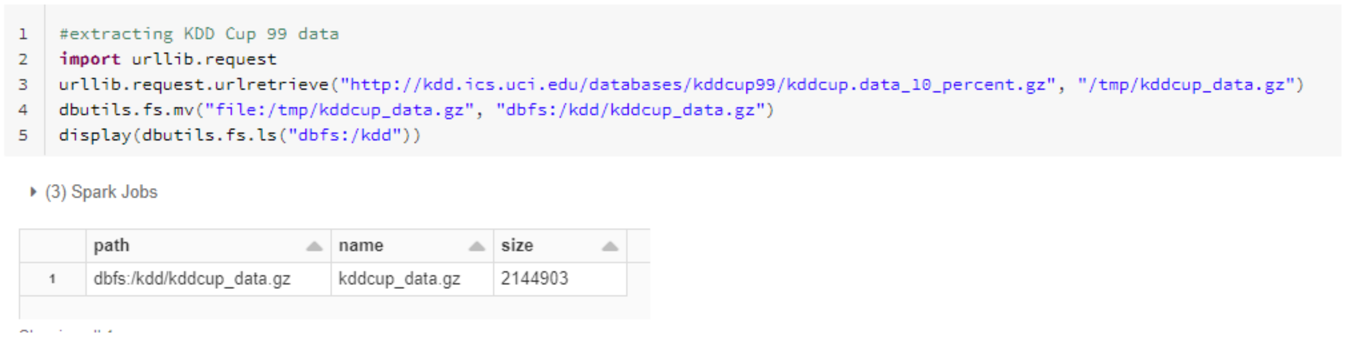
All result code and output can be seen either in attached jupyter notebook or at the end of this PDF.

1. Extracting Data

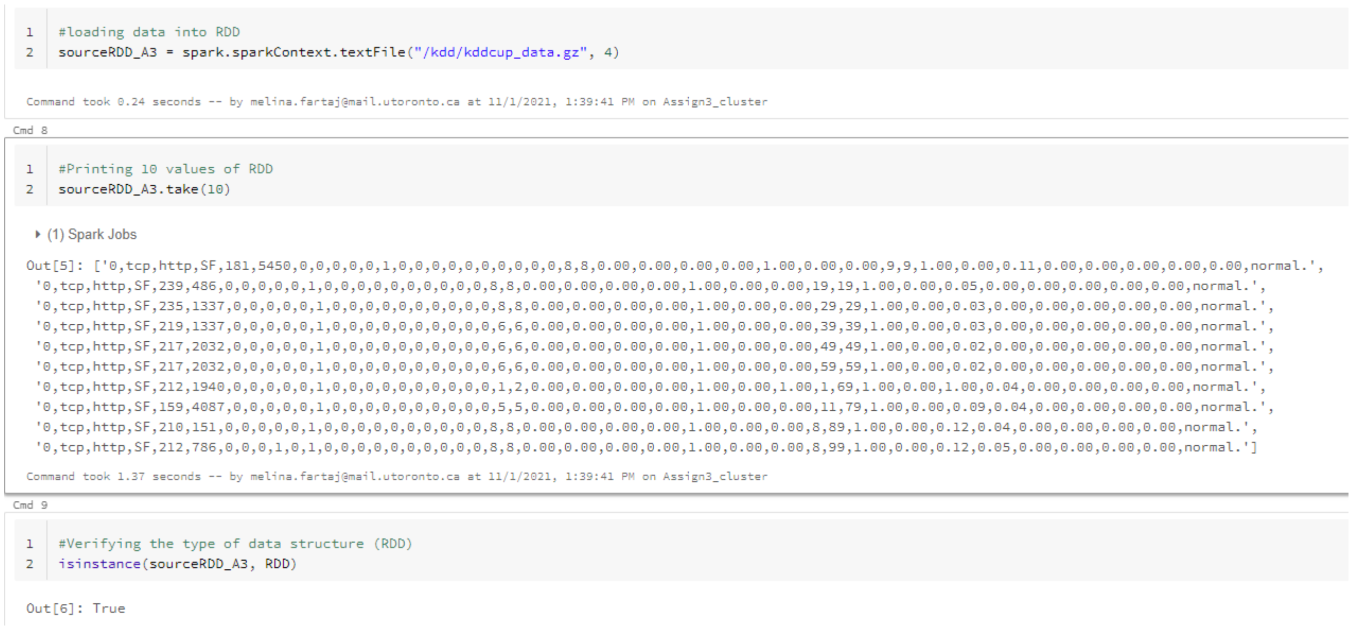


```
#extracting KDD Cup 99 data
import urllib.request
urllib.request.urlretrieve("http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz", "/tmp/kddcup_data.gz")
dbutils.fs.mv("file:/tmp/kddcup_data.gz", "dbfs:/kdd/kddcup_data.gz")
display(dbutils.fs.ls("dbfs:/kdd"))
```

▸ (3) Spark Jobs

	path	name	size
1	dbfs:/kdd/kddcup_data.gz	kddcup_data.gz	2144903

2. RDD values and type of data



```
#loading data into RDD
sourceRDD_A3 = spark.sparkContext.textFile("/kdd/kddcup_data.gz", 4)
```

```
Command took 0.24 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:41 PM on Assign3_cluster
```

Cmd 8

```
#Printing 10 values of RDD
sourceRDD_A3.take(10)
```

▸ (1) Spark Jobs

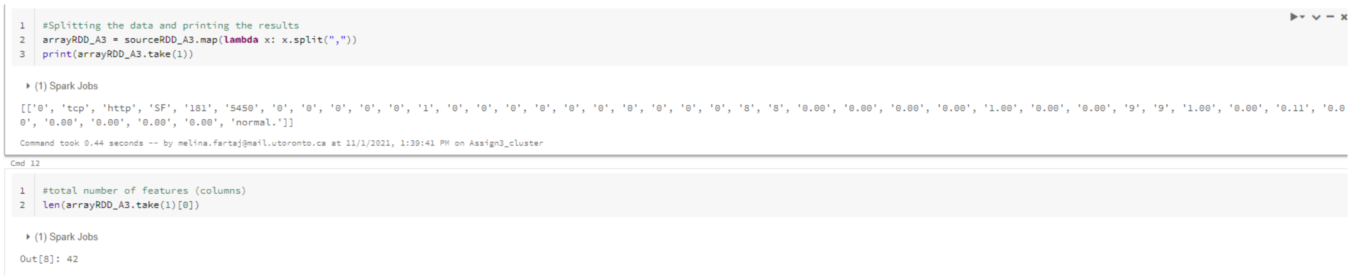
```
Out[5]: ['0,tcp,http,SF,181,5450,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,9,9,1.00,0.00,0.11,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,239,486,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,19,19,1.00,0.00,0.05,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,235,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,29,29,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,219,1337,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,39,39,1.00,0.00,0.03,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,49,49,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,217,2032,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,6,6,0.00,0.00,0.00,0.00,1.00,0.00,0.00,59,59,1.00,0.00,0.02,0.00,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,212,1940,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0.00,0.00,0.00,0.00,1.00,0.00,1.00,1,69,1.00,0.00,1.00,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,159,4087,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,5,0.00,0.00,0.00,0.00,1.00,0.00,0.00,11,79,1.00,0.00,0.09,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,210,151,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,89,1.00,0.00,0.12,0.04,0.00,0.00,0.00,0.00,normal.',
 '0,tcp,http,SF,212,786,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0.00,0.00,0.00,0.00,1.00,0.00,0.00,8,99,1.00,0.00,0.12,0.05,0.00,0.00,0.00,0.00,normal.']
Command took 1.37 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:41 PM on Assign3_cluster
```

Cmd 9

```
#Verifying the type of data structure (RDD)
isinstance(sourceRDD_A3, RDD)
```

```
Out[6]: True
```

3. Splitting the data



```
#Splitting the data and printing the results
arrayRDD_A3 = sourceRDD_A3.map(lambda x: x.split(","))
print(arrayRDD_A3.take(1))
```

▸ (1) Spark Jobs

```
[['0', 'tcp', 'http', 'SF', '181', '5450', '0', '0', '0', '0', '0', '1', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '8', '8', '0.00', '0.00', '0.00', '0.00', '1.00', '0.00', '0.00', '9', '9', '1.00', '0.00', '0.11', '0.0
0', '0.00', '0.00', '0.00', '0.00', 'normal.']]
Command took 0.44 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:41 PM on Assign3_cluster
```

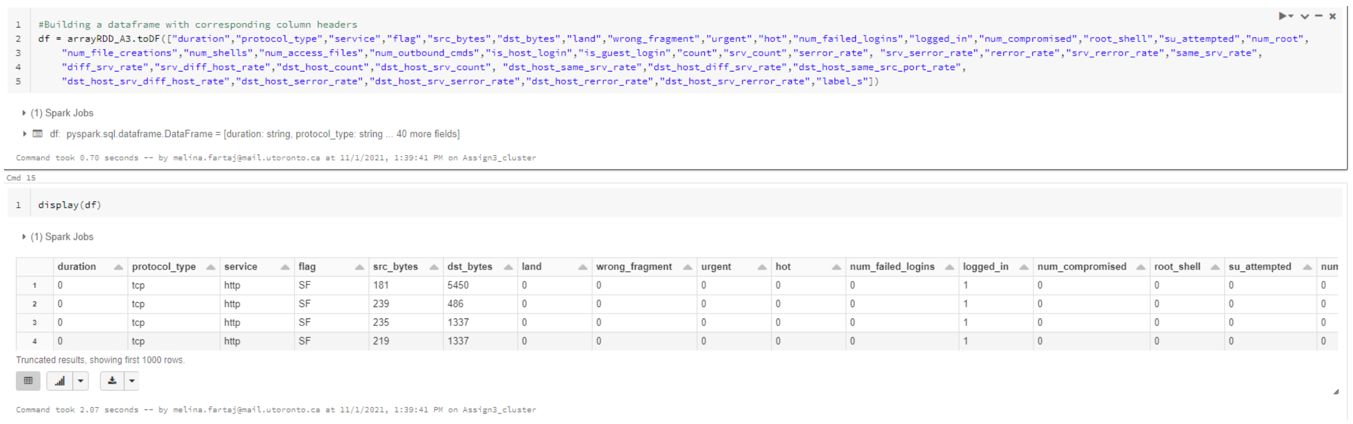
Cmd 12

```
#total number of features (columns)
len(arrayRDD_A3.take(1)[0])
```

▸ (1) Spark Jobs

```
Out[8]: 42
```

4. New Dataframe and RDD



```
#Building a dataframe with corresponding column headers
df = arrayRDD_A3.toDF(["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_s"])
```

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [duration: string, protocol_type: string ... 40 more fields]

```
Command took 0.70 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:41 PM on Assign3_cluster
```

```
Cmd 15
```

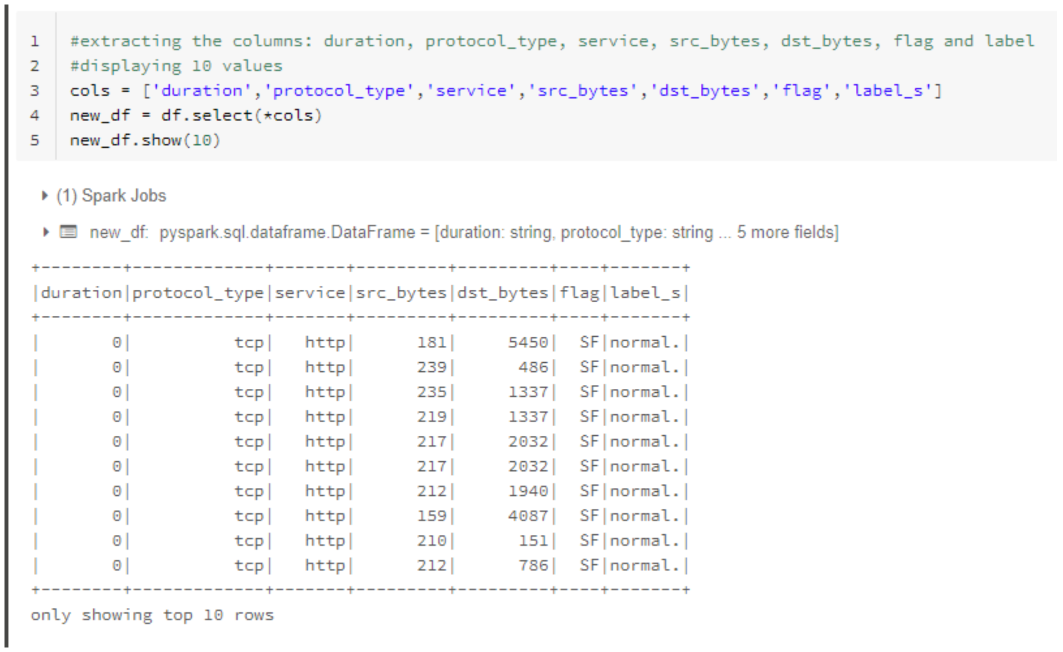
```
display(df)
```

(1) Spark Jobs

	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
1	0	tcp	http	SF	181	5450	0	0	0	0	0	1	0	0	0	0
2	0	tcp	http	SF	239	486	0	0	0	0	0	1	0	0	0	0
3	0	tcp	http	SF	235	1337	0	0	0	0	0	1	0	0	0	0
4	0	tcp	http	SF	219	1337	0	0	0	0	0	1	0	0	0	0

Truncated results, showing first 1000 rows.

```
Command took 2.07 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:41 PM on Assign3_cluster
```

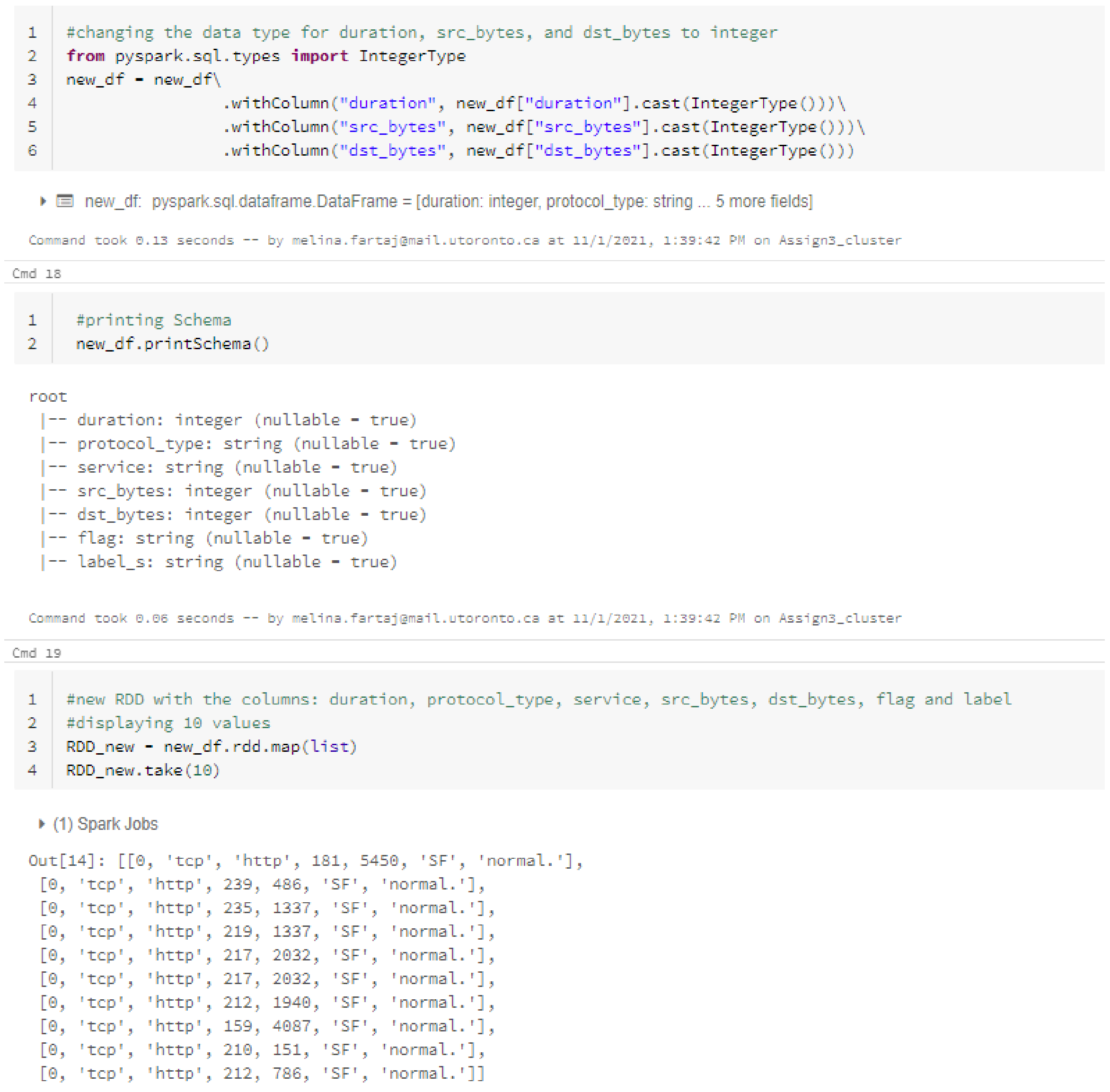


```
#extracting the columns: duration, protocol_type, service, src_bytes, dst_bytes, flag and label
#displaying 10 values
cols = ['duration','protocol_type','service','src_bytes','dst_bytes','flag','label_s']
new_df = df.select(*cols)
new_df.show(10)
```

(1) Spark Jobs

new_df: pyspark.sql.dataframe.DataFrame = [duration: string, protocol_type: string ... 5 more fields]

```
+--------+-------------+-------+---------+---------+----+-------+
|duration|protocol_type|service|src_bytes|dst_bytes|flag|label_s|
+--------+-------------+-------+---------+---------+----+-------+
|       0|          tcp|   http|      181|     5450|  SF|normal.|
|       0|          tcp|   http|      239|      486|  SF|normal.|
|       0|          tcp|   http|      235|     1337|  SF|normal.|
|       0|          tcp|   http|      219|     1337|  SF|normal.|
|       0|          tcp|   http|      217|     2032|  SF|normal.|
|       0|          tcp|   http|      217|     2032|  SF|normal.|
|       0|          tcp|   http|      212|     1940|  SF|normal.|
|       0|          tcp|   http|      159|     4087|  SF|normal.|
|       0|          tcp|   http|      210|      151|  SF|normal.|
|       0|          tcp|   http|      212|      786|  SF|normal.|
+--------+-------------+-------+---------+---------+----+-------+
only showing top 10 rows
```



```
#changing the data type for duration, src_bytes, and dst_bytes to integer
from pyspark.sql.types import IntegerType
new_df = new_df\
                .withColumn("duration", new_df["duration"].cast(IntegerType()))\
                .withColumn("src_bytes", new_df["src_bytes"].cast(IntegerType()))\
                .withColumn("dst_bytes", new_df["dst_bytes"].cast(IntegerType()))
```

```
new_df:  pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 5 more fields]

Command took 0.13 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 18

```
 #printing Schema
 new_df.printSchema()
```

```
root
 |-- duration: integer (nullable = true)
 |-- protocol_type: string (nullable = true)
 |-- service: string (nullable = true)
 |-- src_bytes: integer (nullable = true)
 |-- dst_bytes: integer (nullable = true)
 |-- flag: string (nullable = true)
 |-- label_s: string (nullable = true)

Command took 0.06 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 19

```
#new RDD with the columns: duration, protocol_type, service, src_bytes, dst_bytes, flag and label
#displaying 10 values
RDD_new = new_df.rdd.map(list)
RDD_new.take(10)
```

```
(1) Spark Jobs

Out[14]: [[0, 'tcp', 'http', 181, 5450, 'SF', 'normal.'],
 [0, 'tcp', 'http', 239, 486, 'SF', 'normal.'],
 [0, 'tcp', 'http', 235, 1337, 'SF', 'normal.'],
 [0, 'tcp', 'http', 219, 1337, 'SF', 'normal.'],
 [0, 'tcp', 'http', 217, 2032, 'SF', 'normal.'],
 [0, 'tcp', 'http', 217, 2032, 'SF', 'normal.'],
 [0, 'tcp', 'http', 212, 1940, 'SF', 'normal.'],
 [0, 'tcp', 'http', 159, 4087, 'SF', 'normal.'],
 [0, 'tcp', 'http', 210, 151, 'SF', 'normal.'],
 [0, 'tcp', 'http', 212, 786, 'SF', 'normal.']]
```

5. Connections

```
#Total number of connectiosn based on the protocol_type
protocol_connection = df.groupBy('protocol_type').count().orderBy('count', ascending=False)
display(protocol_connection)
```

▸ (2) Spark Jobs

▸ protocol_connection: pyspark.sql.dataframe.DataFrame = [protocol_type: string, count: long]

	protocol_type	count
1	icmp	283602
2	tcp	190065

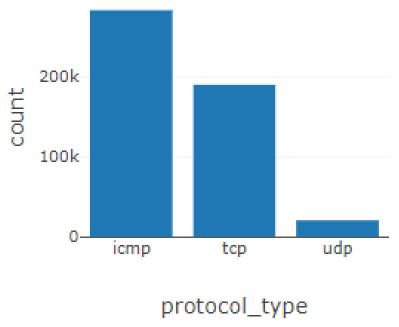
Showing all 3 rows.

```
Command took 11.62 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 22

```
#Bar graph of protocol_type connection
display(protocol_connection)
```

▸ (2) Spark Jobs



```
#Total number of connections based on the service
service_connection = df.groupBy('service').count().orderBy('count', ascending=False)
display(service_connection)
```

▸ (2) Spark Jobs

▸ service_connection: pyspark.sql.dataframe.DataFrame = [service: string, count: long]

	service	count
1	ecr_i	281400
2	private	110893

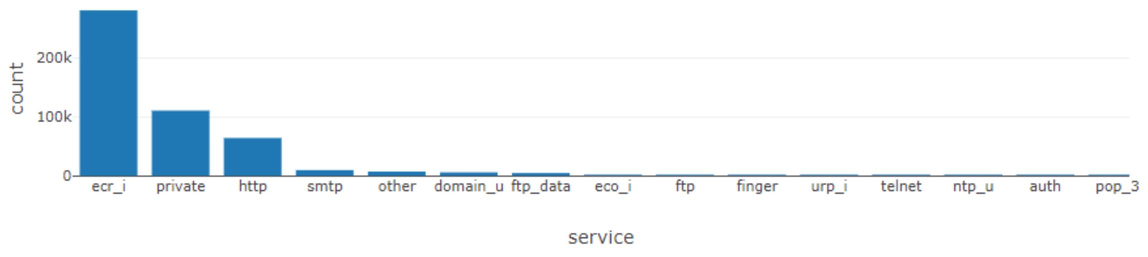
Showing all 66 rows.

```
Command took 11.26 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 24

```
#bar graph of service connections
display(service_connection)
```

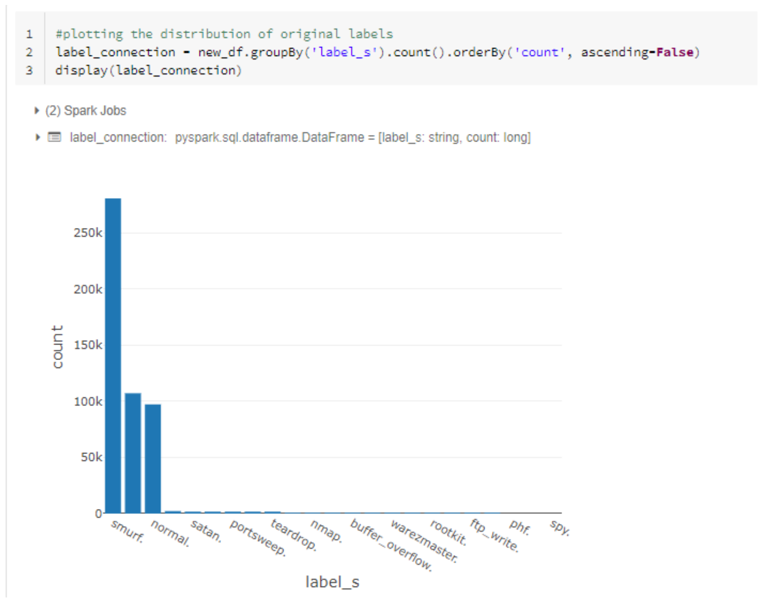
▸ (2) Spark Jobs



6. Exploratory Data Analysis

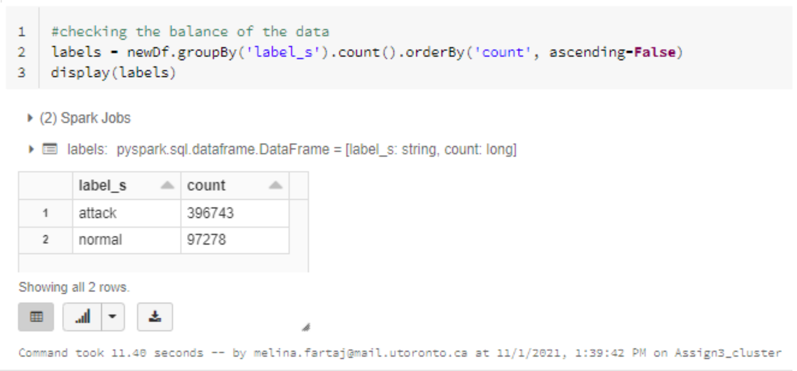
Connections

- Bar Graph showing the count of original labels.
    - Top 3 connections are “smurf.”, “neptume.”, and “normal.”
- Bar Graph showing the count of new labels.
    - Dataset is very unbalanced as there are 396743 attacks and 97278 normal connections.



```
#plotting the distribution of original labels
label_connection = new_df.groupBy('label_s').count().orderBy('count', ascending=False)
display(label_connection)
```

▸ (2) Spark Jobs
▸ label_connection: pyspark.sql.dataframe.DataFrame = [label_s: string, count: long]



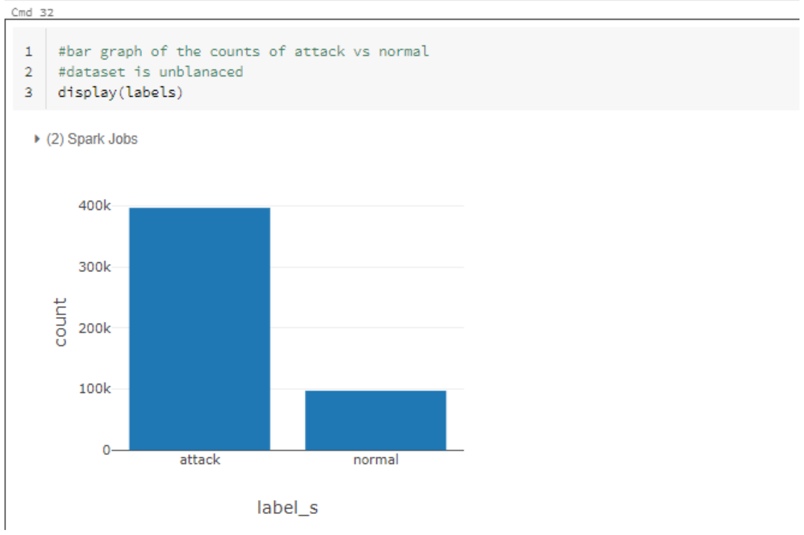
```
#checking the balance of the data
labels = newDf.groupBy('label_s').count().orderBy('count', ascending=False)
display(labels)
```

▸ (2) Spark Jobs
▸ labels: pyspark.sql.dataframe.DataFrame = [label_s: string, count: long]

	label_s	count
1	attack	396743
2	normal	97278

Showing all 2 rows.

```
Command took 11.40 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```



Cmd 32

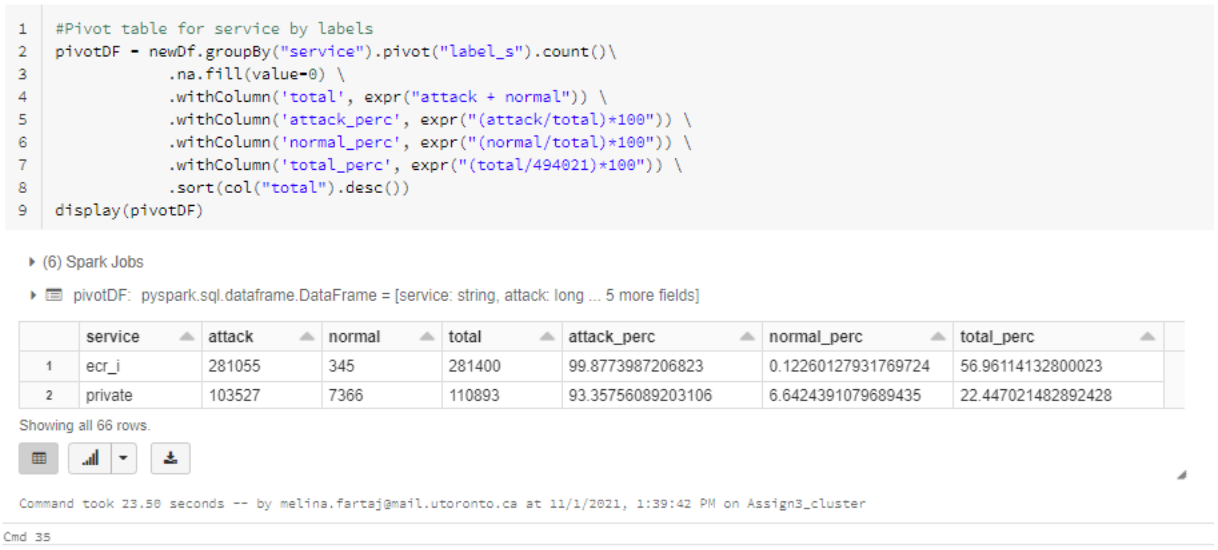
```
#bar graph of the counts of attack vs normal
#dataset is unblanaced
display(labels)
```

▸ (2) Spark Jobs

Connections (Attack or Normal) by Service

- These charts/graphs can help explore and identify relationships between services and connections.

- 3 pie charts for % of service for all connections, normal, and attack.
  - 59% of all connections are service “ecr_i”
  - 73% of normal connections are service “http”
  - 72% of attack connections are service “ecr_i”
- Bar graph showing the top 5 services by connection (attack/normal)
  - Majority is attack for services ecr_i and private
  - Majority is normal for services http, smtp and other



```
#Pivot table for service by labels
pivotDF = newDf.groupBy("service").pivot("label_s").count()\
            .na.fill(value=0) \
            .withColumn('total', expr("attack + normal")) \
            .withColumn('attack_perc', expr("(attack/total)*100")) \
            .withColumn('normal_perc', expr("(normal/total)*100")) \
            .withColumn('total_perc', expr("(total/494021)*100")) \
            .sort(col("total").desc())
display(pivotDF)
```

▸ (6) Spark Jobs

▸ pivotDF: pyspark.sql.dataframe.DataFrame = [service: string, attack: long ... 5 more fields]

	service	attack	normal	total	attack_perc	normal_perc	total_perc
1	ecr_i	281055	345	281400	99.8773987206823	0.12260127931769724	56.96114132800023
2	private	103527	7366	110893	93.35756089203106	6.6424391079689435	22.447021482892428

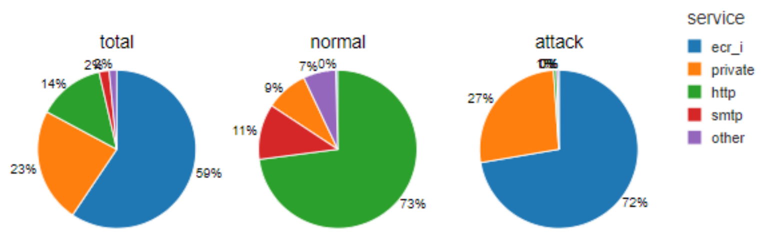
Showing all 66 rows.

```
Command took 23.58 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 35

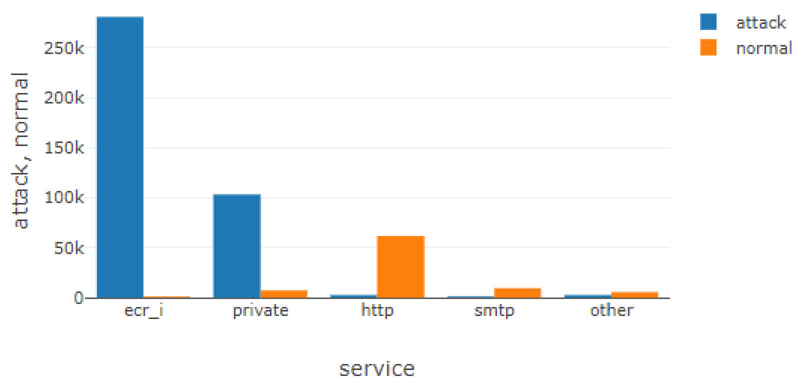
```
#pie charts for % of service for all connections, normal, and attack.
display(pivotDF.take(5))
```

▸ (6) Spark Jobs



```
#Bar graph showing the top 5 service by connection (attack/normal)
display(pivotDF.take(5))
```

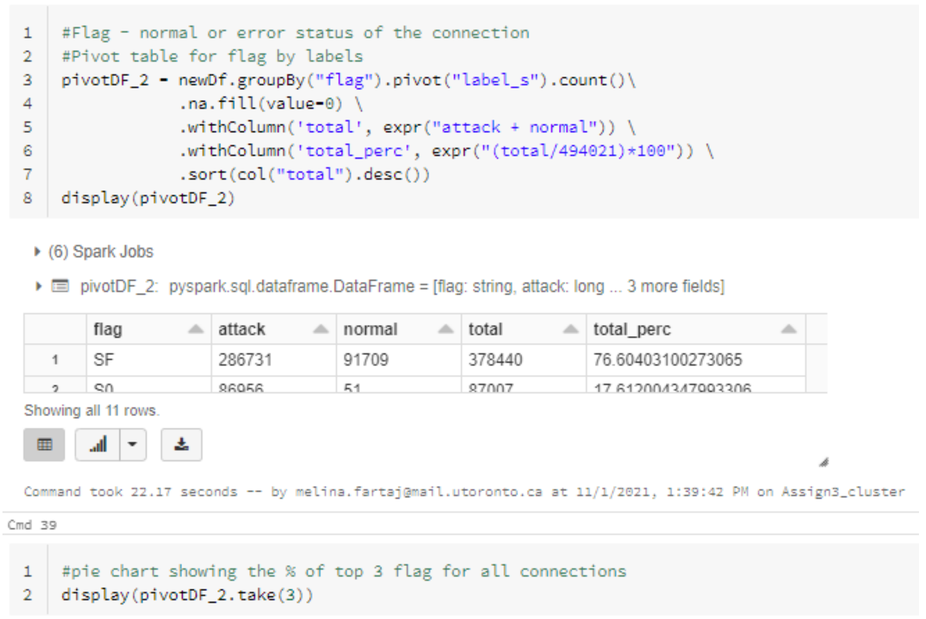
▸ (6) Spark Jobs



Connections (Attack or Normal) by Flag

- These charts/graphs can help explore and identify relationships between flag and connections.

- Pie Chart for % of flag for all connections.
    - 77% of all connections are flag “SF”
- Bar graph showing the top 3 flag by connection (attack/normal)
    - Majority is attack for top 3
    - Top 3 are SF, S0 and REJ



```python
#Flag - normal or error status of the connection
#Pivot table for flag by labels
pivotDF_2 = newDf.groupBy("flag").pivot("label_s").count()\
            .na.fill(value=0) \
            .withColumn('total', expr("attack + normal")) \
            .withColumn('total_perc', expr("(total/494021)*100")) \
            .sort(col("total").desc())
display(pivotDF_2)
```

(6) Spark Jobs

pivotDF_2: pyspark.sql.dataframe.DataFrame = [flag: string, attack: long ... 3 more fields]

	flag	attack	normal	total	total_perc
1	SF	286731	91709	378440	76.60403100273065
2	S0	86956	51	87007	17.612004347993306

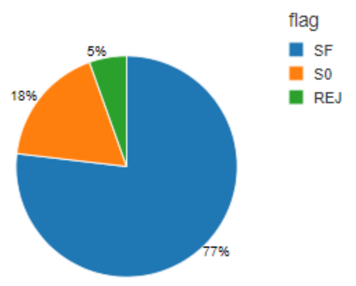
Showing all 11 rows.

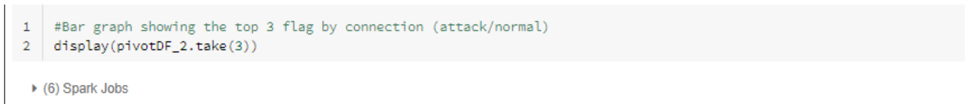
```
Command took 22.17 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 39

```python
#pie chart showing the % of top 3 flag for all connections
display(pivotDF_2.take(3))
```

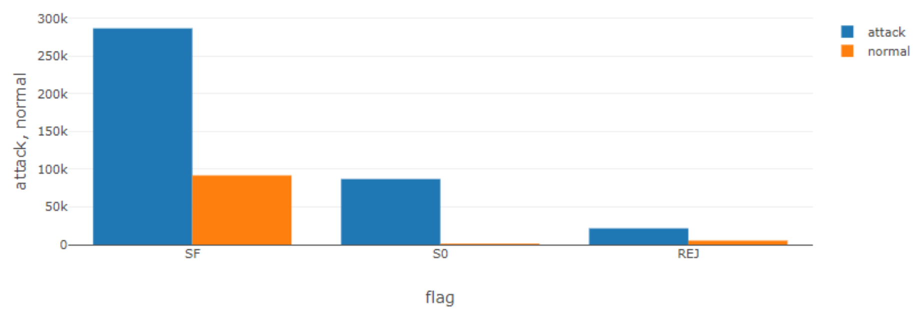
(6) Spark Jobs





```python
#Bar graph showing the top 3 flag by connection (attack/normal)
display(pivotDF_2.take(3))
```

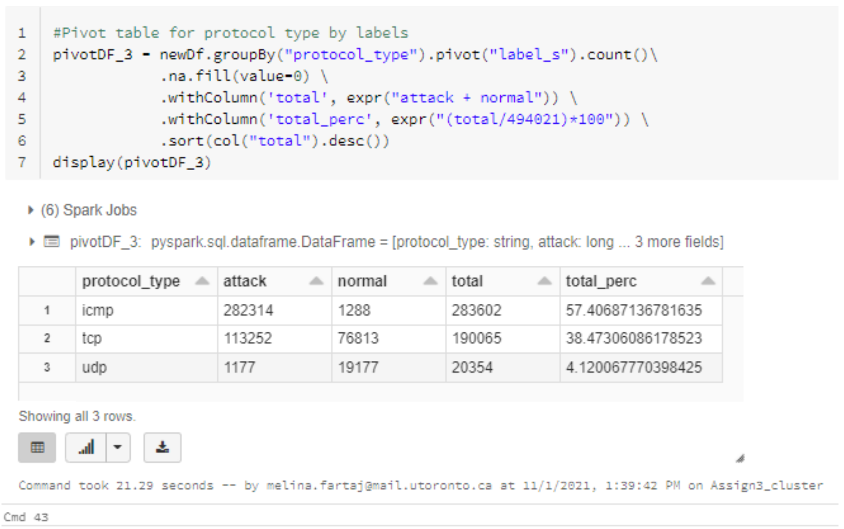
(6) Spark Jobs



Connections (Attack or Normal) by Protocol Type

- This chart can help explore and identify relationships between protocol type and connections.

- Pie Chart for % of protocol type for all connections.
    - 57% of all connections are protocol type “icmp”



```
#Pivot table for protocol type by labels
pivotDF_3 = newDf.groupBy("protocol_type").pivot("label_s").count()\
            .na.fill(value=0) \
            .withColumn('total', expr("attack + normal")) \
            .withColumn('total_perc', expr("(total/494021)*100")) \
            .sort(col("total").desc())
display(pivotDF_3)
```

(6) Spark Jobs

pivotDF_3: pyspark.sql.dataframe.DataFrame = [protocol_type: string, attack: long ... 3 more fields]

	protocol_type	attack	normal	total	total_perc
1	icmp	282314	1288	283602	57.40687136781635
2	tcp	113252	76813	190065	38.47306086178523
3	udp	1177	19177	20354	4.120067770398425

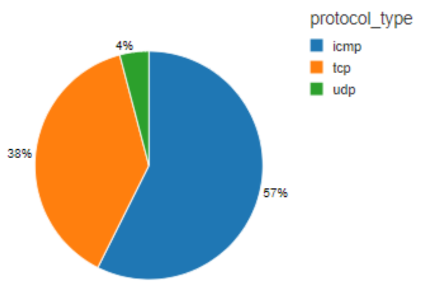
Showing all 3 rows.

```
Command took 21.29 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Cmd 43

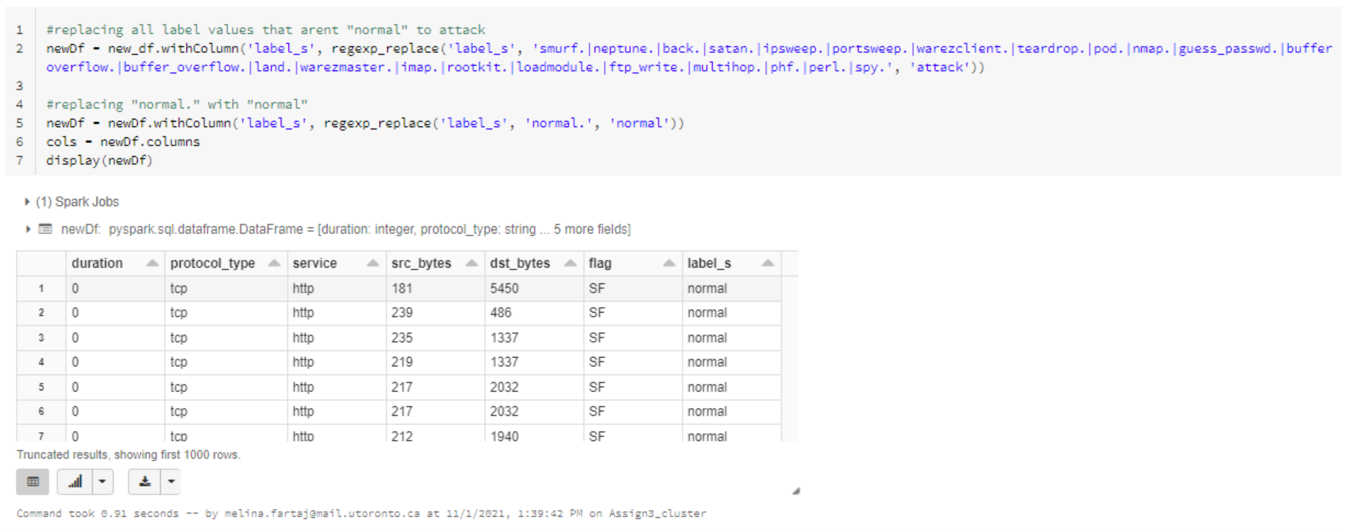
```
#pie chart showing the % of protocol type for all connections
display(pivotDF_3)
```

(3) Spark Jobs



7. Machine Learning Model

New Label Column



```
#replacing all label values that arent "normal" to attack
newDf = new_df.withColumn('label_s', regexp_replace('label_s', 'smurf.|neptune.|back.|satan.|ipsweep.|portsweep.|warezclient.|teardrop.|pod.|nmap.|guess_passwd.|buffer overflow.|buffer_overflow.|land.|warezmaster.|imap.|rootkit.|loadmodule.|ftp_write.|multihop.|phf.|perl.|spy.', 'attack'))

#replacing "normal." with "normal"
newDf = newDf.withColumn('label_s', regexp_replace('label_s', 'normal.', 'normal'))
cols = newDf.columns
display(newDf)
```

(1) Spark Jobs

newDf: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 5 more fields]

	duration	protocol_type	service	src_bytes	dst_bytes	flag	label_s
1	0	tcp	http	181	5450	SF	normal
2	0	tcp	http	239	486	SF	normal
3	0	tcp	http	235	1337	SF	normal
4	0	tcp	http	219	1337	SF	normal
5	0	tcp	http	217	2032	SF	normal
6	0	tcp	http	217	2032	SF	normal
7	0	tcp	http	212	1940	SF	normal

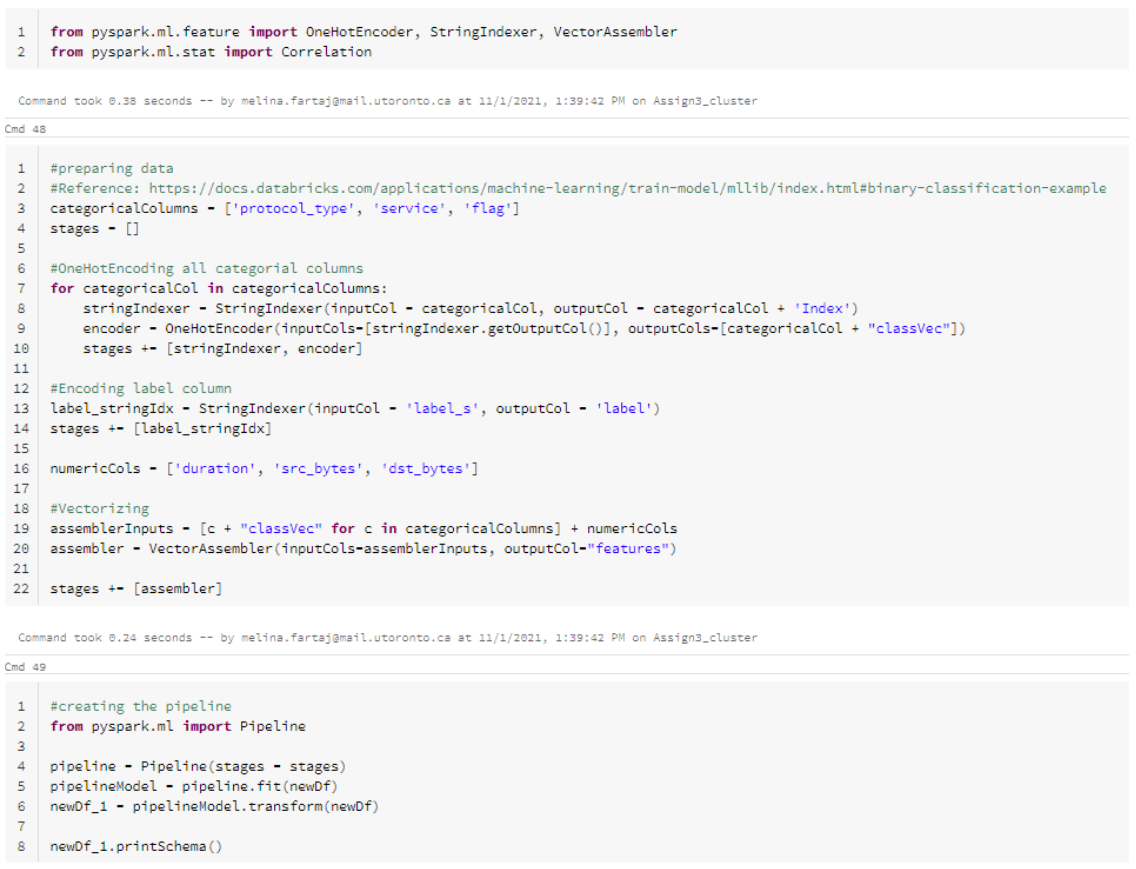
Truncated results, showing first 1000 rows.

```
Command took 0.91 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster
```

Data preprocessing

- This step was done to turn the categorical variables into numeric variables

- Source: https://spark.apache.org/docs/latest/ml-features



```
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.stat import Correlation
```

Command took 0.38 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster

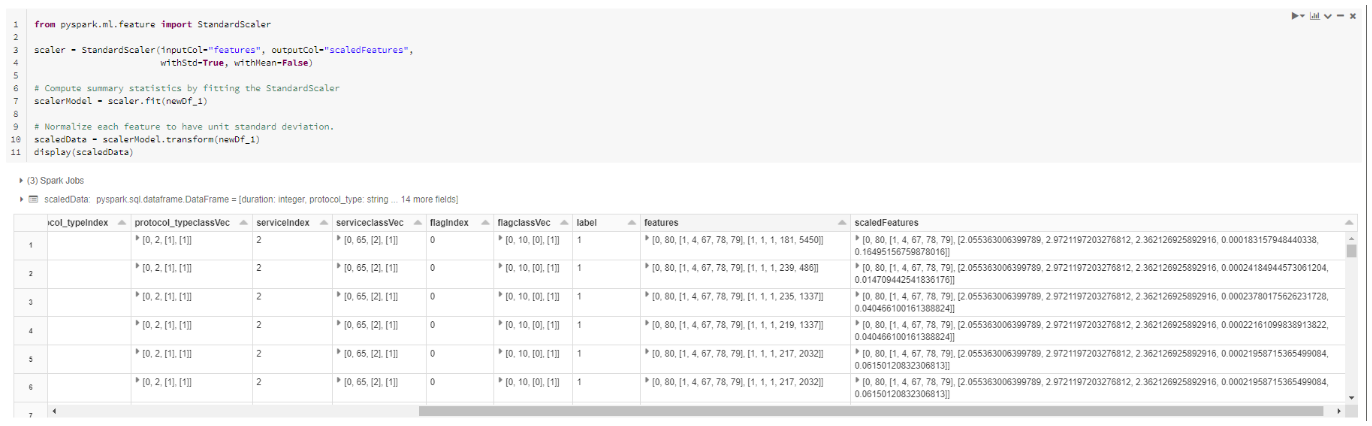
Cmd 48

```
#preparing data
#Reference: https://docs.databricks.com/applications/machine-learning/train-model/mllib/index.html#binary-classification-example
categoricalColumns = ['protocol_type', 'service', 'flag']
stages = []

#OneHotEncoding all categorial columns
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]

#Encoding label column
label_stringIdx = StringIndexer(inputCol = 'label_s', outputCol = 'label')
stages += [label_stringIdx]

numericCols = ['duration', 'src_bytes', 'dst_bytes']

#Vectorizing
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

stages += [assembler]
```

Command took 0.24 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 1:39:42 PM on Assign3_cluster

Cmd 49

```
#creating the pipeline
from pyspark.ml import Pipeline

pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(newDf)
newDf_1 = pipelineModel.transform(newDf)

newDf_1.printSchema()
```



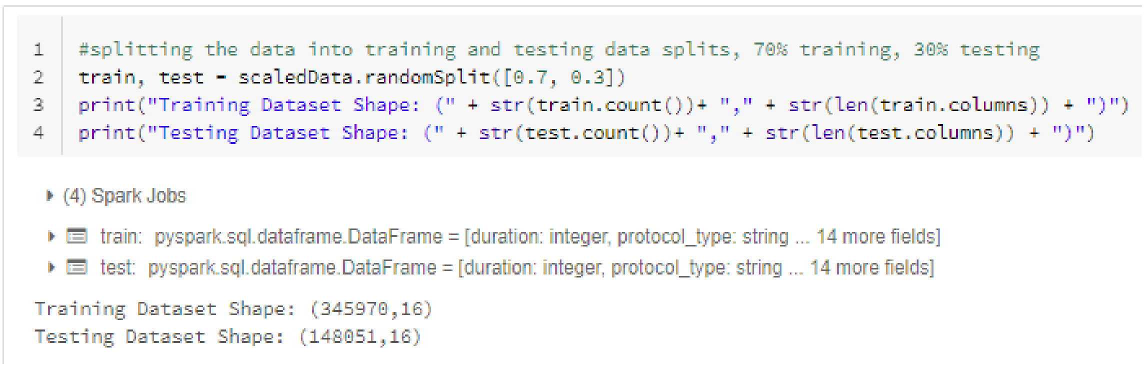
```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(newDf_1)

# Normalize each feature to have unit standard deviation.
scaledData = scalerModel.transform(newDf_1)
display(scaledData)
```

(3) Spark Jobs

scaledData: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 14 more fields]

	col_typeIndex	protocol_typeclassVec	serviceIndex	serviceclassVec	flagIndex	flagclassVec	label	features	scaledFeatures
1		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 181, 5450]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00018315794844033 8, 0.16495156759878016]]
2		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 239, 486]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00024184944573061204, 0.014709442541836176]]
3		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 235, 1337]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00023780175626231728, 0.040466100161388824]]
4		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 219, 1337]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00022161099838913822, 0.040466100161388824]]
5		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 217, 2032]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00021958715365499084, 0.06150120832306813]]
6		[0, 2, [1], [1]]	2	[0, 65, [2], [1]]	0	[0, 10, [0], [1]]	1	[0, 80, [1, 4, 67, 78, 79], [1, 1, 1, 217, 2032]]	[0, 80, [1, 4, 67, 78, 79], [2.055363006399789, 2.9721197203276812, 2.362126925892916, 0.00021958715365499084, 0.06150120832306813]]
7									

Splitting Data



```
#splitting the data into training and testing data splits, 70% training, 30% testing
train, test = scaledData.randomSplit([0.7, 0.3])
print("Training Dataset Shape: (" + str(train.count())+ "," + str(len(train.columns)) + ")")
print("Testing Dataset Shape: (" + str(test.count())+ "," + str(len(test.columns)) + ")")
```

(4) Spark Jobs

train: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 14 more fields]

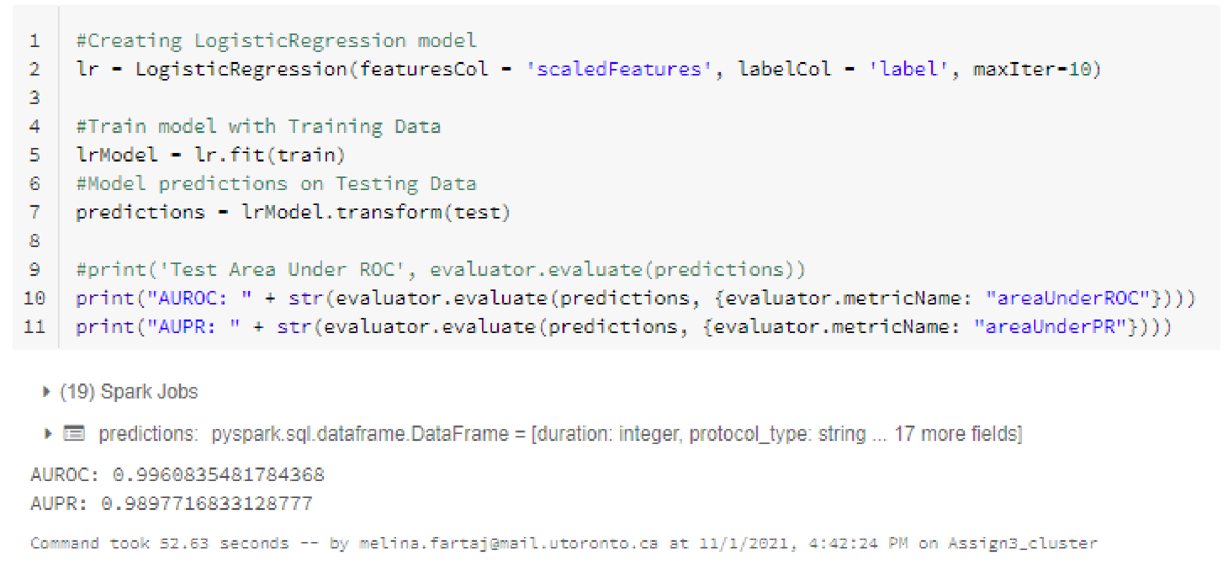
test: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 14 more fields]

```
Training Dataset Shape: (345970,16)
Testing Dataset Shape: (148051,16)
```

Simple Machine Learning Model

- 2 Machine Learning algorithms were used and evaluated based on AUROC (Area under receiver operating characteristic curve) and AUPR (Area under Precision-Recall curve) to determine the best algorithm to use to classify data into attacks and normal. As stated in the paper, which was provided with this assignment, the AUROC is a measure of a classifier's performance and the AUPR shows the trade-off between precision and recall for the different threshold.
- Logistic regression and Random Forest Classifier are popular algorithms used in the literature for Intrusion Detection Systems. The reason these algorithms were chosen is because Logistic Regressions can be used for any classification problem but performs especially well on binary classification and Random Forest has little impact from outliers and performs well on imbalanced datasets.
- The algorithm that was selected was Random Forest Classifier as it had a higher AUROC and AUPR than Logistic Regression.

Model	AUROC	AUPR
Logistic Regression	99.60%	98.97%
Random Forest Classifier	*99.92%*	*99.58%*



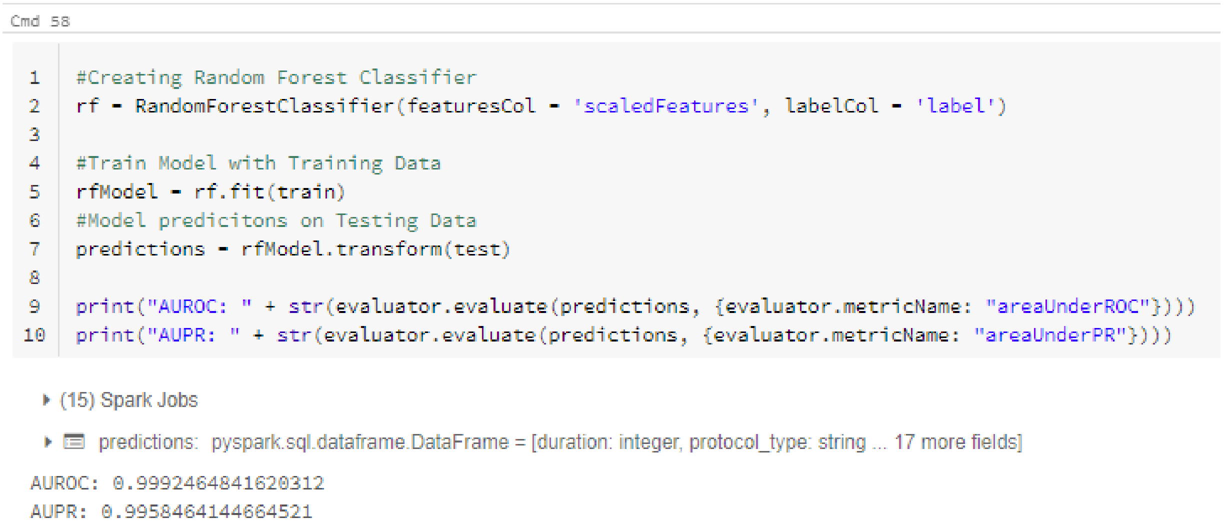
```
#Creating LogisticRegression model
lr = LogisticRegression(featuresCol = 'scaledFeatures', labelCol = 'label', maxIter=10)

#Train model with Training Data
lrModel = lr.fit(train)
#Model predictions on Testing Data
predictions = lrModel.transform(test)

#print('Test Area Under ROC', evaluator.evaluate(predictions))
print("AUROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
print("AUPR: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderPR"})))
```

▸ (19) Spark Jobs

▸ predictions: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 17 more fields]

```
AUROC: 0.9960835481784368
AUPR: 0.9897716833128777
```

Command took 52.63 seconds -- by melina.fartaj@mail.utoronto.ca at 11/1/2021, 4:42:24 PM on Assign3_cluster



Cmd 58

```
#Creating Random Forest Classifier
rf = RandomForestClassifier(featuresCol = 'scaledFeatures', labelCol = 'label')

#Train Model with Training Data
rfModel = rf.fit(train)
#Model predicitons on Testing Data
predictions = rfModel.transform(test)

print("AUROC: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderROC"})))
print("AUPR: " + str(evaluator.evaluate(predictions, {evaluator.metricName: "areaUnderPR"})))
```

▸ (15) Spark Jobs

▸ predictions: pyspark.sql.dataframe.DataFrame = [duration: integer, protocol_type: string ... 17 more fields]

```
AUROC: 0.9992464841620312
AUPR: 0.9958464144664521
```

Part B:

1. **Yes/No**
    a. **A platform as a service (PaaS) solution that hosts web apps in Azure provides professional development services to continuously add features to custom applications.**

Yes – This is true as PaaS provides the hardware and software tools so that clients don't need to worry about configuring or understanding what's going on beyond the application and data. The infrastructure and platform are managed by the cloud provider. Clients can extend or customize cloud-based applications. PaaS can "add development capabilities without adding staff. Platform as a Service components can give your development team new capabilities without your needing to add staff having the required skills" (https://azure.microsoft.com/en-us/overview/what-is-paas/).

    b. **A platform as a service (PaaS) database offering in Azure provides built in high availability.**

Yes – Continuing from the previous answer, PaaS allows clients to create applications using built-in software components. As also stated on the referenced website, "Cloud features such as scalability, high-availability and multi-tenant capability are included, reducing the amount of coding that developers must do."

2. **Multiple Choice**
    a. **A relational database must be used when:**
        i. **A dynamic schema is required**
        ii. **Data will be stored as key/value pairs**
        iii. **Storing large images and videos**
        iv. *Strong consistency guarantees are required*

A relational database is used with structured data. A dynamic schema and data stored as key/value pairs refer to NoSQL databases. Storing large images and videos is not possible with a relational database as those are unstructured data. A relational database is focused on providing strong consistency.

3. **Multiple Choice**
    a. **When you are implementing a Software as a Service solution, you are responsible for:**
        i. **Configuring high availability**
        ii. **Defining scalability rules**
        iii. **Installing the SaaS solution**
        iv. *Configuring the SaaS solution*

For SaaS everything is managed by the cloud provider. When implementing a SaaS solution, the client is responsible for configuring the SaaS solution.

4. **Yes/No**
    a. **To achieve a hybrid cloud model, a company must always migrate from a private cloud model**

No – This is not true as a client can also start with a public cloud model to achieve a hybrid cloud model by combining with an on-premises infrastructure.

    b. **A company can extend the capacity of its internal network by using public cloud**

Yes – A client can use a public cloud to extend the capacity of its internal network by connecting the on-premises network to the cloud environment.

c. **In a public cloud model, only guest users at your company can access the resources in the cloud**

No – This is not true, anyone with an account in the Azure Active Directory can access the resources in the cloud. (source: https://azure.microsoft.com/en-gb/overview/what-is-hybrid-cloud-computing/)

5. **Fill in the Blanks**
   a. **A cloud service that remains available after a failure occurs ______________**
   b. **A cloud service that can be recovered after a failure occurs _______________**
   c. **A cloud service that performs quickly when demand increases ____________**
   d. **A cloud service that can be accessed quickly from the internet _____________**

   Disaster recovery, Fault Tolerance, Low Latency, Dynamic Scalability

a) Fault Tolerance
   - A cloud service that remains available after a failure occurs is fault tolerant. Fault tolerance is the ability to prevent a failure by providing high availability (the ability to ensure a service remains available).

b) Disaster Recovery
   - A cloud service that can be recovered after a failure occurs is disaster recovery. Disaster recovery is the "ability to recover from a disaster and to prevent the loss of data solutions that recover from a disaster" (Lecture 5, MIE 1628 Big Data Science Fall 2021).

c) Dynamic Scalability
   - A cloud service that performs quickly when demand increases is dynamic scalability. Dynamic scalability is the "ability to increase your capacity based on the increasing demand of traffic, memory and computing power" (Lecture 5, MIE 1628 Big Data Science Fall 2021).

d) Low Latency
   - A cloud service that can be accessed quickly from the internet is low latency. Latency is the delay between a client request and a cloud service providers response. It's the amount of time that is taken to send information from one point to the next (https://www.ir.com/guides/what-is-network-latency).