Breast Cancer Detection

Contents

[1 Technical Design Document 4](#_Toc47819924)

[1. Introduction 5](#_Toc47819925)

[High level objectives 5](#_Toc47819926)

[2 Workflow Overall 6](#_Toc47819927)

[Application Flow 6](#_Toc47819928)

[Exception Scenarios Overall 6](#_Toc47819929)

[3 Workflow Data Ingestion and File Conversion 7](#_Toc47819930)

[3.1 Method Definitions 7](#_Toc47819931)

[3.2 Exceptions Scenarios 9](#_Toc47819932)

[3 Data Profiling 11](#_Toc47819933)

[3.1 Method Definition 11](#_Toc47819934)

[4 Graph Based EDA 12](#_Toc47819935)

[4.1 Technical solution design 16](#_Toc47819936)

[5 Data Transformers( Pre-processing steps) 17](#_Toc47819937)

[5.1 Technical solution design 17](#_Toc47819938)

[5.2 Method Definitions 18](#_Toc47819939)

[5.3 Exceptions Scenarios Module Wise 18](#_Toc47819940)

[6 ML Model Selection 19](#_Toc47819941)

[6.1 Technical solution design 19](#_Toc47819942)

[6.2 Exceptions Scenarios Module Wise 19](#_Toc47819943)

[7 Model Tuning and Optimization 21](#_Toc47819944)

[8 Testing Models 22](#_Toc47819945)

[Exceptions Scenarios Module Wise 22](#_Toc47819946)

[9 Data Prediction 23](#_Toc47819947)

[9.1 Technical solution design 26](#_Toc47819948)

[9.2 Exceptions Scenarios Module Wise 26](#_Toc47819949)

[10 Deployment Steps For Azure 27](#_Toc47819950)

[10.1 Exceptions Scenarios 30](#_Toc47819951)

[11 Dockerization 31](#_Toc47819952)

[11.1 Steps to Dockerize App 31](#_Toc47819953)

[12 Logging 35](#_Toc47819954)

[12.1 Technical solution design 35](#_Toc47819955)

[12.2 Common Logging Framework Code 35](#_Toc47819956)

[12.3 Exceptions Scenarios Module Wise 36](#_Toc47819957)

[13 Sample code and standard to be followed: 38](#_Toc47819958)

# 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 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**

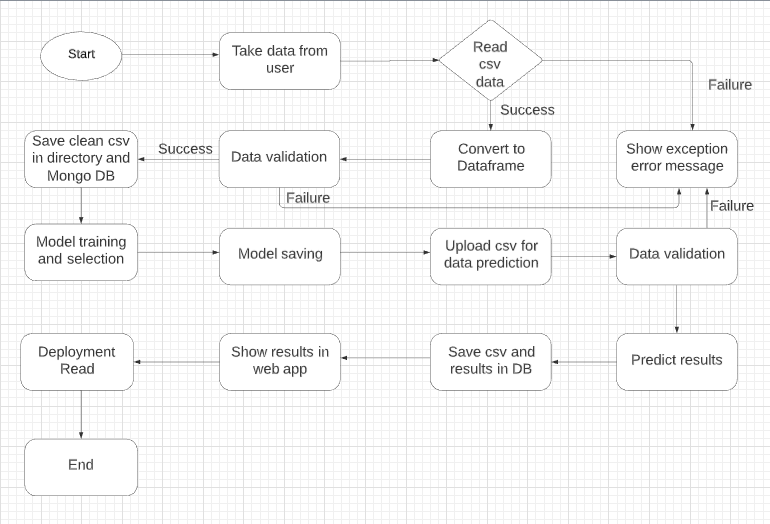
## High level objectives

The high-level objectives are:

1. Loadcsv 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 4models 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 |

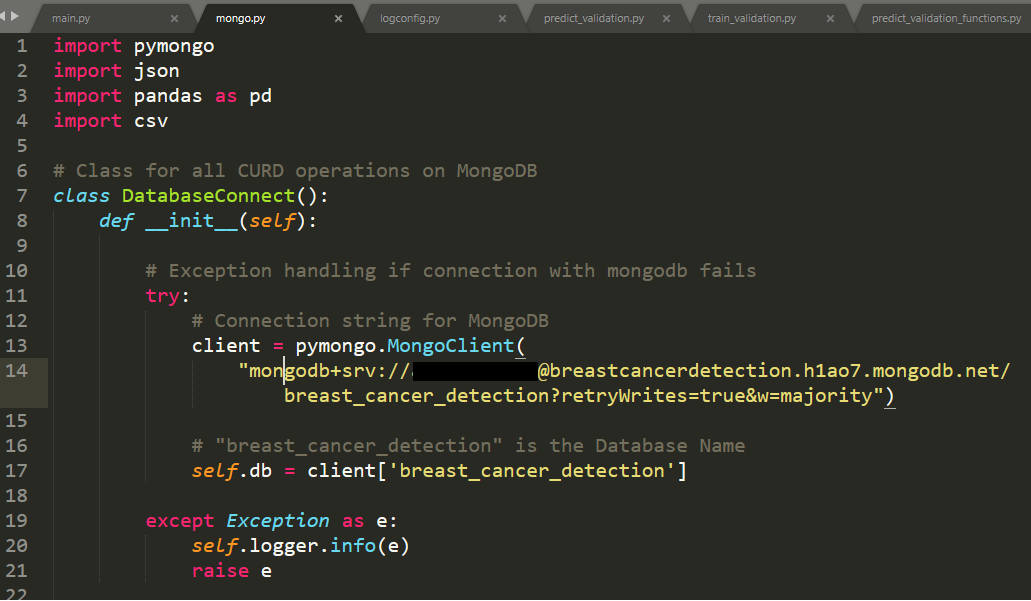
# Workflow Data Ingestion and File Conversion

**Data Sources:**

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

## Method Definitions

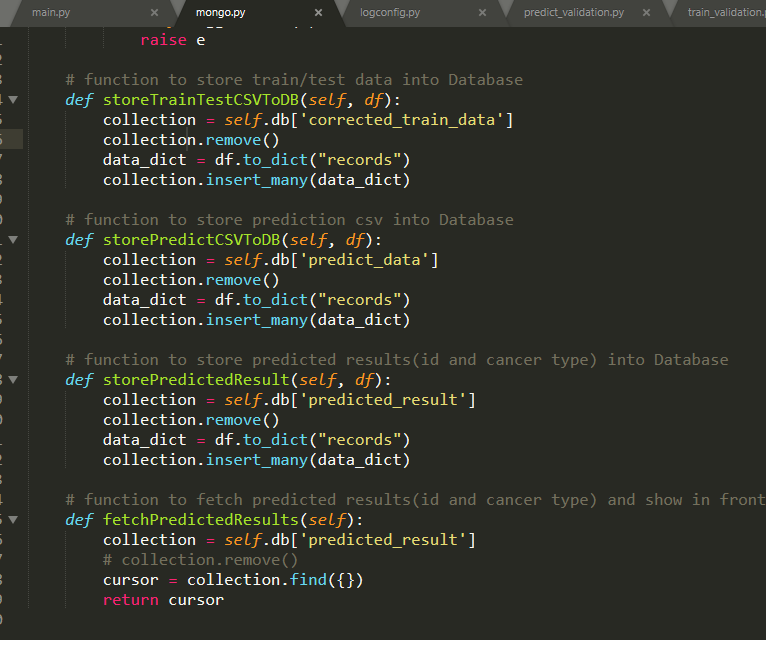
|  |  |  |
| --- | --- | --- |
| **Class Name** | **DatabaseConnect** | As soon as the object of this class is created, connection will be establish with mongo db. |
| Method Name | storeTrainTestCSVToDB |  |
|  | Method Description | This method will be used to to store final cleaned train/test csv 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 |



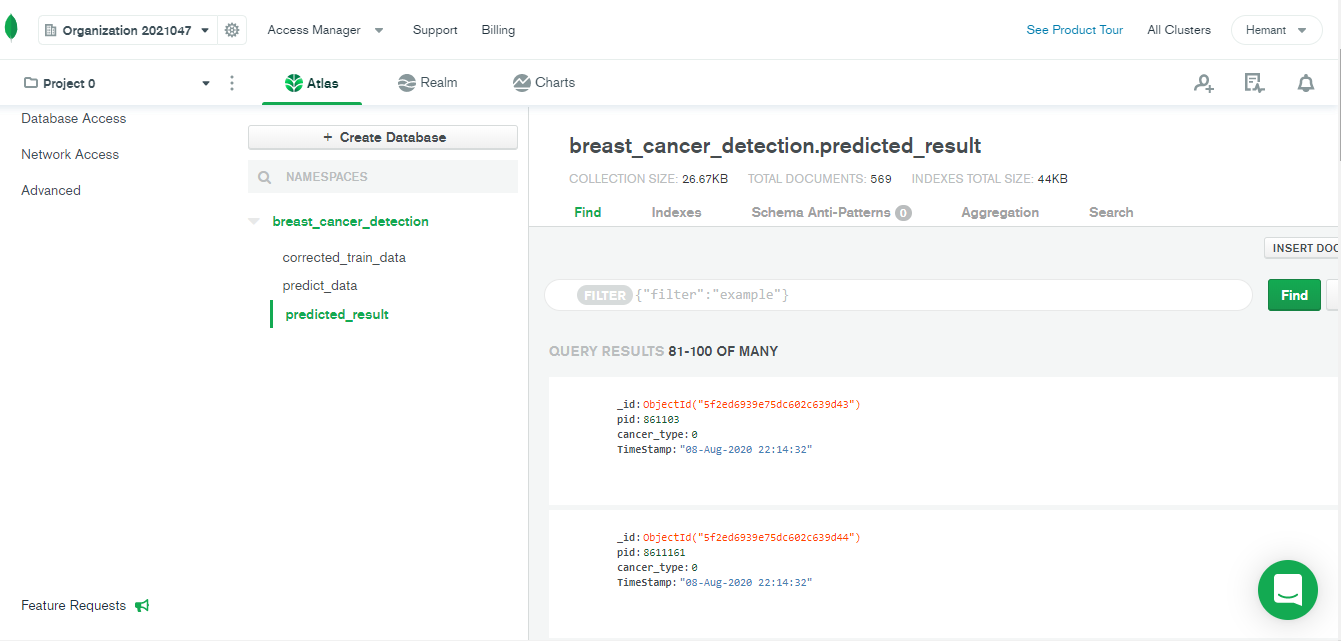
|  |  |  |
| --- | --- | --- |
| Method Name | storePredictCSVToDB |  |
|  | Method Description | This method will be used to to store predicted csv data into database after all data validation process. |
|  | Input parameter names | self, df |
|  | Input Parameter Description | Df: complete dataframe 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 prediction log file |

|  |  |  |
| --- | --- | --- |
| Method Name | storePredictedResult |  |
|  | Method Description | This method will be used to to store patient id and predicted result in DB. |
|  | Input parameter names | self, df |
|  | Input Parameter Description | Df: complete dataframe is getting passed in this method to store in DB |
|  | ouptput | Patient ID and predicted result should be stored in DB. |
|  | On Exception | Failing to store values will be captured in prediction log file |

|  |  |  |
| --- | --- | --- |
| Method Name | fetchPredictedResults |  |
|  | Method Description | This method will be used to fetch Patient ID and predicted result and show in front end. |
|  | Input parameter names | self |
|  | ouptput | Values will be fetched from DB. |
|  | On Exception | Failing to fetch values will be captured in prediction log file |



Mongo DB Screen shot –



## Exceptions Scenarios

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Unable to read or store data to 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 |
|  | output | 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

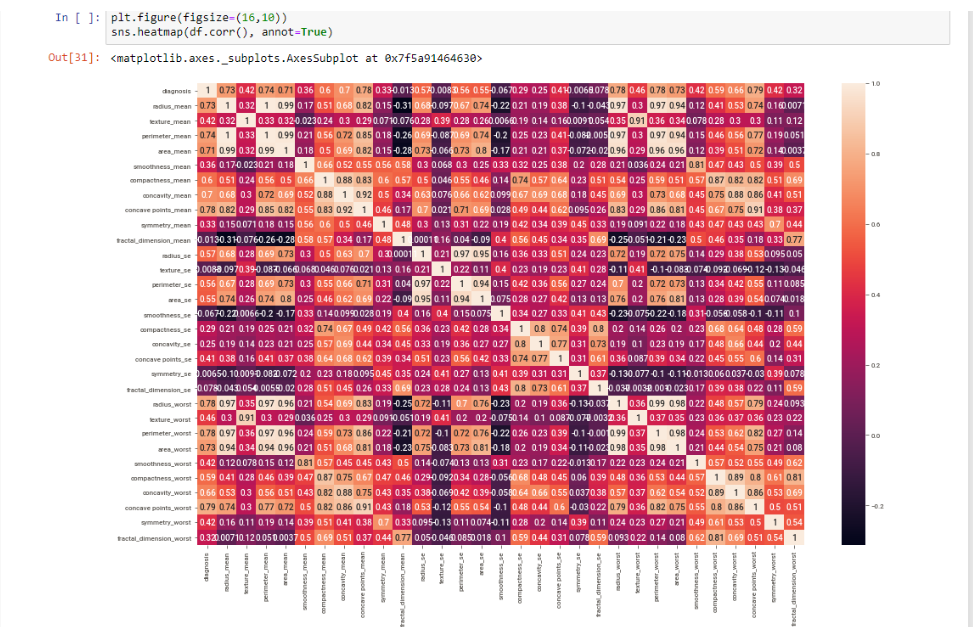
In [statistics](https://en.wikipedia.org/wiki/Statistics), **exploratory data analysis** (**EDA**) is an approach to [analyzing](https://en.wikipedia.org/wiki/Data_analysis) [data sets](https://en.wikipedia.org/wiki/Data_set) to summarize their main characteristics, often with visual methods. A [statistical model](https://en.wikipedia.org/wiki/Statistical_model) can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. Exploratory data analysis was promoted by [John Tukey](https://en.wikipedia.org/wiki/John_Tukey) to encourage statisticians to explore the data, and possibly formulate hypotheses that could lead to new data collection and experiments

EDA Reference Link –

<https://en.wikipedia.org/wiki/Exploratory_data_analysis>

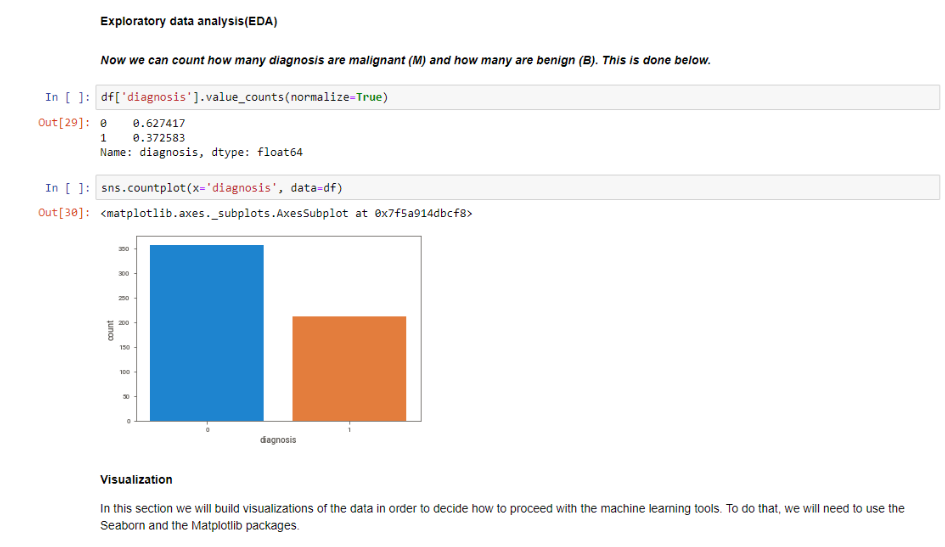
**Correlation Heatmaps:**

Looking at the Heatmap, we can immediately verify the presence of multicollinearity between some of our variables. For instance, the **radius\_mean** column has a correlation of 1 and 0.99 with **perimeter\_mean** and **area\_mean** columns, respectively. This is probably because the three columns essentially contain the same information, which is the physical size of the observation.



**Count plots:**

Out of the 569 observations, 357 (or 62.7%) have been labeled Benign, while the rest 212 (or 37.3%) have been labeled Malignant.

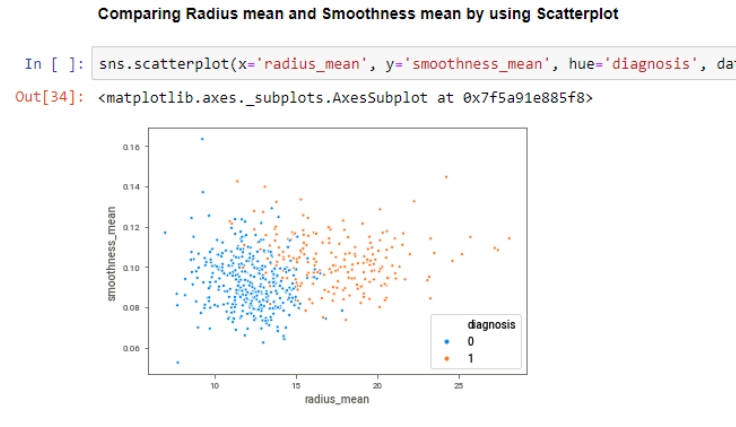


**Pairplots:**

There are some interesting patterns visible. For instance, the almost perfectly linear patterns between the **radius**, **perimeter** and **area** attributes are hinting at the presence of multicollinearity between these variables.

****

**Scatterplot:**

****

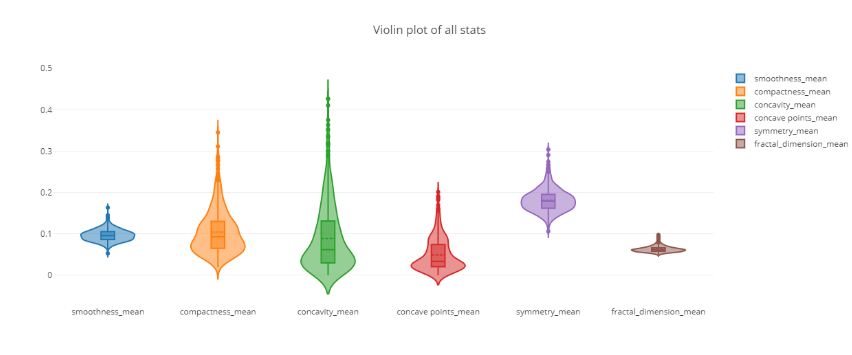
**Donut Chart:**

****

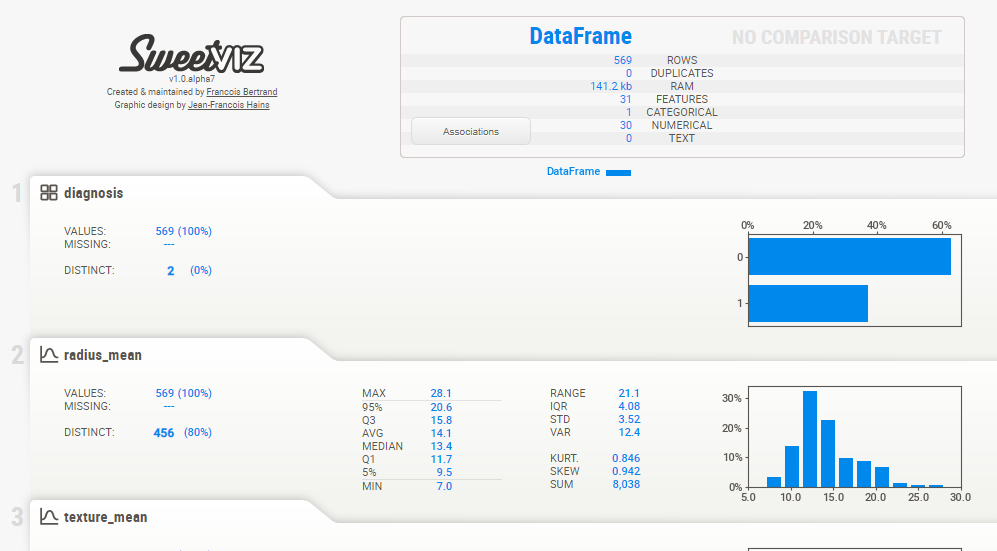
**Violin Chart:**

Just like boxplot by itself illustrates the interquartile spread of the distribution; its length determined by the 25%(Q1) and 75%(Q3) percentiles. The vertical line inside the box marks the median ( 50% ) of the distribution. **A violin plot is a method of plotting numeric data. It is similar to box plot with a rotated kernel density plot on each side**

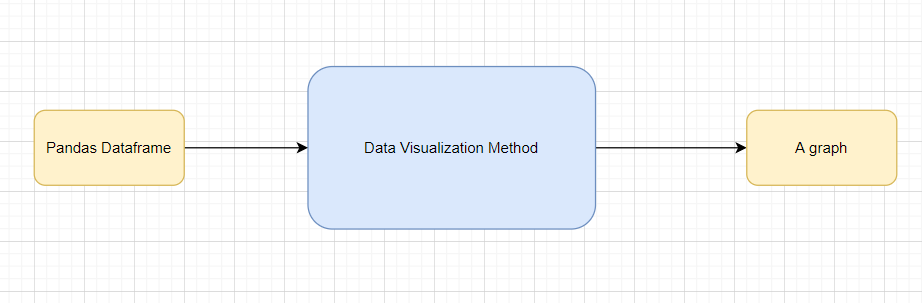
The whiskers are the lines extending from the box. They represent the entire scatter of data points, specifically the points that fall within the interval (Q1−1.5⋅IQR,Q3+1.5⋅IQR) , where IQR=Q3−Q1 is the interquartile range.

****

**EDA using sweetviz**: 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.



## Technical solution design



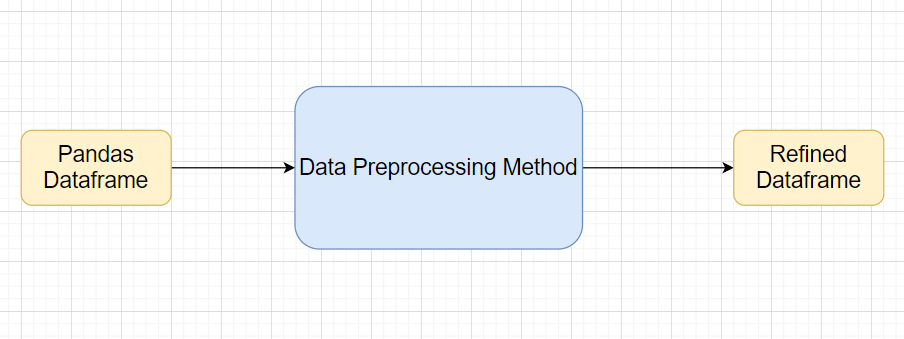
# Data Transformers( Pre-processing steps)

**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 |
|  | output | A pandas Dataframe |
|  | On Exception | Write the exception in the log file.  Raise an exception with the appropriate error message |

## Exceptions Scenarios Module Wise

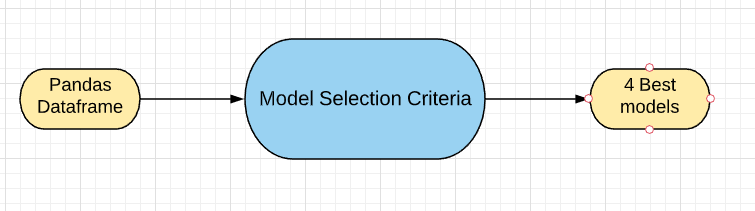
|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Wrong parameters passed to the methods | Handle Internally | Code should never give a wrong input |

# 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.

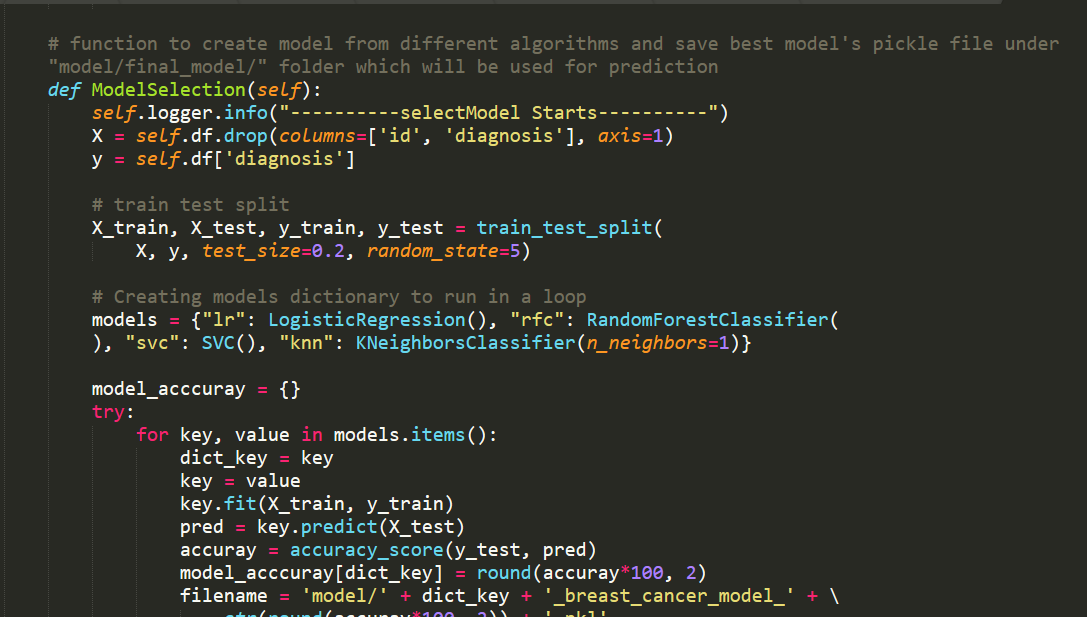
## Technical solution design

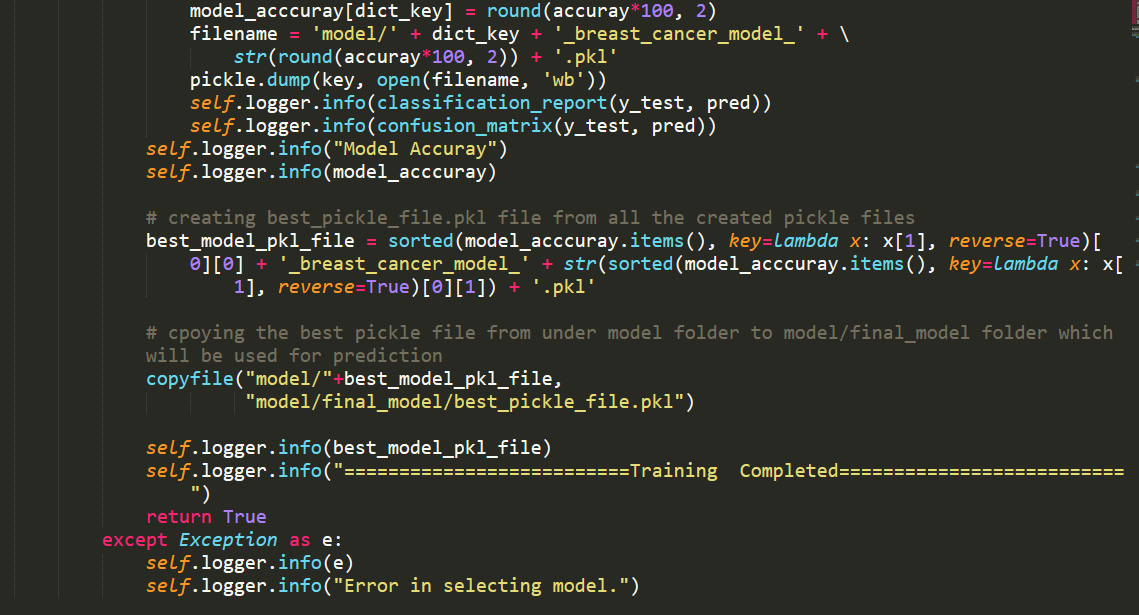


## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **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

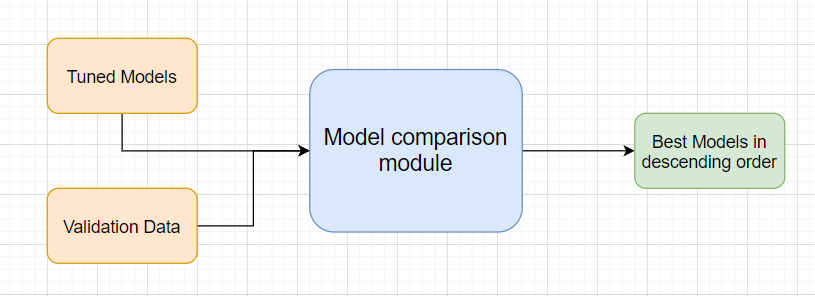
Random Forest

SVM

KNN Classifier

Model selection criteria:

Accuracy, AUC, Precision, Recall



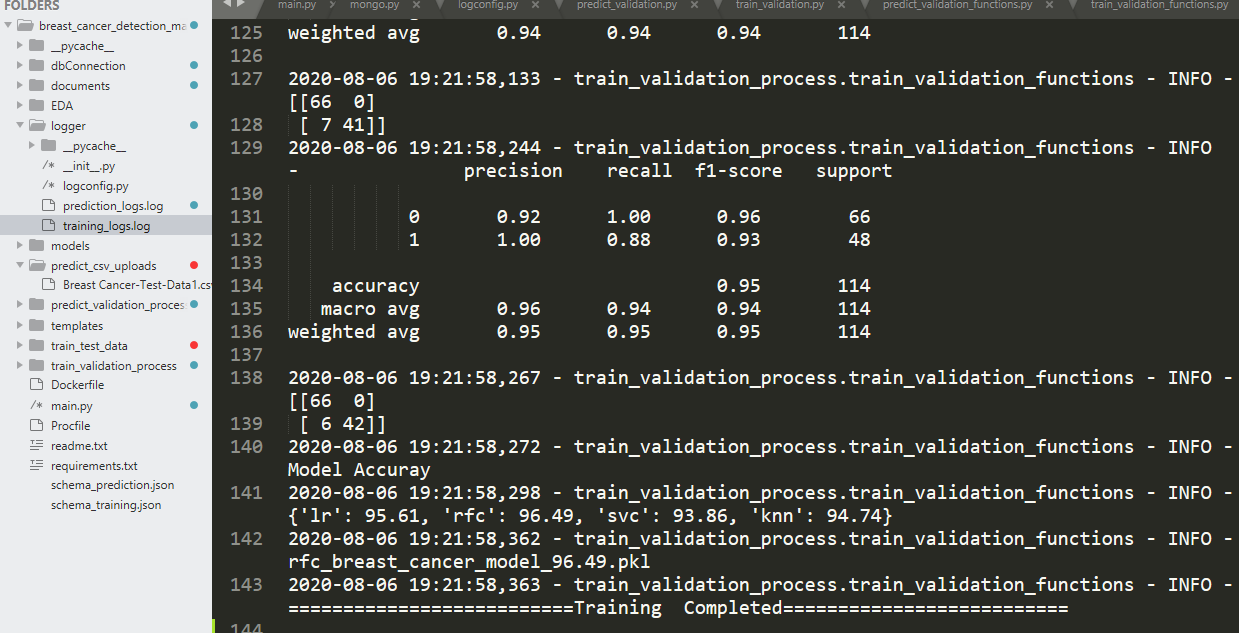
# Testing Models

1. Divide the training csv data itself into train and test sets
2. Use train data to fit into the 4 different models(***Logistic Regression*, *Random Forest*, *SVC, KNN***)
3. Use test data to check the models performance
4. Choose best model based on accuracy and create pickle file of that and put that file inside “models/final\_model” directory.

Give the test report in training logger file -

1. Accuracy
2. Precision
3. Recall
4. F1-score
5. Confusion matrix

Model training logs screenshot -



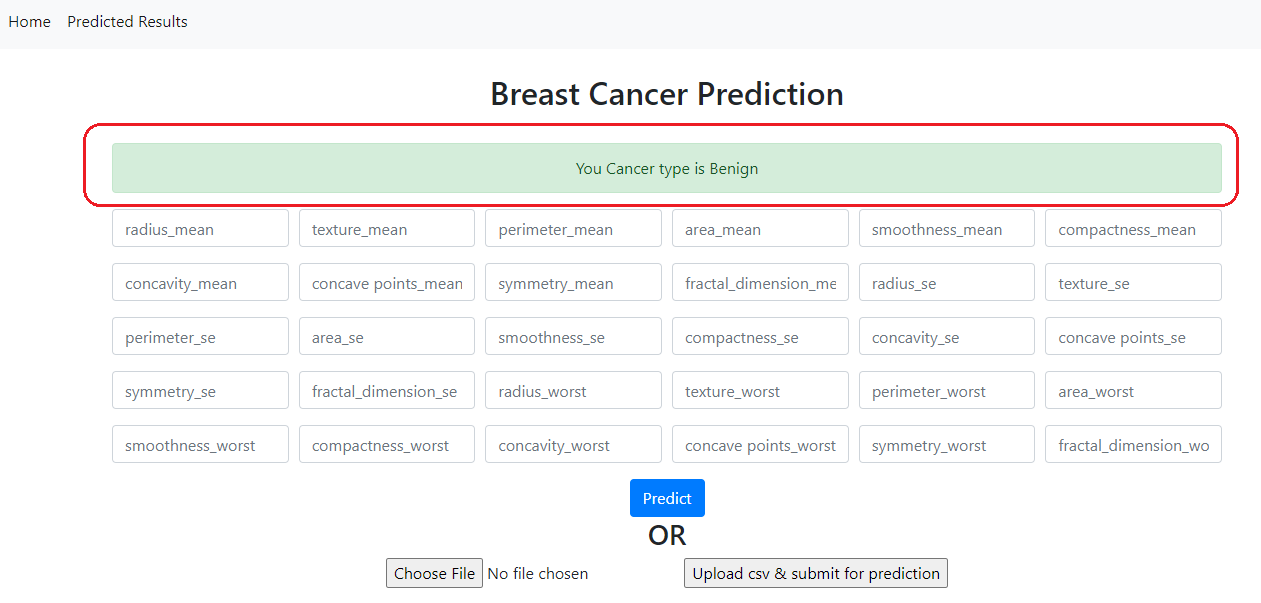
## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Exception** | **Mitigation** |
| Number of columns in train data and schema\_training.json does not match | Handle internally | Check the column numbers in train data csv |
| Column names in train data and schema\_training.json does not match | Handle Internally | Check the column names in train data csv |
| Column types in train data and schema\_training.json does not match | Handle Internally | Check the column types in train data csv |

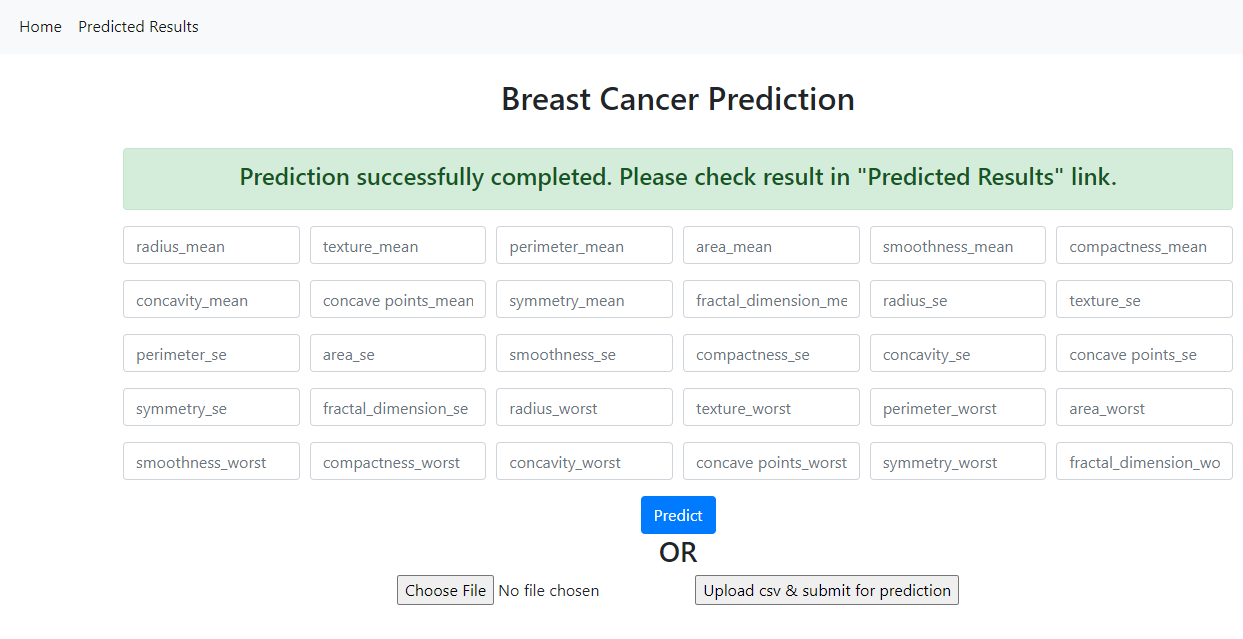
# Data Prediction

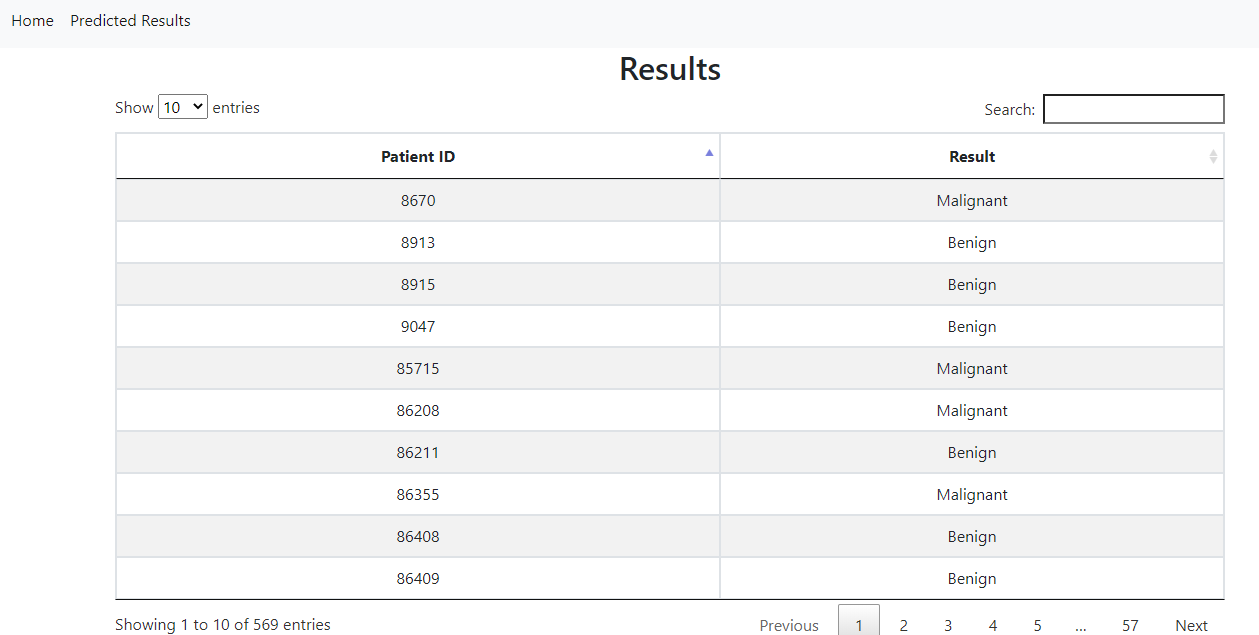
After selecting the best model, next step is to predict the data. Data can be predicted in three ways -

1. Enter individual values of patient in UI and click on Predict button, Cancer type will be shown directly on the screen.



1. Upload csv file and click on “Upload csv& submit for prediction”. Make sure csv file matches with schema\_prediction.json file. Predicted data will be store in DB and can be seen by clicking on link “Predicted Result” from top menu.





1. Send Json request with all the data from postman/insomania and it should result the cancer type.

Json format for sending post request –

Url for sending post data is:

To test in local : Your local server url <http://127.0.0.1:5000/>

To test in Azure: <http://breastcancermodeldetection.azurewebsites.net/>

Note: Use the same key names only change the values.

{

"radius\_mean" : 1,

"texture\_mean" : 1,

"perimeter\_mean" : 1,

"area\_mean" : 1,

"smoothness\_mean" : 1,

"compactness\_mean" : 1,

"concavity\_mean" : 1,

"concave\_points\_mean" : 1,

"symmetry\_mean" : 1,

"fractal\_dimension\_mean" : 1,

"radius\_se" : 1,

"texture\_se" : 1,

"perimeter\_se" : 1,

"area\_se" : 1,

"smoothness\_se" : 1,

"compactness\_se" : 1,

"concavity\_se" : 1,

"concave\_points\_se" : 1,

"symmetry\_se" : 1,

"fractal\_dimension\_se" : 1,

"radius\_worst" : 1,

"texture\_worst" : 1,

"perimeter\_worst" : 1,

"area\_worst" : 1,

"smoothness\_worst" : 1,

"compactness\_worst" : 1,

"concavity\_worst" : 1,

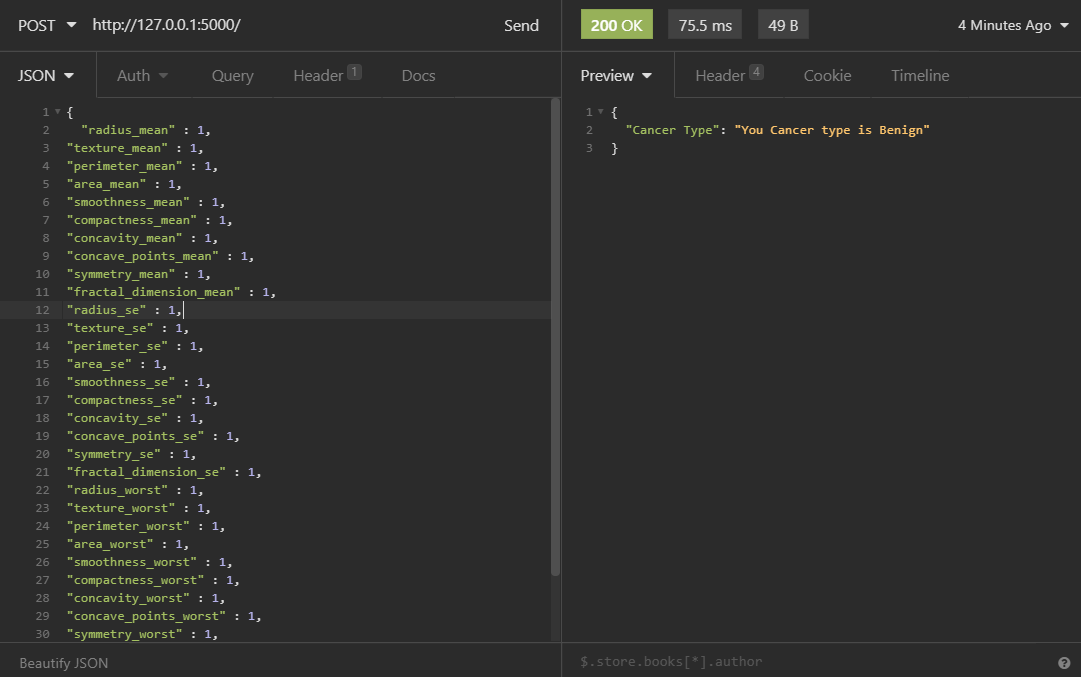
"concave\_points\_worst" : 1,

"symmetry\_worst" : 1,

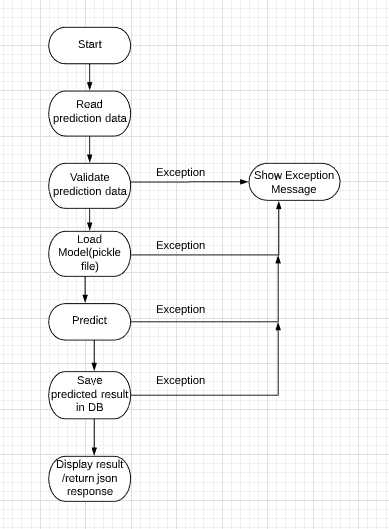
"fractal\_dimension\_worst" : 1

}

Insomania screenshot –



## Technical solution design



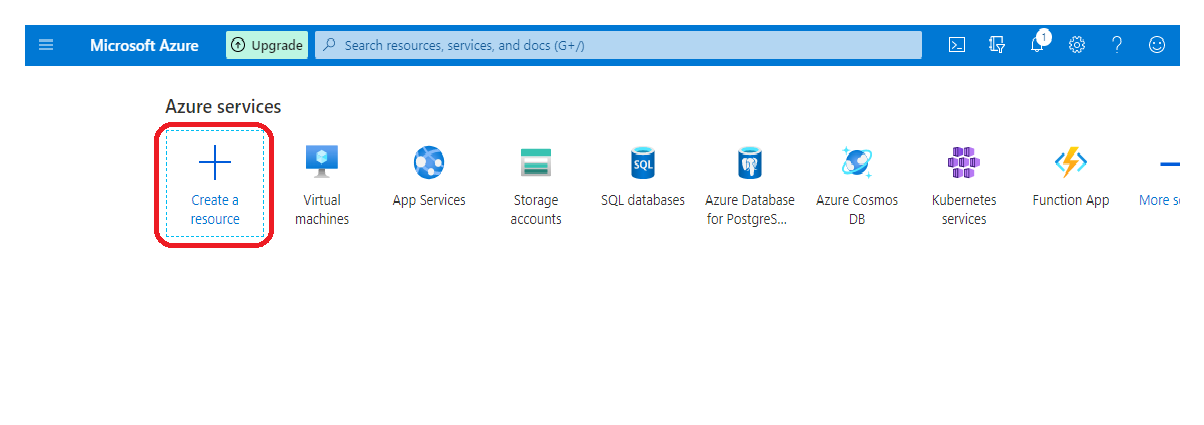
## Exceptions Scenarios Module Wise

|  |  |  |
| --- | --- | --- |
| **Step** | **Type of prediction** | **Exception** |
| User try to enter wrong input values from UI | Single record prediction from UI | Show exception message on UI |
| User try to provide wrong key name and values in json request. | Single record prediction from Json | Show exception message in json response |
| User try to upload csv which doesn’t match with prediction schema.json file | Csv upload data prediction | Show exception message on UI |

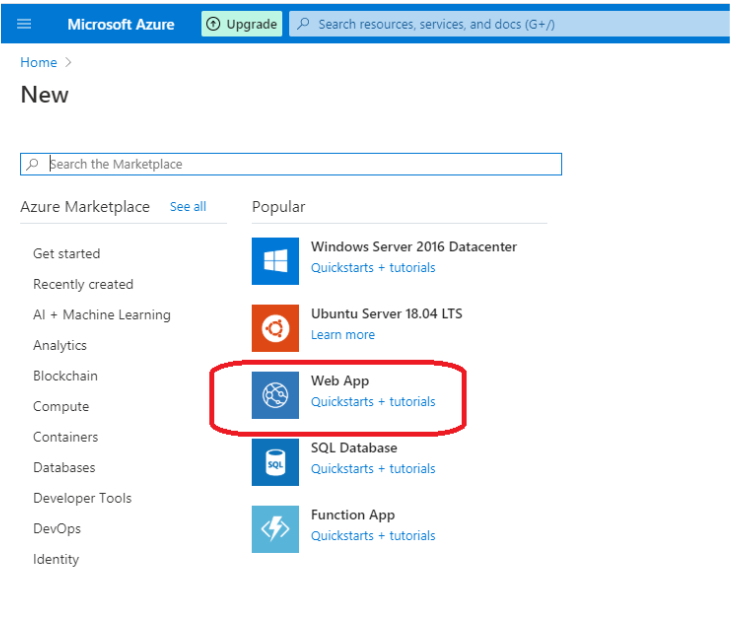
# Deployment Steps For Azure

This project will be deployed on **Azure** by connecting github repository. Steps to deploy in Azure are –

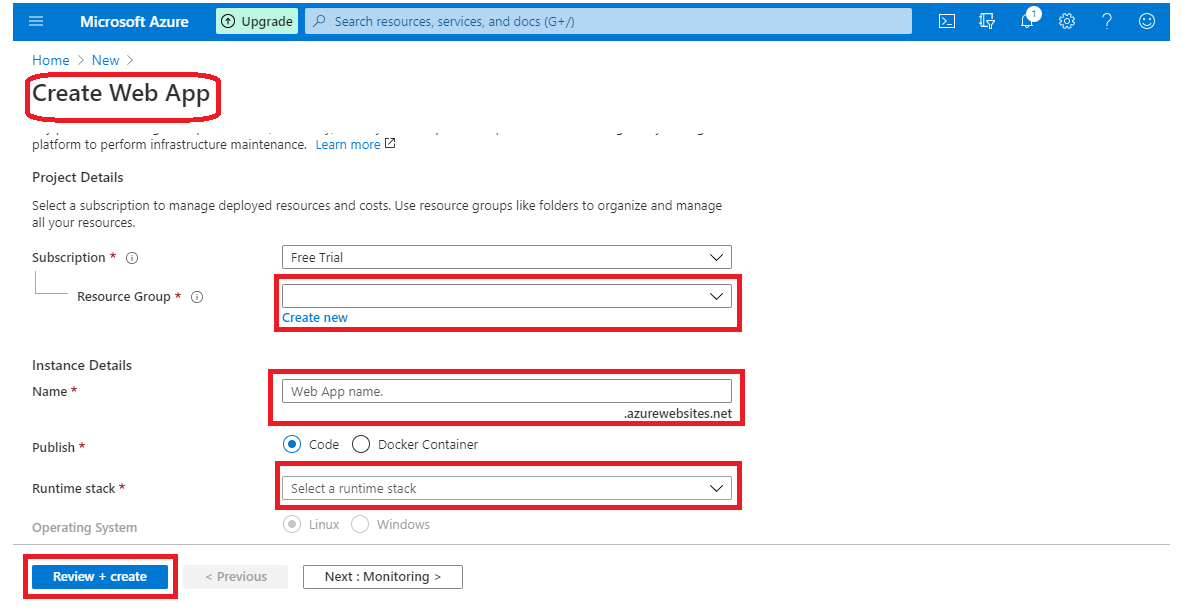
1. Create account in portal.azure.com
2. Click on Create a resource–



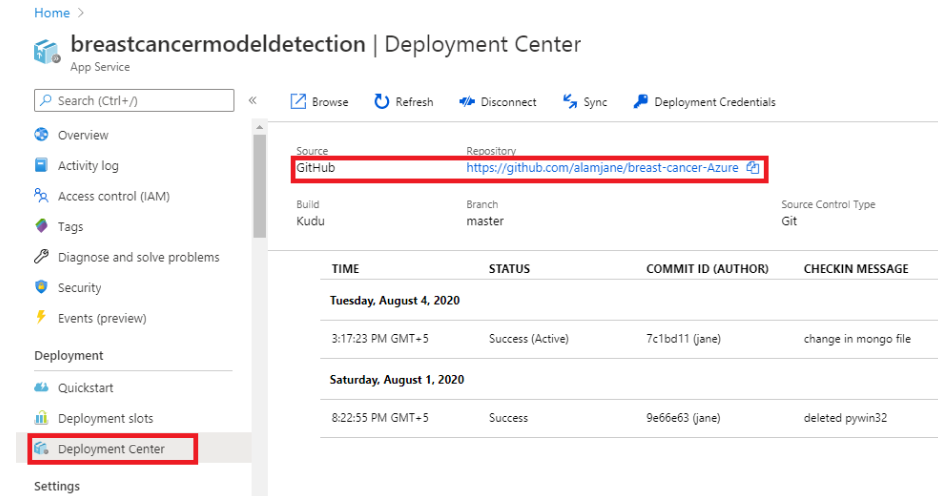
1. Search Web App and click on it –



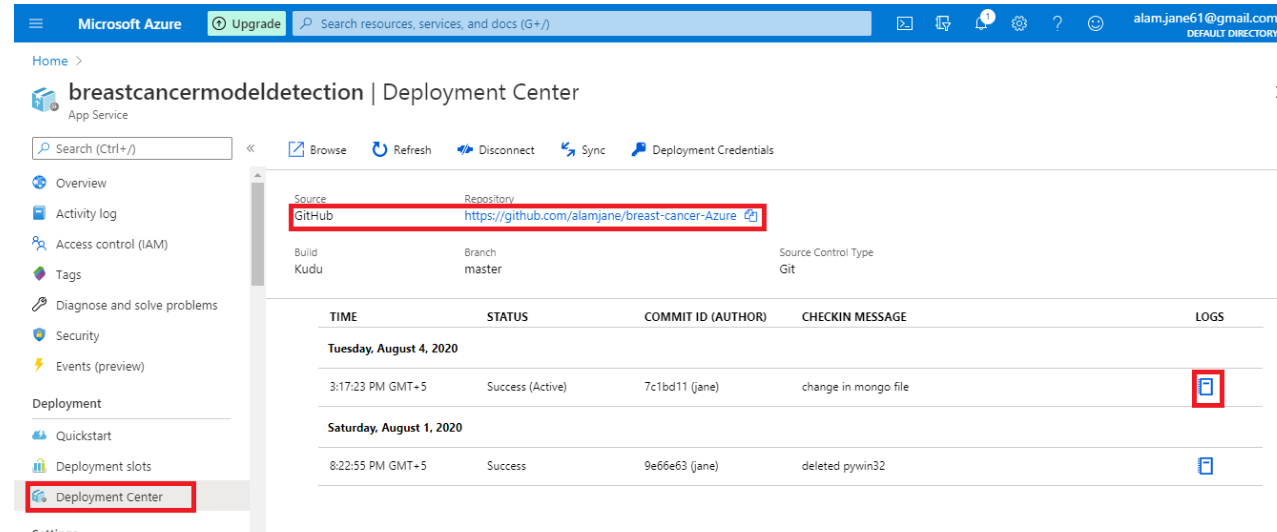
1. Add details to create web app for the project. Create a new resource group if its not created and add that and choose Instance Details name(which is going to be be url of the web app). Click Review+Create after filling all information–

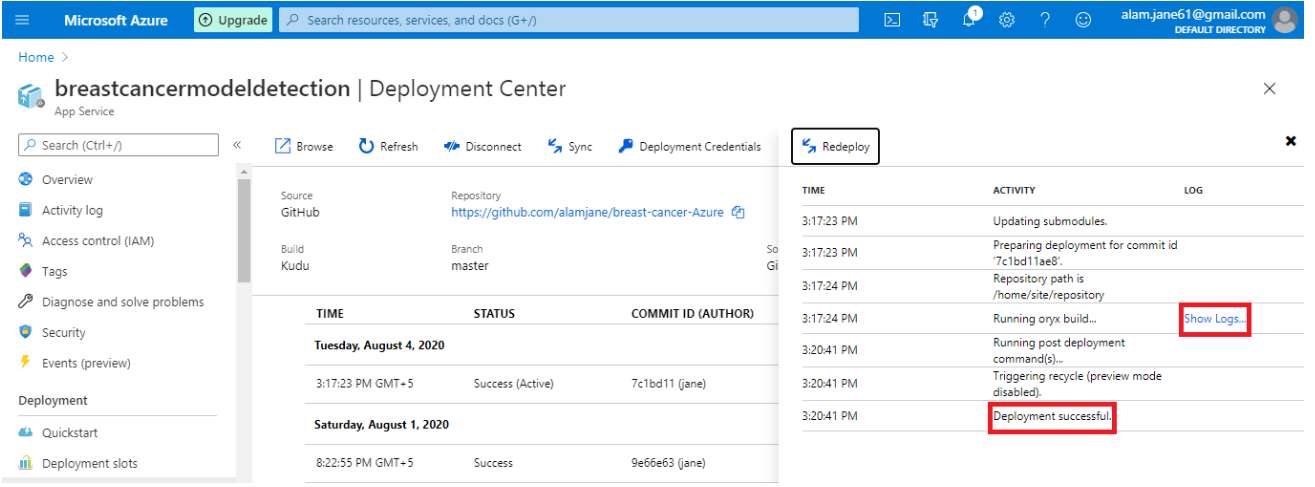


1. Next step is to add git hub url in deployment center.

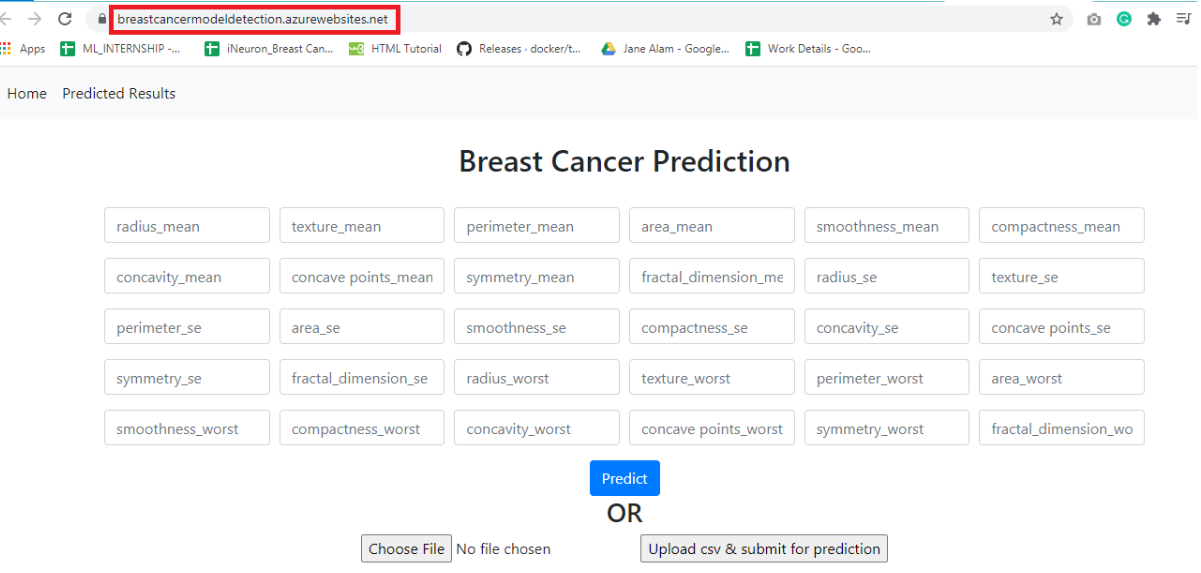


1. To check the deployment logs click on the red highlighted icon under LOGS -





1. If no error shows inside logs, go to the dashboard and open the site url –



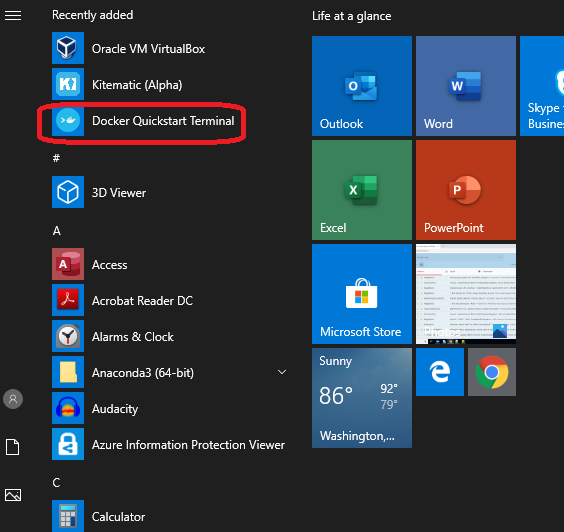
## Exceptions Scenarios

|  |  |
| --- | --- |
| **Step** | **Exception** |
| Missing packages in Requirements.txt | Any package is not included used in project. |

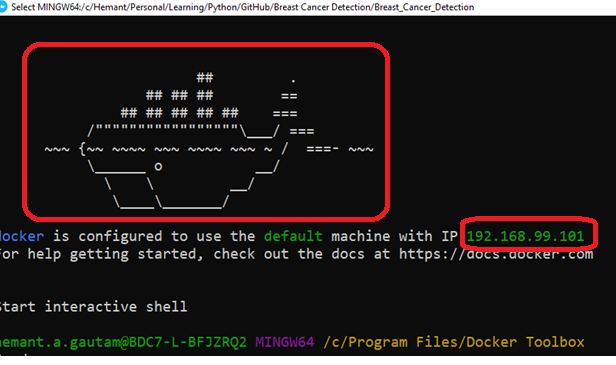
# Dockerization

## Steps to Dockerize App

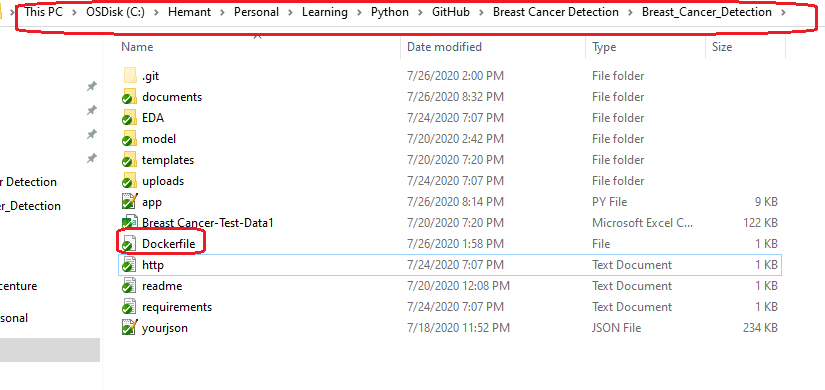
1. Download and run docker toolkit exe file from <https://github.com/docker/toolbox/releases> and if the downloaded version doesn’t support in your machine, try with other lower or higher versions of toolkit.
2. After successful installation, go to start and click on Docker QuickStart Terminal. Refer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

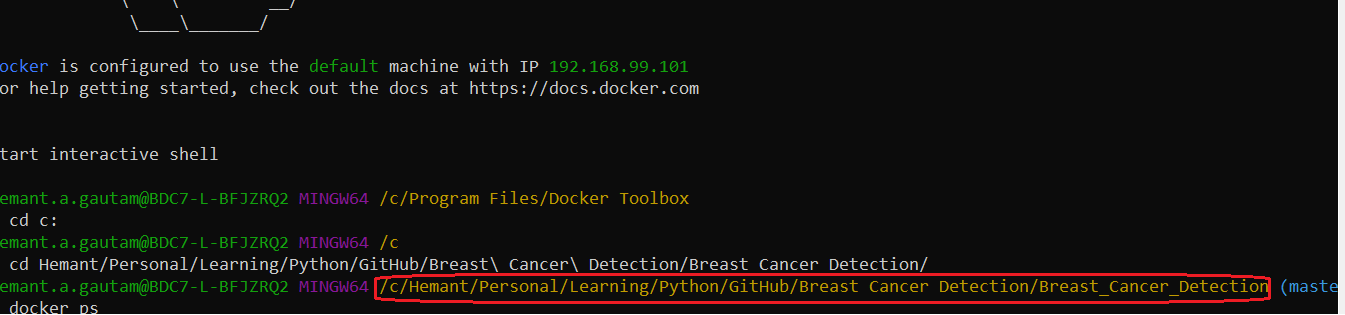


1. This will open a docker command prompt and will take some time to load properly. There will be Docker image at first and machine IP which will be used to access flask app. Refer image (highlighted in red)

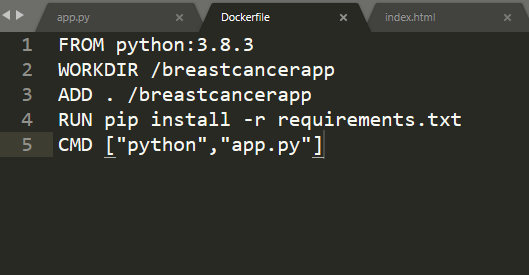


1. Next step is to navigate to the project root folderin docker cmd where Dockerfile is present. Refer image (highlighted in red)

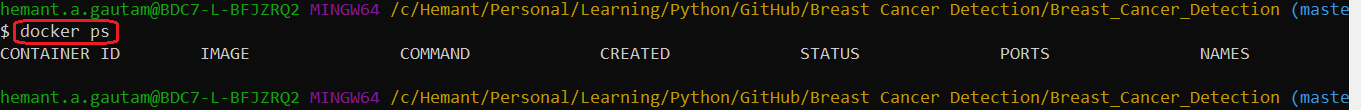




1. Creating docker file is important to dockerize flask app. And to create Dockerfile file, these are the commands should be mentioned. Refer image (highlighted in red



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:

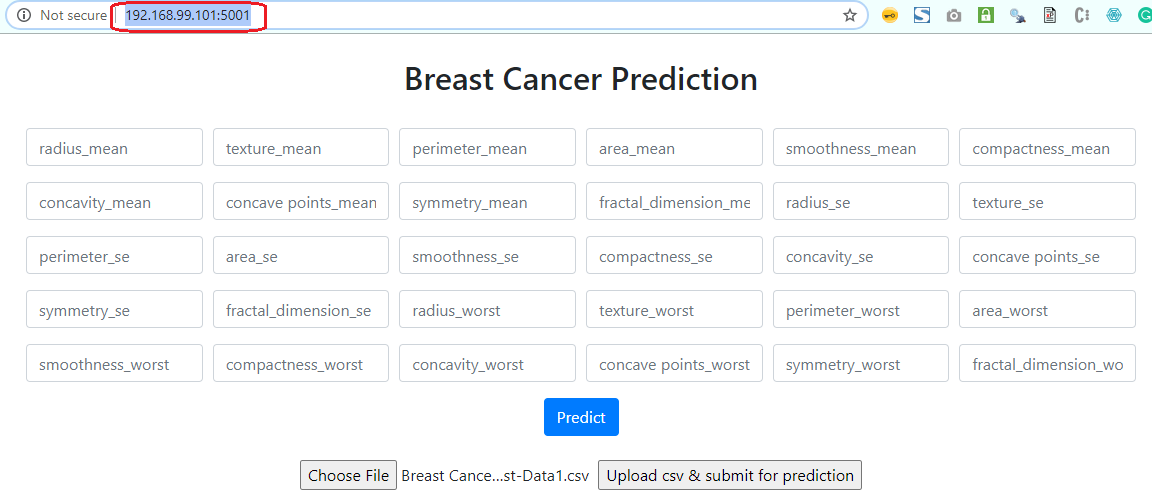
docker image ls

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

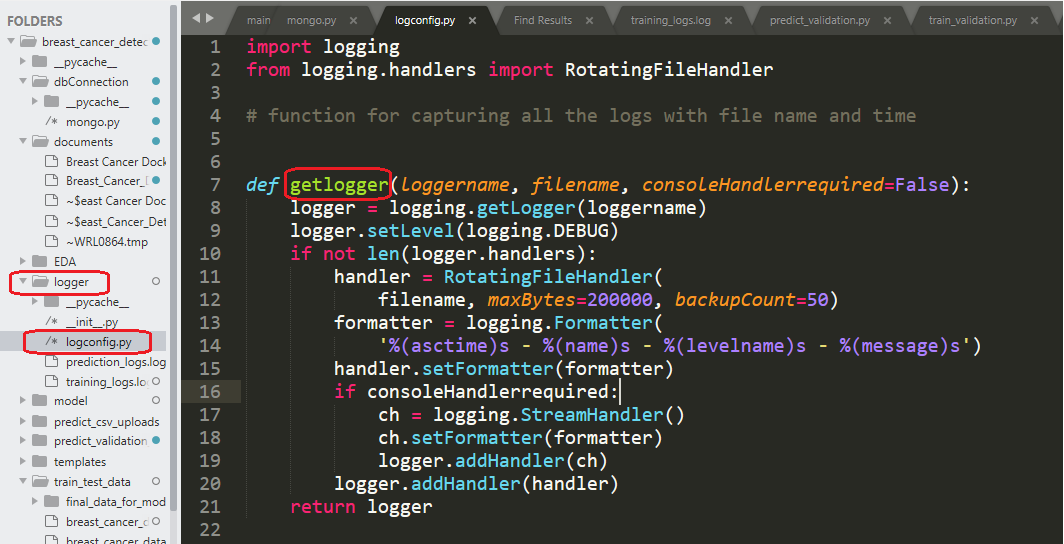
* Separate folder for prediction and trainings logs
* Logging of every step with timestamp
* Entry to the methods
* Exit from the methods with success/ failure message
* Error message Logging
* Model comparisons
* Training start and end
* Prediction start and end

## Technical solution design



## Common Logging Framework Code

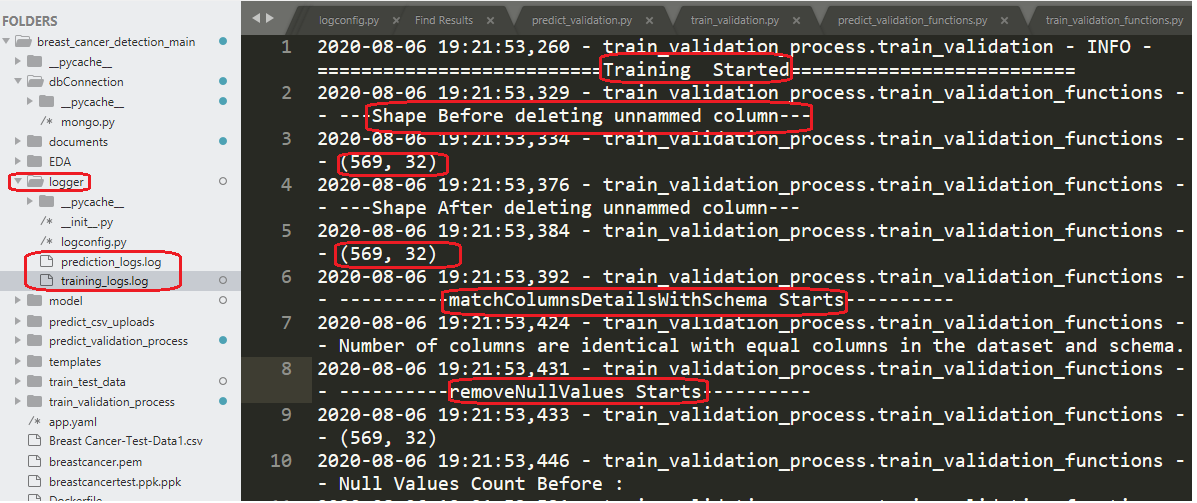
|  |  |
| --- | --- |
| Function name | getlogger |
| Method Description | This function will be used for logging all the prediction and trainings information in separate files. |
| Input parameter names | loggername, filename, consoleHandlerrequired=False |
| Input Parameter Description | loggername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

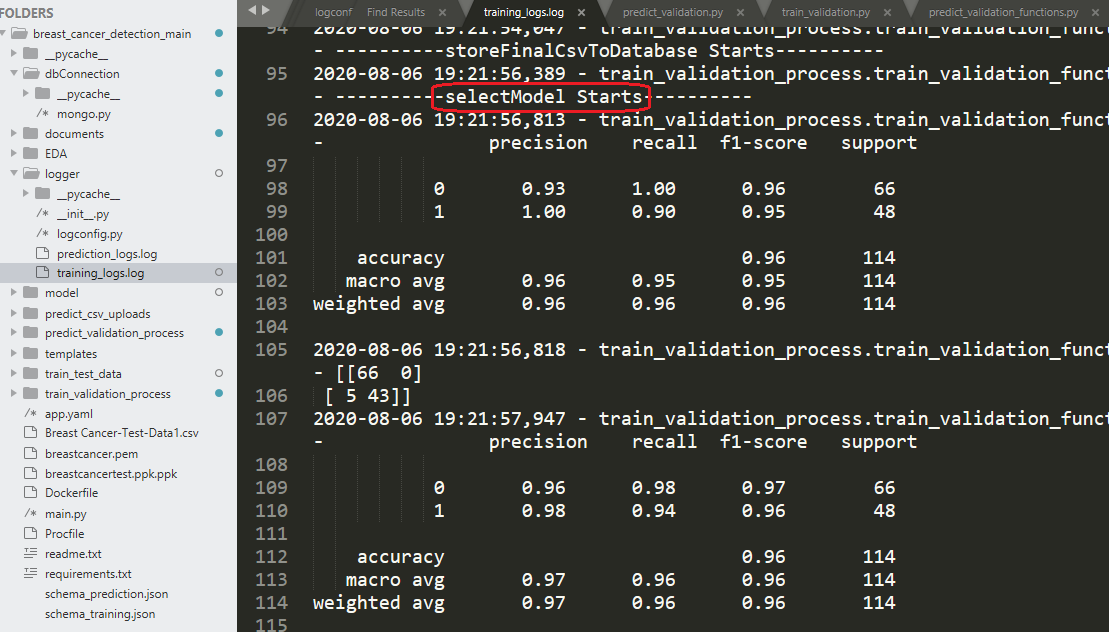


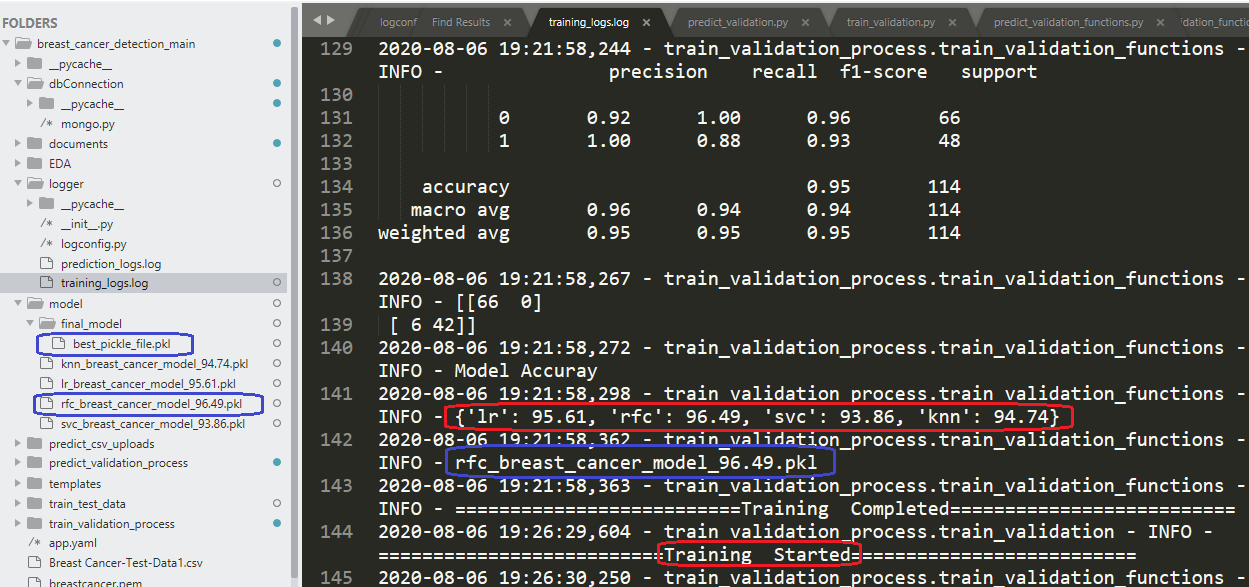
## Exceptions Scenarios Module Wise

Ideally, the logging should never fail.

Training logs screen shot -

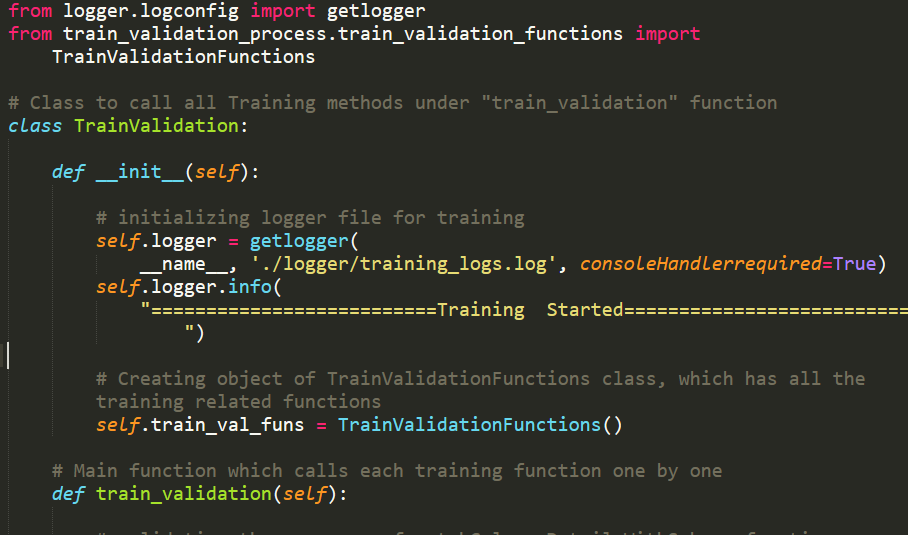


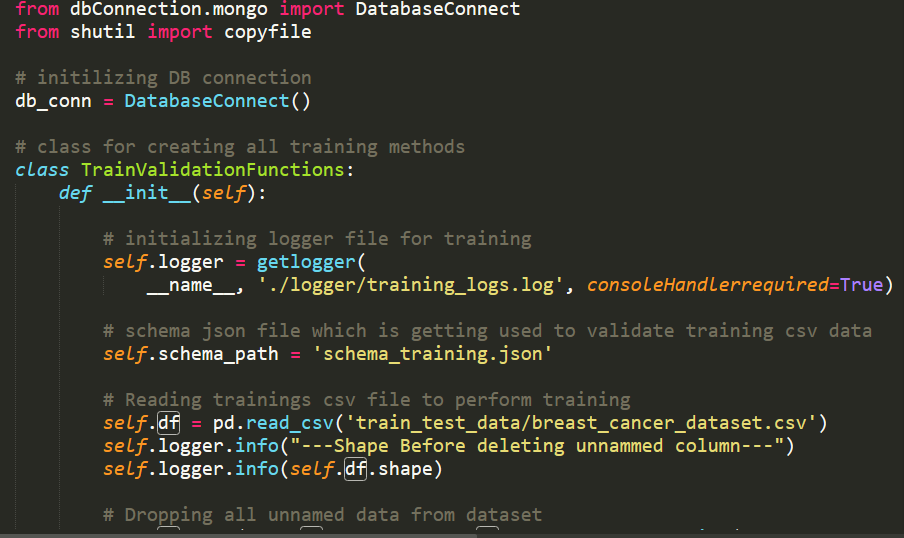




# Sample code and standard to be followed:

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 ==