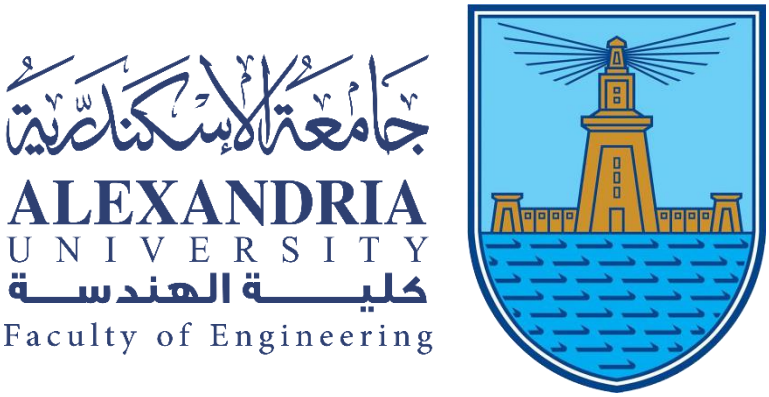


Faculty of Engineering

Computer and Communications Program

Doctor's Assistant

Through Machine Learning

Supervised by:

Prof.Dr.Adel Elfahar

Presented by:

Esraa Mohamed Ibrahim	5630	Mohamed Mahmoud Mousa	5375
Heba Tarek Mahdi	5923	Mostafa Ibrahim Ragab	5567
Khlod Mohamed Ali	5695	Salma Hamdi Saad	5706

Acknowledgments

To begin with, we would like to thank God for the support given throughout our graduation project. We'd also like to thank our spectacular

supervisor, Dr.Adel for his endless support, constant follow up, and academic help given throughout our journey in working on our graduation project. Finally, we would like to thank our families for their constant love, support, and encouragement throughout the time we spent working too. Due to you all we were able to reach our goal of that interesting journey.

Thank you.

Abstract

In this project, we have developed a frontend using Streamlit for the user interface, and a backend using FastAPI for the artificial programming interface which contains the machine learning models used for predicting diagnosis, and then deployed both ends, connectedly, through Heroku. We first built our machine learning models by using the approach of gathering data, preparing data, wrangling data, analysing data, training model and testing model. The diseases that we have built machine learning models for were: Diabetes, Breast Cancer, Lung Cancer, Kidney disease, heart disease, and Erythemato-Squamous disease. We then constructed the frontend to consist of two options the user could make from those diseases, the first is getting a diagnosis prediction for the disease by inserting specific feature inputs, and the second is to obtain more information about the machine learning model used for the diagnosis prediction of the disease. The frontend communicates with the backend by sending requests with the parameters of the user's chosen disease and the user-inserted features, then receiving responses from the API of the backend which contains the prediction class of the disease and its probability. We finally produced docker files to containerize the frontend and backend, and a docker-compose file to connect them for deployment purposes.

Contents

1. Introduction

The investment of machine learning in the medical field and healthcare has been undoubtedly increasing, proving its tremendous success. Learning from training data allows machine learning to find patterns, build models, and make predictions. Because of their custom-designed representation and input features, most machine learning models function effectively. Using the input data generated through that process, machine learning learns algorithms, optimises the weights of each feature, and optimises the final prediction.

In practice, a patient's symptoms, a physician's physical examination, laboratory test results, and imaging investigations are usually required to assess a patient's condition and diagnose a disease. However, for the diagnosis of specific diseases, the predictive power and accuracy that may be reached using machine learning data alone is enormous. As a result, the goal was to create predictive models that physicians might use to make judgments in the hospital setting based on machine learning data, and then to confirm the decision by comparing its predictions to physician diagnoses. Furthermore, early disease diagnosis could be aided, and as a result, machine learning could truly save lives!

2. Prototype

2.1. Definition

A simulation or sample version of a final product, which used for testing before launch.

The goal of the prototype is to test and validate ideas before sharing them, and to make sure that all team members are on the same page

2.2. Reasons to Make Prototypes

- To streamline the design development process, focusing on important interface elements.
- To identify unnecessary elements that are best abandoned.
- Easy to undo with low cost.
- Simulate the real product
- Quick and easy to create

2.3. Prototype Types

There are four types of prototypes as shown in Fig.2.3.1

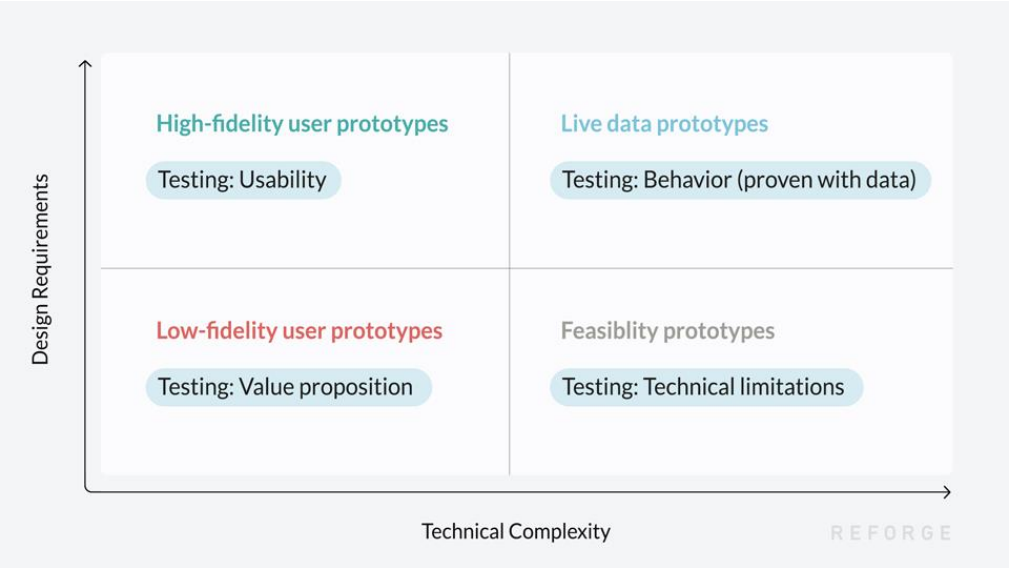


Fig.2.3.1: Prototype types

- Feasibility Prototypes
  - For prototyping new technology (ex. updated algorithm).
  - Engineer writes just enough code to see if it's feasible.
  - Helps understand technical risk, often related to performance.
- Low-Fidelity User Prototypes
  - Essentially an interactive wireframe (doesn't look real).
  - Created by interactive designers to test the workflow.
  - Simulates process to identify usability issues early.
- High-Fidelity User Prototypes
  - Realistic looking, working simulation.
  - Good for communicating a proposed product to stakeholders.
  - Used in defensive user testing, not to see if they'll like it, but to learn if they won't.

- Live-Data Prototypes
  - Very limited implementation created by developers to actually prove it works.
  - Has access to real data and is sent real live traffic.
  - Hasn't been "productized" (no test automation, SEO, localization, etc)

2.4. Our Prototype

We used low fidelity prototype, built it using **EXCALIDRAW** *"a virtual whiteboard for sketching hand-drawn like diagrams"*

It's a simple prototype, constructed from 4 basic screens, as shown in Fig.2.4.1 to Fig.2.4.4

The first screen is used for selecting the disease to predict, then the user will insert the required data in screen 2, prediction will be available in screen 3, and to get more information about our model see screen 4.

Fig.2.4.1: Screen 1

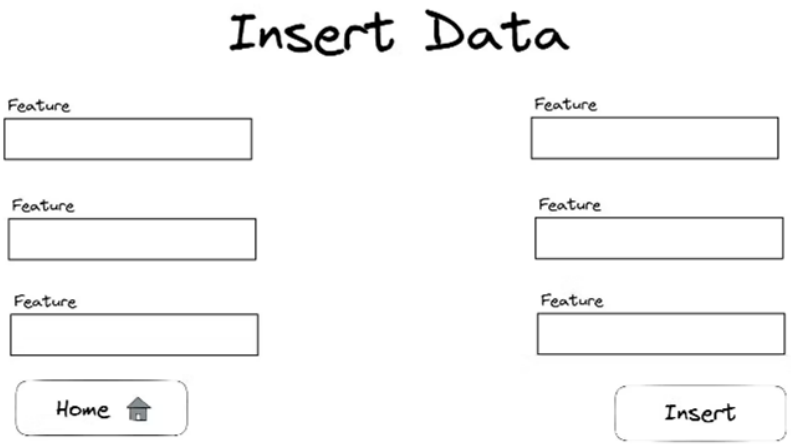


Fig.2.4.2: Screen 2

Fig.2.4.3: Screen 3

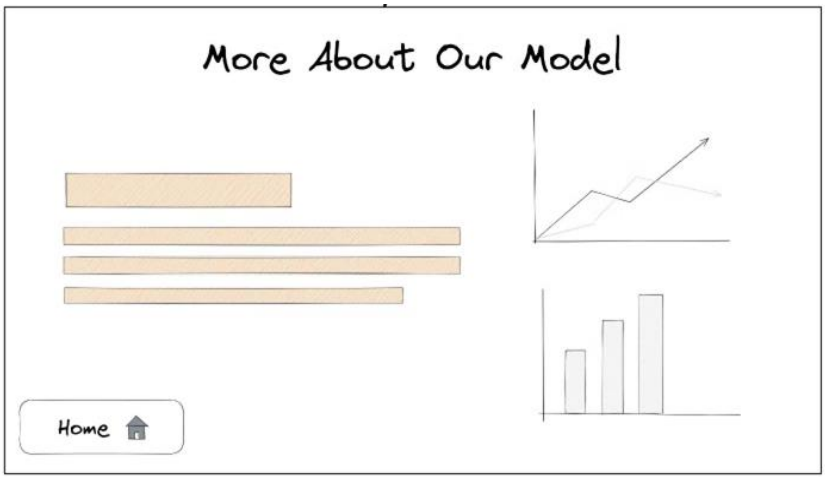


Fig.2.4.4: Screen 4

3. Architecture and Technology Stack

The whole functionality is divided into a frontend and a backend part, both will be containerized with Docker. The containers will be able to communicate via an API: the frontend sends requests to the backend container and receives predictions made by the model in the backend container.

For the frontend we used Streamlit to get a visually appealing example without the need to write HTML and CSS, for the backend we used the FastAPI package. Both parts will be containerized using Docker. Another tool, Docker-compose, used to run both containers on the same network, so they can communicate despite being isolated environments. The following figure visualize our architecture clearly.

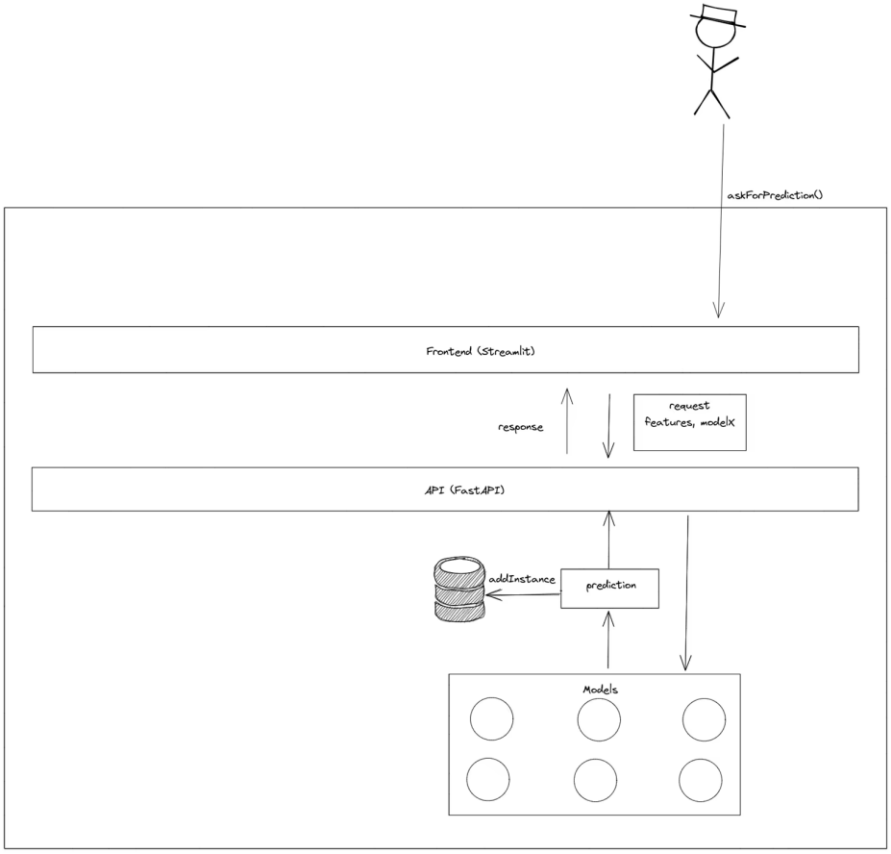


Fig.3.1: Doctor's Assistant Architecture

4. Machine Learning Life-Cycle

Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project.

Machine learning life cycle involves seven major steps, which are shown in Fig.4.1

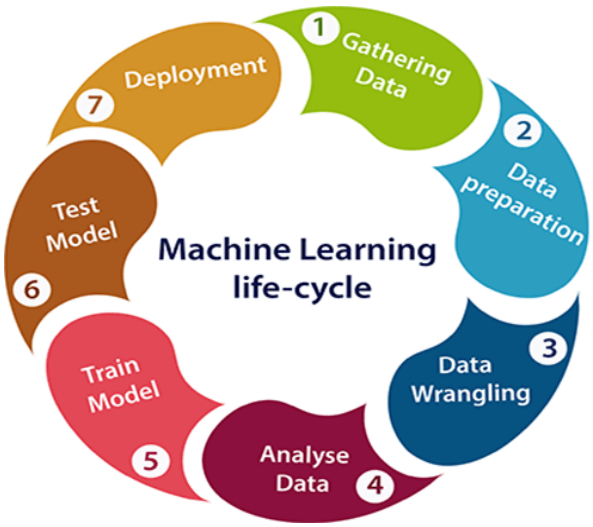


Fig.4.1: ML Life-Cycle

4.1. Gathering Data

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems. In this step, we need to identify the different data sources, as data can be collected from various sources such as files, database, internet, or mobile devices. It is one of the most important steps of the life cycle. The quantity and quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.

This step includes the below tasks:

- Identify various data sources
- Collect data
- Integrate the data obtained from different source.

We used UCI Machine Learning Repository as a main data source for our models.

4.2. Data preparation

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data. This step can be further divided into two processes:

- **Data exploration:** It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data. A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers. As shown in Fig.4.2.1
-

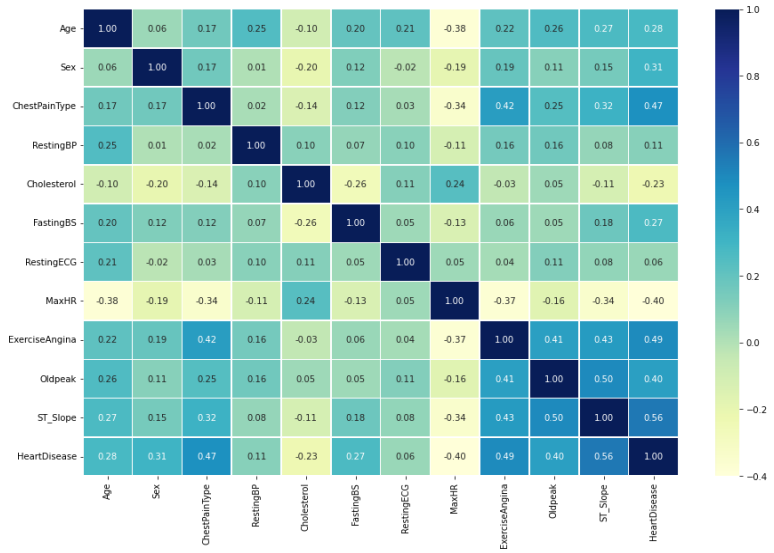


Fig.4.2.1: Correlation Matrix

- **Data pre-processing:** machine learning algorithms work exclusively with numeric data so we need to encode categorical feature into a repersntation compatible with the models. We used a Label Encoder which replaces the categorical value with a numeric value between 0 and the number of classes minus 1.

4.3. Data Wrangling

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step. It is one of the most important steps of the complete process. Cleaning of data is required to address the quality issues.

collected data may have various issues, including:

- Missing Values
- Duplicate data
- Invalid data
- Noise
- Unbalanced data, Fig.4.3.1

So, we use various filtering techniques to clean the data. It is mandatory to detect and remove the above issues because it can negatively affect the quality of the outcome.

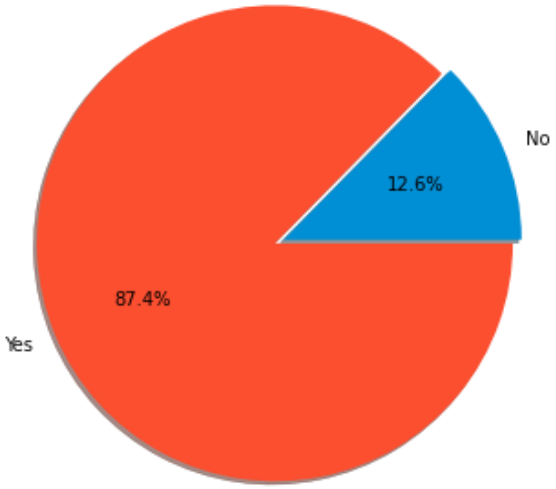


Fig.4.3.1: Example of Unbalanced Data

4.4. Data Analysis

Now the cleaned and prepared data is passed on to the analysis step. This step involves:

- Selection of analytical techniques
- Building models
- Review the result

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as Classification, Regression, Cluster analysis, Association, etc. then build the model using prepared data, and evaluate the model.

4.5. Model Training

In this step we train our model to improve its performance for better outcome of the problem. Training a model is required so that it can understand the various patterns, rules, and, features.

We used various machine learning algorithms for each disease, we will disccuss below the models we used.

4.5.1. Logistic Regression

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable, you can construct a logistic regression model using the **LogisticRegression class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LogisticRegression

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = LogisticRegression()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean())
```

4.5.2. Support Vector Machines (SVM)

Support Vector Machines (or SVM) seek a line that best separates two classes. Those data instances that are closest to the line that best separates the classes are called support vectors and influence where the line is placed. SVM has been extended to support multiple classes. Of particular importance is the use of different kernel functions via the kernel parameter. A powerful Radial Basis Function is used by default. You can construct an SVM model using the **SVC class**.

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.svm import SVC

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = SVC()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.9732836747361887
```

4.5.3. K- Nearst Neighbors

The k-Nearest Neighbors algorithm (or KNN) uses a distance metric to find the k most similar instances in the training data for a new instance and takes the mean outcome of the neighbors as the prediction. You can construct a KNN model using the **KNeighborsClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.neighbors import KNeighborsClassifier
```

```
X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = KNeighborsClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.9738485412787089
```

4.5.4. Naive Bayes

Naive Bayes calculates the probability of each class and the conditional probability of each class given each input value. These probabilities are estimated for new data and multiplied together, assuming that they are all independent (a simple or naive assumption). When working with real-valued data, a Gaussian distribution is assumed to easily estimate the probabilities for

input variables using the Gaussian Probability Density Function.

You can construct a Naive Bayes model using the **GaussianNB class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.naive_bayes import GaussianNB

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = GaussianNB()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.813066418373681
```

4.5.5. Decision Tree

Decision Tree construct a binary tree from the training data. Split points are chosen greedily by evaluating each attribute and each value of each attribute in the training data in order to minimize a cost function (like the Gini index). You can construct a CART model using the **DecisionTreeClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.tree import DecisionTreeClassifier

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = DecisionTreeClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.8280229671011794
```

Compared to Random Forest and ExtraTrees, Decision Tree has High variance

A single decision tree is usually overfits the data it is learning from because it learns from only one pathway of decisions. Predictions from a single decision tree usually don't make accurate predictions on new data.

4.5.6. Random Forest

Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a technique called Bootstrap and Aggregation, commonly known as bagging. The basic idea behind this is to combine multiple decision trees in determining the final output rather than relying on individual decision trees.

Random Forest has multiple decision trees as base learning models. We randomly perform row sampling and feature sampling from the dataset forming sample datasets for every model. This part is called Bootstrap. You can construct a Random Forest model using the **RandomForestClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = RandomForestClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.9532433271260086
```

Compared to Decision Tree and Extra Trees, Random Forest has medium variance

Random forest models reduce the risk of overfitting by introducing randomness by:

- building multiple trees (n_estimators)
- drawing observations with replacement (i.e., a bootstrapped sample)
- splitting nodes on the best split among a random subset of the features selected at every node

4.5.7. Extra Tree Classifier (Extremely Randomized Trees)

ExtraTreesClassifier is an ensemble learning method fundamentally based on decision trees. ExtraTreesClassifier, like RandomForest, randomizes certain decisions and subsets of data to minimize over-learning from the data and overfitting. You can construct an ExtraTree model using the **ExtraTreesClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import ExtraTreesClassifier

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = ExtraTreesClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.967169459962756
```

Compared to Decision Tree and Random Forest, ExtraTrees has Low variance

Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features, but with two key differences: it does not bootstrap observations (meaning it samples without replacement), and nodes are split on random splits, not best splits.

4.5.8. XGBoost

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. Weight of variables predicted wrong by the tree is increased and these the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems. You can construct an XGBoost model using the **XGBClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from xgboost import XGBClassifier

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
```

```
model = XGBClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) # 0.938193668528864
```

4.5.9. Ada Boost

AdaBoost algorithm falls under ensemble boosting techniques, it combines multiple models to produce more accurate results and this is done in two phases:

1. Multiple weak learners are allowed to learn on training data
2. Combining these models to generate a meta-model, this meta-model aims to resolve the errors as performed by the individual weak learners

You can construct an Ada Boost model using the **AdaBoostClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import AdaBoostClassifier

X, y = load_digits(return_X_y=True)
num_folds = 10
kfold = KFold(n_splits = num_folds)
model = AdaBoostClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.2604034761018001
```

4.5.10. Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

You can construct a Gradient Boosting model using the **GradientBoostingClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import GradientBoostingClassifier

X, y = load_digits(return_X_y=True)
num_folds = 2
kfold = KFold(n_splits = num_folds)
model = GradientBoostingClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.8875972064976922
```

4.5.11. Cat Boost

CatBoost or Categorical Boosting is a machine learning algorithm that uses gradient boosting on decision trees.

It is an open-source boosting library developed by Yandex. In addition to regression and classification, CatBoost is used for search, recommendation systems, personal assistant, self-driving cars, weather prediction and many other tasks at Yandex and in other companies, including CERN, Cloudflare, Careem taxi.

You can construct a Cat Boost model using the **CatBoostClassifier class**

```
from sklearn.datasets import load_digits
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

from catboost import CatBoostClassifier

X, y = load_digits(return_X_y=True)
num_folds = 2
kfold = KFold(n_splits = num_folds)
model = CatBoostClassifier()
results = cross_val_score(model, X, y, cv=kfold)
print(results.mean()) #0.9293288013655361
```

4.6. Model Testing

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.

Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

We could also here tune our parameters to get the highest accuracy for example change the maximum depth for a decision tree, change the learning rate for gradient descent, increase number of trees in a random forest, etc.

4.7. Deployement

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system.

If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. But before deploying the project, we will check whether it is improving its performance using available data or not. The deployment phase is similar to making the final report for a project.

4.8. Summary

In this section we explaind what is a machine learning life-cycle, explained each stage in depth and how did we implemented this step in our project

Aslo, we discussed why we should try different algorithms, which classifiers we used, and implemented example for each one.

5. Machine learning Models

5.1. Heart Disease

5.1.1. About the model

A typical goal is to classify the input into one of two states, heart disease or normal according to eleven user-provided features.

5.1.2. Data Set

This dataset was created by combining different datasets already available independently but not combined before. In this dataset, 5 heart datasets are combined over 11 common features which makes it the largest heart disease dataset available so far for research purposes. The five datasets used for its curation are:

- Cleveland: 303 observations
- Hungarian: 294 observations
- Switzerland: 123 observations
- Long Beach VA: 200 observations
- Stalog (Heart) Data Set: 270 observations

Total: 1190 observations
Duplicated: 272 observations

Final dataset: 918 observations
Every dataset used can be found under the Index of heart disease datasets from UCI Machine Learning Repository on the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

5.1.3. Data Set Distribution:

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
ExerciseAngina	918.0	0.404139	0.490992	0.0	0.00	0.0	1.0	1.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
ST_Slope	918.0	0.638344	0.607056	0.0	0.00	1.0	1.0	2.0
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

5.1.4. Features:

Age: age of the patient [years]

Sex: sex of the patient
0: Male
1: Female

ChestPainType:
0: Typical angina: chest pain related decrease blood supply to the heart,
1: Atypical angina: chest pain not related to heart,
2: non-anginal pain: typically, esophageal spasms (non heart related),
3: Asymptomatic: chest pain not showing signs of disease

RestingBP: resting blood pressure [mm Hg]

Cholesterol: serum cholesterol [mm/dl]

FastingBS: fasting blood sugar
1: if FastingBS > 120 mg/dl
0: otherwise

RestingECG: resting electrocardiogramresults
0: Normal: Normal
1:ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
2: LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria

MaxHR: maximum heart rate achieved [Numeric value between 60 and 202],
ExerciseAngina: exercise-induced angina
1: Yes
0: No

Oldpeak: oldpeak = ST [Numeric value measured in depression]

ST_Slope: the slope of the peak exercise ST segment
0: upsloping
1: flat
2: downsloping

5.1.5. Class Target

HeartDisease: output class
1: Heart disease
0: Normal

5.1.6. Data visualization

5.1.6.1. Data balance

The data set appears to be balanced as the number of people with heart disease and the number of people without heart disease are close as shown in fig.5.1.6.1.

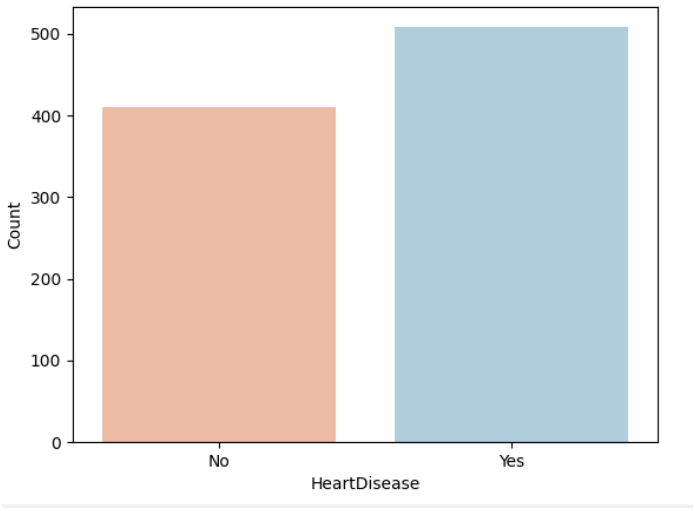


Fig.5.1.6.1: Data balance

5.1.6.2. Histogram of the age

The histogram is a popular graphing tool. It is used to summarize discrete or continuous data that are measured on an interval scale.

The distribution of the age is seeming to be normal distribution as shown in fig.5.1.6.2.

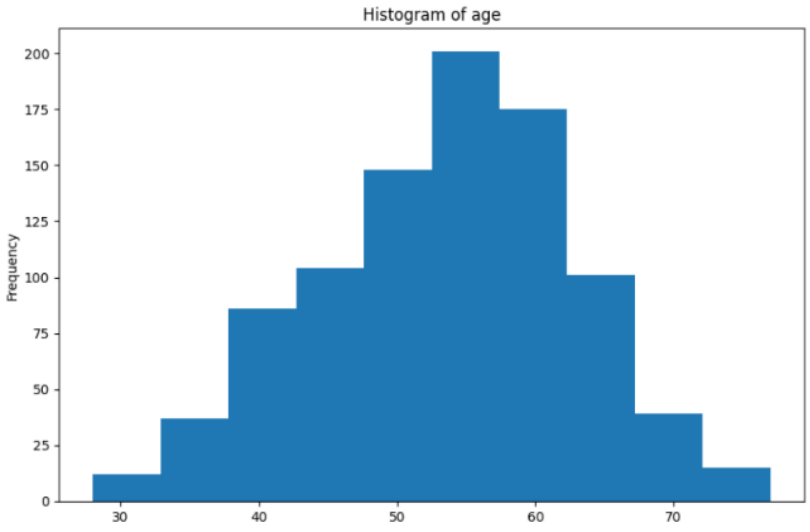


Fig.5.1.6.2: Histogram of age

5.1.6.3. Heart dieses frequency with gender

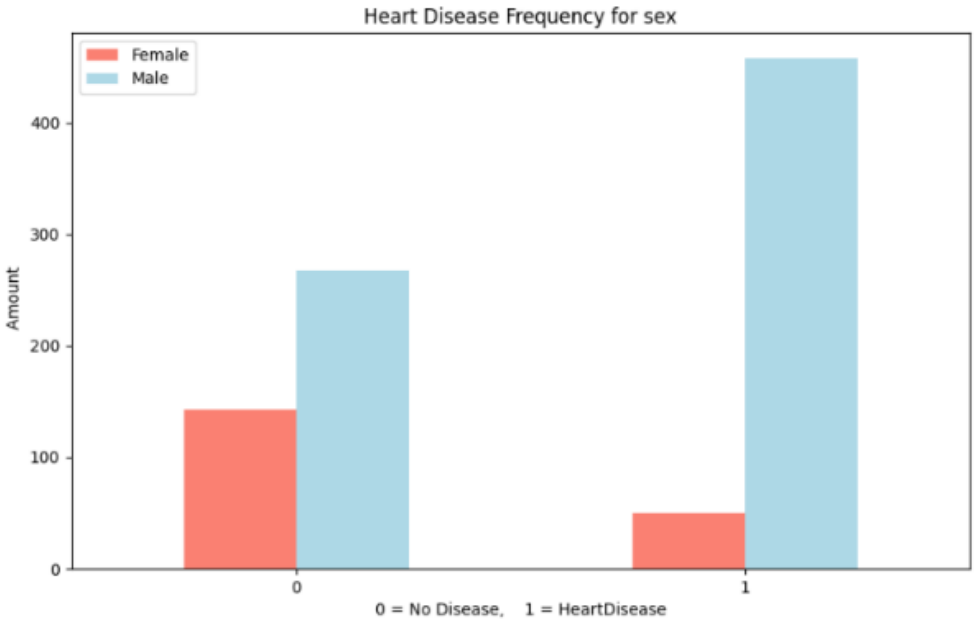


Fig.5.1.6.3: Heart dieses frequency with gender

5.1.6.4. *Heart dieses frequency with Chest pain type*

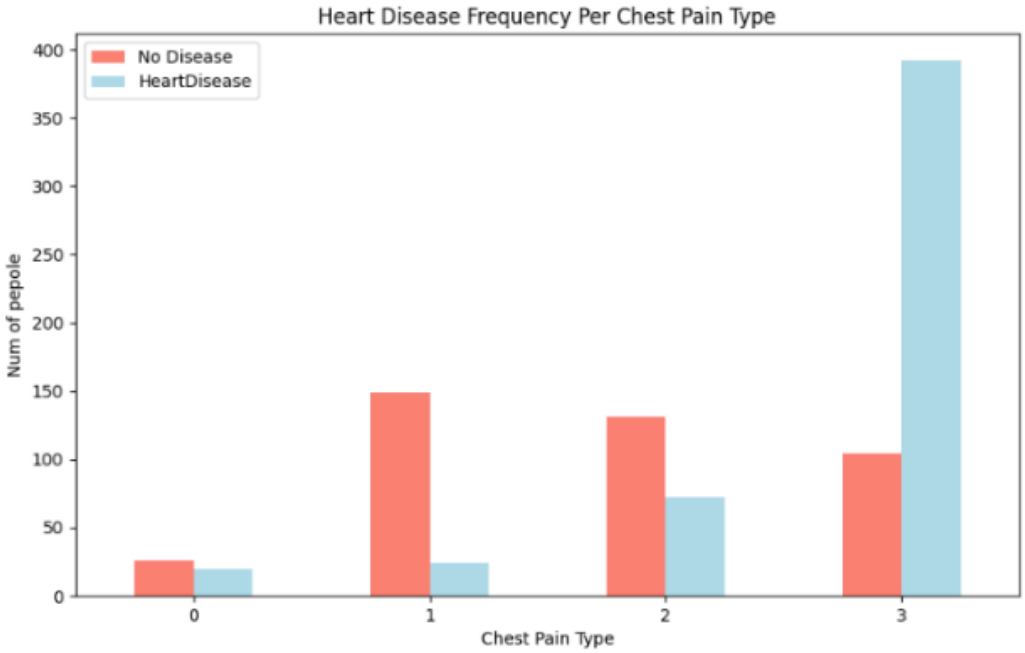


Fig.5.1.6.4: Heart dieses frequency with Chest pain type

5.1.6.5. *Heart Disease representation with Age and Max Heart Rate*

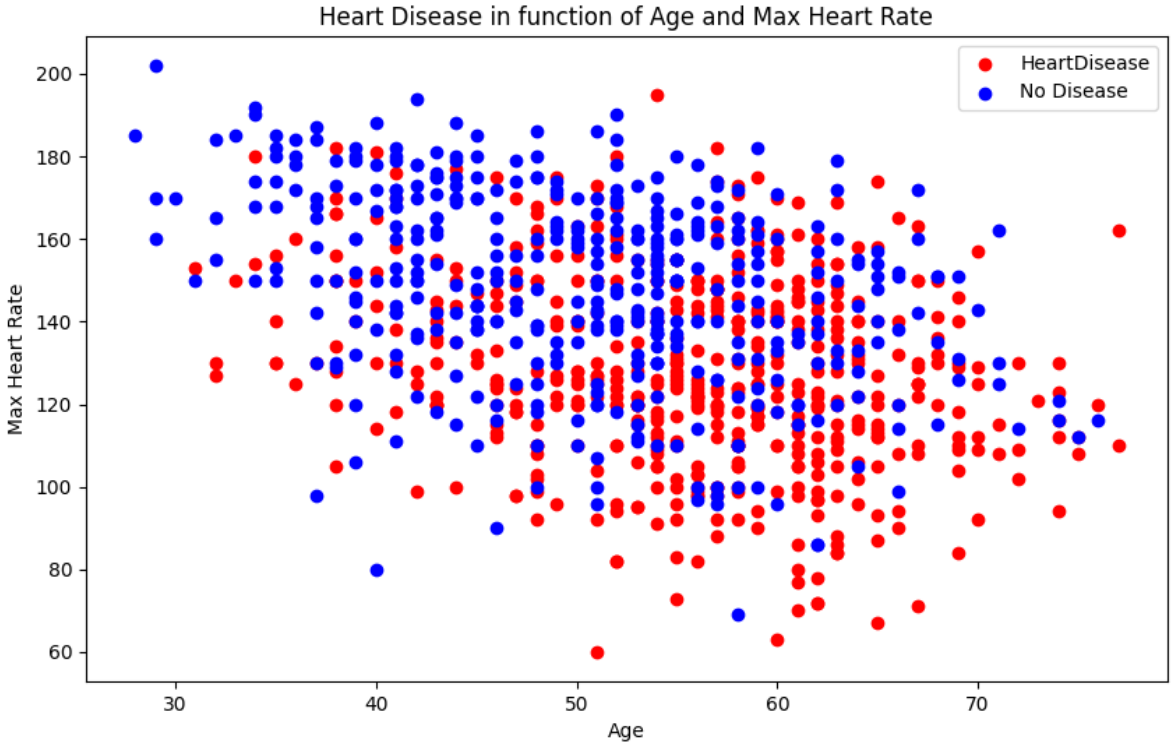


Fig.5.1.6.5: heart disease representation with Age and Max Heart Rate

5.1.7. Models Analysis

The final best model used for heart disease prediction was found to be **the Random Forest Classifier** model using 11 features with an accuracy of **0.913**

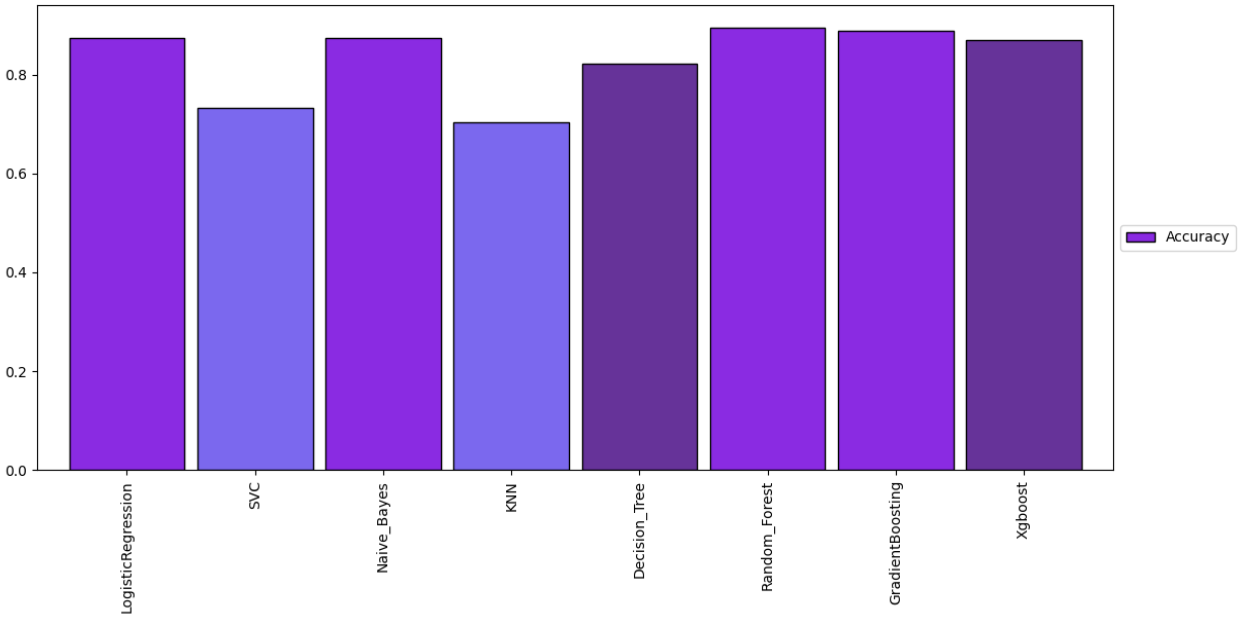


Fig.5.1.7.1: Models Accuracy

5.1.8. Evaluate the best model

5.1.8.1. confusion matrix

A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

The confusion matrix of heart disease model is shown in fig.5.1.8.1

	Predicted true	**Predicted false**
Actual true	109	14
Actual False	10	143

Fig.5.1.8.1: confusion matrix of Random Forest Classifier

5.1.8.2. classification report

Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to learn or train a model and the other used to validate the model.

Cross-Validation of heart disease model is shown in fig.5.1.8.2

Cross-validated accuracy	Cross-validated recall
0.8365	0.8463
Cross-validated precision	Cross-validated f1-score
0.8454	0.8465

Fig.5.1.8.2: cross- validation of Random Forest Classifier

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that were retrieved. Both precision and recall are therefore based on relevance.

The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean.

Precision, recall and f1-score values of the heart disease model is shown in fig.5.1.8.3, fig.5.8.4 and fig.5.1.8.5.

	Precision	Recall	F1-score	Support
0	0.92	0.89	0.90	123
1	0.91	0.93	0.92	153

Fig.5.1.8.3

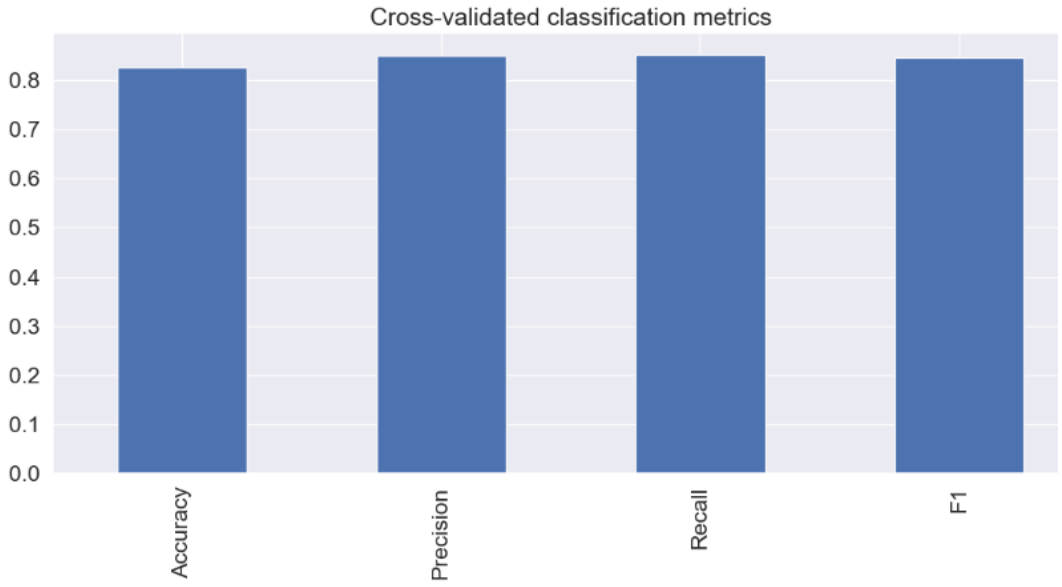


Fig.5.1.8.4

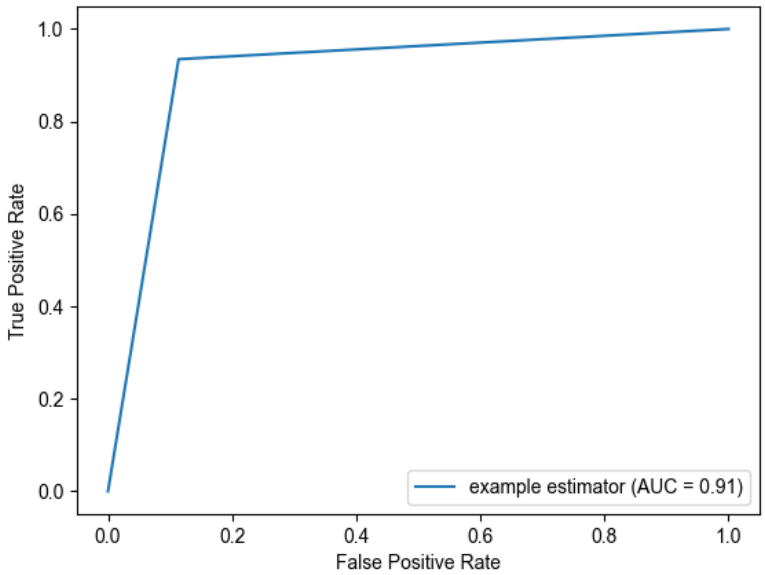


Fig.5.1.8.5

5.2. Lung Cancer Disease

5.2.1. About the Model

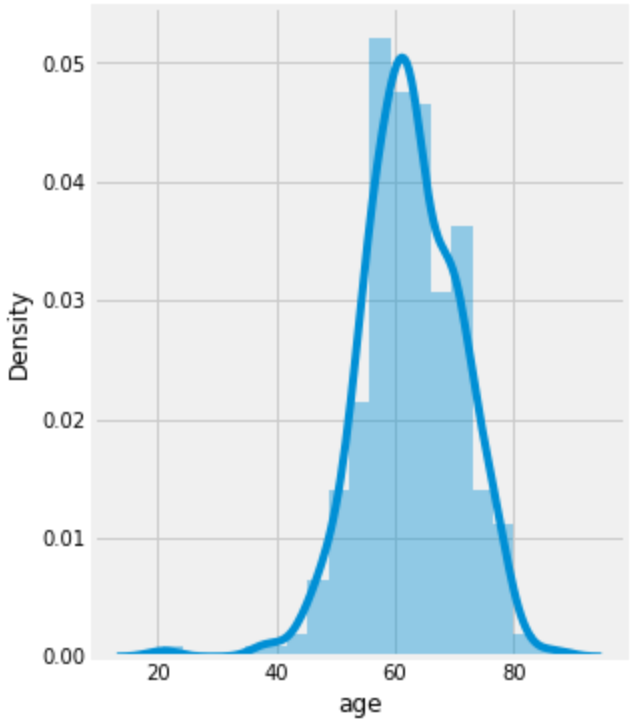
The effectiveness of cancer prediction system helps the people to know their cancer risk with low cost and it also helps the people to take the appropriate decision based on their cancer risk status. The data is collected from the website online lung cancer prediction system

5.2.2. Dataset

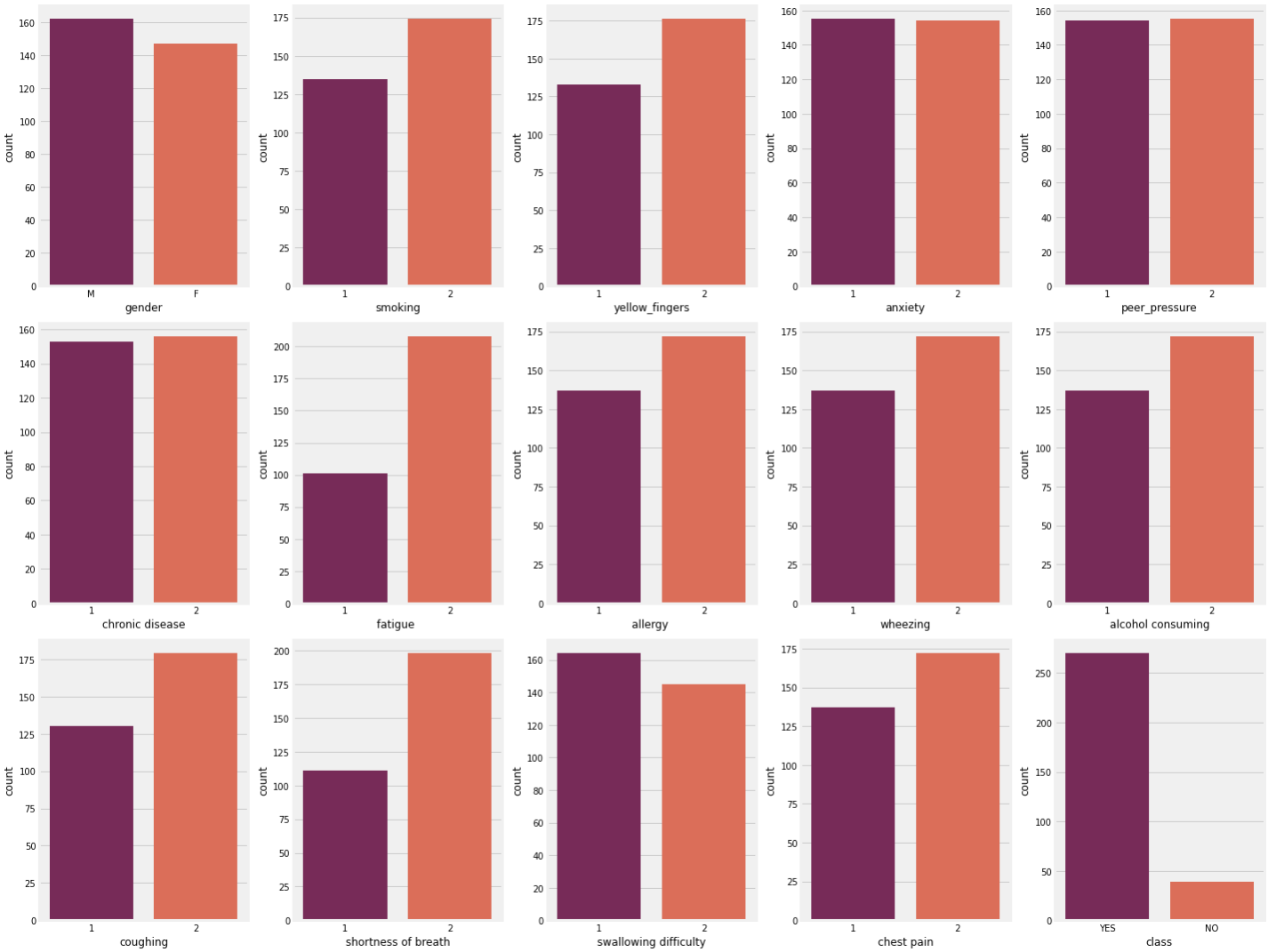
The dataset contains 16 attributes: gender, age, smoking, yellow fingers, anxity, presure, chronic disease, fatigue, allergy, wheezing, alcohol, coughing, shotness of breath, shallowing difficulty, chest pain.

5.2.3. Model Analysis

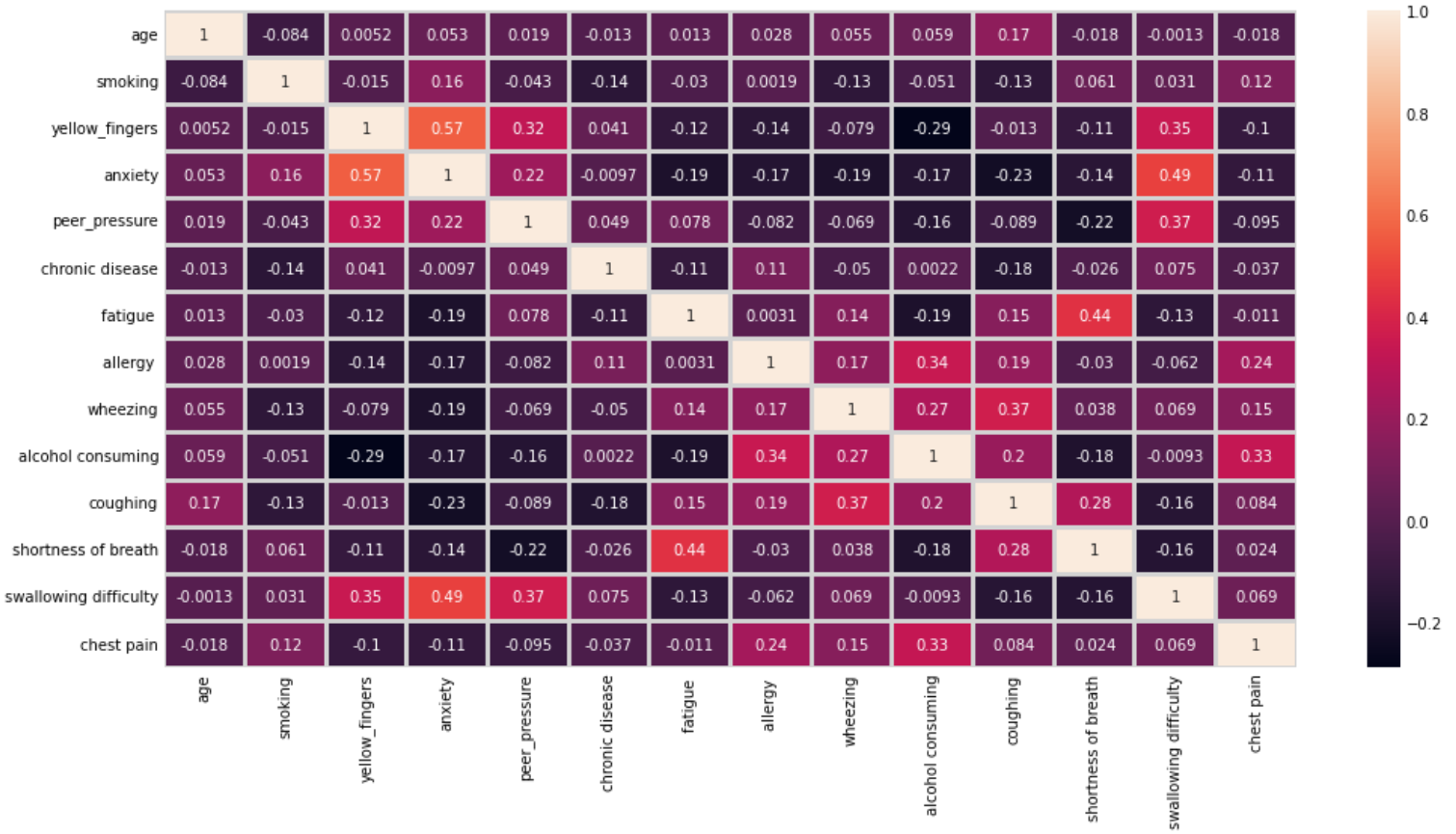
5.2.3.1. Numerical Feature Distribution



5.2.3.2. Categorical Features

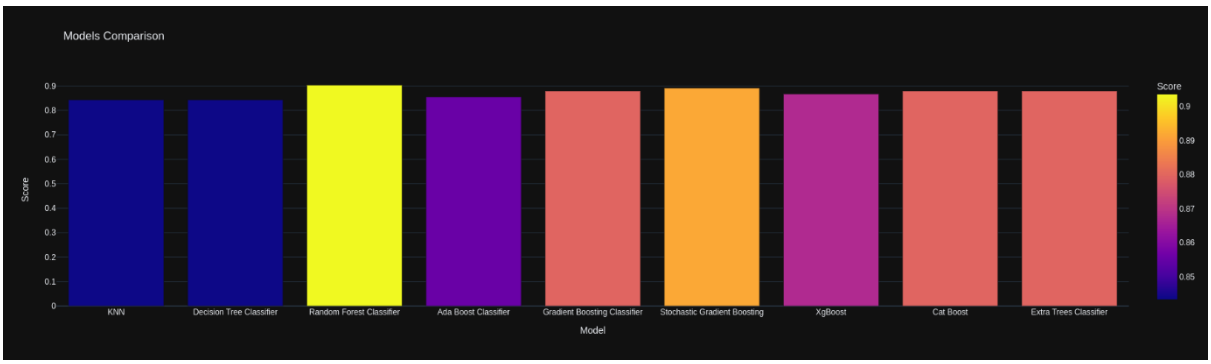


5.2.3.3. *Heatmap of the Features*



5.2.3.4. *Different Models Accuracies*

The final model used for Lung Cancer prediction was found to be the Random Forest Classifier model with all features and an accuracy of 0.903614



5.3. Dermatology Disease

5.3.1. About the Model

The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, cronic dermatitis, and pityriasis rubra pilaris. Usually, a biopsy is necessary for the diagnosis but unfortunately these diseases share many histopathological features as well. Another difficulty for the differential diagnosis is that a disease may show the features of another disease at the beginning stage and may have the characteristic features at the following stages. Patients were first evaluated clinically with 12 features. Afterwards, skin samples were taken for the evaluation of 22 histopathological features. The values of the histopathological features are determined by an analysis of the samples under a microscope.

5.3.2. Dataset information

This database contains 34 attributes, 33 of which are linear valued and one of them is nominal.

In the dataset constructed for this domain, the family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise. The age feature simply represents the age of the patient. Every other feature (clinical and histopathological) was given a degree in the range of 0 to 3. Here, 0 indicates that the feature was not present, 3 indicates the largest amount possible, and 1, 2 indicate the relative intermediate values.

5.3.3. Attribute Information

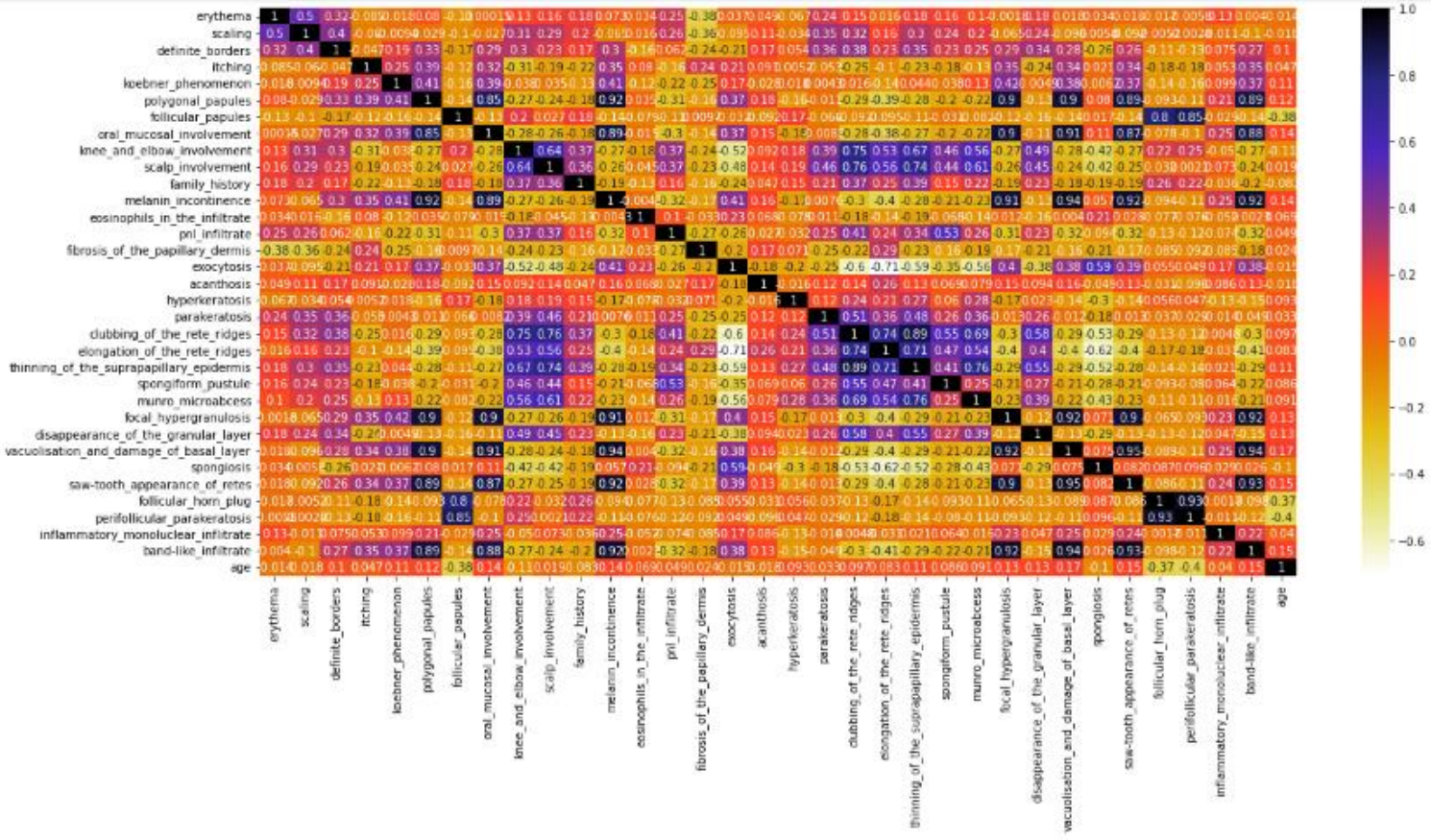
Clinical Attributes: (take values 0, 1, 2, 3, unless otherwise indicated)
1: erythema ,scaling , definite borders, itching, koebner phenomenon, polygonal papules, follicular papules ,oral mucosal involvement, knee and elbow involvement, scalp involvement, family history,(0 or 1),Age (linear)

Histopathological Attributes: (take values 0, 1, 2, 3)
melanin incontinence,eosinophils in the infiltrate,PNL infiltratefibrosis of the papillarydermisexocytosis, acanthosis,hyperkeratosis,parakeratosis,clubbing of the rete ridges,elongation of the rete ridges,thinning of the suprapapillary epidermis,spongiform pustule,munro microabcess,focal hypergranulosis,disappearance of the granular layer,vacuolisation and damage of basal layer,spongiosis,saw-tooth appearance of retes,follicular horn plug,perifollicular parakeratosis,inflammatory monoluclear inflitrateband-like infiltrate

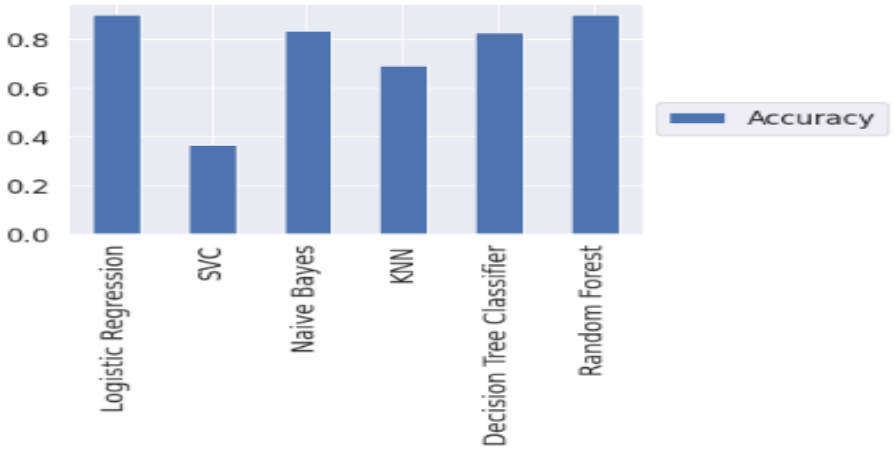
The Source of the dataset can be found at: https://archive.ics.uci.edu/ml/machine-learning-databases/dermatology/

5.3.4. Model Analysis

5.3.4.1. Heatmap of the Features



5.3.4.2. Different Models Accuracies



5.4. Chronic Kidney Disease dataset

5.4.1. About the Model

The data was taken over a 2-month period in India with 25 features (eg, red blood cell count, white blood cell count, etc). The target is the 'classification', which is either 'ckd' or 'notckd' - ckd=chronic kidney disease. There are 400 rows

5.4.2. Dataset

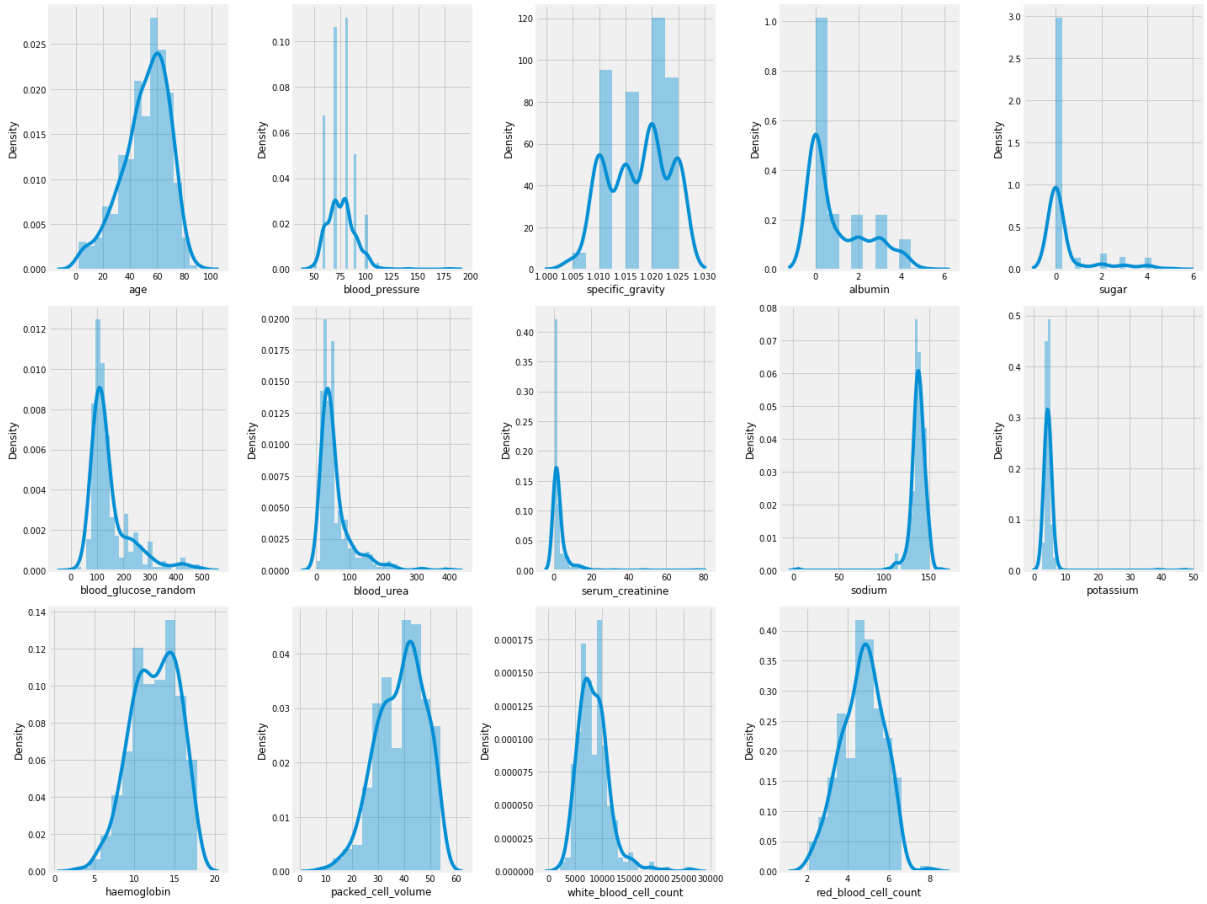
The dataset contains 25 features that help in classifying a patient with chronic kidney disease:

'age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium', 'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count','hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite', 'peda_edema', 'aanemia', 'class'

The Source of the dataset can be found at:
https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease

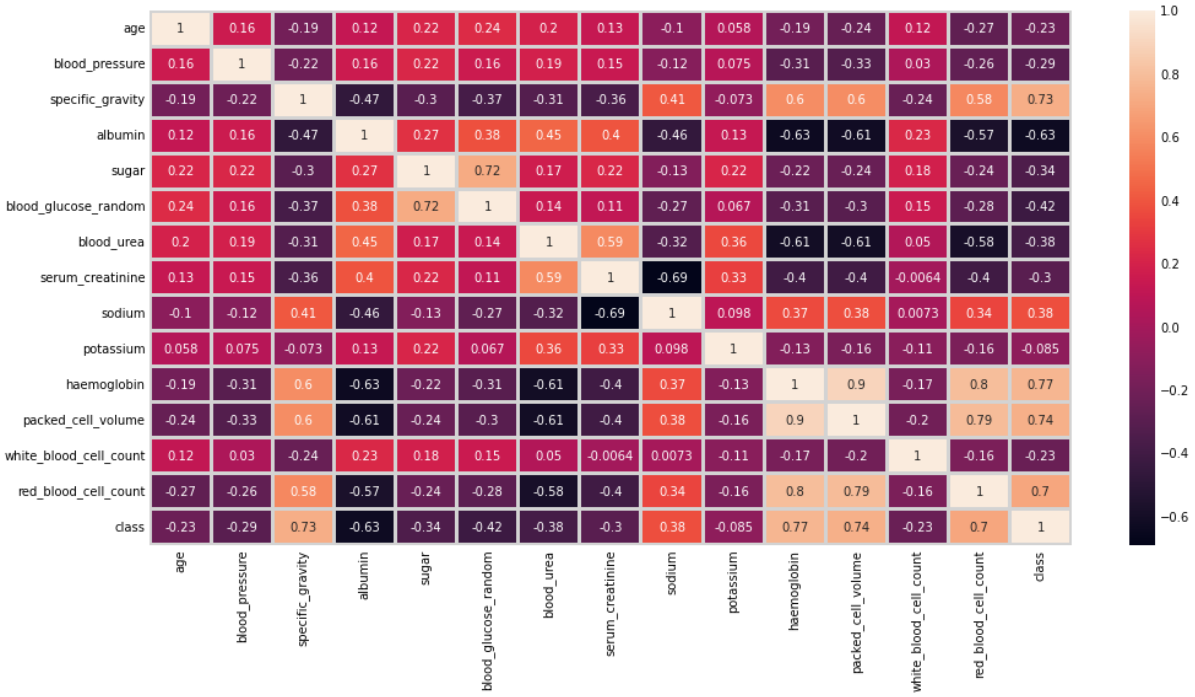
5.4.3. Model Analysis

5.4.3.1. Numerical Features Distribution



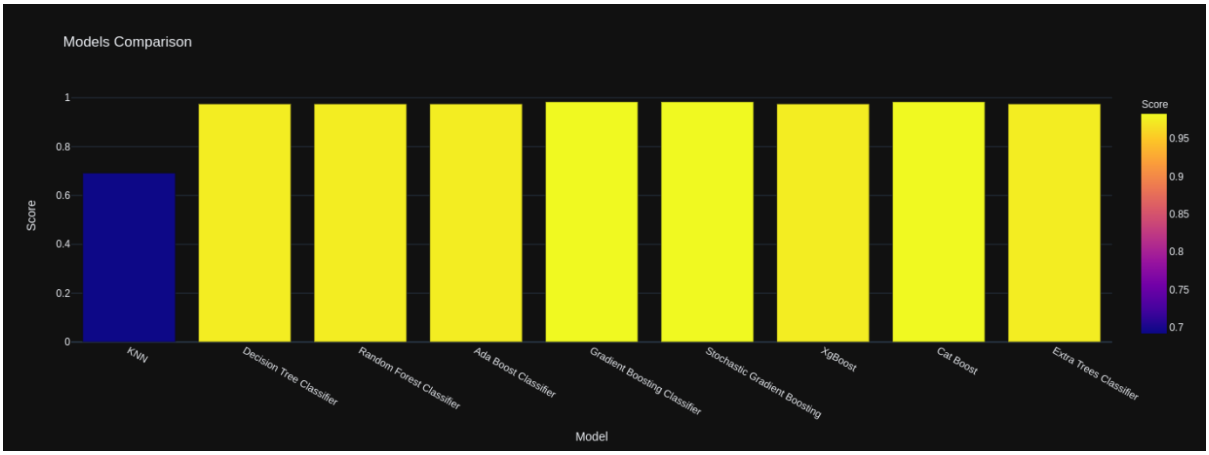
5.4.3.2. Categorical Features

5.4.3.3. Heatmap of the Features



5.4.3.4. Different Models Accuracies

The final best model used for chronic kidney disease prediction was found to be the Gradient Boost Classifier model with all features and an accuracy of 0.99166.



5.5. Breast Cancer Disease Dataset

5.5.1. About the Model

The Wisconsin Diagnostic Breast Cancer dataset is a real-valued multivariate data that consists of two classes, where each class signifies whether a patient has breast cancer or not. The two categories are: malignant and benign. Each record represents data for one breast cancer case. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

5.5.2. Dataset

The dataset contains data of 569 records with 32 attributes consisting of the ID number, diagnosis and 30 real-valued attributes, with no missing attribute value. The attributes are grouped into 3 groups: mean, standard error, and "worst" or largest (mean of the three largest values). Each group is computed for each of the 10 cell nucleus characteristics: radius (mean of distances from centre to points on the perimeter), texture (standard deviation of grey-scale values), perimeter, area, smoothness (local variation in radius lengths), compactness (perimeter^2 / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry and fractal dimension ("coastline approximation" - 1).

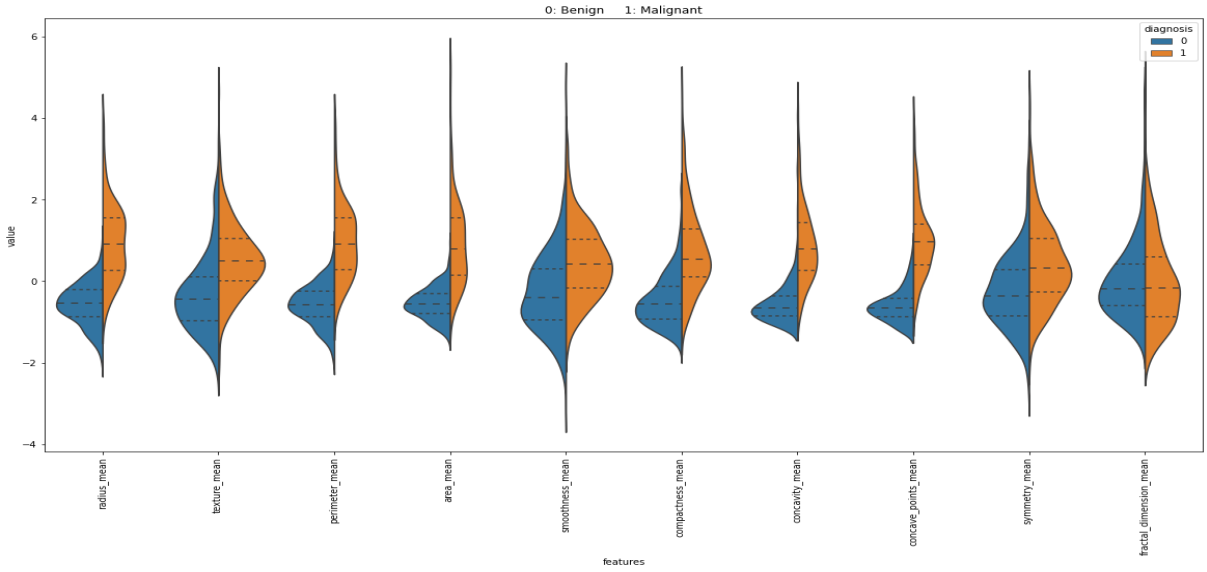
The source of the dataset can be found at:

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

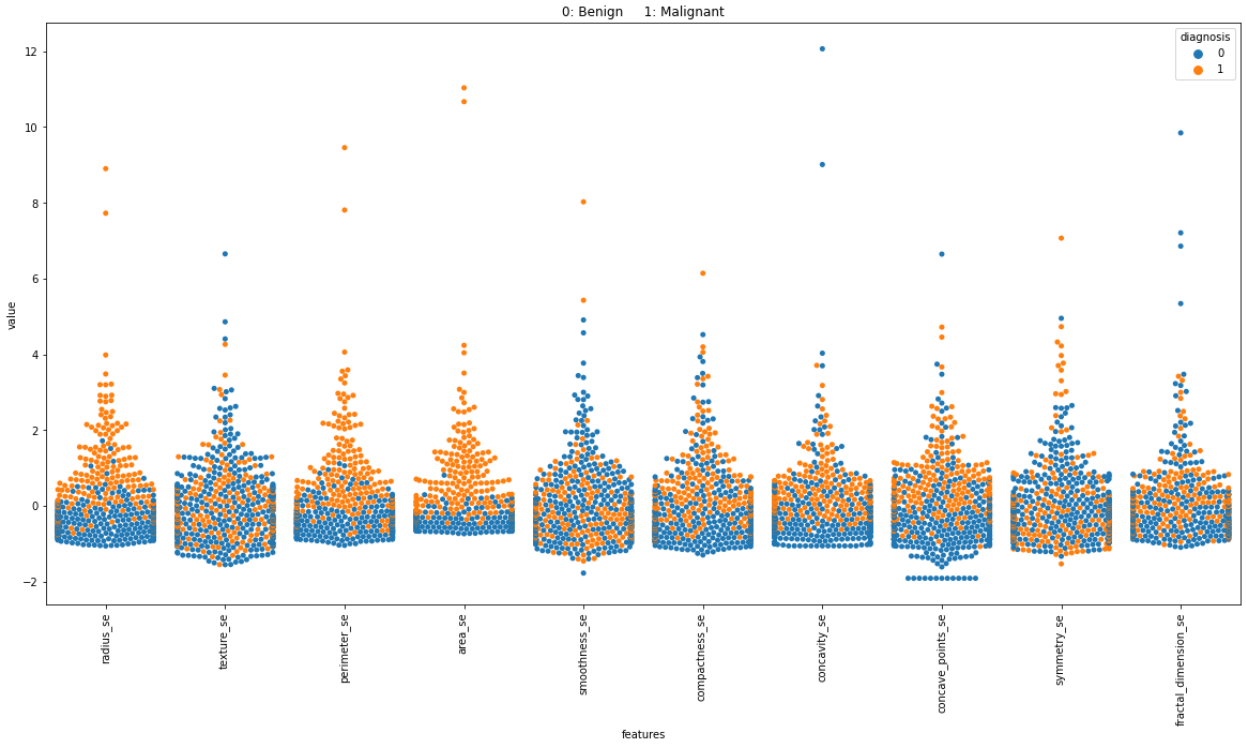
5.5.3. Model Analysis

5.5.3.1. Numerical Features Distribution

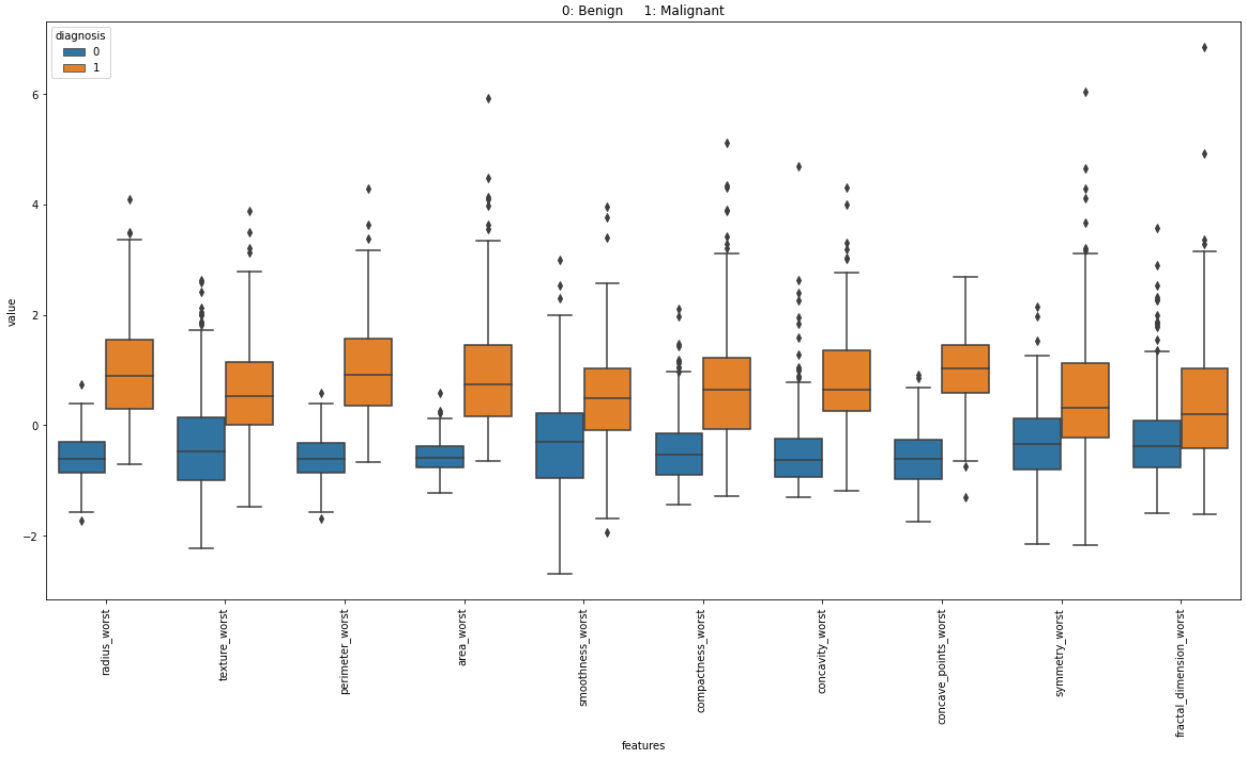
5.5.3.1.1. Mean of Cell Nucleus Characteristics



5.5.3.1.2. Standard Error of Cell Nucleus Characteristics

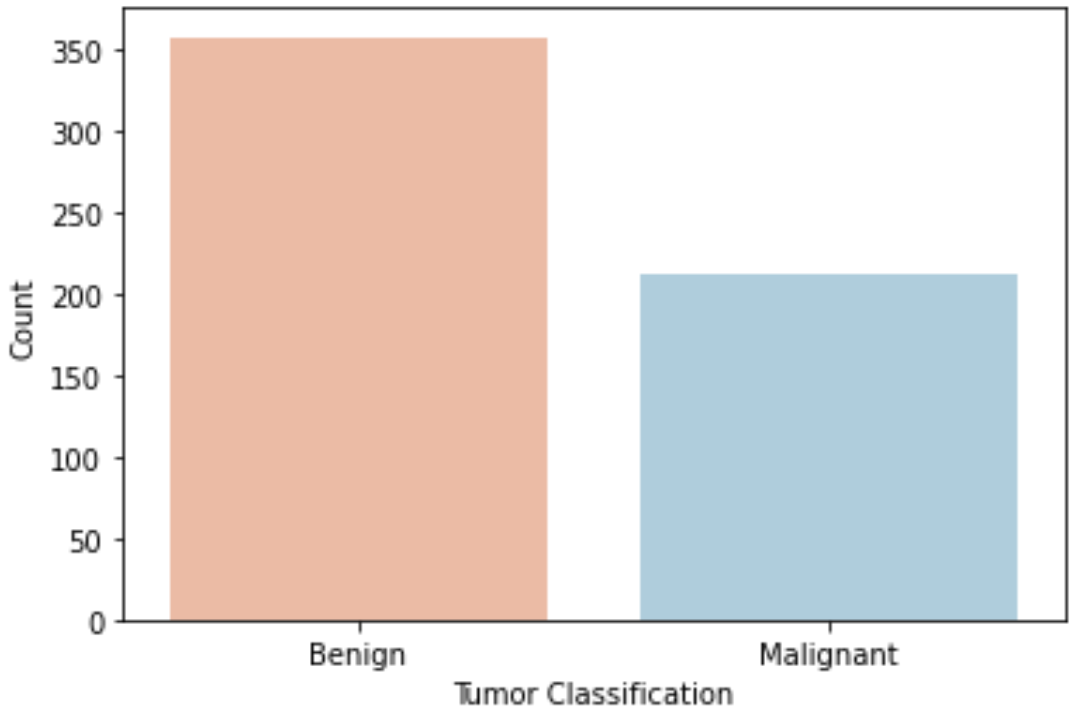


5.5.3.1.3. Worst of Cell Nucleus Characteristics

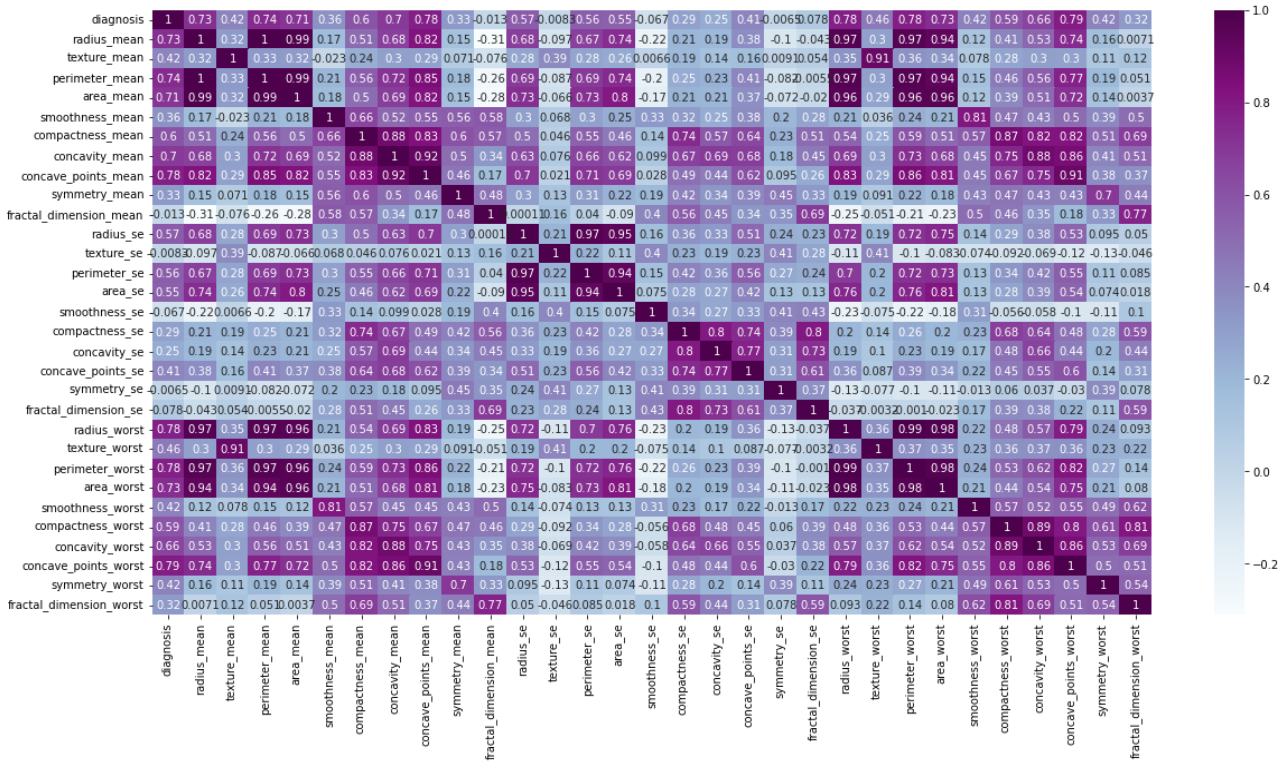


5.5.3.2. Categorical Features

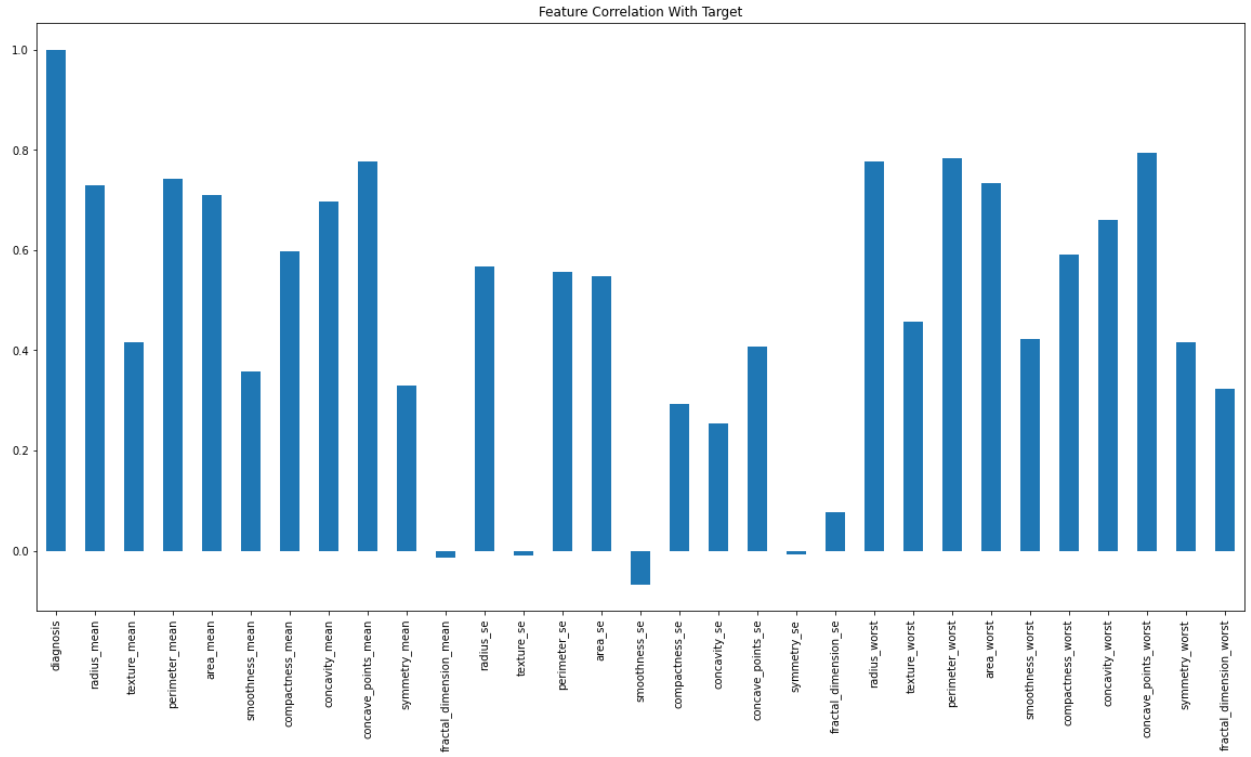
5.5.3.2.1. Target Class (Diagnosis)



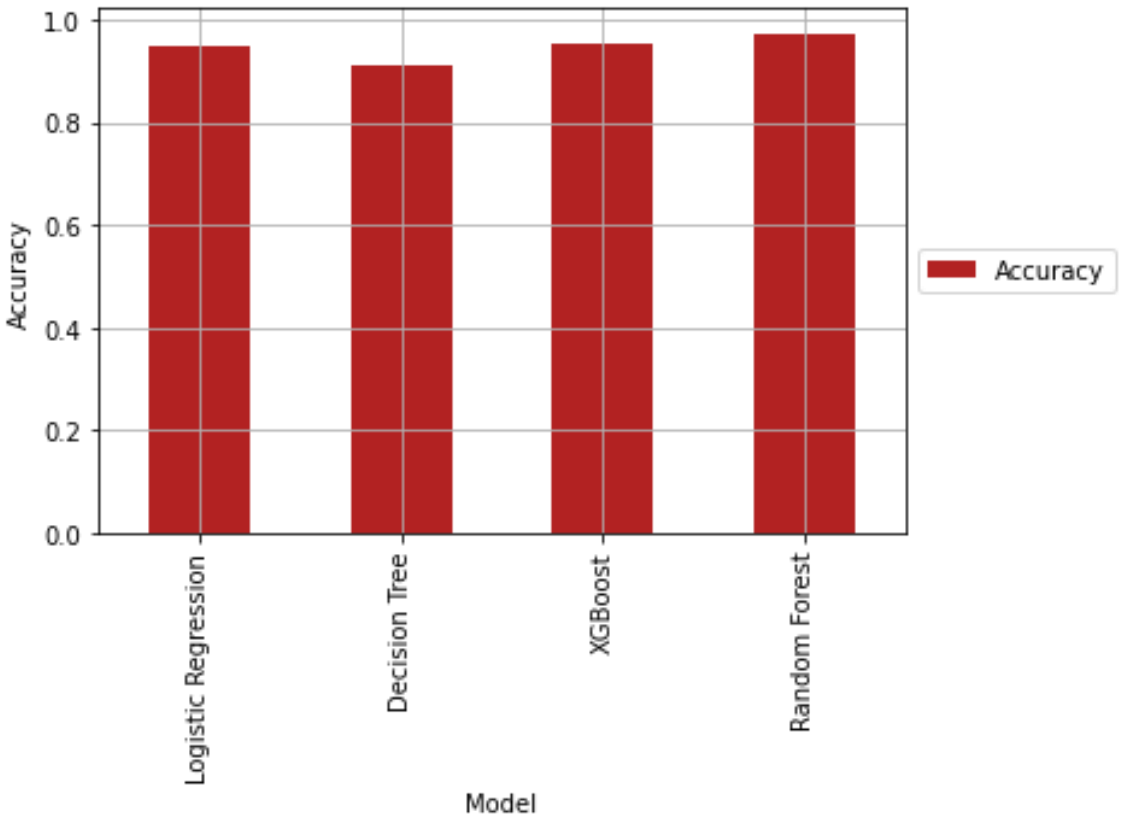
5.5.3.3. Heatmap of the Features



5.5.3.4. Features Correlation with Target Class



5.5.3.5. *Different Model Accuracies*



5.5.3.6. *Best Model Evaluation*

5.5.3.6.1. Model

The final best model used for breast cancer prediction was found to be the Random Forest Classifier model with a feature selection method of Recursive Feature Elimination using 8 features selected out of 30 and an accuracy of 0.9736842105263158.

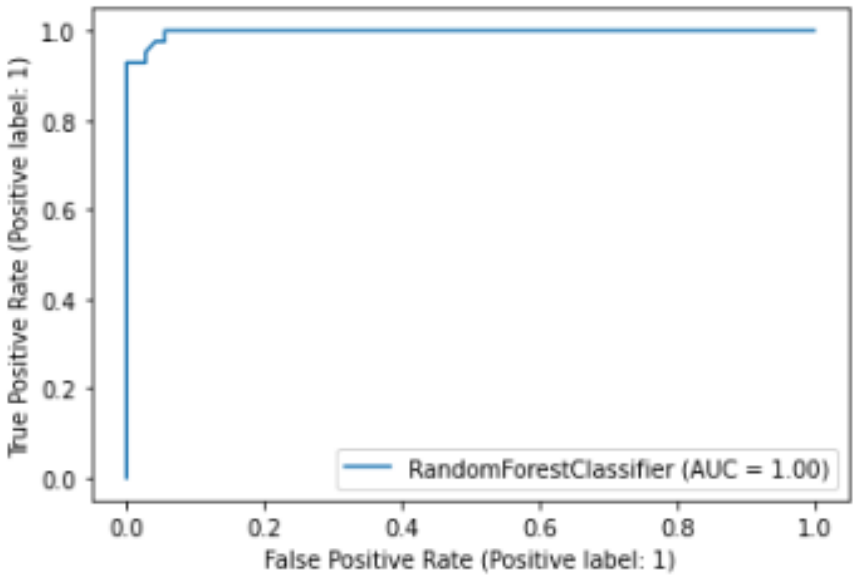
5.5.3.6.2. Selected Features

```
'area_mean', 'concavity_mean', 'concave_points_mean',
'radius_worst', 'texture_worst', 'perimeter_worst',
'area_worst', 'concave_points_worst'
```

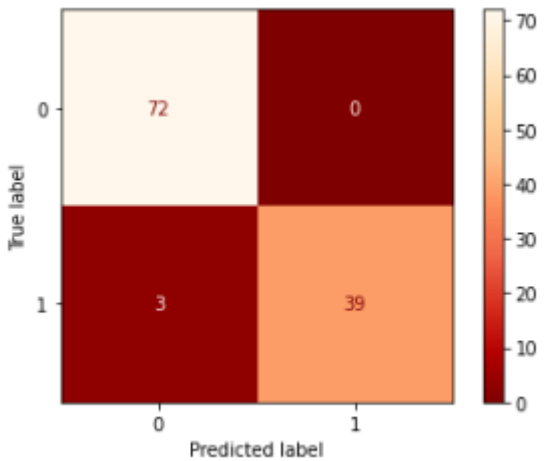
5.5.3.6.3. Classification Report

	precision	recall	f1-score	support
0	0.96	1.00	0.98	72
1	1.00	0.93	0.96	42
accuracy			0.97	114
macro avg	0.98	0.96	0.97	114
weighted avg	0.97	0.97	0.97	114

5.5.3.6.4. ROC Curve



5.5.3.6.5. Confusion Matrix



5.6. Diabetics Disease

5.6.1. About the Model

The data was collected and made available by “National Institute of Diabetes and Digestive and Kidney Diseases” as part of the Pima Indians Diabetes Database. Several constraints were placedon the selection of these instances from a larger database. In particular, all patients here belong tothe Pima Indian heritage (subgroup of Native Americans), and are females of ages 21 and above.

We’ll be using Python and some of its popular data science related packages. First of all, we will import pandas to read our data from a CSV file and manipulate it for further use. We will also use

numpy to convert out data into a format suitable to feed our classification model. We’ll use seaborn and matplotlib for visualizations. We will then import Logistic Regression algorithm from sklearn. Thisalgorithm will help us build our classification model. Lastly, we will use joblib available in sklearn to save our model for future use.

5.6.2. Dataset

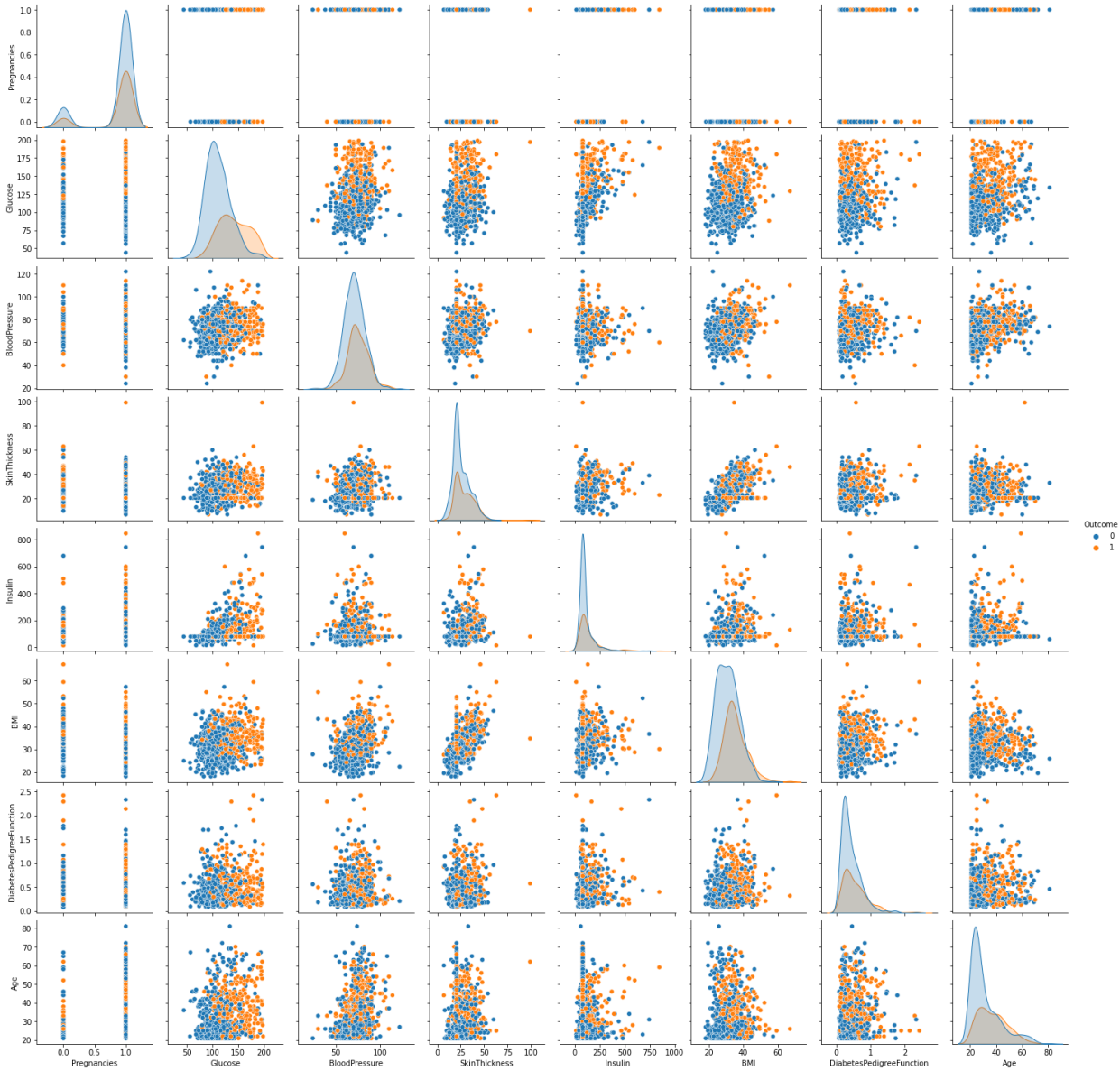
The dataset contains 8 features that help in classifying a patient with Diabetes disease:

Pregnancis GlucoesBloodPressue SkinThickness Insulin BMI DiabetesPedigreeFunction Age

The Source of the dataset can be found at:
https://www.kaggle.com/datasets/kandij/diabetes-dataset

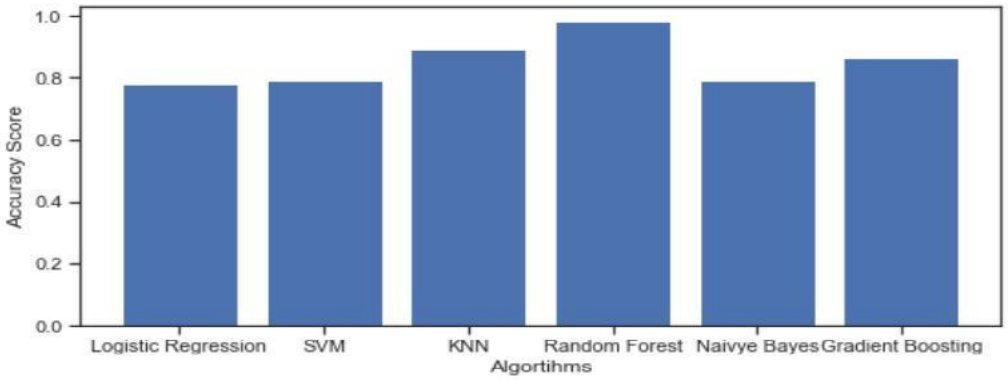
5.6.3. Model Analysis

5.6.3.1. Data visualisation



5.6.3.2. Heatmap of the features

5.6.3.3. Different Models Accuracies



6. Frontend

6.1. Definition

The part of a website that the user interacts with directly is termed the front end. It is also referred to as the 'client side' of the application. It includes everything that users experience directly: text colors and styles, images, graphs and tables, buttons, colors, and navigation menus. The structure, design, behavior, and content of everything seen on browser screens when websites, web applications, or mobile apps are opened up, is implemented by front End developers. Responsiveness and performance are two main objectives of the Front End. The developer must ensure that the site is responsive i.e., it appears correctly on devices of all sizes no part of the website should behave abnormally irrespective of the size of the screen.

From a user standpoint, the frontend is synonymous with the user interface. From a developer standpoint, it is the interface design and the programming that makes the interface function. Conversely, the backend includes functions and data processing that takes place behind the scenes.

Examples of frontend elements include:

- application or page layout
- graphics
- audio and video elements
- text content
- user interface elements (buttons, links, toolbars, navigation bars, etc.)
- input areas (dialog boxes), form fields, text areas, etc.)
- user flow (how one interface leads to the next)
- user preferences, themes, and customizations

One of the primary goals of frontend development is to create a smooth user experience. In other words, the front end of an application or website should be intuitive and easy to use. While this sounds like a simple goal, it can be surprisingly complex since not all users or devices are the same. For example, an app developed for a mobile device requires a significantly different frontend than a desktop application. Websites must work well on multiple devices and screen sizes, which is why modern web development typically involves responsive design.

6.2. Role of the Frontend

It is an aim to provide a user interface that is easy and speedy to use, that provides a way of communication between the user and the backend through requests and responses. For a user to obtain an output, that is, to obtain a probabilistic predicted diagnosis, the user will insert the disease he wishes to predict, along with some specific features that will help our models in the prediction process. The input is received through frontend elements, such as radio buttons, select boxes, numerical inputs and text inputs, by clicking a button, the input is sent as a post request to the API of the backend, which inturn communicates with the models to receive a prediction. Finally, the API will respond to the user with the output as an API response presented to the user as a text element (See figure 6.2.1). To visualize the entire architecture, seek figure 6.2.1 in Architecture and Technology Stack section. The user can also read more information about the model used for each disease, which also consists of some charts for better visualization.

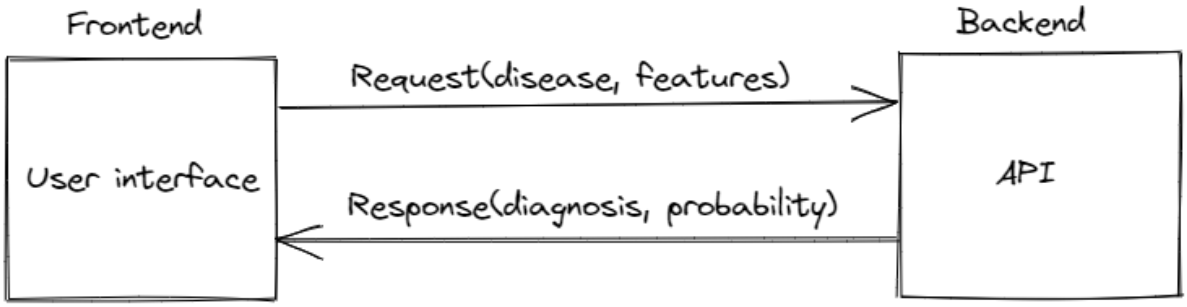


Figure 6.2.1 Simple Architecture of Frontend-Backend Communication

6.3. Typical Frontend Development Technologies

6.3.1. HyperText Markup Language

HyperText Markup Language (HTML) is the backbone of any website development process, without which a web page does not exist. Hypertext means that text has links, termed hyperlinks, embedded in it. When a user clicks on a word or a phrase that has a hyperlink, it will bring up another web-page. A markup language indicates text can be turned into images, tables, links, and other representations. It is the HTML code that provides an overall framework of how the site will look. HTML was developed by Tim Berners-Lee. The latest version of HTML is called HTML5 and was published on October 28, 2014 by the W3C recommendation. This version contains new and efficient ways of handling elements such as video and audio files.

6.3.2. Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) controls the presentation aspect of the site and allows your site to have its own unique look. It does this by maintaining style sheets that sit on top of other style rules and are triggered based on other inputs, such as device screen size and resolution. The CSS can be added externally, internally, or embedded in the HTML tags.

6.3.3. JavaScript

JavaScript is an event-based imperative programming language (as opposed to HTML's declarative language model) that is used to transform a static HTML page into a dynamic interface. JavaScript code can use the Document Object Model (DOM), provided by the HTML standard, to manipulate a web page in response to events, like user input.

Using a technique called AJAX, JavaScript code can also actively retrieve content from the web (independent of the original HTML page retrieval), and also react to server-side events as well, adding a truly dynamic nature to the web page experience.

6.4. Data Scientists Frontend Development Tools and Frameworks

6.4.1. Gradio

Gradio is a powerful tool that is used to create machine learning model user interfaces. It is a python package that is perfectly compatible with several machine learning frameworks such as PyTorch and TensorFlow. It can also be used to create UIs around arbitrary general-purpose Python scripts.

The popularity of Gradio is increasing rapidly. It is being used by machine learning labs at large companies like Cisco and Amazon, as well as academic settings like Stanford medicine.

Pros:

- Gradio offers several customizable UI components that are optimized for machine learning models. For example, Gradio provides easy-to-use drag-and-drop image classification that is highly optimized for the user.
- It is very easy and fast to set up Gradio. It can be installed directly through pip. Moreover, creating interfaces in Gradio requires only a couple of lines of code.
- Gradio creates shareable links which are probably the fastest way to get a deployed machine learning in front of users
- Unlike the other packages, Gradio can be run anywhere from within a Jupyter/Colab notebook or a standalone Python script.

Cons:

- Though Gradio has good documentation on the official website, it is hard to find information and enough examples about the specific features that it offers.
- Gradio has a smaller community than some of the other packages, making it harder to find resources about it.
- Gradio is specific towards building UIs for machine learning models, and less towards dashboards.

6.4.2. Dash

Dash is an open-source Python library used for creating reactive web applications. It is a user-interface library. Dash is a powerful tool for building analytical web applications.

Pros:

- Dash can be used with various domains such as data exploration, data analysis, modeling, visualization, instrument control, and reporting.
- It is very simple to use. It can be installed directly through pip.
- Apart from Python, it can be used with R, Julia, and Jupyter.
- Dash applications are reactive.
- Dash is more customizable than Streamlit. Also, Dash offers better performance.
- Dash has better, regularly-updated, easy-to-follow documentation.

Cons:

- Dash is more focused on the enterprise market and doesn't include all of its available features in the open-source version.
- Dash is more verbose than Gradio or Streamlit; you have to write more code to create simple web applications compared to other frameworks.

6.4.3. Streamlit

Streamlit is another popular tool that is used to create user interfaces. It is an open-source Python library that is used to build powerful, custom web applications for data science and machine learning. Streamlit is compatible with several major libraries and frameworks such as Latex, OpenCV, Vega-Lite, seaborn, PyTorch, NumPy, Altair, and more.

Like Gradio, Streamlit is also popular and used among big industry leaders, such as Uber and Google X.

Pros:

- Streamlit is accessible for everyone who understands Python. There is no requirement for HTML and CSS.
- It has a wide range of UI components. It covers almost every common UI component such as checkbox, slider, a collapsible sidebar, radio buttons, file upload, progress bar, etc. Moreover, these components are very easy to use.
- It supports multiple interactive visualization libraries such Latex, OpenCV, Vega-Lite, etc.

Cons:

- While not difficult, Streamlit does require some time to learn its own syntax.
- Streamlit is not that flexible. It is only based on Python, offers a limited set of widgets, and doesn't integrate with Python Notebooks.
- The data upload limit is only 50Mb.
- There is limited support for video/animation.

6.4.4. Flask

Flask is a python web framework used to build web applications. It offers several modules for web development that makes the developer's job easier. Flask also has resources to deploy a machine learning model, though any front-end work must be done by the developer. This means that as a data scientist, you must also be comfortable with HTML, CSS, and JavaScript.

Pros:

- As it is a framework, you can build everything from scratch according to your own requirements.
- Using flask, it is easier to build customizable applications.
- More generic web applications can be built using Flask.
- Flask is very mature and stable; something you build with flask is more certain to continue working into the future.
- There is extensive documentation about Flask and related libraries out there.

Cons:

- Flask does not provide any sort of UI components for machine learning or data science applications.
- Intermediate Python knowledge is required, as well knowledge of HTML, CSS, and JavaScript.
- Building the solution from scratch can be resource intensive.

	Simplicity	Maturity	Flexibility	Primary Use
Gradio	A	C	B	ML Model Demos
Streamlit	A	C	B	Dashboards
Dash	B	B	B	Dashboards
Flask	C	A	A	Web Interfaces

Figure 6.5.1 Summary Comparison

6.5. Why Streamlit

Streamlit is the simplest and the quickest solution to build a frontend for machine learning. It requires no knowledge about HTML, CSS, and JavaScript, just pure python scripting. It consists of a wide set of UI elements that creates a beautiful web app. Thus, Streamlit was used to build the frontend.

6.6. Frontend design

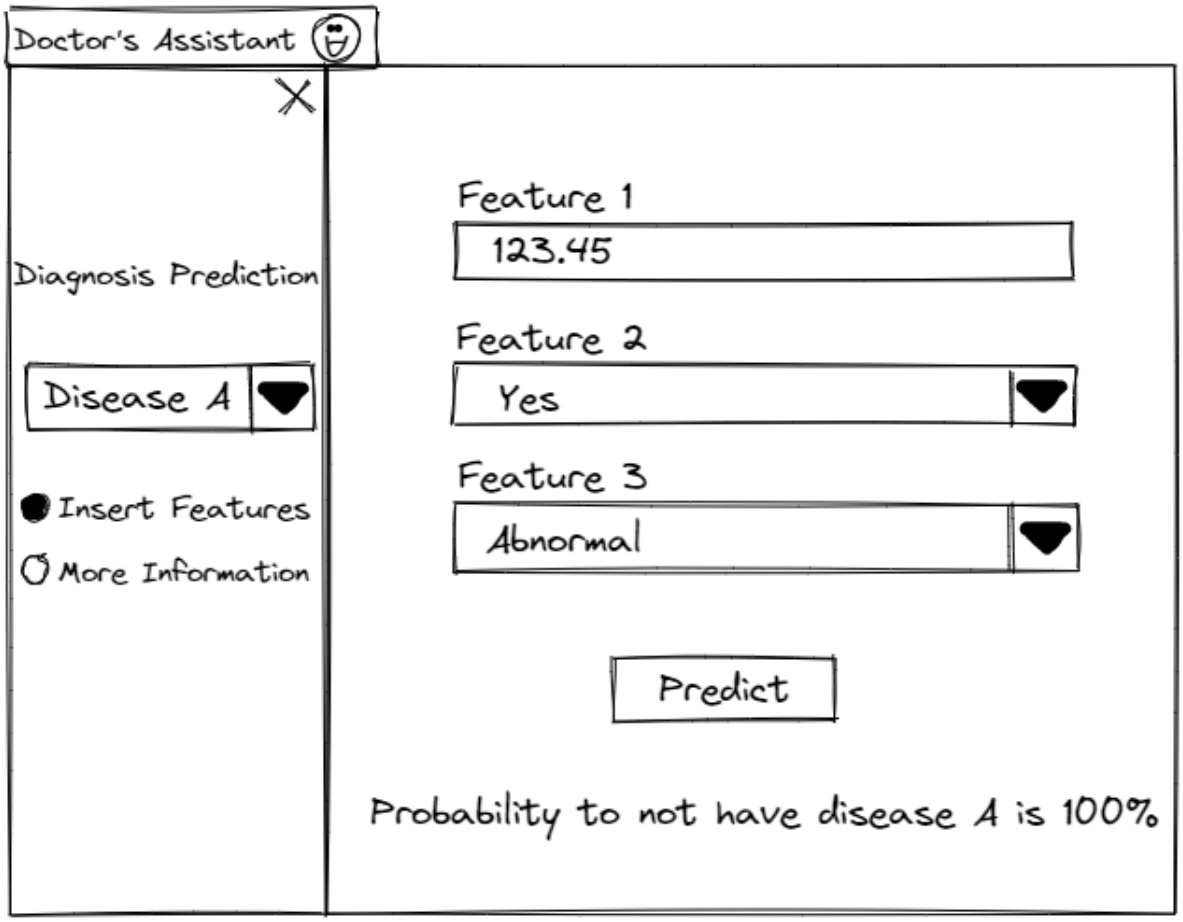


Figure 6.6.1 shows the Insert Features layout

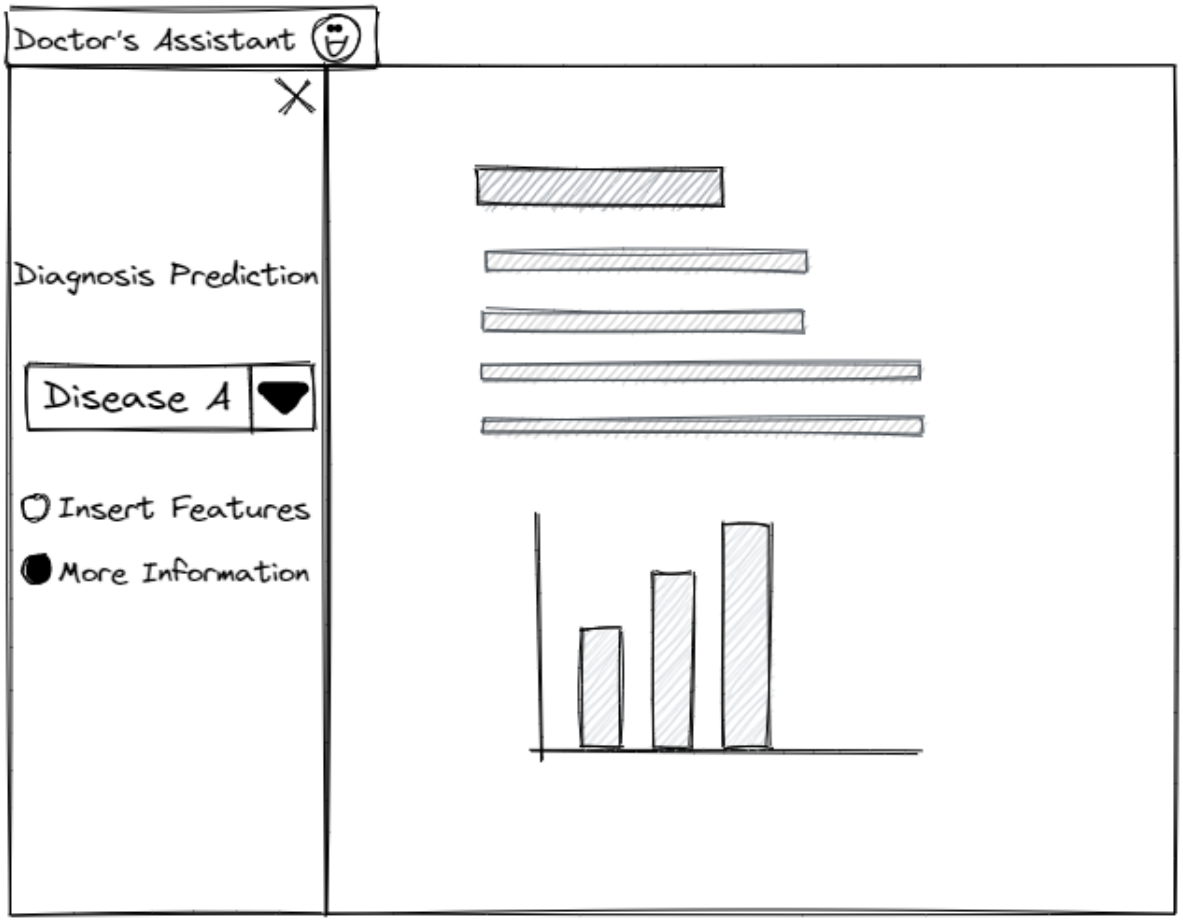


Figure 6.6.2 shows the More Information layout

6.7. Documentation

6.7.1. Dependencies

- Streamlit
- Requests

```
py -m pip install streamlit
py -m pip install requests
```

6.7.2. Run the Code

```
py -m streamlit run main.py
```

6.7.3. Request and Response

6.7.3.1. Request

```
input = {"input_list": [Pregnancies, Glucose, BloodPressure, SkinThickness,
Insulin, BMI, DiabetesPedigreeFunction, Age]}

url = "http://localhost:8000/diabetes_model"

response = requests.post(url, json = input)
```

6.7.3.2. Response

```
response.json()["PredictionClass"]

response.json()["Probability"]
```

6.7.4. Webpage

The webpage will consist of a sidebar that will contain the title, a select box for choosing the disease (See figure 6.7.4.2) and radio buttons to select which kind of action to be done on that chosen disease; Insert Features for predicting diagnosis and More Information to provide more information about the machine learning model used for prediction (See figure 6.7.4.1).

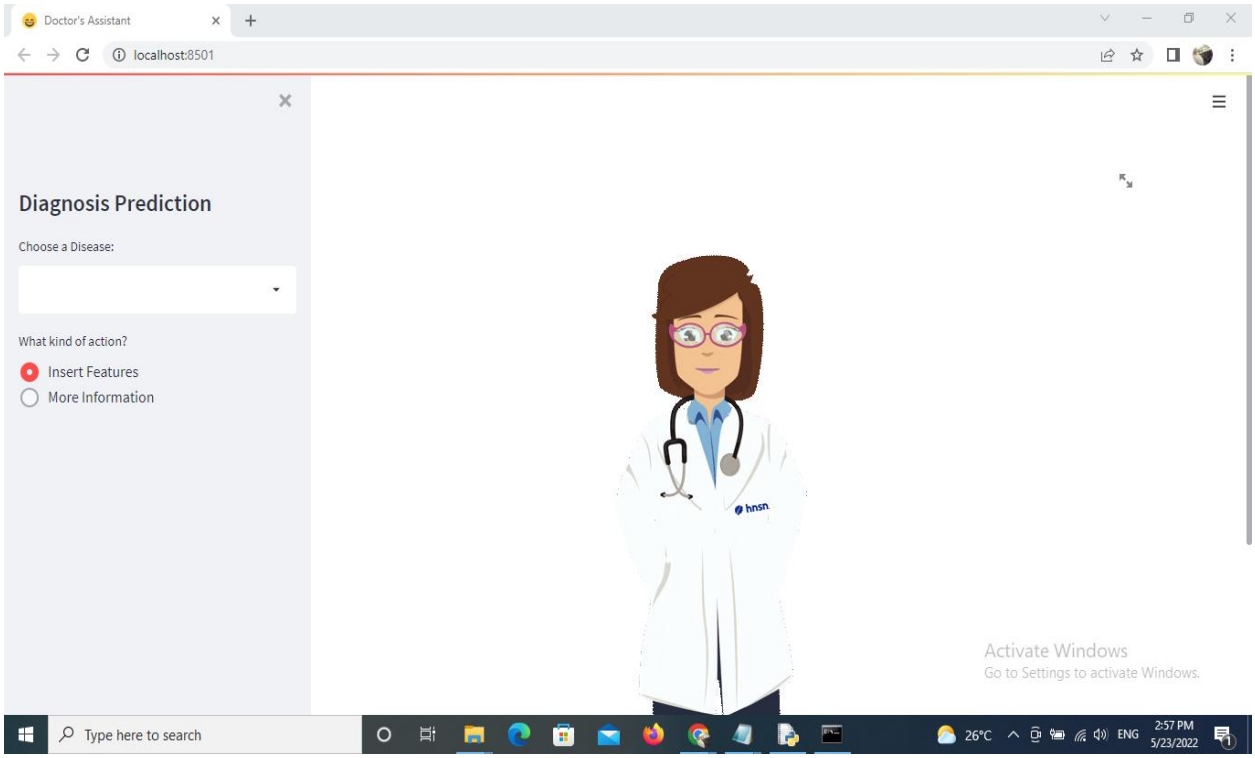


Figure 6.7.4.1 Main Page

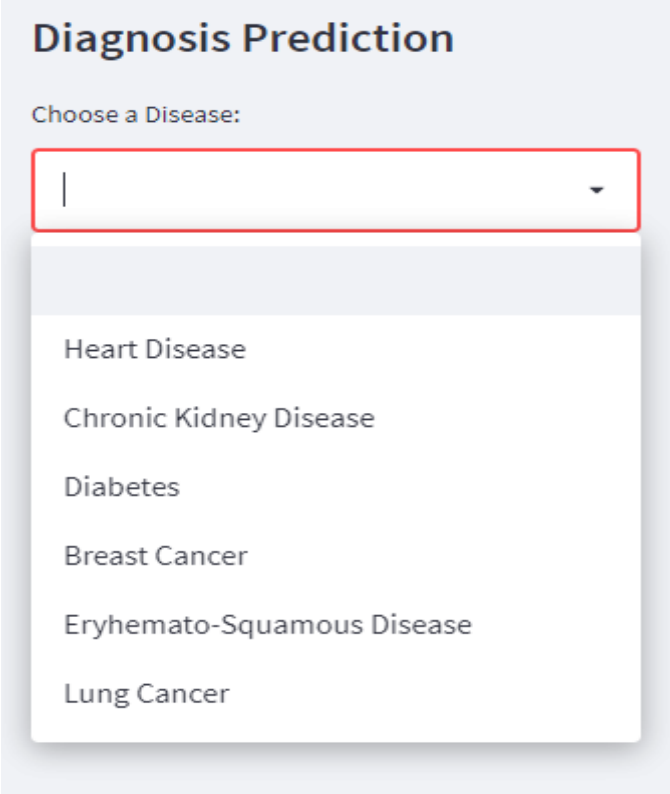


Figure 6.7.4.2 Select Box for Choosing Disease

6.7.5. Input Format for Inserting features

User will either insert numerical features or select a choice of attribute from a select box. The prediction is processed based on the user's input data (See figures 6.7.5.1, 6.7.5.2, 6.7.5.3, 6.7.5.4 and 6.7.5.5).

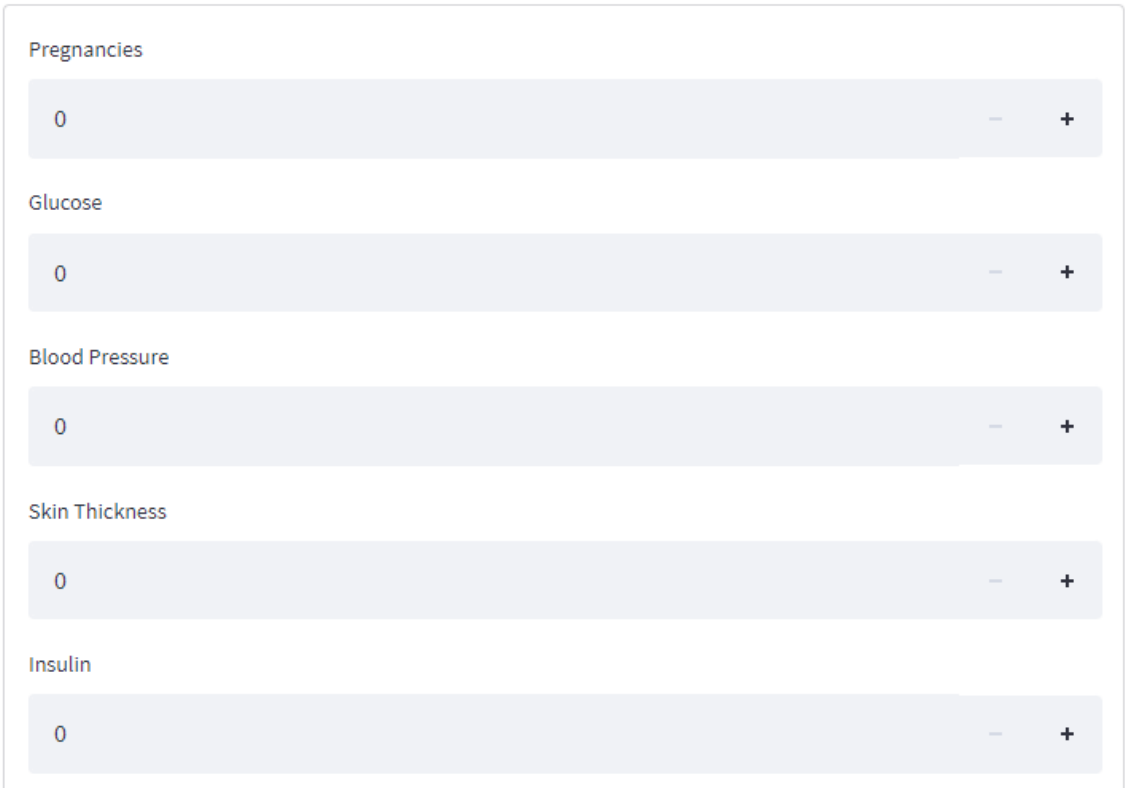


Figure 6.7.5.1 Insert Features for Diabetes

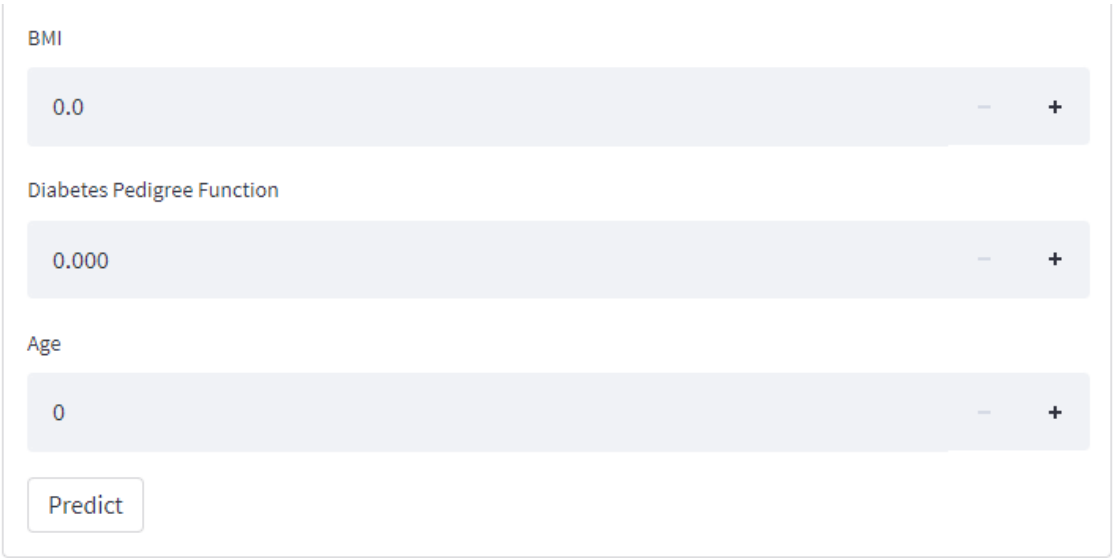


Figure 6.7.5.2 Insert Features for Diabetes continued

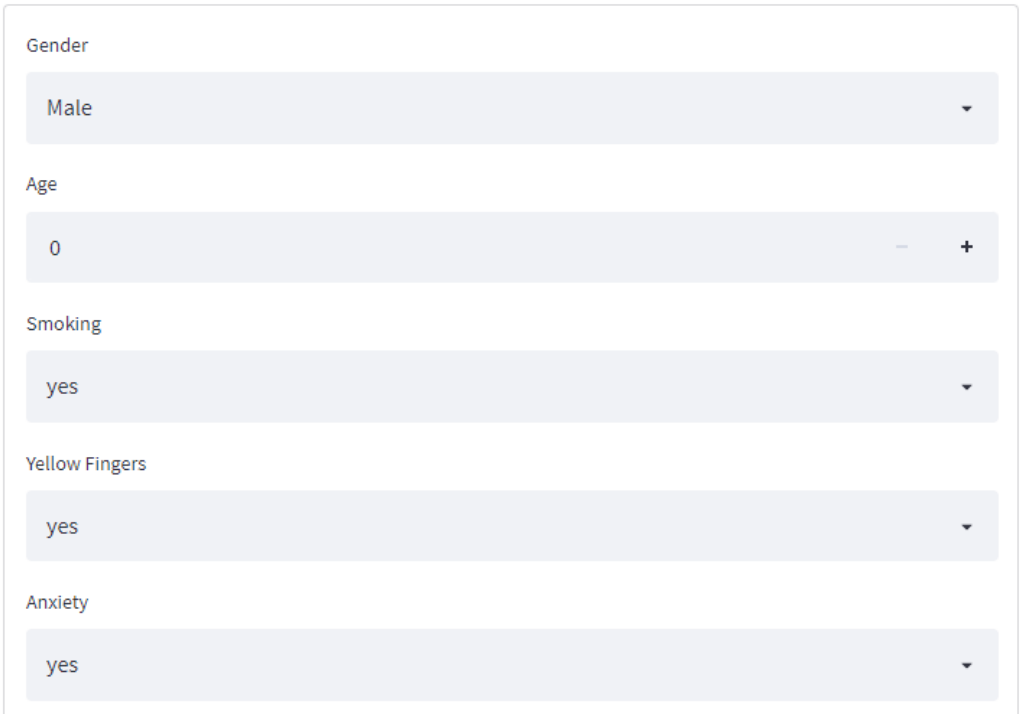


Figure 6.7.5.3 Insert Features for Lung Cancer

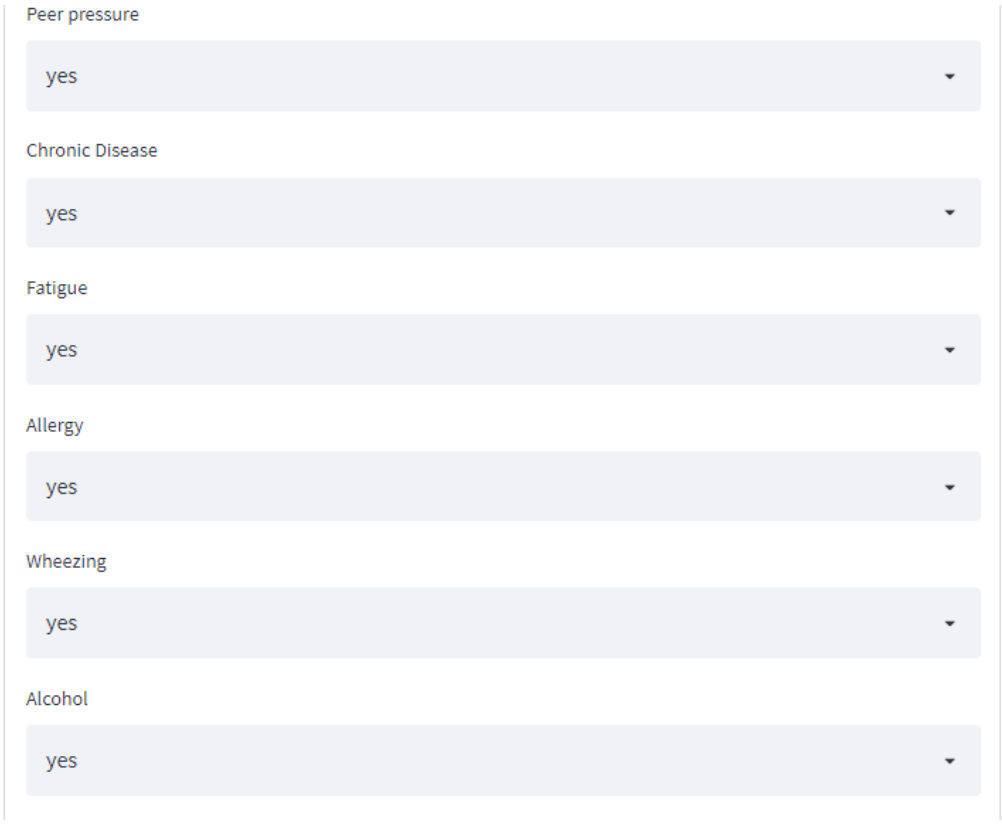


Figure 6.7.5.4 Insert Features for Lung Cancer continued

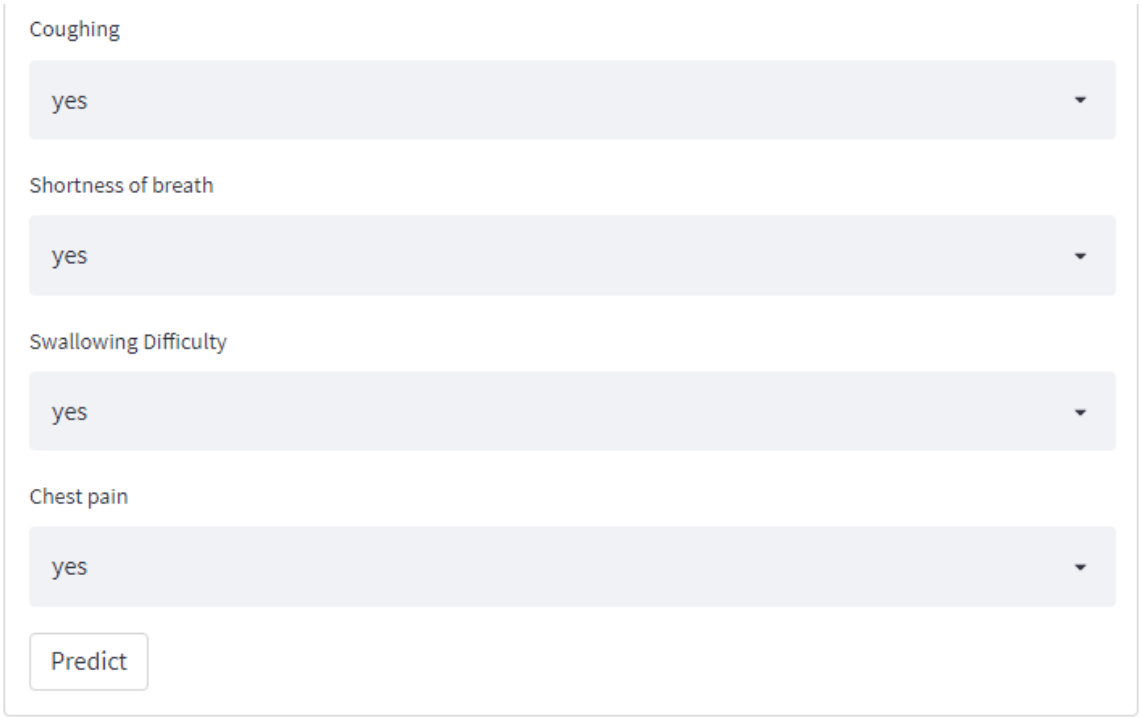


Figure 6.7.5.5 Insert Features for Lung Cancer continued

6.7.6. Output Format for Diagnosis Prediction

After the user has input the features, the user clicks the "Predict" button to receive a successful output prediction of the diagnosis (See figures 6.7.6.1 and 6.7.6.2).

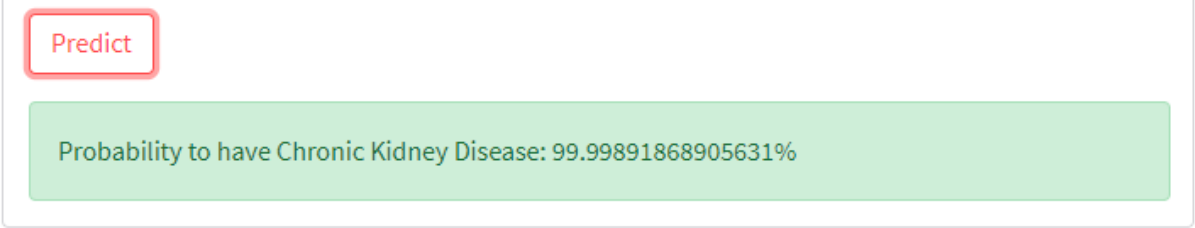


Figure 6.7.6.1 Output prediction for Chronic Kidney Disease

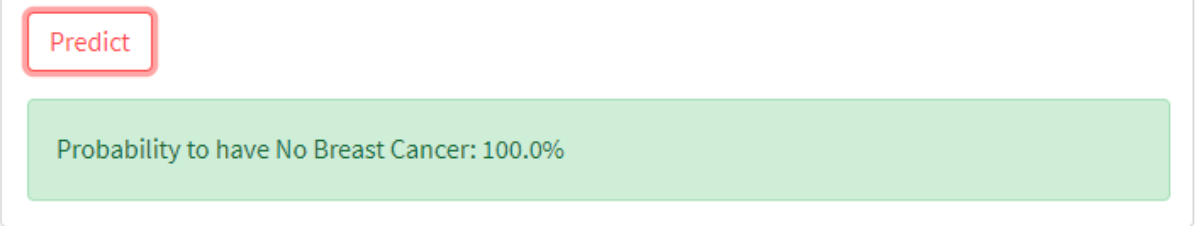


Figure 6.7.6.2 Output prediction for Breast Cancer

6.7.7. More Information of Diseases' Model Format

The user will be able to know more about the machine learning model used for each disease; which includes the accuracy of the model, the model type, feature selection or reduction method used or none, the number of record and attributes, the source of the dataset and some bar charts to visualize the target class distribution in the dataset, as well as, a bar chart to show the several models' accuracies used for comparison (See figures 6.7.7.1 and 6.7.7.2).

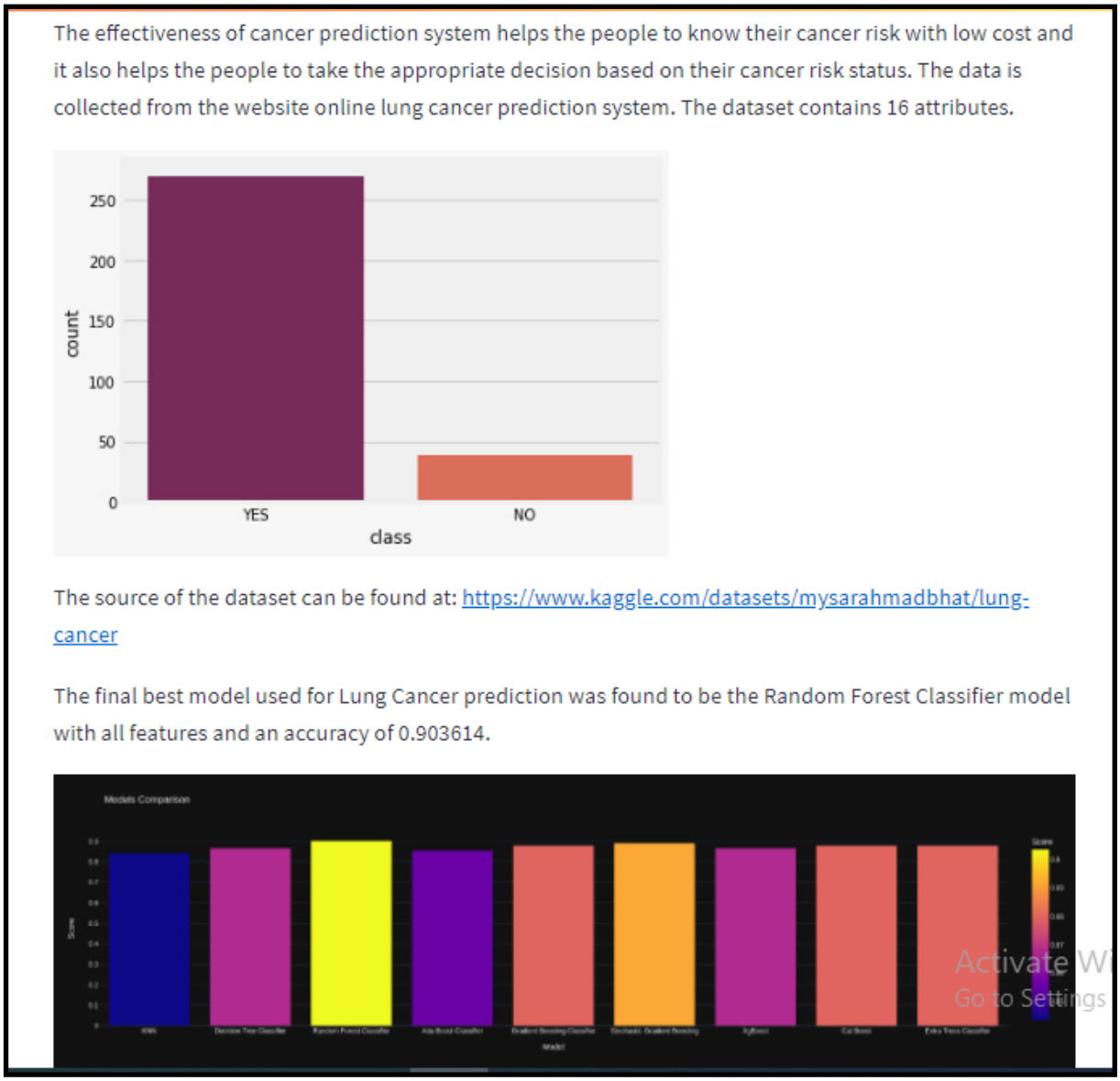
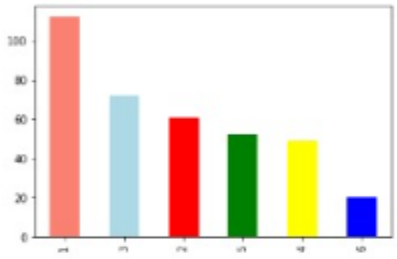


Figure 6.7.7.1 More Information about Lung Cancer model

The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are: 1-psoriasis 2- seboreic dermatitis 3-lichen planus 4- pityriasis rosea 5- cronic dermatitis 6- pityriasis rubra pilaris. The goal of erythemato-squamous predection is to classificate between the previous six diseases.



The source of the dataset can be found at: https://archive.ics.uci.edu/ml/machine-learningdatabases/dermatology/

The final best model used for Eryhemato-Squamous Disease prediction was found to be the Logistic Regression Classifier model with a feature selection method of Pearson Correlation using 17 features selected out of 34 and an accuracy of 0.9090909090909091.

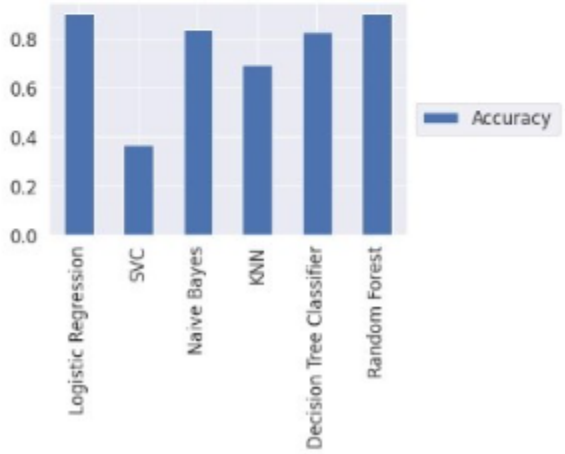


Figure 6.7.7.2 More Information about Erythemato-Squamous Disease model

6.8. Implementation

```
import streamlit as st
import requests

#################################################################
###########

def handle_click_wo_button():
    if st.session_state.kind_of_action:
        st.session_state.type=st.session_state.kind_of_action
    elif st.session_state.disease_chosen:
        st.session_state.type=st.session_state.disease_chosen

#################################################################
###########

def option_chosen(option):
    if option=='Not present':
      return 0
    elif option=='Low intermediate value':
      return 1
    elif option=='High intermediate value':
      return 2
    else:
      return 3
#################################################################
###########

def heart_insert_features():

    form1 = st.form(key='form1')

    Age = form1.number_input('Age', min_value=0, step=1)
    Sex = form1.selectbox('Sex', ['Male', 'Female'])
    ChestPainType = form1.selectbox('Chest Pain Type',
['Typical angina', 'Atypical angina', 'Non-anginal pain',
'Asymptomatic'])
    RestingBP = form1.number_input('Resting BP', min_value=0,
step=1)
    Cholesterol = form1.number_input('Cholesterol',
min_value=0, step=1)
    FastingBS = form1.selectbox('Fasting BS', ['FastingBS >
120 mg/dl', 'otherwise'])
    RestingECG = form1.selectbox('Resting ECG', ['Normal',
'ST', 'LVH'])
```

```
    MaxHR = form1.number_input('Max HR', min_value=0, step=1)
    ExerciseAngina = form1.selectbox('Exercise Angina',
['Yes', 'No'])
    Oldpeak = form1.number_input('Oldpeak', step=0.1,
format="%.1f")
    ST_Slope = form1.selectbox('ST Slope', ['upsloping',
'flat', 'downsloping'])

    if Sex=='Male':
      Sex = 0
    else:
      Sex = 1

    if ChestPainType=='Typical angina':
      ChestPainType = 0
    elif ChestPainType=='Atypical angina':
      ChestPainType = 1
    elif ChestPainType=='Non-anginal pain':
      ChestPainType = 2
    else:
      ChestPainType = 3

    if FastingBS=='FastingBS > 120 mg/dl':
      FastingBS = 1
    else:
      FastingBS = 0

    if RestingECG=='Normal':
      RestingECG = 0
    elif RestingECG=='ST':
      RestingECG = 1
    else:
      RestingECG = 2

    if ExerciseAngina=='Yes':
      ExerciseAngina = 1
    else:
      ExerciseAngina = 0

    if ST_Slope=='upsloping':
      ST_Slope = 0
    elif ST_Slope=='flat':
      ST_Slope = 1
    else:
      ST_Slope = 2

    input = {"input_list": [Age, Sex, ChestPainType,
```

```
RestingBP, Cholesterol, FastingBS, RestingECG, MaxHR,
ExerciseAngina, Oldpeak, ST_Slope]}
    url = "http://localhost:8000/heart_disease_model"

    if form1.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='1':
            form1.success("Probability to have Heart Disease:
" + response.json()["Probability"] + "%")
        else:
            form1.success("Probability to have No Heart
Disease: " + response.json()["Probability"] + "%")


##################################################################
###########


def breast_cancer_insert_features():

    form2 = st.form(key='form2')

    area_mean = form2.number_input('area mean',
min_value=0.0, step=1e-6, format="%.6f")
    concavity_mean = form2.number_input('concavity mean',
min_value=0.0, step=1e-6, format="%.6f")
    concave_points_mean = form2.number_input('concave points
mean', min_value=0.0, step=1e-6, format="%.6f")
    radius_worst = form2.number_input('radius worst',
min_value=0.0, step=1e-6, format="%.6f")
    texture_worst = form2.number_input('texture worst',
min_value=0.0, step=1e-6, format="%.6f")
    perimeter_worst = form2.number_input('perimeter worst',
min_value=0.0, step=1e-6, format="%.6f")
    area_worst = form2.number_input('area worst',
min_value=0.0, step=1e-6, format="%.6f")
    concave_points_worst = form2.number_input('concave points
worst', min_value=0.0, step=1e-6, format="%.6f")

    input = {"input_list": [area_mean, concavity_mean,
concave_points_mean, radius_worst, texture_worst,
perimeter_worst, area_worst, concave_points_worst]}
    url = "http://localhost:8000/breast_cancer_model"

    if form2.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='1':
            form2.success("Probability to have Breast Cancer:
```

```
" + response.json()["Probability"] + "%")
        else:
            form2.success("Probability to have No Breast
Cancer: " + response.json()["Probability"] + "%")

##################################################################
###########

def erythemato_insert_features():

    form3 = st.form(key='form3')

    options_list = ['Not present', 'Low intermediate value',
'High intermediate value', 'Large value']

    erythema = option_chosen(form3.selectbox('erythema',
options_list))
    scaling = option_chosen(form3.selectbox('scaling',
options_list))
    definite_borders =
option_chosen(form3.selectbox('definite borders',
options_list))
    itching = option_chosen(form3.selectbox('itching',
options_list))
    koebner_phenomenon =
option_chosen(form3.selectbox('koebner phenomenon',
options_list))
    polygonal_papules =
option_chosen(form3.selectbox('polygonal papules',
options_list))
    follicular_papules =
option_chosen(form3.selectbox('follicular papules',
options_list))
    knee_and_elbow_involvement =
option_chosen(form3.selectbox('knee and elbow involvement',
options_list))
    family_history = option_chosen(form3.selectbox('family
history', options_list))
    eosinophils_in_the_infiltrate =
option_chosen(form3.selectbox('eosinophils in the
infiltrate', options_list))
    pnl_infiltrate = option_chosen(form3.selectbox('pnl
infiltrate', options_list))
    fibrosis_of_the_papillary_dermis =
option_chosen(form3.selectbox('fibrosis of the papillary
dermis', options_list))
    acanthosis = option_chosen(form3.selectbox('acanthosis',
```

```
options_list))
    hyperkeratosis =
option_chosen(form3.selectbox('hyperkeratosis',
options_list))
    parakeratosis =
option_chosen(form3.selectbox('parakeratosis', options_list))
    inflammatory_monoluclear_infiltrate =
option_chosen(form3.selectbox('inflammatory monoluclear
infiltrate', options_list))
    age = form3.number_input('age', min_value=0, step=1)

    input = {"input_list": [erythema, scaling,
definite_borders, itching, koebner_phenomenon,
polygonal_papules, follicular_papules,
knee_and_elbow_involvement, family_history,
eosinophils_in_the_infiltrate, pnl_infiltrate,
fibrosis_of_the_papillary_dermis, acanthosis, hyperkeratosis,
parakeratosis, inflammatory_monoluclear_infiltrate, age]}
    url = "http://localhost:8000/eryhemato_model"

    if form3.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='1':
            form3.success("Probability to have Psoriasis: " +
response.json()["Probability"] + "%")
        elif response.json()["PredictionClass"]=='2':
            form3.success("Probability to have Seboreic
Dermatitis: " + response.json()["Probability"] + "%")
        elif response.json()["PredictionClass"]=='3':
            form3.success("Probability to have Lichen Planus:
" + response.json()["Probability"] + "%")
        elif response.json()["PredictionClass"]=='4':
            form3.success("Probability to have Pityriasis
Rosea: " + response.json()["Probability"] + "%")
        elif response.json()["PredictionClass"]=='5':
            form3.success("Probability to have Chronic
Dermatitis: " + response.json()["Probability"] + "%")
        else:
            form3.success("Probability to have Pityriasis
Rubra Pilaris: " + response.json()["Probability"] + "%")

##################################################################
############

def kidney_insert_features():

    form4 = st.form(key='form4')
```

```
    age = form4.number_input('age', min_value=0, step=1)
    blood_pressure = form4.number_input('blood pressure',
min_value=0, step=1)
    specific_gravity = form4.number_input('specific gravity',
min_value=0.0, step=0.001, format="%.3f")
    albumin = form4.number_input('albumin', min_value=0.0,
step=0.1, format="%.1f")
    sugar = form4.number_input('sugar', min_value=0, step=1)
    red_blood_cells = form4.selectbox('red blood cells',
['normal', 'abnormal'])
    pus_cell = form4.selectbox('pus cell', ['normal',
'abnormal'])
    pus_cell_clumps = form4.selectbox('pus cell clumps',
['notpresent', 'present'])
    bacteria = form4.selectbox('bacteria', ['notpresent',
'present'])
    blood_glucose_random = form4.number_input('blood glucose
random', min_value=0, step=1)
    blood_urea = form4.number_input('blood urea',
min_value=0.0, step=0.1, format="%.1f")
    serum_creatinine = form4.number_input('serum creatinine',
min_value=0.0, step=0.01, format="%.2f")
    sodium = form4.number_input('sodium', min_value=0.0,
step=0.1, format="%.1f")
    potassium = form4.number_input('potassium',
min_value=0.0, step=0.1, format="%.1f")
    haemoglobin = form4.number_input('haemoglobin',
min_value=0.0, step=0.1, format="%.1f")
    packed_cell_volume = form4.number_input('packed cell
volume', min_value=0.0, step=0.1, format="%.1f")
    white_blood_cell_count = form4.number_input('white blood
cell count', min_value=0.0, step=0.1, format="%.1f")
    red_blood_cell_count = form4.number_input('red blood cell
count', min_value=0.0, step=0.1, format="%.1f")
    hypertension = form4.selectbox('hypertension', ['yes',
'no'])
    diabetes_mellitus = form4.selectbox('diabetes mellitus',
['yes', 'no'])
    coronary_artery_disease = form4.selectbox('coronary
artery disease', ['yes', 'no'])
    appetite = form4.selectbox('appetite', ['good', 'poor'])
    peda_edema = form4.selectbox('peda edema', ['yes', 'no'])
    aanemia = form4.selectbox('aanemia', ['yes', 'no'])

    if red_blood_cells=='normal':
      red_blood_cells = 1
```

```
else:
  red_blood_cells = 0

if pus_cell=='normal':
  pus_cell = 1
else:
  pus_cell = 0

if pus_cell_clumps=='notpresent':
  pus_cell_clumps = 0
else:
  pus_cell_clumps = 1

if bacteria=='notpresent':
  bacteria = 0
else:
  bacteria = 1

if hypertension=='yes':
  hypertension = 1
else:
  hypertension = 0

if diabetes_mellitus=='yes':
  diabetes_mellitus = 1
else:
  diabetes_mellitus = 0

if coronary_artery_disease=='yes':
  coronary_artery_disease = 1
else:
  coronary_artery_disease = 0

if appetite=='good':
  appetite = 0
else:
  appetite = 1

if peda_edema=='yes':
  peda_edema = 1
else:
  peda_edema = 0

if aanemia=='yes':
  aanemia = 1
else:
  aanemia = 0
```

```
    input = {"input_list": [age, blood_pressure,
specific_gravity, albumin, sugar, red_blood_cells, pus_cell,
pus_cell_clumps, bacteria, blood_glucose_random, blood_urea,
serum_creatinine, sodium, potassium, haemoglobin,
packed_cell_volume, white_blood_cell_count,
red_blood_cell_count, hypertension, diabetes_mellitus,
coronary_artery_disease, appetite, peda_edema, aanemia]}
    url = "http://localhost:8000/kidney_model"

    if form4.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='0':
            form4.success("Probability to have Chronic Kidney
Disease: " + response.json()["Probability"] + "%")
        else:
            form4.success("Probability to have No Chronic
Kidney Disease: " + response.json()["Probability"] + "%")

##################################################################
###########

def diabetes_insert_features():

    form5 = st.form(key='form5')

    Pregnancies = form5.number_input('Pregnancies',
min_value=0, step=1)
    Glucose = form5.number_input('Glucose', min_value=0,
step=1)
    BloodPressure = form5.number_input('Blood Pressure',
min_value=0, step=1)
    SkinThickness = form5.number_input('Skin Thickness',
min_value=0, step=1)
    Insulin = form5.number_input('Insulin', min_value=0,
step=1)
    BMI = form5.number_input('BMI', min_value=0.0, step=0.1,
format="%.1f")
    DiabetesPedigreeFunction = form5.number_input('Diabetes
Pedigree Function', min_value=0.0, step=0.001, format="%.3f")
    Age = form5.number_input('Age', min_value=0, step=1)

    input = {"input_list": [Pregnancies, Glucose,
BloodPressure, SkinThickness, Insulin, BMI,
DiabetesPedigreeFunction, Age]}
    url = "http://localhost:8000/diabetes_model"
```

```
    if form5.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='1':
            form5.success("Probability to have Diabetes: " +
response.json()["Probability"] + "%")
        else:
            form5.success("Probability to have No Diabetes: "
+ response.json()["Probability"] + "%")

##################################################################
###########

def lung_cancer_insert_features():

    form6 = st.form(key='form6')

    Gender = form6.selectbox('Gender', ['Male', 'Female'])
    Age = form6.number_input('Age', min_value=0, step=1)
    Smoking = form6.selectbox('Smoking', ['yes', 'no'])
    YellowFingers = form6.selectbox('Yellow Fingers', ['yes',
'no'])
    Anxiety = form6.selectbox('Anxiety', ['yes', 'no'])
    Peerpressure  = form6.selectbox('Peer pressure', ['yes',
'no'])
    ChronicDisease  = form6.selectbox('Chronic Disease',
['yes', 'no'])
    Fatigue = form6.selectbox('Fatigue', ['yes', 'no'])
    Allergy = form6.selectbox('Allergy', ['yes', 'no'])
    Wheezing = form6.selectbox('Wheezing', ['yes', 'no'])
    Alcohol = form6.selectbox('Alcohol', ['yes', 'no'])
    Coughing = form6.selectbox('Coughing', ['yes', 'no'])
    Shortnessofbreath = form6.selectbox('Shortness of
breath', ['yes', 'no'])
    SwallowingDifficulty = form6.selectbox('Swallowing
Difficulty', ['yes', 'no'])
    Chestpain = form6.selectbox('Chest pain', ['yes', 'no'])

    if Gender=='Male':
      Gender = 1
    else:
      Gender = 0

    if Smoking=='yes':
      Smoking = 1
    else:
      Smoking = 0
```

```
if Anxiety=='yes':
  Anxiety = 1
else:
  Anxiety = 0

if YellowFingers=='yes':
  YellowFingers = 1
else:
  YellowFingers = 0

if Peerpressure=='yes':
  Peerpressure = 1
else:
  Peerpressure = 0

if ChronicDisease=='yes':
  ChronicDisease = 1
else:
  ChronicDisease = 0

if Fatigue=='yes':
  Fatigue = 1
else:
  Fatigue = 0

if Allergy=='yes':
  Allergy = 1
else:
  Allergy = 0

if Wheezing=='yes':
  Wheezing = 1
else:
  Wheezing = 0

if Alcohol=='yes':
  Alcohol = 1
else:
  Alcohol = 0

if Coughing=='yes':
  Coughing = 1
else:
  Coughing = 0

if Shortnessofbreath =='yes':
  Shortnessofbreath = 1
```

```
    else:
      Shortnessofbreath = 0

    if SwallowingDifficulty=='yes':
      SwallowingDifficulty = 1
    else:
      SwallowingDifficulty = 0

    if Chestpain=='yes':
      Chestpain = 1
    else:
      Chestpain = 0

    input = {"input_list": [Gender, Age, Smoking,
YellowFingers, Anxiety, Peerpressure , ChronicDisease,
Fatigue, Allergy, Wheezing, Alcohol, Coughing,
Shortnessofbreath, SwallowingDifficulty, Chestpain]}
    url = "http://localhost:8000/lung_cancer_model"

    if form6.form_submit_button('Predict'):
        response = requests.post(url, json = input)
        if response.json()["PredictionClass"]=='1':
            form6.success("Probability to have Lung Cancer: "
+ response.json()["Probability"] + "%")
        else:
            form6.success("Probability to have No Lung
Cancer: " + response.json()["Probability"] + "%")

################################################################
############

def main():

    st.set_page_config(page_title="Doctor's Assistant",
page_icon=":smile:")
    st.sidebar.title("Diagnosis Prediction")
    diseases = ['', 'Heart Disease', 'Chronic Kidney
Disease', 'Diabetes', 'Breast Cancer', 'Erythemato-Squamous
Disease', 'Lung Cancer']
    st.session_state['option'] = st.sidebar.selectbox('Choose
a Disease:', diseases, on_change=handle_click_wo_button,
key='disease_chosen')
    st.session_state['kind'] = st.sidebar.radio('What kind of
action?',['Insert Features', 'More Information'],
on_change=handle_click_wo_button, key='kind_of_action')

    if st.session_state['kind']=='Insert Features':
```

```
        if st.session_state['option']=='Heart Disease':
            heart_insert_features()

        elif st.session_state['option']=='Breast Cancer':
            breast_cancer_insert_features()

        elif st.session_state['option']=='Erythemato-Squamous
Disease':
            erythemato_insert_features()

        elif st.session_state['option']=='Chronic Kidney
Disease':
            kidney_insert_features()

        elif st.session_state['option']=='Diabetes':
            diabetes_insert_features()

        elif st.session_state['option']=='Lung Cancer':
            lung_cancer_insert_features()

        else:
            st.image('doctor.gif')

    else:

        if st.session_state['option']=='Heart Disease':
            st.write('A typical goal is to classify the input
into one of two states, heart disease or normal according to
eleven user-provided features. This dataset was created by
combining different datasets already available independently
but not combined before. In this dataset, 5 heart datasets
are combined over 11 common features which makes it the
largest heart disease dataset available so far for research
purposes.')
            st.image('heart1.png')
            st.write('The source of the dataset can be found
at: https://archive.ics.uci.edu/ml/machine-
learningdatabases/heart-disease/')
            st.write('The final best model used for Heart
Disease prediction was found to be the Random Forest
Classifier model with all features and an accuracy of 0.913')
            st.image('heart2.png')

        elif st.session_state['option']=='Breast Cancer':
            st.write('The Wisconsin Diagnostic Breast Cancer
dataset is a real-valued multivariate data that consists of
```

```
two classes, where each class signifies whether a patient has
breast cancer or not. The two categories are: malignant and
benign. Each record represents data for one breast cancer
case. Features are computed from a digitized image of a fine
needle aspirate (FNA) of a breast mass. They describe
characteristics of the cell nuclei present in the image.')
            st.image('breastcancer1.png')
            st.write('The source of the dataset can be found
at:
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wiscons
in+(Diagnostic)')
            st.write('The final best model used for Breast
Cancer prediction was found to be the Random Forest
Classifier model with a feature selection method of Recursive
Feature Elimination using 8 features selected out of 30 and
an accuracy of 0.9736842105263158.')
            st.image('breastcancer2.png')

        elif st.session_state['option']=='Erythemato-Squamous
Disease':
            st.write('The differential diagnosis of
erythemato-squamous diseases is a real problem in
dermatology. They all share the clinical features of erythema
and scaling, with very little differences. The diseases in
this group are: 1-psoriasis 2- seboreic dermatitis 3-lichen
planus 4- pityriasis rosea 5- cronic dermatitis 6- pityriasis
rubra pilaris. The goal of erythemato-squamous predection is
to classificate between the previous six diseases.')
            st.image('eryhemato1.jpg')
            st.write('The source of the dataset can be found
at: https://archive.ics.uci.edu/ml/machine-
learningdatabases/dermatology/')
            st.write('The final best model used for
Erythemato-Squamous Disease prediction was found to be the
Logistic Regression Classifier model with a feature selection
method of Pearson Correlation using 17 features selected out
of 34 and an accuracy of 0.9090909090909091.')
            st.image('eryhemato2.jpg')

        elif st.session_state['option']=='Chronic Kidney
Disease':
            st.write('The data was taken over a 2-month
period in India with 25 features (eg, red blood cell count,
white blood cell count, etc). The target is the
"classification", which is either "ckd" or "notckd" -
ckd=chronic kidney disease. There are 400 rows in the
dataset.')
```

```
            st.image('kidney1.jpg')
            st.write('0: Chronic Kidney Disease       1: No
Chronic Kidney Disease')
            st.write('The source of the dataset can be found
at:
https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Diseas
e')
            st.write('The final best model used for Chronic
Kidney Disease prediction was found to be the Gradient Boost
Classifier model with all features and an accuracy of
0.99166.')
            st.image('kidney2.png')

        elif st.session_state['option']=='Diabetes':
            st.write('The data was collected and made
available by "National Institute of Diabetes and Digestive
and Kidney Diseases" as part of the Pima Indians Diabetes
Database. Several constraints were placed on the selection of
these instances from a larger database. In particular, all
patients here belong to the Pima Indian heritage (subgroup of
Native Americans), and are females of ages 21 and above.')
            st.image('diabetes1.jpg')
            st.write('0: No Diabetes       1: Diabetes')
            st.write('The source of the dataset can be found
at: https://www.kaggle.com/datasets/kandij/diabetes-dataset')
            st.write('The final best model used for Diabetes
prediction was found to be the Random forest Classifier model
with all features and an accuracy of 0.8226950.')
            st.image('diabetes2.jpg')

        elif st.session_state['option']=='Lung Cancer':
            st.write('The effectiveness of cancer prediction
system helps the people to know their cancer risk with low
cost and it also helps the people to take the appropriate
decision based on their cancer risk status. The data is
collected from the website online lung cancer prediction
system. The dataset contains 16 attributes.')
            st.image('lung1.jpg')
            st.write('The source of the dataset can be found
at: https://www.kaggle.com/datasets/mysarahmadbhat/lung-
cancer')
            st.write('The final best model used for Lung
Cancer prediction was found to be the Random Forest
Classifier model with all features and an accuracy of
0.903614.')
            st.image('lung2.jpg')
```

```
        else:
            st.image('doctor.gif')

##################################################################
###########

if __name__ == '__main__':
    main()
```

6.9. Summary

In this section we explained what a frontend is and what is its role in our web development environment. Furthermore, we explained what are the typical frontend technologies that developers use and mentioned some of the machine learning and data science frontend frameworks or tools to build a frontend. We finally presented the choice of framework, design, documentation, and implementation for our frontend.

7. Back-end

7.1. Definition

In Software Engineering, the term back-end is always used to describe an entity that receives requests from clients (Browsers or Software Applications) to process the requests and may communicate with the database and then forms the response and sends it back to the client.

There are different types of architecture in Software Engineering, but the most common one is having a three-tier system, which consists of three parts:

- **Client:** Browsers and Mobile Applications are examples of clients, also referred to as Front-end
- **Backend**
- **Database Server**

7.2. Role of the Back-end

As we already stated, our solution focuses mainly on providing an API for developers to be able to consume our models by sending to our API requests that contain data of the patient to be predicted by our machine learning models, the API communicates with the machine learning models by sending the data given in the message (request) sent by the client, then the model takes that input and predicts if the patient tests positive or negative and send back the results to the client to be displayed as shown in figure 7.2.1.

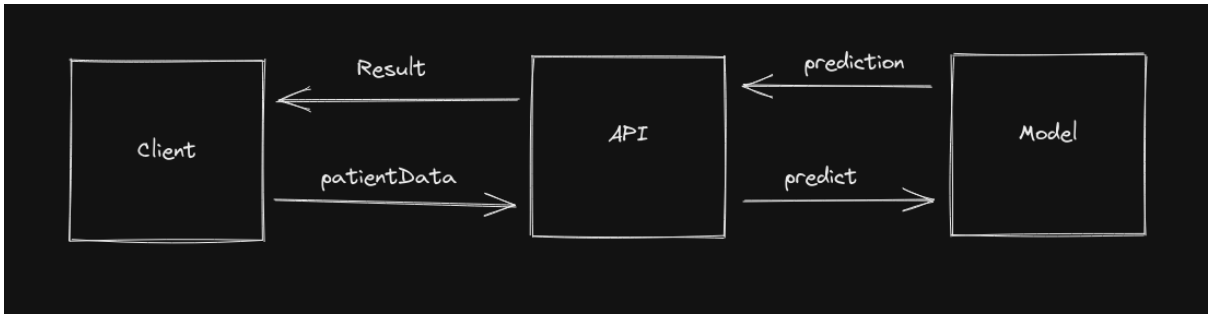


Figure 7.2.1 Simple Solution Architecture

The possible clients are:

- Mobile Applications
- Desktop Applications
- Web Applications

The developer can communicate with the API by sending a request in a format that will be discussed in section 1.3, then receives a response that has the prediction output to use on their client-side.

7.3. Documentation

7.3.1. Dependencies

- FastAPI
- uvicorn
- sklearn
- scikit-learn
- numpy

```
pip3 install "fastapi[all]" "uvicorn[standard]" sklearn scikit-learn numpy
```

7.3.2. Run the Code

```
uvicorn main:app --reload
```

7.3.3. End Points

We have 2 endpoints, one dummy `('/')` and one for serving our models `('/{model}')` we're using 6 different models to predict 6 different diseases

```
breast_cancer_model
```

```
heart_disease_model
```

```
eryhemato_model
```

```
kidney_model
```

```
lung_cancer_model
```

```
diabetes_model
```

7.3.4. Prediction

To predict a disease send the symptoms to the backend as a post request body to `http://localhost:8000/{model}` after deployment, the `localhost` will be replaced by the `baseurl`

7.3.5. Request Body

the symptoms are required as a list of floats.

```
url = "http://localhost:8000/lung_cancer_model"
```

```
input = {"input_list": [1, 55, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]}
```

7.3.6. Response

the server will respond to the post request with the prediction probability, and it's class.

```
{
  "Probability": "78.0375457875458",
  "PredictionClass": "0"
}
```

7.3.7. Input Format

The input format is mentioned in Frontend Section 6.7.5

7.4. Frameworks

Currently, in the Software Industry there is plenty of back-end frameworks, in this section, we discuss the differences between them, the pros and cons of each framework, and in the next section, we state why we chose FastAPI in our solution.

7.4.1. NodeJS

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. Node.js lets developers use JavaScript to write command-line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

The Node.js distributed development project was previously governed by the Node.js Foundation and has now merged with the JS Foundation to form the OpenJS Foundation, which is facilitated by the Linux Foundation's Collaborative Projects program.

Corporate users of Node.js software include GoDaddy, Groupon, IBM, LinkedIn, Microsoft, Netflix, PayPal, Rakuten, SAP, Walmart, Yahoo!, and Amazon Web Services.

Pros	Cons
1. Asynchronous event-driven IO helps concurrent request handling.	1. Node.js doesn't provide scalability. One CPU is not going to be enough; the platform provides no ability to scale out to take advantage of the multiple cores commonly present in today's server-class hardware.
2. Uses JavaScript, which is easy to learn.	2. Dealing with a relational database is a pain if you are using Node.
3. Share the same piece of code with both the server and client-side.	3. Every time using a callback end up with tons of nested callbacks.
4. npm, the Node packaged module has already become huge and still growing.	4. Without diving into depth into JavaScript if someone starts Node, he may face a conceptual problem.
5. Active and vibrant community, with lots of code, shared via Github, etc.	5. Node.js is not suited for CPU-intensive tasks. It is suited for I/O stuff only (like web servers).
6. You can stream big files.	

7.4.2. Django

Django (/ˈdʒæŋgoʊ/ JANG-goh; sometimes stylized as django) is a Python-based web framework, free and open-source, that follows the model–template–views (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established in the US as a 501(c)(3) non-profit.

Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

Some well-known sites that use Django include Instagram, Mozilla, Disqus, Bitbucket, Nextdoor, and Clubhouse.

Benefits of Django Framework

Django alone has lots of advantages but we will be looking at some major ones which also differentiate Django from other frameworks.

- Implemented in Python

If you have learned Python, or have seen the Python code, you will also observe that Python is very easy to read, and that's the main design philosophy behind the creation of Python.

Also, Python is quite powerful and is used in the implementation of scientific computations and high-level Artificial Intelligence.

So, what's not to like about Python and the Django framework is implemented using the same language, so which gives it immense support in the backend while compromising nothing in the front-end.

- Better CDN connectivity and Content Management

This feature is one of the reasons that emerging companies and social services like *Instagram are using Django in the first place*, due to the availability of more control over CDNs.

CDNs are nothing but Content Delivery Networks, as of the name, they are special servers having the multimedia and content that sites such as *Netflix and Amazon Prime use for streaming*. These servers only contain the multimedia and resources for your webpage, as they are geographically located near the client, so they can serve the content more rapidly and thus, increasing client satisfaction.

Django provides libraries and developers to use it as a CMS (Content Management System) due to its great Admin Interface, which makes it rather easy to set up and run CDNs.

Sites highly use CDNs that deliver multimedia content, like YouTube, Instagram, Google, etc.

- Batteries Included Framework

Django is made by web developers for web developers, so of course, it will resolve the general issues and problems that developers face.

Django Framework, comes with so much functionality, you may not even need to create anything other than your unique application, and that's what Django's design philosophy is DRY (Don't Repeat Yourself).

The *Django developers have done all the boring parts of web development themselves so, you get the fun part*.

- Fast Processing

This advantage is decent over other frameworks as Django's Architecture is different from all other frameworks in the industry.

It means Django uses the MTV architecture which makes the whole process of transmitting over the Internet easier and faster as the resources can be put on a CDN. Django server handles things pretty well, while also maintaining the Speed.

The Django Architecture provides substantial differences from other frameworks.

- Offers Rapid development

The reason to fast development speed is that Django's MTV Architecture implements the *philosophy of loosely coupled components*. It means that we can work on different components parallelly and then can integrate much more easily.

This feature of Django makes a great difference from other frameworks and currently, Django is the best for rapid development in Industry.

- Scalable

Django has been made in such a way that it will be able to handle any kind of hardware additions. This advantage is the main reason why the busiest sites of the world like Instagram, Pinterest, Disqus, etc use the Django framework.

Django is based on loosely coupled architecture, which provides it the functionality to add hardware at any point of components as they will manage that change. And, will have little to no effect on other components, which is seriously not the case in other frameworks.

- Security

Django framework is made by the world's best web developers who have great experience and knowledge. So, that leaves a very small possibility of security loopholes in Django, even in the user authentication system.

Like Laravel Framework in PHP transmits data via GET Method and even the passwords which we enter are visible which is a very high risk, but Django takes care of this in-built. It also uses the Get Method to transmit the data but the passwords and all the important information are automatically encrypted with a long security key. Even in the Django database, we cannot see the password.

Limitations of Django Framework

Django has all the functionality you may ever need on the web. But still, every technology has some disadvantages and Django is no exception to that. Although these are not on a performance basis rather, they relate to design.

- Django is Monolithic

Well, this is also a feature for some but for some, it is a drawback. Django has a certain set of files and pre-defined variables. And, you need to learn about those before you create any project through Django.

Django framework has a certain way to define and perform tasks. It is a logical file structure and easy to learn. But, that also makes it mandatory that you can't use your file structure. It is because the framework has a way, popularly known as "The Django way" of doing things. If you don't follow those rules, you may not be able to deploy anything using Django.

Django framework and server look for information in these files and will not change that.

Therefore, lots of developers prefer flask to Django, but for that framework, you need to know, the backend quite well. Also, you need a lot of time and knowledge to make your structures and server patterns which are not being done in Django. Here, you only need to spend more time on your unique project rather than other things.

- Not for smaller projects

All the functionality of Django comes with lots of code. It takes the server's processing and time, which poses some issues for low-end websites which can run on even very little bandwidth.

Also, Django is scalable and makes the developer's work easy. It means that Django has to provide unique functions and features otherwise what's the difference between Django and other frameworks.

7.4.3. Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies, and several common framework-related tools.

Applications that use the Flask framework include Pinterest and LinkedIn.

Advantages of Flask

- Scalable

Size is everything, and Flask's status as a microframework means that you can use it to grow a tech project such as a web app incredibly quickly. If you want to make an app that starts small, but has the potential to grow quickly and in directions you haven't completely worked out yet, then it's an ideal choice. Its simplicity of use and few dependencies enable it to run smoothly even as it scales up and up.

- Flexible

This is the core feature of Flask, and one of its biggest advantages. To paraphrase one of the principles of the Zen of Python, simplicity is better than complexity, because it can be easily rearranged and moved around.

Not only is this helpful in terms of allowing your project to move in another direction easily, but it also makes sure that the structure won't collapse when a part is altered. The minimal nature of Flask and its aptitude for developing smaller web apps means that it's even more flexible than Django itself.

- Easy to negotiate

Like Django, being able to find your way around easily is key for allowing web developers to concentrate on just coding quickly, without getting bogged down. At its core, the microframework is easy to understand for web developers, not just saving them time and effort but also giving them more control over their code and what is possible.

- Lightweight

When we use this term about a tool or framework, we're talking about its design of it—a few constituent parts need to be assembled and reassembled, and it doesn't rely on a large number of extensions to function. This design gives web developers a certain level of control.

Flask also supports modular programming, which is where its functionality can be split into several interchangeable modules. Each module acts as an independent building block, which can execute one part of the functionality. Together this means that the whole constituent parts of the structure are flexible, moveable, and testable on their own.

- Documentation

Following the creator's theory that "nice documentation design makes you write documentation," Flask users will find a healthy number of examples and tips arranged in a structured manner. This encourages developers to use the framework, as they can

easily get introduced to the different aspects and capabilities of the tool. You'll find the Flask documentation on their official website.

Disadvantages of Flask

- Not a lot of tools

Inevitably there are some downsides to this microframework's lightweight nature. Chief among them is that, unlike Django, Flask lacks a large toolbox. This means that developers will have to manually add extensions such as libraries. And, if you add a huge number of extensions, it may start to slow down the app itself due to a multitude of requests.

- Difficult to get familiar with a larger Flask app

Because the fact that development of a web app using Flask can take a variety of twists and turns, a web developer arriving at the project mid-way through can struggle to come to terms with how it's been designed. The modular nature of the microframework that we mentioned earlier can come back to haunt coders, who will have to familiarize themselves with each constituent part.

- Maintenance costs

Because it is so versatile in terms of which technologies it can interface with, quite often a company using Flask will incur extra costs of supporting those technologies. For example, if a technology interfacing with your Flask app becomes obsolete or is discontinued, then the company will have to scramble to find a new compatible one. The more complicated the app becomes, the higher the potential maintenance and implementation costs.

7.4.4. FastAPI

FastAPI is a Web framework for developing RESTful APIs in Python. FastAPI is based on Pydantic and type hints to validate, serialize, and deserialize data, and automatically auto-generate OpenAPI documents.

It fully supports asynchronous programming and can run with Uvicorn and Gunicorn. To improve developer-friendliness, editor support was considered from the earliest days of the project.

Advantages of FastAPI

FastAPI has a high performance, concurrency can be easily supported, and offers a simple and easy-to-use dependency injection system. Inbuilt data validation is another benefit to take into consideration.

- Excellent performance

If we were to name one quality by which FastAPI beats Flask, it's the performance. FastAPI is known as one of the fastest Python web frameworks. Only Starlette and Uvicorn, on which FastAPI is built, are faster. This superior performance is enabled precisely by ASGI, thanks to which FastAPI supports concurrency and asynchronous code. This is achieved by declaring the endpoints with async def syntax.

- Native concurrency support

It used to be very hard to implement concurrent programming in Python - Async I/O was added with Python 3.4. With FastAPI, concurrency can be easily implemented without worrying about Event loop or async/await management.

Developers can simply declare the first path function as coroutines via the async def function wherever they deem appropriate and then declare specific points aws available through await.

- Dependency injection support

FastAPI offers a simple and easy-to-use dependency injection system. Dependency injection is a compositional way of declaring the necessary components required for the code to run properly.

It's a method for achieving inversion of control, which increases the modularity of the code and makes the system more scalable. In FastAPI, developers can simply declare relevant dependencies in the path operation functions assigned to the API endpoints.

- Inbuilt documentation support

FastAPI offers an extremely handy automatic documentation system. It provides a browser-based user interface that interactively documents an API, powered by Swagger UI GUI.

Alternatively, developers can simply type in /redoc to obtain the alternative documentation consisting of all the endpoints listed. The documentation will always allow developers to easily explain the program to others, make it easier for front-end engineers to use your backend, and add convenience when it comes to testing the API endpoints.

- Inbuilt data validation

This is an enormous benefit - inbuilt data validation allows developers to create a compact code by skipping the validation. It can detect invalid data types during the run and returns the reason for bad input in JSON format.

FastAPI makes use of the Pydantic library for this purpose, which greatly simplifies the validation process and ensures faster typing than it would have been by hand. It can also reduce bugs and FastAPI authors claim it reduces developer errors by up to 40%.

Disadvantages of FastAPI

One should take into consideration the lack of an inbuilt security system and the small community of developers.

7.5. Why FastAPI

Our solution only needs a simple API, as we already discussed, most of the backend frameworks offer plenty of features that we don't require in our solution and this may lead to an overhead in our solution that will affect the size of the project and may affect the response time, so we chose FastAPI as it provides a simple API that leverages all security measures without the overhead of the unnecessary features provided by the rest of the frameworks, this also helped a lot in focus on the models and their accuracy and it took less time and most of the time was directed to building our machine learning models and understanding the machine learning algorithms.

7.6. Backend Design

As shown in figure 7.6.1, the API first initializes the models to be ready for usage, then it waits for upcoming requests and gives them to the models, then the models reply with the prediction.

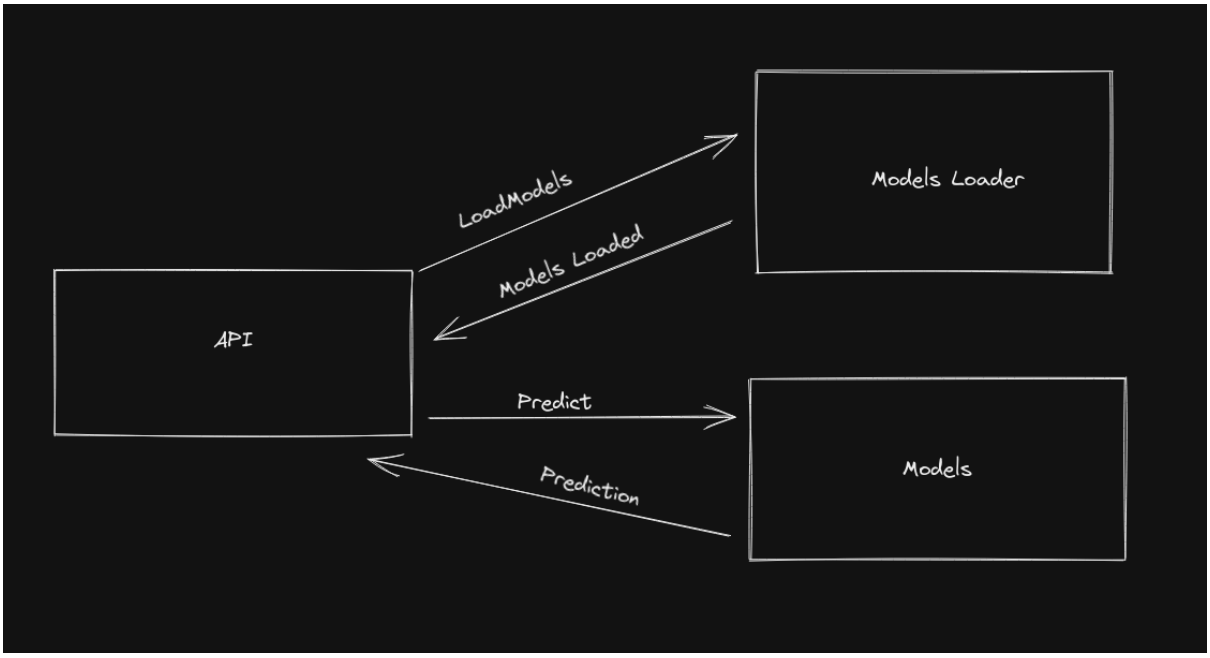


Figure 7.6.1 Backend Design

7.7. Implementation

Main.py

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from models import load_models
from typing import List
# Here we load all the models we need for the project
models = load_models()
class Input(BaseModel):
    input_list: List[float] = []


app = FastAPI()
```

```
@app.get("/")
async def root():
    return {"Response": "Welcome to doctor Helper API"}


@app.post("/{model}")
async def get_predictions(model: str, input_list: Input):
    if model in models:
        prediction_prob = models[model].predict_proba([input_list.input_list]).max() * 100
        prediction_class = models[model].predict([input_list.input_list])[0]
        return {"PredictionClass": str(prediction_class), "Probability": str(prediction_prob)}
    else:
        raise HTTPException(status_code=404, detail="Model not found")
```

In main.py we initialize the models by calling load_models function, then we start listening the upcoming requests using get_prediction function.

Models.py

```
# Here we load all the models we need for the project
import time
import pickle
import warnings
import numpy as np


def load_models():
    print('Loading Models')
    breast_cancer_model = get_model('BreastCancer', [[1001,
0.3001, 0.1471, 25.38, 17.33, 184.6, 2019, 0.2654]])
    heart_disease_model = get_model('HeartDisease',
[[40,1,0,130,289,0,0,172,0,0,0]])
    eryhemato_model = get_model('EryhematoSquamous',
[[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]])
    kidney_model = get_model('Kidney',[[50.0, 80.0, 1.02,
1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 219.0, 176.0, 13.8, 136.0, 4.5,
```

```
8.6, 24.0, 13200.0, 2.7, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0]] )
    lung_cancer_model = get_model('LungCancer', [[1, 55, 1,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]])
    diabetes_model = get_model('Diabetes',
[[6,148,72,35,0,33.6,0.627,50]])

    print('Models Loaded Successfully')

    return {
        'breast_cancer_model': breast_cancer_model,
        'heart_disease_model': heart_disease_model,
        'eryhemato_model': eryhemato_model,
        'kidney_model': kidney_model,
        'lung_cancer_model': lung_cancer_model,
        'diabetes_model': diabetes_model
    }
def get_model(modelName, input):
    warnings.filterwarnings("ignore")
    print(f'Loading {modelName} model')
    filename = f'models/{modelName}.sav'
    model = pickle.load(open(filename, 'rb'))
    print('Testing The Model')
    prediction = model.predict_proba(input)
    np.set_printoptions(suppress=True)
    print('Prediction: ' + str(prediction) + '%')
    print(f'{modelName} Model Loaded Successfully')
    return model
```

In models.py we start by loading our models using pickle, we already saved our models from Google Colab using pickle and now we import the models into the backend to send data to them and ask for prediction, the datastructure used is HashMaps in order to have the key as the model's name and the value is the reference to the model in memory to access it.

7.8. Final Thoughts

In the future, we may use a new tier that will allow us saving the data sent by clients to make our models predictions improve by time.

8. Deployment

8.1. Definition

The IT software "Docker" is containerization technology that enables the creation and use of Linux® containers.

The open-source Docker community works to improve these technologies to benefit all users.

The company, Docker Inc., builds on the work of the Docker community, makes it more secure, and shares those advancements back to the greater community. It then supports the improved and hardened technologies for enterprise customers.

With Docker, you can treat containers like extremely lightweight, modular virtual machines. And you get flexibility with those containers& mdash; you can create, deploy, copy, and move them from environment to environment, which helps.

8.2. Role ofDeployment

Regular software deployments, especially when done on schedule and consistently, can help an organization develop and secure. Updates and patches can assist an organization meet and respond to its ever-changing business needs. The software release cycle looks like this.

"Software deployment" is distinct from "software release," which refers to an application's iterative development process. More functionality, bug-fixing optimization, and other features could be included in a new release.

8.3. Docker

8.3.1. Definition

Docker uses the Linux kernel and technologies such as Cgroups and namespaces to separate processes and allow them to run independently. The ability to run several processes and apps separately from one another to make greater use of your infrastructure while maintaining the security you'd get with separate systems is the goal of containers.

Docker and other container solutions use an image-based deployment methodology. This makes it simple to share an application, or a group of services, across many contexts, along with all of its dependencies. Docker also automates the deployment of an app (or a collection of processes that make up an app) within this container environment.

These tools built on top of Linux containers—what makes Docker user-friendly and unique— gives users unprecedented access to apps, the ability to rapidly deploy, and control over versions and version distribution.

8.3.2. Docker vs LXC (Linux container)

8.3.2.1. LXC

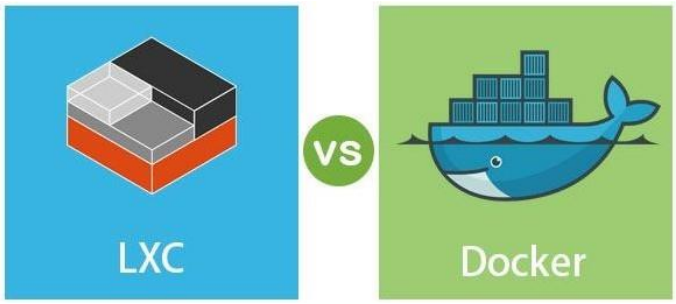
LXC (Linux Containers) is an OS-level virtualization technology that allows creation and running of multiple isolated Linux virtual environments (VE) on a single control host. These isolation levels or
containers can be used to either sandbox specific applications, or to emulate an entirely new host. LXC uses Linux's cgroups functionality, which was introduced in version 2.6.24 to allow the host CPU to better partition memory allocation into isolation levels called namespaces. Note that a VE is distinct from a virtual machine (VM), as we will see below.

8.3.2.2. Docker

Docker, previously called dotCloud, was started as a side project and only open-sourced in 2013. It is really an extension of LXC's capabilities. This it achieves using a high-level API that provides a lightweight virtualization solution to run processes in isolation. Docker is developed in the Go language and utilizes LXC, cgroups, and the Linux kernel itself. Since it's based on LXC, a Docker container does not include a separate operating system; instead, it relies on the operating system's own functionality as provided by the underlying infrastructure. So, Docker acts as a portable container engine, packaging the application and all its dependencies in a virtual container that can run on any Linux server.

8.3.2.3. Docker vs LXC

Docker is not the same as a standard Linux container, despite popular belief. Docker was first constructed on top of LXC, which is most commonly associated with "conventional" Linux containers, although it has subsequently evolved away from that requirement. LXC was beneficial as a lightweight virtualization
solution, but it lacked a good developer and user experience. Docker technology does more than just operate containers; it also simplifies the process of designing and developing containers, shipping images, and image versioning, among other things.



Init systems in traditional Linux containers can manage numerous processes. This means that multiple apps can run simultaneously. The Docker technology encourages and facilitates the separation of applications into their individual processes. This granular approach provides benefits.

8.3.2.3.1. Ease of use

Docker and LXC both provide ample documentation, with helpful guides for creating and deploying containers. Bindings and libraries exist for languages such as Python and Java, making it even easier for developer teams to use. When comparing the two technologies, however, Docker's ever-growing ecosystem will take much more to manage. Docker might have become the standard for running containerized applications, with tools like Kubernetes and Docker Swarm providing the orchestration, however, the ecosystem comes with additional complexity.

Part of this has to do with Docker's key innovation of single-process containers, over and above the standard multiprocess containers that LXC provided. When Docker introduced this innovation, it
inevitably led to downstream complexity for teams porting over traditional applications to a non- standard operating system environment. A lot more planning, architecture decisions and scripting to support applications has to be done.

With LXC, a large part of this complexity is avoided since LXC runs a standard OS init for each container, providing a standard Linux operating system for your apps to live in. As a result, migrating from a VM or bare metal server is often easier to do if you are moving to LXC containers, unlike if you want to move to Docker containers. On the other hand, Docker's approach makes working with containers easier for developer since they don't have to use raw, low-level LXC themselves. This split between a systems admin and developer focus continues to characterize adoption of these tools.

8.3.2.3.2. Popularity

If popularity were the only criteria for deciding between these two containerization technologies, then Docker would handily beat LXC and its REST tool, LXD. It's easy to see why, with Docker taking the devops world by storm since its launch back in 2013. Docker's popularity, however, is not an event in isolation, rather, the application containerization that Docker champions just happens to be a model that tech giants, among them Google, Netflix, Twitter, and other web-scale companies, have gravitated to for its scaling advantages.

LXC, while older, has not been as popular with developers as Docker has proven to be. This is partly due to the difference in use cases that these two technologies focus on, with LXC having a focus on sys admins that's similar to what solutions like the Solaris operating system, with its Solaris Zones, Linux OpenVZ, and FreeBSD, with its BSD Jails virtualization system. These solutions provide OS containers for a whole system, which is achieved, typically, by providing a different root for the filesystem, and creating environments that are isolated from each other and can't share state.

Docker went after a different target market, developers, and sought to take containers beyond the OS level to the more granular world of the application itself. While it started out being built on top of LXC, Docker later moved beyond LXC containers to its own execution environment called libcontainer. Unlike LXC, which launches an operating system init for each container, Docker provides one OS environment, supplied by the

Docker Engine, and enables developers to easily run applications that reside in their own application environment which is specified by a docker image. Just like with LXC, these images can be shared among developers, with a docker file, in the case of Docker, automating the sequence of commands for building an image.

The Docker user base is large and continues to grow, with ZDNet estimating the number of containerized applications at more than 3.5 million and billions of containerized applications downloaded using Docker. Linux powerhouses such as Red Hat and Canonical, the backers of Ubuntu, are firmly on the Docker bandwagon, as are even bigger tech companies like Oracle and Microsoft. With such adoption, it's likely Docker will continue to outstrip LXC in popularity, though system containers like LXC have their place in virtualization of traditional applications that are difficult to port to the microservice architecture that's popular these days.

8.3.3. Why Docker

Advantages of Docker containers

Modularity

The Docker approach to containerization focuses on the ability to update or fix a portion of a program without having to take the entire app down. In addition to this microservices-based strategy, you may use service-oriented architecture (SOA) to share processes across several apps.

Rapid deployment

It used to take days to get new hardware up and running, provisioned, and ready, and the amount of labour and overhead was prohibitive. Containers based on Docker can cut deployment time to seconds. You can quickly share processes with new apps by building a container for each one. Deployment times are significantly reduced because an operating system does not need to boot to add or relocate a container. You can easily and cost-effectively create and remove data created by your containers thanks to faster deployment times.

Rollback

Perhaps the best part about layering is the ability to roll back. Every image has layers. Don't like the current iteration of an image? Roll it back to the previous version. This supports an agile development approach and helps make continuous integration and deployment (CI/CD) a reality from tools perspective.

8.3.4. How it works

Software Deployment is the process of remotely installing software on multiple or all the computers within a network simultaneously, from a central location. The word "Software Deployment" is generally used in the context of a large network (more than 20 computers). An automated deployment using enterprise software deployment tools helps in distributing software without manual intervention saving time and effort.

Software deployment diverges into patch management and IT asset management for software updates and to have complete control and visibility over the software installed across your network.

8.3.5. Docker image

All top-level images, their repository and tags, and their size will be displayed in the default docker images.

Intermediate layers in Docker images boost reusability, reduce disc utilization, and speed up docker builds by caching each phase. By default, these intermediary levels are hidden.

8.3.6. Docker file

A Dockerfile is a text file of instructions which are used to automate installation and configuration of a Docker image. Dockerfiles make it easy to deploy multiple Docker containers without having to maintain the same image across multiple virtual machines.

8.3.6.1. Doker file with Streamlit

This file contains seven main instructions as shown below.

- (FROM): pulls an official python image from docker hub, then our files will be built from that image.
- (WORKDIR): create /app as the working directory for the application
- (COPY source destination): copy file(s) from the source folder to the destination folder.
- (RUN): runs the requirements.txt file in order to install the project dependencies.
- (EXPOSE): used in order to expose the port to be used by the model.
- (ENTRYPOINT) & (CMD): create an entry point in order to finally make the image executable.

8.3.6.2. Doker filewith FastAPI

From the official documentation, FastAPI is a modern [and] fast (high-performance) web framework for building APIs with Python 3.6+ based on standard Python type hints.

As evident from the name, FastAPI is extremely fast and it owes this to the out of the box support of the async feature of Python 3.6+. This is why it is recommended to use the latest versions of Python.

A number of tech giants like Microsoft, Uber and Netflix are already using FastAPI to build their applications.

8.3.7. Docker container

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

8.4. platforms

8.4.1. Heroku

Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage, and scale modern apps. Our platform is elegant, flexible, and easy to use, offering developers the simplest path to getting their apps to market.

Advantages of Heroku

- Heroku offers deployment, environment configuration, and manageability simple and easy to do.
- For beginners, Heroku is an exceptional tool because it is fast and easy toconfigure.
- The Heroku CLI provides a fantastic interface for interacting with the cloudenvironment.
- Pipelines from development to production are very simple. Application rollbacks are also very easy.

Drawbacks of Heroku

- To purchase additional dynos/workers, you need to pay $35 a month, quite costly.
- Limited in types of instances
- Inbound and outbound latency is high.
- It offers low network performance.

8.4.2. Firebase

Firebase is a Backend-as-a-Service (BaaS) that grew into a next-generation app-development platform on Google Cloud Platform. If you are using Firebase, then you don't need to manage the servers and write
APIs. Instead, Firebase is your server, API, and datastore, all written so generically that you can change it to accommodate most needs.

If you are using Firebase Realtime Database, then you are not connecting your database to regular HTTP requests. Instead, you are using a WebSocket to connect your client to the Firebase server.

Advantages of Firebase

- Firebase is blazing fast.
- It provides so many services that it reduces development time and effort.
- Real-time database scalability.
- Google analytics integration.
- Flexible cost.
- Free trial of Firebase services

Drawbacks of Firebase

- If you are building a web application, it does not provide as powerful support as an Android app.
- Limited querying capabilities.
- Limited support for iOS features.
- If your project is enormous, then sometimes you will face real-time synchronization issues.
- Data migration problems.

8.4.3. Digital Ocean

Digital Ocean offers cloud computing technologies to businesses and developers. Through its services, companies can deploy and scale applications effortlessly. These applications can then run parallel to each other across several cloud servers without affecting each other's performance.

Digital Ocean was named one of the largest cloud hosting companies in January 2018. The company is one of the leading cloud service providers in the United States of America. Its headquarters is located in New York City.

DigitalOcean conducted its IPO on March 2021, and it is now a public traded company. The company will use the resources to improve the product offering and to have a more resilient cash position.

8.4.3.1. Features

If you pick Digital Ocean, you can take advantage of the following features that the service provider offers.

- Ease of use
- Developer centric
- Developers can quickly create a scalable VPS to run on their operating system of choice
- Backup and monitor your servers
- Developers can start a virtual private server with any pre-configured web app.
- Allows protection of servers with cloud firewalls.
- Manage your server using API, CLI app, and dashboard.
- Team inclusion option available for building infrastructure together.
- Load balancers and floating IP addresses allow for effective traffic management.

8.4.3.2. Benefits

You should also understand the advantages of using the cloud hosting service provider when comparing to **Heroku**.

Listed below are some of the ways developers benefit while working on Digital Ocean's platform.

- Affordable pricing and the ease of use
- Sever level access and flexibility
- The user interface is simple and aesthetic at the same time. The optimal options present in the UI makes working on Digital Ocean a treat for developers.
- Digital Ocean's performance is better than most cloud hosting companies out there. It was one of the first to use SSD-based virtual machines and IPv6, thereby leading to more convenience for developers.
- The professional and comprehensive documentation helps developers new to the platform to learn all about its utilities.

8.4.4. Why Heroku

Firebase vs. Heroku

To compare Firebase vs. Heroku, Firebase gives developers many in-service tools to use in their applications. On the other hand, the Heroku platform makes it quick and easy to test and deliver new applications.

The Firebase Console is a much-appreciated feature that allows developers to deliver app updates, send or schedule push notifications, and monitor their app from a centralized interface. Heroku provides Heroku CLI that manages the project deployment. It provides valuable and detailed notifications,
metrics, and alerts to help developers monitor their apps and take action when problems arise, or usage patterns change.

Heroku provides the best UX across the board, with an intuitive and useful GUI, API, and command-line interface. However, Firebase users primarily complain about the user experience. The interfaces can be unpleasant to use and unintuitive, especially for the Firestore database system, its real-time database system. We have already compared the Firebase Realtime database vs. Cloud Firestore in this blog.

Digital Ocean vs. Heroku

Apart from the features of Heroku, listed below are some of the benefits of the platform that makes it a contender for **Digital Ocean vs. Heroku** comparison.

- Easy and fast app deployment
- Managed hosting service that takes care of scalability
- It allows developers to shift their focus from infrastructure to coding the application itself.
- Offers a single billing system for projects broken down in the form of teams.
- Support for most Modern Open Source Languages.
- It helps you to focus on innovations instead of thinking about operations.
- Fast permissions and application management
- Impressive CLI and dashboard

Finally, we use Heroku as it is easy to use it as the first step with free login but in the future use digital ocean is better

8.5. Design

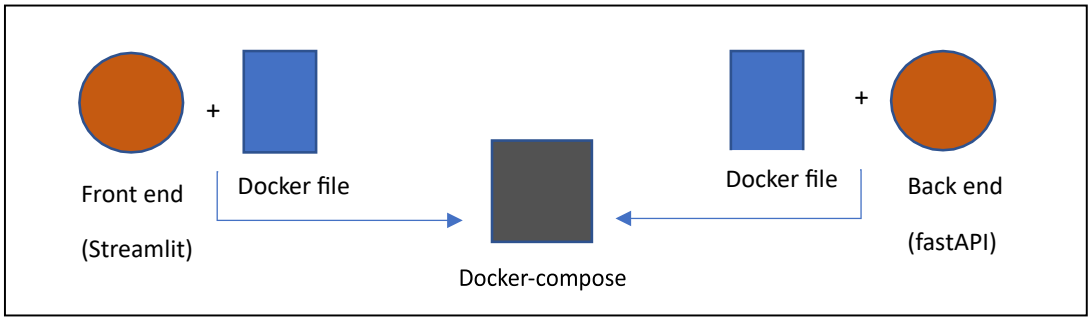


Fig.8.5: Diagram of the Docker app architecture

8.6. Structure of the application

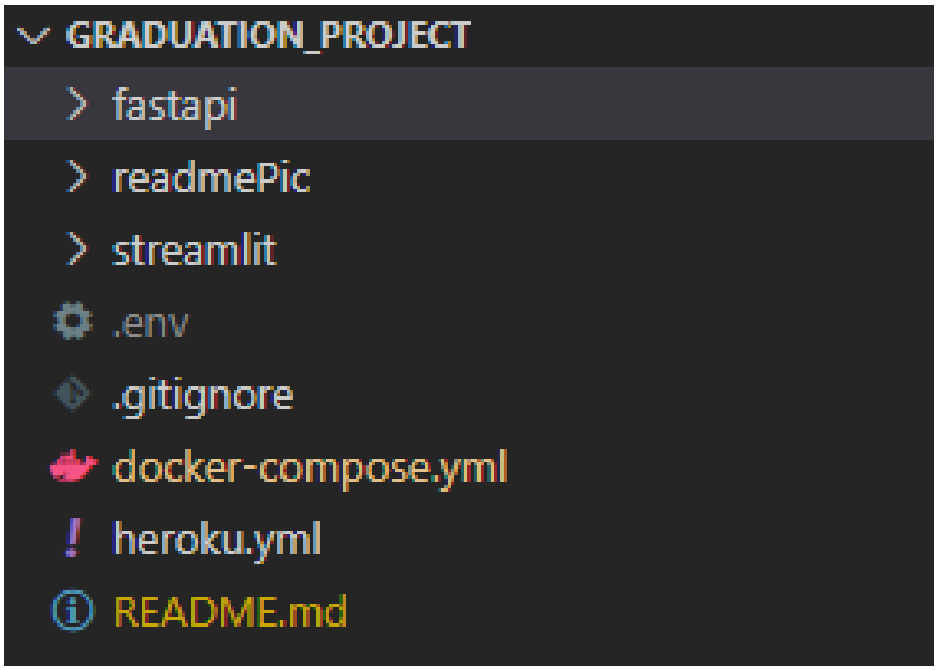


Fig.8.6: Structure of the Docker app

8.7. Implementation of Docker app

8.7.1. Docker file/ FastAPI

```
FROM tiangolo/uvicorn-gunicorn:python3.7

RUN mkdir /fastapi

WORKDIR /fastapi

RUN pip3 install "fastapi[all]" "uvicorn[standard]" sklearn scikit-
learn numpy

COPY . /fastapi

EXPOSE 8000
CMD [ "python","server.py","--precache-models" ]
```

8.7.2. Docker file/ Streamlit

```
#for local docker containers

FROM python:3.7-slim


RUN mkdir /streamlit

COPY requirements.txt /streamlit

WORKDIR /streamlit

RUN pip install -r requirements.txt

COPY . /streamlit
CMD ["streamlit", "run", "app.py"]
```

Requirements.txt

```
streamlit
requests==2.24.0
requests-toolbelt==0.9.1
python-dotenv
```

8.7.3. Docker-Compose

```yaml
version: '3'
services:
  fastapi:
    build: fastapi/
    environment:
        - PORTFastapi=${PORTFastapi} #get ports from .env file
    ports:
        - ${PORTFastapi}:${PORTFastapi}
    networks:
      - deploy_network
    container_name: fastapi

  streamlit:
    build: streamlit/
    depends_on:
      - fastapi
    environment:
        - PORTStreamlit=${PORTStreamlit}
    ports:
        - ${PORTStreamlit}:${PORTStreamlit}
    networks:
      - deploy_network
    container_name: streamlit

networks:
  deploy_network:
    driver: bridge
```

8.7.4. .env

PORTStreamlit=8501

PORTFastapi=8000

8.8. Implementation of Heroku

8.8.1. Heroku.yml

```
build:
  docker:
    web: streamlit/Dockerfile.web  #use diffrent docker file to listen
to heroku port num
    backend: fastapi/Dockerfile
```

8.8.2. Dockerfile.web/ streamlit

```
#for docker heroku containers
FROM python:3.7-slim


RUN mkdir /streamlit

COPY requirements.txt /streamlit

WORKDIR /streamlit

RUN pip install -r requirements.txt

COPY . /streamlit

CMD streamlit run --server.port $PORT app.py
```

8.9. Documentation

8.9.1. Prerequisites

- WSL 2 backend
- Hyper-V backend and Windows containers

- Docker
- install Heroku CLI
- Heroku Account

8.9.2. Structure

1. Streamlit/Dockerfile
2. fastapi/Dockerfile
3. Docker-compose.yml "for local run"
4. heroku.yml

8.9.3. Docker Compose

`docker-compose.yml`

- uses ports num from .env file to run locally

8.9.4. Heroku

`heroku.yml`

- `uses Dockerfile.web instead of using streamlit/Dockerfile`

8.9.5. How to RUN

8.9.5.1. Creat .env file

```
PORTStreamlit=8501
PORTFastapi=8000
```

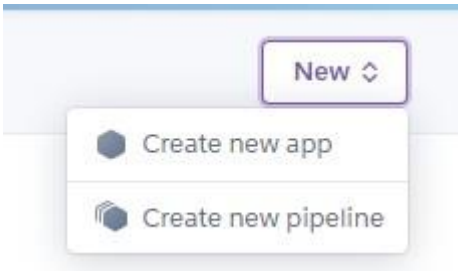
8.9.5.2. Build and play with your docker containers locally

```
docker-compose up
```

`open localhost:8501`

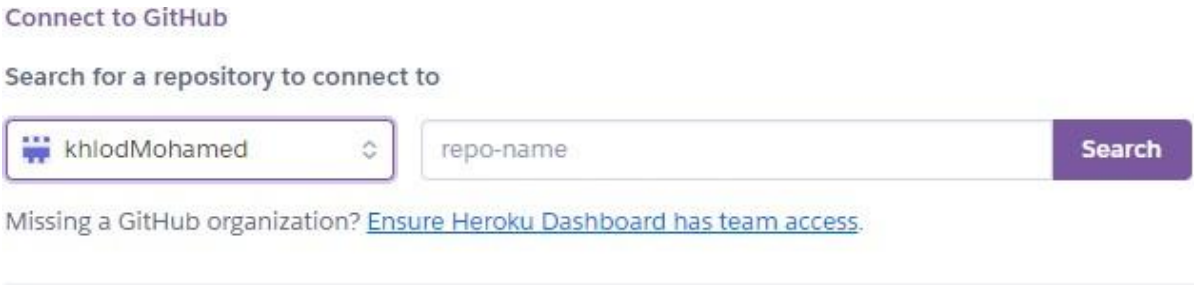
8.9.5.3. Create a heroku pipeline

- Login to heroku
- Creat new pipeline



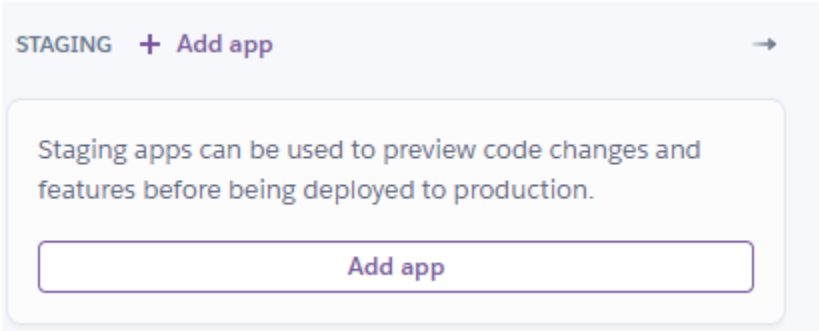
- Enter your pipeline name

- connect your pipeline with your github repo

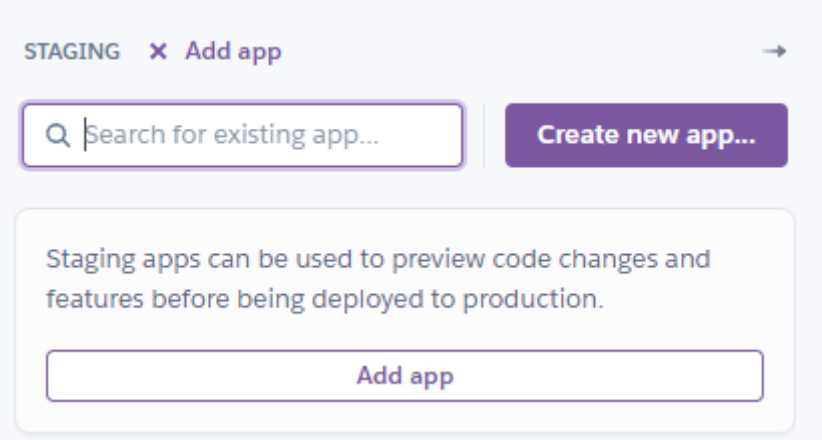


8.9.5.4. Create a heroku app

- Click add app

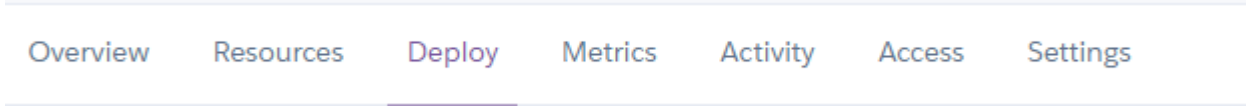


- Enter your app name and click create new app

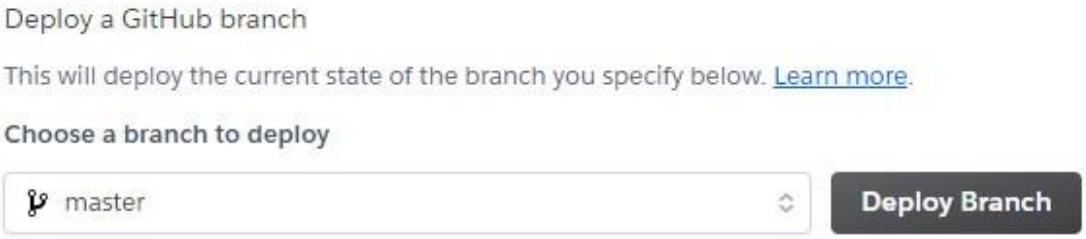


8.9.5.5. Deploy your app

- Go to Deploy tab



- Scroll down click on Deploy Branch



8.9.5.6. Go to <<https://<APP_NAME>.herokuapp.com>> and we are done with deployment.

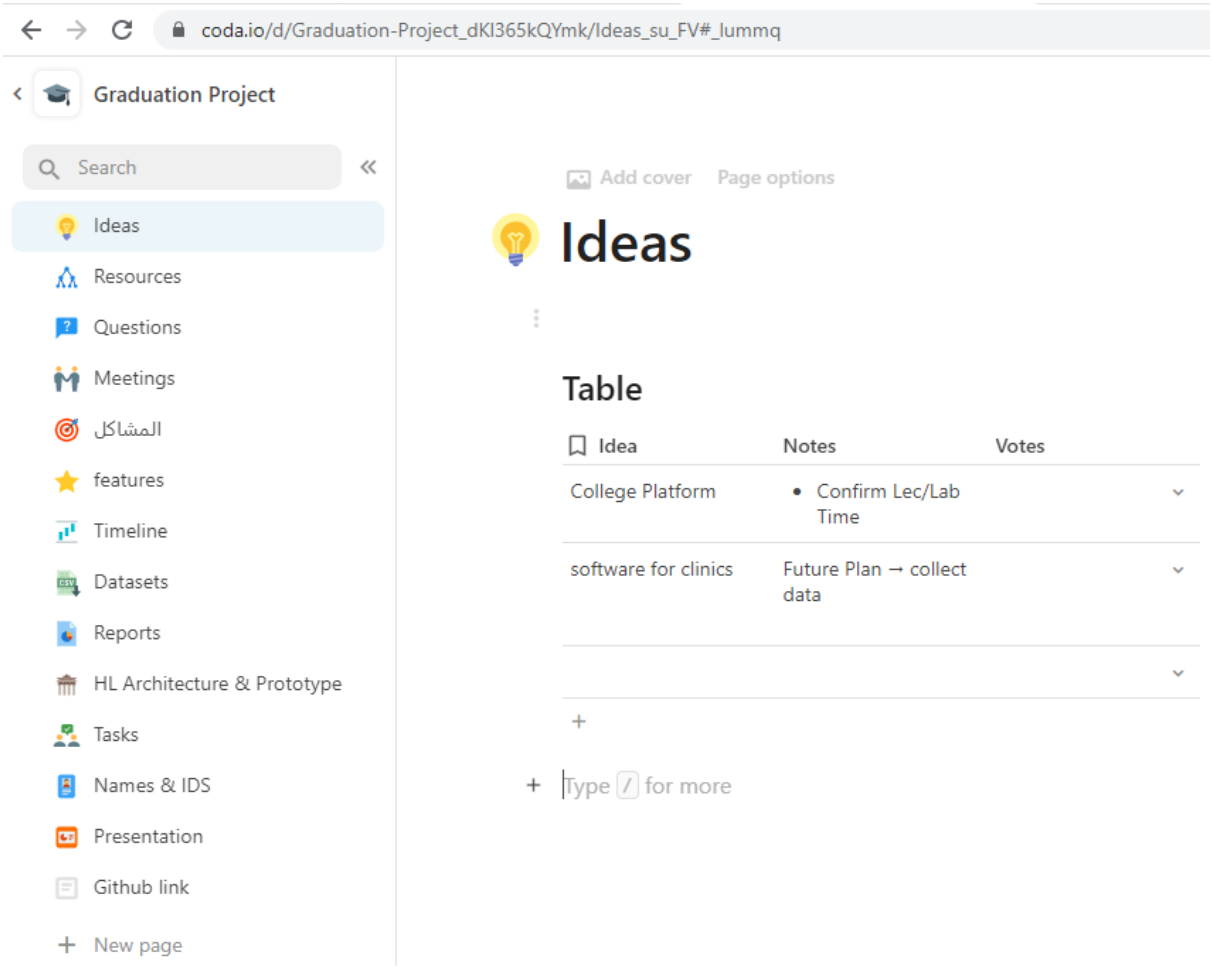
🎉 🎉 🎉

9. Tools

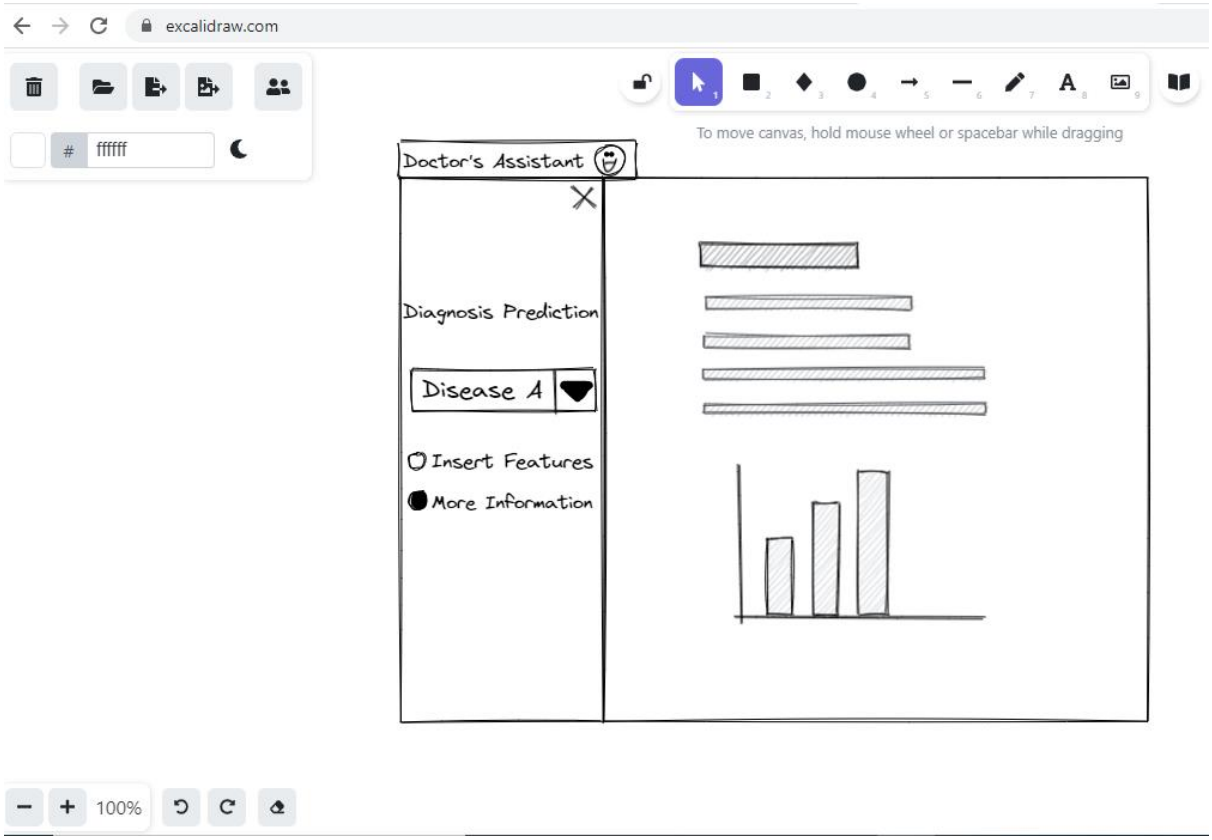
These are the tools that we used throughout our project with their purpose of use and some snapshots.

Zoom: Conducting online audio meetings almost weekly.

Coda: Tracking our progress, setting tasks, deadlines, meetings, creating timelines, recording ideas and documentations, and sharing other information

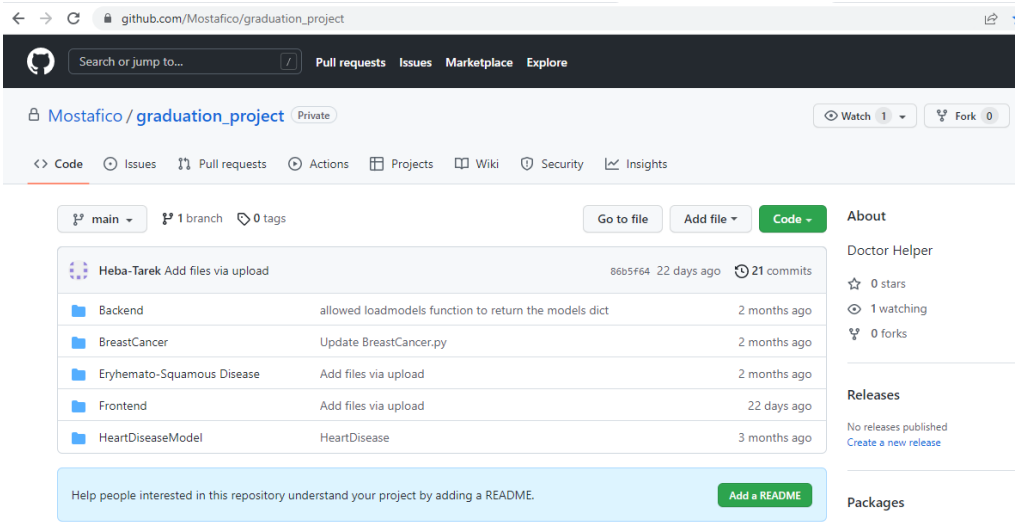


Excali Draw: Digitally sketching prototypes, designs and demonstrational figures.



Google Collaboratory, Microsoft Visual Studio and Python: Programming our project using Python scripting and writing configuration files

GitHub: Sharing project folders and files through a repository accessed among us.



Microsoft Word, Microsoft PowerPoint, Google documents: Documentation.

10. Conclusion

To conclude, by utilizing machine learning, we are not only assisting doctors to obtain diagnosis in a less costly, complex and time-consuming manner, but we are also assisting them in the early detection of diseases, and detecting rare ones.

11. Summary

In this book, we explained what a machine learning life-cycle is, how we implemented each stage in our project, and what each stage entails. We talked about why we should attempt different machine learning algorithms, which classifiers we utilized, and how we implemented each one. Furthermore, we discussed what a frontend, backend and deployment is and what are their roles in the context of our web development process. We mentioned some of the common frontend technologies used by developers, as well as some of the backend and machine learning frontend frameworks that may be used to create the backend and frontend. We explained what a docker file is and platforms we could use for deployment. We presented our choice of frontend, backend frameworks, and deployment platform, as well as, the design, documentation, and implementation for the frontend, backend and deployment.

12. References

12.1. Prototype and Architecture

https://uxdesign.cc/importance-of-prototyping-in-designing-7287c7035a0d
https://www.javatpoint.com/supervised-machine-learning
https://www.reforge.com/brief/the-4-different-types-of-product-prototypes#lysHP7Di-XFU9CjW__w0mw

12.2. Machine Learning Mastery with Python

https://medium.com/@namanbhandari/extratreesclassifier-8e7fc0502c7
https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/
https://www.geeksforgeeks.org/random-forest-classifier-using-scikit-learn/
https://www.geeksforgeeks.org/ml-xgboost-extreme-gradient-boosting/
https://www.geeksforgeeks.org/ml-gradient-boosting/
https://www.geeksforgeeks.org/implementing-the-adaboost-algorithm-from-scratch/
https://www.javatpoint.com/machine-learning-life-cycle

12.3. Frontend

https://www.geeksforgeeks.org/frontend-vs-backend/

https://techterms.com/definition/frontend

https://en.wikipedia.org/wiki/Front-end_web_development

https://towardsdatascience.com/gradio-vs-streamlit-vs-dash-vs-flask-d3defb1209a2

12.4. Backend

https://en.wikipedia.org/wiki/Node.js

https://www.voidcanvas.com/describing-node-js/

https://en.wikipedia.org/wiki/Django_(web_framework)

https://data-flair.training/blogs/django-advantages-and-disadvantages/

https://en.wikipedia.org/wiki/Flask_(web_framework)

https://careerfoundry.com/en/blog/web-development/what-is-flask/

https://en.wikipedia.org/wiki/FastAPI

https://www.netguru.com/blog/python-flask-versus-fastapi

12.5. Deployment

https://www.pagerduty.com/resources/learn/what-is-software-deployment/

https://www.upguard.com/blog/docker-vs-lxc

https://towardsdatascience.com/create-an-awesome-streamlit-app-deploy-it-with-docker-a3d202a636e8

https://www.redhat.com/en/topics/containers/what-is-

docker?sc_cid=7013a000002wLw6AAE&gclid=Cj0KCQjwvqeUBhCBARIsAOdt45ZdrPkIIFvUZv-cCO-CzSl4vu5UVozSs-kRqehNBKiNLXb863swEWgaArK4EALw_wcB&gclsrc=aw.ds

https://blog.back4app.com/firebase-vs-heroku/

https://blog.back4app.com/digitalocean-vs-heroku/