Breast Cancer Detection

Contents

[1 Technical Design Document 4](#_Toc47604148)

[1. Introduction 5](#_Toc47604149)

[High level objectives 5](#_Toc47604150)

[2 Workflow Overall 6](#_Toc47604151)

[Application Flow 6](#_Toc47604152)

[Exception Scenarios Overall 6](#_Toc47604153)

[3 Workflow Data Ingestion and File Conversion 7](#_Toc47604154)

[3.1 Method Definitions 7](#_Toc47604155)

[3.2 Exceptions Scenarios 7](#_Toc47604156)

[3 Data Profiling 8](#_Toc47604157)

[3.1 Method Definition 8](#_Toc47604158)

[4 Graph Based EDA 9](#_Toc47604159)

[4.1 Technical solution design 9](#_Toc47604160)

[5 EDA using sweetviz library 10](#_Toc47604161)

[6 Data Transformers( Pre-processing steps) 11](#_Toc47604162)

[6.1 Technical solution design 11](#_Toc47604163)

[6.2 Method Definitions 12](#_Toc47604164)

[6.3 Exceptions Scenarios Module Wise 12](#_Toc47604165)

[7 ML Model Selection 13](#_Toc47604166)

[7.1 Technical solution design 13](#_Toc47604167)

[7.2 Exceptions Scenarios Module Wise 13](#_Toc47604168)

[8 Model Tuning and Optimization 14](#_Toc47604169)

[9 Testing Modules 15](#_Toc47604170)

[Exceptions Scenarios Module Wise 15](#_Toc47604171)

[10 Prediction Pipeline 16](#_Toc47604172)

[10.1 Technical solution design 17](#_Toc47604173)

[10.2 Exceptions Scenarios Module Wise 18](#_Toc47604174)

[11 Deployment Strategy 19](#_Toc47604175)

[11.1 Exceptions Scenarios Module Wise 19](#_Toc47604176)

[12 Dockerization 20](#_Toc47604177)

[12.1 Steps to Dockerize App 20](#_Toc47604178)

[13 Logging 24](#_Toc47604179)

[13.1 Technical solution design 24](#_Toc47604180)

[13.2 Common Logging Framework Code 24](#_Toc47604181)

[13.3 Exceptions Scenarios Module Wise 24](#_Toc47604182)

[14 Sample code and standard to be followed: 25](#_Toc47604183)

# Technical Design Document

Version 1.0

Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 18th July 2020 | 1.0 | Initial Draft | Jane Alam  Hemant Gautam |

Contributors

The content of this document has been authored with the combined input of the following group of key individuals.

|  |  |
| --- | --- |
| Name | Section Worked Upon |
| Jane Alam  Hemant Gauam | Initial Draft |

# Introduction

The goal here is to build an end to end Machine Learning solution for breast cancer detection where the user will only give the data(either by adding single patient data or csv file) and the result will be the shown wheather the cancer type is M(1) = malignant or B(0) = benign on UI.

This project shall be delivered in one phase:

Phase1: Integration of UI to all the functionalities(predicting the result).

The technical design document gives a design blueprint of the Breast Cancer detection project. This document communicates the technical details of the solution proposed.

In addition, this document also captures the different workflows involved to build the solution, exceptions in the workflows and any assumptions that have been considered.

Once agreed as the basis for the building of the project, the flowchart and assumptions will be used as a platform from which the solution will be designed.

**Note: All the code will be written in python version 3.8.3**

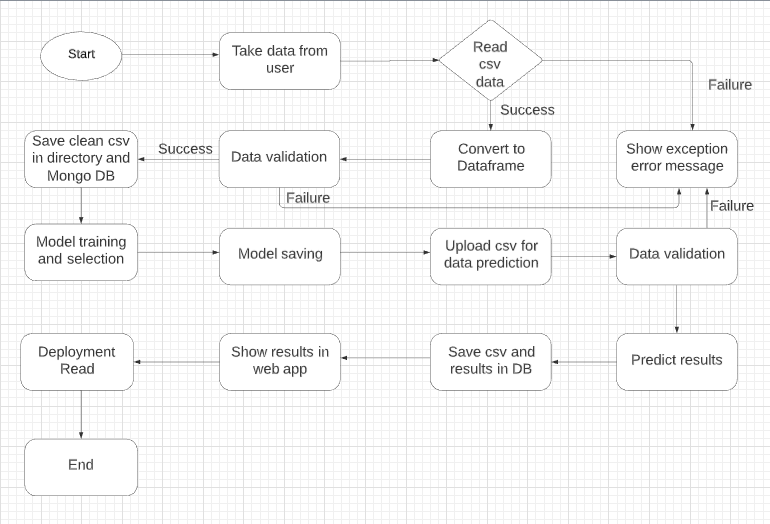
## High level objectives

The high-level objectives are:

1. Load csv data into mongo db after cleaning it and convert that data into dataframe.
2. Perform data cleaning operation with all the steps required and showcase a report on screen.
3. After data cleaning showcase the graphical analysis once again for comparison.
4. Choose the appropriate ML model for training.
5. Perform model Tuning.
6. Create a list of top 4 models and show multiple metrics for them.

# Workflow Overall

## Application Flow



## Exception Scenarios Overall

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Unable to read data from database | Give proper error message | Database credentials needs to be checked |

# Workflow Data Ingestion and File Conversion

**Data Sources:**

|  |  |
| --- | --- |
| Data Storage | Provided File Type |
| [Mongo](https://help.tableau.com/current/pro/desktop/en-us/examples_access.htm) DB | CSV |

## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DatabaseConnect** | **As soon as the object of this class is created, connection will be establish with mongo db.** |
| Method Name | storeTrainTestCSVToDB |  |
|  | Method Description | This method will be used to to store final cleaned train/test csv data file into DB |
|  | Input parameter names | self, df |
|  | Input Parameter Description | Df: complete dataframe is getting passed in this method to store in DB |
|  | ouptput | DF should be store in DB |
|  | On Exception | Failing to store DF will be captured in training log file |

|  |  |  |
| --- | --- | --- |
| Method Name | storePredictCSVToDB |  |
|  | Method Description | This method will be used to to store predicted csv data into database after all data validation process. |
|  | Input parameter names | self, df |
|  | Input Parameter Description | Df: complete dataframe is getting passed in this method to store in DB |
|  | ouptput | DF should be store in DB |
|  | On Exception | Failing to store DF will be captured in training log file |

|  |  |  |
| --- | --- | --- |
| Method Name | storePredictedResult |  |
|  | Method Description | This method will be used to to store final cleaned train/test csv data file into DB |
|  | Input parameter names | self, df |
|  | Input Parameter Description | Df: complete dataframe is getting passed in this method to store in DB |
|  | ouptput | DF should be store in DB |
|  | On Exception | Failing to store DF will be captured in training log file |

|  |  |  |
| --- | --- | --- |
| Method Name | fetchPredictedResults |  |
|  | Method Description | This method will be used to to store final cleaned train/test csv data file into DB |
|  | Input parameter names | self |
|  | ouptput | DF should be store in DB |
|  | On Exception | Failing to store DF will be captured in training log file |

## Exceptions Scenarios

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Unable to read data from database | Give proper error message | Database credentials needs to be checked |

# Data Profiling

After reading the data, automatically the following details should be shown:

1. The number of rows
2. The number of columns
3. Total missing values and it’s percentage
4. Number of categorical columns and their list
5. Number of numerical columns and their list
6. Number of duplicate rows

## Method Definition

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataProfiler** |  |
| Method Name | get\_data\_profile |  |
|  | Method Description | This method will be used to give various insighst about data. |
|  | Input parameter names | self, dataframe |
|  | Input Parameter Description | dataframe: the inpt data just loaded from source |
|  | ouptput | 1. The number of rows 2. The number of columns 3. Total missing values and it’s percentage 4. Number of categorical columns and their list 5. Number of numerical columns and their list 6. Number of duplicate rows |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

# Graph Based EDA

Create the following graphs :

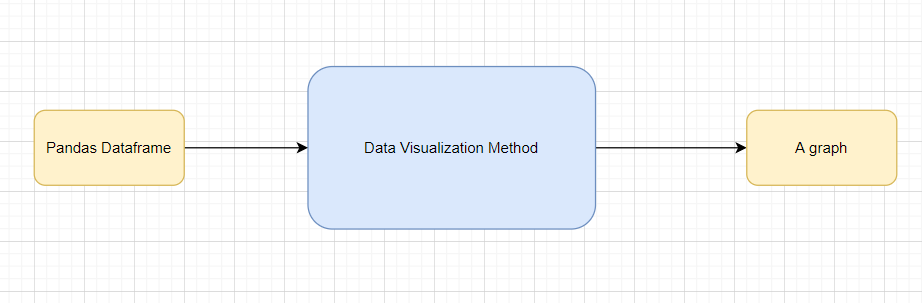
Correlation Heatmaps

Count plots

Barplots

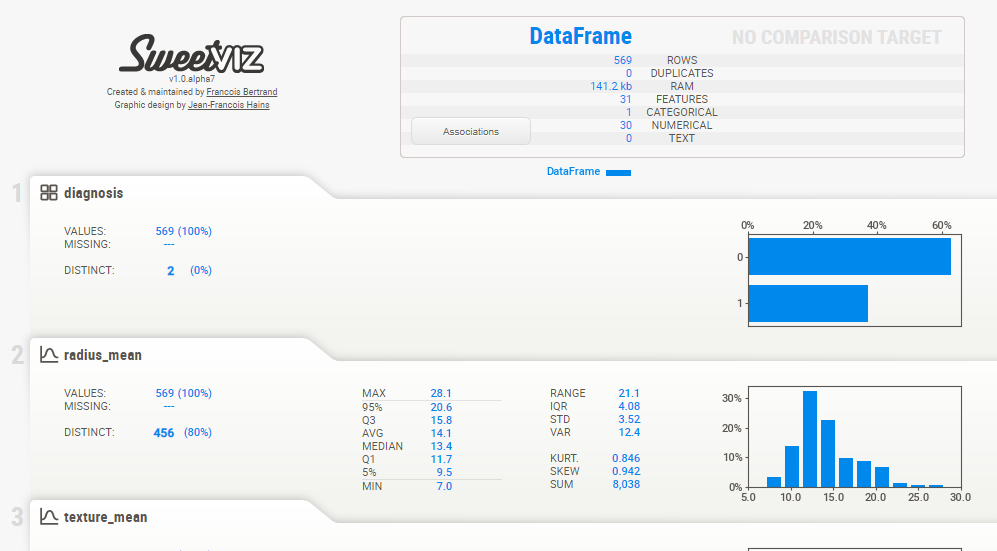
Scatterplot

## Technical solution design



# EDA using sweetviz library

Sweetviz is a automatic EDA library which provides information about each columns with graphs in just few seconds by creating html file in local working directory.



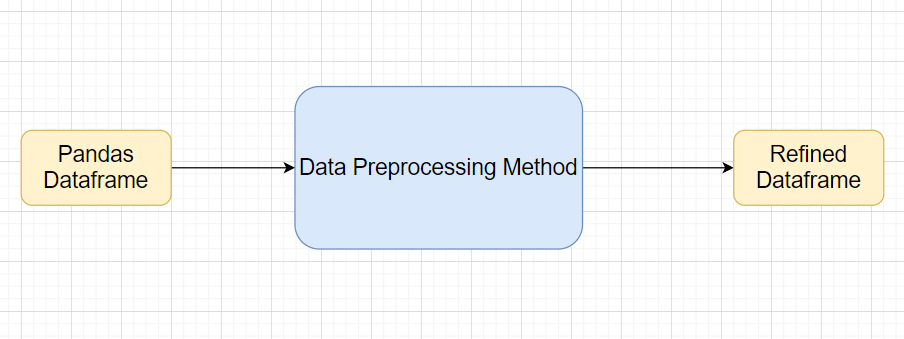
# Data Transformers( Pre-processing steps)

**MVP:**

Null value handling

Categorical to numerical

## Technical solution design



## Method Definitions

|  |  |  |
| --- | --- | --- |
| **Class Name** | **DataPreprocessor** |  |
| Method Name | impute\_missing\_values |  |
|  | Method Description | This method will be used to read data from a csv file or a flat file |
|  | Input parameter names | self,file\_name, header,names, use\_cols, separator |
|  | Input Parameter Description | file\_name: name of the file to be read  header: Row number(s) to be used as column names  names : array-like, optional  List of column names to use. If file contains no header row, then you  should explicitly pass ``header=None``.  Use\_cols: To load a subset of columns  Separator: Delimiter to use |
|  | ouptput | A pandas Dataframe |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

## Exceptions Scenarios Module Wise

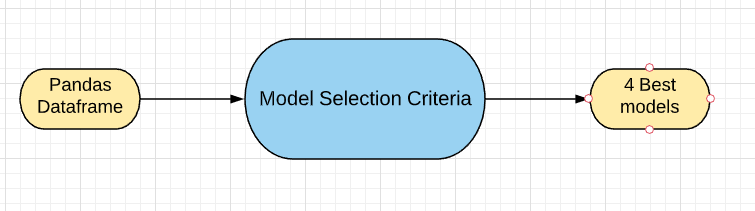
|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# ML Model Selection

4 Models— **Logistic Regression**, **Random Forest**, **SVC, KNN**

Model Selection criteria: train and test split data will be passed in all 4 algorithms and best performed model’s pickle file will be chosen for prediction and kept inside “model/final\_model” folder.

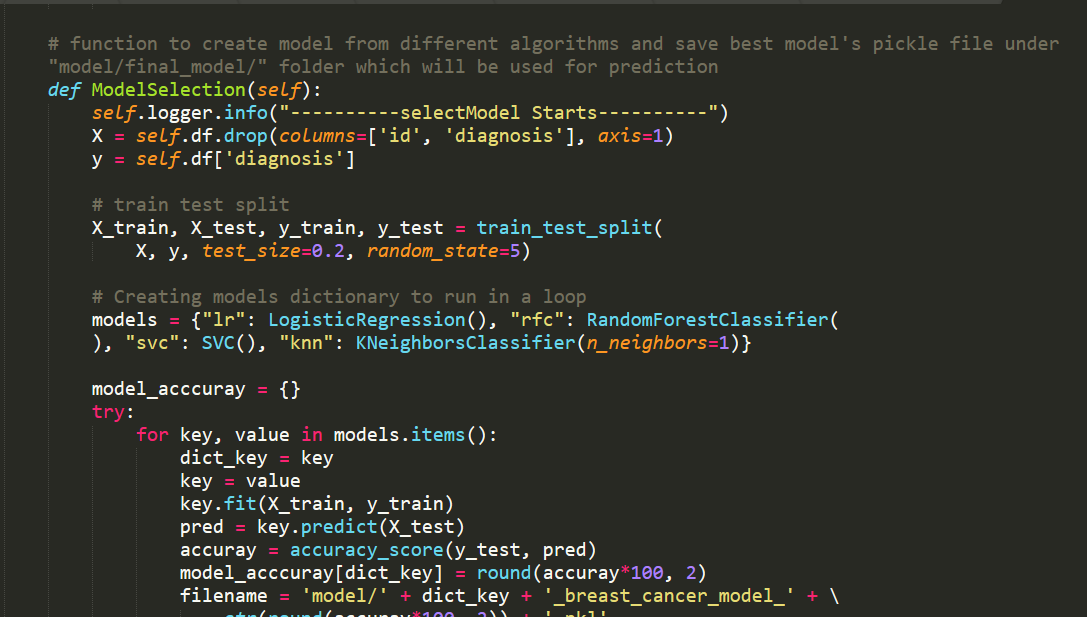
## Technical solution design

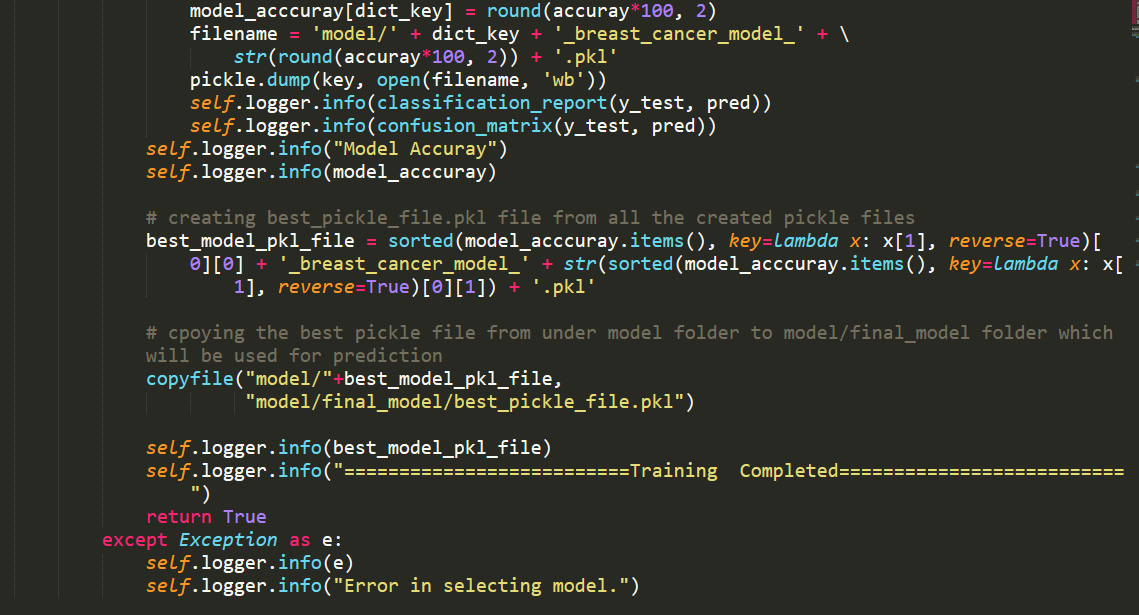


## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

Model selection method screenshot:





# Model Tuning and Optimization

**Note:** The data should have been divided into train and validation set before this.

**Classification:**

Logistic Regression

Decision Tree

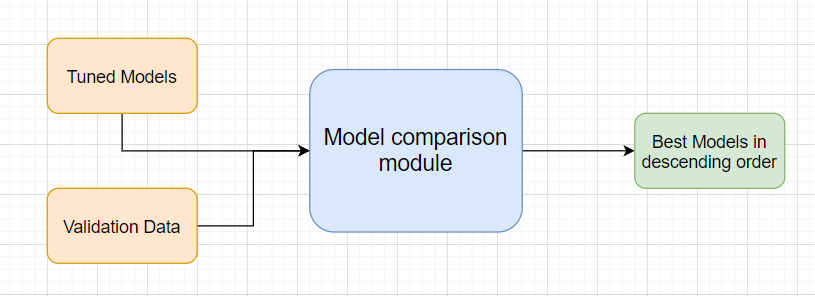
Random Forest

SVM

KNN Classifier

Model selection criteria:

Accuracy, AUC, Precision, Recall



# Testing Modules

Divide the training data itself into train and test sets

Use test data to have tests run on the three best models

Give the test report

1. Accuracy
2. Precision
3. Recall
4. F Beta
5. Confusion matrix

**Note**: Save the best model after validation is completed.

## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Number of Parameters do not match | Handle internally | Check the test data creation and verify the columns |
| Only once class present in test data | Handle Internally |  |

# Prediction Pipeline

Use the existing data read modules

Use the existing pre-processing module

Load the model into memory

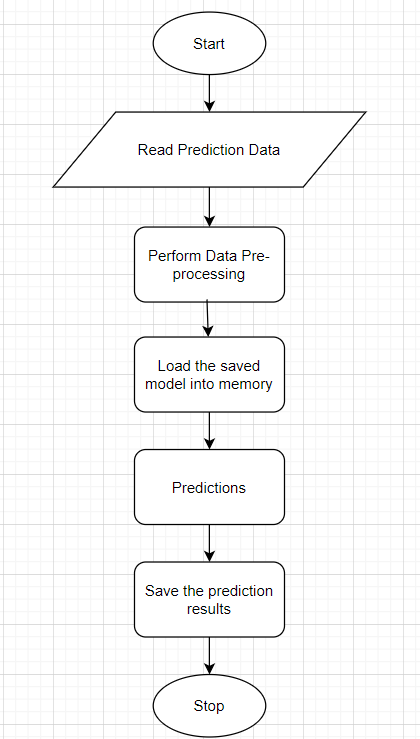
Do predictions

Store prediction results(show sample predictions)

Phase 2:

UI for predictions

## Technical solution design



## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Columns don’t match in training and Prediction data | Show error message | The user enters the correct data |
|  |  |  |

# Deployment Strategy

Take the cloud name as input

Prepare the metadata files based on cloud

Phase 2:

Accept the user credentials

Prepare a script file to push changes

Docker instance

Push of the docker instance to cloud

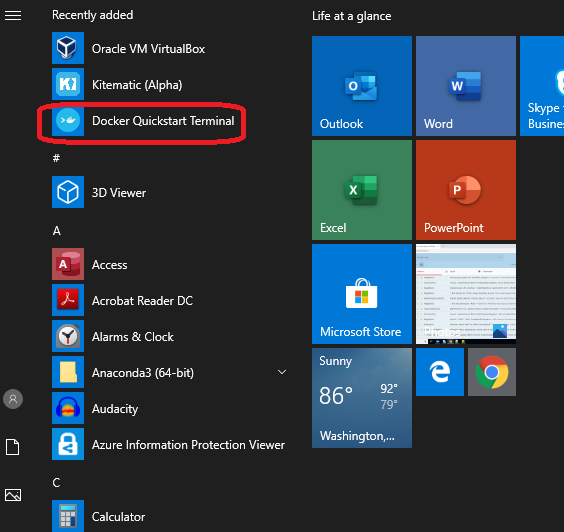
## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
|  |  |  |

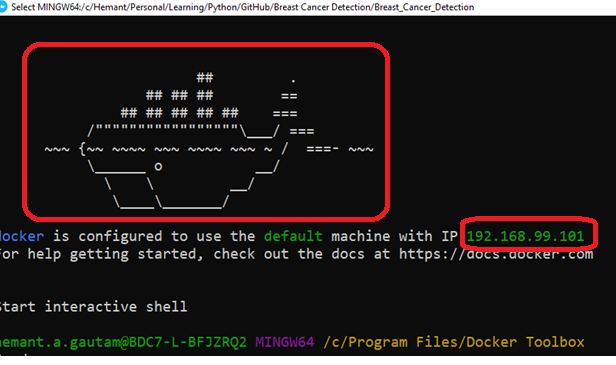
# Dockerization

## Steps to Dockerize App

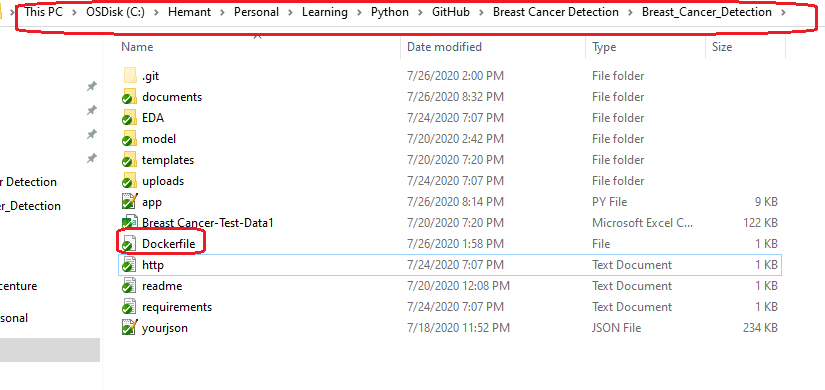
1. Download and run docker toolkit exe file from <https://github.com/docker/toolbox/releases> and if the downloaded version doesn’t support in your machine, try with other lower or higher versions of toolkit.
2. After successful installation, go to start and click on Docker QuickStart Terminal. Refer image (highlighted in red)

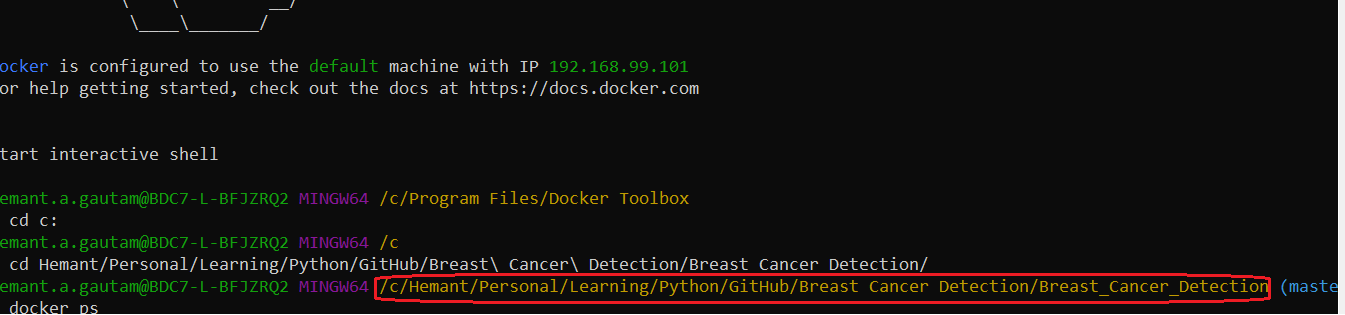


1. This will open a docker command prompt and will take some time to load properly. There will be Docker image at first and machine IP which will be used to access flask app. Refer image (highlighted in red)

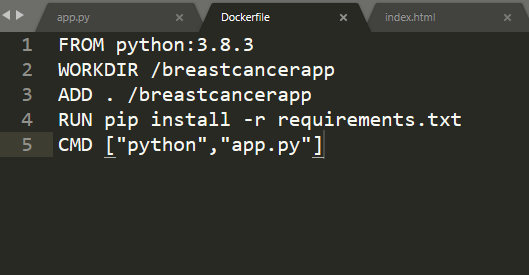


1. Next step is to navigate to the project root folderin docker cmd where Dockerfile is present. Refer image (highlighted in red)

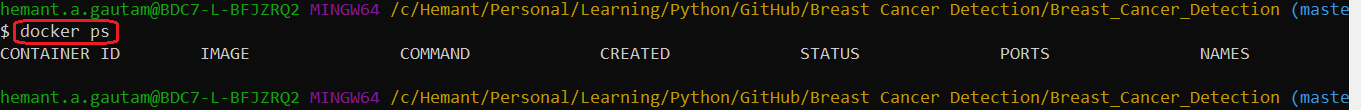




1. Creating docker file is important to dockerize flask app. And to create Dockerfile file, these are the commands should be mentioned. Refer image (highlighted in red



1. Command to check if any existing docker image is running or not. There is no image showing in the below image –



1. Command to create image, this will download python version mentioned in Dockerfile and all the other dependent packages(from requirements.txt):

docker image build -t breastcancer-image .

1. Command to check if image is created successfully:

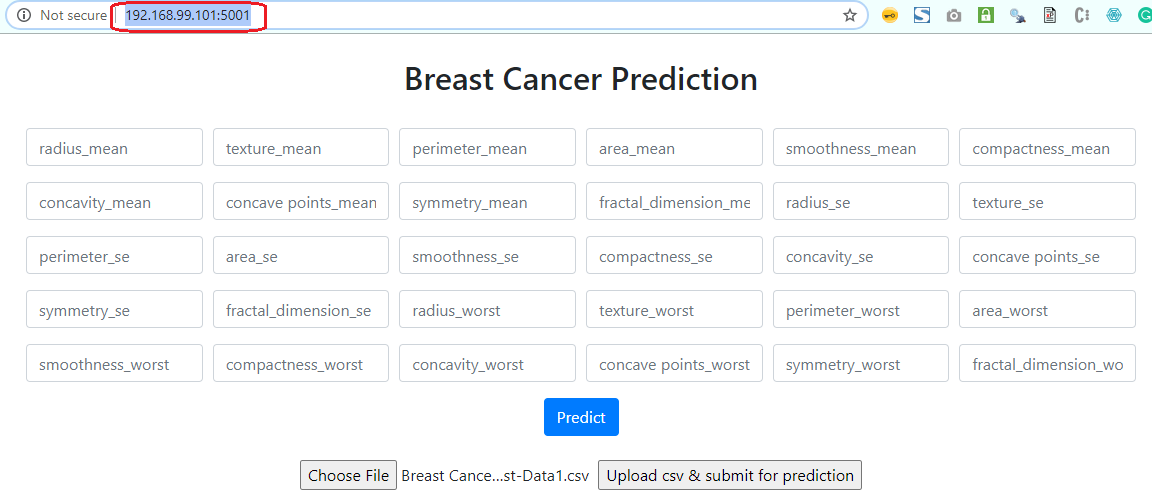
docker image ls

1. Finally to run the created image locally, run this command –

docker run -p 5001:5000 -d breastcancer-image

1. To access the dockerize flask app, use the IP address which shown at the time of opening docker toolkit along with port 5001, and paste that in browser -

<http://192.168.99.101:5001/>



1. Dockerization is completed.

Docker Reference Link –

<https://medium.com/@tasnuva2606/dockerize-flask-app-4998a378a6aa>

# Logging

For logging in this project, python logging package is used

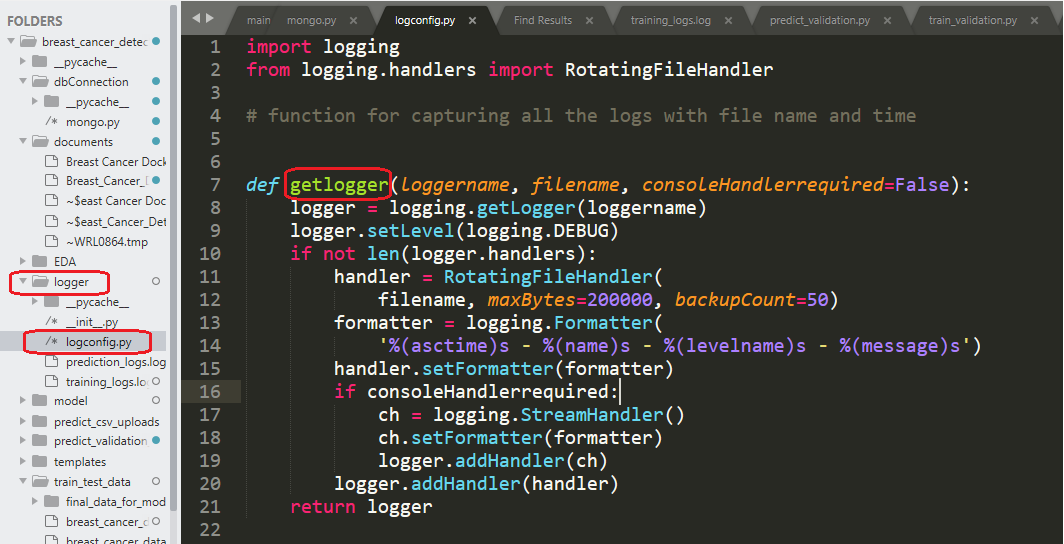
* Separate folder for prediction and trainings logs
* Logging of every step with timestamp
* Entry to the methods
* Exit from the methods with success/ failure message
* Error message Logging
* Model comparisons
* Training start and end
* Prediction start and end

## Technical solution design



## Common Logging Framework Code

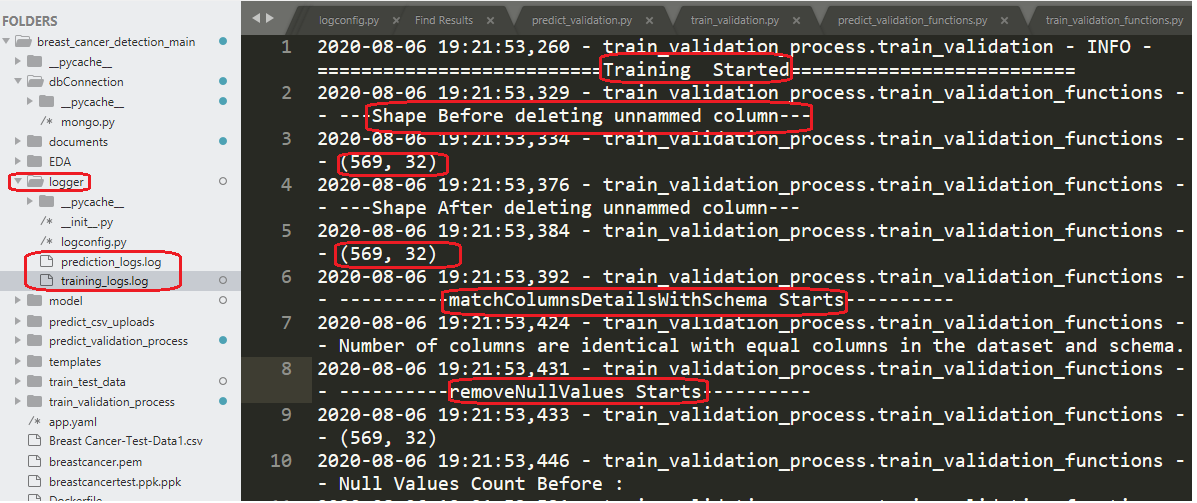
|  |  |
| --- | --- |
| Function name | getlogger |
| Method Description | This function will be used for logging all the prediction and trainings information in separate files. |
| Input parameter names | loggername, filename, consoleHandlerrequired=False |
| Input Parameter Description | loggername : Message to log in files  filename: From which file message is getting logged |
| ouptput | A log file with messages |

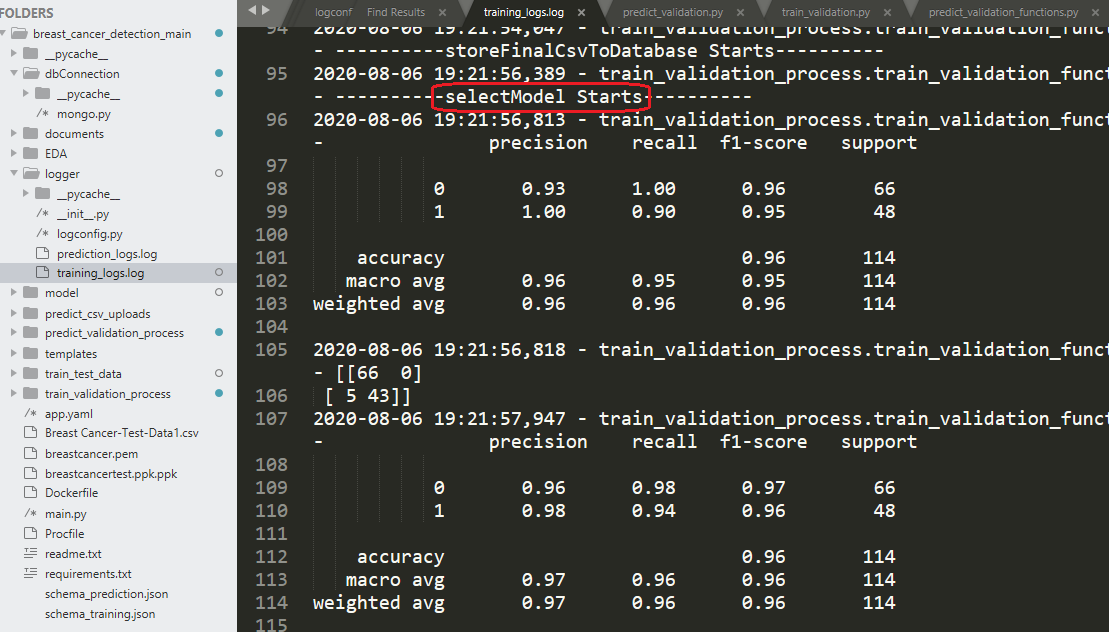


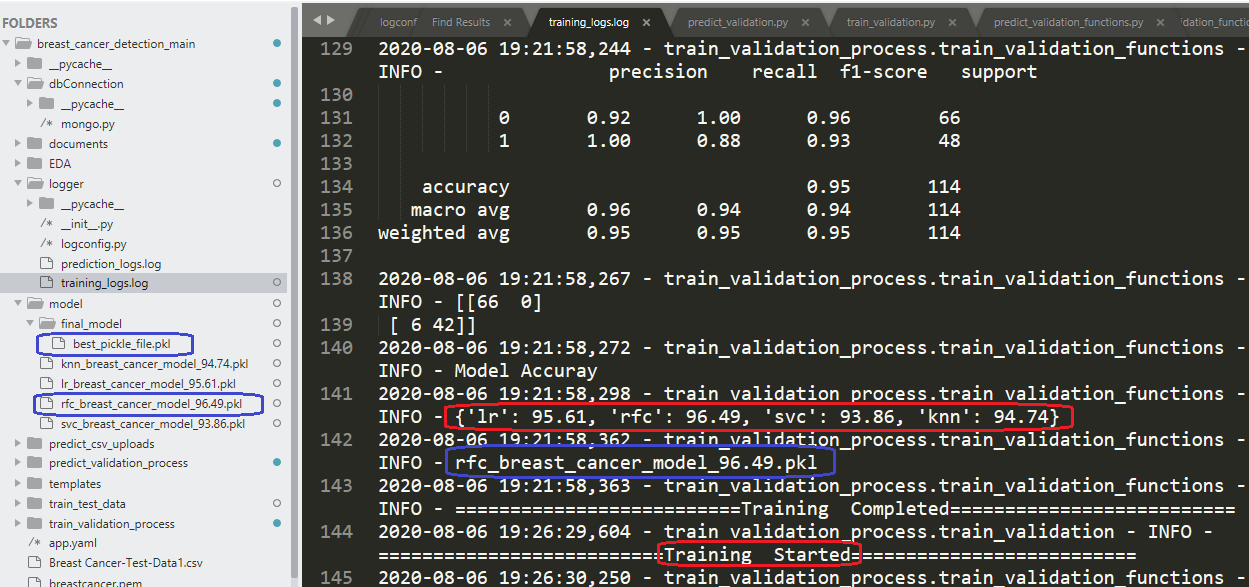
## Exceptions Scenarios Module Wise

Ideally, the logging should never fail.

Training logs screen shot -

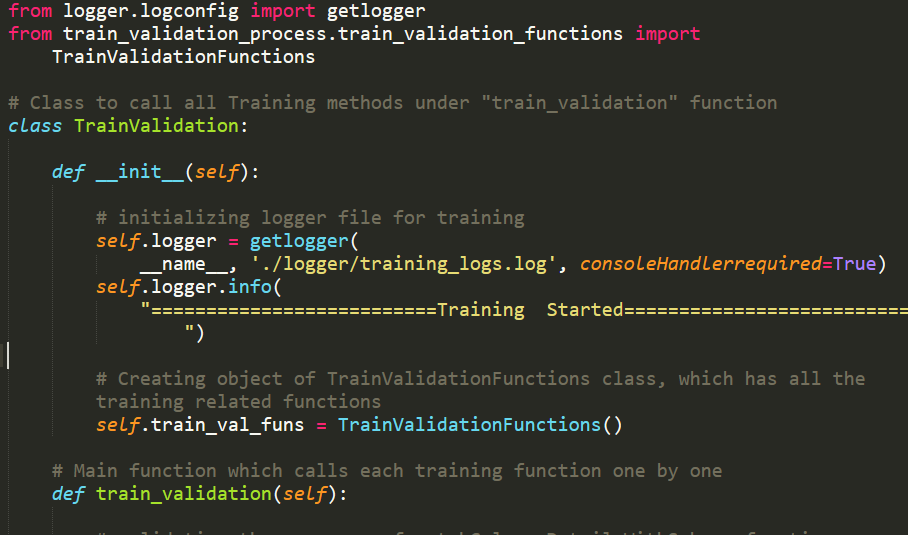


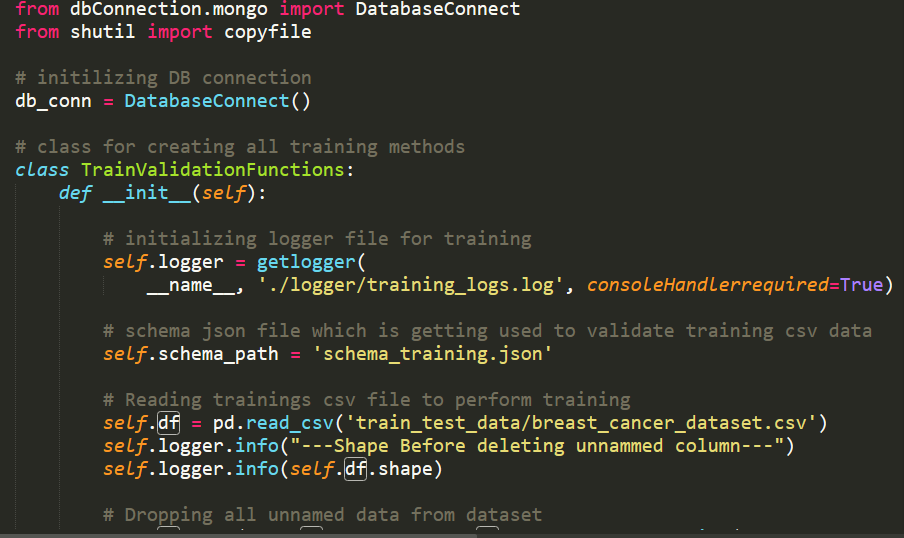




# Sample code and standard to be followed:

Sample Code:





Coding Standard:

1. Imports should usually be on separate lines
2. Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing.
3. Compound statements (multiple statements on the same line) are generally discouraged
4. Comments should be complete sentences. Always make a priority of keeping the comments up-to-date when the code changes. Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.
5. Never use the characters 'l' (lowercase letter el), 'O' (uppercase letter oh), or 'I' (uppercase letter eye) as single character variable names.
6. The name of the variables should start with small case capital letters and a multi word variable should be named as: word1\_word2\_word3.
7. The variable name should be appropriate based on the things that they do. DO NOT USE NAMES LIKE x, k, y etc. Always use a meaningful English word. For example, customer\_name, nearest\_neighbour etc.
8. Method names should start with small case characters. They should start with a verb and make a meaningful sense of what they are supposed to accomplish. For e.g.: load\_data\_from\_sql()
9. Always use self for the first argument to instance methods.
10. Class names should normally use the CapWords convention. Class name should also represent the functionality of the class. For e.g. DataLoader()
11. Modules/Packages/Folders should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. For e.g.: data\_ingestion
12. Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL.
13. Comparisons to singletons like None should always be done with is or is not, never the equality operators
14. The code should be properly enclosed withing try and exception blocks and the exceptions should be handled with proper error messages.
15. Additionally, for all try/except clauses, limit the try clause to the absolute minimum amount of code necessary. Again, this avoids masking bugs
16. When a resource is local to a particular section of code, use a with statement to ensure it is cleaned up promptly and reliably after use.
17. Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. If any return statement returns an expression, any return statements where no value is returned should explicitly state this as return None, and an explicit return statement should be present at the end of the function (if reachable)
18. Object type comparisons should always use isinstance() instead of comparing types directly
19. Don't compare boolean values to True or False using ==