Bachelor Project

Czech
Technical
University
in Prague

Faculty of Electrical Engineering, Department of Telecommunications

Identifying suspicious behavior of network devices using machine learning methods

Katsiaryna Zubaryk

Supervisor: Ing. Pavel Bezpalec, Ph.D.
May 2023

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Zubaryk Katsiaryna**	Personal ID number:	**491881**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Telecommunications Engineering**		
Study program:	**Electronics and Communications**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Identifying Suspicious Behavior of Network Devices Using Machine Learning Methods

Bachelor’s thesis title in Czech:

Identifikace podezřelého chování síťového zařízení pomocí metod strojového učení

Guidelines:

Analyse security incident detection and prevention systems. Design the coding of analysed incidents into an appropriate data space.
Perform a search for data clustering methods suitable for the proposed space. Assess their suitability for anomaly detection and apply one method in combination with the proposed metrics on the specified data. Find suitable visualization techniques to project the data into a lower dimension and use them to display the detected anomalies against the remaining data.

Bibliography / sources:

[1] AGGARWAL, CHARU C. a CHANDAN K. REDDY, ed. Data clustering: algorithms and applications. Boca Raton, Fla.: CRC Press, c2014, xxvi, 622 s. Chapman & Hall/CRC data mining and knowledge discovery series. ISBN 978-1-4665-5821-2.
[2] AXELSSON, Stefan. (2000). Intrusion Detection Systems: A Survey and Taxonomy.
[3] KOHOUT, J., ŠKARDA, Č., SHCHERBIN, K., KOPP, M., & BRABEC, J. (2021). A framework for comprehensible multi-modal detection of cyber threats.

Name and workplace of bachelor’s thesis supervisor:

Ing. Pavel Bezpalec, Ph.D. Department of Telecommunications Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **01.03.2023** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **16.02.2025**

Ing. Pavel Bezpalec, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Abstract

The purpose of this thesis is to investigate the application of machine learning techniques to detect suspicious behavior of network devices. With the increasing complexity of cyber threats, it has become imperative to develop effective methods to detect abnormal activities that could potentially indicate security or malicious network activities. This research focuses on the study of machine learning algorithms and approaches to detect and classify suspicious behavior of network devices.

Keywords: Clustering, Unsupervised, feature selection…

Keywords:

Supervisor: Ing. Pavel Bezpalec, Ph.D.
Technická 2, Praha

Abstrakt

Cílem této práce je prozkoumat použití technik strojového učení k detekci podezřelého chování síťových zařízení. S rostoucí složitostí kybernetických hrozeb se stalo nezbytným vyvinout účinné metody pro detekci abnormálních aktivit, které by mohly potenciálně indikovat bezpečnostní nebo škodlivé aktivity v síti. Tento výzkum se zaměřuje na studium algoritmů a přístupů strojového učení k detekci a klasifikaci podezřelého chování síťových zařízení.

Klíčová slova: Clusterind, učení bez dohledu, výběr prvků…

Klíčová slova:

Překlad názvu: Identifikace podezřelého chování sítového zařízení pomocí metod strojového učení

Contents

Figures

Tables

Acknowledgements

would like to thank my supervisor Ing. Pavel Bezpalec, Ph.D. and to Lukáš Bátrla from CISCO Systems for there professional guidance and valuable advice.

Declaration

declare that I have prepared the submitted work independently and that I have listed all the information sources used in accordance with in accordance with the Methodological Guideline on the Observance of Ethical Principles in the Preparation of a University Thesis.

Chapter 1

Introduction

Identification of suspicious device action is a vital aspect of modern-day security systems. With the increasing use of Internet of Things (IoT) devices in our daily lives, it has become necessary to take reliable security measures to protect our privacy and security. Unsupervised and supervised identification of suspicious device activity are measures that can help us detect any abnormal behavior of IoT devices and prevent any security breaches.

1.1 Unsupervised identification

Unsupervised identification is a machine learning technique that finds and identifies patterns or anomalies in data without the need for labeled training examples. It is the opposite of supervised learning, where a model is trained on data with explicit labels showing the correct answers[1].

The use of unsupervised identification methods is becoming popular in data analysis, especially for detecting patterns and relationships that may previously have been undetected. According to Ella Rabinovich and Shuly Wintner from Department of Computer Science at the University of Haifa, "We then show that unsupervised classification is highly accurate on this task"[2] which emphasizes the effectiveness of these methods in detecting hidden ideas in datasets. Methods of unsupervised identification are [3]:

1. Dimensionality reduction. This is a technique that is used when the number of features, or dimensions, in a given dataset is too large. It reduces the number of inputs to an acceptable size while maintaining the integrity of the dataset as much as possible[5]. It is usually used in the preprocessing phase of the data, and there are several different methods of dimensionality reduction, such as:

   a. Principal component analysis (PCA)[3]

   b. PCA Kernel[3]

2. Clustering. This is a data mining technique which groups unlabeled data based on their similarities or differences. Clustering algorithms are used to process raw, unclassified data objects into groups represented by

structures or patterns in the information[5]. Clustering algorithms can be categorized as:

a. K-means[3]
b. Hierarchical cluster analysis (HCA)[3]

With the exponential growth of data in recent years, it has become increasingly difficult for humans to manually identify patterns and extract insights from them. This is where unsupervised identification methods can play a crucial role. These methods allow computers to automatically detect patterns in large datasets that may not be obvious to humans, saving valuable time and resources.

Unsupervised identification is widely used in a wide variety of work areas, including:

1. Cybersecurity: In this field, detection of anomalous or malicious behaviour in computer networks, security systems is required. Unsupervised learning models can detect unusual network packets, burst traffic or unexpected access to systems and prevent cyberattacks[3].

2. Financial services sector: The financial industry also needs to detect fraudulent transactions, anomalies in transactions or changes in patterns of behavior in markets. Unsupervised learning models can help identify unusual financial transactions and take timely action[4].

3. Medicine and healthcare: In the medical field, unsupervised identification can be applied to detect outliers in medical data, identify rare or new diseases, and analyse images. Unsupervised learning models can help doctors and researchers detect pathologies or anomalies that may go unnoticed[5].

4. Marketing and data analysis: In marketing and data analysis can be used to segment customers, identify behavioral patterns, predict consumer preferences or personalise recommendations. Unsupervised learning models can help identify groups of similar customers and determine their characteristics for a more effective marketing strategy[5].

In conclusion, unsupervised identification techniques offer a promising solution for organizations that want to extract valuable insights from large and complex datasets. Using machine learning algorithms, such as clustering and anomaly detection, may help reduce human error and ultimately save valuable time and resources. These methods are especially useful when working with unlabeled data or when the goal is to identify previously unknown patterns in the data.

1.2 Supervised identification

Supervised identification is the process of identifying patterns, or anomalies in the data using labeled examples or a previously defined target variable.

In supervised learning, a model is trained on a dataset where each data instance is associated with a known label or target value. The model learns the relationship between the input features and the corresponding labels, which allows it to make predictions or classifications on new, unseen data[1].

Various algorithms and techniques are used in supervised machine learning processes:

1. Linear regression. It is used to identify the relationship between a dependent variable and one or more independent variables and is typically leveraged to make predictions about future outcomes[6].
2. Logistic regression. While linear regression is leveraged when dependent variables are continuous, logistic regression is selected when the dependent variable is categorical, meaning they have binary outputs[6].
3. K-nearest neighbor. Also known as the KNN algorithm, is a non-parametric algorithm that classifies data points based on their proximity and association to other available data. This algorithm assumes that similar data points can be found near each other. As a result, it seeks to calculate the distance between data points[6].

Those techniques have proven useful in various fields such as:

1. Image- and object-recognition[5]. It can be used to identify objects in pictures or videos. For example, this method is used in machines with an automatic control system. Or it can be used in medicine where existing medical images of patients will be used as labels.
2. Customer sentiment analysis[5]. It is possible to predict the sentiment or opinion expressed by customers about a product, service or brand based on their reviews or feedback.
3. Spam detection[5]. Classifying emails or messages based on their content.

In conclusion, supervised identification is an essential process in machine learning that allows for the accurate classification of information. The use of human guidance in the training phase ensures that machines can identify patterns and make predictions with high accuracy, making it a valuable tool in fields such as automation, healthcare, and marketing.

1.3 Selected identification for thesis

In this thesis I will use unsupervised identification. Because the provided data does not contain the specified labels, and the task of research is to find anomalies among the various devices. I will use the clustering method. This is one of the unsupervised learning methods that I wrote about in chapter 1.1. A separate paragraph about clustering is described in chapter 4.

Chapter 2

Introduction to data analysis for unsupervised identification

Data analysis is an important part of unsupervised identification. It helps to detect hidden patterns, structures and anomalies in the data without the use of explicit labels or classification. In this chapter I will look at data analysis in the context of unsupervised identification, focusing on a specific dataset.

2.1 Programming language

I have used Python programming language, Jupyter notebook platform and relevant libraries for data analysis, machine learning and visualisation. The specified libraries are in figure 2.1. The history of programming languages used for machine learning tasks goes back more than 60 years. From the time the first machine learning algorithms were created. During the development of this industry, libraries and programming languages began to be created to simplify the work of creating machine learning algorithms, and later the creation of artificial intelligence and neural networks. As a result of this evolution, the most successful language became Python, which is still one of the most widely used for development in the industry. Python is voted by data scientists as the "language most desirable to learn" in 2018 StackOverflow survey [7].

Programming expertise in the organisation, interoperability with existing code/data frameworks might be reasons to stick to traditional languages such as C++ or Java. But the advantages that new-age languages such as Python and R bring to Machine Learning are plenty. Ease of coding, strong supporting developer community, and excellent data manipulation features are key reasons for Python's suitability for Machine Learning [8].

2.2 Description of the dataset

The dataset contains information about various events. Each record represents an individual event and contains the following attributes (see table 2.1):

```
import pandas as pd
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
from collections import defaultdict
from scipy.spatial import distance
import openpyxl
import itertools
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.metrics import calinski_harabasz_score
import plotly.graph_objects as go
import plotly.offline as pyo
import scipy.cluster.hierarchy as shc
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import linkage,
   dendrogram, fcluster
```

Figure 2.1: Python libraries

Attributes	Description
eventId	the unique identifier of the event
eventTitle	the name of the event
assetId	the identifier of the device, associated with the event
sourceId	the source identifier of the event
startDay	the date of the observation event on the device
from	start time of the event
to	time the event ended

Table 2.1: Description of the dataset

2.3 Data analysis

I started by loading the dataset into an appropriate software environment and did the necessary preprocessing of the data, such as removing null values (see figure 2.2).

In the next step, I analysed the main characteristics of the data. Such as how much data we have. How many unique values are in the columns carrying information about the name of the observed event and on which devices they appeared (see figure 2.3). The result is given in the table 2.2.

I was looking if one *eventTitle* had more than one *eventId* (figure 2.4). I had assumed that if more than one *eventId* per *eventTitle*, it could be seen as some kind of suspicious activity. The output is in the table 2.3. Based on the result, we can see that the most frequent events are: Anomalous destination, Non-user activity, Unexpected application atd others. Which makes my hypothesis correct.

```
df=pd.read_csv(r'D:\A_CVUT_FEL\Bakalarska_prace\
    security_event_data.csv')
df = df.dropna()
```

Figure 2.2: Loading the dataset

```
print('all the data we have :',len(df['eventTitle'])
   )
print(len(df['eventId'].unique()))
print(len(df['eventTitle'].unique()))
print(len(df['assetId'].unique()))
```

Figure 2.3: Python code for unique data

The histogram was created (see code 2.5) to give a more accurate view of the unique *eventTitles* to which several *eventIds* belong (figure 2.6). The histogram shows the distribution of *eventTitles* and how many *eventIds* relate to them. Columns with lower heights have two or more different *eventIds*. Which can be seen as suspicious events. Because each unique *eventTitle* should also have a unique *eventId*.

Further analysis was done on the *eventTitles* (look at 2.7. I found the events that appear most frequently in the dataset. Later, during a search for a outliers, it will be possible to see whether the suspicious *assetId* has these events. Because they appear in the dataset more often than others. The result is given in the table 2.4.

Next, I decided to see how long each event lasts (code 2.8). The result is in the table appendixA.1. From the table A.1 we can see that 54 events out of 180, which is 30% last more than 0 ms. Since most of the events lasted 0 ms, the events from the table can be considered suspicious. All events that lasted longer than 0 milliseconds were extracted (see code 2.9) from the 2.10.

To begin with, calculations were made on how many unique *assetIds* we have in our data. And how many different *eventTitles* belong to them (see the figure 2.11).

Next, find the average number of titles per asset. And find the asset that has more *eventTitles* than the average (see code 2.12). The result of running the code is: *The assetId which have more then average eventTitle is: b8368210-4879-3d5b-8247-2746b956109e 28.* Perhaps this particular asset will turn out to be the abnormal one.

After analysing the data, a dataset must be compiled to search for anomalies among the devices or assets. For this, I took the unique *assetIds* and unique *eventTitles* (see figure 2.13). Then they were grouped together in such a way that each *assetId* had several events associated with it (see figure 2.14). This was created to further dataset (see figure 2.15) where if an event is present in a given *assetId* it is set to one, otherwise it is set to zero.

Prepare a dataset in the form of a one-hot-encoded vector for each asset (see figure 2.16). After analysing the data and compiling the dataset, we can now

	Count
all the data we have	886186
Unique eventId	180
Unique eventTitle	131
Unique assetId	4923

Table 2.2: Amount of data

```python
idForTitle = df.drop_duplicates().set_index('eventId
    ')['eventTitle'].to_dict()
values=idForTitle.values()
countIdforTitle=dict(Counter(values))
sameName={}
for i in countIdforTitle:
    if countIdforTitle[i]>1:
        print(i, countIdforTitle[i])
        sameName[i]=countIdforTitle[i]
```

Figure 2.4: Events with lots of eventIds

proceed to the main topic of this thesis. And that is to find anomalies among all devices or among *assetIds*. But In real-life data science problems, it is almost rare that all the variables in the dataset are useful for building a model. Adding redundant variables reduces the model's generalization capability and may also reduce the overall accuracy of a classifier. Furthermore, adding more variables to a model increases the overall complexity of the model. In order to reduce the dimensionality of a dataset, a feature selection is used. The next chapter of the thesis is written about this.

2.4 Visualisation technique

To visualise the data in this section, I have used histograms with the help of the library *matplotlib.pyplot*. A histogram is a way of presenting statistical data graphically - in the form of a bar chart. It is sometimes called the frequency distribution, as the histogram shows the frequency of occurrence of the measured values of an object. Histograms in Python present several advantages that help visualise and analyse data:

1. Ease of use: Python has many libraries, such as Matplotlib and Plotly, which provide a simple and intuitive interface for creating histograms.

2. Big data visualisation: Histograms can visualise large amounts of data. This helps trends in the data that might be missed by simply looking at numerical values.

3. Interactivity: Some libraries in Python, such as Plotly, allow you to create interactive histograms that can be zoomed in, selected specific

Event	Count
Unusual file extension	2
Known malicious hostnames	3
Commands that run scripts	2
Usage of wmic	2
Unusual execution directory	2
Non-user activity	9
Anomalous destination	18
Repetitive requests or burst	3
TLS inconsistency	4
Unexpected application	5
HTTP to IP address	2
Unexpected user destination	4
Inconsistent user time activity	5
Anomalous HTTP	2

Table 2.3: Events with lots of eventIds

intervals or categories for more detailed analysis. These histograms will be used in the chapter about clustering.

Histograms are powerful data visualisation tools that help to explore, analyse and understand data. They are widely used in a variety of fields including statistics, finance, marketing and more.

```python
dict_countIdforTitle = {}
for i in countIdforTitle.values():
    dict_countIdforTitle[i] = dict_countIdforTitle.get(i, 0) + 1

counts = list(dict_countIdforTitle.values())
titles = list(dict_countIdforTitle.keys())

# Sort the titles and counts based on the counts in descending order
sorted_indices = sorted(range(len(counts)), key=lambda k: counts[k], reverse=True)
titles = [titles[i] for i in sorted_indices]
counts = [counts[i] for i in sorted_indices]

plt.bar(range(len(titles)), counts)

plt.xlabel('Number of eventIds')
plt.ylabel('Counts')
plt.title('Histogram')

plt.xticks(range(len(titles)), titles)

for i, count in enumerate(counts):
    plt.text(i, count + 1, str(count), ha='center')

plt.tight_layout()
plt.show()
```

Figure 2.5: Histogram for code 2.6

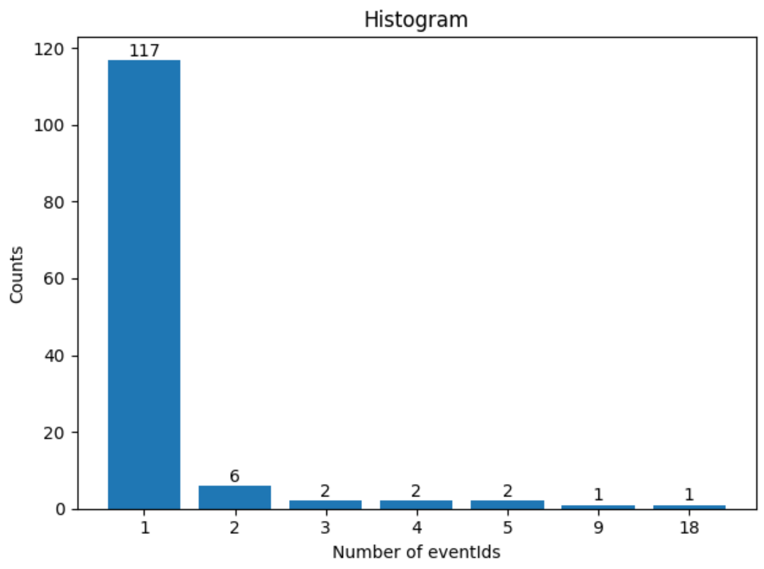


Figure 2.6: How many eventTitles have the given number of eventIds.

```python
mainEventTitle = dict(Counter(df['eventTitle']))
average=0
for i in mainEventTitle:
    average+=mainEventTitle[i]
average=average/len(mainEventTitle)
print(average)
maxx=0
for i in mainEventTitle:
    if mainEventTitle[i]>average:
        print(i, mainEventTitle[i])
        if maxx < mainEventTitle[i]:
            maxx =mainEventTitle[i]
            r=i
print('The main eventTitle is:', r, maxx)
```

Figure 2.7: Events that have more than the average occurrence value

Event	Occurrence value
Product Update Started	7151
ClamAV exceeded max file scan time	10770
Product Update Failed	7078
Quarantine Delete Started	16898
Quarantine Delete Success	13649
HTTPS communication	177248
Known site	208838
No Server Traffic	10513
Non-user activity	54927
Domain letter to non-letter ratio	69767
Data upload	6820
Anomalous destination	132513
TLS inconsistency	13733
Unexpected application	61637
Multimedia streaming	11998
Communication blocked by proxy	17323
Inconsistent user time activity	11221
The main eventTitle is: Known site	208838

Table 2.4: Events that have more than the average occurrence value

```python
dict_from = df.groupby('eventTitle')['from'].unique
   ().to_dict()
dict_to = df.groupby('eventTitle').to.unique().
   to_dict()
minus=0
dict_lasts=defaultdict(list)
for i in dict_to:
    for j, jj in enumerate (dict_to[i]):
        minus=dict_to[i][j]-dict_from[i][j]
        dict_lasts[i].append(minus)
notNullTime =defaultdict(list)
for i in dict_lasts:
    if len(dict_lasts[i])!=1:
        if(all(x==dict_lasts[i][0] for x in
           dict_lasts[i]))==False :
                    notNullTime[i].append(dict_lasts
                       [i])
        else:
            if 0 not in dict_lasts[i]:
                notNullTime[i].append(dict_lasts[i])
    else:
        if dict_lasts[i]!=[0]:
            notNullTime[i].append(dict_lasts[i])
```

Figure 2.8: Python code for considering the timing of eventTitles

```
dict_source = df.groupby('eventTitle').sourceId.
    unique().to_dict()
source={}
for i in notNullTime:
    if list(dict_source[i])[0] not in source: source
        [list(dict_source[i])[0]]=1
    else: source[list(dict_source[i])[0]]+=1
counts = list(source.values())
titles = list(source.keys())

plt.bar(titles, counts)
plt.xlabel('Source')
plt.ylabel('Event counts')
plt.title('Sources that last longer than 0 ms')
plt.show()
```

Figure 2.9: Histogram code for longer-lasting events

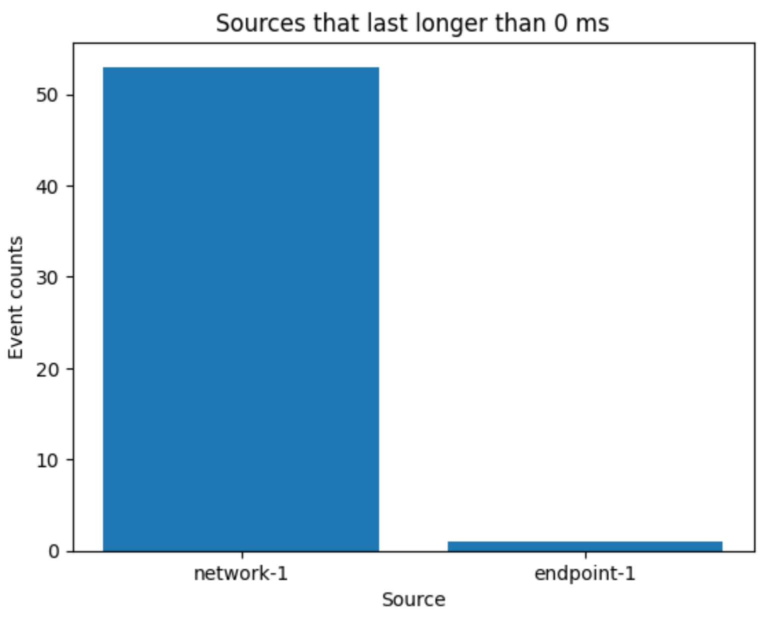


Figure 2.10: Events from sources that last longer than 0 ms

```
countSourceId=df.groupby('assetId').eventTitle.
    nunique()
countSourceId=countSourceId.to_dict()
```

Figure 2.11: Python code for longer-lasting events

```
countSourceId = df.groupby('assetId').eventTitle.
   nunique().to_dict()
average=0
for i in countSourceId:
    average+=countSourceId[i]
average=average/len(countSourceId)
maxx=0
for i in countSourceId:
    if countSourceId[i]>average:
        if maxx < countSourceId[i]:
            maxx =countSourceId[i]
            r=i
print('The assetId which have more then average
   eventTitle is:', r, maxx)
```

Figure 2.12: Code to find assetId which have more then average eventTitle

```
uniqueAssets = df['assetId'].unique()
uniqueEventTitle = df['eventTitle'].unique()
```

Figure 2.13: Python code to find unique values

```
groupedData = df.groupby('assetId').eventTitle.
   unique()
groupedData = groupedData.to_dict()
```

Figure 2.14: Python code for grouped data

```
dataset=defaultdict(list)
for i in uniqueAssets:
    for j in uniqueEventTitle:
        if j in groupedData[i]: dataset[i].append(1)
        else: dataset[i].append(0)
```

Figure 2.15: Python code for a practical dataset

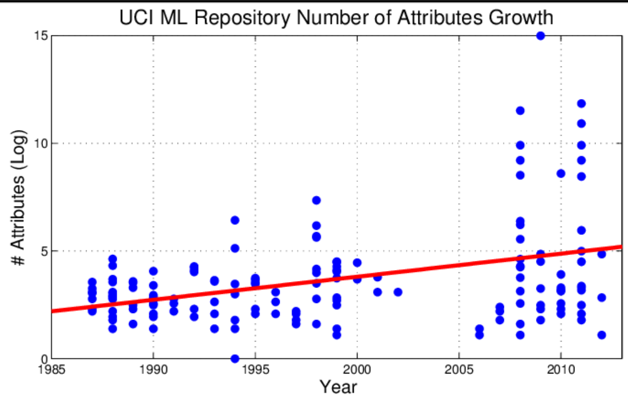
```
oneEncodedVector = pd.DataFrame(dataset).T
```

Figure 2.16: Python code for one-hot-encoded vector

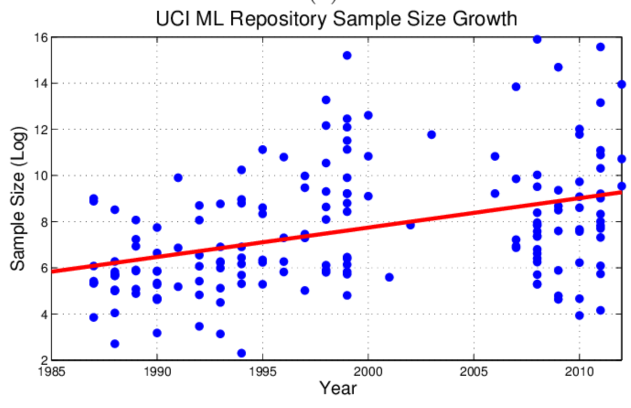
Chapter 3

Feature selection

The growth of high-throughput technologies nowadays has led to exponential growth in the harvested data with respect to dimensionality and sample size. As a consequence, storing and processing these data becomes more challenging [11]. Figure 3.1 shows the trend of this growth for UCI Machine Learning Repository.



(a)



(b)

.

Figure 3.1: Plot (a) shows the dimensionality growth trend in UCI Machine Learning Repository from mid '80s to 2012 while (b) shows the growth in the sample size for the same period [11]

The production of an ever-expanding amount of data on a daily basis has made feature selection a fundamental aspect of both data analysis and machine learning. Machine learning relies on feature selection to identify the most significant and informative attributes from large-scale datasets. As highlighted by Younes Bouchlaghem, Yassine Akhiat and Souad Amjad (2022) [10] from Department of Informatics in Morocco, feature selection is a crucial area of study in data mining and machine learning that helps reduce complexity, increase interpretability, and improve model performance[10].

There numerous feature selection methods that have been categorized into three groups: filters, wrappers and embedded [10].

1. Filter model When it comes to selecting the most relevant features for machine learning models, filter methods are often favored due to their simplicity and efficiency. To cause the classification model, it first ranks features individually based on a particular criterion measure such as distance, Pearson correlation, and entropy. Second, it selects the best-ranked features using a threshold value. The remaining features are deemed to be unnecessary and uninformative [10]. However, some researchers argue that filter methods may not always be the best approach since they do not take into account the interactions between different variables. In these cases, wrapper or embedded methods may provide better results by evaluating subsets of features together rather than individually [11].

   Let's, discuss some techniques from the filter model.

   a. Information gain.

      Information gain is a measure used in the selection of features for categorical variables. It determines the reduction of entropy or uncertainty about the target variable when a particular attribute is known [13].

   b. Chi-square test.

      The chi-square test is used to evaluate the independence between two categorical variables. In feature selection, it measures the relationship between each feature and the target variable. The chi-square test calculates a test statistic that compares the observed frequencies of the categories of each feature to the expected frequencies under the assumption of independence. Features with a higher chi-square test statistic or a lower p-value indicate a stronger association with the target variable [13].

   c. Correlation matrix.

      Correlation is a measure of the linear relationship between 2 or more variables. Through correlation, we can predict one variable from the other. The logic behind using correlation for feature selection is that good variables correlate highly with the target. Furthermore, variables should be correlated with the target but uncorrelated among themselves [13].

If two variables show a correlation, we can make predictions about one variable based on the other. Consequently, if two variables are correlated, only one of them is needed for the model, as the second variable provides no additional information.

2. Wrapper feature selection model.

   Wrapper feature selection methods involve evaluating subset of features using a particular machine learning algorithm. Instead of relying on statistical measures like correlation or information gain, the wrapper method estimate subsets of features through the formation and evaluation of a machine learning model. The selection process involves iterating over various combinations of features and evaluating their performance based on a chosen evaluation metric, such as accuracy or error rate. This method may be computationally intensive but produces more accurate results because it takes into account the interaction between the features in the context of the chosen algorithm [10] and [13].

3. Embedded model.

   Embedded methods combines the advantageous aspects of both Filter and Wrapper methods. Embedded methods are selecting features based on the learning procedure of the machine learning model. However, Wrapper methods consider unimportant features iteratively based on the evaluation metric, while embedded methods perform feature selection and training of the algorithm in parallel[13] and [14].

In the realm of machine learning, feature selection is a critical process that aids in pinpointing the most vital characteristics for predictive models. By selecting meaningful features, it can reduce computational expenses while simultaneously enhancing model accuracy and generalization performance. The approach involves numerous techniques which are utilized based on how intricate the dataset is; these include filter methods, wrapper methods and embedded methods. In my research I used a filter model, and in particular, I used a correlation matrix.

3.1 Correlation matrix

Correlation matrix feature selection has become an essential tool in the data analysis toolbox. This technique allows for identifying and selecting the most relevant features from a large dataset while considering their interdependencies. It is particularly useful when dealing with high-dimensional datasets, where the number of variables is often much larger than the number of observations. The idea behind this method is to measure the correlation between each pair of features and exclude those that are highly correlated or redundant. Correlation values range from -1 to 1, where -1 indicates a perfect negative correlation, 1 indicates a perfect positive correlation and 0 indicates no

```
strong_corr = oneEncodedVector.corr().abs() > 0.8
strong_corr_vars = strong_corr.sum()[strong_corr.sum
   () > 1].index
strong_corr_matrix = oneEncodedVector[
   strong_corr_vars]
plot_corr = strong_corr_matrix.corr()
plt.figure(figsize=(100,35))
sns.heatmap(plot_corr, annot =True)
```

Figure 3.2: Code for correlation matrix

correlation [16]. The correlation coefficient *p(x, y)* between two random vectors x and y is defined as (Equation (3.1):[15]:

$$\text{p(x, y)} = \frac{\sum_{i=1}^{n}((x_i - \text{mean}(x)) \cdot (y_i - \text{mean}(y)))}{(n-1) \cdot \text{var}(x) \cdot \text{var}(y)} \tag{3.1}$$

Where var(x), var(y) denotes the variance of the random vector x or y. And mean(x), mean(y) represent the mean values of variables x and y, respectively.

Then remove redundant variables that are highly correlated with other variables. If two variables have a very high correlation coefficient, this may indicate that they contain similar information and only one of them should be left. In this way, we can keep variables that carry information while reducing the dimensionality of the data without significant loss of information. A strong correlation is considered to be a value between 0.8 for positive correlation and -0.8 for negative correlation. However, it should be remembered that correlation coefficients measure only linear relationships between variables and do not reflect possible non-linear relationships [16]. In my thesis, I applied the correlation matrix to an existing dataset (see figure 3.2).

Correlation matrix result is available in appendix A.2. From here I find a strong correlation in the matrix and select only one value to remove from the dataset (figure 3.3). Remove unnecessary columns from the dataset (figure 3.4).

After all the manipulation, I have a dataset for clustering.

```python
eventTitle_pairs=defaultdict()
eventTitle_numb=[]
for i in plot_corr.columns:
    for j in strong_corr.index:
        if i!=j:
            if (strong_corr[i][j]>=0.8) | (strong_corr[i][j]<=-0.8):
                if (uniqueEventTitle[j], uniqueEventTitle[i]) not in eventTitle_pairs:
                    eventTitle_pairs[(uniqueEventTitle[i], uniqueEventTitle[j])]=strong_cor[i][j]
                    eventTitle_numb.append(j)
```

Figure 3.3: Values with a strong correlation coeficient

```python
updateDataSet = oneEncodedVector.drop(
    eventTitle_numb, axis=1)
```

Figure 3.4: Updated dataset

Chapter 4

Clustering

The machine learning technique known as clustering is a highly universal method that groups data points based on their similarities or distance measures [11]. Clustering's usefulness derives from its adaptability to various fields such as computer science, data science, statistics, pattern recognition, artificial intelligence, and machine learning [17]. Clustering provides a valuable tool for machine learning applications such as dynamic trend detection, biological data analysis, and customer segmentation [11]. Despite its usefulness in various fields including healthcare and finance sectors, there are still some challenges associated with clustering algorithms like determining the optimal number of clusters needed when working with large datasets. However, researchers continue to develop new methods to overcome these limitations.

Over the past years, dozens of data clustering techniques have been proposed and implemented to solve data clustering problems [17]. In general, clustering analysis techniques can be divided into three main groups:

1. Distance-Based Algorithms These algorithms rely on the concept of distance or similarity between data points to group similar instances together. The distance measure plays a fundamental role in these algorithms, as it determines the proximity or dissimilarity between data points [11]. Distance-based methods are often desirable because of their simplicity and ease of implementation in a wide variety of scenarios. Distance-based algorithms can be generally divided into two types:

   a. Flat: In this case, the data is divided into several clusters in one shot, typically with the use of partitioning representatives. The choice of the partitioning representative and distance function is crucial and regulates the behavior of the underlying algorithm. In each iteration, the data points are assigned to their closest partitioning representatives, and then the representative is adjusted according to the data points assigned to the cluster [11]. Some common methods for creating the partitions are as follows:

      (i) K-means. This is an iterative clustering algorithm that splits the dataset into k clusters. The purpose of it is to minimise the sum of the squares of the distances within a cluster. Euclidean distance is usually used as a distance metric to distribute data points across clusters and update cluster centroids. The k -Means method is considered one of the simplest and most classical methods for data clustering and is also perhaps one of the most widely used methods because of its simplicity [11].

  (ii) K-medians. It is similar to the well-known K-means algorithm, but uses a different measure of central tendency. K-medians use the median to represent the center of each cluster. The median is the average of a data point set, which makes it robust to outliers and less sensitive to extremes [11].

b. Hierarchical: In these methods, the clusters are represented hierarchically through a dendogram , at varying levels of granularity

  (i) Agglomerative: Involves creating a hierarchy of clusters by iteratively merging smaller clusters into larger ones. This technique requires the calculation of distances between pairs of data points, which can be done using various distance metrics [11] and [19].

  (ii) Divisive: Opposite to agglomerative clustering. Starts by considering all the data points as a single cluster. And then splits the cluster into smaller sub-clusters [11] and [19].

2. Density and Grid-Based Methods. Density-based methods are a popular approach for clustering data points, which can be beneficial in several applications. According to Brown's (2020) research on density-based methods [20], these techniques rely on the distribution of data points and their proximity to identify clusters. A major advantage of these methods is that since they explore the data space at a high level of granularity, they can be used to reconstruct the entire shape of the data distribution [11].

3. Leveraging Dimensionality Reduction Methods Dimensionality reduction methods are closely related to both feature selection and clustering, in that they attempt to use the closeness and correlations between dimensions to reduce the dimensionality of representation. Thus, dimensionality reduction methods can often be considered a vertical form of clustering, in which columns of the data are clustered with the use of either correlation or proximity analysis, as opposed to the rows [11].

The last two methods have the disadvantage that they are very complicated to describe the algorithm. Therefore, I will explore well-known algorithms like hierarchical clustering and K-means with the application of various distances.

4.1 Distances

Distance measures clustering, as a technique for grouping data points based on their similarity or proximity, is an effective way to analyze and understand large datasets. According to Sujan Dahal, there are several different distance measures used in clustering, including Euclidean, Squared Euclidean, Manhattan, Chebyshev and Mahalanobis distance (2015) [21].

1. Euclidean distance. The most common distance function. Represents the geometric distance in multidimensional space [18]:

$$D(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (y_{1_i} - y_{2_i})^2} \tag{4.1}$$

   In this formula (4.1), y_{1_i} and y_{2_i} represent the i-th coordinate values of x_1 and x_2 respectively. The Euclidean distance is calculated by taking the square root of the sum of the squared differences between corresponding coordinates.

2. City block distance (Manhattan distance). This distance is the average of the coordinate differences. In most cases, this distance measure leads to the same results as the regular Euclidean distance. However, for this measure, the influence of individual large differences (outliers) is reduced (since they are not squared). The formula (4.2) for calculating Manhattan distance between two points x_1 and x_2 in an n-dimensional space is calculated as the sum of the absolute differences between corresponding coordinates [18]:

$$D(x_1, x_2) = \sum_{i=1}^{n} |y_{1_i} - y_{2_i}| \tag{4.2}$$

3. Jaccard distance. The Jaccard distance is commonly used to measure the dissimilarity or distance between two sets. It is particularly useful when dealing with binary or categorical data, where the presence or absence of elements in the sets is of interest. The Jaccard distance between two sets A and B is calculated as the complement of the Jaccard similarity coefficient [18]. It is defined as (4.3):

$$D(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{4.3}$$

In Sujan Dahal master's thesis on this topic evaluates the impact of these different measures on clustering output. By applying the K-medoid method (K-medoids method is the modified form of the K-means method) with various distance measures to a spatial dataset in his research area, Dahal found that the result of clustering varies depending on which measure is used. This suggests that it is crucial to choose the appropriate measure for each specific task when using this technique [21].

4.2 K-means clustering

When data scientists want to cluster data points, they often refer to a popular algorithm called K-means clustering. This powerful tool allows you to divide

1) Clustering select K points as initial centroids.
2) repeat
3) Form K clusters by assigning each point to its closest centroid.
4) Recompute the centroid of each cluster.
5) until convergence criterion is met.

Table 4.1: K-means algorithm

n observations into k clusters based on similarity or distance metrics. But where did this technique come from? K-means clustering was first proposed by Stuart Lloyd in 1957 as a pulse-code modulation quantization (PCM) coding technique [22]. In the years since its creation, it has become widespread in various fields such as image processing, data mining and market segmentation.

But what makes K-means so effective? By iteratively assigning each observation to the cluster whose mean is closest, it is able to minimize inter-cluster variance and maximize inter-cluster partitioning [11]. Because of its ability to efficiently handle large datasets in a variety of domains and applications, K-means remains one of the most commonly used algorithms today.

K-means clustering is a powerful unsupervised learning technique that has been widely used in various fields, including education. In fact, a study by Sari et al (2017) demonstrates the use of K-means algorithm to classify students' activities using e-learning platforms [23]. By grouping similar activity patterns, this approach yielded clusters of students' behavior and allowed for an improvement in student's ability. This application shows how Kmeans can be instrumental in identifying relationships between data points and providing insights into complex systems. The beauty of K-means lies in its simplicity yet effectiveness. It works by iteratively assigning each data point to the nearest centroid and updating their positions until convergence [23].

Although it is not immune to limitations such as sensitivity to initial conditions or outliers, K-means remains one of the most popular clustering algorithms due to its flexibility and ease of use.

K-means clustering is a greedy algorithm that is guaranteed to converge to a local minimum but the minimization of its score function is known to be NP-Hard [11]. Typically, the convergence condition is relaxed and a weaker condition may be used. In practice, it follows the rule that the iterative procedure must be continued until 1% of the points change their cluster memberships. Detailed proof of the mathematical convergence of K-means can be found in [24].

The K-means algorithm [11]: 4.1.

The algorithm is simple enough, but there are factors that greatly affect how it works and the final result.

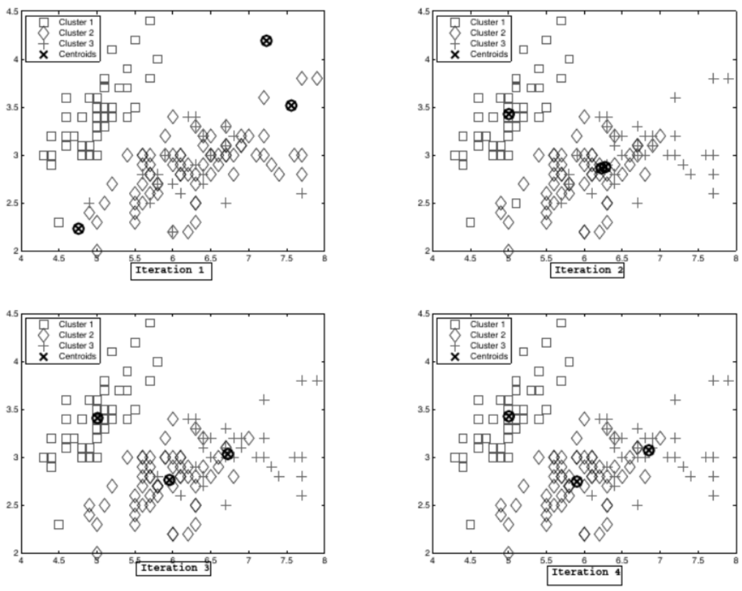


Figure 4.1: An illustration of 4 iterations of K-means over the Fisher Iris dataset[11].

4.2.1 Factors affecting K-means

The major factors that can impact the performance of the K -means algorithm are the following:

Choosing the initial centroids

Choosing K initial centroids in K-means is a critical step in the clustering process that can significantly impact the quality of results. There are several initialization methods:

1. Hartigan and Wong. First one entire cluster is allocated. Then it splits into two smaller clusters and will continue to split until it reaches a given number of clusters. The division will be considered completed if it leads to a deviation and variance reduction [11].

2. Bradley and Fayyad. This initialization works by selecting random subsamples from the data. And to each subsample it applies k-means clustering. The centroids obtained from these subsamples are then clustered again using k-means. The set of centroids giving the minimum sum of squares errors (SSE) is chosen as the initial set of seeds for k-means clustering [11].

3. K-means ++. The algorithm begins by randomly selecting the first centroid from the data points. Then, for each subsequent centroid, we calculate the distance between each data point and the closest already selected centroid. The probability of choosing a data point as the next centroid is higher if it is far away from the existing centroids [11]

```
ch_scores = []
k_range = range(2, 15)
for k in k_range:
    kmeans_model = KMeans(n_clusters=k, n_init='auto
       ', random_state=1).fit(updateDataSet)
    ch_score = calinski_harabasz_score(updateDataSet
       , kmeans_model.labels_)

    ch_scores.append(ch_score)

plt.plot(k_range, ch_scores)
plt.xlabel('Number of clusters K')
plt.ylabel('Calinski-Harabasz Score')
plt.title('Calinski-Harabasz Score')
plt.show()
best_k = np.argmax(ch_scores) + 2 # Add 2 because
   k_range starts from 2
print("Best value of K =", best_k)
```

Figure 4.2: Code for Calinski-Harbasz score

For my thesis, I used the initialization of K-means++ as it is built into the library *sklearn.cluster.kmeans_plusplus*. Which I will use for clustering.

Calculation the number of clusters K

In my research, I used one method to find the best value. Namely:

1. Calinski-Harbasz score [26]. The index is defined by Equation (4.4):

$$\mathrm{CH}(K) = \frac{B(K) \cdot (K-1)}{W(K) \cdot (N-K)} \tag{4.4}$$

To find the best value of K (figure 4.2) was written a code. As a result, you can see a graph that represents the dependence of the Calinski-Harbasz score on the number of clusters.

Where a higher value indicates better-defined and well-separated clusters. From the graph 4.3, you can see that the best number of clusters is 3.

4.2.2 Anomaly detection in K-means

This is the topic that has sparked the curiosity of many researchers, data scientists, and analysts. The potential to uncover hidden insights or identify outlying behaviors within datasets has made this area of study critical for various industries such as finance, healthcare, and cybersecurity [11] [17].

However, before delving into the intricacies of detecting anomalies with the K-means clustering algorithm, let me first explain what an anomaly is.

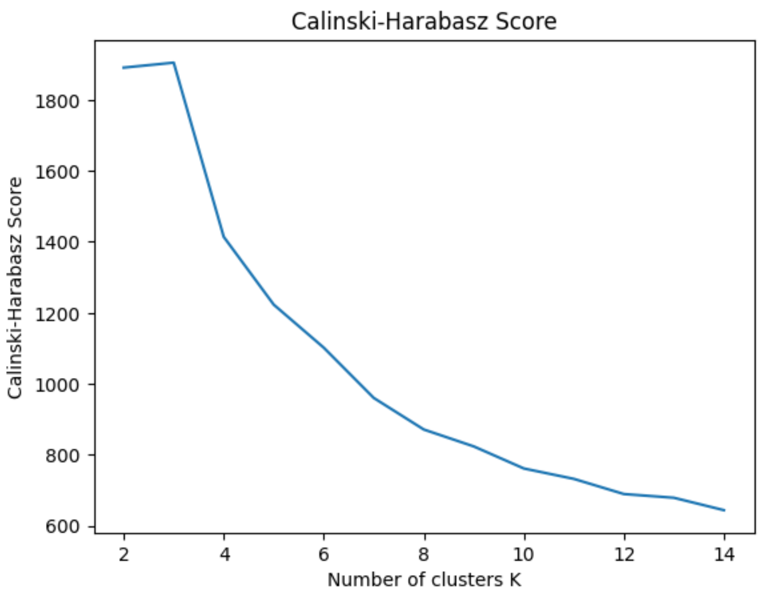


Figure 4.3: Calinski-Harbasz score

```python
d={}
j=0
for i in uniqueEventTitle:
    if j not in eventTitle_numb:
        if j!=40:
            d[j]=i
            j+=1
        else: j+=1
    else: j+=1
```

Figure 4.4: Categories oe eventTitles

In its simplest form, an anomaly can be defined as something that deviates from what is standard or expected. In the world of data analysis though, identifying these anomalies can prove to be challenging due to their sporadic nature and often ambiguous definition. This is where K-means clustering comes in by grouping similar observations together based on specific features [27].

One challenge in anomaly detection with K-means is determining the appropriate threshold for identifying outliers, as it may based on the specific data being analyzed and the goals of the analysis.

Now I have everything for finding anomalies. Filtered dataset without unnecessary data. A choice of initialized method and the required number of clusters.

To begin with, I created a dictionary made up of categories or *eventTitles* (code 4.4). Next, since we have three clusters. For our convenience, we will compare first and second 4.5, first and third, and finally second and third clusters.

For the other clusters, we will use the same code (4.5), but we will change the *cluster_1* and *cluster_2* variables respectively on *cluster_1* and *cluster_3* or *cluster_2* and the last one *cluster_3*:

As a result, we get three plotly graphs (see figure 4.6, appendix A.3 figure A.3, A.4). These plots show in which *eventTitles* the clusters differ by more than 20%.

And the last part is to find the distances from the center centroids to each *assetId* or outliers.

4.2.3 Outlier detection

To do this, I created a function (figure 4.7) that calculates two metrics. The Euclidean distance and the Manhattan distance. Let's see and compare what suspicious *assetIds* each metric find.

Consider the results of the Euclidean 4.8 distance and the Manhattan 4.10 distance about the first and second clusters.

The result is histograms (4.9 and 4.11) that represent how many outliers are near or far from the cluster center. For example, in histogram 4.9, you can see that the farthest outliers are more than at a distance of two point three. And in histogram 4.11, outliers are at a distance of seven. To see those *assetIds*, let's run the code (figure 4.12 and 4.13).

We end up with two dataframes (4.14 and 4.15) with the same result. This tells us that the metrics are quite similar to themselves. Let me remember the formulas by which they are calculated (Chapter 5.1). The formulas are very similar.

1. Euclidean distance (equation 4.1): $\mathrm{D}(\mathrm{x}_1, x_2) = \sqrt{\sum_{i=1}^{n}(y_{1_i} - y_{2_i})^2}$

2. City block (Manhattan) distance (equation 4.2): $\mathrm{D}(\mathrm{x}_1, x_2) = \sum_{i=1}^{n}|y_{1_i} - y_{2_i}|$

If we want to see how the outliers look compared to the cluster center, then we have to run the following code (figure 4.16).

All you have to do is set the number of clusters (*label*), the threshold and the categories. The code 4.17 searches for *assetIds* using the Euclidean metric. For the Manhattan metric, the code will be identical, except for specifying the metric and the other threshold.

And the result is shown in the figure (4.17). You can immediately see that all of the asset values are much larger than the center of the cluster.

However, if we choose a different threshold, the result will change. As seen in the diagram in the (appendix figures A.9, A.10, A.11, A.12). Assets are almost identical in each cluster using Euclidean and Manhattan metrics. The result depends only on the choice of threshold. As you can see from the (figures A.9, A.10), our first asset is *b8368210-4879-3d5b-8247-2746b956109e*. In the chapter on data analytics (3.3) I indicated that this asset has more then average *eventTitles*. This shows that my suggestion was correct.

4.3 Hierarchical Agglomerative clustering

In the past few years, Agglomerative clustering has gained widespread recognition for its ability to manage data that is not linearly separable. This common challenge can prove difficult for other clustering techniques. This method proves especially useful when handling complex datasets with intricate structures since traditional algorithms often fail in classifying them with ease. Aljumily (2016) [28] highlights Agglomerative clustering's functionality by creating a vector-distance matrix and constructing a hierarchical tree. The creation of this layout allows patterns and similarities in the data to be identified regardless of their linearity or separation boundaries. Agglomerative clustering works recursively by combining similar objects into clusters until all elements belong to one large group or individual clusters without any coherence between them (Aljumily, 2016) [28]. Therefore, this particular technique is efficient at grouping observations together based on shared attributes without being constrained by separation limits.

4.3.1 Anomaly detection in Agglomerative clustering

This clustering method is considered fast and easy thanks to the Python libraries. We calculate pairwise distances than set up a traverse and calculate the coupling matrix.

This time I decided to change the metrics and use Euclidean distance (figure 4.18) and Jaccard (figure 4.19) distance. The result of executing the code (4.20 and 4.21) is the number of assets in each cluster. And if there is one asset in the cluster, we can consider it as suspicious *assetId*. The result is two figures (4.20 and 4.21).

Based on the result that clustering with the Euclidean (4.20) metric gave us too many *assetIds*, this metric is not suitable here. Because we want to get about 0.1% of the *assestIds* as anomalous. Because it would be easier for the analyst to deal with a small amount of data in the future and not make a mistake without spending a lot of time.

4.4 Visualisation technique

This time I used a tool like plotly. Plotly is a Python data visualization library that provides many features and benefits that make it very user-friendly:

1. Interactivity: One of the main benefits of Plotly is the ability to create interactive graphs. We can scale graphs to move, hover over data, and more.

2. Wide range of charts: Plotly supports a wide range of chart types, including line, bar, pie, point, heat map, bar graphs. You can choose from a variety of styles and options to customize the appearance of the charts to your needs.

```python

categories = list(d.values())
# Perform K-Means clustering with k=3
kmeans = KMeans(n_clusters=3, init='k-means++', n_init='auto').fit(updateDataSet)
fig = go.Figure()

# Add the data for each cluster
cluster_1 = updateDataSet[kmeans.labels_ == 0].mean()
cluster_2 = updateDataSet[kmeans.labels_ == 1].mean()

difference_index = abs(cluster_1 - cluster_2) >=0.2


categories = [d[i] for i in list(cluster_1.index[difference_index])]

cluster_1 = list(cluster_1[difference_index])
cluster_2 = list(cluster_2[difference_index])

fig.add_trace(go.Scatterpolar(r=cluster_1, theta=categories, fill='toself', name=f'Cluster 1'))
fig.add_trace(go.Scatterpolar(r=cluster_2, theta=categories, fill='toself', name=f'Cluster 2'))

fig.update_layout(title=go.layout.Title(text='Clusters'),
                  polar={'radialaxis': {'visible': True}},
                  showlegend=True)

pyo.plot(fig)
```

Figure 4.5: A comparison between the 1 and the 2 clusters

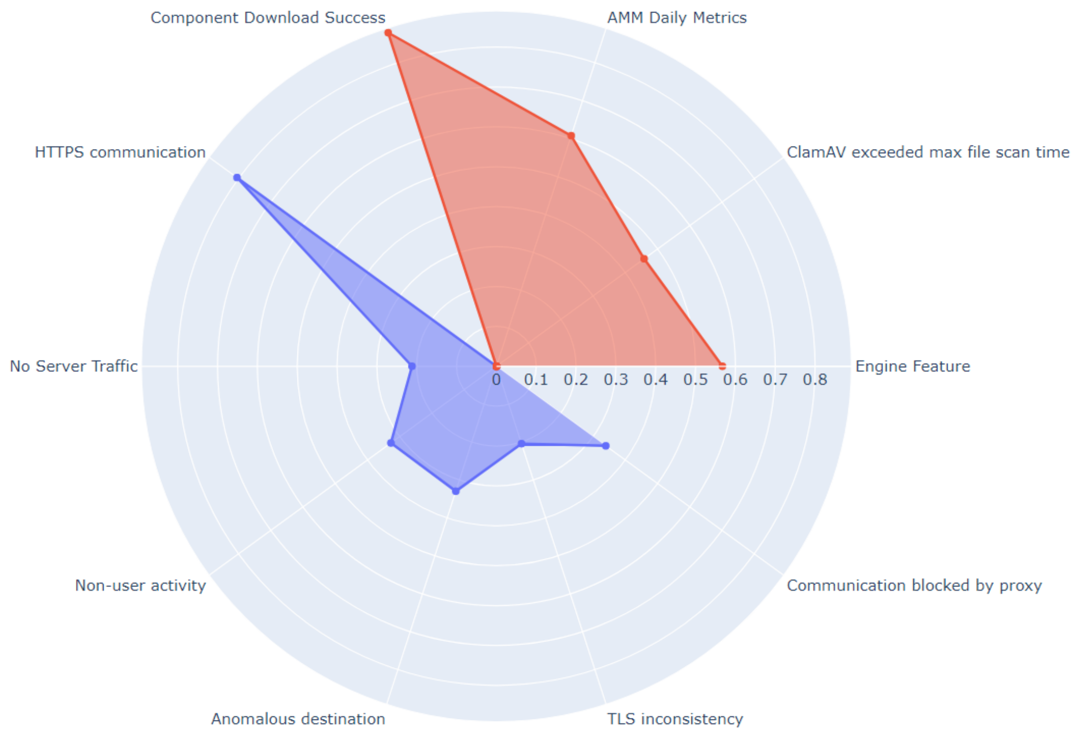


Figure 4.6: Polar Plot between the 1 and the 2 clusters

```
def distance_from_center(row):
    label = row['label']
    center_cluster =  kmeans.cluster_centers_[label]
    data_vector = row[:center_cluster.size].to_numpy
       ()

    distance_euclidean = distance.euclidean(
       data_vector, center_cluster)
    distance_cityblock = distance.cityblock(
       data_vector, center_cluster)
    return np.round(distance_euclidean, 2), np.round
       (distance_cityblock, 2)

updateDataSet['label'] = kmeans.labels_
#updateDataSet['distance'] = updateDataSet.apply(
   distance_from_center, axis=1)
updateDataSet['distance_euclidean'], updateDataSet['
   distance_cityblock'] = zip(*updateDataSet.apply(
   distance_from_center, axis=1))
```

Figure 4.7: Function that calculates distance measures

```
updateDataSet[updateDataSet.label == 0].
   distance_euclidean.plot(kind='hist', bins=20, log
   =False)
```

Figure 4.8: Code for 1 cluster

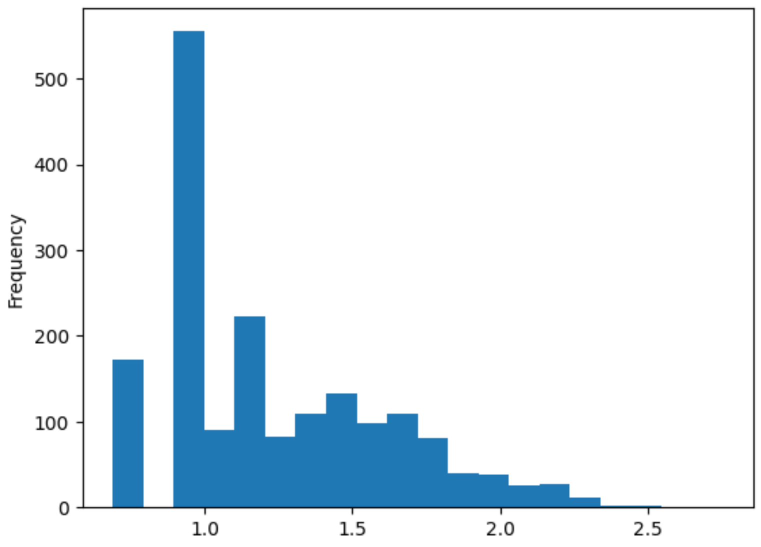


Figure 4.9: Histogram of Euclidean distance for the 1 cluster.

```
updateDataSet[updateDataSet.label == 0].
    distance_cityblock.plot(kind='hist', bins=20, log
    =False)
```

Figure 4.10: Code for 1 cluster

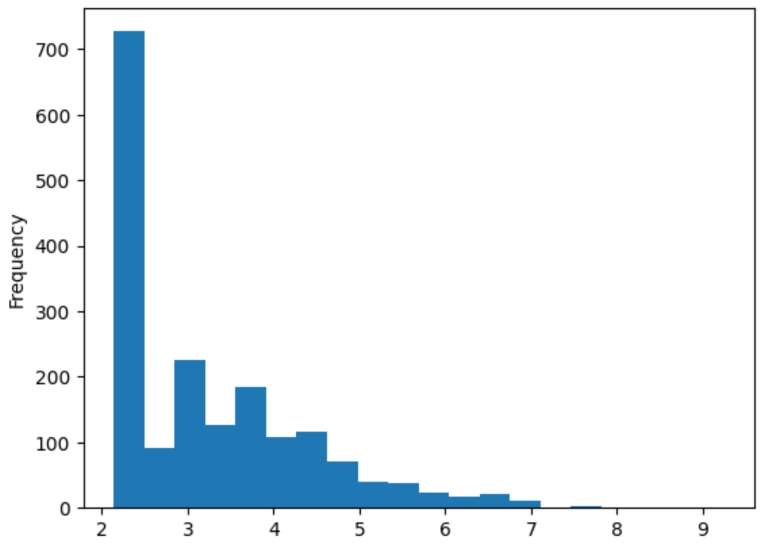


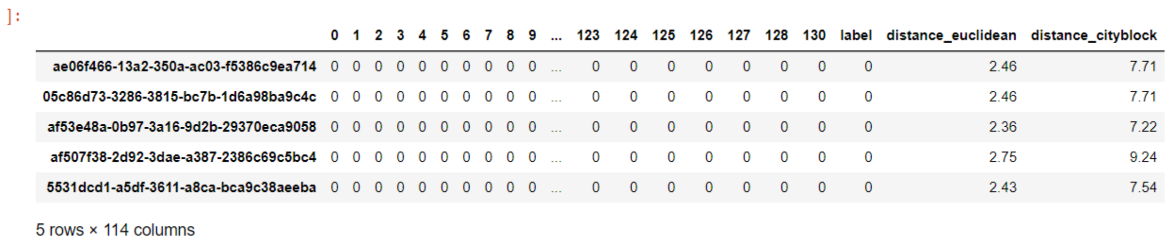
Figure 4.11: Histogram of Manhatten distance for the 1 cluster.

```
threshold = 2.3
updateDataSet[updateDataSet.label == 0][
    updateDataSet.distance_euclidean > threshold]
```

Figure 4.12: AssetIds that farther from the center than 2.3

```
threshold =7.0
updateDataSet[updateDataSet.label == 0][
    updateDataSet.distance_cityblock > threshold]
```

Figure 4.13: AssetIds that farther from the center than 7.0

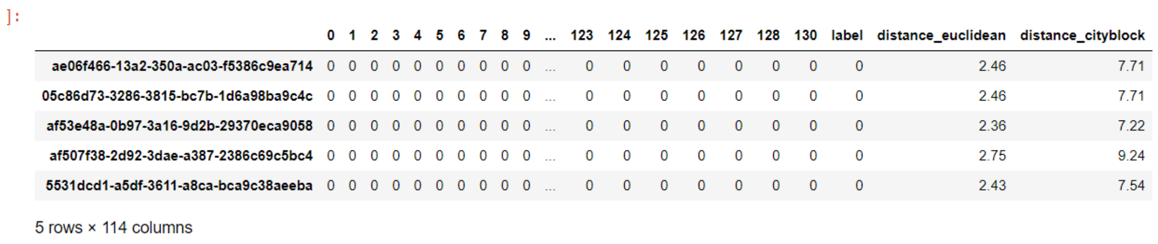


]:

	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
ae06f466-13a2-350a-ac03-f5386c9ea714	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.46	7.71
05c86d73-3286-3815-bc7b-1d6a98ba9c4c	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.46	7.71
af53e48a-0b97-3a16-9d2b-29370eca9058	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.36	7.22
af507f38-2d92-3dae-a387-2386c69c5bc4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.75	9.24
5531dcd1-a5df-3611-a8ca-bca9c38aeeba	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.43	7.54

5 rows × 114 columns

Figure 4.14: Dataframe with Euclidean distance for the 1 cluster.



]:

	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
ae06f466-13a2-350a-ac03-f5386c9ea714	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.46	7.71
05c86d73-3286-3815-bc7b-1d6a98ba9c4c	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.46	7.71
af53e48a-0b97-3a16-9d2b-29370eca9058	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.36	7.22
af507f38-2d92-3dae-a387-2386c69c5bc4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.75	9.24
5531dcd1-a5df-3611-a8ca-bca9c38aeeba	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.43	7.54

5 rows × 114 columns

Figure 4.15: Dataframe with Manhattan distance for the 1 cluster.

```

def cluster_comparison (data, label, distance_threshold, d):
    fig = go.Figure()

    # Add the data for each cluster
    asset = data[data.label == label][data.distance_euclidean > distance_threshold].iloc[0]
    label = int(asset['label'])
    vector_size = kmeans.cluster_centers_[label].size
    cluster_center = data[kmeans.labels_ == label].mean()[:vector_size]
    asset_vector = asset[:vector_size]

    difference_index = abs(cluster_center - asset_vector) >= 0.2   # calculates the absolute difference between the corresponding elements

    categories = [d[i] for i in list(cluster_center.index[difference_index])]

    cluster_center = list(cluster_center[difference_index])
    asset_vector = list(asset_vector[difference_index])

    fig.add_trace(go.Scatterpolar(r=cluster_center, theta=categories, fill='toself', name=f'Cluster center'))
    fig.add_trace(go.Scatterpolar(r=asset_vector, theta=categories, fill='toself', name=f'Asset'))

    fig.update_layout(title=go.layout.Title(text='Clusters'),
                      polar={'radialaxis': {'visible': True}},
                      showlegend=True)

    pyo.plot(fig)
label = 0
threshold = 2.3
cluster_comparison (updateDataSet, label, threshold, d)
```

Figure 4.16: Asset compared to the center of the cluster

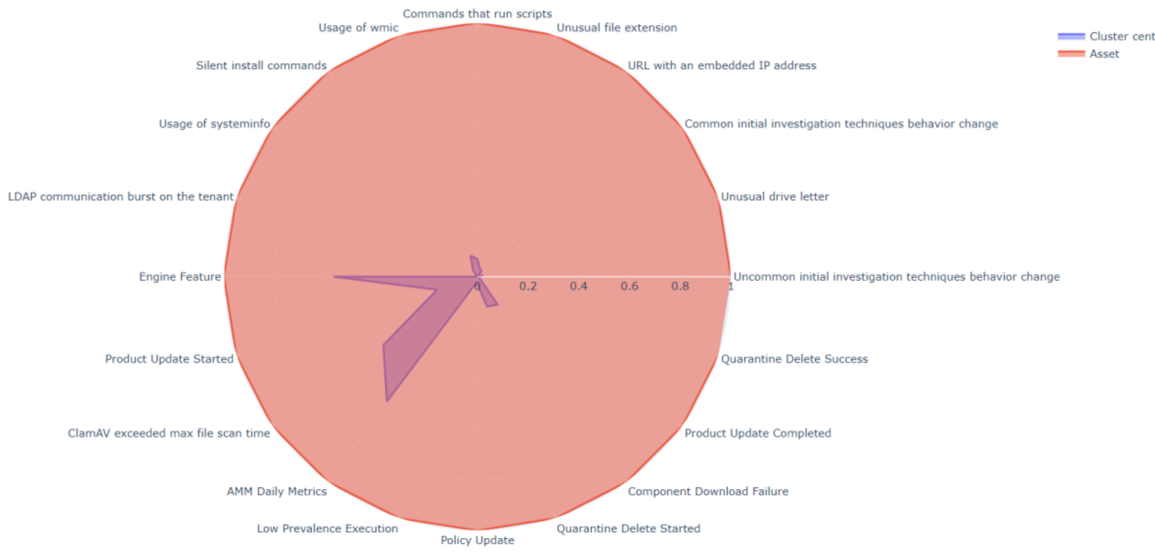


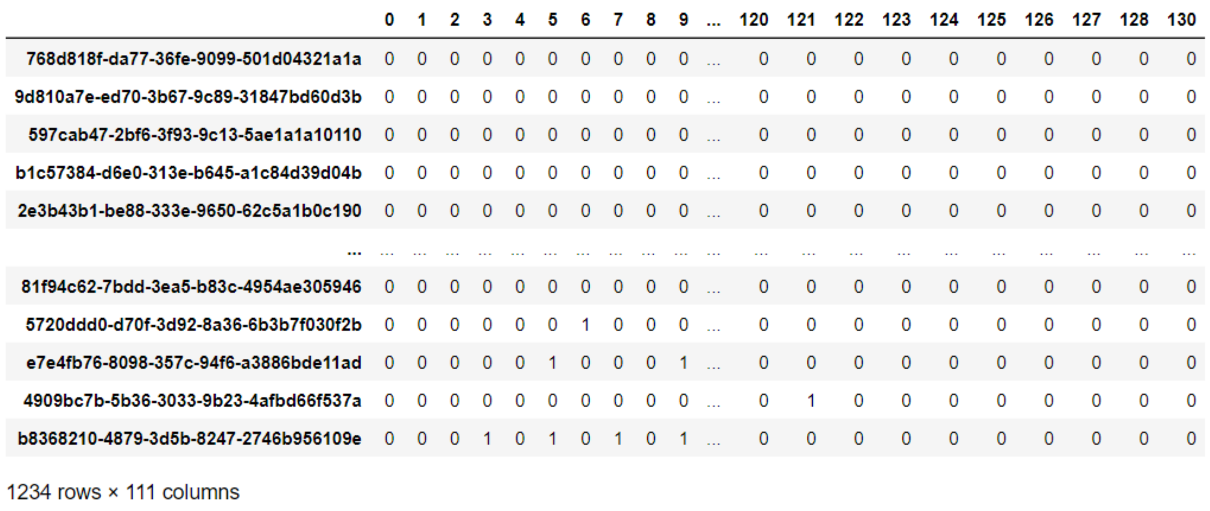
Figure 4.17: Asset compared to the center of the cluster

```python
# Compute the pairwise distances between rows using
    Euclidean distance
dists = pdist(updateDataSet, metric='euclidean')

# Perform hierarchical clustering using single
    linkage
linkage_matrix = linkage(dists, method='single')

# Extract the cluster labels for a given distance
    threshold
threshold = 0.6  # Adjust this value to obtain
    different number of clusters
clusters = fcluster(linkage_matrix, threshold,
    criterion='distance')

# Create a dictionary to store the cluster data
cluster_data = {}

# Compute the mean values for each cluster
for cluster_label in range(1, max(clusters) + 1):
    cluster_data[cluster_label] = updateDataSet[
        clusters == cluster_label].mean()
unique_values ,counts = np.unique(clusters,
    return_counts=True)
single_asset_clusters = []
for cluster_label, count in zip(unique_values,
    counts):
    if count == 1:
        cluster_rows = updateDataSet[clusters ==
            cluster_label]
        single_asset_clusters.append(cluster_rows)

df_single_asset_clusters = pd.concat(
    single_asset_clusters)
df_single_asset_clusters
```

Figure 4.18: The use of Euclidean metric in Hierarchical Clustering

```python
# Compute the pairwise distances between rows using Jaccard distance
dists = pdist(updateDataSet, metric='jaccard')

# Perform hierarchical clustering using single linkage
linkage_matrix = linkage(dists, method='single')

# Extract the cluster labels for a given distance threshold
threshold = 0.6  # Adjust this value to obtain different number of clusters
clusters = fcluster(linkage_matrix, threshold, criterion='distance')

# Create a dictionary to store the cluster data
cluster_data = {}

# Compute the mean values for each cluster
for cluster_label in range(1, max(clusters) + 1):
    cluster_data[cluster_label] = updateDataSet[clusters == cluster_label].mean()
unique_values ,counts = np.unique(clusters, return_counts=True)
single_asset_clusters = []
for cluster_label, count in zip(unique_values, counts):
    if count == 1:
        cluster_rows = updateDataSet[clusters == cluster_label]
        single_asset_clusters.append(cluster_rows)

df_single_asset_clusters = pd.concat(single_asset_clusters)
df_single_asset_clusters
```

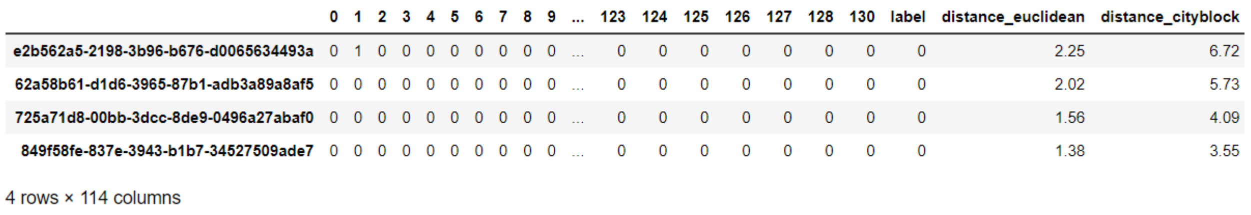
Figure 4.19: The use of Jaccard metric in Hierarchical Clustering



	0	1	2	3	4	5	6	7	8	9	...	120	121	122	123	124	125	126	127	128	130
768d818f-da77-36fe-9099-501d04321a1a	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9d810a7e-ed70-3b67-9c89-31847bd60d3b	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
597cab47-2bf6-3f93-9c13-5ae1a1a10110	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
b1c57384-d6e0-313e-b645-a1c84d39d04b	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2e3b43b1-be88-333e-9650-62c5a1b0c190	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
81f94c62-7bdd-3ea5-b83c-4954ae305946	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5720ddd0-d70f-3d92-8a36-6b3b7f030f2b	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
e7e4fb76-8098-357c-94f6-a3886bde11ad	0	0	0	0	0	1	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
4909bc7b-5b36-3033-9b23-4afbd66f537a	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	0
b8368210-4879-3d5b-8247-2746b956109e	0	0	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0

1234 rows × 111 columns

Figure 4.20: Agglomerative clustering with Euclidean distance



	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
e2b562a5-2198-3b96-b676-d0065634493a	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.25	6.72
62a58b61-d1d6-3965-87b1-adb3a89a8af5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	2.02	5.73
725a71d8-00bb-3dcc-8de9-0496a27abaf0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1.56	4.09
849f58fe-837e-3943-b1b7-34527509ade7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1.38	3.55

4 rows × 114 columns

Figure 4.21: Agglomerative clustering with Jaccard distance

Chapter 5
Conclusion

As a result, I would like to compare the results obtained with K-means and Hierarchical clustering.

It is immediately obvious that the second method found the assets that K-means did not find. Because from the (figure 4.21) these assets are located close to the center of the cluster. This happened because of applying different metrics, namely Euclidean and Manhattan metrics, which are similar to each other, and Jaccard metric.

For the future it will be possible to see what *assetsIds* K-means method will find using Jaccard distance. In the future development of this bachelor thesis, it may be possible to write a neural network that will search for these anomalies without human assistance. Because now a human sets the right metric, cluster and threshold. At a later time, all this could be automated.

Bibliography

[1] Johnston, Benjamin, et al. Applied Unsupervised Learning with Python : Discover Hidden Patterns and Relationships in Unstructured Data with Python, Packt Publishing, Limited, 2019. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=5781350.

[2] Rabinovich, Ella and Wintner, Shuly, 2015, page 1, "Unsupervised Identification of Translationese". https://aclanthology.org/Q15-1030.pdf

[3] Parisi, Alessandro. Hands-On Artificial Intelligence for Cybersecurity : Implement Smart AI Systems for Preventing Cyber Attacks and Detecting Threats and Network Anomalies, Packt Publishing, Limited, 2019. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=5847212. page 13

[4] The Applications of Sentiment Analysis for Russian Language Texts: Current Challenges and Future Perspectives Sergey Smetanin Journal: IEEE Access, 2020, Volume 8, Page 110693 DOI:10.1109/ACCESS.2020.3002215 available in https://link.springer.com/chapter/10.1007/978-3-030-34872-4_35

[5] IBM. (n.d.). Unsupervised Learning. Retrieved from https://www.ibm.com/topics/unsupervised-learning

[6] IBM. (n.d.). Supervised Learning. Retrieved from https://www.ibm.com/topics/supervised-learning

[7] Stack Overflow. (2018). Stack Overflow Developer Survey 2018. Retrieved from https://insights.stackoverflow.com/survey/2018

[8] Devopedia. (n.d.). Machine Learning in Python. Retrieved from https://devopedia.org/machine-learning-in-pythonStackOverflow-2018

[9] Medium. (n.d.). Feature Selection. Retrieved from https://medium.com/analytics-vidhya/feature-selection-85539d6a2a88

[10] Y. BBI, H. Yli, and S. SMA, 2022, "Feature Selection: A Review and Comparative Study", EDP Sciences. https://www.e3s-conferences.org/articles/e3sconf/pdf/2022/18/e3sconf_icies2022_01046.pdf

[11] Data Clustering : Algorithms and Applications, edited by Charu C. Aggarwal, and Chandan K. Reddy, CRC Press LLC, 2013. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=1355921.

[12] Bolón-Canedo, Verónica, et al. Feature Selection for High-Dimensional Data, Springer International Publishing AG, 2015. ProQuest Ebook Central, https://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=4085623.

[13] Analytics Vidhya. (2020, October). Feature Selection Techniques in Machine Learning. Retrieved from https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/

[14] Medium. (n.d.). Feature Selection. Retrieved from https://medium.com/analytics-vidhya/feature-selection-embedded-methods-a7940036973f

[15] Zhang, XD. (2020). Machine Learning. In: A Matrix Algebra Approach to Artificial Intelligence. Springer, Singapore. https://doi.org/10.1007/978-981-15-2770-8_6

[16] Mark A. Hall, 1999, Correlation-based Feature Selection for Machine Learning. https://www.cs.waikato.ac.nz/ mhall/thesis.pdf

[17] Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide O. Oyelade, Laith Abualigah, Jeffery O. Agushaka, Christopher I. Eke, Andronicus A. Akinyelu, Engineering Applications of Artificial Intelligence, Volume 110,2022, 104743, ISSN 0952-1976, https://doi.org/10.1016/j.engappai.2022.104743. (https://www.sciencedirect.com/science/article/pii/S095219762200046X)

[18] "Measures of Distance in Data Mining." GeeksforGeeks, URL: https://www.geeksforgeeks.org/measures-of-distance-in-data-mining/ (accessed May 22, 2023).

[19] Nielsen, F. (2016). Hierarchical Clustering. In: Introduction to HPC with MPI for Data Science. Undergraduate Topics in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-319-21903-5_8

[20] Brown, Daniel, 2020, "Fast Clustering Using a Grid-Based Underlying Density Function Approximation". https://core.ac.uk/download/pdf/322808625.pdf

[21] Dahal, Sujan, 2015, "Effect of Different Distance Measures in Result of Cluster Analysis", Aalto University School of Engineering, Department of Real Estate, Planning and Geoinformatics. https://core.ac.uk/download/pdf/80716648.pdf

[22] Lloyd, S. Least squares quantization in pcm. Information Theory. IEEE Transactions.

[23] Sari, Herlina Latipa et al, 2017, "IMPLEMENTATION OF K-MEANS CLUSTERING METHOD FOR ELECTRONIC LEARNING MODEL", Journal of Physics: Conference Series. https://iopscience.iop.org/article/10.1088/1742-6596/930/1/012021/pdf

[24] M. de Hoon, S. Imoto, J. Nolan, and S. Miyano. "Software for Clustering Gene Expression Data." University of Tokyo, URL: http://bonsai.hgc.jp/ mdehoon/software/cluster/software.htm

[25] P. Berkhin. A survey of clustering data mining techniques. Grouping Multidimensional Data , pages 25– 71, 2006.

[26] B. Mirkin. Clustering for Data Mining: A Data Recovery Approach . Chapman and Hall/CRC, Boca raton, FL, 2005.

[27] Ke-Wei Wang, 2016, "A hybrid approach for anomaly detection using K-means and PSO", Atlantis Press. https://www.atlantis-press.com/article/25861007.pdf

[28] Aljumily, Refat, 2016, "Agglomerative Hierarchical Clustering: An Introduction to Essentials. (1) Proximity Coefficients and Creation of a Vector-Distance Matrix and (2) Construction of the Hierarchical Tree and a Selection of Methods", Global Journals Inc. (USA). https://asels.org/wp-content/themes/asels/uploads/20118_5f9ec67805003.pdf

Appendix A

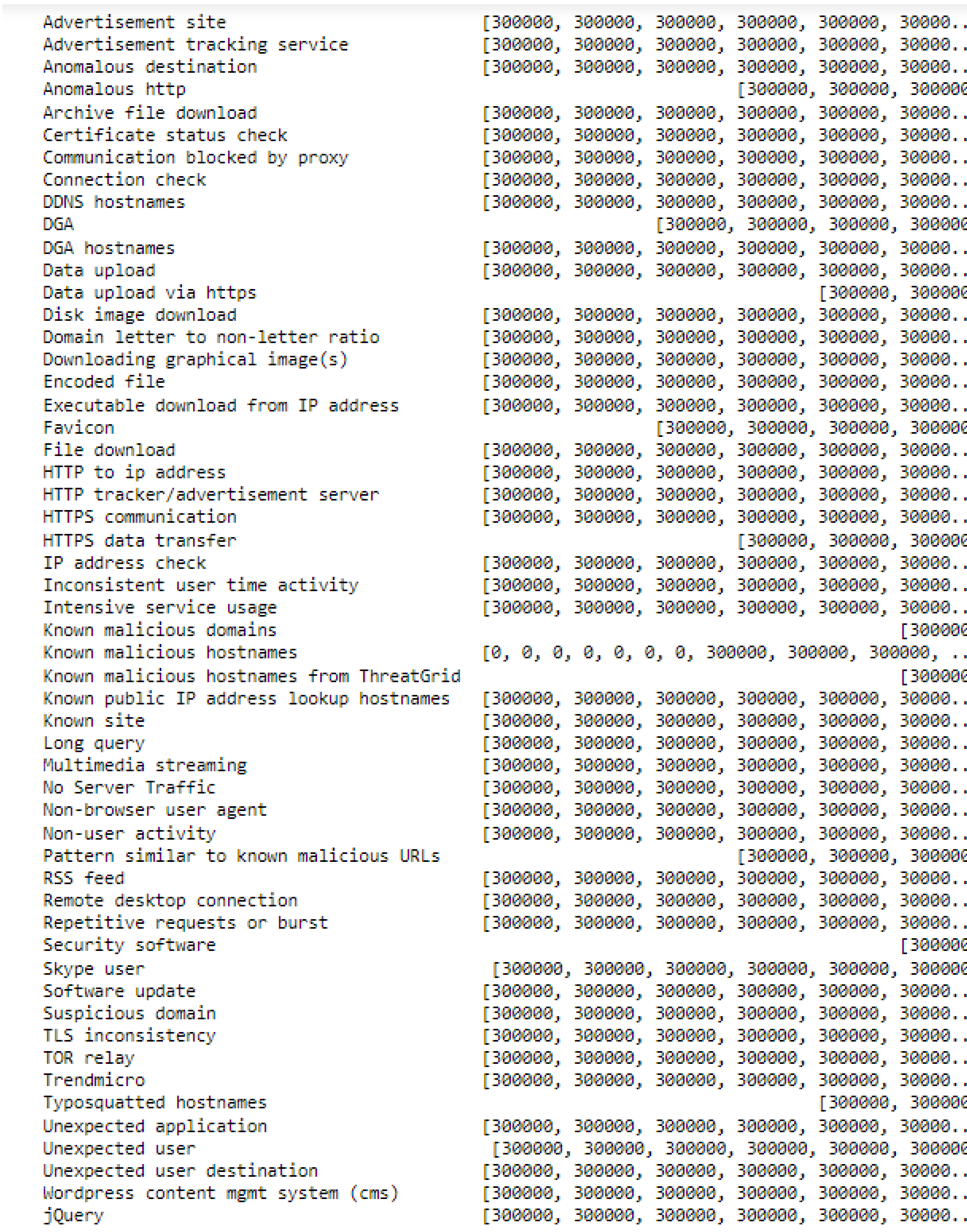
Appendix

A.1 Time table

A.2 Correlation

A.3 Plotly graphs

A.4 Outliers



```
Advertisement site                          [300000, 300000, 300000, 300000, 300000, 30000..
Advertisement tracking service              [300000, 300000, 300000, 300000, 300000, 30000..
Anomalous destination                       [300000, 300000, 300000, 300000, 300000, 30000..
Anomalous http                                                          [300000, 300000, 30000
Archive file download                       [300000, 300000, 300000, 300000, 300000, 30000..
Certificate status check                    [300000, 300000, 300000, 300000, 300000, 30000..
Communication blocked by proxy              [300000, 300000, 300000, 300000, 300000, 30000..
Connection check                            [300000, 300000, 300000, 300000, 300000, 30000..
DDNS hostnames                              [300000, 300000, 300000, 300000, 300000, 30000..
DGA                                                         [300000, 300000, 300000, 30000
DGA hostnames                               [300000, 300000, 300000, 300000, 300000, 30000..
Data upload                                 [300000, 300000, 300000, 300000, 300000, 30000..
Data upload via https                                                           [300000, 30000
Disk image download                         [300000, 300000, 300000, 300000, 300000, 30000..
Domain letter to non-letter ratio           [300000, 300000, 300000, 300000, 300000, 30000..
Downloading graphical image(s)              [300000, 300000, 300000, 300000, 300000, 30000..
Encoded file                                [300000, 300000, 300000, 300000, 300000, 30000..
Executable download from IP address         [300000, 300000, 300000, 300000, 300000, 30000..
Favicon                                                     [300000, 300000, 300000, 30000
File download                               [300000, 300000, 300000, 300000, 300000, 30000..
HTTP to ip address                          [300000, 300000, 300000, 300000, 300000, 30000..
HTTP tracker/advertisement server           [300000, 300000, 300000, 300000, 300000, 30000..
HTTPS communication                         [300000, 300000, 300000, 300000, 300000, 30000..
HTTPS data transfer                                                     [300000, 300000, 30000
IP address check                            [300000, 300000, 300000, 300000, 300000, 30000..
Inconsistent user time activity             [300000, 300000, 300000, 300000, 300000, 30000..
Intensive service usage                     [300000, 300000, 300000, 300000, 300000, 30000..
Known malicious domains                                                                 [30000
Known malicious hostnames                   [0, 0, 0, 0, 0, 0, 0, 300000, 300000, 300000, ..
Known malicious hostnames from ThreatGrid                                               [30000
Known public IP address lookup hostnames    [300000, 300000, 300000, 300000, 300000, 30000..
Known site                                  [300000, 300000, 300000, 300000, 300000, 30000..
Long query                                  [300000, 300000, 300000, 300000, 300000, 30000..
Multimedia streaming                        [300000, 300000, 300000, 300000, 300000, 30000..
No Server Traffic                           [300000, 300000, 300000, 300000, 300000, 30000..
Non-browser user agent                      [300000, 300000, 300000, 300000, 300000, 30000..
Non-user activity                           [300000, 300000, 300000, 300000, 300000, 30000..
Pattern similar to known malicious URLs                                 [300000, 300000, 30000
RSS feed                                    [300000, 300000, 300000, 300000, 300000, 30000..
Remote desktop connection                   [300000, 300000, 300000, 300000, 300000, 30000..
Repetitive requests or burst                [300000, 300000, 300000, 300000, 300000, 30000..
Security software                                                                       [30000
Skype user                                   [300000, 300000, 300000, 300000, 300000, 30000
Software update                             [300000, 300000, 300000, 300000, 300000, 30000..
Suspicious domain                           [300000, 300000, 300000, 300000, 300000, 30000..
TLS inconsistency                           [300000, 300000, 300000, 300000, 300000, 30000..
TOR relay                                   [300000, 300000, 300000, 300000, 300000, 30000..
Trendmicro                                  [300000, 300000, 300000, 300000, 300000, 30000..
Typosquatted hostnames                                                          [300000, 30000
Unexpected application                      [300000, 300000, 300000, 300000, 300000, 30000..
Unexpected user                              [300000, 300000, 300000, 300000, 300000, 30000
Unexpected user destination                 [300000, 300000, 300000, 300000, 300000, 30000..
Wordpress content mgmt system (cms)         [300000, 300000, 300000, 300000, 300000, 30000..
jQuery                                      [300000, 300000, 300000, 300000, 300000, 30000..
```

Figure A.1: Events that last longer than 0 ms

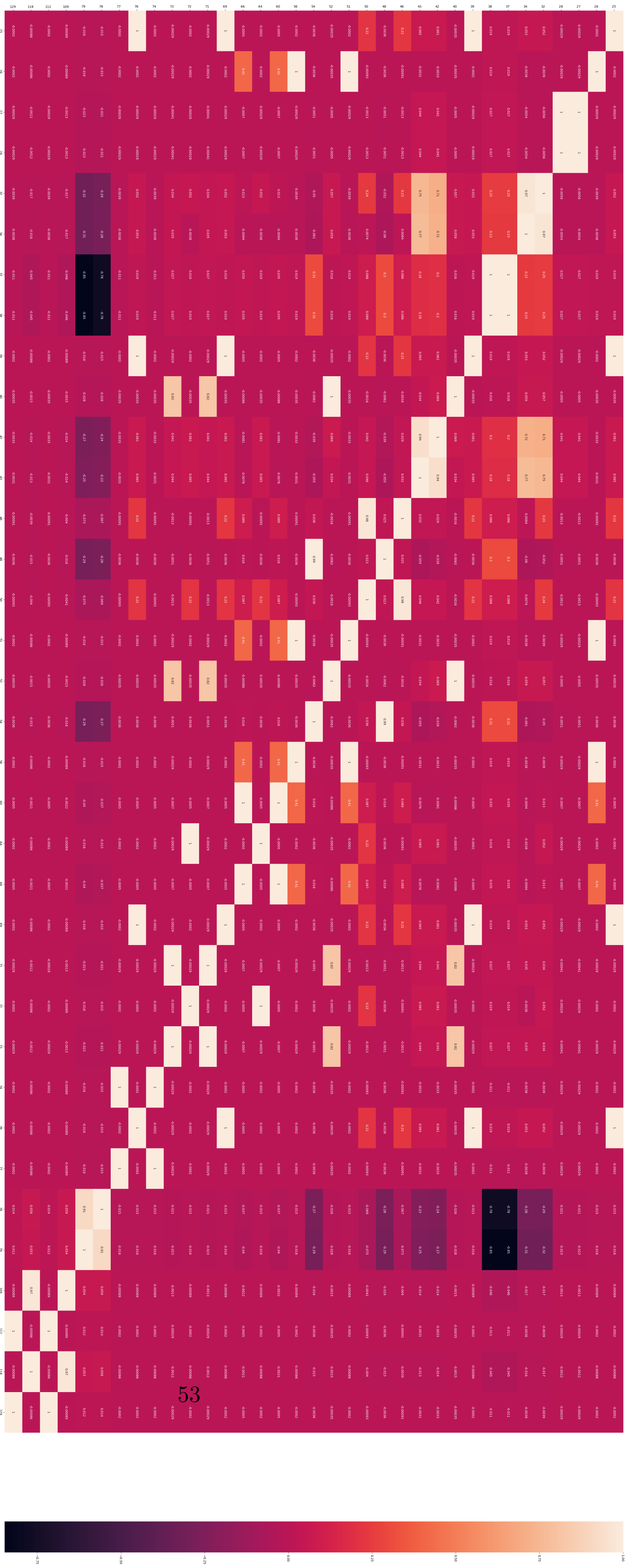


Figure A.3: Correlation

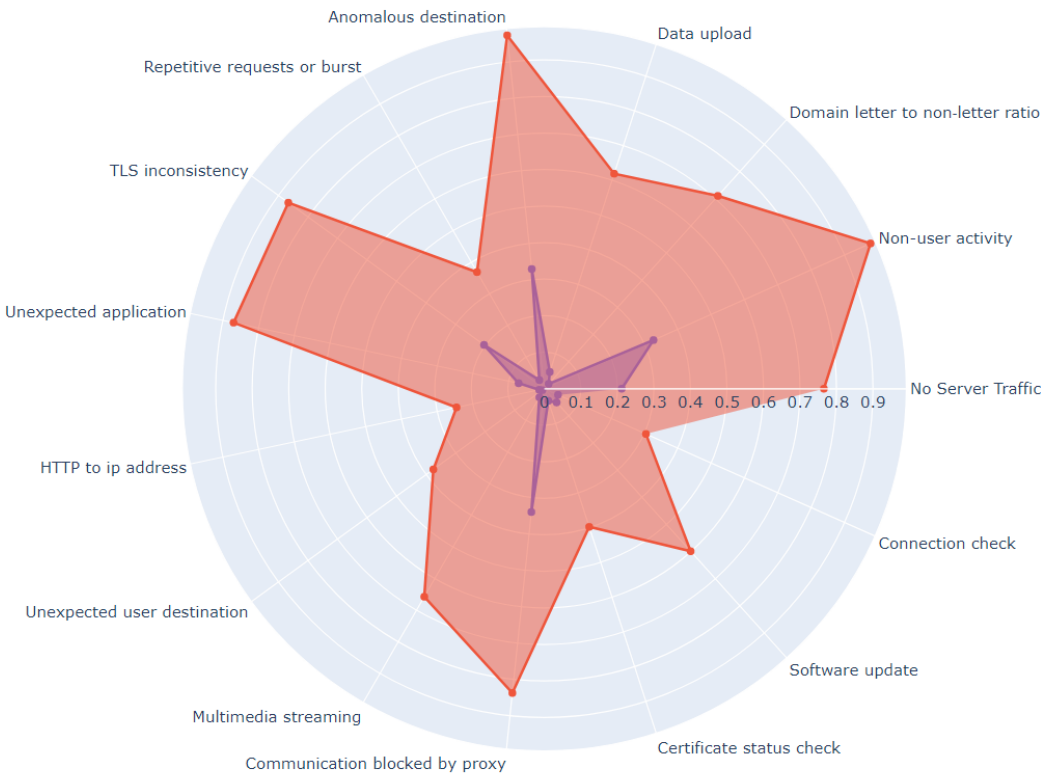


Figure A.3: Polar Plot between the 1 and the 3 clusters

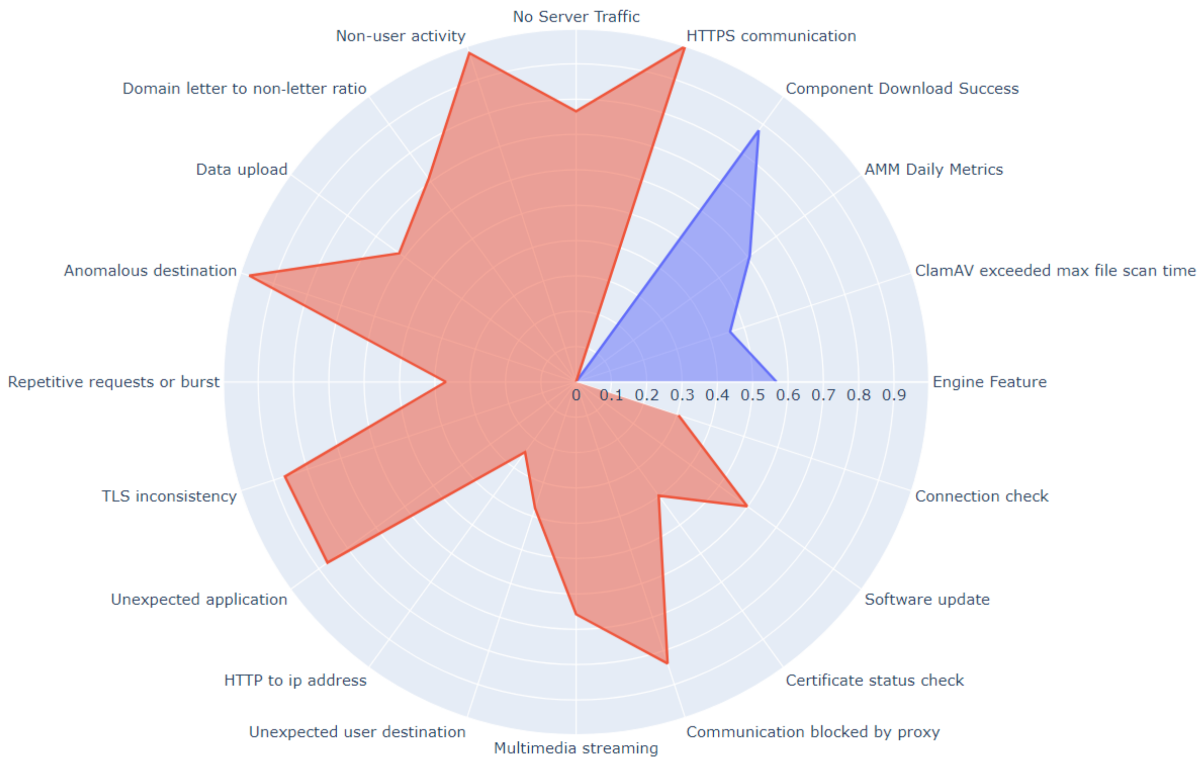


Figure A.4: Polar Plot between the 2 and the 3 clusters

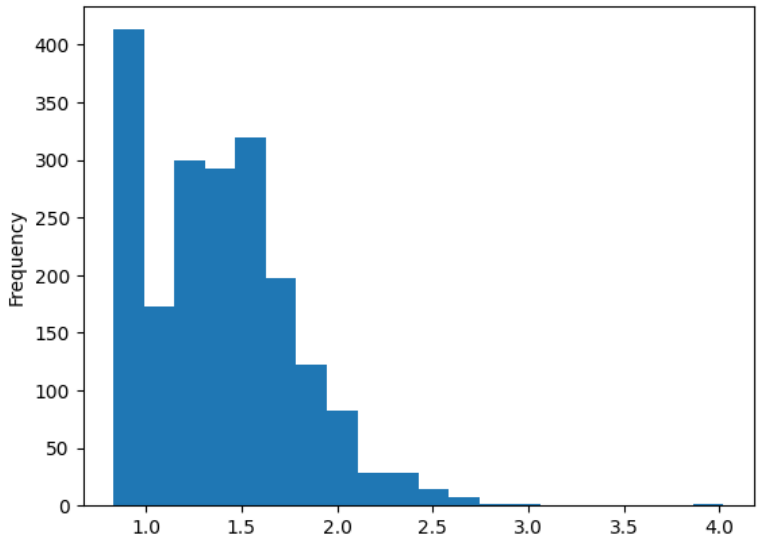


Figure A.5: Polar Plot

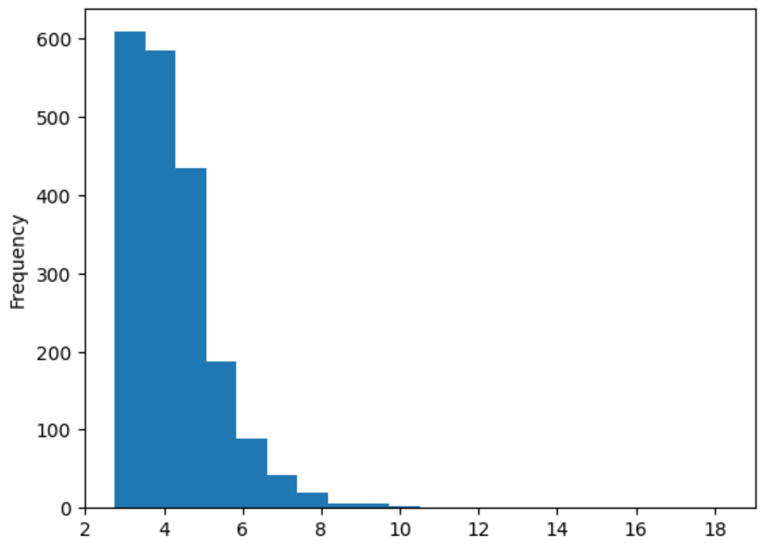


Figure A.6: Polar Plot

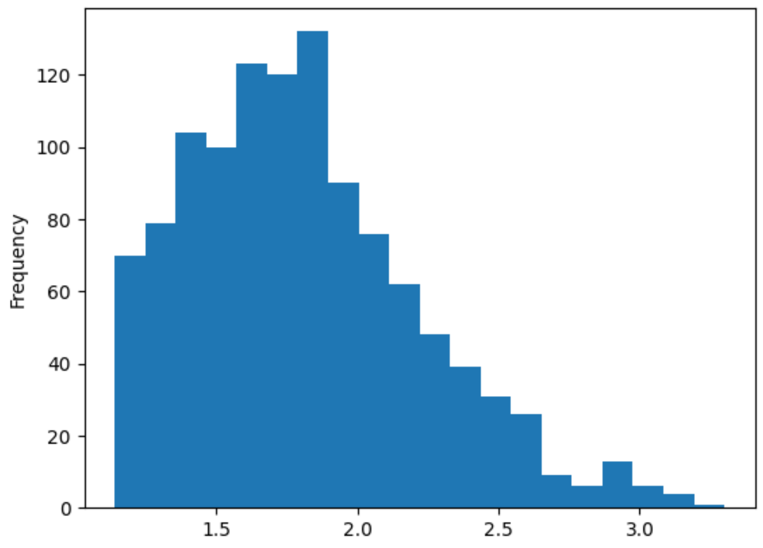


Figure A.7: Polar Plot

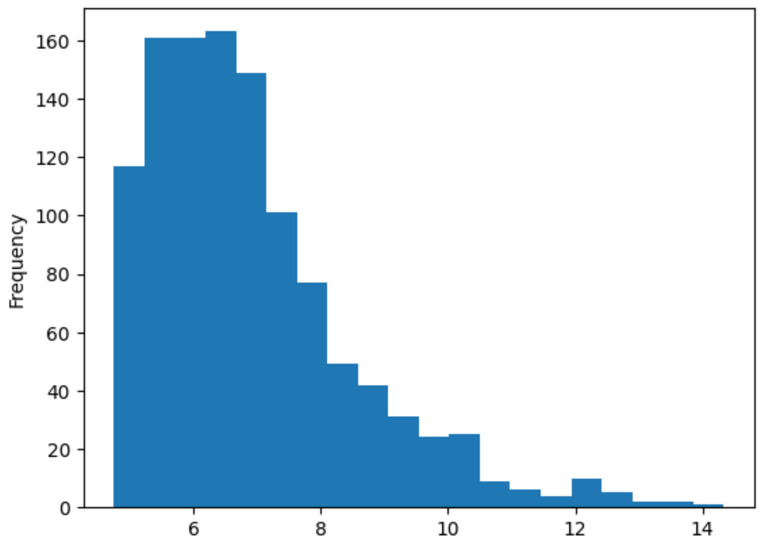
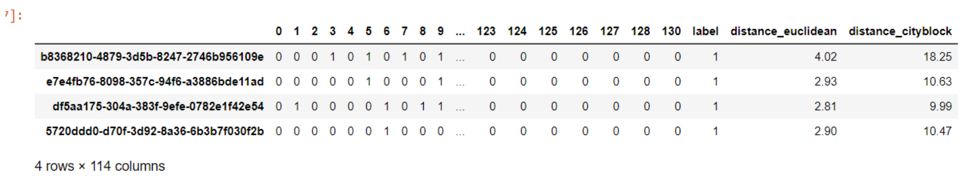


Figure A.8: Polar Plot

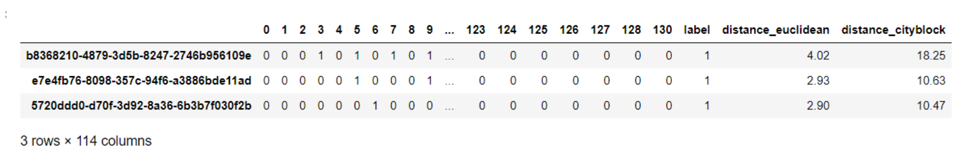


]:

	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
b8368210-4879-3d5b-8247-2746b956109e	0	0	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	1	4.02	18.25
e7e4fb76-8098-357c-94f6-a3886bde11ad	0	0	0	0	0	1	0	0	0	1	...	0	0	0	0	0	0	0	1	2.93	10.63
df5aa175-304a-383f-9efe-0782e1f42e54	0	1	0	0	0	0	1	0	1	1	...	0	0	0	0	0	0	0	1	2.81	9.99
5720ddd0-d70f-3d92-8a36-6b3b7f030f2b	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	1	2.90	10.47

4 rows × 114 columns

Figure A.9: Dataframe with Euclidean distance > 2.2 for the 2 cluster.

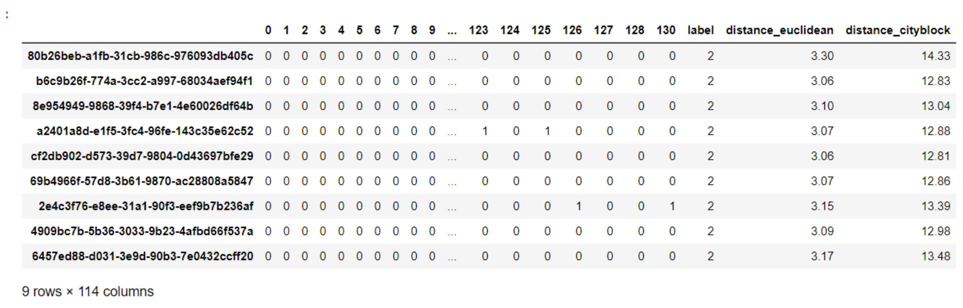


:

	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
b8368210-4879-3d5b-8247-2746b956109e	0	0	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	1	4.02	18.25
e7e4fb76-8098-357c-94f6-a3886bde11ad	0	0	0	0	0	1	0	0	0	1	...	0	0	0	0	0	0	0	1	2.93	10.63
5720ddd0-d70f-3d92-8a36-6b3b7f030f2b	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	1	2.90	10.47

3 rows × 114 columns

Figure A.10: Dataframe with Manhattan distance > 10.0 for the 2 cluster.

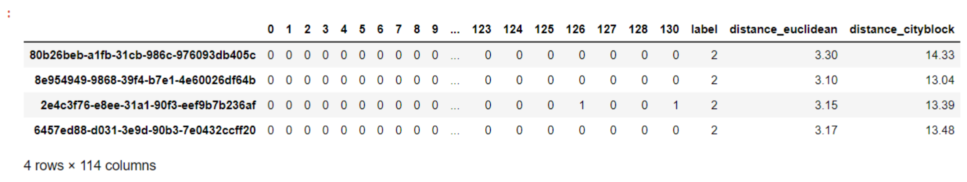


:

	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
80b26beb-a1fb-31cb-986c-976093db405c	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.30	14.33
b6c9b26f-774a-3cc2-a997-68034aef94f1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.06	12.83
8e954949-9868-39f4-b7e1-4e60026df64b	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.10	13.04
a2401a8d-e1f5-3fc4-96fe-143c35e62c52	0	0	0	0	0	0	0	0	0	0	...	1	0	1	0	0	0	0	2	3.07	12.88
cf2db902-d573-39d7-9804-0d43697bfe29	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.06	12.81
69b4966f-57d8-3b61-9870-ac28808a5847	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.07	12.86
2e4c3f76-e8ee-31a1-90f3-eef9b7b236af	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	1	2	3.15	13.39
4909bc7b-5b36-3033-9b23-4afbd66f537a	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.09	12.98
6457ed88-d031-3e9d-90b3-7e0432ccff20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.17	13.48

9 rows × 114 columns

Figure A.11: Dataframe with Euclidean distance >3.0 for the 3 cluster.



	0	1	2	3	4	5	6	7	8	9	...	123	124	125	126	127	128	130	label	distance_euclidean	distance_cityblock
80b26beb-a1fb-31cb-986c-976093db405c	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.30	14.33
8e954949-9868-39f4-b7e1-4e60026df64b	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.10	13.04
2e4c3f76-e8ee-31a1-90f3-eef9b7b236af	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	1	2	3.15	13.39
6457ed88-d031-3e9d-90b3-7e0432ccff20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	2	3.17	13.48

4 rows × 114 columns

Figure A.12: Dataframe with Manhattan distance > 13.0 for the 3 cluster.