Network Intrusion Detection - Apache Spark

Team 2 - Shubham Chawla, Chintan Vajani, Archit Sinha

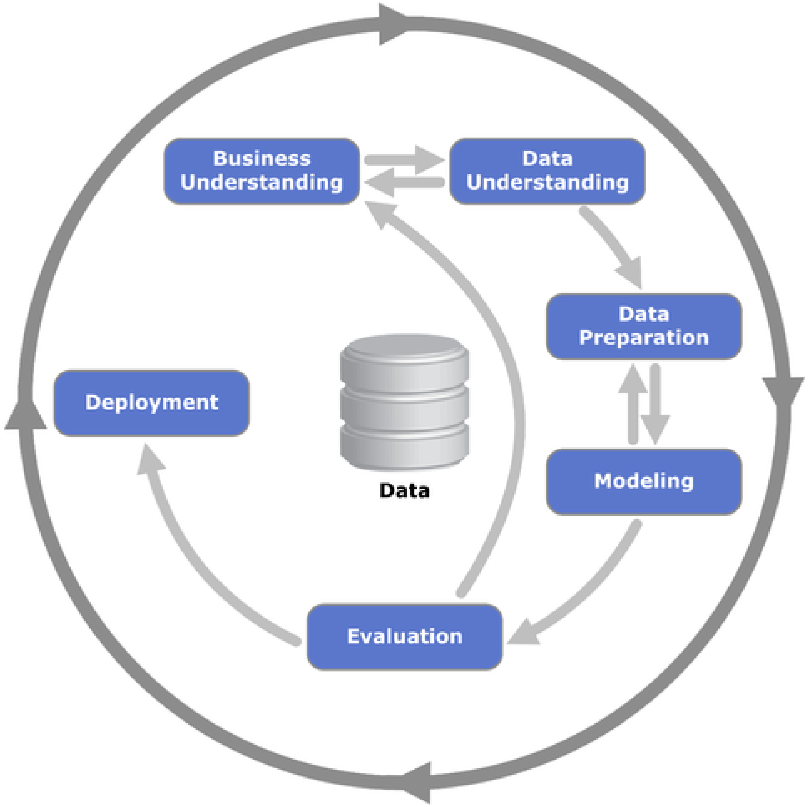
Project Summary

A system's security and privacy are jeopardised when an intrusion occurs. The Intrusion Detection System (IDS) is essential for network security since it can identify various kinds of attacks. Therefore, utilising machine learning algorithms, we will propose an intrusion detection system here. The suggested system will be tested using the KDD99 Dataset. Our project includes implementation and performance evaluation of a Big Data project whose goal is to use Apache Spark and MLlib in order to perform network traffic analysis on KDD99 dataset. Further the project was deployed on Flask and hosted on Heroku.

Dataset link: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

(http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html)

Data Mining Process



Ref: https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining (https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining)

The process of generating and implementing data-driven solutions to business problems is iterative, and the model highlights this by highlighting a number of distinct stages that are frequently revisited during this process.

- Business understanding
- Data understanding
- Data exploration and preparation

- Model Building
- Deployment

Business Understanding

Numerous firms are now at danger as a result of the growing cyber security risks. Any organisation could be the target of attackers who can cause significant harm to it.

For these reasons, it's essential to offer a system that aids in the detection of network intrusions. In order to facilitate the deployment and to obtain findings as quickly as possible, this intrusion detection system should be non-invasive to other systems.

Data Understanding

The dataset includes samples of network connections that are either normal or under attack; in this instance, the attack type is mentioned. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type. Each connection record consists of about 100 bytes.

It consists of several attack methods like Denial of Service (DoS), port scanning, and buffer overflow along with normal samples that have all been combined into a single comma-separated values (CSV) file with 4898431 samples and 42 characteristics. The resulting features are categorised into the following groups:

Attacks fall into four main categories:

- **DOS:** denial-of-service, e.g. syn flood
- **R2L:** unauthorized access from a remote machine, e.g. guessing password
- **U2R:** unauthorized access to local superuser (root) privileges, e.g., various ``buffer overflow'' attacks
- **probing:** surveillance and other probing, e.g., port scanning.

Features essential for distniguishing *normal* connections from *attacks*

Basic Features of each network connection vector

No	Feature name	description	type
1	duration	length (number of seconds) of the connection	continuous
2	protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
3	service	network service on the destination, e.g., http, telnet, etc.	discrete
4	flag	normal or error status of the connection. The possible status are this: SF, S0, S1, S2, S3, OTH, REJ, RSTO, RSTOS0, SH, RSTRH, SHR	discrete
5	src_bytes	number of data bytes from source to destination	continuous
6	dst_bytes	number of data bytes from destination to source	continuous
7	land	1 if connection is from/to the same host/port; 0 otherwise	discrete
8	wrong_fragment	sum of bad checksum packets in a connection	continuous
9	urgent	number of urgent packets. Urgent packets are packets with the urgent bit activated	continuous

Content Related Features of each network connection vector

No.	Feature name	description	type
10	hot	sum of hot actions in a connection such as: entering a system direc- tory, creating programs and executing programs	continuous
11	num_failed_logins	number of failed login attempts	continuous
12	logged_in	1 if successfully logged in; 0 otherwise	discrete
13	num_compromised	number of "compromised" conditions	continuous
14	root_shell	1 if root shell is obtained; 0 otherwise	discrete
15	su_attempted	1 if "su root" command attempted; 0 otherwise	discrete
16	num_root	number of "root'" accesses	continuous
17	num_file_creations	number of file creation operations	continuous
18	num_shells	number of shell prompts	continuous
19	num_access_files	number of operations on access control files	continuous
20	num_outbound_cmds	number of outbound commands in an ftp session	continuous
21	is_hot_login	1 if the login belongs to the "hot" list; 0 otherwise	discrete
22	is_guest_login	1 if the login is a "guest" login; 0 otherwise	discrete

Time Related Traffic Features of each network connection vector

Col	Feature name	description	type
23	count	sum of connections to the same destination IP address	continuous
24	srv_count	sum of connections to the same destination port number	continuous
25	serror_rate	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23)	continuous
26	srv_serror_rate	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24)	continuous
27	rerror_rate	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in count (23)	continuous
29	same_srv_rate	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24)	continuous
28	srv_rerror_rate	the percentage of connections that were to the same service, among the connections aggregated in count (23)	continuous
30	diff_srv_rate	the percentage of connections that were to different services, among the connections aggregated in count (23)	continuous
31	srv_diff_host_rate	the percentage of connections that were to different destination ma- chines among the connections aggregated in srv_count (24)	continuous

Host Based Traffic Features of each network connection vector

Col	Feature name	description	type
32	dst_host_count	sum of connections to the same destination IP address	continuous
33	dst_host_srv_count	sum of connections to the same destination port number	continuous
34	dst_host_same_srv_rate	the percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32)	continuous
35	dst_host_diff_srv_rate	the percentage of connections that were to different services, among the connections aggregated in dst_host_count (32)	continuous
36	dst_host_same_src_port_rate	the percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_count (33)	continuous
37	dst_host_srv_diff_host_rate	the percentage of connections that were to different destination ma- chines, among the connections aggregated in dst_host_srv_count (33)	continuous
38	dst_host_serror_rate	the percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32)	continuous
39	dst_host_srv_serror_rate	the percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count (33)	continuous

Col	Feature name	description	type
40	dst_host_rerror_rate	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32)	continuous
41	dst_host_srv_rerror_rate	the percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_count (33)	continuous

The 42nd attribute is our target variable where we will be using Multinomial Classification to predict whether our class is Normal or DOS or PROBE or R2L or U2R

Initializing Apache Spark and Importing Necessary Libraries

In [1]:

```
import pyspark
from pyspark.context import SparkContext
import warnings
warnings.filterwarnings("ignore")
import time
from pyspark.sql import SparkSession

# To perform feature engineering
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml import Pipeline

# To classify label class
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, \
                                      NaiveBayes, RandomForestClassifier

# Importing MulticlassClassificationEvaluator to evaluate the performance of the classifiers (models),
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.sql.functions import regexp_replace,lit
# Spark Session
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
spark.conf.set('spark.sql.shuffle.partitions', 6)
spark.conf.set('num-executors', 16)
spark
```

```
SparkSession - in-memory

SparkContext
```

Spark UI (http://ShubhamPC:4040)

Version

v3.3.0

Master

local[*]

AppName

pyspark-shell

Data Exploration & Preparation

In [2]:

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

In [3]:

```
# Defining attribute names as they were not present in the data file
columns = ["duration", "protocol_type", "service", "flag", "src_bytes", "dst_bytes",
            "land", "wrong_fragment", "urgent", "hot", "num_failed_logins", "logged_in",
            "num_compromised", "root_shell", "su_attempted", "num_root", "num_file_creations",
            "num_shells", "num_access_files", "num_outbound_cmds", "is_host_login",
            "is_guest_login", "count", "srv_count", "serror_rate", "srv_serror_rate", "rerror_rate",
            "srv_rerror_rate", "same_srv_rate", "diff_srv_rate", "srv_diff_host_rate", "dst_host_count",
            "dst_host_srv_count", "dst_host_same_srv_rate", "dst_host_diff_srv_rate",
            "dst_host_same_src_port_rate", "dst_host_srv_diff_host_rate", "dst_host_serror_rate",
            "dst_host_srv_serror_rate", "dst_host_rerror_rate", "dst_host_srv_rerror_rate", "label"]

# Reading csv file into spark dataframe
sparkdf = spark.read.csv("kddcup.data.corrected", inferSchema=True, header=False)

# Adding the list of features to our dataset
sparkdf = sparkdf.toDF(*columns)
sparkdf = sparkdf.withColumn("label", regexp_replace("label", "\.", ""))

print("Dataset sizes: {row} rows, {cols} columns".format(row=sparkdf.count(), cols=len(sparkdf.columns)))
```

```
Dataset sizes: 4898431 rows, 42 columns
```

https://stackoverflow.com/questions/39067505/pyspark-display-a-spark-data-frame-in-a-table-format (https://stackoverflow.com/questions/39067505/pyspark-display-a-spark-data-frame-in-a-table-format) https://sparkbyexamples.com/pyspark/pyspark-find-count-of-null-none-nan-values/#:~:text=In%20PySpark%20DataFrame%20you%20can,count()%20and%20when() (https://sparkbyexamples.com/pyspark/pyspark-find-count-of-null-none-nan-values/#:~:text=In%20PySpark%20DataFrame%20you%20can,count()%20and%20when()).

In [4]:

```
# To convert large dataframes to pandas
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")

# Checking nulls in all the columns
from pyspark.sql.functions import col,isnan, when, count
sparkdf.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in sparkdf.columns]
   ).toPandas()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	c
0	0	0	0	0	0	0	0	0	0	0	...	0	0

```
1 rows × 42 columns
```

As we can see, there are no null values present in the dataset. So there is no need of cleaning with respect to null or missing values.

Using Pandas API on Spark to explore data further

In [5]:

```
import pyspark.pandas as ps
psdf = sparkdf.to_pandas_on_spark()
```

```
WARNING:root:'PYARROW_IGNORE_TIMEZONE' environment variable was not set. It is required to set this environment variable to '1' in both dr
iver and executor sides if you use pyarrow>=2.0.0. pandas-on-Spark will set it for you but it does not work if there is a Spark context al
ready launched.
```

In [6]:

```
type(psdf)
```

```
pyspark.pandas.frame.DataFrame
```

In [7]:

```
psdf.head()
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logge
0	0	tcp	http	SF	215	45076	0	0	0	0	0	1
1	0	tcp	http	SF	162	4528	0	0	0	0	0	1
2	0	tcp	http	SF	236	1228	0	0	0	0	0	1
3	0	tcp	http	SF	233	2032	0	0	0	0	0	1
4	0	tcp	http	SF	239	486	0	0	0	0	0	1

In [8]:

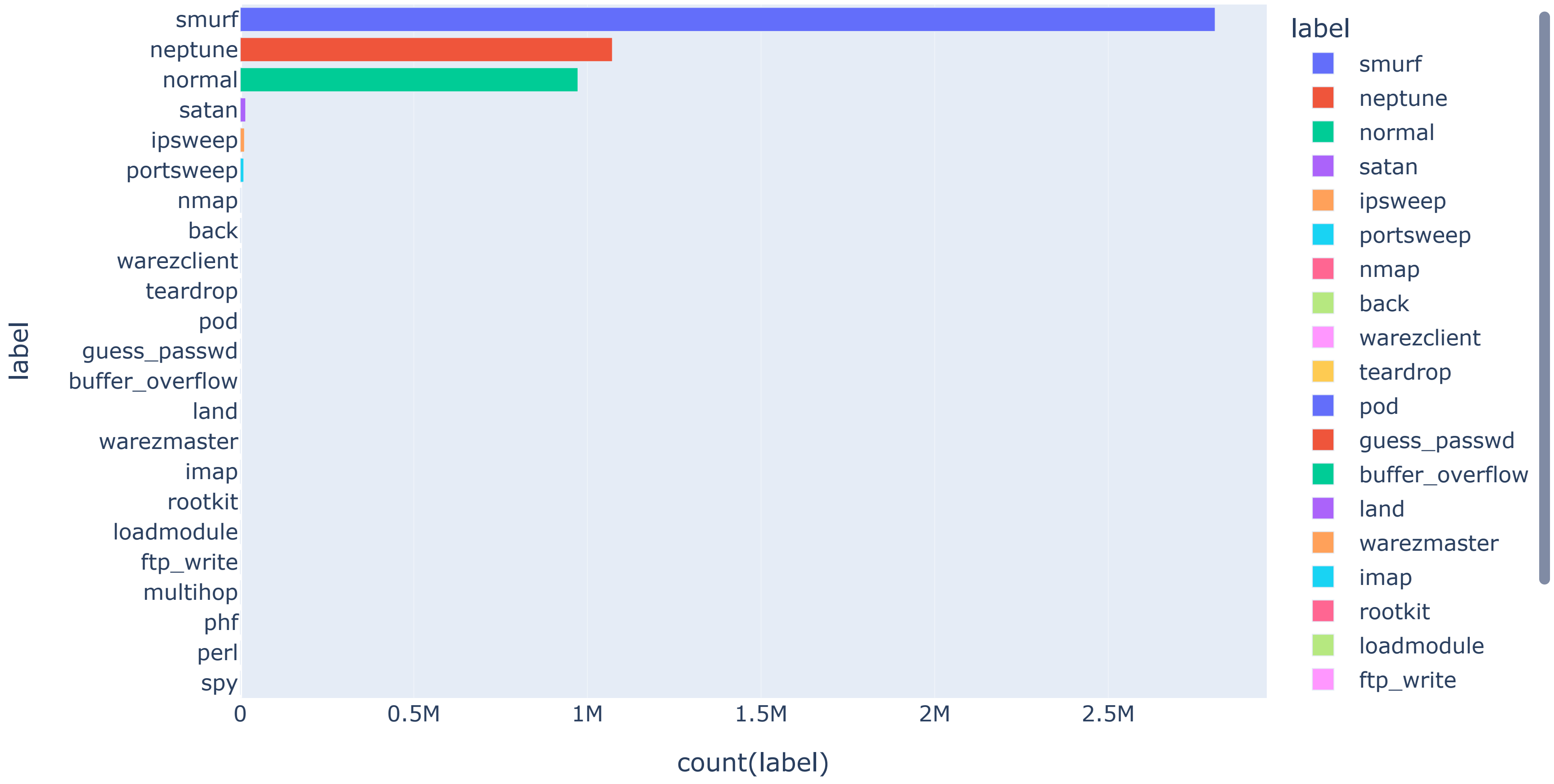
```
# Distribution of label - attack type class
from pyspark.sql import functions as F
df1 = sparkdf.groupby('label').agg(F.count('label')).orderBy('count(label)', ascending= False)
df1.show(23)
```

```
+---------------+------------+
|          label|count(label)|
+---------------+------------+
|          smurf|     2807886|
|        neptune|     1072017|
|         normal|      972781|
|          satan|       15892|
|        ipsweep|       12481|
|      portsweep|       10413|
|           nmap|        2316|
|           back|        2203|
|    warezclient|        1020|
|       teardrop|         979|
|            pod|         264|
|   guess_passwd|          53|
|buffer_overflow|          30|
|           land|          21|
|    warezmaster|          20|
|           imap|          12|
|        rootkit|          10|
|     loadmodule|           9|
|      ftp_write|           8|
|       multihop|           7|
|            phf|           4|
```

Plotting Horizontal Bar Chart for all label categories

In [9]:

```
psdf = df1.to_pandas_on_spark()
psdf.plot.bar(y = 'label', x = 'count(label)', color = 'label').update_yaxes(categoryorder="total ascending'
```



There are way too many categories in the label to predict. It is better to encode them according to 5 main categories - Normal, DOS, Probe, R2L and U2R

Using Pyspark sql to encode 23 different label categories into 5 categories

- 0 - Normal
- 1 - DOS
- 2 - Probe
- 3 - R2L
- 4 - U2R

In [10]:

```
# Using pyspark sql to encode attack categories into 5 different categories
#0- normal, 1- DOS, 2- Probe, 3 - R2L(remote to user attack), 4 - U2R(User to Root attack)
# Creating a temp view
sparkdf.createOrReplaceTempView("network")

# adding label_num column
df_sql = spark.sql("""
SELECT *,
CASE
    WHEN label = 'normal' THEN 0
    WHEN label = 'back' OR label = 'smurf' OR label = 'pod' OR label = 'land' OR label = 'neptune' OR label
    WHEN label = 'satan' OR label = 'ipsweep' OR label = 'nmap' OR label = 'portsweep' THEN 2
    WHEN label = 'guess_passwd' OR label = 'ftpwrite' OR label = 'imap' OR label = 'phf' OR label = 'multih
    ELSE 4
END AS label_num,
CASE
    WHEN label = 'normal' THEN 'normal'
    WHEN label = 'back' OR label = 'smurf' OR label = 'pod' OR label = 'land' OR label = 'neptune' OR label
    WHEN label = 'satan' OR label = 'ipsweep' OR label = 'nmap' OR label = 'portsweep' THEN 'Probe'
    WHEN label = 'guess_passwd' OR label = 'ftpwrite' OR label = 'imap' OR label = 'phf' OR label = 'multih
    ELSE 'U2L'
END AS label_cat
FROM network;
""")
df_sql.select(['label_num','label_cat']).distinct().orderBy('label_num',ascending=True).show()
```

```
+---------+---------+
|label_num|label_cat|
+---------+---------+
|        0|   normal|
|        1|      DOS|
|        2|    Probe|
|        3|      R2L|
|        4|      U2L|
+---------+---------+
```

In [11]:

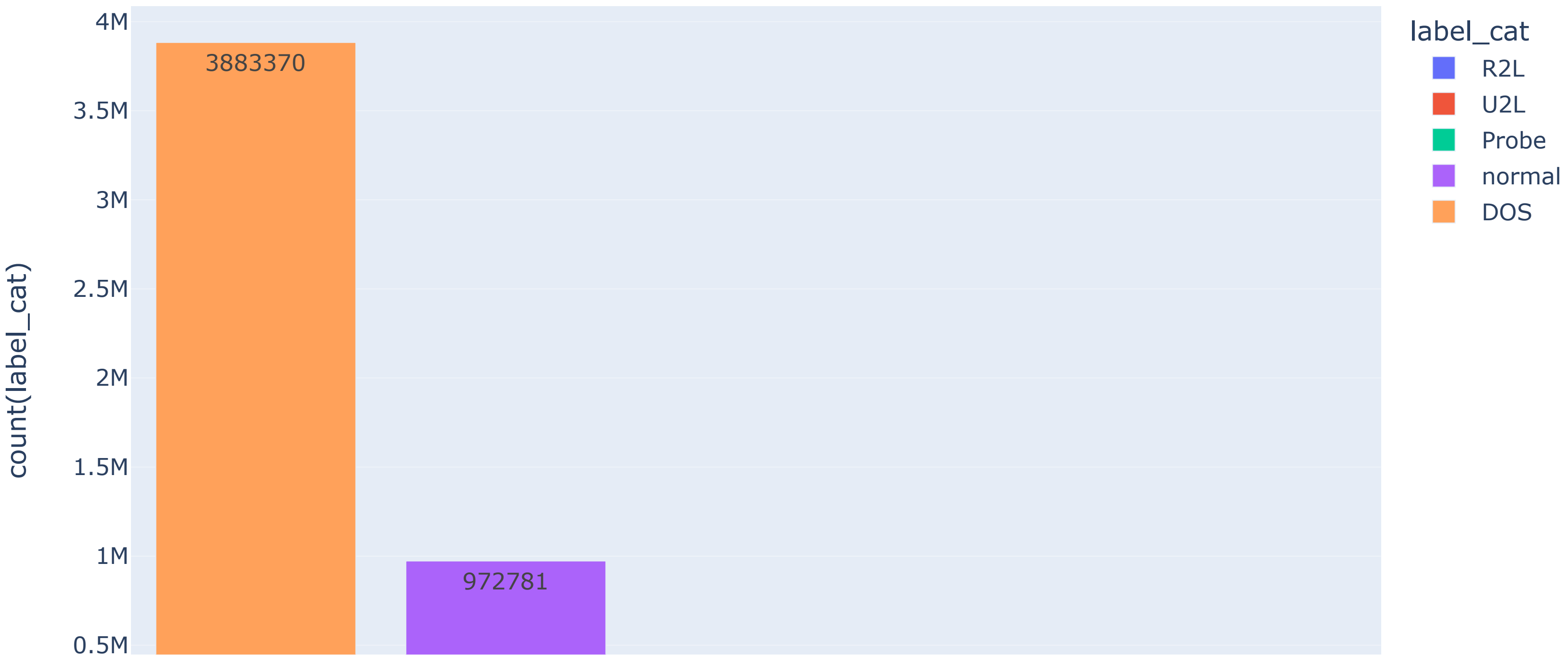
```
# Checking count of new label categories
df_sql.createOrReplaceTempView("label")

df_sql1 = spark.sql("""
SELECT label_cat, count(label_cat)
FROM label
group by label_cat
order by count(label_cat)
;
""")
#df_sql.select(['label_cat']).distinct().orderBy('label_num',ascending=True).show()
df_sql1.show()
```

```
+---------+----------------+
|label_cat|count(label_cat)|
+---------+----------------+
|      R2L|              98|
|      U2L|            1080|
|    Probe|           41102|
|   normal|          972781|
|      DOS|         3883370|
+---------+----------------+
```

Plotting Bar Chart for encoded label categories

In [12]:
```
psdf1 = df_sql1.to_pandas_on_spark()
psdf1.plot.bar(y = 'count(label_cat)', x = 'label_cat', color = 'label_cat', text = 'count(label_cat)').upda
```



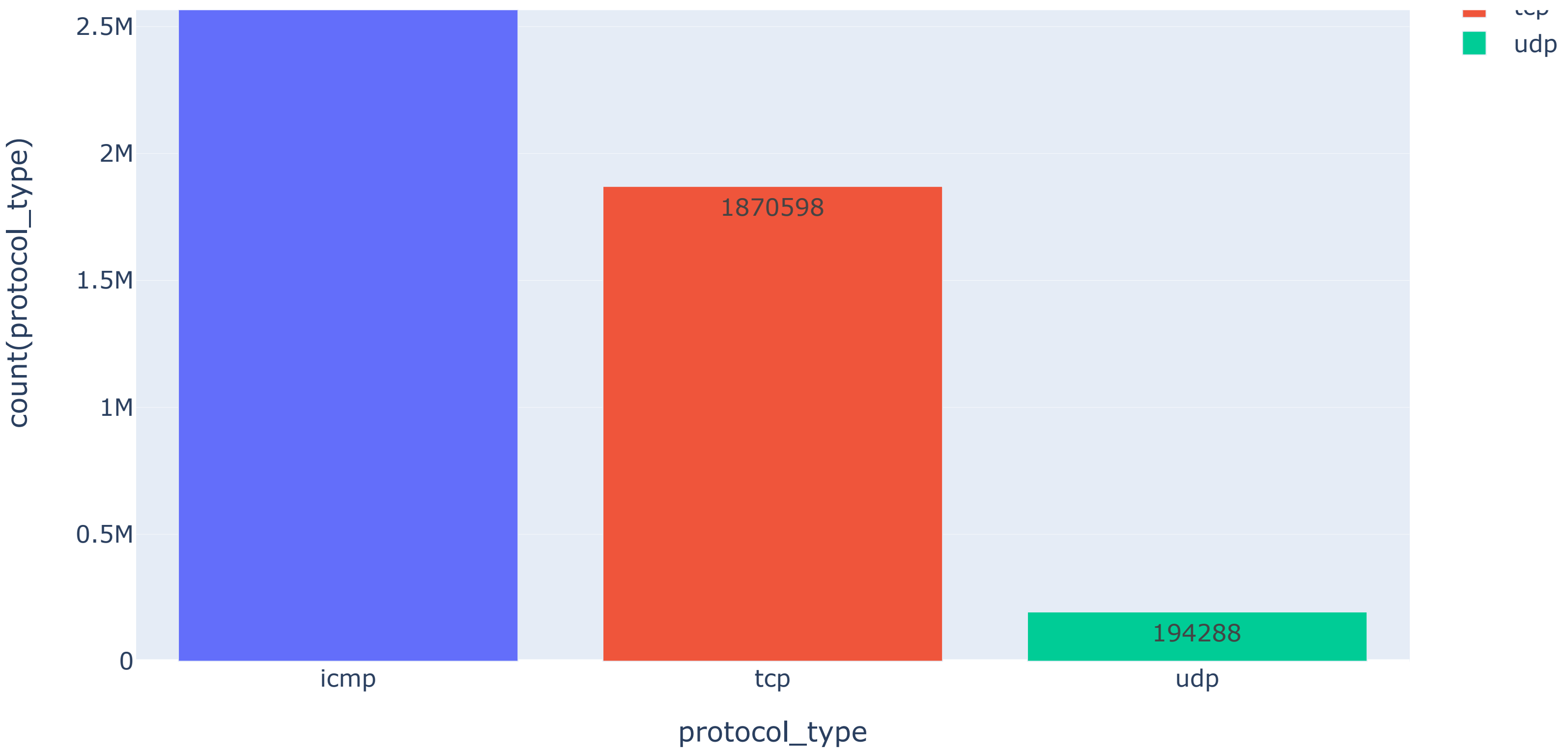
As we can see label of category 1-DOS is present in most of the data followed by normal category.

Plotting Bar Graph for different protocol types

In [13]:

```
### Plotting Bar chart for protocol types


df3 = df_sql.groupby('protocol_type').agg(F.count('protocol_type')).orderBy('protocol_type',
                                                                               ascending= True)
psdf2 = df3.to_pandas_on_spark()
psdf2.plot.bar(y = 'count(protocol_type)', x = 'protocol_type', color = 'protocol_type',
    text = 'count(protocol_type)')
```

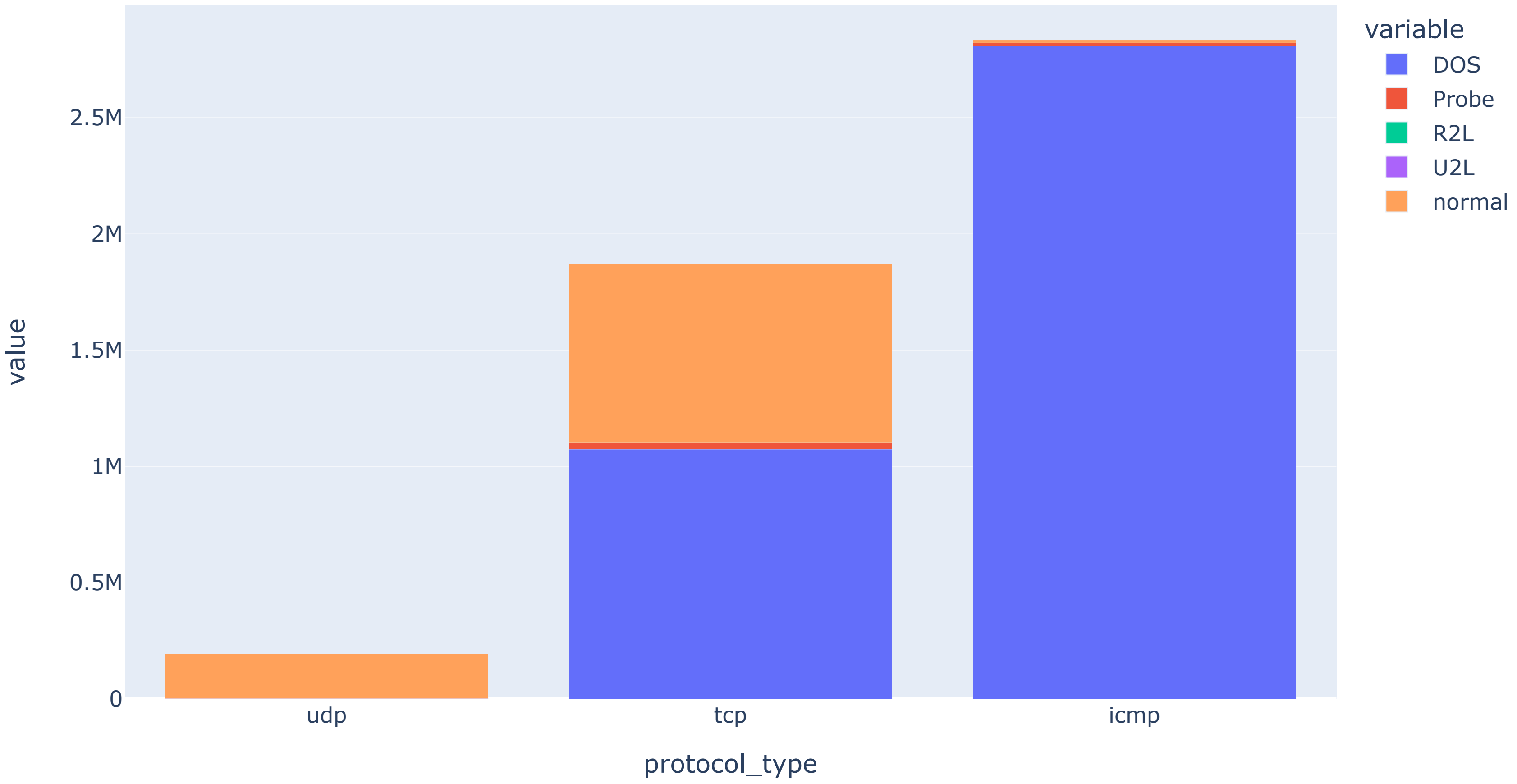


In [15]:

```
# Creating pivot table to check relationship between label and protocol type
df3 = df_sql.groupby('protocol_type').pivot('label_cat').agg(F.count('protocol_type')).orderBy('protocol_typ
psdf2 = df3.to_pandas_on_spark()
psdf2 = psdf2.fillna(0)
#psdf2.head()
df_pd = psdf2.set_index('protocol_type')
df_pd
```

	DOS	Probe	R2L	U2L	normal
protocol_type					
udp	979	1958	0	3	191348
tcp	1074241	26512	98	1077	768670
icmp	2808150	12632	0	0	12763

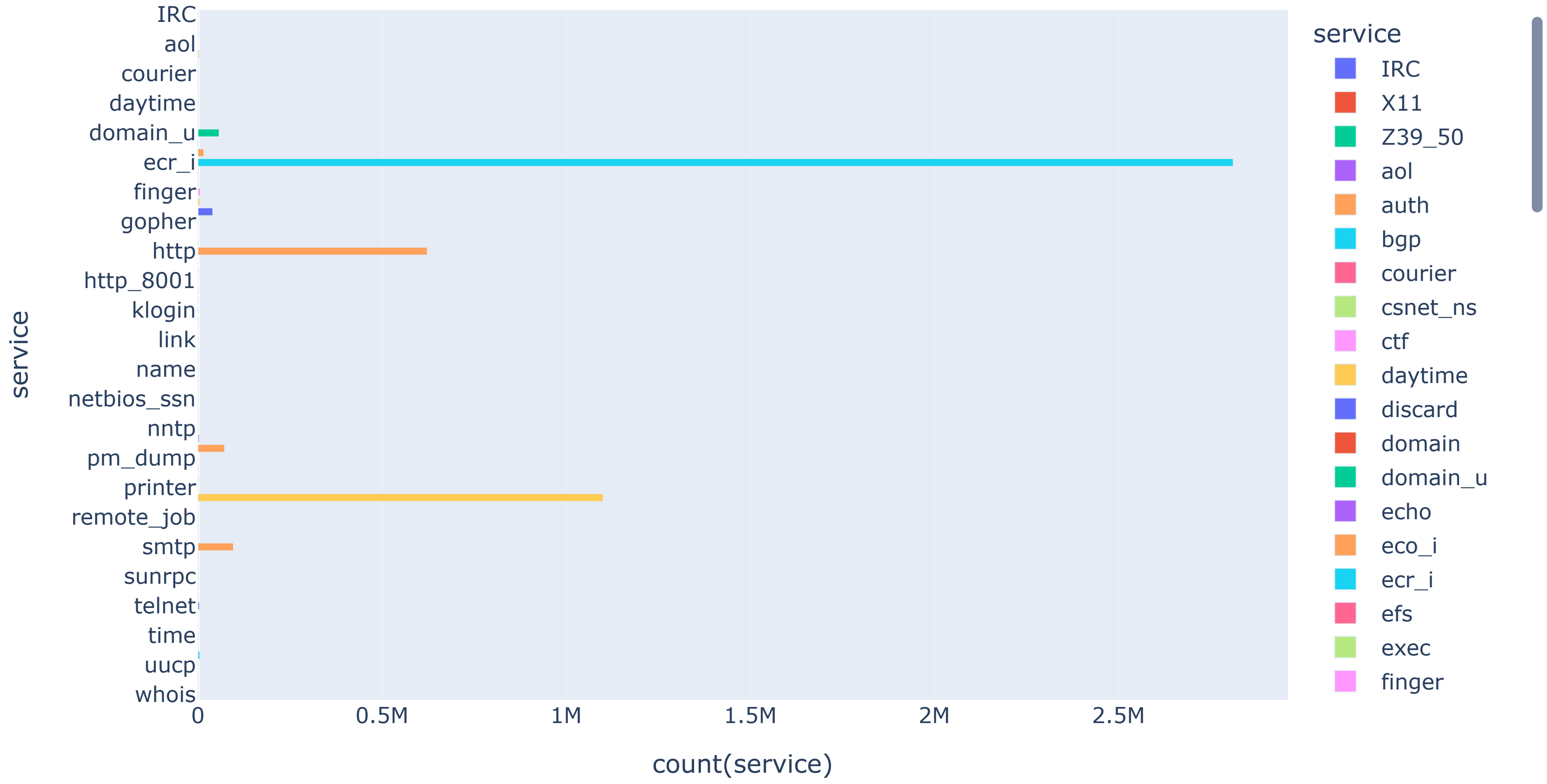
```
In [16]: # Plotting protocol type against our target variable
         df_pd.plot.bar(barmode = 'stack')
```



In [17]:

```
### Plotting Bar chart for service type


df3 = df_sql.groupby('service').agg(F.count('service')).orderBy('service', ascending= True)
psdf2 = df3.to_pandas_on_spark()
psdf2.plot.bar(y = 'service', x = 'count(service)', color = 'service')
```



In [18]:

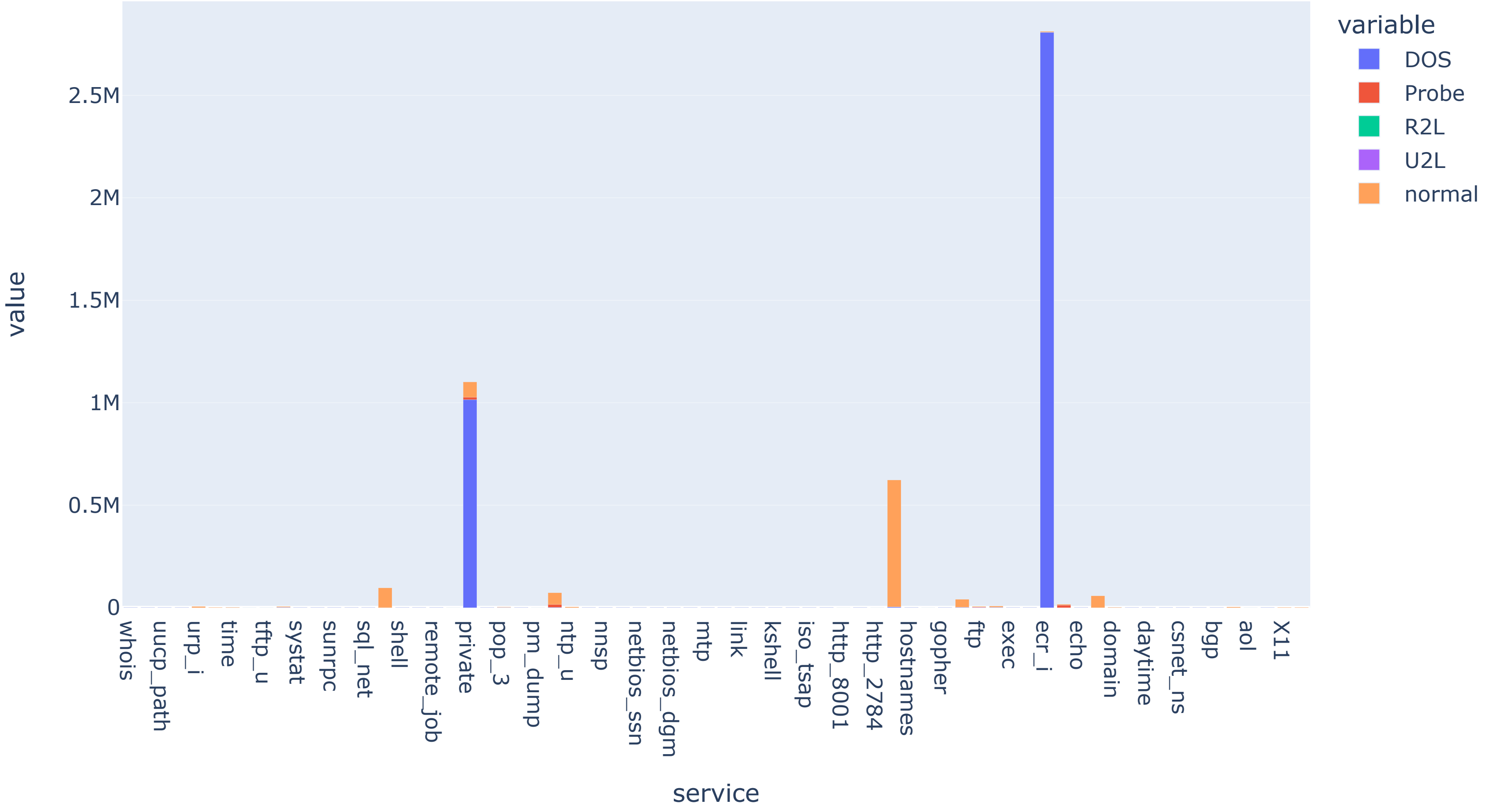
```
# Creating pivot table to check relationship between label and service

df3 = df_sql.groupby('service').pivot('label_cat').agg(F.count('service')).orderBy('service', ascending= Fa
psdf2 = df3.to_pandas_on_spark()
psdf2 = psdf2.fillna(0)
#psdf2.head()
df_pd = psdf2.set_index('service')
df_pd.head(3)
```

	DOS	Probe	R2L	U2L	normal
service					
whois	1042	31	0	0	0
vmnet	1041	12	0	0	0
uucp_path	1044	13	0	0	0

In [19]:

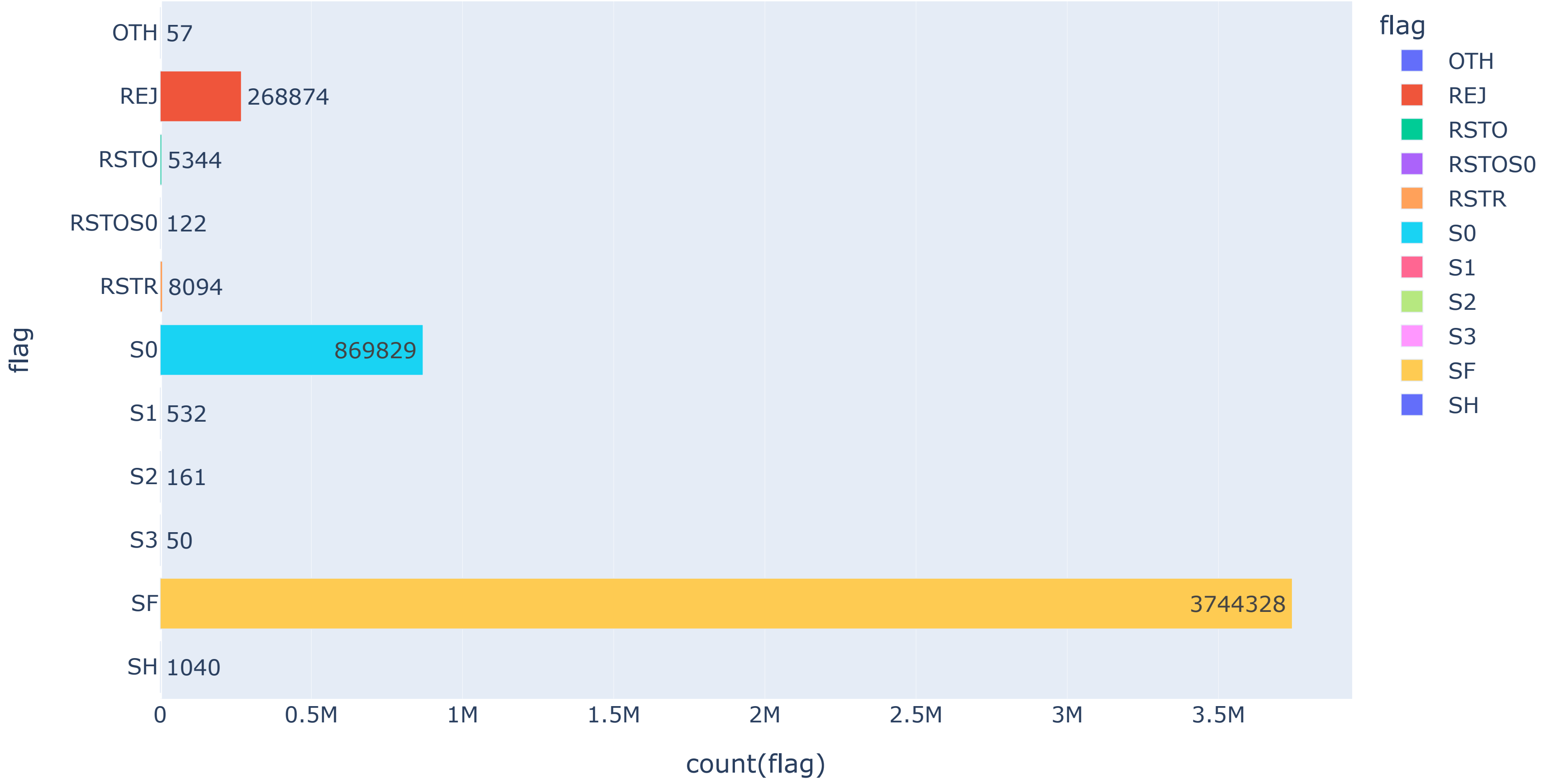
```
# Plotting service against our target variable
df_pd.plot.bar(barmode = 'stack')
```



In [21]:

```
### Plotting Bar chart for flag type


df3 = df_sql.groupby('flag').agg(F.count('flag')).orderBy('flag', ascending= True)
psdf2 = df3.to_pandas_on_spark()
psdf2.plot.bar(y = 'flag', x = 'count(flag)', color = 'flag', text = 'count(flag)')
```



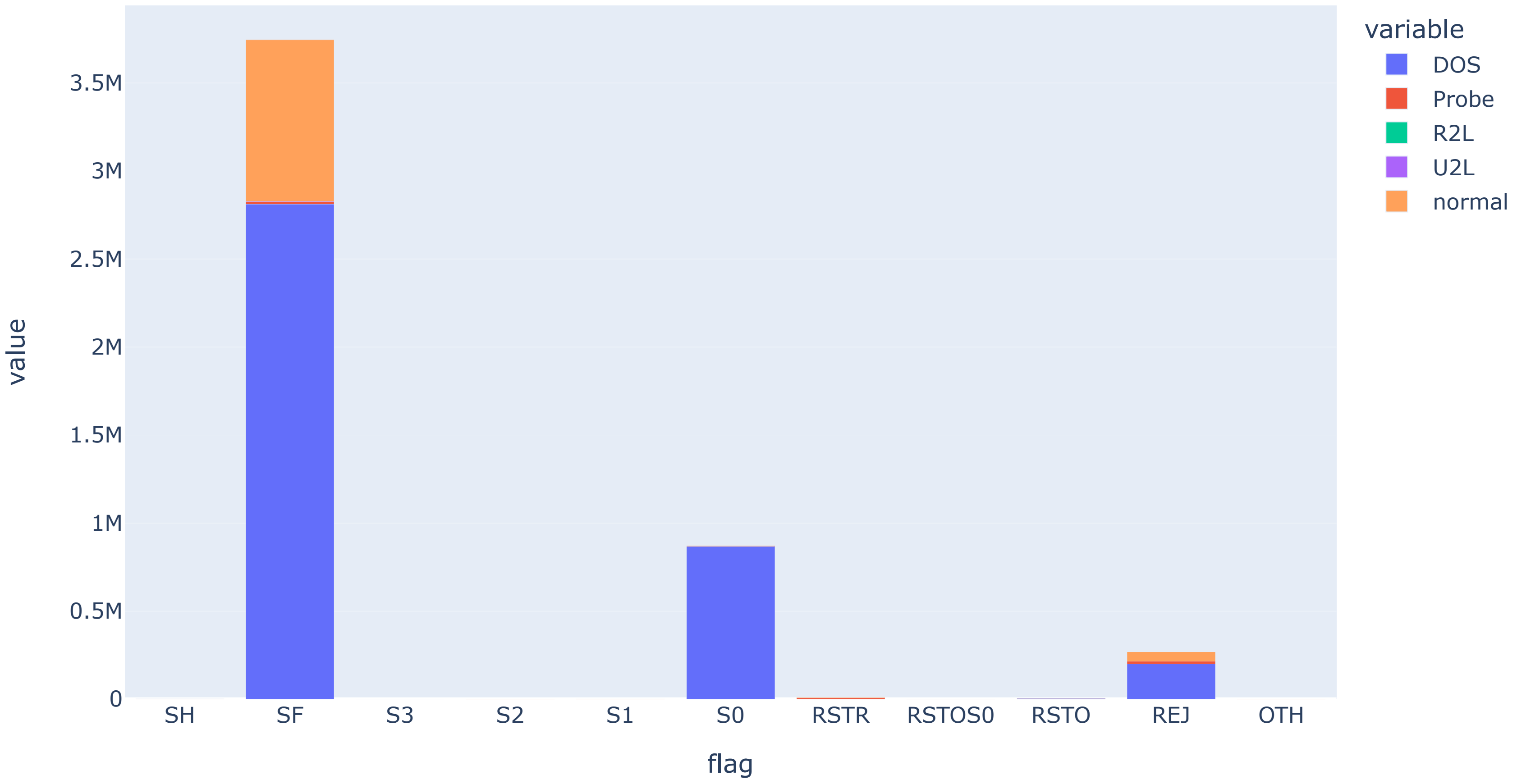
In [22]:

```
# Creating pivot table to check relationship between label and flag
df3 = df_sql.groupby('flag').pivot('label_cat').agg(F.count('flag')).orderBy('flag', ascending= False)
psdf2 = df3.to_pandas_on_spark()
psdf2 = psdf2.fillna(0)
#psdf2.head()
df_pd = psdf2.set_index('flag')
df_pd.head(3)
```

	DOS	**Probe**	**R2L**	**U2L**	**normal**
flag					
SH	0	1034	4	0	2
SF	2811235	14769	41	1075	917208
S3	0	1	2	1	46

In [23]:

```
# Plotting flag against our target variable
df_pd.plot.bar(barmode = 'stack')
```



Identifying relationships between target & numerical independent variables by comparing means

In [25]:

```
psdf4 = df_sql.to_pandas_on_spark()
psdf4.groupby('label_num').mean().T
```

label_num	4	2	1	0	3
duration	585.213889	590.519464	0.000074	217.824724	25.091837
src_bytes	283585.966667	109376.218870	707.446083	1477.846250	157.132653
dst_bytes	966.800000	51350.530655	4.670547	3234.650111	822842.908163
land	0.000000	0.000000	0.000005	0.000007	0.000000
wrong_fragment	0.000000	0.000000	0.000818	0.000000	0.000000
urgent	0.003704	0.000000	0.000000	0.000036	0.000000
hot	7.673148	0.000462	0.001114	0.049535	1.091837
num_failed_logins	0.000926	0.000097	0.000000	0.000099	0.571429
logged_in	0.992593	0.002141	0.000568	0.719268	0.132653
num_compromised	0.059259	0.000170	0.000548	0.038389	0.775510
root_shell	0.024074	0.000000	0.000000	0.000310	0.061224
su_attempted	0.000000	0.000000	0.000000	0.000184	0.010204
num_root	0.039815	0.000170	0.000000	0.064970	1.112245
num_file_creations	0.039815	0.000487	0.000000	0.005887	0.336735
num_shells	0.006481	0.000000	0.000000	0.000363	0.040816
num_access_files	0.003704	0.000000	0.000000	0.005131	0.071429
num_outbound_cmds	0.000000	0.000000	0.000000	0.000000	0.000000
is_host_login	0.000000	0.000000	0.000000	0.000002	0.000000
is_guest_login	0.286111	0.000024	0.000000	0.003882	0.051020
count	1.520370	171.770936	418.668688	8.159029	1.438776
srv_count	1.331481	7.349496	369.633827	10.912790	12.887755
serror_rate	0.004148	0.070904	0.223364	0.001483	0.105510
srv_serror_rate	0.002444	0.073539	0.223360	0.001725	0.095408
rerror_rate	0.002787	0.567238	0.052708	0.055941	0.500000
srv_rerror_rate	0.002778	0.565939	0.052737	0.056206	0.507959

label_num	4	2	1	0	3
same_srv_rate	0.993620	0.586935	0.743029	0.985257	1.000000
diff_srv_rate	0.009241	0.407238	0.017760	0.018535	0.000000
srv_diff_host_rate	0.011565	0.220565	0.000119	0.132494	0.046735
dst_host_count	82.085185	169.054182	254.867245	148.498415	44.816327
dst_host_srv_count	38.618519	52.952825	187.495878	202.014806	27.724490
dst_host_same_srv_rate	0.738676	0.318464	0.735483	0.844879	0.931327
dst_host_diff_srv_rate	0.021824	0.589753	0.018337	0.056500	0.001837
dst_host_same_src_port_rate	0.658407	0.589580	0.722970	0.134940	0.368163
dst_host_srv_diff_host_rate	0.098639	0.189274	0.000026	0.024340	0.010204
dst_host_serror_rate	0.010352	0.070218	0.223382	0.002039	0.133571
dst_host_srv_serror_rate	0.003537	0.073449	0.223338	0.001050	0.131429
dst_host_rerror_rate	0.005648	0.547672	0.052784	0.057784	0.477857
dst_host_srv_rerror_rate	0.001407	0.564952	0.052707	0.056016	0.475510

Observations:

- Duration of connection for attack is higher than normal except for DOS and R2L.
- Count of outbound commands in an ftp session are 0 for normal and attack connections.
- Wrong fragments in the connection is only present in DOS attack.

Correlation Matrix

In [28]:

```
# Creating Correlation Matrix
psdf_corr = df_sql.to_pandas_on_spark()
corrm=psdf_corr[numerical_variables].corr()
corrm
```

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_log
duration	1.000000	4.122055e-02	0.020392	-0.000160	-0.001012	3.765465e-03	0.004450	0.007412
src_bytes	0.041221	1.000000e+00	0.000239	-0.000005	-0.000027	-8.677487e-08	0.000782	-0.000007
dst_bytes	0.020392	2.393376e-04	1.000000	-0.000004	-0.000026	1.645208e-04	0.000126	0.000632
land	-0.000160	-4.659182e-06	-0.000004	1.000000	-0.000036	-2.638263e-06	-0.000063	-0.000010
wrong_fragment	-0.001012	-2.714937e-05	-0.000026	-0.000036	1.000000	-1.670585e-05	-0.000402	-0.000066
urgent	0.003765	-8.677487e-08	0.000165	-0.000003	-0.000017	1.000000e+00	0.003591	0.031005
hot	0.004450	7.822053e-04	0.000126	-0.000063	-0.000402	3.590666e-03	1.000000	0.004475
num_failed_logins	0.007412	-6.921906e-06	0.000632	-0.000010	-0.000066	3.100532e-02	0.004475	1.000000
logged_in	-0.020624	1.998519e-04	0.002119	-0.000979	-0.006197	2.534183e-03	0.064579	0.001792
num_compromised	0.027126	4.832481e-06	0.001307	-0.000005	-0.000032	1.767953e-02	0.002688	0.019542
root_shell	0.026378	-5.620801e-06	0.000988	-0.000020	-0.000125	8.908420e-02	0.017916	0.023673
su_attempted	0.052088	-4.122588e-	0.001204	-0.000011	-0.000069	1.330230e-01	0.001926	0.069186

Creating a Pipeline to encode the qualitative variables which are of String datatype namely protocol_type, service, flag and label into variables of datatype Double by using StringIndexer Transformers.

This step is important as Logistic Regression and NaiveBayes classifiers needs to be trained on numerical values.

https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html

(https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html)

In [26]:

```
# Defining Categorical Variables
qualitative_variables = ["protocol_type", "service", "flag"]

# Using String Indexer to encode Categorical variables to Numerical Variables
indexers = [StringIndexer(inputCol=column, outputCol=column + "_num") for column in qualitative_variables]

#Creating a pipeline
pipeline = Pipeline(stages=indexers)
#Transforming our spark dataframe
df_sql = pipeline.fit(df_sql).transform(df_sql)

# Excluding Non-Numeric Attributes
not_needed = qualitative_variables + ["label", "label_num","label_cat"]
#print(not_needed)
```

In [27]:
```
numerical_variables = [col for col in df_sql.columns if col not in not_needed]
print(numerical_variables)
df_sql.printSchema()
```

```
['duration', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'hot', 'num_failed_logins', 'logged_in', 'num_compromised',
'root_shell', 'su_attempted', 'num_root', 'num_file_creations', 'num_shells', 'num_access_files', 'num_outbound_cmds', 'is_host_login',
'is_guest_login', 'count', 'srv_count', 'serror_rate', 'srv_serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_r
ate', 'srv_diff_host_rate', 'dst_host_count', 'dst_host_srv_count', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate', 'dst_host_same_s
rc_port_rate', 'dst_host_srv_diff_host_rate', 'dst_host_serror_rate', 'dst_host_srv_serror_rate', 'dst_host_rerror_rate', 'dst_host_srv_
rerror_rate', 'protocol_type_num', 'service_num', 'flag_num']
root
 |-- duration: integer (nullable = true)
 |-- protocol_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- flag: string (nullable = true)
 |-- src_bytes: integer (nullable = true)
 |-- dst_bytes: integer (nullable = true)
 |-- land: integer (nullable = true)
 |-- wrong_fragment: integer (nullable = true)
 |-- urgent: integer (nullable = true)
 |-- hot: integer (nullable = true)
 |-- num_failed_logins: integer (nullable = true)
 |-- logged_in: integer (nullable = true)
 |-- num_compromised: integer (nullable = true)
 |-- root_shell: integer (nullable = true)
 |-- su_attempted: integer (nullable = true)
 |-- num_root: integer (nullable = true)
 |-- num_file_creations: integer (nullable = true)
 |-- num_shells: integer (nullable = true)
 |-- num_access_files: integer (nullable = true)
 |-- num_outbound_cmds: integer (nullable = true)
 |-- is_host_login: integer (nullable = true)
 |-- is_guest_login: integer (nullable = true)
 |-- count: integer (nullable = true)
 |-- srv_count: integer (nullable = true)
 |-- serror_rate: double (nullable = true)
 |-- srv_serror_rate: double (nullable = true)
 |-- rerror_rate: double (nullable = true)
 |-- srv_rerror_rate: double (nullable = true)
 |-- same_srv_rate: double (nullable = true)
 |-- diff_srv_rate: double (nullable = true)
 |-- srv_diff_host_rate: double (nullable = true)
 |-- dst_host_count: integer (nullable = true)
 |-- dst_host_srv_count: integer (nullable = true)
 |-- dst_host_same_srv_rate: double (nullable = true)
 |-- dst_host_diff_srv_rate: double (nullable = true)
 |-- dst_host_same_src_port_rate: double (nullable = true)
 |-- dst_host_srv_diff_host_rate: double (nullable = true)
 |-- dst_host_serror_rate: double (nullable = true)
 |-- dst_host_srv_serror_rate: double (nullable = true)
 |-- dst_host_rerror_rate: double (nullable = true)
 |-- dst_host_srv_rerror_rate: double (nullable = true)
 |-- label: string (nullable = true)
 |-- label_num: integer (nullable = false)
 |-- label_cat: string (nullable = false)
 |-- protocol_type_num: double (nullable = false)
 |-- service_num: double (nullable = false)
 |-- flag_num: double (nullable = false)
```

Now we will use Spark's Vector Assembler to create a feature vector for each numerical variable as we need to combine all the input columns into a single vector which would essentially act as the input feature for the classifiers.

In [29]:

```
df_assembler = VectorAssembler(inputCols=numerical_variables, outputCol="features")
df_sql = df_assembler.transform(df_sql)
#df_sql.printSchema()
```

We can see a features column has been created in the dataframe schema

Now we will create a new dataframe *ml_df* containing two columns - features and label_num. Furthermore, we will randomly split it into the training dataset and testing dataset in 75:25 ratio respectively.

https://medium.com/udemy-engineering/pyspark-under-the-hood-randomsplit-and-sample-inconsistencies-examined-7c6ec62644bc (https://medium.com/udemy-engineering/pyspark-under-the-hood-randomsplit-and-sample-inconsistencies-examined-7c6ec62644bc) - The signature function of randomSplit() includes a weight list and a seed specification. The weight list is to specify the number of splits and percentage (approximate) in each and the seed is for reproducibility. The ratio is approximate due to the nature of how it is calculated.

In [30]:

```
# Selecting vectorized features column and label_num
ml_df = df_sql.select(["features","label_num"])
ml_df.printSchema()

train_set, test_set = ml_df.randomSplit([0.75, 0.25], seed=3000)
print("Training dataset count: " + str(train_set.count()))
print("Test dataset count: " + str(test_set.count()))
```

```
root
 |-- features: vector (nullable = true)
 |-- label_num: integer (nullable = false)

Training dataset count: 3675127
Test dataset count: 1223304
```

In [31]:

```
# Selecting vectorized features column and label_num
ml_df = df_sql.select(["features","label_num"])
ml_df.printSchema()

train_set, test_set = ml_df.randomSplit([0.75, 0.25], seed=2000)
print("Training dataset count: " + str(train_set.count()))
print("Test dataset count: " + str(test_set.count()))
```

```
root
 |-- features: vector (nullable = true)
 |-- label_num: integer (nullable = false)

Training dataset count: 3672779
Test dataset count: 1225652
```

From above two blocks of code we can verify the random split by checking the dataset shape

Building Machine Learing Models

KPI(Key Performance Indicator)

- Accuracy
- Weighted Precision
- Weighted Recall
- F1 Score

In [35]:

```
metrics = ["accuracy", "weightedPrecision", "weightedRecall", "f1"]
```

Evaluation

Logistic Regression

In [88]:

```
# Logistic Regression model
lr = LogisticRegression(maxIter=20, regParam=0.3, elasticNetParam=0.8, featuresCol="features",
                        labelCol="label_num", family="multinomial")

lrlist = []

print("\nLogistic Regression Model Evaluation:")
print("-"*30)
lr_model = lr.fit(train_set)

# make predictions
predictions = lr_model.transform(test_set)
predictions.cache()

evaluator = MulticlassClassificationEvaluator(labelCol="label_num", predictionCol="prediction")

for m in metrics:
    evaluator.setMetricName(m)
    metric = evaluator.evaluate(predictions)
    print("{name} = {value:.2f}".format(name=m, value=metric))
    lrlist.append(metric)

predictions.show(5)
```

```
Logistic Regression Model Evaluation:
------------------------------
accuracy = 0.79
weightedPrecision = 0.63
weightedRecall = 0.79
f1 = 0.70
```

```
+--------------------+---------+--------------------+--------------------+----------+
|            features|label_num|       rawPrediction|         probability|prediction|
+--------------------+---------+--------------------+--------------------+----------+
|(41,[0,1,2,6,8,10...|        0|[4.04992757798011...|[0.38493120112223...|       1.0|
|(41,[0,1,2,6,8,18...|        0|[4.04952262397637...|[0.38452734500661...|       1.0|
|(41,[0,1,2,6,8,18...|        0|[4.04972510097824...|[0.38472925408982...|       1.0|
|(41,[0,1,2,6,8,18...|        0|[4.04972510097824...|[0.38472925408982...|       1.0|
|(41,[0,1,2,6,8,18...|        0|[4.04972510097824...|[0.38472925408982...|       1.0|
+--------------------+---------+--------------------+--------------------+----------+
only showing top 5 rows
```

Decision Tree

In [89]:

```
# Decision Tree model
dt = DecisionTreeClassifier(labelCol="label_num", featuresCol="features",  maxBins=70)

dtlist = []

print("\nDecision Tree Model Evaluation:")
print("{:-<30}".format(""))
dt_model = dt.fit(train_set)

# make predictions
predictions = dt_model.transform(test_set)
predictions.cache()

evaluator = MulticlassClassificationEvaluator(labelCol="label_num", predictionCol="prediction")

for m in metrics:
    evaluator.setMetricName(m)
    metric = evaluator.evaluate(predictions)
    print("{name} = {value:.2f}".format(name=m, value=metric))
    dtlist.append(metric)

predictions.show(5)
```

```
Decision Tree Model Evaluation:
------------------------------
accuracy = 1.00
weightedPrecision = 1.00
weightedRecall = 1.00
f1 = 1.00
+--------------------+---------+--------------------+--------------------+----------+
|            features|label_num|       rawPrediction|         probability|prediction|
+--------------------+---------+--------------------+--------------------+----------+
|(41,[0,1,2,6,8,10...|        0|[708161.0,706.0,4...|[0.99177212353438...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[708161.0,706.0,4...|[0.99177212353438...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[708161.0,706.0,4...|[0.99177212353438...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[708161.0,706.0,4...|[0.99177212353438...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[708161.0,706.0,4...|[0.99177212353438...|       0.0|
+--------------------+---------+--------------------+--------------------+----------+
only showing top 5 rows
```

Random Forest Classifiier

In [90]:

```
# Random Forest model
rf = RandomForestClassifier(labelCol="label_num", featuresCol="features", numTrees=20, maxBins=70)

rflist = []

print("\nRandom Forest Classifier Model Evaluation:")
print("{:-<30}".format(""))
rf_model = rf.fit(train_set)

# make predictions
predictions = rf_model.transform(test_set)
predictions.cache()

evaluator = MulticlassClassificationEvaluator(labelCol="label_num", predictionCol="prediction")

for m in metrics:
    evaluator.setMetricName(m)
    metric = evaluator.evaluate(predictions)
    print("{name} = {value:.2f}".format(name=m, value=metric))
    rflist.append(metric)

predictions.show(5)
```

```

Random Forest Classifier Model Evaluation:
------------------------------
accuracy = 1.00
weightedPrecision = 1.00
weightedRecall = 1.00
f1 = 1.00
```

```
+--------------------+---------+--------------------+--------------------+----------+
|            features|label_num|       rawPrediction|         probability|prediction|
+--------------------+---------+--------------------+--------------------+----------+
|(41,[0,1,2,6,8,10...|        0|[16.9687825289452...|[0.84843912644726...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[19.6540507623080...|[0.98270253811540...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[19.5035822798782...|[0.97517911399391...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[19.6420975918839...|[0.98210487959419...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[19.6520642213605...|[0.98260321106802...|       0.0|
+--------------------+---------+--------------------+--------------------+----------+
only showing top 5 rows
```

Naive Bayes Multinomial

In [91]:

```
# Naive Bayes Multinomial
nb = NaiveBayes(labelCol="label_num", featuresCol="features", smoothing=1.0, modelType="multinomial")

nblist = []

print("\nNaive Bayes Multinomial Model Evaluation:")
print("{:-<30}".format(""))
nb_model = nb.fit(train_set)

# make predictions
predictions = nb_model.transform(test_set)
predictions.cache()

evaluator = MulticlassClassificationEvaluator(labelCol="label_num", predictionCol="prediction")

for m in metrics:
    evaluator.setMetricName(m)
    metric = evaluator.evaluate(predictions)
    print("{name} = {value:.2f}".format(name=m, value=metric))
    nblist.append(metric)

predictions.show(5)
```

```
Naive Bayes Multinomial Model Evaluation:
------------------------------
accuracy = 0.93
weightedPrecision = 0.95
weightedRecall = 0.93
f1 = 0.93
+--------------------+---------+--------------------+--------------------+----------+
|            features|label_num|       rawPrediction|         probability|prediction|
+--------------------+---------+--------------------+--------------------+----------+
|(41,[0,1,2,6,8,10...|        0|[-384791.27476208...|[0.0,0.0,0.0,1.0,...|       3.0|
|(41,[0,1,2,6,8,18...|        0|[-1136.0742561634...|[1.0,0.0,2.097648...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[-4548.3207874904...|[1.0,0.0,0.0,0.0,...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[-1706.7583981789...|[1.0,0.0,0.0,0.0,...|       0.0|
|(41,[0,1,2,6,8,18...|        0|[-1325.6372574940...|[1.0,0.0,3.609994...|       0.0|
+--------------------+---------+--------------------+--------------------+----------+
only showing top 5 rows
```

Comparing our models

In [92]:

```
l = [round(item, 2) for item in lrlist]
l.insert(0, 'Logistic Regression')
d = [round(item, 2) for item in dtlist]
d.insert(0, 'Decision Tree')
r = [round(item, 2) for item in rflist]
r.insert(0, 'Random Forest Classifier')
n = [round(item, 2) for item in nblist]
n.insert(0, 'Naive Bayes')

data = [l,d,r,n]

#giving column names of dataframe
columns = ["Models","Accuracy", "WeightedPrecision", "WeightedRecall","F1"]
#creating a dataframe
dataframe = spark.createDataFrame(data, columns)

#show data frame
dataframe.show()
```

```
+--------------------+--------+-----------------+--------------+----+
|              Models|Accuracy|WeightedPrecision|WeightedRecall|  F1|
+--------------------+--------+-----------------+--------------+----+
| Logistic Regression|    0.79|             0.63|          0.79| 0.7|
|       Decision Tree|     1.0|              1.0|           1.0| 1.0|
|Random Forest Cla...|     1.0|              1.0|           1.0| 1.0|
|         Naive Bayes|    0.93|             0.95|          0.93|0.93|
+--------------------+--------+-----------------+--------------+----+
```

In [93]:

```
psdf_results = dataframe.to_pandas_on_spark()
psdf_results = psdf_results.set_index('Models')
psdf_results.head()
```

	Accuracy	WeightedPrecision	WeightedRecall	F1
Models				
Logistic Regression	0.79	0.63	0.79	0.70
Decision Tree	1.00	1.00	1.00	1.00
Random Forest Classifier	1.00	1.00	1.00	1.00
Naive Bayes	0.93	0.95	0.93	0.93

In [94]:

```
psdf_results.plot.bar(barmode='group').update_xaxes(categoryorder="total ascending")
```

Saving our model file

Saving Decision Tree Model file using pickle as it has the highest accuracy. https://suziepyspark.blogspot.com/2021/03/using-pickle-to-save-model-in-incorta.html (https://suziepyspark.blogspot.com/2021/03/using-pickle-to-save-model-in-incorta.html)

In [380]:

```
#!pip install prophet
from pyspark.sql.functions import *
from pyspark.sql.types import *
from prophet import Prophet
import pickle

import pickle
pkl_path = "model.pkl"
with open(pkl_path, "wb") as f:
    #Saving our decision tree model
    pickle.dump(dt_model, f)

# save the model
print("*** Data Saved ***")
```

```
*** Data Saved ***
```

In [384]:

```
import pickle
model=pickle.load(open('model.pkl', 'rb'))

# Parameters for DOS
model.predict([[1,0,0,1000,0.06,0.00,0.04,10,0,0,21,0,0.04,0.00,0]])
```

```
array([1.])
```

```
In [382]: # Parameters for probe
          model.predict([[0,0,0,0,0.06,0.00,0.04,10,0,0,21,0,0.04,0.00,0]])
```

```
array([2.])
```

References

- https://sparkbyexamples.com/pyspark/different-ways-to-create-dataframe-in-pyspark/
- https://github.com/biagiom/spark-network-traffic-classifier
- https://unsplash.com/
- https://www.barracuda.com/glossary/intrusion-detection-system
- https://www.ecb.torontomu.ca/~bagheri/papers/cisda.pdf