

task_2_EDA

February 22, 2026

```
[1]: !pip install pyspark
```

Requirement already satisfied: pyspark in /Users/jju/Documents/SIM/Semester 1, 2026/CSCI316 Big Data Mining/Assignments/.venv/lib/python3.9/site-packages (3.5.1)

Requirement already satisfied: py4j==0.10.9.7 in /Users/jju/Documents/SIM/Semester 1, 2026/CSCI316 Big Data Mining/Assignments/.venv/lib/python3.9/site-packages (from pyspark) (0.10.9.7)

[notice] A new release of pip is available: 25.3 -> 26.0.1

[notice] To update, run:

```
pip install --upgrade pip
```

```
[2]: # Better view for charts
# High quality image
%config InlineBackend.figure_format = 'svg'
# # Mount Google Drive
# from google.colab import drive
# drive.mount('/content/drive')

# 0. Create Spark Session
import os
import sys
import pandas as pd
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.sql.functions import regexp_extract, col, when, avg, lag, abs as spark_abs
from pyspark.sql.functions import col, log1p, abs as spark_abs

from pyspark.sql.window import Window
from pyspark.sql.types import DoubleType
import matplotlib.pyplot as plt
import seaborn as sns

# Updated train and test split
from sklearn.preprocessing import StandardScaler, LabelEncoder
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import GridSearchCV, train_test_split
```

```
spark = SparkSession.builder \
    .appName("UNSW_Preprocessing") \
    .getOrCreate()
```

26/02/22 14:13:46 WARN Utils: Your hostname, ijuwon-ui-MacBookPro.local resolves to a loopback address: 127.0.0.1; using 192.168.0.8 instead (on interface en0)

26/02/22 14:13:46 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address

Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

26/02/22 14:13:47 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

```
[3]: def load_and_prep_spark_data():
    # Update paths
    train_path = '/content/drive/MyDrive/University/CSCI316 - Big Data Mining_
↳Techniques/Group Assignment/UNSW_NB15_training-set.csv'
    test_path = '/content/drive/MyDrive/University/CSCI316 - Big Data Mining_
↳Techniques/Group Assignment/UNSW_NB15_testing-set.csv'

    # Juwon's Local file path
    test_path = '/Users/jju/Documents/SIM/Semester 1, 2026/CSCI316 Big Data_
↳Mining/Assignments/Group_Assignment_Database/UNSW_NB15_testing-set.csv'
    train_path = '/Users/jju/Documents/SIM/Semester 1, 2026/CSCI316 Big Data_
↳Mining/Assignments/Group_Assignment_Database/UNSW_NB15_training-set.csv'

    print("Loading data...")
    df_train_orig = spark.read.csv(train_path, header=True, inferSchema=True)
    df_test_orig = spark.read.csv(test_path, header=True, inferSchema=True)

    # 1. Combine Datasets (Fixes Split Mismatch)
    # Drop ID and attack_cat (Fixes Leakage)
    df_full = df_train_orig.unionByName(df_test_orig).drop('id', 'attack_cat')

    # 2. Feature Engineering (Matches RF Features)
    # Add pkt_ratio and ttl_gap so SVM has the same info as RF
    df_full = df_full.withColumn("pkt_ratio", (col("spkts") + 1) /_
↳(col("dpkts") + 1))
    df_full = df_full.withColumn("ttl_gap", spark_abs(col("sttl") -_
↳col("dttl")))

    print(f"Total Records: {df_full.count()}")
```

```

# 3. Stratified Split (70% Train, 15% Val, 15% Test)
# We split by label to ensure balance
zeros = df_full.filter(col("label") == 0)
ones = df_full.filter(col("label") == 1)

train_0, val_0, test_0 = zeros.randomSplit([0.7, 0.15, 0.15], seed=42)
train_1, val_1, test_1 = ones.randomSplit([0.7, 0.15, 0.15], seed=42)

train_data = train_0.union(train_1)
val_data = val_0.union(val_1)
test_data = test_0.union(test_1)

print(f"Split Sizes -> Train: {train_data.count()}, Val: {val_data.
count()}, Test: {test_data.count()}")
return train_data, val_data, test_data

train_df, val_df, test_df = load_and_prep_spark_data()

```

Loading data...

Total Records: 257673

26/02/22 14:13:51 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.sql.debug.maxToStringFields'.

Split Sizes -> Train: 180973, Val: 38573, Test: 38127

0.1 Add new columns

```

[4]: from pyspark.sql.functions import isnan, when, count

def missing_value_report(df):
    return df.select([
        count(when(isnan(c) | col(c).isNull(), c)).alias(c)
        for c in df.columns
    ])

print("train_df missing value report")
missing_value_report(train_df).show(truncate=False)
print("test_df missing value report")
missing_value_report(test_df).show(truncate=False)
print("val_df missing value report")
missing_value_report(val_df).show(truncate=False)

```

train_df missing value report

```

+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

```


test_df missing value report

dur	proto	service	state	spkts	dpkts	sbytes	dbytes	rate	sttl	dttl	sload	dload	sloss	dloss											
oss	dloss	sinpkt	dinpkt	sjit	djit	swin	stcpb	dtcpb	dwin	tcprrt	synack	ackdat	smean	dmean											
an	dmean	trans_depth	response_body_len	ct_srv_src	ct_state_ttl	ct_dst_ltm	ct_src_dport_ltm	ct_dst_sport_ltm	ct_dst_src_ltm	is_ftp_login	ct_ftp_cmd	ct_flw_http_mthd	ct_src_ltm	ct_srv_dst											
is_sm_ips_ports	label	pkt_ratio	ttnl_gap																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0											

```

--+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+
--+-----+-----+-----+-----+-----+-----+-----+

val_df missing value report
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
-----+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
|dur|proto|service|state|spkts|dpkts|sbytes|dbytes|rate|sttl|dttl|sload|dload|sl
oss|dloss|sinpkt|dinpkt|sjit|djit|swin|stcpb|dtcpb|dwin|tcprrt|synack|ackdat|sme
an|dmean|trans_depth|response_body_len|ct_srv_src|ct_state_ttl|ct_dst_ltm|ct_src
_dport_ltm|ct_dst_sport_ltm|ct_dst_src_ltm|is_ftp_login|ct_ftp_cmd|ct_flw_http_m
thd|ct_src_ltm|ct_srv_dst|is_sm_ips_ports|label|pkt_ratio|ttl_gap|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
-----+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+
-----+---+---+---+---+---+---+---+---+---+---+---+
--+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

0.2 Feature Engineering

```

[5]: # Add new columns pkt_ratio and ttl_gap
spark_train_df = train_df \
    .withColumn("pkt_ratio", (col("spkts") + 1) / (col("dpkts") + 1)) \
    .withColumn("ttl_gap", spark_abs(col("sttl") - col("dttl")))

spark_test_df = test_df \
    .withColumn("pkt_ratio", (col("spkts") + 1) / (col("dpkts") + 1)) \
    .withColumn("ttl_gap", spark_abs(col("sttl") - col("dttl")))

spark_val_df = val_df \
    .withColumn("pkt_ratio", (col("spkts") + 1) / (col("dpkts") + 1)) \
    .withColumn("ttl_gap", spark_abs(col("sttl") - col("dttl")))

```

```
[6]: # Exclude_columns
# trans_depth(only zeros)
# response_body_len(only zeros)
excluded = {"attack_cat", "id"}

selected_cols = [c for c in train_df.columns if c not in excluded]

# select dataframe with only selected columns
df_filtered = train_df.select(*selected_cols)

# Extract Numeric Columns
from pyspark.sql.types import NumericType

numeric_cols = [
    f.name for f in df_filtered.schema.fields
    if isinstance(f.dataType, NumericType) and f.name != "label"
]

print("Numeric columns:", numeric_cols)

# only filter the any nulls detected columns
null_summary = df_filtered.select([
    count(
        when(col(c).isNull() |.isnan(col(c)), c)
    ).alias(c)
    for c in numeric_cols
])

# Show all data
null_summary.show(truncate=False)
```

```
Numeric columns: ['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl',
'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit',
'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprrt', 'synack', 'ackdat', 'smean',
'dmean', 'trans_depth', 'response_body_len', 'ct_srv_src', 'ct_state_ttl',
'ct_dst_ltm', 'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm',
'is_ftp_login', 'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm', 'ct_srv_dst',
'is_sm_ips_ports', 'pkt_ratio', 'ttl_gap']
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|dur|spkts|dpkts|sbytes|dbytes|rate|sttl|dttl|sload|dload|sloss|dloss|sinpkt|din
pkt|sjit|djit|swin|stcpb|dtcpb|dwin|tcprrt|synack|ackdat|smean|dmean|trans_depth
```

```
|response_body_len|ct_srv_src|ct_state_ttl|ct_dst_ltm|ct_src_dport_ltm|ct_dst_sp
ort_ltm|ct_dst_src_ltm|is_ftp_login|ct_ftp_cmd|ct_flw_http_mthd|ct_src_ltm|ct_sr
v_dst|is_sm_ips_ports|pkt_ratio|ttl_gap|
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
|0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |0  |
+---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

0.3 Visualization

0.3.1 Distributions

```
[7]: import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

sample_pd = spark_train_df.select(
    "spkts", "sbytes", "dbytes", "rate"
).sample(fraction=0.05, seed=42).toPandas()

fig, axes = plt.subplots(2, 2, figsize=(10,8))

sns.histplot(np.log1p(sample_pd["spkts"]), bins=50, ax=axes[0,0])
axes[0,0].set_title("spkts (log scaled)")

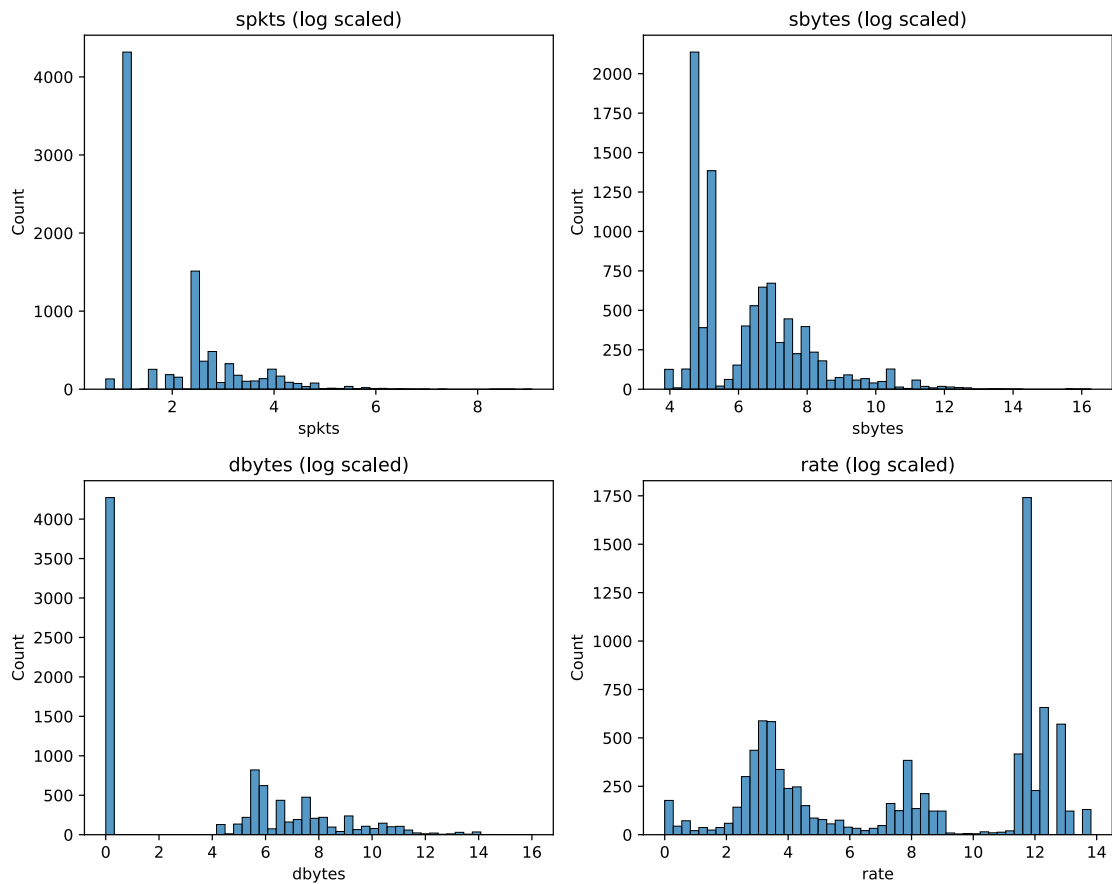
sns.histplot(np.log1p(sample_pd["sbytes"]), bins=50, ax=axes[0,1])
axes[0,1].set_title("sbytes (log scaled)")

sns.histplot(np.log1p(sample_pd["dbytes"]), bins=50, ax=axes[1,0])
axes[1,0].set_title("dbytes (log scaled)")

sns.histplot(np.log1p(sample_pd["rate"]), bins=50, ax=axes[1,1])
axes[1,1].set_title("rate (log scaled)")

plt.tight_layout()
plt.savefig("EDA_images/task2_distribution.png", dpi=300)
```

```
plt.show()
```



0.3.2 Correlation Visualization

```
[8]: # Correlation Check to identify unnecessary columns
# Calculate correlation of each labels
corr_rows = []

for c in numeric_cols:
    corr_val = spark_train_df.stat.corr(c, "label") # =>
    corr_rows.append((c, corr_val))

# convert to spark DF
corr_df = spark.createDataFrame(
    corr_rows,
    ["feature", "correlation"]
)

# convert to pandas DF
```

```

corr_pd = (
    corr_df
    .toPandas()
    .sort_values(by="correlation", key=abs, ascending=False)
)

sns.set_theme(style="whitegrid")
plt.figure(figsize=(8, 10))
sns.barplot(
    data=corr_pd,
    x="correlation",
    y="feature",
    palette="coolwarm"
)

plt.axvline(0, color="black", linewidth=0.8)
plt.title("Correlation between Numerical Features and Label")
plt.xlabel("Pearson Correlation")
plt.ylabel("Feature")
plt.tight_layout()
plt.savefig("EDA_images/task2_correlation.png", dpi=300)
plt.show()

```

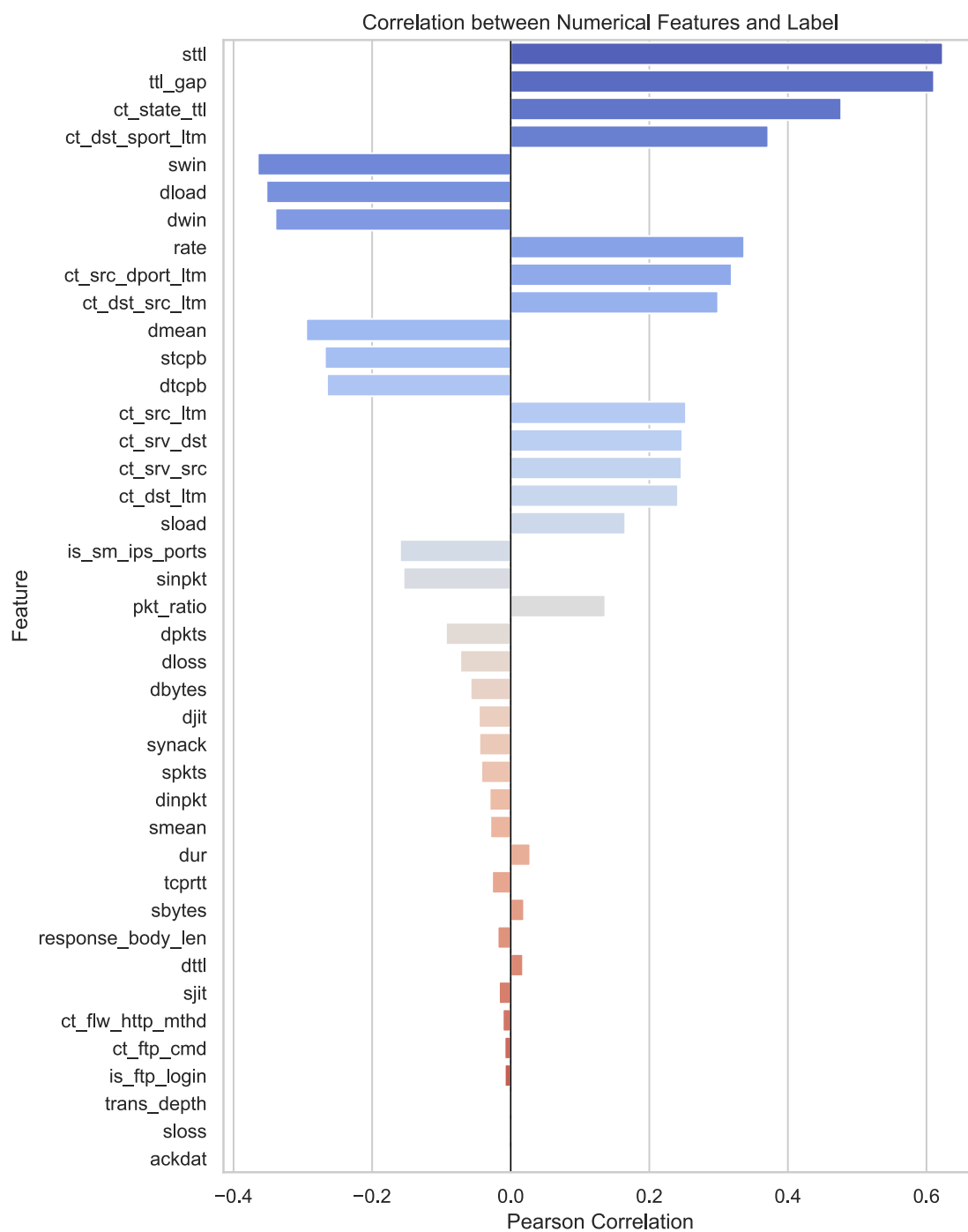
/var/folders/vy/xvrzy34d5r3dcmcbf4jj5sz80000gn/T/ipykernel_42645/2561798699.py:2
5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```

sns.barplot(

```

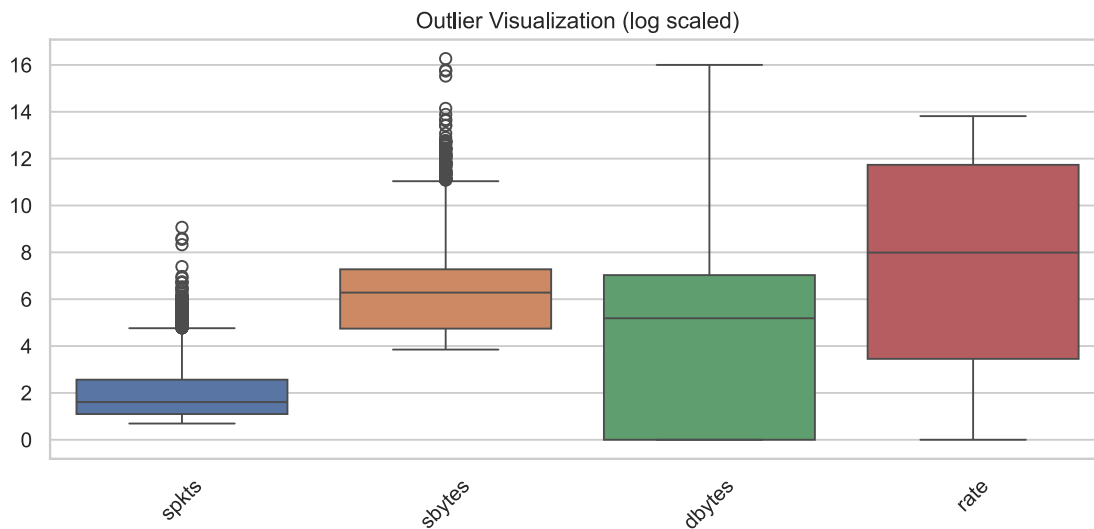


0.3.3 Outlier Visualization

```
[9]: box_cols = ["spkts", "sbytes", "dbytes", "rate"]

box_pd = spark_train_df.select(box_cols) \
    .sample(0.05, seed=42) \
    .toPandas()

plt.figure(figsize=(10,4))
sns.boxplot(data=np.log1p(box_pd))
plt.xticks(rotation=45)
plt.title("Outlier Visualization (log scaled)")
plt.savefig("EDA_images/task2_outliers_log_scaled.png", dpi=300)
plt.show()
```



0.3.4 Label Distribution

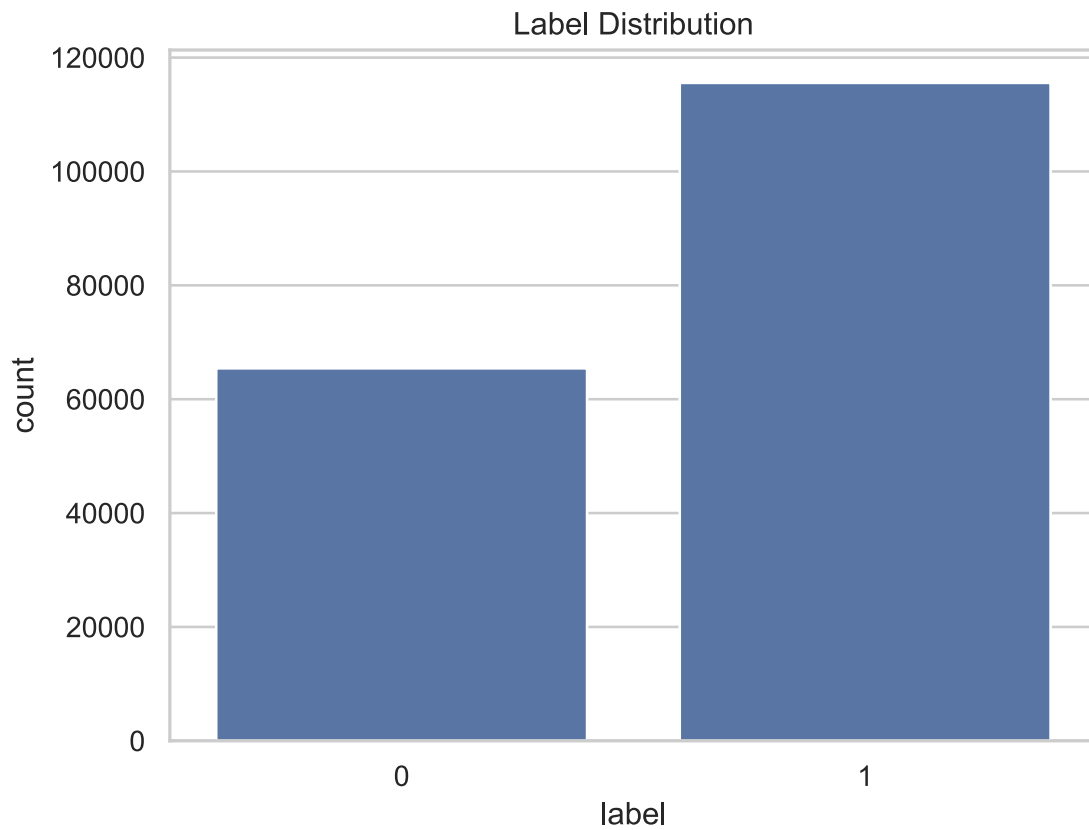
```
[10]: label_count_df = spark_train_df.groupBy("label").count()
label_count_df.show()

label_pd = label_count_df.toPandas()

sns.barplot(x="label", y="count", data=label_pd)
plt.title("Label Distribution")
plt.savefig("EDA_images/task2_label_distribution.png", dpi=300)
plt.show()
```

+-----+-----+

label	count
0	65421
1	115552

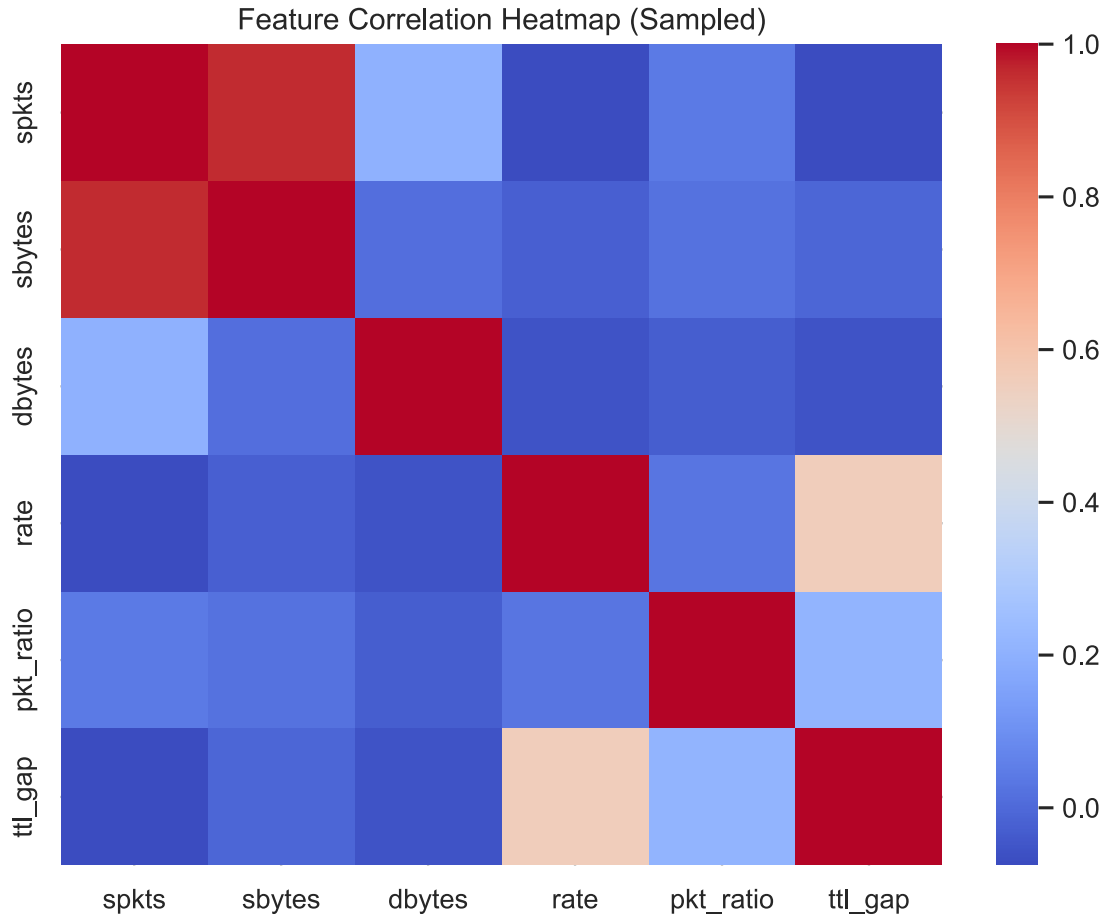


0.3.5 Correlation heatmap

```
[11]: corr_cols = ["spkts", "sbytes", "dbytes", "rate", "pkt_ratio", "ttl_gap"]

corr_pd = spark_train_df.select(corr_cols) \
    .sample(0.05, seed=42) \
    .toPandas()

plt.figure(figsize=(8,6))
sns.heatmap(corr_pd.corr(), cmap="coolwarm", annot=False)
plt.title("Feature Correlation Heatmap (Sampled)")
plt.savefig("EDA_images/task2_feature_correlation_heatmap.png", dpi=300)
plt.show()
```



Opinion

After removing id (which is primary key) and attack_cat which includes the answers of the data to check any unnecessary data or data trend.

Opinion

StandardScaler was considered prior to PCA, however the intrusion detection requires to identify any kind of outliers available therefore, standardizing the data is not an appropriate way to preprocess. However, since some features have same direction of correlation values, we applied PCA method to reduce feature dimensions.

0.4 TRY 1: Preprocessing with Numerical data only

0.4.1 RFormula → features → PCA → pca_features → Model

```
[12]: from pyspark.sql.functions import col
```

```
preparedDF = train_df \  
    .drop("id", "attack_cat") \  
    .withColumn("label", col("label").cast("integer")) \  
    .cache()
```

preparedDF ↓ (stratified split) train_data / val_data / test_data ↓ (StringIndexer fit on train, transform all) train_data / val_data / test_data ↓ (VectorAssembler) trainDF / valDF / testDF
↓ model.fit(trainDF)

```
[13]: from pyspark.ml.feature import PCA, StringIndexer, VectorAssembler  
from pyspark.sql.types import StringType  
from pyspark.ml.classification import LogisticRegression
```

```
## Step 0: Categorical_str_cols
```

```
categorical_str_cols = [  
    f.name for f in preparedDF.schema.fields  
    if isinstance(f.dataType, StringType)  
]
```

```
df_labeled = preparedDF.withColumn("label", col("label").cast("integer"))
```

```
## Step 1: Stratified split
```

```
zeros = df_labeled.filter(col("label") == 0)  
ones = df_labeled.filter(col("label") == 1)
```

```
train_0, val_0, test_0 = zeros.randomSplit([0.7, 0.15, 0.15], seed=42)
```

```
train_1, val_1, test_1 = ones.randomSplit([0.7, 0.15, 0.15], seed=42)
```

```
# These data can be used in next step
```

```
train_data = train_0.union(train_1).cache()
```

```
val_data = val_0.union(val_1).cache()
```

```
test_data = test_0.union(test_1).cache()
```

```
## Step1: StringIndexer (preparedDF)
```

```
for col_name in categorical_str_cols:  
    idx_col = f"{col_name}_idx"  
    if idx_col in train_data.columns:  
        continue #  
    indexer = StringIndexer(  
        inputCol=col_name,  
        outputCol=idx_col,  
        handleInvalid="keep"  
    )
```

```

train_data = indexer.fit(train_data).transform(train_data)
val_data = indexer.fit(val_data).transform(val_data)
test_data = indexer.fit(test_data).transform(test_data)

## Step 2: VectorAssembler
feature_cols = [
    c for c in train_data.columns
    if c not in ["id", "label", "attack_cat", "proto", "service", "state"]
]

assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol="features",
    handleInvalid="skip"
)

trainDF = assembler.transform(train_data) \
    .select("features", col("label").cast("int"))

valDF = assembler.transform(val_data) \
    .select("features", col("label").cast("int"))

testDF = assembler.transform(test_data) \
    .select("features", col("label").cast("int"))

## Step 4: PCA (train_only)
pca = PCA(k=30, inputCol="features", outputCol="pca_features")
pca_model = pca.fit(trainDF)

pca_train = pca_model.transform(trainDF).cache()
pca_test = pca_model.transform(testDF).cache()
pca_val = pca_model.transform(valDF).cache()

## Step 5: Logistic Regression
lr = LogisticRegression(
    featuresCol="pca_features",
    labelCol="label",
    maxIter=20,
    regParam=0.01
)

model = lr.fit(pca_train)

## Step 6: Prediction

```

```

predictions = model.transform(pca_test)

predictions.select(
    "label", "probability", "prediction"
).show(10, truncate=False)

```

26/02/22 14:14:28 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
 26/02/22 14:14:28 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK

```

+-----+-----+-----+-----+
|label|probability|prediction|
+-----+-----+-----+
|0| [0.9657963456913502,0.0342036543086498]|0.0|
|0| [0.965796342758693,0.03420365724130703]|0.0|
|0| [0.21987171483516488,0.7801282851648351]|1.0|
|0| [0.8959041335156306,0.10409586648436941]|0.0|
|0| [0.8692014392933969,0.13079856070660312]|0.0|
|0| [0.22632904988800479,0.7736709501119952]|1.0|
|0| [0.6813521179065707,0.3186478820934293]|0.0|
|0| [0.8739641223041081,0.12603587769589186]|0.0|
|0| [0.8761169850157817,0.12388301498421828]|0.0|
|0| [0.8961283929367126,0.10387160706328735]|0.0|
+-----+-----+-----+-----+

```

only showing top 10 rows

0.4.2 Test PCA prediction

[14]: *# Evaluate Multiple Matrix for Testing (For me, Model is not necessary, but in order to test my preprocessed data)*

```

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

metrics = { "accuracy": "accuracy",
            "precision": "weightedPrecision",
            "recall": "weightedRecall",
            "f1": "f1"
          }

print("="*40)
print("PCA TRAIN RESULTS")
print("="*40)
for name, metric in metrics.items():
    evaluator = MulticlassClassificationEvaluator(
        labelCol="label",
        predictionCol="prediction",

```

```

        metricName=metric
    )
    print(f"\n{name}: {evaluator.evaluate(predictions):.4f}")

```

```

=====
PCA TRAIN RESULTS
=====

```

accuracy: 0.8866

precision: 0.8988

recall: 0.8866

f1: 0.8813

0.5 TRY 2 : Preprocessing with Categorical data + numerical data for tree

0.5.1 RFormula → features → Model

```

[15]: from pyspark.sql.functions import col
      from pyspark.sql.types import StringType
      from pyspark.ml.feature import StringIndexer, VectorAssembler
      from pyspark.ml.classification import LogisticRegression

      ## Step 1: Stratified split
      df_labeled = preparedDF.withColumn("label", col("label").cast("integer"))

      zeros = df_labeled.filter(col("label") == 0)
      ones  = df_labeled.filter(col("label") == 1)

      train_0, val_0, test_0 = zeros.randomSplit([0.7, 0.15, 0.15], seed=42)
      train_1, val_1, test_1 = ones.randomSplit([0.7, 0.15, 0.15], seed=42)

      # These data can be used in next step
      train_data = train_0.union(train_1).cache()
      val_data   = val_0.union(val_1).cache()
      test_data  = test_0.union(test_1).cache()

      # Find string columns
      categorical_str_cols = [
          f.name for f in train_data.schema.fields
          if isinstance(f.dataType, StringType)
      ]

```

```

## Step1: StringIndexer (preparedDF)
for col_name in categorical_str_cols:
    idx_col = f"{col_name}_idx"
    if idx_col in train_data.columns:
        continue #
    indexer = StringIndexer(
        inputCol=col_name,
        outputCol=idx_col,
        handleInvalid="keep"
    )
    train_data = indexer.fit(train_data).transform(train_data)
    val_data = indexer.fit(val_data).transform(val_data)
    test_data = indexer.fit(test_data).transform(test_data)

## Step 2: VectorAssembler
feature_cols = [
    c for c in train_data.columns
    if c not in ["id", "label", "attack_cat", "proto", "service", "state"]
]

assembler = VectorAssembler(
    inputCols=feature_cols,
    outputCol="features",
    handleInvalid="skip"
)

trainDF = assembler.transform(train_data) \
    .select("features", col("label").cast("int"))

valDF = assembler.transform(val_data) \
    .select("features", col("label").cast("int"))

testDF = assembler.transform(test_data) \
    .select("features", col("label").cast("int"))

# 5) LR train + predict
lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=20,
    ↪ regParam=0.01)
model = lr.fit(trainDF)

pred = model.transform(testDF)
pred.select("label", "probability", "prediction").show(10, truncate=False)

```

```

26/02/22 14:14:32 WARN CacheManager: Asked to cache already cached data.
26/02/22 14:14:32 WARN CacheManager: Asked to cache already cached data.
26/02/22 14:14:32 WARN CacheManager: Asked to cache already cached data.

```

```

+-----+-----+-----+-----+-----+

```

	label probability	prediction
+	-----+	-----+
0	[0.9904557577543591,0.009544242245640877]	0.0
0	[0.9904557602884948,0.009544239711505198]	0.0
0	[0.2181936901552711,0.7818063098447289]	1.0
0	[0.8263723852983746,0.17362761470162535]	0.0
0	[0.7858537461869733,0.21414625381302665]	0.0
0	[0.212772365907601,0.7872276340923989]	1.0
0	[0.5011668471969392,0.49883315280306084]	0.0
0	[0.7949987054094111,0.2050012945905889]	0.0
0	[0.8304592779681389,0.1695407220318611]	0.0
0	[0.8373646950097853,0.16263530499021472]	0.0
+	-----+	-----+

only showing top 10 rows

0.6 Test Model Output

```
[16]: # Evaluate Multiple Matrix for Testing (For me, Model is not necessary, but in
      ↪ order to test my preprocessed data)
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

metrics = { "accuracy": "accuracy",
            "precision": "weightedPrecision",
            "recall": "weightedRecall",
            "f1": "f1"
          }

print("="*40)
print("Normal Cat+Num TRAIN RESULTS")
print("="*40)
for name, metric in metrics.items():
    evaluator = MulticlassClassificationEvaluator(
        labelCol="label",
        predictionCol="prediction",
        metricName=metric
    )
    print(f"{name}: {evaluator.evaluate(predictions):.4f}")
```

```
=====
Normal Cat+Num TRAIN RESULTS
=====
accuracy: 0.8866
precision: 0.8988
recall: 0.8866
f1: 0.8813
```