# Empirical Software Engineering (SE-404)

## LAB A1-G2

## Laboratory Manual

## Department of Software Engineering

## DELHI TECHNOLOGICAL UNIVERSITY(DTU)

Shahbad Daulatpur, Bawana Road, Delhi-110042

| **Submitted to: -** | **Submitted by:-** |
|---|---|
| Ms. Priya Singh | Name: ASHISH KUMAR |
| | Roll number: 2K18/SE/041 |

# INDEX

| S.No. | EXPERIMENT | DATE | REMARKS |
|---|---|---|---|
| **10.** | Perform a comparison of the following data analysis tools. WEKA, KEEL, SPSS, MATLAB, R. | 04-01-2022 | |
| **1.** | Consider any empirical study of your choice (Experiments, Survey Research, Systematic Review, Postmortem analysis and case study). Identify the following components for an empirical study:<br>a. Identify parametric and nonparametric tests<br>b. Identify Independent, dependent and confounding variables<br>c. Is it Within-company and cross-company analysis?<br>d. What type of dataset is used? Proprietary and open-source software | 18-01-2022 | |
| **2.** | Defect detection activities like reviews and testing help in identifying the defects in the artifacts (deliverables). These defects must be classified into various buckets before carrying out the root cause analysis. Following are some of the defect categories: Logical, User interface, Maintainability, and Standards. In the context of the above defect categories, classify the following statements under the defect categories. | 25-01-2022 | |
| **3.** | Consider any prediction model of your choice.<br>a. Analyze the dataset that is given as a input to the prediction model<br>b. Find out the quartiles for the used dataset<br>c. Analyze the performance of a model using various performance metrics. | 25-01-2022 | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Empirical Software Engineering LAB – A1 G2
# EXPERIMENT 2

- ASHISH KUMAR
- 2K18/SE/041

**Experiment Objective:-** Defect detection activities like reviews and testing help in identifying the defects in the artifacts (deliverables). These defects must be classified into various buckets before carrying out the root cause analysis. Following are some of the defect categories: Logical, User interface, Maintainability, and Standards. In the context of the above defect categories, classify the following statements under the defect categories.

## Introduction:-

**Root cause analysis (RCA)** is the process of discovering the root causes of problems in order to identify appropriate solutions. **The first goal** of root cause analysis is to discover the root cause of a problem or event. **The second goal** is to fully understand how to fix, compensate, or learn from any underlying issues within the root cause. **The third goal** is to apply what we learn from this analysis to systematically prevent future issues or to repeat successes.

## Result:-

**a. Divide by Zero Error is not guarded**
**Logical Defect:**
Logical defects are mistakes done regarding the implementation of the code. They are related to the core of the software and happen when the programmer does not take care of the corner cases or doesn't understand the problem clearly or thinks in a wrong way. Not handling corner cases can lead to low-quality software causing crashes and other kinds of defects.
Dividing a number by Zero is a mathematical error (not defined) and we can use exception handling to gracefully overcome such operations.

**b. Usage of 3.14 in the statement Circle_Area = 3.14 * Radius * Radius;**
**Logical Defect:**
Using 3.14 can lead to loss of precision for some applications. When the programmer doesn't understand the problem clearly or thinks in a wrong way then such types of defects happen. Solution would be 3.14 can be declared as macro.

**c. 3500 lines of code in a single function**
**Maintainability:**
Managing large monolithic codebases is tough. And modifying such codebases is tougher. Decomposing a system into subsystems reduces the complexity developers have to deal with by simplifying the parts and increasing their coherence.

**d. A pointer is declared but not initialized. It is used in the program for storing a value.**
**Logical Defect and Standards:**
A pointer is a variable that holds the address of another variable. So solution would be to initialize the pointer properly and assign a value to it.

**e. A program designed to handle 1000 simultaneous users, crashed when 1001 the user logged in.**
**Logical Defect, User Interface:**
Solution would be to do proper load and stress testing before hosting to the internet for user usage.

**f. A "while" loop never exits**
**Logical Defect:**
Solution would be to write appropriate condition to end the loop otherwise the program would run infinite times.

**g. User interface displays "MALFUNCTION 54" when something goes wrong in the back-end**
**User Interface:**
Solution would be to do proper error handling and make an appropriate display message so that user would get to know what exact problem is. One of the main aspects of GUI testing is checking whether correct error messages are being displayed. UI Testing is done to uncover such User Interface Bugs.

**h. No documentation (comments) for the source code**
**Standards:**
Solution: It is considered to be a good coding standard to name the functions according to what they perform and to add comments explaining what function are actually doing and proper documentations should be done.

**i. Hungarian Notation not followed while coding, even though the coding guidelines mandate to use Hungarian Notation**
**Standards:**
Solution: It is always good to follow standards i.e. Hungarian Notation while coding. All coding guidelines should be followed for successful software development.

**j. Pressing of “Tab” key moves the cursor in different fields of a web form randomly.**
**User Interface:**
Solution would be to design forms properly and to perform form-based testing before hosting to internet.

**Learning from experiment:**- We have successfully learned about RCA and from which we learned different types of testing and found different defects in the different scenarios. We also discussed solutions to prevent these defects.

# Empirical Software Engineering LAB – A1 G2
# EXPERIMENT 3

- ASHISH KUMAR
- 2K18/SE/041

**Experiment Objective:-** Consider any prediction model of your choice.
a. Analyze the dataset that is given as a input to the prediction model
b. Find out the quartiles for the used dataset
c. Analyze the performance of a model using various performance metrics.

## Introduction:-

The predictive model used for this experiment is from a research paper "**Software Fault Proneness Prediction Using Support Vector Machines by Yogesh Singh, Arvinder Kaur, Ruchika Malhotra,** 2009 [Proceedings of the World Congress on Engineering 2009, London, U.K.].
Here is the link of the research paper:
http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=47D7383DC75311874ED04A78F4B955A1?doi=10.1.1.149.3711&rep=rep1&type=pdf

## Code and Result:-

a) Link of the dataset used for this experiment:
http://promise.site.uottawa.ca/SERepository/datasets/kc1-class-level-numericdefect.arff

This study makes use of the public domain data set KC1 from the NASA Metrics Data Program. The data in KC1 was collected from a storage management system for receiving/processing ground data, which was implemented in the C++ programming language. This system consists of 145 classes that comprise 2107 methods, with 40K lines of code. KC1 provides both class-level and method-level static metrics. At the class level, values of 10 metrics are computed including seven metrics given by Chidamber and Kemerer.

Feature Used as the Response Variable:
=======================================
NUMDEFECTS: The number of defects recorded for the class,

Features at Class Level Originally
===================================
PERCENT_PUB_DATA: The percentage of data that is public and protected data in a class. In general, lower values indicate greater encapsulation. It is measure of encapsulation.

ACCESS_TO_PUB_DATA: The amount of times that a class's public and protected data is accessed. In general, lower values indicate greater encapsulation.  It is a measure of encapsulation.

COUPLING_BETWEEN_OBJECTS: The number of distinct non-inheritance-related classes on which a class depends. If a class that is heavily dependent on many classes outside of its hierarchy is introduced into a library, all the classes upon which it depends need to be introduced as well. This may be acceptable, especially if the classes which it references are already part of a class library and are even more fundamental than the specified class.

DEPTH: The level for a class. For instance, if a parent has one child the  depth for the child is two. Depth indicates at what level a class is located within its class hierarchy. In general, inheritance increases when depth increases.

LACK_OF_COHESION_OF_METHODS: For each data field in a class, the percentage of the methods in the class using that data field; the percentages are averaged then subtracted from 100%. The LOCM metric indicates low or high percentage of cohesion. If the percentage is low, the class is cohesive. If it is high, it may indicate that the class could be split into separate classes that will individually have greater cohesion.

NUM_OF_CHILDREN: The number of classes derived from a specified class.

DEP_ON_CHILD: Whether a class is dependent on a descendant.

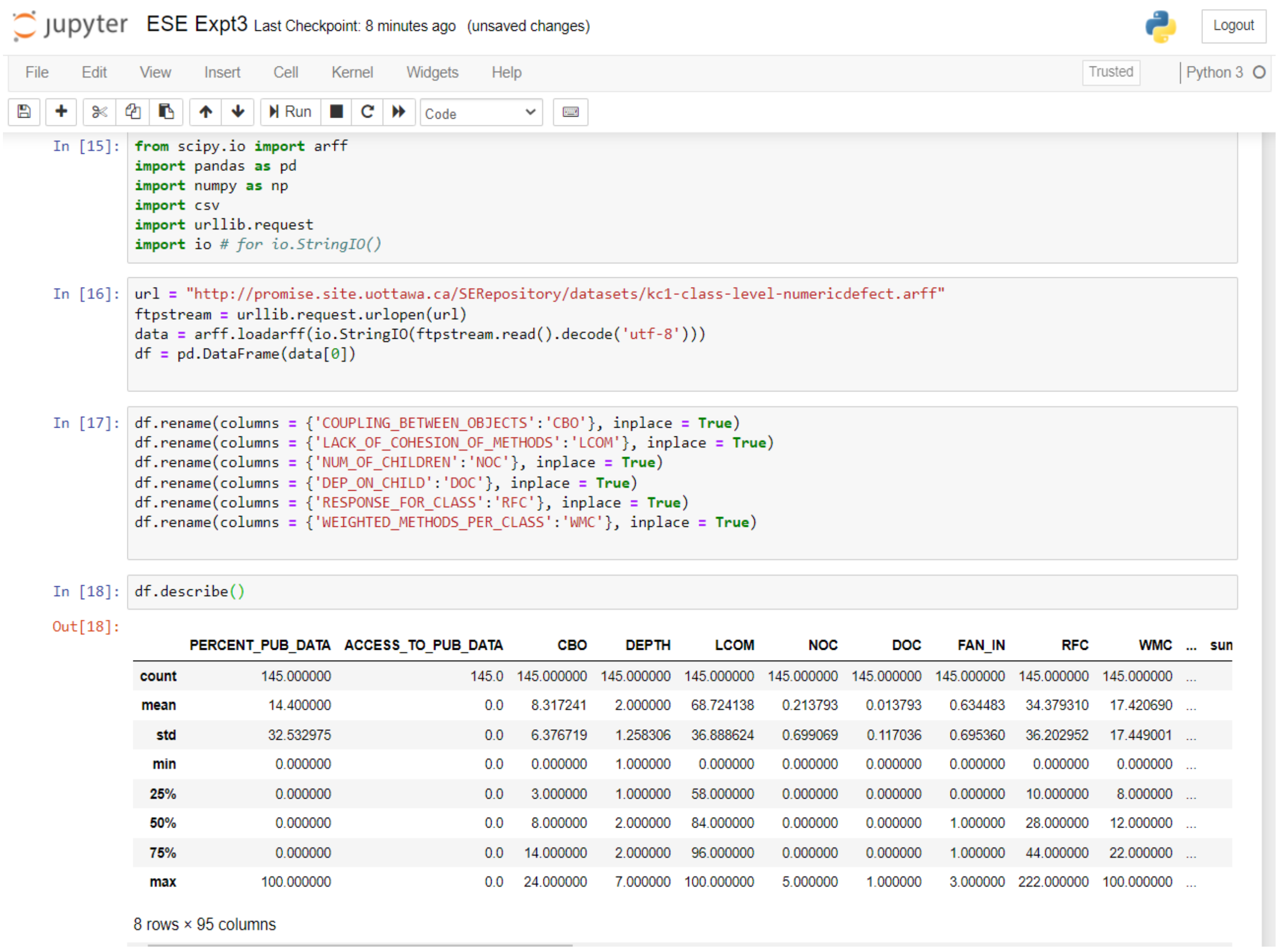
FAN_IN: This is a count of calls by higher modules.

RESPONSE_FOR_CLASS: A count of methods implemented within a class plus the number of methods accessible to an object class due to inheritance. In general, lower values indicate greater polymorphism.

WEIGHTED_METHODS_PER_CLASS: A count of methods implemented within a class (rather than all methods accessible within the class hierarchy). In general, lower values indicate greater polymorphism.

**Note:-** **I have used python language to analyze the dataset and used pandas, numpy libraries to extract and analyze the data.**

Below is the screenshot of the Statistical Summary of dataset after extracting it from arff file:



jupyter ESE Expt3 Last Checkpoint: 8 minutes ago (unsaved changes)

```
In [15]: from scipy.io import arff
         import pandas as pd
         import numpy as np
         import csv
         import urllib.request
         import io # for io.StringIO()
```

```
In [16]: url = "http://promise.site.uottawa.ca/SERepository/datasets/kc1-class-level-numericdefect.arff"
         ftpstream = urllib.request.urlopen(url)
         data = arff.loadarff(io.StringIO(ftpstream.read().decode('utf-8')))
         df = pd.DataFrame(data[0])
```

```
In [17]: df.rename(columns = {'COUPLING_BETWEEN_OBJECTS':'CBO'}, inplace = True)
         df.rename(columns = {'LACK_OF_COHESION_OF_METHODS':'LCOM'}, inplace = True)
         df.rename(columns = {'NUM_OF_CHILDREN':'NOC'}, inplace = True)
         df.rename(columns = {'DEP_ON_CHILD':'DOC'}, inplace = True)
         df.rename(columns = {'RESPONSE_FOR_CLASS':'RFC'}, inplace = True)
         df.rename(columns = {'WEIGHTED_METHODS_PER_CLASS':'WMC'}, inplace = True)
```

```
In [18]: df.describe()
```

Out[18]:

| | PERCENT_PUB_DATA | ACCESS_TO_PUB_DATA | CBO | DEPTH | LCOM | NOC | DOC | FAN_IN | RFC | WMC | ... | sun |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 145.000000 | 145.0 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | ... | |
| mean | 14.400000 | 0.0 | 8.317241 | 2.000000 | 68.724138 | 0.213793 | 0.013793 | 0.634483 | 34.379310 | 17.420690 | ... | |
| std | 32.532975 | 0.0 | 6.376719 | 1.258306 | 36.888624 | 0.699069 | 0.117036 | 0.695360 | 36.202952 | 17.449001 | ... | |
| min | 0.000000 | 0.0 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | ... | |
| 25% | 0.000000 | 0.0 | 3.000000 | 1.000000 | 58.000000 | 0.000000 | 0.000000 | 0.000000 | 10.000000 | 8.000000 | ... | |
| 50% | 0.000000 | 0.0 | 8.000000 | 2.000000 | 84.000000 | 0.000000 | 0.000000 | 1.000000 | 28.000000 | 12.000000 | ... | |
| 75% | 0.000000 | 0.0 | 14.000000 | 2.000000 | 96.000000 | 0.000000 | 0.000000 | 1.000000 | 44.000000 | 22.000000 | ... | |
| max | 100.000000 | 0.0 | 24.000000 | 7.000000 | 100.000000 | 5.000000 | 1.000000 | 3.000000 | 222.000000 | 100.000000 | ... | |

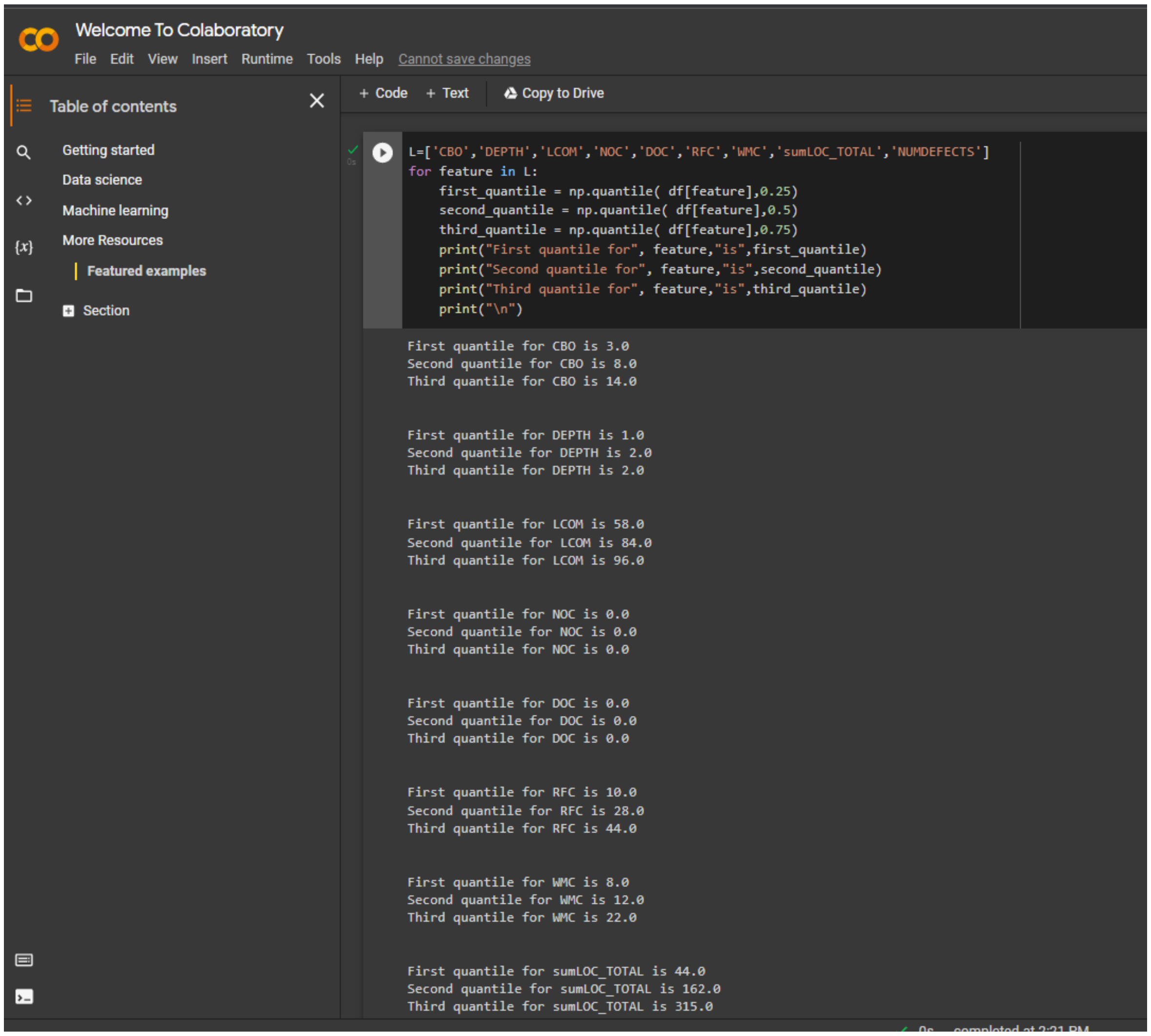
8 rows × 95 columns

b)
The quantile( ) function of NumPy is used to find quartiles of the dataset for each of the metrics or attributes used. Quantiles are the set of values/points that divides the dataset into groups of equal size.

**<u>Note:</u> I have used Google Colab for this because jupyter notebook is showing error for quantile function.**

Below is the code of it and screenshot of output:

```
L=['CBO','DEPTH','LCOM','NOC','DOC','RFC','WMC','sumLOC_TOTAL','NUMDEFECTS']
for feature in L:
    first_quantile = np.quantile( df[feature],0.25)
    second_quantile = np.quantile( df[feature],0.5)
    third_quantile = np.quantile( df[feature],0.75)
    print("First quantile for", feature,"is",first_quantile)
    print("Second quantile for", feature,"is",second_quantile)
    print("Third quantile for", feature,"is",third_quantile)
    print("\n")
```



```
L=['CBO','DEPTH','LCOM','NOC','DOC','RFC','WMC','sumLOC_TOTAL','NUMDEFECTS']
for feature in L:
    first_quantile = np.quantile( df[feature],0.25)
    second_quantile = np.quantile( df[feature],0.5)
    third_quantile = np.quantile( df[feature],0.75)
    print("First quantile for", feature,"is",first_quantile)
    print("Second quantile for", feature,"is",second_quantile)
    print("Third quantile for", feature,"is",third_quantile)
    print("\n")
```

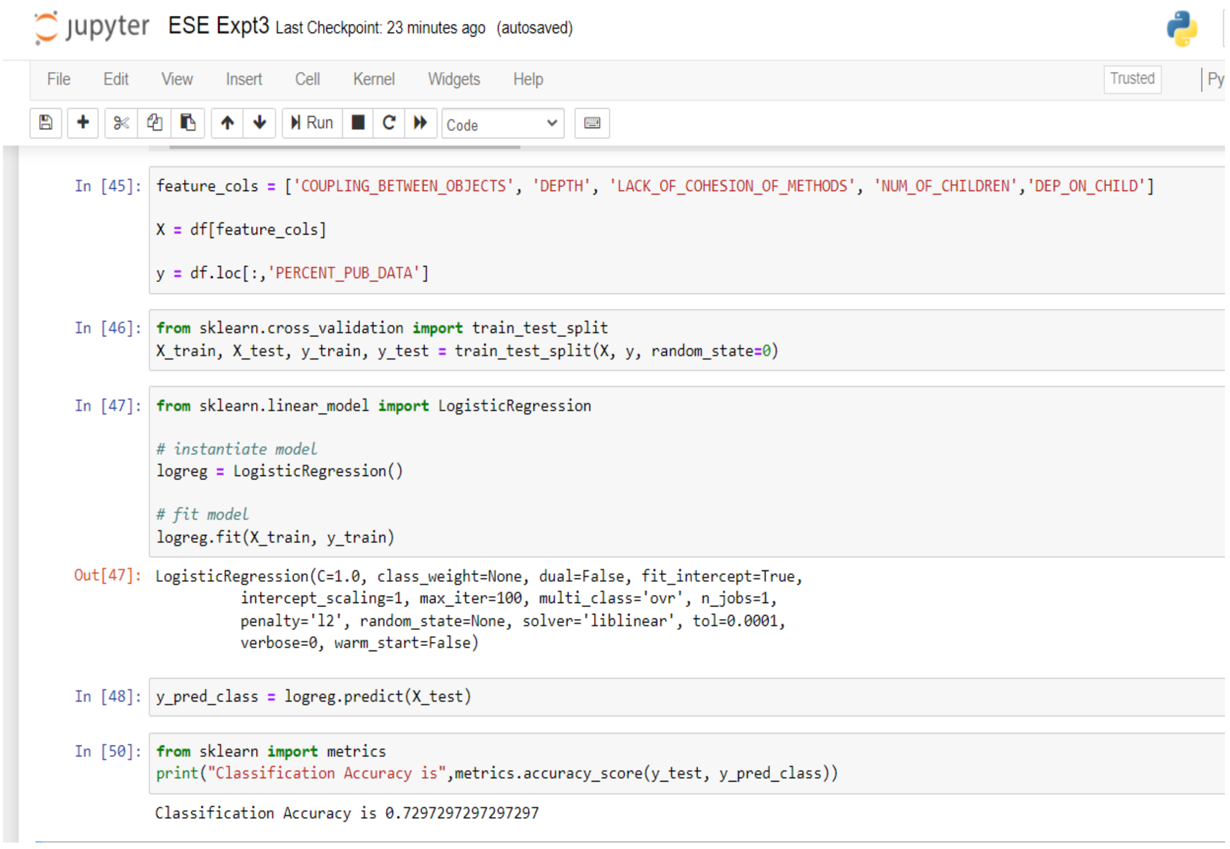
```
First quantile for CBO is 3.0
Second quantile for CBO is 8.0
Third quantile for CBO is 14.0


First quantile for DEPTH is 1.0
Second quantile for DEPTH is 2.0
Third quantile for DEPTH is 2.0


First quantile for LCOM is 58.0
Second quantile for LCOM is 84.0
Third quantile for LCOM is 96.0


First quantile for NOC is 0.0
Second quantile for NOC is 0.0
Third quantile for NOC is 0.0


First quantile for DOC is 0.0
Second quantile for DOC is 0.0
Third quantile for DOC is 0.0


First quantile for RFC is 10.0
Second quantile for RFC is 28.0
Third quantile for RFC is 44.0


First quantile for WMC is 8.0
Second quantile for WMC is 12.0
Third quantile for WMC is 22.0


First quantile for sumLOC_TOTAL is 44.0
Second quantile for sumLOC_TOTAL is 162.0
Third quantile for sumLOC_TOTAL is 315.0
```

c) The Model evaluation metrics used are:

**Classification Accuracy:** Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification problems.

Below is the code of it and output:

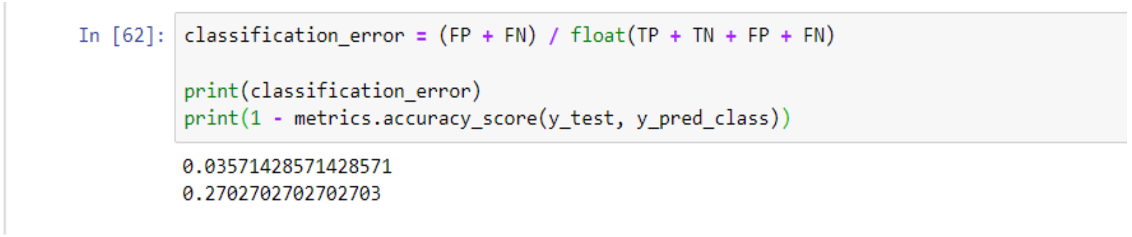
**Note:** Classification Accuracy comes out to be 0.7297297297297297 i.e. 72%.


jupyter ESE Expt3 Last Checkpoint: 23 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Run
Code

```
In [45]: feature_cols = ['COUPLING_BETWEEN_OBJECTS', 'DEPTH', 'LACK_OF_COHESION_OF_METHODS', 'NUM_OF_CHILDREN','DEP_ON_CHILD']

         X = df[feature_cols]

         y = df.loc[:,'PERCENT_PUB_DATA']

In [46]: from sklearn.cross_validation import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

In [47]: from sklearn.linear_model import LogisticRegression

         # instantiate model
         logreg = LogisticRegression()

         # fit model
         logreg.fit(X_train, y_train)

Out[47]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)

In [48]: y_pred_class = logreg.predict(X_test)

In [50]: from sklearn import metrics
         print("Classification Accuracy is",metrics.accuracy_score(y_test, y_pred_class))

         Classification Accuracy is 0.7297297297297297
```

**Classification Error:** Overall, how often is the classifier incorrect?

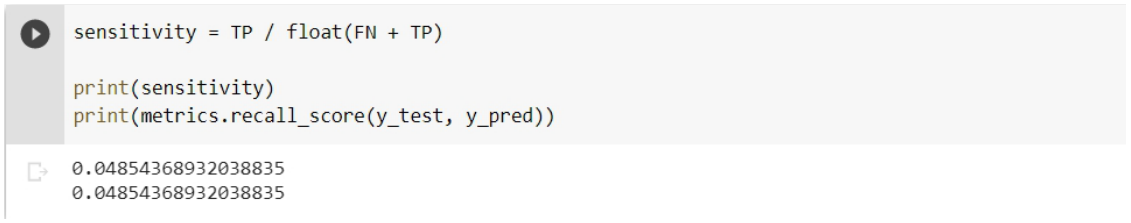
- Also known as "Misclassification Rate"



```
In [62]: classification_error = (FP + FN) / float(TP + TN + FP + FN)

         print(classification_error)
         print(1 - metrics.accuracy_score(y_test, y_pred_class))

         0.03571428571428571
         0.2702702702702703
```

**Sensitivity:** When the actual value is positive, how often is the prediction correct?

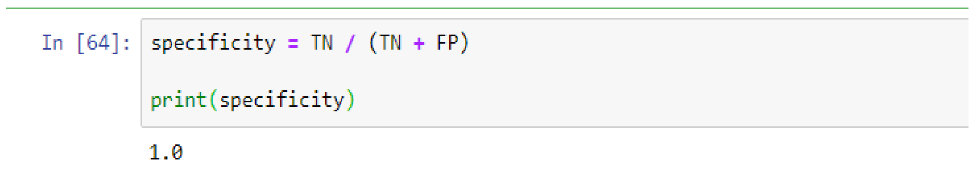
- How "sensitive" is the classifier to detecting positive instances?



```
sensitivity = TP / float(FN + TP)

print(sensitivity)
print(metrics.recall_score(y_test, y_pred))

0.04854368932038835
0.04854368932038835
```

**Specificity:** When the actual value is negative, how often is the prediction correct?

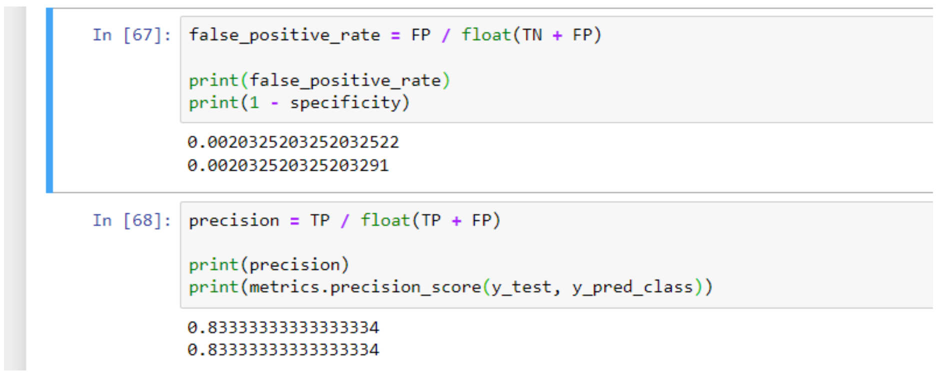
- Something we want to maximize
- How "specific" (or "selective") is the classifier in predicting positive instances?



```
In [64]: specificity = TN / (TN + FP)

         print(specificity)

1.0
```

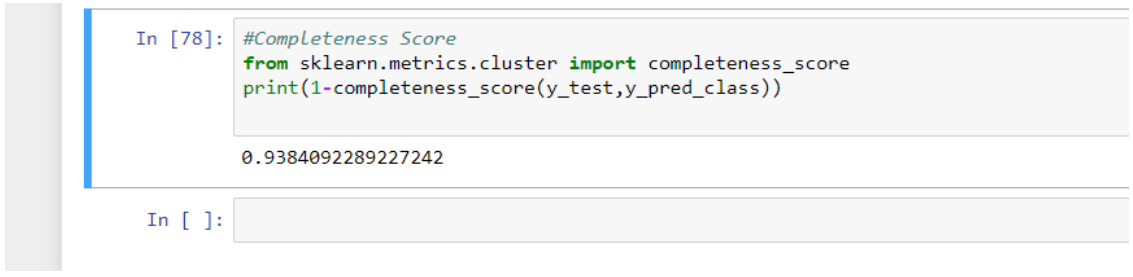
**False Positive Rate**: When the actual value is negative, how often is the prediction incorrect?

**Precision:** When a positive value is predicted, how often is the prediction correct?



```
In [67]: false_positive_rate = FP / float(TN + FP)

         print(false_positive_rate)
         print(1 - specificity)

0.0020325203252032522
0.002032520325203291

In [68]: precision = TP / float(TP + FP)

         print(precision)
         print(metrics.precision_score(y_test, y_pred_class))

0.8333333333333334
0.8333333333333334
```

**Completeness Score:** It is the Completeness metric of a cluster labeling given a ground truth. A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster.



```
In [78]: #Completeness Score
         from sklearn.metrics.cluster import completeness_score
         print(1-completeness_score(y_test,y_pred_class))

0.9384092289227242

In [ ]:
```

**Note:** I have used this site as a source to evaluate c point: https://www.ritchieng.com/machine-learning-evaluate-classification-model/

**Learning from experiment:**- Using the Support Vector Machine Predictive Model for Software Fault Proneness Prediction using OO metrics of KC1 NASA dataset, we analyzed the dataset, found out quartiles and analyzed the performance of the model using various performance metrics using Python language.