# Configuration Manual

MSc Research Project
Data Analytics

AKINWALE S. OBAFEMI
Student ID: x20200854

School of Computing
National College of Ireland

Supervisor: Jorge Basilio

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | AKINWALE S. OBAFEMI |
| **Student ID:** | X20200854 |
| **Programme:** | Data Analytics |
| **Year:** | 2022 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Jorge Basilio |
| **Submission Due Date:** | 15/08/2022 |
| **Project Title:** | Configuration Manual |
| **Word Count:** | 1310 |
| **Page Count:** | 12 |

| | |
|---|---|
| **Signature:** | |
| **Date:** | 15th August 2021 |

| **Office Use Only** | |
|---|---|
| Signature: | |
| Date: | |
| Penalty Applied (if applicable): | |

# A Predictive Model for Predicting Blood Pressure Levels Using Machine Learning Techniques

Akinwale Sunday Obafemi
x20200854

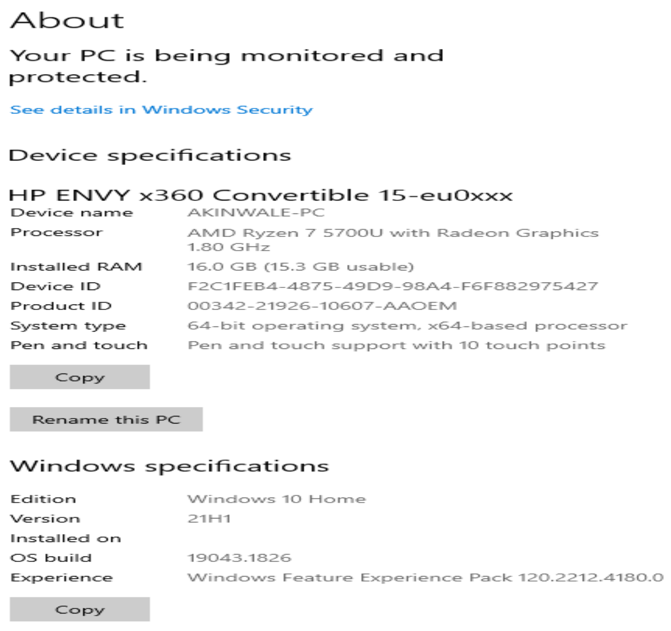
## 1 Introduction

Configuration Manual is a document which gives a walkthrough on the software and hardware requirements needed to perform the modelling and run the codes right from the preparation of data stage up to the Implementation phase. The manual would serve as a guide to help in replicating this research work titled A Predictive Model for Predicting Blood Pressure Levels Using Machine Learning Techniques.

## 2 System Configuration

### 2.1 Hardware Environment

The machine specifications of the device used for the research work would be discussed in this section. For this work, a laptop which runs a 64-bit Microsoft Windows 10 Operating System, with 1.80 Ghz processor and 16GB RAM was used. Full description is shown in Figure 1 below.



About

Your PC is being monitored and protected.

See details in Windows Security

Device specifications

HP ENVY x360 Convertible 15-eu0xxx

| | |
|---|---|
| Device name | AKINWALE-PC |
| Processor | AMD Ryzen 7 5700U with Radeon Graphics 1.80 GHz |
| Installed RAM | 16.0 GB (15.3 GB usable) |
| Device ID | F2C1FEB4-4875-49D9-98A4-F6F882975427 |
| Product ID | 00342-21926-10607-AAOEM |
| System type | 64-bit operating system, x64-based processor |
| Pen and touch | Pen and touch support with 10 touch points |

Windows specifications

| | |
|---|---|
| Edition | Windows 10 Home |
| Version | 21H1 |
| Installed on | |
| OS build | 19043.1826 |
| Experience | Windows Feature Experience Pack 120.2212.4180.0 |

Fig. 1: Hardware Configuration.

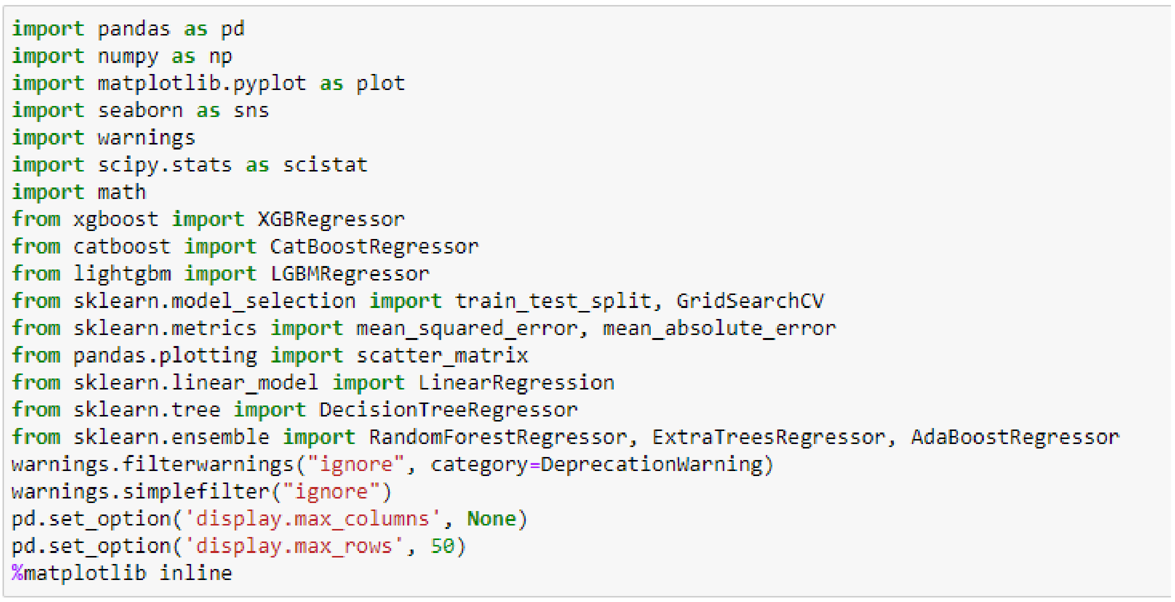
### 2.2 Software Configuration

This part describes the software specifications that were used in implementing this project. The programming language used was Python, while the IDE used is Google Colab (which is a cloud based Jupyter Notebook). The web browser used is Google Chrome. Also, the software used for the documentation of the report is Overleaf. After signing in onto the Google Colab, your google drive has to be mounted to be able to access your data, then all necessary libraries would be imported thereafter.

The libraries needed to be imported include:

1. Pandas.
2. NumPy
3. Seaborn
4. Matplotlib
5. SciPy.Stats
6. Math
7. Sklearn

All the libraries imported are shown in Figure 2 below.



```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plot
import seaborn as sns
import warnings
import scipy.stats as scistat
import math
from xgboost import XGBRegressor
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error
from pandas.plotting import scatter_matrix
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor, AdaBoostRegressor
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter("ignore")
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 50)
%matplotlib inline
```

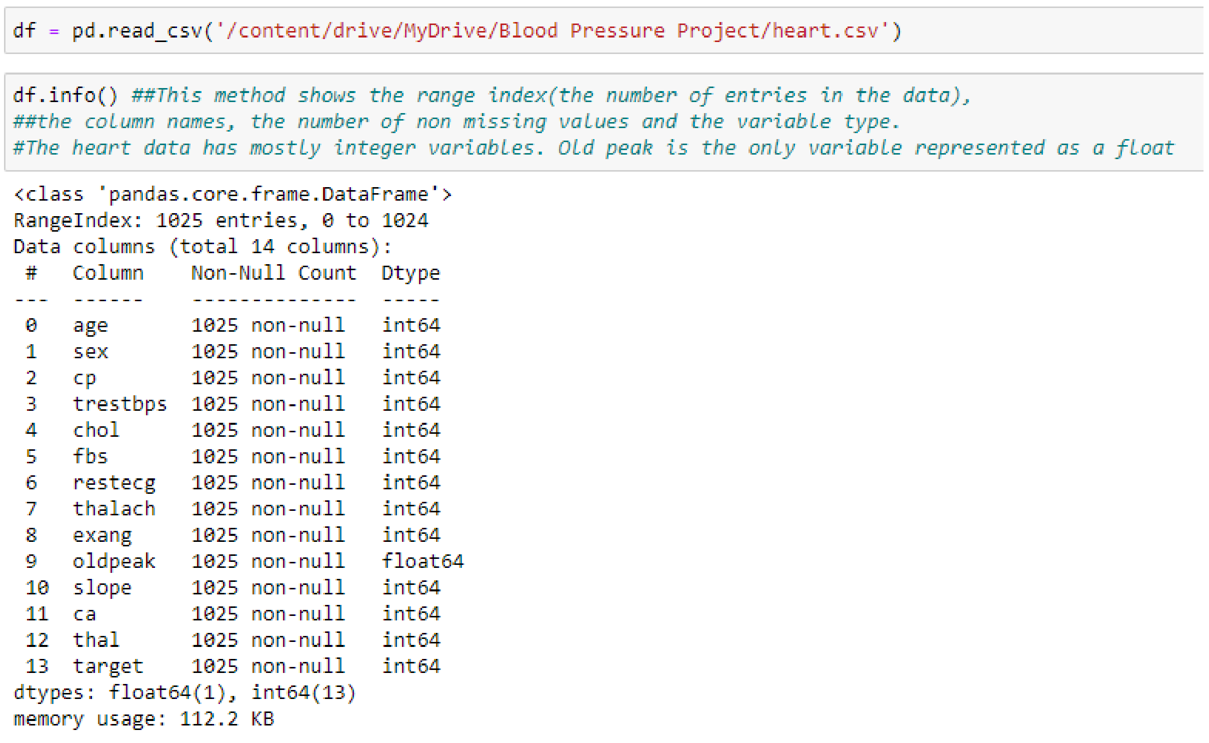
Fig. 2: All libraries used.

## 3 Data Preparation

In this section, the whole process of how our dataset was uploaded and read into the work environment was described. After that, how the data was investigated (or inspected) and cleaned were shown. The dataset was used for the research work was gotten from Kaggle and we discovered from this stage that there were no missing values here.

### 3.1 Reading the Dataset

The dataset used was first mounted on my drive, then it was read into the environment using the python code as shown below in Figure 3. It can also be seen in the figure that inspection of the data for missing values showed no missing values in the data so there was not much cleaning to do in the data.



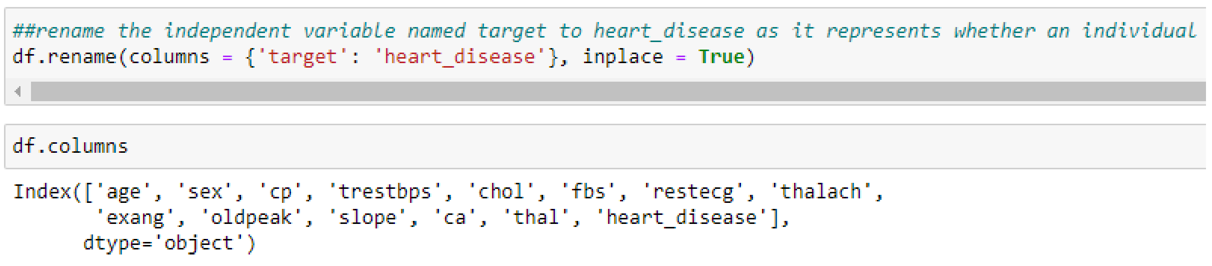
```
df = pd.read_csv('/content/drive/MyDrive/Blood Pressure Project/heart.csv')
```

```
df.info() ##This method shows the range index(the number of entries in the data),
##the column names, the number of non missing values and the variable type.
#The heart data has mostly integer variables. Old peak is the only variable represented as a float
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Fig. 3: Reading data from drive and showing its properties.

As part of the preprocessing stage also, the Heart_disease column was renamed appropriately to avoid confusion as it is not our "Target" variable for this experiment. This was done as seen in Figure 4.



```
##rename the independent variable named target to heart_disease as it represents whether an individual
df.rename(columns = {'target': 'heart_disease'}, inplace = True)
```

```
df.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'heart_disease'],
      dtype='object')
```

Fig. 4: Renaming the heart_disease column.

## 3.2 Exploratory Data Analysis

This stage is necessary for exploring one's data. Exploration is key in data analysis because it aims to assist in understanding the general landscape of the data and detect patterns using visual graphics. Here, the relationships between variables are shown in the Figures 5 to 7.

```
## barchart of sex vs. mean trestbps
# compute mean trestbps per sex = (0,1)
ax = df.groupby('sex').mean().trestbps.plot(kind = 'bar')
ax.set_ylabel('Avg. trestbps')
plot.show()
```

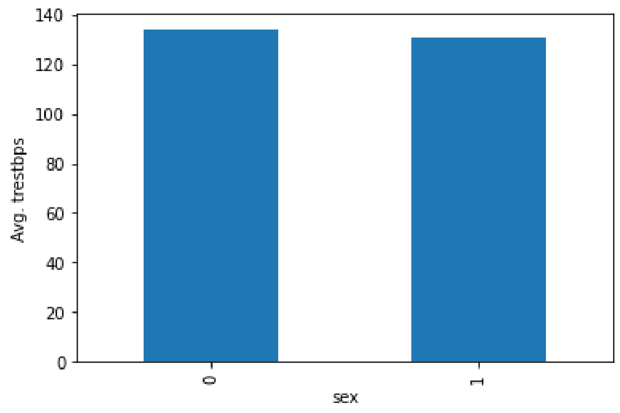


Fig. 5: Bar chart for Average resting BP against Sex.

```
## scatter plot with axes names
#colour the points by value of df.sex
#plot first the data points for df.sex of 0 and then 1
#setting colour to 'none' gives open circles
_,ax = plot.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
  subset_df = df[df.sex == catValue]
  ax.scatter(subset_df.age, subset_df.trestbps, color = 'none', edgecolor = color)
ax.set_xlabel('age')
ax.set_ylabel('trestbps')
ax.legend(['female 0', 'male 1'])
plot.show()
```

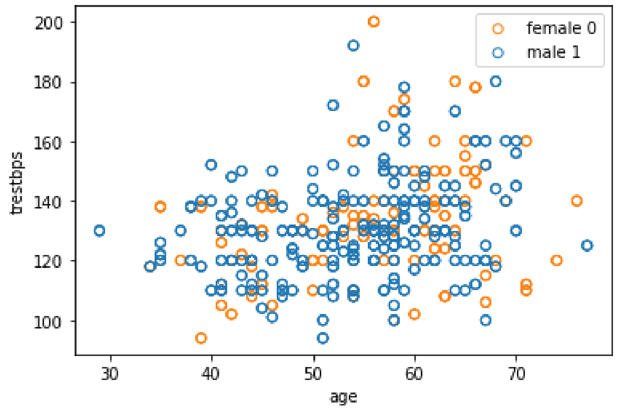


Fig. 6: Scatter plot of Resting BP against Age.

```
## scatter plot with axes names
_,ax = plot.subplots()
for catValue, color in (0, 'C1'), (1, 'C0'):
  subset_df = df[df.sex == catValue]
  ax.scatter(subset_df.age, subset_df.chol, color = 'none', edgecolor = color)
ax.set_xlabel('age')
ax.set_ylabel('chol')
ax.legend(['female 0', 'male 1'])
plot.show()
```

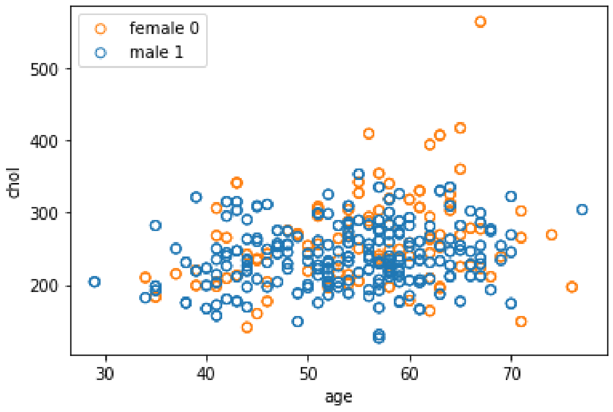


Fig. 7: Code and Output for Scatter plot of Cholesterol levels against Age.

Similarly, some distribution plots were derived to test for the measures of central tendency, shape and spread amongst some selected predictor variables. The aim of doing this in the research is to understand the distribution of values in the data and then know the level of importance of the features to our models. The “getdistprops” function was used to generate these measures as seen in the code lines in Figure 8. This function also shows the skewness and level of deviation of the features.

```
def getdistprops(seriestotest):
  out = {}
  normstat, normpvalue = scistat.shapiro(seriestotest)
  if (not math.isnan(normstat)):
    out['normstat'] = normstat
    if(normpvalue>=0.05):
      out['normpvalue'] = str(round(normpvalue, 2)) + ":Accept Normal"
    elif (normpvalue<0.05):
      out['normpvalue'] = str(round(normpvalue, 2)) + ": Reject Normal"
      out['mean'] = seriestotest.mean()
      out['median'] = seriestotest.median()
      out['std'] = seriestotest.std()
      out['kurtosis'] = seriestotest.kurtosis()
      out['skew'] = seriestotest.skew()
      out['count'] = seriestotest.count()
      return out

def dist_plot(data, variable, title):
  #function to plot a distribution plot
   sns.distplot(df[variable])
   plot.title(title)
   plot.show()
```

Fig. 8: Distribution Measures

The graphs of the distribution plots obtained along with the skewness values for some of the variables are then also shown below in Figures 9 to11.

```
dist_age = getdistprops(df.age)
print(dist_age)
```

```
{'normstat': 0.9843646287918091, 'normpvalue': '0.0: Reject Normal', 'mean': 54.43414634146342, 'median': 56.0, 'std': 9.072
290233244278, 'kurtosis': -0.5256178128684339, 'skew': -0.24886590174584555, 'count': 1025}
```

```
dist_plot(df, 'age', 'Distribution Plot for Age')
```

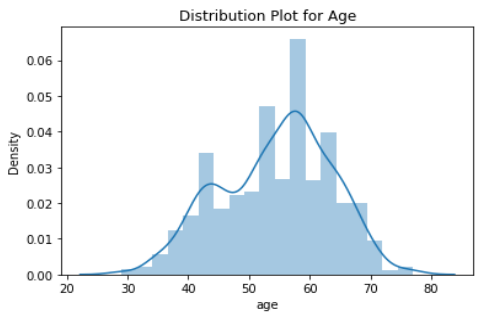


Fig. 9: Distribution plot for Age.

```
dist_trestbps = getdistprops(df.trestbps)
print(dist_trestbps)
```

```
{'normstat': 0.9633121490478516, 'normpvalue': '0.0: Reject Normal', 'mean': 131.61170731707318, 'median': 130.0, 'std': 17.
516718005376408, 'kurtosis': 0.9912207431245537, 'skew': 0.739768226050074, 'count': 1025}
```

```
dist_plot(df, 'trestbps', 'Distribution Plot for Resting Blood Pressure')
```

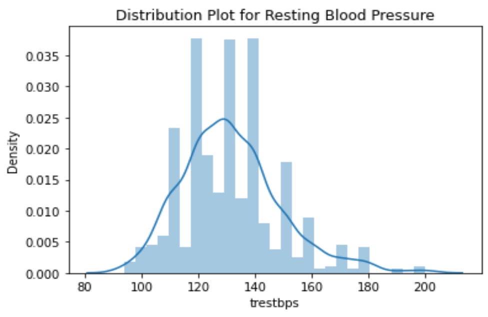


Fig. 10: Distribution Plot for Resting BP.

```
dist_exang = getdistprops(df.exang)
print(dist_exang)
```

```
{'normstat': 0.5965365171432495, 'normpvalue': '0.0: Reject Normal', 'mean': 0.33658536585365856, 'median': 0.0, 'std': 0.47
277237600371186, 'kurtosis': -1.5232047382747014, 'skew': 0.692655170469321, 'count': 1025}
```

```
dist_plot(df, 'exang', 'Distribution Plot for Exercise Induced Angina')
```

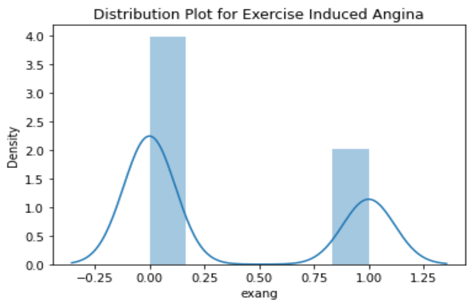


Fig. 11: Distribution Plot for Exercise Induced Angina.

Next, box plots of the variables were constructed to help find which features have outliers that could affect our models. It is very important to treat outliers in data to avoid possibility of bias in the modelling stage. The box plots derived before treating outliers is seen in Figure 12, while the one derived after treating the outliers is shown in Figure 14.

```
plot.figure(figsize=(35,7))
sns.boxplot(x="variable", y="value", data=pd.melt(df))
plot.title('Boxplot showing the distribution of the values for each of the features in the dataset')
plot.show()
```

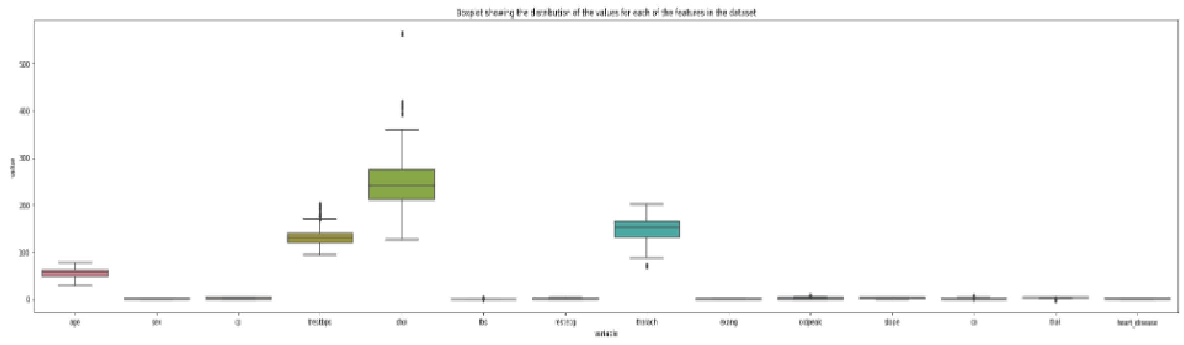


Fig. 12: Box Plot showing the features with outliers.

From the graph above, it is observed that the “trestbps”, “chol” and “thalach” are the three features which have very distinct number of outliers. The outliers in these three features were then trimmed using the code below in Figure 13, while the result of the trim is seen in Figure 14.

```
#Trimming the values of the specified columns to be between the 5th and 95th quantile with the code below
out_cols = ['chol', 'thalach', 'trestbps']
df[out_cols] = df[out_cols].clip(lower = df[out_cols].quantile(0.05),
                                upper = df[out_cols].quantile(.95), axis = 1)
```

Fig. 13: Trimming the Outliers.

```
plot.figure(figsize=(35,7))
sns.boxplot(x="variable", y="value", data=pd.melt(df))
plot.title('Boxplot showing the distribution of the values for each of the features in the dataset after treating outliers')
plot.show()
```

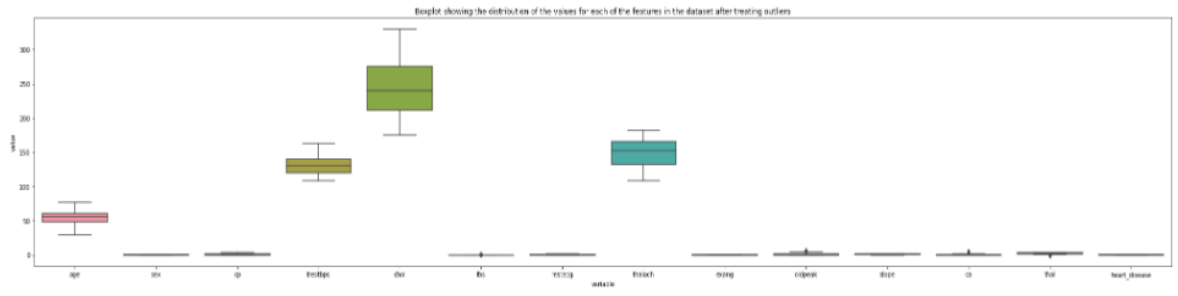
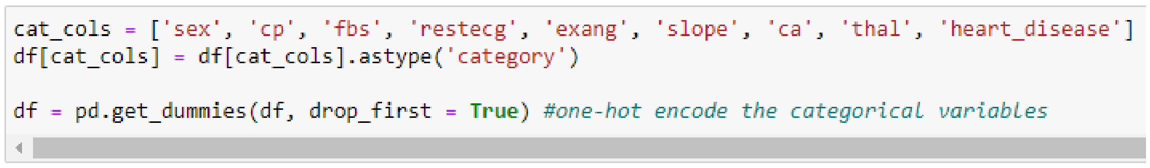


Fig. 14: Box Plot showing features without outliers.

## 3.3 Feature Engineering

Here, we changed the datatype of the categorical columns and then one-hot encode them. The categorical columns in our data include sex, cp, fbs, restecg, exang, slope, ca, thal, and heart_disease. This would generate more columns in our dataset by making each categorical data to have a column based on the number of its distinct values. The aim of doing this is to know the multicollinearity level of each of the features and the drop the ones who has little to no correlation with our target feature. The code for doing this is seen below in Figure 15.

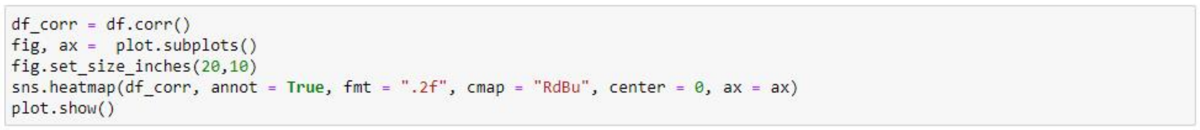


```
cat_cols = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal', 'heart_disease']
df[cat_cols] = df[cat_cols].astype('category')

df = pd.get_dummies(df, drop_first = True) #one-hot encode the categorical variables
```

Fig. 15: Feature Engineering

## 3.4 Correlation Analysis and Feature Selection

Correlation analysis is a good way to detect duplication of variables in the data. One way of finding redundancies in our data is to look at a correlation matrix. A correlation heatmap has been generated to try and identify strong correlations and multicollinearity. The correlation heatmap is seen in Figure 16.



```
df_corr = df.corr()
fig, ax =  plot.subplots()
fig.set_size_inches(20,10)
sns.heatmap(df_corr, annot = True, fmt = ".2f", cmap = "RdBu", center = 0, ax = ax)
plot.show()
```

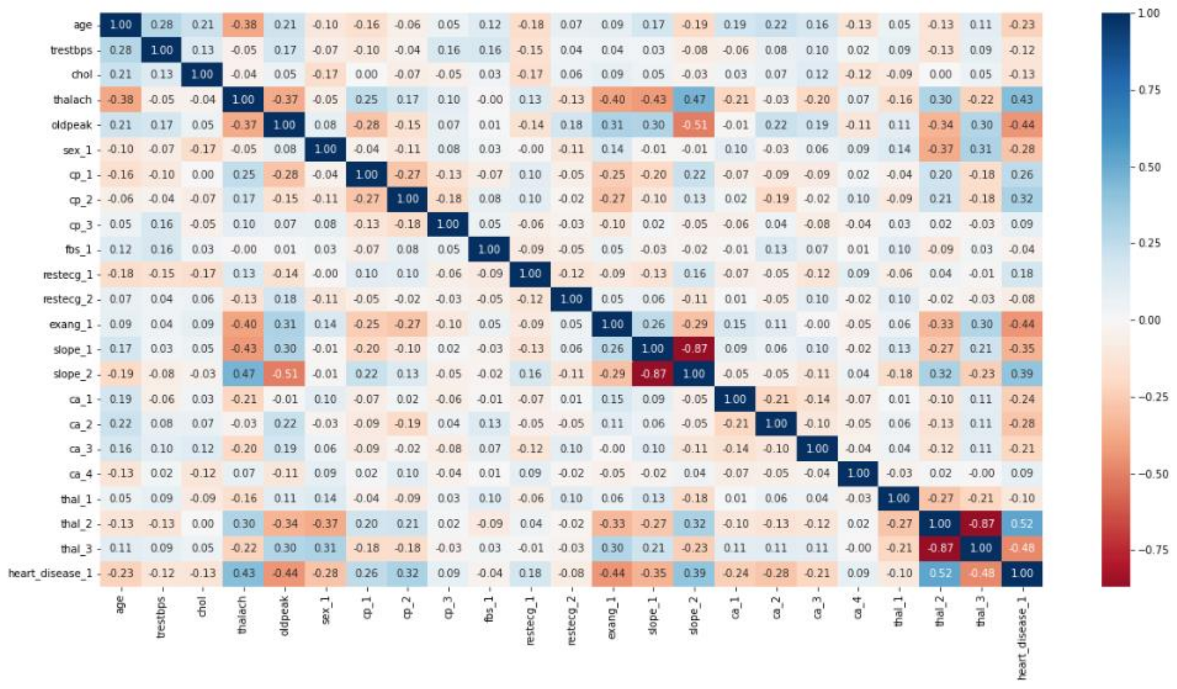


Fig. 16: Correlation Heatmap.

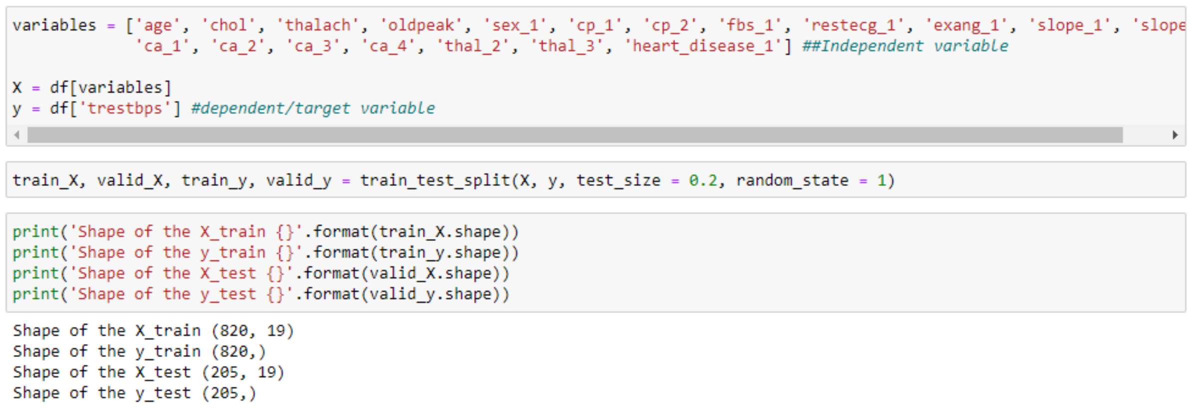
Using the correlation table, we focus on the correlation figures of predictors with the target variable (trestbps). We select only the predictors that reduces the effect of multicollinearity, and these variables are age, trestbps, chol, thalach, oldpeak, sex_1, cp_1, cp_2, fbs_1, restecg_1, exang_1, slope_1, slope_2, ca_1, ca_2, ca_3, ca_4, thal_2, thal_3, heart_disease_1. The listed independent variables are ones to be used for modelling.

# 4 Implementation of the Models.

## 4.1 Model Building

This section of the report describes the models that were used in the research. Here, we show the codes of how each of the models were built and the results of each of the models were then evaluated. The models used in this research include Decision Tree Regressor, ExtraTrees, Random Forest, Light GBM, XGBoost and CatBoost. After evaluation of the models, hyperparameter tuning was done on the best performing model which is CatBoost to further get a better performance from the model.

All the necessary libraries and models have been imported at the beginning, so the dataset is split into the training and test data first here. Then the shapes of the data that would be fed into the models are also seen below in Figure 17.



```
variables = ['age', 'chol', 'thalach', 'oldpeak', 'sex_1', 'cp_1', 'cp_2', 'fbs_1', 'restecg_1', 'exang_1', 'slope_1', 'slope
             'ca_1', 'ca_2', 'ca_3', 'ca_4', 'thal_2', 'thal_3', 'heart_disease_1'] ##Independent variable

X = df[variables]
y = df['trestbps'] #dependent/target variable
```

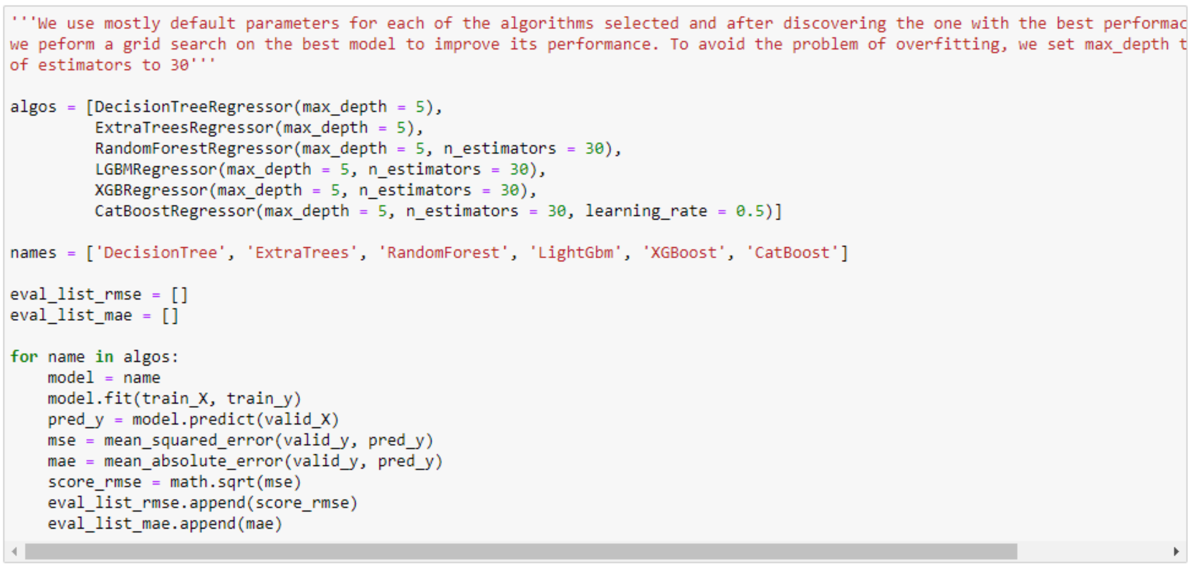
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size = 0.2, random_state = 1)
```

```
print('Shape of the X_train {}'.format(train_X.shape))
print('Shape of the y_train {}'.format(train_y.shape))
print('Shape of the X_test {}'.format(valid_X.shape))
print('Shape of the y_test {}'.format(valid_y.shape))
```

```
Shape of the X_train (820, 19)
Shape of the y_train (820,)
Shape of the X_test (205, 19)
Shape of the y_test (205,)
```

Fig. 17: Splitting of the dataset

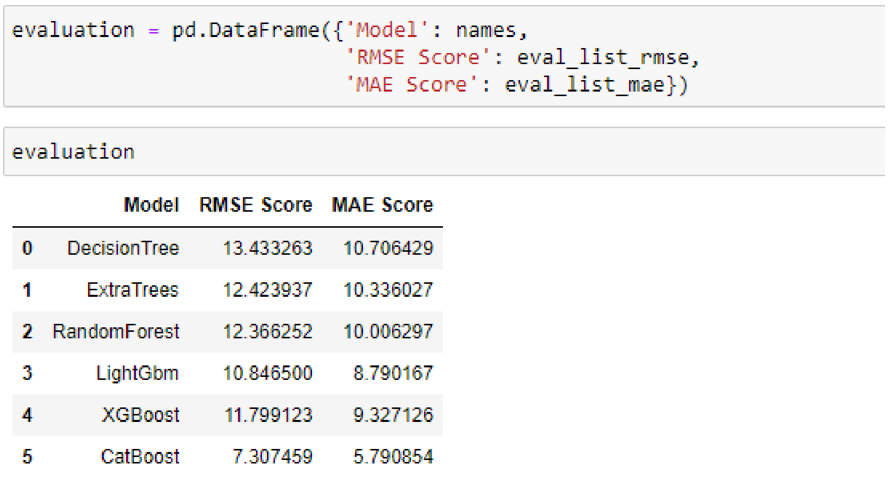
The code for building the six models is shown below in Figure 18.



```python
'''We use mostly default parameters for each of the algorithms selected and after discovering the one with the best performac
we peform a grid search on the best model to improve its performance. To avoid the problem of overfitting, we set max_depth t
of estimators to 30'''

algos = [DecisionTreeRegressor(max_depth = 5),
         ExtraTreesRegressor(max_depth = 5),
         RandomForestRegressor(max_depth = 5, n_estimators = 30),
         LGBMRegressor(max_depth = 5, n_estimators = 30),
         XGBRegressor(max_depth = 5, n_estimators = 30),
         CatBoostRegressor(max_depth = 5, n_estimators = 30, learning_rate = 0.5)]

names = ['DecisionTree', 'ExtraTrees', 'RandomForest', 'LightGbm', 'XGBoost', 'CatBoost']

eval_list_rmse = []
eval_list_mae = []

for name in algos:
    model = name
    model.fit(train_X, train_y)
    pred_y = model.predict(valid_X)
    mse = mean_squared_error(valid_y, pred_y)
    mae = mean_absolute_error(valid_y, pred_y)
    score_rmse = math.sqrt(mse)
    eval_list_rmse.append(score_rmse)
    eval_list_mae.append(mae)
```

Fig. 18: Building of the Model.

The results of the evaluation of the models are then shown below in Figure 19.



```python
evaluation = pd.DataFrame({'Model': names,
                           'RMSE Score': eval_list_rmse,
                           'MAE Score': eval_list_mae})
```

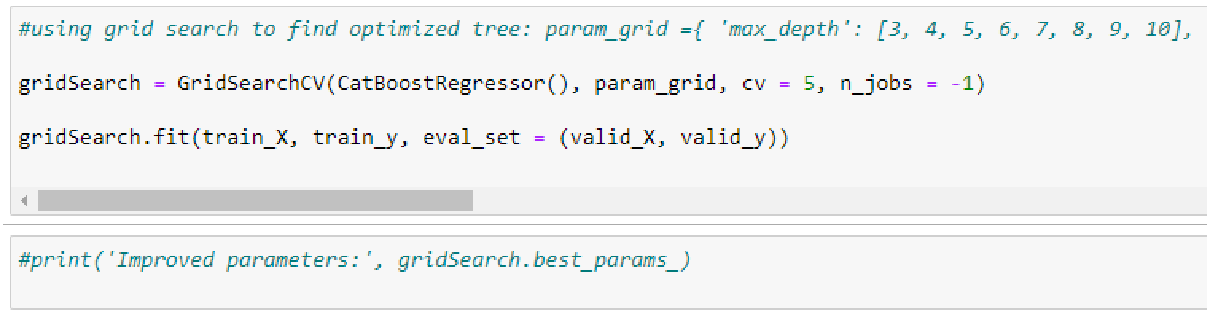
```python
evaluation
```

| | Model | RMSE Score | MAE Score |
|---|---|---|---|
| 0 | DecisionTree | 13.433263 | 10.706429 |
| 1 | ExtraTrees | 12.423937 | 10.336027 |
| 2 | RandomForest | 12.366252 | 10.006297 |
| 3 | LightGbm | 10.846500 | 8.790167 |
| 4 | XGBoost | 11.799123 | 9.327126 |
| 5 | CatBoost | 7.307459 | 5.790854 |

Fig. 19: Results of the Data Modelling.

## 4.2 Hyperparameter Optimization

From the results gotten using the RMSE and MAE metrics, CatBoost gave the best performance of all the models used. We then applied hyperparameter optimization using Grid Search on this model for better performance. The code used to obtain the parameters for the hyperparameter optimization is shown below in Figure 20.

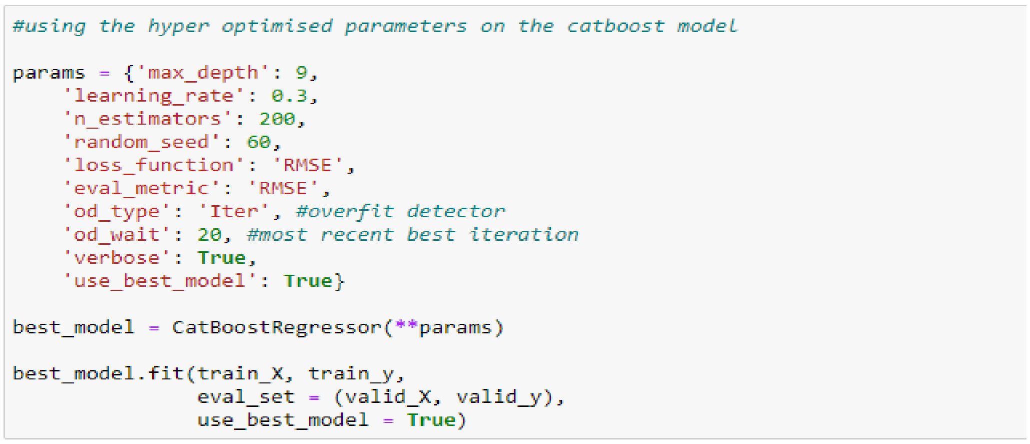


```
#using grid search to find optimized tree: param_grid ={ 'max_depth': [3, 4, 5, 6, 7, 8, 9, 10],

gridSearch = GridSearchCV(CatBoostRegressor(), param_grid, cv = 5, n_jobs = -1)

gridSearch.fit(train_X, train_y, eval_set = (valid_X, valid_y))
```

```
#print('Improved parameters:', gridSearch.best_params_)
```

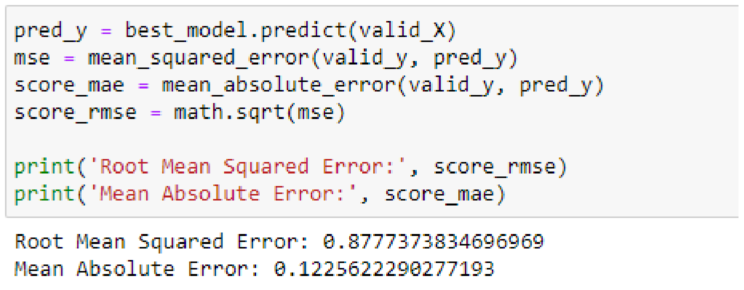
Fig. 20: Obtaining Improved parameters for the Optimization.

Then we would run the hyperparameter optimization using the new improved parameters as shown below in Figure 21. After that, the model is re-evaluated to obtain the new values of RMSE and MAE for the CatBoost model. The code for that is seen in Figure 22.



```
#using the hyper optimised parameters on the catboost model

params = {'max_depth': 9,
     'learning_rate': 0.3,
     'n_estimators': 200,
     'random_seed': 60,
     'loss_function': 'RMSE',
     'eval_metric': 'RMSE',
     'od_type': 'Iter', #overfit detector
     'od_wait': 20, #most recent best iteration
     'verbose': True,
     'use_best_model': True}

best_model = CatBoostRegressor(**params)

best_model.fit(train_X, train_y,
               eval_set = (valid_X, valid_y),
               use_best_model = True)
```

Fig. 21: Hyperparameter optimization of the model.



```
pred_y = best_model.predict(valid_X)
mse = mean_squared_error(valid_y, pred_y)
score_mae = mean_absolute_error(valid_y, pred_y)
score_rmse = math.sqrt(mse)

print('Root Mean Squared Error:', score_rmse)
print('Mean Absolute Error:', score_mae)
```

```
Root Mean Squared Error: 0.8777373834696969
Mean Absolute Error: 0.1225622290277193
```

Fig. 22: Re-evaluation of the Final Model.

From the figure above, we can see how the performance of the model has been greatly improved to obtain an RMSE value of 0.8777 and MAE value of 0.1225 as the closer the values are to zero, the better the performance of the model.

Finally, we looked at the most important features for predicting blood pressure levels, and the features who are most important are shown accordingly in the chart below in Figure 23.

```
fea_imp = pd.DataFrame({'imp':best_model.feature_importances_, 'col': X.columns})
fea_imp = fea_imp.sort_values(['imp', 'col'], ascending=[True, False]).iloc[-35:]
_ = fea_imp.plot(kind='barh', x='col', y='imp', figsize=(10, 5))
plot.title('Feature Importance of Predictors with Respect to the Target Variable(trestbps)')
plot.show()
```

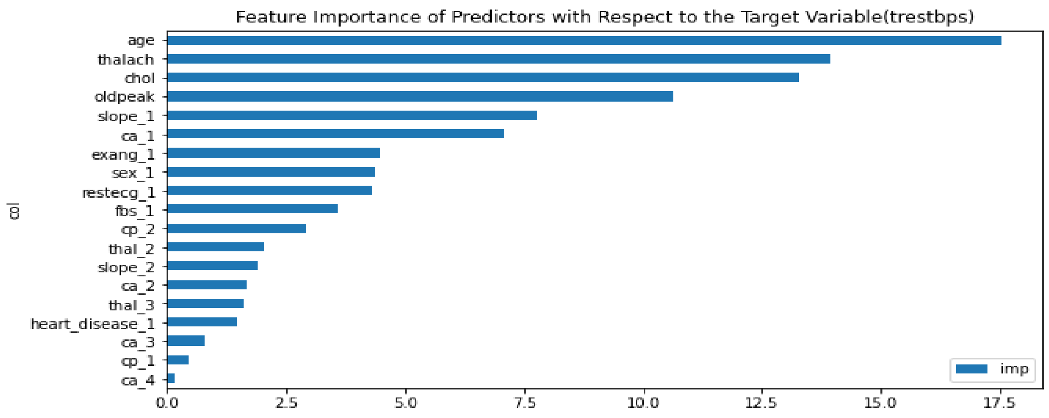


Fig. 23: Feature Importance of Predictors.