Configuration Manual

MSc Research Project
Programme Name

Snehal Sarjerao Bhosale
Student ID: x19213948

School of Computing
National College of Ireland

Supervisor: Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Snehal Sarjerao Bhosale		
Student ID:	X19213948		
Programme:	MSc. in Cybersecurity	**Year:**	2021-2022
Module:	Research Project		
Lecturer:	Imran Khan		
Submission Due Date:	26/04/2022		
Project Title:	Cryptojacking Detection Using CPU Utilization as a target attribute with machine learning techniques.		
Word Count:	**1300 Page Count: 15**		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Snehal Sarjerao Bhosale

Date: 26/04/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Cryptojacking detection using CPU Utilization as a target attribute with machine learning techniques.

Snehal Sarjerao Bhosale
Student ID: x19213948

1. Introduction

The criteria for deploying the method that was developed to identify bitcoin crypto-hijacking through monitoring the CPU usage employing deep learning algorithms are outlined in the accompanying configuration manual. In addition, the document will deal into the software and hardware prerequisites which were required to execute the assignment efficiently.

2. System Configuration

The hardware and software specifications necessary to complete this research are mentioned below:

2.1. Hardware Requirements:

- Operating system: Windows 10
- System Compatibility: 64-bit
- CPU: 10th Gen Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
- RAM: 8.0 GB
- Storage: 456 GB

2.2. Software Requirements:

- Programming Language tools: Google Colab Community (Cloud-based Jupyter Notebook Environment) (Version: 2021.2(64 bits)), Python (Version: 3.8(64 bits))
- Email Account: Gmail for Google Drive
- We Browser: Google Chrome
- Other software: Microsoft Word

3. Project Development

This category describes how to build up the infrastructure and how to obtain information.

3.1. Google Colab Community Environment setup

Colab is a web-based Python editor that empowers anyone to develop and run unconstrained Python code. It's notably useful for deep learning, data processing, and education.

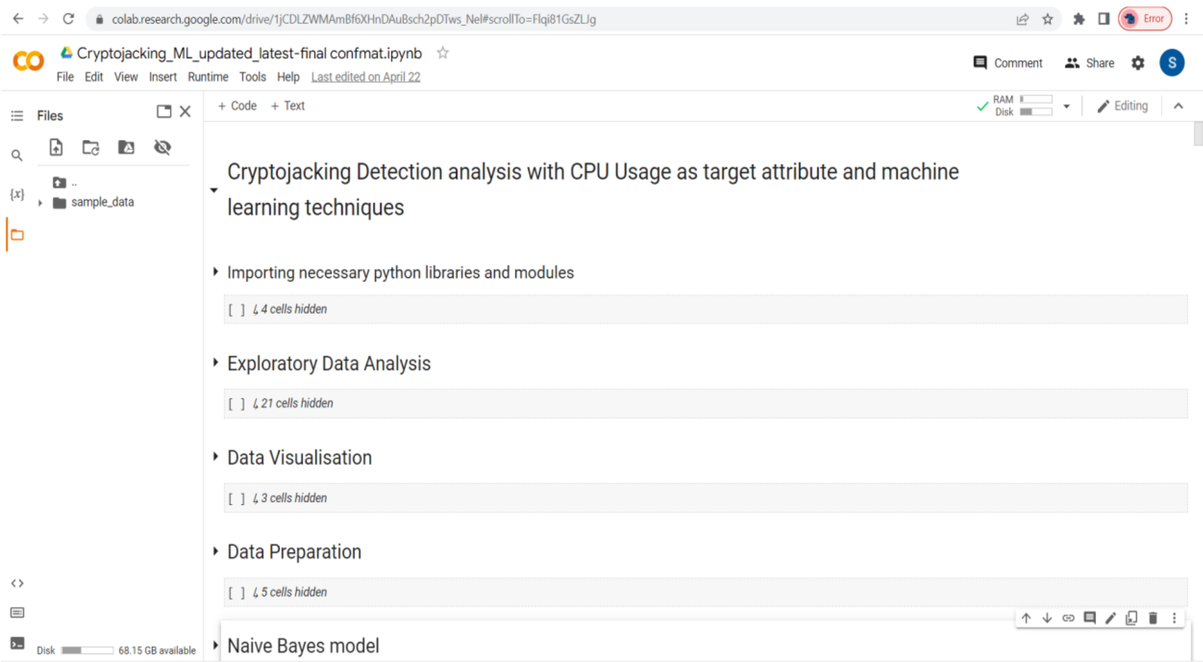

Figure 1: Google Colab Community

3.2. Dataset

The dataset is sourced from kaggle, a publicly accessible website. Kaggle is a website that allows you to search datasets and create predictive models. Upon downloading the data from Kaggle, it was published to Google Drive together with python code created in Google Colab.

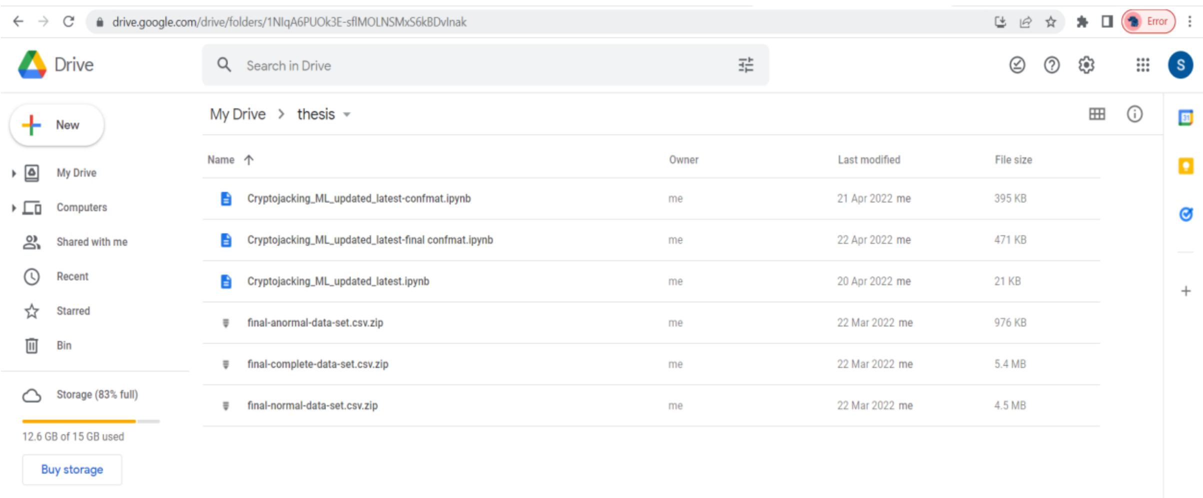

Figure 2: Google Drive Data

3.2.1. Dataset Summary

The dataset consists of three files namely normal, abnormal and complete data.

3.2.1.1. Abnormal dataset

The abnormal dataset includes the time-series performance data during a crypto-hijacking attack.

	cpu_guest	cpu_guest_nice	cpu_idle	cpu_steal	cpu_iowait	cpu_irq	cpu_nice	cpu_softirq	cpu_system	cpu_total
0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
1	0	0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
2	0	0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
3	0	0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
4	0	0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	100.0

5 rows × 82 columns

Figure 3: Data row analysis for abnormal dataset file

```
# Analysis the shape of abnormal dataset file
nRow, nCol = data_abnormal.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 14461 rows and 82 columns
```

Figure 4: Data shape analysis for abnormal data file

3.2.1.2. Normal dataset

The normal dataset includes the time-series performance data during no cryptojacking attack.

	cpu_guest	cpu_guest_nice	cpu_idle	cpu_iowait	cpu_irq	cpu_nice	cpu_softirq	cpu_steal	cpu_system
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.5
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.5
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.3
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0

5 rows × 82 columns

Figure 5: Data row analysis for normal data file

```python
# Analysis the shape of normal dataset file
nRow, nCol = data_normal.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 80851 rows and 82 columns
```

Figure 6: Data shape analysis for normal data file

3.2.1.3. Complete dataset

A collective of abnormal and normal datasets can be found in the complete dataset.

	cpu_guest	cpu_guest_nice	cpu_idle	cpu_iowait	cpu_irq	cpu_nice	cpu_softirq	cpu_steal	cpu_system	cpu_total
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	100.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	100.0

5 rows × 82 columns

Figure 7: Data row analysis for complete data file

```python
# Analysis the shape of complete dataset file
nRow, nCol = data_complete.shape
print(f'There are {nRow} rows and {nCol} columns')
```

```
There are 95312 rows and 82 columns
```

Figure 8: Data shape analysis for complete data file

4. Implementation

This proposal's libraries must be installed before it can be implemented. Those libraries that aren't present can be imported with the pip command.

```
!pip install pefile
```

```
# Importing required libraries

# Libraries for basic mathematical functions and data manipulation
import os
import pandas as pd
import numpy as numpy
import pickle
import pefile

# Libraries for machine learning model training, testing and evaluation
import sklearn.ensemble as ek
from sklearn import model_selection, tree, linear_model
from sklearn.feature_selection import SelectFromModel
#from sklearn.externals import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn import svm
from sklearn.linear_model import LinearRegression
```

Figure 9: Imported Libraries

The dataset is compressed and in csv format. As a result, it must initially be unzipped.

```
# !unzip /content/final-anormal-data-set.csv.zip
# !unzip /content/final-complete-data-set.csv.zip
# !unzip /content/final-normal-data-set.csv.zip
```

Figure 10: Unzipping the dataset

All the operations are performed on complete dataset.

```
# Data type analysis for complete data file
data_complete.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95312 entries, 0 to 95311
Data columns (total 82 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   cpu_guest                       95311 non-null  float64
 1   cpu_guest_nice                  95311 non-null  float64
 2   cpu_idle                        95311 non-null  float64
 3   cpu_iowait                      95311 non-null  float64
 4   cpu_irq                         95310 non-null  float64
 5   cpu_nice                        95310 non-null  float64
 6   cpu_softirq                     95311 non-null  float64
 7   cpu_steal                       95310 non-null  float64
 8   cpu_system                      95311 non-null  float64
 9   cpu_total                       95310 non-null  float64
 10  cpu_user                        95311 non-null  float64
 11  diskio_sda1_disk_name           95312 non-null  object
 12  diskio_sda1_key                 95312 non-null  object
 13  diskio_sda1_read_bytes          95312 non-null  float64
 14  diskio_sda1_time_since_update   95312 non-null  object
 15  diskio_sda1_write_bytes         95312 non-null  int64
 16  diskio_sda_disk_name            95312 non-null  object
 17  diskio_sda_key                  95312 non-null  object
 18  diskio_sda_read_bytes           95312 non-null  float64
 19  diskio_sda_time_since_update    95312 non-null  object
 20  diskio_sda_write_bytes          95312 non-null  int64
 21  fs_/_device_name                95312 non-null  object
 22  fs_/_free                       95312 non-null  int64
 23  fs_/_fs_type                    95312 non-null  object
 24  fs_/_key                        95312 non-null  object
 25  fs_/_mnt_point                  95312 non-null  object
 26  fs_/_percent                    95312 non-null  object
```

Figure 11: Data type analysis for complete datafile

```
# Statistical analysis of the complete dataset
data_complete.describe()
```

	cpu_guest	cpu_guest_nice	cpu_idle	cpu_iowait	cpu_irq	cpu_nice	cpu_softirq	cpu_steal	cpu_system
count	95311.0	95311.0	95311.000000	95311.000000	95310.0	95310.000000	95311.000000	95310.0	95311.000000
mean	0.0	0.0	50.093351	0.004353	0.0	0.004094	0.068611	0.0	3.348590
std	0.0	0.0	45.256234	0.319006	0.0	0.466298	0.271453	0.0	1.521691
min	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.000000
25%	0.0	0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	2.500000
50%	0.0	0.0	88.900000	0.000000	0.0	0.000000	0.000000	0.0	2.800000
75%	0.0	0.0	91.700000	0.000000	0.0	0.000000	0.000000	0.0	3.700000
max	0.0	0.0	100.000000	96.900000	0.0	75.500000	3.000000	0.0	48.000000

8 rows × 58 columns

Figure 12: Statistical data analysis of complete data file

4.1. Data Description for complete dataset

```
# Selecting the Numerical Datatype data attributes
result = data_complete.select_dtypes(include='number')
print(result)
```

```
       cpu_guest  cpu_guest_nice  cpu_idle  cpu_iowait  cpu_irq  cpu_nice  \
0            0.0             0.0       0.0         0.0      0.0       0.0
1            0.0             0.0       0.0         0.0      0.0       0.0
2            0.0             0.0       0.0         0.0      0.0       0.0
3            0.0             0.0       0.0         0.0      0.0       0.0
4            0.0             0.0       0.0         0.0      0.0       0.0
...          ...             ...       ...         ...      ...       ...
95307        0.0             0.0      91.7         0.0      0.0       0.0
95308        0.0             0.0      88.4         0.0      0.0       0.0
95309        0.0             0.0      91.7         0.0      0.0       0.0
95310        0.0             0.0      92.6         0.0      0.0       0.0
95311        0.0             0.0      91.7         0.0      0.0       0.0

       cpu_softirq  cpu_steal  cpu_system  cpu_total  ...  percpu_0_nice  \
0              0.0        0.0         2.0      100.0  ...            0.0
1              0.0        0.0         2.0      100.0  ...            0.0
2              0.0        0.0         2.0      100.0  ...            0.0
3              0.0        0.0         2.0      100.0  ...            0.0
4              0.0        0.0         1.0      100.0  ...            0.0
...            ...        ...         ...        ...  ...            ...
95307          0.0        0.0         2.8        9.9  ...            0.0
95308          0.9        0.0         4.5       10.0  ...            0.0
95309          0.0        0.0         2.8        9.1  ...            0.0
```

Figure 13: Numerical data analysis for the complete dataset

The figure above depicts the numerical data analysis for the complete dataset. For the numerical data analysis data columns with integer or float data types are considered. The data columns with object data types are utilized after label encoding.

4.2. Label encoding for complete dataset

```
#Label encoding
from sklearn.preprocessing import LabelEncoder

df = pd.DataFrame(result)
```

```
#data fitting into label encoding
le = preprocessing.LabelEncoder()
df["load_cpucore"] = le.fit_transform(df["load_cpucore"].astype(str))
```

Figure 14: Label encoding for the complete dataset

The figure above depicts the label encoding for the complete dataset. Label encoding basically includes transforming the data labels into numerical values so that the machine can understand the data attributes and features.

4.3. Data Pre-processing for complete dataset

```
#Splitting the dataset into training and testing
#X = df
X = df.iloc[:,:-1]
#y = df["load_cpucore"].values
y = df.iloc[:,-1]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.33, random_state=1)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(63857, 57) (31453, 57) (63857,) (31453,)
```

Figure 15: Data splitting for the complete dataset

4.4. Data Description, Encoding and Pre-processing for selected 8 column datasets

```
data_dev = data_complete[['cpu_total','cpu_user','load_cpucore','mem_shared','mem_used','processcount_total','timestamp']]

# Selecting the Numerical Datatype data attributes
result_dev = data_dev.select_dtypes(include='number')
print(result_dev)
```

```
   cpu_total  cpu_user  load_cpucore  mem_shared    mem_used  \
0      100.0       6.0             1   9035776.0  3055915008
1      100.0      39.0             1   9035776.0  3059118080
2      100.0      42.0             1   9035776.0  3059699712
3      100.0      40.0             1   9035776.0  3065937920
4      100.0      37.0             1   9035776.0  3068030976
```

Figure 16: Numerical data analysis for the selected 8 column dataset

```
#Label encoding
from sklearn.preprocessing import LabelEncoder

df_dev = pd.DataFrame(result_dev)
```

```
#data fitting into label encoding
le = preprocessing.LabelEncoder()
df_dev["load_cpucore"] = le.fit_transform(df["load_cpucore"].astype(str))
df_dev
```

	cpu_total	cpu_user	load_cpucore	mem_shared	mem_used	processcount_total
0	100.0	6.0	0	9035776.0	3055915008	113.0
1	100.0	39.0	0	9035776.0	3059118080	113.0
2	100.0	42.0	0	9035776.0	3059699712	112.0

Figure 17: Label encoding for the selected 8 column dataset

```
#Splitting the dataset into training and testing
X = df_dev.iloc[:,:-1]
y = df_dev.iloc[:,-1]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.33, random_state=1)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(63857, 5) (31453, 5) (63857,) (31453,)
```

Figure 18: Data splitting for the selected 8 column dataset

4.5. Data Description, Encoding and Pre-processing for selected 16 column datasets

```
data_dev_1 = data_complete[['load_cpucore','mem_used','mem_active','mem_cached','mem_inactive','mem_shared','mem_total','mem_used']]

# Selecting the Numerical Datatype data attributes
result_dev_1 = data_dev_1.select_dtypes(include='number')
print(result_dev_1)
```

```
       load_cpucore    mem_used  mem_active  mem_cached  mem_inactive  \
0                 1  3055915008   368939008   361664512     240955392
1                 1  3059118080   372154368   361693184     240943104
2                 1  3059699712   372920320   362090496     241266688
3                 1  3065937920   379072512   362209280     241238016
4                 1  3068030976   381206528   362266624     241274880
...             ...         ...         ...         ...           ...
95307             1   683212800   580444160   889622528     332763136
95308             1   683712512   580808704   889622528     332759040
95309             1   683753472   580837376   889622528     332759040
95310             1   683745280   580845568   889622528     332759040
95311             1   683712512   580833280   889622528     332759040
```

Figure 19: Numerical data analysis for the selected 16 column dataset

```
#Label encoding
from sklearn.preprocessing import LabelEncoder

df_dev_1 = pd.DataFrame(result_dev_1)
```

```
#data fitting into label encoding
le = preprocessing.LabelEncoder()
df_dev_1["load_cpucore"] = le.fit_transform(df["load_cpucore"].astype(str))
df_dev_1
```

	load_cpucore	mem_used	mem_active	mem_cached	mem_inactive	mem_shared	mem_total	mem_used
0	0	3055915008	368939008	361664512	240955392	9035776.0	3.973603e+09	3055915008
1	0	3059118080	372154368	361693184	240943104	9035776.0	3.973603e+09	3059118080
2	0	3059699712	372920320	362090496	241266688	9035776.0	3.973603e+09	3059699712

Figure 20: Label encoding for the selected 16 column dataset

```
#Splitting the dataset into training and testing
X = df_dev_1.iloc[:,:-1]
y = df_dev_1.iloc[:,-1]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.33, random_state=1)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(63857, 7) (31453, 7) (63857,) (31453,)
```

Figure 21: Data splitting for the selected 16 column dataset

5. Models & Output

The fittings of all four models are mentioned below:

5.1. Model Training

5.1.1. Naïve Bayes model

Naive Bayes model

```
[ ]  from sklearn.naive_bayes import GaussianNB
     nb = GaussianNB()
     nb.fit(X_train,y_train)
     y_pred1=nb.predict(X_test)
```

Figure 22: Naive Bayes model fitting

5.1.2. K-Nearest Neighbor model

K-Nearest Neighbour model

```
[ ]  from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier()
     knn.fit(X_train,y_train)
     y_pred2=knn.predict(X_test)
```

Figure 23: K-Nearest Neighbour model fitting

5.1.3. Decision tree model

Decision Tree model

```
[ ]  from sklearn.tree import DecisionTreeClassifier
     tr = DecisionTreeClassifier()
     tr.fit(X_train,y_train)
     y_pred3=tr.predict(X_test)
```

Figure 24: Decision tree model fitting

5.1.4. Random forest model

Random Forest model

```
[ ]  from sklearn.ensemble import RandomForestClassifier
     rf = RandomForestClassifier()
     rf.fit(X_train,y_train)
     y_pred4=rf.predict(X_test)
```

Figure 25: Random Forest model fitting

5.2. Comparative Analysis

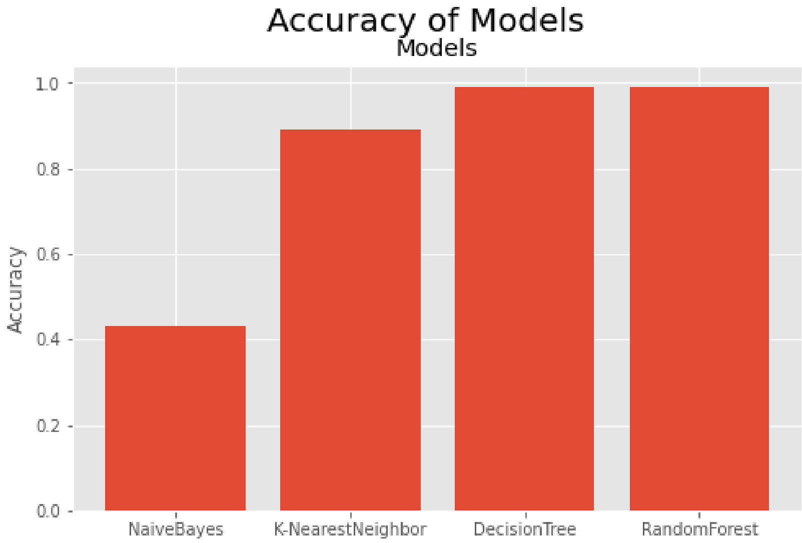

Figure 26: Comparative analysis for complete dataset with respect to accuracy value

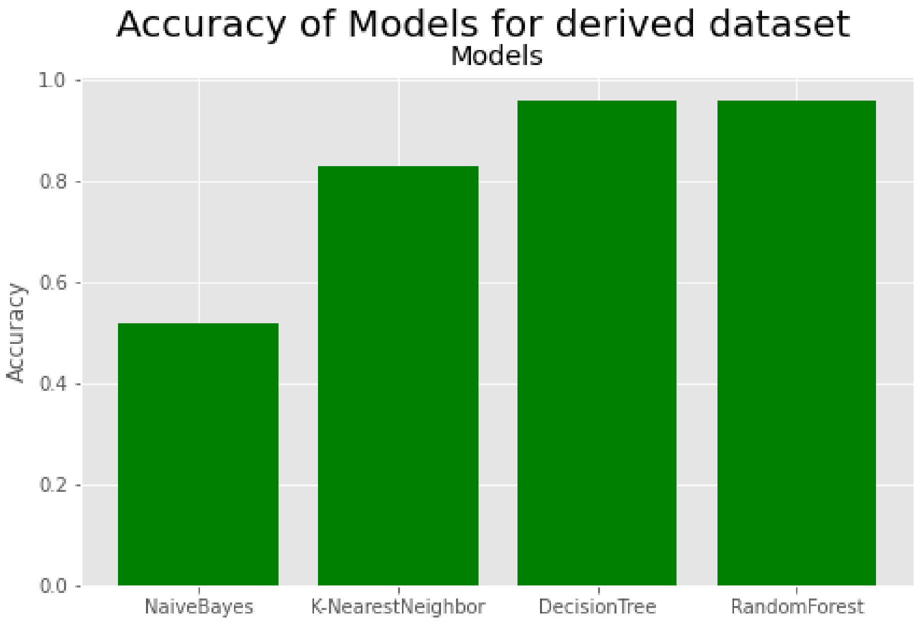

Figure 27: Comparative analysis for selected 8 datasets with respect to accuracy value

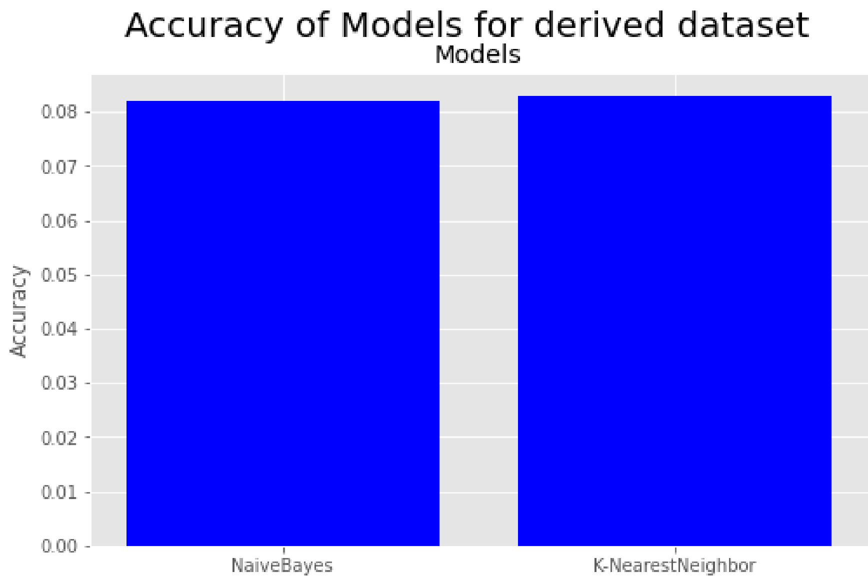

Figure 28: Comparative analysis for selected 16 columns dataset with respect to accuracy value

6. Results

Random Forest and Decision Tree are two of the four algorithms in this categorization approach that have a 99 percent accuracy rate. The KNN model has an accuracy rate of around 89 percent, while the Nave Bayes model has a score of 43 percent.

References

[1] Androulaki, E. et al., 2018. *Hyperledger fabric: a distributed operating system for permissioned blockchains.* s.l., Proceedings of the Thirteenth EuroSys Conference.

[2] Eskandari, S., Andreas Leoutsarakos, Troy Mursch & Jeremy Clark, 2018. *A First Look at Browser-Based Cryptojacking.* London, UK, 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).

[3] Pastrana, S. & Guillermo Suarez-Tangil, 2019. *A First Look at the Crypto-Mining Malware Ecosystem: A Decade of Unrestricted Wealth.* s.l., Proceedings of the Internet Measurement Conference.