Name: Subramanian Venkataraman

Date: Apr-21-2021

Project : Disease Prediction using Machine Learning Algorithms

Description: This project predicts the factors that are responsible for the disease from the existing set of attributes of the table.

Following are the major factors which are affecting the health and leads to disease.

1. Age
2. Cholestrol_too high
3. Weight
4. Glucose too High
5. Smoke
6. Alcohcol

We have used the following machine learning algorithms to predict the factors for the disease.

1 GradientBoostingClassifier 2 RandomForestClassifier 3 KNeighborsClassifier 4 GaussianNB

Of which GradientBoosting provides the highest performance with the accuracy 80.74%.

```
# Dataframes
import numpy as np
import pandas as pd

# Graph libraries
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
import matplotlib.mlab as mlab

# binarize the column values

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import neighbors
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb
from xgboost import XGBClassifier
```

```
# Algorithm Evaluation
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score



#For evaluation
from math import sqrt
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV

from sklearn.neighbors import KNeighborsRegressor

# For KNN - impriving accuracy using scaling approach.
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import roc_auc_score

import sys; sys.path
from time import time
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import (RandomForestClassifier,
                              AdaBoostClassifier,
                              GradientBoostingClassifier,
                              HistGradientBoostingClassifier)
import xgboost as xgb
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
```

1. Data Preparation

```
df=pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/Disease Prediction T
```

```
df.head(5)
```

	Age	Gender	Height	Weight	High Blood Pressure	Low Blood Pressure	Cholesterol	Glucose	Smoke
0	59	female	167	88.0	130	68	normal	normal	0
1	64	female	150	71.0	140	100	normal	normal	0
2	41	female	166	83.0	100	70	normal	normal	0
3	50	male	172	110.0	130	80	normal	normal	1

1.1 Feature Engineering

Shape provides the 2D view of the dataframe

```
print("The shape of our feature is:", df.shape)
```

```
The shape of our feature is: (49000, 12)
```

1.2 Describe

Describe provides the count of the values, mean,std, min, percentile and max values of the data

```
df.describe()
```

	Age	Height	Weight	High Blood Pressure	Low Blood Pressure	Smo
count	49000.000000	49000.000000	49000.000000	49000.000000	49000.000000	49000.0000
mean	52.853306	164.366878	74.190527	128.698939	96.917367	0.0882
std	6.763065	8.216637	14.329934	147.624582	200.368069	0.2836
min	29.000000	55.000000	10.000000	-150.000000	0.000000	0.0000
25%	48.000000	159.000000	65.000000	120.000000	80.000000	0.0000
50%	53.000000	165.000000	72.000000	120.000000	80.000000	0.0000
75%	58.000000	170.000000	82.000000	140.000000	90.000000	0.0000

If you look at the High Blood pressure column, the min value is -150. There is no scope for Bllood Pressure to show in negative number. We will convert it to positive value, by assuming that the root cause of the issue is either data entry or data manipulation at source side

Display top 5 rows to study the data values

```
df.head(5)
```

	Age	Gender	Height	Weight	High Blood Pressure	Low Blood Pressure	Cholesterol	Glucose	Smoke
0	59	female	167	88.0	130	68	normal	normal	0
1	64	female	150	71.0	140	100	normal	normal	0
2	41	female	166	83.0	100	70	normal	normal	0
3	50	male	172	110.0	130	80	normal	normal	1

Let me know the column names of the data frame

```
# display column names
columns= list(df)
print(columns)
```

```
['Age', 'Gender', 'Height', 'Weight', 'High Blood Pressure', 'Low Blood Pressu
```

Display the null counts null for each column of the data frame. There are no null values in any of the column as per the report below.

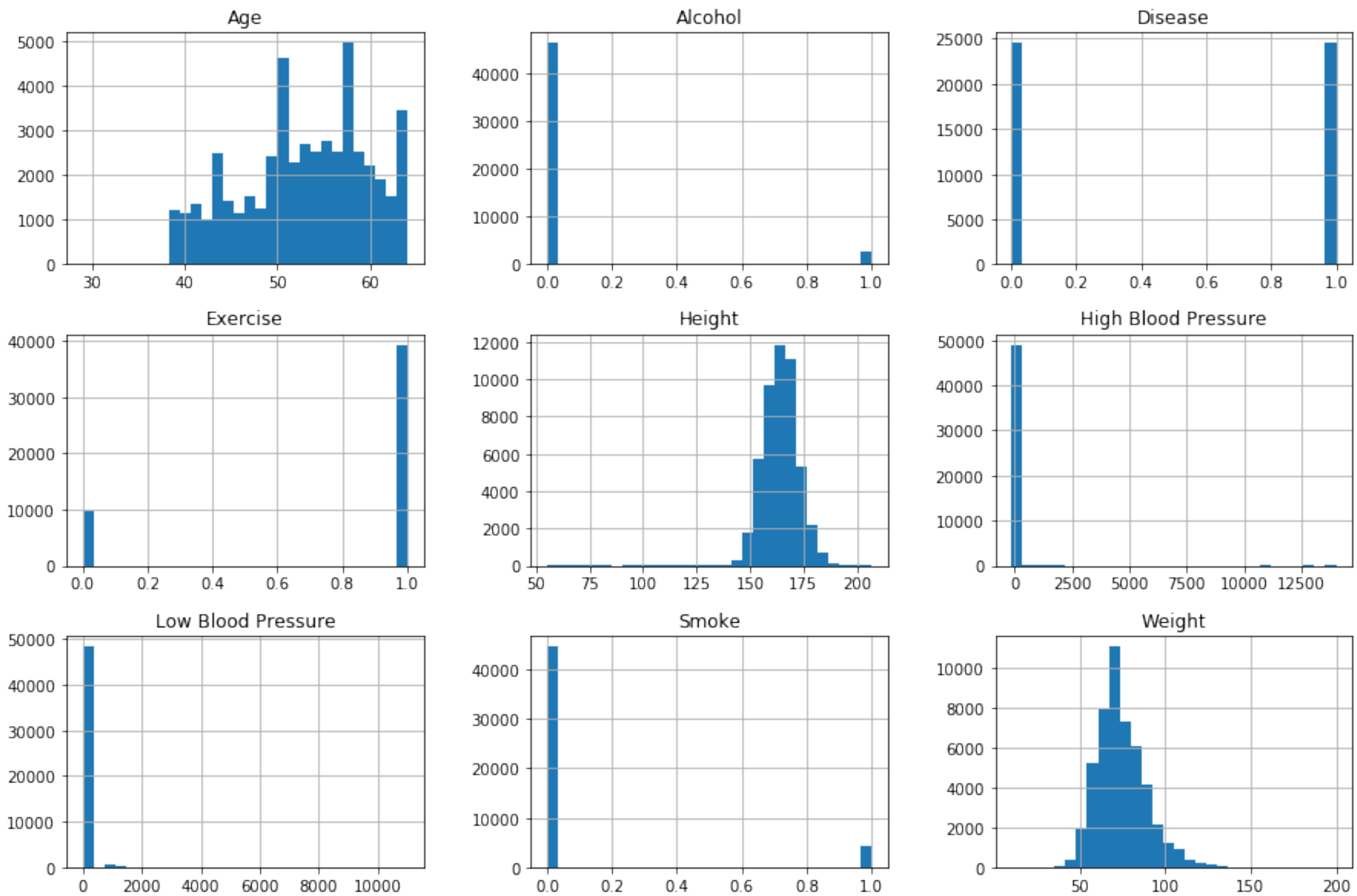
```
df.isnull().sum().sort_values(ascending=False)
```

```
Disease                0
Exercise               0
Alcohol                0
Smoke                  0
Glucose                0
Cholesterol            0
Low Blood Pressure     0
High Blood Pressure    0
Weight                 0
Height                 0
Gender                 0
Age                    0
dtype: int64
```

Display histogram for all columns

```
df.hist(bins=30, figsize=(15, 10))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7faf712aa278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf711f5828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf7124add8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf711883c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf711b4908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf6c9d3eb8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf6c1634a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf6f435a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf6f435ac8>]],
      dtype=object)
```



1.2 Data Cleaning

The categorical columns Gender , Cholestrol and Glocose have categorical values. ie, the values of these column belongs to a defined efined category. We need to create one columns for each category type type against each column name. For example, the column name Gender have two values Male and Female. In order to clearly identify which gender has impact the target attriubute , Disease, we will create two columns gender_male and gender_female where the binary values will be applied as 1 or 0.

1.2.1 Create categorical variables for Cholesterol and Glucose

```
df=pd.get_dummies(df, columns=["Gender","Cholesterol","Glucose"])
```

```
df.head(5)
```

	Age	Height	Weight	High Blood Pressure	Low Blood Pressure	Smoke	Alcohol	Exercise	Disease	G
0	59	167	88.0	130	68	0	0	1	0	
1	64	150	71.0	140	100	0	0	0	1	
2	41	166	83.0	100	70	0	1	1	0	
3	50	172	110.0	130	80	1	0	1	0	
4	39	162	61.0	110	80	0	0	1	0	

```
# display column names
columns= list(df)
print(columns)
```

```
['Age', 'Height', 'Weight', 'High Blood Pressure', 'Low Blood Pressure', 'Smol
```

1.2.2 Move Disease to last column for since it is a dependent value

```
# Move Disease to last column position
df1 = df.pop('Disease') # remove column Disease and store it in df1
df['Disease']=df1 # add Disease series as a 'new' column.
```

1.2.3 Review all columns for boundary values

```
# Print unique values for all columns
for (columnName, columnData) in df.iteritems():
    print('Colunm Name : ', columnName)
    print('Unique values : ', columnData.unique() )

# Below, we can notice that 'High Blood Pressure'  column has -ve values which is
```

```
Colunm Name :  Age
Unique values :  [59 64 41 50 39 54 48 51 42 56 63 52 45 58 57 49 43 46 62 5
 60 40 29 30]
Colunm Name :  Height
Unique values :  [167 150 166 172 162 163 159 171 161 170 165 168 178 156 16
 153 154 151 175 164 149 169 174 179 180 173 176 158 177 183 185 157 181
 145 152 198 184 148 120 188 186 130 144 147  98 190 187 146 133 143 140
 189 142 193  59 192 197 131 195 135  70  55 110 191  68 138 134 108 132
 100 194 109 137 128 111  67 125  91  75  96 117  81 207  72 196 136  71
 105  76 122 119  99  60 139  57  65 104  66]
Colunm Name :  Weight
Unique values :  [ 88.    71.    83.   110.    61.    89.    72.    43.    7
  85.   106.    68.    74.    80.    65.    76.    51.   119.    86.
  81.    69.    67.    95.    84.5   78.    64.    66.    90.    96.
 121.    73.    60.    82.   100.    53.    46.    94.    59.    62.
 103.   120.    99.    70.   105.    58.    93.    40.    79.    45.
  92.    57.   114.    91.    55.    48.    84.    77.    97.    50.
 108.   135.    52.    87.   126.    54.    56.    47.   112.   102.
 123.    49.   104.    98.   101.   125.    67.5  107.    42.    41.
  59.5  141.    80.8  131.   116.    62.5  109.   127.   130.   150.
  44.   168.   148.   115.    39.    51.5  122.    79.94  71.3   67.9
  78.5   34.    30.   118.   138.   111.   117.   170.    36.   134.
  89.1   61.5   66.5   68.5  124.    76.5   82.5   70.3  200.   113.
  38.    73.5   71.5  105.5  136.   133.   128.    68.2  144.    57.6
  31.   149.    74.3   64.5   29.    69.5  140.    37.   146.   153.
 129.    64.7   35.   147.    60.5  132.    82.1   10.    33.    63.8
  79.5   88.5  139.    84.3  106.4   94.7   89.9  143.    92.2   59.2
  50.7  165.    23.    53.2   73.2   90.5  160.   145.    75.5   84.9
  52.3   28.    62.4   60.6   78.2   56.2  121.8  156.    76.7   84.8
 154.   155.    32.    73.8   35.45  53.3   63.4   54.35  82.3   64.1
  72.1   57.8  164.   114.6   80.7   84.6   70.2  171.    53.9   80.5
  65.3   75.6  137.   178.   162.    53.6   74.2  167.   177.    72.5
  65.5  180.    58.5   84.7   83.5   74.5   85.5  158.    70.5   67.8
  96.5  142.   181.   152.    62.3   64.3   80.6  161.    99.9   55.4
  69.8   94.5   81.1   11.    75.2  109.7   61.2   55.2  175.    70.8
  86.5  166.   121.3   68.4   45.8  159.    61.3   22.  ]
```

```
Colunm Name :  High Blood Pressure
Unique values :  [  130   140   100   110   120   150    90   160   155   14
   200   169   190   170   105   125   126   128   153   124    12    80
   141   220   115   134   135    99   165   172    11   191   133 13010
    95   132   147   210   119   144    85    14   148   187   163   103
   143   121   117   127   175   137   138   139   118  1420   146   101
   113   185   149   108   151    93   131   168   129 11020   123   106
  1400   176   156    13    15   162   122   111   230   116   171    97
   907   240   161   112   154   166   109     1   152   136   159   158
   142  1130 14020   178   157   164    70   174   188   104   167   114
  1202    20   179   960   102   196  -140    10   906   107  2000  1500
   207   701  -120   177    17  -100   181  -150  -115  1110  1300   202
   215   195   401   902   199    16   173   909    96  1205     7]
Colunm Name :  Low Blood Pressure
Unique values :  [   68   100    70    80    90    60    91  1000   120    6
    95    86    59    69   130  1100    83    85   160   140  1008    89
   110  1177    75   106    99    57    65    94   105    76    77    82
    96  1007    78 11000   101    84   112   107   180    72    40    20
    74    67  5700    50    81    62    88  1200   109   710   802    73
    64    92    98    56  9100  1125  1120  1110   103     0  9011   820
    71   150    97    93  8100     6   700  1088    87 10000   108  1003
  1002    10    66   190  1101   809   111     8    30   170  8099  8000
```

We can notice above that the column name 'High Blood Pressure' has negative values. We will convert them to positive now.

1.2.4 Convert High Blood Pressure column to absolute values.

```
# Print unique values for all columns
df['High Blood Pressure'].unique()
```

```
array([  130,   140,   100,   110,   120,   150,    90,   160,   155,
         145,   180,  1409,   200,   169,   190,   170,   105,   125,
         126,   128,   153,   124,    12,    80,   141,   220,   115,
         134,   135,    99,   165,   172,    11,   191,   133, 13010,
          95,   132,   147,   210,   119,   144,    85,    14,   148,
         187,   163,   103,   143,   121,   117,   127,   175,   137,
         138,   139,   118,  1420,   146,   101,   113,   185,   149,
         108,   151,    93,   131,   168,   129, 11020,   123,   106,
        1400,   176,   156,    13,    15,   162,   122,   111,   230,
         116,   171,    97,   907,   240,   161,   112,   154,   166,
         109,     1,   152,   136,   159,   158,   142,  1130, 14020,
         178,   157,   164,    70,   174,   188,   104,   167,   114,
        1202,    20,   179,   960,   102,   196,  -140,    10,   906,
         107,  2000,  1500,   207,   701,  -120,   177,    17,  -100,
         181,  -150,  -115,  1110,  1300,   202,   215,   195,   401,
         902,   199,    16,   173,   909,    96,  1205,     7])
```

```
df['High Blood Pressure'] = df['High Blood Pressure'].abs()
```

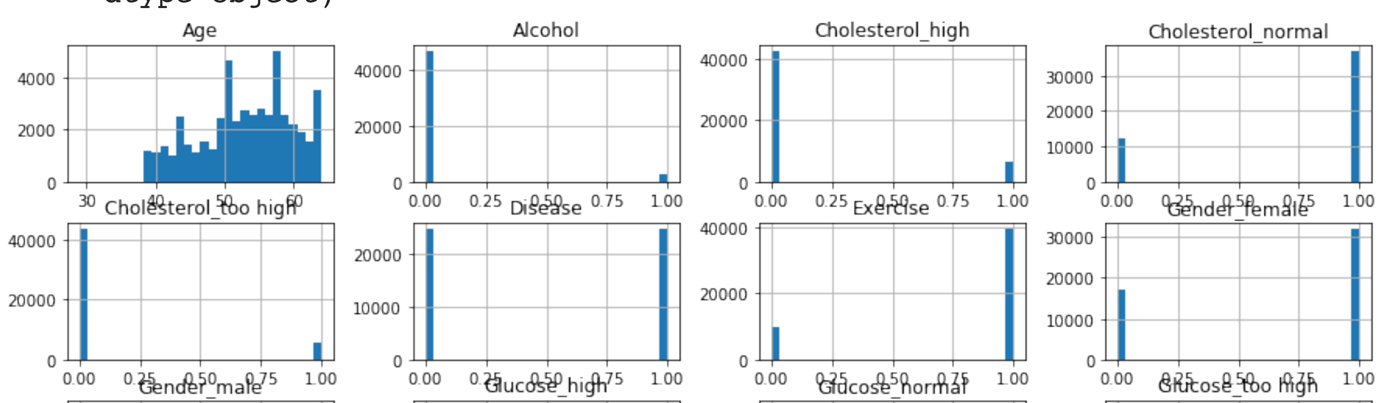
Now all -ve values of High Blood Pressure are converted to positive.

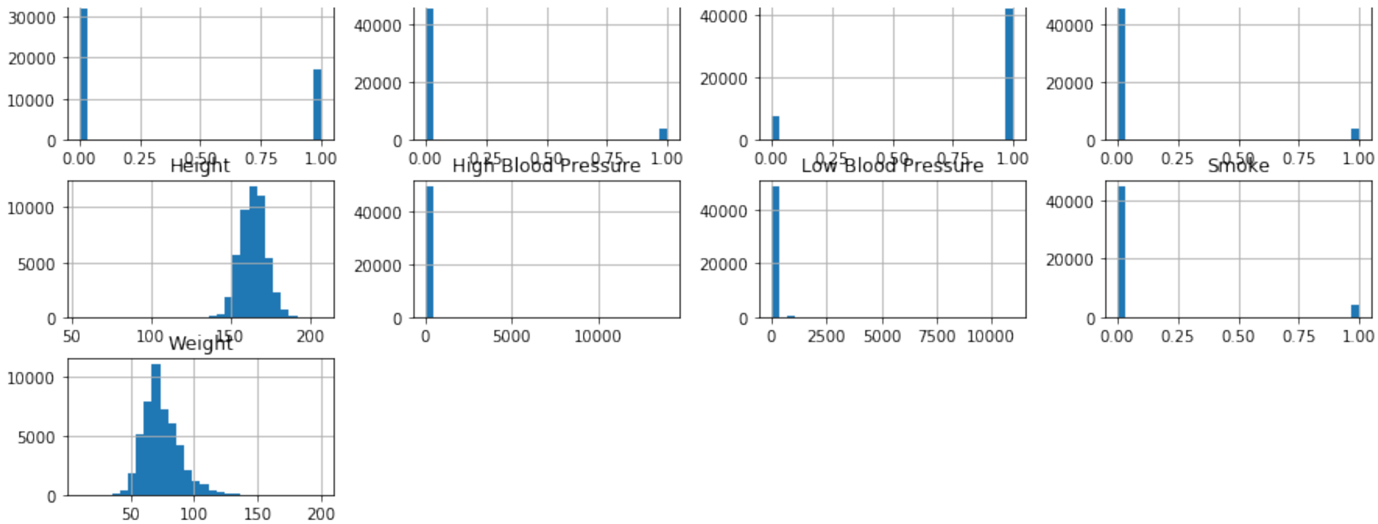
```
df['High Blood Pressure'].describe()
```

```
count    49000.000000
mean       128.733429
std        147.594506
min          1.000000
25%        120.000000
50%        120.000000
75%        140.000000
max      14020.000000
Name: High Blood Pressure, dtype: float64
```

```
df.hist(bins=30, figsize=(15, 10))
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7faf693c7da0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf705c8128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf705e6278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf7095c3c8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf7098f518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf709c4668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf709f67b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70a2b940>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf70a2b978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70a95ba8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70ac9cf8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70afce48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf70b31f98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70b71470>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf70ba0a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf71030fd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7faf7106b5c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf710a0b70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf710dc160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7faf7110c710>]],
      dtype=object)
```

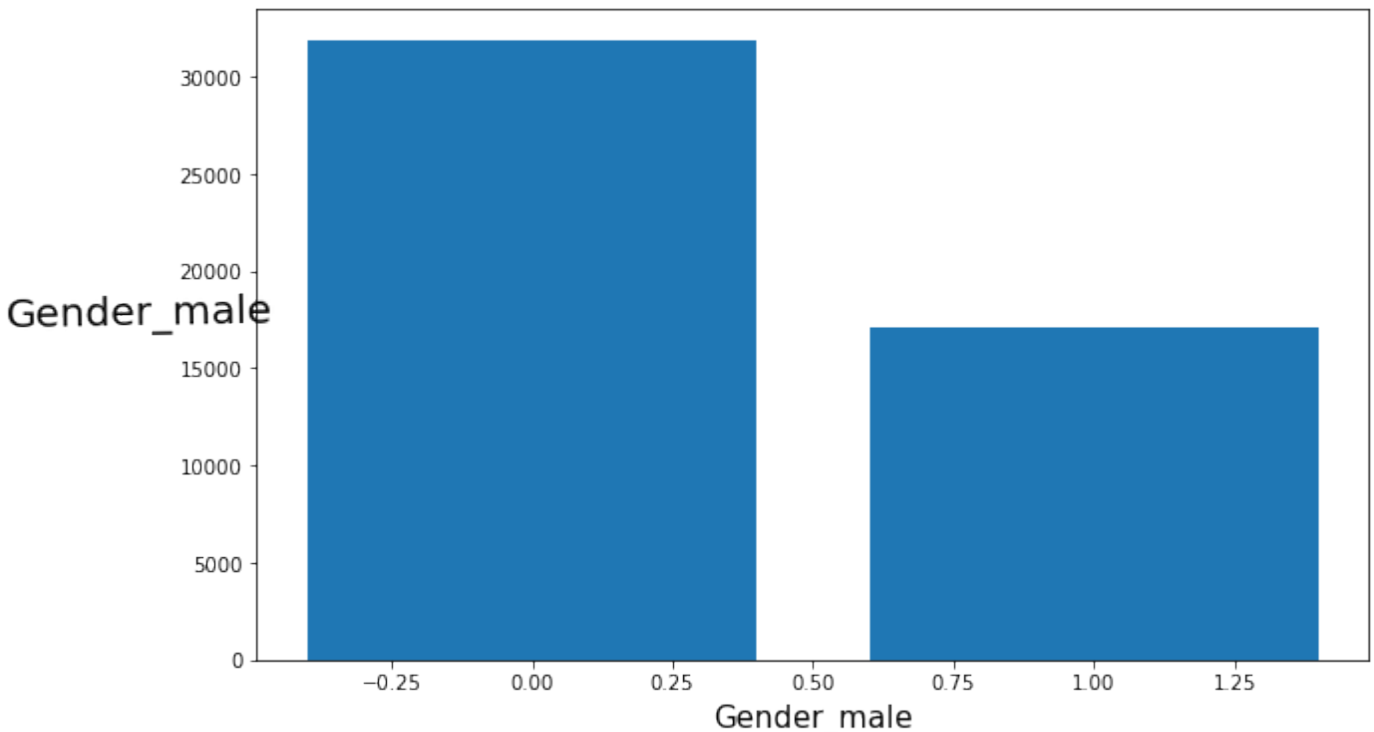




Height
High Blood Pressure
Low Blood Pressure
Smoke
Weight

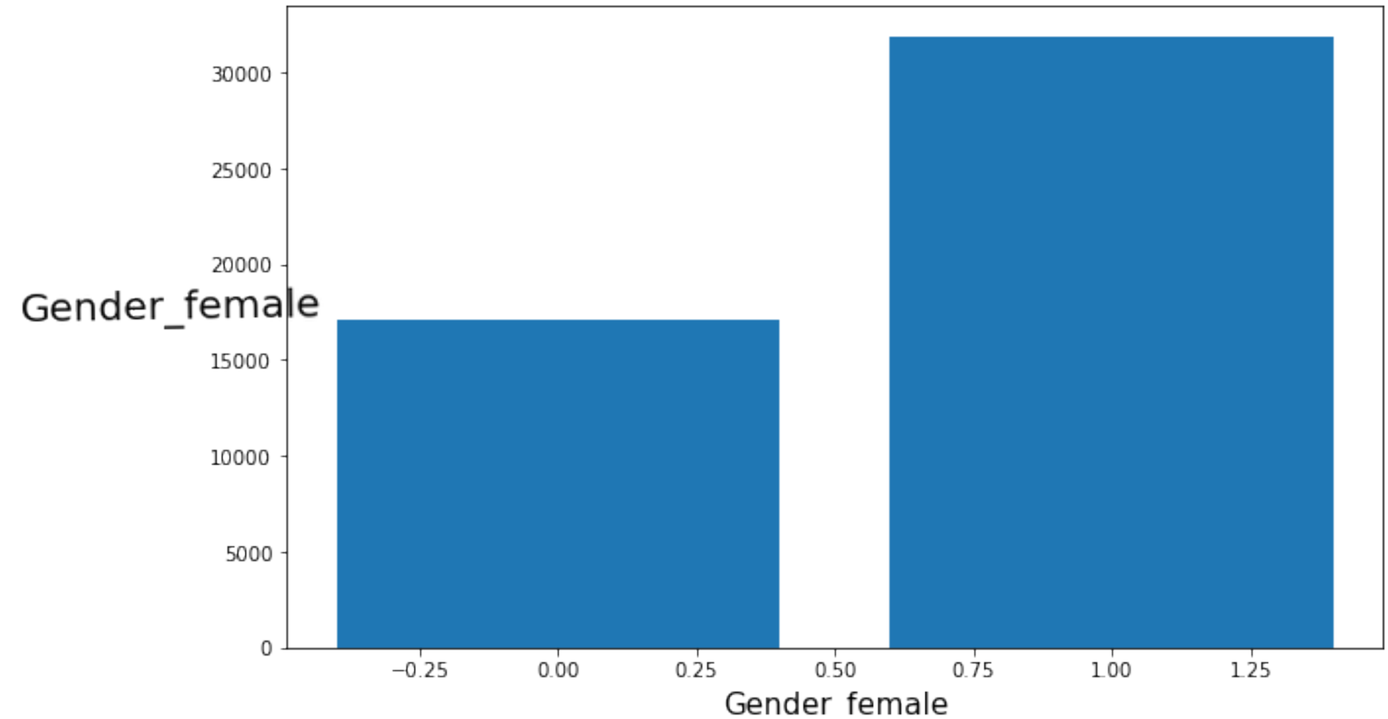
```
# Ratio between Gender and disease

df1 = df.groupby(['Gender_male'])['Disease'].count().to_frame('Disease').reset_in
df1=df1.sort_values(by='Disease', ascending=False)
plt.figure(figsize=(10,6))
# make bar plot with matplotlib
plt.bar('Gender_male', 'Disease',data=df1)
plt.xlabel("Gender_male", size=15)
plt.ylabel("Gender_male", rotation=1, fontsize=20, labelpad=20)
```

```
Text(0, 0.5, 'Gender_male')
```

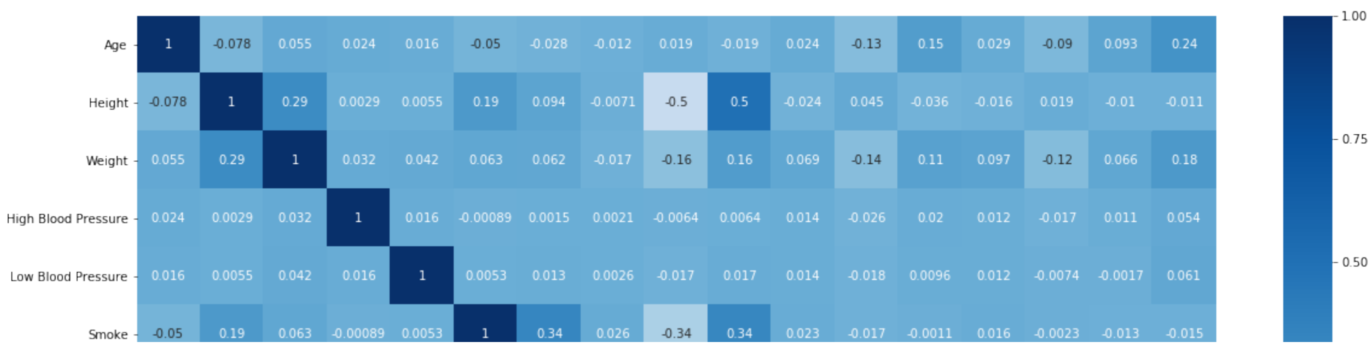


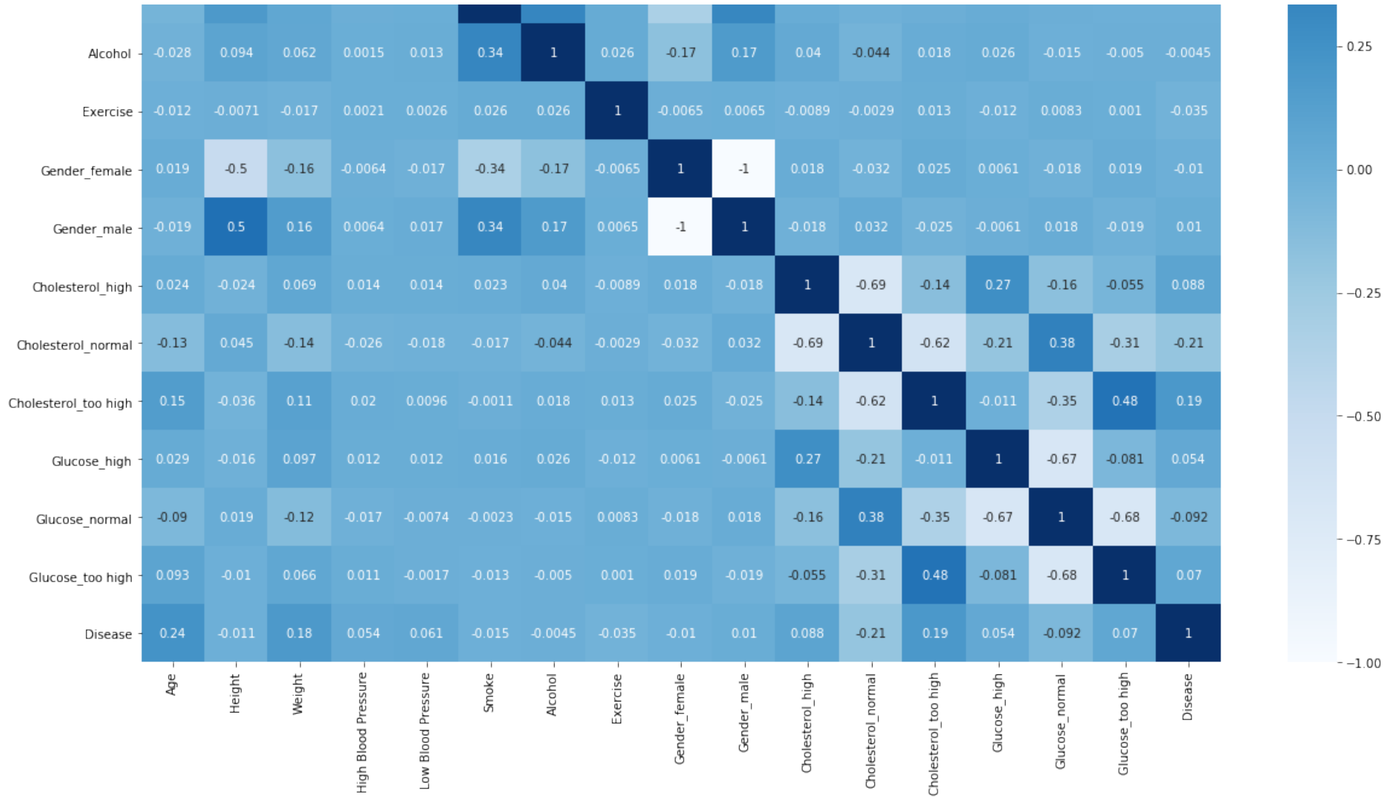
```
# Ratio between Gende_femaler and disease

df1 = df.groupby(['Gender_female'])['Disease'].count().to_frame('Disease').reset_
df1=df1.sort_values(by='Disease', ascending=False)
plt.figure(figsize=(10,6))
# make bar plot with matplotlib
plt.bar('Gender_female', 'Disease',data=df1)
plt.xlabel("Gender_female", size=15)
plt.ylabel("Gender_female", rotation=1, fontsize=20, labelpad=20)
```

```
Text(0, 0.5, 'Gender_female')
```




```
features = df.columns
X = df[features]

plt.subplots(figsize=(20, 15))
sns.heatmap(X.corr(), annot=True, cmap='Blues')
plt.show()
```




Alcohol
Exercise
Gender_female
Gender_male
Cholesterol_high
Cholesterol_normal
Cholesterol_too high
Glucose_high
Glucose_normal
Glucose_too high
Disease
Age
Height
Weight
High Blood Pressure
Low Blood Pressure
Smoke
Alcohol
Exercise
Gender_female
Gender_male
Cholesterol_high
Cholesterol_normal
Cholesterol_too high
Glucose_high
Glucose_normal
Glucose_too high
Disease
0.25
0.00
−0.25
−0.50
−0.75
−1.00

From the above heat map, we can conclude that the following features have high correleation with disease

1. Age - 0.24
2. Cholestrol_too high - 0.19
3. Weight - 0.18
4. Glucose too High - 0.07
5. Smoke - -0.015 -> Negatively correleates with disease
6. Alcohcol - -0.0045 -> Negatively correleates with disease
7. Height - -0.011 -> Negatively correleates with disease

2. Machine Learning Models

2.1 Navie Bayes Models

2.1.1 Gaussian Navie Bayes model

The Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.
GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

```
# Gaussian Navie Bayes model
X = df.drop(['Disease'], axis=1)
y = df["Disease"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
y_pred = gaussian.predict(X_test)
acc_gaussian = round(accuracy_score(y_pred, y_test) * 100, 2)
print("GaussianNB gives accuracy of", acc_gaussian)
```

```
    GaussianNB gives accuracy of 60.04
```

The Gaussian Naive Bayes algorithm provides the accuracy of 60%.

BernoulliNB implements the naive Bayes training and classification algorithms for multivariate Bernoulli distributions. It expects the input data to be in 0s and 1s. If the input data is not in the format, it binarize the data before applying the model.

2.2 Random Forest Machine Learning Model

The random forest uses many trees, and it it predicts the target values by averaging each component of the tree. Generally, the accuracy of Random Forest is better than Deccision Trees.

2.2.1 Algorithm implementation

```
# Random Forest

X = df.drop(['Disease'], axis=1)
y = df["Disease"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

rf = RandomForestClassifier(n_estimators=100)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
base_accuracy = round(accuracy_score(y_pred, y_test) * 100, 2)
print("Random Forest gives accuracy of", base_accuracy)
```

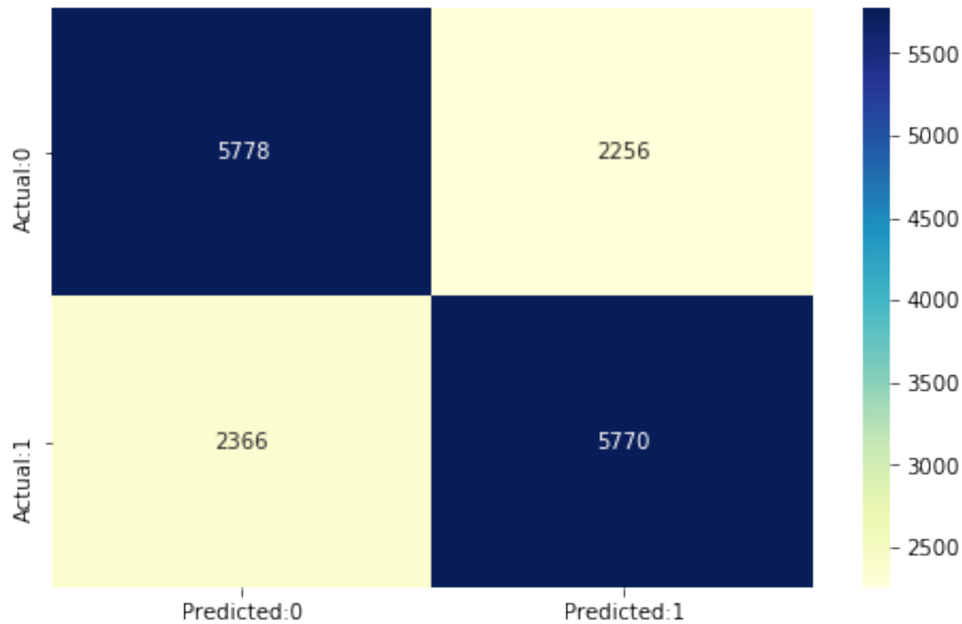
```
Random Forest gives accuracy of 71.42
```

2.2.2 Confusion Matrix

Confusion Matrix provides how is our prediction rate in terms of
TP - How many we predicted positive is True
TN - How many we predicted negative is True
FP - How many we predicted positive is False
FN - How many we predicted negative is False
Lets go ahead with Confusion Matrix and finding Hyperparameters.

```
cm=confusion_matrix(y_test,y_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Ac
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faf73509f28>
```



If we look at the above diagram, the cells highlighted in blue are TP and TN and are predicted correctly.
where as the cells highlighted in yellow are FP and FN which were not predicted correctly.

2.2.3 Classification Report

The classifcation report provides Recall, Precision, Accuracy and F-Score which are explained below.

1. Recall : Recall indicates that from the overall positive values, how much were predicted correctly. It should be higher as much as possible.
2. Precision : From the overall positive classes, how many were predicted correctly.
3. Accuracy : From the overall classes (including positive and negative) how much were predicted correctly
4. F1-measure : F-score helps to meaure precision and recall at the same time by using mean values.

   From the below report, we can notices that the accuracy and F1-Score lies at 71%.

```
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
    Classification Report
                  precision    recall  f1-score   support

               0       0.71      0.72      0.71      8034
               1       0.72      0.71      0.71      8136

        accuracy                           0.71     16170
       macro avg       0.71      0.71      0.71     16170
    weighted avg       0.71      0.71      0.71     16170
```

2.2.3 Finding Hyperparameters

Hyperparameters govern the machine learning models drom the point of training process. Hyperparameters control the execution of machine learning models with various machine learning model parameters. Hence, its important to know which hyperparameters are really boosting the accuracy of our training model. In order to accomplish this, we will run a model alogorithm by setting a range of values for each parameter. The best performing hyperparameters are choosen based on for which set of combination of input machine learning model parameters, wea re able to get higher accuracy and higher validation results.

```
X = df.drop(['Disease'], axis=1)
y = df["Disease"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
rfc=RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [10,20,30,40],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini',
max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=None,
                                              oob_score=False,
random_state=42,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [10, 20, 30, 40]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

2.2.4. The Hyperparameters are

```
CV_rfc.best_params_
```

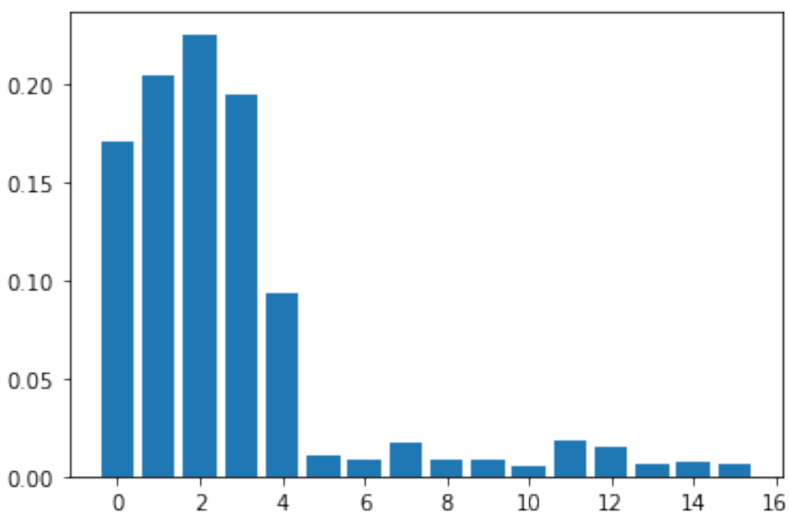
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 30}
```

2.2.5 Feature Importance from random Forest

Feature importance is the process of selecting the features which impacts the prediction results to a greater extent positively for improving accuracy. Keeping irrevalent features in the preduition model will provide lesser accuracy in results. So, now, lets see ehat are all the important features in our data set which can help to increase the accuracy.

```
importance = rf.feature_importances_

cols=X.columns
for i,v in enumerate(importance):
    print('Feature:%d %s, Score: %.5f' % (i,cols[i],v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Feature:0 Age, Score: 0.17092
Feature:1 Height, Score: 0.20436
Feature:2 Weight, Score: 0.22549
Feature:3 High Blood Pressure, Score: 0.19450
Feature:4 Low Blood Pressure, Score: 0.09334
Feature:5 Smoke, Score: 0.01024
Feature:6 Alcohol, Score: 0.00861
Feature:7 Exercise, Score: 0.01737
Feature:8 Gender_female, Score: 0.00861
Feature:9 Gender_male, Score: 0.00858
Feature:10 Cholesterol_high, Score: 0.00534
Feature:11 Cholesterol_normal, Score: 0.01838
Feature:12 Cholesterol_too high, Score: 0.01492
Feature:13 Glucose_high, Score: 0.00584
Feature:14 Glucose_normal, Score: 0.00762
Feature:15 Glucose_too high, Score: 0.00588
```



Lets select the first 5 features base don the feature importance to prepare the model again along with the hyperparameters.

The best performing 5 features are

Feature:2 Weight

Feature:1 Height

Feature:3 High Blood Pressure

Feature:0 Age

Feature:4 Low Blood Pressure

Feature:5 Smoke

Feature:13 Glucose_high

2.2.6 Lets apply the Hyperparameters now

```
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

rf_best=RandomForestClassifier(random_state=42, max_features='auto', n_estimators
rf_best.fit(X_train, y_train)
y_pred=rf_best.predict(X_test)
rf_hyper_acc = round(accuracy_score(y_test,y_pred) * 100,2)
print("Accuracy for Random Forest on CV data: ",rf_hyper_acc)
```

```
Accuracy for Random Forest on CV data:  73.54
```

2.2.6 Increase in accuracy after applying hyperparameters and feature importance

```
formatted_str = 'The increase in accuracy after applying hyper parameters  :  %1.
print(formatted_str)
```

```
The increase in accuracy after applying hyper parameters  :  2.12 %
```

2.3 Gradient Boosting Algorithm

Double-click (or enter) to edit

2.3.1 Algorithm implementation

Lets develop a model Gradient Boosting Alogorithm for the training data

```
# Gradient boosting

# Important features derived from heat map based on the correlation values
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]

#Apply scaling
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

gb = GradientBoostingClassifier(n_estimators=100, learning_rate=.25, max_features
gb.fit(X_train, y_train)
acc_train = gb.score(X_train, y_train)*100
acc_test = gb.score(X_test, y_test)*100
format_str = 'Accuracy-Training Data: %1.2f%%  Accuracy-Test Data :  %1.2f%%' % (
print(format_str)
```

```
Accuracy-Training Data: 76.03%  Accuracy-Test Data :  73.19%
```

2.3.2 Find Hyper Parameters fngor Gradient Boosting algorithm

The accuracy we have got above may be low. Lets review and tune the Hyperparameters using GridSearchCV

```
# Important features derived from heat map based on the correlation values
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
```

```
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]

#Apply scaling
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

gbc = GradientBoostingClassifier()
parameters = {
    "n_estimators":[100],
    "max_depth":[10,15],
    "learning_rate":[0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]
}
from sklearn.model_selection import GridSearchCV
cv = GridSearchCV(gbc,parameters,cv=5)
cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  criterion='friedman_mse',
                                                  init=None,
learning_rate=0.1,
                                                  loss='deviance',
max_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=None,
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,

min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=None,
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0,
warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.05, 0.075, 0.1, 0.25, 0.5, 0.75,
                                           1],
                         'max_depth': [10, 15], 'n_estimators': [100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

2.3.3. The Best Hyperparameters for Gradient Boosting Algorithm are:

```
print(f'Best parameters are: {cv.best_params_}')
print("\n")
mean_score = cv.cv_results_['mean_test_score']
std_score = cv.cv_results_['std_test_score']
params = cv.cv_results_['params']
for mean,std,params in zip(mean_score,std_score,params):
    print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
```

```
Best parameters are: {'learning_rate': 0.05, 'max_depth': 10, 'n_estimators':


0.725 + or -0.005 for the {'learning_rate': 0.05, 'max_depth': 10, 'n_estimat
0.699 + or -0.003 for the {'learning_rate': 0.05, 'max_depth': 15, 'n_estimat
0.722 + or -0.005 for the {'learning_rate': 0.075, 'max_depth': 10, 'n_estima
0.691 + or -0.007 for the {'learning_rate': 0.075, 'max_depth': 15, 'n_estima
0.718 + or -0.005 for the {'learning_rate': 0.1, 'max_depth': 10, 'n_estimato
0.69 + or -0.006 for the {'learning_rate': 0.1, 'max_depth': 15, 'n_estimators
0.699 + or -0.003 for the {'learning_rate': 0.25, 'max_depth': 10, 'n_estimat
0.683 + or -0.003 for the {'learning_rate': 0.25, 'max_depth': 15, 'n_estimat
0.678 + or -0.005 for the {'learning_rate': 0.5, 'max_depth': 10, 'n_estimato
0.676 + or -0.004 for the {'learning_rate': 0.5, 'max_depth': 15, 'n_estimato
0.672 + or -0.005 for the {'learning_rate': 0.75, 'max_depth': 10, 'n_estimat
0.674 + or -0.004 for the {'learning_rate': 0.75, 'max_depth': 15, 'n_estimat
0.662 + or -0.003 for the {'learning_rate': 1, 'max_depth': 10, 'n_estimators
0.67 + or -0.007 for the {'learning_rate': 1, 'max_depth': 15, 'n_estimators'
```

2.3.4 Apply Hyperparameters

```
# Gradient boosting

# Important features derived from heat map based on the correlation score
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]

lr = 0.05
feature_list = list(range(1,len(columns)))

#Apply scaling
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

gb_best = GradientBoostingClassifier(n_estimators=100, learning_rate=lr, max_feat
gb_best.fit(X_train, y_train)
y_pred = gb_best.predict(X_test)
gb_acc_hyper_train = gb_best.score(X_train, y_train)*100
gb_acc_hyper_test = gb_best.score(X_test, y_test)*100
format_str = 'Accuracy-Training Data: %1.2f%%  Accuracy-Test Data :  %1.2f%%' % (
print(format_str)
```

```
Accuracy-Training Data: 80.74%  Accuracy-Test Data :  72.87%
```

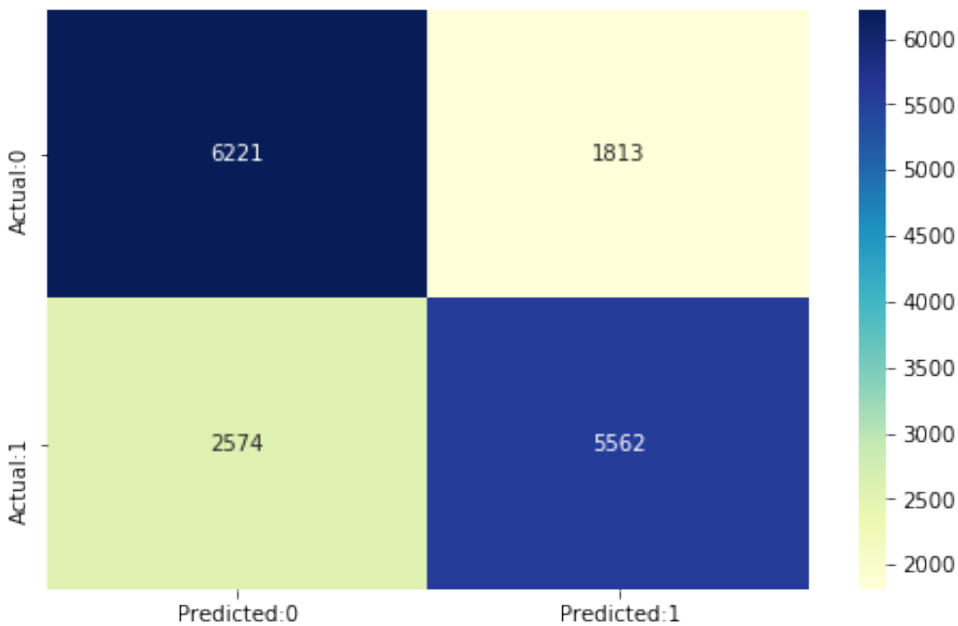
2.3.5 Increase in accuracy after applying hyperparameters and feature importance

```
formatted_str = 'The increase in accuracy after applying hyper parameters  :  %1.
print(formatted_str)
```

```
The increase in accuracy after applying hyper parameters  :  4.72 %
```

```
cm=confusion_matrix(y_test,y_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Ac
plt.figure(figsize = (8,5))
sns.heatmap(conf_matrix, annot=True,fmt='d',cmap="YlGnBu")

print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.71      0.77      0.74      8034
           1       0.75      0.68      0.72      8136

    accuracy                           0.73     16170
   macro avg       0.73      0.73      0.73     16170
weighted avg       0.73      0.73      0.73     16170
```



The classifcation report provides Recall, Precision, Accuracy and F-Score which are explained below.

1. Recall : Recall indicates that from the overall positive values, how much were predicted correctly. It should be higher as much as possible.
2. Precision : From the overall positive classes, how many were predicted correctly.
3. Accuracy : From the overall classes (including positive and negative) how much were predicted correctly
4. F1-measure : F-score helps to meaure precision and recall at the same time by using mean values.

   From the below report, we can notices that the accuracy and F1-Score lies at 73%.

2.4. K-Nearest Neighbours Algorithm

2.4.1 A model KNN Algorithm

```
# Features selected based on HeatMap from the correalation score.
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]

#Apply scaling
scaler = MinMaxScaler()
X = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

rmse_val = []
k_range =[]
scores=[]
 #Create KNN Classifier
for i in range(1, 35):

    knn = KNeighborsClassifier(n_neighbors=i)

    #Train the model using the training sets
```

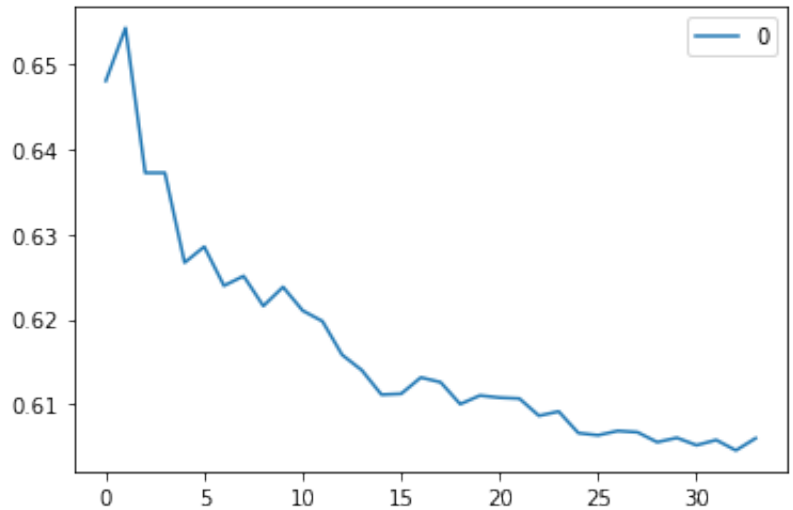
```
knn.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn.predict(X_test)

error = sqrt(mean_squared_error(y_test,y_pred)) #calculate rmse
acc_score=metrics.accuracy_score(y_test, y_pred)
k_range.append(i)
rmse_val.append(error)
scores.append(acc_score)

# Model Accuracy
print("Neighbours# " , i,"    Accuracy:",acc_score, "     RMSE value :  ",erro
```

```
Neighbours#  1     Accuracy: 0.5800247371675943      RMSE value :   0.64805498
Neighbours#  2     Accuracy: 0.5719233147804577      RMSE value :   0.65427569
Neighbours#  3     Accuracy: 0.593939393939394      RMSE value :   0.637228849
Neighbours#  4     Accuracy: 0.593939393939394      RMSE value :   0.637228849
Neighbours#  5     Accuracy: 0.6072974644403216      RMSE value :   0.62665982
Neighbours#  6     Accuracy: 0.604947433518862      RMSE value :   0.628532072
Neighbours#  7     Accuracy: 0.6106988249845393      RMSE value :   0.62394004
Neighbours#  8     Accuracy: 0.6092764378478664      RMSE value :   0.62507884
Neighbours#  9     Accuracy: 0.6136672850958566      RMSE value :   0.62155668
Neighbours#  10     Accuracy: 0.6108843537414966      RMSE value :   0.6237913
Neighbours#  11     Accuracy: 0.6143475572047      RMSE value :   0.6210092131
Neighbours#  12     Accuracy: 0.6158936301793445      RMSE value :   0.6197631
Neighbours#  13     Accuracy: 0.6207792207792208      RMSE value :   0.6158090
Neighbours#  14     Accuracy: 0.6230055658627087      RMSE value :   0.6139987
Neighbours#  15     Accuracy: 0.6265306122448979      RMSE value :   0.6111214
Neighbours#  16     Accuracy: 0.6264069264069264      RMSE value :   0.6112226
Neighbours#  17     Accuracy: 0.624056895485467      RMSE value :   0.61314199
Neighbours#  18     Accuracy: 0.6247371675943104      RMSE value :   0.6125869
Neighbours#  19     Accuracy: 0.6278911564625851      RMSE value :   0.6100072
Neighbours#  20     Accuracy: 0.6266542980828695      RMSE value :   0.6110202
Neighbours#  21     Accuracy: 0.6269635126777984      RMSE value :   0.6107671
Neighbours#  22     Accuracy: 0.62708719851577      RMSE value :   0.610665867
Neighbours#  23     Accuracy: 0.629560915275201      RMSE value :   0.60863707
Neighbours#  24     Accuracy: 0.6289424860853432      RMSE value :   0.6091449
Neighbours#  25     Accuracy: 0.6320346320346321      RMSE value :   0.6066014
Neighbours#  26     Accuracy: 0.6323438466295609      RMSE value :   0.6063465
Neighbours#  27     Accuracy: 0.6317254174397031      RMSE value :   0.6068563
Neighbours#  28     Accuracy: 0.6319109461966604      RMSE value :   0.6067034
Neighbours#  29     Accuracy: 0.6333333333333333      RMSE value :   0.6055300
Neighbours#  30     Accuracy: 0.6327149041434755      RMSE value :   0.6060405
Neighbours#  31     Accuracy: 0.6337662337662338      RMSE value :   0.6051725
Neighbours#  32     Accuracy: 0.6330241187384045      RMSE value :   0.6057853
Neighbours#  33     Accuracy: 0.634508348794063      RMSE value :   0.60455905
Neighbours#  34     Accuracy: 0.6327767470624613      RMSE value :   0.6059894
```

2.4.2 Display Elbow curve

```
#plotting the rmse values against k values
elbow = pd.DataFrame(rmse_val) #elbow curve
elbow.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7faf7349b0b8>
```



2.4.3 Find Hyper Params using GridSearch

```
# display the neighnor with highest accuracy

params = {}
srange = range(1,35)
params = {'n_neighbors':list(k_range)}
knn = neighbors.KNeighborsRegressor()
model = GridSearchCV(knn, params, cv=5)
model.fit(X_train,y_train)
model.best_params_
```

```
{'n_neighbors': 34}
```

2.4.4. Prepare KNN Model based on best fit

```
#Prepare model based on best fit

# Features selected based on HeatMap from the correalation score.
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df, columns=columns)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]

series_val=[]
rmse_val=[]

knn_best_fit=list(model.best_params_.values())[0]

knn_best = KNeighborsClassifier(n_neighbors=knn_best_fit)

X = df1.drop(['Disease'], axis=1)
y = df1["Disease"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, rando

#Train the model using the training sets
knn_best.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = knn_best.predict(X_test)

error = sqrt(mean_squared_error(y_test,y_pred)) #calculate rmse
knn_hyper_acc = metrics.accuracy_score(y_test, y_pred)*100
formatted_str = 'Neighbours#  %d     Accuracy: %1.4f     RMSE value :  %1.4f'  %
print(formatted_str)
```

```
    Neighbours#  34     Accuracy: 71.7069     RMSE value :  0.5319
```

3. Summary of Model Evaluation

```
models = pd.DataFrame({
    'Model': ['GaussianNB', 'RandomForestClassifier', 'GradientBoostingClassifier
    'Score': [acc_gaussian,  rf_hyper_acc, gb_acc_hyper_train, knn_hyper_acc]})
models.sort_values(by='Score', ascending=False)
```

	Model	Score
2	GradientBoostingClassifier	80.743223
1	andomForestClassifier	73.540000
3	KNeighborsClassifier	71.706865
0	GaussianNB	60.040000

4. Apply the highest accurate model to predict the probability of disease for the test data

4.1 Read Test data

```
# Read Test data

# data frame for processing
df_test = pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/Disease Predi

# dataframe for generating final result
gaussian_test_result = pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/
rf_test_result = pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/Diseas
gb_test_result = pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/Diseas
knn_test_result = pd.read_csv('/Users/vsubu/Documents/MS/Disease Prediction/Disea
```

4.2 Shape

```
df_test.shape
```

```
(21000, 12)
```

4.2 Check for null values

```
df_test.isnull().sum().sort_values(ascending=False)
```

```
Exercise               0
Alcohol                0
Smoke                  0
Glucose                0
Cholesterol            0
Low Blood Pressure     0
High Blood Pressure    0
Weight                 0
Height                 0
Gender                 0
Age                    0
ID                     0
dtype: int64
```

```
df_test.describe()
```

	ID	Age	Height	Weight	High Blood Pressure	Low Bloo Pressu
count	21000.000000	21000.000000	21000.000000	21000.000000	21000.000000	21000.0000
mean	10499.500000	52.811190	164.341381	74.241070	129.093429	95.9608
std	6062.322162	6.775489	8.195082	14.548468	167.975674	157.2574
min	0.000000	29.000000	64.000000	21.000000	10.000000	-70.0000
25%	5249.750000	48.000000	159.000000	65.000000	120.000000	80.0000
50%	10499.500000	53.000000	165.000000	72.000000	120.000000	80.0000
75%	15749.250000	58.000000	170.000000	82.000000	140.000000	90.0000

From the above list, we can notice that the Low Blood Pressure column has -ve values.
We will convert them to absolute values

4.3 Apply Data Cleaning

1. Create Categorical values for Gender, Cholesterol and Glucose
2. Convert the negative values of Low Blood Pressure Column to absolute value

```
# Apply the Data cleaning techniques.

# Convert negative values to absolute values for HBP
df_test['Low Blood Pressure'] = df_test['High Blood Pressure'].abs()

# Create categorical variables for Cholesterol and Glucose
df_test=pd.get_dummies(df_test, columns=["Gender","Cholesterol","Glucose"])

# Keep the columns based on feature importance.
columns = ['Age', 'Height','Weight','High Blood Pressure','Low Blood Pressure','C
df1 = pd.DataFrame(df_test, columns=columns)
```

4.4 Run the algorithms

```
# We removed Gaussian from the prediction requirement, since it was producing low

# Predict based on Random Forest
rf_test_result['Disease'] = rf_best.predict(df1)

# predict the target on Gradient Boosting algorithm
gb_test_result['Disease'] = gb_best.predict(df1)

# predict the target on KNN algorithm
knn_test_result['Disease'] = knn_best.predict(df1)
```

```
Target on test data [1 1 1 ... 1 1 1]
```

4.4 Display the top 10 rows of Random Forest prediction results

```
rf_test_result.head(10)
```

	ID	Age	Gender	Height	Weight	High Blood Pressure	Low Blood Pressure	Cholesterol	Glucose	Sm
0	0	44	female	160	59.0	100	80	high	normal	
1	1	41	female	169	74.0	120	70	normal	normal	
2	2	63	male	168	84.0	120	80	normal	high	
3	3	55	female	158	108.0	160	100	normal	normal	
4	4	55	female	167	67.0	120	80	normal	normal	
5	5	58	female	162	95.0	130	70	normal	normal	
6	6	45	female	161	68.0	120	70	normal	normal	
7	7	52	female	149	85.0	160	90	normal	normal	
8	8	58	male	168	64.0	140	90	normal	normal	

4.5 Display the top 10 rows of Gaussian Navie Bayes prediction results

```
gb_test_result.head(10)
```

	ID	Age	Gender	Height	Weight	High Blood Pressure	Low Blood Pressure	Cholesterol	Glucose	Sm
0	0	44	female	160	59.0	100	80	high	normal	
1	1	41	female	169	74.0	120	70	normal	normal	
2	2	63	male	168	84.0	120	80	normal	high	
3	3	55	female	158	108.0	160	100	normal	normal	
4	4	55	female	167	67.0	120	80	normal	normal	
5	5	58	female	162	95.0	130	70	normal	normal	
6	6	45	female	161	68.0	120	70	normal	normal	
7	7	52	female	149	85.0	160	90	normal	normal	
8	8	58	male	168	64.0	140	90	normal	normal	

4.6 Display the top 10 rows of KNN prediction results

```
knn_test_result.head(10)
```

	ID	Age	Gender	Height	Weight	High Blood Pressure	Low Blood Pressure	Cholesterol	Glucose	Sm
0	0	44	female	160	59.0	100	80	high	normal	
1	1	41	female	169	74.0	120	70	normal	normal	
2	2	63	male	168	84.0	120	80	normal	high	
3	3	55	female	158	108.0	160	100	normal	normal	
4	4	55	female	167	67.0	120	80	normal	normal	
5	5	58	female	162	95.0	130	70	normal	normal	
6	6	45	female	161	68.0	120	70	normal	normal	
7	7	52	female	149	85.0	160	90	normal	normal	
8	8	58	male	168	64.0	140	90	normal	normal	

4.7 Display the unique values of Disease from the prediction results data frame for all models.

```
# Print unique values for Disease column to ensure that it has only 1s and 0s
rf_test_result.groupby(['Disease'])['ID'].count()
```

```
Disease
0     8960
1    12040
Name: ID, dtype: int64
```

```
# Print unique values for Disease column to ensure that it has only 1s and 0s
knn_test_result.groupby(['Disease'])['ID'].count()
```

```
Disease
0     3578
1    17422
Name: ID, dtype: int64
```

```
gb_test_result.groupby(['Disease'])['ID'].count()
```

```
Disease
1    21000
Name: ID, dtype: int64
```

```
# Since Gradient boosting model has produced all 1s for the prediction results, its
rf_test_result.to_csv('rf_test_result.csv', index = True)
knn_test_result.to_csv('knn_test_result.csv', index = True)
```

4.7 References

Ye, Zhishan, et al. "Using Machine Learning Algorithms Based on GF-6 and Google E

1.9. Naive Bayes — scikit-learn 0.16.1 documentation. https://scikit-learn.org/0.

Lwanga, Victor Kwome. "Stock Market Price Prediction Using Sentiment Analysis: A