PROJECT REPORT
ON

Cardiovascular Stroke Prediction System

Carried Out at

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING
KNOWLEDGE PARK, BANGALORE

UNDER THE SUPERVISION OF

Mr. Abhay Mane

C-DAC Bangalore

Submitted By

Ankit Bhoge (210950125010)

Abhishek Pandey (210950125004)

Rohit Goyal (210950125051)

Acharya Sarang (210950125006)

Sheikh Mohsin (210950125060)

PG DIPLOMA IN BIG DATA ANALYTICS
C-DAC, BANGALORE

Declaration

We hereby certify that the work being presented in the report entitled stroke prediction System, in partial fulfillment of the requirements for the award of PG Diploma Certificate and submitted in the department of PG-DBDA of the C-DAC Bangalore, is an authentic record of our work carried out during the period, 14th March 2021 to 14th April 2021 under the supervision of Mr. Abhay Mane, C-DAC Bangalore. The matter presented in the report has not been submitted by us for the award of any degree of this or any other Institute/University.

Ankit Bhoge PRN :- 210950125010 _____

Abhishek Pandey PRN :- 210950125004 _____

Rohit Goyal PRN :- 210950125051 _____

Acharya Sarang PRN :- 210950125006 _____

Sheikh Mohsin PRN :- 210950125060 _____

Date :../../....

Acknowledgement

This acknowledgment is a token of gratitude, I feel towards the people who have helped me to make this report a rich experience. I am delighted to offer our sincere thanks to my Institute and our **Course Coordinator, PG-DBDA Ms. Uma Prasad** for enabling me to add a formal project Report on my PG - Diploma project work, which has made me acquire tremendous knowledge. I would like to express profound gratitude to my project guide **Mr. Abhay Mane** Guide for his invaluable support, encouragement, supervision and useful suggestions throughout this project work. There moral support and continuous guidance enabled us to complete my work successfully. We are highly grateful to **Mr. Ramesh Naidu (ACTS training Centre, C-DAC)**, for his guidance and support whenever necessary while doing this course Post-Graduate Diploma in Big Data Analytics (PG-DBDA) through C-DAC ACTS, Bangalore. His regular suggestions made my work easy and proficient. Last but not the least, I am thankful and indebted all those who helped me directly or indirectly in completion of this project report.

Ankit Bhoge 210950125010
Abhishek Pandey 210950125004
Rohit Goyal 210950125051
Acharya Sarang 210950125006
Sheikh Mohsin 210950125060

Abstract

A heart attack is another term for a cardiovascular stroke. According to the World Health Organization, stroke is the second leading cause of death worldwide, accounting for 11% of all deaths. As a result of the high number of COVID-19 patients who experience breathing issues as a side effect of therapy, the risk of a stroke has increased. There isn't always a system in place to check for the odds of having a stroke. A modest fluctuation in blood pressure that we underestimate can sometimes lead to a stroke. So, we created a system that will predict the possibility of having a stroke. We experimented with various of machine learning techniques to come up with the optimal solution. We chose the best performing algorithm in terms of accuracy and type-1 errors among the majority of machine learning algorithms that we tested with implementation. We devised a method for predicting the likelihood of suffering a stroke based on a few simple characteristics that may be measured at home. As a result, if our algorithm predicts that you may have a stroke, you should seek medical advice as soon as possible.

Contents

List of Figures

Chapter 1

Introduction and Overview of Project

The Prediction of Cardiovascular Stroke is a classification problem. We must classify the provided data across multiple classes based on patterns in the input attributes in a classification issue. ”Stroke” and ”no stroke” are the two classifications available in our problem. We used multiple classification machine learning algorithms, as well as numerous sets of trials based on various feature extraction methodologies and algorithms, to categories this. We also spoke with medical professionals for data preparation to obtain precise details required for data cleansing. We implemented the algorithms after cleaning and preprocessing the data. We must perform hyperparameter optimization to improve the algorithm's efficiency. We used a combination of random search and grid search to fine-tune the algorithm, and we also used TPE (Tree Parzen Estimators) to verify the results. After adjusting with both hyperparameter optimization techniques, the Random Forest classifier was the best algorithm. This Random Forest model is integrated into the front-end user interface. The Django framework in Python was used to create the front end. On the front end, we built a web-based application that would accept input for the model's various features and then display the result, which is predicted by the model.

1.1 Objectives of the project

- Predicted the possibility of getting a stroke.
- Learn the implementation of different machine learning algorithms.

1.2 Software Required

- Ensemble Machine Learning Algorithm Random Forest Classifier
- Python Django Web-Framework
- AWS (Amazon Web Services)

1.3 Machine Learning

In a world where nearly, all manual tasks are being automated, the definition of ”manual” is changing. We are living in an era of constant technological progress, and one such advancement is in the field of machine learning.

Machine learning is a data analytics technique that teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases. Machine learning algorithms find natural patterns in data that generate insight and help you make better decisions and predictions.

Machine learning uses two types of techniques: Supervised learning, which trains a model on known input and output data so that it can predict future outputs, and Unsupervised learning, which finds hidden patterns or intrinsic structures in input data.

Supervised machine learning builds a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data.

Supervised learning uses classification and regression techniques to develop machine learning models. Classification techniques predict discrete responses. we use classification if our data can be tagged, categorized, or separated into specific groups or classes. Here in our problem, we have two discrete responses as ”stroke” or ”no stroke”. So, we used classification technique.

Regression techniques predict continuous responses. Unsupervised learning finds hidden pat-

terns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data without labeled responses

1.3.1 Advantages of Machine Learning

- Machine learning algorithms are continuously improving in accuracy and efficiency, which makes better decisions.
- Machine learning is responsible for cutting the workload and time. By automating things, we let the algorithm do the hard work for us.
- Machine learning algorithms are good at handling data that is multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments
- Machine learning has a wide variety of applications. This means that we can apply Machine learning to any of the major fields. Machine learning has a role everywhere, from medical, business, and banking to science and tech.

1.3.2 Random Forest Classifier

Random forests or random decision forests is an learning method for classification and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.
The fundamental concept behind random forest is a simple but powerful one - the wisdom of crowds. In data science speak, the reason that the random forest model works so well is: A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models. Random forests are frequently used as "Blackbox" models in businesses, as they generate reasonable predictions across a wide range of data while requiring little configuration.

Before understanding the working of the random forest, we must look into the ensemble technique. Ensemble simply means combining multiple models. Thus, a collection of models is used to make predictions rather than an individual model. Ensemble uses two types of methods: - Bagging creates a different training subset from sample training data with replacement the final output is based on majority voting. For example, Random Forest. Boosting combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy.
A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the max_samples parameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

Important Features of Random Forest

1. **Diversity-** Not all attributes/variables/features are considered while making an individual tree, each tree is different.

2. **Immune to the curse of dimensionality-** Since each tree does not consider all the features, the feature space is reduced.

3. **Parallelization-** Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. **Train-Test split-** In a random forest we don't have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

1.3.3 Python Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source. A Web framework is a set of components that provide a standard way

to develop websites fast and easily. Django's primary goal is to ease the creation of complex database-driven websites. Some well-known sites that use Django include PBS, Instagram, Disqus, Washington Times, Bitbucket and Mozilla.
Django follows the MVT design pattern (Model View Template).
The model provides data from the database. In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases. The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it. Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements. The models are usually located in a file called models.py.
A view is a function or method that takes http requests as arguments, imports the relevant model(s), and finds out what data to send to the template, and returns the final result. The views are usually located in a file called views.py.
A template is a file where you describe how the result should be represented. Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.

Design Philosophies of Django

1. **Loosely Coupled :-** Django aims to make each element of its stack independent of the others.

2. **Less Coding :-** Less code so in turn a quick development.

3. **Don't Repeat Yourself (DRY) :-** Everything should be developed only in exactly one place instead of repeating it again and again.

4. **Fast Development :-** Django's philosophy is to do all it can to facilitate hyper-fast development.

5. **Clean Design :-** Django strictly maintains a clean design throughout its own code and makes it easy to follow best web-development practices.

Advantages of Django

- **Object-Relational Mapping (ORM) Support** Django provides a bridge between the data model and the database engine, and supports a large set of database systems including MySQL, Oracle, Postgres, etc. Django also supports NoSQL database through Django-nonreal fork. For now, the only NoSQL databases supported are MongoDB and google app engine.
- **Multilingual Support** Django supports multilingual websites through its built-in internationalization system. So you can develop your website, which would support multiple languages.
- **Framework Support -** Django has built-in support for Ajax, RSS, Caching and various other frameworks.
- **Administration GUI -** Django provides a nice ready-to-use user interface for administrative activities.
- **Development Environment -** Django comes with a lightweight web server to facilitate end-to-end application development and testing.

Chapter 2

Evaluation Metrics

2.1 Accuracy

Accuracy is one metric for evaluating classification models. Informally, Accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

Accuracy=TP+TNTP+TN+FP+FN

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

2.2 Precision and Recall

Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.
In the field of information retrieval precision is the fraction of retrieved documents that are relevant to the query:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Recall calculates how many of the Actual Positives our model capture through labeling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

2.3 F1 Score

F1 is a function of Precision and Recall.
F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

2.4 Support

Support is the number of actual occurrences of the class in the specified data set. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or re-balancing.

2.5 Confusion matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.
For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values: Let's decipher the matrix:

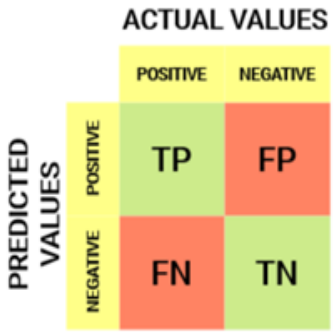


Figure 2.1: Confusion Matrix

- The target variable has two values: **Positive** or **Negative**
- The columns represent the **actual values** of the target variable
- The rows represent the **predicted values** of the target variable

Chapter 3

Data Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.
When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data preprocessing task.
A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.
Before going into pre-processing and data exploration we will explain some of the concepts that allowed us to select our features.

3.1 Exploratory Data Analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

3.1.1 Imports and Read in Data

Here we are importing the all of the basic libraries for data preprocessing as well as sklearn.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random

from sklearn.model_selection import train_test_split, RandomizedSearchCV, GridSearchCV
from sklearn.preprocessing import StandardScaler
from category_encoders.leave_one_out import LeaveOneOutEncoder
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

from sklearn.ensemble import RandomForestClassifier
```

3.1.2 Loading Dataset

```
df = pd.read_csv("CVA_dataset.csv", sep=";")
df.head()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

3.1.3 EDA

It is a way of visualizing, summarizing and interpreting the information that is hidden in rows and column format. EDA is one of the crucial steps in data science that allows us to achieve certain insights and statistical measure that is essential for the business continuity, stockholders and data scientists. It performs to define and refine our important features.

1. Understanding Data
2. Handle Missing value
3. Removing duplicates
4. Outlier Treatment
5. Normalizing and Scaling
6. Encoding Categorical variables
7. Bivariate Analysis

Understanding Data

The very first step in exploratory data analysis is to identify the type of variables in the dataset. Variables are of two types Numerical and Categorical. info() method to identify the data type of the variables in the dataset as well as it will tell the count of non-null values as well.
We can get the size of the dataset using the shape method.

```
1 df.shape
```

```
(70000, 13)
```

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   id           70000 non-null  int64
 1   age          70000 non-null  int64
 2   gender       70000 non-null  int64
 3   height       70000 non-null  int64
 4   weight       70000 non-null  float64
 5   ap_hi        70000 non-null  int64
 6   ap_lo        70000 non-null  int64
 7   cholesterol  70000 non-null  int64
 8   gluc         70000 non-null  int64
 9   smoke        70000 non-null  int64
 10  alco         70000 non-null  int64
 11  active       70000 non-null  int64
 12  cardio       70000 non-null  int64
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

describe() function to get various summary statistics that exclude NaN values. This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

```
1 df.describe()
```

	count	mean	std	min	25%	50%	75%	max
id	70000.0	49972.419900	28851.302323	0.0	25006.75	50001.5	74889.25	99999.0
age	70000.0	19468.865814	2467.251667	10798.0	17664.00	19703.0	21327.00	23713.0
gender	70000.0	1.349571	0.476838	1.0	1.00	1.0	2.00	2.0
height	70000.0	164.359229	8.210126	55.0	159.00	165.0	170.00	250.0
weight	70000.0	74.205690	14.395757	10.0	65.00	72.0	82.00	200.0
ap_hi	70000.0	128.817286	154.011419	-150.0	120.00	120.0	140.00	16020.0
ap_lo	70000.0	96.630414	188.472530	-70.0	80.00	80.0	90.00	11000.0
cholesterol	70000.0	1.366871	0.680250	1.0	1.00	1.0	2.00	3.0
gluc	70000.0	1.226457	0.572270	1.0	1.00	1.0	1.00	3.0
smoke	70000.0	0.088129	0.283484	0.0	0.00	0.0	0.00	1.0
alco	70000.0	0.053771	0.225568	0.0	0.00	0.0	0.00	1.0
active	70000.0	0.803729	0.397179	0.0	1.00	1.0	1.00	1.0
cardio	70000.0	0.499700	0.500003	0.0	0.00	0.0	1.00	1.0

Handle Missing Value

As we can see in .info() screenshot in understanding data section, there are no null values present in the dataset. So, no need to handle missing values.

Removing Duplicates

.duplicates() method of pandas library is used to find out the number of duplicate records. After finding the number of duplicate records we will drop all those records using dropDuplicates() method.

```
print(f"Number of duplicate records = {df.duplicated().sum()}")
# Dropping duplicate records
df.drop_duplicates(inplace=True)
```

```
Number of duplicate records = 24
```

Outlier Treatment

Here we have medical problem so it is best practice to consult with the doctor while treating the outlier. After consulting with doctor, we manually look into different values of different columns. Also, there is chance of human error while recording the data, so we need to consider that also. For example, in ap_hi (systolic blood pressure) columns we have negative value but number is valid range of Systolic blood pressure. In such case we just have to make it positive value.

```
#removing outlier from height
df['height'] = np.where(df['height'] < 91.44, np.nan, df['height'])
df['height'] = np.where(df['height'] > 213, np.nan, df['height'])
df.dropna(how='any', inplace=True)
```

```
#removing outlier from weight
df['weight'] = np.where(df['weight'] < 30, np.nan, df['weight'])
df['weight'] = np.where(df['weight'] > 150, np.nan, df['weight'])
df.dropna(how='any', inplace=True)

# replacing values from ap_hi with appropriate values
df['ap_hi'] = np.where(df['ap_hi'] < 0, df['ap_hi']*-1, df['ap_hi'])
df['ap_hi'] = np.where(df['ap_hi'] == 1, df['ap_hi'] * 100, df['ap_hi'])
df['ap_hi'] = np.where(df['ap_hi'] == 7, df['ap_hi'] * 10, df['ap_hi'])
df['ap_hi'] = np.where(((df['ap_hi'] > 8) & (df['ap_hi'] < 30)), df['ap_hi']*10, df['ap_hi'])
df['ap_hi'] = np.where(df['ap_hi'] > 500, df['ap_hi'] / 10, df['ap_hi'])
df['ap_hi'] = np.where(df['ap_hi'] > 500, df['ap_hi'] / 10, df['ap_hi'])

#replacing values from ap_lo with appropriate values
df['ap_lo'] = np.where(df['ap_lo'] < 0, df['ap_lo'] * -1, df['ap_lo'])
df['ap_lo'] = np.where(df['ap_lo'] == 0, np.nan, df['ap_lo'])
df.dropna(how = 'any', inplace= True)
df['ap_lo'] = np.where(df['ap_lo'] < 10, df['ap_lo'] * 10, df['ap_lo'])
df['ap_lo'] = np.where(((df['ap_lo'] > 500)), df['ap_lo'] / 10, df['ap_lo'])
df['ap_lo'] = np.where(((df['ap_lo'] > 500)), df['ap_lo'] / 10, df['ap_lo'])
```

Normalizing or Scaling

Need for Normalizing is we have values of different columns lies within different range of value. Normalizing or scaling will make all data values to fall within common limits. The machine learning model which are about implement may divert because of different range of values from different columns. We normalize or scale data from numerical columns only.

```
# dependent and independent variable
X = df.iloc[:, :-1]
Y = df.iloc[:, -1]
# collecting all numerical columns
numeric_cols = X.select_dtypes(include=np.number).columns.to_list()
# standard scaling the data
scaler = StandardScaler()
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])
```

Encoding Categorical Variable

If categorical data is in form of string data type, then it is most important to encode the data. Even if the data is in numerical data type, then also it is good to encode data. Encoding categorical data is one of such tasks which is considered crucial. All models basically perform mathematical operations which can be performed using different tools and techniques. But the harsh truth is that mathematics is totally dependent on numbers. So in short we can say most of the models require numbers as the data, not strings or not anything else and these numbers can be float or integer. Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the models to give and improve the predictions.

```
# dependent and independent variable
X = df.iloc[:, :-1]
Y = df.iloc[:, -1]
# collecting categorical columns
X['gender'] = X['gender'].astype(object)
X['cholesterol'] = X['cholesterol'].astype(object)
X['gluc'] = X['gluc'].astype(object)
X['smoke'] = X['smoke'].astype(object)
X['alco'] = X['alco'].astype(object)
X['active'] = X['active'].astype(object)
categorical_cols = X.select_dtypes('object').columns.to_list()
# Encoding categorical variable with TargetEncoder()
encoder = TargetEncoder()
encoder.fit(X[categorical_cols], Y)
X[categorical_cols] = encoder.transform(X[categorical_cols])
```

3.2 Data Preprocessing

3.2.1 Separating Data in train and test

Any machine learning algorithm needs to be tested for accuracy. In order to do that, we divide our data set into two parts: training set and testing set. As the name itself suggests, we use the training set to make the algorithm learn the behaviors present in the data and check the correctness of the algorithm by testing-on-testing set.

Here we will check the distribution of class label in complete dataset as well distribution in training and testing dataset.

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, random_state=2)
print("distribution of records")
print(f"shape of x_train = {x_train.shape}")
print(f"shape of x_test = {x_test.shape}\n")
print("Percentage distribution")
print(f"train size = {round((len(x_train)/len(df))*100, 2)}")
print(f"test size = {round((len(x_test)/len(df))*100, 2)}\n")
print(f"Check Distribution Of Label in complete dataset")
print('Not cardio', round(df['cardio'].value_counts()[0]/len(df) * 100,2), '% of the dataset')
print('cardio', round(df['cardio'].value_counts()[1]/len(df) * 100,2), '% of the dataset\n')
print("train data distribution")
print(f"no cardio {round(y_train.value_counts()[0]/len(y_train)*100, 2 )}% of dataset")
print(f"cardio {round(y_train.value_counts()[1]/len(y_train)*100, 2 )}% of dataset\n")
print("test data distribution")
print(f"no cardio {round(y_test.value_counts()[0]/len(y_test)*100, 2 )}% of dataset")
print(f"cardio {round(y_test.value_counts()[1]/len(y_test)*100, 2 )}% of dataset\n")
```

```
distribution of records
shape of x_train = (52482, 11)
shape of x_test = (17494, 11)

Percentage distribution
train size = 75.0
test size = 25.0

Check Distribution Of Label in complete dataset
Not cardio 50.02 % of the dataset
cardio 49.98 % of the dataset

train data distribution
no cardio 49.91% of dataset
cardio 50.09% of dataset

test data distribution
no cardio 50.36% of dataset
cardio 49.64% of dataset
```

Chapter 4

Model Building

The modeling process was divided into two main parts: Building different model of machine learning to find out best algorithm and hyperparameter optimization of selected model.

4.1 Building Different Model

In Machine Learning, we have different classification algorithm. Few of them were basic algorithms such as Naïve-Bayes, Decision Tree, Logistic Regression. Also based on this basic algorithm we have different ensemble algorithms namely Random Forest, XGBoost, Adaboost, Support Vector Machine, Bagging Classifier, Gradient Boosting. We also implemented Multilayer Perceptron neural network algorithm.
We have tried implementing all this algorithm to find out the best performing model which will further tuned to get best model. And then this model will be integrated with Django framework. Here we have attached excel sheet of record of all algorithms.

Algorithm Name	Accuracy	Precision	Recall	F-1 Score	False Positive	False Negative
Random forest	74.1341	76.9348	69.1921	72.8583	1818	2700
AdaBoost	73.4757	76.6334	66.5468	71.2361	1749	2884
Multi-Layer Percerptron	72.8803	73.107	72.508	72.843	2337	2400
Decision Tree	73.24	73.19	73.56	73.37	3450	3551
NB	73.25	73.19	73.56	73.37	3450	2551
XGBoost	73.64	71	78	75	2255	3279
SVM	73	71	79	75	2170	3520
Bagging	72.87	75	69	72	2448	3248
Gradient Boosting	73.11	75	69	72	2375	3271
Logistic Regression	69	67	77	72	2054	3296

Figure 4.1: Different algorithms with their best metrics

From above chart we can see that Random Forest is the best choice among all other machine learning model. Now this model will be used for hyperparameter optimization.

4.2 Hyperparameter Optimization

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Machine learning models are not intelligent enough to know what hyperparameters would lead to the highest possible accuracy on the given dataset. However, hyperparameter values when set right can build highly accurate models, and thus we allow our models to try different combinations of hyperparameters during the training process and make predictions with the best

combination of hyperparameter values. Some of the hyperparameters in Random Forest Classifier are n_estimators (total number of trees in a forest), max_depth (the depth of each tree in the forest), and criterion (the method to make splits in each tree). n_estimators set to 1 or 2 doesn't make sense as a forest must have a higher number of trees, but how do we know what number of trees will yield the best results? And for this purpose, we try different values like [100, 200, 300]. The model will try all three of the given values and we can easily identify the optimal number of trees in our forest.

4.2.1 Random Search and Grid Search

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm. In this case, the optimization problem is said to have a low intrinsic dimensionality. Random Search is also embarrassingly parallel, and additionally allows the inclusion of prior knowledge by specifying the distribution from which to sample.

```
# Number of trees in random forest
n_estimators = [int(x) for x in range(25, 1025, 25)]
criterion = ['gini', 'entropy']
# Number of features to consider at every split
max_features = ['auto', 'sqrt', 'log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in range(2,11)]
# Minimum number of samples required at each leaf node
min_samples_leaf = [int(x) for x in range(1,6)]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'criterion' : criterion,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450, 475, 500, 525, 550, 57
5, 600, 625, 650, 675, 700, 725, 750, 775, 800, 825, 850, 875, 900, 925, 950, 975, 1000], 'criterion': ['gini', 'entropy'], 'ma
x_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split':
[2, 3, 4, 5, 6, 7, 8, 9, 10], 'min_samples_leaf': [1, 2, 3, 4, 5], 'bootstrap': [True, False]}
```

Figure 4.2: Creating search space for random search

```
rf=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=rf,
                               param_distributions=random_grid,
                               n_iter=25,
                               cv=10,
                               verbose=2,
                               random_state=100,
                               n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(x_train,y_train)
```

```
rf_randomcv.best_params_
```

```
{'n_estimators': 400,
 'min_samples_split': 4,
 'min_samples_leaf': 3,
 'max_features': 'auto',
 'max_depth': 10,
 'criterion': 'entropy',
 'bootstrap': True}
```

Figure 4.3: Performing random search and getting best params

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a hold-out validation set. Since the parameter space of a machine learner may include real-valued or unbounded value spaces for certain parameters, manually set bounds and discretization may be necessary before applying grid search.

```
param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth'] - 3,
                  rf_randomcv.best_params_['max_depth'] - 2,
                  rf_randomcv.best_params_['max_depth']-1,
                  rf_randomcv.best_params_['max_depth'],
                  rf_randomcv.best_params_['max_depth'] + 1,
                  rf_randomcv.best_params_['max_depth'] + 2,
                  rf_randomcv.best_params_['max_depth'] + 3],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 2,
                          rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1,
                          rf_randomcv.best_params_['min_samples_split'] + 2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 30,
                     rf_randomcv.best_params_['n_estimators'] - 20,
                     rf_randomcv.best_params_['n_estimators'] - 10,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 10,
                     rf_randomcv.best_params_['n_estimators'] + 20,
                    rf_randomcv.best_params_['n_estimators'] - 30],
        'bootstrap': [rf_randomcv.best_params_['bootstrap']],
         'max_features' : [rf_randomcv.best_params_['max_features']] }

print(param_grid)
```

Figure 4.4: Creating search space for grid search using best params of random search

```
#### Fit the grid_search to the data
rf=RandomForestClassifier(criterion='entropy',
                          max_features='auto',
                          bootstrap=True)

grid_search=GridSearchCV(estimator=rf,
                          param_grid=param_grid,
                          cv=10,
                          n_jobs=-1,
                          verbose=2)
grid_search.fit(x_train,y_train)
```

```
Fitting 10 folds for each of 588 candidates, totalling 5880 fits

GridSearchCV(cv=10, estimator=RandomForestClassifier(criterion='entropy'),
             n_jobs=-1,
             param_grid={'max_depth': [7, 8, 9, 10, 11, 12, 13],
                         'min_samples_leaf': [4, 5, 6, 7],
                         'min_samples_split': [3, 4, 5],
                         'n_estimators': [370, 380, 390, 400, 410, 420, 430]},
             verbose=2)
```

```
grid_search.best_params_
```

```
{'max_depth': 10,
 'min_samples_leaf': 6,
 'min_samples_split': 4,
 'n_estimators': 430}
```

Figure 4.5: Performing grid search and getting best estimator

For hyperparameter tuning, instead giving wide range of hyperparameter to grid search directly; we will implement Random search on wide range of parameter to find out best parameter randomly. Then we will feed few nearby values of params given by random search to Grid search. In this way we can get best result in less time compared to direct grid search.

4.2.2 Tree Parzen Estimators

This method handles categorical hyper-parameters in a tree-structured fashion. For instance, the number of layers of a neural net and the number of neurons in each define a tree structure (there cannot be a third layer if the second one is not there, and setting the number of neurons only makes sense if this layer exists in the graph). Another obvious example can be taken from the choice of the optimizer (each optimizer can have its own set of parameters).
The idea of Parzen estimators is close to Bayesian optimization while standing on an opposite theoretical ground. While Bayesian optimization tries to figure out p(y—x) (y is the value on the response surface, i.e., the validation loss, and x is the hyper-parameter), a tree of Parzen estimators models p(x—y) and p(y).

As for Bayesian optimization, the first step in TPE is to start sampling the response surface by random search to initialize the algorithm.
For this we have used Microsoft tool namely NNI (Neural Network Intelligence). NNI comes with different tuners and TPE is one of them. For Optimizing using NNI we have defined search space first.

Random_forest > {} search_space.json > ...

```
{
    "criterion": {"_type" : "choice", "_value" : ["gini", "entropy"]},
    "max_depth": {"_type" : "uniform", "_value" : [5, 100]},
    "max_features": {"_type" : "choice", "_value" : ["auto", "sqrt", "log2"]},
    "min_samples_leaf":{"_type": "uniform", "_value":[0, 0.5]},
    "min_samples_split":{"_type": "uniform", "_value":[0, 1]},
    "n_estimators":{"_type":"randint", "_value":[25, 1025]}
}
```

Figure 4.6: search_space.json

Then we have to write python code to run hyperparameter optimization which select different set of parameters using nni.get_next_parameter() method.
After that we have create config.yml file in which configuration details for experiments will be there.

Random_forest > ! config.yml

```yaml
experimentName: Random_Forest-TPE
searchSpaceFile: search_space.json
trialCommand: python main.py
trialConcurrency: 2
maxTrialNumber: 50
maxExperimentDuration: 10h
tuner:
  name: TPE
  classArgs:
    optimize_mode: maximize
trainingService:  # For other platforms, check mnist-pytorch example
  platform: local

```

Figure 4.7: config.yml

After that we have to create experiments using “nnictl create “ command. Then we can visualize the experiment details in webUI.

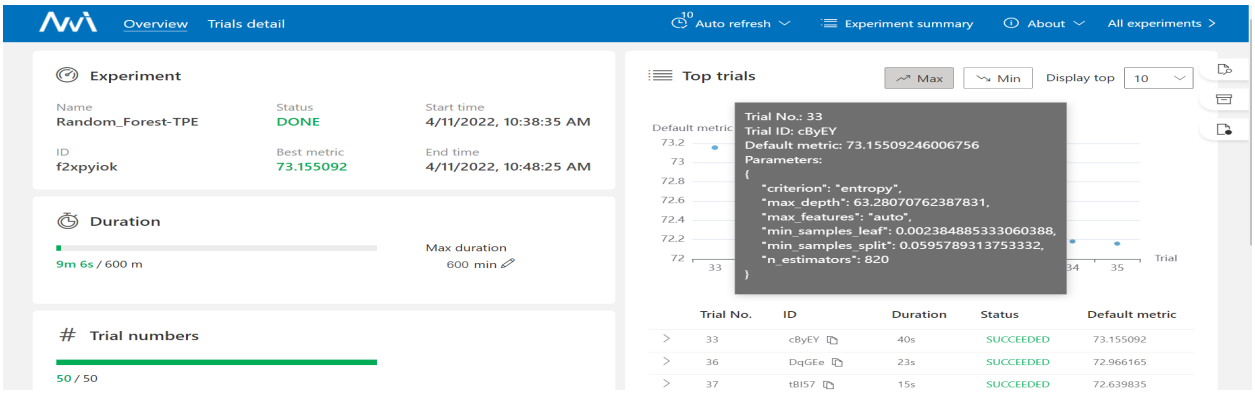


Figure 4.8: NNI Web UI

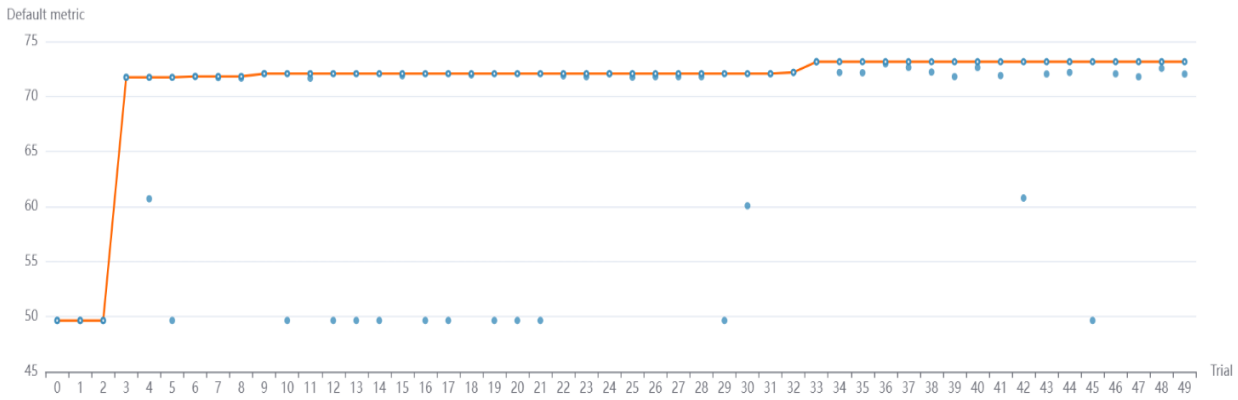


Figure 4.9: Optimization curve for iterations

visualization of different combinations of Hyperparameter.

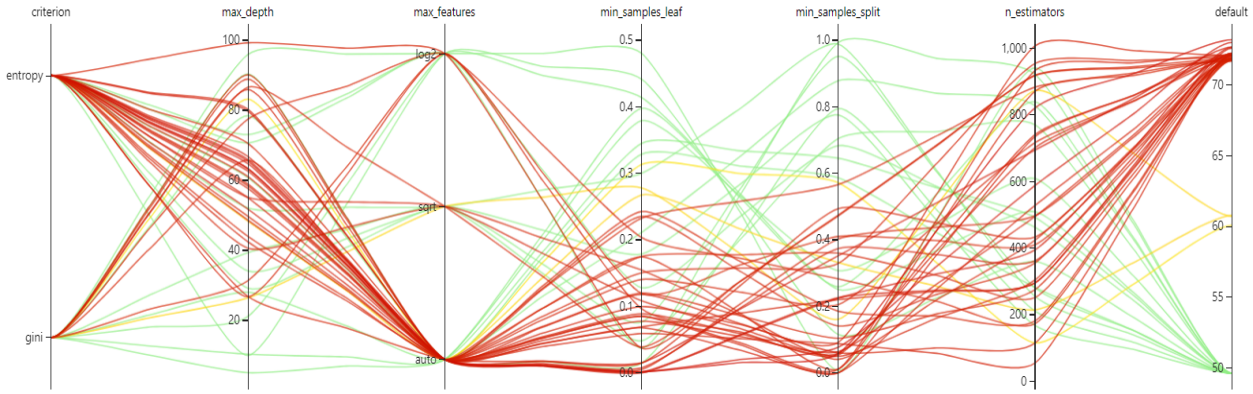


Figure 4.10: Combination of Hyperparameter

4.3 Integration with Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites.with the help of Django, HTML and CSS, we build interactive user interfaces for our project's cardiovascular stroke detection system.

In this, we have designed three pages. The first page is the home page, (basically the welcome page), and the second page is for taking input for different attributes from the user. On the predication page, we have created a form page for taking 11 feature attributes details from the user and storing them in the same variable in the dataset. After this, we have created one dictionary as a key value pair, and then we created one dataframe, and with this dataframe, we have appended that key value pair. We apply scaler transformation on the numerical column and encoder transformation on the categorical column, and finally we apply our model for prediction.

We have provided all the path details in the urls.py python file for redirection from one page to another. Here in urls, we have two files, namely tts/urls.py which is the main file from where we process the request from the user; and the other file is main/urls.py in which we write urls for HTML pages.

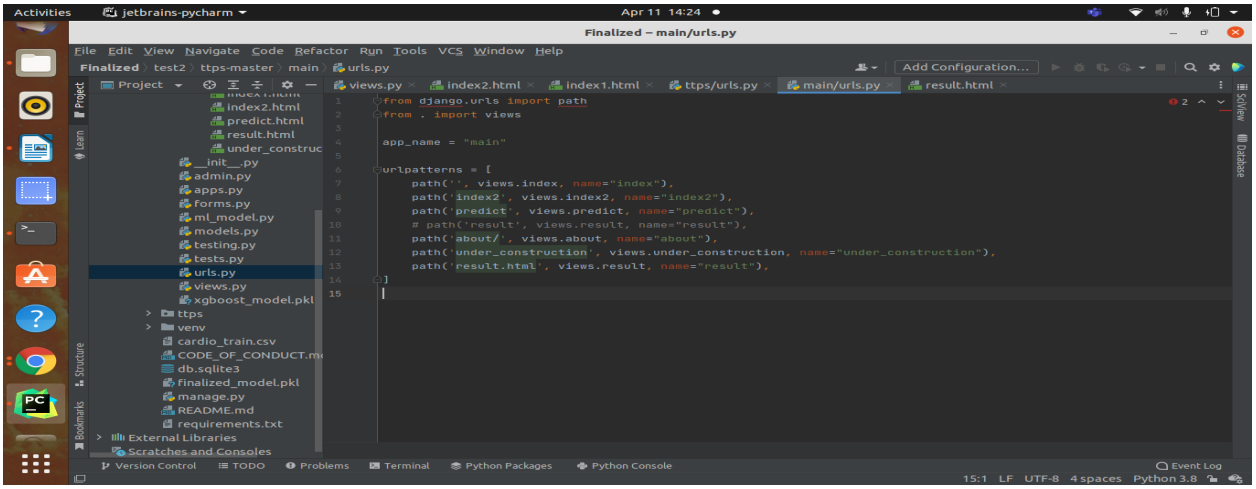
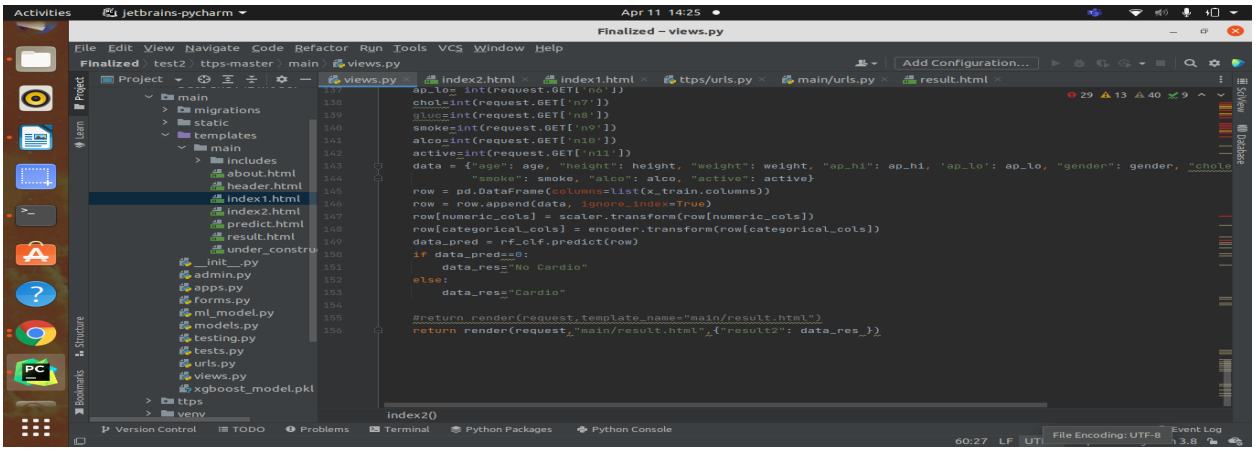


Figure 4.11: urls.py

Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function (or a class-based view). The view gets passed the following arguments: An instance of HttpRequest If the matched URL pattern contains no named groups, then the matches from the regular expression are provided as positional arguments. All function we have created in under views.py python file. In this file we write all functions and methods that we are going use.this file contain all working code for project.



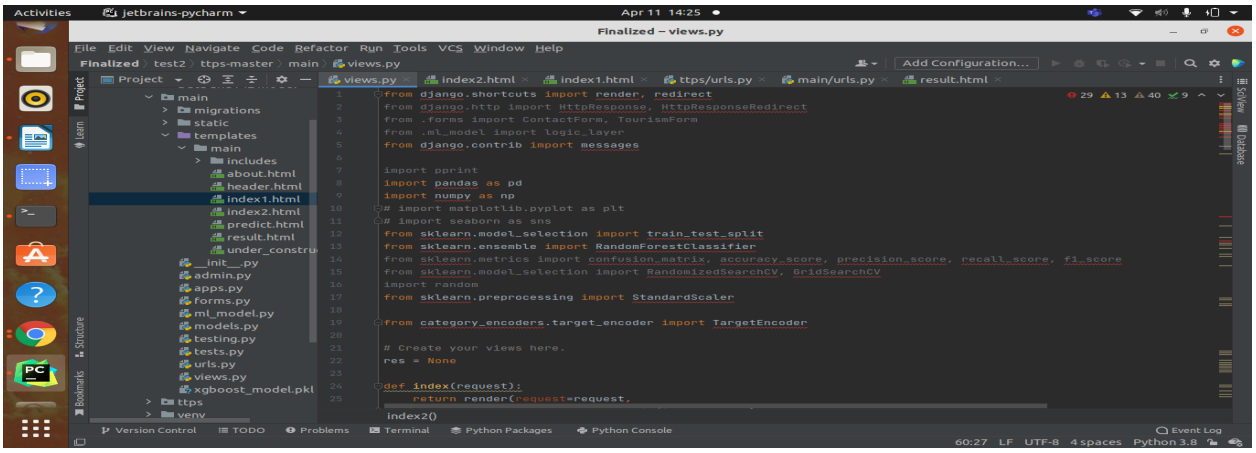


Figure 4.12: Views.py

Another file whichc we craete is result.html in which we will display the result got from views.py.

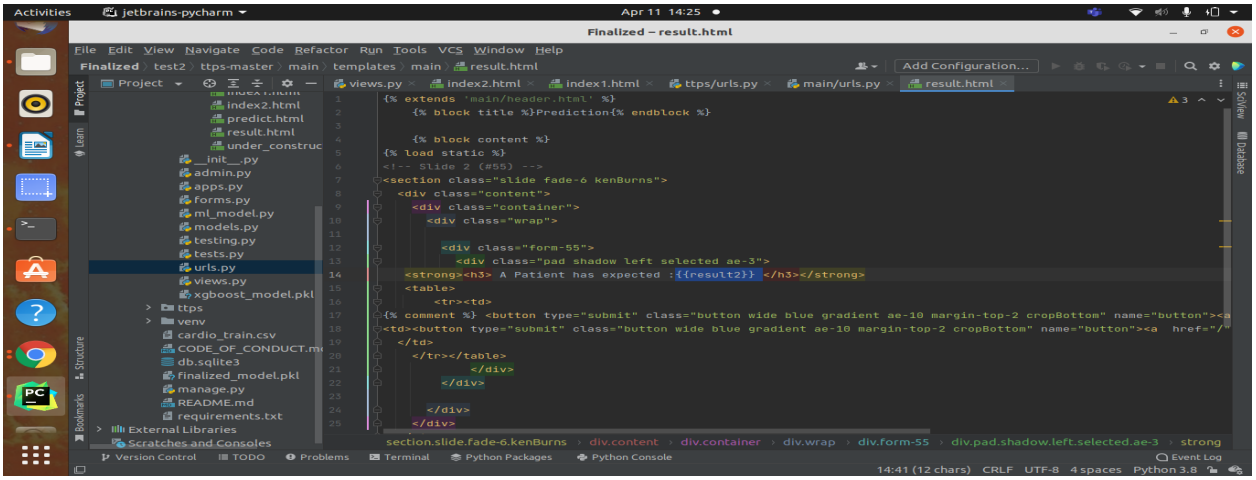


Figure 4.13: Result.html

4.4 AWS Deployment

We use AWS (The task of designing a scalable, efficient, and cost-effective deployment solution should not be limited to the issue of how you will update your application version, but should also consider how you will manage supporting infrastructure throughout the complete application lifecycle).

AWS provides a number of services that provide management capabilities for one or more aspects of your application lifecycle. Depending on your desired balance of control (i.e., manual management of resources) versus convenience (i.e., AWS management of resources) and the type of application, these services can be used on their own or combined to create a feature-rich deployment solution.

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Instance types comprise varying combinations of CPU, memory, storage, and networking capacity and give you the flexibility to choose the appropriate mix of resources for your applications. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

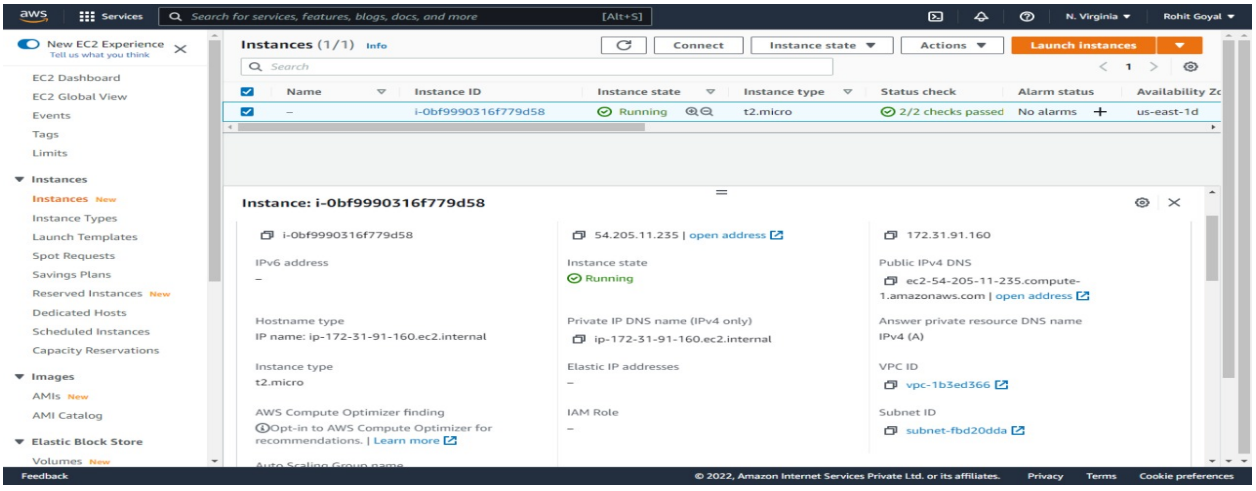


Figure 4.14: Amazon EC2 instance

We made Ubuntu virtual machine and hosted Django web application for that we have used following steps:

1. Launching an EC2 instance
2. Connecting the EC2 instance on local machine
3. Configure EC2 instance
4. Clone the Django Application Repository.
5. Create a virtual Environment
6. Install the necessary packages
7. Test run the application

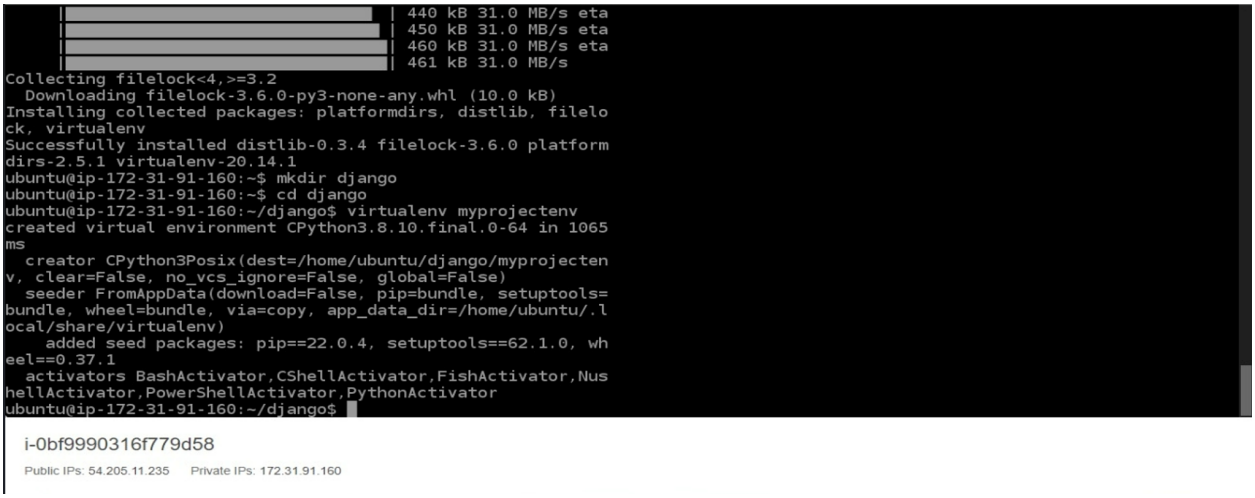


Figure 4.15: starting SSH server

4.5 Sample Output

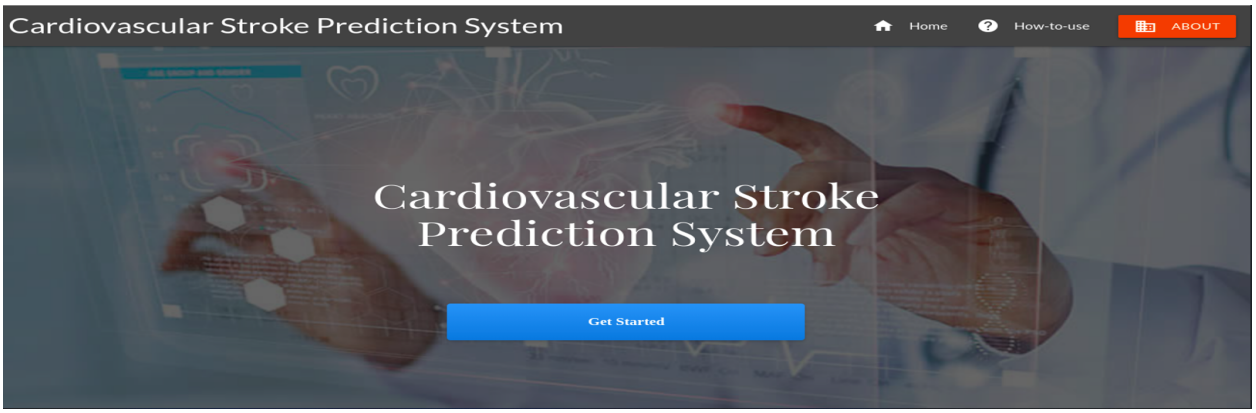


Figure 4.16: welcome Page

Figure 4.17: User detail input - 1

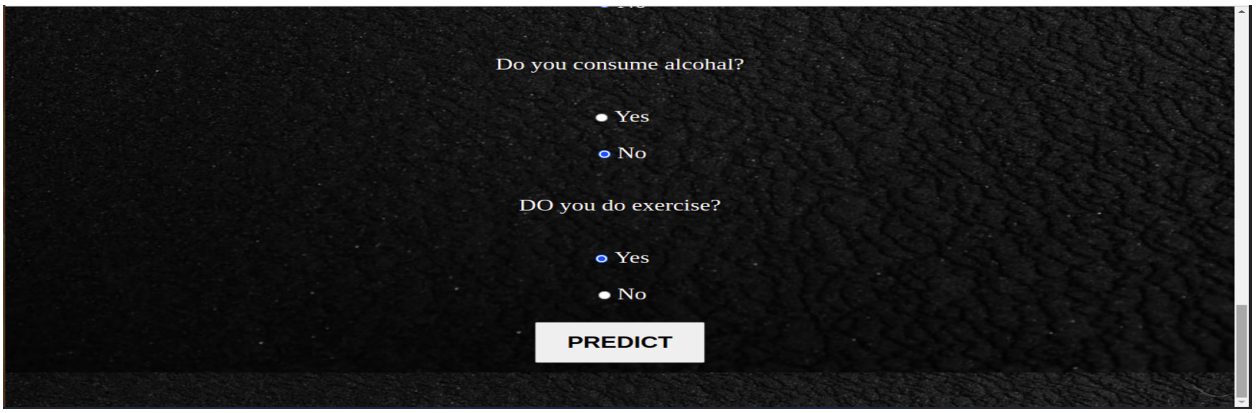


Figure 4.18: User detail input - 2

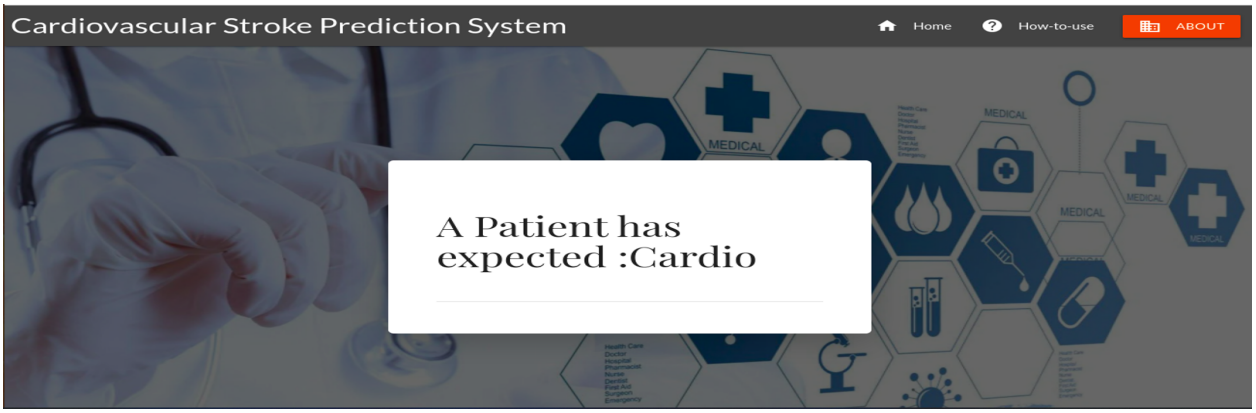


Figure 4.19: final predict page

Chapter 5

Conclusion

- We did data visualization and data analysis of the target variable, age features, and what-not along with its univariate analysis and bivariate analysis.
- We also did a complete feature engineering part in this article which summons all the valid steps needed for further steps i.e. model building.
- From the above model accuracy, Random Forest is giving us the better performing metrics.
- In this what we found is during small datasets in some other cases most of time decision trees direct us to a solution which is not accurate, but when we look at Naïve Bayes results we are getting more accurate results with probabilities of all other possibilities but due to guidance to only one solution decision trees may miss lead.
- In this what we found is during small datasets in some other cases most of time decision trees direct us to a solution which is not accurate, but when we look at Naïve Bayes results we are getting more accurate results with probabilities of all other possibilities but due to guidance to only one solution decision trees may miss lead.

References

[1] Dataset from Kaggle named ”Cardiovascular Disease dataset” from https://www.kaggle.com/datasets/sulianova/cardiovascular-disease-dataset

[2] ”A Tour of Machine Learning Algorithms” by Jason Brownlee from https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

[3] Django Web Framework (Python) from developer.mozilla on https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django

[4] Django Tutorial from W3school https://www.w3schools.com/django/index.php

[5] ”Hyperparameter tuning for machine learning models” by JEREMY JORDAN from https://www.jeremyjordan.me/hyperparameter-tuning/

[6] ”Alternative Hyperparameter Optimization Technique You need to Know – Hyperopt” from analytics Vidhya on https://www.analyticsvidhya.com/blog/2020/09/alternative-hyperparameter-optimization-technique-you-need-to-know-hyperopt/

[7] ”Tree-structured Parzen Estimator” from optunity on https://optunity.readthedocs.io/en/latest/user/solvers/TPE.html

[8] ”NNI doumentation” from https://nni.readthedocs.io/en/stable/index.html

[9] AWs Documentation from https://docs.aws.amazon.com/

[10] Random Forest and other libraries from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html