**Advance Software Engineering (SE-406)**

**LAB A1-G3**

**Laboratory Manual**



**Department of Software Engineering**

**DELHI TECHNOLOGICAL UNIVERSITY(DTU)**

Shahbad Daulatpur, Bawana Road, Delhi-110042

**Submitted to: - Submitted by:-**

Ms. Parul Sharma Name: ASHISH KUMAR

Roll number: 2K18/SE/041

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **EXPERIMENT** | **DATE** | **REMARKS** |
| **1.** | Identify a research paper (in software context only) and discuss the applications of formal methods for the same. | 09-01-2022 |  |
| **2.** | To create test cases from the requirement specification document (any case study) using the following Black Box Testing techniques. | 02-02-2022 |  |
| **3.** | To Compute Chidamber and Kemerer Metrics for a given source code. | 23-02-2022 |  |
| **4.** | To count Halstead metrics on a selected source code (searching/sorting). | 03-03-2022 |  |
| **5.** | Data Acquisition for software defect prediction. | 09-03-2022 |  |
| **6.** | Preprocessing and cleaning of data collected in experiment 5. | 09-03-2022 |  |
| **7.** | Training a support vector machine classifier to predict defect in the dataset collected in experiment 5. | 09-03-2022 |  |
| **8.** | To train and test fault prediction model using SVM. | 23-02-2022 |  |
| **9.** | To train and test fault prediction model using Decision tree**.** | 30-03-2022 |  |
| **10.** | To train and test fault prediction model using Random Forest. | 30-03-2022 |  |

**EXPERIMENT 1**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- Identify a research paper (in software context only) and discuss the applications of formal methods for the same.

**Introduction:-** I have chosen this research paper “**A FORMAL METHOD FOR MAPPING SOFTWARE ENGINEERING PRACTICES TO ESSENCE**” written by Murat Pasa Uysal

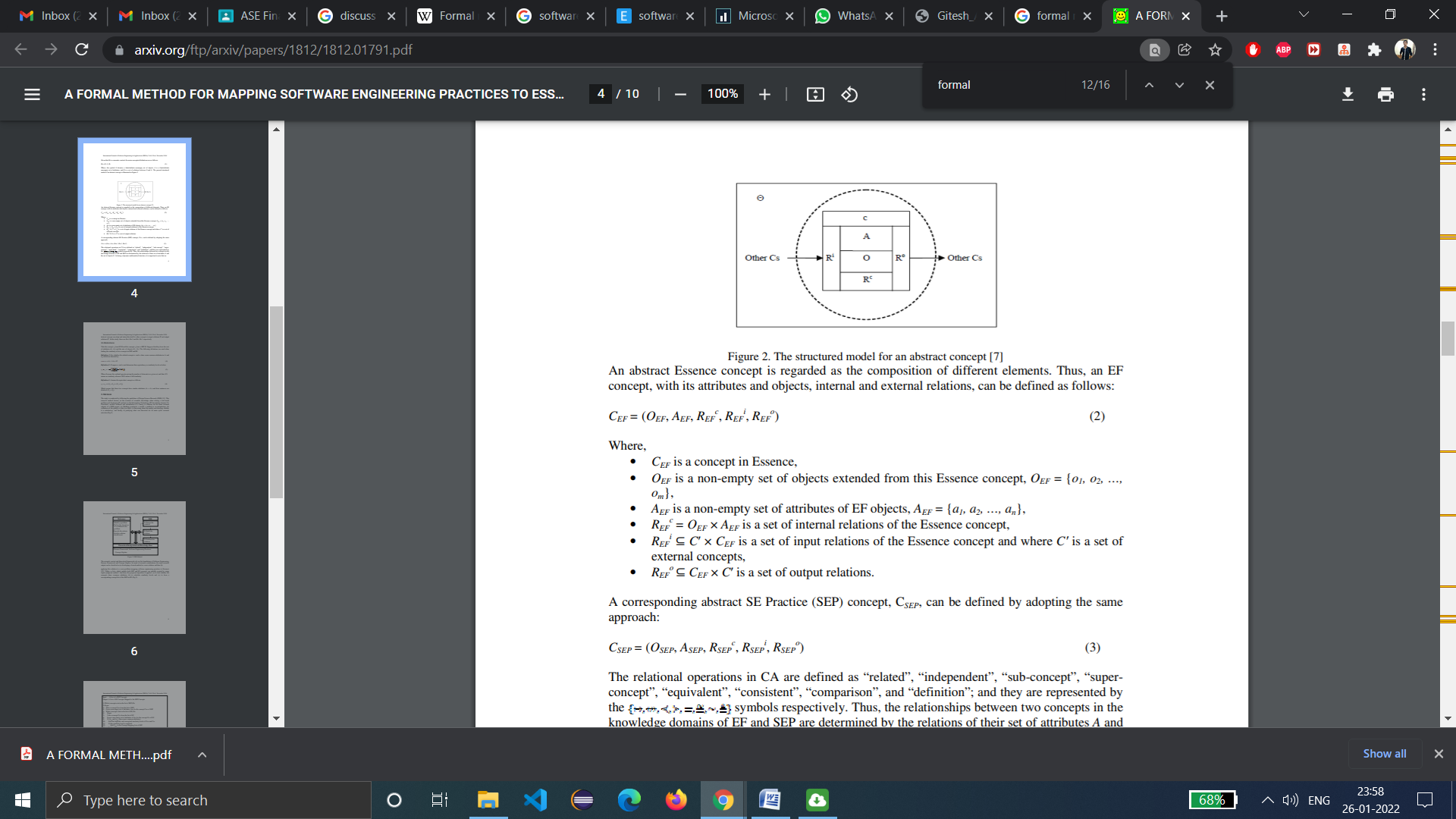
Here is the link: <https://arxiv.org/ftp/arxiv/papers/1812/1812.01791.pdf>

In this study the formal method employs an algorithm based on Concept Algebra and it is applied in a Scrum case study. In this study conceptual mapping and semantic evaluations is done using formal methods.

But first let’s talk about what are formal methods in software engineering?

The Encyclopedia of Software Engineering [MAR94] defines formal methods in the following manner: **Formal methods** used in developing computer systems are mathematically based techniques for describing system properties. Such formal methods provide frameworks within which people can specify, develop, and verify systems in a systematic, rather than ad hoc manner. A method is formal if it has a sound mathematical basis, typically given by a formal specification language. This basis provides a means of precisely defining notions like consistency and completeness, and more relevantly, specification, implementation and correctness.

EXAMPLE OF FORMAL METHOD USED in the paper – (set theory)



Below section shows how the formal mapping is applied in this study:

• The theoretical background of mapping is based on Concept Algebra principles and definitions. • A content analysis for the EF specification document and resources related to Scrum Practice is conducted.

• An attribute comparison list is created, which includes two sets of core attributes for the “Requirements” concept and “Product Backlog” concept.

• Note that a concept in linguistics is assumed as a noun or noun-phrase, which serves as the subject of a to-be statement. By using a Linguistic Typological Analysis (LTA) (assuming that a simple sentence is made of “subject”, “predicate” and “object” parts), an initial similarity level is determined on a scale ranging from 0 to 3.

• “0” level indicates no-typological similarity where none of the parts of two attributes is similar. “1” indicates that one similar part exists. “2” means that two of linguistic parts are similar. Finally, 3 points out a full linguistic similarity where both of the sentences have similar “subject”, “predicate” and “object” parts. Note that the level 2 or 3 is regarded as satisfactory for EF mapping procedures in this study.

**Learning from experiment:**- We have successfully learned about formal methods and its applications. This paper revealed interesting details about the benefits of formal analysis. The main argument of this paper is that formal methods can provide more accurate transformations as well as they can enable application of more systematic mapping procedures. In this study, a formal method is using an algorithm and CA definitions is proposed as a complete solution

**EXPERIMENT 2**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To create test cases from the requirement specification document(any case study) using the following Black Box Testing techniques.

**Introduction:-** I have chosen this requirement specification document “**E-Commerce System”** written by Andrew Blossom, Derek Gebhard, Steven Emelander, Robert Meyer.

Here is the link: <https://cse.msu.edu/~chengb/RE-491/Papers/SRS-BECS-2007.pdf>

This Software Requirements Specification (SRS) illustrates, in clear terms, the system’s primary uses and required functionality as specified by our customer.

But first let’s talk about what are SRS in software engineering?

The Software Requirements Specification is designed to document and describe the agreement between the customer and the developer regarding the specification of the software product requested. Its primary purpose is to provide a clear and descriptive “statement of user requirements” that can be used as a reference in further development of the software system.

Now in this experiment we need to write test cases using black box testing. **Black Box Testing** is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

## Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones –

* **Functional testing** – This black box testing type is related to the functional requirements of a system; it is done by software testers.
* **Non-functional testing**– This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

**Note:-** I am doing **Functional Testing.**

## General Test Cases for E-commerce system:

1. Verify that the user is able to navigate through all the products across different categories.
2. Verify that all the links and banners are redirecting to correct product/category pages and none of the links are broken.
3. Verify that the company logo is clearly visible.
4. Verify that all the text – product, category name, price, and product description are clearly visible.
5. Verify that all the images – product and banner are clearly visible.
6. Verify that category pages have a relevant product listed specific to the category.
7. Verify that the correct count of total products is listed on the category pages.
8. Search – Verify that on searching the product satisfying the search criteria are visible on the search result page.
9. Search – Verify the more relevant product for the search term is displayed on the top for a particular search term.
10. Search – Verify that count of products is correctly displayed on the search result page for a particular search term.
11. Filtering – Verify that filtering functionality correctly filters products based on the filter applied.
12. Filtering – Verify that filtering works correctly on category pages.
13. Filtering – Verify that filtering works correctly on the search result page.
14. Filtering – Verify that the correct count of total products is displayed after a filter is applied.

## Product Buy Flow – Test cases for E-commerce system:

1. Verify that on the product page, the user can select the desired attribute of the product e.g. size, color, etc.
2. Verify that the user can add to cart one or more products.
3. Verify that users can add products to the wishlist.
4. Verify that the user can buy products added to the cart after signing in to the application (or as per the functionality of the website).
5. Verify that the user can successfully buy more than one product that was added to his/her cart.
6. Verify that the user cannot add more than the available inventory of the product.
7. Verify that the limit to the number of products a user can by is working correctly by displaying an error message and preventing the user from buying more than the limit.
8. Verify that the delivery can be declined for the places where shipping is not available.
9. Verify that the Cash on Delivery / online payment option is working fine.
10. Verify that product return/refund functionality works fine.

## User Registration – Test cases:

1. Verify that all the specified fields are present on the registration page.
2. Verify that the required/mandatory fields are marked with \* against the field.
3. Verify that for better user interface dropdowns, radio buttons and checkboxes; etc fields are displayed wherever possible instead of just textboxes.
4. Verify the page has both submit and cancel/reset buttons at the end.
5. Verify that clicking submits button after entering all the required fields, submits the data to the server.
6. Verify that clicking cancels/reset button after entering all the required fields, cancels the submit request, and reset all the fields.
7. Verify that not filling the mandatory fields and clicking the submit button will lead to validation error.
8. Verify that not filling the optional fields and clicking the submit button will still send data to the server without any validation error.
9. Check the upper limit of the textboxes.
10. Check validation on the date and email fields (only valid dates and valid email Ids should be allowed.
11. Check validation on numeric fields by entering alphabets and special characters.
12. Verify that entering blank spaces on mandatory fields leads to validation error.

**SHOPPING CART TEST CASES:**

1. Website users can easily add/remove products to/from a shopping cart.
2. Users can add the same product multiple times and change their quantity in the cart directly.
3. It is possible to add the same products in different variations – color, size, etc.
4. Users can add products from different categories to the cart.
5. Items in the cart are displayed with correct names, images, and prices.
6. The items are clickable, and the links lead to corresponding product pages.
7. An order price updates when a user adds/removes a new item to/from the cart.
8. Shipping charges are included in the total price but displayed separately along with the other parameters.
9. If an item is out of stock, a user cannot add it to the cart.
10. When a user removes all items from the cart, nothing is displayed and the total price equals zero.
11. When a user closes a tab with a shopping cart, the items should remain in the cart.

**CHECKOUT FLOW TEST CASES:**

1. There is a prompt to log in or register before completing the purchase.
2. Customers can check out and pay as guest users without registration.
3. Returning logged-in users can use pre-saved shipping and billing information.
4. All types of supported payment methods work correctly.
5. Sensitive information, including payment details, isn’t stored after the payment.
6. An order confirmation page appears after a successful payment.
7. A user receives an order confirmation message via an email or text message. Order status is available in the account for registered users.
8. After completing the payment, a user is able to continue navigating through the site.

**UI TEST CASES:**

1. A user can enter their credentials in the signing and logging forms.
2. Users can type texts in the text fields, such as comments, reviews, etc.
3. All buttons are clickable and perform the programmed actions.
4. Breadcrumbs, tags, and other navigational elements are clickable and linked to the correct pages.
5. Toggles change their position after a click/slide, and each position is visually distinctive.
6. The performance and design of UI elements don’t vary depending on a user’s device.

**UX TEST CASES:**

1. Sorting functionality considers all the key parameters.
2. If search results go beyond one page, it is easy to navigate between the pages.
3. The font color and size make information easy to read.
4. Product descriptions are relevant and free of mistakes.
5. All category pages feature relevant products.
6. A user has a good overview of the page, without pop-up elements covering much information.
7. The design is consistent across different devices and screens, as well as the branding in general.

**Learning from experiment:**- We have successfully learnt about black box testing and its methods. We are successfully able to test e-commerce system by creating test cases of respective use case scenarios.

**EXPERIMENT 3**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To Compute Chidamber and Kemerer Metrics for a given source code.

**Introduction:-**

The Chidamber & Kemerer metrics suite originally consists of 6 metrics calculated for each class which are:

1. Weighted Methods Per Class (WMC)
2. Depth of Inheritance Tree (DIT)
3. Number of Children (NOC)
4. Coupling between Object Classes (CBO)
5. RFC and RFC ́ Response for a Class
6. Lack of Cohesion of Methods (LCOM1)

**1. Weighted Methods Per Class (WMC)**

Despite its long name, WMC is simply the method count for a class.

*WMC = number of methods defined in class*

A high WMC has been found to lead to more faults. Classes with many methods are likely to be more application specific, limiting the possibility of reuse. WMC is a predictor of how much time and effort is required to develop and maintain the class. A large number of methods also means a greater potential impact on derived classes, since the derived classes inherit (some of) the methods of the base class.

**2. Depth of Inheritance Tree (DIT)**

A high DIT has been found to increase faults. The deeper a class is in the hierarchy, the more methods and variables it is likely to inherit, making it more complex. Deep trees as such indicate greater design complexity.

Setting limit for DIT: An increase in DIT increases the density of bugs and decreases quality. A recommended DIT is 5 or less. The Visual Studio .NET documentation recommends that DIT <= 5 because excessively deep class hierarchies are complex to develop. Some sources allow up to 8.

**3. Number of Children (NOC)**

NOC measures the breadth of a class hierarchy, whereas maximum DIT measures the depth.

*NOC = number of immediate child classes derived from a base class.*

Depth is generally better than breadth, since it promotes reuse of methods through inheritance. NOC and DIT are closely related. Inheritance levels can be added to increase the depth and reduce the breadth.

A high NOC is an indicator of:

* High reuse of base class (Inheritance)
* Base class may require more testing.
* Improper abstraction of the parent class.
* Misuse of sub-classing.

Setting limit for NOC:

1. A redesign may be required for a class with a high NOC and a high WMC indicates complexity at the top of the class hierarchy. This type of class is potentially influencing a large number of descendant classes.
2. All classes need not have the same number of subclasses. Classes higher up in the hierarchy should have more subclasses than those lower down.

**4. Coupling between Object Classes (CBO)**

High CBO is undesirable. Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

Setting limit for CBO: In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A high coupling has been found to indicate fault-proneness.

**5. RFC and RFC ́ Response for a Class**

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC is simply the number of methods in the set.

*RFC = M + R (First-step measure)*

*RFC’ = M + R’ (Full measure)*

Where,

M = number of methods in the class

R = number of remote methods directly called by methods of the class

R’ = number of remote methods called, recursively through the entire call tree

Since RFC specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.

RFC is the original definition of the measure. It counts only the first level of calls outside of the class. RFC’ measures the full response set, including methods called by the callers, recursively, until no new remote methods can be found. If the called method is polymorphic, all the possible remote methods executed are included in R and R’.

Setting limit for RFC:

1. A large RFC has been found to indicate more faults. Classes with a high RFC are more complex and harder to understand. Testing and debugging is complicated. A worst case value for possible responses will assist in appropriate allocation of testing time.
2. An increase in RFC increases the density of bugs and decreases quality, though the optimum level has not been suggested by any researcher.

**6. Lack of Cohesion of Methods (LCOM1)**

In low cohesion only one class is responsible to execute lots of jobs that are not in common which reduces the chance of reusability and maintenance. But in high cohesion, there is a separate class for all the jobs to execute a specific job, which results in better usability and maintenance.

* High cohesion is when you have a class that does a well-defined job. Low cohesion is when a class does a lot of jobs that don’t have much in common.
* High cohesion gives us better-maintaining facility and Low cohesion results in monolithic classes that are difficult to maintain, understand and reduce re-usability

**Source Code(in C++):**

#include<bits/stdc++.h>

using namespace std;

class employee{

public:

int empid;

string emailid;

vector<string> getmeetinglist(){

}

void getinfo(){

}

};

class SDE: public employee{

public:

SDE(int eid,string email){

empid=eid;

emailid=email;

}

string managername;

string businessunitid;

bool ishandlingsecurity(){

}

int getdepid(){

string info = getinfo();

}

};

int main(){

SDE ashish(1,"abc@gmail.com");

ashish.businessunitid=2;

ashish.getdepid();

ashish.getmeetinglist();

ashish.getinfo();

return 0;

}

**Result:-**

1. WMC

There are two class name “SDE” and “employee”. In employee class, there are 2 methods & 2 attributes whereas in SDE (i.e. derived class) there are 2 attributes & 2 methods.

**So, Total WMC = Total methods in class = 2 + 2 = 4**

1. DIT

Since SDE is the derived class, so its depth would be 1 and for employee which is the base class, its depth would be 0.

**So, Total DIT = 0 + 1 = 1**

1. NOC

Since NOC is the opposite of DIT. For SDE class, NOC is 0 and for employee, NOC is 1.

**So, Total NOC = 0 + 1 = 1**

1. CBO

CBO is defined as “Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class”. In SDE class, I have used one method which is defined in base class i.e. employee.

So, **CBO would be 1 and these two classes are coupled.**

1. RFC

Since RFC is simply the number of methods in the class, so in employee class, there are 2 methods whereas in SDE (i.e. derived class) there are 2 methods.

1. LCOM

For LCOM calculation:

M1= {0} //since all the 4 methods are not using any attributes

M2= {0}

M3= {0}

M4= {0}

**LCOM= no. of ones – no. of zeroes = 0 – 4 = 0**

**Learning from experiment:**- We have successfully learnt about Chidamber and Kemerer Metrics and we compute the 6 metrics of a given source code.

**EXPERIMENT 4**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To count Halstead metrics on a selected source code(searching/sorting).

**Introduction:-**

Halstead’s metrics are included in a number of current commercial tools that count software lines of code. By counting the tokens and determining which operators are and which are operands, the following base measures can be collected:

n1 = Number of distinct operators.

n2 = Number of distinct operands.

N1 = Total number of occurrences of operators.

N2 = Total number of occurrences of operands.

In addition to the above, Halstead defines the following:

n1\* = Number of potential operators.

n2\* = Number of potential operands.

Halstead refers to n1\* and n2\* as the minimum possible number of operators and operands for a module and a program respectively.

**Halstead metrics are:**

1. Halstead Program Length – The total number of operator occurrences and the total number of operand occurrences.

**N = N1 + N2**

And estimated program length is, **N^ = n1 log2n1 + n2 log2n2**

1. Halstead Vocabulary – The total number of unique operator and unique operand occurrences.

**n = n1 + n2**

1. Program Volume – Proportional to program size represents the size, in bits, of space necessary for storing the program. This parameter is dependent on specific algorithm implementation. The properties V, N, and the number of lines in the code are shown to be linearly connected and equally valid for measuring relative program size.

**V = N \* log2 (n)**

1. Program Difficulty – This parameter shows how difficult to handle the program is.

**D = (n1 / 2) \* (N2 / n2)**

As the volume of the implementation of a program increases, the program level decreases and the difficulty increases. Thus, programming practices such as redundant usage of operands, or the failure to use higher-level control constructs will tend to increase the volume as well as the difficulty.

1. Programming Effort – Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.

**E = D \* V = Difficulty \* Volume**

Advantages of Halstead Metrics:

* It is simple to calculate.
* It measures overall quality of the programs.
* It predicts the rate of error.
* It does not require the full analysis of programming structure.
* It is useful in scheduling and reporting projects.
* It can be used for any programming language.

Disadvantages of Halstead Metrics:

* It depends on the complete code.
* It has no use as a predictive estimating model.

**Source Code (in C++):**

int sort (int x[ ], int n)

{

int i, j, save, im1;

/\*This function sorts array x in ascending order \*/

If (n< 2) return 1;

for (i=2; i< =n; i++)

{

im1=i-1;

for (j=1; j< =im1; j++)

if (x[i] < x[j])

{

Save = x[i];

x[i] = x[j];

x[j] = save;

}

}

return 0;

}

**Result:-**

|  |  |  |  |
| --- | --- | --- | --- |
| **operators** | **occurrences** | **operands** | **occurrences** |
| int | 4 | sort | 1 |
| () | 5 | x | 7 |
| , | 4 | n | 3 |
| [] | 7 | i | 8 |
| if | 2 | j | 7 |
| < | 2 | save | 3 |
| ; | 11 | im1 | 3 |
| for | 2 | 2 | 2 |
| = | 6 | 1 | 3 |
| – | 1 | 0 | 1 |
| <= | 2 | – | – |
| ++ | 2 | – | – |
| return | 2 | – | – |
| {} | 3 | – | – |
| **n1=14** | **N1=53** | **n2=10** | **N2=38** |

[Table: List of the operators and operands]

Here N1=53 and N2=38.

1. **The program length N=N1+N2**

= 53+38=91

1. **Vocabulary of the program** **n=n1+n2**

=14+10=24

1. **Volume V= N \* log2N**

= 91 \* log2 24 = 417 bits.

1. **Purity Ratio = N^ / N**

**Where, N^ = n1 log2 (n1) + n2 log2 (n2)**

N^ = 14 log2 (14) + 10 log2 (10) = 86.498

So, Purity ratio = 86.498 / 91 = 0.95

1. **Program Difficulty, D= (n1 / 2) \* (N2 / n2)**

= (14 / 2) \* (38 / 10) = 26.6

1. **Effort, E = D \* V**

= 26.6 \* 417 = 11092.2 PM

**Learning from experiment:**- We have successfully learnt about Halstead Metrics and we compute the metrics of a given source code written in C++ language. We listed out the advantages and disadvantages of Halstead Metrics as well.

**EXPERIMENT 5**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- Data Acquisition for software defect prediction.

**About dataset:-**

Link to dataset: <http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff>

This is a PROMISE data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering.

1. Sources:

-- Creators: NASA, then the NASA Metrics Data Program,

-- http://mdp.ivv.nasa.gov. Contacts: Mike Chapman, Galaxy Global Corporation

-- Donor: Tim Menzies (tim@barmag.net)

-- Date: December 2 2004

2. Number of instances: 10885

3. Number of attributes: 22 (5 different lines of code measure, 3 McCabe metrics, 4 base   
 Halstead measures, 8 derived Halstead measures, a branch-count, and 1 goal field)

4. Attribute Information:

1. loc : numeric % McCabe's line count of code

2. v(g) : numeric % McCabe "cyclomatic complexity"

3. ev(g) : numeric % McCabe "essential complexity"

4. iv(g) : numeric % McCabe "design complexity"

5. n : numeric % Halstead total operators + operands

6. v : numeric % Halstead "volume"

7. l : numeric % Halstead "program length"

8. d : numeric % Halstead "difficulty"

9. i : numeric % Halstead "intelligence"

10. e : numeric % Halstead "effort"

11. b : numeric % Halstead

12. t : numeric % Halstead's time estimator

13. lOCode : numeric % Halstead's line count

14. lOComment : numeric % Halstead's count of lines of comments

15. lOBlank : numeric % Halstead's count of blank lines

16. lOCodeAndComment : numeric

17. uniq\_Op : numeric % unique operators

18. uniq\_Opnd : numeric % unique operands

19. total\_Op : numeric % total operators

20. total\_Opnd : numeric % total operands

21: branchCount : numeric % of the flow graph

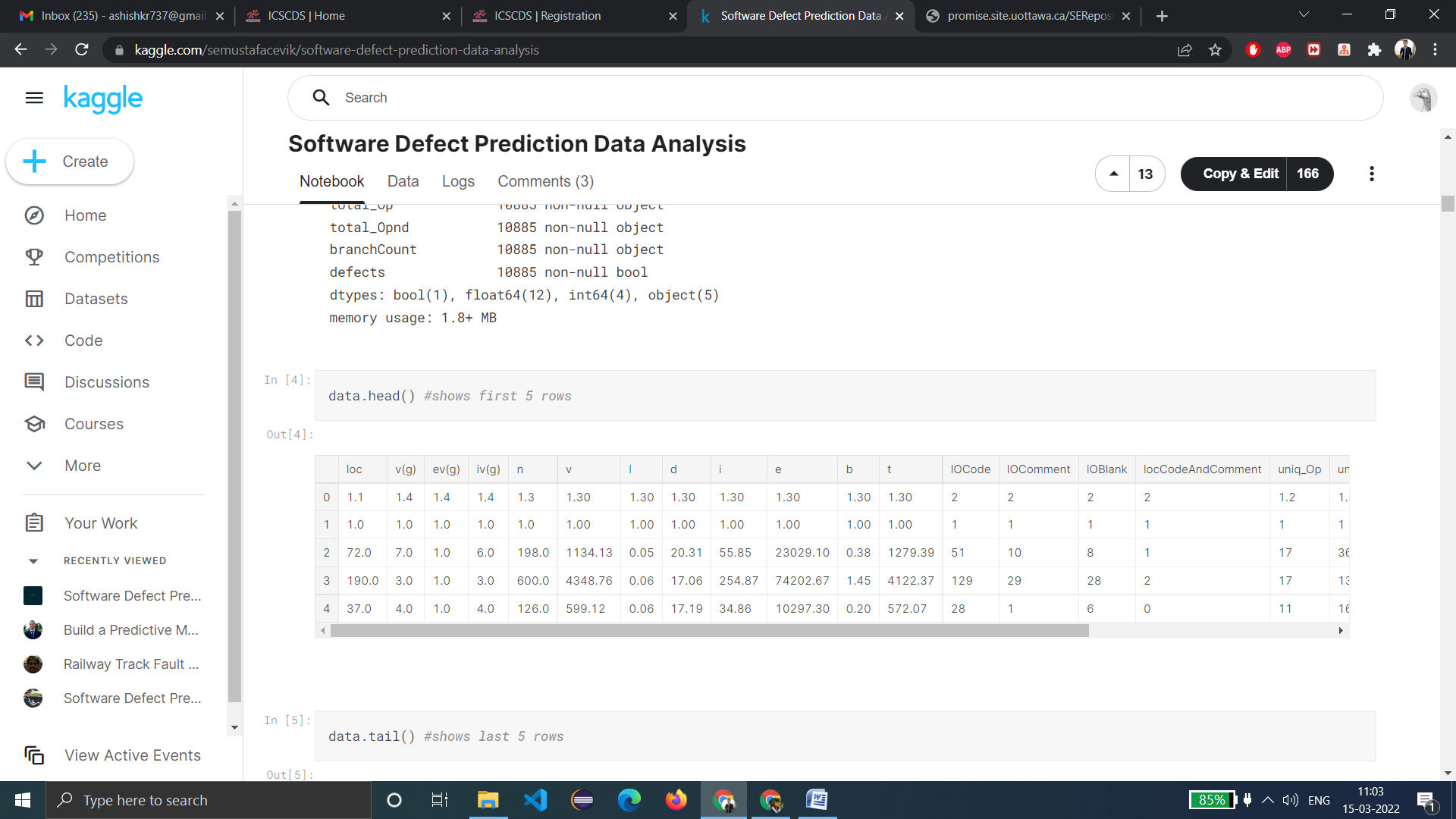
22. defects : {false,true} % module has/has not one or more reported defects

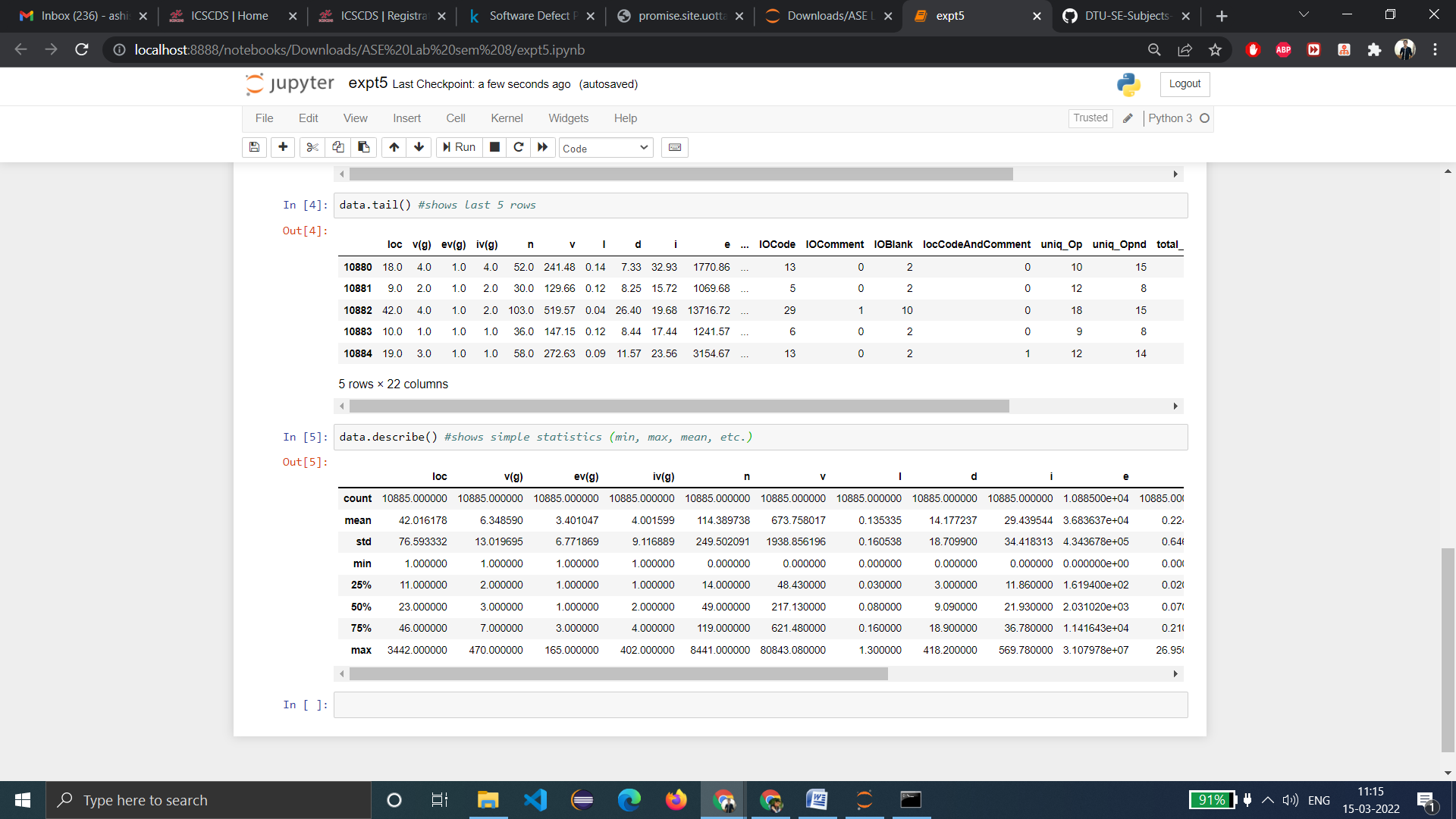
5. Missing attributes: none

6. Class Distribution: the class value (defects) is discrete

false: 2106 = 19.35%

true: 8779 = 80.65%

****

****

**Learning from experiment:**- We are successful in finding a defect dataset and we analyzed it properly.

**EXPERIMENT 6**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- Preprocessing and cleaning of data collected in experiment 5.

**Introduction:- Data Preprocessing** is the most important step when we are building our model. In **Data Preprocessing**step, the data is transformed into a form where it becomes suitable for model ingestion. **Data Cleaning** is the process of analyzing data for finding incorrect, corrupt, and missing values and ablating it to make it suitable for input to data analytics and various machine learning algorithms.

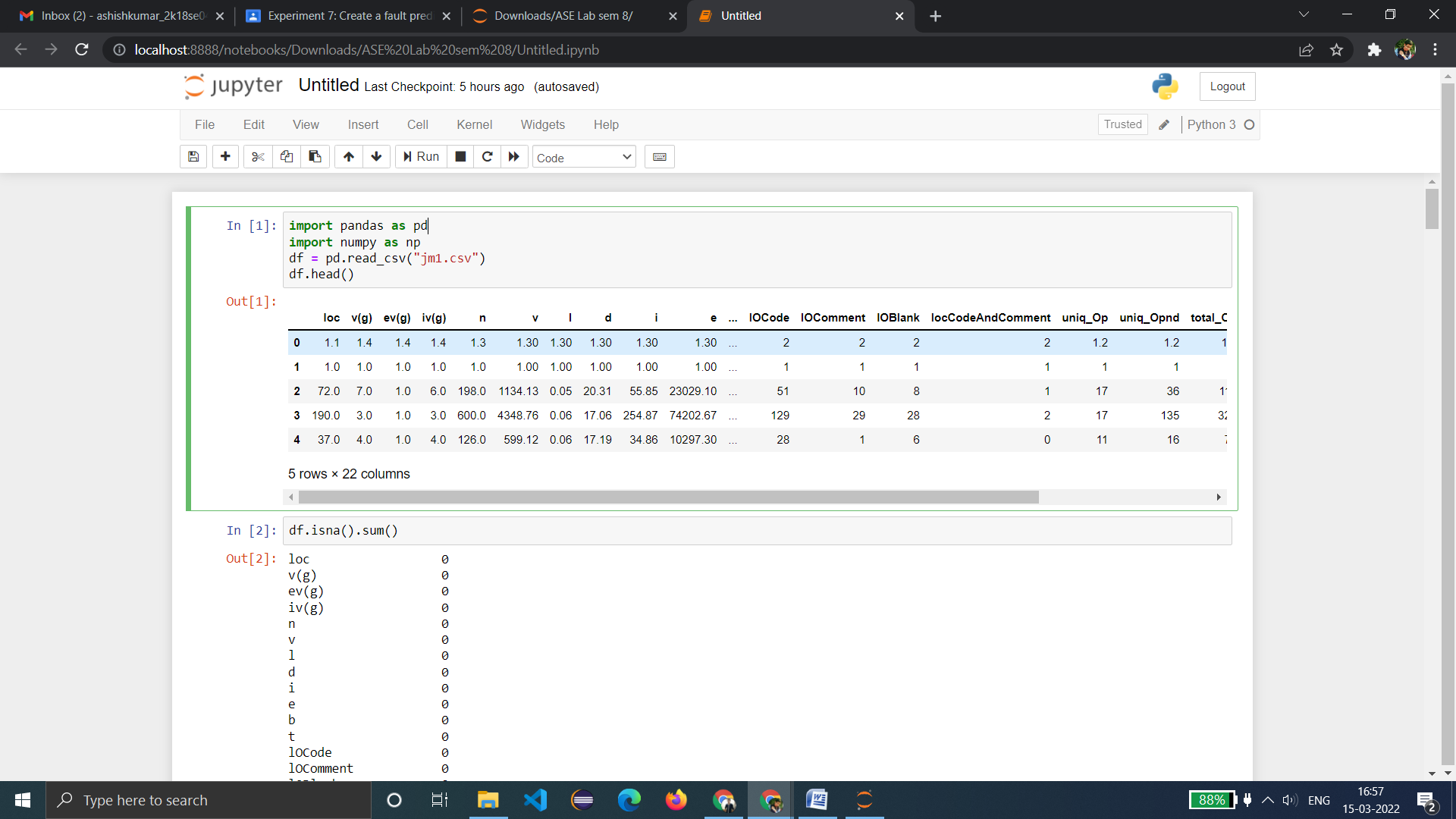
**Code & Output:-**

import pandas as pd

import numpy as np

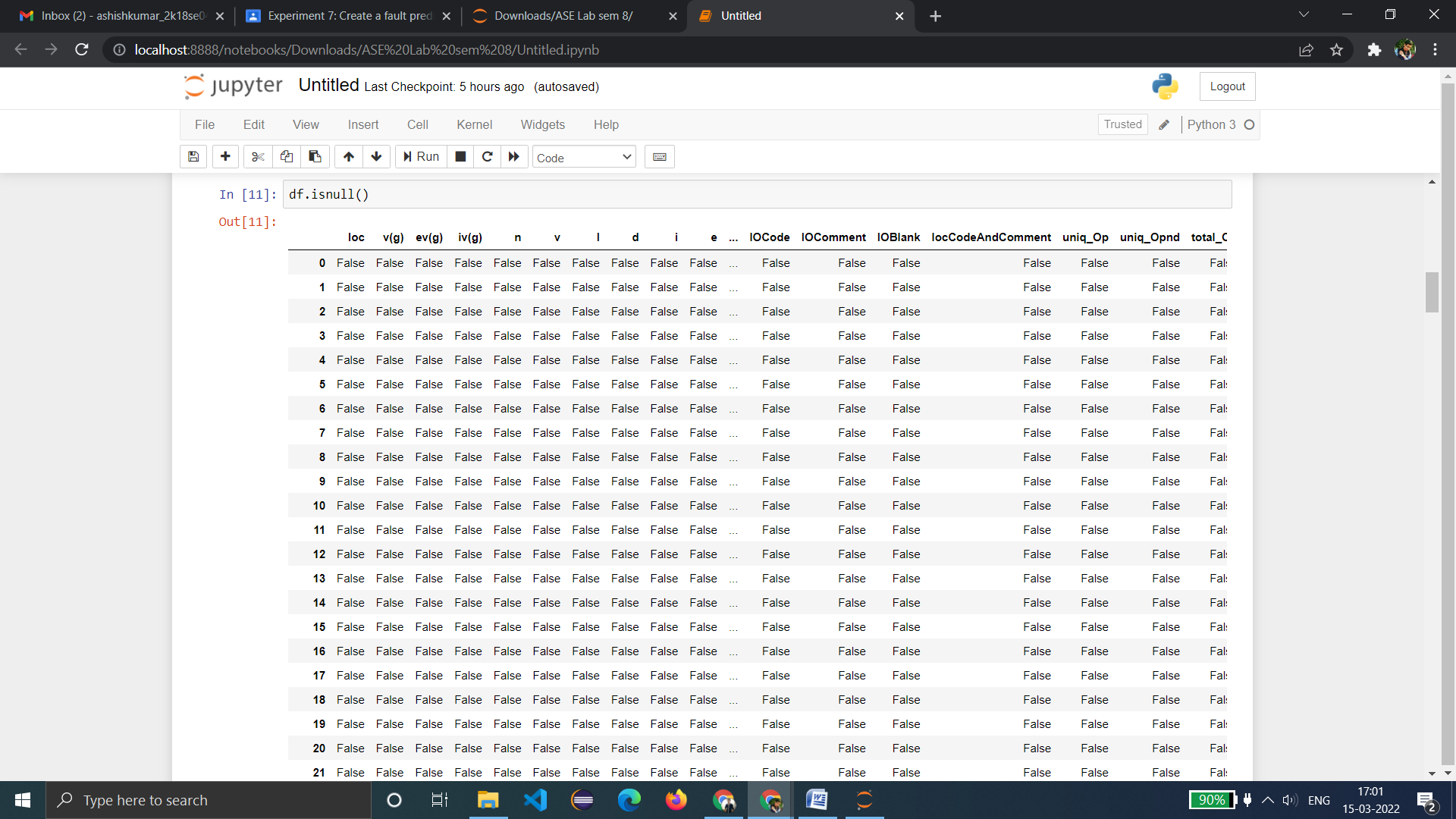
df = pd.read\_csv("jm1.csv")

df.head()



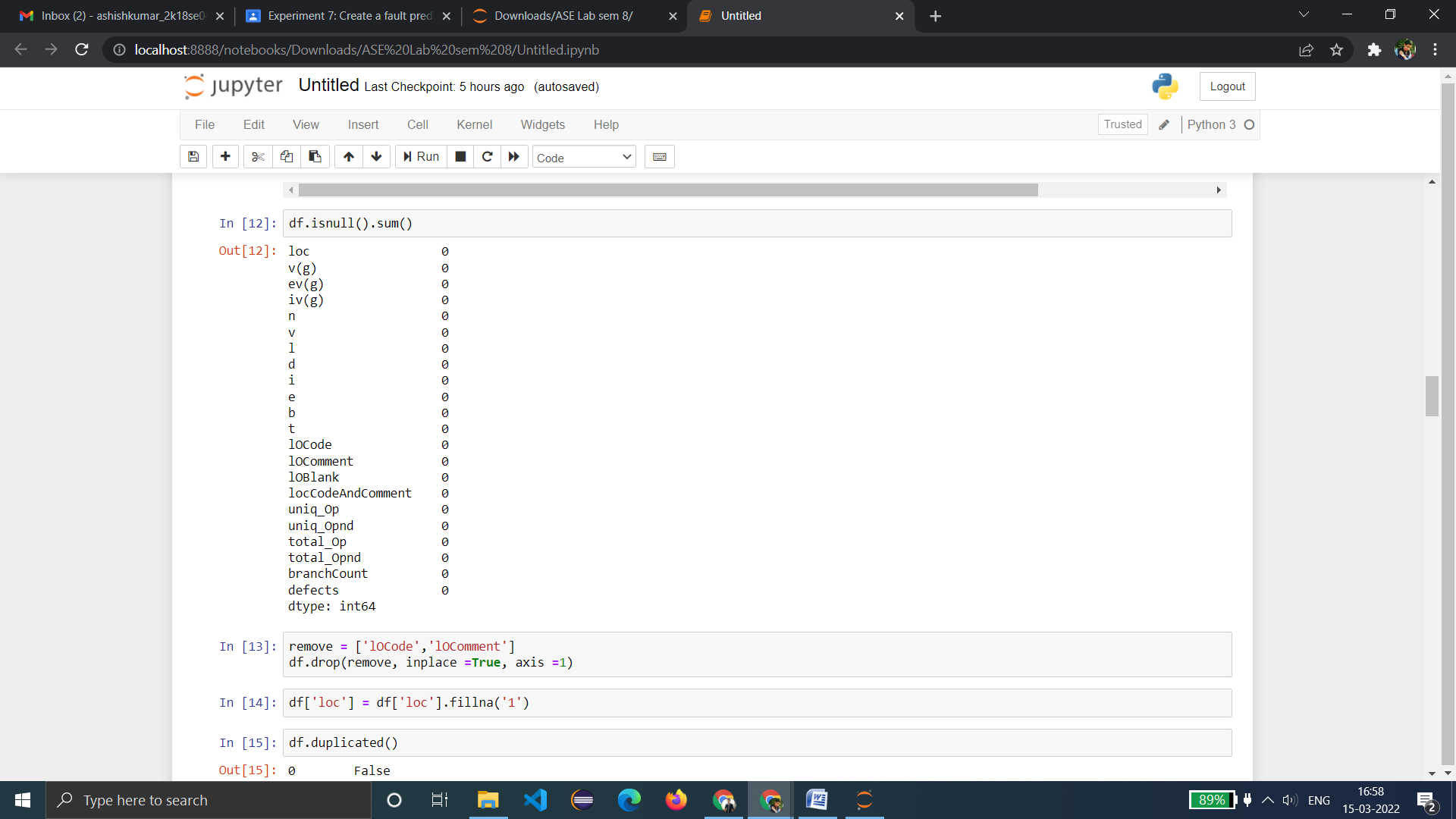
Note: 'isnull' helps us find where in our dataset there are missing values. This is useful information as this is what we need to correct while data cleaning.

df.isnull()



df.isnull().sum() #shows how many of the null values

#it shows zero missing values.

****

**Result:**- Since there are no missing value, no **data cleaning needed** because the data is all important.

**Learning from experiment:**- We are successful in Preprocessing and cleaning of a dataset collected in experiment 5 i.e. NASA Metrics Data Program.

**EXPERIMENT 7**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- Training a support vector machine classifier to predict defect in the dataset collected in experiment 5.

**Introduction:-** A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classiﬁcation and regression. SVMs are known as maximum margin classiﬁers as they ﬁnd the best separating hyperplane between two classes. This process can also be applied recursively to allow the separation of any number of classes. Only those data points that are located nearest to this dividing hyperplane, known as the support vectors, are used by the classiﬁer. This enables SVMs to be used successfully with both large and small data sets.

The aim of this experiment is to observe the classiﬁcation performance of the Support Vector Machine (SVM) for defect prediction in the context of data sets from the NASA Metrics Data Program (MDP) repository; a collection of data sets generated from NASA software systems and intended for defect prediction research.

**Code & Output:-**

# Importing the dataset

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('jm1.csv')

X = dataset.iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

dataset.head(5)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25)

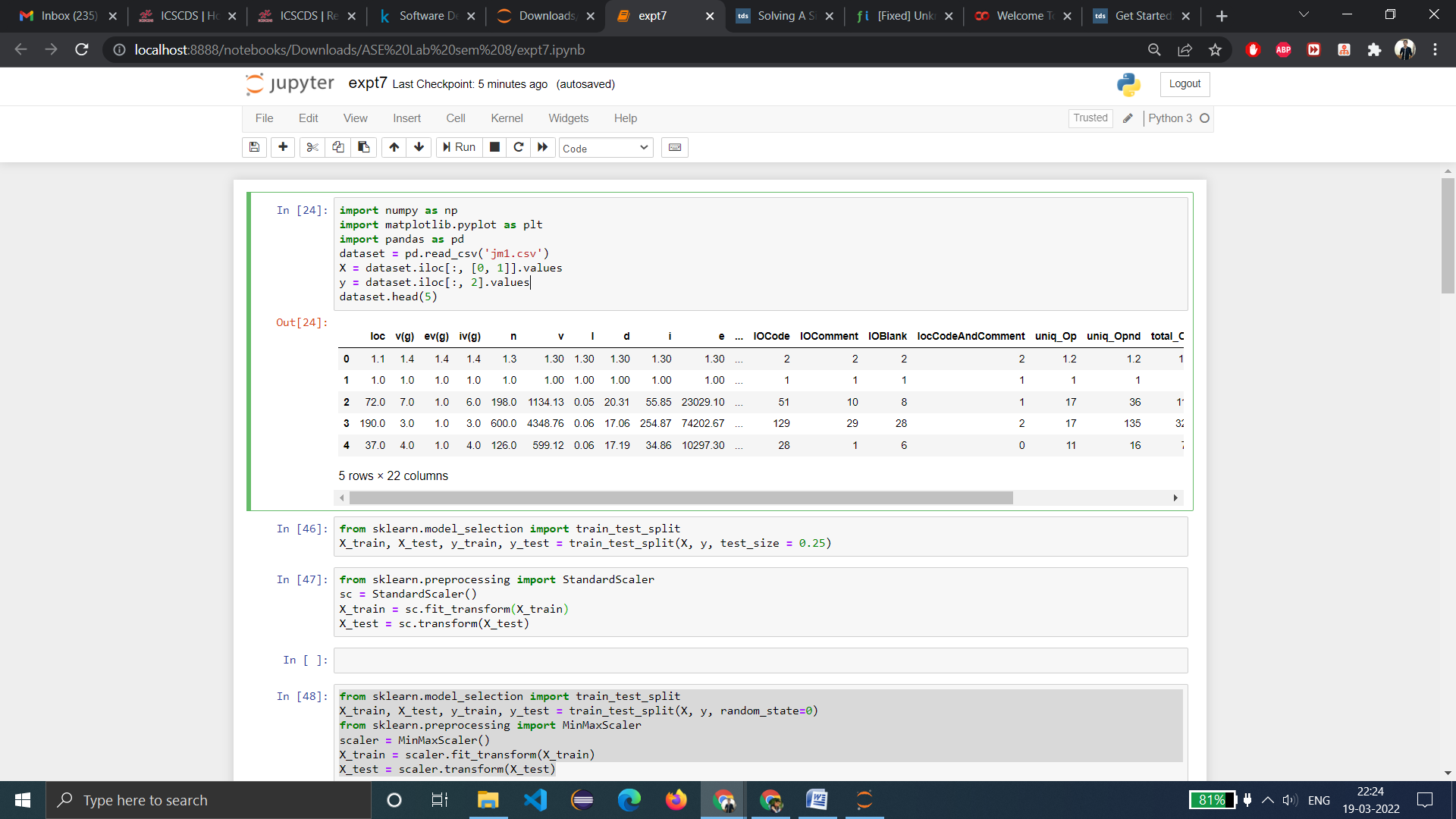
# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)



# Create Training and Test Sets and Apply Scaling

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Scatter-matrix for each input variable

from pandas.tools.plotting import scatter\_matrix

from matplotlib import cm

feature\_names = ['n','v','l','d']

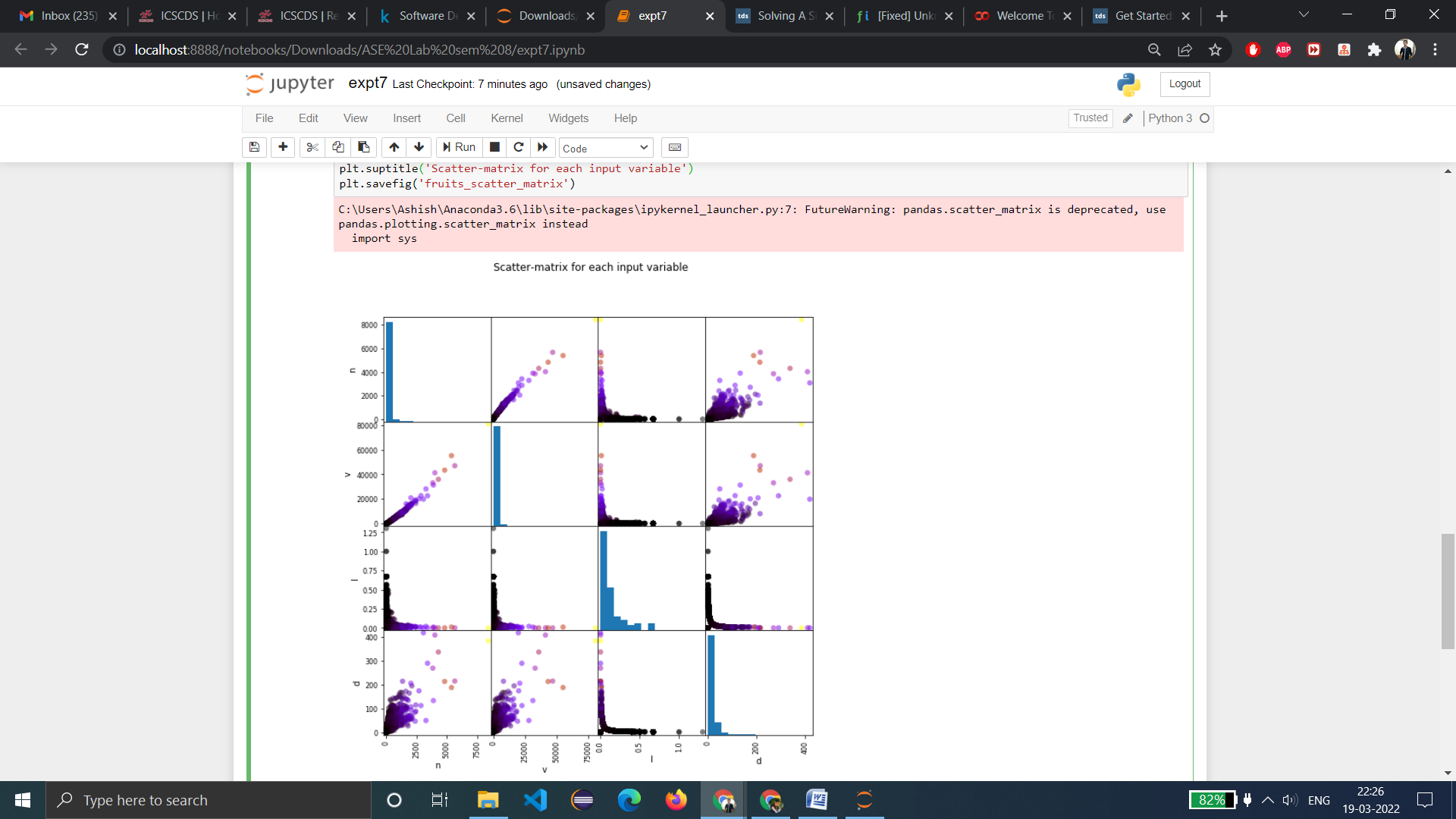
X = dataset[feature\_names]

y = dataset['loc']

cmap = cm.get\_cmap('gnuplot')

scatter = pd.scatter\_matrix(X, c = y, marker = 'o', s=40, hist\_kwds={'bins':15}, figsize=(9,9), cmap = cmap)

plt.suptitle('Scatter-matrix for each input variable')



# #Applying SVM Classifier on the Training Set

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn import svm

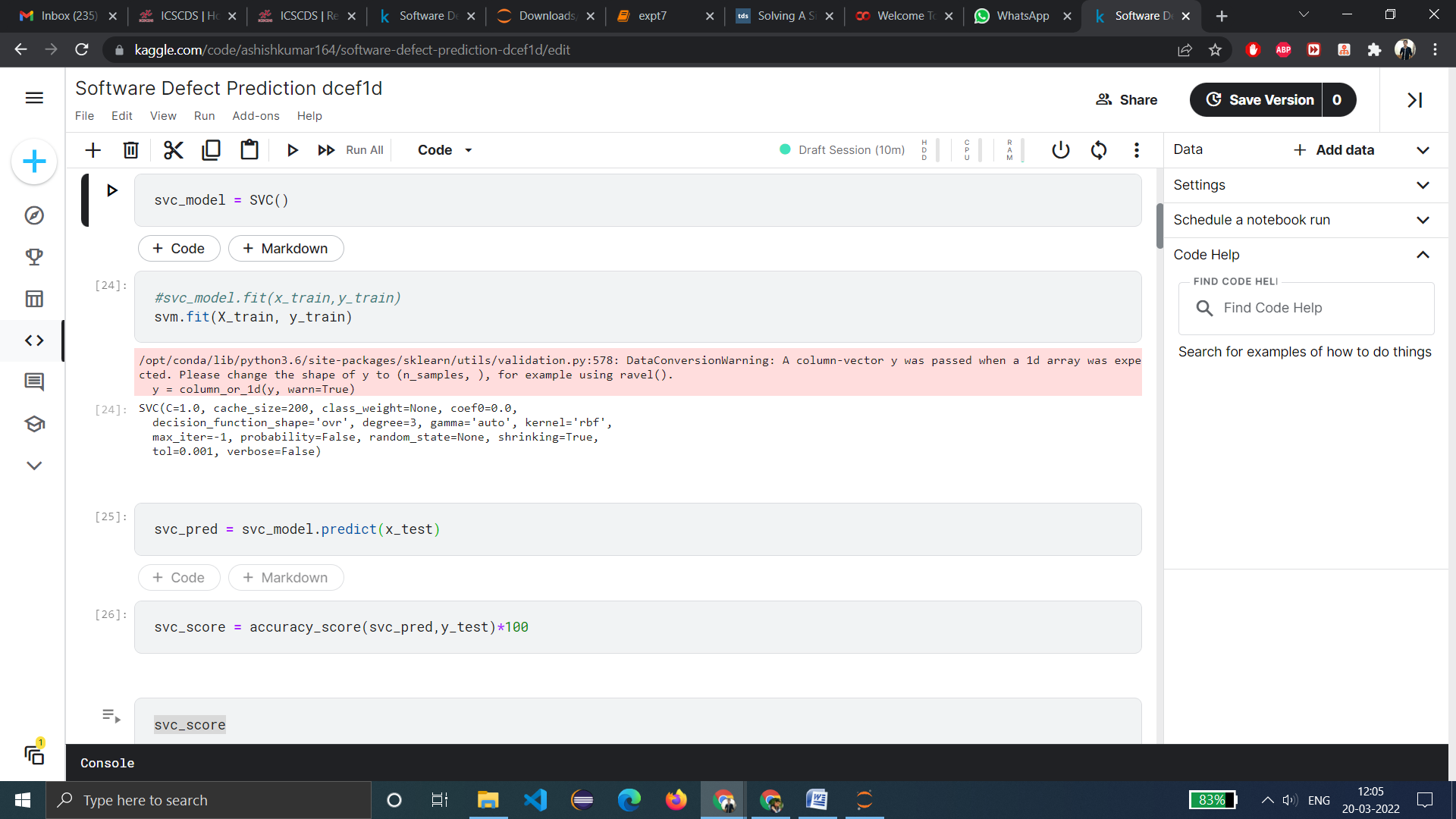
from sklearn.preprocessing import StandardScaler

from sklearn.svm import LinearSVC, SVC

svm= SVC()

svm.fit(X\_train, y\_train)

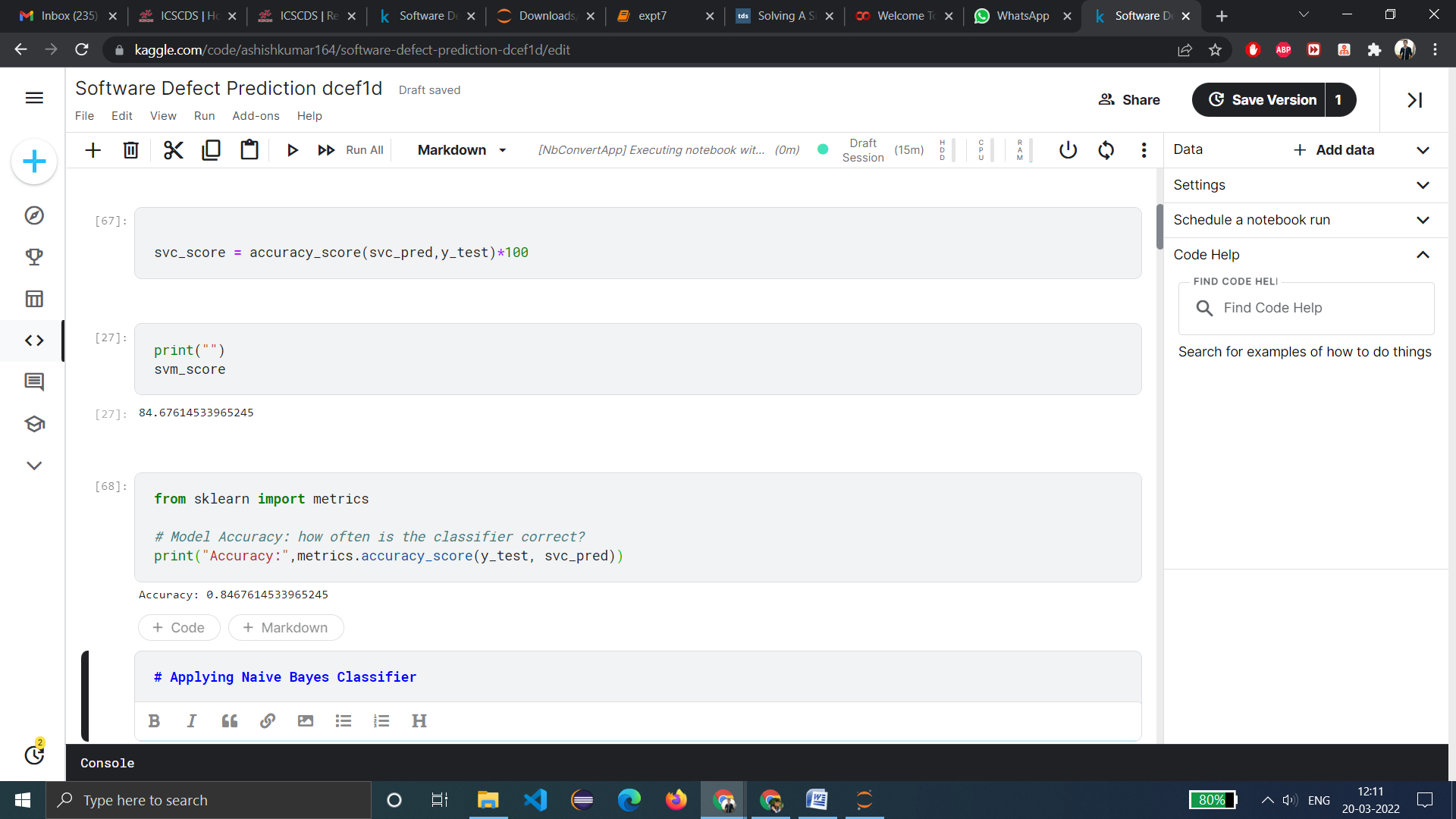
svm\_pred = svm.predict(X\_test)



# Model Accuracy: how often is the classifier correct?

from sklearn import metrics

print("Accuracy:",metrics.accuracy\_score(y\_test, svm\_pred))



**Result:**- The accuracy of the SVM Classifier comes out to be 84.67%.

**Learning from experiment:**- We have successfully been able to build a **SVM Classification**Model that is able to predict defect in the dataset.

**EXPERIMENT 8**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To train and test fault prediction model using SVM.

**Introduction:-** A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classiﬁcation and regression. SVMs are known as maximum margin classiﬁers as they ﬁnd the best separating hyperplane between two classes. This process can also be applied recursively to allow the separation of any number of classes. Only those data points that are located nearest to this dividing hyperplane, known as the support vectors, are used by the classiﬁer. This enables SVMs to be used successfully with both large and small data sets.

The aim of this experiment is to observe the classiﬁcation performance of the Support Vector Machine (SVM) for fault prediction in the context of data sets from the NASA Metrics Data Program (MDP) repository; a collection of data sets generated from NASA software systems and intended for fault prediction research.

**Code & Output:-**

# Importing the dataset

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv('jm1.csv')

X = dataset.iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

dataset.head(5)

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25)

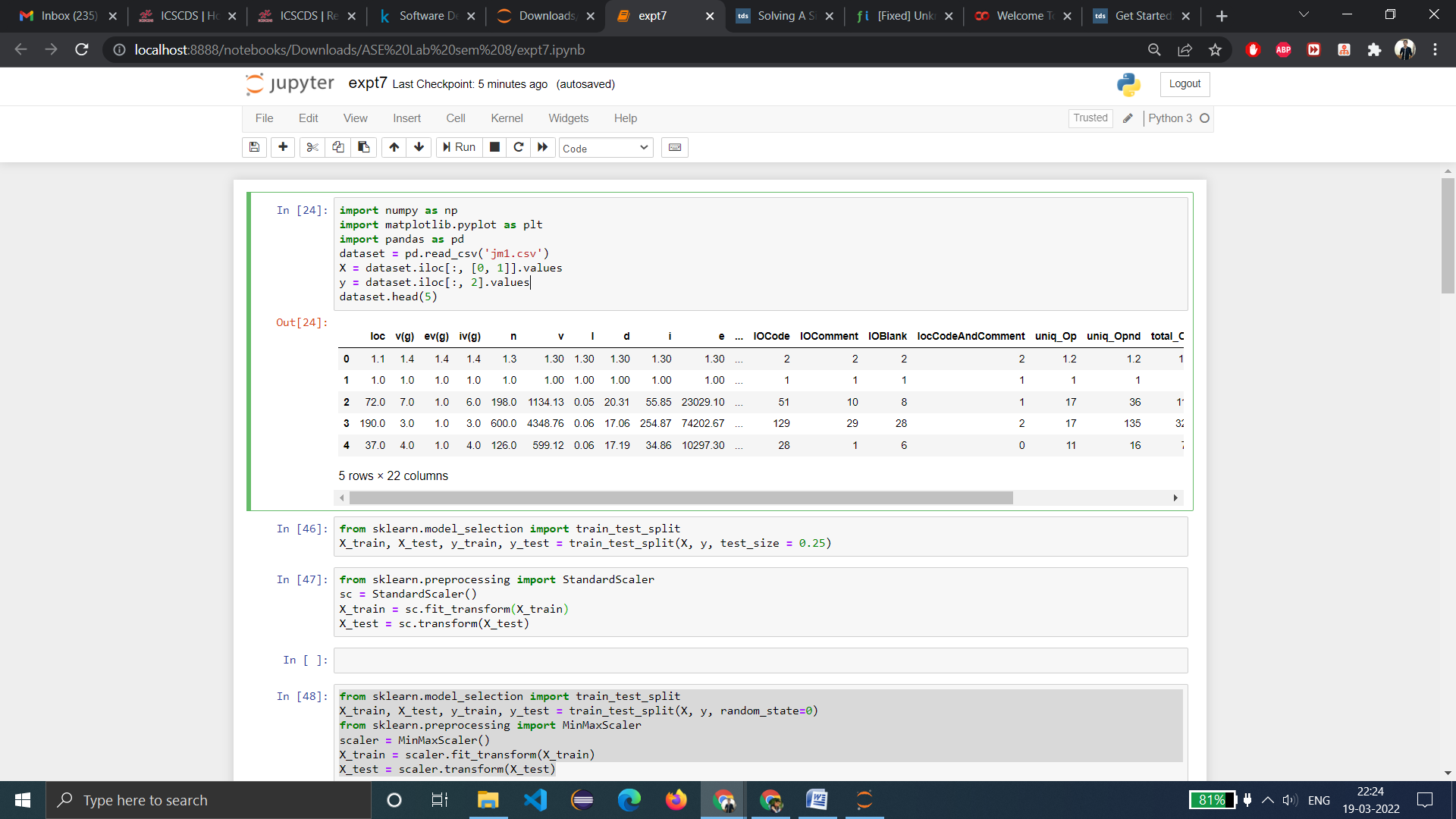
# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)



# Create Training and Test Sets and Apply Scaling

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=0)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# #Applying SVM Classifier on the Training Set

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score

from sklearn import svm

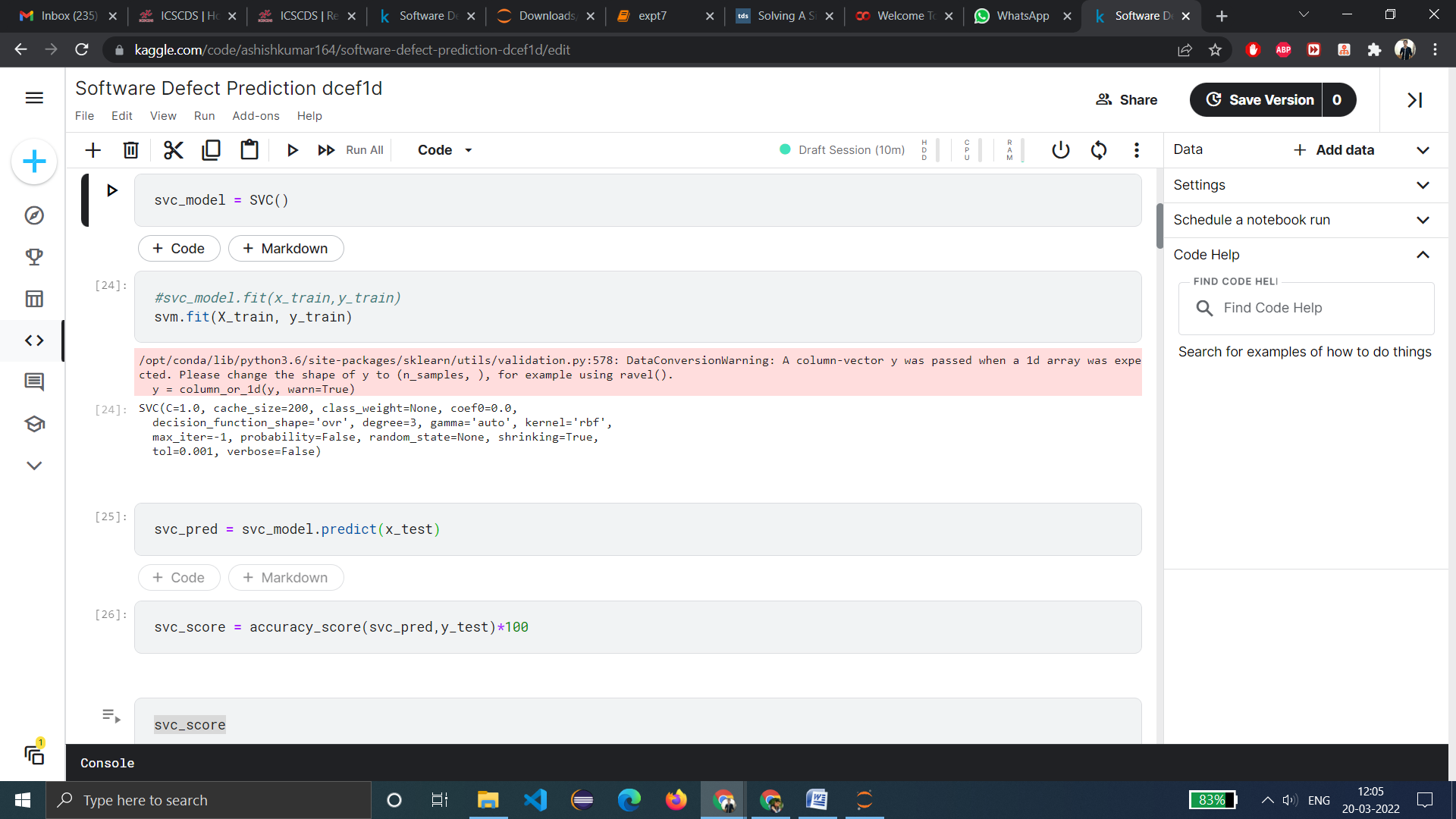
from sklearn.preprocessing import StandardScaler

from sklearn.svm import LinearSVC, SVC

svm= SVC()

svm.fit(X\_train, y\_train)

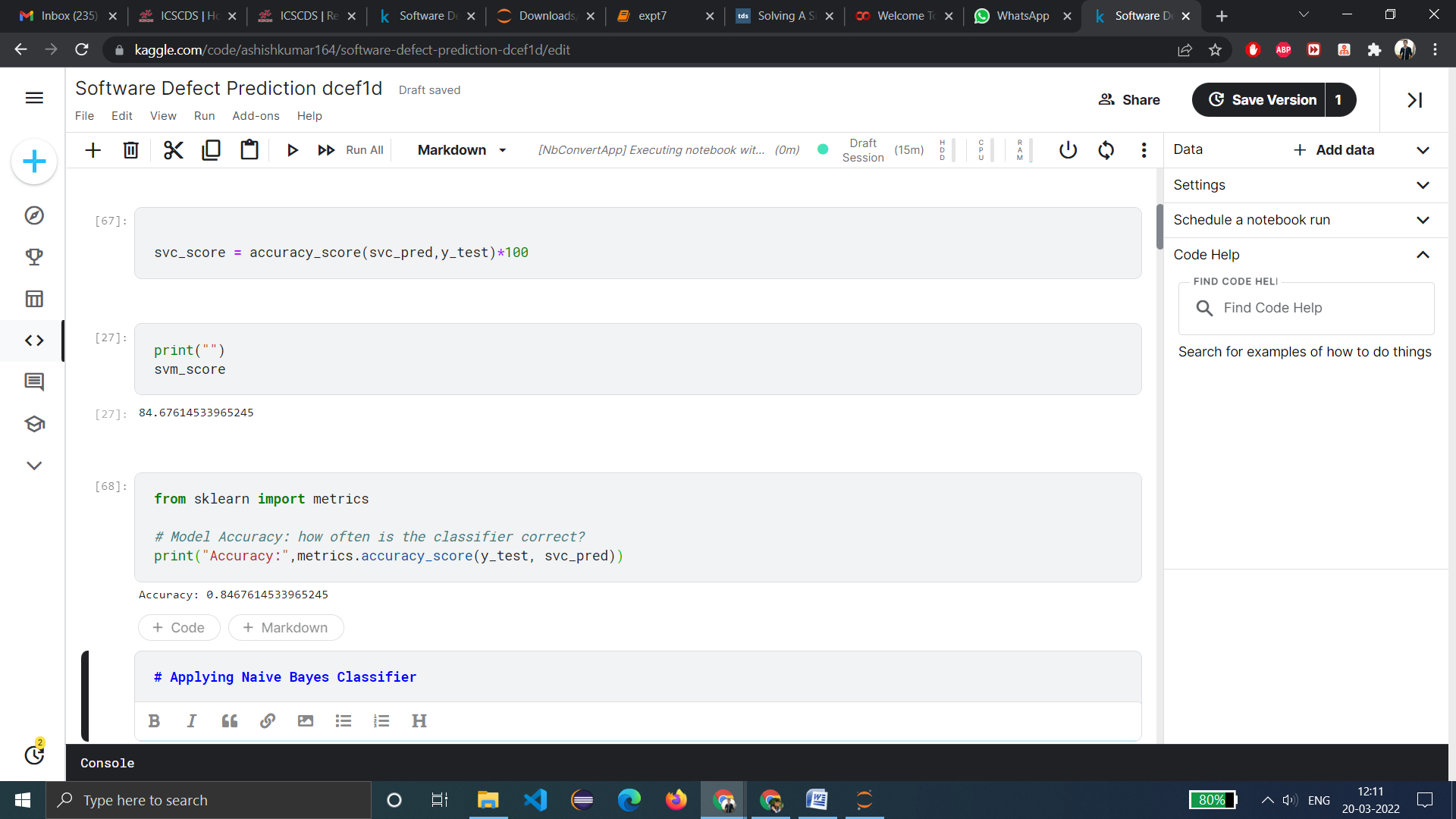
svm\_pred = svm.predict(X\_test)



# Model Accuracy: how often is the classifier correct?

from sklearn import metrics

print("Accuracy:",metrics.accuracy\_score(y\_test, svm\_pred))



#visualize support-vectors of svm classifier

from sklearn.preprocessing import LabelEncoder

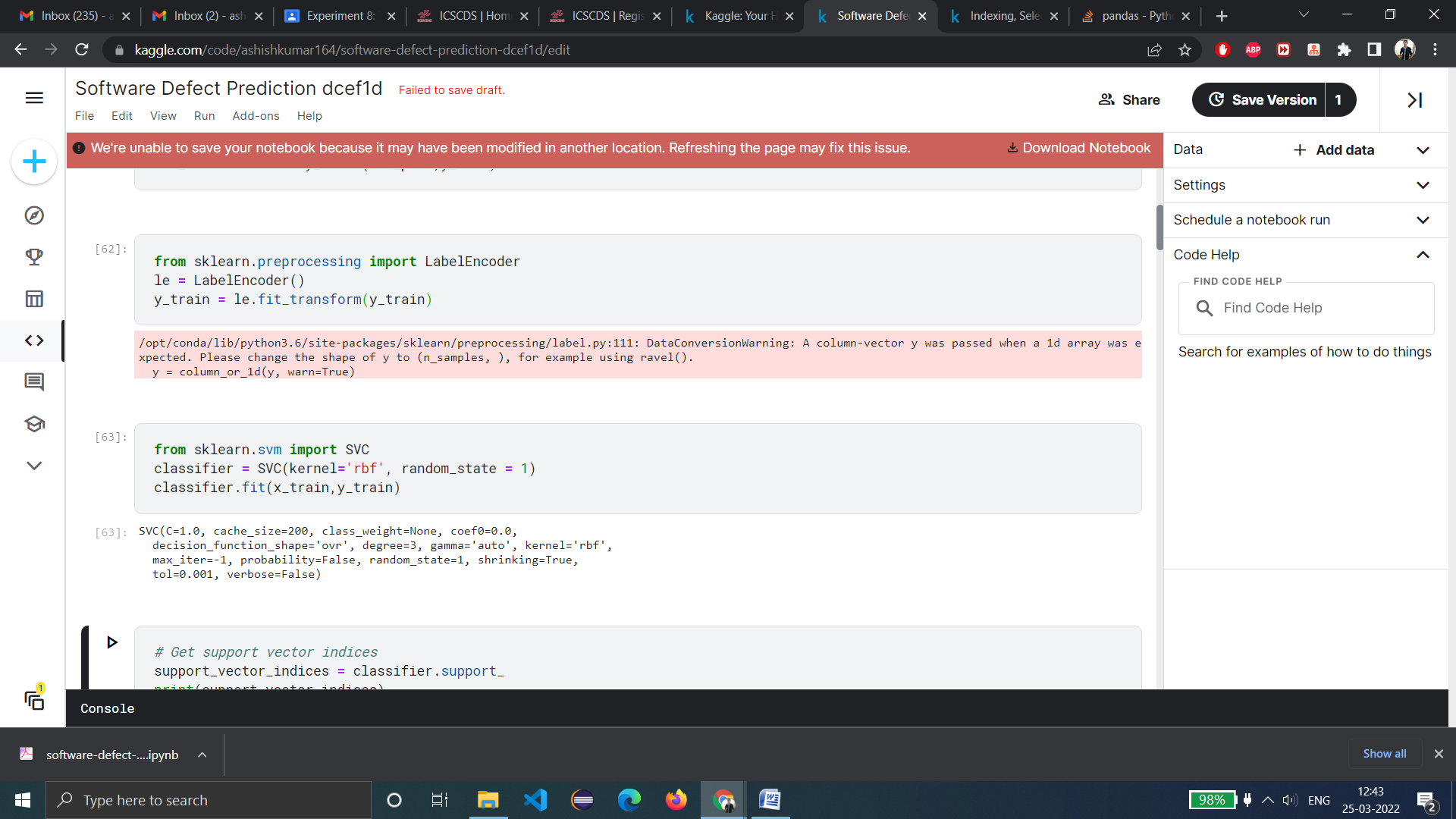
le = LabelEncoder()

y\_train = le.fit\_transform(y\_train)

from sklearn.svm import SVC

classifier = SVC(kernel='rbf', random\_state = 1)

classifier.fit(x\_train,y\_train)

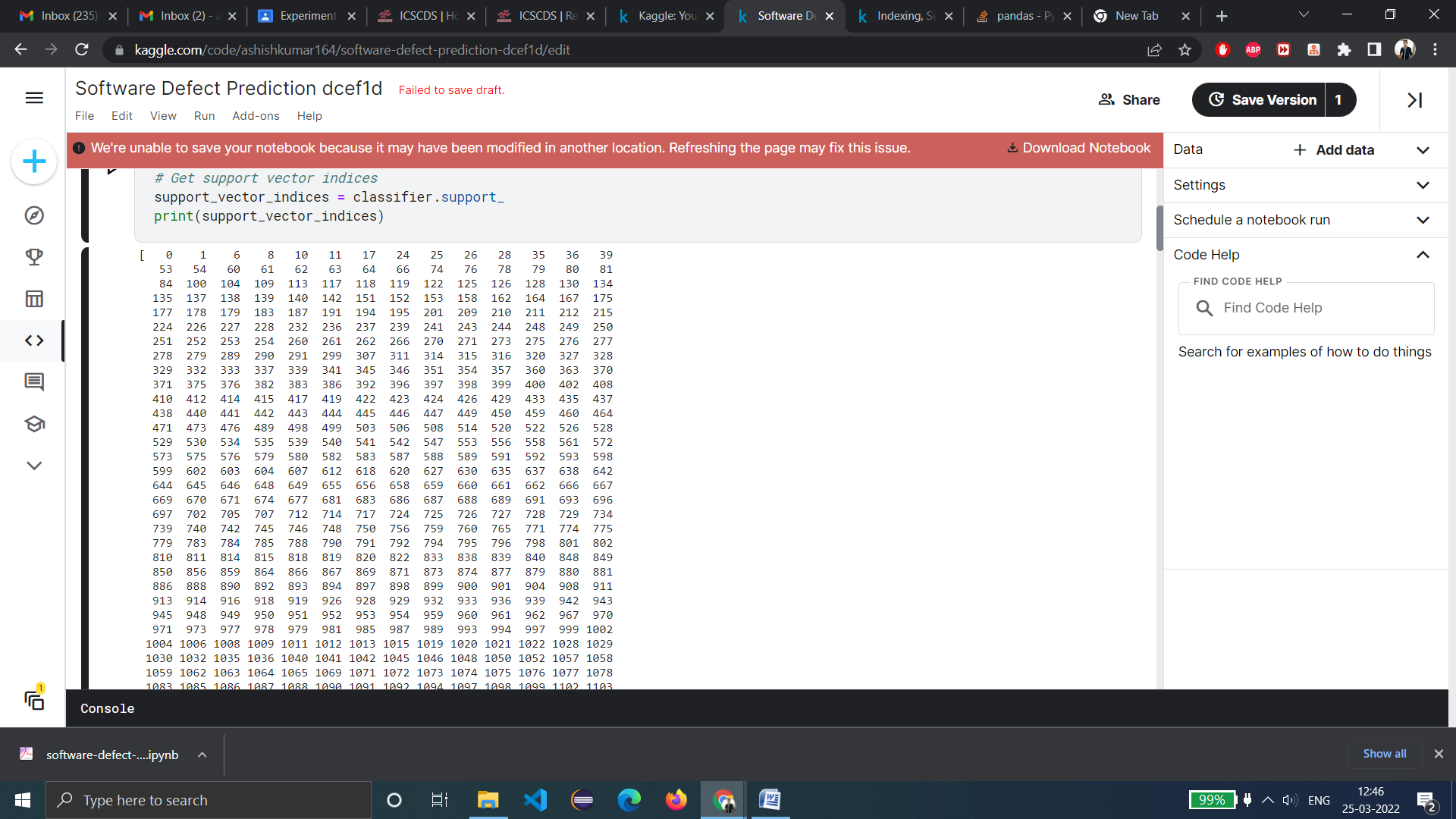


#Visualizing SVM's support vectors

# Get support vector indices

support\_vector\_indices = classifier.support\_

print(support\_vector\_indices)



# Get support vectors themselves

support\_vectors = classifier.support\_vectors\_

# Visualize support vectors

plt.scatter(x\_train[:,0].values, x\_train[:,1].values)

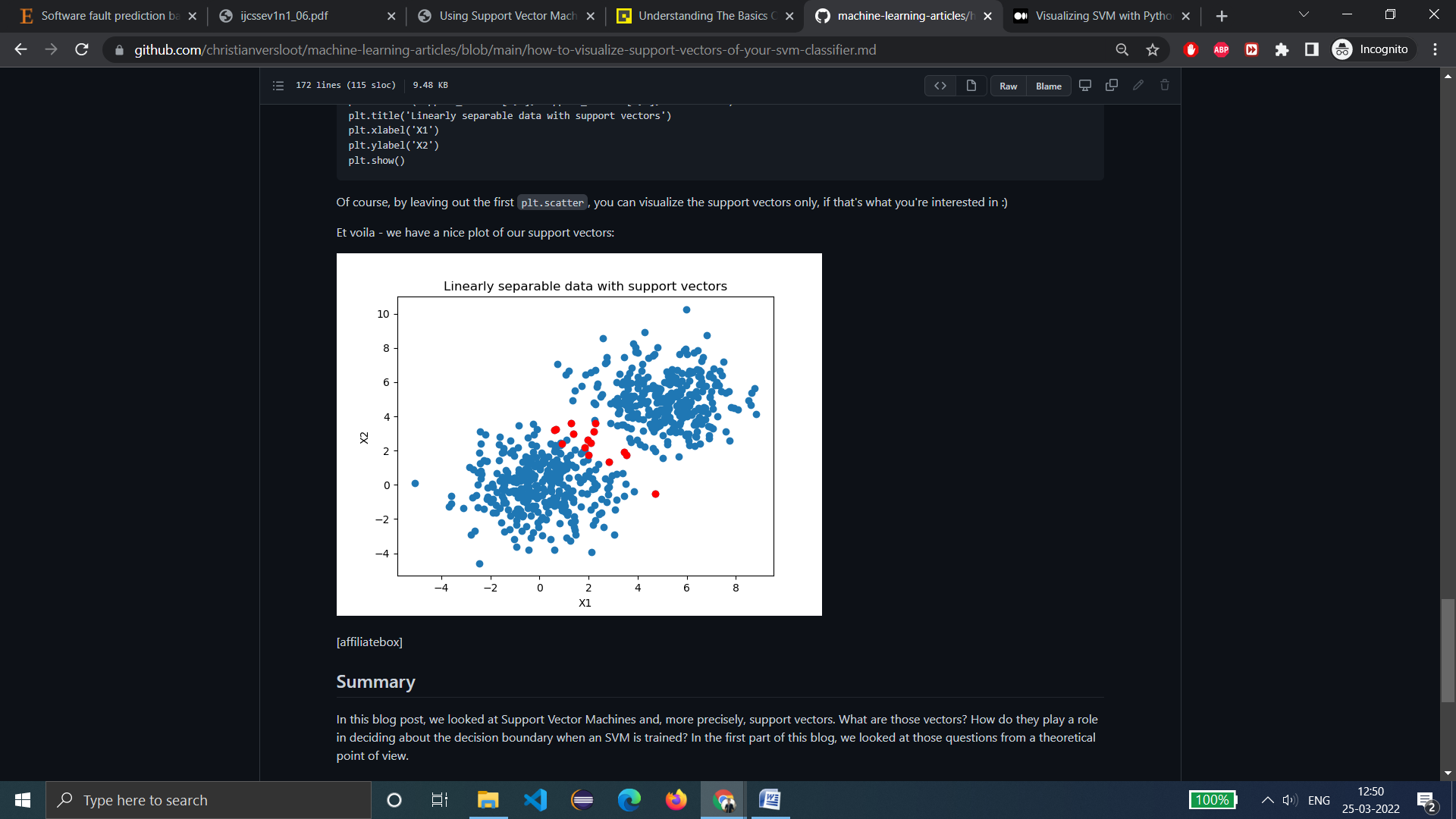
plt.scatter(support\_vectors[:,0].values, support\_vectors[:,1].values, color='red')

plt.title('Linearly separable data with support vectors')

plt.xlabel('X1')

plt.ylabel('X2')

plt.show()



**Result:**- The accuracy of the SVM Classifier comes out to be 84.67%.

**Learning from experiment:**- We have successfully been able to train and test fault prediction model using SVM.

**EXPERIMENT 9**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To train and test fault prediction model using Decision tree.

**Introduction:-** Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

## Pros

* Decision trees are easy to interpret and visualize.
* It can easily capture Non-linear patterns.
* It requires fewer data preprocessing from the user, for example, there is no need to normalize columns.
* It can be used for feature engineering such as predicting missing values, suitable for variable selection.
* The decision tree has no assumptions about distribution because of the non-parametric nature of the algorithm.

## Cons

* Sensitive to noisy data. It can overfit noisy data.
* The small variation (or variance) in data can result in the different decision tree. This can be reduced by bagging and boosting algorithms.
* Decision trees are biased with imbalance dataset, so it is recommended that balance out the dataset before creating the decision tree.

**Code & Output:-**

Link to dataset: <http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff>

This is a PROMISE data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering.

# Importing the dataset

import numpy as np

import matplotlib.pyplot as plt

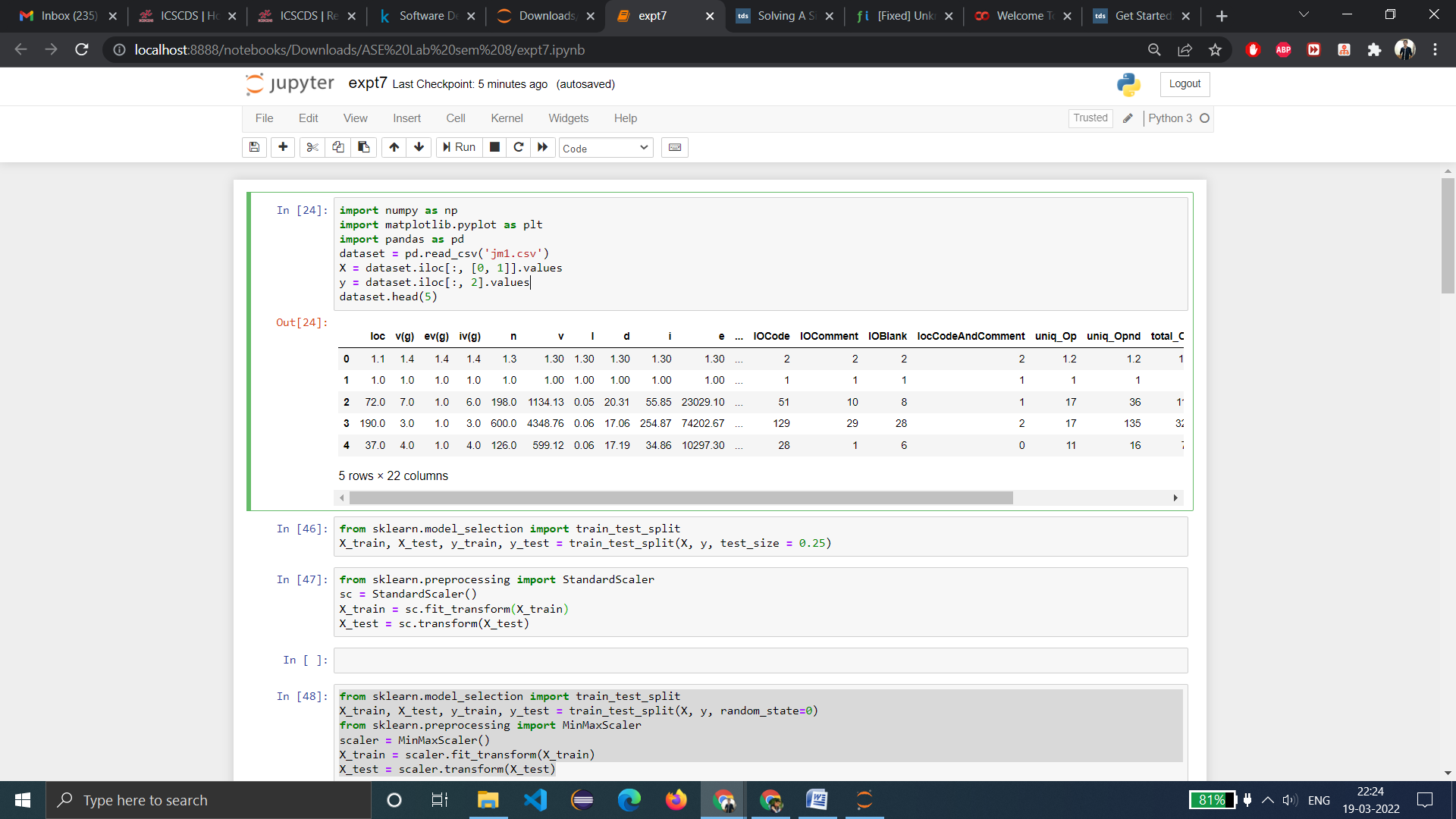
import pandas as pd

dataset = pd.read\_csv('jm1.csv')

X = dataset.iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

dataset.head(5)



# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

#Applying Decision tree Classifier on the Training Set

# Create Decision Tree classifer object

from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer

clf = clf.fit(x\_train,y\_train)

#Predict the response for test dataset

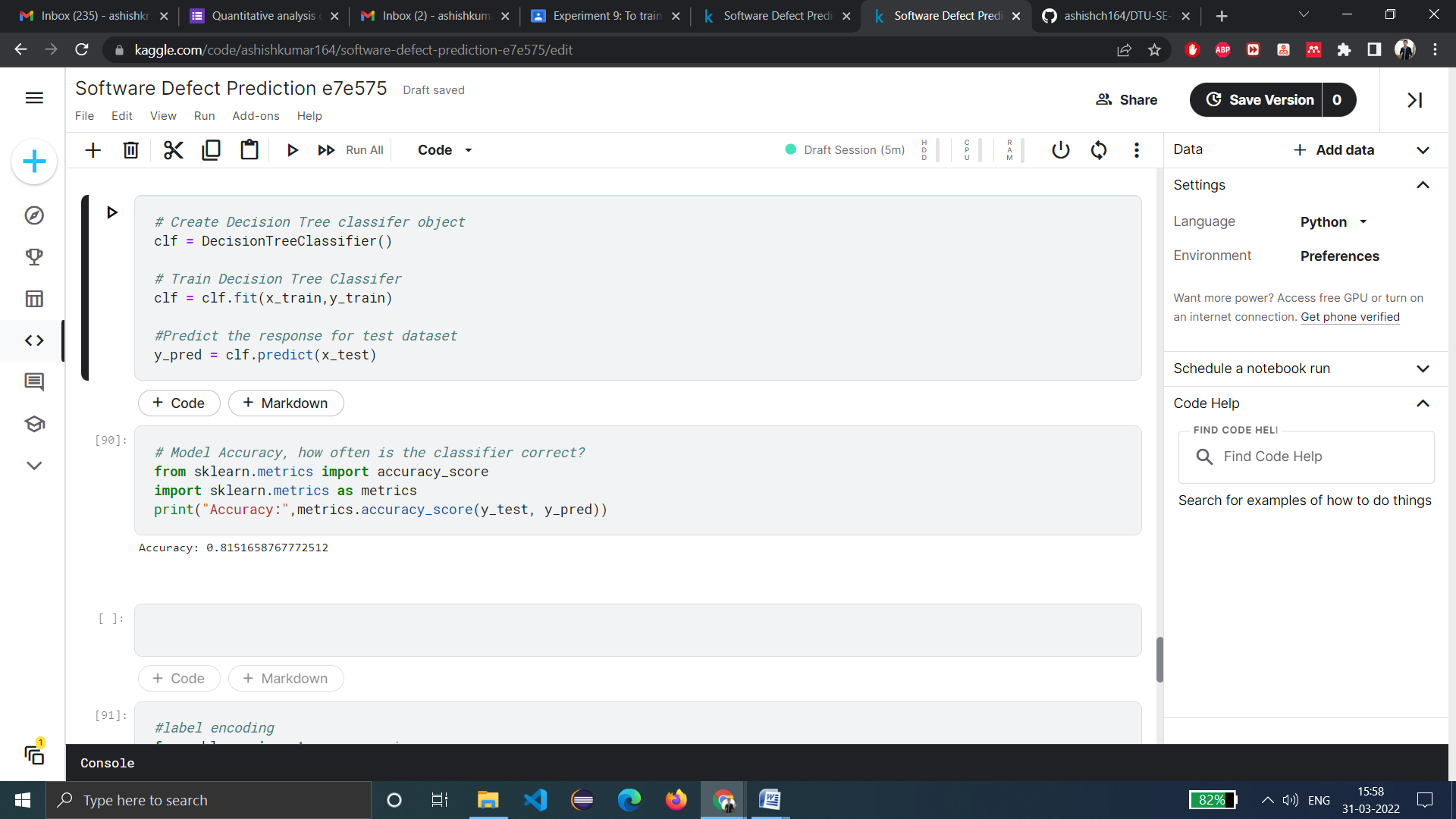
y\_pred = clf.predict(x\_test)

# Model Accuracy, how often is the classifier correct?

from sklearn.metrics import accuracy\_score

import sklearn.metrics as metrics

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))



**Result:**- The accuracy of the Decision tree classifier comes out to be 81.51%.

**Learning from experiment:**- We have successfully been able to train and test fault prediction model using Decision tree and learnt its advantages as well as its disadvantages.

**EXPERIMENT 10**

**-** ASHISH KUMAR  
 - 2K18/SE/041

# Aim:- To train and test fault prediction model using Random Forest.

**Introduction:-** Technically it is an ensemble method (based on the divide-and-conquer approach) of decision trees generated on a randomly split dataset. This collection of decision tree classifiers is also known as the forest. The individual decision trees are generated using an attribute selection indicator. Each tree depends on an independent random sample. It is simpler and more powerful compared to the other non-linear classification algorithms.

## Advantages:

* Random forest is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
* It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
* The algorithm can be used in both classification and regression problems.
* Random forest can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
* You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

## Disadvantages:

* Random forest is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
* The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.

**Code & Output:-**

Link to dataset: <http://promise.site.uottawa.ca/SERepository/datasets/cm1.arff>

This is a PROMISE data set made publicly available in order to encourage repeatable, verifiable, refutable, and/or improvable predictive models of software engineering.

# Importing the dataset

import numpy as np

import matplotlib.pyplot as plt

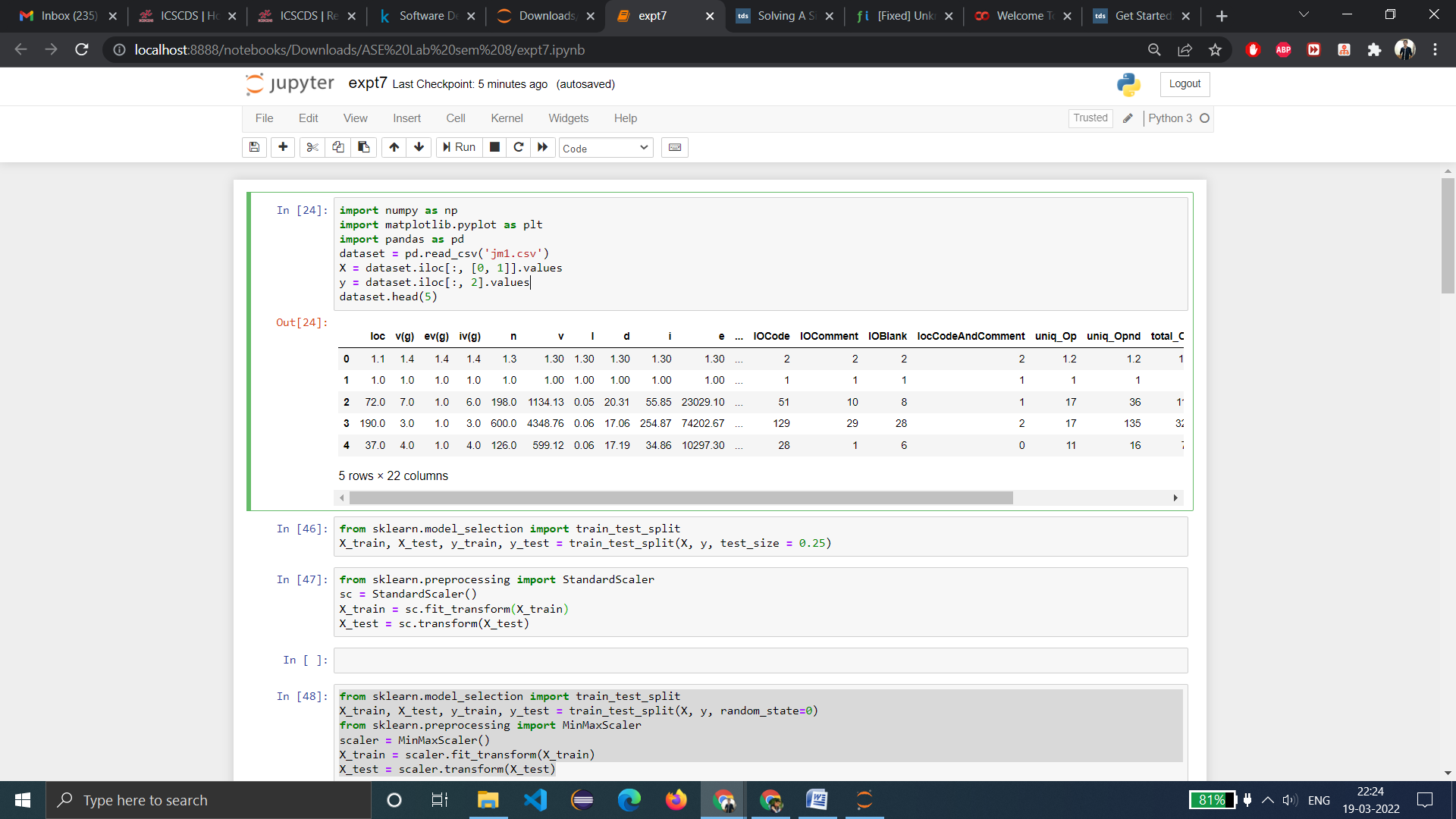
import pandas as pd

dataset = pd.read\_csv('jm1.csv')

X = dataset.iloc[:, [0, 1]].values

y = dataset.iloc[:, 2].values

dataset.head(5)



# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.3,random\_state=1)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

x\_train = sc.fit\_transform(x\_train)

x\_test = sc.transform(x\_test)

# #Applying Random Forest Classifier on the Training Set

from sklearn.ensemble import RandomForestClassifier

# creating a RF classifier

clf = RandomForestClassifier(n\_estimators = 100)

# Training the model on the training dataset

clf.fit(x\_train, y\_train)

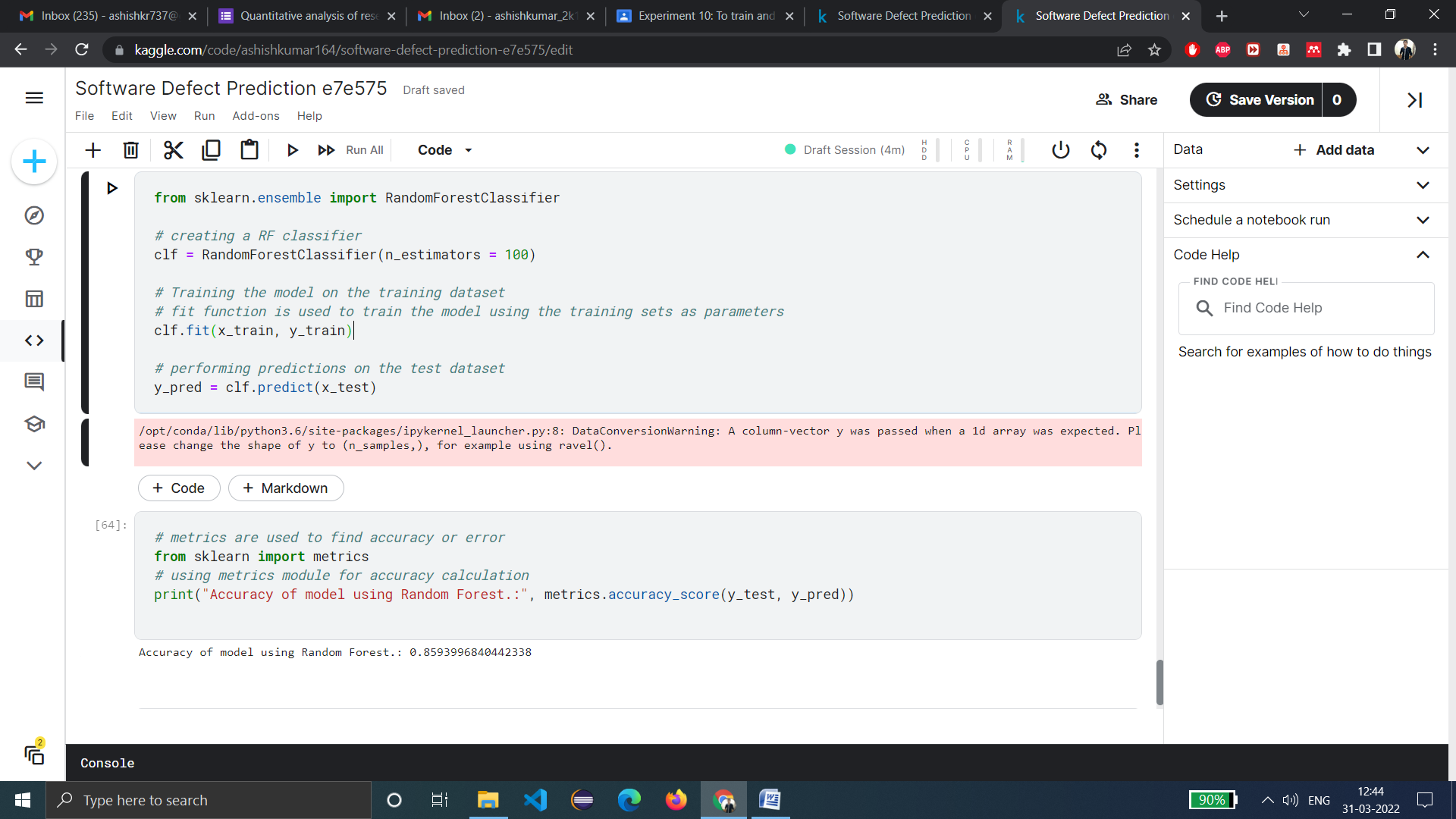
y\_pred = clf.predict(x\_test)

# Model Accuracy: how often is the classifier correct?

from sklearn import metrics

# using metrics module for accuracy calculation

print("Accuracy of model using Random Forest:", metrics.accuracy\_score(y\_test, y\_pred))



**Result:**- The accuracy of the Random Forest classifier comes out to be 85.93%.

**Learning from experiment:**- We have successfully been able to train and test fault prediction model using Random Forest and learnt its advantages as well as its disadvantages.