Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Software Defect Prediction Tool

PROJECT REPORT

Submitted for the Course
CSE3001-Software Engineering

by

SYED AYAZ IMAM (18BCE0660)

AISHIKA SAHA (18BCE2168)

Slot: D2
Name of faculty: Dr. SWATHI J.N

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

ACKNOWLEDGEMENTS

I take immense pleasure in thanking Dr. G. Viswanathan, my beloved Chancellor, VIT University and respected Dean, Dr. R. Saravanan, for having permitted me to carry out the project. I express gratitude to my guide, Dr. Swathi J.N, for guidance and suggestions that helped me to complete the project on time. Words are inadequate to express my gratitude to the faculty and staff members who encouraged and supported me during the project.

[illegible] [illegible]

Signature of Student 1 Signature of Student 2

Executive Summary

Software Defect Prediction [SDP] plays an important role in the active research areas of software engineering. A software defect is an error, bug, flaw, fault, malfunction or mistake in software that causes it to create a wrong or unexpected outcome. The major risk factors related with a software defect which is not detected during the early phase of software development are time, quality, cost, effort and wastage of resources. Defects may occur in any phase of software development. Booming software companies focus concentration on software quality, particularly during the early phase of the software development .Thus the key objective of any organization is to determine and correct the defects in an early phase of Software Development Life Cycle [SDLC]. To improve the quality of software, datamining techniques have been applied to build predictions regarding the failure of software components by exploiting past data of software components and their defects.

Keywords: Software Defect Prediction, Machine Learning, SDLC, Waterfall Model, Gantt Chart, Work Breakdown Structure, Activity Network, Timeline Chart.

Table of Contents

1. INTRODUCTION

1.1. OBJECTIVE

If a developer or a tester can predict the software defects properly, it then reduces the cost, time and effort. In this thesis, we put forward a comparative analysis of software defect prediction based on classification rule mining. We propose an innovative scheme for this process and we choose different classification algorithms that helped in showing the comparison of predictions in software defects analysis. This evaluation analyses the prediction performance of competing learning schemes with the given historical data set (NASA MDP Data Set).

1.2. MOTIVATION

Our motivation to follow such defect prediction techniques is to provide and fill in as early quality standard and indicator of the product entering framework testing and to help the testing teams to oversee and control test execution exercises. Matrices gathered from earlier stages to framework testing are recognized, dissected and analysed to ascertain potential predictors for designing the model. As a stand-alone testing team, it is essential to design and deal with the test execution exercises all together to fulfil the tight time constraint for discharging the product to end-clients. In order to deliver the best to the client there needs to be fast defect detection methodologies to provide them with an error free software system. A wide scope of prediction models has been proposed. A wide range of prediction models have been proposed. Furthermore, data from defect identification and the testing procedure can be utilized to anticipate absconds. As of late extensive complex multivariate measurable models have been created trying to locate a solitary multifaceted nature metric that will represent defects. This project shall help us gain a better understandability of such models.

1.3. BACKGROUND STUDY

There has been a huge spurt in the demand for software quality, especially over the last one or two decades. As a consequence, the issues related to testing are becoming increasingly critical. The ability to measure software defects can be extremely crucial for minimizing the cost and improving the overall effectiveness of the testing process. A majority of faults in a software system are found in a few of its own components. Although there are several definitions for software quality, it is totally agreed that a project with numerous defects lacks quality in its software. Knowing the causes of possible defects as well as identifying general software process areas that may need attention starting from the initialization of a project could save money, time and effort. The possibility of early estimation of the probable faultiness of software could help on planning, controlling and executing software development activities. A low-cost method for defect analysis is learning from past mistakes to prevent future ones. Today, several data sets exist, which could be mined to gain useful knowledge with regard to the defects that transpired during the rapid growth in software development. The defects in software are due to various reasons. In the process of software development, testing of software is the main phase which reduces the defects in the software. If a developer or a tester can predict the software defects properly, it then reduces the cost, time and effort. In this thesis, we put forward a comparative analysis of software defect prediction based on classification rule mining. We propose an innovative scheme for this process and we choose different classification algorithms that helped in showing the comparison of predictions in software defects analysis. This evaluation analyses the prediction performance of competing learning schemes with the given historical data set (NASA MDP Data Set). The result of this scheme evaluation shows that we have to choose different classifier rules for different data sets. Using this knowledge one should ideally be able to:

- Identify potential fault-prone software;
- Estimate the distinct number of faults; and
- Discover the possible causes of faults.

2. PROJECT DESCIPTION AND GOALS

The goal of this research is to help developers identify defects based on the existing software metrics using data mining techniques and thereby improve software quality, which ultimately leads to reducing the cost of software development both in the developing and maintenance phases. The focus is identifying defective modules so that the scope of software that needs to be examined for defects could be prioritized. This allows the developer to run test cases in the predicted modules. The proposed methodology helps in identifying modules that require immediate attention and the reliability of the software is vastly improved as the high priority defects are handled first.

3. TECHNICAL SPECIFICATION

>> **Python:** Python offers a concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems. Developers get to put all their effort into solving an ML problem instead of focusing on the technical nuances of the language.

>> **Pandas:** Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Pandas is Python Data Analysis Library, pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools.

>> **Google Colab**: Colab is a free Jupyter Notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that we create can be simultaneously edited by our team members - just the we edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in our notebook.

>> **Heroku App**- Heroku is a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud. We have deployed the tool on the Heroku web infrastructure.

>> **Flask-** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. We have used Flask to deploy the model locally.

>> **Scikit-Learn:** The sklearn library contains a lot of efficient tools for machine learning and statistical modelling including classification, regression, clustering and dimensionality reduction.

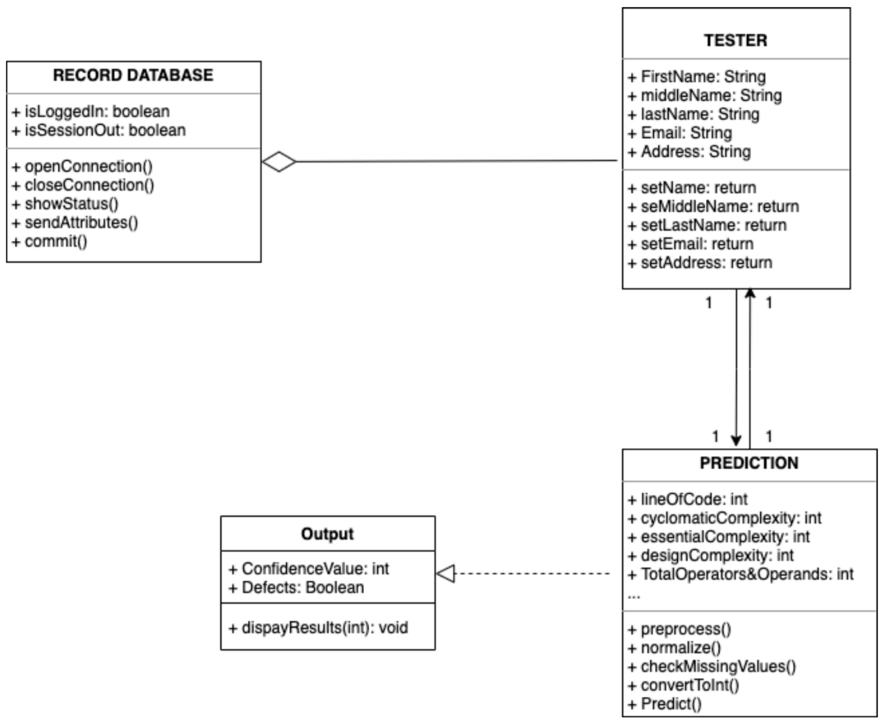
>> **HTML:** The HyperText Markup Language is the standard mark-up language for documents designed to be displayed in a web browser. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

>> **CSS:** CSS is the language for describing the presentation of Web pages, including colors, layout, and fonts. It allows one to adapt the presentation to different types of devices, such as large screens, small screens, or printers. CSS is independent of HTML and can be used with any XML-based markup language.

>> **JAVASCRIPT:** JavaScript is the Programming Language for the Web. JavaScript can update and change both HTML and CSS. JavaScript can calculate, manipulate and validate data.

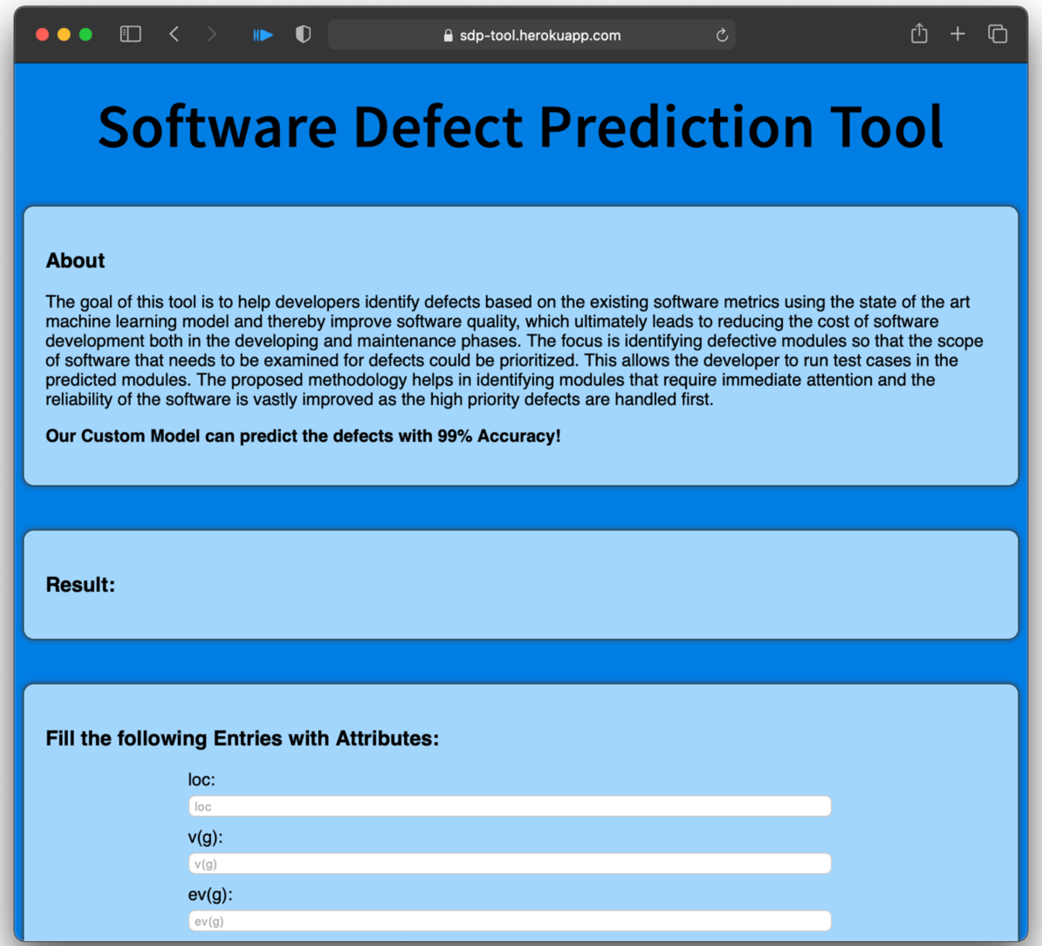
4. DESIGN & APPROACH DETAILS

4.1. Analysis & Design



The prediction module generates the classification result of form the submitted attributes and classifies if the software has gaps/defects.

For this project, we are going to move forward with waterfall model. We have chosen this specific model because it's simplicity and ease of use. Also, the model is rigid and each phase in waterfall model has specific deliverables and a review process. This model will also be very handy to us because no two stages will be occurring simultaneously thus no confusion or other complications will arise. For this scale of product (small product with a two member team for development), task arrangement becomes easier. We will clearly define the software development stages and document the processes and results. We have not used other models because it would increase the complexity of our SDLC and since our project is of a small scale, we do not required added complexity to it.



Software Defect Prediction Tool

About

The goal of this tool is to help developers identify defects based on the existing software metrics using the state of the art machine learning model and thereby improve software quality, which ultimately leads to reducing the cost of software development both in the developing and maintenance phases. The focus is identifying defective modules so that the scope of software that needs to be examined for defects could be prioritized. This allows the developer to run test cases in the predicted modules. The proposed methodology helps in identifying modules that require immediate attention and the reliability of the software is vastly improved as the high priority defects are handled first.

Our Custom Model can predict the defects with 99% Accuracy!

Result:

Fill the following Entries with Attributes:

loc:

v(g):

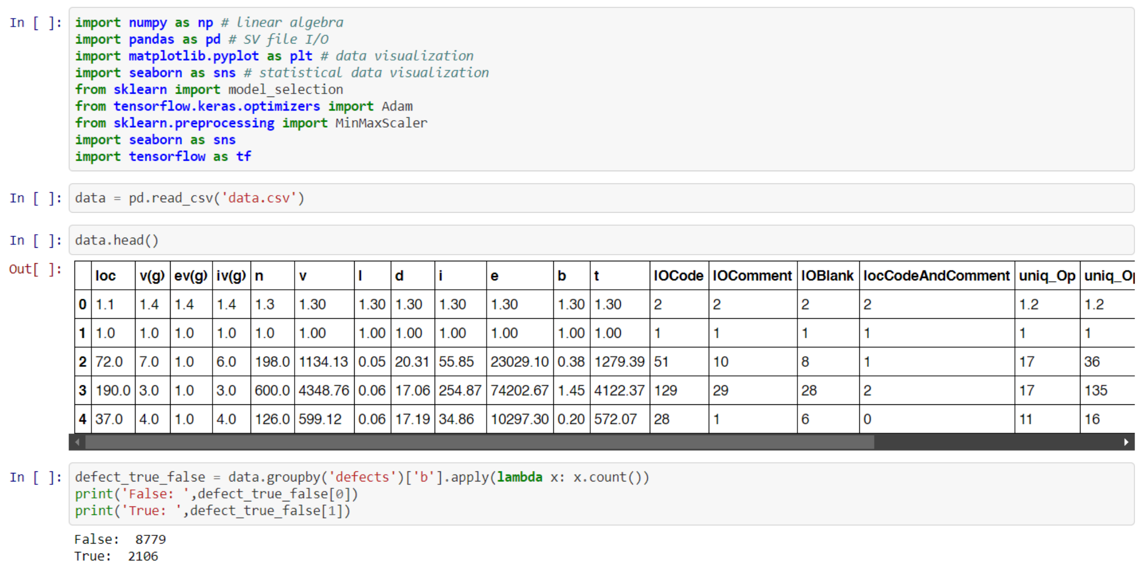
ev(g):

The development will begin with the gathered data and dissect it and to analyse potential predictors for designing the model. As a stand-alone testing team, it is essential to design and deal with the test execution exercises all together to fulfil the tight time constraint for discharging the product to end-clients. In order to deliver the best to the client there needs to be fast defect detection methodologies to provide them with an error free software system.

4.2 CODE:

>> DefectDataAnalysis.ipynb

>> IMPORTING LIBRARIES



```
In [ ]: import numpy as np # linear algebra
        import pandas as pd # SV file I/O
        import matplotlib.pyplot as plt # data visualization
        import seaborn as sns # statistical data visualization
        from sklearn import model_selection
        from tensorflow.keras.optimizers import Adam
        from sklearn.preprocessing import MinMaxScaler
        import seaborn as sns
        import tensorflow as tf
```

```
In [ ]: data = pd.read_csv('data.csv')
```

```
In [ ]: data.head()
```

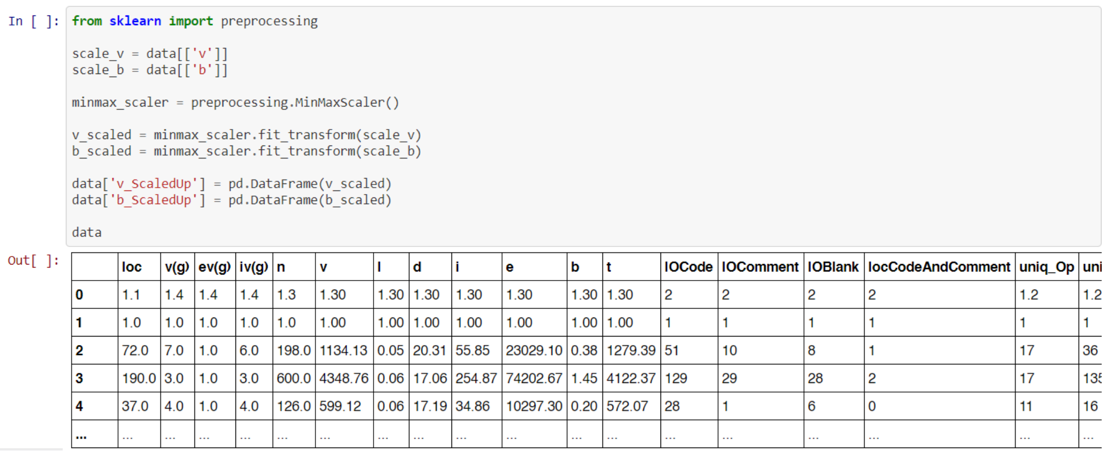
Out[ ]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uniq_O
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	1.30	1.30	2	2	2	2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	0.38	1279.39	51	10	8	1	17	36
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	1.45	4122.37	129	29	28	2	17	135
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	0.20	572.07	28	1	6	0	11	16

```
In [ ]: defect_true_false = data.groupby('defects')['b'].apply(lambda x: x.count())
        print('False: ',defect_true_false[0])
        print('True: ',defect_true_false[1])

        False:  8779
        True:  2106
```

>> DATA PREPROCESSING

$$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$



```
In [ ]: from sklearn import preprocessing

        scale_v = data[['v']]
        scale_b = data[['b']]

        minmax_scaler = preprocessing.MinMaxScaler()

        v_scaled = minmax_scaler.fit_transform(scale_v)
        b_scaled = minmax_scaler.fit_transform(scale_b)

        data['v_ScaledUp'] = pd.DataFrame(v_scaled)
        data['b_ScaledUp'] = pd.DataFrame(b_scaled)

        data
```

Out[ ]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b	t	lOCode	lOComment	lOBlank	locCodeAndComment	uniq_Op	uni
0	1.1	1.4	1.4	1.4	1.3	1.30	1.30	1.30	1.30	1.30	1.30	1.30	2	2	2	2	1.2	1.2
1	1.0	1.0	1.0	1.0	1.0	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1	1	1	1	1	1
2	72.0	7.0	1.0	6.0	198.0	1134.13	0.05	20.31	55.85	23029.10	0.38	1279.39	51	10	8	1	17	36
3	190.0	3.0	1.0	3.0	600.0	4348.76	0.06	17.06	254.87	74202.67	1.45	4122.37	129	29	28	2	17	13
4	37.0	4.0	1.0	4.0	126.0	599.12	0.06	17.19	34.86	10297.30	0.20	572.07	28	1	6	0	11	16
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

>> DATA ANALYSIS

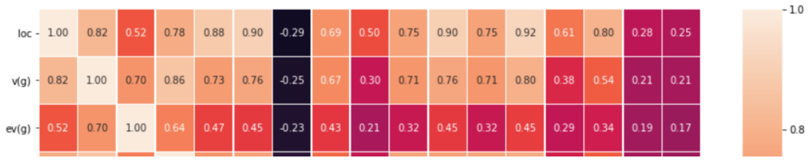
```
In [ ]: print("The shape of the data is: ", data.shape)

        The shape of the data is:  (10885, 22)
```

```
In [ ]: #Simple Statistics
        data.describe()
```

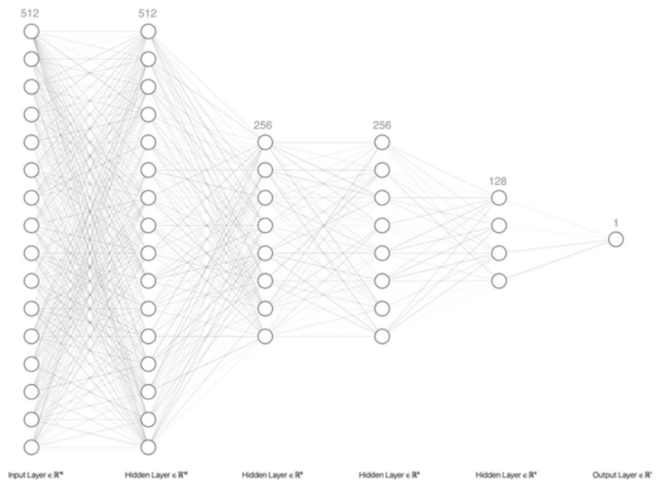
Out[ ]:

	loc	v(g)	ev(g)	iv(g)	n	v	l	d	i	e	b
count	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	10885.000000	1.088500e+04	1088
mean	42.016178	6.348590	3.401047	4.001599	114.389738	673.758017	0.135335	14.177237	29.439544	3.683637e+04	0.22
std	76.593332	13.019695	6.771869	9.116889	249.502091	1938.856196	0.160538	18.709900	34.418313	4.343678e+05	0.64
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.00
25%	11.000000	2.000000	1.000000	1.000000	14.000000	48.430000	0.030000	3.000000	11.860000	1.619400e+02	0.02
50%	23.000000	3.000000	1.000000	2.000000	49.000000	217.130000	0.080000	9.090000	21.930000	2.031020e+03	0.07
75%	46.000000	7.000000	3.000000	4.000000	119.000000	621.480000	0.160000	18.900000	36.780000	1.141643e+04	0.21
max	3442.000000	470.000000	165.000000	402.000000	8441.000000	80843.080000	1.300000	418.200000	569.780000	3.107978e+07	26.9

```
In [ ]: f,ax = plt.subplots(figsize = (15, 15))
        sns.heatmap(data.corr(), annot = True, linewidths = .5, fmt = '.2f')
        plt.show()
```



```
In [ ]: from sklearn.preprocessing import StandardScaler
        sc = StandardScaler()
        X_train = sc.fit_transform(X_train)
        X_test = sc.fit_transform(X_test)
```



```
In [ ]: model = tf.keras.models.Sequential()
        model.add(tf.keras.layers.Dense(units = 512, input_dim=21, activation = 'relu'))
        model.add(tf.keras.layers.Dense(units = 512, input_dim=21, activation = 'relu'))
        model.add(tf.keras.layers.Dense(units = 256, input_dim=21, activation = 'relu'))
        model.add(tf.keras.layers.Dense(units = 256, activation = 'relu'))
        model.add(tf.keras.layers.Dense(units = 128, activation = 'relu'))
        model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))


        opt = Adam(lr=1e-5)

        model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
In [ ]: model.summary()
```

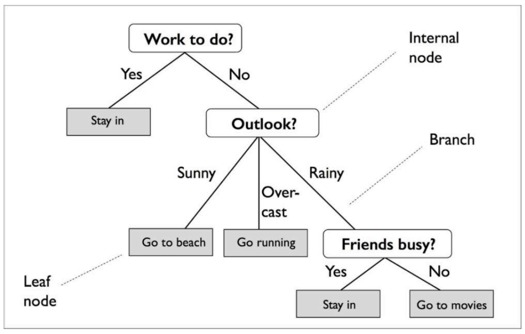
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               11264
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 256)               65792
_________________________________________________________________
dense_4 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 504,065
Trainable params: 504,065
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: model.fit(X_train, Y_train, batch_size = 128, epochs =100)
```

```
Epoch 1/100
```

>> ModelOptimization.ipynb

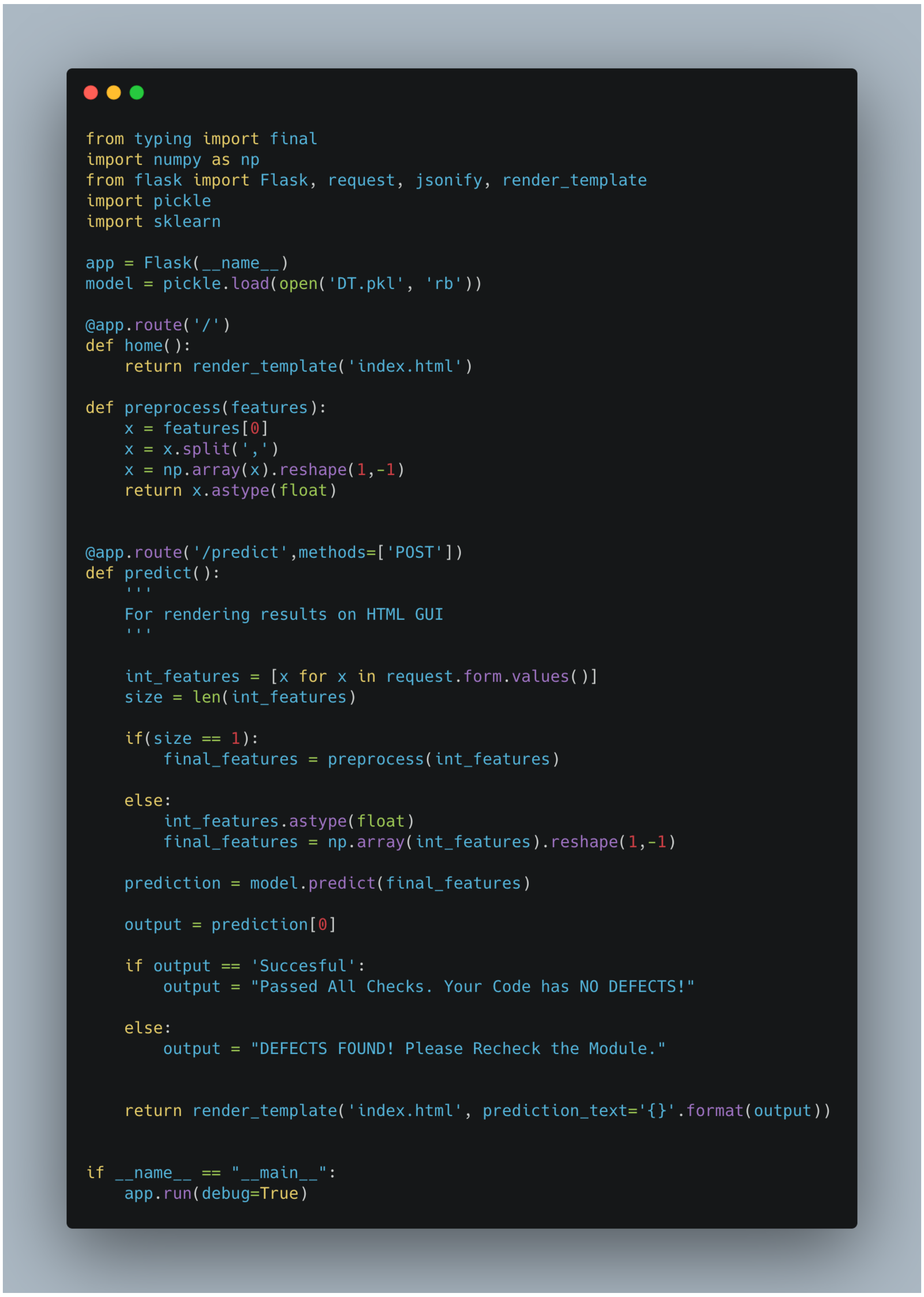
```
In [22]: from sklearn import tree

         model = tree.DecisionTreeClassifier()
```



```
In [23]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)

         model.fit(X_train, y_train)

         y_pred = model.predict(X_test)

         #Summary of the predictions made by the classifier
         print("Decision Tree Algorithm")
         print(classification_report(y_test, y_pred))
         print(confusion_matrix(y_test, y_pred))
         #Accuracy score
         from sklearn.metrics import accuracy_score
         print("ACC: ",accuracy_score(y_pred,y_test))
```

```
Decision Tree Algorithm
              precision    recall  f1-score   support

    Redesign       1.00      1.00      1.00       319
   Succesful       1.00      1.00      1.00      1858

    accuracy                           1.00      2177
   macro avg       1.00      1.00      1.00      2177
weighted avg       1.00      1.00      1.00      2177

[[ 319    0]
 [   1 1857]]
ACC:  0.9995406522737712
```

>> App.py

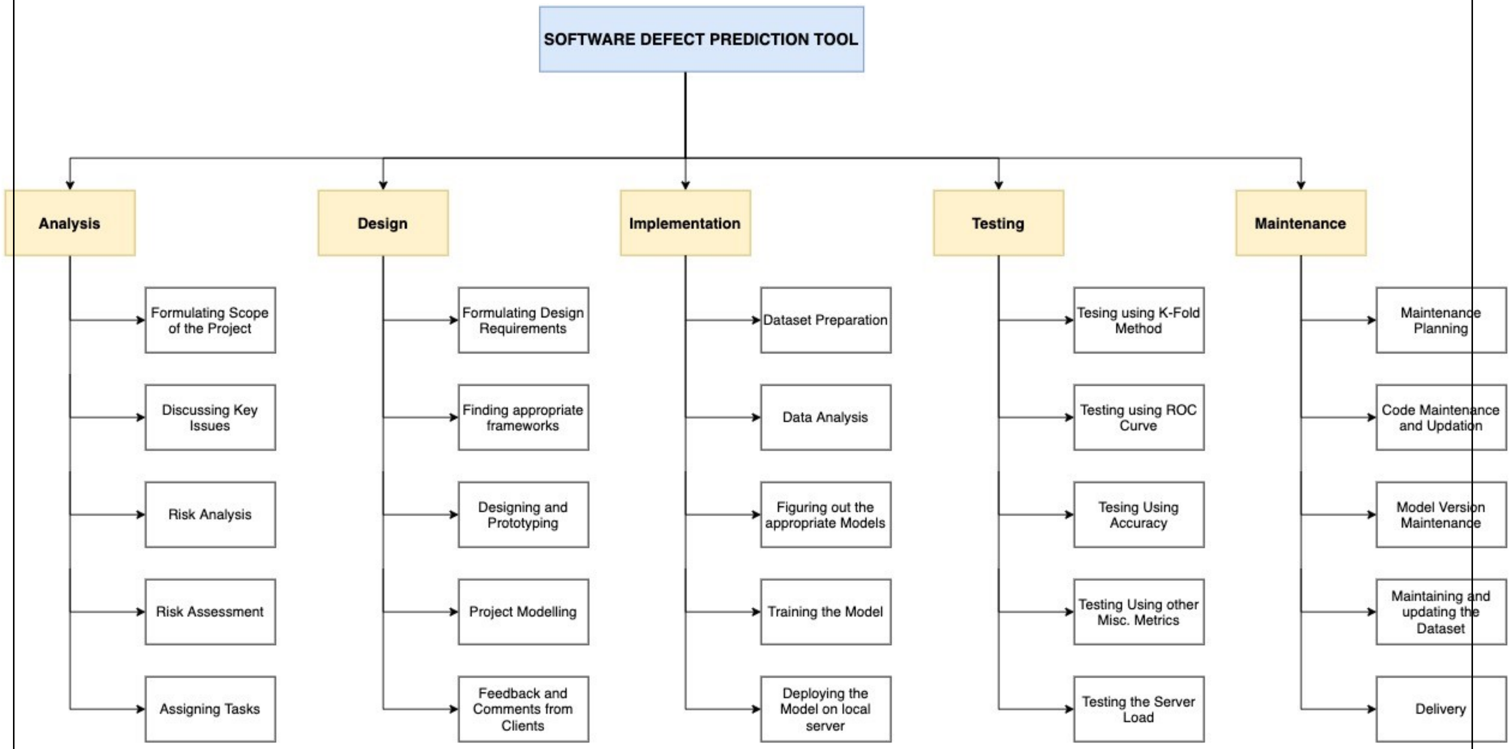


```python
from typing import final
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
import sklearn

app = Flask(__name__)
model = pickle.load(open('DT.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

def preprocess(features):
    x = features[0]
    x = x.split(',')
    x = np.array(x).reshape(1,-1)
    return x.astype(float)


@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''

    int_features = [x for x in request.form.values()]
    size = len(int_features)

    if(size == 1):
        final_features = preprocess(int_features)

    else:
        int_features.astype(float)
        final_features = np.array(int_features).reshape(1,-1)

    prediction = model.predict(final_features)

    output = prediction[0]

    if output == 'Succesful':
        output = "Passed All Checks. Your Code has NO DEFECTS!"

    else:
        output = "DEFECTS FOUND! Please Recheck the Module."


    return render_template('index.html', prediction_text='{}'.format(output))


if __name__ == "__main__":
    app.run(debug=True)
```

5. SCHEDULE, TASKS AND MILESTONES

WORK BREAKDOWN STRUCTURE FOR PROCESS

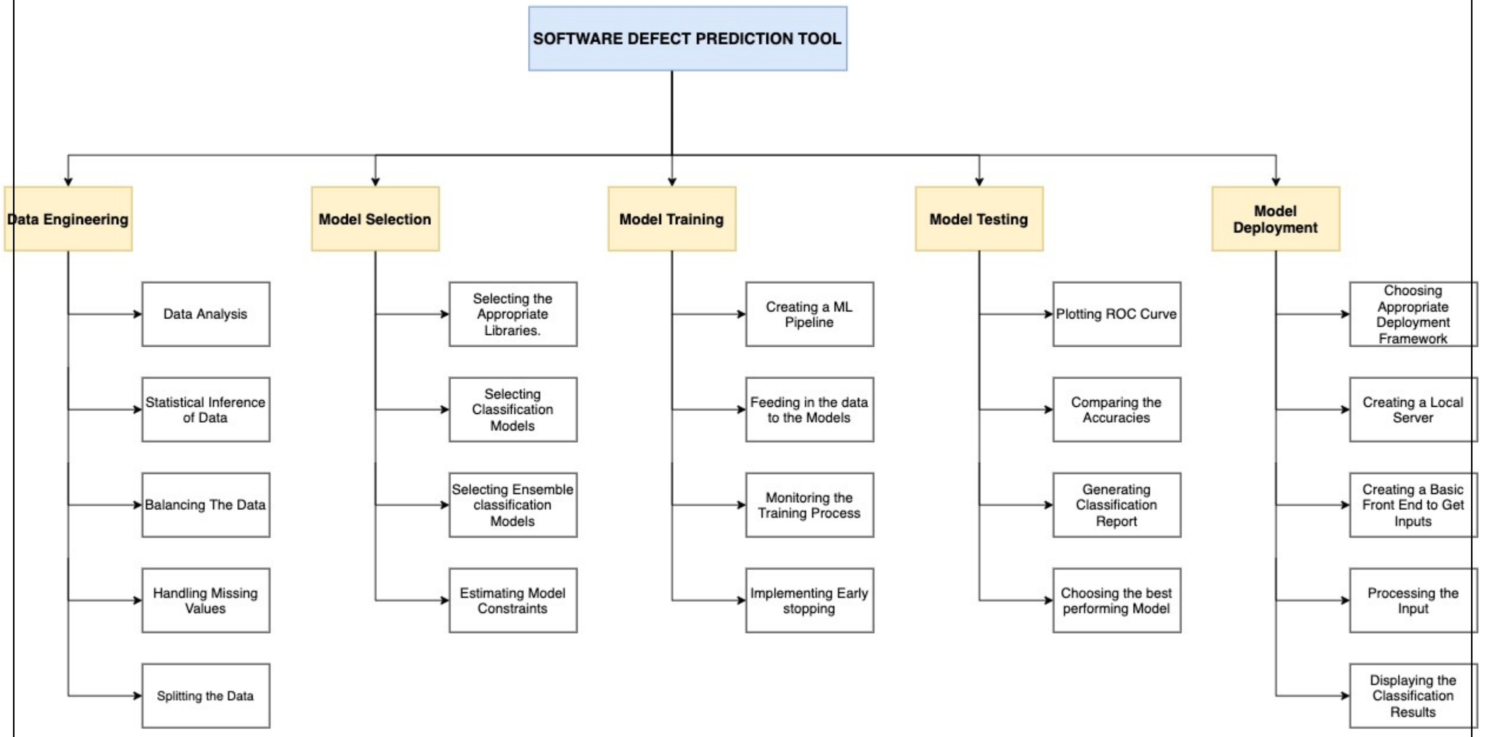
We have used Draw.io open-sourced tool to draw the WBS for the process.



SOFTWARE DEFECT PREDICTION TOOL
Analysis
Formulating Scope of the Project
Discussing Key Issues
Risk Analysis
Risk Assessment
Assigning Tasks
Design
Formulating Design Requirements
Finding appropriate frameworks
Designing and Prototyping
Project Modelling
Feedback and Comments from Clients
Implementation
Dataset Preparation
Data Analysis
Figuring out the appropriate Models
Training the Model
Deploying the Model on local server
Testing
Tesing using K-Fold Method
Testing using ROC Curve
Tesing Using Accuracy
Testing Using other Misc. Metrics
Testing the Server Load
Maintenance
Maintenance Planning
Code Maintenance and Updation
Model Version Maintenance
Maintaining and updating the Dataset
Delivery

WORK BREAKDOWN STRUCTURE FOR PRODUCT

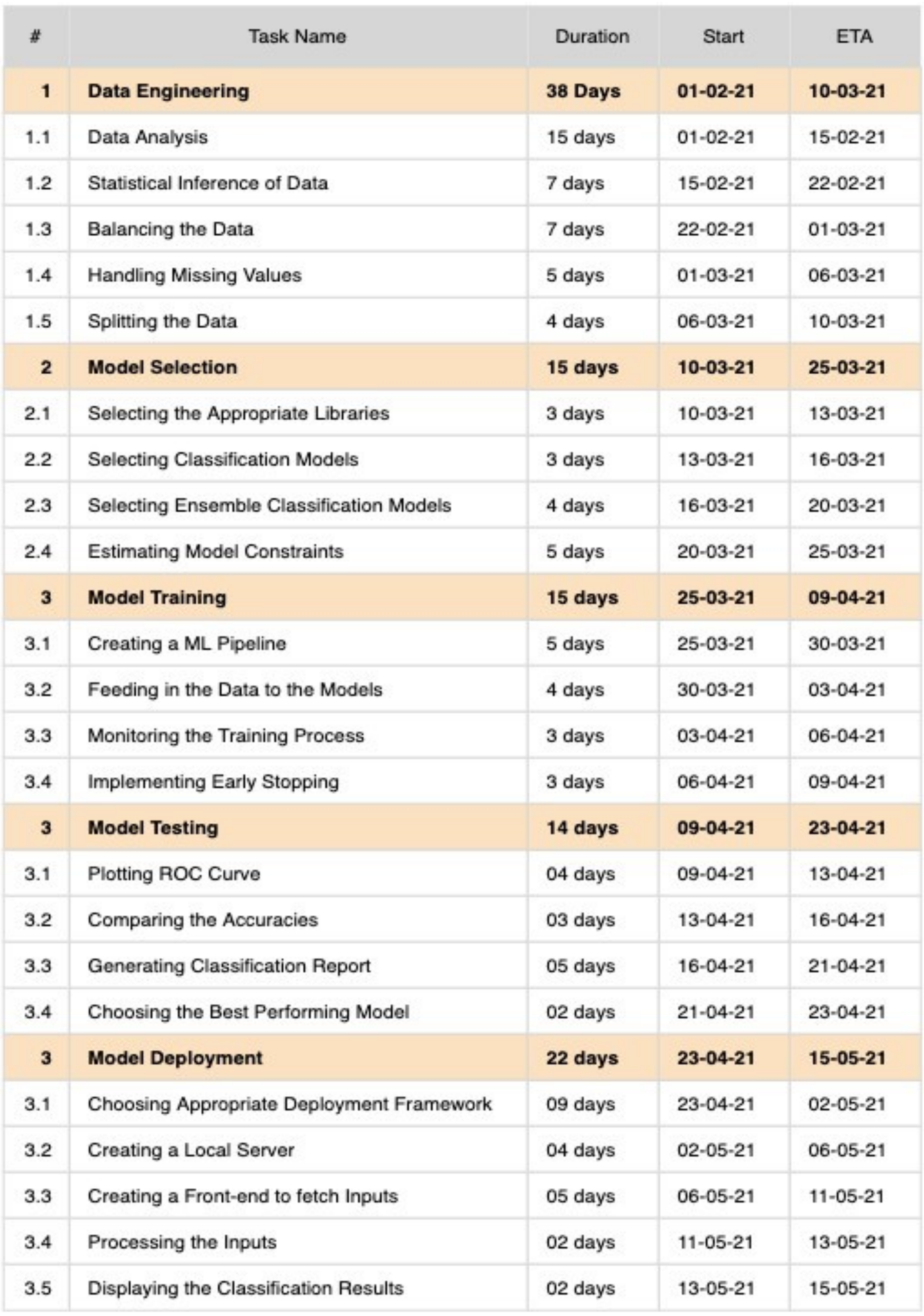
We have used Draw.io open-sourced tool to draw the WBS for the product.



SOFTWARE DEFECT PREDICTION TOOL
Data Engineering
Data Analysis
Statistical Inference of Data
Balancing The Data
Handling Missing Values
Splitting the Data
Model Selection
Selecting the Appropriate Libraries.
Selecting Classification Models
Selecting Ensemble classification Models
Estimating Model Constraints
Model Training
Creating a ML Pipeline
Feeding in the data to the Models
Monitoring the Training Process
Implementing Early stopping
Model Testing
Plotting ROC Curve
Comparing the Accuracies
Generating Classification Report
Choosing the best performing Model
Model Deployment
Choosing Appropriate Deployment Framework
Creating a Local Server
Creating a Basic Front End to Get Inputs
Processing the Input
Displaying the Classification Results

TIMELINE CHART

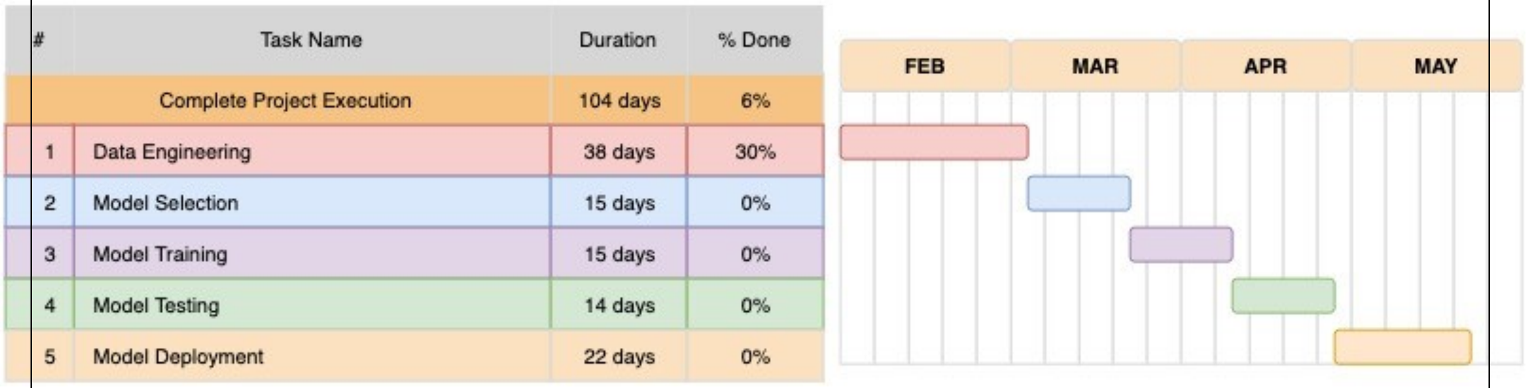
We have used Draw.io open-sourced tool to draw the Gantt chart.



#	Task Name	Duration	Start	ETA
1	**Data Engineering**	**38 Days**	**01-02-21**	**10-03-21**
1.1	Data Analysis	15 days	01-02-21	15-02-21
1.2	Statistical Inference of Data	7 days	15-02-21	22-02-21
1.3	Balancing the Data	7 days	22-02-21	01-03-21
1.4	Handling Missing Values	5 days	01-03-21	06-03-21
1.5	Splitting the Data	4 days	06-03-21	10-03-21
2	**Model Selection**	**15 days**	**10-03-21**	**25-03-21**
2.1	Selecting the Appropriate Libraries	3 days	10-03-21	13-03-21
2.2	Selecting Classification Models	3 days	13-03-21	16-03-21
2.3	Selecting Ensemble Classification Models	4 days	16-03-21	20-03-21
2.4	Estimating Model Constraints	5 days	20-03-21	25-03-21
3	**Model Training**	**15 days**	**25-03-21**	**09-04-21**
3.1	Creating a ML Pipeline	5 days	25-03-21	30-03-21
3.2	Feeding in the Data to the Models	4 days	30-03-21	03-04-21
3.3	Monitoring the Training Process	3 days	03-04-21	06-04-21
3.4	Implementing Early Stopping	3 days	06-04-21	09-04-21
3	**Model Testing**	**14 days**	**09-04-21**	**23-04-21**
3.1	Plotting ROC Curve	04 days	09-04-21	13-04-21
3.2	Comparing the Accuracies	03 days	13-04-21	16-04-21
3.3	Generating Classification Report	05 days	16-04-21	21-04-21
3.4	Choosing the Best Performing Model	02 days	21-04-21	23-04-21
3	**Model Deployment**	**22 days**	**23-04-21**	**15-05-21**
3.1	Choosing Appropriate Deployment Framework	09 days	23-04-21	02-05-21
3.2	Creating a Local Server	04 days	02-05-21	06-05-21
3.3	Creating a Front-end to fetch Inputs	05 days	06-05-21	11-05-21
3.4	Processing the Inputs	02 days	11-05-21	13-05-21
3.5	Displaying the Classification Results	02 days	13-05-21	15-05-21

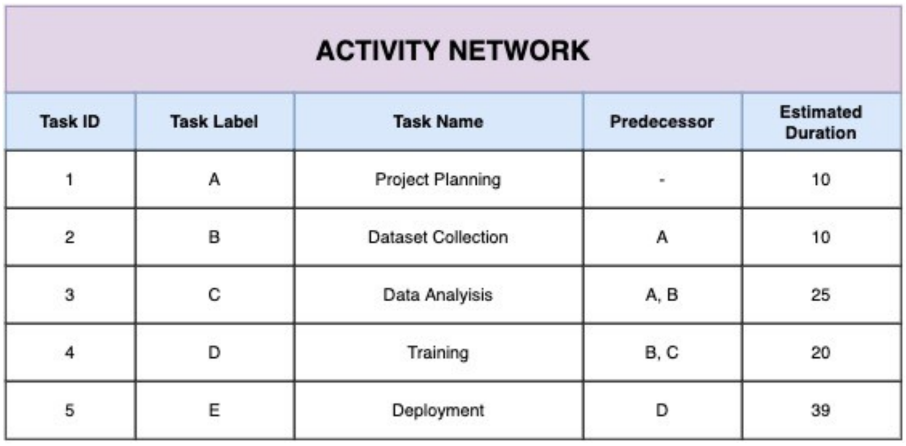
GANTT CHART

We have used Draw.io open-sourced tool to draw the Gantt chart.



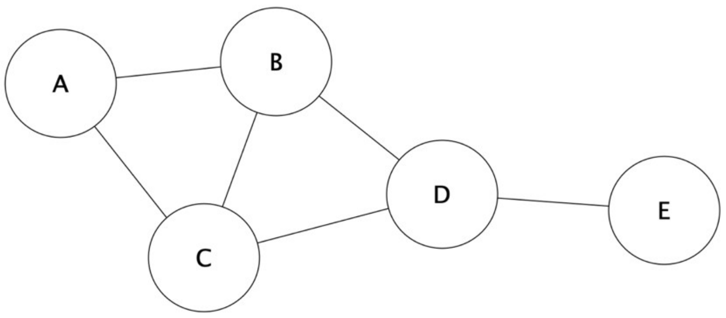
#	Task Name	Duration	% Done
	Complete Project Execution	104 days	6%
1	Data Engineering	38 days	30%
2	Model Selection	15 days	0%
3	Model Training	15 days	0%
4	Model Testing	14 days	0%
5	Model Deployment	22 days	0%

ACTIVITY NETWORK

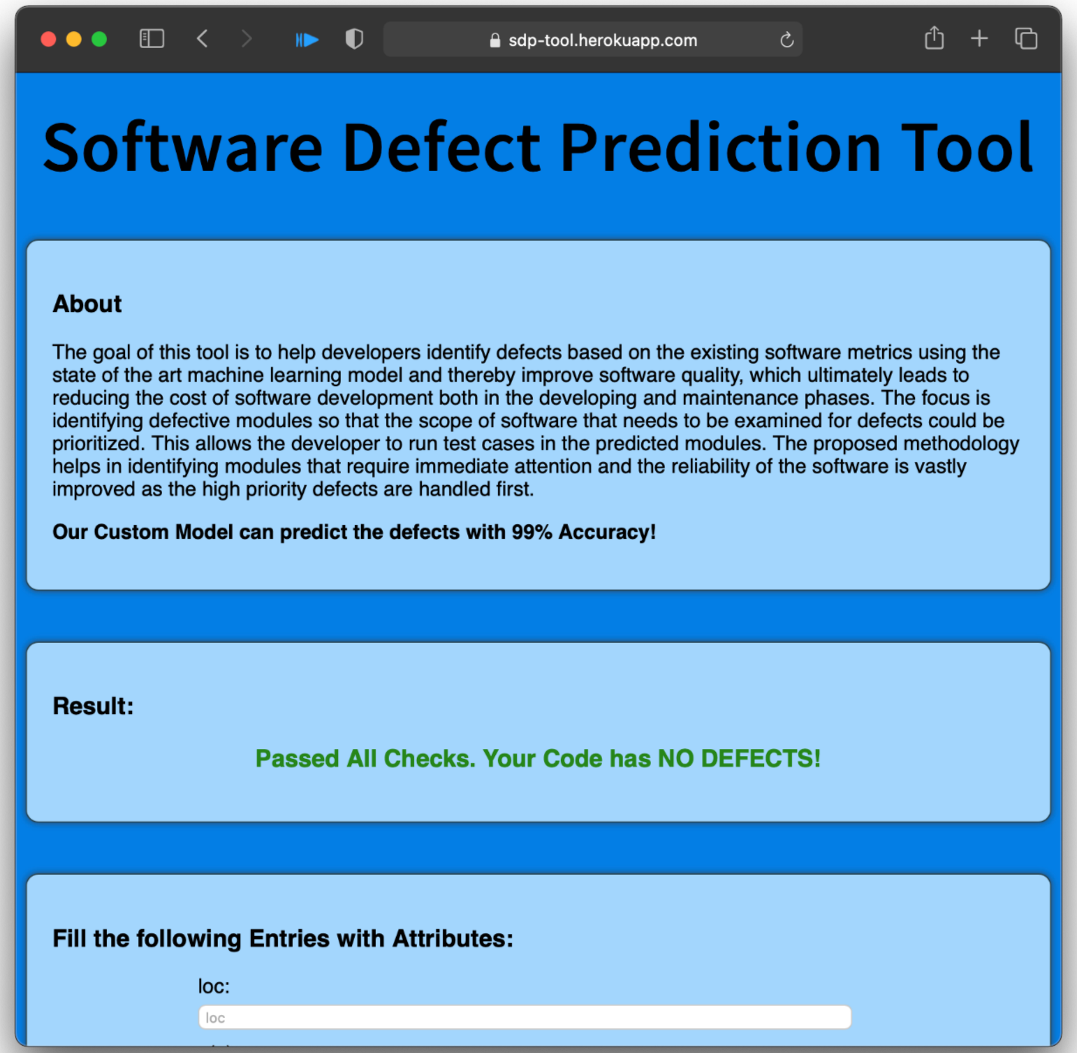


ACTIVITY NETWORK

Task ID	Task Label	Task Name	Predecessor	Estimated Duration
1	A	Project Planning	-	10
2	B	Dataset Collection	A	10
3	C	Data Analyisis	A, B	25
4	D	Training	B, C	20
5	E	Deployment	D	39



6. PROJECT DEMONSTRATION & RESULTS



Software Defect Prediction Tool

About

The goal of this tool is to help developers identify defects based on the existing software metrics using the state of the art machine learning model and thereby improve software quality, which ultimately leads to reducing the cost of software development both in the developing and maintenance phases. The focus is identifying defective modules so that the scope of software that needs to be examined for defects could be prioritized. This allows the developer to run test cases in the predicted modules. The proposed methodology helps in identifying modules that require immediate attention and the reliability of the software is vastly improved as the high priority defects are handled first.

Our Custom Model can predict the defects with 99% Accuracy!

Result:

Passed All Checks. Your Code has NO DEFECTS!

Fill the following Entries with Attributes:

loc:

loc

The novelty of our project is that we can predict the defects probablity in the software with a 99% accuracy rate. This is possible by carefully tuning the hyperparameters of our model and following good data pre-processing techniques. We have also opensourced our project so that other interested researchers can contribute to the project to enhance it's functionalities and build upon the features.

Furthermore, we have provided option to enter the attribuites one by one, or the tester can enter all of them in as comma-seperated values.

7. CONCLUSION

Now a day the development of software based system are increasing from the previous years due to its benefit. However, the quality of the system is required before it is delivered to end users. In order to enhance the software quality, we have various quality metrics such as software testing, CMM and ISO standards. Currently software testing becomes more and more important in the software reliability. Software defects prediction can effectively improve the efficiency of software testing and guide the allocation of resources. For the error-prone modules, we should spend more resource and time.

8. REFERENCES

I. Dalal, S., & Chhillar, R. S. (2012). Case studies of most common and severe types of software system failure. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(8).

II. Rajkumar, G., & Alagarsamy, K. (2013). The most common factors for the failure of software development project. *The International Journal of Computer Science & Applications (TIJCSA)*, *1*(11), 74-77.

III. Chomal, V. S., & Saini, J. R. (2014). Cataloguing most severe causes that lead software projects to fail. *International Journal on Recent and Innovation Trends in Computing and Communication*, *2*(5), 1143-1147.

IV. Park, M., & Hong, E. (2014). Software fault prediction model using clustering algorithms determining the number of clusters automatically. *International Journal of Software Engineering and Its Applications*, *8*(7), 199-204.

V. Erturk, E., & Sezer, E. A. (2015). A comparison of some soft computing methods for software fault prediction. *Expert systems with applications*, *42*(4), 1872-1879.

VI. Phuc, N. V. (2010). *The application of machine learning methods in software verification and validation* (Doctoral dissertation).