Big Data Course Project

SHAKIBA SADAT MIRBAGHERI

JULY 2023

Introduction

The purpose of this project is to analyze Bitcoin Heist Ransomware Address Dataset from UCI using Spark library.

- Setting up the VM
- Creating spark session
- Loading the dataset
- Preprocessing the dataset
- Creating RFM features
- Creating ML algorithms
- Evaluation of the models

Setting Up The VM

1. Installing Oracle and Vagrant
2. Creating a vagrant file and setting up the Ubuntu/jammy64
3. Creating and starting the VM
4. Connecting to the VM with vagrant user
5. Setting up the ports
6. Creating the link to access Jupyter notebook

```
PS C:\Users\Lenovo\desktop\test> vagrant init ubuntu/jammy64
```

```
PS C:\Users\Lenovo\desktop\test> vagrant up --provision
```

```
PS C:\Users\Lenovo\desktop\test> vagrant ssh -- -L 8888:localhost:8888
```

Creating spark session

Entry point to PySpark for loading and creating the dataset

```
!pip install pyspark
!pip install findspark
```

[1] Python

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pyspark
  Downloading pyspark-3.3.1.tar.gz (281.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 281.4/281.4 MB 4.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting py4j==0.10.9.5
  Downloading py4j-0.10.9.5-py2.py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.7/199.7 KB 15.8 MB/s eta 0:00:00
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.3.1-py2.py3-none-any.whl size=281845512 sha256=a346fedf564ab862fadbd80c96a9ad0191128291ee3d22cb8bea93cec0dfef74
  Stored in directory: /root/.cache/pip/wheels/43/dc/11/ec201cd671da62fa9c5cc77078235e40722170ceba231d7598
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9.5 pyspark-3.3.1
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting findspark
  Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Installing collected packages: findspark
Successfully installed findspark-2.0.1
```

```
os.environ["JAVA_HOME"] = "C:\Program Files\Java\jdk-19"
os.environ["SPARK_HOME"] = 'C:\\Users\\Nima_S_H\\Desktop\\BDAVM\\spark'
findspark.init()
```

[ ] Python

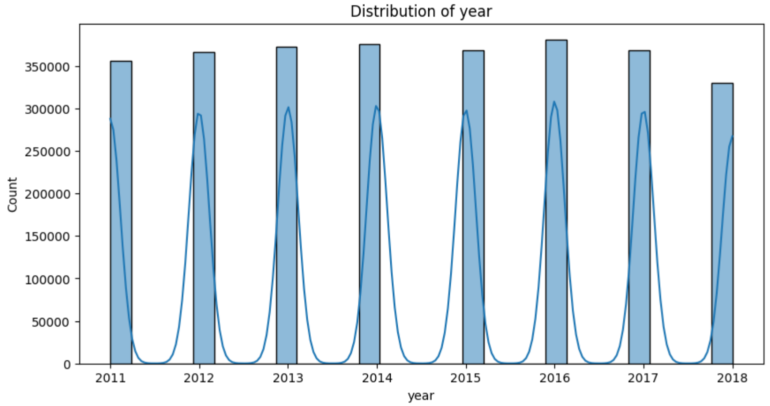
Data Understanding and Preprocessing

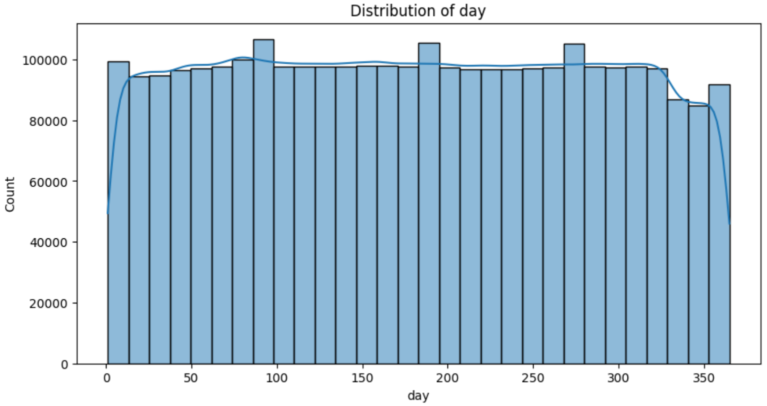
The dataset is quite substantial, containing 2,916,697 rows and 10 columns. Upon conducting a cleanliness check for any null or 'NaN' values, I found dataset to be in excellent shape with no missing values, which streamlined the preprocessing stage.

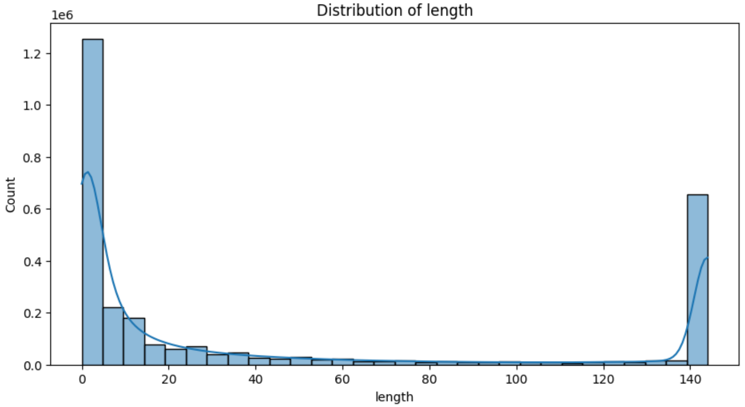
Attributes

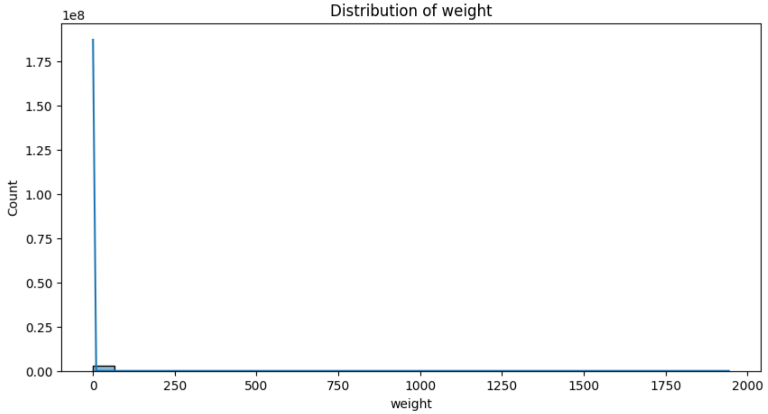
- **Address**: A string representing a Bitcoin address.
- **Year:** An integer indicating the year of the observation.
- **Day:** An integer representing the day of the year (1 to 365).
- **Length:** An integer value indicating a measurement (specific context not provided).
- **Weight:** A floating-point value (float) associated with the data point (specific context not provided).
- **Count:** An integer representing a quantity (specific context not provided).
- **Looped:** An integer representing a looped value (specific context not provided).
- **Neighbors:** An integer indicating the number of linked addresses.
- **Income:** An integer representing the amount of Satoshi received (smallest unit of Bitcoin).
- **Label:** A categorical string indicating the ransomware family name or "white" if not known to be ransomware.

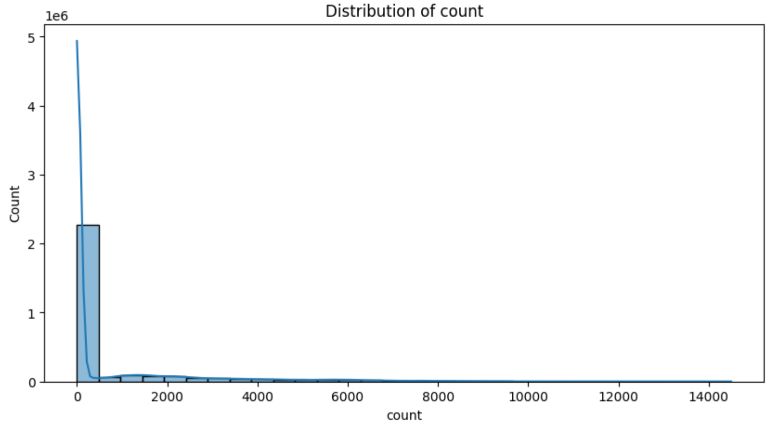
Chart

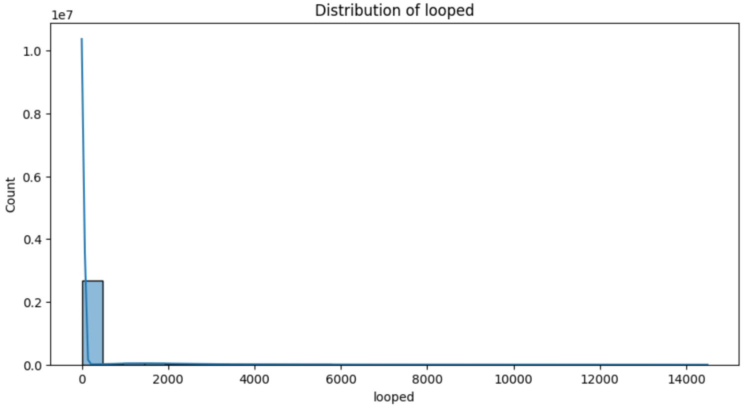

Distribution of year
Count
0
50000
100000
150000
200000
250000
300000
350000
2011
2012
2013
2014
2015
2016
2017
2018
year


Distribution of day
Count
0
20000
40000
60000
80000
100000
0
50
100
150
200
250
300
350
day


Distribution of length
1e6
Count
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
20
40
60
80
100
120
140
length


Distribution of weight
1e8
Count
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
250
500
750
1000
1250
1500
1750
2000
weight

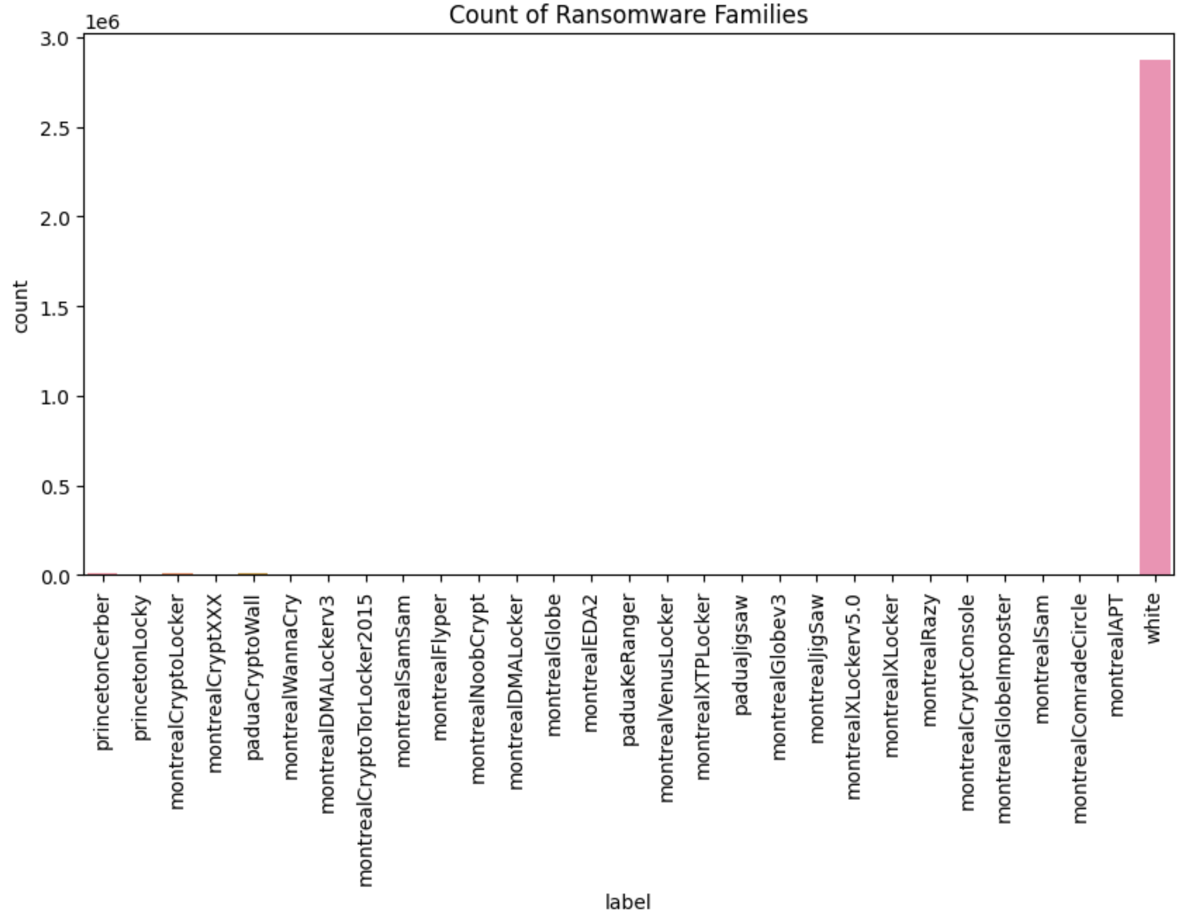

Distribution of count
1e6
Count
0
1
2
3
4
5
0
2000
4000
6000
8000
10000
12000
14000
count


Distribution of looped
1e7
Count
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000
10000
12000
14000
looped

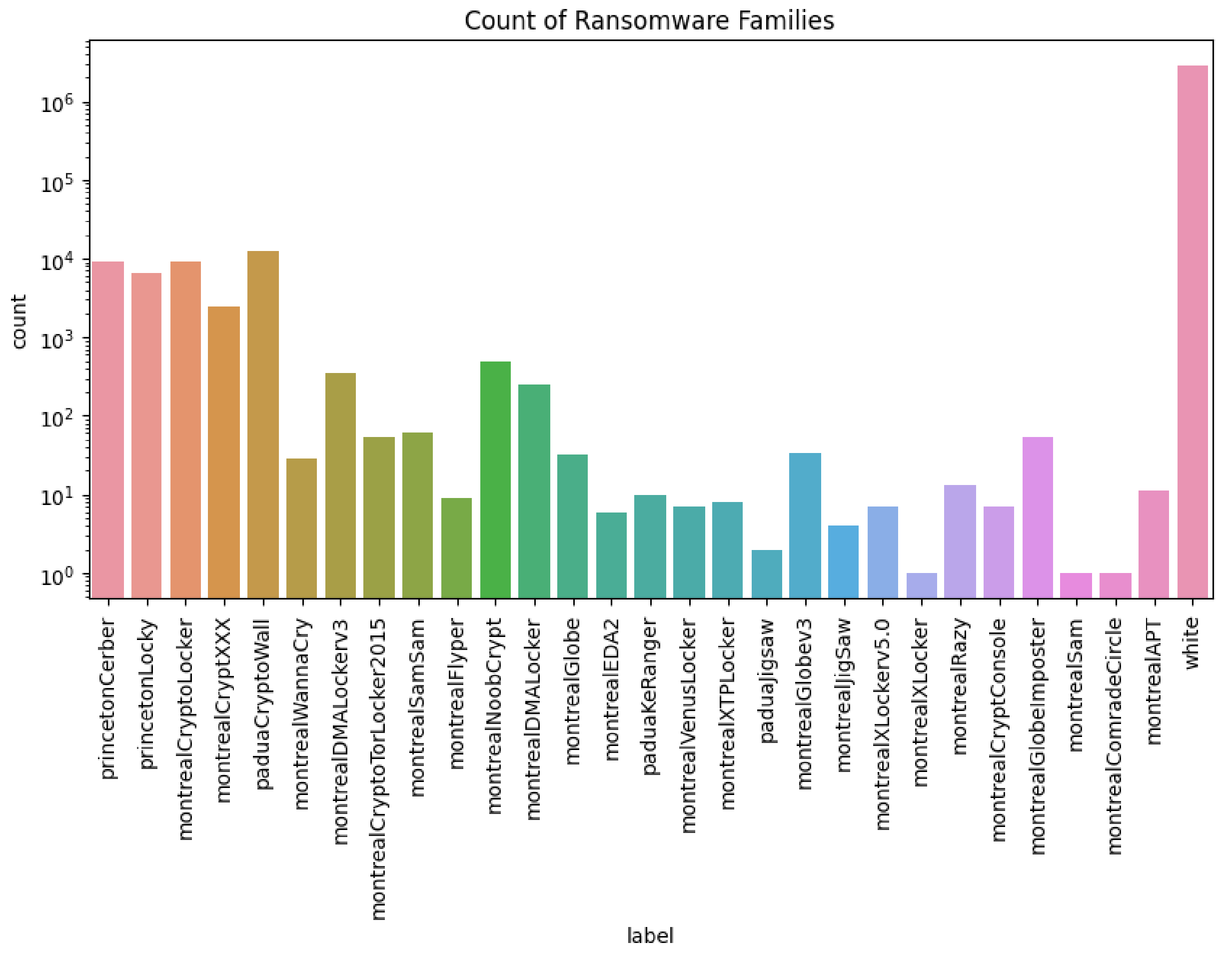
top 10 labels

Label	count
white	2875284
paduaCryptoWall	12390
montrealCryptoLocker	9315
princetonCerber	9223
princetonLocky	6625
montrealCryptXXX	2419
montrealNoobCrypt	483
montrealDMALockerv3	354
montrealDMALocker	251
montrealSamSam	62

Chart

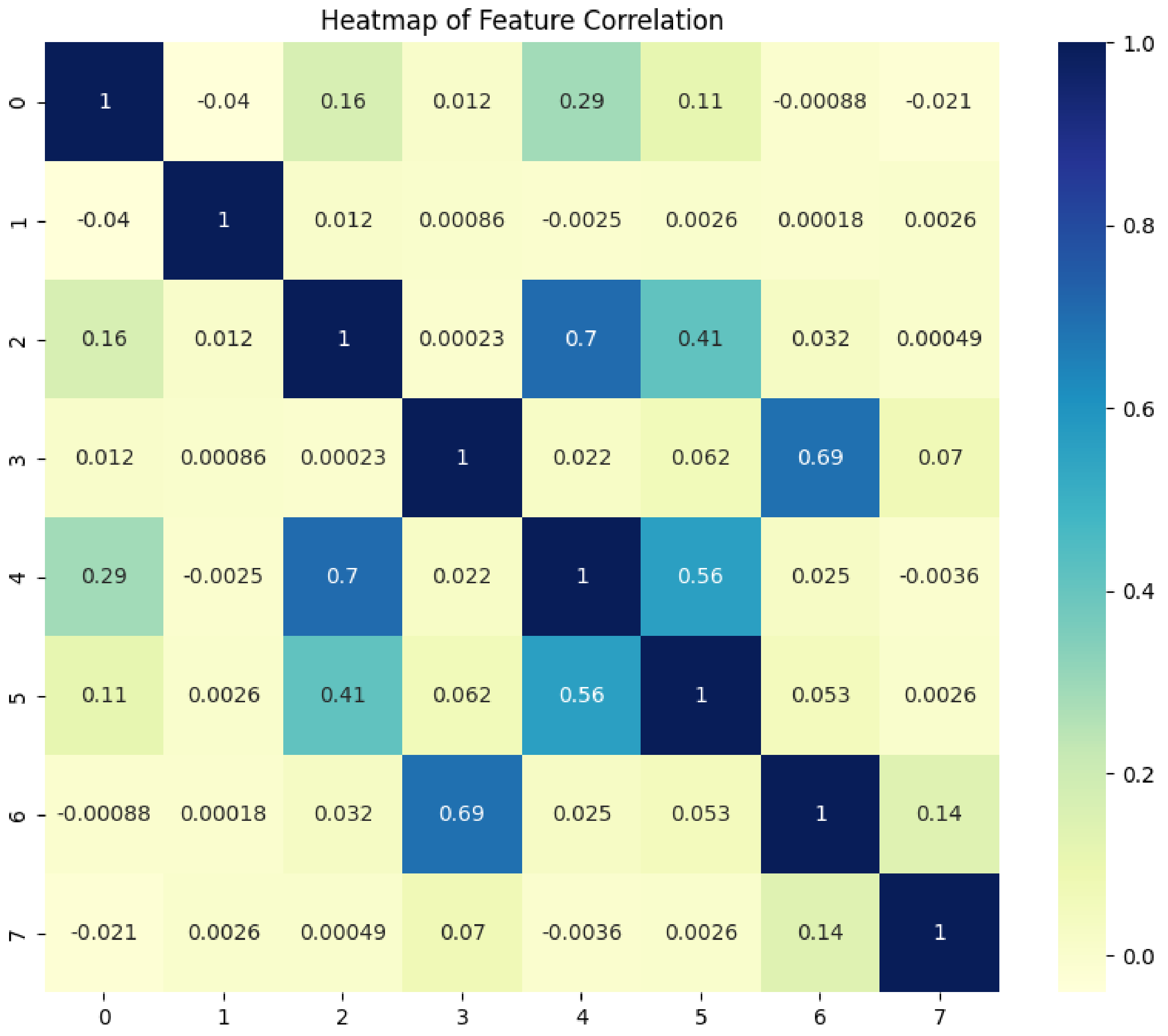


Count of Ransomware Families
1e6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
count
princetonCerber
princetonLocky
montrealCryptoLocker
montrealCryptXXX
paduaCryptoWall
montrealWannaCry
montrealDMALockerv3
montrealCryptoTorLocker2015
montrealSamSam
montrealFlyper
montrealNoobCrypt
montrealDMALocker
montrealGlobe
montrealEDA2
paduaKeRanger
montrealVenusLocker
montrealXTPLocker
paduaJigsaw
montrealGlobev3
montrealJigSaw
montrealXLockerv5.0
montrealXLocker
montrealRazy
montrealCryptConsole
montrealGlobeImposter
montrealSam
montrealComradeCircle
montrealAPT
white
label



Count of Ransomware Families
10^6
10^5
10^4
10^3
10^2
10^1
10^0
count
princetonCerber
princetonLocky
montrealCryptoLocker
montrealCryptXXX
paduaCryptoWall
montrealWannaCry
montrealDMALockerv3
montrealCryptoTorLocker2015
montrealSamSam
montrealFlyper
montrealNoobCrypt
montrealDMALocker
montrealGlobe
montrealEDA2
paduaKeRanger
montrealVenusLocker
montrealXTPLocker
paduaJigsaw
montrealGlobev3
montrealJigSaw
montrealXLockerv5.0
montrealXLocker
montrealRazy
montrealCryptConsole
montrealGlobeImposter
montrealSam
montrealComradeCircle
montrealAPT
white
label

Display Data



Transition to Unsupervised Learning

K-means

Feature Scaling

- In this phase, I prepare my dataset for unsupervised machine learning, specifically, K-means clustering. For optimal performance of my clustering algorithm, I need to standardize our features to ensure they're on the same scale.

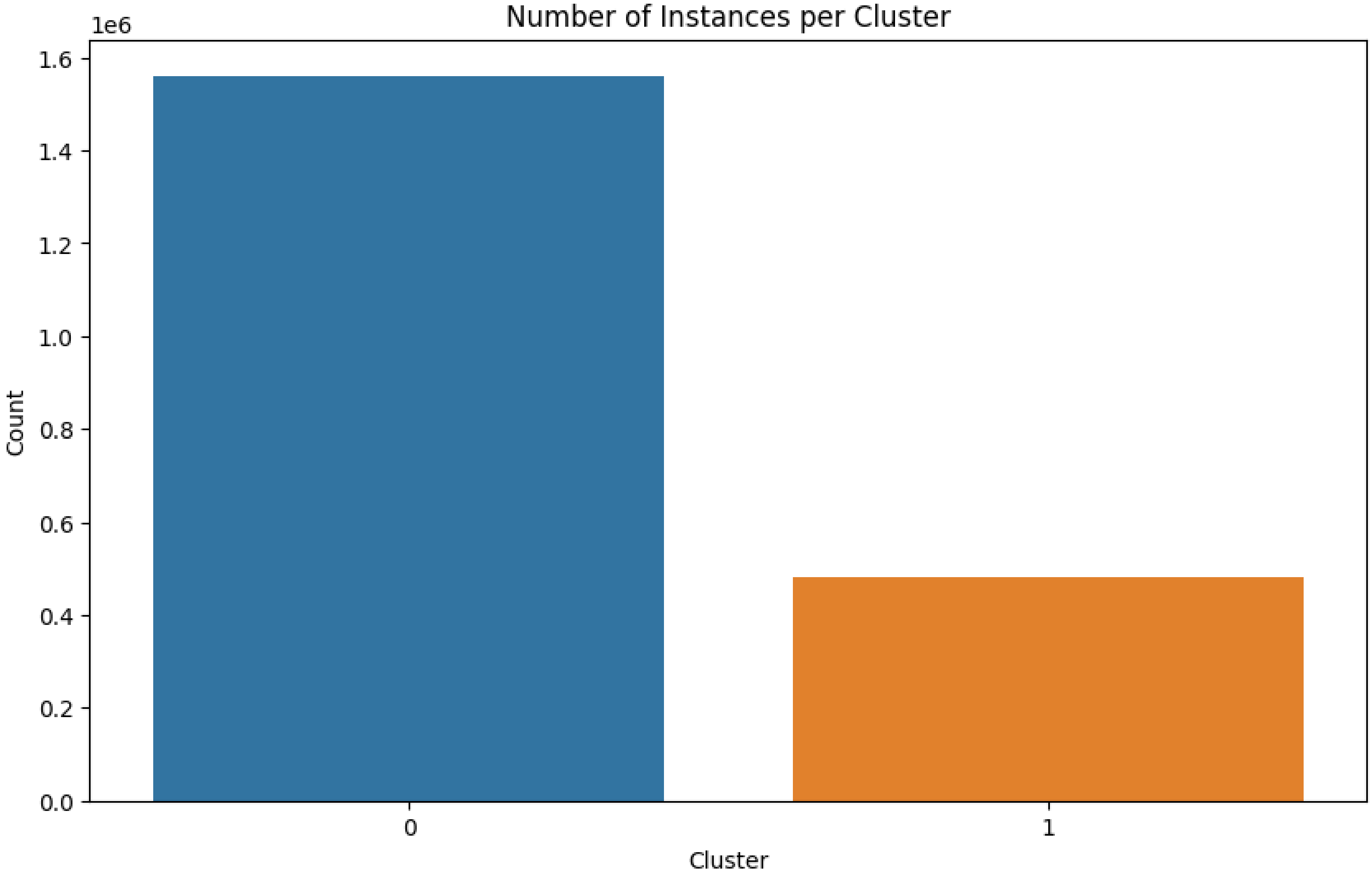
Assembling for Clustering

- To get my data ready for machine learning, I use the **VectorAssembler**, a transformer that combines a given list of columns into a single vector column. In this case, it's transforming my features into one single vector column, which I have labeled as 'features'.

Majority Label Identification

- Since K-means is an unsupervised learning algorithm, I need a way to evaluate its performance. Here, I group our predictions by the 'prediction' and 'label' columns and count the number of instances.
- Then join this DataFrame with my original DataFrame that contains my K-means predictions and labels. This allows me to compare the original label to the majority label for each cluster.

K-means

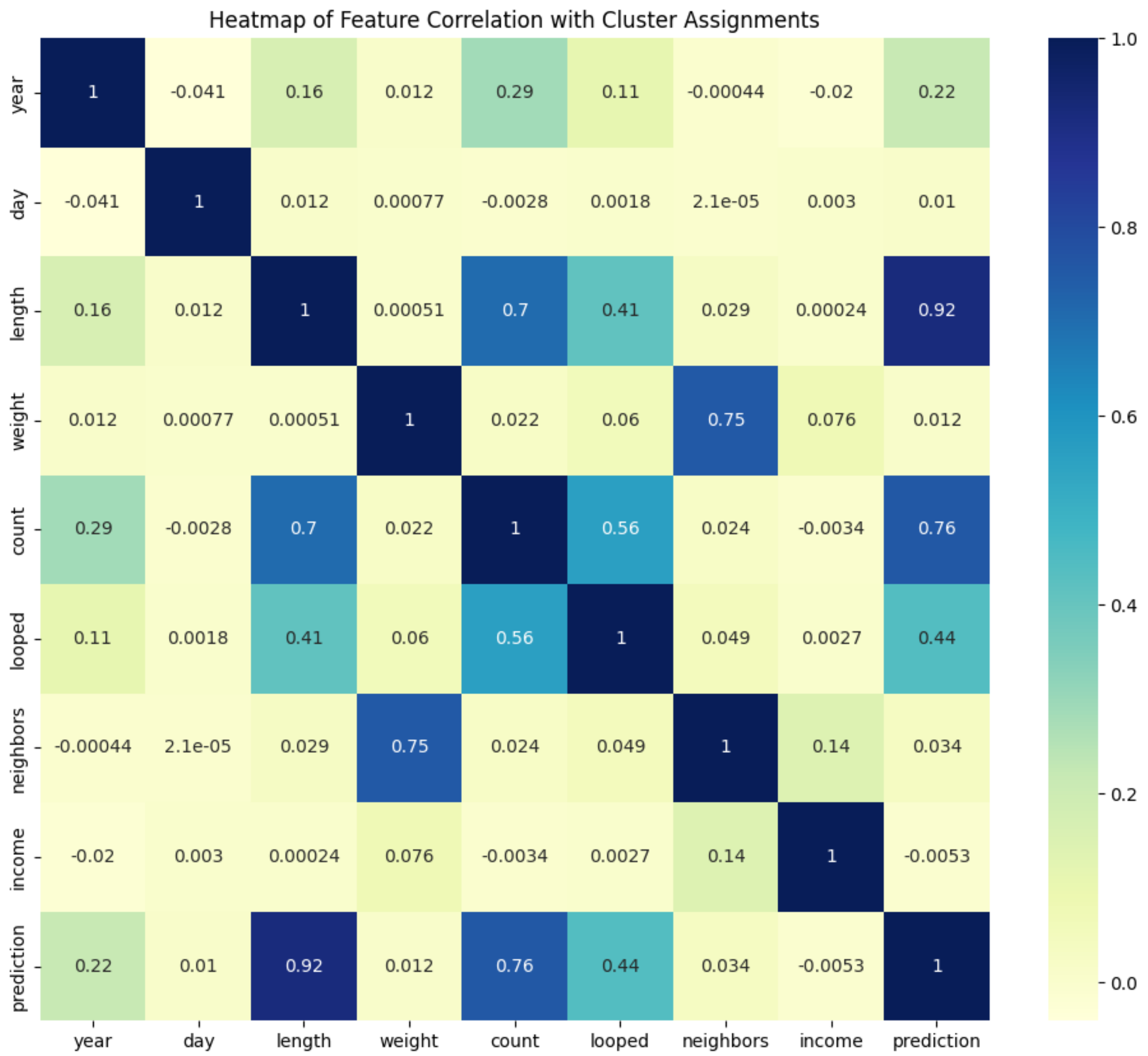

Number of Instances per Cluster
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Count
0
1
Cluster

Accuracy Evaluation

84.80% for training

85% for test

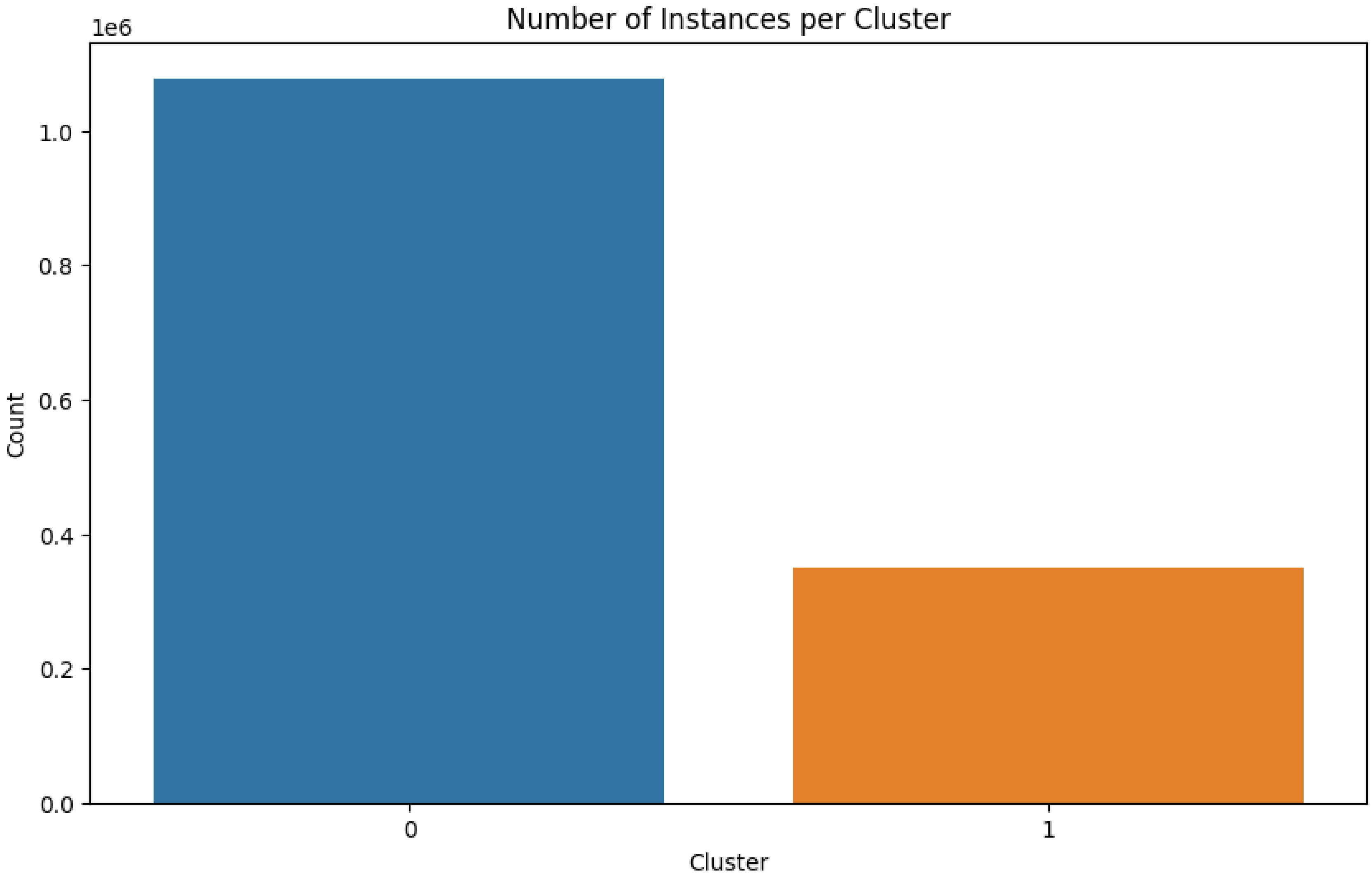
Analyzing Feature Correlation with Cluster Assignments



GMM

84.91% for training

15% for test



Concolusion

The K-means clustering method gave balanced accuracy results between the training and test sets, which indicates a good generalization of the model. On the other hand, the GMM, while it performed slightly better on the training set, had a significantly lower accuracy on the test set, suggesting a potential overfitting issue.

This comparison gives us valuable insights into the strengths and weaknesses of each method, and how they perform with this specific dataset. For future work, it might be beneficial to investigate the reasons behind the low test accuracy of the GMM and adjust the model's parameters or the dataset accordingly.

Thank you