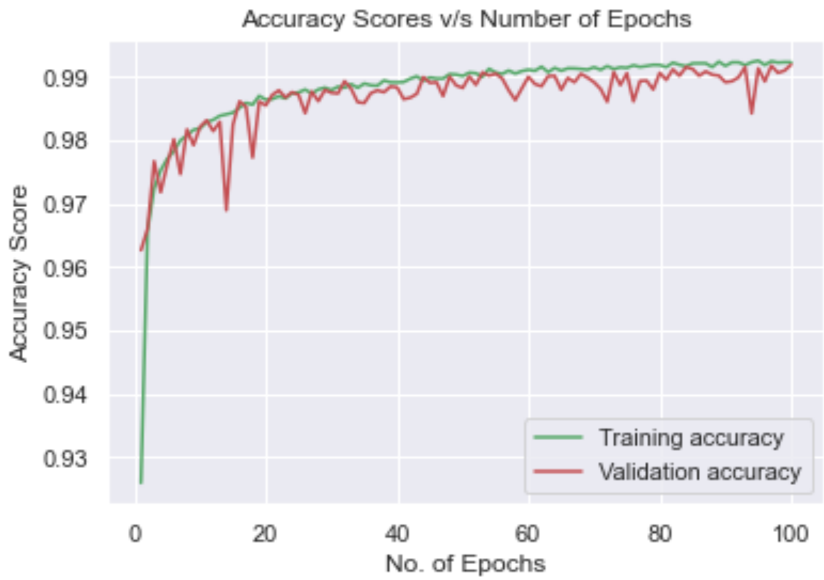
Plotting Loss v/s Epochs

In [42]:
```
loss = history_org.history['loss']
val_loss = history_org.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g', label = 'Training Loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
plt.title('Loss v/s No. of epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Plotting Accuracy v/s Epochs

In [43]:
```
loss = history_org.history['accuracy']
val_loss = history_org.history['val_accuracy']
plt.plot(epochs, loss, 'g', label = 'Training accuracy')
plt.plot(epochs, val_loss, 'r', label = 'Validation accuracy')
plt.title('Accuracy Scores v/s Number of Epochs')
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy Score')
plt.legend()
plt.show()
```



Model Evaluation

In [44]:
```
loss, accuracy = model.evaluate(X_test, y_test)
print('Accuracy of Deep neural Network : %.2f' % (accuracy*100))
Classifier_accuracy.append(accuracy*100)
```

```
974/974 [==============================] - 0s 502us/step - loss: 0.0195 - accuracy: 0.9919
Accuracy of Deep neural Network : 99.19
```

K-Nearest Neighbor Classifier

In [45]:
```
knn_clf = KNeighborsClassifier()
knn_clf.fit(X_train, y_train)
y_pred = knn_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of KNN Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of KNN Classifier : 96.46
```

SVM Classifier

In [46]:
```
svc_clf = SVC()
svc_clf.fit(X_train,y_train)
y_pred = svc_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SVM Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of SVM Classifier : 97.44
```

Decision Tree Classifier

In [47]:
```
dt_clf = DecisionTreeClassifier(max_depth=5)
dt_clf.fit(X_train,y_train)
y_pred = dt_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Decision Tree Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of Decision Tree Classifier : 96.63
```

Naive Bayes Classifier

In [48]:
```
nb_clf = CategoricalNB()
nb_clf.fit(X_train,y_train)
y_pred = nb_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Naive Bayes Classifier : %.2f" % (accuracy*100) )
```

```
Accuracy of Naive Bayes Classifier : 71.32
```

Quadratic Discriminant Analysis Classifier

In [49]:
```
qda_clf=QuadraticDiscriminantAnalysis()
qda_clf.fit(X_train,y_train)
y_pred=qda_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of QDA Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of QDA Classifier : 50.14
```

Stochastic Gradient Classifier

In [50]:
```
sgd_clf=SGDClassifier(loss="hinge", penalty="l2")
sgd_clf.fit(X_train,y_train)
y_pred=sgd_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of SGD Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of SGD Classifier : 83.91
```

Logistic Regression

In [51]:
```
lr_clf = LogisticRegression()
lr_clf.fit(X_train,y_train)
y_pred=lr_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of Logistic Regression Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of Logistic Regression Classifier : 83.69
```

XGBoost Classifier

In [52]:
```
xgb_clf=xgb.XGBClassifier(eval_metric = 'error',objective='binary:logistic',max_depth=2, learning_rate=0
xgb_clf.fit(X_train,y_train)
y_pred=xgb_clf.predict(X_test)
accuracy = metrics.accuracy_score(y_test, y_pred)
Classifier_accuracy.append(accuracy*100)
print("Accuracy of XGBoost Classifier : %.2f" % (accuracy*100))
```

```
Accuracy of XGBoost Classifier : 98.18
```

Comparitive analysis of models

In [53]:
```
assifier_names = ["DNN", "KNN", "RBF_SVM", "Decision Tree","Naive Bayes","Quadratic","SGD","Logistic Regr
```

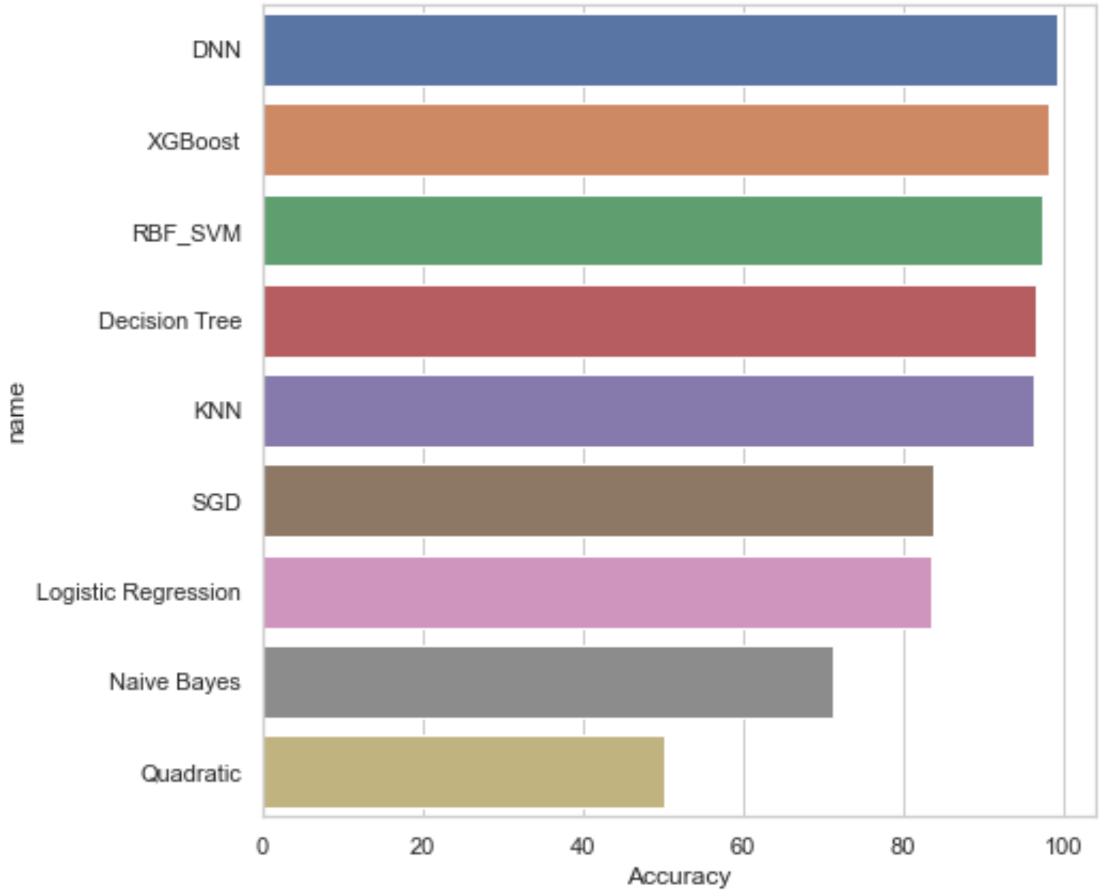
In [54]:
```
df_clf = pd.DataFrame()
df_clf['name'] = Classifier_names
df_clf['Accuracy'] = Classifier_accuracy
df_clf = df_clf.sort_values(by=['Accuracy'], ascending=False)
df_clf.head(10)
```

Out[54]:

	name	Accuracy
0	DNN	99.187851
8	XGBoost	98.179892
2	RBF_SVM	97.444787
3	Decision Tree	96.632640
1	KNN	96.456086
6	SGD	83.911145
7	Logistic Regression	83.689651
4	Naive Bayes	71.318053
5	Quadratic	50.144453

Visualize accuracies of the models

In [55]:
```
sns.set(style="whitegrid",rc={'figure.figsize':(7,7)})
ax = sns.barplot(y="name", x="Accuracy", data=df_clf)
```



In [56]:
```
print(f"The best baseline Classifier is {df_clf.name[0]} with an accuracy of {df_clf.Accuracy[0]}.")
```

```
The best baseline Classifier is DNN with an accuracy of 99.18785095214844.
```

Hyperparameter tuning

In [83]:
```
def model_builder(hp):
    model = keras.Sequential()

    model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
    model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
    model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
    opt = keras.optimizers.Adam(learning_rate=0.01)

    model.compile(optimizer=keras.optimizers.Adam(hp.Choice('learning_rate',[1e-2, 1e-3, 1e-4])), loss='

    return history, model.layers, model
```

In [58]:
```
from keras_tuner.tuners import RandomSearch
tuner = RandomSearch(model_builder, objective='val_accuracy', max_trials=3, executions_per_trial=2, dire
```

In [59]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 1
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
```

In [60]:
```
tuner.search(X_train, y_train, epochs=100, validation_data=(X_test,y_test), batch_size = 32)
```

```
Trial 3 Complete [00h 09m 31s]
val_accuracy: 0.9936600923538208

Best val_accuracy So Far: 0.9936600923538208
Total elapsed time: 00h 29m 23s
INFO:tensorflow:Oracle triggered exit
```

In [61]:
```
tuner.results_summary()
```

```
Results summary
Results in ddos\ddos_isa
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
learning_rate: 0.001
Score: 0.9936600923538208
Trial summary
Hyperparameters:
learning_rate: 0.01
Score: 0.9922637343406677
Trial summary
Hyperparameters:
learning_rate: 0.0001
Score: 0.9879140853881836
```

Best Hyperparameters

In [62]:
```
modified_model = tuner.get_best_models(num_models=1)[0]
modified_hparam=tuner.get_best_hyperparameters(num_trials=1)[0]
tuner.get_best_hyperparameters()[0].values
```

Out[62]:
```
{'learning_rate': 0.001}
```

Model Evaluation

In [63]:
```
loss, accuracy = modified_model.evaluate(X_test, y_test)
```

```
974/974 [==============================] - 1s 739us/step - loss: 0.0159 - accuracy: 0.9938
```

Get Best value for epoch

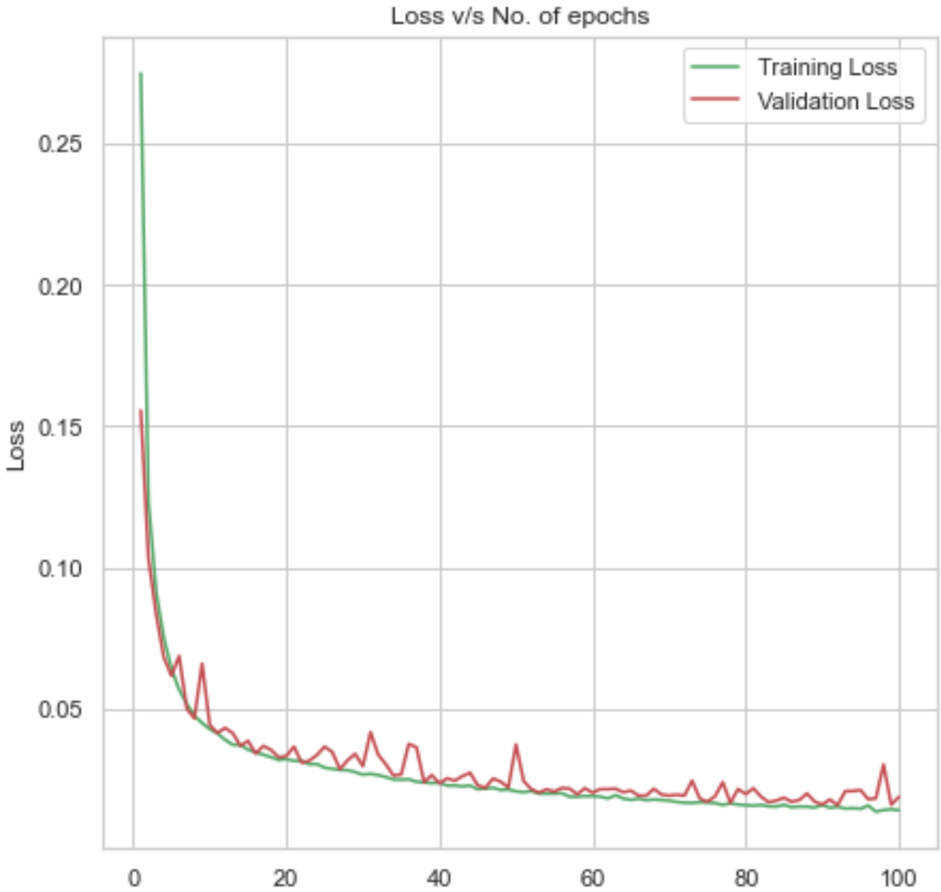
In [64]:
```
model = tuner.hypermodel.build(modified_hparam)
history = model.fit(X_train, y_train, batch_size=32, epochs=100, verbose=1, validation_data=(X_test,y_te
val_per_epoch = history.history['val_accuracy']
modified_epoch = val_per_epoch.index(max(val_per_epoch)) + 1
print('Best epoch value: %d' % (modified_epoch,))
```

```
0.0210 - val_accuracy: 0.9918
Epoch 95/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0147 - accuracy: 0.9939 - val_loss:
0.0213 - val_accuracy: 0.9908
Epoch 96/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0159 - accuracy: 0.9935 - val_loss:
0.0181 - val_accuracy: 0.9922
Epoch 97/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0137 - accuracy: 0.9944 - val_loss:
0.0184 - val_accuracy: 0.9929
Epoch 98/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0143 - accuracy: 0.9942 - val_loss:
0.0304 - val_accuracy: 0.9884
Epoch 99/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0145 - accuracy: 0.9938 - val_loss:
0.0161 - val_accuracy: 0.9928
Epoch 100/100
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0142 - accuracy: 0.9940 - val_loss:
0.0189 - val_accuracy: 0.9920
Best epoch value: 78
```

Plot of Loss v/s Epochs for hypermodel

In [65]:

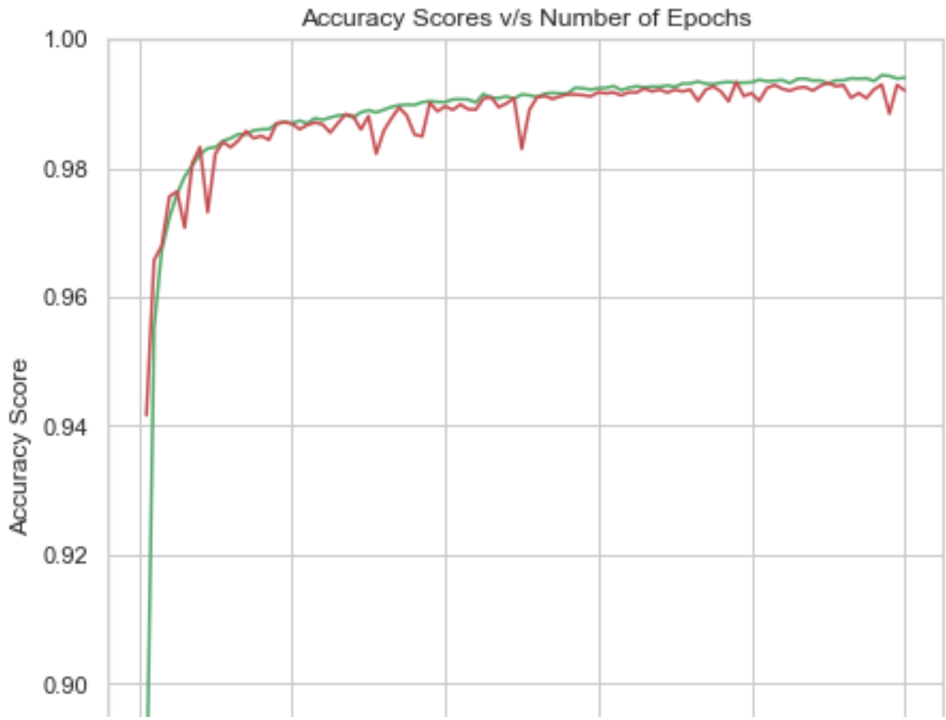
```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g', label = 'Training Loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
plt.title('Loss v/s No. of epochs')
plt.xlabel('Number of Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

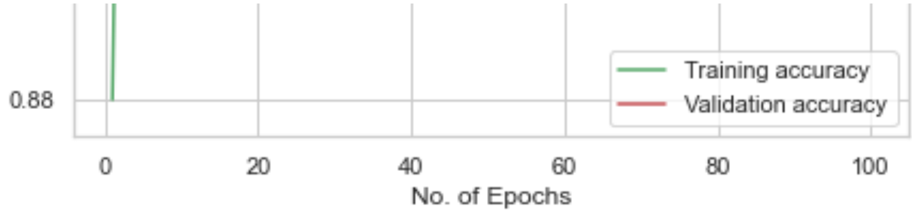


Number of Epochs

Plot for Accuracy v/s Epochs for hypermodel

In [66]:
```
loss = history.history['accuracy']
val_loss = history.history['val_accuracy']
plt.plot(epochs, loss, 'g', label = 'Training accuracy')
plt.plot(epochs, val_loss, 'r', label = 'Validation accuracy')
plt.title('Accuracy Scores v/s Number of Epochs')
plt.xlabel('No. of Epochs')
plt.ylabel('Accuracy Score')
plt.legend()
plt.show()
```





Final Model

In [67]:
```
hypermodel = tuner.hypermodel.build(modified_hparam)
```

Fitting the hypermodel

In [68]:
```
hypermodel.fit(X_train, y_train, batch_size=32, epochs=modified_epoch, validation_data=(X_test, y_test),
```

```
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0179 - accuracy: 0.9924 - val_loss:
0.0236 - val_accuracy: 0.9905
Epoch 71/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0179 - accuracy: 0.9923 - val_loss:
0.0223 - val_accuracy: 0.9910
Epoch 72/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0194 - accuracy: 0.9920 - val_loss:
0.0228 - val_accuracy: 0.9896
Epoch 73/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0177 - accuracy: 0.9926 - val_loss:
0.0231 - val_accuracy: 0.9895
Epoch 74/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0182 - accuracy: 0.9925 - val_loss:
0.0212 - val_accuracy: 0.9907
Epoch 75/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0175 - accuracy: 0.9927 - val_loss:
0.0236 - val_accuracy: 0.9903
Epoch 76/78
2272/2272 [==============================] - 3s 1ms/step - loss: 0.0175 - accuracy: 0.9927 - val_loss:
0.0209 - val_accuracy: 0.9913
```

In [69]:
```
hypermodel.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Hidden_Layer_1 (Dense)       (None, 28)                1596      
_________________________________________________________________
Hidden_Layer_2 (Dense)       (None, 10)                290       
_________________________________________________________________
Output_Layer (Dense)         (None, 1)                 11        
=================================================================
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
_________________________________________________________________
```

Printing the final accuracy and loss values of the hypermodel

In [70]:
```
result_final = hypermodel.evaluate(X_test, y_test, batch_size=32)
print("[Loss, Accuracy]:", result_final)
```

```
974/974 [==============================] - 1s 779us/step - loss: 0.0185 - accuracy: 0.9921
[Loss, Accuracy]: [0.018542751669883728, 0.9921353459358215]
```

Making Sample Predictions

In [71]:
```
classes = model.predict(X_test)
print(classes)
```

```
[[9.9998575e-01]
 [2.7547706e-34]
 [9.9986565e-01]
 ...
 [5.6841223e-11]
 [0.0000000e+00]
 [9.9976611e-01]]
```

In [72]:
```
y_pred = []
for i in classes:
    if i > 0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [73]:
```
y_pred[:20]
```

Out[73]:
```
[1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1]
```

In [74]:
```
y_test[:20]
```

Out[74]:
```
38224    1
7165     0
80513    1
38347    1
10413    1
18464    0
67941    0
47189    0
60882    0
59075    0
97322    0
26155    0
17114    0
34611    1
17038    0
81781    0
40626    1
38474    1
54097    0
62925    1
Name: label, dtype: int64
```

Classification Report

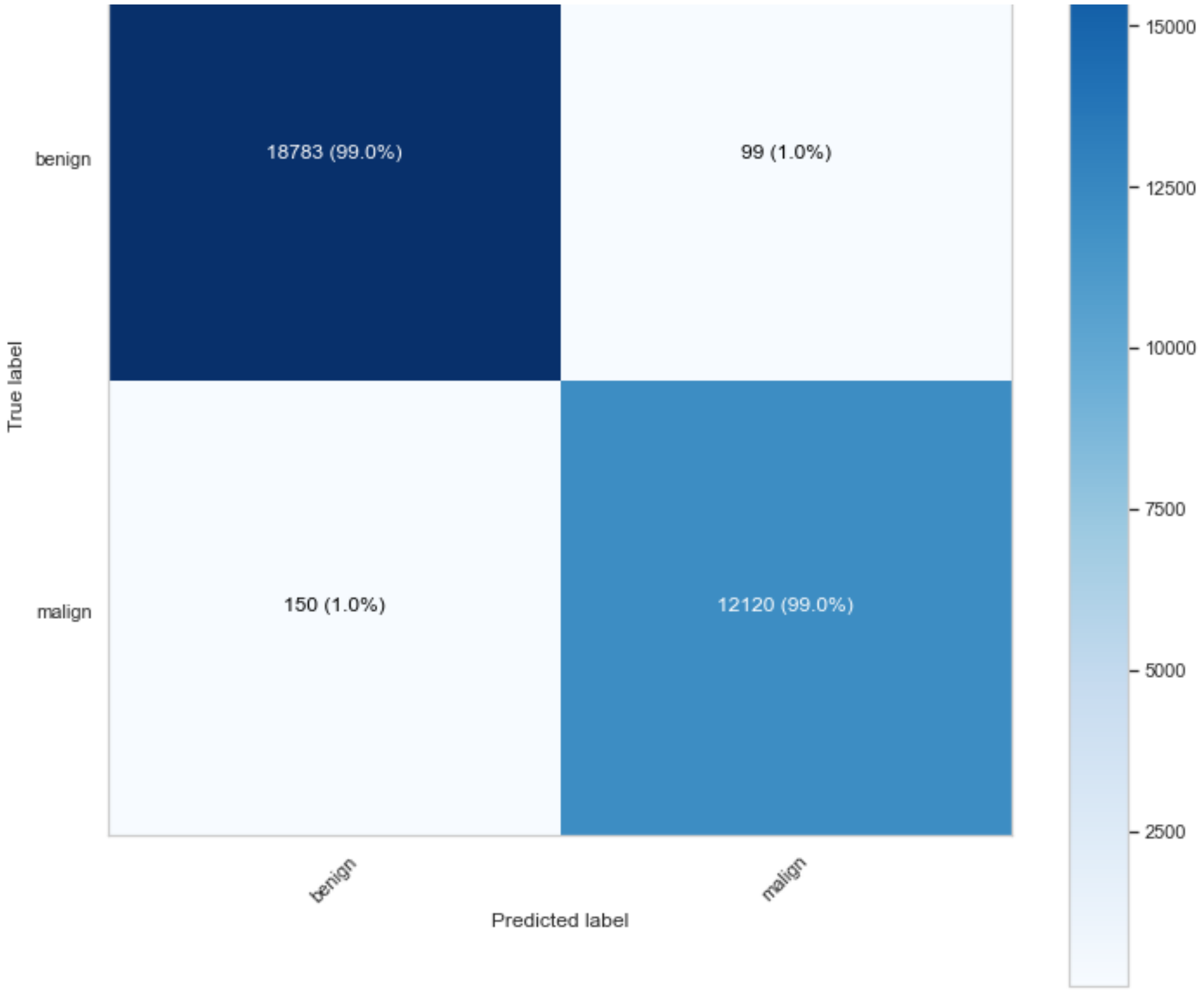
In [93]:
```
print(classification_report(y_test, y_pred, target_names = labels))
```

```
              precision    recall  f1-score   support

      benign       0.99      0.99      0.99     18882
      malign       0.99      0.99      0.99     12270

    accuracy                           0.99     31152
   macro avg       0.99      0.99      0.99     31152
weighted avg       0.99      0.99      0.99     31152
```

Plotting Confusion Matrix

In [125]:
```
from itertools import product
def plot_confusion_matrix(cm, classes, normalize=True, title='Confusion matrix', cmap=plt.cm.Blues):
    plt.figure(figsize=(10,10))
    plt.grid(False)
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    cm1 = cm
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm = np.around(cm, decimals=2)
        cm[np.isnan(cm)]
        thresh = cm.max() / 2.
    for i, j in product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, str(cm1[i, j])+ " ("+ str(cm[i, j]*100)+"%)",
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [126]:
```
confusion_mtx = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(confusion_mtx, classes = labels)
```



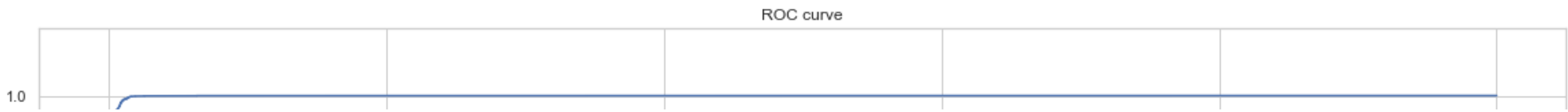
benign
malign
True label
18783 (99.0%)
99 (1.0%)
150 (1.0%)
12120 (99.0%)
benign
malign
Predicted label
15000
12500
10000
7500
5000
2500

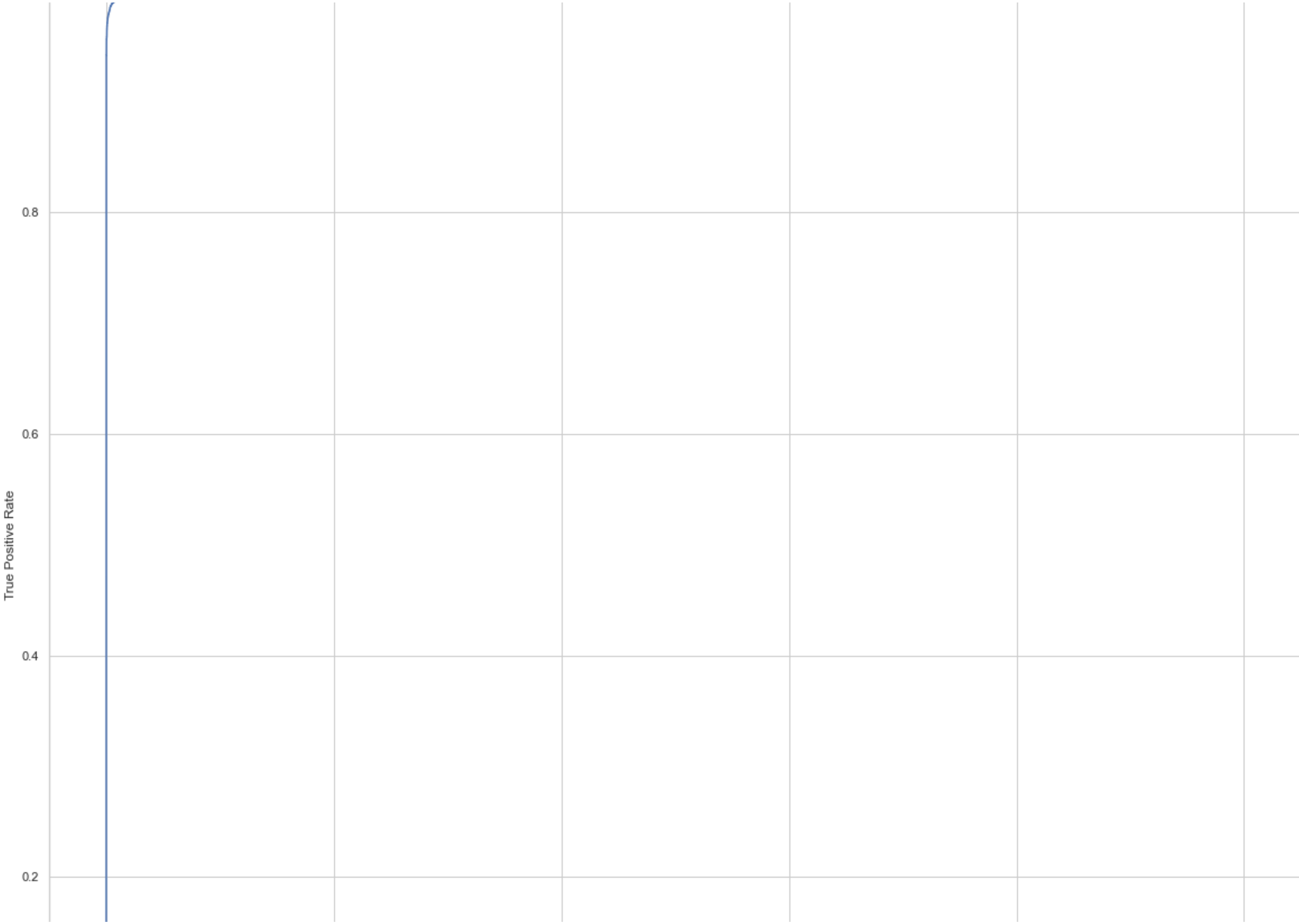
Displaying ROC-AUC curve

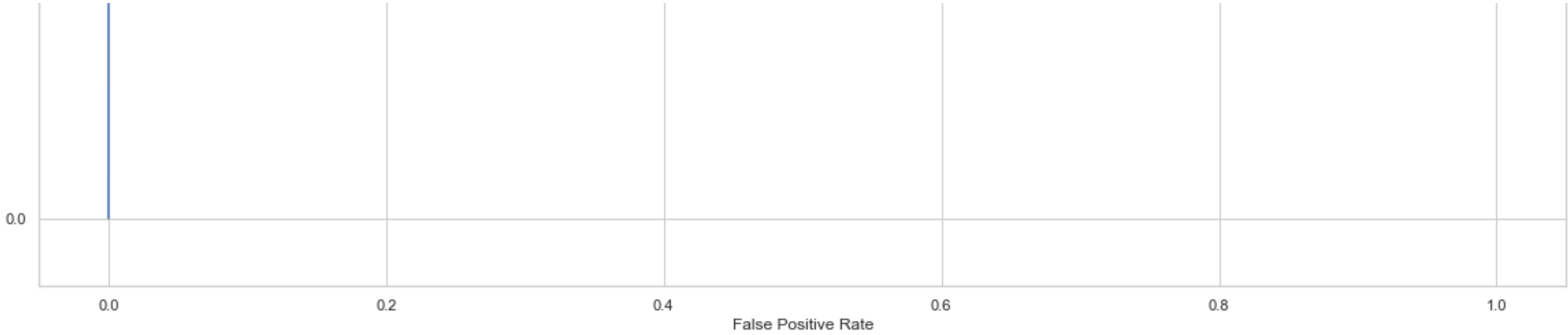
In [87]:
```
def model_builder_crv(X_train, X_test, y_train, y_test):
    model = keras.Sequential()

    model.add(Dense(28 , input_shape=(56,) , activation="relu" , name="Hidden_Layer_1"))
    model.add(Dense(10 , activation="relu" , name="Hidden_Layer_2"))
    model.add(Dense(1 , activation="sigmoid" , name="Output_Layer"))
    opt = keras.optimizers.Adam(learning_rate=0.01)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss='binary_crossentropy', metr
    history = model.fit(X_train,y_train,epochs=100,verbose=0,callbacks=None,validation_data=(X_test,y_te

    return history, model.layers, model
```

In [88]:
```
from sklearn.metrics import roc_curve, auc
plt.figure(figsize=(20,20))
history,model_layers,model = model_builder_crv(X_train, X_test, y_train, y_test)
y_predicted = model(X_test)
fpr, tpr, keras_thr = roc_curve(y_test, y_predicted)
auc_crv = auc(fpr, tpr)
print(f"Area under the curve(AUC) is: {auc_crv}")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(fpr, tpr)
plt.title("ROC curve")
plt.show()
```

```
Area under the curve(AUC) is: 0.9998218809615622
```




0.8
0.6
True Positive Rate
0.4
0.2


0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate