Zero Day Malware Detection using Machine Learning Algorithms

Configuration Manual

MSc Internship

Cybersecurity

School of Computing

National College of Ireland

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mohammed Mustafa Raza Choudhry
Student ID:	X20119607
Program:	MSC. Cyber Security **Year:** 2021
Module:	Intership
Lecturer:	Dr. Imran Khan
Submission Due Date:	28th January 2022
Project Title:	Zero Day Malware Detection using supervised and unsupervised Machine learning Algorithms
Word Count: 1919	**Page Count:** 22

I hereby certify that the information contained in this (my submission) is research information I conducted for this project. All information other than my 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.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: Mohammed Mustafa Raza Choudhry

Date: 14-12-2021

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 reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the 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):	

Zero Day Malware Detection using Machine Learning Algorithms

INTRODUCTION

The configuration manual contains comprehensive guidance about how the research program is designed and executed, this involves system requirements and coding requirements. For developing the algorithms, Python coding language is used as it supports many machine learning libraries. 2 datasets one is UCI and Virus Share are considered for evaluation of performance metrics likeaccuracy, recall, F1 score, precision based on the actual number of malware and normal samples in the dataset with the output sample produced which is a true positive or a false positive. The lesserthe value of false positive, the greater is the accuracy of the algorithm. First, the dataset considered is passed through a pre-processing algorithm wherein only relevant features havingimportance according to the statistics are extracted from the whole sample which helps in optimizing our training set. This set is tested with unsupervised and supervised algorithms likeLogistic Regression, Decision Tree, Naïve Bayes, KNN, Random Forest, and the output determines whether the application is malicious or harmless. Among which Random forest gives highest accuracy.

INITIAL SETUP

- RAM: 8GB
- DISK SPACE REQUIRED: 16GB
- OS: WINDOWS 10
- Softwares Used: Google Collab, ANACONDA, JUPYTER NOTEBOOK, SPYDER
- The prototype is tested on localhost

Anaconda [1]

The version of anaconda we have installed is 4.10.1. This will be used to manage all our environments created in python.

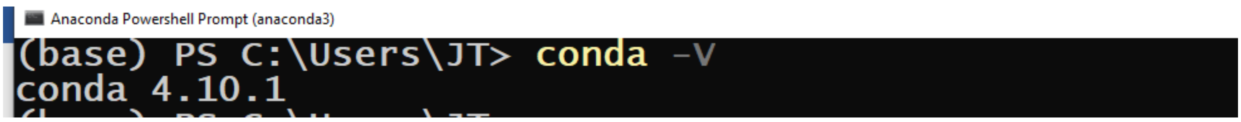


Fig. 1. Conda version

The version of python we have installed is 3.7.8. Python is installed from within Anaconda Navigator.

Fig. 2. Python version

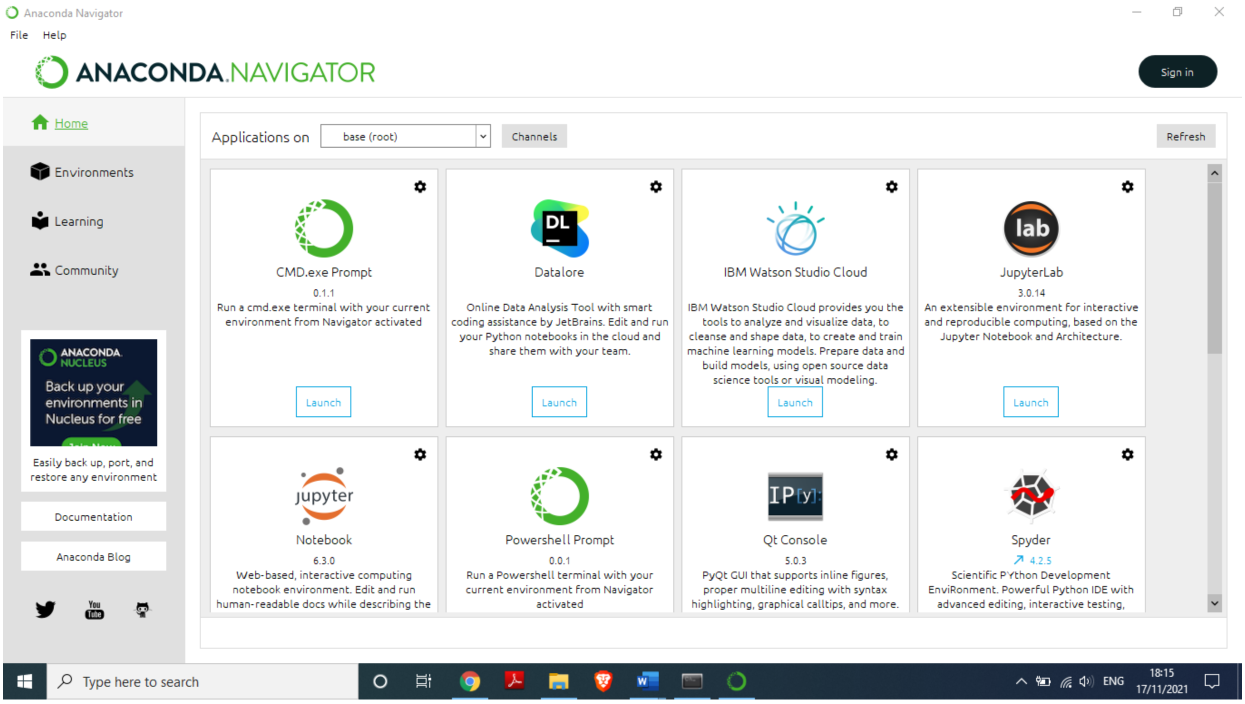


Fig. 3. Anaconda GUI

Jupyter is used for coding and managing our algorithms.

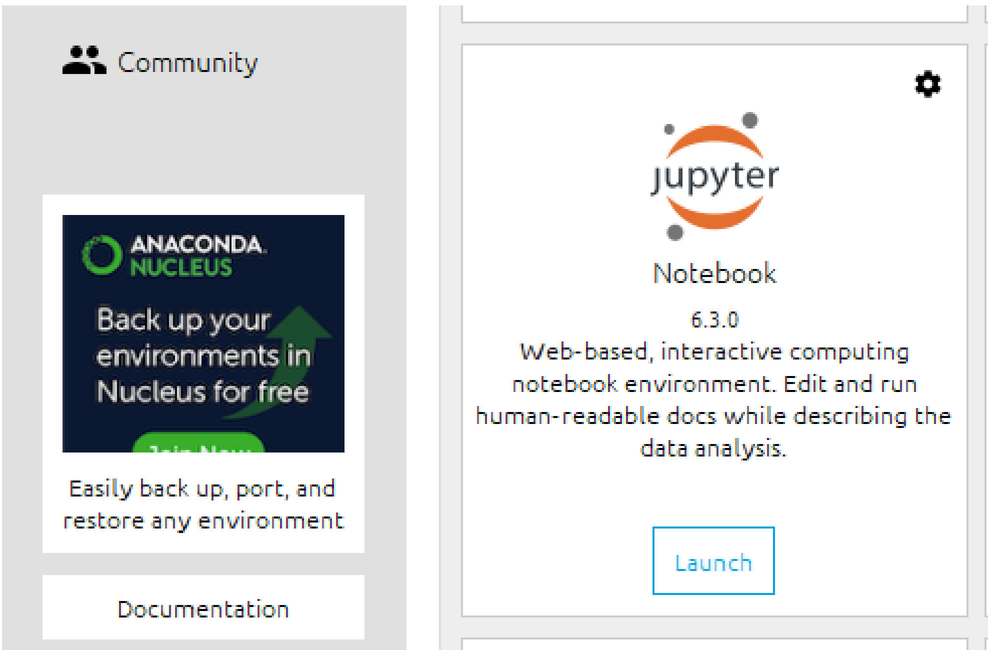
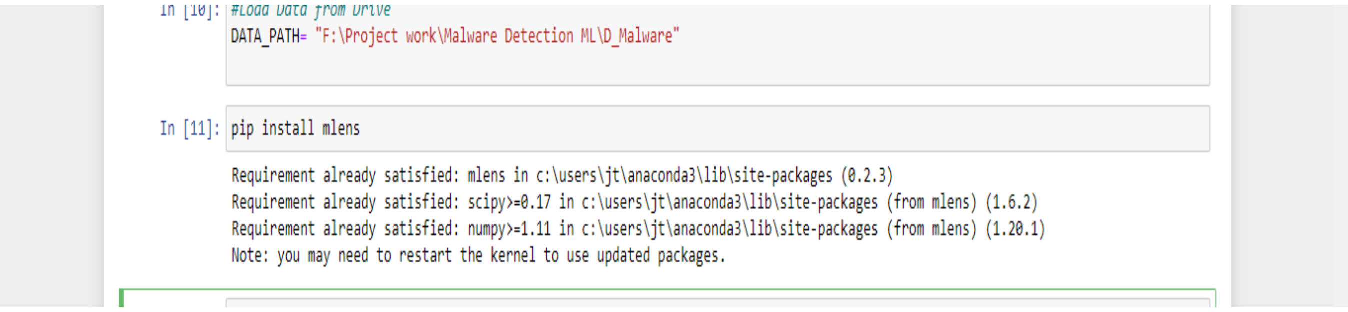


Fig. 3. Jupyter Notebook Version

For Dataset we have used panda library function to read dataset and used dataframe in which our dataset will be inserted. Here we have created DATA_PATH Variable which will used in loading and reading of Data.



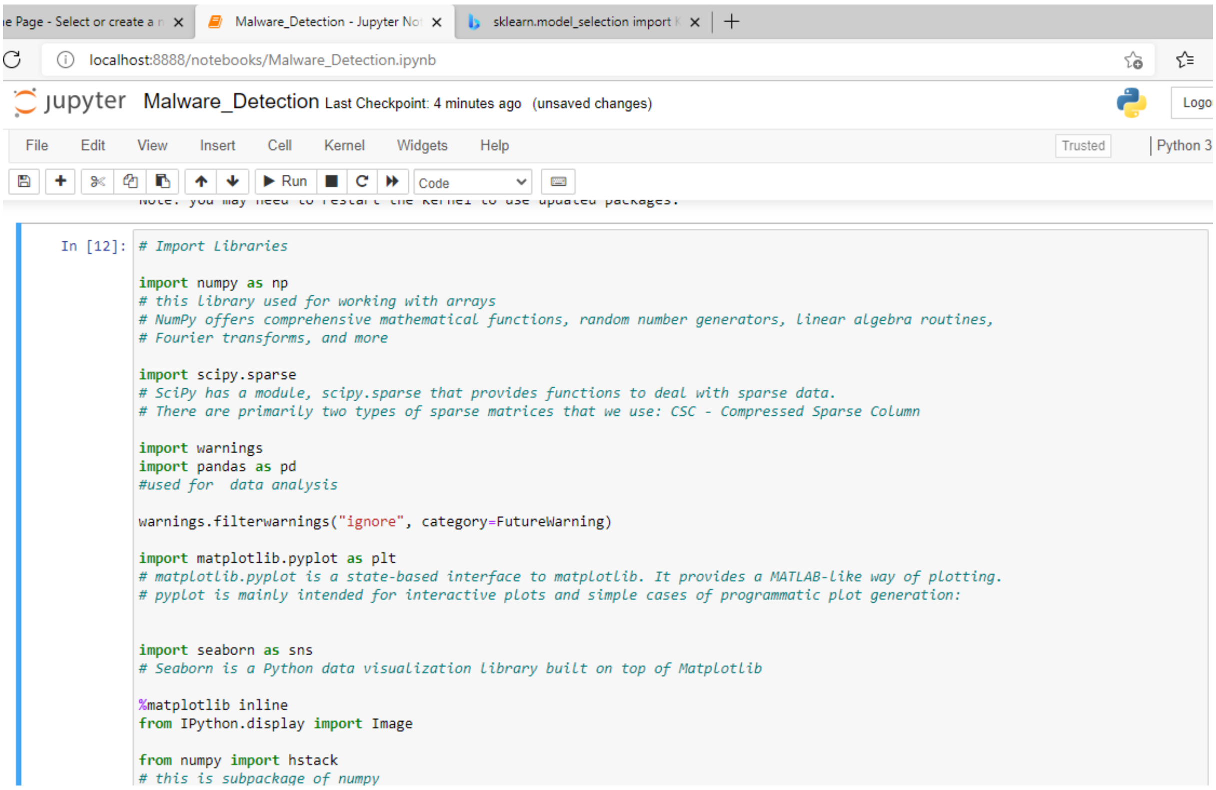
```
In [10]: #Load Data from Drive
         DATA_PATH= "F:\Project work\Malware Detection ML\D_Malware"

In [11]: pip install mlens

         Requirement already satisfied: mlens in c:\users\jt\anaconda3\lib\site-packages (0.2.3)
         Requirement already satisfied: scipy>=0.17 in c:\users\jt\anaconda3\lib\site-packages (from mlens) (1.6.2)
         Requirement already satisfied: numpy>=1.11 in c:\users\jt\anaconda3\lib\site-packages (from mlens) (1.20.1)
         Note: you may need to restart the kernel to use updated packages.
```

Fig. 4. DATA_PATH for Dataset.

For the pre-processing of data and also for the whole experiment we will require a few essential Python libraries which will ease the whole process of this study. Few popularly used Machine Learning libraries are used. Libraries along with their basic functions are listed in the figure below. [5]




```
In [12]: # Import Libraries

import numpy as np
# this library used for working with arrays
# NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines,
# Fourier transforms, and more

import scipy.sparse
# SciPy has a module, scipy.sparse that provides functions to deal with sparse data.
# There are primarily two types of sparse matrices that we use: CSC - Compressed Sparse Column

import warnings
import pandas as pd
#used for  data analysis

warnings.filterwarnings("ignore", category=FutureWarning)

import matplotlib.pyplot as plt
# matplotlib.pyplot is a state-based interface to matplotlib. It provides a MATLAB-like way of plotting.
# pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation:


import seaborn as sns
# Seaborn is a Python data visualization library built on top of Matplotlib

%matplotlib inline
from IPython.display import Image

from numpy import hstack
# this is subpackage of numpy
```

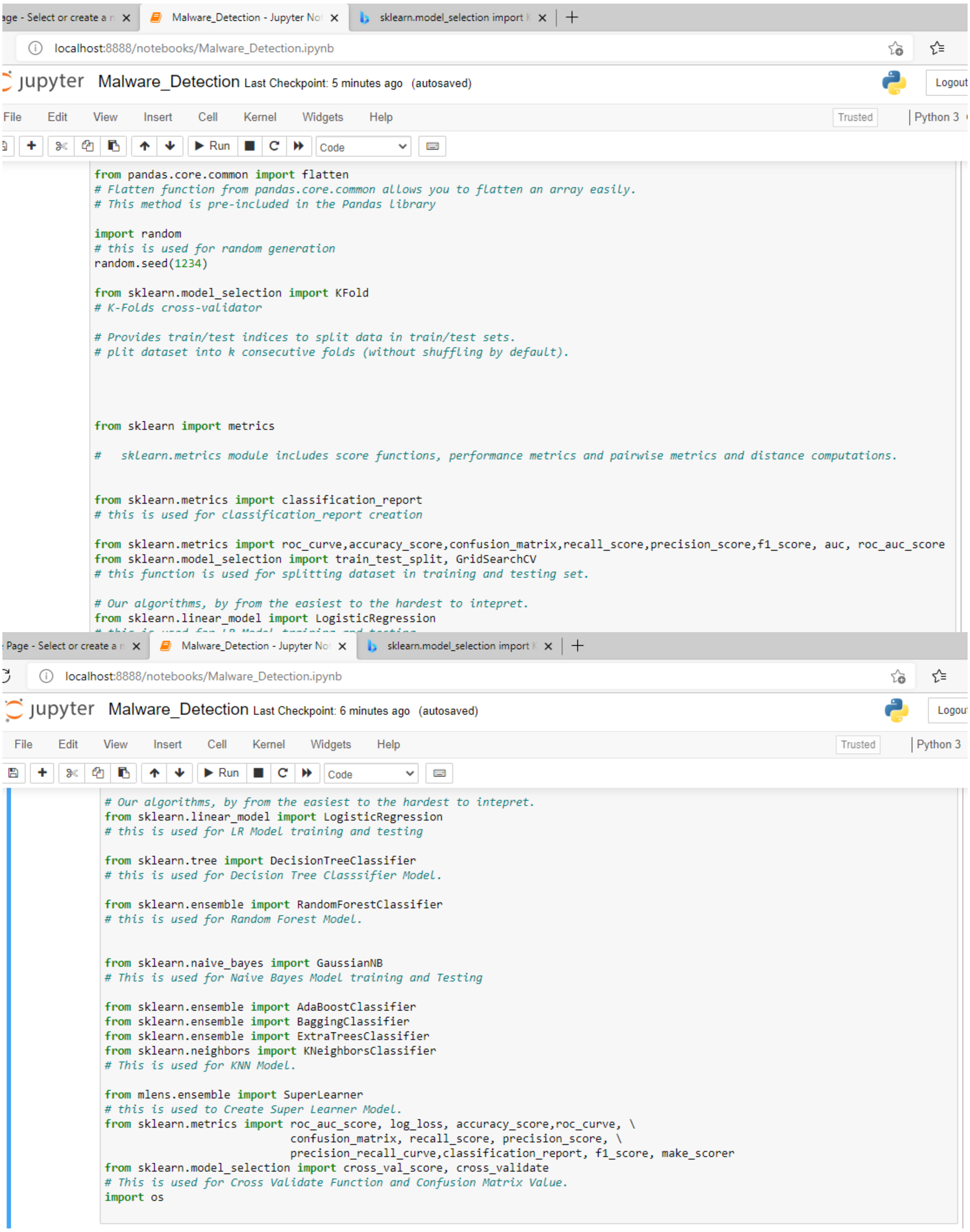


Fig. 5. Installed Libraries with their use commented

After importing all the libraries, our environment is ready for developing the algorithms. So now at first, we will feed our dataset to the pre-processing stage wherein all the data samples are analyzed and optimized before going through the algorithms to improve the efficiency of machine learning algorithms and also to greatly reduce the time taken for algorithms to generate output.

In this pre-processing stage, the data in the samples is

- First, eliminates duplicate and not null values,
- Calculating coefficient
- Identifying important features
- Model Training
- Creating Super Learne

```
In [15]: # Read Data

df =pd.read_csv(r"F:\Project work\Malware Detection ML\D_Malware\uci_malware_detection.csv")
print(df.shape)
df.head()

(373, 532)
```

Out[15]:

	Label	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	...	F_522	F_523	F_524	F_525	F_526	F_527	F_528	F_529	F_530	F_531
0	non-malicious	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
1	non-malicious	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
2	non-malicious	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
3	non-malicious	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
4	non-malicious	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0

5 rows × 532 columns

Fig. 6. Data Reading

In above figure we can see that we have created data frame with using our UCI dataset .

```
In [16]: # Labeling Malicious and Non Malicious

df['Label'] = df['Label'].map({'non-malicious': 1, 'malicious': 0})
print(df.shape)
df.head()

(373, 532)
```

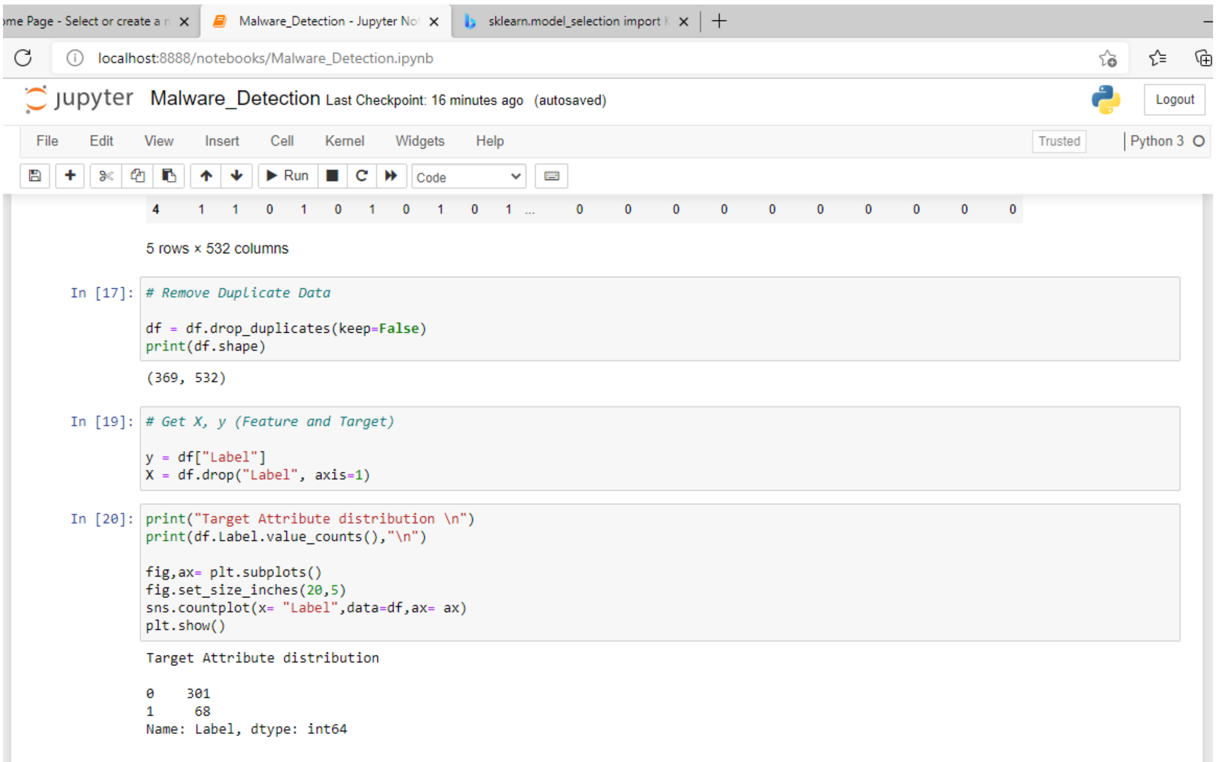
Out[16]:

	Label	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	...	F_522	F_523	F_524	F_525	F_526	F_527	F_528	F_529	F_530	F_531
0	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
2	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
4	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0

5 rows × 532 columns

Fig. 7. Data Labeling

In this step we have labelled our dataset for malicious and non-malicious files using 0 and 1.



me Page - Select or create a | Malware_Detection - Jupyter No | sklearn.model_selection import

localhost:8888/notebooks/Malware_Detection.ipynb

jupyter Malware_Detection Last Checkpoint: 16 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

4	1	1	0	1	0	1	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0

5 rows × 532 columns

In [17]:
```
# Remove Duplicate Data

df = df.drop_duplicates(keep=False)
print(df.shape)
```

```
(369, 532)
```

In [19]:
```
# Get X, y (Feature and Target)

y = df["Label"]
X = df.drop("Label", axis=1)
```

In [20]:
```
print("Target Attribute distribution \n")
print(df.Label.value_counts(),"\n")

fig,ax= plt.subplots()
fig.set_size_inches(20,5)
sns.countplot(x= "Label",data=df,ax= ax)
plt.show()
```

```
Target Attribute distribution

0    301
1     68
Name: Label, dtype: int64
```

Fig. 8. Data Cleaning and Target distribution

In above figure we have removed duplicate data and defined feature and target attributes and checked the distribution of data. In this dataset we have 369 total entries including malicious and harmless files out of which we found there are 68 harmless and 301 malicious files data contained in our Dataset.

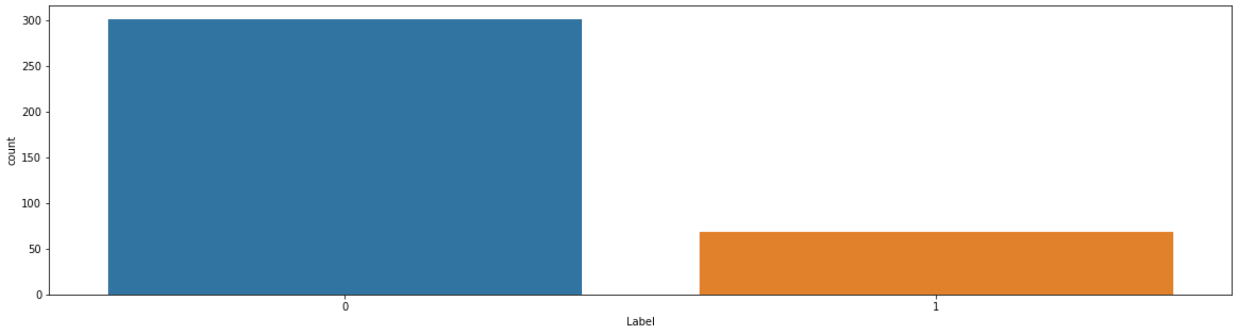


Fig. 9. Distribution of Data

After our new and optimized dataset is ready, we will split our dataset into proportions of 30% - 70% as test dataset and training dataset. Here we found dataset is randomly splitted in training and testing dataset and we have 295 samples for training and remaining 74 samples used for testing of our models.

```
In [21]: # Training and Testing Split Data 80-20

         from sklearn.model_selection import train_test_split

         X_train, X_test, y_train, y_test= train_test_split(X,y, test_size=0.2, random_state=42)

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

         (295, 531) (295,)
         (74, 531) (74,)
```

Fig. 9. Splitting of Data

```
In [22]: #Display binary confusion matrix using Seaborn heatmap
         def confusion_plot(matrix, labels=None):
             labels = labels if labels else ['Negative (0)', 'Positive (1)']

             fig, ax = plt.subplots(nrows=1, ncols=1)
             sns.heatmap(data=matrix, cmap='BuPu', annot=True, fmt='d',
                         xticklabels=labels, yticklabels=labels, ax=ax)
             ax.set_xlabel('PREDICTED')
             ax.set_ylabel('ACTUAL')
             ax.set_title('Confusion Matrix')
             plt.close()

             return fig
```

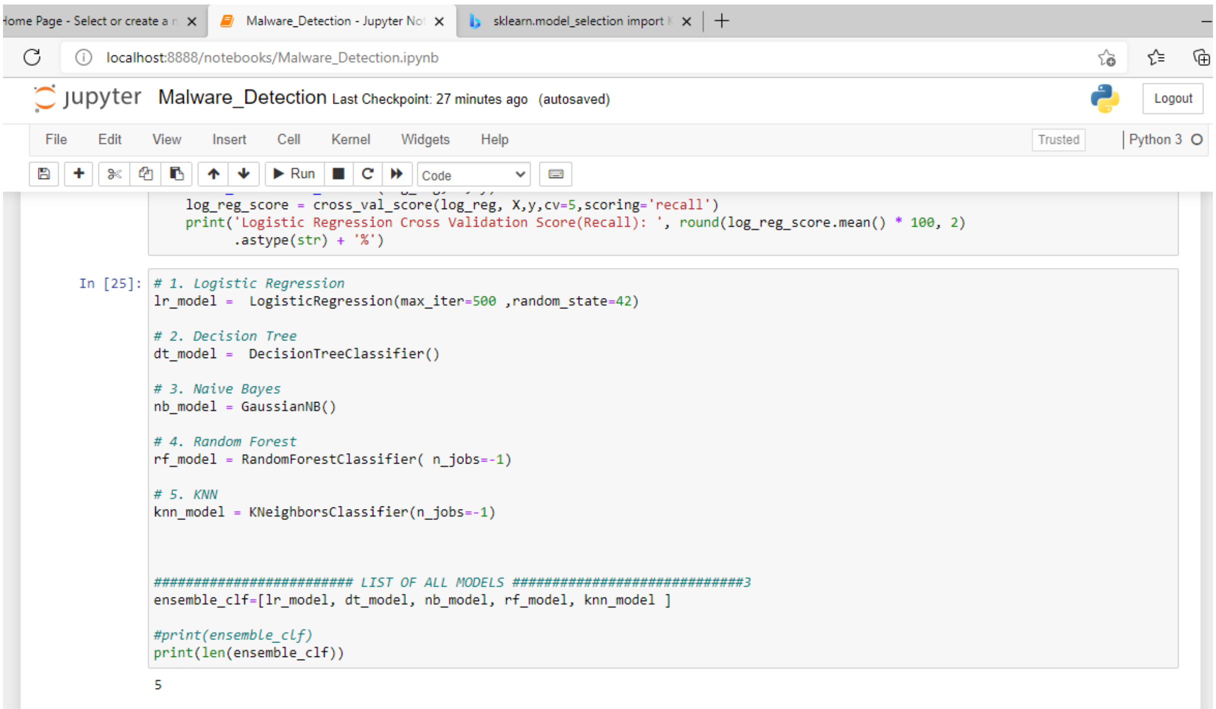
Fig. 10. Function for Confusion matrix

In above figure we have created function for displaying confusion matrix of our models which will be used after the training of dataset.

```
In [24]: #  Function to print cross validation score
         scoring = {'recall' : make_scorer(recall_score)}
         def cross_validation_metrics(log_reg, X, y):
             log_reg_score = cross_val_score(log_reg, X,y,cv=5,scoring='recall')
             print('Logistic Regression Cross Validation Score(Recall): ', round(log_reg_score.mean() * 100, 2)
                   .astype(str) + '%')
```

Fig. 10. Function for Cross Validation

In above figure we have created function for displaying cross validate score of our models which will be used after the training of dataset.




```
    log_reg_score = cross_val_score(log_reg, X,y,cv=5,scoring='recall')
    print('Logistic Regression Cross Validation Score(Recall): ', round(log_reg_score.mean() * 100, 2)
          .astype(str) + '%')

In [25]: # 1. Logistic Regression
lr_model =  LogisticRegression(max_iter=500 ,random_state=42)

# 2. Decision Tree
dt_model =  DecisionTreeClassifier()

# 3. Naive Bayes
nb_model = GaussianNB()

# 4. Random Forest
rf_model = RandomForestClassifier( n_jobs=-1)

# 5. KNN
knn_model = KNeighborsClassifier(n_jobs=-1)


######################### LIST OF ALL MODELS ##############################3
ensemble_clf=[lr_model, dt_model, nb_model, rf_model, knn_model ]

#print(ensemble_clf)
print(len(ensemble_clf))

5
```

Fig. 10. Model Creation and adding models in ensemble.

Here we have created five models using skilit learn library function and add into ensemble so all model can be trained and tested in single function.

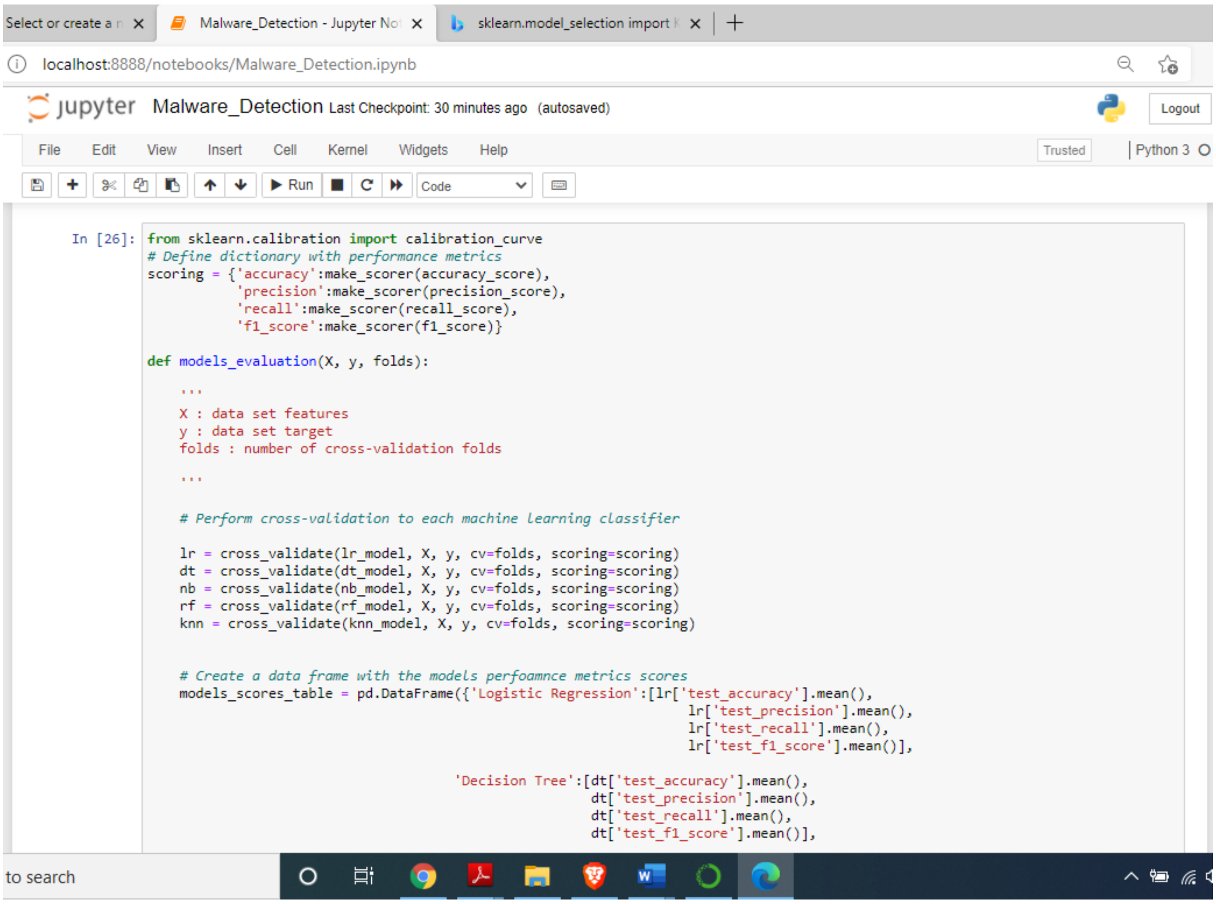


Fig. 10. Model evaluate function

The more is the accuracy of an algorithm, the greater is the probability of a sample being ham.

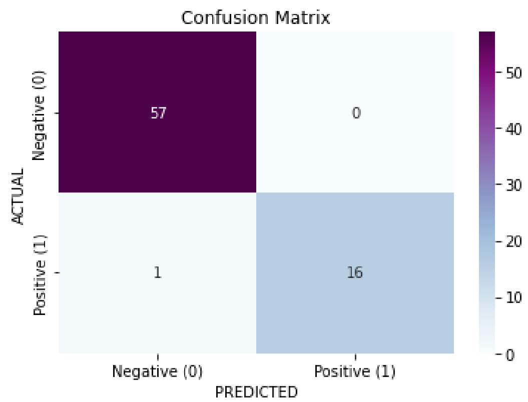


Figure 11: CF Matrix of LR

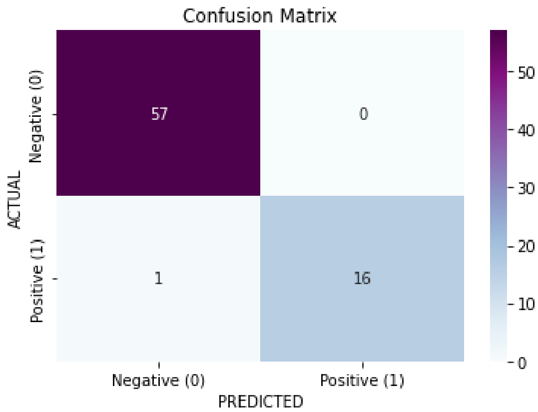


Figure 12: CF Matrix of DT

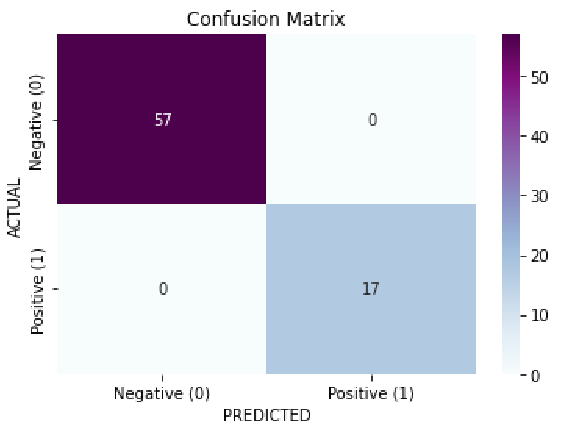


Figure 13: CF Matrix of NB

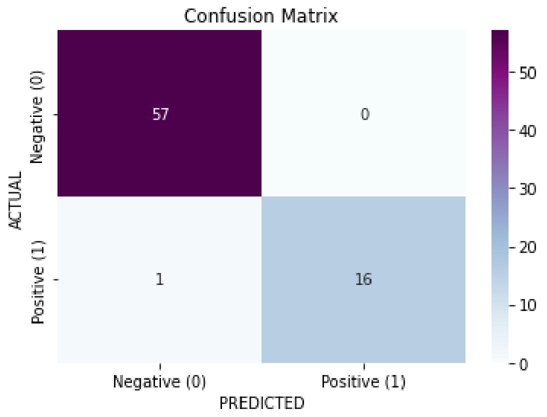


Figure 14: CF Matrix of RF

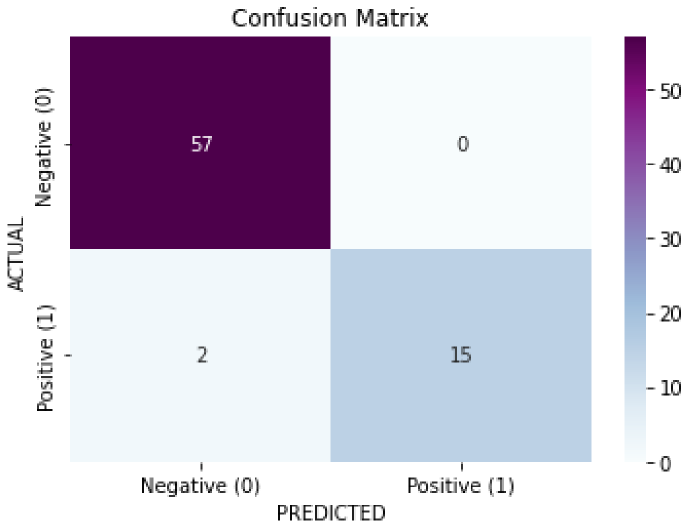


Figure 15: CF Matrix of KNN

In above figure we have displayed different confusion matrix of all five ML models on UCI malware dataset which is taken from Kaggle, we can see that in this dataset Naïve Bayes provide lowest rate of False Positive and false negative and KNN provide highest rate of false positive and false negative

Table 1: ML Model Result Comparison (UCI Malware Dataset)

	Logistic Regression	Decision Tree	Random Forest	Naive Bayes	KNN	Best Score
Accuracy	0.994595	0.989152	0.991892	0.997297	0.981044	Naive Bayes
Precision	0.985714	0.985714	0.985714	0.986667	0.983333	Naive Bayes
Recall	0.985714	0.957143	0.971429	1.000000	0.913187	Naive Bayes
F1 Score	0.985714	0.969801	0.978307	0.993103	0.945846	Naive Bayes

In above table best score in accuracy, precision,recall and F1 Score has been achived in Naïve bayes. Naïve bayes provides good result than random forest in this dataset.

In figure 16 we can see that testing accuracy of naïve bayes model remains high than all other ML models and provides good testing results.

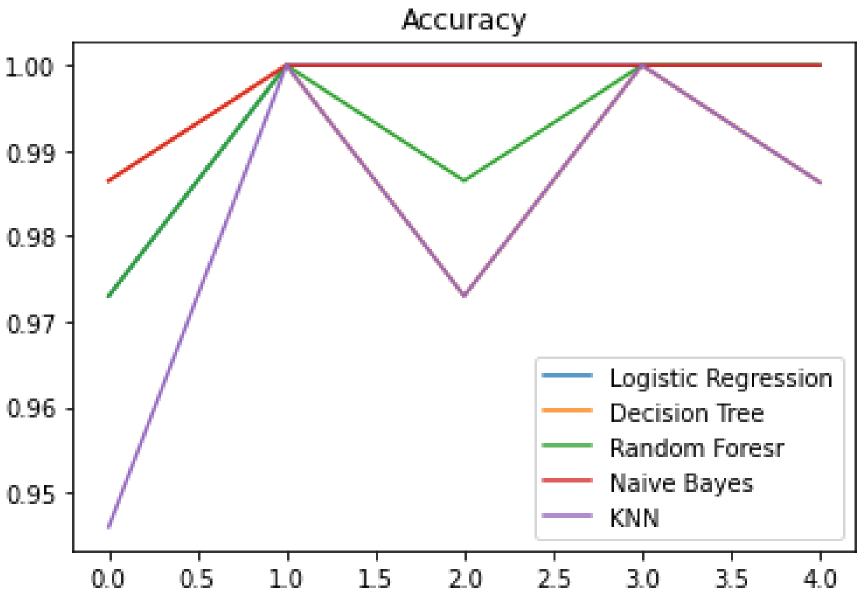


Figure 16: Accuracy of Testing of ML Models

In figure 16 we have displayed calibration plots of all five models on UCI dataset.

Virus Share Dataset

As virus share dataset is very large so we have used google colab platform as machine learning algorithms require lots of processing so google is providing **Google Colaboratory** is a free online cloud-based Jupyter notebook environment that allows us to train our machine learning and deep learning models on CPUs, GPUs, and TPUs.

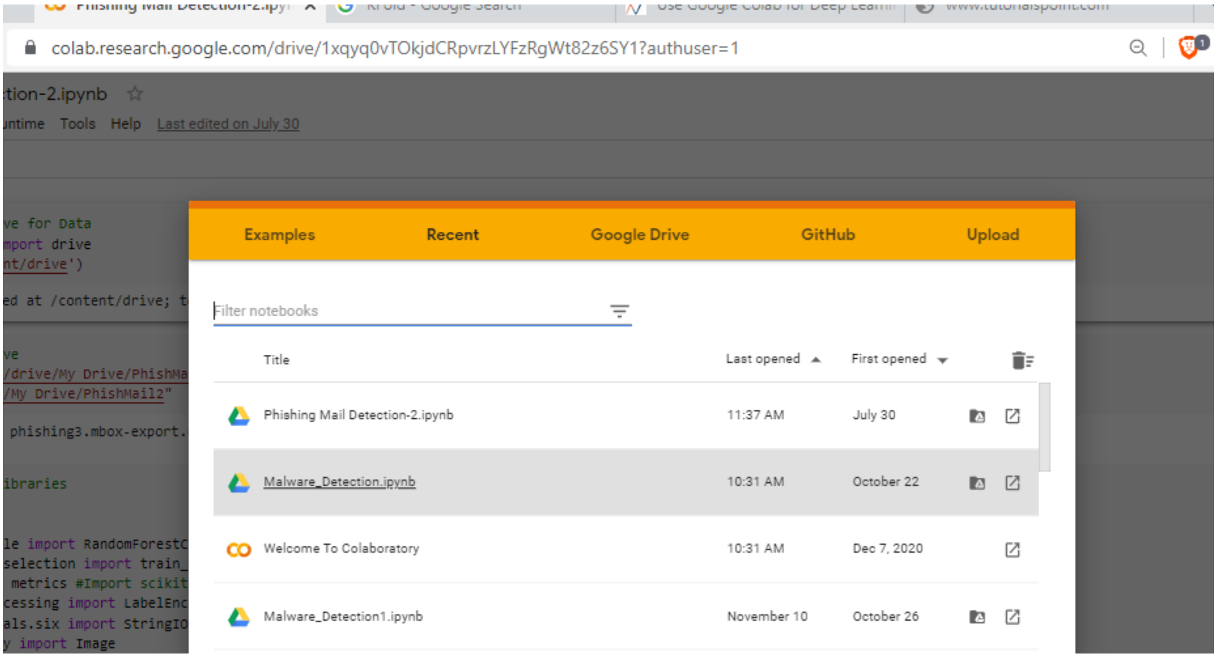


Figure 17: Google Colab

Google Colab Runtimes – Choosing the GPU or TPU Option

The ability to choose different types of runtimes is what makes Colab so popular and powerful.

Here are the steps to change the runtime of your notebook:

Step 1: Click 'Runtime' on the top menu and select 'Change Runtime Type':

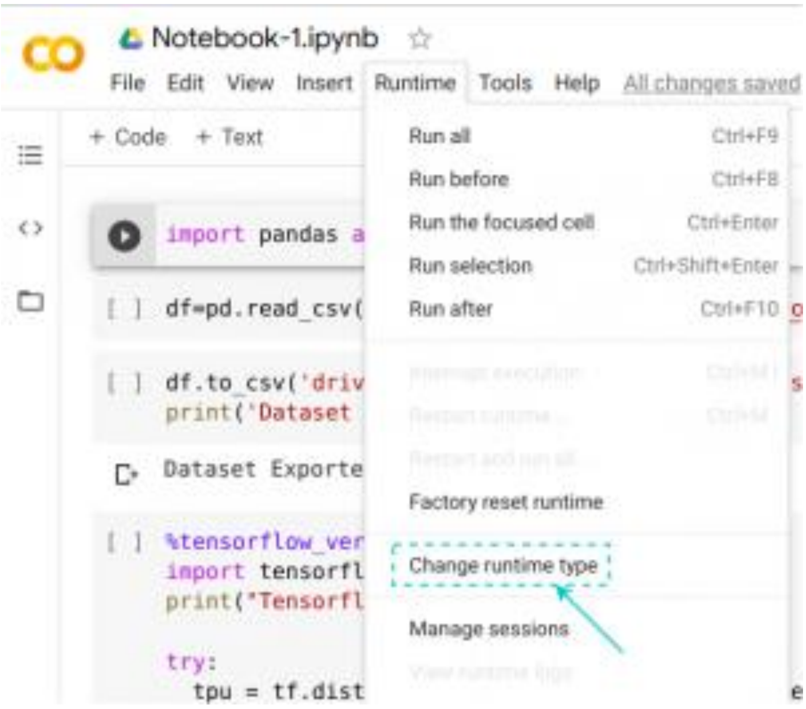


Figure 18: Google Colab Runtime Type

Step 2: Here you can change the runtime according to your need:

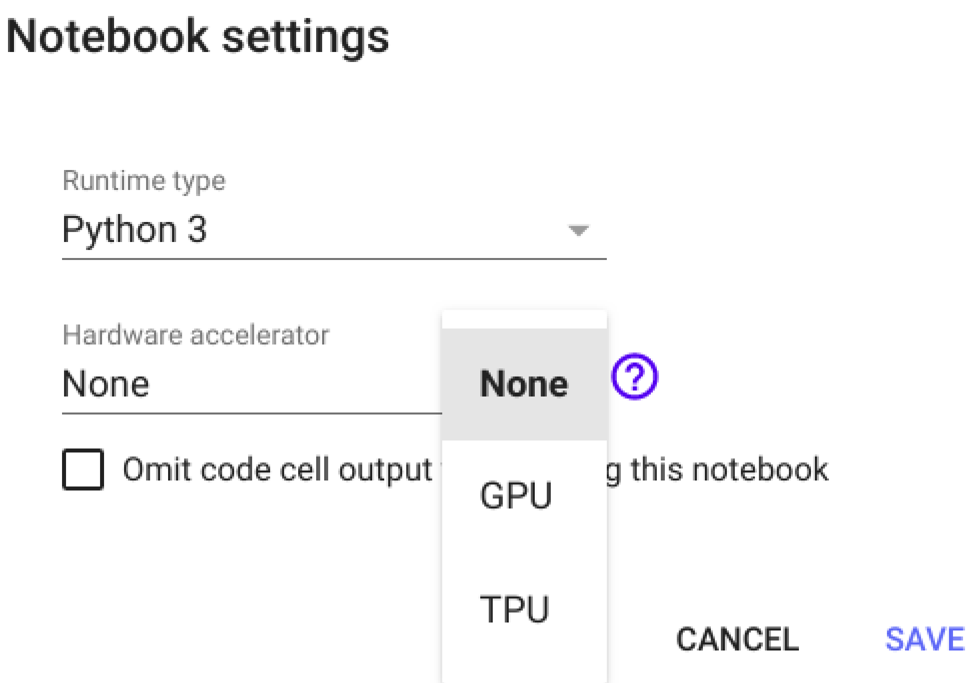


Figure 19: Google Colab Runtime Type

For Implementation we have used google colab platform which is freely available and providing computing resource for implementing various deep learning and machine learning algorithms. We have first uploaded our two datasets in google drive and then drive is mounted using google authenticator.

Reading Data

```
df = pd.read_csv(DATA_PATH+"MalwareData.csv",sep ="|")
```

```
df.head()
```

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfUnin
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	115712	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	19968	
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	621568	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	369152	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	247296	

Figure 20: Reading Virus Share Dataset

In above figure we can see that we have created a dataframe in which we have added our virusshare dataset.

```
# This method prints information about a DataFrame including the index dtype
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 138047 entries, 0 to 138046
Data columns (total 58 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   index                         138047 non-null  int64
 1   Name                          138047 non-null  object
 2   md5                           138047 non-null  object
 3   Machine                       138047 non-null  int64
 4   SizeOfOptionalHeader          138047 non-null  int64
 5   Characteristics               138047 non-null  int64
 6   MajorLinkerVersion            138047 non-null  int64
 7   MinorLinkerVersion            138047 non-null  int64
 8   SizeOfCode                    138047 non-null  int64
 9   SizeOfInitializedData         138047 non-null  int64
 10  SizeOfUninitializedData       138047 non-null  int64
 11  AddressOfEntryPoint           138047 non-null  int64
 12  BaseOfCode                    138047 non-null  int64
 13  BaseOfData                    138047 non-null  int64
 14  ImageBase                     138047 non-null  float64
 15  SectionAlignment              138047 non-null  int64
 16  FileAlignment                 138047 non-null  int64
 17  MajorOperatingSystemVersion   138047 non-null  int64
 18  MinorOperatingSystemVersion   138047 non-null  int64
 19  MajorImageVersion             138047 non-null  int64
 20  MinorImageVersion             138047 non-null  int64
 21  MajorSubsystemVersion         138047 non-null  int64
 22  MinorSubsystemVersion         138047 non-null  int64
 23  SizeOfImage                   138047 non-null  int64
 24  SizeOfHeaders                 138047 non-null  int64
 25  CheckSum                      138047 non-null  int64
 26  Subsystem                     138047 non-null  int64
 27  DllCharacteristics            138047 non-null  int64
 28  SizeOfStackReserve            138047 non-null  int64
 29  SizeOfStackCommit             138047 non-null  int64
 30  SizeOfHeapReserve             138047 non-null  int64
```

Figure 21: Data Cleaning

In above figure we have checked for any null values in our features so we can clean dataset and we found there are no null values in any feature. If null values or data cleaning will not done than some data overfitting and underfitting issues may reside. In our dataset there were total 58 features and all are checked against null values.

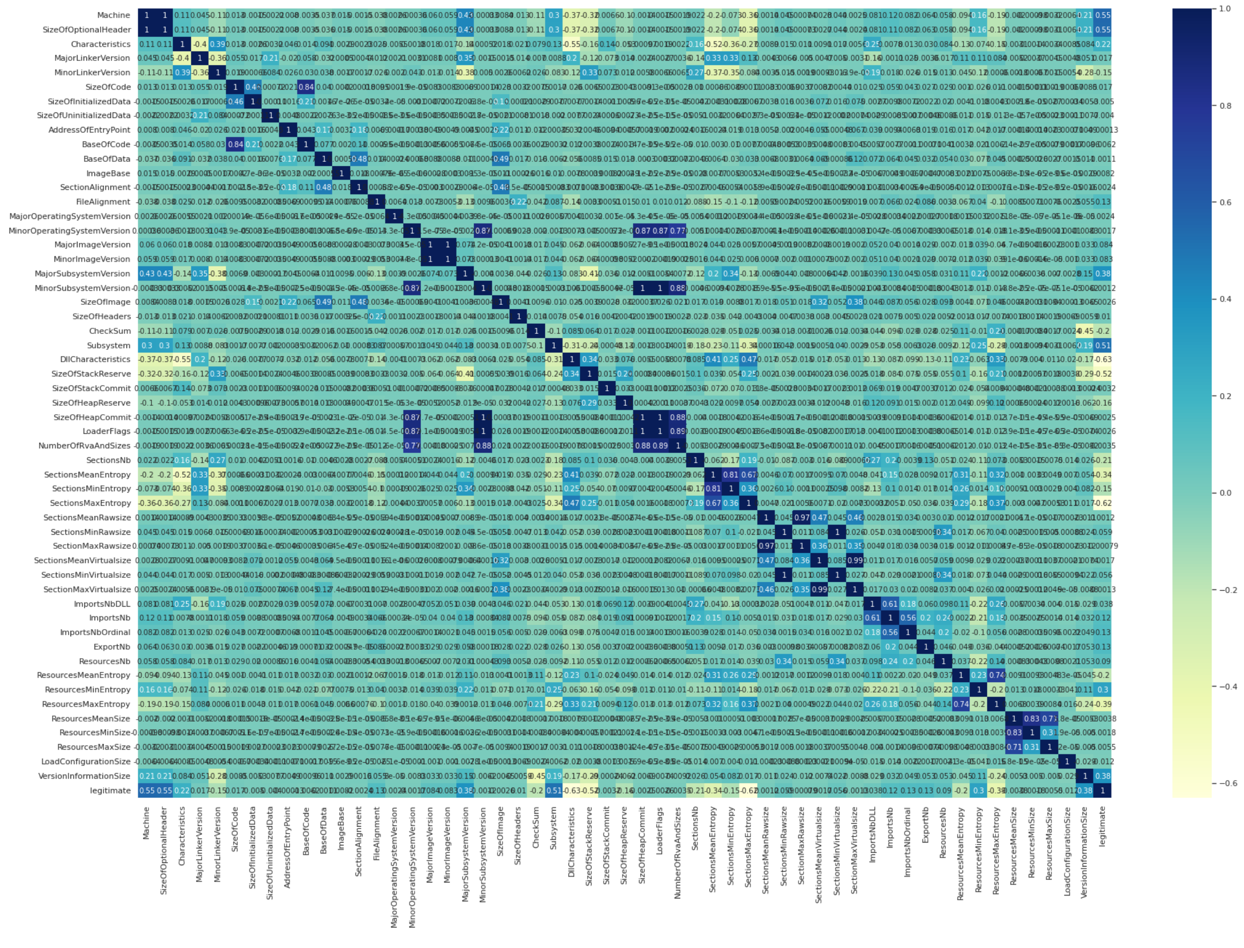


Figure 22: Feature correlation

Correlation is a proportion of the straight relationship of at least 2 factors. Through relationship, we can foresee one variable from the other. The rationale behind utilizing relationship for include choice is that the acceptable factors are profoundly corresponded with the objective. Moreover, factors ought to be corresponded with the objective yet ought to be uncorrelated among themselves. If two factors are associated, we can foresee one from the other. Along these lines, if two highlights are connected, the model just actually needs one of them, as the subsequent one doesn't add extra data. We have utilized the Pearson Correlation here.

Feature extraction can be used to characterized as changing the huge, ambiguous assortment of contributions to the arrangement of highlights. Progressed identification mainly depends on highlighting of extraction of the malicious files being examined. Feature could contain various plaintext strings found in the dismantled documents, the size of the malware, n-gram byte arrangements, framework asset data like the arrangement of DLLs, and so forth by utilizing AI calculation, these highlights are given as sources of info.

```
#Selecting highly correlated features
relevant_features = corr_target[corr_target>0.2]
relevant_features
df1 = relevant_features.sort_values()
df1
```

```
ResourcesMeanEntropy       0.202432
SectionsNb                 0.207782
Characteristics            0.221956
ResourcesMinEntropy        0.299112
SectionsMeanEntropy        0.343933
VersionInformationSize     0.379646
MajorSubsystemVersion      0.380393
ResourcesMaxEntropy        0.392855
Subsystem                  0.514352
SizeOfStackReserve         0.521642
SizeOfOptionalHeader       0.547498
Machine                    0.548835
SectionsMaxEntropy         0.624229
DllCharacteristics         0.630177
legitimate                 1.000000
Name: legitimate, dtype: float64
```

Figure 23: Feature Selection

We have used coefficient to find important features and used the features in which coefficient is grater than 0.2 for training and testing and other features were not used as only important features have been selected. From this method we have identified 14 important features out of 58 features. In below figure we have checked the distribution of values in 14 important features.

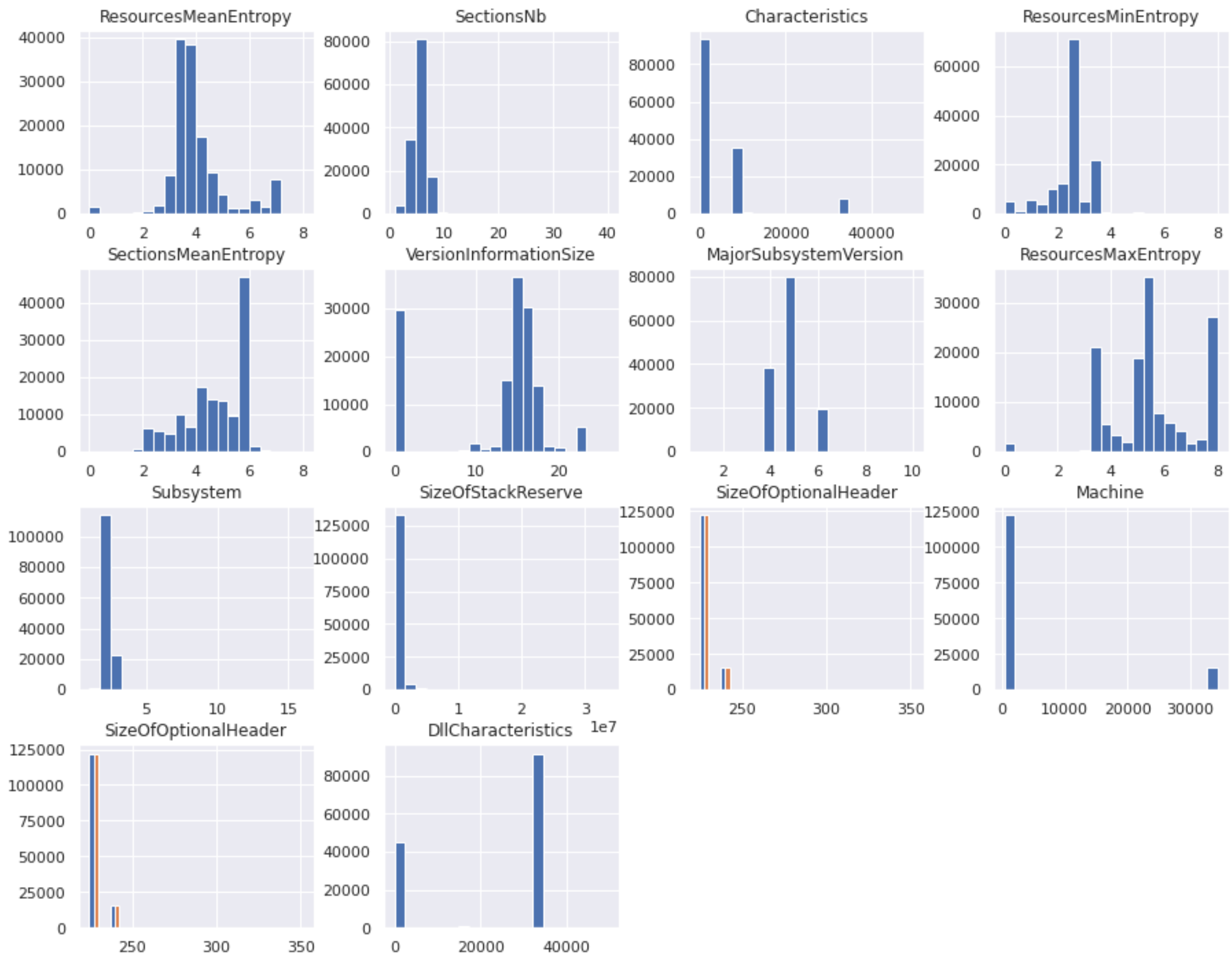


Figure 24: Values distribution of important features

Split the data in Training and testing

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=7)
```

```
[ ] X_train.shape, X_test.shape, Y_train.shape, Y_test.shape

((96632, 14), (41415, 14), (96632,), (41415,))
```

Figure 25: Training and Testing Data Split

In above figure we have split the data for training and testing in 70/30 using sklearn library function. Here we found we have 966632 samples for training and 41415 random samples for testing.

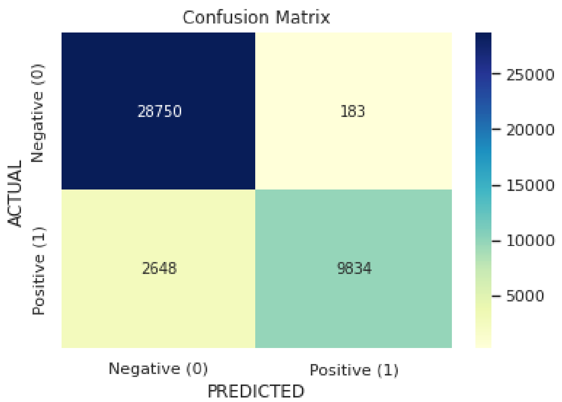


Figure 10: CF Matrix of LR

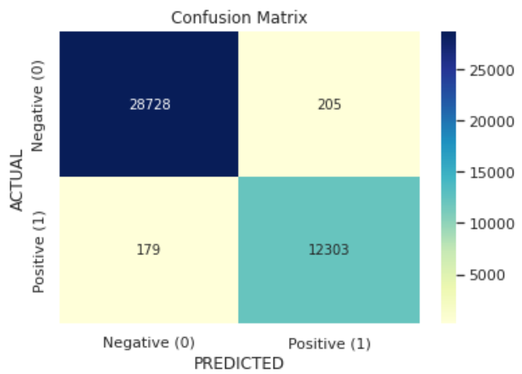


Figure 11: CF Matrix of DT

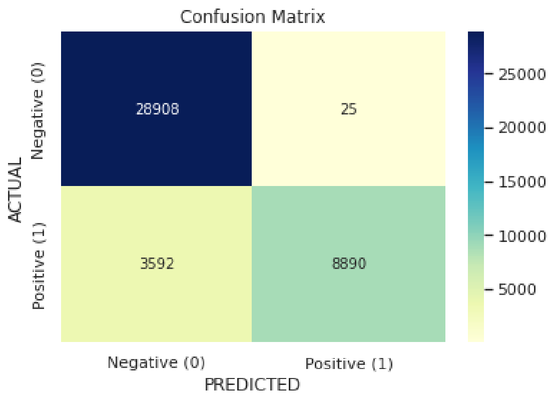


Figure 12: CF Matrix of NB

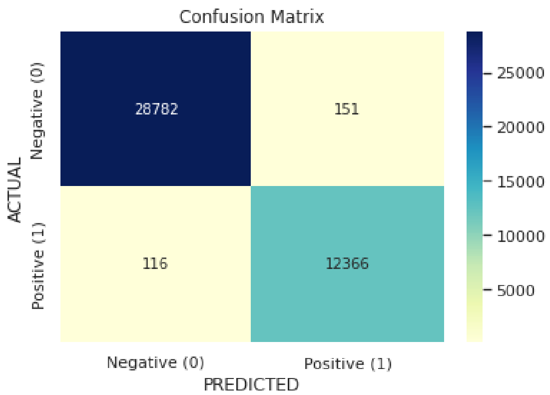


Figure 13: CF Matrix of RF

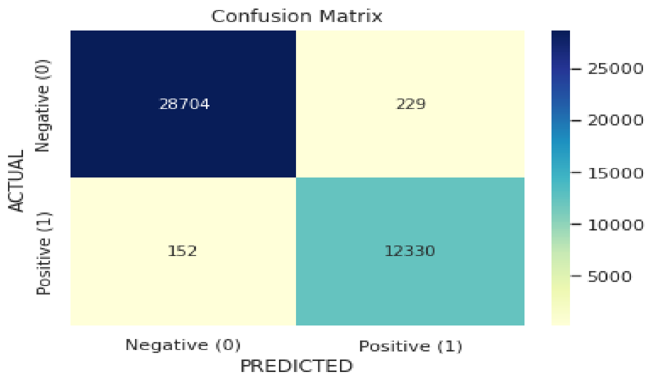


Figure 12: CF Matrix of KNN

In above figures confusion matrix of each model are displayed and based on True Positive, True Negative, False Positive and False Negative confusion matrix resulted from which accuracy, precision, recall can be calculated.

Table 2: ML Model Result Comparison

	Logistic Regression	Decision Tree	Random Forest	Naive Bayes	KNN	Best Score
Accuracy	0.937079	0.990170	0.993741	0.913508	0.991076	Random Forest
Precision	0.975584	0.982637	0.987870	0.996787	0.982756	Naive Bayes
Recall	0.810855	0.984561	0.991264	0.713356	0.987513	Random Forest
F1 Score	0.884753	0.983596	0.989564	0.831573	0.985128	Random Forest

In above table best score in accuracy, recall and F1 Score has been achived in Random Forest. Only Precision is good in Naïve Bayes. Random Forest provides best result in accuracy and F1 Score in terms of different dataset.

In figure 13 we can see that testing accuracy of Random forest model remains high than all other ML models and provides good testing results.

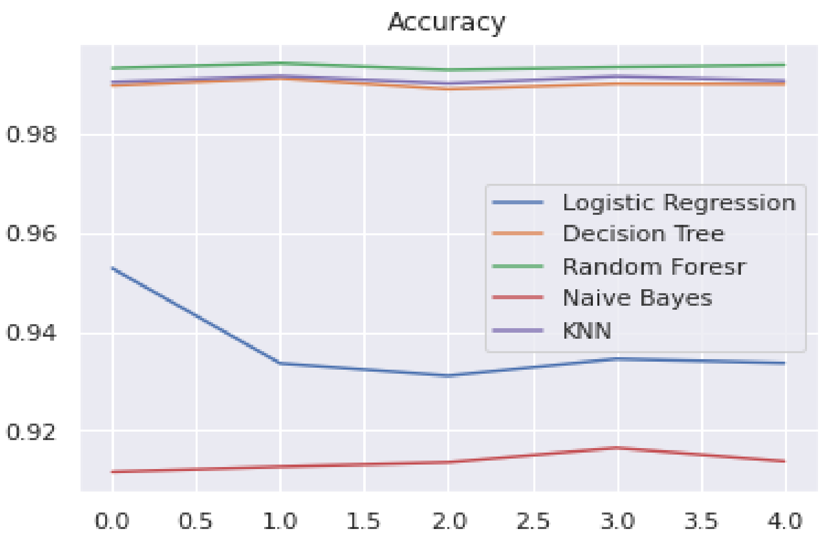


Figure 13: Accuracy of Testing of ML Model

REFERENCES

[1] "Anaconda | Individual Edition." Anaconda, https://www.anaconda.com/products/individual.

[2] https://colab.research.google.com/?utm_source=scs-index

[3] https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/

[4] https://www.tutorialspoint.com/google_colab/google_colab_tutorial.pdf

[5] https://scikit-learn.org/stable/modules/ensemble.html#forest