‘DETECTION OF PHISHING WEBSITES USING ENSEMBLE MACHINE LEARNING ALGORITHMS’

SHARMILA DEVI. M

(20PCS006)

Project Report Submitted

In partial fulfillment of the requirements for the Award of

Master of Science in Computer Science

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND

HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

‘DETECTION OF PHISHING WEBSITES USING ENSEMBLE MACHINE LEARNING ALGORITHMS’

SHARMILA DEVI. M

(20PCS006)

Project Report Submitted

In partial fulfillment of the requirements for the Award of

Master of Science in Computer Science

DEPARTMENT OF COMPUTER SCIENCE

AVINASHILINGAM INSTITUTE FOR HOME SCIENCE AND HIGHER EDUCATION FOR WOMEN

COIMBATORE – 641043

MAY – 2022

Signature of the Head of the Department **Signature of the Supervisor**

Viva-voce Examination held on _____

Signature of the Examiner

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

I would like to express my sincere thanks to **God** Almighty, for his constant love and grace that he has shown upon me, which kept me in good health, and sound mind without which my project would not have reached a successful end.

I would like to express my deep sense of reverential gratitude and sincere thanks to **Dr.S.P.Thyagarajan,** Chancellor, Avinashilingam Institute of Home Science and Higher Education for Women, Coimbatore, for the opportunity given to me for undertaking this study and for providing all the needed facilities during my study.

I owe my great deal of gratitude to **Dr.V.BharathiHarishankar,** Ph.D., FRSA Vice-Chancellor, Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for extending all resources that facilitated the conduct of the present study.

I express my gratitude to **Dr.S.Kowsalya,** Registrar, M.Sc., M.Phil., Ph.D. Avinashilingam Institute for Home Science and Higher Education for Women, Coimbatore, for providing all facilities necessary for the study.

I would express my boundless thanks to **Dr. G. Padmavathi,** M.Sc., M.Phil., Ph.D., and Dean**,** School of Physical Sciences & Computational Sciences for granting the facility required.

I wish to place on record my deep sense of gratitude to **Dr. VasanthaKalyani David,** M.Sc., M.Phil(Mathematics)., M.Phil(CS)., Ph.D., Professor and Head, Department of Computer Science for support and encouragement to complete the project.

I heartily thank my project guide **Dr.I.Elizabeth Shanthi** M.Sc., M.Phil., Ph.D., Professor Department of Computer Science, for imparting the tremendous knowledge and well-timed support for the successful completion of my project. Her guidance and constant supervision helped me in completing the project in time.

I express my honorable thanks to my project coordinator Dr**.I.Elizabeth Shanthi** M.Sc., M.Phil., Ph.D., Professor Department of Computer Science**,** for her kind advice and knowledgeable suggestions which helped me to complete my project successfully.

I have great pleasure in expressing my deep sense of gratitude to all the teaching and non-teaching staff members who stood behind the screen for the completion of the project.

Finally, I would like to thank my parents, family members, friends, and all well-wishers for their kind inspiration, blessings, and encouragement during the project time.

ABSTRACT

ABSTRACT

Phishing is one of the most common approaches of cyber-attack. Social media, spoof email, development of clone website is the main medium used by various phishers in order to steal the private information of an individual. Uniform Resource Locators (URLs) are the main source for sharing malwares, trojans and false information. Therefore, the accurate classification between phishing url is very much important. Traditional methods of detecting phishing url were mainly relying on the blacklisting and signature-based methods. Both of these methods are time consuming process and cannot work effectively on new set of URLS. The main objective of this project is to apply the Feature Selection technique such as Filter method, Wrapper method and Embedded method and selecting the best feature for model building to detect the Phishing website using ensemble learning algorithms such as Vote Classifier, Bagging Classifier, Random Forest, Ada Boost Classifier, Gradient Boosting and XG Boost. For Detecting Phishing website, the dataset has been taken from Mendeley repository. Bagging (i.e.) (Random Forest) technique is observed as outperformed. The whole project is developed with the help of python Jupyter notebook.

CONTENTS

TABLE OF CONTENTS

INTRODUCTION

CHAPTER 1

1. INTRODUCTION

Phishing is defined as spoofing a legitimate website in order to deceive consumers by collecting personal information such as usernames, passwords, account numbers, social security numbers, and so on. Phishing scams are maybe the most common type of cybercrime today. Phishing attacks can occur in a variety of areas, including the online payment sector, webmail, financial institutions, file hosting or cloud storage, and many more. Phishing has wreaked havoc on the webmail and online payment industries more than any other. Phishing can take the form of email phishing schemes and spear phishing; therefore, users should be aware of the risks and not put their trust in popular security applications naively.

1.1 ABOUT THE PROJECT

Phishing is one in all the foremost serious cyber dangers that companies face. consistent with Proofpoint's 2021 State of the Phish Report, phishing attacks affected quite eightieth of companies last year.

One of the foremost intensifying aspects of this can be that whereas the majority perceive what phishing is and the way it works, many of us area unit still caught off guard. Phishing scams are getting a lot of subtle, that has contributed to the current. they'll still wish to steal our personal data or infect our gadgets, however there are a unit currently a excess of how to try to to thus.

The UK government released the Cyber Security Breaches Survey 2021 on March 24, 2021, which included a study of 1,419 UK firms and an overview of the cyber threat landscape in 2020. The paper demonstrates how the Covid-19 epidemic has made it considerably more difficult for businesses to protect themselves from modern cyber-attacks while employees work from home.

The number of cyber-attacks continues to rise. phishing efforts were the most common sort of danger at 83 percent of organisations that recognised breaches or assaults, with 27 percent experiencing them once a week.

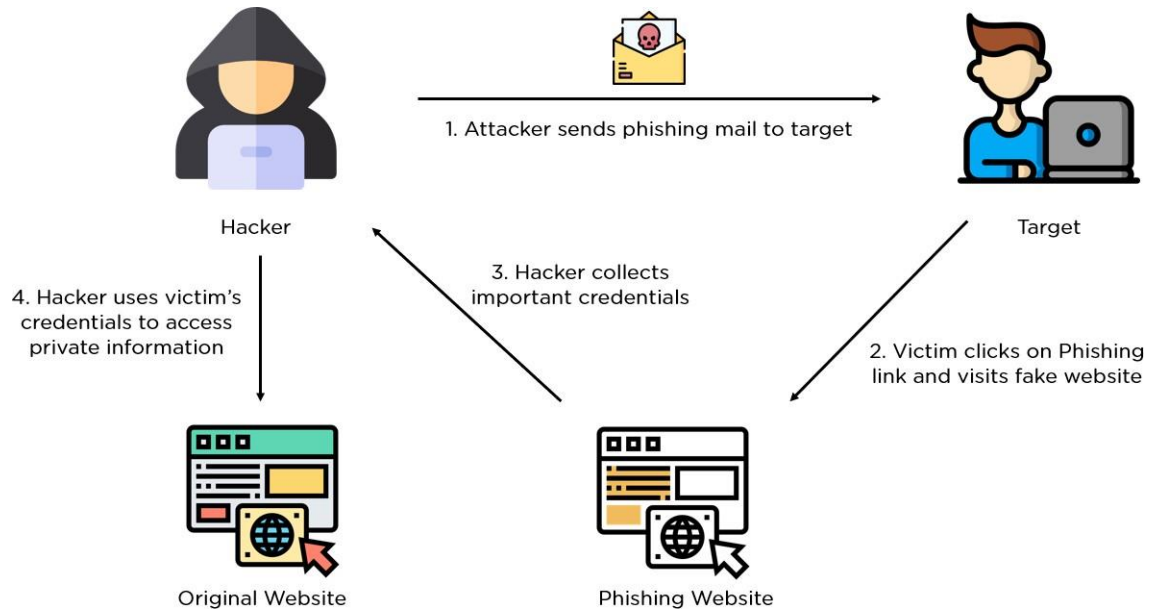


Figure 1.1

1.1.1 TYPES OF PHISHING ATTACK

There are 5 types of Phishing attack they are

- Email Phishing
- Spear Phishing
- Whaling
- Smishing and Vishing
- Angler Phishing

1.1.1.1. EMAIL PHISHING

The most widespread sort of phishing is email phishing, that has been around since the 1990s. Hackers send these emails to no matter address they'll get their hands on. the e-mail commonly warns you that your account has been hacked which you want to respond immediately by clicking on a link given. The language within the email typically contains orthography and/or grammatical issues, creating these attacks easy to note. Some phishing emails are troublesome to identify, particularly once

the choice of words and synchronic linguistics or additional fastidiously ready. If you search for suspicious choice of words within the email supply and also the link you are being directed to, you will be ready to tell if the supply is authentic.

1.1.1.2. SPEAR PHISHING

Spear phishing is a type of phishing that is targeted and customised to a certain person, group, or organisation. Regular phishing emails, on the other hand, use a broad approach, sending bulk emails to large lists of unwitting recipients. These phishing letters are usually hastily written and do not contain any personal information about the recipient.

1.1.1.3. WHALING

Whaling is a more specific sort of phishing that targets whales, a marine species that is even larger than a fish. These attacks usually target a CEO, CFO, or any CXX in a given industry or business. A whaling email can say that the company is facing legal action and that you should click on the link to learn more.

The link will take you to a page where you will be required to enter sensitive information about the business, such as the company's tax ID and bank account details.

1.1.1.4. SMISHING AND VISHING

Smishing is an assault that's dispensed over text electronic communication or short message service (SMS). Delivering a message to a movable via SMS that contains a clickable link or a comeback number may be a typical smishing technique.

Vishing serves constant aim as alternative phishing schemes. Your sensitive personal or company info continues to be being wanted by the attackers. A voice decision is employed to hold out this attack. As a result, the name encompasses a "v" instead of a "ph" in it.

1.1.1.5. ANGLER PHISHING

Social media, a comparatively new attack vector, provides thieves with a range of techniques to deceive folks. Fake URLs, cloned websites, posts, and tweets, and instant electronic communication (which is effectively constant as smishing) will all be accustomed trick people into revealing sensitive

data or downloading malware. Criminals, on the opposite hand, might develop extremely targeted attacks exploitation the information that people volitionally disclose on social media.

1.2 MOTIVATION AND JUSTIFICATION

According to Proofpoint's 2021 State of the Phish Report, more than 80% of organisations fell victim to a phishing attack,reflecting cyber criminal's continued focus on compromising individual end user.

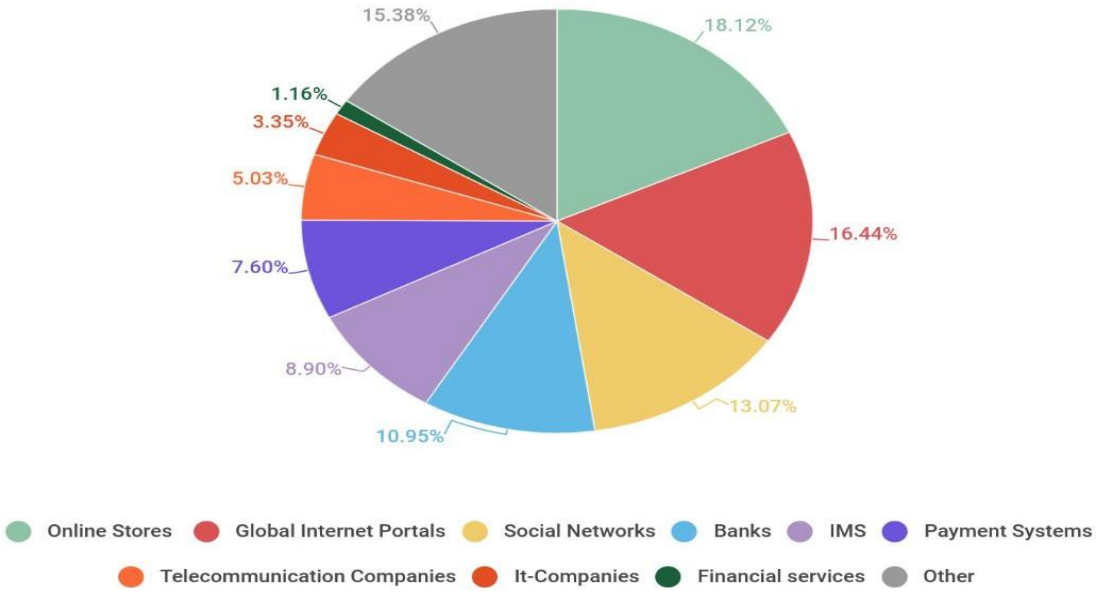


Figure 1.2

1.3 PROBLEM STATEMENT

Hackers attempt to trap the end-users through various forms such as phishing, malware, ransomware, web trojan, and so on. Among all these attacks, phishing reports to be the most deceiving attack. So, it is needed to identify the phishing urls to proceed the transactions in the safer way.

1.4 OBJECTIVE

To design an efficient and adaptive phishing detection algorithm that has the ability to adapt to newer data or phishing attach vendors discovered in the wild.

SYSTEM CONFIGURATION

CHAPTER 2

2. SYSTEM CONFIGURATION

2.1. HARDWARE SPECIFICATION

RAM	: 8.00 GB
PROCESSOR	: 11th Gen Intel (R) Core™ i5 -1135G7
PROCESSOR SPEED	: 2.40 GHz 2.42 GHz
SYSTEM TYPE	: 64-bit Operating System, x64 based processor

2.2 SOFTWARE SPECIFICATION

ANACONDA PLATFORM

Anaconda Distribution is a Python/R data science distribution that includes a package and environment manager as well as over 7,500 open-source tools. You can use Anaconda on Windows, Mac OS X, or Linux because it is platform agnostic. Anaconda is a free, user-friendly distribution with free community support.

Anaconda Embedded allows you to give customers a smooth Python interface or use Anaconda behind the scenes to support your product or service offering. All Embedded partners have access to Anaconda's premium repository, experts, and developers, as well as additional benefits including SLAs, co-marketing, and bespoke development opportunities.

The most recent version of our Anaconda repository is Anaconda Server. With support for all major operating systems, the repository serves as a common conda, PyPI, and CRAN packaging resource for desktop users, development clusters, CI/CD systems, and production containers.

ANACONDA NAVIGATOR

Anaconda Navigator is a graphical user interface (GUI) for desktops that comes with the anaconda distribution that permits users to run programmes and manage conda packages, environments, and channels while not having to use command-line commands. Navigator could search for packages on anaconda Cloud or in a very native anaconda Repository, then install, run, and update them in a very given setting. Windows, Mac OS X, and UNIX operating system square measure all supported.

Navigator comes with the following applications pre-installed:

• JupyterLab

• Jupyter Notebook

• QtConsole[19]

• Spyder

• Glue

• Orange

• RStudio

• Visual Studio Code

JUPYTER NOTEBOOK

Jupyter is a free, open-source, interactive web platform that allows academics to combine software code, computational output, explanatory prose, and multimedia materials in one document. Computational notebooks are essentially lab notebooks for scientific computing. Rather of pasting DNA gels alongside lab instructions, researchers incorporate code, data, and text to illustrate their computational methodologies.

Jupyter Notebook is an open-source web application that allows a user, scientist, scholar, or analyst to create and share a Notebook that comprises live programmes, documentation, graphs, plots, and visualisations.Jupyter notebook is a fantastic tool for data analysis and visualisation. Data scientists utilise Jupyter notebook to conduct their daily data analysis duties. Surprisingly, the notebook is the first tool that analysts are taught to in a data science course. The Jupyter Notebook file browser interface is the most common way to open a Jupyter notebook (.ipynb) file.

When you open one in a text editor, it appears as a JSON file rather than interactive code blocks. To view a notebook using the Jupyter interface, first launch Jupyter Notebook (which will open in a browser window), and then open the file from within Jupyter Notebook.Jupyter Notebook is a programme that allows you to write code in Python. Unfortunately, making Jupyter Notebook the default software application for double-clicking.ipynb files is not possible. A notebook is made up of cells, which are boxes that carry human-readable code or text. Every cell has a type, which can be selected from the drop-down menu.

Jupyter Notebook is a great tool for data scientists, especially when it comes to education. This tool can be used by analysts to write tiny code snippets and in scenarios when code

production isn't required. As a result, the Jupyter notebook comes in handy for this project.

LIBRARIES USED

- import pandas as pd

Pandas is typically installed using the pd alias. Instead of pandas, the Pandas package is nowknown as pd

- import matplotlib.pyplot as plt
- import numpy as np
- import seaborn as sns

Matplotlib is helps in visualizing the data in the graphical format, with numpy. Seaborn is a Python data visualization library built on top of Matplotlib.

- from sklearn.model_selection import train_test_split

Sklearn model selection has a method called train test split that splits data arrays into twosubsets: training data and testing data. It choses randomly the sets.

- from sklearn.metrics import accuracy_score

The accuracy classification score is a numerical value that indicates how accurate the categorization is. This function computes subset accuracy in multilabel classification: the labelset predicted for a sample must exactly match the label set in y true.

- from sklearn.metrics import classification_report

In machine learning, a classification report is a performance evaluation indicator. It's used to demonstrate trained classification model's precision, recall, F1 Score, and support.

- from sklearn import metrics

To quantify classification performance, the sklearn. metrics module includes many loss, score,and utility methods.

- from sklearn.feature_selection import VarianceThreshold

VarianceThreshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples.

- import mlxtend
- import statsmodels.api as sm

Mlxtend (machine learning extensions) is a Python library of useful tools for the day-to-day data science tasks. Statsmodels supports specifying models using R-style formulas and pandas DataFrames.

- from mlxtend.feature_selection import SequentialFeatureSelector as SFS

The Sequential Feature Selector adds (forward selection) or removes (backward selection) features to form a feature subset in a greedy fashion. At each stage, this estimator chooses the best feature to add or remove based on the cross-validation score of an estimator.

- from sklearn.feature_selection import RFECV

Recursive Feature Elimination, Cross-Validated (RFECV) feature selection. Selects the best subset of features for the supplied estimator by removing 0 to N features (where N is the number of features) using recursive feature elimination, then selecting the best subset based on the cross-validation score of the model.

- from sklearn.linear_model import Lasso, LogisticRegression

The Lasso is a linear model that estimates sparse coefficients. Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier.

- from sklearn.preprocessing import StandardScaler

Remove the mean and scale to unit variance to standardise characteristics. By computing the necessary statistics on the samples in the training set, each feature is individually centred and scaled.

- from sklearn import ensemble

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

- from sklearn.ensemble import VotingClassifier

A voting classifier is a classification method that employs multiple classifiers to make predictions.

- from sklearn.ensemble import BaggingClassifier

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.

- from sklearn.ensemble import RandomForestClassifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

- from sklearn.ensemble import AdaBoostClassifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

- from sklearn.ensemble import GradientBoostingClassifier

Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT) is a generalization of boosting to arbitrary differentiable loss functions. GBDT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems in a variety of areas including Web search ranking and ecology.

LITERATURE REVIEW

CHAPTER 3

3. LITERATURE REVIEW

[1] Suhas R. Sharma, S. R., Parthasarathy, et.al (2020), used Phishing datasets from UCI and Mendeley, where comparison is done between several Machine Learning Algorithms (that can be used for binary classification) and Feature Selection Techniques. The purpose of their experiment is to achieve similar/comparable accuracies while reducing the number of characteristics significantly. To evaluate system improvements, they employed the F1 Score and Time of Execution as metrics. The information is subsequently presented in the form of tables and graphs to demonstrate the same. Hence, Random Forest worked well when compared to other algorithms used.

[2] Junaid Rashid, Toqeer Mahmood, et.al (2020), propose machine learning-based phishing detection. Additionally, machine learning algorithms build classifiers that recognise real phishing websites. SVM was employed in the proposed technique, which had a 95.66 percent accuracy and an extremely low false-positive rate. The proposed technique can detect new temporary phishing sites and lessen phishing attack impact. The suggested machine learning-based solution outperforms existing phishing detection technologies in terms of effectiveness.

[3] Amani Alswailem, Bashayr Alabdullah, et.al (2016), presented an intelligent technique for detecting phishing websites in this paper. The technology works as an addon to an internet browser that warns the user when it finds a phishing website. The system is based on supervised learning, which is a type of machine learning. They chose the Random Forest approach because of its high classification accuracy. Their goal is to develop a higher-performing classifier by examining the characteristics of phishing websites and selecting the best mix of those characteristics to train the classifier. As a result, they arrive at a 98.8% accuracy rate and a 26-feature combination.

[4] Mohammed Hazim Alkawaz, Stephanie Joanne Steven, et.al (2020), presented a phishing detection method to detect blacklisted URLs, also known as phishing websites, so that users can be informed while browsing or accessing a specific website. As a result, it may be used for identification and authentication, and it can also be used to protect people from being duped.

[5] Sharad Rajendra Parmar (2020) advocated for the employment of ensemble learning techniques. Bagging, AdaBoost, Random Forest, and Gradient boosting methods were employed. The findings were then compared to non-ensemble learning methods including decision trees, K-nearest neighbour, and logistic regression. Using the random forest classifier, an ensemble learning method, they attained the best accuracy of 96.15 percent after training the models. The lowest performing model is K-nearest neighbour classifier, which provides the lowest accuracy of 54.31%

[6] Mehmet Korkmaz, Ozgur Koray Sahingoz, et.al (2020), proposed a machine learning-based phishing detection system that analyses URLs using eight distinct algorithms and compares the

findings to previous works using three different datasets. The experimental results show that the proposed models operate exceptionally well, with a high success rate.

[7] Mohammad Nazmul Alam, Dhiman Sarma, et.al (2020), Using machine learning (ML) algorithms such as random forest (RF) and decision tree, the suggested research effort has constructed a model to detect phishing attempts (DT). The suggested model uses feature selection procedures like principal component analysis to examine the dataset's properties (PCA). Finally, using the random forest technique, a maximum accuracy of 97 percent was reached.

[8] Mohammad Alshira. H, Mohammad Al-Fawa reh, (2020), They were used to classify the state of URLs using lexical features. Various machine learning and statistical detection algorithms used the dataset as input. Based on linguistic properties, the models were used to predict Phishing URLs. The result suggests a high level of accuracy. In comparison to the other detection models, the Random Forest (RF)model produced the highest accuracy (98%).

Phishing means

Phishing is a type of social engineering wherever an attacker sends a fraudulent (e.g., spoofed, fake, or otherwise deceptive) message designed to trick someone into revealing sensitive data to the offender or to deploy malicious code on the victim's infrastructure like ransomware. Phishing attacks became increasingly subtle and infrequently transparently mirror the positioning being targeted, permitting the offender to look at everything whereas the victim is navigating the positioning, and transversal any extra security boundaries with the victim. As of 2020, phishing is far and away the foremost common attack performed by cybercriminals, with the FBI's web Crime grievance Centre recording over doubly as several incidents of phishing than the other style of pc crime.

ENSEMBLE LEARNING

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.

There are two techniques of ensemble learning. They are

- Simple Technique
- Advanced Technique

SIMPLE TECHNIQUE

A few simple but powerful techniques, namely:

- Max Voting
- Averaging
- Weighted Averaging

MAX VOTING

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.

AVERAGING

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

WEIGHTED AVERAGING

This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction.

ADVANCED TECHNIQUE

In Advanced Technique there are 4 types.They are

- Stacking
- Blending
- Bagging
- Boosting

STACKING

Stacking is an ensemble learning technique that uses predictions from multiple models (for example decision tree, knn or svm) to build a new model. This model is used for making predictions on the test set.

BLENDING

Blending follows the same approach as stacking but uses only a holdout (validation) set from the train set to make predictions. In other words, unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model which is run on the test set.

BAGGING

Bagging (or Bootstrap Aggregating) technique uses these subsets (bags) to get a fair idea of the distribution (complete set). The size of subsets created for bagging may be less than the original set.

Bagging algorithms:

- Bagging meta-estimator
- Random forest

BAGGING META-ESTIMATOR

Bagging meta-estimator is an ensemble algorithm that can be used for both classification (Bagging Classifier) and regression (Bagging Regressor) problems. It follows the typical bagging technique to make predictions.

RANDOM FOREST

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

BOOSTING

Boosting is a sequential process, where each subsequent model attempts to correct the errors of the previous model. The succeeding models are dependent on the previous model.

Boosting Algorithms

- Ada Boost
- Gradient Boosting
- XGBoost

ADA BOOST

Adaptive boosting or AdaBoost is one of the simplest boosting algorithms. Usually, decision trees are used for modelling. Multiple sequential models are created, each correcting the errors from the last model. AdaBoost assigns weights to the observations which are incorrectly predicted and the subsequent model works to predict these values correctly.

GRADIENT BOOSTING

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Regression trees used as a base learner, each subsequent tree in series is built on the errors calculated by the previous tree.

XGBOOST

XGBoost (extreme Gradient Boosting) is an advanced implementation of the gradient boosting algorithm. XGBoost has proved to be a highly effective ML algorithm, extensively used in machine learning competitions and hackathons. XGBoost has high predictive power and is almost 10 times faster than the other gradient boosting techniques. It also includes a variety of regularization which reduces overfitting and improves overall performance. Hence it is also known as ‘regularized boosting ‘technique.

3.1 BACKGROUND STUDY

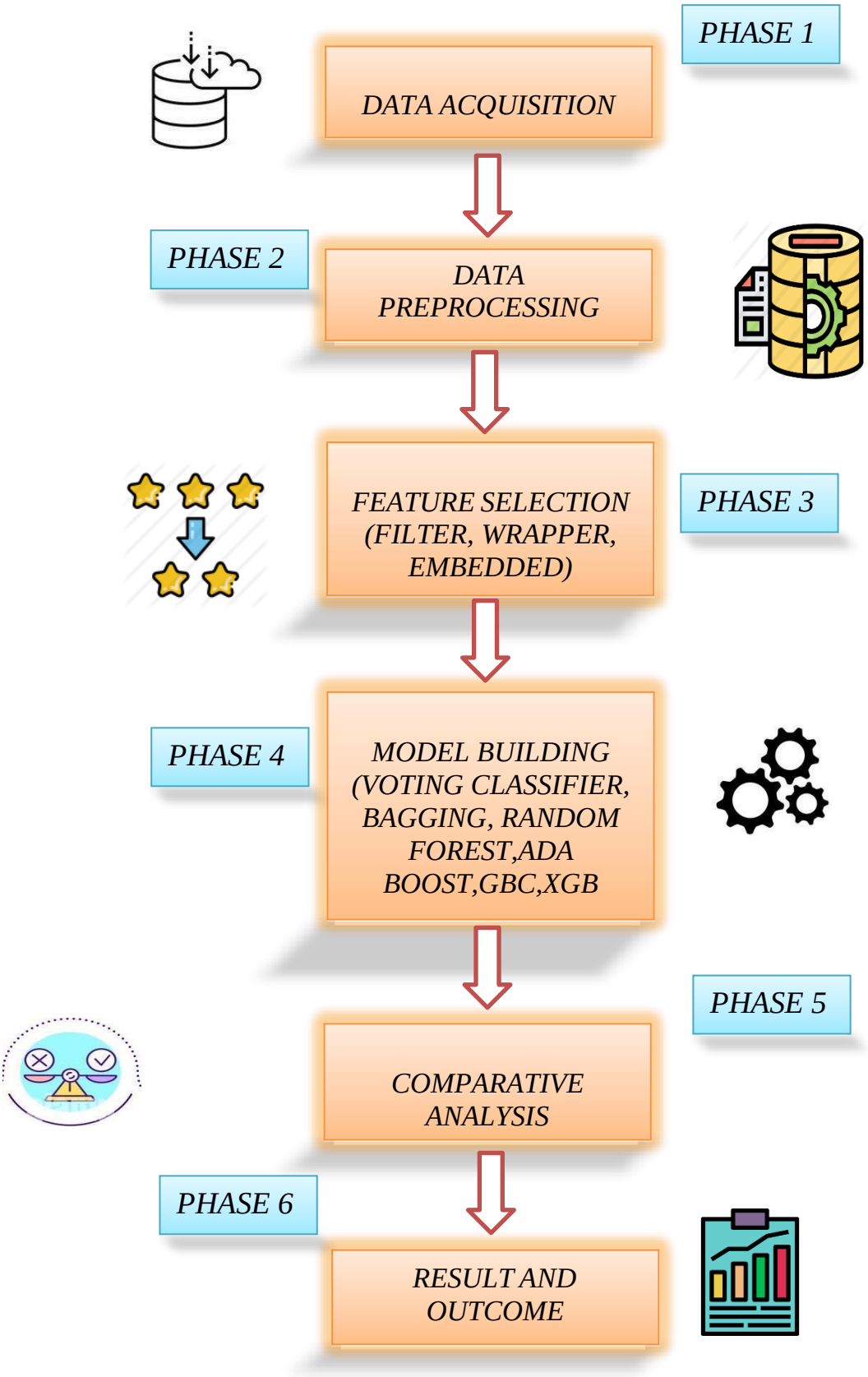
Paper Name	Author Name	Algorithms and methods used	Result
A Feature Selection Comparative Study for Web Phishing Datasets	Suhas R. Sharma, Rahul Parthasarathy, Prasad B. Honnavalli	SVM, K Nearest Neighbours, Decision Tree, Naïve Bayes, Multi-layer perceptron, Logistic Regression, Random Forest Classifier, Stochastic Gradient Descent.	Random Forest Classifier gives the best accuracy 96%
Phishing Detection Using Machine Learning Technique(IEEE)	Junaid Rashid, Toqeer Mahmood, Muhammad Wasif Nisar, Tahira Nazir	Principle Component Analysis, Support vector Machine	Support vector machine classifier has the best performance of accurately 95.66%
Detecting Phishing Websites Using Machine Learning	Amani Alswailem, Bashayr Alabdullah, Norah Alrumayh, Dr.Aram Alsedrani	Decision Tree, ANN, Naïve Bayes, SVM, and KNN	ANN gives the best accuracy 96.01%
Detecting Phishing Website Using Machine Learning	Mohammed Hazim Alkawaz, Stephanie Joanne Steven, Asif Iqbal Hajamydeen	No algorithms used here. The Software was implemented	The Software was implemented successfully
Detection of Phishing URL using Ensemble Learning Techniques	Sharad Rajendra Parmar	Bagging classifier, AdaBoost, Gradient Boosting, Decision Tree, K nearest Neighbours, Logistic Regression	Random Forest Classifier gives the best accuracy 96.15%

Detection of Phishing Websites by Using Machine Learning-Based URL Analysis	Mehmet Korkmaz, Ozgur Koray Sahingoz, Banu Diri	XGBoost, Random forest, Logistic regression, KNN,SVM ,Decision tree, ANN ,Naïve Bayes	Random Forest Classifier gives the best accuracy 94.59%
Phishing Attacks Detection using Machine Learning Approach	Mohammad Nazmul Alam, Dhiman Sarma, Farzana Firoz Lima, Ishita Saha, Rubaiath-E-Ulfath, Sohrab Hossain	Principle Component Analysis, Random Forest Classifier	Random Forest Classifier gives the best accuracy 97%
Detecting Phishing URLs using Machine Learning & Lexical Feature-based Analysis	Mohammad Alshira, Mohammad Al-Fawa	(RF, DT, GNB, KNN: k-nearest neighbour, Logistic regression, SVC, QDA, Perceptron, SMOTE.	Random forest (RF)model has produced the best accuracy (98%)

METHODOLOGY AND IMPLEMENTATION

CHAPTER 4

4. METHODOLOGY



IMPLEMENTATION

4.1 DATA ACQUISITION (PHASE 1)

Data acquisition is the process of obtaining data from relevant sources before it is saved, cleansed, pre-processed, and used in subsequent operations. It is the process of gathering important business data, transforming it into the appropriate business format, and loading it into the appropriate system.

"The act of sampling data that measures real-world physical attributes and transforming the resulting samples into digital numeric values that a computer can manage is known as data acquisition."

DATASET LINK

https://data.mendeley.com/datasets/h3cgnj8hft/1

4.1.1 DATASET USED

In this project the phishing website dataset has been collected from Mendeley Repositary.

Data Set Information

The Phishing website containing 48 fields with three classes attributes. This dataset also consists of 1000 instances. The preview is shown in Fig 4.1.1

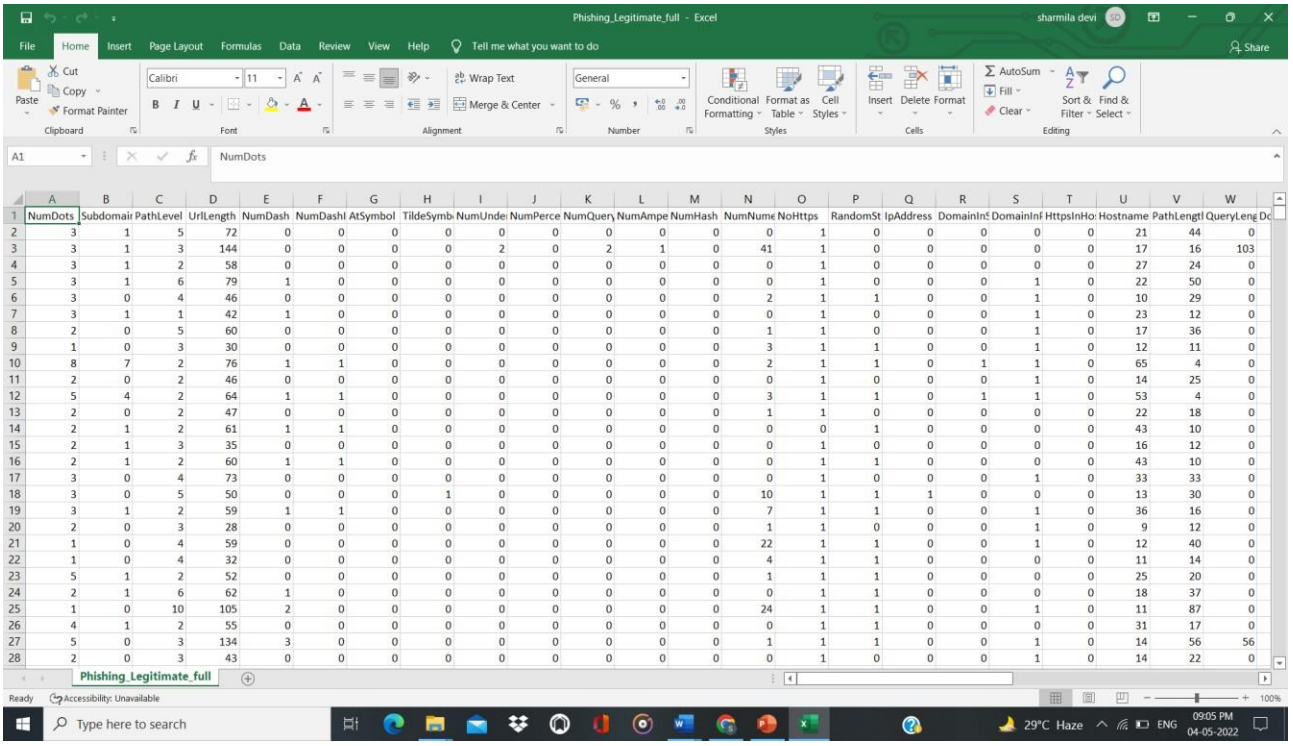


Fig. 4.1.1 DATASET PREVIEW

DATASET DESCRIPTION

- The 48 fields are classified into 3 types. They are
    - **Lexical Based Features – 27 Features**
    - **Host Based Features – 15 Features**
    - **Correlation Based Features – 6 Features**
- Figure 4.1.2 gives the classification of fields

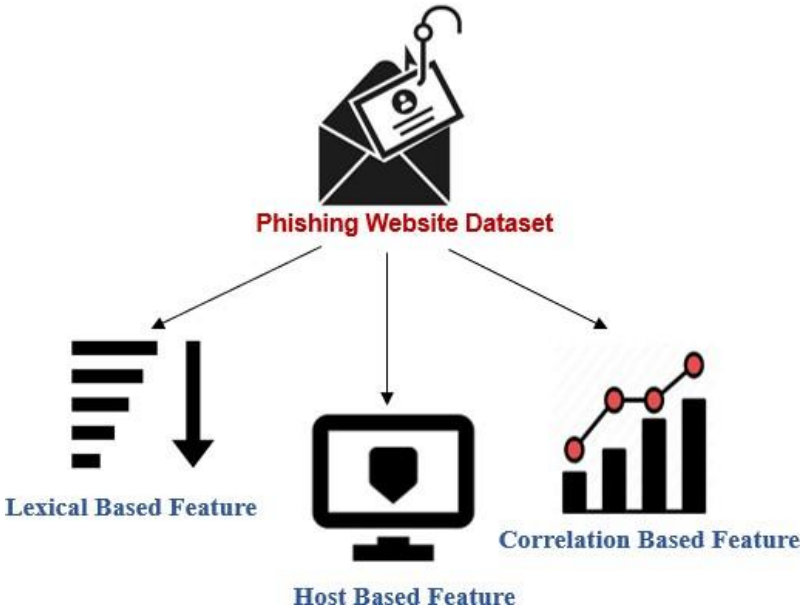


Fig. 4.1.2 Classification of fields

HOST BASED FEATURES

These are characteristics of the host-name properties of the URL. These provide information about the host of the webpage, for example, country of registration, domain name properties, open ports, named servers, connection speed, time to live from registration, etc.

CORRELATION BASED FEATURES

Ranks attributes accordingto a heuristic evaluation function based on correlations

FEATURES DESCRIPTION

Table 1 shows the list of features under lexical, host and correlation-based features.

Lexical based feature	Host based feature	Correlation based feature
NumDots	PctExtResourceUrls	UrlLengthRT
SubdomainLevel	ExtFavicon	PctExtResourceUrlsRT
PathLevel	InsecureForms	AbnormalExtFormActionR
UrlLength	RelativeFormAction	ExtMetaScriptLinkRT
NumDash	ExtFormAction	SubdomainLevelRT
NumDashInHostname	AbnormalFormAction	PctExtNullSelfRedirectHyp
AtSymbol	PctNullSelfRedirectHyperlinks	erlinksRT
TildeSymbol	FrequentDomainName	
NumUnderscore	Mismatch	
NumPercent	FakeLinkInStatusBar	
NumQueryComponents	RightClickDisabled	
NumAmpersand	PopUpWindow	
NumHash	SubmitInfoToEmail	
NumNumericChars	IframeOrFrame	
NoHttps	MissingTitle	
RandomString	ImagesOnlyInForm	
IpAddress		
DomainInSubdomains		
DomainInPaths		
HttpsInHostname		
HostnameLength		
PathLength		
QueryLength		
DoubleSlashInPath		
NumSensitiveWords		
EmbeddedBrandName		
PctExtHyperlinks		

Table 1

Feature Description

4.2 DATA PRE-PROCESSING (PHASE 2)

Data preparation is an important phase in Machine Learning since it improves data quality and makes it easier to extract useful insights from the data. Data pre-processing in Machine Learning refers to the act of organising and handling raw data in order to prepare it for the creation and training of Machine Learning models. The first stage in constructing a Machine Learning model is data preparation, which signals the start of the process. Data from the real world is frequently incomplete, inconsistent, erroneous (owing to errors or outliers), and lacking in precise attribute values and trends. This is where data preparation comes in; it cleans, prepares, and summaries raw data so that Machine Learning models can work with it.

DATA PRE-PROCESSING IN MACHINE LEARNING

There are four significant steps in data preprocessing.

1) ***Acquire the dataset***

   - First Step is to acquire the dataset. This dataset will be made up of data acquired from a variety of sources, which will be merged in a logical manner to make a dataset.
   - For Detecting the phishing websites, the dataset has been collected from Mendeley repository.

2) ***Import all the crucial libraries***

   - The three core Python libraries used for this data pre-processing in Machine Learning are: NumPy, Pandas, Matplotlib.
   - Importing the dataset into python jupyter notebook using Panda's libraries

3) ***Handling Missing Values***

   - In data preprocessing it is major important to check and identify the Missing values. If we fail to do this, faulty conclusion an inference from the data.

4) Splitting the dataset

- Every dataset for Machine Learning model must be split into two separate sets – training set and test set.
- The Dataset are splitted into 70:30 ratio or 80:20 ratio.

4.3 FEATURE SELECTION (PHASE 3)

- It is a method of determining which required properties have the most impact on the output variable.
- It means that only those features (independent variables) that are closely related to the output variable should be chosen.
- It is the most crucial step in the creation of a machine learning model.

TYPES OF FEATURE SELECTION

Feature Selection technique is classified into 3 categories. They are

- Filter Method
- Wrapper Method and
- Embedded Method.

4.3.1 FILTER BASED FEATURE SELECTION

Filter methods are generally used as a preprocessing step. The selection of features is independent of any machine learning algorithms. Instead, features are selected on the basis of their scores in various statistical tests for their correlation with the outcome variable. Filter based feature selection are classified into,

- Basic Filter Methods
- Correlation Filter Methods
- Statistical & Ranking Filter Methods

In this project in Basic Filter methods (Quasi Constant) and Correlation

(Pearson Correlation) Filter methods are used for detecting Phishing websites. The working of Filter based Feature Selection is shown in Fig 4.3.1

Figure 4.3.1

Working of Filter based Feature Selection

Basic Filter Methods

These basic and intuitive methods help to remove

- Constant Feature
- Quasi Constant Feature
- Duplicated Features

In this project Quasi-Constant feature is implemented.

4.3.1.1. QUASI CONSTANT FEATURE

- Quasi – Constant features have the same values for a very large subset of the outputs. Such features are not very useful for making predictions. There is no rule as to what should be the threshold for the variance of quasi-constant features.
- Quasi – Constant feature are also mainly depending on variance threshold then fit the model with training set of the data.
- Variance Threshold is a simple baseline approach to feature selection. It removes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e., features that have the same value in all samples.

Pseudocode for Quasi - Constant based Feature Selection

STEP 1: Start the Process.

STEP 2: Import all the necessary libraries and importing the dataset.

STEP 3: Splitting the dataset as train and test model and dropping the target value from thedataset.

STEP 4: Create variance threshold model which will search for the features having 99% of samevalue in all samples.

STEP 5: Using get_support() return True and False value for each feature.
True: Not a quasi-constant feature
False: Quasi constant feature.

STEP 6*:* Using X_train=varModel.transform(X_train) and X_test=varModel.transform(X_test)dropping the constant values in the dataset.

STEP 7*:* Stop the Process

4.3.1.2. PEARSON CORRELATION BASED FEATURE SELECTION

- It is used as a measure for quantifying linear dependence between two continuous variables X and Y. Its value varies from -1 to +1.
- Pearson's Correlation method is used for finding the association between the continuous features and the class feature.

Pseudocode for Pearson Correlation based Feature Selection

STEP 1: Start the Process.

STEP 2: Import all the necessary libraries and importing the dataset.

STEP 3: Splitting the dataset as train and test model and dropping the target value from the dataset.

STEP 4: X_train.corr() is used to find the pairwise correlation of all columns in the dataframe.

STEP 5: def correlation (dataset, threshold) function will select highly correlated features and will remove the first feature that is correlated with anything another feature

STEP 6*:* Dropping the correlated features in the dataset.

STEP 7*:* Stop the Process

4.3.2 WRAPPER BASED FEATURE SELECTION

In wrapper methods, the feature selection process is based on a specific machine learning algorithm that we are trying to fit on a given dataset. The working of Wrapper based Feature Selection is shown in Fig 4.3.2

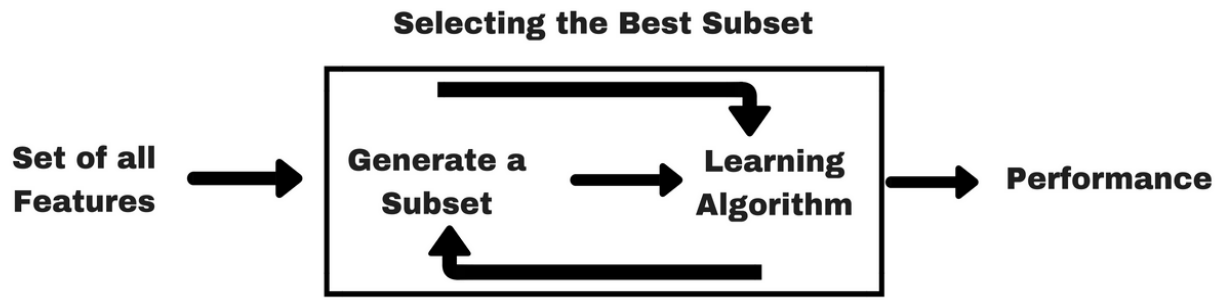


Figure 4.3.2

Working of Wrapper Based Feature Selection

- In this project in wrapper-based feature selection (Stepwise Feature Selection Method) and (Recursive Feature Elimination Methods) are used to detect Phishing Websites.

Methods involved in Wrapper based Feature Selection

Wrapper based Feature Selection are Classified into three categories, They are

- Forward Selection Method
- Backward Selection Method
- Stepwise Feature Selection

4.3.2.1. STEPWISE FEATURE SELECTION

- It's similar to forward selection, but instead of adding a new feature, it checks the significance of previously added features, and if any of the previously picked features are found to be minor, it simply removes that feature by backward elimination. As a result, it's a hybrid of forward and backward elimination.

Pseudocode for Stepwise Feature Selection

STEP 1: Import all the necessary libraries and importing the dataset.

STEP 2: Splitting the dataset as train and test model and dropping the target value from the dataset.

STEP 3: Choose a *significance level* to enter and exit the model (e.g. SL_in = 0.05 and SL_out = 0.05 with 95% confidence).

STEP 4: Perform the next step of *forward selection* (newly added feature must have *p-value* <SL_in to enter).

STEP 5: Perform all steps of *backward elimination* (any previously added feature with *p-value*>SL_out is ready to exit the model).

STEP 6: Repeat steps 2 and 3 until we get a final *optimal* set of features.

4.3.2.2. RECURSIVE FEATURE ELIMINATION

- Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached.
- Features are ranked by the model's coef and feature important attributes, and by recursively eliminating a small number of features per loop, RFE attempts to eliminate dependencies and collinearity that may exist in the model.
- Recursive feature elimination with cross-validation to determine an optimal number of features for a model. Visualizes the feature subsets with respect to the cross-validation score.

Pseudocode for Recursive Feature Elimination

STEP 1: Start the Process.

STEP 2: Import all the necessary libraries and importing the dataset.

STEP 3: Splitting the dataset as train and test model and dropping the target value from the dataset.

STEP 4: Create Linear Model
dtree = DecisionTreeRegressor()

STEP 5: Create recursive feature eliminator that scores feature by mean squared errors.

STEP 6: Stop the Process.

4.3.3. EMBEDDED BASED FEATURE SELECTION

- Embedded methods combine the qualities of filter and wrapper methods. It's implemented by algorithms that have their own built-in feature selection methods.

- Some of the most popular examples of these methods are LASSO and RIDGE regression which have inbuilt penalization functions to reduce overfitting.
    - Lasso regression performs L1 regularization which adds penalty equivalent to absolute value of the magnitude of coefficients.
    - Ridge regression performs L2 regularization which adds penalty equivalent to square of the magnitude of coefficients.

- The working of Embedded based Feature Selection is shown in Fig 4.3.3

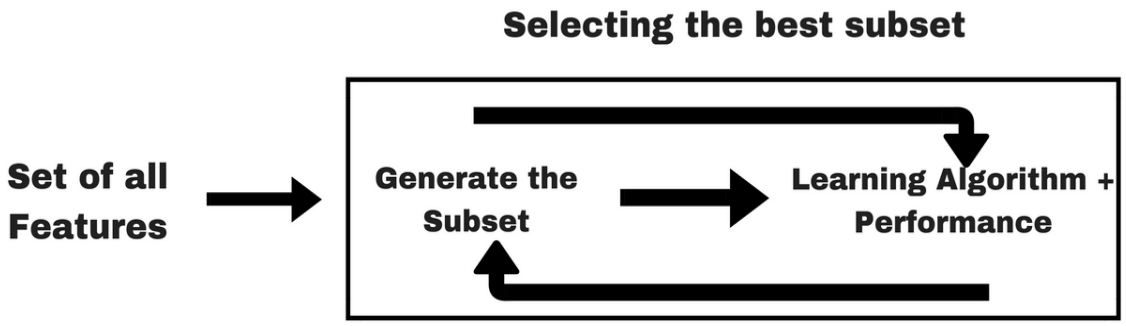


Figure 4.3.3

Working of Embedded Based Feature Selection

4.3.3.1 LASSO REGRESSION

- LASSO stands for *Least Absolute Shrinkage and Selection Operator*.
- Performs L1 regularization, i.e., adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = LS Obj + α * (sum of absolute value of coefficients)

LASSO REGRESSION WITH LOGISTIC REGRESSION

- The penalty is given to the coefficients that multiply each of the predictors in Logistic Regression. Lasso or L1 is one of the regularization types that has the ability to decrease part of the coefficients to zero. As a result, the model's feature can be eliminated.
- Both model fitting and feature selection are done using logistic regression. The Logistic Regression uses the Lasso (L1) penalty and the selectFromModel object from sklearn to choose which coefficients are non-zero in the features.
- The output labels are organized by index. True is true for characteristics that Lasso judged were relevant (non-zero features), while False is true for features whose weights were reduced to zero and are not important to Lasso.

Pseudocode for Lasso Regression

STEP 1: Start the Process

STEP 2: Import all the necessary libraries and importing the dataset.

STEP 3: Splitting the dataset as train and test model and dropping the target value from the dataset.

STEP 4: Using Logistic Regression model, in order to select the Lasso (l1) penalty and SelectFromModel class from sklearn, which will select the features which coefficients are non-zero.

STEP 5: Visualize the index of the features that were selected.

STEP 6: Stop the Process

4.3.3.2. TREE BASED METHOD

- The most typical embedded technique is tree-based methods.
- Tree based methods select a feature in each recursive step of the tree growth processand divide the sample set into smaller subsets.

TREE BASED METHOD WITH RANDOM FOREST CLASSIFIER

- "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."

Pseudocode for Tree Based Method

STEP 1: Start the Process

STEP 2: Import all the necessary libraries and importing the dataset.

STEP 3: Splitting the dataset as train and test model and dropping the target value from the dataset.

STEP 4: Create Random Forest Classifier model

STEP 5: Use RFE to eliminate the less importance features

STEP 6: Visualize the index of the features that were selected.

STEP 7: Stop the Process

4.4 MODULE BUILDING (PHASE 4)

4.4.1. ENSEMBLE LEARNING

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.

There are two techniques of ensemble learning. They are

- Simple Technique
- Advanced Technique

4.4.1.1. SIMPLE TECHNIQUE

A few simple but powerful techniques, namely:

- Max Voting
- Averaging
- Weighted Averaging

4.4.1.1.1. VOTING CLASSIFIER

A Voting Classifier is a machine learning model that learns from an ensemble of models and predicts an output (class) based on the highest probability of the output being the chosen class.

It simply adds up the results of each classifier supplied into Voting Classifier and predicts the output class based on the most votes. Rather than constructing separate specialized models and determining their performance, we create a single model that trains on various models and predicts output based on the cumulative majority of votes for each output class.

Voting Classifier supports two types of voting's.

- Hard Voting
- Soft Voting

PROCEDURE

STEP 1: Apply 3 Classifiers (K Nearest Classifier, Decision Tree Classifier and Logistic Regression Classifier) on the training data.

STEP 2: Compare the performance of the 3 Classifier.

STEP 3: Performing Majority Voting for every observation.

STEP 4: Compare the performance of the Majority Voting with the K Nearest Classifier, Decision Tree Classifier and Logistic Regression Classifier.

STEP 5: Stop the Process

4.4.1.2. ADVANCED TECHNIQUE

In Advanced Technique there are 4 categories. They are

- Stacking
- Blending
- Bagging
- Boosting

4.4.1.2.1. BAGGING

The ensemble learning method of tagging, also known as bootstrap aggregation, is often used to reduce variance within a noisy dataset.

Bagging is the process of selecting a random sample of data from a training set with replacement—that is, the individual data points might be chosen many times.
Bagging, which frequently examines homogeneous weak learners, learns them in parallel and then combines them using some form of deterministic averaging technique. The working of Bagging is shown in Fig 4.4.1

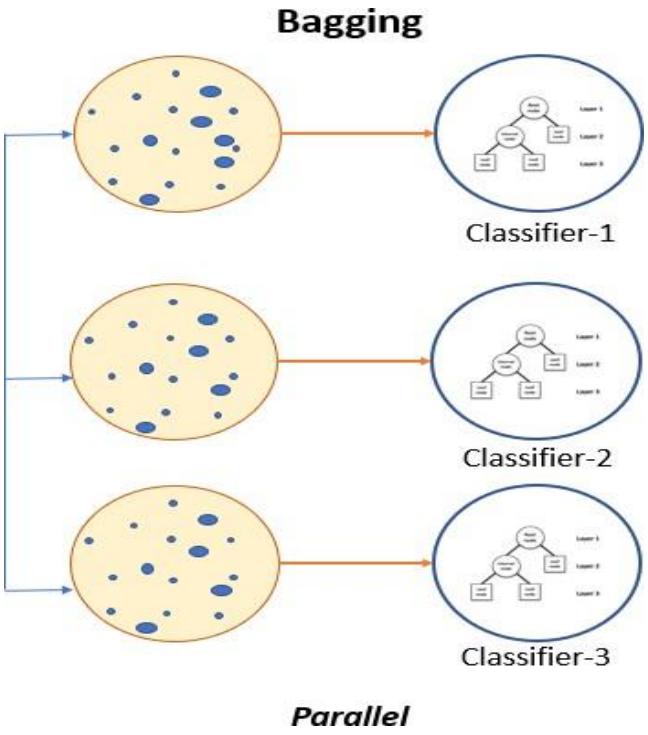


Figure 4.4.1

Working of Bagging Classifier

The size of subsets created for bagging may be less than the original set.

Bagging algorithms:

- Bagging meta-estimator
- Random forest

4.4.1.2.1.1. BAGGING META-ESTIMATOR

Bagging meta-estimator is an ensemble algorithm that can be used for both classification (Bagging Classifier) and regression (Bagging Regressor) problems.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction.Bagging reduces overfitting (variance) by averaging or voting

PROCEDURE

STEP 1: Assume you have N observations and M characteristics. With replacement, a random sample from observation is chosen (Bootstrapping).

STEP 2: To develop a model with a sample of observations and a subset of features, a subset of features is chosen.

STEP 3: The feature from the subset that delivers the best split on the training data is chosen.

STEP 4: This process is repeated to construct many models, each of which is trained in parallel.

STEP 5: A prediction is made based on the sum of all the models' predictions.

4.4.1.2.1.2. RANDOM FOREST

Random Forest is another ensemble machine learning algorithm that follows the bagging technique. It is an extension of the bagging estimator algorithm. The base estimators in random forest are decision trees. Unlike bagging meta estimator, random forest randomly selects a set of features which are used to decide the best split at each node of the decision tree.

PROCEDURE

STEP 1: Select N records at random from the dataset.

STEP 2: Create a decision tree using the N records.

STEP 3: Repeat steps 1 and 2 with the number of trees you want in your algorithm.

STEP 4: In the case of a regression problem, each tree in the forest predicts a value for Y for a new record (output). The ultimate value can be computed by averaging all of the anticipated values from all of the trees in the forest. Alternatively, each tree in the forest forecasts the category to which the new record belongs in the event of a classification challenge. Finally, the new record is given to the category that receives the greatest number of votes.

4.4.1.2.2. BOOSTING

Boosting is a sequential procedure in which each subsequent model seeks to rectify the prior model's mistakes. The models that follow are reliant on the prior model.
Boosting is an ensemble learning strategy for minimizing training errors by combining a group of weak learners into a strong learner. A random sample of data is chosen, fitted with a model, and then trained progressively in boosting—that is, each model attempts to compensate for the shortcomings of its predecessor. The working of Boosting is shown in Fig 4.3.1

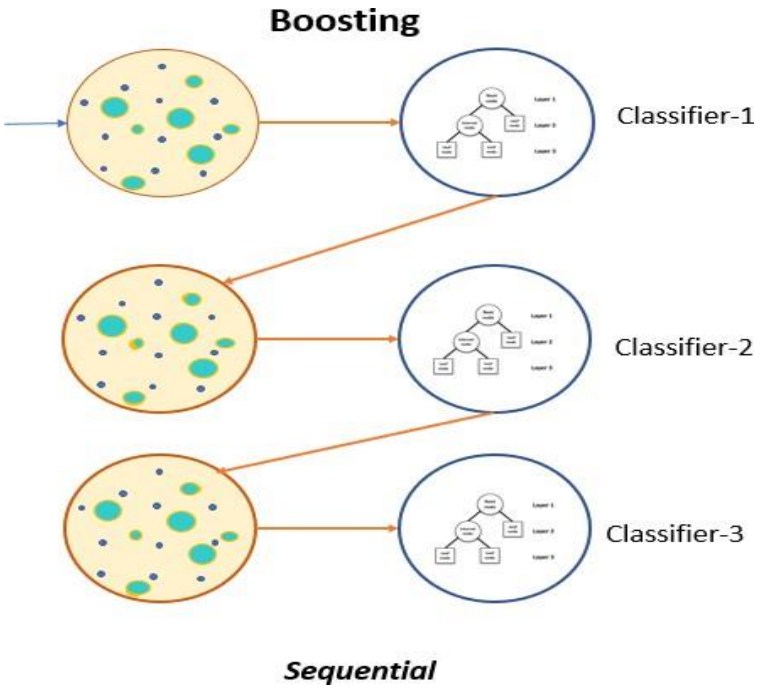


Figure 4.4.2

Working of Boosting

Boosting Algorithms:

- Ada Boost
- Gradient Boosting
- XGBoost
- Light GBM
- CatBoost

4.4.1.2.2.1. ADABOOST

AdaBoost, also known as Adaptive Boosting, is a Machine Learning approach that is employed as part of an Ensemble Method. The most frequent AdaBoost algorithm is one-level decision trees, which is decision trees with only one split. Decision Stumps is another name for these trees.

PROCEDURE

STEP 1: All observations in the dataset are given equal weights at the start.

STEP 2: A model is constructed using a subset of data.

STEP 3: This model is used to make predictions across the entire dataset.

STEP 4: Errors are determined by comparing predicted and actual results.

STEP 5: When developing the next model, the data points that were mistakenly forecasted are assigned higher weights.

STEP 6: The error value can be used to calculate weights. For example, the greater the mistake, the more the weight given to the observation.

STEP 7: The method is repeated

4.4.1.2.2.2. GRADIENT BOOSTING

GBM (Gradient Boosting Machine Learning) is another ensemble machine learning technique that may be used to solve both regression and classification problems. GBM employs the boosting strategy, which entails combining several weak learners into a single strong learner. As a base learner, regression trees are utilised, with each succeeding tree in the series being built on the errors calculated by the preceding tree.

PROCEDURE

STEP 1: The mean is assumed to be the predicted value for all observations in the dataset.

STEP 2: The errors are calculated using this mean prediction and actual values of the mean.

STEP 3: A tree model is created using the errors. Our objective is to find the best split to minimize the error.

STEP 4: New errors are calculated using this predicted value and actual value.

STEP 5: Steps 2 to 4 are repeated till the maximum number of iterations is reached (or error function does not change).

4.4.1.2.2.3. XGBOOST

Extreme Gradient Boosting (XGBoost) is a more advanced version of the gradient boosting method. XGBoost has shown to be a powerful machine learning algorithm that has been widely used in competitions and hackathons. XGBoost is about 10 times faster than other gradient boosting approaches and has a good predictive potential. It also features regularisation techniques that decrease overfitting and boost overall performance. As a result, the approach is also known as "regularised boosting."

PROCEDURE

STEP 1: Build a Model in training dataset.

STEP 2: In XGBoost model the Decision tree are created in sequential form.

STEP 3: Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.

4.5. COMPARATIVE ANALYSIS (PHASE 5)

A Comparative Analysis of feature selection namely Filter Method (Quasi Constant and Pearson Correlation), Wrapper Method (Stepwise Feature selection and Recursive Feature Elimination) and Embedded Method (Lasso and Tree based) gives and selects the best features for model building

A Comparative Analysis for six algorithms namely Voting classifier, Bagging Classifier, Random Forest, Ada Boost, Gradient Boosting and XGBoost gives the best accuracy levels for this Phishing websites detection.

The Feature Selection is compared and select the highly reduced features for model building.

The model building is made in selected Feature Selection data. The six different Algorithms are compared based on their accuracy level, and different classification metrics precision score, recall score, and f1 score. And finally based on the higher level of test accuracy the model is chosen as the best model to fit. One model is chosen to be the best one. Based on their performance the accuracy is improved. Comparative Analysis of Feature Selection (Filter, Wrapper and Embedded) is shown in the Fig 4.5.1

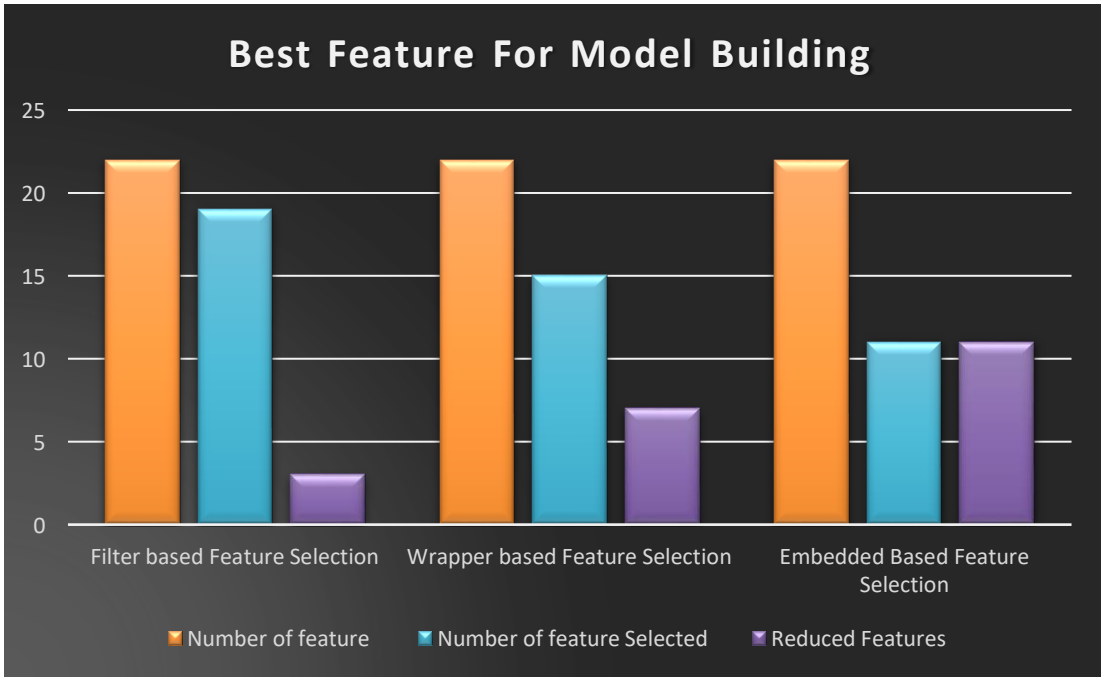


Figure 4.5.1

Comparative Analysis of Feature Selection (Filter, Wrapper and Embedded)

	Model	Test_Accuracy	Precision	f1_score	recall_score
0	Voting Classifier	93.10%	95.52%	93.00%	90.61%
1	Bagging Classifier	94.80%	93.24%	94.96%	96.74%
2	Random Forest	95.45%	94.15%	95.57%	97.04%
3	Ada Boost	93.60%	95.47%	93.55%	91.70%
4	Gradient Boosting	90.15%	91.62%	90.11%	88.64%
5	XG Boost	93.25%	95.25%	93.19%	91.21%

Figure 4.5.2

Comparative Analysis of Model Building

4.6. RESULT AND DISCUSSION

PHASE 3: FEATURE SELECTION

Phishing Websites have been detected using Feature Selection Techniques. Filter, Wrapper, and Embedded were utilized in the feature selection process.

FILTER METHOD (QUASI CONSTANT)

Quasi-Constant and Pearson Correlation based feature selection techniques have been used in Filter based Feature selection. Fig 4.6.1 shows the dropped features in quasi constant feature selection.

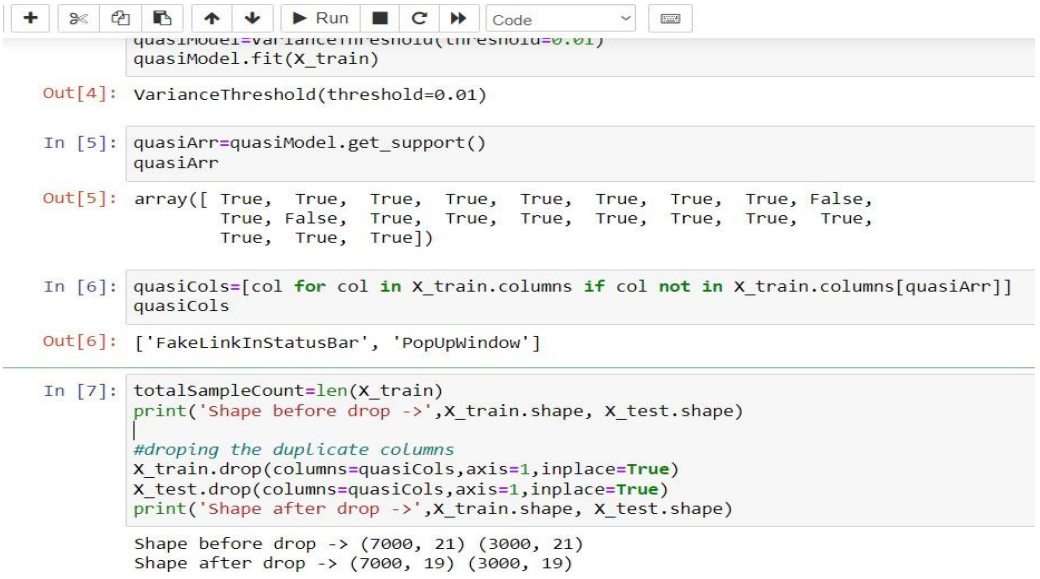
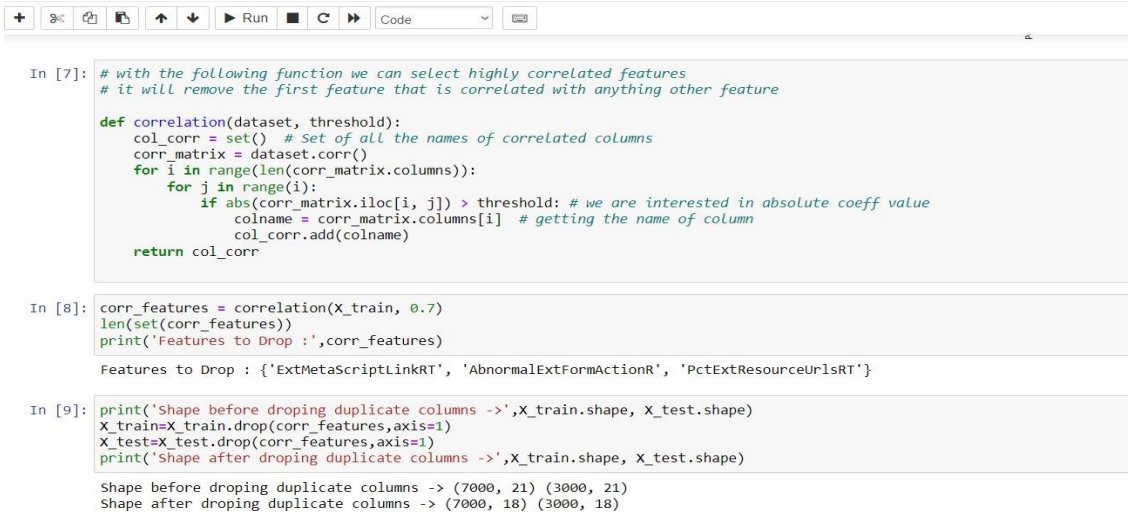


Figure 4.6.1

Quasi Constant Feature Selection

FILTER METHOD (PEARSON CORRELATION)

Fig 4.6.2 shows the dropped features in Pearson Correlation Feature Selection.



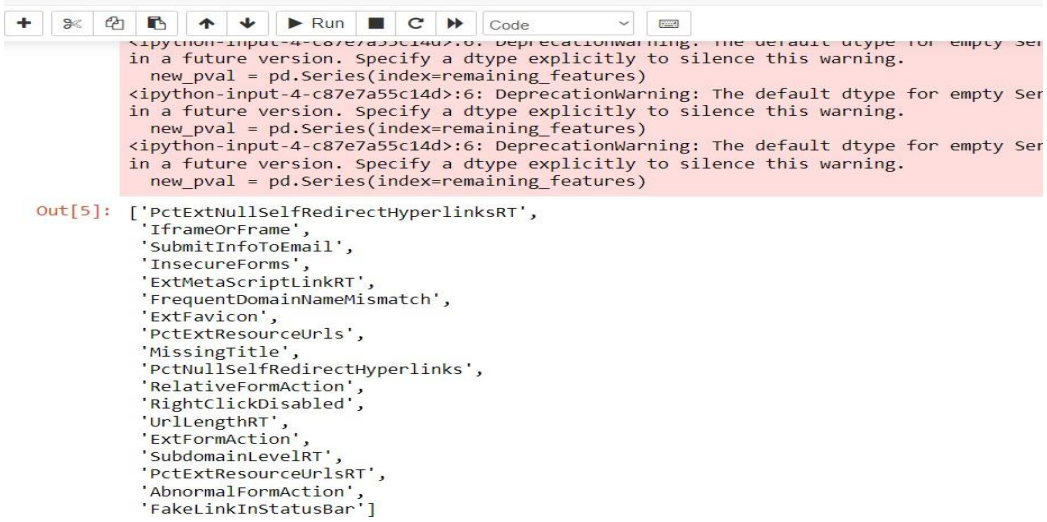
```
In [7]: # with the following function we can select highly correlated features
        # it will remove the first feature that is correlated with anything other feature

        def correlation(dataset, threshold):
            col_corr = set()  # Set of all the names of correlated columns
            corr_matrix = dataset.corr()
            for i in range(len(corr_matrix.columns)):
                for j in range(i):
                    if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                        colname = corr_matrix.columns[i]  # getting the name of column
                        col_corr.add(colname)
            return col_corr

In [8]: corr_features = correlation(X_train, 0.7)
        len(set(corr_features))
        print('Features to Drop :',corr_features)

        Features to Drop : {'ExtMetaScriptLinkRT', 'AbnormalExtFormActionR', 'PctExtResourceUrlsRT'}

In [9]: print('Shape before droping duplicate columns ->',X_train.shape, X_test.shape)
        X_train=X_train.drop(corr_features,axis=1)
        X_test=X_test.drop(corr_features,axis=1)
        print('Shape after droping duplicate columns ->',X_train.shape, X_test.shape)

        Shape before droping duplicate columns -> (7000, 21) (3000, 21)
        Shape after droping duplicate columns -> (7000, 18) (3000, 18)
```

Figure 4.6.2

Pearson Correlation Feature Selection

WRAPPER METHOD (STEPWISE FEATURE SELECTION)

Wrapper-based feature selection uses a step-by-step feature selection technique as well as recursive feature selection. Fig 4.6.3 shows the selected features in Stepwise Feature Selection.



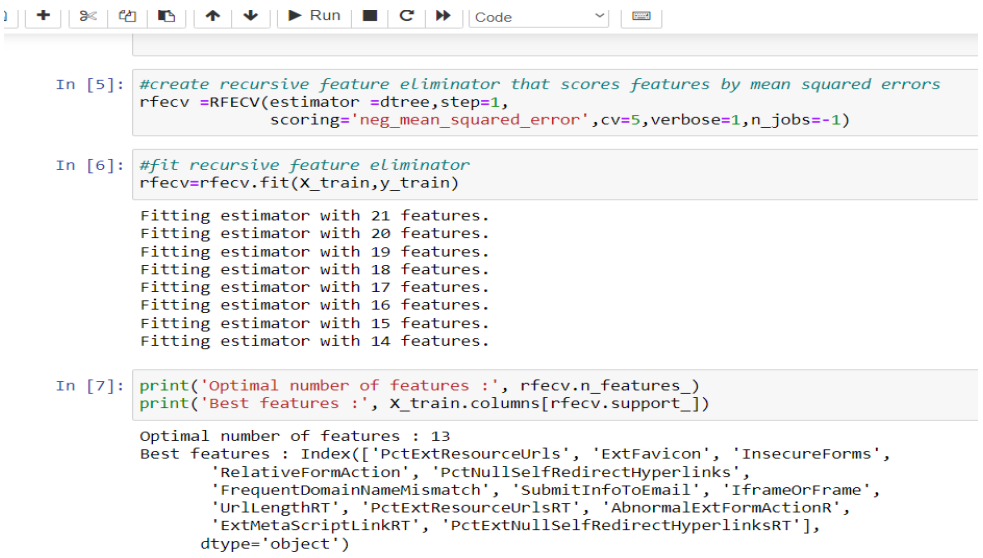
```
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Ser
in a future version. Specify a dtype explicitly to silence this warning.
  new_pval = pd.Series(index=remaining_features)
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Ser
in a future version. Specify a dtype explicitly to silence this warning.
  new_pval = pd.Series(index=remaining_features)
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Ser
in a future version. Specify a dtype explicitly to silence this warning.
  new_pval = pd.Series(index=remaining_features)

Out[5]: ['PctExtNullSelfRedirectHyperlinksRT',
         'IframeOrFrame',
         'SubmitInfoToEmail',
         'InsecureForms',
         'ExtMetaScriptLinkRT',
         'FrequentDomainNameMismatch',
         'ExtFavicon',
         'PctExtResourceUrls',
         'MissingTitle',
         'PctNullSelfRedirectHyperlinks',
         'RelativeFormAction',
         'RightClickDisabled',
         'UrlLengthRT',
         'ExtFormAction',
         'SubdomainLevelRT',
         'PctExtResourceUrlsRT',
         'AbnormalFormAction',
         'FakeLinkInStatusBar']
```

Figure 4.6.3

Stepwise Feature Selection

WRAPPER METHOD (RECURSIVE FEATURE ELIMINATION)

Fig 4.6.4 shows the selected features in Recursive Feature Elimination.



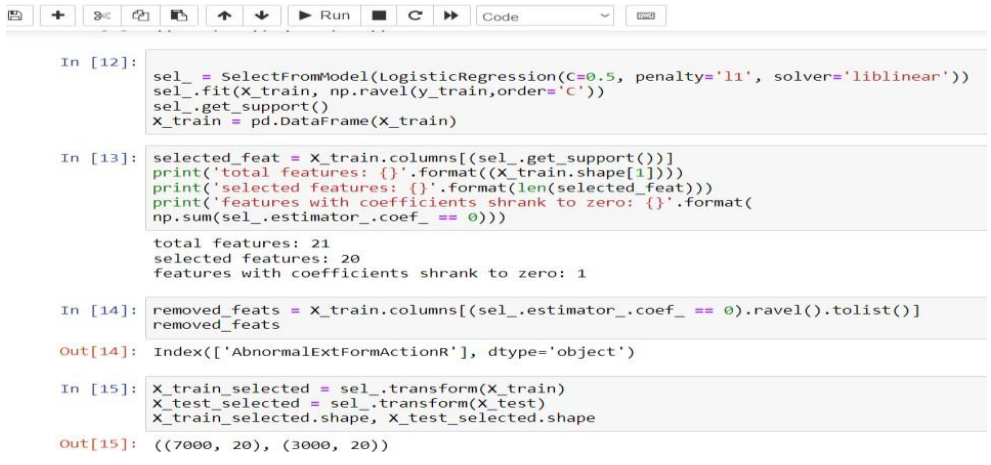
```
In [5]: #create recursive feature eliminator that scores features by mean squared errors
        rfecv =RFECV(estimator =dtree,step=1,
                     scoring='neg_mean_squared_error',cv=5,verbose=1,n_jobs=-1)

In [6]: #fit recursive feature eliminator
        rfecv=rfecv.fit(X_train,y_train)

        Fitting estimator with 21 features.
        Fitting estimator with 20 features.
        Fitting estimator with 19 features.
        Fitting estimator with 18 features.
        Fitting estimator with 17 features.
        Fitting estimator with 16 features.
        Fitting estimator with 15 features.
        Fitting estimator with 14 features.

In [7]: print('Optimal number of features :', rfecv.n_features_)
        print('Best features :', X_train.columns[rfecv.support_])

        Optimal number of features : 13
        Best features : Index(['PctExtResourceUrls', 'ExtFavicon', 'InsecureForms',
               'RelativeFormAction', 'PctNullSelfRedirectHyperlinks',
               'FrequentDomainNameMismatch', 'SubmitInfoToEmail', 'IframeOrFrame',
               'UrlLengthRT', 'PctExtResourceUrlsRT', 'AbnormalExtFormActionR',
               'ExtMetaScriptLinkRT', 'PctExtNullSelfRedirectHyperlinksRT'],
              dtype='object')
```

Figure 4.6.4

Recursive Feature Elimination

EMBEDDED METHOD (LASSO REGRESSION)

Two techniques, LASSO and Tree-based feature selection, have been implemented in Embedded based feature selection. Fig 4.6.5 shows the dropped feature in Lasso Regression.



```
In [12]: sel_ = SelectFromModel(LogisticRegression(C=0.5, penalty='l1', solver='liblinear'))
         sel_.fit(X_train, np.ravel(y_train,order='C'))
         sel_.get_support()
         X_train = pd.DataFrame(X_train)

In [13]: selected_feat = X_train.columns[(sel_.get_support())]
         print('total features: {}'.format((X_train.shape[1])))
         print('selected features: {}'.format(len(selected_feat)))
         print('features with coefficients shrank to zero: {}'.format(
         np.sum(sel_.estimator_.coef_ == 0)))

         total features: 21
         selected features: 20
         features with coefficients shrank to zero: 1

In [14]: removed_feats = X_train.columns[(sel_.estimator_.coef_ == 0).ravel().tolist()]
         removed_feats

Out[14]: Index(['AbnormalExtFormActionR'], dtype='object')

In [15]: X_train_selected = sel_.transform(X_train)
         X_test_selected = sel_.transform(X_test)
         X_train_selected.shape, X_test_selected.shape

Out[15]: ((7000, 20), (3000, 20))
```

Figure 4.6.5

Lasso Regression

EMBEDDED METHOD (TREE BASED)

Fig 4.6.6 shows the selected features in Tree based Feature Selection.

```
#Reduce X to the selected features and then predict using the predict
y_pred_rf=sel_rfe_tree.predict(X_test)
metrics.accuracy_score(y_test,y_pred_rf)
```

```
[ True  True  True False False False  True  True False False False  True
  True False False False  True False False  True  True]
[ 1  1  1  4  6  9  1  1 12 10 11  1  1  7  8  5  1  2  3  1  1]
```

Out[16]: 0.9493333333333334

In [13]:
```
#find the number of selected features

selected_cols = [column for column in X_train.columns
                 if column in X_train.columns[sel_rfe_tree.get_support()]]
selected_cols
```

Out[13]:
```
['PctExtResourceUrls',
 'ExtFavicon',
 'InsecureForms',
 'PctNullSelfRedirectHyperlinks',
 'FrequentDomainNameMismatch',
 'SubmitInfoToEmail',
 'IframeOrFrame',
 'UrlLengthRT',
 'ExtMetaScriptLinkRT',
 'PctExtNullSelfRedirectHyperlinksRT']
```

Figure 4.6.6

Tree based Feature Selection

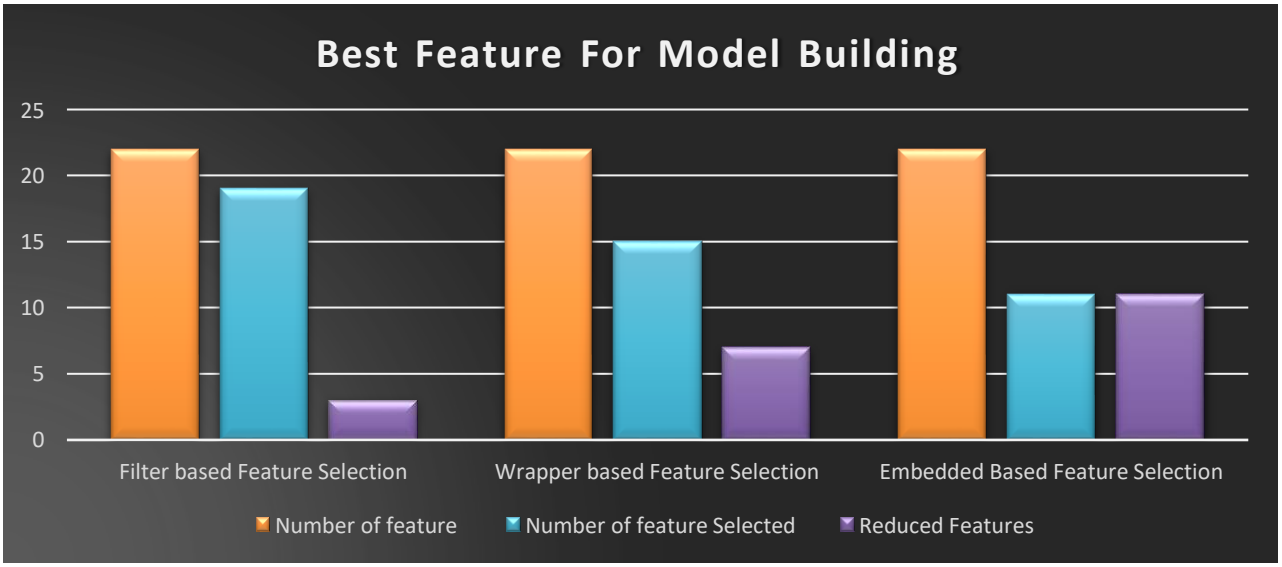


Figure 4.6.7

Comparative analysis for Feature Selection

PHASE 4: MODEL BUILDING

CLASSIFICATION AND PERFORMANCE EVALUATION METRIX ACCURACY

The classification metrics and performance evaluation metrics of Accuracy, Precision, F1 score and Recall score are shown in the below figures

ACCURACY

It is a measure of all the correctly identified cases. Accuracy is useful whenall the classes are equally important, Table 2 gives the **Accuracy** formula

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

TP-True Positive
TN- True Negative
FP-False Positive
FN- False Negative

Table 2 Accuracy formula

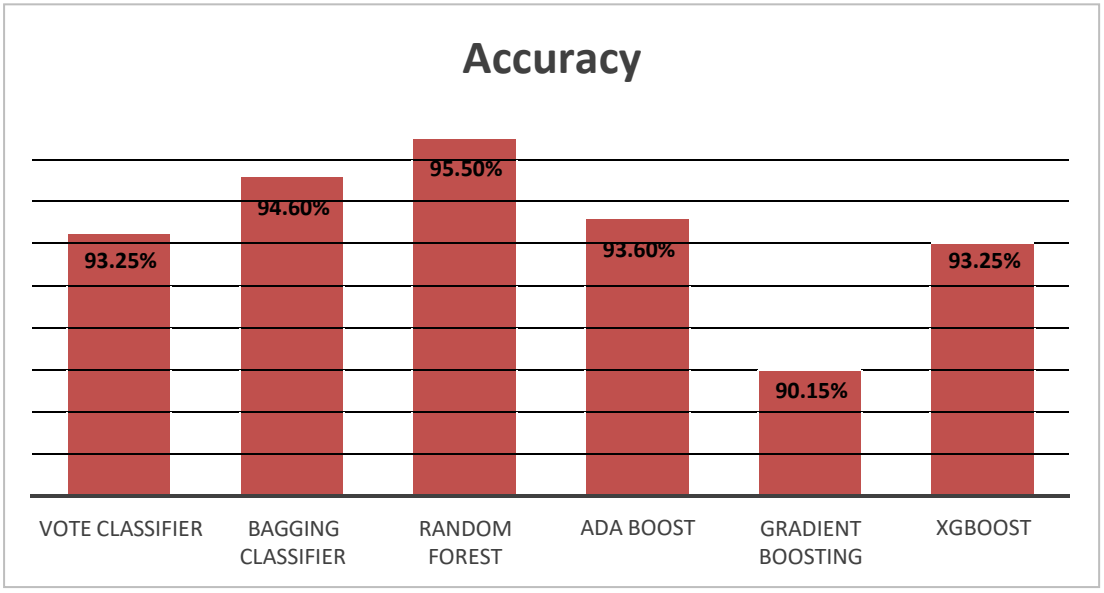


Figure 4.6.8

PRECISION

It is the ratio of correctly identified positive cases to all the predicted positivecases. This metric proves to be useful when the cost of False Positives is high., Table 3 gives the **Precision** formula

$$Precision = \frac{TP}{TP+FP}$$

TP-True Positive
FP-Flase Positive

Table 3 Precision formula

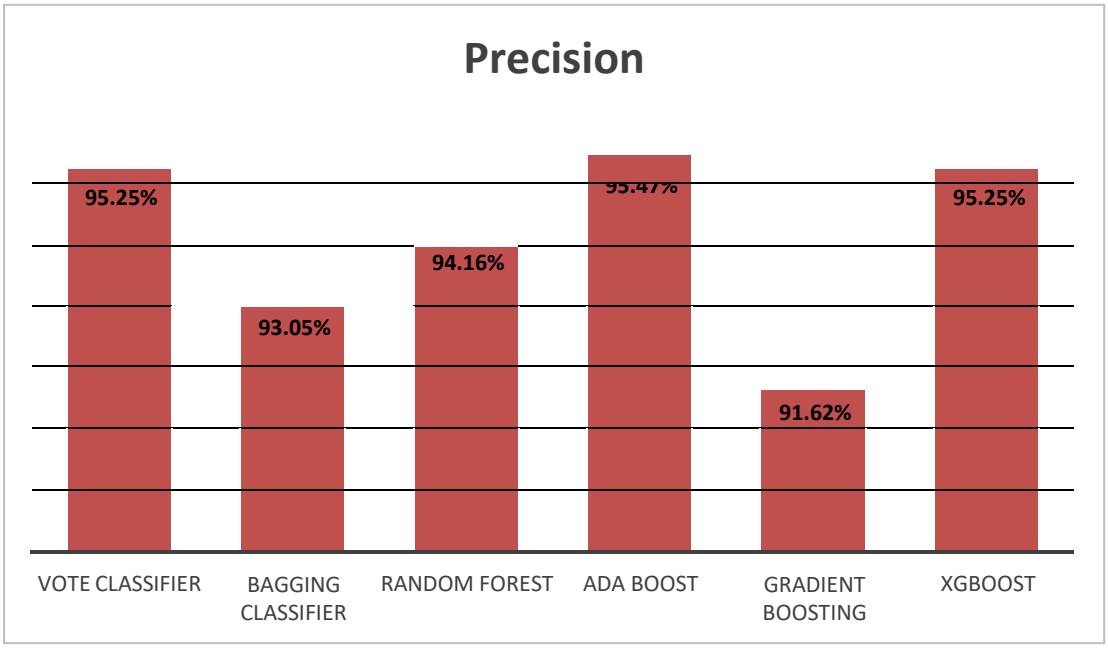


Figure 4.6.9

F1 SCORE

It is the harmonic mean of Precision and Recall and gives us a better measureof the incorrectly classified cases than the Accuracy metric, Table 4 gives the **F1- Score** formula

$$F1\ Score = \frac{Precision * Recall}{Precision + Recall}$$

Table 4 F1-Score formula

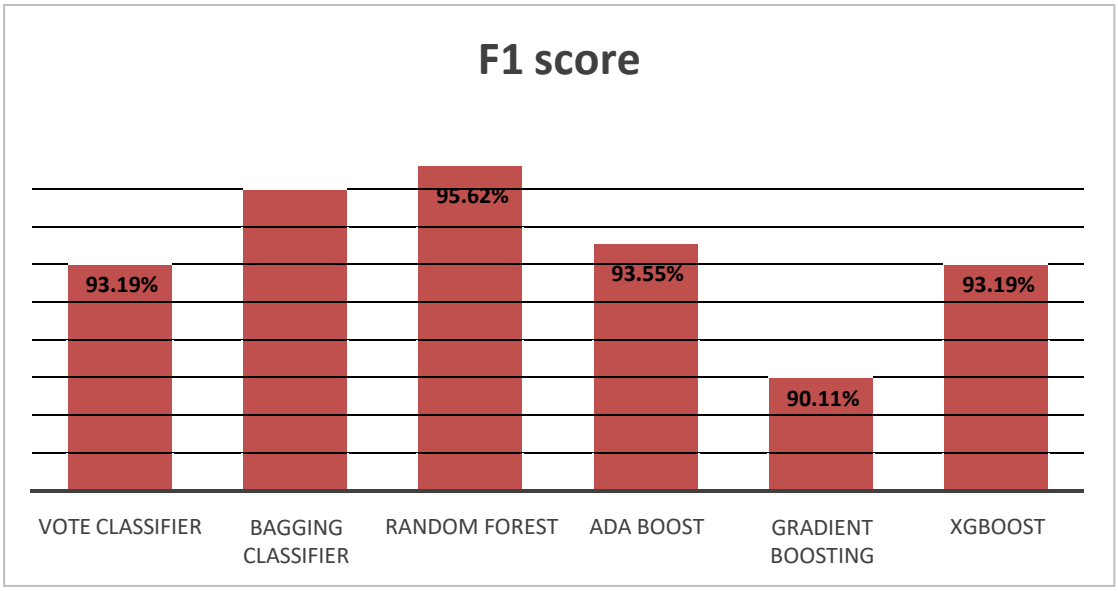


Figure 4.6.10

RECALL SCORE

It is the ratio of correctly identified positive cases to all the actual positive cases.This metric is particularly useful when the cost of False Negatives is high., Table 28 gives the **Recall** formula

$$Recall = \frac{TP}{TP+FP}$$

TP- True Positive
FN-False Negative

Table 5 Recall formula

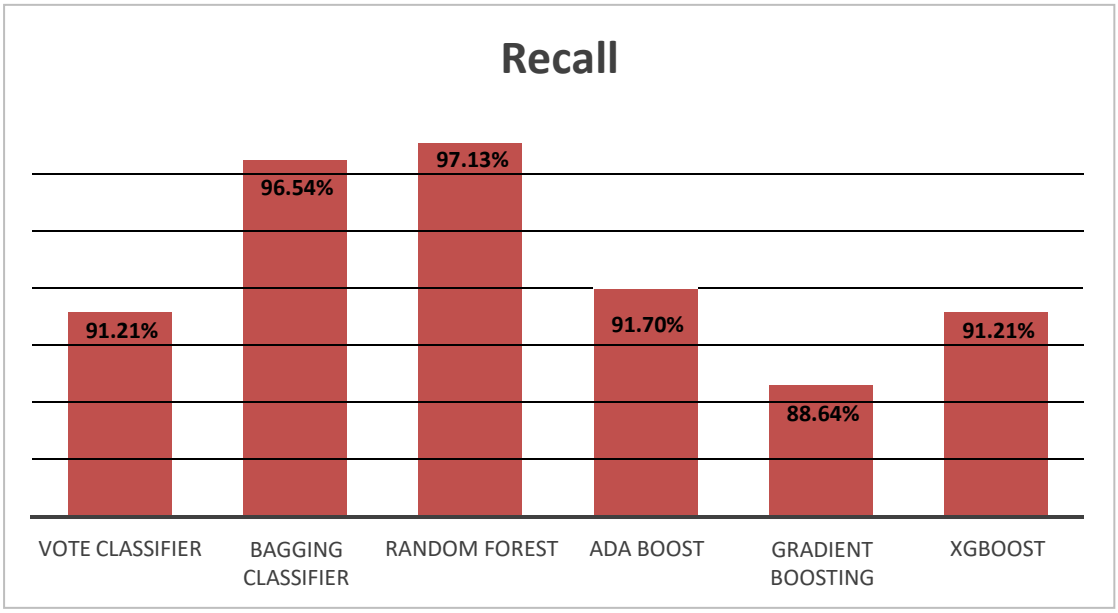


Figure 4.6.11

The results based on comparison; Random Forest gives the best accuracy (96%).

CONCLUSION

CHAPTER 5

5 CONCLUSION

Phishing attack is the most common cyber-attack where the attackers steal personal information in the form of email, website, SMS etc. In this project various Machine learning techniques such as (Filter, Wrapper and Embedded Feature Selection) have been implemented to analyze the dataset and selected the best features for model building and the performance was evaluated using Ensemble Learning classification algorithm. The Bagging Classifier ie (Random Forest Classification) gives the best accuracy for feature selected datasets.

5.1 SCOPE OF FUTURE ENHANCEMENT

Feature Selection techniques and Classification algorithm had been used in this project to detect the Phishing Websites. Whereas, Host based Feature and Correlation based Feature have been implemented. In future enhancements, compare machine learning with ensemble machine learning algorithms can be explored to achieve much better results.

REFERENCES

CHAPTER 6

6 REFERENCES

[1] Sharma, S. R., Parthasarathy, R., & Honnavalli, P. B. (2020). A Feature Selection Comparative Study for Web Phishing Datasets. *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. https://doi.org/10.1109/conecct50063.2020.9198349

[2] Rashid, J., Mahmood, T., Nisar, M. W., & Nazir, T. (2020). Phishing Detection Using Machine Learning Technique. *2020 First International Conference of Smart Systems and Emerging Technologies (SMARTTECH)*. https://doi.org/10.1109/smart-tech49988.2020.00026

[3] Alswailem, A., Alabdullah, B., Alrumayh, N., & Alsedrani, A. (2019). Detecting Phishing Websites Using Machine Learning. *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*. https://doi.org/10.1109/cais.2019.8769571

[4] Alkawaz, M. H., Steven, S. J., & Hajamydeen, A. I. (2020a). Detecting Phishing Website Using Machine Learning. 2020 16th IEEE International Colloquium on Signal Processing & Its Applications (CSPA). https://doi.org/10.1109/cspa48992.2020.9068728

[5] Korkmaz, M., Sahingoz, O. K., & Diri, B. (2020). Detection of Phishing Websites by Using Machine Learning-Based URL Analysis. 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). https://doi.org/10.1109/icccnt49239.2020.9225561

[6] Alam, M. N., Sarma, D., Lima, F. F., Saha, I., Ulfath, R. E., & Hossain, S. (2020b). Phishing Attacks Detection using Machine Learning Approach. *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*. https://doi.org/10.1109/icssit48917.2020.9214225

[7] https://www.warse.org/IJATCSE/static/pdf/file/ijatcse242942020.pdf

APPENDIX

CHAPTER 7

7.1 CODING

DATA PREPROCESSING

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.info()
dataset.head()
dataset.isnull().sum()
dataset.isnull().values.any()
X_features = dataset.drop(labels=['CLASS_LABEL'],axis=1)
y_labels = dataset['CLASS_LABEL']
X_train,X_test,y_train,y_test=train_test_split(X_features,y_labels,test_size=0.2,random_state
=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

FEATURE SELECTION

FILTER METHOD (QUASI CONSTANT)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
```

```
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
#Spillting the dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(
dataset.drop(labels=["CLASS_LABEL"],axis = 1),
dataset["CLASS_LABEL"],
 test_size=0.3,
random_state=0)
X_train.shape, X_test.shape
X_train_org=X_train.copy()
X_test_org=X_test.copy()
#Quasi -Constant
quasiModel=VarianceThreshold(threshold=0.01)
quasiModel.fit(X_train)
quasiArr=quasiModel.get_support()
quasiArr
quasiCols=[col for col in X_train.columns if col not in X_train.columns[quasiArr]]
quasiCols
totalSampleCount=len(X_train)
print('Shape before drop ->',X_train.shape, X_test.shape)
#droping the duplicate columns
X_train.drop(columns=quasiCols,axis=1,inplace=True)
X_test.drop(columns=quasiCols,axis=1,inplace=True)
print('Shape after drop ->',X_train.shape, X_test.shape)
```

FILTER METHOD (PEARSON CORRELATION)

```
import pandas as pd
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
X_train, X_test, y_train, y_test = train_test_split(
 dataset.drop(labels=["CLASS_LABEL"],axis = 1),
 dataset["CLASS_LABEL"],
 test_size=0.3,
random_state=0)
X_train.shape, X_test.shape
X_train.corr()
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#Using Pearson Correlation
plt.figure(figsize=(20,15))
cor = X_train.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
```

```
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
    return col_corr
corr_features = correlation(X_train, 0.7)
len(set(corr_features))
print('Features to Drop :',corr_features)
print('Shape before droping duplicate columns ->',X_train.shape, X_test.shape)
X_train=X_train.drop(corr_features,axis=1)
X_test=X_test.drop(corr_features,axis=1)
print('Shape after droping duplicate columns ->',X_train.shape, X_test.shape)
```

WRAPPER METHOD (STEPWISE FEATURE SELECTION)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
import mlxtend
import statsmodels.api as sm
%matplotlib inline
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
```

```
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
def stepwise_selection(data,CLASS_LABEL,SL_in=0.05,SL_out = 0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(CLASS_LABEL,
sm.add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<SL_in):
            best_features.append(new_pval.idxmin())
            while(len(best_features)>0):
                best_features_with_constant = sm.add_constant(data[best_features])
                p_values = sm.OLS(CLASS_LABEL,
best_features_with_constant).fit().pvalues[1:]
                max_p_value = p_values.max()
                if(max_p_value >= SL_out):
                    excluded_feature = p_values.idxmax()
                    best_features.remove(excluded_feature)
                else:
                    break
        else:
            break
```

```
return best_features
stepwise_selection(X,y)
```

WRAPPER METHOD (RECURSIVE FEATURE ELIMINATION)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFECV
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
X_train, X_test, y_train, y_test = train_test_split(
 dataset.drop(labels=["CLASS_LABEL"],axis = 1),
 dataset["CLASS_LABEL"],
 test_size=0.3,
random_state=0)
X_train.shape, X_test.shape
#Create Linear Model
dtree = DecisionTreeRegressor()
#create recursive feature eliminator that scores features by mean squared errors
rfecv =RFECV(estimator =dtree,step=1,
            scoring='neg_mean_squared_error',cv=5,verbose=1,n_jobs=-1)
```

```
#fit recursive feature eliminator
rfecv=rfecv.fit(X_train,y_train)
print('Optimal number of features :', rfecv.n_features_)
print('Best features :', X_train.columns[rfecv.support_])
```

EMBEDDED METHOD (LASSO REGULARIZATION)

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso, LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.preprocessing import StandardScaler
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
X_train, X_test, y_train, y_test = train_test_split(
 dataset.drop(labels=["CLASS_LABEL"],axis = 1),
 dataset["CLASS_LABEL"],
 test_size=0.3,
random_state=0)
X_train.shape, X_test.shape
sel_ = SelectFromModel(LogisticRegression(C=0.5, penalty='l1', solver='liblinear'))
sel_.fit(X_train, np.ravel(y_train,order='C'))
sel_.get_support()
```

```
X_train = pd.DataFrame(X_train)
selected_feat = X_train.columns[(sel_.get_support())]
print('total features: {}'.format((X_train.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
print('features with coefficients shrank to zero: {}'.format(
np.sum(sel_.estimator_.coef_ == 0)))
removed_feats = X_train.columns[(sel_.estimator_.coef_ == 0).ravel().tolist()]
removed_feats
X_train_selected = sel_.transform(X_train)
X_test_selected = sel_.transform(X_test)
X_train_selected.shape, X_test_selected.shape
```

EMBEDDED METHOD (TREE METHOD)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
#import the RFE from sklearn library
from sklearn.feature_selection import RFE,SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
```

```
df.head()
X_train, X_test, y_train, y_test = train_test_split(
 dataset.drop(labels=["CLASS_LABEL"],axis = 1),
 dataset["CLASS_LABEL"],
test_size=0.3,
random_state=0)
X_train.shape, X_test.shape
model_tree = RandomForestClassifier(n_estimators=100,random_state=42)
#use RFE to eliminate the less importance features
sel_rfe_tree = RFE(estimator=model_tree,n_features_to_select=10,step=1)
X_train_rfe_tree = sel_rfe_tree.fit_transform(X_train,y_train)
print(sel_rfe_tree.get_support())
print(sel_rfe_tree.ranking_)
#Reduce X to the selected features and then predict using the predict
y_pred_rf=sel_rfe_tree.predict(X_test)
metrics.accuracy_score(y_test,y_pred_rf)
#find the number of selected features
selected_cols = [column for column in X_train.columns
        if column in X_train.columns[sel_rfe_tree.get_support()]]
selected_cols
```

ENSEMBLE LEARNING

MODEL BUILDING IN EMBEDDED FEATURE SELECTED DATASET

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.metrics import classification_report
from sklearn import ensemble
from sklearn import metrics
import warnings
dataset=pd.read_excel("Emb_feat_sel.xlsx")
dataset.info()
for col in dataset:
  plt.hist(dataset[col],bins = 3)
  plt.xlabel(col)
  plt.ylabel("Frequency")
  plt.title(col)
   plt.show()
X_features = dataset.drop(labels=['CLASS_LABEL'],axis=1)
y_labels = dataset['CLASS_LABEL']
X_train,X_test,y_train,y_test=train_test_split(X_features,y_labels,
                                    test_size=0.2,random_state=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

SIMPLE TECHNIQUE

VOTING CLASSIFIER

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()
model1.fit(X_train,y_train)
```

```
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)
pred1=model1.predict(X_test)
pred2=model2.predict(X_test)
pred3=model3.predict(X_test)
import statistics
final_pred = np.array([])
for i in range(0,len(X_test)):
    final_pred = np.append(final_pred, statistics.mode([pred1[i], pred2[i], pred3[i]]))
from sklearn.ensemble import VotingClassifier
model1  =  LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
model.fit(X_train,y_train)
model.score(X_test,y_test)
y_predict = model.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_predict)
#cm vizualization
import seaborn as sns
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_predict")
plt.ylabel("y_test")
plt.show()
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
print('Voting Classifier Accuracy: {:.02%}'.format(model.score(X_test, y_test)))
```

```
print("Accuracy on test set for Voting Classifier: {:.02%}"
  .format(accuracy_score(y_test,model.predict(X_test))))
precision_vc = metrics.precision_score(y_test,y_predict)
precision = {"Voting Classifier":precision_vc}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_vc = metrics.recall_score(y_test,y_predict)
recall = {"Voting CLASSIFIER":recall_vc}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_vc = metrics.f1_score(y_test,y_predict)
f1_score = {"Voting Classifier":f1_score_vc}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
mae_vc = mean_absolute_error(y_test, y_predict)
print("MAE for VC:", mae_vc)
mse_vc = mean_squared_error(y_test, y_predict)
print("MSE for VC:", mse_vc)
rmse_vc = np.sqrt(mse_vc)
print("RMSE for VC:", rmse_vc)
```

ADVANCED TECHNIQUE

BAGGING

BAGGING META – ESTIMATOR

```
from sklearn.ensemble import BaggingClassifier
model1 = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
model1.fit(X_train, y_train)
```

```
model1.score(X_test,y_test)
y_bag_predict = model1.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_bag_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_bag_predict")
plt.ylabel("y_test")
plt.show()
print(classification_report(y_test, y_bag_predict))
print('Bagging Classifier Accuracy: {:.02%}'
     .format(model1.score(X_test, y_test)))
precision_bag = metrics.precision_score(y_test,y_bag_predict)
precision = {"Bagging Classifier":precision_bag}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_bag = metrics.recall_score(y_test,y_bag_predict)
recall = {"Bagging CLASSIFIER":recall_bag}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_bag = metrics.f1_score(y_test,y_bag_predict)
f1_score = {"Bagging Classifier":f1_score_bag}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
mae_bag = mean_absolute_error(y_test, y_bag_predict)
print("MAE for BAG:", mae_bag)
mse_bag = mean_squared_error(y_test, y_bag_predict)
print("MSE for BAG:", mse_bag)
```

```
rmse_bag = np.sqrt(mse_bag)
print("RMSE for BAG:", rmse_bag)
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier()
model2.fit(X_train,y_train)
model2.score(X_test,y_test)
y_rf_predict = model2.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_rf_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_rf_predict")
plt.ylabel("y_test")
plt.show()
print(classification_report(y_test, y_rf_predict))
print('Random Forest Accuracy: {:.02%}'
     .format(model2.score(X_test, y_test)))
precision_rf = metrics.precision_score(y_test,y_rf_predict)
precision = {"Random Forest":precision_rf}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_rf = metrics.recall_score(y_test,y_rf_predict)
recall = {"Random Forest":recall_rf}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_rf = metrics.f1_score(y_test,y_rf_predict)
```

```
f1_score = {"Random Forest":f1_score_rf}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
mae_rf = mean_absolute_error(y_test, y_rf_predict)
print("MAE for RF:", mae_rf)
mse_rf = mean_squared_error(y_test, y_rf_predict)
print("MSE for RF:", mse_rf)
rmse_rf = np.sqrt(mse_rf)
print("RMSE for RF:", rmse_rf)
```

BOOSTING

ADABOOST

```
from sklearn.ensemble import AdaBoostClassifier
model3 = AdaBoostClassifier(random_state=1)
model3.fit(X_train, y_train)
model3.score(X_test,y_test)
y_ada_predict=model3.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_ada_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5,
        linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_ada_predict")
plt.ylabel("y_test")
plt.show()
print(classification_report(y_test, y_ada_predict))
print('Ada Boost Accuracy: {:.02%}'
    .format(model3.score(X_test, y_test)))
precision_ada = metrics.precision_score(y_test,
```

```
                                    y_ada_predict)
precision = {"Ada Boost":precision_ada}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_ada = metrics.recall_score(y_test,
                                  y_ada_predict)
recall = {"Ada Boost":recall_ada}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_ada = metrics.f1_score(y_test,y_ada_predict)
f1_score = {"Ada Boost":f1_score_ada}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
mae_ada = mean_absolute_error(y_test, y_ada_predict)
print("MAE for ADA:", mae_ada)
mse_ada = mean_squared_error(y_test, y_ada_predict)
print("MSE for ADA:", mse_ada)
rmse_ada = np.sqrt(mse_ada)
print("RMSE for ADA:", rmse_ada)
```

GRADIENT BOOSTING

```
from sklearn.ensemble import GradientBoostingClassifier
model4= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model4.fit(X_train, y_train)
model4.score(X_test,y_test)
y_gb_predict=model4.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_gb_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_gb_predict")
```

```
plt.ylabel("y_test")
plt.show()
print(classification_report(y_test, y_gb_predict))
print('Gradient Boosting Accuracy: {:.02%}'
     .format(model4.score(X_test, y_test)))
precision_gb = metrics.precision_score(y_test,y_gb_predict)
precision = {"Gradient Boost":precision_gb}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_gb = metrics.recall_score(y_test,y_gb_predict)
recall = {"Gradient Boost":recall_gb}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_gb = metrics.f1_score(y_test,y_gb_predict)
f1_score = {"Gradient Boost":f1_score_gb}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
mae_gb = mean_absolute_error(y_test, y_gb_predict)
print("MAE for GB:", mae_gb)
mse_gb = mean_squared_error(y_test, y_gb_predict)
print("MSE for GB:", mse_gb)
rmse_gb = np.sqrt(mse_gb)
print("RMSE for GB:", rmse_gb)
```

XGBOOST

```
import xgboost as xgb
model5=xgb.XGBClassifier(random_state=1,learning_rate=0.01)
model5.fit(X_train, y_train)
model5.score(X_test,y_test)
y_xgb_predict=model5.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_xgb_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
```

```
plt.xlabel("y_xgb_predict")
plt.ylabel("y_test")
plt.show()
print(classification_report(y_test, y_xgb_predict))
print('XG Boost Accuracy: {:.02%}'.format(model5.score(X_test, y_test)))
precision_xgb = metrics.precision_score(y_test,y_xgb_predict)
precision = {"XG Boost":precision_xgb}
print("PRECISION")
print(pd.DataFrame(precision.items()))
recall_xgb = metrics.recall_score(y_test,y_xgb_predict)
recall = {"XG Boost":recall_xgb}
print("RECALL")
print(pd.DataFrame(recall.items()))
f1_score_xgb = metrics.f1_score(y_test,y_xgb_predict)
f1_score = {"XG Boost":f1_score_xgb}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
mae_xgb = mean_absolute_error(y_test, y_xgb_predict)
print("MAE for XGB:", mae_xgb)
mse_xgb = mean_squared_error(y_test, y_xgb_predict)
print("MSE for XGB:", mse_xgb)
rmse_xgb = np.sqrt(mse_xgb)
print("RMSE for XGB:", rmse_xgb)
```

COMPARATIVE ANALYSIS FOR MODEL BUILDING

```
Analysis_Model = []
Model_Test_Accuracy = []
Model_precision = []
Model_f1_score = []
Model_recall = []
def storeResults(model, a1, b,c,d,):
    Analysis_Model.append(model)
    Model_Test_Accuracy.append(a1)
    Model_precision.append(b)
```

```
    Model_f1_score.append(c)
    Model_recall.append(d)
VC_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model.predict(X_test))))
VC_Precision=("{:.02%}".format(metrics.precision_score(y_test,model.predict(X_test))))
VC_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model.predict(X_test))))
VC_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model.predict(X_test))))
BAG_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model1.predict(X_test))))
BAG_Precision=("{:.02%}".format(metrics.precision_score(y_test,model1.predict(X_test))))
BAG_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model1.predict(X_test))))
BAG_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model1.predict(X_test))))
RF_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model2.predict(X_test))))
RF_Precision=("{:.02%}".format(metrics.precision_score(y_test,model2.predict(X_test))))
RF_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model2.predict(X_test))))
RF_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model2.predict(X_test))))
ADA_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model3.predict(X_test))))
ADA_Precision=("{:.02%}".format(metrics.precision_score(y_test,model3.predict(X_test))))
ADA_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model3.predict(X_test))))
ADA_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model3.predict(X_test))))
GB_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model4.predict(X_test))))
GB_Precision=("{:.02%}".format(metrics.precision_score(y_test,model4.predict(X_test))))
GB_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model4.predict(X_test))))
GB_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model4.predict(X_test))))
XGB_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model5.predict(X_test))))
XGB_Precision=("{:.02%}".format(metrics.precision_score(y_test,model5.predict(X_test))))
XGB_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model5.predict(X_test))))
XGB_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model5.predict(X_test))))
storeResults('Voting Classifier' , VC_Test_Accuracy,
                                VC_Precision, VC_f1_score, VC_recall_score)
storeResults('Bagging Classifier' , BAG_Test_Accuracy,
BAG_Precision, BAG_f1_score, BAG_recall_score)
storeResults('Random Forest' , RF_Test_Accuracy,
 RF_Precision, RF_f1_score, RF_recall_score)
storeResults('Ada Boost' , ADA_Test_Accuracy,
ADA_Precision, ADA_f1_score, ADA_recall_score)
```

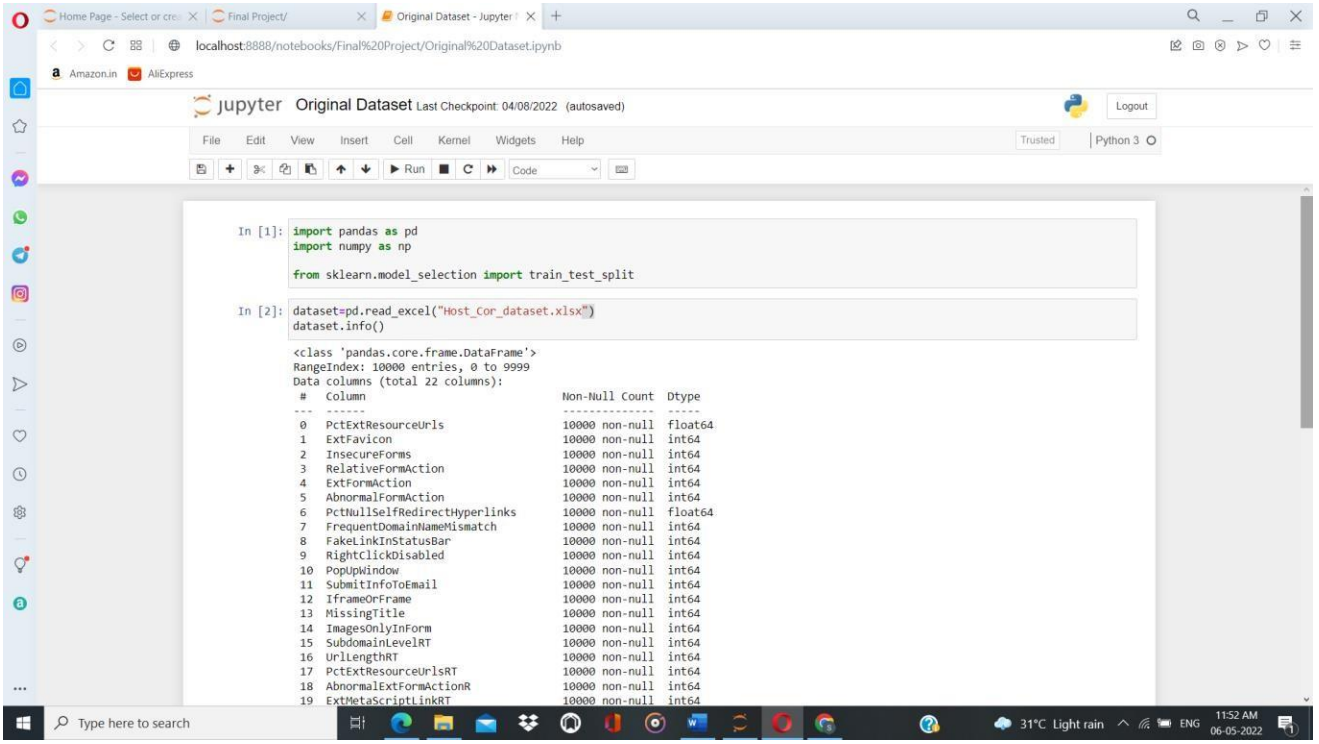
```
storeResults('Gradient Boosting' , GB_Test_Accuracy,
GB_Precision, GB_f1_score, GB_recall_score)
storeResults('XG Boost' , XGB_Test_Accuracy,
XGB_Precision, XGB_f1_score, XGB_recall_score)
result = pd.DataFrame({'Model': Analysis_Model,
          'Test_Accuracy' : Model_Test_Accuracy,'Precision' : Model_precision,
         'f1_score' : Model_f1_score,'recall_score' : Model_recall})
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
result
pip install colour
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_max(s):
    '''
    highlight the maximum in a Series green.
    '''
    is_max = s == s.max()
    return ['background-color: lightBlue' if v else '' for v in is_max]
print("\nHighlight the maximum value in each column:")
result.style.apply(highlight_max,subset=pd.IndexSlice[:, ['Test_Accuracy']])
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_max(s):
    '''
    highlight the maximum in a Series green.
    '''
  is_max = s == s.max()
    return ['background-color: lightBlue' if v else '' for v in is_max]
print("\nHighlight the maximum value in each column:")
result.style.apply(highlight_max,subset=pd.IndexSlice[:, ['Test_Accuracy',
                                        'Precision','f1_score','recall_score']])
# Highlight the Min values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_min(s):
    '''
```

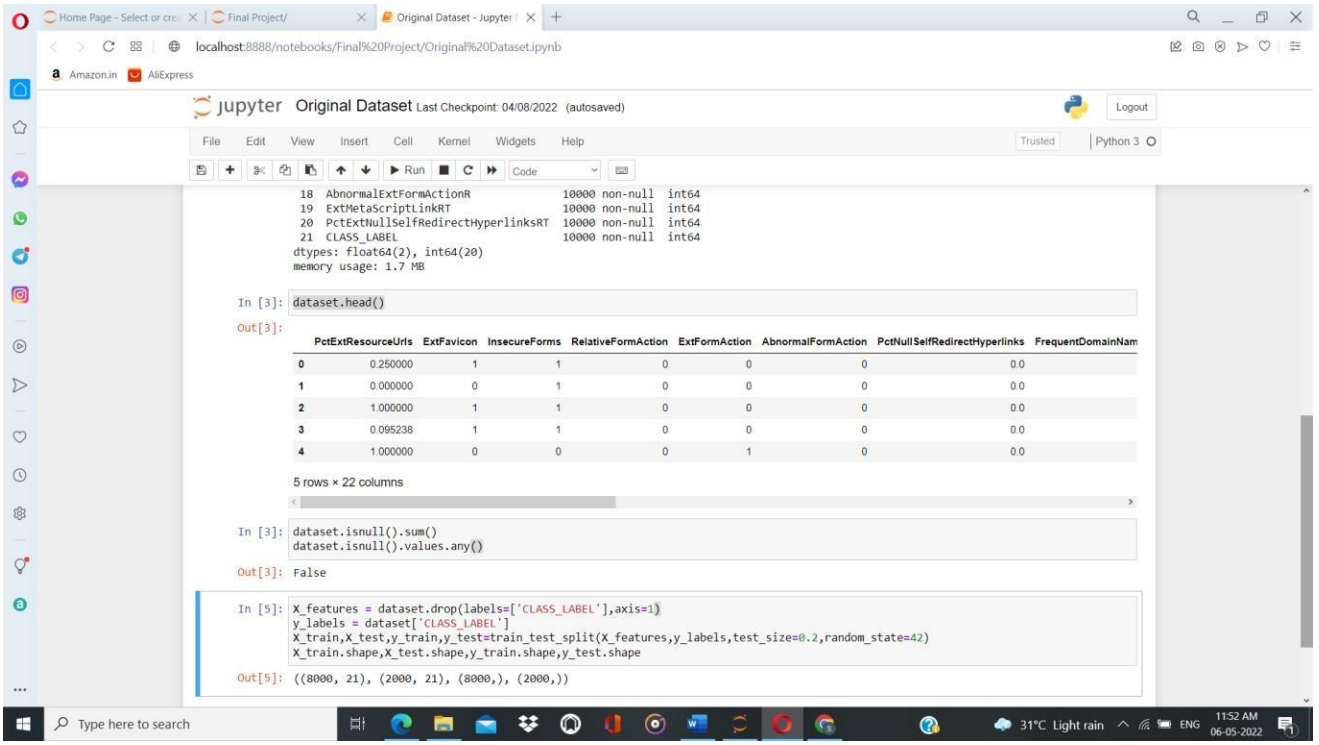
```
    highlight the minimum in a Series green.
    '''
    is_min = s == s.min()
    return ['background-color:cyan ' if v else '' for v in is_min]
print("\nHighlight the minimum value in each column:")
result.style.apply(highlight_min,subset=pd.IndexSlice[:, ['Test_Accuracy']])
# Highlight the Min values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_min(s):
    '''
    highlight the minimum in a Series green.
    '''
    is_min = s == s.min()
    return ['background-color: cyan' if v else '' for v in is_min]
print("\nHighlight the minimum value in each column:")
result.style.apply(highlight_min,subset=pd.IndexSlice[:, ['Test_Accuracy',
                                        'Precision', 'f1_score', 'recall_score']])
# Creating error model
Error_Model = []
MAE_Model = []
MSE_Model = []
RMSE_Model = []
def storeResults(E_Model, m1, m2, r):
    Error_Model.append(E_Model)
    MAE_Model.append(m1)
    MSE_Model.append(m2)
    RMSE_Model.append(r)
storeResults('VOTING CLASSIFIER', mae_vc, mse_vc, rmse_vc)
storeResults('BAGGING CLASSIFIER', mae_bag, mse_bag, rmse_bag)
storeResults('RANDOM FOREST', mae_rf, mse_rf, rmse_rf)
storeResults('ADA BOOST', mae_ada, mse_ada, rmse_ada)
storeResults('Gradient BOOSTINg', mae_gb, mse_xgb, rmse_gb)
storeResults('XG BOOST', mae_xgb, mse_xgb, rmse_xgb)
error_result = pd.DataFrame({'E_Model': Error_Model, 'MAE SCORE' :
                    MAE_Model, 'MSE SCORE' : MSE_Model,
```

```
                    'RMSE SCORE' : RMSE_Model})
print("COMPARITIVE ANALYSIS OF DIFFERENT ERROR RATES")
error_result
error_result.plot(kind="bar",figsize=(18, 8))
plt.title("Comparitive analysis of Error Rates",size = 20)
plt.xlabel("Different Ensemble Machine Learning Models",size = 18)
plt.legend(loc ="lower center")
plt.ylabel("Error_rates",size = 18)
plt.xticks(rotation=75)
```

7.2 SCREENSHOTS

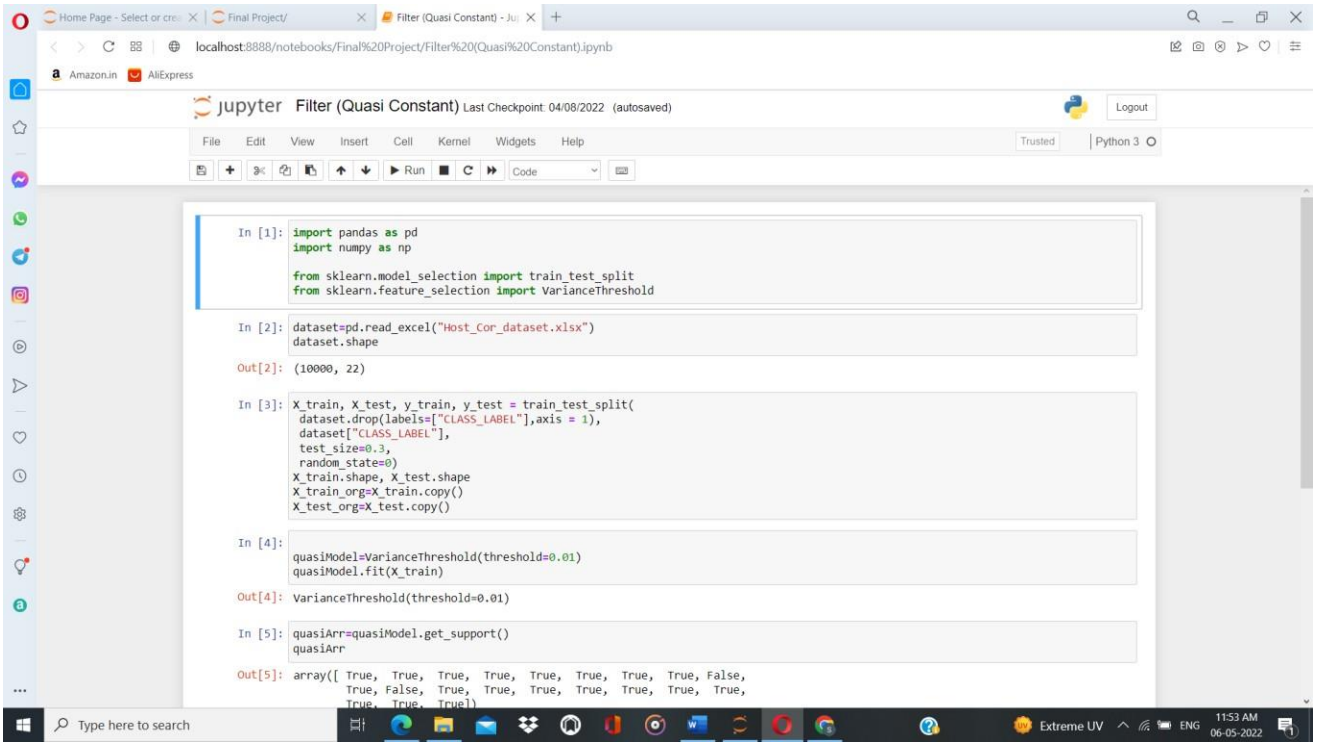
DATA PREPROCESSING



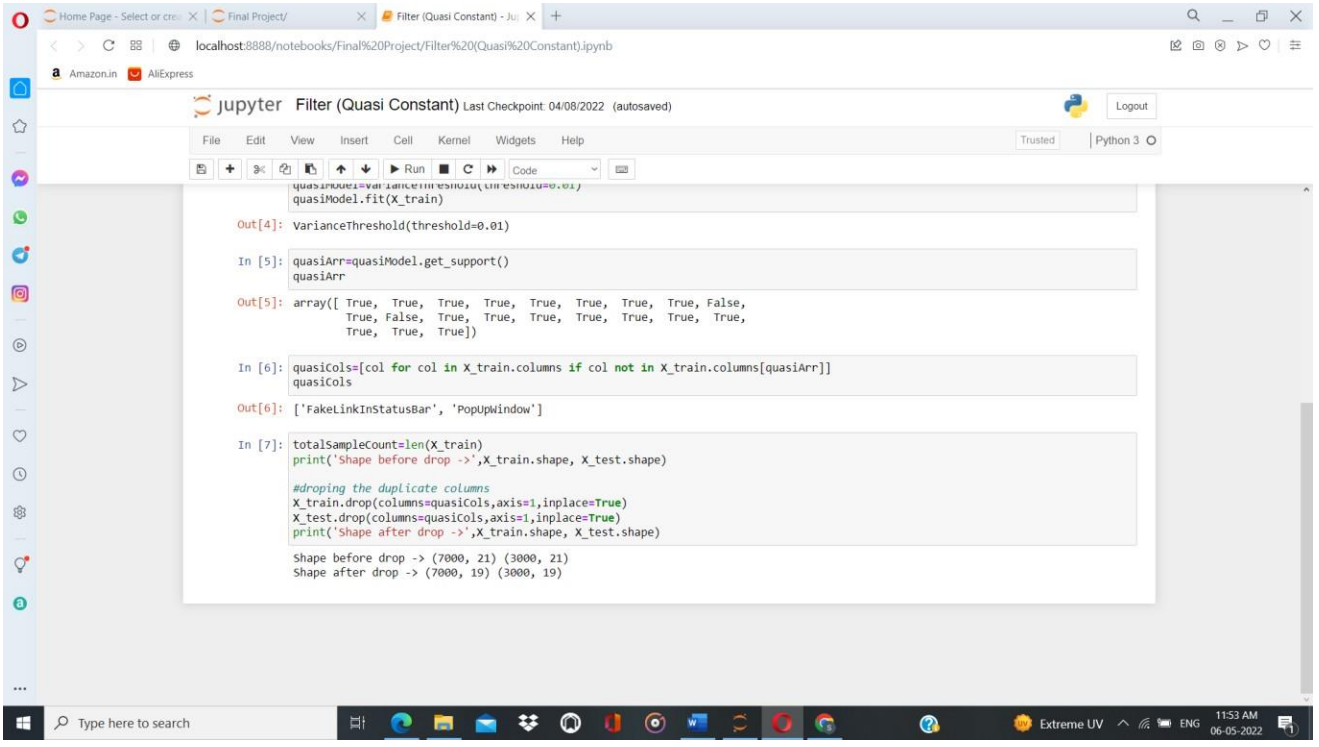


FEATURE SELECTION

FILTER METHOD (QUASI CONSTANT)

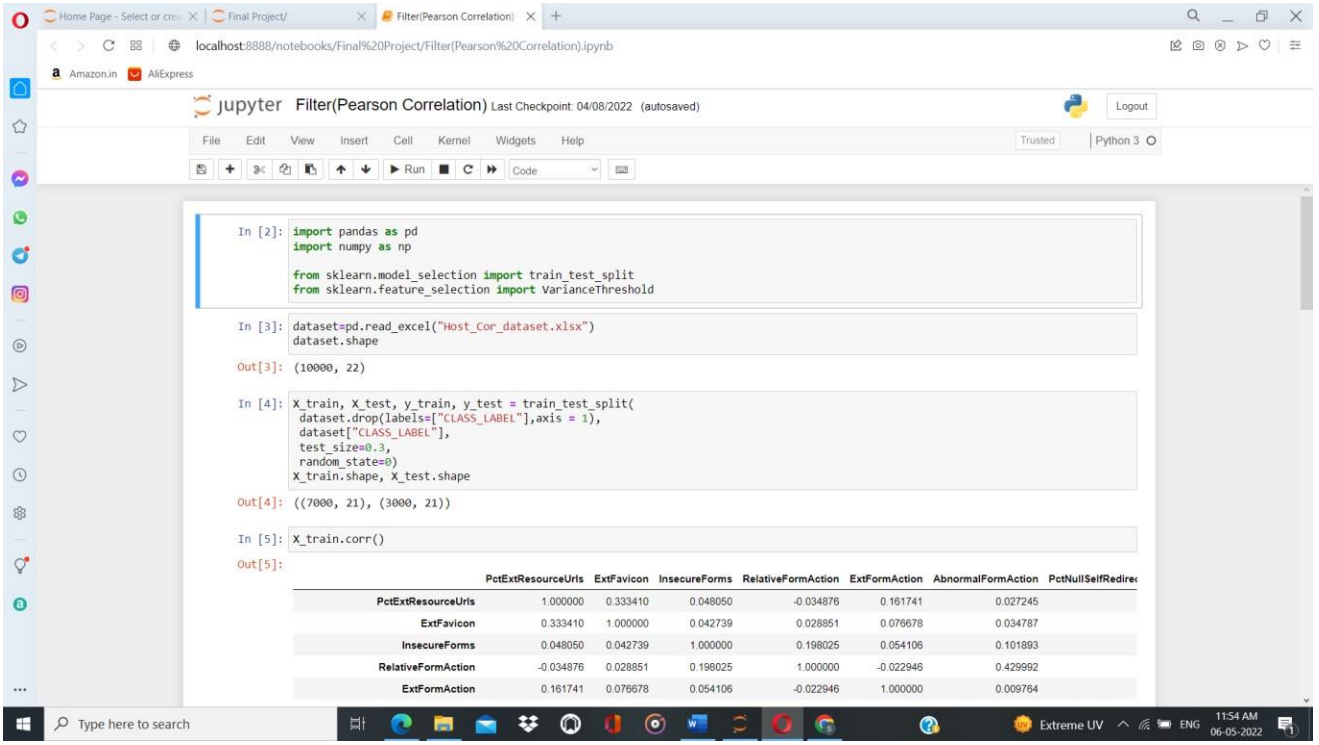


```
In [1]: import pandas as pd
        import numpy as np

        from sklearn.model_selection import train_test_split
        from sklearn.feature_selection import VarianceThreshold

In [2]: dataset=pd.read_excel("Host_Cor_dataset.xlsx")
        dataset.shape

Out[2]: (10000, 22)

In [3]: X_train, X_test, y_train, y_test = train_test_split(
         dataset.drop(labels=["CLASS_LABEL"],axis = 1),
         dataset["CLASS_LABEL"],
         test_size=0.3,
         random_state=0)
        X_train.shape, X_test.shape
        X_train_org=X_train.copy()
        X_test_org=X_test.copy()

In [4]:
        quasiModel=VarianceThreshold(threshold=0.01)
        quasiModel.fit(X_train)

Out[4]: VarianceThreshold(threshold=0.01)

In [5]: quasiArr=quasiModel.get_support()
        quasiArr

Out[5]: array([ True,  True,  True,  True,  True,  True,  True,  True, False,
                True, False,  True,  True,  True,  True,  True,  True,  True,
                True,  True,  True])
```



```
        quasiModel.fit(X_train)

Out[4]: VarianceThreshold(threshold=0.01)

In [5]: quasiArr=quasiModel.get_support()
        quasiArr

Out[5]: array([ True,  True,  True,  True,  True,  True,  True,  True, False,
                True, False,  True,  True,  True,  True,  True,  True,  True,
                True,  True,  True])

In [6]: quasiCols=[col for col in X_train.columns if col not in X_train.columns[quasiArr]]
        quasiCols

Out[6]: ['FakeLinkInStatusBar', 'PopUpWindow']

In [7]: totalSampleCount=len(X_train)
        print('Shape before drop ->',X_train.shape, X_test.shape)

        #droping the duplicate columns
        X_train.drop(columns=quasiCols,axis=1,inplace=True)
        X_test.drop(columns=quasiCols,axis=1,inplace=True)
        print('Shape after drop ->',X_train.shape, X_test.shape)

        Shape before drop -> (7000, 21) (3000, 21)
        Shape after drop -> (7000, 19) (3000, 19)
```

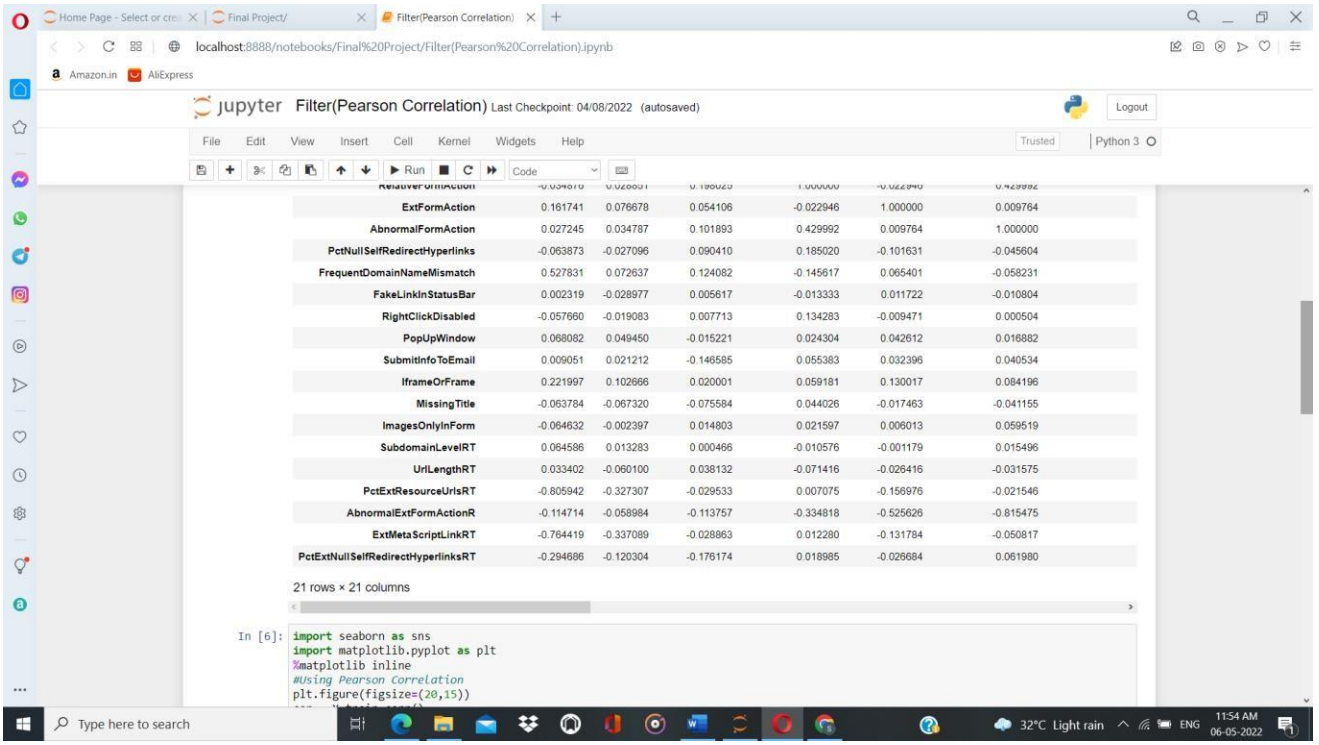
FILTER METHOD (PEARSON CORRELATION)



Filter(Pearson Correlation) Last Checkpoint: 04/08/2022 (autosaved)

```
In [2]: import pandas as pd
        import numpy as np

        from sklearn.model_selection import train_test_split
        from sklearn.feature_selection import VarianceThreshold

In [3]: dataset=pd.read_excel("Host_Cor_dataset.xlsx")
        dataset.shape

Out[3]: (10000, 22)

In [4]: X_train, X_test, y_train, y_test = train_test_split(
         dataset.drop(labels=["CLASS_LABEL"],axis = 1),
         dataset["CLASS_LABEL"],
         test_size=0.3,
         random_state=0)
        X_train.shape, X_test.shape

Out[4]: ((7000, 21), (3000, 21))

In [5]: X_train.corr()

Out[5]:
```

	PctExtResourceUrls	ExtFavicon	InsecureForms	RelativeFormAction	ExtFormAction	AbnormalFormAction	PctNullSelfRedire
PctExtResourceUrls	1.000000	0.333410	0.048050	-0.034876	0.161741	0.027245	
ExtFavicon	0.333410	1.000000	0.042739	0.028851	0.076678	0.034787	
InsecureForms	0.048050	0.042739	1.000000	0.198025	0.054106	0.101893	
RelativeFormAction	-0.034876	0.028851	0.198025	1.000000	-0.022946	0.429992	
ExtFormAction	0.161741	0.076678	0.054106	-0.022946	1.000000	0.009764	

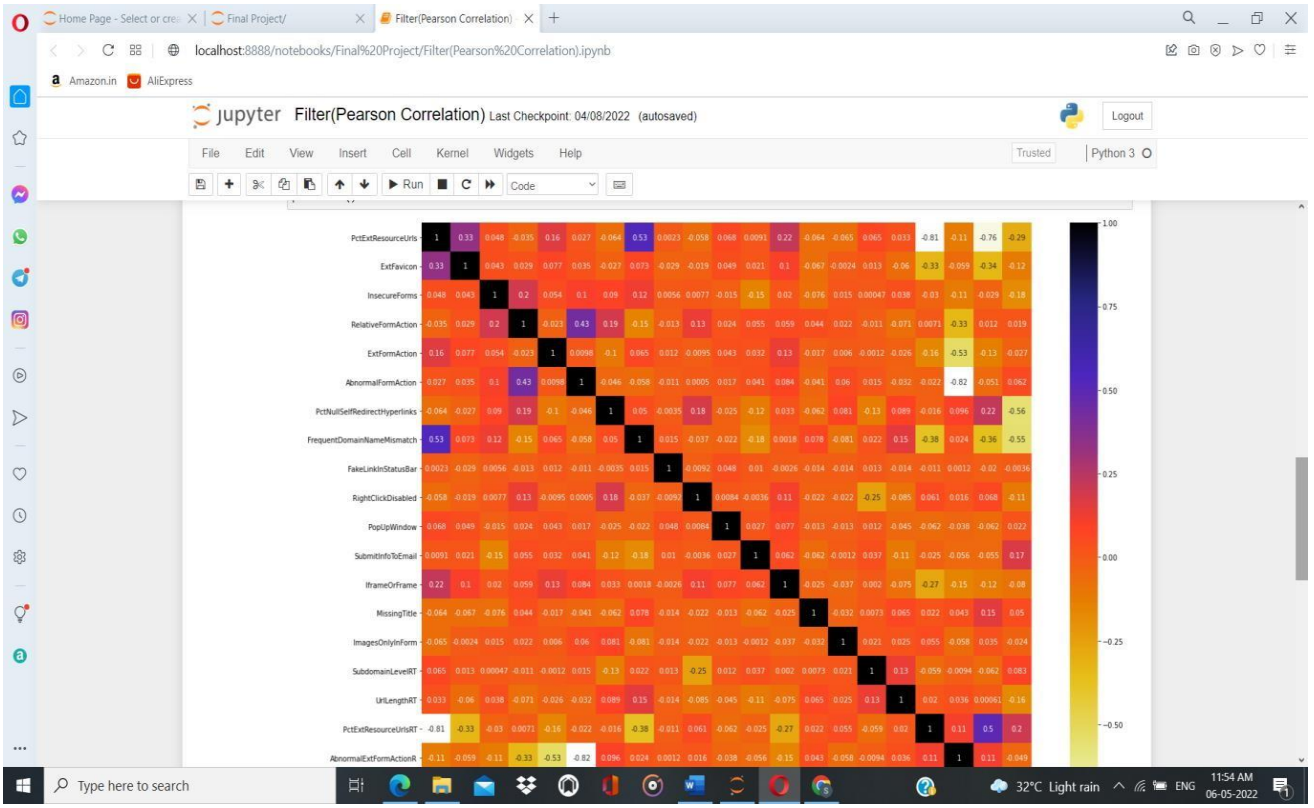


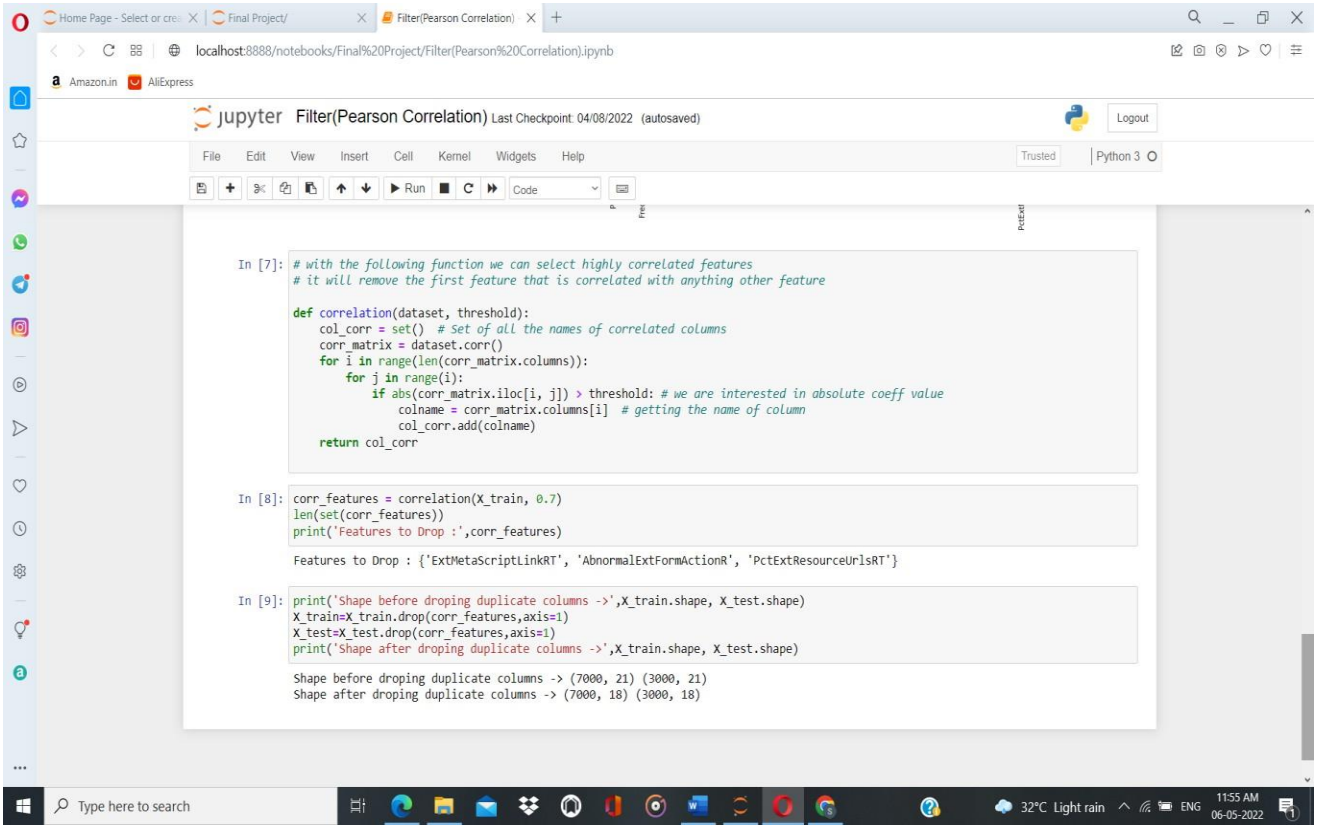
Filter(Pearson Correlation) Last Checkpoint: 04/08/2022 (autosaved)

ExtFormAction	0.161741	0.076678	0.054106	-0.022946	1.000000	0.009764
AbnormalFormAction	0.027245	0.034787	0.101893	0.429992	0.009764	1.000000
PctNullSelfRedirectHyperlinks	-0.063873	-0.027096	0.090410	0.185020	-0.101631	-0.045604
FrequentDomainNameMismatch	0.527831	0.072637	0.124082	-0.145617	0.065401	-0.058231
FakeLinkInStatusBar	0.002319	-0.028977	0.005617	-0.013333	0.011722	-0.010804
RightClickDisabled	-0.057660	-0.019083	0.007713	0.134283	-0.009471	0.000504
PopUpWindow	0.068082	0.049450	-0.015221	0.024304	0.042612	0.016882
SubmitInfoToEmail	0.009051	0.021212	-0.146585	0.055383	0.032396	0.040534
IframeOrFrame	0.221997	0.102666	0.020001	0.059181	0.130017	0.084196
MissingTitle	-0.063784	-0.067320	-0.075584	0.044026	-0.017463	-0.041155
ImagesOnlyInForm	-0.064632	-0.002397	0.014803	0.021597	0.006013	0.059519
SubdomainLevelRT	0.064586	0.013283	0.000466	-0.010576	-0.001179	0.015496
UrlLengthRT	0.033402	-0.060100	0.038132	-0.071416	-0.026416	-0.031575
PctExtResourceUrlsRT	-0.805942	-0.327307	-0.029533	0.007075	-0.156976	-0.021546
AbnormalExtFormActionR	-0.114714	-0.058984	-0.113757	-0.334818	-0.525626	-0.815475
ExtMetaScriptLinkRT	-0.764419	-0.337089	-0.028863	0.012280	-0.131784	-0.050817
PctExtNullSelfRedirectHyperlinksRT	-0.294686	-0.120304	-0.176174	0.018985	-0.026684	0.061980

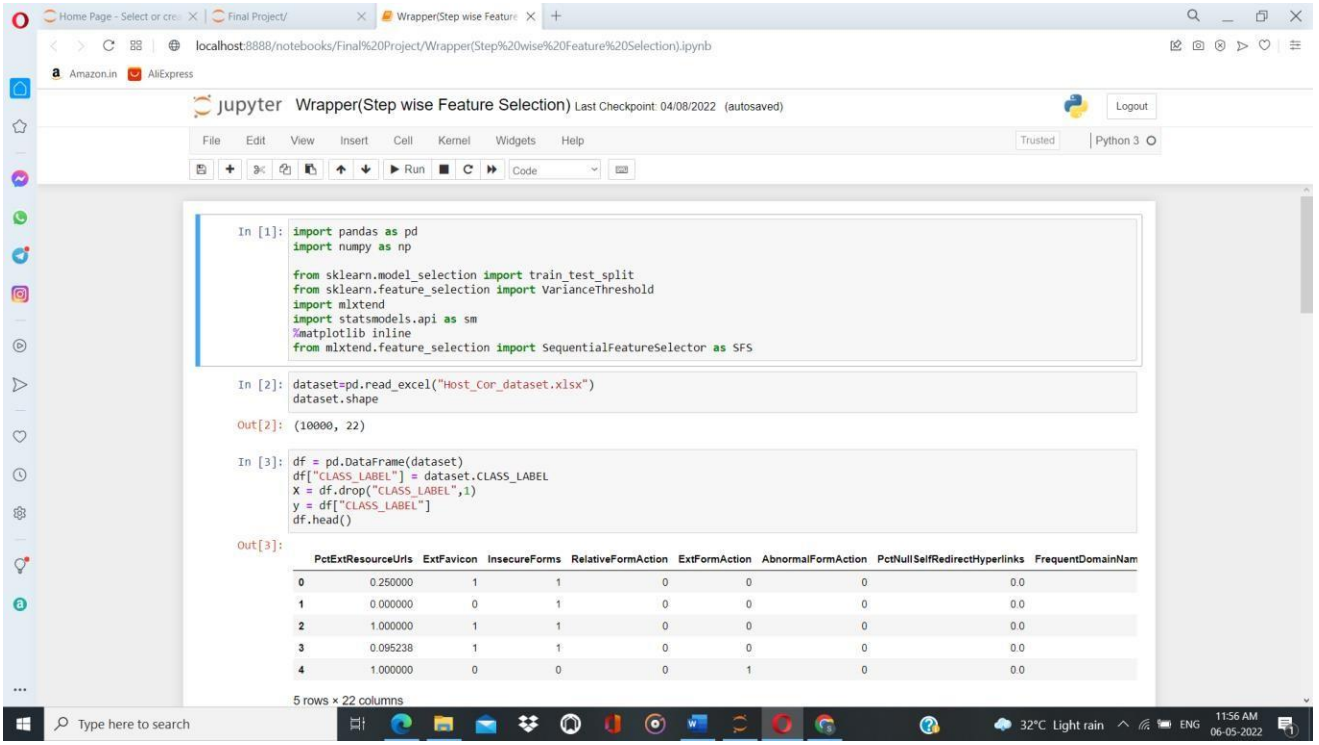
21 rows × 21 columns

```
In [6]: import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        #Using Pearson Correlation
        plt.figure(figsize=(20,15))
```


Home Page - Select or cre
Final Project/
Filter(Pearson Correlation)
localhost:8888/notebooks/Final%20Project/Filter(Pearson%20Correlation).ipynb
Amazon.in
AliExpress
jupyter Filter(Pearson Correlation) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
PctExtResourceUrls
ExtFavicon
InsecureForms
RelativeFormAction
ExtFormAction
AbnormalFormAction
PctNullSelfRedirectHyperlinks
FrequentDomainNameMismatch
FakeLinkInStatusBar
RightClickDisabled
PopUpWindow
SubmitInfoToEmail
IframeOrFrame
MissingTitle
ImagesOnlyInForm
SubdomainLevelRT
UrlLengthRT
PctExtResourceUrlsRT
AbnormalExtFormActionR
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Type here to search
32°C Light rain
ENG
11:54 AM
06-05-2022


Home Page - Select or cre
Final Project/
Filter(Pearson Correlation)
localhost:8888/notebooks/Final%20Project/Filter(Pearson%20Correlation).ipynb
Amazon.in
AliExpress
jupyter Filter(Pearson Correlation) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [7]:
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything other feature
def correlation(dataset, threshold):
col_corr = set() # Set of all the names of correlated columns
corr_matrix = dataset.corr()
for i in range(len(corr_matrix.columns)):
for j in range(i):
if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute coeff value
colname = corr_matrix.columns[i] # getting the name of column
col_corr.add(colname)
return col_corr
In [8]:
corr_features = correlation(X_train, 0.7)
len(set(corr_features))
print('Features to Drop :',corr_features)
Features to Drop : {'ExtMetaScriptLinkRT', 'AbnormalExtFormActionR', 'PctExtResourceUrlsRT'}
In [9]:
print('Shape before droping duplicate columns ->',X_train.shape, X_test.shape)
X_train=X_train.drop(corr_features,axis=1)
X_test=X_test.drop(corr_features,axis=1)
print('Shape after droping duplicate columns ->',X_train.shape, X_test.shape)
Shape before droping duplicate columns -> (7000, 21) (3000, 21)
Shape after droping duplicate columns -> (7000, 18) (3000, 18)
Type here to search
32°C Light rain
ENG
11:55 AM
06-05-2022

WRAPPER METHOD (STEPWISE FEATURE SELECTION)



Home Page - Select or cre | Final Project/ | Wrapper(Step wise Feature

localhost:8888/notebooks/Final%20Project/Wrapper(Step%20wise%20Feature%20Selection).ipynb

Amazon.in AliExpress

jupyter Wrapper(Step wise Feature Selection) Last Checkpoint: 04/08/2022 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [1]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
import mlxtend
import statsmodels.api as sm
%matplotlib inline
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

In [2]:
```
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape
```

Out[2]: (10000, 22)

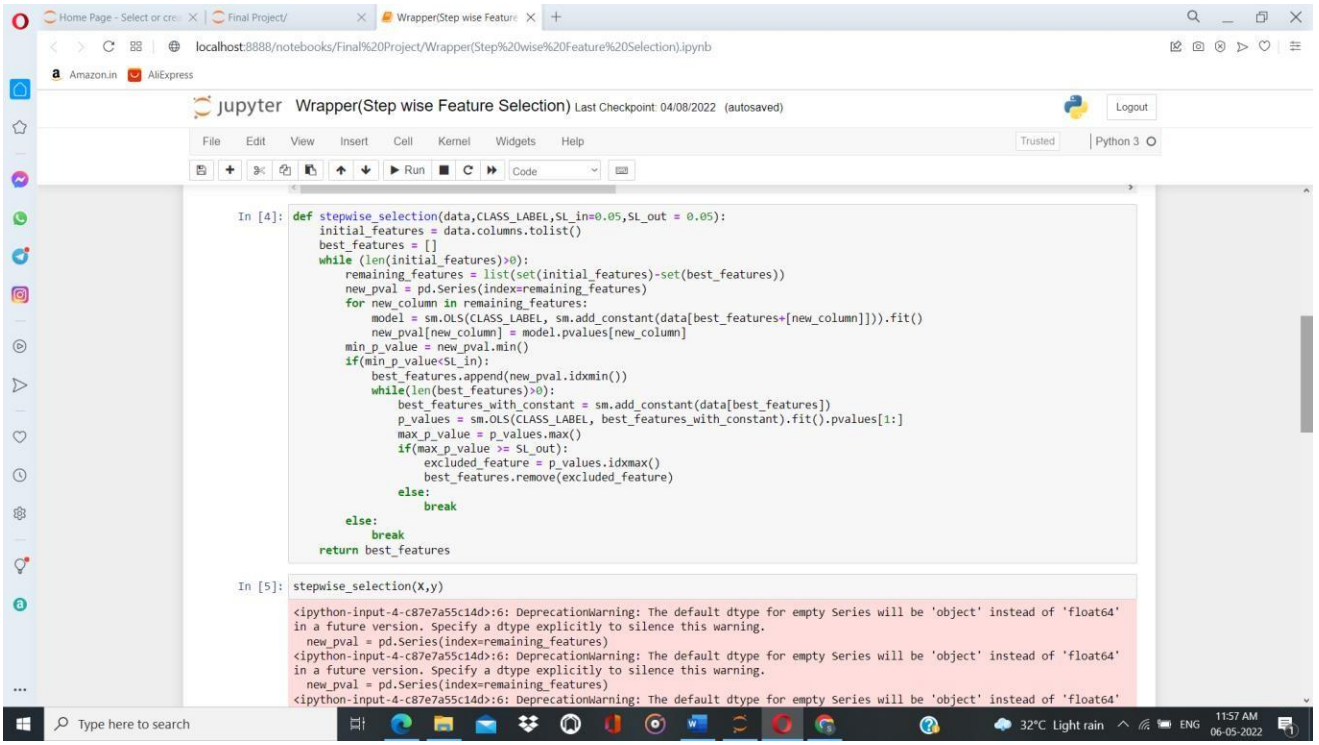
In [3]:
```
df = pd.DataFrame(dataset)
df["CLASS_LABEL"] = dataset.CLASS_LABEL
X = df.drop("CLASS_LABEL",1)
y = df["CLASS_LABEL"]
df.head()
```

Out[3]:

	PctExtResourceUrls	ExtFavicon	InsecureForms	RelativeFormAction	ExtFormAction	AbnormalFormAction	PctNullSelfRedirectHyperlinks	FrequentDomainNam
0	0.250000	1	1	0	0	0	0.0	
1	0.000000	0	1	0	0	0	0.0	
2	1.000000	1	1	0	0	0	0.0	
3	0.095238	1	1	0	0	0	0.0	
4	1.000000	0	0	0	1	0	0.0	

5 rows × 22 columns

Type here to search 32°C Light rain ENG 11:56 AM 06-05-2022



Home Page - Select or cre | Final Project/ | Wrapper(Step wise Feature

localhost:8888/notebooks/Final%20Project/Wrapper(Step%20wise%20Feature%20Selection).ipynb

Amazon.in AliExpress

jupyter Wrapper(Step wise Feature Selection) Last Checkpoint: 04/08/2022 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

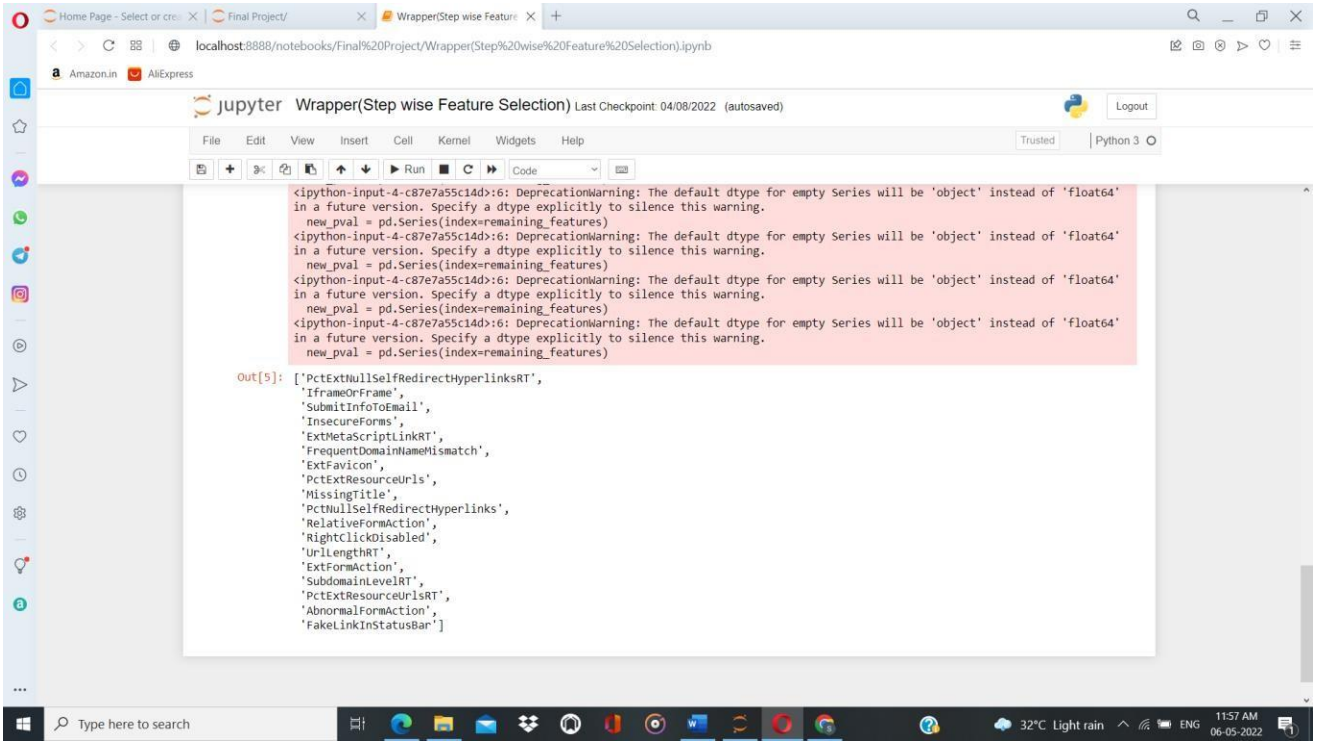
Run Code

In [4]:
```
def stepwise_selection(data,CLASS_LABEL,SL_in=0.05,SL_out = 0.05):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(CLASS_LABEL, sm.add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<SL_in):
            best_features.append(new_pval.idxmin())
            while(len(best_features)>0):
                best_features_with_constant = sm.add_constant(data[best_features])
                p_values = sm.OLS(CLASS_LABEL, best_features_with_constant).fit().pvalues[1:]
                max_p_value = p_values.max()
                if(max_p_value >= SL_out):
                    excluded_feature = p_values.idxmax()
                    best_features.remove(excluded_feature)
                else:
                    break
        else:
            break
    return best_features
```

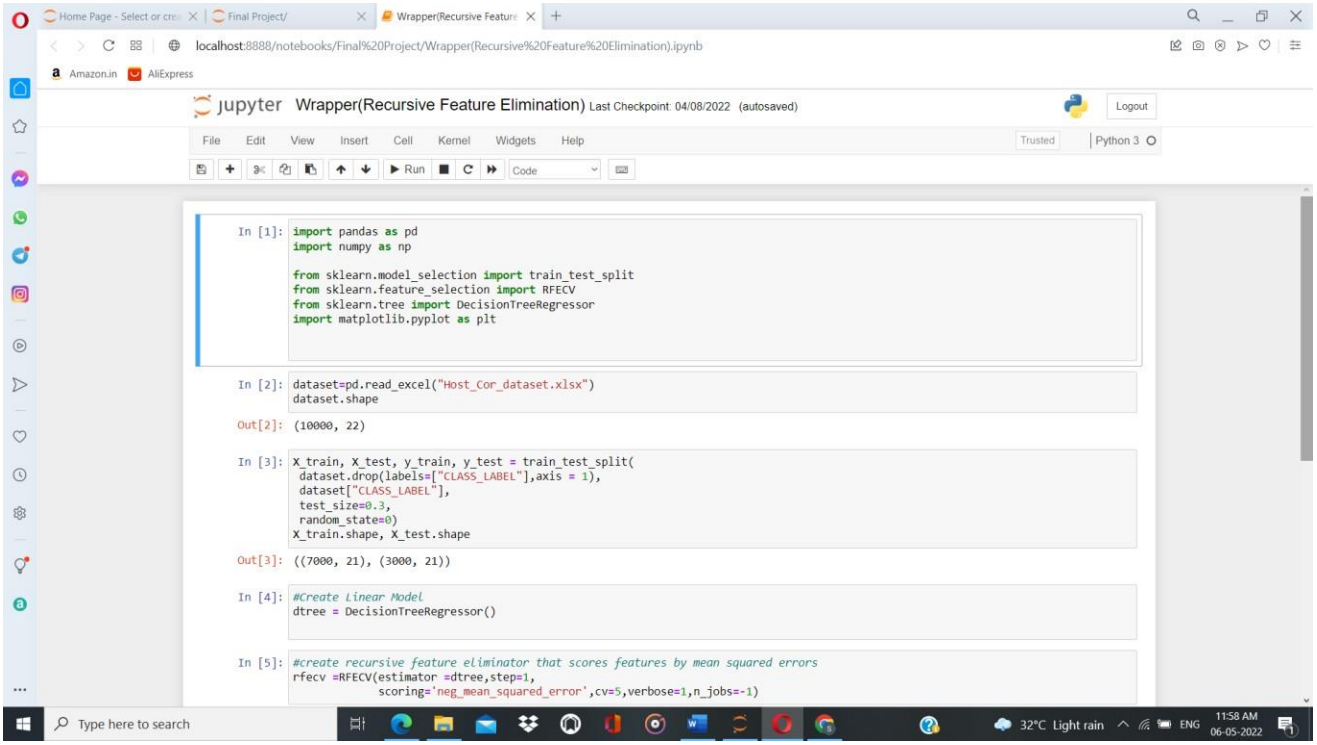
In [5]:
```
stepwise_selection(X,y)
```

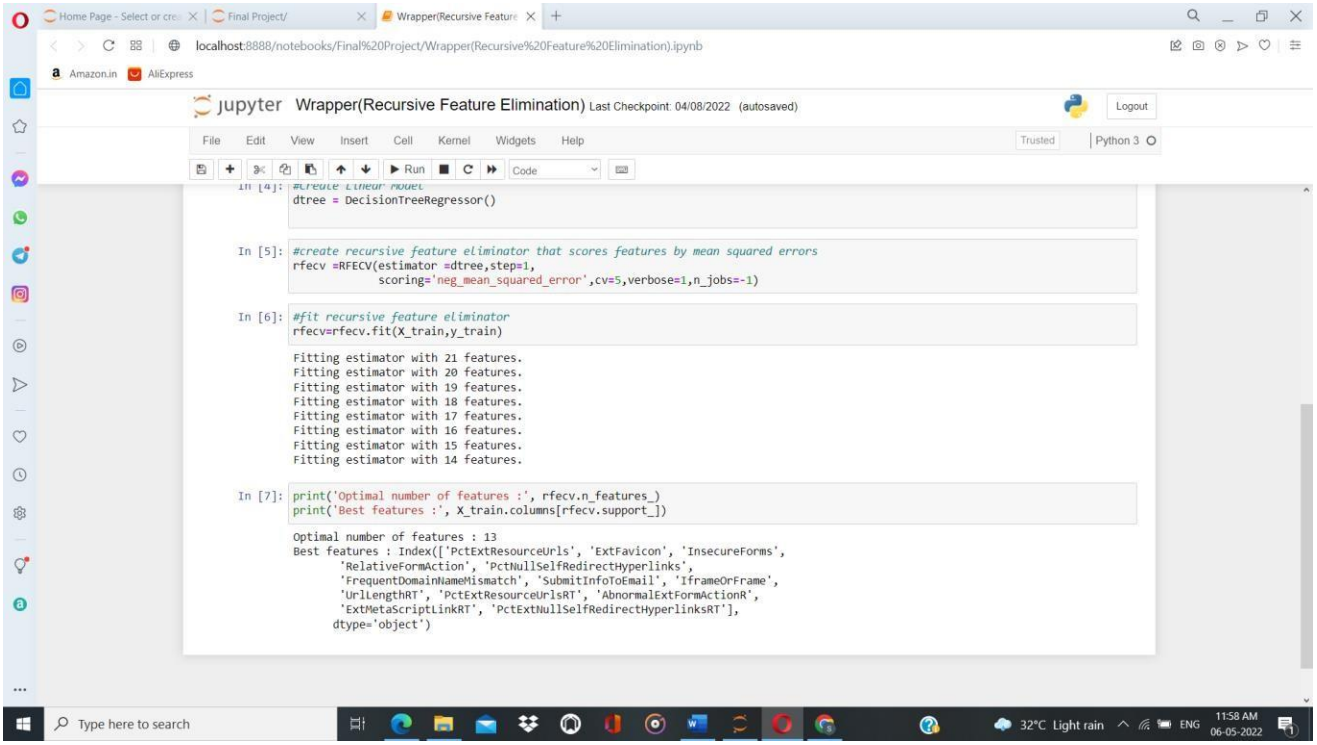
```
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64'
in a future version. Specify a dtype explicitly to silence this warning.
  new_pval = pd.Series(index=remaining_features)
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64'
in a future version. Specify a dtype explicitly to silence this warning.
  new_pval = pd.Series(index=remaining_features)
<ipython-input-4-c87e7a55c14d>:6: DeprecationWarning: The default dtype for empty Series will be 'object' instead of 'float64'
```

Type here to search 32°C Light rain ENG 11:57 AM 06-05-2022



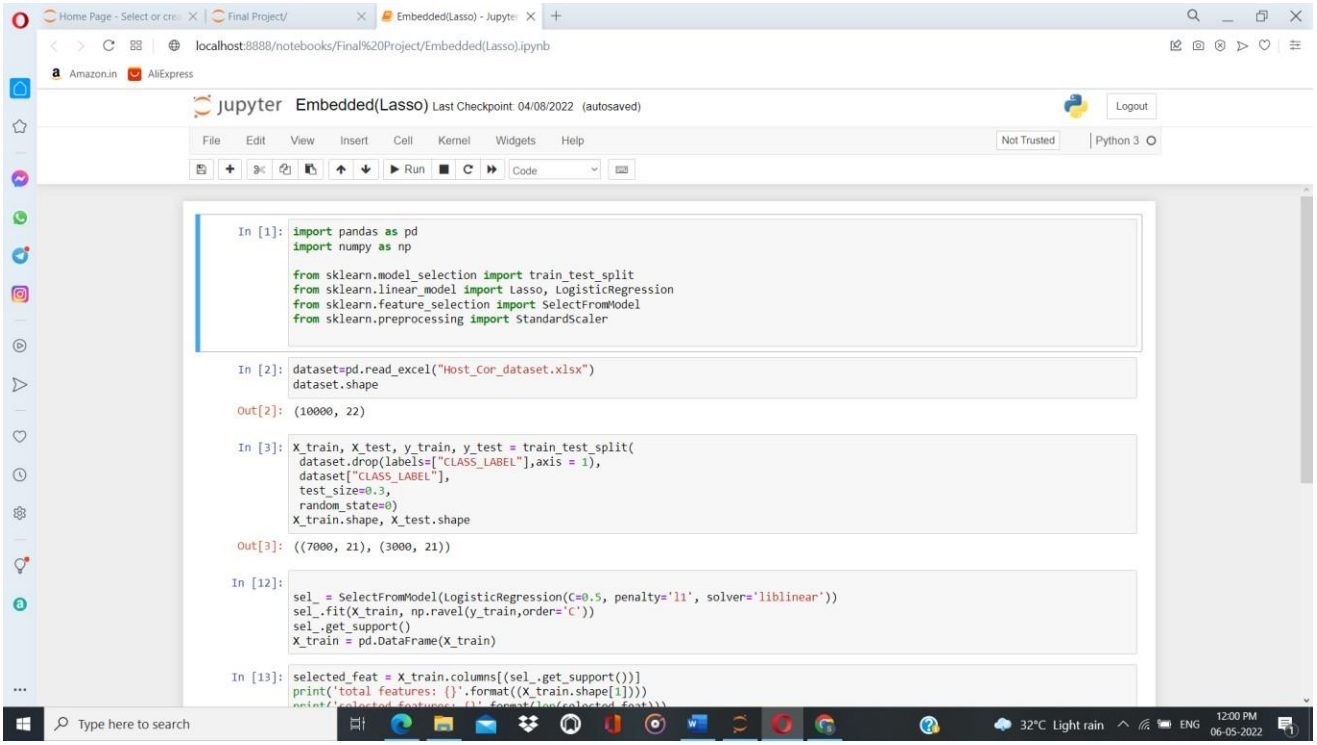
WRAPPER METHOD (RECURSIVE FEATURE ELIMINATION)



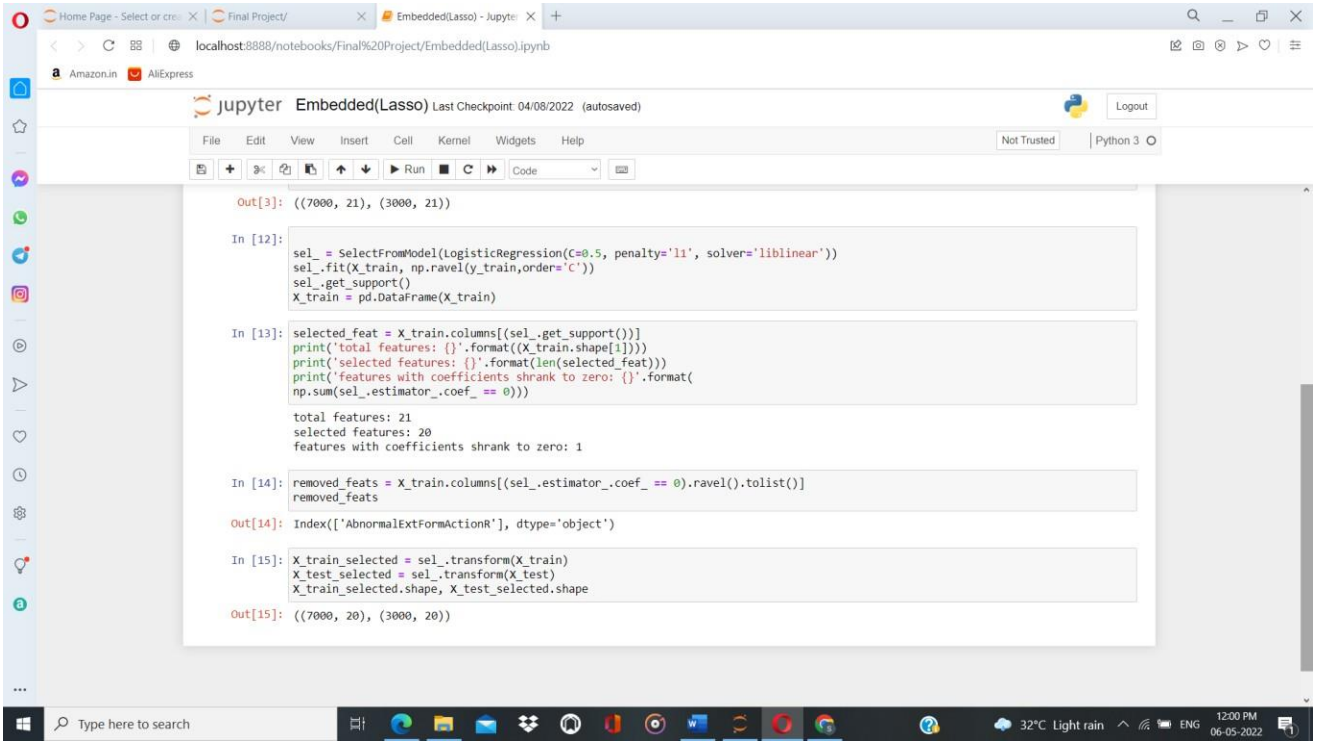


```
In [4]: #create linear model
        dtree = DecisionTreeRegressor()

In [5]: #create recursive feature eliminator that scores features by mean squared errors
        rfecv =RFECV(estimator =dtree,step=1,
                     scoring='neg_mean_squared_error',cv=5,verbose=1,n_jobs=-1)

In [6]: #fit recursive feature eliminator
        rfecv=rfecv.fit(X_train,y_train)

        Fitting estimator with 21 features.
        Fitting estimator with 20 features.
        Fitting estimator with 19 features.
        Fitting estimator with 18 features.
        Fitting estimator with 17 features.
        Fitting estimator with 16 features.
        Fitting estimator with 15 features.
        Fitting estimator with 14 features.

In [7]: print('Optimal number of features :', rfecv.n_features_)
        print('Best features :', X_train.columns[rfecv.support_])

        Optimal number of features : 13
        Best features : Index(['PctExtResourceUrls', 'ExtFavicon', 'InsecureForms',
               'RelativeFormAction', 'PctNullSelfRedirectHyperlinks',
               'FrequentDomainNameMismatch', 'SubmitInfoToEmail', 'IframeOrFrame',
               'UrlLengthRT', 'PctExtResourceUrlsRT', 'AbnormalExtFormActionR',
               'ExtMetaScriptLinkRT', 'PctExtNullSelfRedirectHyperlinksRT'],
              dtype='object')
```

EMBEDDED METHOD (LASSO REGULARIZATION)

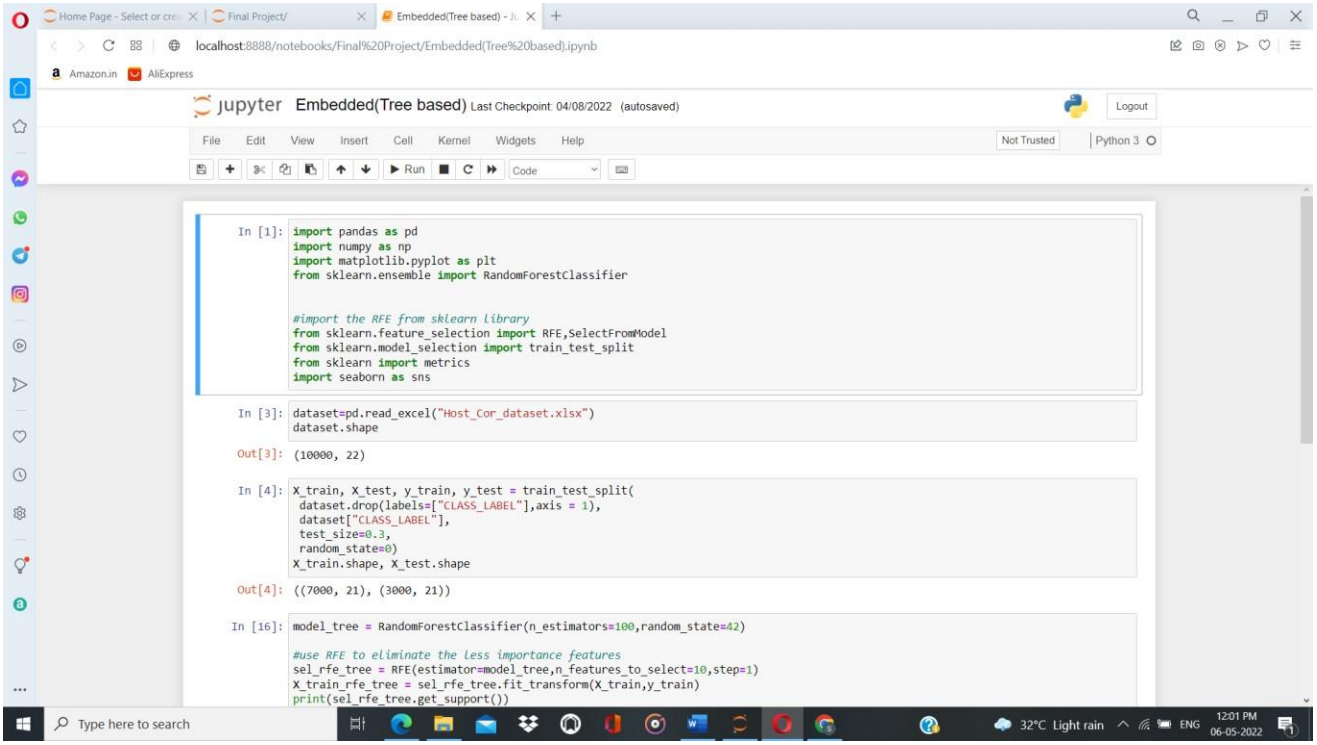


```
In [1]: import pandas as pd
        import numpy as np

        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import Lasso, LogisticRegression
        from sklearn.feature_selection import SelectFromModel
        from sklearn.preprocessing import StandardScaler

In [2]: dataset=pd.read_excel("Host_Cor_dataset.xlsx")
        dataset.shape

Out[2]: (10000, 22)

In [3]: X_train, X_test, y_train, y_test = train_test_split(
         dataset.drop(labels=["CLASS_LABEL"],axis = 1),
         dataset["CLASS_LABEL"],
         test_size=0.3,
         random_state=0)
        X_train.shape, X_test.shape

Out[3]: ((7000, 21), (3000, 21))

In [12]:
        sel_ = SelectFromModel(LogisticRegression(C=0.5, penalty='l1', solver='liblinear'))
        sel_.fit(X_train, np.ravel(y_train,order='C'))
        sel_.get_support()
        X_train = pd.DataFrame(X_train)

In [13]: selected_feat = X_train.columns[(sel_.get_support())]
         print('total features: {}'.format((X_train.shape[1])))
```

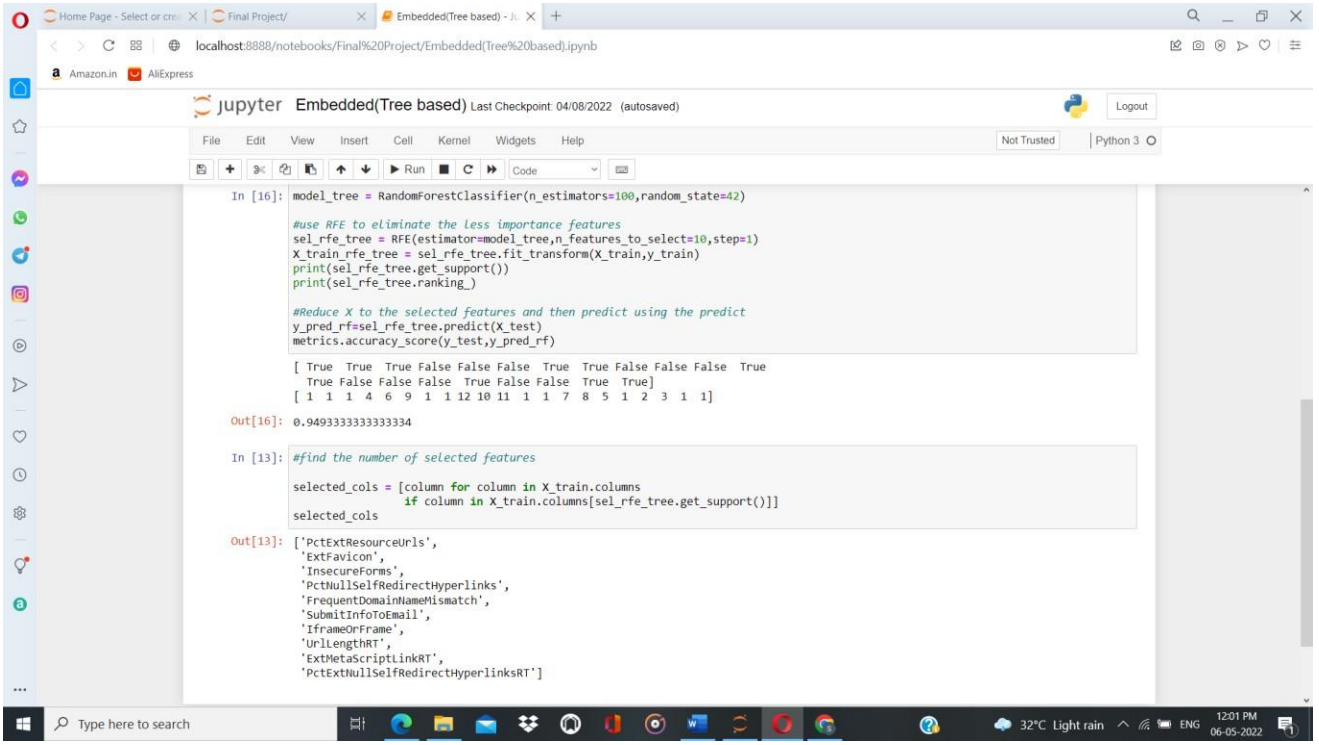


```
Out[3]: ((7000, 21), (3000, 21))

In [12]:
sel_ = SelectFromModel(LogisticRegression(C=0.5, penalty='l1', solver='liblinear'))
sel_.fit(X_train, np.ravel(y_train,order='C'))
sel_.get_support()
X_train = pd.DataFrame(X_train)

In [13]:
selected_feat = X_train.columns[(sel_.get_support())]
print('total features: {}'.format((X_train.shape[1])))
print('selected features: {}'.format(len(selected_feat)))
print('features with coefficients shrank to zero: {}'.format(
np.sum(sel_.estimator_.coef_ == 0)))

total features: 21
selected features: 20
features with coefficients shrank to zero: 1

In [14]:
removed_feats = X_train.columns[(sel_.estimator_.coef_ == 0).ravel().tolist()]
removed_feats

Out[14]: Index(['AbnormalExtFormActionR'], dtype='object')

In [15]:
X_train_selected = sel_.transform(X_train)
X_test_selected = sel_.transform(X_test)
X_train_selected.shape, X_test_selected.shape

Out[15]: ((7000, 20), (3000, 20))
```

EMBEDDED METHOD (TREE BASED)

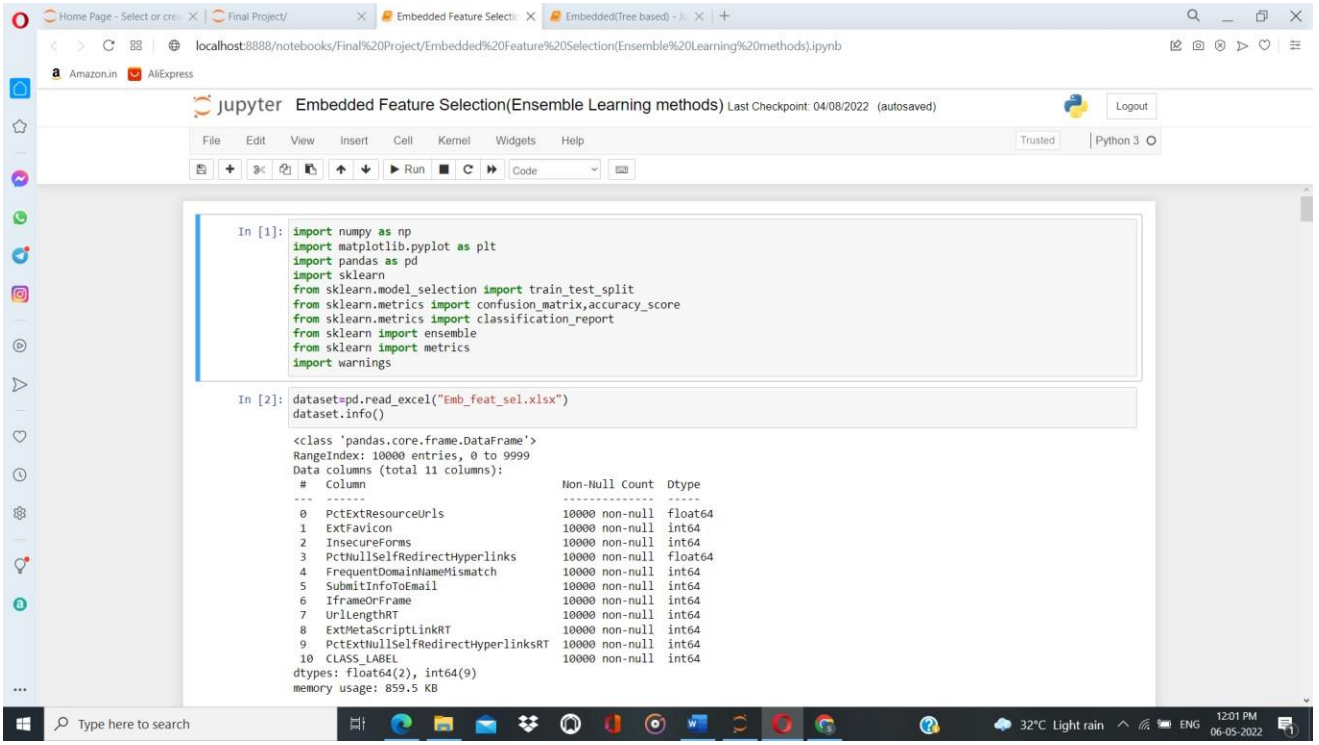


```
In [1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier


#import the RFE from sklearn library
from sklearn.feature_selection import RFE,SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns

In [3]:
dataset=pd.read_excel("Host_Cor_dataset.xlsx")
dataset.shape

Out[3]: (10000, 22)

In [4]:
X_train, X_test, y_train, y_test = train_test_split(
 dataset.drop(labels=["CLASS_LABEL"],axis = 1),
 dataset["CLASS_LABEL"],
 test_size=0.3,
 random_state=0)
X_train.shape, X_test.shape

Out[4]: ((7000, 21), (3000, 21))

In [16]:
model_tree = RandomForestClassifier(n_estimators=100,random_state=42)

#use RFE to eliminate the less importance features
sel_rfe_tree = RFE(estimator=model_tree,n_features_to_select=10,step=1)
X_train_rfe_tree = sel_rfe_tree.fit_transform(X_train,y_train)
print(sel_rfe_tree.get_support())
```

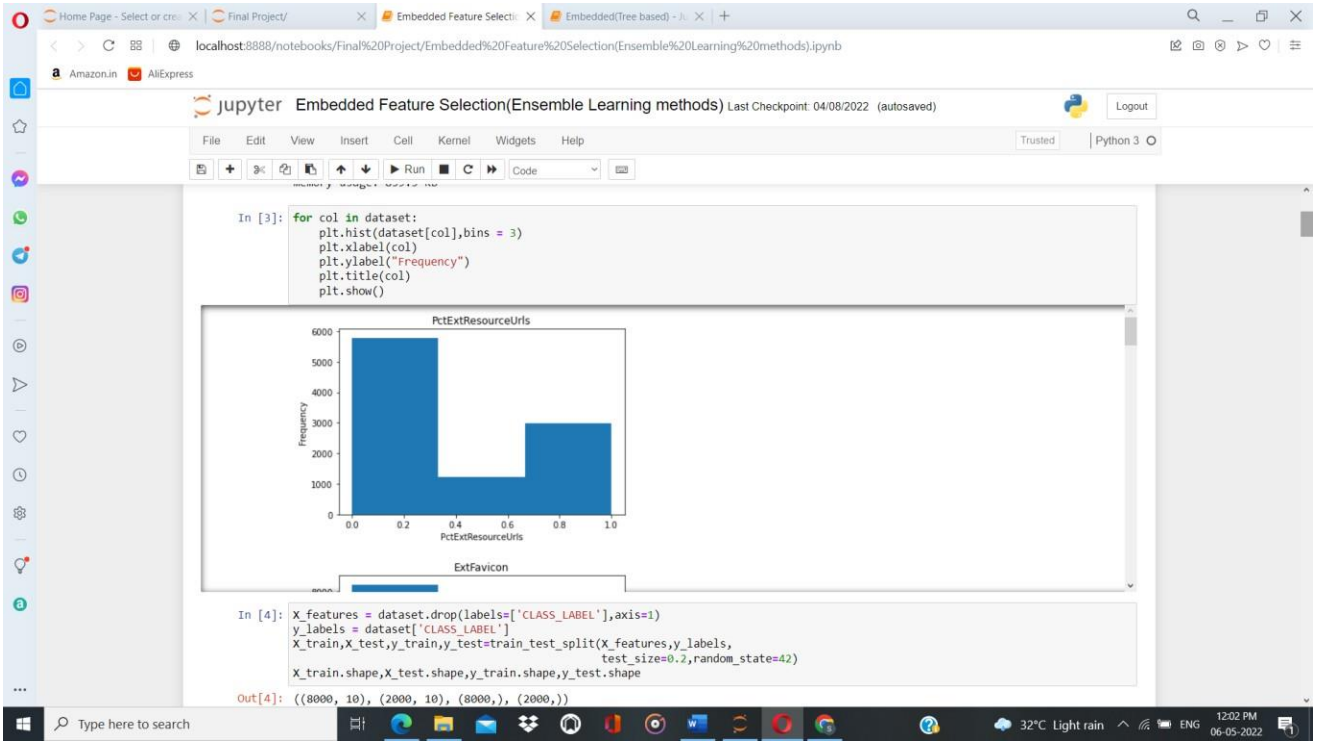


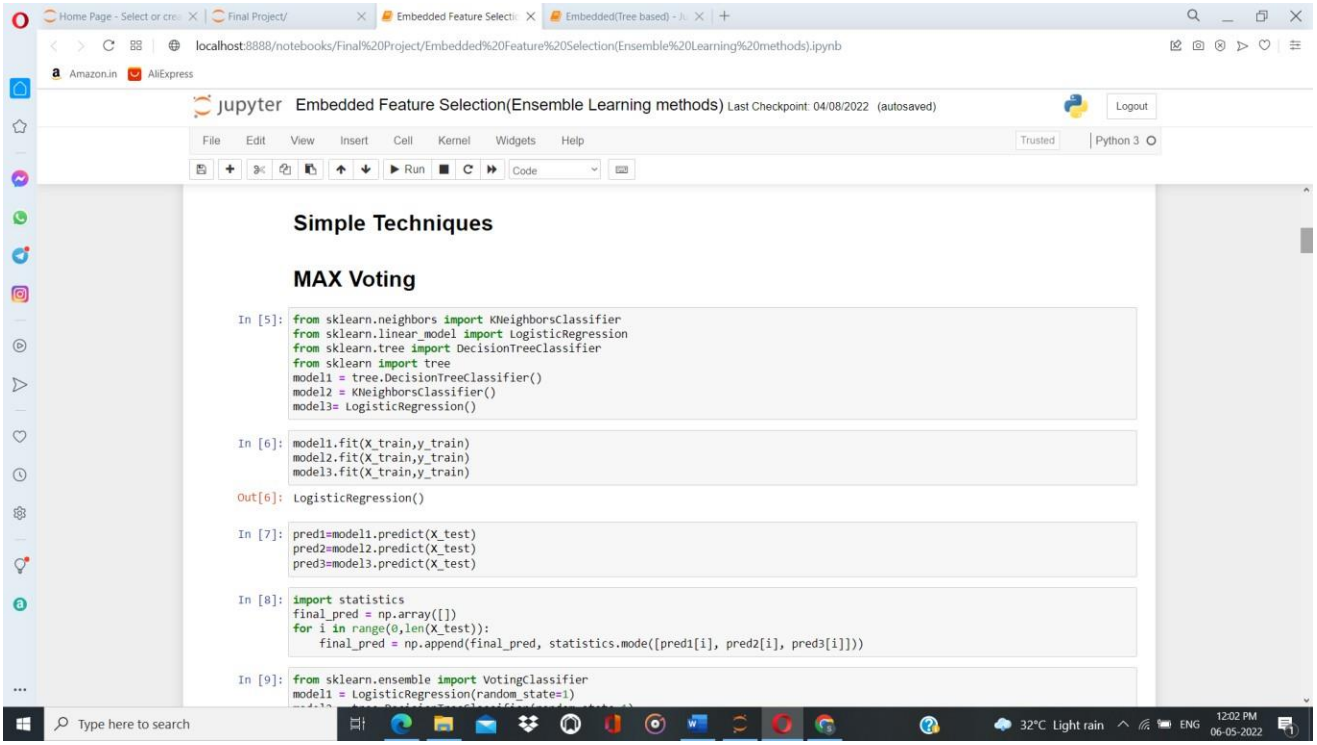
```
In [16]: model_tree = RandomForestClassifier(n_estimators=100,random_state=42)

         #use RFE to eliminate the less importance features
         sel_rfe_tree = RFE(estimator=model_tree,n_features_to_select=10,step=1)
         X_train_rfe_tree = sel_rfe_tree.fit_transform(X_train,y_train)
         print(sel_rfe_tree.get_support())
         print(sel_rfe_tree.ranking_)

         #Reduce X to the selected features and then predict using the predict
         y_pred_rf=sel_rfe_tree.predict(X_test)
         metrics.accuracy_score(y_test,y_pred_rf)

         [ True  True  True False False False  True  True False False False  True
           True False False False  True False False  True  True]
         [ 1  1  1  4  6  9  1  1 12 10 11  1  1  7  8  5  1  2  3  1  1]

Out[16]: 0.9493333333333334

In [13]: #find the number of selected features

         selected_cols = [column for column in X_train.columns
                          if column in X_train.columns[sel_rfe_tree.get_support()]]
         selected_cols

Out[13]: ['PctExtResourceUrls',
          'ExtFavicon',
          'InsecureForms',
          'PctNullSelfRedirectHyperlinks',
          'FrequentDomainNameMismatch',
          'SubmitInfoToEmail',
          'IframeOrFrame',
          'UrlLengthRT',
          'ExtMetaScriptLinkRT',
          'PctExtNullSelfRedirectHyperlinksRT']
```

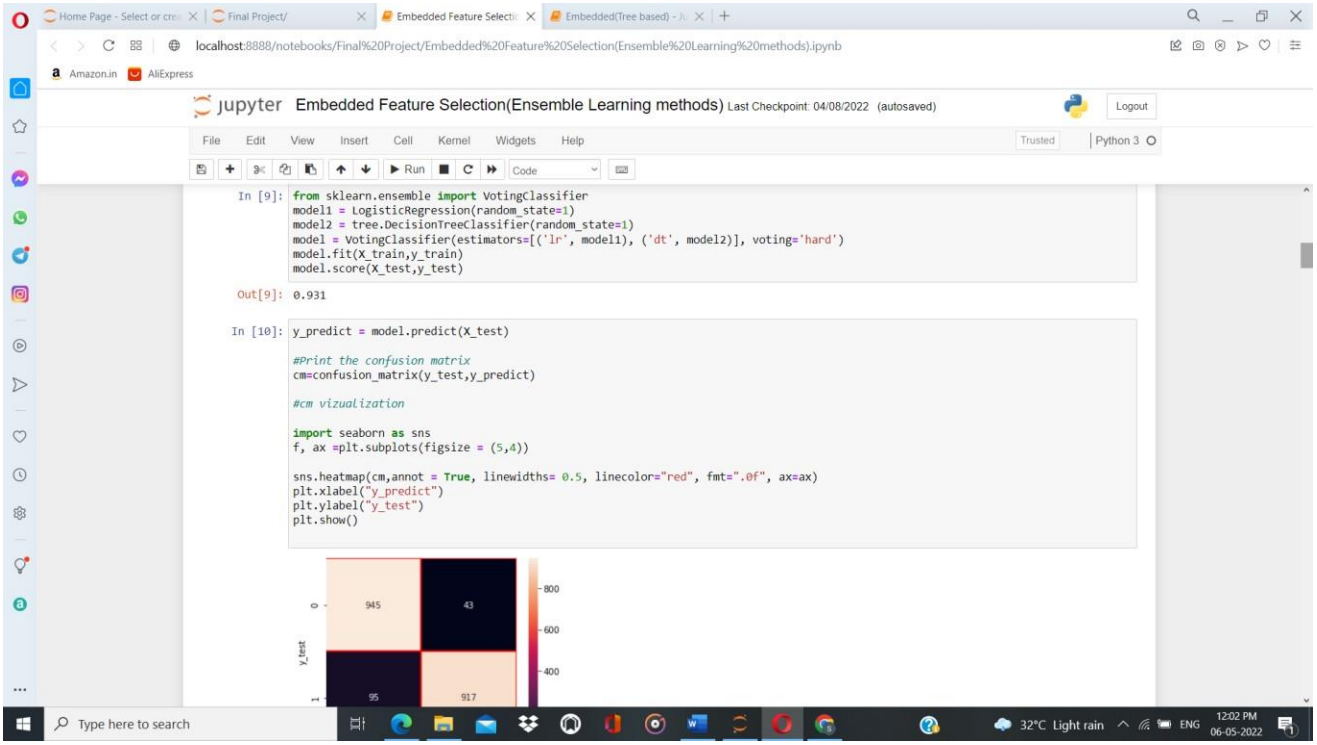
MODEL BUILDING

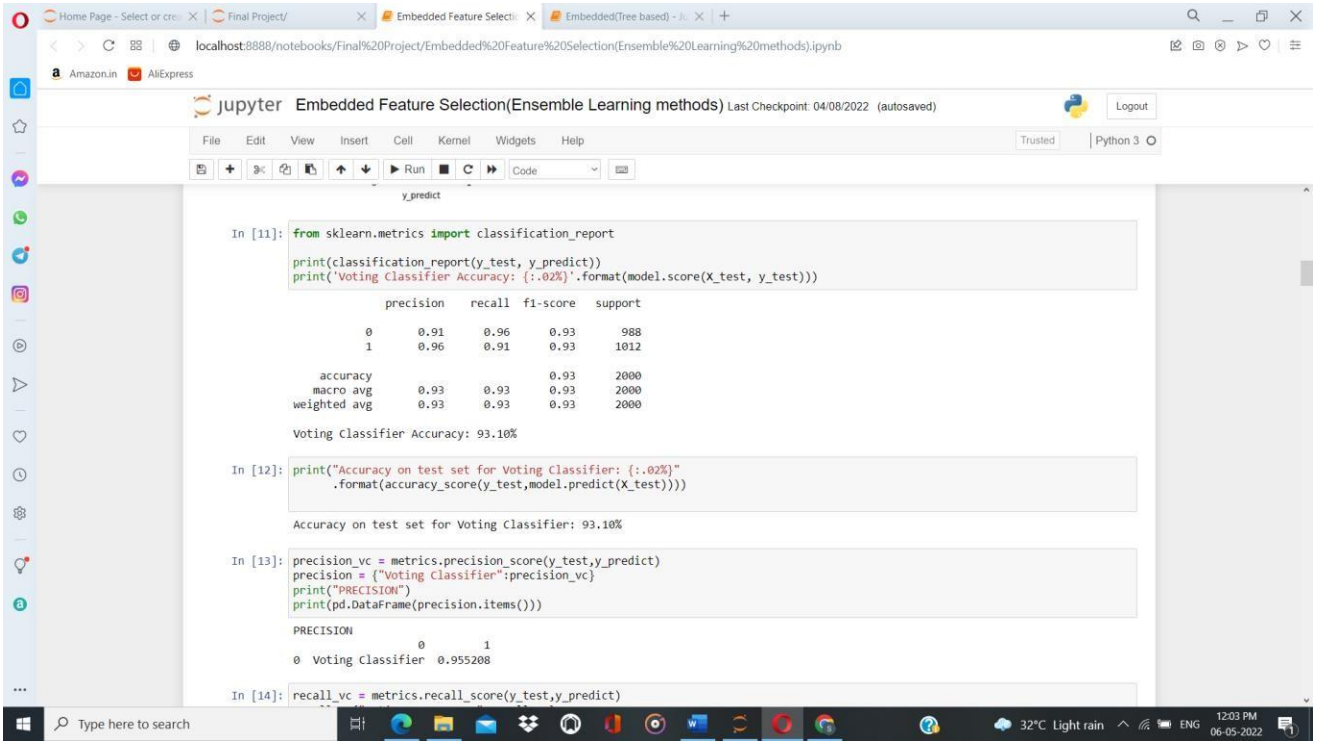


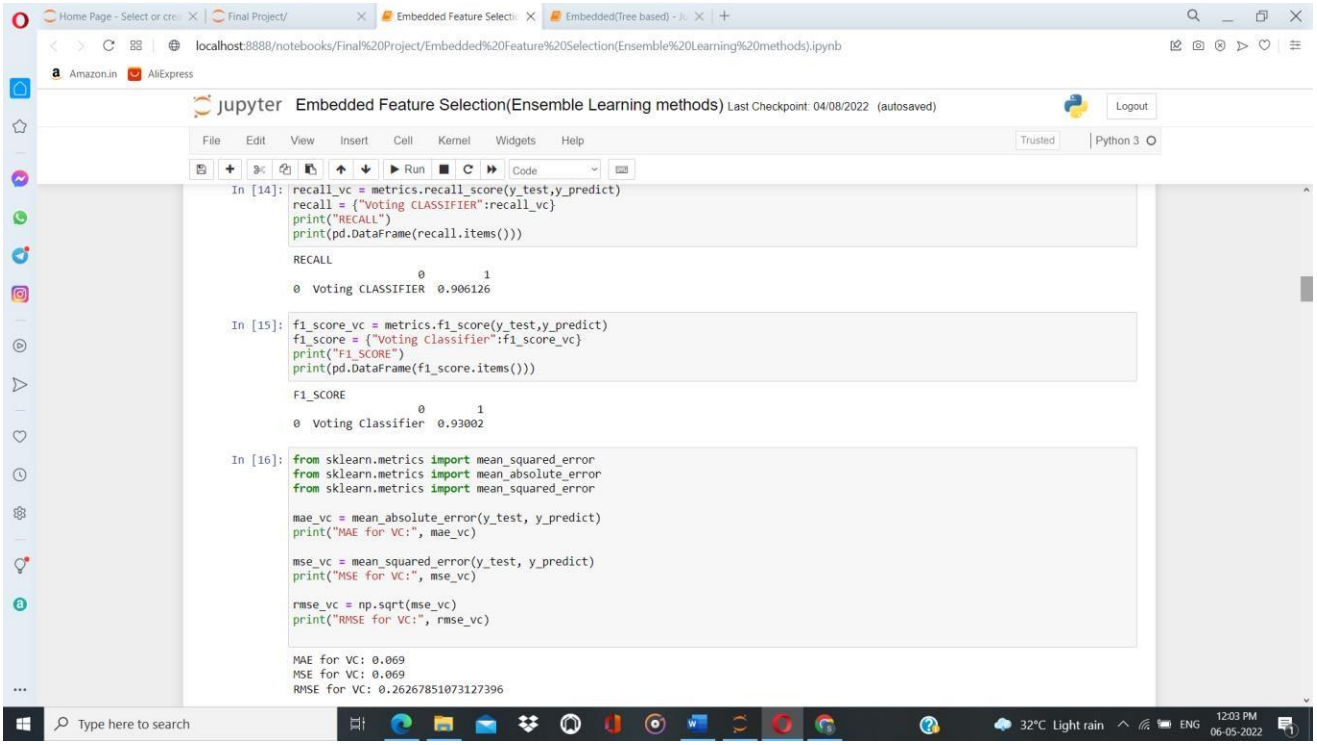
```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        import sklearn
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix,accuracy_score
        from sklearn.metrics import classification_report
        from sklearn import ensemble
        from sklearn import metrics
        import warnings

In [2]: dataset=pd.read_excel("Emb_feat_sel.xlsx")
        dataset.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 10000 entries, 0 to 9999
        Data columns (total 11 columns):
         #   Column                          Non-Null Count  Dtype
        ---  ------                          --------------  -----
         0   PctExtResourceUrls              10000 non-null  float64
         1   ExtFavicon                      10000 non-null  int64
         2   InsecureForms                   10000 non-null  int64
         3   PctNullSelfRedirectHyperlinks   10000 non-null  float64
         4   FrequentDomainNameMismatch      10000 non-null  int64
         5   SubmitInfoToEmail               10000 non-null  int64
         6   IframeOrFrame                   10000 non-null  int64
         7   UrlLengthRT                     10000 non-null  int64
         8   ExtMetaScriptLinkRT             10000 non-null  int64
         9   PctExtNullSelfRedirectHyperlinksRT  10000 non-null  int64
         10  CLASS_LABEL                     10000 non-null  int64
        dtypes: float64(2), int64(9)
        memory usage: 859.5 KB
```

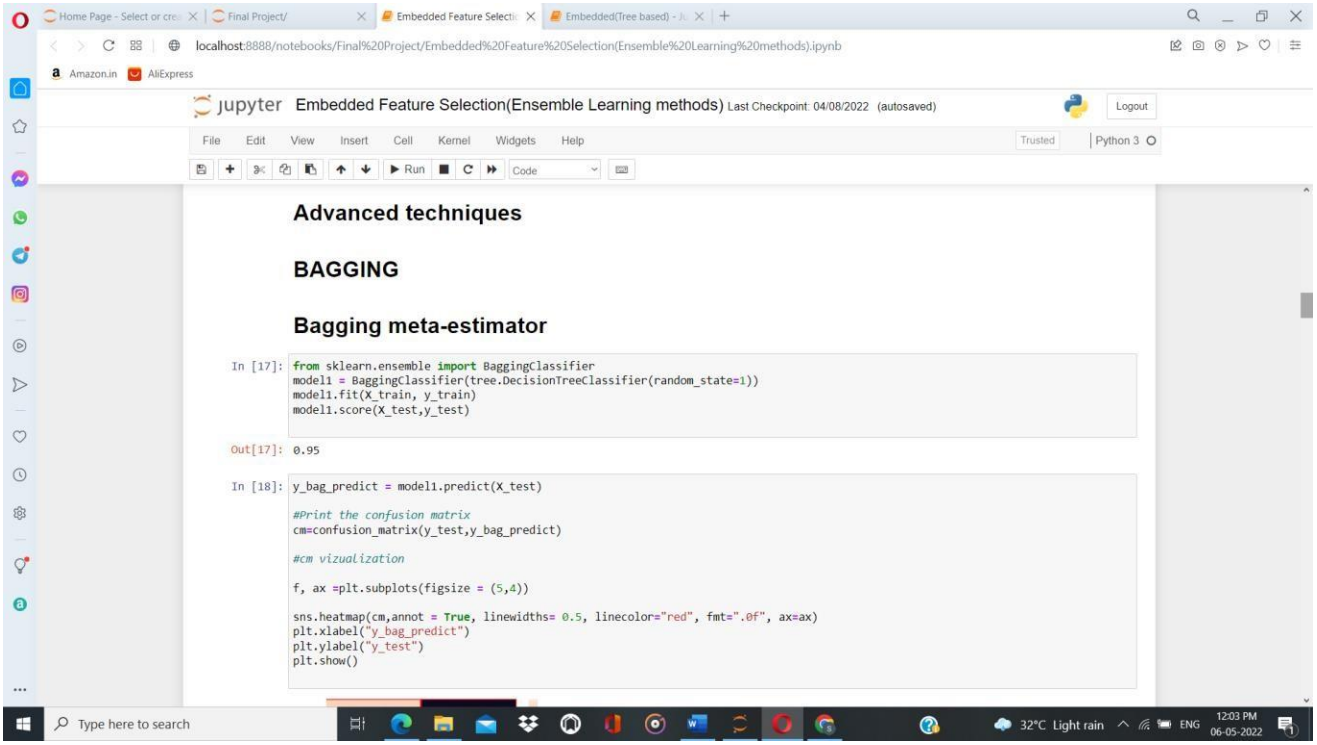

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
In [3]:
for col in dataset:
plt.hist(dataset[col],bins = 3)
plt.xlabel(col)
plt.ylabel("Frequency")
plt.title(col)
plt.show()
PctExtResourceUrls
Frequency
PctExtResourceUrls
ExtFavicon
In [4]:
X_features = dataset.drop(labels=['CLASS_LABEL'],axis=1)
y_labels = dataset['CLASS_LABEL']
X_train,X_test,y_train,y_test=train_test_split(X_features,y_labels,
test_size=0.2,random_state=42)
X_train.shape,X_test.shape,y_train.shape,y_test.shape
Out[4]: ((8000, 10), (2000, 10), (8000,), (2000,))

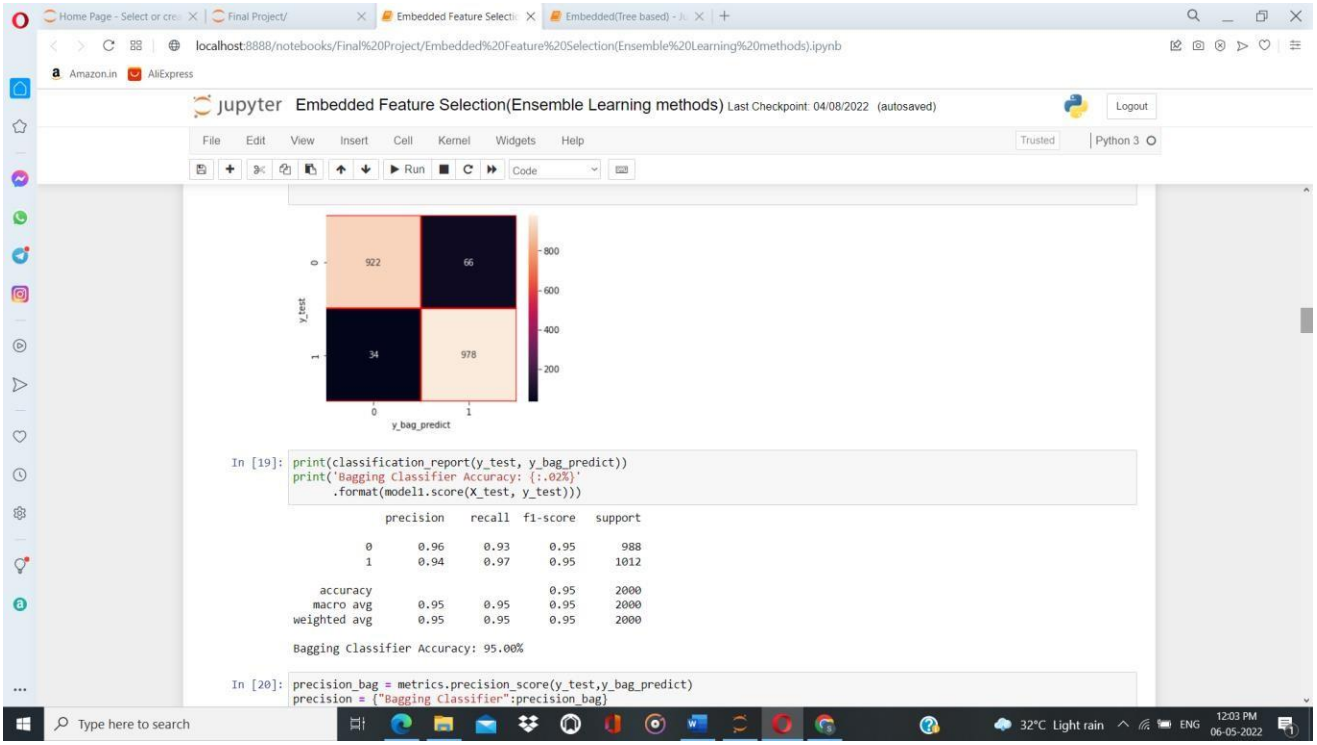

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
Simple Techniques
MAX Voting
In [5]:
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()
In [6]:
model1.fit(X_train,y_train)
model2.fit(X_train,y_train)
model3.fit(X_train,y_train)
Out[6]: LogisticRegression()
In [7]:
pred1=model1.predict(X_test)
pred2=model2.predict(X_test)
pred3=model3.predict(X_test)
In [8]:
import statistics
final_pred = np.array([])
for i in range(0,len(X_test)):
final_pred = np.append(final_pred, statistics.mode([pred1[i], pred2[i], pred3[i]]))
In [9]:
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
In [9]:
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
model.fit(X_train,y_train)
model.score(X_test,y_test)
Out[9]: 0.931
In [10]:
y_predict = model.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_predict)
#cm vizualization
import seaborn as sns
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_predict")
plt.ylabel("y_test")
plt.show()
945
43
95
917
y_test
800
600
400


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
In [11]:
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
print('Voting Classifier Accuracy: {:.02%}'.format(model.score(X_test, y_test)))
precision recall f1-score support
0 0.91 0.96 0.93 988
1 0.96 0.91 0.93 1012
accuracy 0.93 2000
macro avg 0.93 0.93 0.93 2000
weighted avg 0.93 0.93 0.93 2000
Voting Classifier Accuracy: 93.10%
In [12]:
print("Accuracy on test set for Voting Classifier: {:.02%}"
.format(accuracy_score(y_test,model.predict(X_test))))
Accuracy on test set for Voting Classifier: 93.10%
In [13]:
precision_vc = metrics.precision_score(y_test,y_predict)
precision = {"Voting Classifier":precision_vc}
print("PRECISION")
print(pd.DataFrame(precision.items()))
PRECISION
0 1
0 Voting Classifier 0.955208
In [14]:
recall_vc = metrics.recall_score(y_test,y_predict)


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
In [14]:
recall_vc = metrics.recall_score(y_test,y_predict)
recall = {"Voting CLASSIFIER":recall_vc}
print("RECALL")
print(pd.DataFrame(recall.items()))
RECALL
0 1
0 Voting CLASSIFIER 0.906126
In [15]:
f1_score_vc = metrics.f1_score(y_test,y_predict)
f1_score = {"Voting Classifier":f1_score_vc}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
F1_SCORE
0 1
0 Voting Classifier 0.93002
In [16]:
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
mae_vc = mean_absolute_error(y_test, y_predict)
print("MAE for VC:", mae_vc)
mse_vc = mean_squared_error(y_test, y_predict)
print("MSE for VC:", mse_vc)
rmse_vc = np.sqrt(mse_vc)
print("RMSE for VC:", rmse_vc)
MAE for VC: 0.069
MSE for VC: 0.069
RMSE for VC: 0.26267851073127396

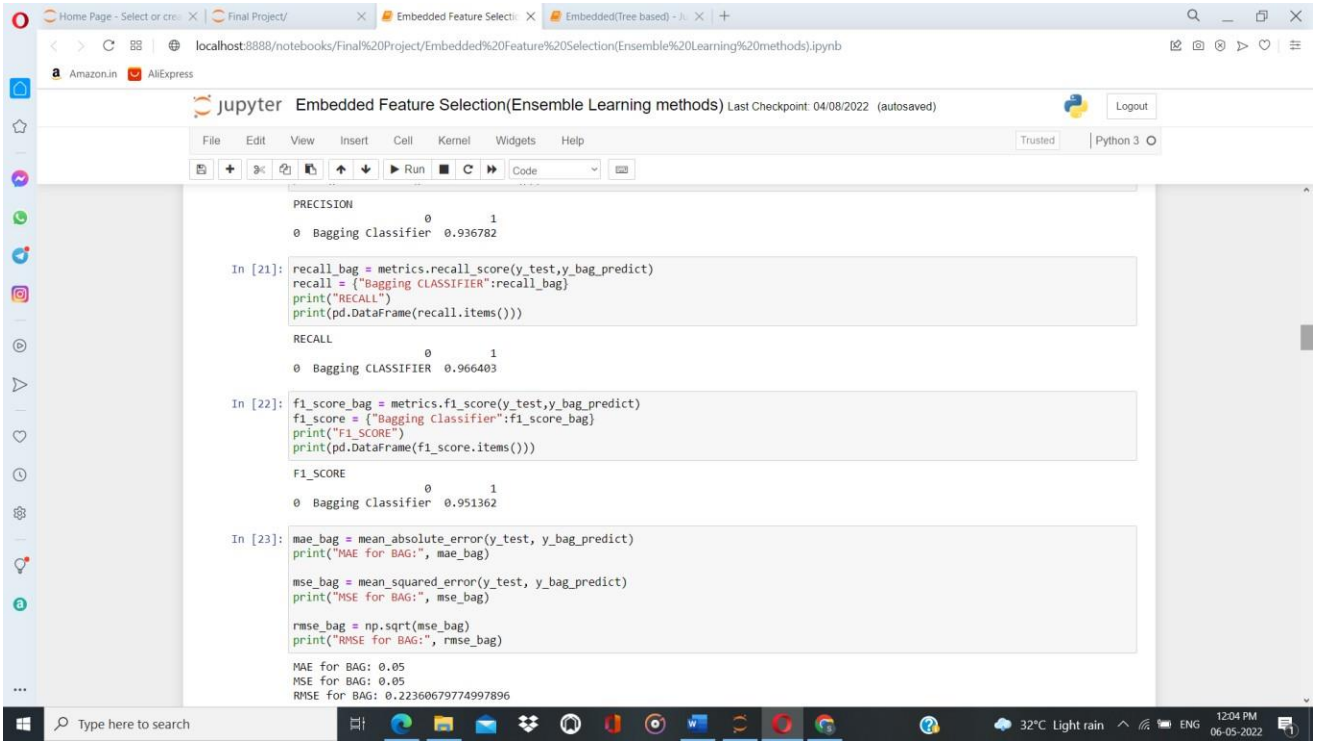

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
Advanced techniques
BAGGING
Bagging meta-estimator
In [17]:
from sklearn.ensemble import BaggingClassifier
model1 = BaggingClassifier(tree.DecisionTreeClassifier(random_state=1))
model1.fit(X_train, y_train)
model1.score(X_test,y_test)
Out[17]: 0.95
In [18]:
y_bag_predict = model1.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_bag_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_bag_predict")
plt.ylabel("y_test")
plt.show()




```
In [19]: print(classification_report(y_test, y_bag_predict))
         print('Bagging Classifier Accuracy: {:.02%}'
               .format(model1.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.95       988
           1       0.94      0.97      0.95      1012

    accuracy                           0.95      2000
   macro avg       0.95      0.95      0.95      2000
weighted avg       0.95      0.95      0.95      2000

Bagging Classifier Accuracy: 95.00%
```

```
In [20]: precision_bag = metrics.precision_score(y_test,y_bag_predict)
         precision = {"Bagging Classifier":precision_bag}
```



```
PRECISION
                    0         1
0  Bagging Classifier  0.936782
```

```
In [21]: recall_bag = metrics.recall_score(y_test,y_bag_predict)
         recall = {"Bagging CLASSIFIER":recall_bag}
         print("RECALL")
         print(pd.DataFrame(recall.items()))
```

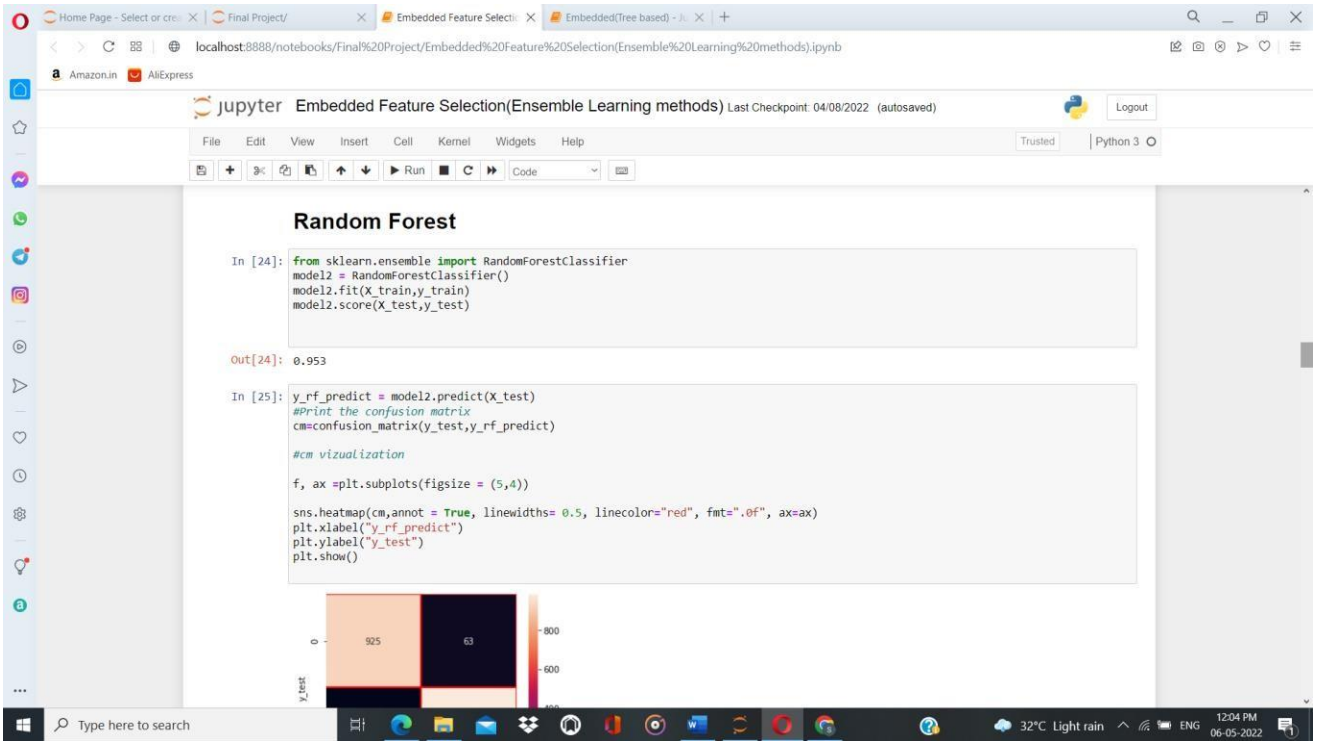
```
RECALL
                    0         1
0  Bagging CLASSIFIER  0.966403
```

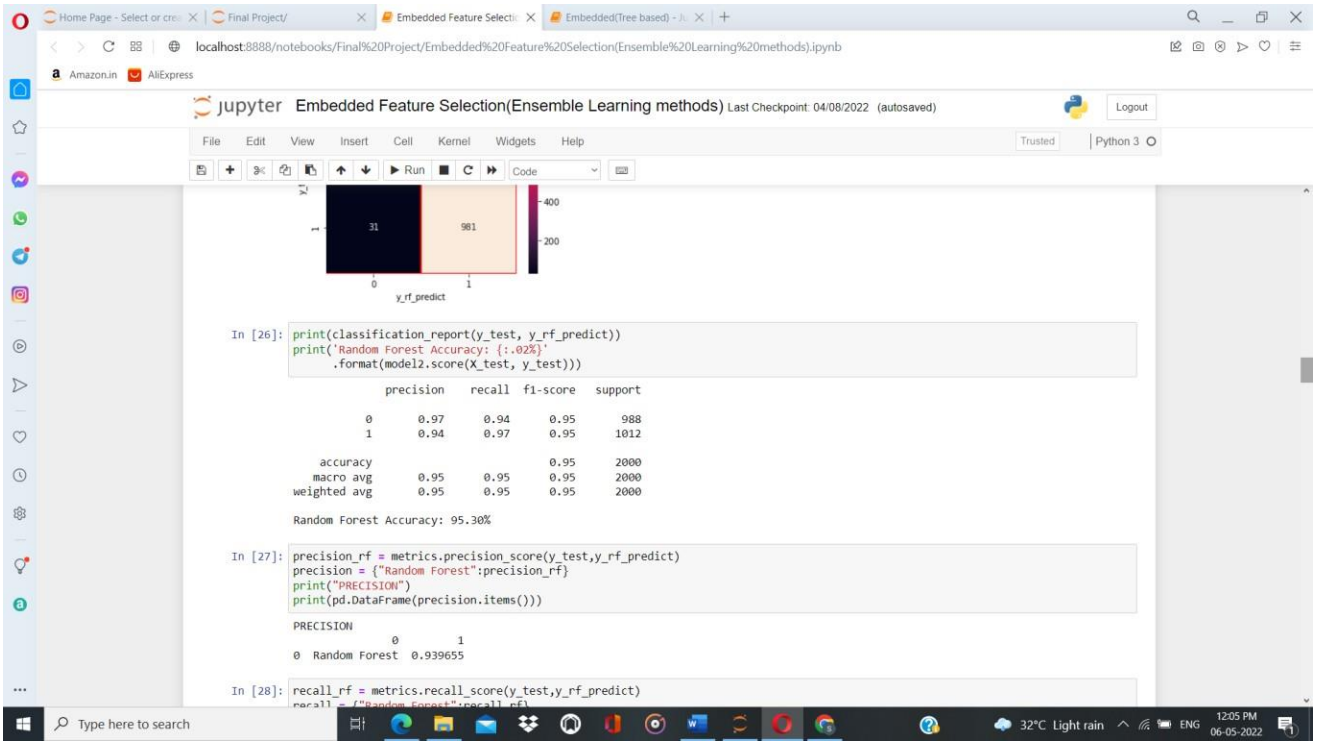
```
In [22]: f1_score_bag = metrics.f1_score(y_test,y_bag_predict)
         f1_score = {"Bagging Classifier":f1_score_bag}
         print("F1_SCORE")
         print(pd.DataFrame(f1_score.items()))
```

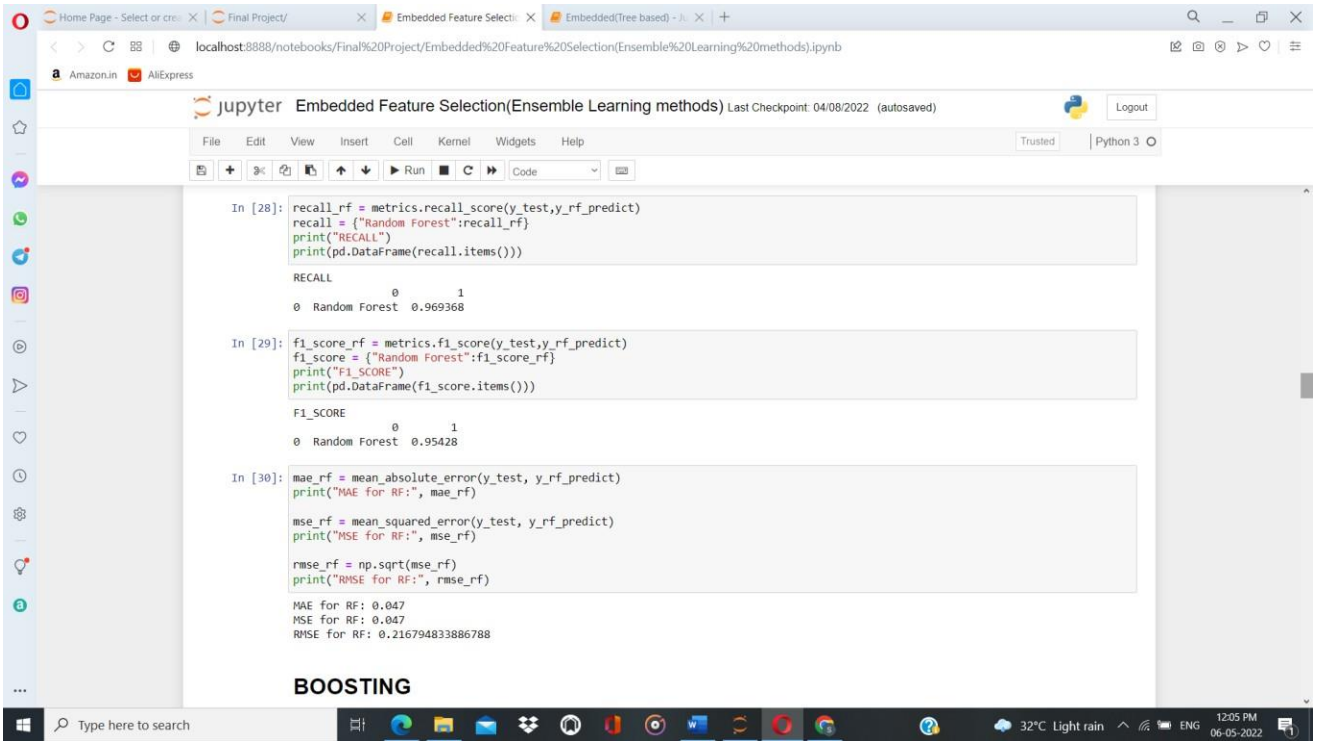
```
F1_SCORE
                    0         1
0  Bagging Classifier  0.951362
```

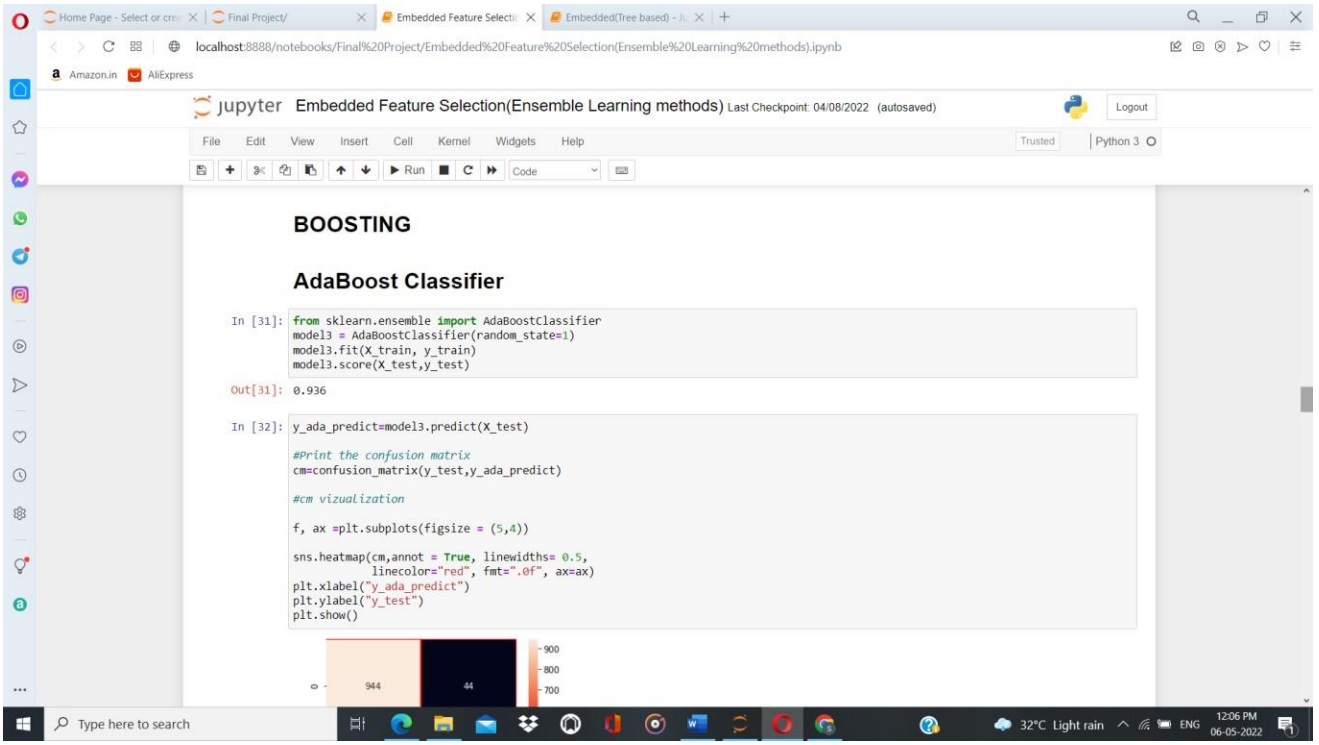
```
In [23]: mae_bag = mean_absolute_error(y_test, y_bag_predict)
         print("MAE for BAG:", mae_bag)

         mse_bag = mean_squared_error(y_test, y_bag_predict)
         print("MSE for BAG:", mse_bag)

         rmse_bag = np.sqrt(mse_bag)
         print("RMSE for BAG:", rmse_bag)
```

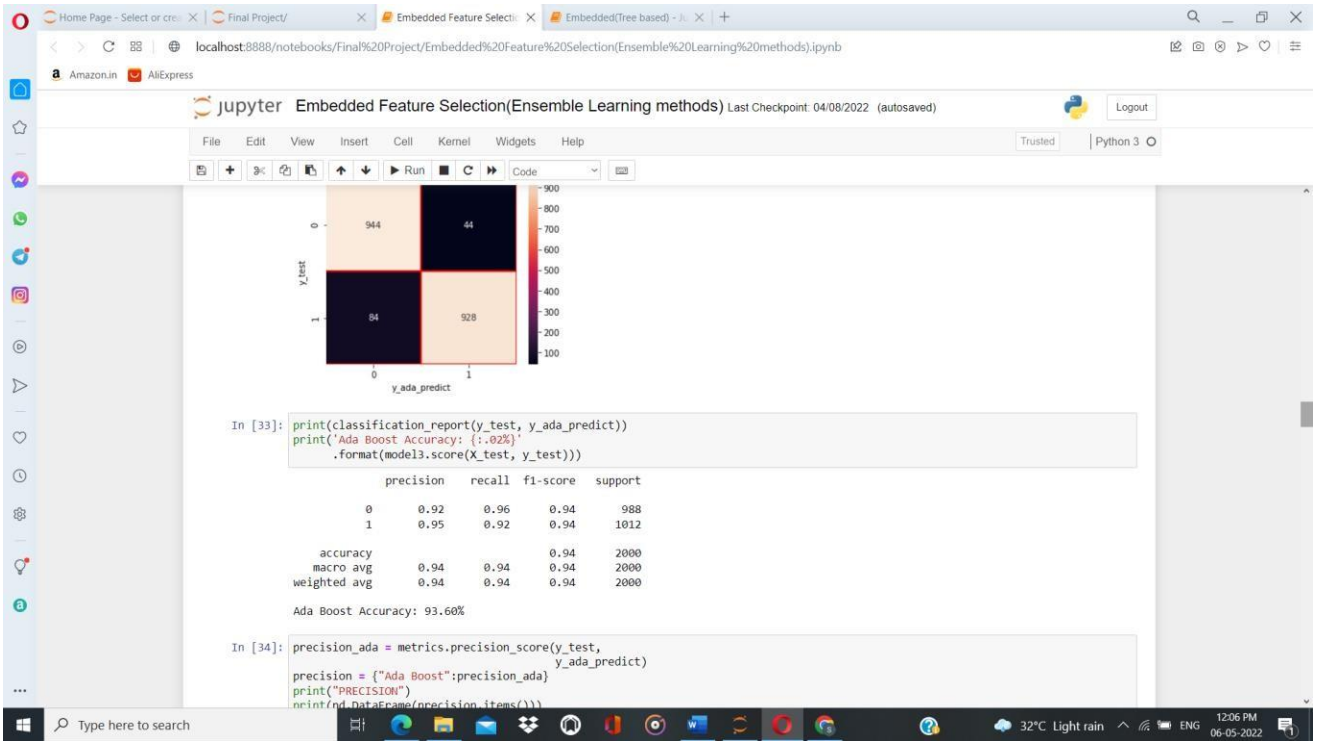
```
MAE for BAG: 0.05
MSE for BAG: 0.05
RMSE for BAG: 0.22360679774997896
```

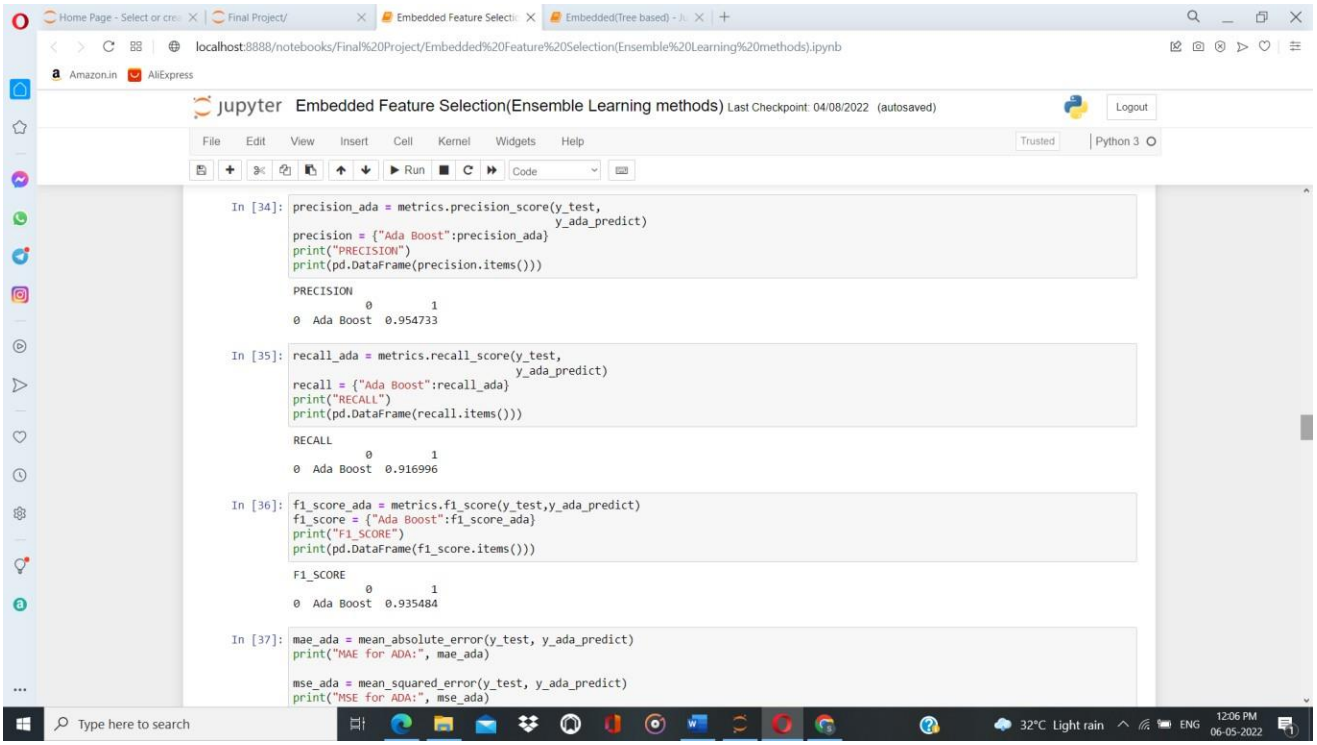

Embedded Feature Selection(Ensemble Learning methods)
Random Forest
In [24]:
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier()
model2.fit(X_train,y_train)
model2.score(X_test,y_test)
Out[24]: 0.953
In [25]:
y_rf_predict = model2.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_rf_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_rf_predict")
plt.ylabel("y_test")
plt.show()
925
63
800
600
y_test


Embedded Feature Selection(Ensemble Learning methods)
31
981
400
200
y_rf_predict
In [26]:
print(classification_report(y_test, y_rf_predict))
print('Random Forest Accuracy: {:.02%}'
.format(model2.score(X_test, y_test)))
precision recall f1-score support
0 0.97 0.94 0.95 988
1 0.94 0.97 0.95 1012
accuracy 0.95 2000
macro avg 0.95 0.95 0.95 2000
weighted avg 0.95 0.95 0.95 2000
Random Forest Accuracy: 95.30%
In [27]:
precision_rf = metrics.precision_score(y_test,y_rf_predict)
precision = {"Random Forest":precision_rf}
print("PRECISION")
print(pd.DataFrame(precision.items()))
PRECISION
0 1
0 Random Forest 0.939655
In [28]:
recall_rf = metrics.recall_score(y_test,y_rf_predict)


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Logout
In [28]:
recall_rf = metrics.recall_score(y_test,y_rf_predict)
recall = {"Random Forest":recall_rf}
print("RECALL")
print(pd.DataFrame(recall.items()))
RECALL
0 1
0 Random Forest 0.969368
In [29]:
f1_score_rf = metrics.f1_score(y_test,y_rf_predict)
f1_score = {"Random Forest":f1_score_rf}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
F1_SCORE
0 1
0 Random Forest 0.95428
In [30]:
mae_rf = mean_absolute_error(y_test, y_rf_predict)
print("MAE for RF:", mae_rf)
mse_rf = mean_squared_error(y_test, y_rf_predict)
print("MSE for RF:", mse_rf)
rmse_rf = np.sqrt(mse_rf)
print("RMSE for RF:", rmse_rf)
MAE for RF: 0.047
MSE for RF: 0.047
RMSE for RF: 0.216794833886788
BOOSTING
Type here to search
32°C Light rain
ENG
12:05 PM
06-05-2022


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Logout
BOOSTING
AdaBoost Classifier
In [31]:
from sklearn.ensemble import AdaBoostClassifier
model3 = AdaBoostClassifier(random_state=1)
model3.fit(X_train, y_train)
model3.score(X_test,y_test)
Out[31]: 0.936
In [32]:
y_ada_predict=model3.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_ada_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5,
linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_ada_predict")
plt.ylabel("y_test")
plt.show()
944
44
900
800
700
Type here to search
32°C Light rain
ENG
12:06 PM
06-05-2022


Embedded Feature Selection(Ensemble Learning methods)
944
44
84
928
y_test
y_ada_predict
In [33]:
print(classification_report(y_test, y_ada_predict))
print('Ada Boost Accuracy: {:.02%}'
.format(model3.score(X_test, y_test)))
precision recall f1-score support
0 0.92 0.96 0.94 988
1 0.95 0.92 0.94 1012
accuracy 0.94 2000
macro avg 0.94 0.94 0.94 2000
weighted avg 0.94 0.94 0.94 2000
Ada Boost Accuracy: 93.60%
In [34]:
precision_ada = metrics.precision_score(y_test,
y_ada_predict)
precision = {"Ada Boost":precision_ada}
print("PRECISION")


In [34]:
precision_ada = metrics.precision_score(y_test,
y_ada_predict)
precision = {"Ada Boost":precision_ada}
print("PRECISION")
print(pd.DataFrame(precision.items()))
PRECISION
0 1
0 Ada Boost 0.954733
In [35]:
recall_ada = metrics.recall_score(y_test,
y_ada_predict)
recall = {"Ada Boost":recall_ada}
print("RECALL")
print(pd.DataFrame(recall.items()))
RECALL
0 1
0 Ada Boost 0.916996
In [36]:
f1_score_ada = metrics.f1_score(y_test,y_ada_predict)
f1_score = {"Ada Boost":f1_score_ada}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
F1_SCORE
0 1
0 Ada Boost 0.935484
In [37]:
mae_ada = mean_absolute_error(y_test, y_ada_predict)
print("MAE for ADA:", mae_ada)
mse_ada = mean_squared_error(y_test, y_ada_predict)
print("MSE for ADA:", mse_ada)

```
print("MAE for ADA:", mae_ada)

mse_ada = mean_squared_error(y_test, y_ada_predict)
print("MSE for ADA:", mse_ada)

rmse_ada = np.sqrt(mse_ada)
print("RMSE for ADA:", rmse_ada)
```

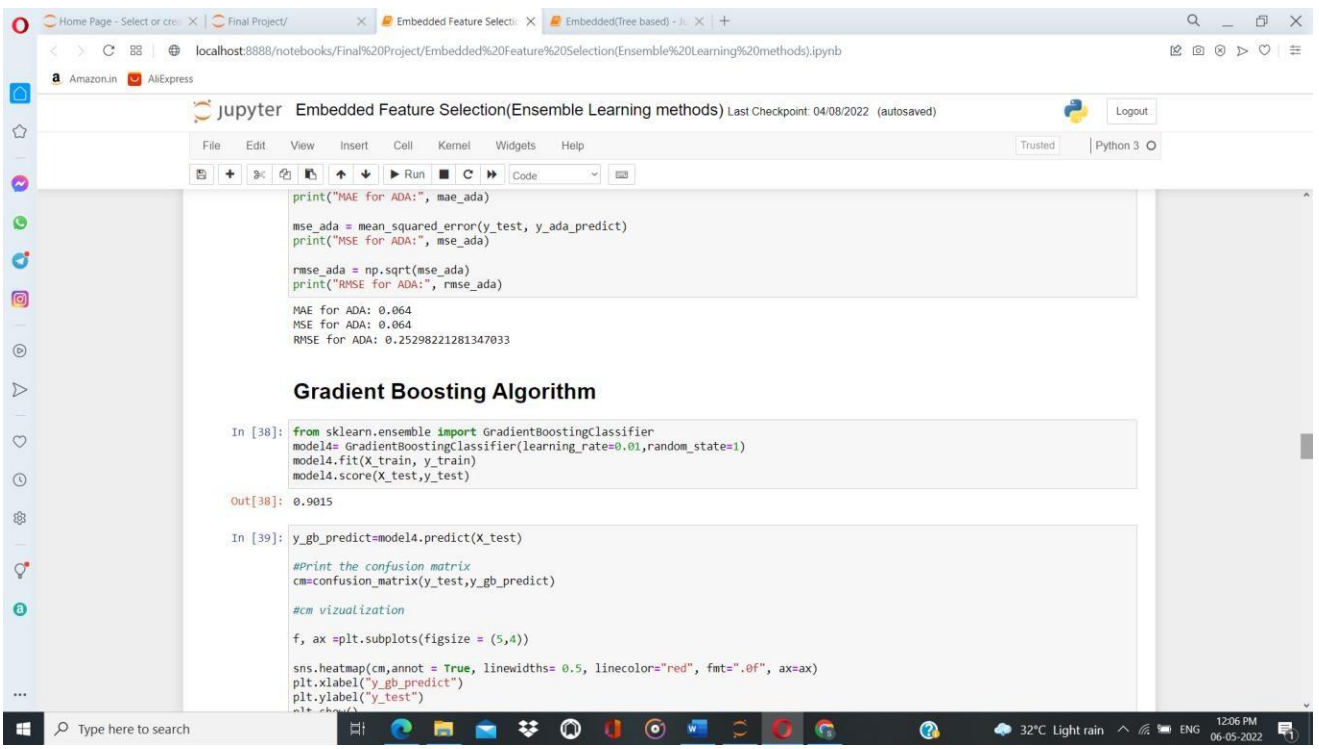
```
MAE for ADA: 0.064
MSE for ADA: 0.064
RMSE for ADA: 0.25298221281347033
```

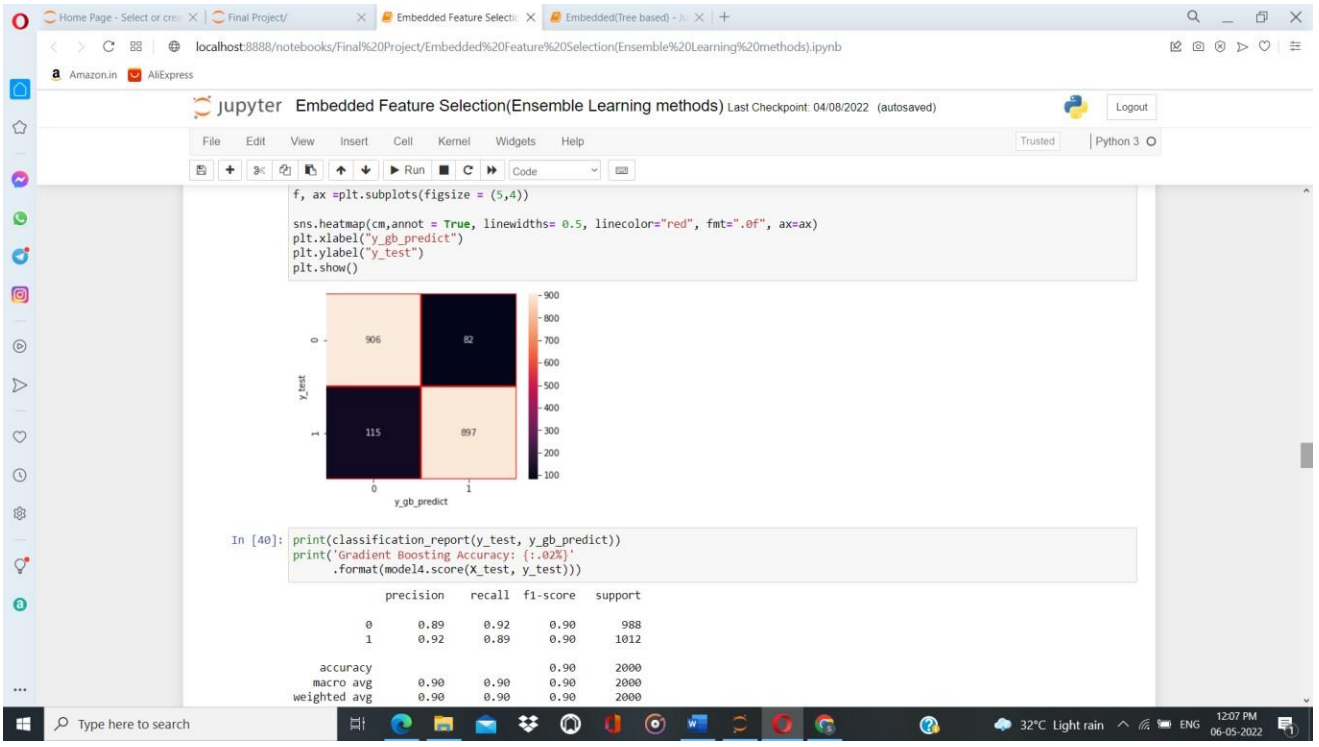
Gradient Boosting Algorithm

In [38]:
```
from sklearn.ensemble import GradientBoostingClassifier
model4= GradientBoostingClassifier(learning_rate=0.01,random_state=1)
model4.fit(X_train, y_train)
model4.score(X_test,y_test)
```

Out[38]: 0.9015

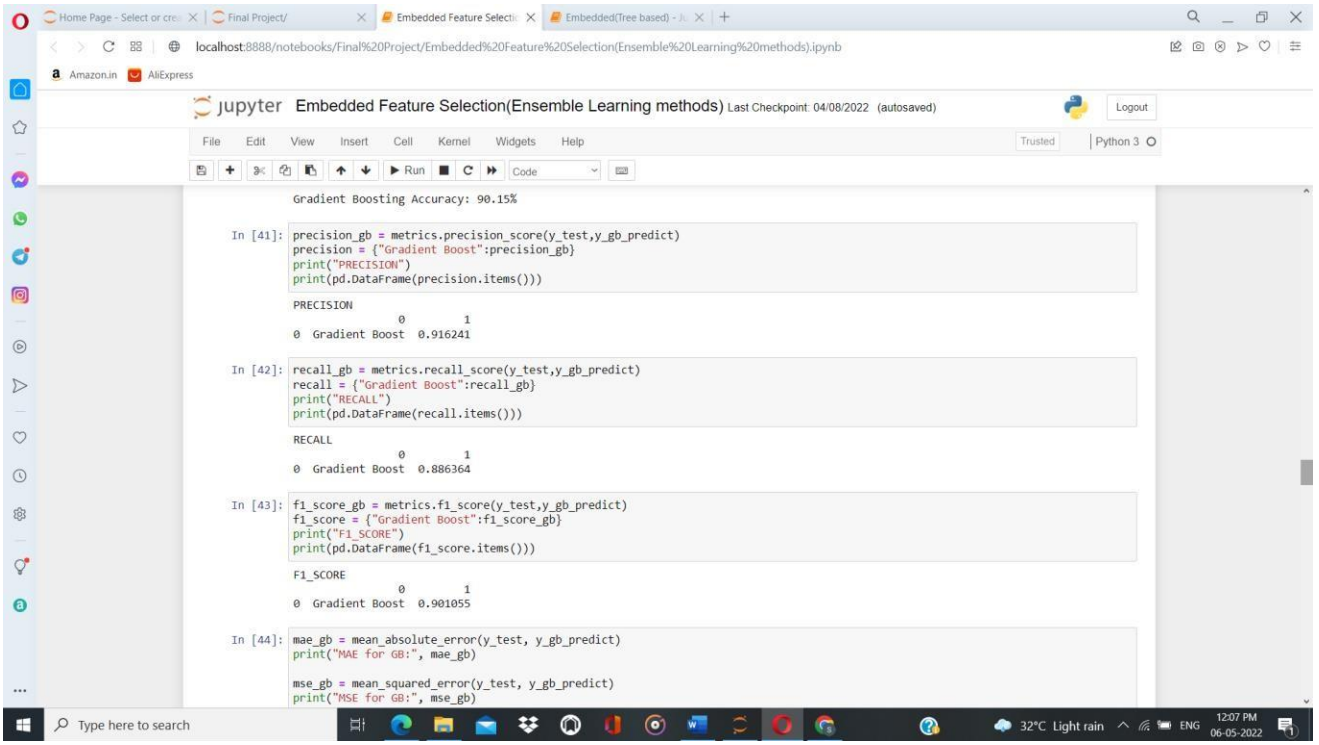
In [39]:
```
y_gb_predict=model4.predict(X_test)

#Print the confusion matrix
cm=confusion_matrix(y_test,y_gb_predict)

#cm vizualization

f, ax =plt.subplots(figsize = (5,4))

sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_gb_predict")
plt.ylabel("y_test")
```

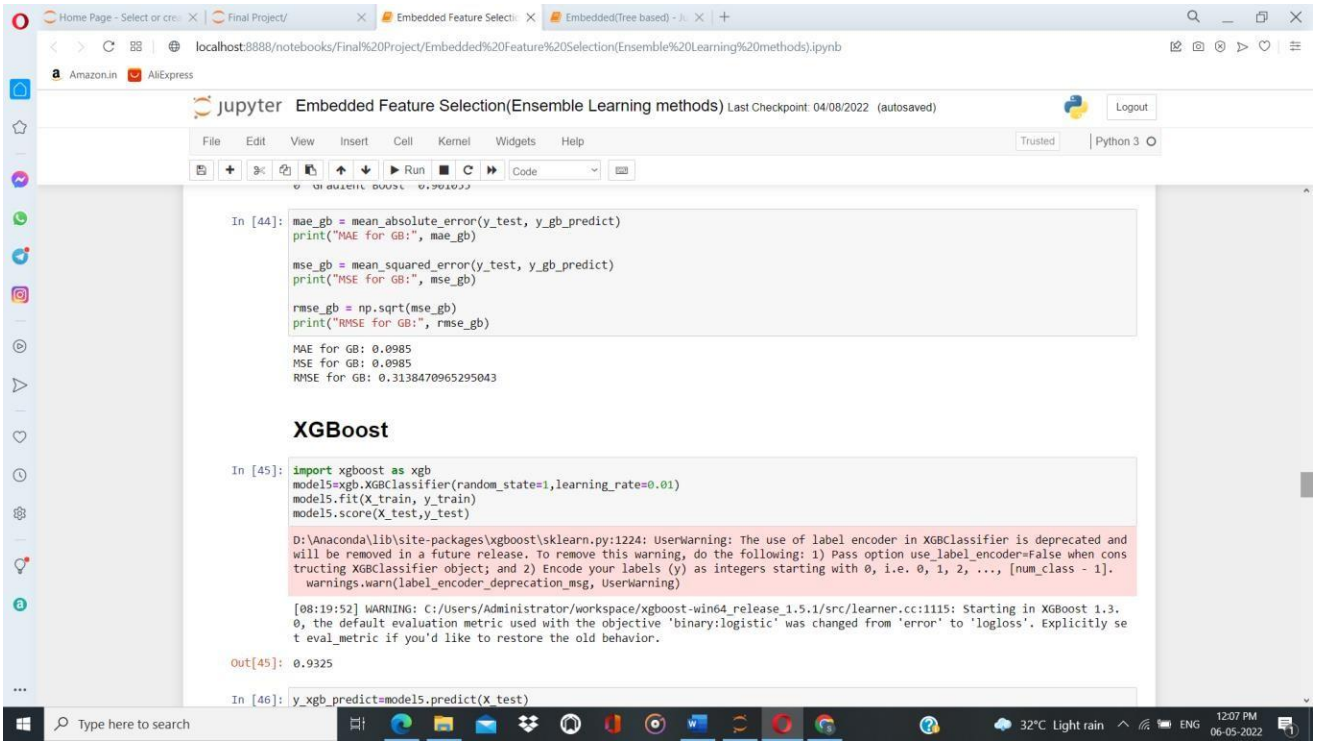


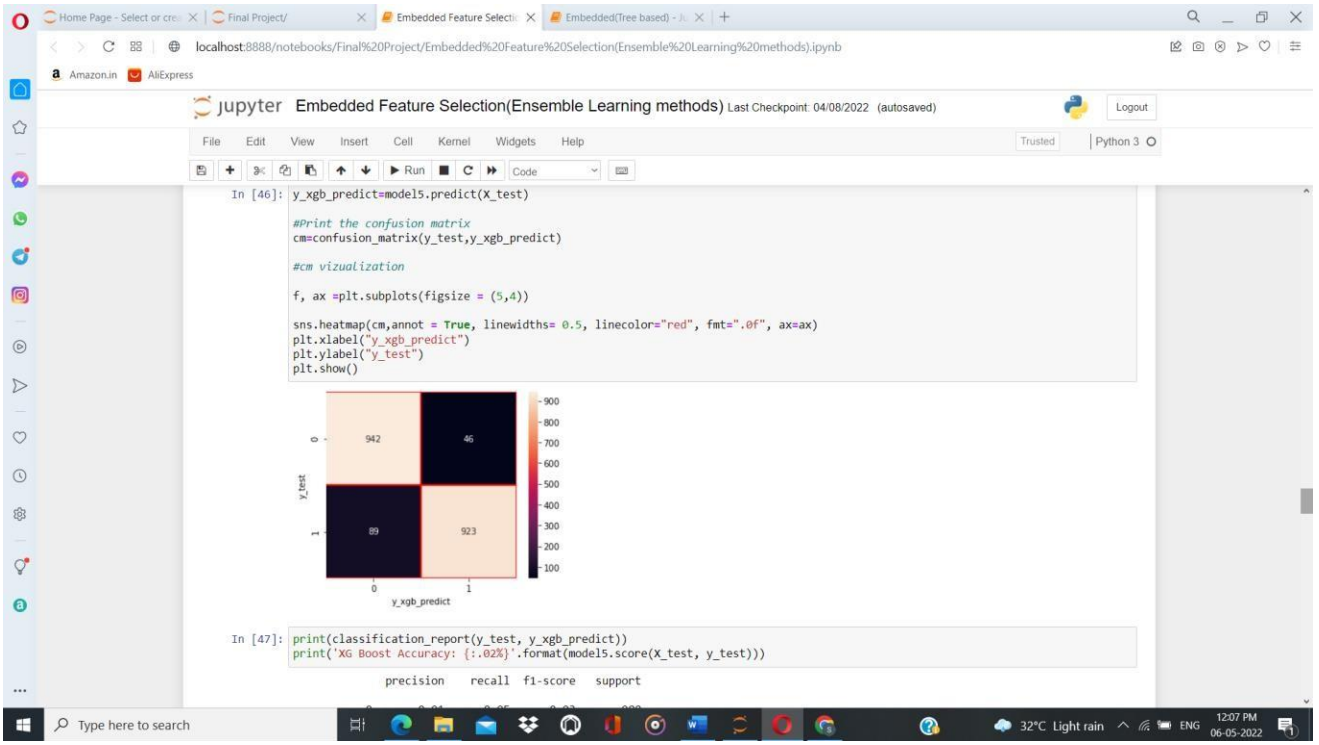


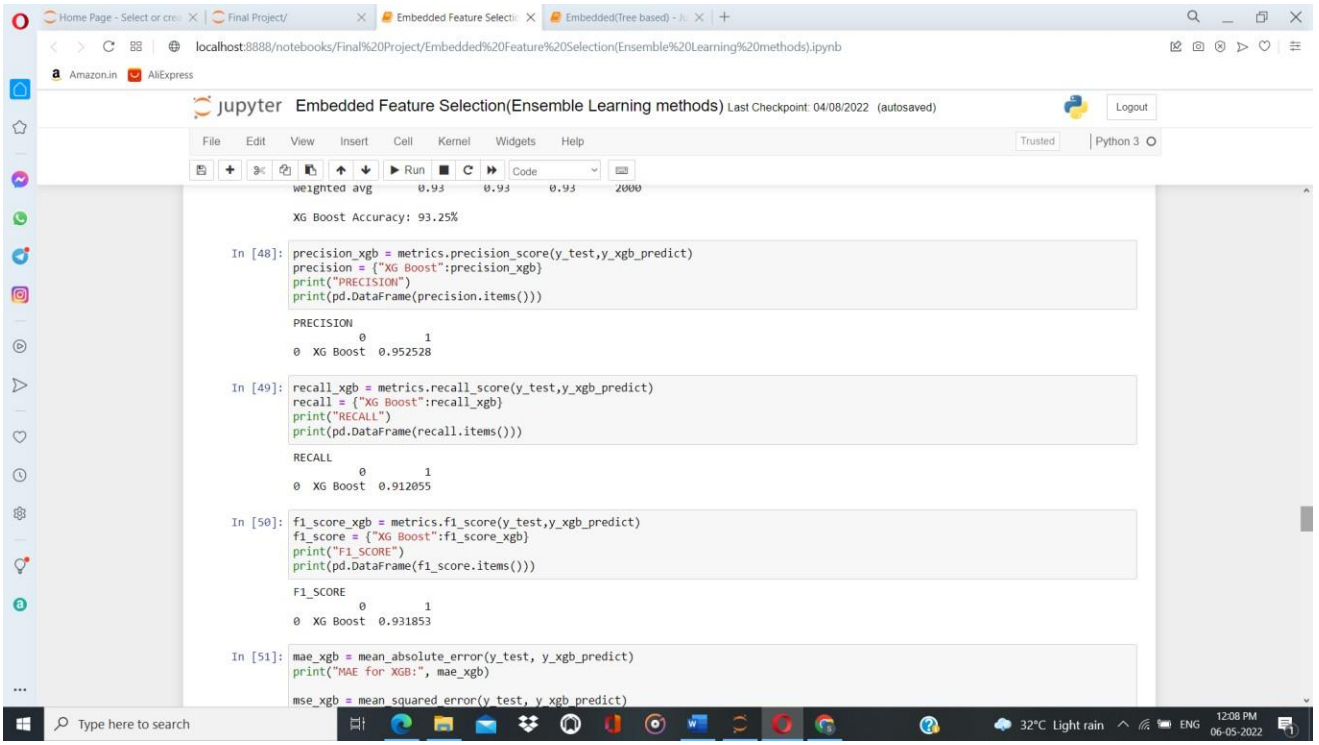
In [40]:
```
print(classification_report(y_test, y_gb_predict))
print('Gradient Boosting Accuracy: {:.02%}'
      .format(model4.score(X_test, y_test)))
```

```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90       988
           1       0.92      0.89      0.90      1012

    accuracy                           0.90      2000
   macro avg       0.90      0.90      0.90      2000
weighted avg       0.90      0.90      0.90      2000
```

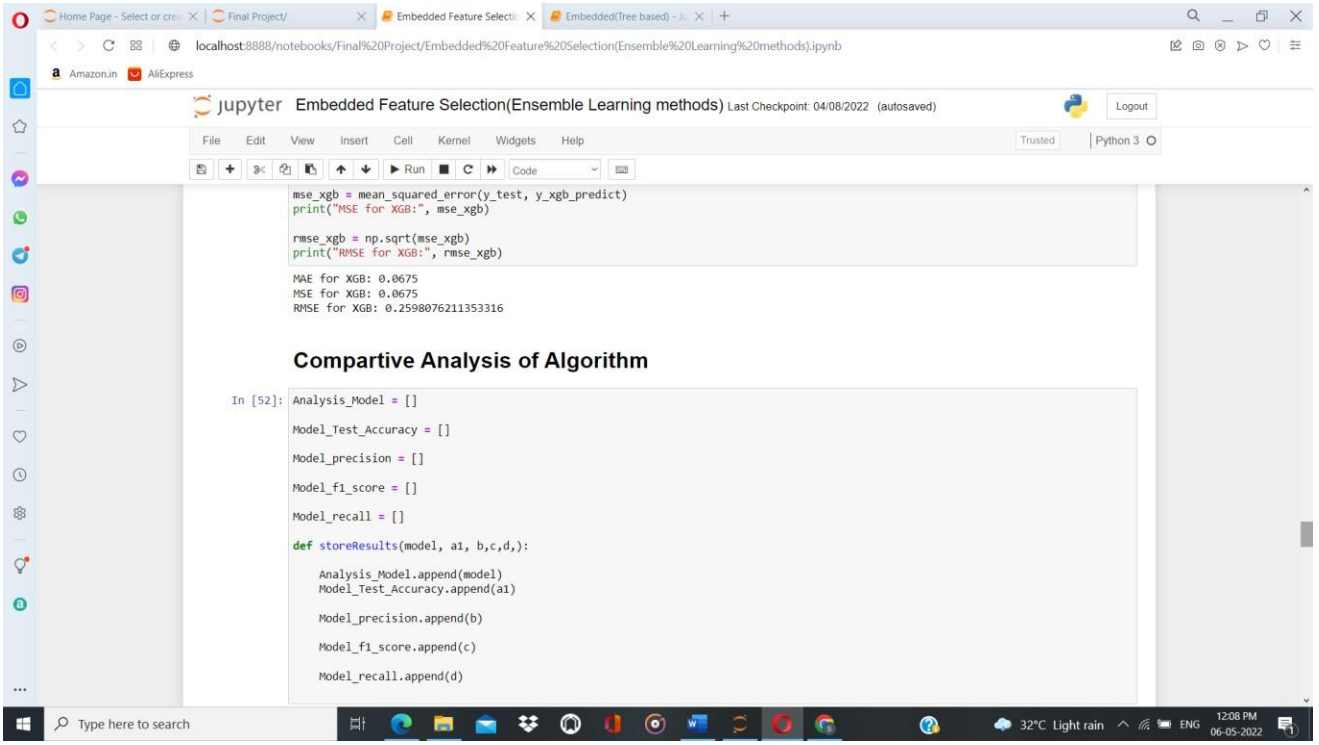

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Gradient Boosting Accuracy: 90.15%
In [41]:
precision_gb = metrics.precision_score(y_test,y_gb_predict)
precision = {"Gradient Boost":precision_gb}
print("PRECISION")
print(pd.DataFrame(precision.items()))
PRECISION
0 1
0 Gradient Boost 0.916241
In [42]:
recall_gb = metrics.recall_score(y_test,y_gb_predict)
recall = {"Gradient Boost":recall_gb}
print("RECALL")
print(pd.DataFrame(recall.items()))
RECALL
0 1
0 Gradient Boost 0.886364
In [43]:
f1_score_gb = metrics.f1_score(y_test,y_gb_predict)
f1_score = {"Gradient Boost":f1_score_gb}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
F1_SCORE
0 1
0 Gradient Boost 0.901055
In [44]:
mae_gb = mean_absolute_error(y_test, y_gb_predict)
print("MAE for GB:", mae_gb)
mse_gb = mean_squared_error(y_test, y_gb_predict)
print("MSE for GB:", mse_gb)
Type here to search
32°C Light rain
ENG
12:07 PM
06-05-2022

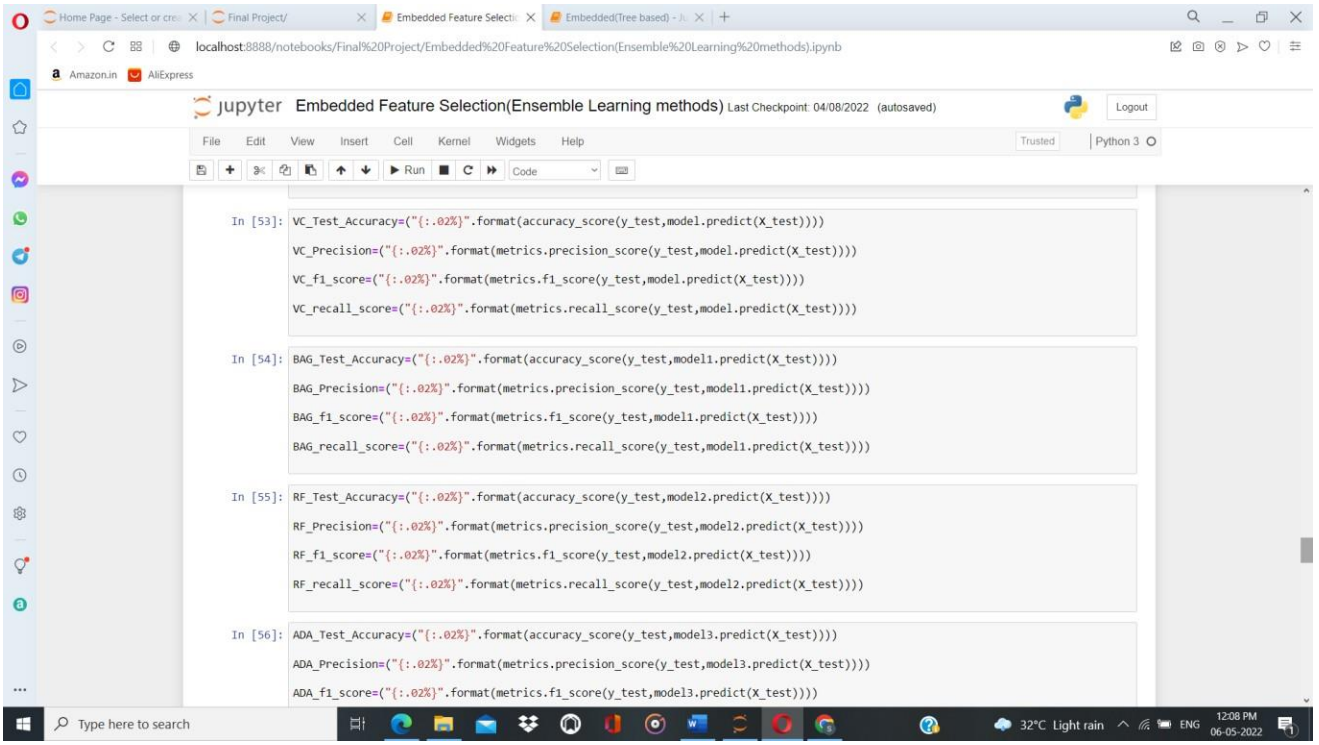

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [44]:
mae_gb = mean_absolute_error(y_test, y_gb_predict)
print("MAE for GB:", mae_gb)
mse_gb = mean_squared_error(y_test, y_gb_predict)
print("MSE for GB:", mse_gb)
rmse_gb = np.sqrt(mse_gb)
print("RMSE for GB:", rmse_gb)
MAE for GB: 0.0985
MSE for GB: 0.0985
RMSE for GB: 0.3138470965295043
XGBoost
In [45]:
import xgboost as xgb
model5=xgb.XGBClassifier(random_state=1,learning_rate=0.01)
model5.fit(X_train, y_train)
model5.score(X_test,y_test)
D:\Anaconda\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
warnings.warn(label_encoder_deprecation_msg, UserWarning)
[08:19:52] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
Out[45]: 0.9325
In [46]:
y_xgb_predict=model5.predict(X_test)
Type here to search
32°C Light rain
ENG
12:07 PM
06-05-2022

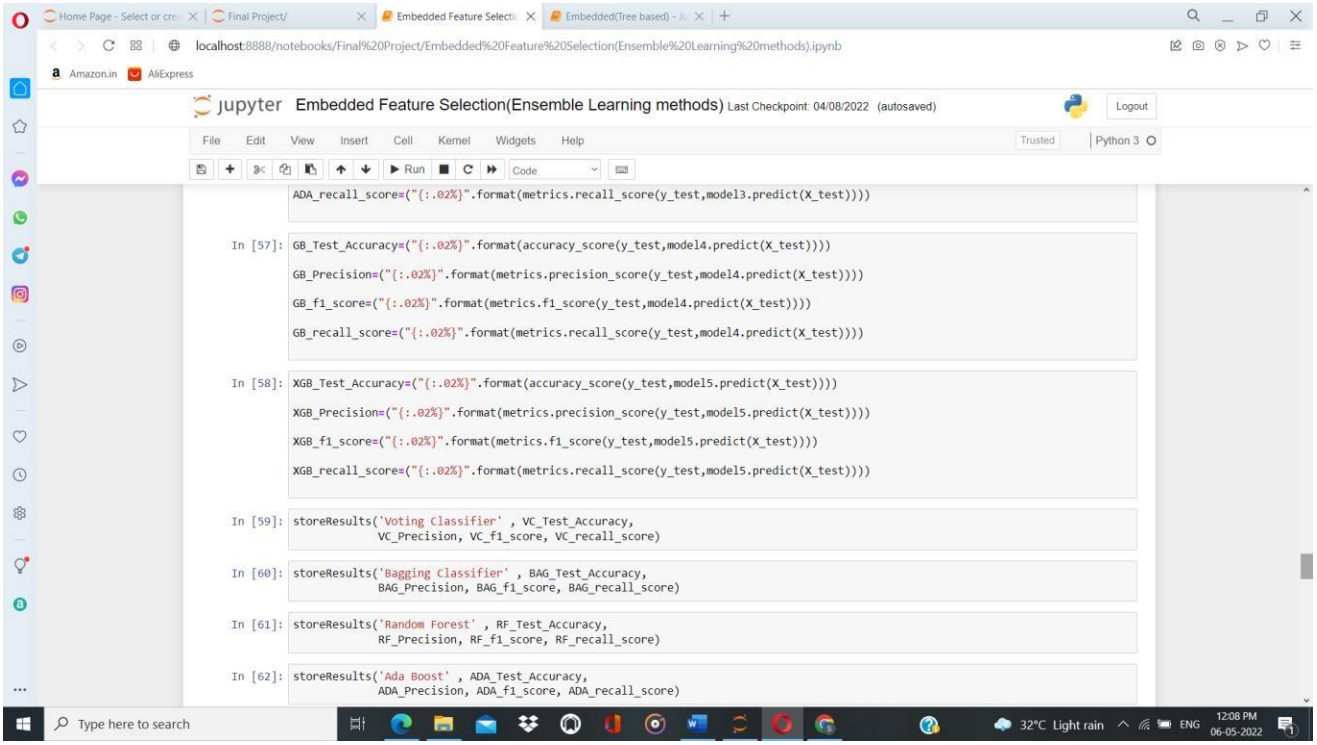

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
In [46]:
y_xgb_predict=model5.predict(X_test)
#Print the confusion matrix
cm=confusion_matrix(y_test,y_xgb_predict)
#cm vizualization
f, ax =plt.subplots(figsize = (5,4))
sns.heatmap(cm,annot = True, linewidths= 0.5, linecolor="red", fmt=".0f", ax=ax)
plt.xlabel("y_xgb_predict")
plt.ylabel("y_test")
plt.show()
942
46
89
923
y_test
y_xgb_predict
In [47]:
print(classification_report(y_test, y_xgb_predict))
print('XG Boost Accuracy: {:.02%}'.format(model5.score(X_test, y_test)))
precision recall f1-score support
Type here to search
32°C Light rain
ENG
12:07 PM
06-05-2022

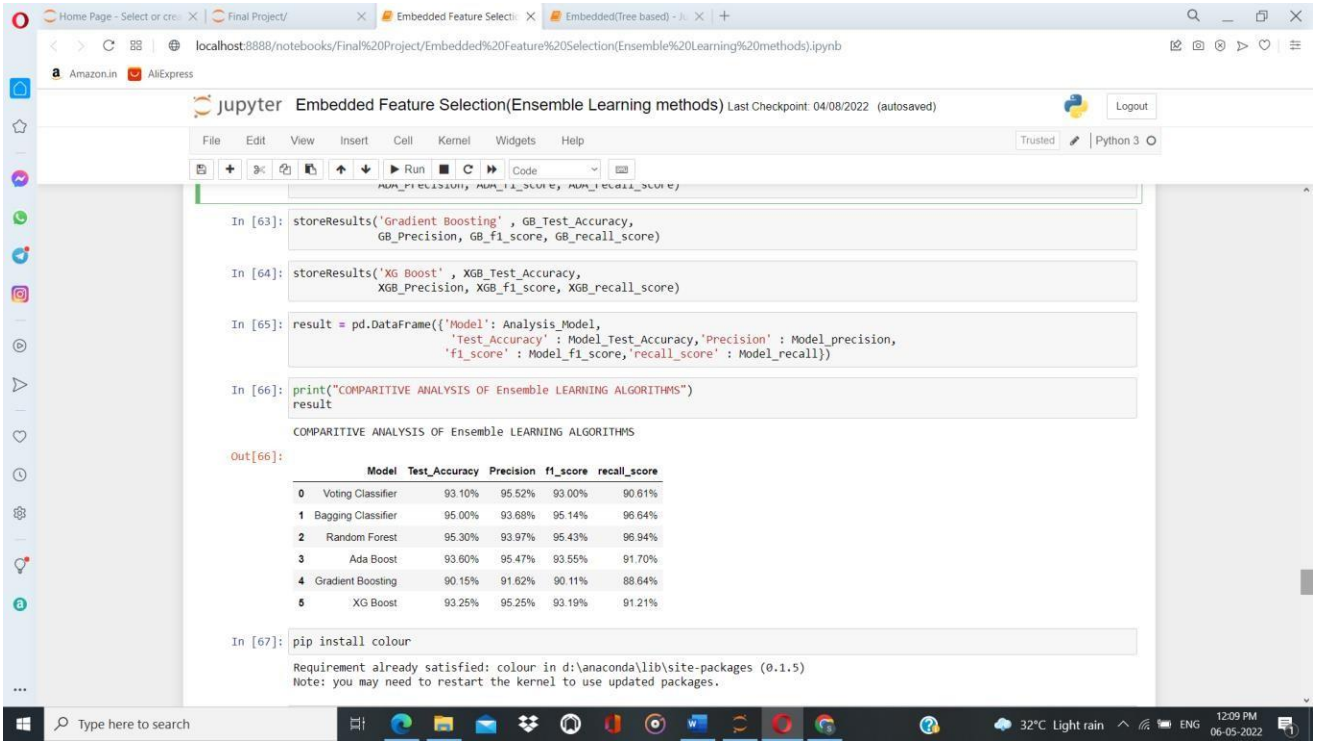

Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3
weighted avg 0.93 0.93 0.93 2000
XG Boost Accuracy: 93.25%
In [48]:
precision_xgb = metrics.precision_score(y_test,y_xgb_predict)
precision = {"XG Boost":precision_xgb}
print("PRECISION")
print(pd.DataFrame(precision.items()))
PRECISION
0 1
0 XG Boost 0.952528
In [49]:
recall_xgb = metrics.recall_score(y_test,y_xgb_predict)
recall = {"XG Boost":recall_xgb}
print("RECALL")
print(pd.DataFrame(recall.items()))
RECALL
0 1
0 XG Boost 0.912055
In [50]:
f1_score_xgb = metrics.f1_score(y_test,y_xgb_predict)
f1_score = {"XG Boost":f1_score_xgb}
print("F1_SCORE")
print(pd.DataFrame(f1_score.items()))
F1_SCORE
0 1
0 XG Boost 0.931853
In [51]:
mae_xgb = mean_absolute_error(y_test, y_xgb_predict)
print("MAE for XGB:", mae_xgb)
mse_xgb = mean_squared_error(y_test, y_xgb_predict)
Type here to search
32°C Light rain
ENG
12:08 PM
06-05-2022

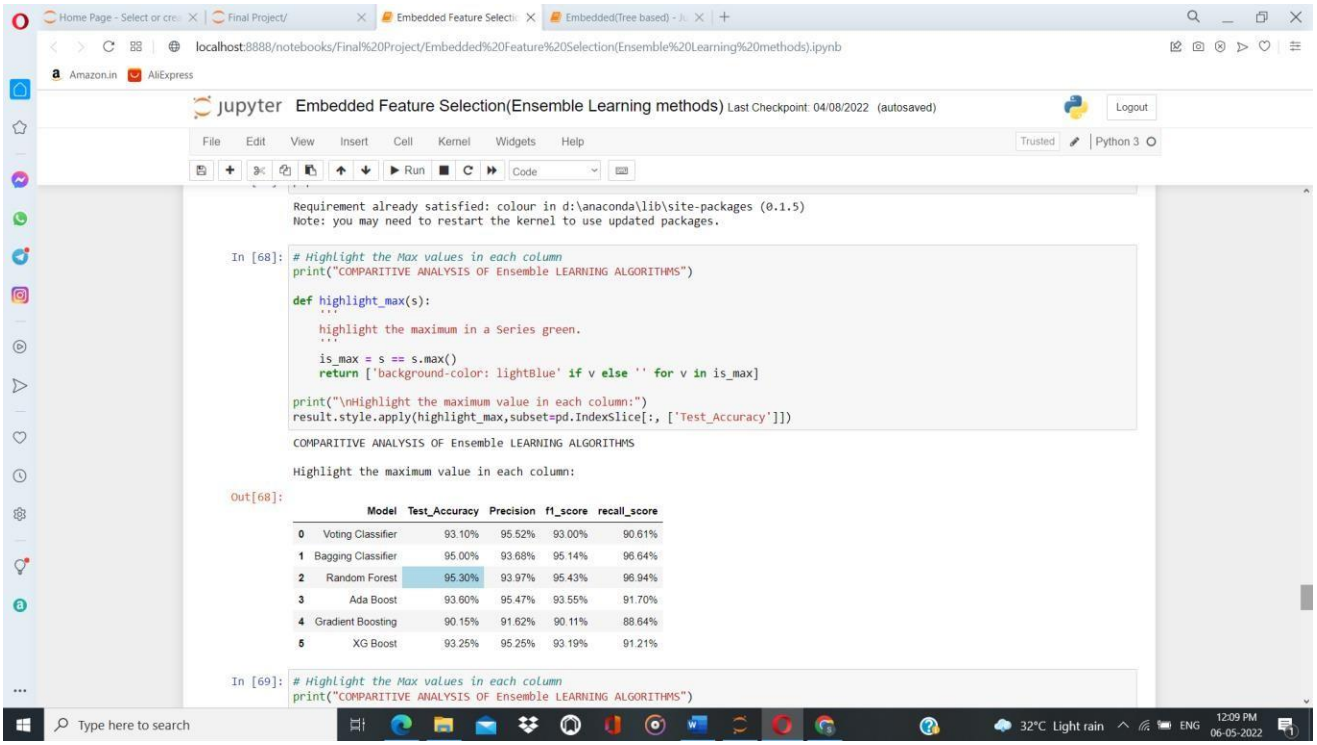
COMPARATIVE ANALYSIS

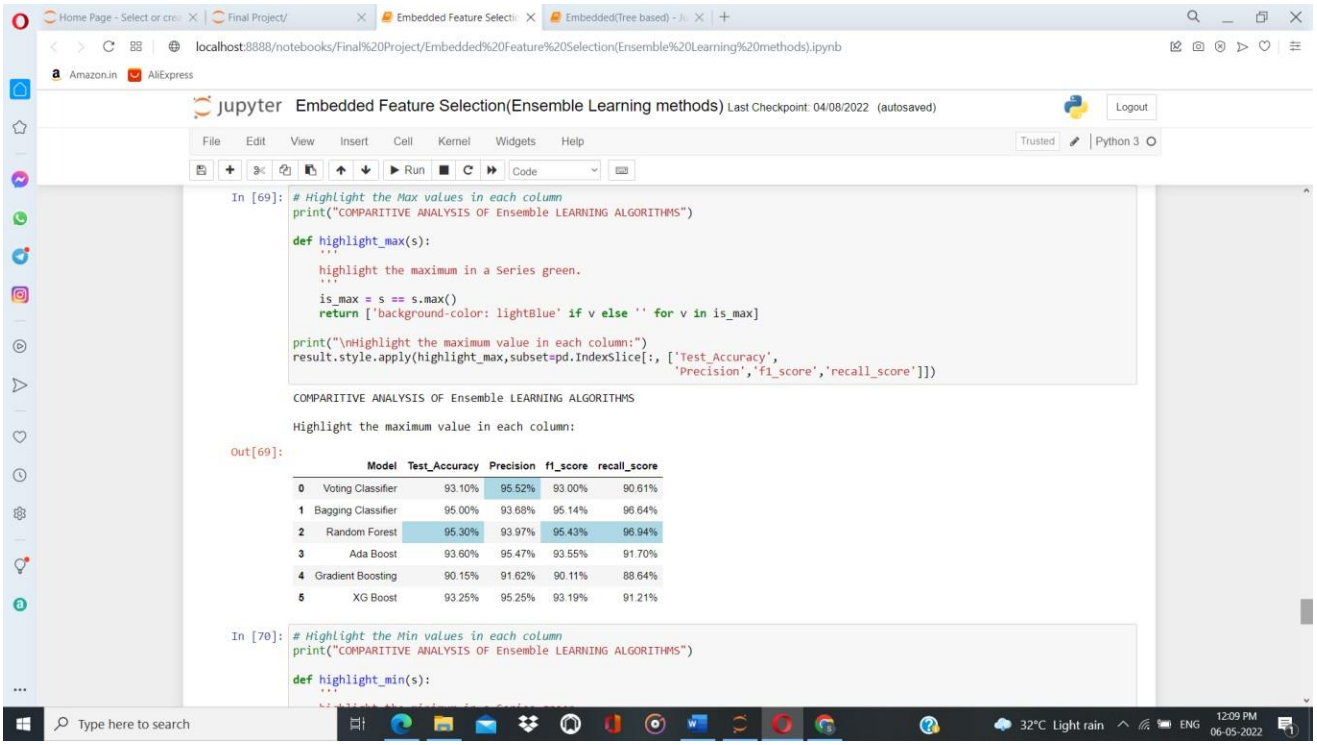


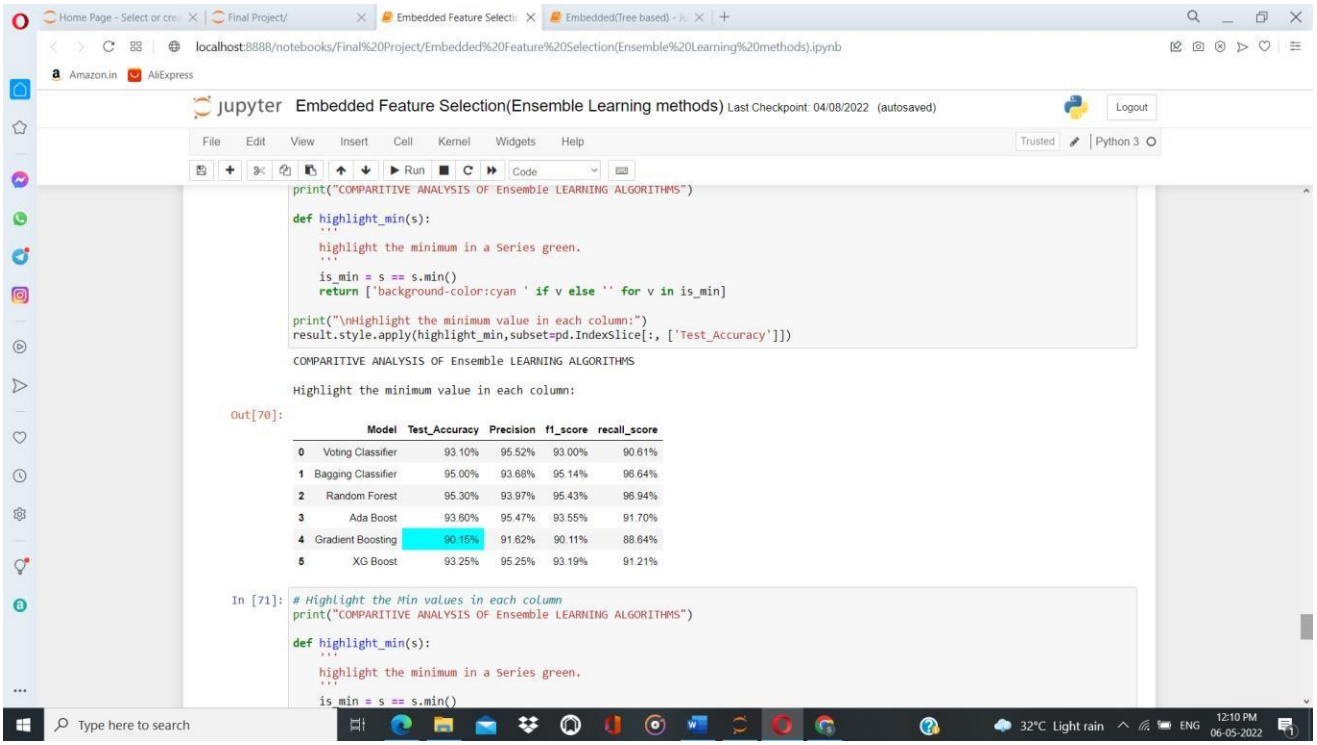



Embedded Feature Selection(Ensemble Learning methods)
ADA_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model3.predict(X_test))))
In [57]:
GB_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model4.predict(X_test))))
GB_Precision=("{:.02%}".format(metrics.precision_score(y_test,model4.predict(X_test))))
GB_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model4.predict(X_test))))
GB_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model4.predict(X_test))))
In [58]:
XGB_Test_Accuracy=("{:.02%}".format(accuracy_score(y_test,model5.predict(X_test))))
XGB_Precision=("{:.02%}".format(metrics.precision_score(y_test,model5.predict(X_test))))
XGB_f1_score=("{:.02%}".format(metrics.f1_score(y_test,model5.predict(X_test))))
XGB_recall_score=("{:.02%}".format(metrics.recall_score(y_test,model5.predict(X_test))))
In [59]:
storeResults('Voting Classifier' , VC_Test_Accuracy,
VC_Precision, VC_f1_score, VC_recall_score)
In [60]:
storeResults('Bagging Classifier' , BAG_Test_Accuracy,
BAG_Precision, BAG_f1_score, BAG_recall_score)
In [61]:
storeResults('Random Forest' , RF_Test_Accuracy,
RF_Precision, RF_f1_score, RF_recall_score)
In [62]:
storeResults('Ada Boost' , ADA_Test_Accuracy,
ADA_Precision, ADA_f1_score, ADA_recall_score)


Embedded Feature Selection(Ensemble Learning methods)
In [63]:
storeResults('Gradient Boosting' , GB_Test_Accuracy,
GB_Precision, GB_f1_score, GB_recall_score)
In [64]:
storeResults('XG Boost' , XGB_Test_Accuracy,
XGB_Precision, XGB_f1_score, XGB_recall_score)
In [65]:
result = pd.DataFrame({'Model': Analysis_Model,
'Test_Accuracy' : Model_Test_Accuracy,'Precision' : Model_precision,
'f1_score' : Model_f1_score,'recall_score' : Model_recall})
In [66]:
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
result
COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS
Out[66]:
Model Test_Accuracy Precision f1_score recall_score
0 Voting Classifier 93.10% 95.52% 93.00% 90.61%
1 Bagging Classifier 95.00% 93.68% 95.14% 96.64%
2 Random Forest 95.30% 93.97% 95.43% 96.94%
3 Ada Boost 93.60% 95.47% 93.55% 91.70%
4 Gradient Boosting 90.15% 91.62% 90.11% 88.64%
5 XG Boost 93.25% 95.25% 93.19% 91.21%
In [67]:
pip install colour
Requirement already satisfied: colour in d:\anaconda\lib\site-packages (0.1.5)
Note: you may need to restart the kernel to use updated packages.


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
Requirement already satisfied: colour in d:\anaconda\lib\site-packages (0.1.5)
Note: you may need to restart the kernel to use updated packages.
In [68]:
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_max(s):
'''
highlight the maximum in a Series green.
'''
is_max = s == s.max()
return ['background-color: lightBlue' if v else '' for v in is_max]
print("\nHighlight the maximum value in each column:")
result.style.apply(highlight_max,subset=pd.IndexSlice[:, ['Test_Accuracy']])
COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS
Highlight the maximum value in each column:
Out[68]:
Model Test_Accuracy Precision f1_score recall_score
0 Voting Classifier 93.10% 95.52% 93.00% 90.61%
1 Bagging Classifier 95.00% 93.68% 95.14% 96.64%
2 Random Forest 95.30% 93.97% 95.43% 96.94%
3 Ada Boost 93.60% 95.47% 93.55% 91.70%
4 Gradient Boosting 90.15% 91.62% 90.11% 88.64%
5 XG Boost 93.25% 95.25% 93.19% 91.21%
In [69]:
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
Type here to search
32°C Light rain
ENG
12:09 PM
06-05-2022


Embedded Feature Selection(Ensemble Learning methods) Last Checkpoint: 04/08/2022 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted Python 3
Run Code
In [69]:
# Highlight the Max values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_max(s):
'''
highlight the maximum in a Series green.
'''
is_max = s == s.max()
return ['background-color: lightBlue' if v else '' for v in is_max]
print("\nHighlight the maximum value in each column:")
result.style.apply(highlight_max,subset=pd.IndexSlice[:, ['Test_Accuracy',
'Precision','f1_score','recall_score']])
COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS
Highlight the maximum value in each column:
Out[69]:
Model Test_Accuracy Precision f1_score recall_score
0 Voting Classifier 93.10% 95.52% 93.00% 90.61%
1 Bagging Classifier 95.00% 93.68% 95.14% 96.64%
2 Random Forest 95.30% 93.97% 95.43% 96.94%
3 Ada Boost 93.60% 95.47% 93.55% 91.70%
4 Gradient Boosting 90.15% 91.62% 90.11% 88.64%
5 XG Boost 93.25% 95.25% 93.19% 91.21%
In [70]:
# Highlight the Min values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")
def highlight_min(s):
Type here to search
32°C Light rain
ENG
12:09 PM
06-05-2022



```
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")

def highlight_min(s):
    '''
    highlight the minimum in a Series green.
    '''
    is_min = s == s.min()
    return ['background-color:cyan ' if v else '' for v in is_min]

print("\nHighlight the minimum value in each column:")
result.style.apply(highlight_min,subset=pd.IndexSlice[:, ['Test_Accuracy']])
```

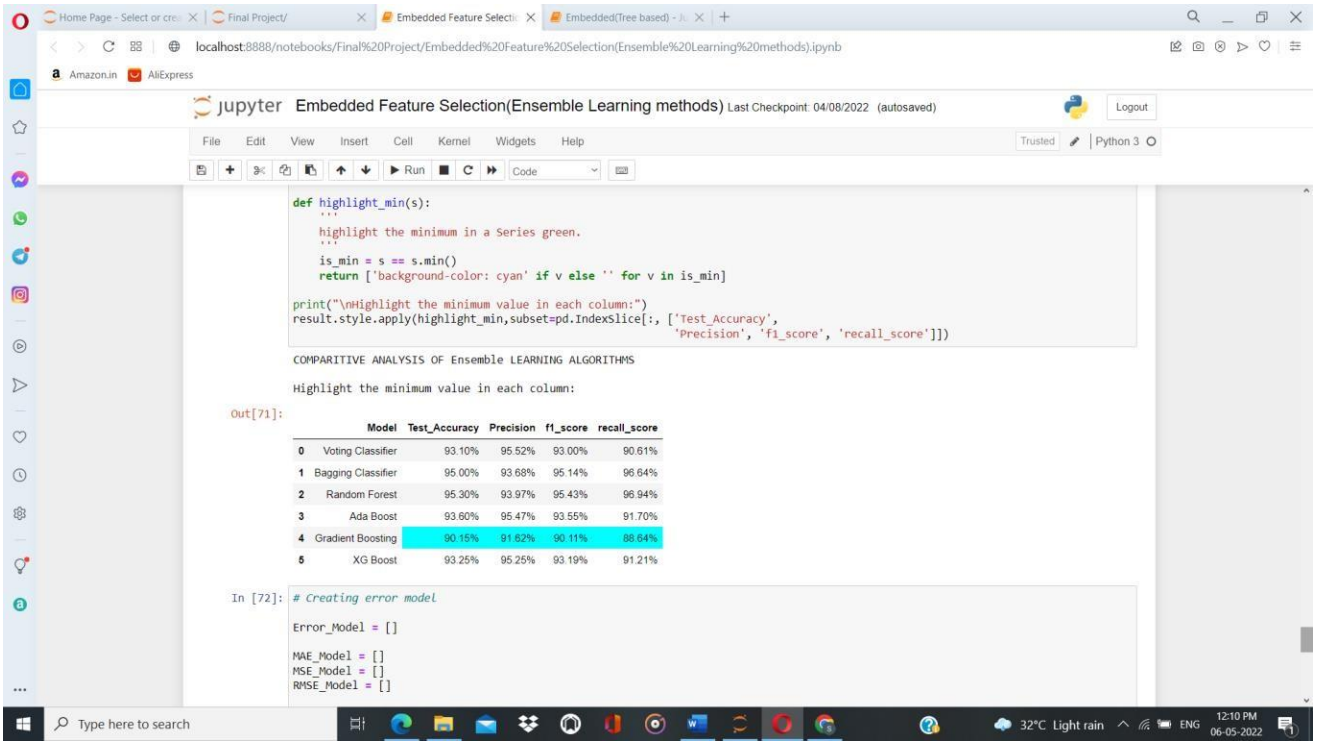
```
COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS

Highlight the minimum value in each column:
```

Out[70]:

	Model	Test_Accuracy	Precision	f1_score	recall_score
0	Voting Classifier	93.10%	95.52%	93.00%	90.61%
1	Bagging Classifier	95.00%	93.68%	95.14%	96.64%
2	Random Forest	95.30%	93.97%	95.43%	96.94%
3	Ada Boost	93.60%	95.47%	93.55%	91.70%
4	Gradient Boosting	90.15%	91.62%	90.11%	88.64%
5	XG Boost	93.25%	95.25%	93.19%	91.21%

In [71]:

```
# Highlight the Min values in each column
print("COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS")

def highlight_min(s):
    '''
    highlight the minimum in a Series green.
    '''
    is_min = s == s.min()
```



```
def highlight_min(s):
    '''
    highlight the minimum in a Series green.
    '''
    is_min = s == s.min()
    return ['background-color: cyan' if v else '' for v in is_min]

print("\nHighlight the minimum value in each column:")
result.style.apply(highlight_min,subset=pd.IndexSlice[:, ['Test_Accuracy',
                                                          'Precision', 'f1_score', 'recall_score']])
```

```
COMPARITIVE ANALYSIS OF Ensemble LEARNING ALGORITHMS

Highlight the minimum value in each column:
```

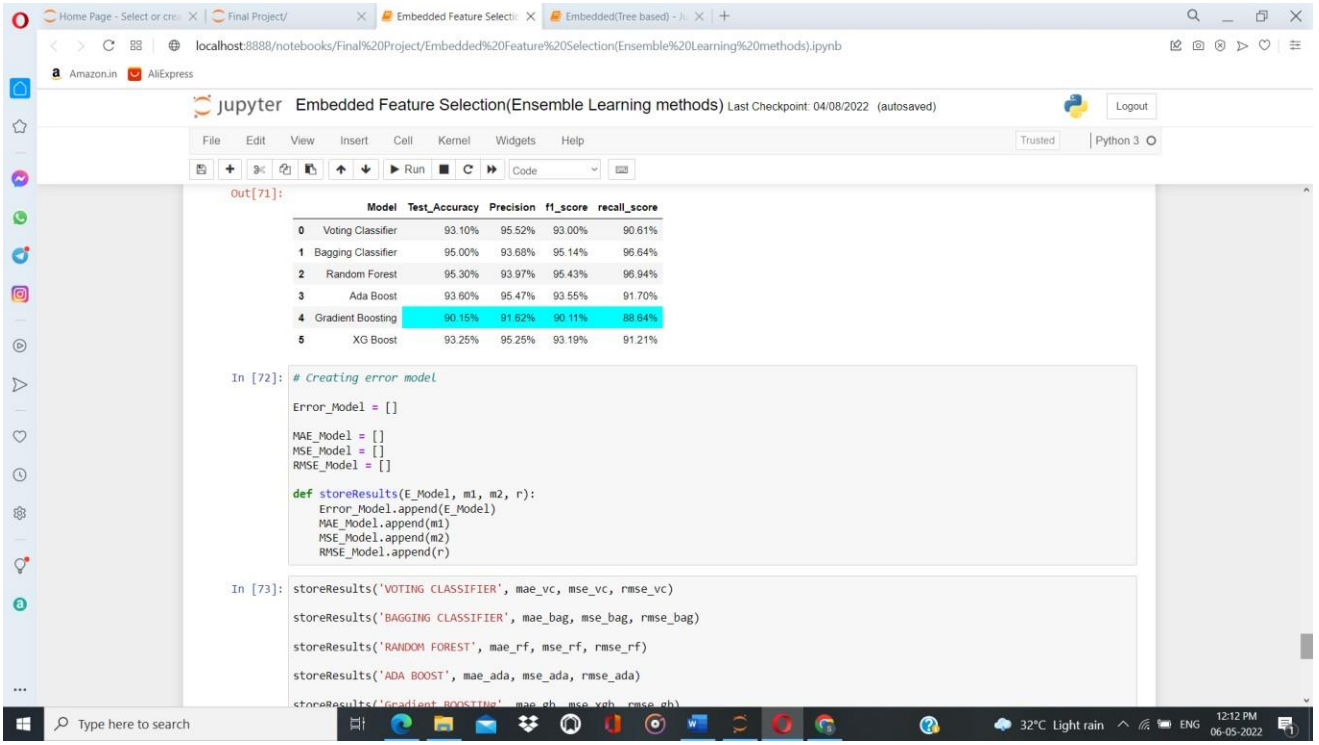
Out[71]:

	Model	Test_Accuracy	Precision	f1_score	recall_score
0	Voting Classifier	93.10%	95.52%	93.00%	90.61%
1	Bagging Classifier	95.00%	93.68%	95.14%	96.64%
2	Random Forest	95.30%	93.97%	95.43%	96.94%
3	Ada Boost	93.60%	95.47%	93.55%	91.70%
4	Gradient Boosting	90.15%	91.62%	90.11%	88.64%
5	XG Boost	93.25%	95.25%	93.19%	91.21%

In [72]:

```
# Creating error model

Error_Model = []

MAE_Model = []
MSE_Model = []
RMSE_Model = []
```

COMPARATIVE ANALYSIS OF ERROR RATE

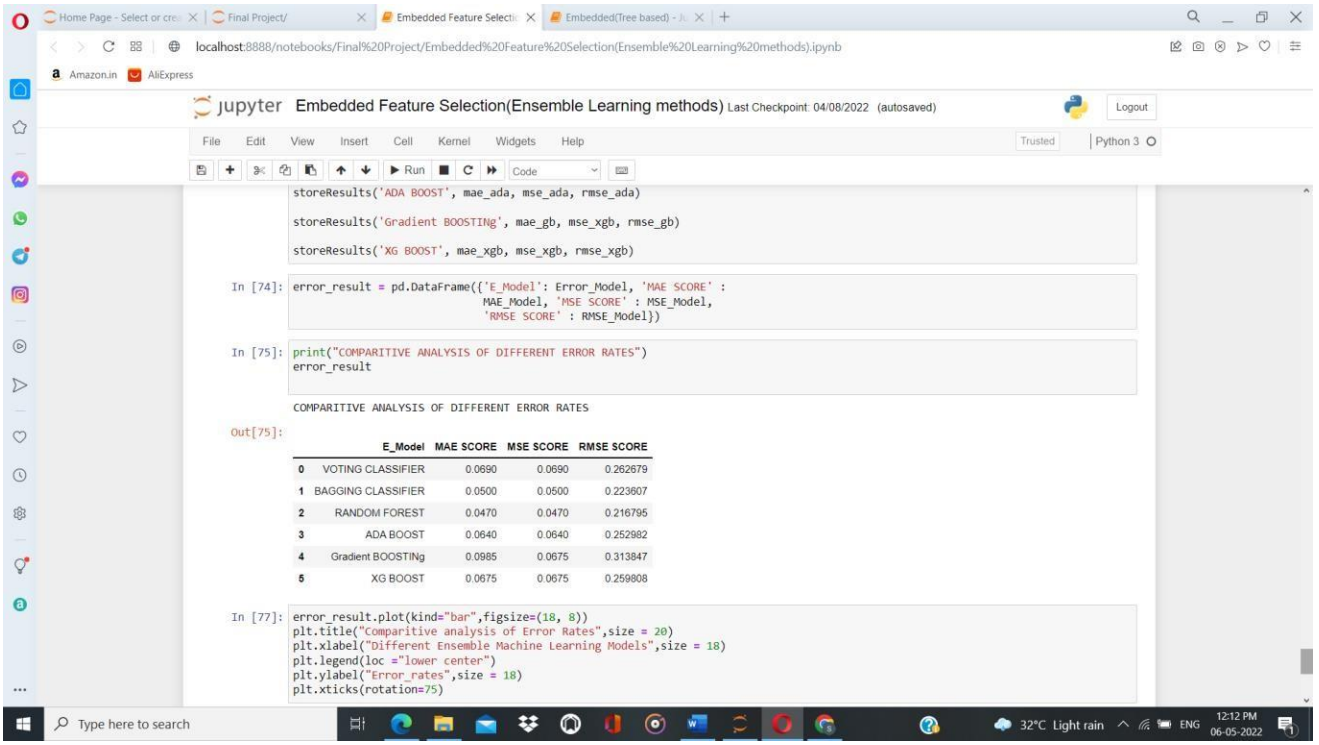


Out[71]:

	Model	Test_Accuracy	Precision	f1_score	recall_score
0	Voting Classifier	93.10%	95.52%	93.00%	90.61%
1	Bagging Classifier	95.00%	93.68%	95.14%	96.64%
2	Random Forest	95.30%	93.97%	95.43%	96.94%
3	Ada Boost	93.60%	95.47%	93.55%	91.70%
4	Gradient Boosting	90.15%	91.62%	90.11%	88.64%
5	XG Boost	93.25%	95.25%	93.19%	91.21%

In [72]:
```
# Creating error model

Error_Model = []

MAE_Model = []
MSE_Model = []
RMSE_Model = []

def storeResults(E_Model, m1, m2, r):
    Error_Model.append(E_Model)
    MAE_Model.append(m1)
    MSE_Model.append(m2)
    RMSE_Model.append(r)
```

In [73]:
```
storeResults('VOTING CLASSIFIER', mae_vc, mse_vc, rmse_vc)

storeResults('BAGGING CLASSIFIER', mae_bag, mse_bag, rmse_bag)

storeResults('RANDOM FOREST', mae_rf, mse_rf, rmse_rf)

storeResults('ADA BOOST', mae_ada, mse_ada, rmse_ada)
```



In [74]:
```
storeResults('ADA BOOST', mae_ada, mse_ada, rmse_ada)

storeResults('Gradient BOOSTINg', mae_gb, mse_xgb, rmse_gb)

storeResults('XG BOOST', mae_xgb, mse_xgb, rmse_xgb)
```

In [74]:
```
error_result = pd.DataFrame({'E_Model': Error_Model, 'MAE SCORE' :
                             MAE_Model, 'MSE SCORE' : MSE_Model,
                             'RMSE SCORE' : RMSE_Model})
```

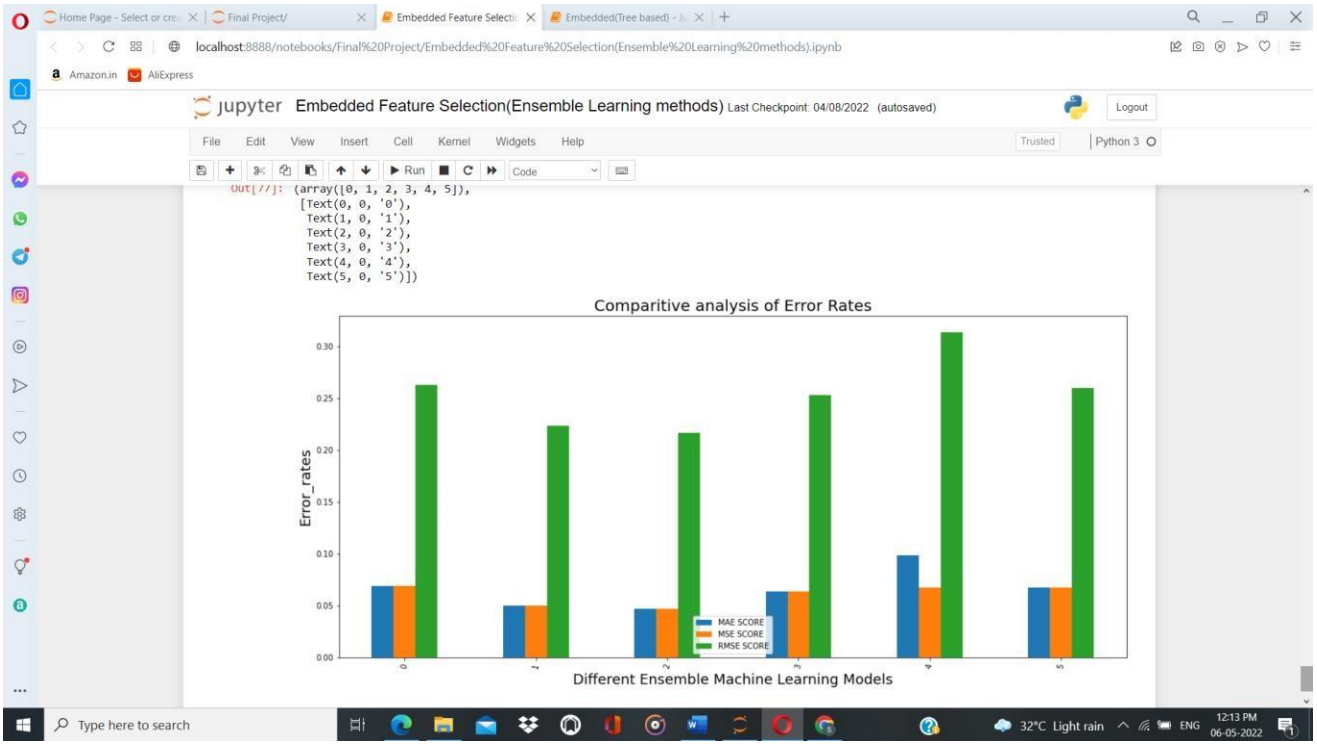
In [75]:
```
print("COMPARITIVE ANALYSIS OF DIFFERENT ERROR RATES")
error_result
```

```
COMPARITIVE ANALYSIS OF DIFFERENT ERROR RATES
```

Out[75]:

	E_Model	MAE SCORE	MSE SCORE	RMSE SCORE
0	VOTING CLASSIFIER	0.0690	0.0690	0.262679
1	BAGGING CLASSIFIER	0.0500	0.0500	0.223607
2	RANDOM FOREST	0.0470	0.0470	0.216795
3	ADA BOOST	0.0640	0.0640	0.252982
4	Gradient BOOSTINg	0.0985	0.0675	0.313847
5	XG BOOST	0.0675	0.0675	0.259808

In [77]:
```
error_result.plot(kind="bar",figsize=(18, 8))
plt.title("Comparitive analysis of Error Rates",size = 20)
plt.xlabel("Different Ensemble Machine Learning Models",size = 18)
plt.legend(loc ="lower center")
plt.ylabel("Error_rates",size = 18)
plt.xticks(rotation=75)
```


Embedded Feature Selection(Ensemble Learning methods)
Out[77]: (array([0, 1, 2, 3, 4, 5]),
[Text(0, 0, '0'),
Text(1, 0, '1'),
Text(2, 0, '2'),
Text(3, 0, '3'),
Text(4, 0, '4'),
Text(5, 0, '5')])
Comparitive analysis of Error Rates
Error_rates
0.30
0.25
0.20
0.15
0.10
0.05
0.00
MAE SCORE
MSE SCORE
RMSE SCORE
0
1
2
3
4
5
Different Ensemble Machine Learning Models