Cardiovascular Diseases Risk Prediction

I. Introduction

The dataset utilized in this project is a Cardiovascular Diseases Risk Prediction Dataset sourced from the 2021 Behavioral Risk Factor Surveillance System (BRFSS). BRFSS is recognized as the primary system in the United States for the conduct of health-related telephone surveys. It is committed to the collection of data from U.S. residents across various domains, encompassing behaviors with potential health implications, the presence of chronic health conditions, and the utilization of preventive healthcare services. This extensive program functions as an indispensable source of health-related information [1].

The dataset employed in this study has been pre-processed and cleaned to ensure its quality and reliability, as the original database contained 304 distinct variables. The author conducted a thorough selection process to carefully choose 19 lifestyle-related variables believed to have the potential to influence an individual's risk of developing cardiovascular diseases. [1].

The fields of the dataset are as follows:
General_Health, Checkup, Exercise, Heart_Disease, Skin_Cancer, Other_Cancer, Depression, Diabetes, Arthritis, Sex, Age_Category, Height_(cm), Weight_(kg), BMI, Smoking_History, Alcohol_Consumption, Fruit_Consumption, Green_Vegetables_Consumption, FriedPotato_Consumption [1].

II. Preprocessing

```
In [3]: data1 = pd.read_csv('C:/Users/pabas/Desktop/Machine Learning Project 01/Project5/CVD_cleaned_Over_Sampling.csv')

In [16]: data1.shape

Out[16]: (198290, 19)

In [4]: data1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 198290 entries, 0 to 198289
Data columns (total 19 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   General_Health               198290 non-null  object
 1   Checkup                      198290 non-null  object
 2   Exercise                     198290 non-null  object
 3   Skin_Cancer                  198290 non-null  object
 4   Other_Cancer                 198290 non-null  object
 5   Depression                   198290 non-null  object
 6   Diabetes                     198290 non-null  object
 7   Arthritis                    198290 non-null  object
 8   Sex                          198290 non-null  object
 9   Age_Category                 198290 non-null  object
 10  Height_(cm)                  198290 non-null  int64
 11  Weight_(kg)                  198290 non-null  float64
 12  BMI                          198290 non-null  float64
 13  Smoking_History              198290 non-null  object
 14  Alcohol_Consumption          198290 non-null  int64
 15  Fruit_Consumption            198290 non-null  int64
 16  Green_Vegetables_Consumption 198290 non-null  int64
 17  FriedPotato_Consumption      198290 non-null  int64
 18  Heart_Disease                198290 non-null  object
dtypes: float64(2), int64(5), object(12)
memory usage: 28.7+ MB
```

Fig. 1. python code to import the packages, to read the dataset and get information on dataset.

According to the above figure 1, after loading relevant packages, the data set was loaded as data1, the shape of the data set and information on the dataset are obtained.

```
In [5]: data1.describe()
```

Out[5]:

	Height_(cm)	Weight_(kg)	BMI	Alcohol_Consumption	Fruit_Consumption	Green_Vegetables_Consumption	FriedPotato_Consumption
count	198290.000000	198290.000000	198290.000000	198290.000000	198290.000000	198290.000000	198290.000000
mean	170.849644	84.922764	29.002042	4.682581	29.009320	14.589142	6.156085
std	10.664331	21.541352	6.572486	8.257350	24.639082	14.515923	8.527821
min	91.000000	25.400000	12.110000	0.000000	0.000000	0.000000	0.000000
25%	163.000000	69.850000	24.410000	0.000000	12.000000	4.000000	1.000000
50%	170.000000	81.650000	27.960000	0.000000	30.000000	12.000000	4.000000
75%	178.000000	96.620000	32.280000	5.000000	30.000000	20.000000	8.000000
max	234.000000	293.020000	98.440000	30.000000	120.000000	124.000000	128.000000

```
In [6]: data1.isnull().sum()
```

```
Out[6]: General_Health                0
        Checkup                       0
        Exercise                      0
        Skin_Cancer                   0
        Other_Cancer                  0
        Depression                    0
        Diabetes                      0
        Arthritis                     0
        Sex                           0
        Age_Category                  0
        Height_(cm)                   0
        Weight_(kg)                   0
        BMI                           0
        Smoking_History               0
        Alcohol_Consumption           0
        Fruit_Consumption             0
        Green_Vegetables_Consumption  0
        FriedPotato_Consumption       0
        Heart_Disease                 0
        dtype: int64
```

Fig. 2. Descriptive statistics and Null values count of the data set.

According to the above figure 2, the descriptive statistics of the dataset were obtained, and the dataset was checked for any null values. According to the above output it is confirmed that there are no null values on this dataset.

```
In [19]: data1.duplicated()
```

```
Out[19]: 0         False
         1         False
         2         False
         3         False
         4         False
                   ...
         198285    False
         198286    False
         198287    False
         198288    False
         198289    False
         Length: 198290, dtype: bool
```

Fig. 3. Python code to check for duplicates and the output of the data set.

The above output of duplicate values confirmed that there are no duplicate values on this dataset.

```
In [3]: columns_to_extract = ['Height_(cm)', 'Weight_(kg)', 'BMI', 'Alcohol_Consumption',
                              'Fruit_Consumption', 'Green_Vegetables_Consumption',
                              'FriedPotato_Consumption', 'Heart_Disease', 'Exercise','Diabetes','Arthritis','Other_Cancer',
                              'Depression','Age_Category','General_Health']
```

```
In [4]: data2 = data1[columns_to_extract].copy()

        data2['Heart_Disease']=data2['Heart_Disease'].map({"Yes":1,"No":0})
        data2['Exercise']=data2['Exercise'].map({"Yes":1,"No":0})
        data2['Diabetes']=data2['Diabetes'].map({"Yes":1,"No":0, "No, pre-diabetes or borderline diabetes":0,
                                                 "Yes, but female told only during pregnancy":1})
        data2['Arthritis']=data2['Arthritis'].map({"Yes":1,"No":0})
        data2['Other_Cancer']=data2['Other_Cancer'].map({"Yes":1,"No":0})
        data2['Depression']=data2['Depression'].map({"Yes":1,"No":0})
        data2['Age_Category']=data2['Age_Category'].map({"18-24":20,"25-29":25,"30-34":30,"35-39":35,"40-44":40
                                                        ,"45-49":45,"50-54":50,"55-59":55,"60-64":60,"65-69":65
                                                        ,"70-74":70,"75-79":75,"80+":80})
        data2['General_Health']=data2['General_Health'].map({"Poor":1,"Fair":2,"Good":3,"Very Good":4,"Excellent":5})

        data2.head()
```

Out[4]:

Green_Vegetables_Consumption	FriedPotato_Consumption	Heart_Disease	Exercise	Diabetes	Arthritis	Other_Cancer	Depression	Age_Category	General_Health
0	4	1	0	1	0	0	0	70	4
30	8	1	1	1	0	0	0	75	1
8	0	1	1	0	1	0	0	60	2
20	2	1	1	0	1	0	0	75	2
8	30	1	0	1	0	0	0	75	2

Fig. 4. Python code to extract columns and convert categorical data to numerical data.

According to the above figure 4, relevant columns of the data set were extracted and created a new dataset as data2. Then all the categorical columns in data2 were converted into numerical fields for further analysis.

III. DESCRIPTIVE DATA MINING

Descriptive data mining is conducted to visualize the nature of the data set. Below given are figures of plots which can be used to describe the features of the dataset in a more descriptive way.

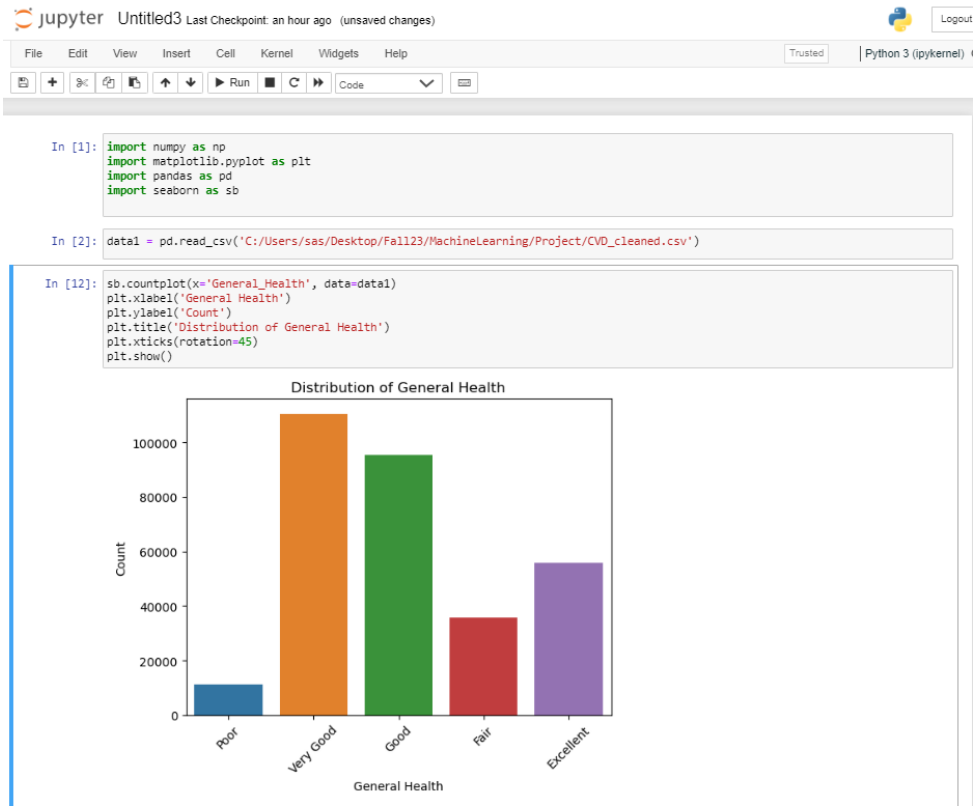

Fig. 5. python code to import the packages, to read the dataset and to show the boxplot for “Distribution of General Health.”

According to figure 5, it can be seen that a significant proportion of the sample under this study has good general health conditions.

```
In [4]: plt.scatter(data1['Weight_(kg)'], data1['BMI'], alpha=0.9, c='blue', edgecolors='k')
        plt.xlabel('Weight_(kg)')
        plt.ylabel('BMI')
        plt.title('Scatter Plot of Age vs. BMI')
        plt.xticks(rotation=45)
        plt.grid(True)
        plt.show()
```

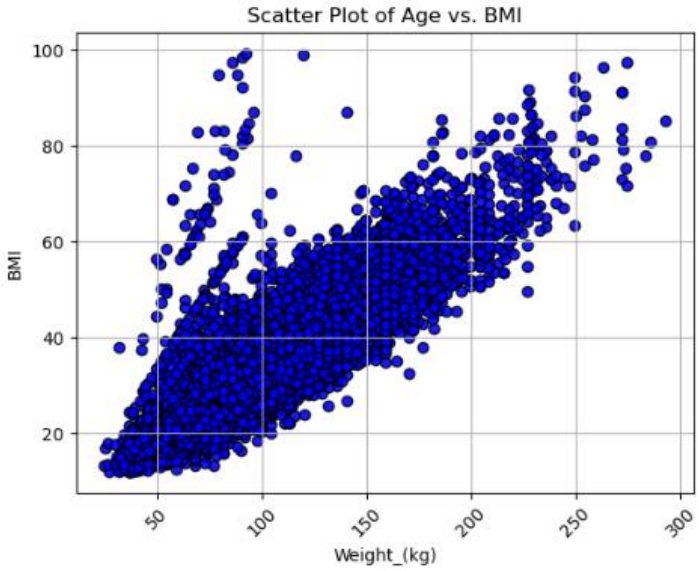

Fig. 6. Scatter plot of "Age vs. BMI" and the executed python code.

According to the above Figure 6, it can be seen that there is a significant positive correlation between the two variables, Weight and BMI.

```
In [5]: heart_disease_counts = data1['Heart_Disease'].value_counts()
        plt.pie(heart_disease_counts, labels=heart_disease_counts.index, autopct='%1.1f%%')
        plt.title('Distribution of Heart Disease')
        plt.show()
```

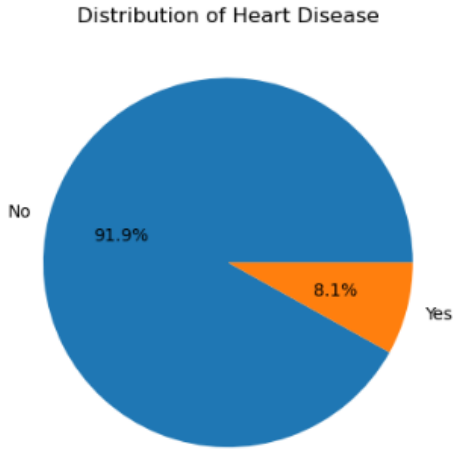

Fig. 7. Pie chart of "Distribution of Heart Disease" and the executed python code.

As of figure 7 it can be seen that a significant proportion of the sample has heart disease.

```
In [9]: bmi_by_age = data1.groupby('Age_Category')['BMI'].mean().reset_index()

plt.figure(figsize=(10, 6))
plt.plot(bmi_by_age['Age_Category'], bmi_by_age['BMI'], marker='o', linestyle='-')
plt.xlabel('Age Category')
plt.ylabel('Average BMI')
plt.title('Average BMI by Age Category')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

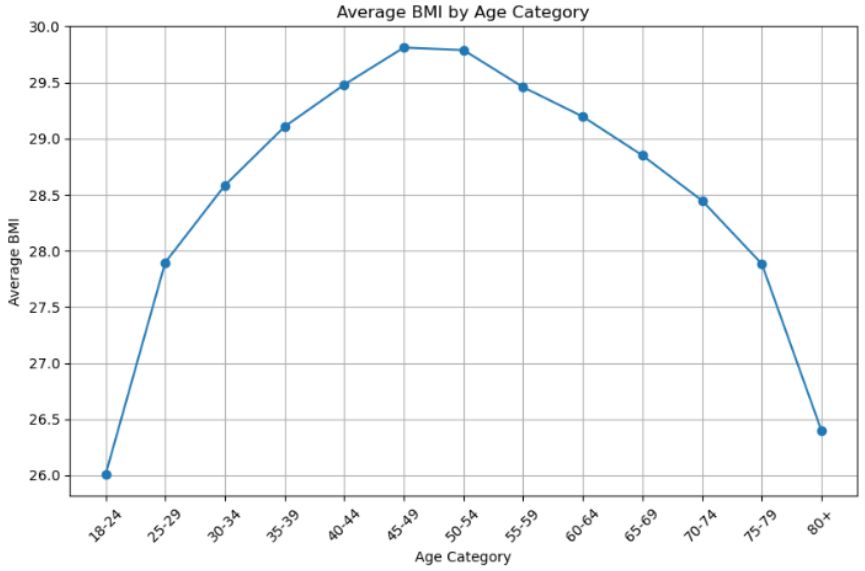

Fig. 8. Line chart of "Average BMI by Age Category" and the executed python code.

The above graph of "Average BMI by Age Category" shows that the highest BMI value is given by the two age categories, 45-49 and 50-54 years respectively. Furthermore, according to the graph in the very young ages like 18-24 and very old ages like 80+, the BMI values are significantly low.

```
In [14]: sb.jointplot(x='Weight_(kg)', y='BMI', data=data1, kind='scatter')
plt.xlabel('Weight (kg)')
plt.ylabel('BMI')
plt.title('Jointplot of BMI and Weight')
plt.show()
```

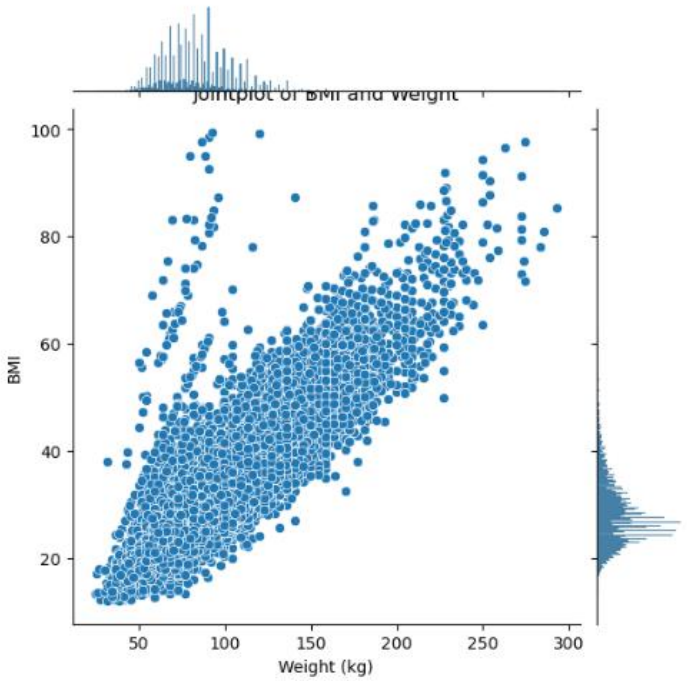

Fig. 9. Joint plot of "BMI and Weight" and the executed python code.

```
In [7]: sb.stripplot(x='AgeCategory', y='FriedPotatoConsumption', data=data1, size=10,jitter=True)
        plt.title('Strip Plot of Age Category vs Fried Potato Consumption')
        plt.show()
```

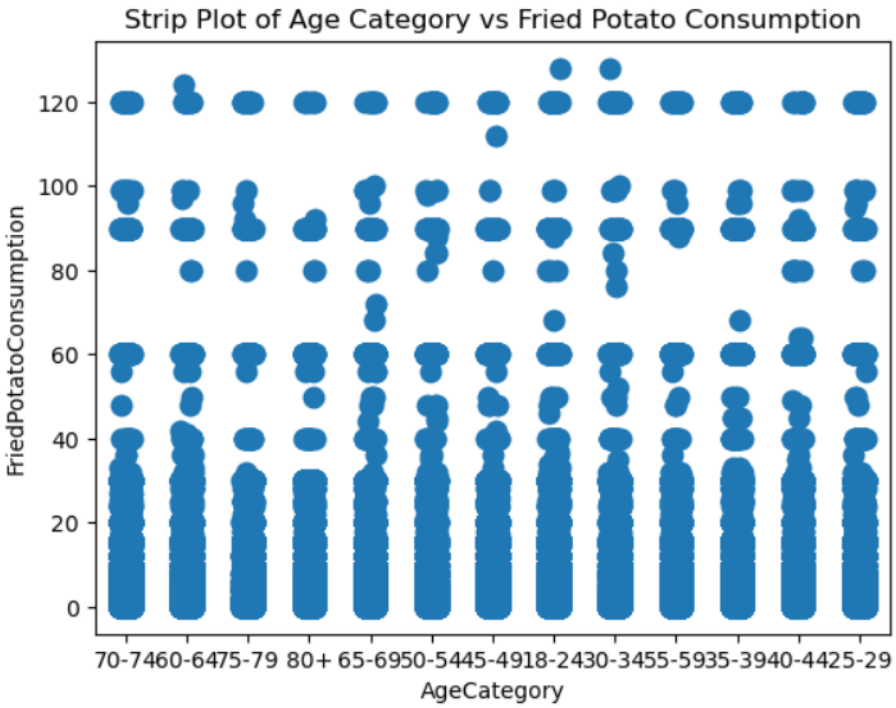

Fig. 10. Strip plot of "BMI and Weight" and the executed python code.

According to the above graph, it can be seen that the fried potato consumption in almost every age category under the sample is significantly low.

```
In [17]: plt.figure(figsize=(10, 6))
         sb.violinplot(x='General_Health', y='BMI', data=data1, palette='Set2')
         plt.xlabel('General Health')
         plt.ylabel('BMI')
         plt.title('Distribution of BMI by General Health')
         plt.xticks(rotation=45)
         plt.grid(True)
         plt.show()
```

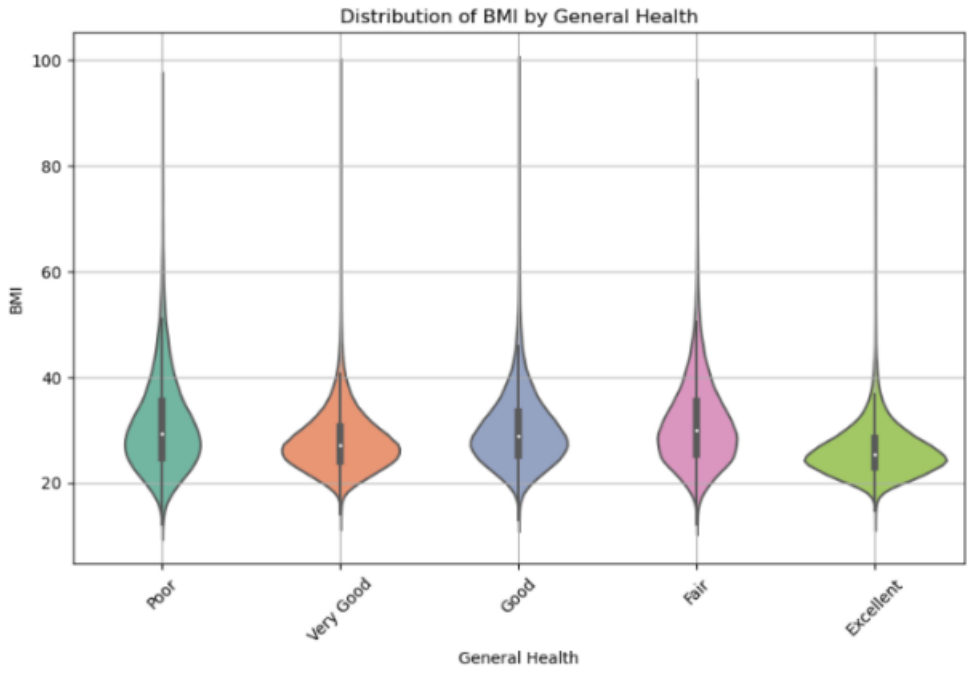

Fig. 11. Violin plot of the "Distribution of BMI and Weight" and the executed python code.

```
In [15]: sb.countplot(x='Age_Category', hue='Heart_Disease', data=data1, palette='Set1')
         plt.xlabel('Age Category')
         plt.ylabel('Count')
         plt.title('Distribution of Heart Disease by Age Category')
         plt.legend(title='Heart Disease', loc='upper right', labels=['No', 'Yes'])
         plt.xticks(rotation=45)
         plt.show()
```

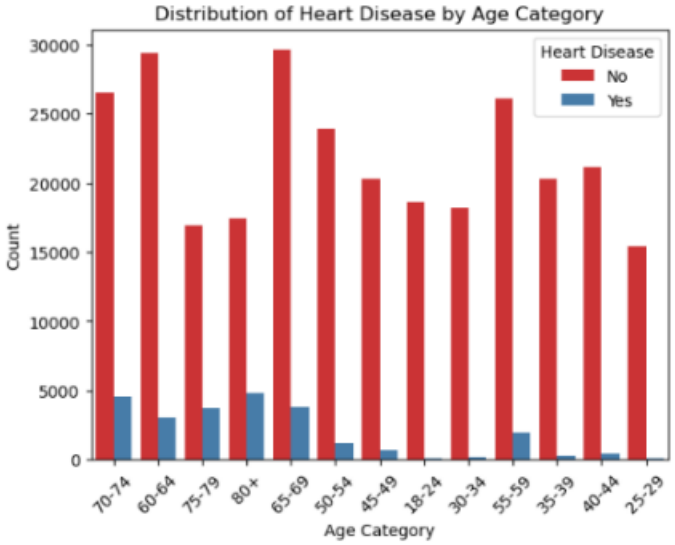

Fig. 12. Bar chart for “Distribution of heart disease by Age Category” and the executed python code.

IV. CLUSTERING

After preprocessing the dataset, it was checked for eligible clustering.

```
In [20]: from sklearn.cluster import KMeans

In [77]: clustering1 = KMeans(n_clusters=5)

In [78]: clustering1.fit(data2)

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warnin
g

Out[78]: KMeans
         KMeans(n_clusters=5)

In [79]: labels=clustering1.predict(data2)

In [80]: print(labels)

         [4 3 4 ... 2 3 2]

In [74]: centroid1=clustering1.cluster_centers_

In [81]: plt.scatter(data2.iloc[:, 0],data2.iloc[:, 4], c=labels)
         plt.scatter(centroid1[:, 0], centroid1[:, 4],marker='x', s=100, c='crimson')
         plt.show()
```

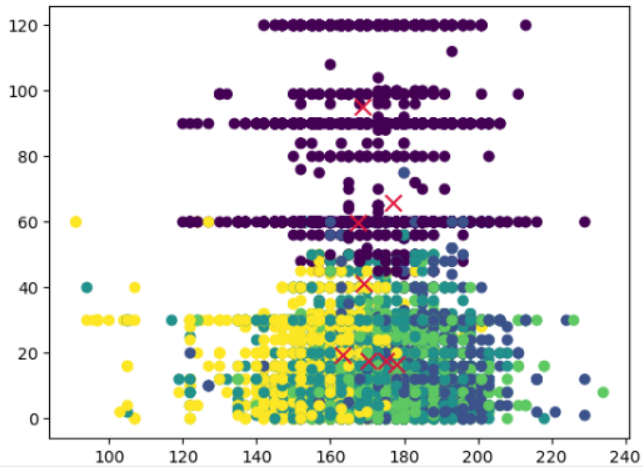

Fig. 13. Python code import libraries and obtain KMeans clustering.

Based on the insights derived from Figure 13, it is apparent that optimal K-means clustering was achieved by adjusting the number of clusters. However, it is noted that this might not be considered a favorable practice, suggesting potential limitations or challenges associated with the chosen clustering approach.

```
In [14]: data2.corr()
```

Out[14]:

	Height_(cm)	Weight_(kg)	BMI	Alcohol_Consumption	Fruit_Consumption	Green_Vegetables_Consumption	FriedPotato_Cor
Height_(cm)	1.000000	0.473043	-0.027278	0.129110	-0.044111	-0.016637	
Weight_(kg)	0.473043	1.000000	0.859273	-0.031895	-0.077652	-0.061649	
BMI	-0.027278	0.859273	1.000000	-0.107910	-0.062519	-0.061127	
Alcohol_Consumption	0.129110	-0.031895	-0.107910	1.000000	-0.010121	0.060897	
Fruit_Consumption	-0.044111	-0.077652	-0.062519	-0.010121	1.000000	0.267009	
Green_Vegetables_Consumption	-0.016637	-0.061649	-0.061127	0.060897	0.267009	1.000000	
FriedPotato_Consumption	0.106254	0.093303	0.045644	0.016056	-0.052608	0.006808	
Heart_Disease	0.031430	0.092007	0.086219	-0.072870	-0.035009	-0.047802	
Exercise	0.090383	-0.088351	-0.152272	0.104782	0.136680	0.133754	
Diabetes	-0.004791	0.191976	0.220463	-0.142825	-0.028329	-0.037559	
Arthritis	-0.092959	0.083875	0.145588	-0.054642	-0.011378	-0.038729	
Other_Cancer	-0.032196	-0.029337	-0.013747	-0.015697	-0.000668	-0.012243	
Depression	-0.094799	0.052252	0.115696	-0.039639	-0.042769	-0.049266	
Age_Category	-0.083115	-0.070389	-0.036660	-0.020911	0.024449	0.003203	
General_Health	0.061759	-0.160035	-0.215533	0.147125	0.098435	0.127182	

Fig. 14. Correlation Matrix of all variables.

```
In [9]: sb.pairplot(data1)
        plt.show()
```

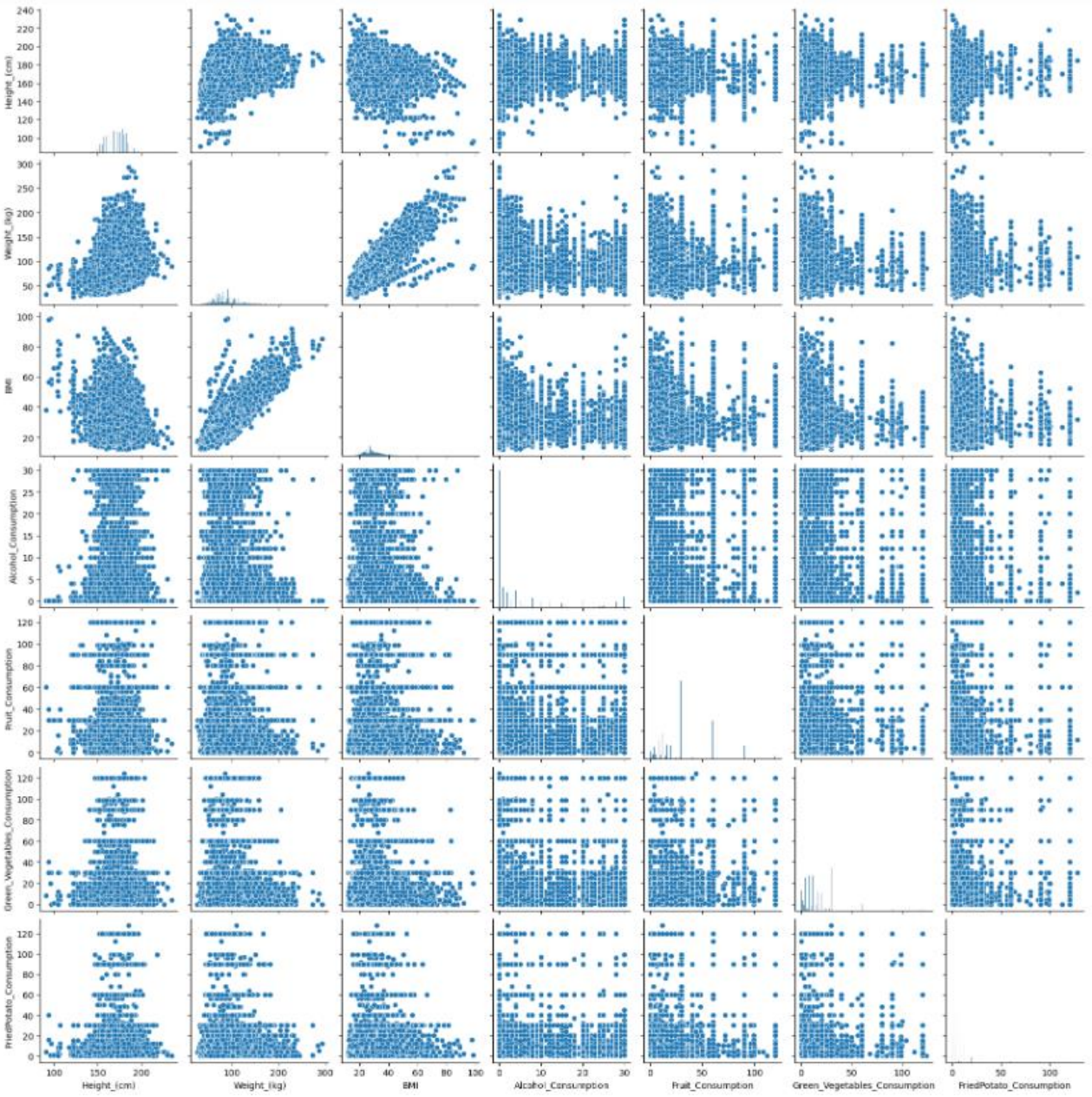

Fig. 15. Python code to obtain pair plot.

Based on the information presented in Figure 14, it is evident that no relationship, aside from height and BMI, exhibits a correlation exceeding 0.5. This observation signifies that there is no substantial correlation among the features. Furthermore, the findings in Figure 15 reinforce this conclusion by confirming the absence of multicollinearity within this dataset.

V. CLASSIFICATION

In this part of the study, the dataset was analyzed using three distinct classification algorithms: Decision Tree, K-Nearest Neighbors (KNN), and Naive Bayes. These algorithms were employed to predict the risk of cardiovascular diseases based on a range of clinical and demographic features. The investigation aimed to provide valuable insights into the predictive capabilities of each method and to identify the most influential risk factors contributing to cardiovascular disease outcomes. By leveraging these diverse classification approaches, the research contributes to a comprehensive understanding of the dataset and offers a foundation for informed decision-making in healthcare and risk assessment.

A. *Decision Tree*

```
In [5]: y = data2['Heart_Disease']
        x = data2.drop(columns=['Heart_Disease'])

In [6]: x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3, random_state=42,stratify=y)
```

Fig. 16. Python code to split the dataset.

Decision Tree

```
In [16]: tree1=DecisionTreeClassifier()

In [17]: tree1.fit(x_train, y_train)
Out[17]: DecisionTreeClassifier
         DecisionTreeClassifier()

In [18]: pre1=tree1.predict(x_test)

In [19]: from sklearn import metrics

In [20]: metrics.accuracy_score(y_test, pre1)
Out[20]: 0.8961789970918016
```

Fig. 16 Code and Output of Decision Tree Classifier

B. *Naïve Bayes*

Naive Bayes

```
In [11]: gnb1=GaussianNB()

In [12]: y_pred=gnb1.fit(x_train,y_train).predict(x_test)

In [13]: print(gnb1.score(x_test,y_test))
         0.7177198379477869

In [14]: print(confusion_matrix(y_test,y_pred))
         [[21360  8162]
          [ 8630 21335]]
```

Fig. 17 Code and Output of Naïve Bayes Classifier

C. KNN

KNN

```
In [7]: knn1=KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=2)

In [8]: knn1.fit(x_train,y_train)

Out[8]: KNeighborsClassifier
        KNeighborsClassifier(n_neighbors=3)

In [9]: y_predict=knn1.predict(x_test)

In [10]: print(knn1.score(x_test,y_test))
         0.7929799788189016
```

Fig. 18 Code and Output of KNN Classifier

Based on Figures 16, 17, and 18 above, it is evident that the model achieves accuracies of 0.79 and 0.71 when employing the decision tree and KNN, respectively. While Naïve Bayes demonstrates a relatively favorable accuracy of 0.89, there is room for improvement in the model's accuracy through the application of ensemble methods, as outlined below.

VI. Ensemble Methods

A. Random Forest

Random Forest

```
In [21]: RF1 =RandomForestRegressor(n_estimators=100,random_state=0)

In [22]: RF1.fit(x_train,y_train)

Out[22]: RandomForestRegressor
         RandomForestRegressor(random_state=0)

In [23]: pre1=RF1.predict(x_test)
         RF1.score(x_test,y_test)

Out[23]: 0.7342413628020581

In [24]: RF2 =RandomForestClassifier(max_depth=4, oob_score=True)
         RF2.fit(x_train,y_train)
         print(RF2.score(x_test,y_test))
         print(RF2.oob_score_)

         0.7426664649419201
         0.7394652853324496

In [25]: y_pred=RF2.predict(x_test)

In [26]: cm=confusion_matrix(y_test,y_pred)
         print(cm)

         [[20147  9375]
          [ 5933 24032]]
```

Fig. 19. python code to import the packages, to read the dataset and to show the accuracy of “Random Forest”.

B. Bagging

Bagging

```
In [27]: Bag1=BaggingClassifier(n_estimators = 100, random_state = 22)
         Bag1.fit(x_train,y_train)

Out[27]: BaggingClassifier
         BaggingClassifier(n_estimators=100, random_state=22)

In [28]: Bag1.predict(x_test)

Out[28]: array([1, 0, 0, ..., 1, 1, 1], dtype=int64)

In [29]: Bag1.score(x_test,y_test)

Out[29]: 0.9098626590683678

In [30]: Bag2=BaggingClassifier(n_estimators = 20, random_state = 22)
         Bag2.fit(x_train,y_train)

Out[30]: BaggingClassifier
         BaggingClassifier(n_estimators=20, random_state=22)

In [31]: pre3=Bag2.predict(x_test)

In [32]: Bag2.score(x_test,y_test)

Out[32]: 0.9119471481163952

In [33]: cm=confusion_matrix(y_test,pre3)
         print(cm)

         [[24758  4764]
          [  474 29491]]
```

Fig. 20. python code to import the packages, to read the dataset and to show the accuracy of “Bagging”.

As depicted in Figure 5, the model achieves a maximum accuracy of 0.91 when applying Bagging with 100 and 20 estimators, showcasing negligible differences in accuracy between the two scenarios.

C. Boosting

Ada Boost

```
In [34]: Ada1 = AdaBoostClassifier(n_estimators =100, learning_rate = 0.2).fit(x_train, y_train)
         score = Ada1.score(x_test, y_test)

In [35]: pre4=Ada1.predict(x_test)

In [36]: score
Out[36]: 0.7525845983155983

In [37]: cm=confusion_matrix(y_test,pre4)
         print(cm)

         [[21228  8294]
          [ 6424 23541]]
```

XGBoost

```
In [38]: from xgboost import XGBClassifier
         xg1 = XGBClassifier(n_estimators = 1000, learning_rate = 0.05)
         xg1.fit(x_train, y_train, early_stopping_rounds = 5, eval_set = [(x_test,y_test)], verbose=False)
         score_xgb = xg1.score(x_test,y_test)

         C:\ProgramData\anaconda3\lib\site-packages\xgboost\sklearn.py:835: UserWarning: `early_stopping_rounds` in `fit` method is depr
         ecated for better compatibility with scikit-learn, use `early_stopping_rounds` in constructor or`set_params` instead.
           warnings.warn(

In [39]: score_xgb
Out[39]: 0.7812295123304251

In [40]: pre5=xg1.predict(x_test)

In [41]: cm=confusion_matrix(y_test,pre5)
         print(cm)

         [[21349  8173]
          [ 4841 25124]]
```

Fig. 21. python code to import the packages, to read the dataset and to show the accuracy of “Boosting”.

As indicated by Figure 6 above, the model's accuracy is observed to be 0.75 when employing Ada Boosting and 0.78 when employing XGBoosting.

D. Logistic Regression

Logistic Regression

```
In [42]: reg1 = LogisticRegression()

In [43]: reg1.fit(x_train,y_train)

C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Out[43]: ▾ LogisticRegression
         LogisticRegression()

In [44]: pre2=reg1.predict(x_test)

In [45]: cm=confusion_matrix(y_test,pre2)

In [46]: print(cm)

         [[21061  8461]
          [ 6353 23612]]

In [47]: cm1=normalize(cm,norm='l1',axis=1)

In [48]: print(cm1)

         [[0.71340018 0.28659982]
          [0.21201402 0.78798598]]

In [49]: score = reg1.score(x_test, y_test)

In [50]: score

Out[50]: 0.7509708003429321
```

Fig. 22. python code to import the packages, to read the dataset and to show the accuracy of "Logistic Regression".

As indicated by Figure 7 above, the model's accuracy is also observed to be 0.75 when employing Logistic Regression.

When assessing the accuracy of decision trees, KNN, and Naive Bayes in comparison to Random Forest, Bagging, and Boosting, the latter trio consistently demonstrates superior performance. Decision trees, KNN, and Naive Bayes exhibit inherent limitations, which are effectively addressed by the ensemble methods. Random Forest, Bagging, and Boosting leverage the combined capabilities of multiple models, resulting in enhanced predictive accuracy. Empirical evidence suggests that, in diverse scenarios, the adoption of ensemble methods such as Random Forest, Bagging, and Boosting proves advantageous, positioning them as preferred choices for various machine learning applications.

According to the above study. It is confirmed that Bagging gives the highest accurate model with accurate confusion matrix compared to every other classification and ensemble methods.

VII. Feature Importance Methods

Feature importance in machine learning refers to the assessment of the significance of various input variables (features) in predicting the model's output or outcome. It enables the identification of features that contribute most substantially to the model's performance, offering valuable insights into underlying patterns. This understanding facilitates improved model interpretation, optimization, and decision-making by allowing practitioners to concentrate on the most influential factors for enhanced predictive accuracy.

The two types of Feature Importance used in this study are applying Random Forest and applying XGBoost. Then Neural Network with Keras was implemented and accuracy was calculated by changing the number of epos and batch size.

A. Feature Importance by applying Random Forest

Feature Importance by applying Random Forest

In [8]:
```
RF1 =RandomForestRegressor(n_estimators=100,random_state=0)
```

In [9]:
```
RF1.fit(x_train,y_train)
```

Out[9]:
```
RandomForestRegressor(random_state=0)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [10]:
```
RF1.feature_importances_
```

Out[10]:
```
array([0.06984612, 0.10162776, 0.13361714, 0.05954762, 0.08028843,
       0.08382707, 0.0767902 , 0.01659525, 0.01408395, 0.02070953,
       0.01452991, 0.01459244, 0.20896638, 0.1049782 ])
```

In [11]:
```
print(x.columns)
```

```
Index(['Height_(cm)', 'Weight_(kg)', 'BMI', 'Alcohol_Consumption',
       'Fruit_Consumption', 'Green_Vegetables_Consumption',
       'FriedPotato_Consumption', 'Exercise', 'Diabetes', 'Arthritis',
       'Other_Cancer', 'Depression', 'Age_Category', 'General_Health'],
      dtype='object')
```

In [12]:
```
list1=[]
for i in range(14):
    list1.append(x.columns[i])
list1
```

Out[12]:
```
['Height_(cm)',
 'Weight_(kg)',
 'BMI',
 'Alcohol_Consumption',
 'Fruit_Consumption',
 'Green_Vegetables_Consumption',
 'FriedPotato_Consumption',
 'Exercise',
 'Diabetes',
 'Arthritis',
 'Other_Cancer',
 'Depression',
 'Age_Category',
 'General_Health']
```

```
In [14]: plt.barh(x.columns, RF1.feature_importances_)
         plt.show()
```

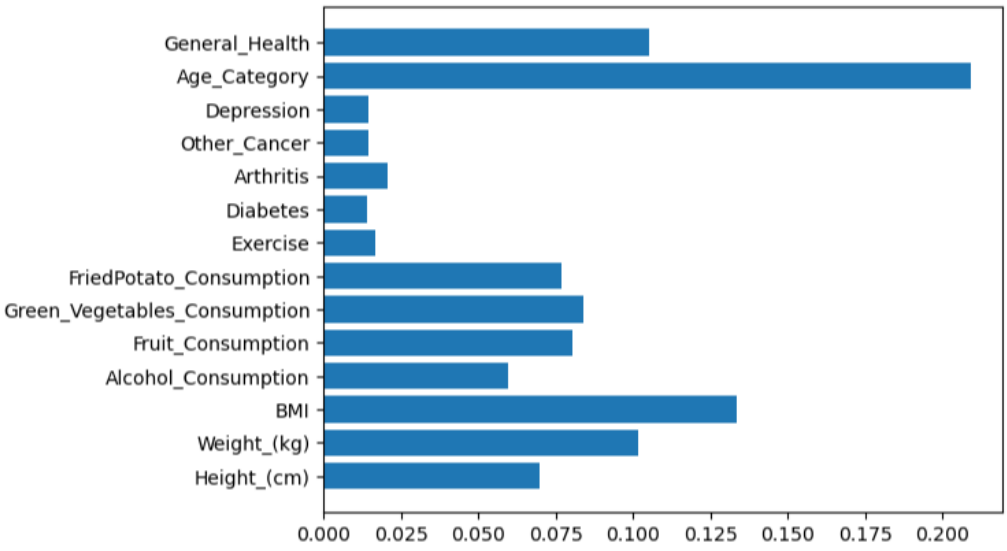

Fig. 23. python code to import the packages, to read the dataset and obtain feature Importance by applying Random Forest.

B. Feature Importance by applying XGBoost

Feature Importance by applying XGBoost

```
In [7]: xgb1=XGBClassifier(n_estimator=100,learning_rate=0.05)

In [8]: xgb1.fit(x_train,y_train)

[12:11:42] WARNING: C:\Users\dev-admin\croot2\xgboost-split_1675461376218\work\src\learner.cc:767:
Parameters: { "n_estimator" } are not used.

Out[8]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.05, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=None, max_leaves=None,
                      min_child_weight=None, missing=nan, monotone_constraints=None,
                      n_estimator=100, n_estimators=100, n_jobs=None,
                      num_parallel_tree=None, predictor=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [9]: xgb1.get_booster().get_score(importance_type='gain')

Out[9]: {'Height_(cm)': 27.797964096069336,
         'Weight_(kg)': 27.808765411376953,
         'BMI': 11.878717422485352,
         'Alcohol_Consumption': 13.385190963745117,
         'Fruit_Consumption': 11.240407943725586,
         'Green_Vegetables_Consumption': 11.078418731689453,
         'FriedPotato_Consumption': 12.036109924316406,
         'Exercise': 12.019018173217773,
         'Diabetes': 55.66769790649414,
         'Arthritis': 73.12091064453125,
         'Other_Cancer': 17.691173553466797,
         'Depression': 20.627023696899414,
         'Age_Category': 381.68853759765625,
         'General_Health': 289.53582763671875}
```

Fig. 24. python code to import the packages, to read the dataset and obtain feature Importance by applying Random Forest".

C. Neural Network with Keras

```
In [7]: model3=Sequential()
        model3.add(Dense(12, input_shape=(14,), activation='relu'))
        model3.add(Dense(8, activation='relu'))
        model3.add(Dense(1,activation='sigmoid'))
        model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
        model3.fit(x,y, epochs=50, batch_size=40)

Epoch 1/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5900 - accuracy: 0.7270
Epoch 2/50
4958/4958 [==============================] - 10s 2ms/step - loss: 0.5240 - accuracy: 0.7440
Epoch 3/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5208 - accuracy: 0.7449
Epoch 4/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5177 - accuracy: 0.7463
Epoch 5/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5165 - accuracy: 0.7483
Epoch 6/50
4958/4958 [==============================] - 13s 3ms/step - loss: 0.5157 - accuracy: 0.7491
Epoch 7/50
4958/4958 [==============================] - 12s 2ms/step - loss: 0.5149 - accuracy: 0.7495
Epoch 8/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5144 - accuracy: 0.7489
Epoch 9/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5139 - accuracy: 0.7498
Epoch 10/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5137 - accuracy: 0.7501
Epoch 11/50
4958/4958 [==============================] - 10s 2ms/step - loss: 0.5130 - accuracy: 0.7507
Epoch 12/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5130 - accuracy: 0.7503
Epoch 13/50
4958/4958 [==============================] - 12s 2ms/step - loss: 0.5124 - accuracy: 0.7508
Epoch 14/50
4958/4958 [==============================] - 12s 2ms/step - loss: 0.5126 - accuracy: 0.7501
Epoch 15/50
4958/4958 [==============================] - 13s 3ms/step - loss: 0.5121 - accuracy: 0.7510
Epoch 16/50
4958/4958 [==============================] - 12s 2ms/step - loss: 0.5121 - accuracy: 0.7509
Epoch 17/50
4958/4958 [==============================] - 12s 2ms/step - loss: 0.5118 - accuracy: 0.7508
Epoch 18/50
4958/4958 [==============================] - 11s 2ms/step - loss: 0.5116 - accuracy: 0.7507
```

Fig. 25. Code and output to create Nural Network using Keras.

```
In [8]: _, accuracy = model3.evaluate(x,y)
        print('Accuracy: %.2f' % (accuracy*100))

6197/6197 [==============================] - 12s 2ms/step - loss: 0.5090 - accuracy: 0.7524
Accuracy: 75.24
```

Fig. 26. Output and accuracy of Neuro Network.

The output values from Random Forest and XGBoost are identical; the only distinction lies in their presentation format. Random Forest is visualized through a vertical bar chart, while XGBoost values are exhibited in textual representation. In the case of the Neural Network implemented with Keras, varying combinations of epochs and batch size were explored to identify the optimal accuracy. The ultimate accuracy achieved with the Neural Network stands at 75.24%.

Conclusion

Following the thorough analysis conducted above, it becomes evident that the models' highest accuracy reached 91%, attributed to the Bagging ensemble method. All other obtained accuracies fell below this benchmark. Therefore, the conclusion can be drawn that, for this dataset, the Bagging ensemble method excels in creating the most effective model.

References

[1] Alphiree, “Cardiovascular diseases risk prediction dataset,” Kaggle, https://www.kaggle.com/datasets/alphiree/cardiovascular-diseases-risk-prediction-dataset/data, July 2023.