ADA MidSem P2

- Sampad Kumar Kar
- MCS202215

0. Imports

In [ ]:
```
import os, sys
import pandas as pd
import numpy as np

# for plotting
import matplotlib.pyplot as plt
import seaborn as sns

# sklearn imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_sco

# for statistical tests
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [ ]:
```
# ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

0.1 Some helper functions

In [ ]:
```
from pandas.api.types import is_datetime64_dtype, is_categorical_dtype

# helper function to reduce memory usage by compressing datatype
def reduce_mem_usage(df, use_float16=False):
    """
    Iterate through all the columns of a dataframe and modify the data type to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        if is_datetime64_dtype(df[col]) or is_categorical_dtype(df[col]):
            # skip datetime type or categorical type
            continue
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if use_float16 and c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('object')

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

    return df
```

In [ ]:
```
# helper function to find the column with highest VIF and drop it from the dataframe, if it is greater than thres
def drop_correlated_columns(df, vif_threshold):
    # select the columns which do not have the datatype as object
    df = df.select_dtypes(exclude=['object'])

    # variable set to True if we need to continue dropping columns
    flag = True

    while flag:
        # calculate VIF for each feature
        vif = pd.DataFrame()
        vif["features"] = df.columns
        vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]

        # select the column with highest VIF
        max_col = vif.sort_values(by=['VIF'], ascending=False)['features'].iloc[0]

        # check if the VIF is greather than threshold
        if vif[vif['features'] == max_col]['VIF'].iloc[0] > vif_threshold:
            # drop the column
            df = df.drop(max_col, axis=1)
        else:
            # if no column has VIF greater than threshold, exit the loop
            flag = False

    return vif
```

1. Data Loading

In [ ]:
```
data_dir_path = os.path.join('data', 'raw')
```

In [ ]:
```
# read the features
dff = pd.read_csv(os.path.join(data_dir_path, 'NUSW-NB15_features.csv'), encoding='cp1252', index_col=0)

dff
```

Out[ ]:

No.	Name	Type	Description
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number
5	proto	nominal	Transaction protocol
6	state	nominal	Indicates to the state and its dependent proto...
7	dur	Float	Record total duration
8	sbytes	Integer	Source to destination transaction bytes
9	dbytes	Integer	Destination to source transaction bytes
10	sttl	Integer	Source to destination time to live value
11	dttl	Integer	Destination to source time to live value
12	sloss	Integer	Source packets retransmitted or dropped
13	dloss	Integer	Destination packets retransmitted or dropped
14	service	nominal	http, ftp, smtp, ssh, dns, ftp-data ,irc and ...
15	Sload	Float	Source bits per second
16	Dload	Float	Destination bits per second
17	Spkts	integer	Source to destination packet count
18	Dpkts	integer	Destination to source packet count
19	swin	integer	Source TCP window advertisement value
20	dwin	integer	Destination TCP window advertisement value
21	stcpb	integer	Source TCP base sequence number
22	dtcpb	integer	Destination TCP base sequence number
23	smeansz	integer	Mean of the ?ow packet size transmitted by the...
24	dmeansz	integer	Mean of the ?ow packet size transmitted by the...

25	trans_depth	integer	Represents the pipelined depth into the connec...
26	res_bdy_len	integer	Actual uncompressed content size of the data t...
27	Sjit	Float	Source jitter (mSec)
28	Djit	Float	Destination jitter (mSec)
29	Stime	Timestamp	record start time
30	Ltime	Timestamp	record last time
31	Sintpkt	Float	Source interpacket arrival time (mSec)
32	Dintpkt	Float	Destination interpacket arrival time (mSec)
33	tcprtt	Float	TCP connection setup round-trip time, the sum ...
34	synack	Float	TCP connection setup time, the time between th...
35	ackdat	Float	TCP connection setup time, the time between th...
36	is_sm_ips_ports	Binary	If source (1) and destination (3)IP addresses ...
37	ct_state_ttl	Integer	No. for each state (6) according to specific r...
38	ct_flw_http_mthd	Integer	No. of flows that has methods such as Get and ...
39	is_ftp_login	Binary	If the ftp session is accessed by user and pas...
40	ct_ftp_cmd	integer	No of flows that has a command in ftp session.
41	ct_srv_src	integer	No. of connections that contain the same servi...
42	ct_srv_dst	integer	No. of connections that contain the same servi...
43	ct_dst_ltm	integer	No. of connections of the same destination add...
44	ct_src_ ltm	integer	No. of connections of the same source address ...
45	ct_src_dport_ltm	integer	No of connections of the same source address (...
46	ct_dst_sport_ltm	integer	No of connections of the same destination addr...
47	ct_dst_src_ltm	integer	No of connections of the same source (1) and t...
48	attack_cat	nominal	The name of each attack category. In this data...
49	Label	binary	0 for normal and 1 for attack records

In [ ]:

```
# read the data, while setting the 'Name' column as the column names and 'Type' column as the data type of each
df1 = reduce_mem_usage(pd.read_csv(os.path.join(data_dir_path, 'UNSW-NB15_1.csv'), names=dff['Name'].values))
df2 = reduce_mem_usage(pd.read_csv(os.path.join(data_dir_path, 'UNSW-NB15_2.csv'), names=dff['Name'].values))
df3 = reduce_mem_usage(pd.read_csv(os.path.join(data_dir_path, 'UNSW-NB15_3.csv'), names=dff['Name'].values))
df4 = reduce_mem_usage(pd.read_csv(os.path.join(data_dir_path, 'UNSW-NB15_4.csv'), names=dff['Name'].values))

# concatenate the dataframes into a single dataframe
df = pd.concat([df1, df2, df3, df4], ignore_index=True)

df.head()
```

```
Memory usage of dataframe is 261.69 MB
Memory usage after optimization is: 116.16 MB
Decreased by 55.6%
Memory usage of dataframe is 261.69 MB
Memory usage after optimization is: 122.83 MB
Decreased by 53.1%
Memory usage of dataframe is 261.69 MB
Memory usage after optimization is: 122.83 MB
Decreased by 53.1%
Memory usage of dataframe is 164.51 MB
Memory usage after optimization is: 77.22 MB
Decreased by 53.1%
```

Out[ ]:

	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	sttl	...	ct_ftp_cmd	ct_srv_src	ct_srv_dst
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	0.001055	132	164	31	...	0	3	7
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	0.036133	528	304	31	...	0	2	4
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	0.001119	146	178	31	...	0	12	8
3	59.166.0.5	3593	149.171.126.5	53	udp	CON	0.001209	132	164	31	...	0	6	9
4	59.166.0.3	49664	149.171.126.0	53	udp	CON	0.001169	146	178	31	...	0	7	9

5 rows × 49 columns

In [ ]:

```
# print the shape of the dataframe
```

```
df.shape
```

Out[ ]: (2540047, 49)

In [ ]:
```
# print info about the columns
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2540047 entries, 0 to 2540046
Data columns (total 49 columns):
 #   Column            Dtype  
---  ------            -----  
 0   srcip             object 
 1   sport             object 
 2   dstip             object 
 3   dsport            object 
 4   proto             object 
 5   state             object 
 6   dur               float32
 7   sbytes            int32  
 8   dbytes            int32  
 9   sttl              int16  
 10  dttl              int16  
 11  sloss             int16  
 12  dloss             int16  
 13  service           object 
 14  Sload             float32
 15  Dload             float32
 16  Spkts             int16  
 17  Dpkts             int16  
 18  swin              int16  
 19  dwin              int16  
 20  stcpb             int64  
 21  dtcpb             int64  
 22  smeansz           int16  
 23  dmeansz           int16  
 24  trans_depth       int16  
 25  res_bdy_len       int32  
 26  Sjit              float32
 27  Djit              float32
 28  Stime             int32  
 29  Ltime             int32  
 30  Sintpkt           float32
 31  Dintpkt           float32
 32  tcprtt            float32
 33  synack            float32
 34  ackdat            float32
 35  is_sm_ips_ports   int8   
 36  ct_state_ttl      int8   
 37  ct_flw_http_mthd  float32
 38  is_ftp_login      float32
 39  ct_ftp_cmd        object 
 40  ct_srv_src        int8   
 41  ct_srv_dst        int8   
 42  ct_dst_ltm        int8   
 43  ct_src_ ltm       int8   
 44  ct_src_dport_ltm  int8   
 45  ct_dst_sport_ltm  int8   
 46  ct_dst_src_ltm    int8   
 47  attack_cat        object 
 48  Label             int8   
dtypes: float32(12), int16(11), int32(5), int64(2), int8(10), object(9)
memory usage: 455.4+ MB
```

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

Out[ ]:

	dur	sbytes	dbytes	sttl	dttl	sloss	dloss	Sload
count	2.540047e+06	2.540047e+06	2.540047e+06	2.540047e+06	2.540047e+06	2.540047e+06	2.540047e+06	2.540047e+06
mean	6.587917e-01	4.339600e+03	3.642759e+04	6.278197e+01	3.076681e+01	5.163921e+00	1.632944e+01	3.695645e+07
std	1.392493e+01	5.640599e+04	1.610960e+05	7.462277e+01	4.285089e+01	2.251707e+01	5.659474e+01	1.186043e+08
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.037000e-03	2.000000e+02	1.780000e+02	3.100000e+01	2.900000e+01	0.000000e+00	0.000000e+00	1.353963e+05
50%	1.586100e-02	1.470000e+03	1.820000e+03	3.100000e+01	2.900000e+01	3.000000e+00	4.000000e+00	5.893038e+05
75%	2.145545e-01	3.182000e+03	1.489400e+04	3.100000e+01	2.900000e+01	7.000000e+00	1.400000e+01	2.039923e+06
max	8.786638e+03	1.435577e+07	1.465753e+07	2.550000e+02	2.540000e+02	5.319000e+03	5.507000e+03	5.988000e+09

8 rows × 40 columns

2. Data Summarization

2.1 Total no. of flows.

Each flow is determined by the 5-tuple of the following features:

- `srcip`
- `sport`
- `dstip`
- `dsport`
- `proto`

In [ ]:
```
columns_of_interest = ['srcip', 'sport', 'dstip', 'dsport', 'proto']
unique_flows = df[columns_of_interest].drop_duplicates()

# extract the no. of unique flows
num_unique_flows = unique_flows.shape[0]
print("Total no. of unique flows: ", num_unique_flows)
```

```
Total no. of unique flows:  1965893
```

2.2 Total flow duration.

Total flow duration is just the sum of all the entries in the `dur` column.

In [ ]:
```
# total flow duration
total_flow_duration = df['dur'].sum()
print("Total flow duration (in s): ", total_flow_duration)
```

```
Total flow duration (in s):  1673361.9
```

3. Data Preprocessing

3.1 Handling Missing Values

In [ ]:
```
# check for missing values and only display the columns with non zero missing values
df.isnull().sum()[df.isnull().sum() > 0]
```

Out[ ]:
```
ct_flw_http_mthd    1348145
is_ftp_login        1429879
attack_cat          2218764
dtype: int64
```

In [ ]:
```
# display the proportion of missing values
df.isnull().sum()[df.isnull().sum() > 0] / df.shape[0]
```

Out[ ]:
```
ct_flw_http_mthd    0.530756
is_ftp_login        0.562934
attack_cat          0.873513
dtype: float64
```

Based on the information above we choose to employ the following strategy to handle missing values corresponding to these 3 columns:

- `ct_flw_http_mthd` : This columns contains integers, so we impute the missing values with the closest integer to the mean.
- `is_ftp_login` : This is a boolean columnm, so we impute the missing values with the mode.
- `attack_cat` : This is a nominal column with distict categories and this column also has `87.35%` of it's values missing. So we drop this column.

In [ ]:
```
# impute with closest integer to the mean
mean_ct_flw_http_mthd = df['ct_flw_http_mthd'].mean()
df['ct_flw_http_mthd'].fillna(round(mean_ct_flw_http_mthd), inplace=True)

# impute with mode
mode_is_ftp_login = df['is_ftp_login'].mode()[0]
df['is_ftp_login'].fillna(mode_is_ftp_login, inplace=True)

# drop column
df.drop('attack_cat', axis=1, inplace=True)
```

In [ ]:
```
# check for missing values and only display the columns with non zero missing values
df.isnull().sum()[df.isnull().sum() > 0]
```

Out[ ]:
```
Series([], dtype: int64)
```

So, we do not have missing values anymore.

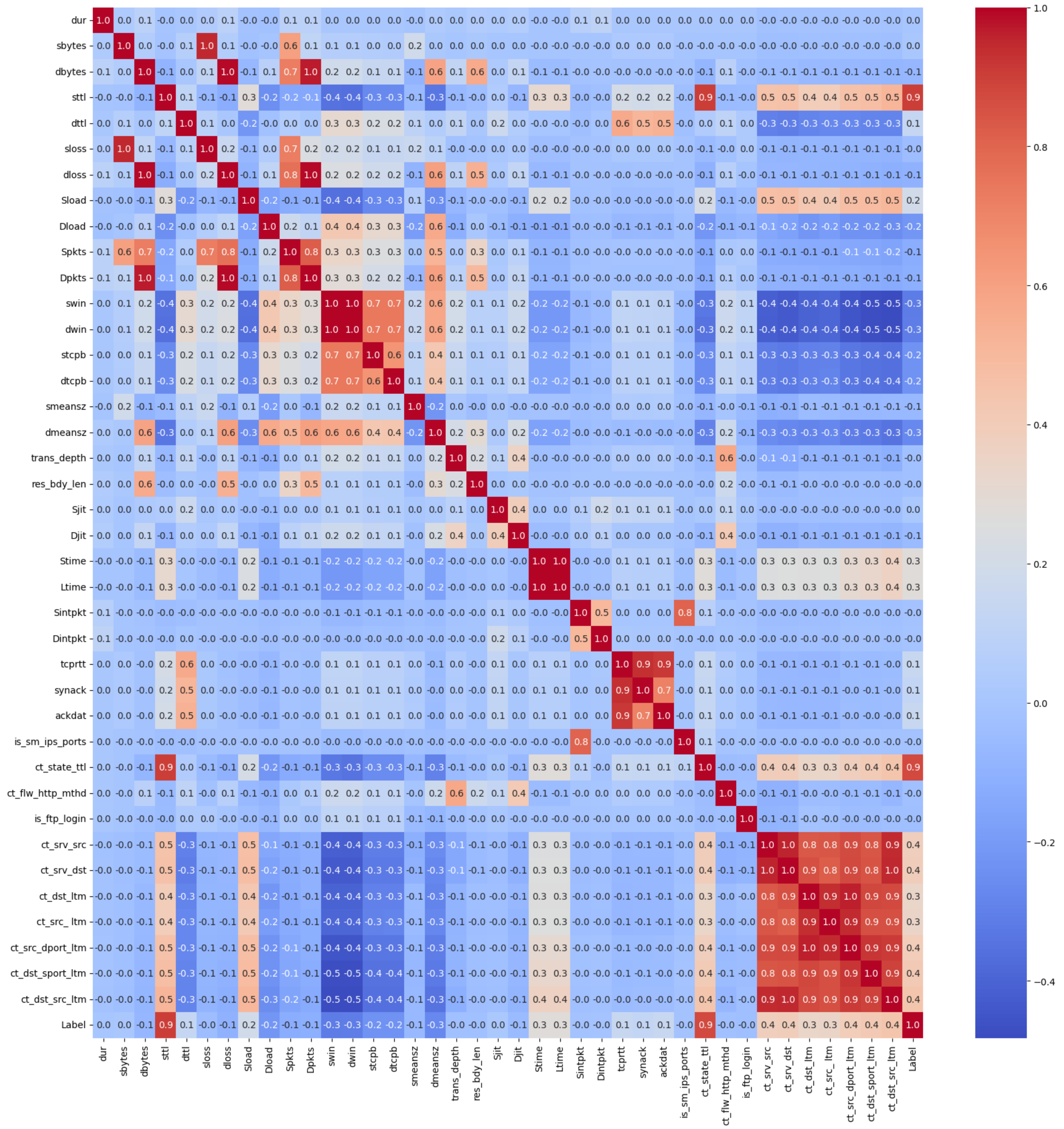
3.2 Check for Multicollinearity

3.2.1 Correlation Matrix

We check for multicollinearity using the correlation matrix. Columns with correlation values close to `1` or `-1` are considered to be highly correlated.

In [ ]:
```
# compute the correlation matrix for the numerical features
corr_matrix = df.select_dtypes(exclude=['object']).corr()

# plot the heatmap
plt.figure(figsize=(20, 20))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".1f")
plt.show()
```



3.2.2 Variance Inflation Factor (VIF)

We also check for multicollinearity using the VIF values. Columns with VIF values greater than `10` are considered to be highly correlated.

Here is how we calculate the VIF values:

$$VIF_i = \frac{1}{1 - R_i^2}$$

where R_i^2 is the R^2 value of the regression of the i^{th} feature on all the other features.

We drop the variables one after another, in decreasing order of their VIF values, until all the VIF values are less than `10` for the remaining columns. For this we use the helper function `drop_correlated_columns`.

In [ ]:

```
%%time

# we set a threshold of 10
threshold = 10
vif = drop_correlated_columns(df, 10)
vif
```

```
CPU times: user 5h 49s, sys: 28min 9s, total: 5h 28min 59s
Wall time: 57min 40s
```

Out[ ]:

	features	VIF
0	dur	1.022104
1	sbytes	4.254992
2	dbytes	7.467053
3	dttl	2.017366
4	Sload	1.423300
5	Dload	2.128370
6	Spkts	9.827260
7	swin	4.917358
8	stcpb	2.338797
9	dtcpb	2.338982
10	smeansz	1.405013
11	dmeansz	4.452056
12	trans_depth	1.712197
13	res_bdy_len	1.632796
14	Sjit	1.256313
15	Djit	1.596539
16	Ltime	0.000004
17	Dintpkt	1.052766
18	synack	2.184516
19	ackdat	2.211254
20	is_sm_ips_ports	1.070685
21	ct_state_ttl	4.862982
22	ct_flw_http_mthd	1.653777
23	is_ftp_login	1.118511
24	ct_srv_src	4.631355
25	ct_src_ ltm	4.837234
26	ct_dst_sport_ltm	5.361229
27	Label	5.056842

In [ ]:

```
# columns to remove from the dataframe
non_object_columns = [col for col in df.columns if df[col].dtype != 'object']
columns_to_remove = [col for col in non_object_columns if (col not in vif['features'].values)]
```

In [ ]:

```
columns_to_remove
```

Out[ ]:

```
['sttl',
 'sloss',
 'dloss',
 'Dpkts',
 'dwin',
 'Stime',
 'Sintpkt',
 'tcprtt',
 'ct_srv_dst',
 'ct_dst_ltm',
 'ct_src_dport_ltm',
 'ct_dst_src_ltm']
```

We remove the above listed columns from the dataframe due to them having high multicollinearity.

In [ ]:

```
# columns of dataframe before removing correlated columns
df.columns
```

Out[ ]:
```
Index(['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'dur', 'sbytes',
       'dbytes', 'sttl', 'dttl', 'sloss', 'dloss', 'service', 'Sload', 'Dload',
       'Spkts', 'Dpkts', 'swin', 'dwin', 'stcpb', 'dtcpb', 'smeansz',
       'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit', 'Djit', 'Stime',
       'Ltime', 'Sintpkt', 'Dintpkt', 'tcprtt', 'synack', 'ackdat',
       'is_sm_ips_ports', 'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login',
       'ct_ftp_cmd', 'ct_srv_src', 'ct_srv_dst', 'ct_dst_ltm', 'ct_src_ ltm',
       'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'Label'],
      dtype='object')
```

In [ ]:
```python
# remove the columns from the dataframe
df.drop(columns_to_remove, axis=1, inplace=True)

# columns of dataframe after removing correlated columns
df.columns
```

Out[ ]:
```
Index(['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'dur', 'sbytes',
       'dbytes', 'dttl', 'service', 'Sload', 'Dload', 'Spkts', 'swin', 'stcpb',
       'dtcpb', 'smeansz', 'dmeansz', 'trans_depth', 'res_bdy_len', 'Sjit',
       'Djit', 'Ltime', 'Dintpkt', 'synack', 'ackdat', 'is_sm_ips_ports',
       'ct_state_ttl', 'ct_flw_http_mthd', 'is_ftp_login', 'ct_ftp_cmd',
       'ct_srv_src', 'ct_src_ ltm', 'ct_dst_sport_ltm', 'Label'],
      dtype='object')
```

3.3 Handling Categorical Variables

In [ ]:
```python
# we check all the columns with datatype as object
df_nominal = df.select_dtypes(include=['object'])

df_nominal.head()
```

Out[ ]:

	srcip	sport	dstip	dsport	proto	state	service	ct_ftp_cmd
0	59.166.0.0	1390	149.171.126.6	53	udp	CON	dns	0
1	59.166.0.0	33661	149.171.126.9	1024	udp	CON	-	0
2	59.166.0.6	1464	149.171.126.7	53	udp	CON	dns	0
3	59.166.0.5	3593	149.171.126.5	53	udp	CON	dns	0
4	59.166.0.3	49664	149.171.126.0	53	udp	CON	dns	0

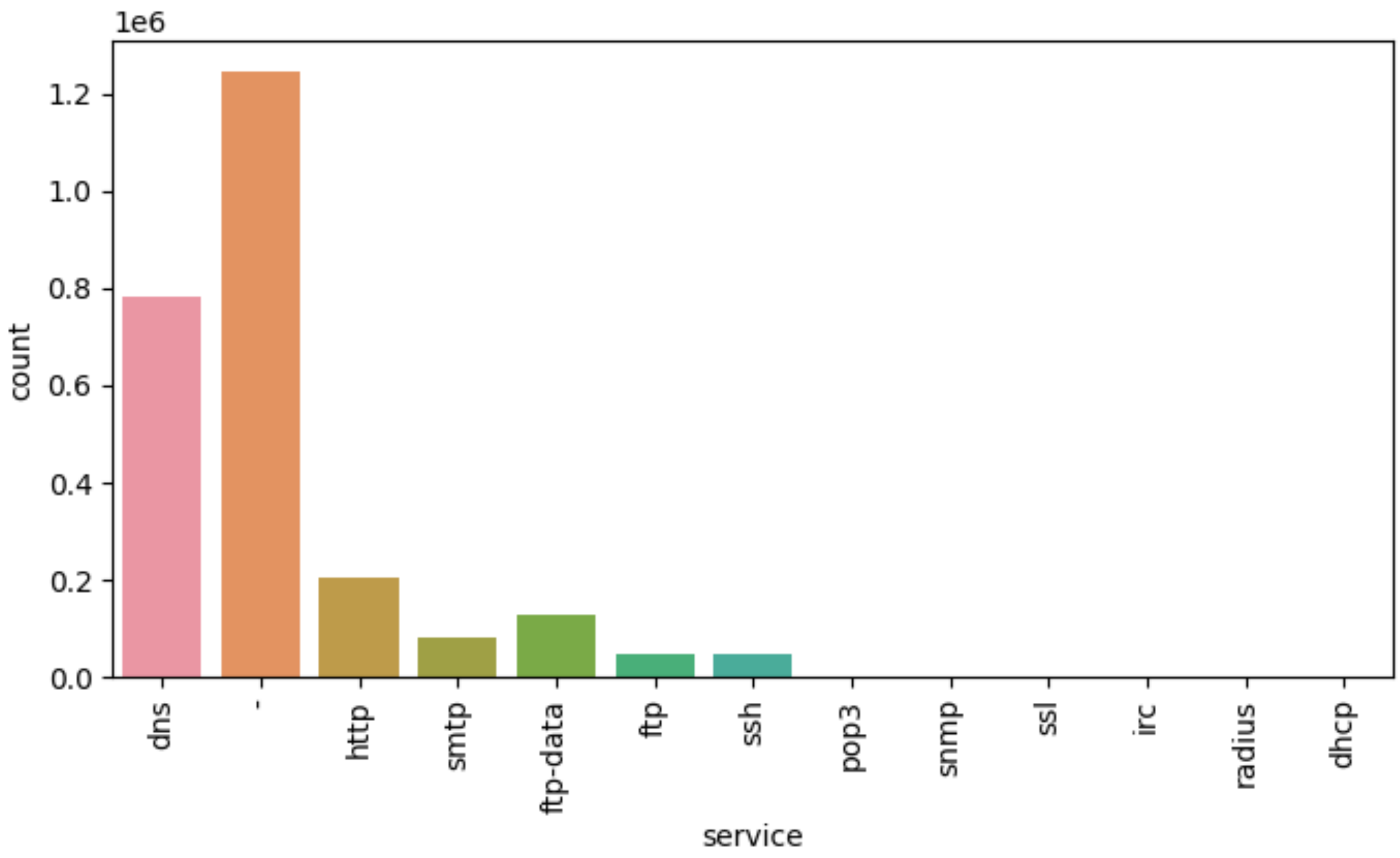
3.3.1 Handling the target variable

In this dataset, the `service` column is the target variable. This is of object type. We first find all the unique values in this column and then encode them.

In [ ]:
```python
# check the unique values in 'service' column
df_nominal['service'].value_counts()
```

Out[ ]:
```
service
-           1246397
dns          781668
http         206273
ftp-data     125783
smtp          81645
ftp           49090
ssh           47160
pop3           1533
dhcp            172
ssl             142
snmp            113
radius           40
irc              31
Name: count, dtype: int64
```

In [ ]:
```python
# plot a barchart for 'service' column counts
plt.figure(figsize=(8, 4))
sns.countplot(x='service', data=df_nominal)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:
```
# we use the unique values in 'service' column as class labels
class_labels = df_nominal['service'].unique()
class_labels
```

Out[ ]:
```
array(['dns', '-', 'http', 'smtp', 'ftp-data', 'ftp', 'ssh', 'pop3',
       'snmp', 'ssl', 'irc', 'radius', 'dhcp'], dtype=object)
```

3.3.2 Label Encoding the nominal columns

In [ ]:
```
# select the columns of interest (columns in df_nominal except 'service' column)
nominal_columns = [col for col in df_nominal.columns if col != 'service']
nominal_columns
```

Out[ ]:
```
['srcip', 'sport', 'dstip', 'dsport', 'proto', 'state', 'ct_ftp_cmd']
```

In [ ]:
```
# columns made up of multiple datatypes
compound_columns = ['sport', 'dsport', 'ct_ftp_cmd']
```

In [ ]:
```
le = LabelEncoder()

df_processed = df.copy()

# iterate through the columns of interest
for col in nominal_columns:
    if col in compound_columns:
        # convert the column to string datatype
        df_processed[col] = df_processed[col].astype(str)
        # fit the label encoder on the column
        df_processed[col] = le.fit_transform(df_processed[col])
    else:
        # fit the label encoder on the column
        df_processed[col] = le.fit_transform(df_processed[col])


df_processed.head()
```

Out[ ]:

	srcip	sport	dstip	dsport	proto	state	dur	sbytes	dbytes	dttl	...	ackdat	is_sm_ips_ports	ct_state_ttl	ct_flw_
0	33	4276	24	47344	120	2	0.001055	132	164	29	...	0.0	0	0	
1	33	26036	27	253	120	2	0.036133	528	304	29	...	0.0	0	0	
2	39	5091	25	47344	120	2	0.001119	146	178	29	...	0.0	0	0	
3	38	28534	23	47344	120	2	0.001209	132	164	29	...	0.0	0	0	
4	36	43654	8	47344	120	2	0.001169	146	178	29	...	0.0	0	0	

5 rows × 36 columns

In [ ]:
```
df_processed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2540047 entries, 0 to 2540046
Data columns (total 36 columns):
 #   Column            Dtype  
---  ------            -----  
 0   srcip             int64  
 1   sport             int64  
 2   dstip             int64  
 3   dsport            int64  
 4   proto             int64  
 5   state             int64  
 6   dur               float32
 7   sbytes            int32  
 8   dbytes            int32  
 9   dttl              int16  
 10  service           object 
 11  Sload             float32
 12  Dload             float32
 13  Spkts             int16  
 14  swin              int16  
 15  stcpb             int64  
 16  dtcpb             int64  
 17  smeansz           int16  
 18  dmeansz           int16  
 19  trans_depth       int16  
 20  res_bdy_len       int32  
 21  Sjit              float32
 22  Djit              float32
 23  Ltime             int32  
 24  Dintpkt           float32
 25  synack            float32
 26  ackdat            float32
 27  is_sm_ips_ports   int8   
 28  ct_state_ttl      int8   
 29  ct_flw_http_mthd  float32
 30  is_ftp_login      float32
 31  ct_ftp_cmd        int64  
 32  ct_srv_src        int8   
 33  ct_src_ ltm       int8   
 34  ct_dst_sport_ltm  int8   
 35  Label             int8   
dtypes: float32(10), int16(6), int32(4), int64(9), int8(6), object(1)
memory usage: 373.0+ MB
```

3.3.3 Splitting Features and Target

In [ ]:
```
# we split the data into features and labels
features = df_processed.drop('service', axis=1)
labels = df_processed['service']

# print the shape of features and labels
print("Shape of features: ", features.shape)
print("Shape of labels: ", labels.shape)
```

```
Shape of features:  (2540047, 35)
Shape of labels:  (2540047,)
```

We have encoded all the nominal columns using the `LabelEncoder`. Now, we need to use `StandardScaler` to scale the data.

In [ ]:
```
# scale the features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

3.4 Train-Test Split

We split the data into training and testing sets using the `train_test_split` function from `sklearn.model_selection`. We use a `test_size` of `0.2` and use stratified sampling to ensure that the distribution of the target variable is the same in both the training and testing sets.

In [ ]:
```
# stratified split of data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features_scaled, labels, test_size=0.2, random_state=42, stra
```

In [ ]:
```
# check the shape of train and test sets
print("Shape of X_train: ", X_train.shape)
print("Shape of y_train: ", y_train.shape)

print("Shape of X_test: ", X_test.shape)
print("Shape of y_test: ", y_test.shape)
```

```
Shape of X_train:  (2032037, 35)
Shape of y_train:  (2032037,)
Shape of X_test:  (508010, 35)
Shape of y_test:  (508010,)
```

4. Model Fit and Evaluation

We fit a simple Logistic Regression model to the data. We use the `LogisticRegression` class from `sklearn.linear_model`. We use the `predict` method to predict the target variable for the test set.

In [ ]:
```
log_reg = LogisticRegression()

# fit the model on train data
log_reg.fit(X_train, y_train)
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
# predict on test data
y_test_pred = log_reg.predict(X_test)
```

4.1 Classification Report

In [ ]:
```
# print the classification report
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           -       0.98      0.97      0.97    249280
        dhcp       0.32      0.29      0.31        34
         dns       0.97      1.00      0.99    156334
         ftp       0.98      0.95      0.97      9818
    ftp-data       0.95      0.97      0.96     25157
        http       0.98      0.98      0.98     41255
         irc       0.00      0.00      0.00         6
        pop3       0.70      0.50      0.58       307
      radius       0.00      0.00      0.00         8
        smtp       0.98      0.97      0.97     16329
        snmp       0.00      0.00      0.00        22
         ssh       0.79      0.78      0.78      9432
         ssl       0.00      0.00      0.00        28

    accuracy                           0.97    508010
   macro avg       0.59      0.57      0.58    508010
weighted avg       0.97      0.97      0.97    508010
```

4.3 Accuracy Score

In [ ]:
```
# print the accuracy score
print("Accuracy score: ", accuracy_score(y_test, y_test_pred))
```

```
Accuracy score:  0.973600913367847
```

4.4 Precision Score

In [ ]:
```
# print the weighted precision score
print("Weighted precision score: ", precision_score(y_test, y_test_pred, average='weighted'))
```

```
Weighted precision score:  0.9734779902861929
```

4.5 Recall Score

In [ ]:
```
# print the weighted recall score
print("Weighted recall score: ", recall_score(y_test, y_test_pred, average='weighted'))
```

```
Weighted recall score:  0.973600913367847
```

We get the following scores:

- Accuracy: `97.36 %`
- Weighted Precision: `97.34 %`
- Weighted Recall: `97.36 %`

Because of the high score, we can conclude that the model is performing well.