INFOSYS 722

Iteration 4 BDAS

Analysis of Cardiovascular Disease

Submitted By :
Di Zhao
Student ID:
144356542

Contents

1 Business Understanding

Github Link:

https://github.com/VirtueZhao/INFOSYS_722_Iteration4_BDAS

Definition and Facts about CVDs:

Before I analyze the statistics of cardiovascular disease, I should know what the cardiovascular disease actually means. This is the definition of cardiovascular disease from **World Health Organization**: **Cardiovascular diseases (CVDs)** are disorders of the heart and blood vessels and include coronary heart disease, cerebrovascular disease, rheumatic heart disease and other conditions. Four out of five CVD deaths are due to heart attacks and strokes. Individuals at risk of CVD may demonstrate raised blood presure, glucose, and lipids as well as overweight and obesity [18].

These are the facts about cardiovascular diseases:

- CVDs are the number 1 cause of death globally: more people die annually from CVDs than from any other cause [18].
- An estimated 17.9 million people died from CVDs om 2016, representing 31% of all global deaths. Of these deaths, 85% are due to heart attack and stroke [18].
- Over three quaters of CVD deaths take place in low- and middle-income countries.
- Out of the 17 million premature deaths (under the age of 70) due to noncommunicable diseases in 2015, 82% are in low- and middle-income countries, and 37% are caused by CVDs [18].

These facts show that there are lots of people dead per year caused by cardiovascular diseases. Therefore, reduce the risk of cardiovascular diseases is urgent. If I can find any useful information such as the relations between different features (**age**, **gender**, et al.) and the probability of getting cardiovascular diseases, I can save lots of people's life.

1.1 Identify The Objectives Of The Business/Situation

1.1.1 - UN 17 Sustainable Goals - Good health and Well-being

From the World Health Organization's statistics, I can see that the CVDs problem should be treated seriously. The United Nation also make it as one of it's 17 sustainable goals (**Good health and Well-being**).

The targets related to cardiovascular diseases are:

- **3.8** Achieve universal health coverage, including financial risk protection, access to quality essential health-care services and access to safe, effective, quality and affordable essential medicines and vaccines for all [17].
- **3.B** Support the research and development of vaccines and medicines for the communicable and noncommunicable diseases that primarily affect developing countries, provide access to affordable essential medicines and vaccines [17].
- **3.C** Substantially increase health financing and the recruitment, development, training and retention of the health workforce in developing countries, especially in least developed countries and small island developing states [17].
- **3.D** Strengthen the capacity of all countries, in particular developing countries, for early warning, risk reduction and management of national and global health risks [17].

1.1.2 - The Necessity and Benefits of CVDs Analysis

Despite numerous congresses and publications, the CVD is still under-recognized and underfunded. No pertinent representation of the CVD is provided. CVD is being ignored by policy makers and development aid agencies, and limited funds mean limited action directed at prevention and control [2] [3].

There are lots of benefits for CVDs analysis. First, analysis of the most recent global burden of disease data clearly reveals the predominance of CVD. Second, CVD affects many people working age at high rates in countries with low and middle incomes, which clearly impacts economic growth. Third, health systems cannot be built vertically, disease by disease; by integrating CVD diseases, weak health systems can be strengthened. Finally cost-effective policy, program and treatment initiatives already exist for CVD that could notably improve poverty and general health in many countries [2].

1.1.3 - Research Objectives

Thus this research is commissioned with the following objectives:

1. **Explanation:** Finding out important features relevant to whether a person get cardiovascular diseases. Explain the relation between features and response variable.
2. **Explanation:** Describing how the features will affect the probability of a person get cardiovascular diseases. Explain the latitude of the features which affect the response variable.

3. **Prediction:** Given a person's details, predict whether this person has cardiovascular diseases. Predict the response variable by provided features.

1.1.4 - Research Judgment

Tentatively, this research will be judged a success if:

1. Finding out which features are important and how they affect the predicted result (Objective 1 and Objective 2).
2. The high accuracy for prediction (Objective 3).
3. The research finishes on time (Since this is a course assignment, it should be finished on time).

1.2 Assess The Situation

1.2.1 - Personnel [1]

Doctors are professional in the medical areas. They can record patients details and provide a good treatment plan to patients by the patients' personal detail and patients' medical records. However, their knowledge of data collection, data management and data analysis are still lacking. Therefore, a data scientist is needed. Because

- For the patients, a specific data analysis will give them an early warning, which can reduce the chance to get cardiovascular disease (3.D).
- For the doctors and hospitals, a specific data analysis will give them better understanding for patients' situation and provide better treatment (3.8, 3.B).
- For the medical company, they can develop new medicines with better efficiency which can help more people (3.B).
- For the government (in particular developing countries' government), a specific data analysis allow them warn people earlier, reduce people's health risk(3.C, 3.D).

This research is long-term effective and will benefit more and more people over time.

1.2.2 - Data Sources

The dataset should be collected from the reliable, credible and authoritative dataset websites. For example:

- **Kaggle**: https://www.kaggle.com/datasets

- **UCI Machine Learning**: https://archive.ics.uci.edu/ml/datasets.php
- **NZ Government Data Portal**: https://catalogue.data.govt.nz/dataset
- **KD Nuggets**: https://www.kdnuggets.com/datasets/index.html

1.2.3 - Requirements

I don't have to worry about the security and legal restrictions such as privacy policy problems on the project results because the instances in the data set are anonymous.

For now, there are not deployment requirements. But if the result will be deployed, the requirements will be discussed further.

1.2.4 - Assumptions

Even the data set may not contain all features which relevant to the response variable, we assume my data set is representative and qualified.

Except the data set quality, there are no other factors can affect the research.

1.2.5 - Constraints

There is no limitation for data access and use since the data set is collected from Kaggle which is a open source platform for machine learning and data mining. All data sets provided by Kaggle is free to access and use. There are no financial constraints covered in the project budget.

1.2.6 - Risks

1. Due to the **Privacy Policy**, the records in the datasets may not representative enough. Since the datasets only contains the records which is allowed to be displayed.
2. The features contained in dataset may not representative enough either. I should not ignore the features which are not included in the features.
3. The information contained in the dataset is enormous and complex. The number of features recorded for each instance is also large. All these factors may cause the dataset difficult to analyse.

1.2.7 - Contingency Plan

1. For **Risk 1**, we should try to collect as more qualified data as I can which will make the data more representative.

2. For **Risk 2**, my explanation, inference and prediction should only based on the truth I have. I should clarify my research result for the person or institutions who use it.
3. For **Risk 3**, I should do more work on data preprocessing which will make the data easier to analyse and gives me a more clear result.

1.3 Determine Data Mining Objectives

1.3.1 - Data Mining Goals

The goals for this research to be completed are:

- **Prediction**: This research uses the datasets to build a efficient model to predict whether a patient has cardiovascular disease or not.
- **Explanation**: This research uses the datasets to find the relationship between provided features and cardiovascular disease.
- **Desired Outcome**: I desire the prediction accuracy as higher as possible and a rank for important features.

1.3.2 - Success Criteria

- **Prediction**: The prediction will be judged as success if the prediction accuracy is higher than 70%.
- **Explanation**: The explanation will be judged as success if I get a rank for the important features.

1.4 Project Plan

Phase	**Time**	**Resources**	**Risks**
Business Understanding	0.5 week	All analysts	Economic change
Data Understanding	0.5 week	All analysts	Data problems, technology problems
Data Preparation	0.5 week	Python, Data mining consultant, some database analyst time	Data problems, technology problems
Modeling	0.5 week	Python, Data mining consultant, some database analyst time	Technology problems, inability to find adequate model
Evaluation	0.5 week	All analysts	Economic change, inability to implement results
Deployment	0.5 week	Data mining consultant, some database analyst time	Economic change, inability to implement results

Table 1: Project Plan Overview

Since the time limit, I only have 3 weeks for each iteration. Therefore, I spend the first week to understand the business, collect data and understand the data. Then I use the second week to process the data and build models for my mining goal. Then I spend the last week to evaluate my models and deploy the best model.

2 Data Understanding

2.1 Collect Initial Data

The dataset is downloaded from **Kaggle Dataset**:
https://www.kaggle.com/sulianova/cardiovascular-disease-dataset

The data is uploaded by **Svetlana Ulianova**.
Her personal kaggle homepage is: https://www.kaggle.com/sulianova

Kaggle is an authoritative platform for data mining learners to collect and download suitable, verified datasets. Therefore, I can say the dataset is reliable and credible.

2.2 Describe The Data

2.2.1 - Data Amount

The Cardiovascular Disease Dataset contains 70.0k rows(instances) and 13 columns(features).

2.2.2 - Value Types and Coding Schemes

Feature Name	Feature Type	Unit	Coding Schemes
id	Continuous		
age	Continuous	days	
gender	Categorical		1: women, 2: men
height	Continuous	cm	
weight	Continuous	kg	
ap_hi(Systolic blood pressure)	Continuous		
ap_lo(Diastolic blood pressure)	Continuous		
cholesterol	Categorical		1: normal, 2: above normal, 3: well above normal
gluc(Glucose)	Categorical		1: normal, 2: above normal, 3: well above normal
smoke	Flag		1: Yes, 2: Not
alco(Alcohol intake)	Flag		1: Yes, 2: Not
active(Physical activity)	Flag		1: Yes, 2: Not
cardio (Presence or absence)	Flag (Response)		1: Yes, 2: Not

Table 2: Features Value Type and Coding Schemes

2.3 Explore The Data

Important: Since the pyspark dataframe is hard to manipulate, we choose pandas to describe and visualize our data, then convert it back to the pyspark dataframe and build the pyspark model. The pyspark data frame and pandas data frame can easily convert to each other by the following code:

```
[1]: import findspark
     import pandas as pd
     findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
     import pyspark
     from pyspark.sql import *
     spark = SparkSession.builder.appName('dataframe_transformation').getOrCreate()

[4]: # Read Data into PySprak Data Frame
     original_df_in_pyspark = spark.read.csv('cardio_train_new.csv')

[6]: # Convert PySpark Dataframe to Pandas Dataframe
     df_in_pandas = original_df_in_pyspark.toPandas()

[7]: # Convert Pandas Dataframe to PySpark Dataframe
     df_in_pyspark = spark.createDataFrame(df_in_pandas)
```

Figure 1: DataFrame Transformation

2.3.1 - Code for Exploring The Data

```
In [5]: cardio_data = pd.read_csv('cardio_train_new.csv')
        cardio_data
```

Out[5]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	NaN	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228.0	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857.0	1	165	64.0	130	70	3	1	0	0	0	1
3	3	NaN	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474.0	1	156	56.0	100	60	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	99993	19240.0	2	168	76.0	120	80	1	1	1	0	1	0
69996	99995	22601.0	1	158	126.0	140	90	2	2	0	0	1	1
69997	99996	19066.0	2	183	105.0	180	90	3	1	0	1	0	1
69998	99998	22431.0	1	163	72.0	135	80	1	2	0	0	0	1
69999	99999	20540.0	1	170	72.0	120	80	2	1	0	0	1	0

70000 rows × 13 columns

Figure 2: Code for Exploring the Data and Snap shot of the Dataset

```
In [6]: cardio_data.dtypes
Out[6]: id              int64
        age           float64
        gender          int64
        height          int64
        weight        float64
        ap_hi           int64
        ap_lo           int64
        cholesterol     int64
        gluc            int64
        smoke           int64
        alco            int64
        active          int64
        cardio          int64
        dtype: object
```

Figure 3: Code for Exploring the Data and Snap shot of the Dataset

From the output, I can see that there are 70,000 rows of the data and 13 columns. For now, all the data type are set to numeric (float64 and int64) but they are actually not. I should modify them later.

2.3.2 - Exploration

First, I will check whether the distribution of response variable is balanced. In Figure 4, I can see that the distribution for both class is balanced. Therefore, I can say that this data set is representative.

```
In [7]: plt.hist(cardio_data.cardio)
Out[7]: (array([35021.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                    0., 34979.]),
         array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
         <a list of 10 Patch objects>)
```

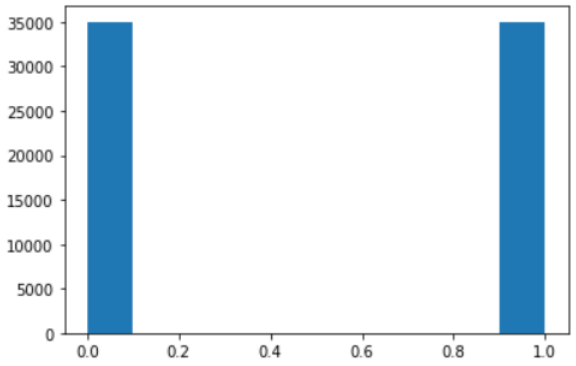


Figure 4: Distribution for response variable (cardio)

Now, let's check the statistics of numeric features and distribution of categorical features.

```
In [9]: print("age")
        print("Count: " + str(len(cardio_data.age)))
        print("Min: " + str(np.nanmin(cardio_data.age)))
        print("Max: " + str(np.nanmax(cardio_data.age)))
        print("Range: " + str(np.nanmax(cardio_data.age) - np.nanmin(cardio_data.age)))
        print("Variance: " + str(np.nanvar(cardio_data.age)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.age)))

age
Count: 70000
Min: 10798.0
Max: 23713.0
Range: 12915.0
Variance: 6087339.625698899
Standard Deviation: 2467.2534579363546
```

Figure 5: Statistics of age

```
In [10]: print("height")
         print("Count: " + str(len(cardio_data.height)))
         print("Min: " + str(np.nanmin(cardio_data.height)))
         print("Max: " + str(np.nanmax(cardio_data.height)))
         print("Range: " + str(np.nanmax(cardio_data.height) - np.nanmin(cardio_data.height)))
         print("Variance: " + str(np.nanvar(cardio_data.height)))
         print("Standard Deviation: " + str(np.nanstd(cardio_data.height)))

height
Count: 70000
Min: 55
Max: 250
Range: 195
Variance: 67.40521197632654
Standard Deviation: 8.210067720568848
```

Figure 6: Statistics of height

```
In [11]: print("weight")
         print("Count: " + str(len(cardio_data.weight)))
         print("Min: " + str(np.nanmin(cardio_data.weight)))
         print("Max: " + str(np.nanmax(cardio_data.weight)))
         print("Range: " + str(np.nanmax(cardio_data.weight) - np.nanmin(cardio_data.weight)))
         print("Variance: " + str(np.nanvar(cardio_data.weight)))
         print("Standard Deviation: " + str(np.nanstd(cardio_data.weight)))

weight
Count: 70000
Min: 10.0
Max: 200.0
Range: 190.0
Variance: 207.23484980675718
Standard Deviation: 14.39565385131072
```

Figure 7: Statistics of weight

```
In [12]: print("ap_hi")
         print("Count: " + str(len(cardio_data.ap_hi)))
         print("Min: " + str(np.nanmin(cardio_data.ap_hi)))
         print("Max: " + str(np.nanmax(cardio_data.ap_hi)))
         print("Range: " + str(np.nanmax(cardio_data.ap_hi) - np.nanmin(cardio_data.ap_hi)))
         print("Variance: " + str(np.nanvar(cardio_data.ap_hi)))
         print("Standard Deviation: " + str(np.nanstd(cardio_data.ap_hi)))

ap_hi
Count: 70000
Min: -150
Max: 16020
Range: 16170
Variance: 23719.17847263265
Standard Deviation: 154.01031937059494
```

Figure 8: Statistics of ap_hi

```
In [13]: print("ap_lo")
         print("Count: " + str(len(cardio_data.ap_lo)))
         print("Min: " + str(np.nanmin(cardio_data.ap_lo)))
         print("Max: " + str(np.nanmax(cardio_data.ap_lo)))
         print("Range: " + str(np.nanmax(cardio_data.ap_lo) - np.nanmin(cardio_data.ap_lo)))
         print("Variance: " + str(np.nanvar(cardio_data.ap_lo)))
         print("Standard Deviation: " + str(np.nanstd(cardio_data.ap_lo)))

ap_lo
Count: 70000
Min: -70
Max: 11000
Range: 11070
Variance: 35521.38722068551
Standard Deviation: 188.471184059223
```

Figure 9: Statistics of ap_lo

I can see that the statistic for numeric features does not all in the reasonable range. For example, a human cannot have a negative blood pressure or the blood pressure more than 10000. I can also see that the age has much larger statistics than other features. I may need to normalize it later.

```
In [17]: plt.hist(cardio_data.smoke)
Out[17]: (array([63831.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                    0.,  6169.]),
          array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
          <a list of 10 Patch objects>)
```

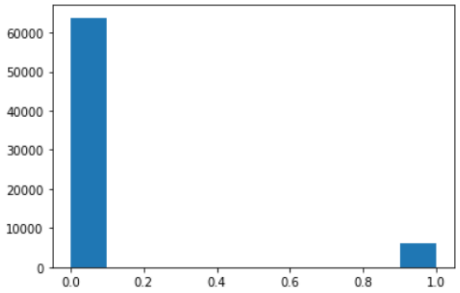


Figure 10: Distribution of smoke

```
In [19]: plt.hist(cardio_data.alco)
Out[19]: (array([66236.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                     0.,  3764.]),
          array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
          <a list of 10 Patch objects>)
```

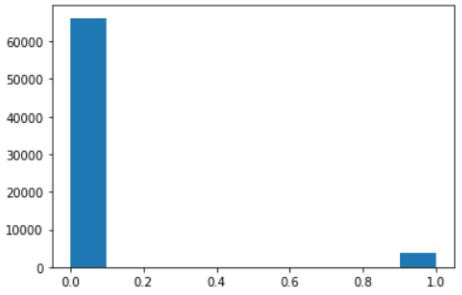


Figure 11: Distribution of alco

```
In [20]: plt.hist(cardio_data.cholesterol)
Out[20]: (array([52385.,     0.,     0.,     0.,     0.,  9549.,     0.,     0.,
                     0.,  8066.]),
          array([1. , 1.2, 1.4, 1.6, 1.8, 2. , 2.2, 2.4, 2.6, 2.8, 3. ]),
          <a list of 10 Patch objects>)
```

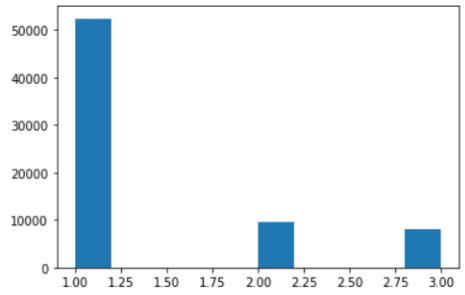


Figure 12: Distribution of cholesterol

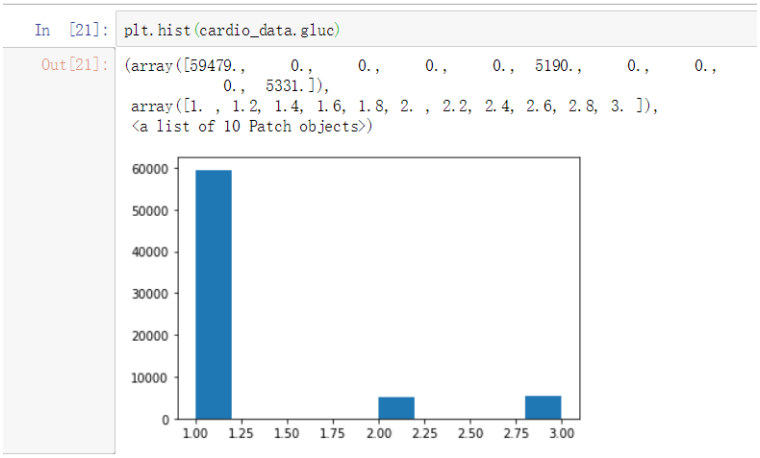


Figure 13: Distribution of gluc

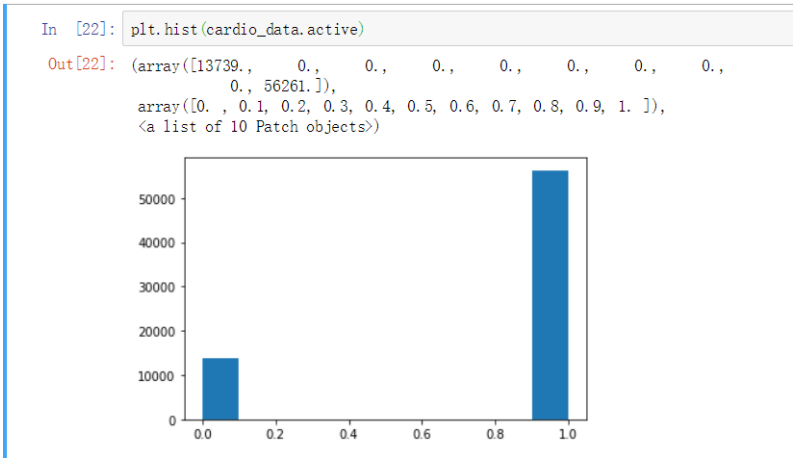


Figure 14: Distribution of active

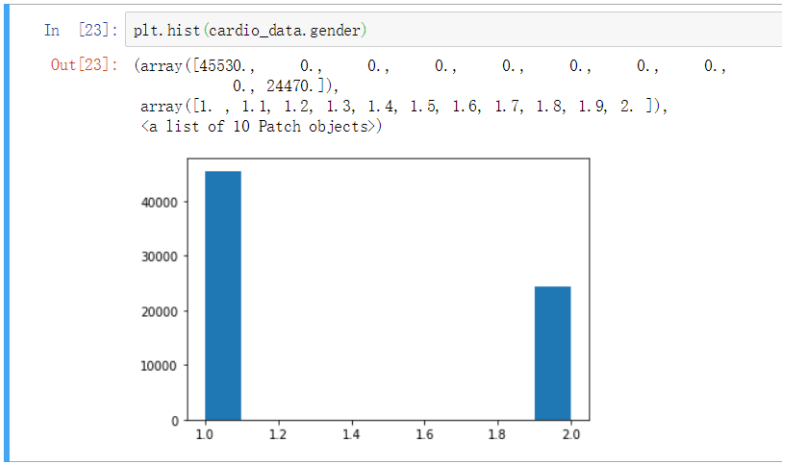


Figure 15: Distribution of gender

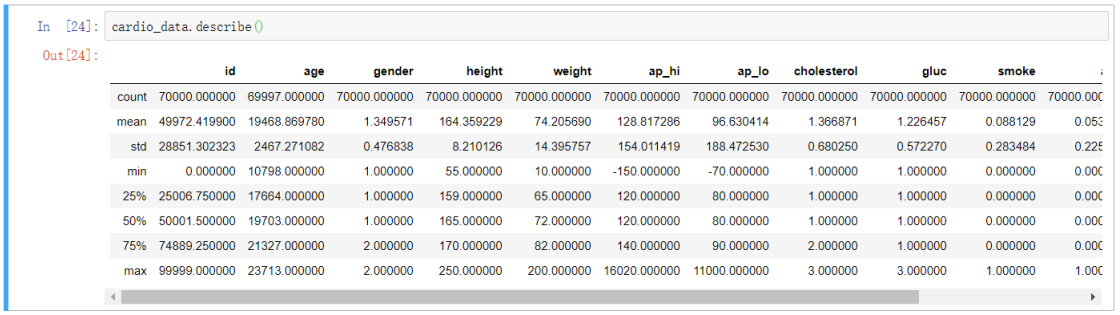
From the distribution of categorical features, I can see that most of them are imbalance. These imbalanced features may affect the model precision. I should balance them before I creating model.

Hypothesis and Features Analysis

- **Hypotheses 1:** The unbalanced features will affect the model performance.
- **Hypotheses 2:** The age feature should be normalized, otherwise, it will affect the model performance.
- **Hypotheses 3:** Not all the features are useful for building model.

2.4 Verify The Data Quality

From the figure 14,15,16 I can see that there is missing value in **age** feature. There are also outliers and extreme values in features **age**, **height**, **weight**, **ap_hi**, and **ap_lo**. These outliers and extreme values may caused by **Data errors** or **Measurement errors**. I should take care of these instances when I building the model (I may choose a suitable model to handle these outliers and extreme values or simply remove these instances and see how the performance(accuracy) of the model changed).

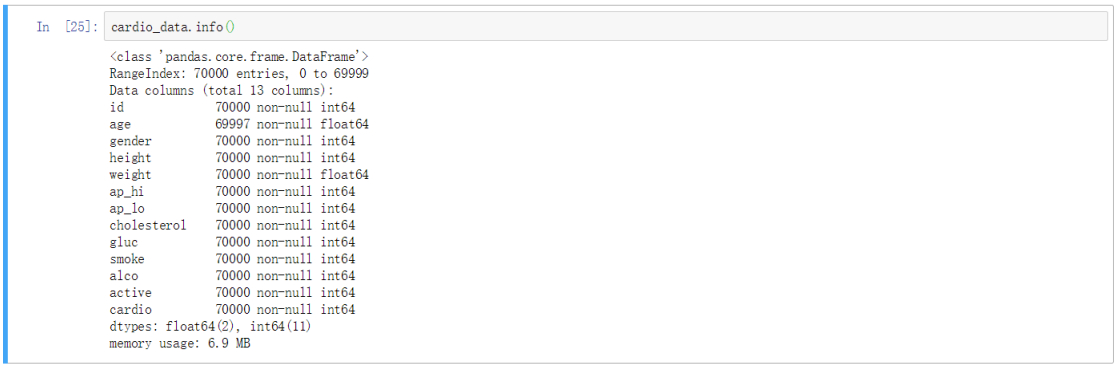


```
In [24]: cardio_data.describe()
```

Out[24]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	
count	70000.000000	69997.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000000	70000.000
mean	49972.419900	19468.869780	1.349571	164.359229	74.205690	128.817286	96.630414	1.366871	1.226457	0.088129	0.053
std	28851.302323	2467.271082	0.476838	8.210126	14.395757	154.011419	188.472530	0.680250	0.572270	0.283484	0.225
min	0.000000	10798.000000	1.000000	55.000000	10.000000	-150.000000	-70.000000	1.000000	1.000000	0.000000	0.000
25%	25006.750000	17664.000000	1.000000	159.000000	65.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000
50%	50001.500000	19703.000000	1.000000	165.000000	72.000000	120.000000	80.000000	1.000000	1.000000	0.000000	0.000
75%	74889.250000	21327.000000	2.000000	170.000000	82.000000	140.000000	90.000000	2.000000	1.000000	0.000000	0.000
max	99999.000000	23713.000000	2.000000	250.000000	200.000000	16020.000000	11000.000000	3.000000	3.000000	1.000000	1.000

Figure 16: Description of data



```
In [25]: cardio_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
id              70000 non-null int64
age             69997 non-null float64
gender          70000 non-null int64
height          70000 non-null int64
weight          70000 non-null float64
ap_hi           70000 non-null int64
ap_lo           70000 non-null int64
cholesterol     70000 non-null int64
gluc            70000 non-null int64
smoke           70000 non-null int64
alco            70000 non-null int64
active          70000 non-null int64
cardio          70000 non-null int64
dtypes: float64(2), int64(11)
memory usage: 6.9 MB
```

Figure 17: Information of data

```
In [26]: cardio_data.isnull()
```

Out[26]:

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	False	True	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	True	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69995	False	False	False	False	False	False	False	False	False	False	False	False	False
69996	False	False	False	False	False	False	False	False	False	False	False	False	False
69997	False	False	False	False	False	False	False	False	False	False	False	False	False
69998	False	False	False	False	False	False	False	False	False	False	False	False	False
69999	False	False	False	False	False	False	False	False	False	False	False	False	False

70000 rows × 13 columns

Figure 18: Null value of data

3 Data Preparation

3.1 Select the Data

The original dataset contains 70,000 rows(instances) and 13 columns(features). It is easy to see that the feature **id** is useless since it is unique identifier. Therefore I should remove it during the mining process. The codes and results are shown in Figure 19.

```
In [2]: cardio_data = pd.read_csv('cardio_train_new.csv')
        cardio_data = cardio_data.drop('id',axis=1)
        print(cardio_data.dtypes)

age            float64
gender           int64
height           int64
weight         float64
ap_hi            int64
ap_lo            int64
cholesterol      int64
gluc             int64
smoke            int64
alco             int64
active           int64
cardio           int64
dtype: object
```

Figure 19: Remove **id** feature

Since different model will give me different feature importance, I will not do the feature selection here.

3.2 Clean the data

3.2.1 - Missing Values

As discussed previously in section 2.4, there are only 3 missing values, therefore, I can simply delete these three data and don't need to worry about them will affect the model performance. I use the **dropna** function to do that. The codes and results are shown in Figure 20.

```
In [3]: cardio_data.dropna(inplace=True)
        print(cardio_data.describe())
```

	age	gender	height	weight	ap_hi	\
count	69997.000000	69997.000000	69997.000000	69997.000000	69997.000000	
mean	19468.869780	1.349558	164.359215	74.205485	128.817235	
std	2467.271082	0.476834	8.210225	14.395786	154.014683	
min	10798.000000	1.000000	55.000000	10.000000	-150.000000	
25%	17664.000000	1.000000	159.000000	65.000000	120.000000	
50%	19703.000000	1.000000	165.000000	72.000000	120.000000	
75%	21327.000000	2.000000	170.000000	82.000000	140.000000	
max	23713.000000	2.000000	250.000000	200.000000	16020.000000	

	ap_lo	cholesterol	gluc	smoke	alco	\
count	69997.000000	69997.000000	69997.000000	69997.000000	69997.000000	
mean	96.630841	1.366859	1.226467	0.088132	0.053774	
std	188.476548	0.680234	0.572281	0.283489	0.225572	
min	-70.000000	1.000000	1.000000	0.000000	0.000000	
25%	80.000000	1.000000	1.000000	0.000000	0.000000	
50%	80.000000	1.000000	1.000000	0.000000	0.000000	
75%	90.000000	2.000000	1.000000	0.000000	0.000000	
max	11000.000000	3.000000	3.000000	1.000000	1.000000	

	active	cardio
count	69997.000000	69997.000000
mean	0.803720	0.499707
std	0.397185	0.500003
min	0.000000	0.000000
25%	1.000000	0.000000
50%	1.000000	0.000000
75%	1.000000	1.000000
max	1.000000	1.000000

Figure 20: Drop Missing Value

3.2.2 - Extreme Values and Outliers

From previous research in section 2.4, I can see that there are outliers and extreme values in **age**, , **height**, **weight**, **ap_hi**, and **ap_lo** features which may caused by **Data errors** or **Measurement errors**. I should do further research in these features and the boxplot will help me to do that.

```
In [4]: plt.boxplot(cardio_data.age)

Out[4]: {'whiskers': [<matplotlib.lines.Line2D at 0x2004efbc400>,
  <matplotlib.lines.Line2D at 0x2004efbc748>],
 'caps': [<matplotlib.lines.Line2D at 0x2004efbca90>,
  <matplotlib.lines.Line2D at 0x2004efbcdd8>],
 'boxes': [<matplotlib.lines.Line2D at 0x2004efbc128>],
 'medians': [<matplotlib.lines.Line2D at 0x2004efd4160>],
 'fliers': [<matplotlib.lines.Line2D at 0x2004efd44a8>],
 'means': []}
```

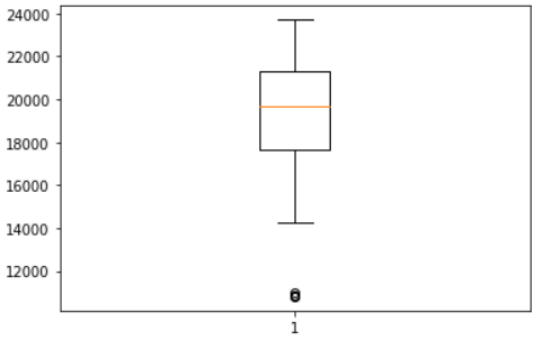


Figure 21: Boxplot for age

```
In [5]: plt.boxplot(cardio_data.height)

Out[5]: {'whiskers': [<matplotlib.lines.Line2D at 0x2004f06f748>,
  <matplotlib.lines.Line2D at 0x2004f06fb00>],
 'caps': [<matplotlib.lines.Line2D at 0x2004f06fe48>,
  <matplotlib.lines.Line2D at 0x2004f07a1d0>],
 'boxes': [<matplotlib.lines.Line2D at 0x2004f06f438>],
 'medians': [<matplotlib.lines.Line2D at 0x2004f07a518>],
 'fliers': [<matplotlib.lines.Line2D at 0x2004f07a860>],
 'means': []}
```

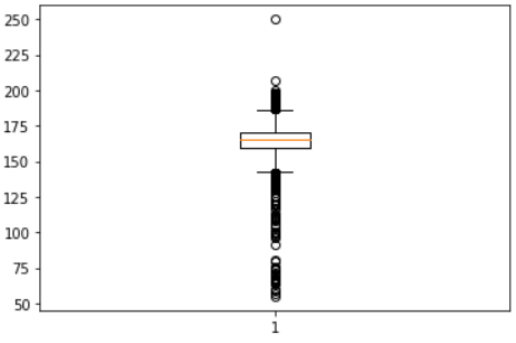


Figure 22: Boxplot for height

```
In [6]: plt.boxplot(cardio_data.weight)

Out[6]: {'whiskers': [<matplotlib.lines.Line2D at 0x2004f0e0748>,
  <matplotlib.lines.Line2D at 0x2004f0e0ac8>],
 'caps': [<matplotlib.lines.Line2D at 0x2004f0e0e10>,
  <matplotlib.lines.Line2D at 0x2004f0ea198>],
 'boxes': [<matplotlib.lines.Line2D at 0x2004f0e05f8>],
 'medians': [<matplotlib.lines.Line2D at 0x2004f0ea4e0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2004f0ea828>],
 'means': []}
```

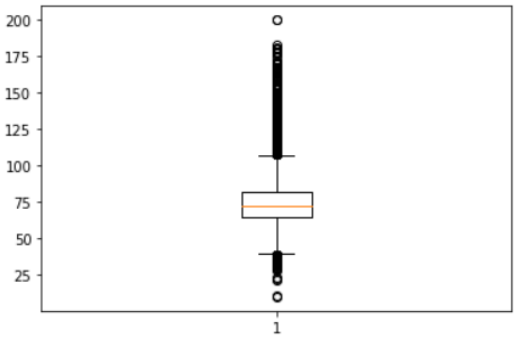


Figure 23: Boxplot for weight

```
In [7]: plt.boxplot(cardio_data.ap_hi)

Out[7]: {'whiskers': [<matplotlib.lines.Line2D at 0x2004f151080>,
  <matplotlib.lines.Line2D at 0x2004f151400>],
 'caps': [<matplotlib.lines.Line2D at 0x2004f151748>,
  <matplotlib.lines.Line2D at 0x2004f151a90>],
 'boxes': [<matplotlib.lines.Line2D at 0x2004f142ef0>],
 'medians': [<matplotlib.lines.Line2D at 0x2004f151dd8>],
 'fliers': [<matplotlib.lines.Line2D at 0x2004f15e160>],
 'means': []}
```

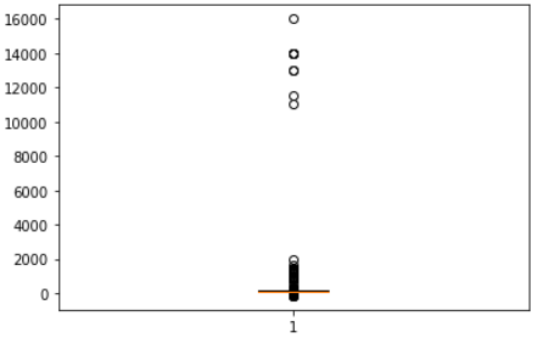


Figure 24: Boxplot for ap_hi

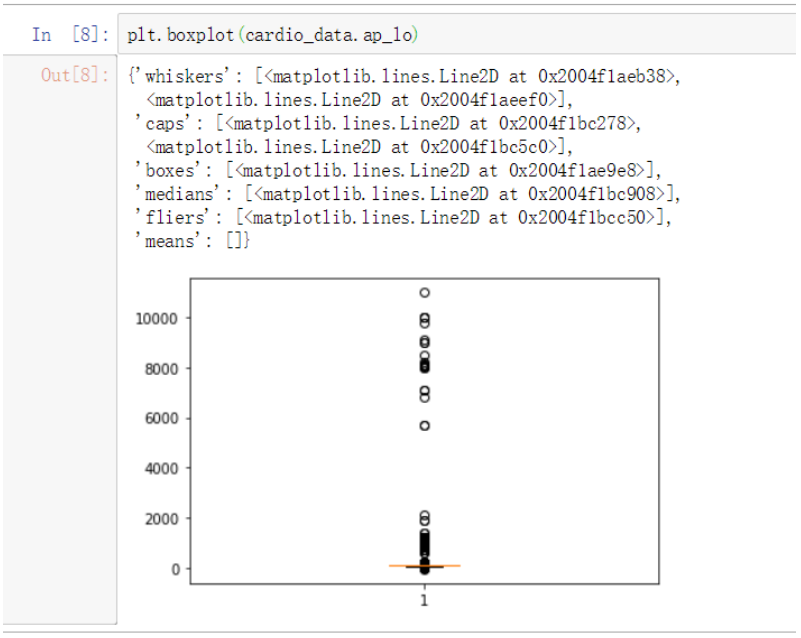


Figure 25: Boxplot for ap_lo

From the given plots, I can see that the outliers for **age**, **height**, and **weight** are in reasonable range. I can ignore them. However, **ap_hi** and **ap_lo** are not. I should take care of them.

There are different ways to handler the outliers and extreme values:

- **Coerce**: Replaces outliers and extreme values with the nearest value that would not be considered extreme. For example if an outlier is defined to be anything above or below three standard deviations, then all outliers would be replaced with the highest or lowest value within this range [9].
- **Discard**: Discards records with outlying or extreme values for the specified field [9].
- **Nullify**: Replaces outliers and extremes with the null or system-missing value [9].
- **Coerce outliers / discard extremes**: Discards extreme values only [9]. **Coerce outliers / nullify extremes**: Nullifies extreme values only [9].

In my cases, I can see that in both **ap_hi** and **ap_lo** features, almost all the instances are in a reasonable range. The outliers are in a ridiculous position. Therefore, I choose to discard these outliers and extreme values instead replacing them with nearest value.

The medical paper published by Professor Peter [20] shows that normally, the limitation of human blood pressure is between 40 to 220. Therefore, the instances with blood pressure larger than 220 or less than 40 must be caused by **Data errors** or **Measurement errors**, I need to remove them. The following code will help me remove the outlier and the results are shown in Figure 26, 27. After I remove these outliers, all things look fine.

```
In [9]: cardio_data = cardio_data.loc[cardio_data.ap_hi <= 220]
        cardio_data = cardio_data.loc[cardio_data.ap_hi >= 40]
        cardio_data = cardio_data.loc[cardio_data.ap_lo <= 220]
        cardio_data = cardio_data.loc[cardio_data.ap_lo >= 40]
        cardio_data.shape
Out[9]: (68764, 12)
```

Figure 26: Remove outliers and extreme values

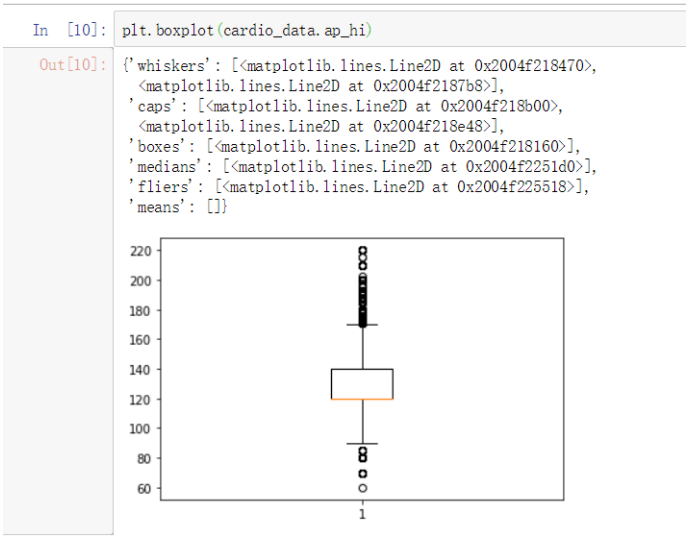


Figure 27: Boxplot for ap_hi after removing outliers

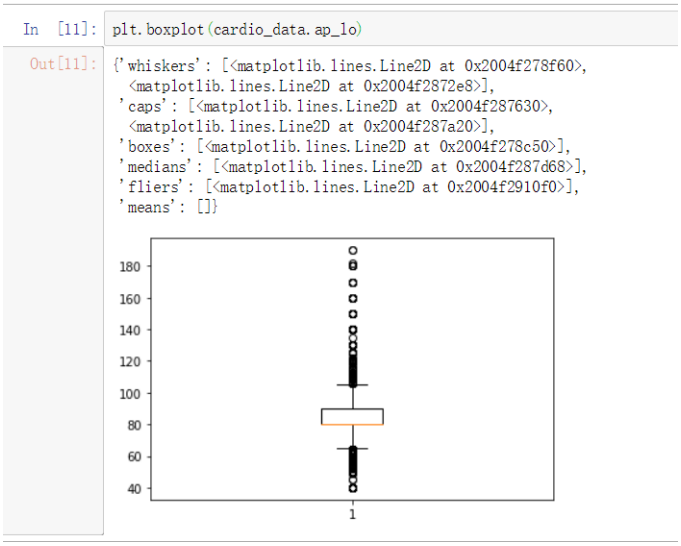


Figure 28: Boxplot for ap_lo after removing outliers

3.3 Construct the data

In this research, all features are collected by professional staffs. I cannot make new feature because the lack of professional medical skills.

3.4 Integrate various data sources

The data was originally stored in two separate files **cardio_train_1.csv** and **cardio_train_2.csv**, the following codes will help me to integrate these two files into one new file - **cardio_train_new.csv**

```
In [ ]: file_1_path = "cardio_train_1.csv"
        file_2_path = "cardio_train_2.csv"

        data_list = []
        with open('cardio_train_1.csv') as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=',')
            next(csv_reader)
            for row in csv_reader:
                temp_row = row[0].split(';')
                data_list.append(temp_row)
        with open('cardio_train_2.csv') as csv_file:
            csv_reader = csv.reader(csv_file, delimiter=',')
            for row in csv_reader:
                temp_row = row[0].split(';')
                data_list.append(temp_row)

        with open ('cardio_train_new.csv', mode='w', newline="") as csv_file:
            csv_writer = csv.writer(csv_file)
            csv_writer.writerow(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardi
            for row in data_list:
                id = row[0]
                age = row[1]
                gender = row[2]
                height = row[3]
                weight = row[4]
                ap_hi = row[5]
                ap_lo = row[6]
                cholesterol = row[7]
                gluc = row[8]
                smoke = row[9]
                alco = row[10]
                active = row[11]
                cardio = row[12]
                csv_writer.writerow([id, age, gender, height, weight, ap_hi, ap_lo, cholesterol, gluc, smoke, alco, active, cardio])
```

Figure 29: File Integration Code

3.5 Format the data as required

When I first time load the data into program, the data format are all numeric as shown in Figure 30. I should set them correctly as discussed in section 2.2.2.

```
In [12]: cardio_data.gender= cardio_data.gender.astype('category')
         cardio_data.height = cardio_data.height.astype('float64')
         cardio_data.ap_hi = cardio_data.ap_hi.astype('float64')
         cardio_data.ap_lo = cardio_data.ap_lo.astype('float64')
         cardio_data.cholesterol = cardio_data.cholesterol.astype('category')
         cardio_data.gluc = cardio_data.gluc.astype('category')
         cardio_data.smoke = cardio_data.smoke.astype('bool')
         cardio_data.alco = cardio_data.alco.astype('bool')
         cardio_data.active = cardio_data.active.astype('bool')
         cardio_data.cardio = cardio_data.cardio.astype('bool')
```

Figure 30: Initial Formatting

After setting the format correctly, I can go to the next step. The codes and results are shown in Figure 31.

```
In [13]: cardio_data.dtypes

Out[13]: age             float64
         gender         category
         height          float64
         weight          float64
         ap_hi           float64
         ap_lo           float64
         cholesterol    category
         gluc           category
         smoke              bool
         alco               bool
         active             bool
         cardio             bool
         dtype: object
```

Figure 31: Codes and Results for Formatting

4 Data Transformation

4.1 Reduce the data

As shown in Figure 32 and 33, the distribution of **cholesterol** and **gluc** is unbalanced, therefore, I merge the category 2 (above normal) and category 3 (well above normal) into one single category - abnormal which will make the distribution of these two features more balance and reduce the redundant category.

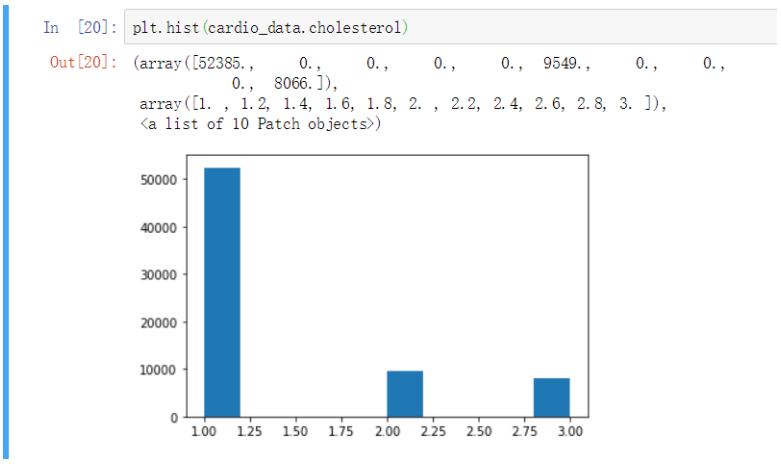


Figure 32: Distribution of cholesterol

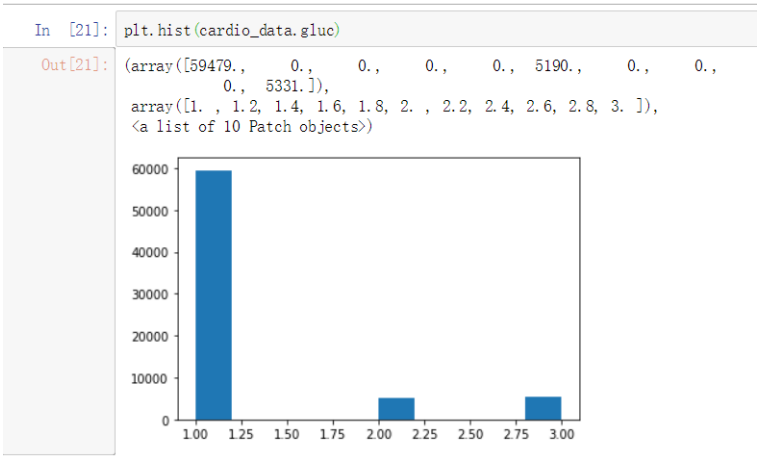


Figure 33: Distribution of gluc

I can reclassify the categories by following code:

```
In [5]: cholesterol = cardio_data.cholesterol
new_cholesterol = []
for c in cholesterol:
    if c == 1:
        new_cholesterol.append(1)
    else:
        new_cholesterol.append(2)
cardio_data['new_cholesterol'] = new_cholesterol
cardio_data.new_cholesterol = cardio_data.new_cholesterol.astype('category')
gluc = cardio_data.gluc
new_gluc = []
for g in gluc:
    if g == 1:
        new_gluc.append(1)
    else:
        new_gluc.append(2)
cardio_data['new_gluc'] = new_gluc
cardio_data.new_gluc = cardio_data.new_gluc.astype('category')
```

Figure 34: Reclassify Categories

After I reclassify the categories, the distribution of **cholesterol** and **gluc** is shown in Figure 35 and 36. I can see that after I reclassify the categories, the distribution looks much better than before and the redundant category is removed. Since the distribution of **cardio** is balanced, the imbalanced distribution of these two features won't affect the model performance.

```
In [4]: plt.hist(cardio_data.new_cholesterol)
Out[4]: (array([51570.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
                0.,  17194.]),
 array([1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. ]),
 <a list of 10 Patch objects>)
```

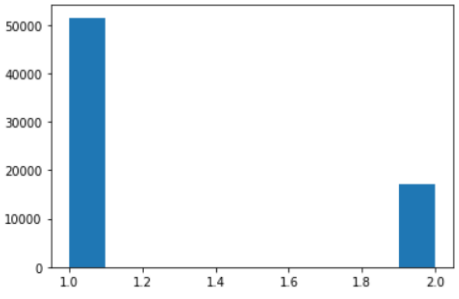


Figure 35: Distribution of new_cholesterol

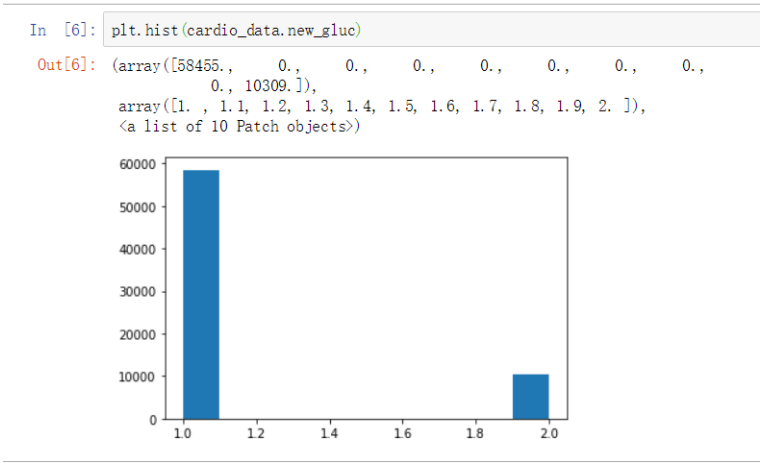


Figure 36: Distribution of new_gluc

4.2 Project the data

In this research, I don't need to project the response data because it is a classification problem. If I use classical logistic regression model, it will automatically take log for the response variable since it calculates odds instead of probability. If I use decision trees algorithms or neural network, there is no need for response variable transformation since they are not based on statistics.

4.3 Feature Selection

For the feature selection, I use two ways in python **ANOVA F-Test** and **Correlation Matrix**. As shown in Figure 37, 38 and 39 the feature importance and correlation of each feature is quite different. However, since we only have 13 features in the dataset. Therefore, we should not remove any feature now. Leaving them for further researching.

In [17]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
bestfeatures = SelectKBest(score_func=f_classif , k=10)
X = cardio_data.drop('cardio', axis=1)
Y = cardio_data.cardio
bestfeatures.fit(X, Y)
scores = pd.DataFrame(bestfeatures.scores_)
columns = pd.DataFrame(X.columns)
feature_scores = pd.concat([columns, scores], axis=1)
print(feature_scores)
```

```
                0            0
0             age  4188.463674
1          gender     3.541935
2          height     8.023304
3          weight  2295.291204
4           ap_hi 15209.465756
5           ap_lo  8694.172007
6     cholesterol  3549.471647
7            gluc   563.333216
8           smoke    18.312320
9            alco     4.704894
10         active    95.319684
11  new_cholesterol  3029.384442
12        new_gluc   582.015629
```

Figure 37: ANOVA F-Test Result

In [20]:
```
import seaborn as sns
plt.figure(figsize=(14,14))
sns.heatmap(cardio_data.corr(),
            vmin=-1,
            cmap='coolwarm',
            annot=True);
```

Figure 38: Correlation Matrix Code

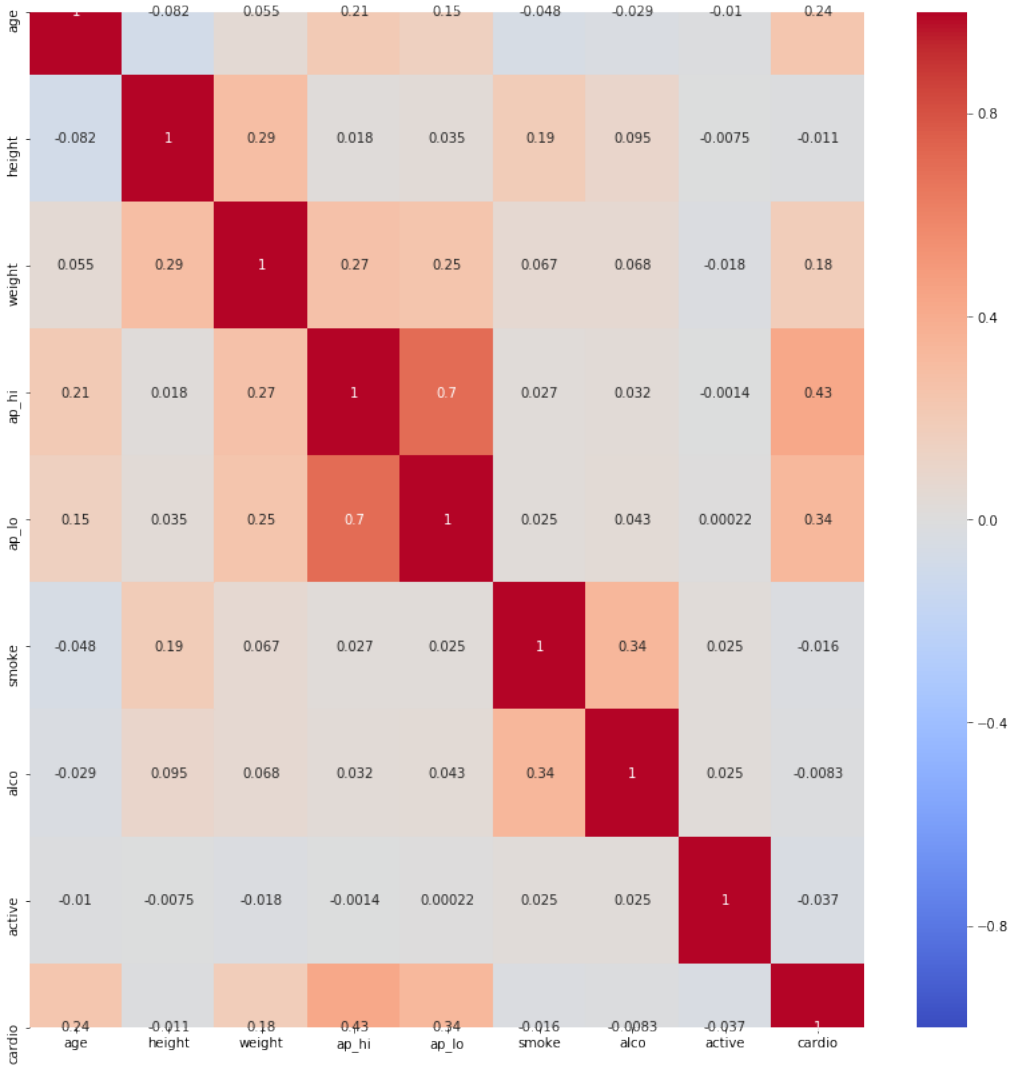


Figure 39: Correlation Matrix Result

5 Data Mining Methods Selection

5.1 Match and Discuss the Objectives of Data Mining to Data Mining Methods

This research focus on the following data mining goals:

1. **Explanation:** Finding out important features relevant to whether a person get cardiovascular diseases. Explain the relation between features and response variable.
2. **Explanation:** Describing how the features will affect the probability of a person get cardiovascular diseases. Explain the latitude of the features which affect the response variable.
3. **Prediction:** Given a person's details, predict whether this person has cardiovascular diseases. Predict the response variable by provided features.

5.2 Select the Appropriate Data Mining Methods Based on Distribution

There are three types methods for data mining:

- **Supervised Learning - Classification**: It predicts discrete number of values. In Classification the data is categorized under different labels according to some parameters and then the labels are predicted for the data. Classifying emails as either spam or not spam is example of classification problem [15].
- **Supervised Learning - Regression**: It predicts continuous valued output. The Regression analysis is the statistical model which is used to predict the numeric data instead of labels. It can also identify the distribution trends based on the available data or historic data. Predicting a person's income from their age, education is an example of regression task [15].
- **Unsupervised Learning - Clustering**: Clustering is the task of partitioning the dataset into groups. The goal is to split up the data in such a way that points within single cluster are very similar and points in different clusters are different. It determines grouping among unlabeled data [15].

In our dataset, the reponse variable **cardio** is labeled. Therefore, I won't use clustering. On the other hand, the type of our reponse variable **cardio** is binary, 0 stand for no cardiovascular disease, 1 stand for with cardiovascular disease. Therefore, I should use classification in our research.

6 Data Mining Algorithms Selection

sklearn offers a variety of modeling methods taken from machine learning, artificial intelligence, and statistics. The methods available on the Modeling palette allow us to derive new information from our data and to develop predictive models. Each method has certain strengths and is best suited for particular types of problems [22].

6.1 Conduct Exploratory Analysis and Discuss

Firstly, I should double check the data quality to make sure it is ready to move forward. Based on the previous steps, the dataset is completely processed and ready for modeling.

Secondly, I should use various graphs and tables to summarize the dataset. Visualization makes the dataset easier to understand and interpret.

Finally, because I want to use classification methods in our research, I should choose the classification models carefully. Classification models use the values of one or more **inputs (features)** fields to predict the value of **target (reponse variable)**. Sklearn provides various classification models: decision trees (C&R Tree, C5.0 algorithms), regression (linear, logistic, and generalized linear regression algorithms), neural networks, support vector machines, and Bayesian networks. All these algorithms have their own advantages and disadvantages.

6.1.1 - Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features [21].

Use decision tree models to develop classification systems that predict or classify future observations based on a set of decision rules. If I have data divided into classes, I can use our data to build rules that I can use to classify old or new cases with maximum accuracy.

This approach, sometimes known as rule induction, has several advantages. First, the reasoning process behind the model is clearly evident when browsing the tree. This is in contrast to other black box modeling techniques in which the internal logic can be difficult to work out [7].

Second, the process automatically include in its rule only the attributes that really matter in making a decision. Attributes that do not contribute to the accuracy of the tree are ignored. This can yield very useful information about the data and

can be used to reduce the data to relevant fields before training another learning technique, such as a neural net [7].

Last but not least, decision tree model nuggets can be converted into a collection of if-then rules (a rule set), which in many cases show the information in a more comprehensible form. The decision-tree presentation is useful when I want to see how attributes in the data can split, or partition, the population into subsets relevant to the problem. The rule set presentation is useful if I want to see how particular groups of items relate to a specific conclusion [7].

Advantage of decision trees:

- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values [21].
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree [21].
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information [21].
- Able to handle multi-output problems [21].
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret [21].
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model [21].
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated [21].

Disadvantage of decision trees:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem [21].
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by

using decision trees within an ensemble [21].

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement [21].
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems [21].
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree [21].

6.1.1.1 - CART Tree

- **Definition**: The Classification and Regression (C&R) Tree node is a tree-based classification and prediction method. Similar to C5.0, this method uses recursive partitioning to split the training records into segments with similar response variable [6].
- **Requirements**: To train a C&R Tree model, I need one or more features and exactly one response variable. Response variable and features can be continuous or categorical [6].
- **Strengths**: C&R Tree are quite robust in the presence of problems such as missing data and large number of fields. They usual do not require long training times to estimate. In addition, C&R Tree models tend to be easier to understand than some other model types - the rules derived from the model have a very straightforward interpretation. Unlike C5.0, C&R Tree can accommodate continuous as well as categorical response variable [6].

6.1.1.2 - C5.0

- **Definition**: A C5.0 model works by splitting the sample based on the field that provides the maximum information gain. Each subsample defined by the first split is then split again, usually based on a different field, and the process repeats until the subsamples cannot be split any further. Finally, the lowest-level splits are reexamined, and those that do not contribute significantly to the value of the model are removed or pruned [5].

- **Requirements**: To train a C5.0 model, there must be one categorical response variable, and one ore more input features of any type [5].
- **Strengths**: C5.0 models are quite robust in the presence of problems such as missing data and large number of input features. They usually do not require long training times to estimate. In addition, C5.0 models tend to be easier to understand than some other model types, since the rules derived from the model have a very straightforward interpretation. C5.0 also offers the powerful boosting method to increase accuracy of classification [5].

6.1.2 - Statistical Models

Statistical models use mathematical equations to encode information extracted from the data. In some cases, statistical modeling techniques can provide adequate models very quickly. Even for problems in which more flexible machine-learning techniques (such as neural networks) can ultimately give better results, I can use some statistical models as baseline predictive models to judge the performance of more advanced techniques [13].

6.1.2.1 - Linear Regression

- **Definition**: Linear Regression is a common statistical technique for classifying instances based on the values of numeric input features. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual response variables [10].
- **Requirements**: Only numeric fields can be used in a linear regression model. I must have exactly one response variable and one or more features [10].
- **Strengths**: Linear regression models are relatively simple and give an easily interpreted mathematical formula for generating predictions. Because linear regression is a long-established statistical procedure, the properties of these models are well understood. Linear models are also typically very fast to train. The linear node provides methods for automatic feature selection in order to eliminate nonsignificant input features from the equation [10].

6.1.2.2 - Logistic Regression

- **Definition**: Logistic regression, also known as nominal regression, is a statistical technique for classifying instances based on values of input features. It is analogous to linear regression but takes a categorical response variable instead of a numeric one. Both binomial models (for response variable with two discrete categories) and multinomial models (for response variable with

more than two categories) are supported [11]. Logistic Regression works by building a set of equations that relate the input feature values to the probabilities associated with each of the output field categories. Once the model is generated, it can be used to estimate probabilities for new data. For each instance, a probability of membership is computed for each possible output category. The response variable with the highest probability is assigned as the predicted output value for that instance [11].

- **Requirements**: One or more input features and exactly one categorical response variable with two or more categories. For a binomial model response variable must have a measurement level of Flag [11].
- **Strengths**: Logistic regression models are often quite accurate. They can handle symbolic and numeric input features. They can give predicted probabilities for all target categories so that a second best guess can easily be identified. Logistic models are most effective when group membership is a truly categorical field [11].

6.1.2.3 - Generalized Linear Regression

- **Definition**: The generalized linear model expands the general linear model expands the general linear model so that the dependent variable is linearly related to the factors and covariates via a specified link function. Moreover, the model allows for the dependent variable to have a non-normal distribution. It covers widely used statistical models, such as linear regression for normally distribution responses, logistic models for binary data, loglinear models for count data, complementary log-log models for interval-censored survival data, plus many other statistical models through its very general model formulation [8].
- **Requirements**: I need one or more input features and exactly one response variable with two or more categories [8].
- **Strengths**: The generalized linear model is extremely flexible, but the process of choosing the model structure is not automated and thus demands a level of familiarity with our data that is not required by "black box" algorithms [8].

6.1.3 - Neural Network

- **Definition**: A neural network can approximate a wide range of predictive models with minimal demands on model structure and assumption. The form of the relationships is determined during the learning process. If a linear relationship between the response and predictors is appropriate, the results

of the neural network should closely approximate those of a traditional linear model. If a nonlinear relationship is more appropriate, the neural network will automatically approximate the "correct" model structure [12],

- **Requirements**: There must be at least one response variable and one input feature [12].
- **Strengths**: The trade-off for this flexibility is that the neural network is not easily interpretable. If I are trying to explain an underlying process that produces the relationships between the response variable and features, it would be better to use a more traditional statistical model. However, if the model interpretability is not important, I can obtain good predictions using a neural network [12].

6.1.4 - Support Vector Machine

Support Vector Machine (SVM) is a robust classification and regression technique that maximizes the predictive accuracy of a model without overfitting the training data. SVM is particular suited to analyzing data with very large numbers of features [14].

The advantages of SVM:

- Effective in high dimensional spaces [23].
- Still effective in cases where number of dimensions is greater than the number of samples [23].
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient [23].
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient [23].

The disadvantages of SVM:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial [23].
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below) [23].

6.1.5 - Bayesian Networks

- **Definition**: The Bayesian Network node enables you to build a probability model by combining observed and recorded evidence with "common-sense" real-world knowledge to establish the likelihood of occurrences by using seemingly unlinked attributes. The node focuses on Tree Augmented Naive Bayes (TAN) and Markov Blanket networks that are primarily used for classification [4].
- **Requirements**: Response variable must be categorical and can have a measurement level of Nominal, Ordinal or Flag. Input features can be fields of any type. Continuous (numeric range) input features will be automatically binned; however, if the distribution is skewed, I may obtain better results by manually binning the fields using a Binning node before the Bayesian Network node [4].

6.2 Select Data Mining Algorithms Based on Discussion

In this research ,we are mainly focus on explain the relation between different features and response variable and predict the whether a person get cardiovascular disease by provided features. Therefore, I have two clearly data mining goals: **explanation** and **prediction**. Since I have two data mining goals, I may use different algorithms for each goal.

Firstly, I will use **Logistic Regression Model** as benchmark because logistic regression model is the most effective classification algorithm when group membership is a truly categorical field. It is easy to apply and have statistic explanation for the generated model. I won't use Linear Regression and Cox Regression here because they are not suitable for classification. The Generalized Linear Regression is also acceptable but I already know I need classification, therefore, I don't need to use it.

After that, I will use decision tree to build a explanation model compared to logistic model. I will use C5.0 as my decision tree algorithm than others because it can generate robust model. The C5.0 model tends to be easier to understand than some other model types, since the rules derived from the model have a very straightforward interpretation. C5.0 also offers the powerful boosting methods to increase accuracy of classification. I will also use **Random Forest** algorithm which is an ensemble algorithm based on decision tree and it always performs better than single decision tree algorithm.

6.3 Build/Select Appropriate Models and Choose Relevant Parameters

6.3.1 - Logistic Regression Model Parameters Setting

As shown in Figure 40, I build 3 logistic model. For logistic model 1 and logistic model 2, I'm trying to discover how the regularization parameter affect the model performance. For logistic model 1 and logistic model 3, I'm trying to discover how the number of iteration affect the model performance. The rest of parameters I will leave them default.

```
Logistic_model_1 = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
Logistic_model_2 = LogisticRegression(regParam=1, maxIter=100, labelCol='cardio')
Logistic_model_3 = LogisticRegression(regParam=0, maxIter=200, labelCol='cardio')
```

Figure 40: Parameters Setting for Logistic Regression Model

6.3.2 - C5.0 Decision Tree

As shown in Figure 41, I build 2 decision tree classifier model. I'm trying to find how the impurity function affect the model performance. The rest of parameters I will leave them default.

```
decision_tree_model_1 = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
decision_tree_model_2 = DecisionTreeClassifier(impurity='entropy', labelCol='cardio')
```

Figure 41: Parameters Setting for Decision Tree Model

6.3.3 - Random Forest

As shown in Figure 42, I build 3 random forest classifier model. For random forest model 1 and random forest model 2, I'm trying to find how the number of trees affect the model performance. For random forest model 1 and random forest model 3, I'm trying to find how the impurity function affect the model performance. The rest of parameters I will leave them default.

```
random_forest_model_1 = RandomForestClassifier(numTrees=5, impurity='gini', labelCol='cardio')
random_forest_model_2 = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
random_forest_model_3 = RandomForestClassifier(numTrees=5, impurity='entropy', labelCol='cardio')
```

Figure 42: Parameters Setting for Random Forest Model

7 Data Mining

7.1 Create and Justify Test Designs

If I use the same data for **model selection**, **model fitting** and to **evaluate the precision of the predictions**. It is almost certain that model will fit the data better than it will fit the population that the data came from. As a result our evaluation of precision is almost certainly too optimistic.

Therefore, I need to separate our data into training and testing sets. Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when I separate a dataset into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to help ensure that the testing and training sets are similar. By using similar data for training and testing, I can minimize the effects of data discrepancies and better understand the characteristics of the model [16].

After a model has been processed by using the training set, I test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that I want to predict, it is easy to determine whether the model's guesses are correct [16].

By Pareto principle [19], I usually separate our data by 80% training set and 20% test set. The following code will split the data into training set and testing set.

```
train_data, test_data = cardio_data_pyspark.randomSplit([0.8,.2])
```

Figure 43: Split Dataset

7.2 Conduct Data Mining

7.2.0 - Pandas to Pyspark

In previous section, I use Pandas dataframe to preprocess the data since it is more convenient. However, if I want to build model by pyspark machine learning module, I should convert the pandas dataframe to pyspark dataframe. As shown in Figure 44,45,46. The pyspark does not support categorical type, therefore, I have to convert all categorical feature to string. The pyspark also requires the target(response) variable to be numeric, however, it is boolean type in my dataset. Therefore, I should also transform it to numeric type.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

cardio_data = pd.read_csv('cardio_train_new.csv')
cardio_data = cardio_data.drop('id', axis=1)
cardio_data.dropna(inplace=True)

cardio_data = cardio_data.loc[cardio_data.ap_hi <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_hi >= 40]
cardio_data = cardio_data.loc[cardio_data.ap_lo <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_lo >= 40]

cardio_data.gender= cardio_data.gender.astype('str')
cardio_data.height = cardio_data.height.astype('float64')
cardio_data.ap_hi = cardio_data.ap_hi.astype('float64')
cardio_data.ap_lo = cardio_data.ap_lo.astype('float64')
cardio_data.cholesterol = cardio_data.cholesterol.astype('category')
cardio_data.gluc = cardio_data.gluc.astype('category')
cardio_data.smoke = cardio_data.smoke.astype('bool')
cardio_data.alco = cardio_data.alco.astype('bool')
cardio_data.active = cardio_data.active.astype('bool')
cardio_data.cardio = cardio_data.cardio.astype('int64')

cholesterol = cardio_data.cholesterol
new_cholesterol = []
for c in cholesterol:
    if c == 1:
        new_cholesterol.append(1)
    else:
        new_cholesterol.append(2)
cardio_data['new_cholesterol'] = new_cholesterol
cardio_data.new_cholesterol = cardio_data.new_cholesterol.astype('str')

gluc = cardio_data.gluc
new_gluc = []
for g in gluc:
    if g == 1:
        new_gluc.append(1)
    else:
        new_gluc.append(2)
cardio_data['new_gluc'] = new_gluc
cardio_data.new_gluc = cardio_data.new_gluc.astype('str')

cardio_data = cardio_data.drop('cholesterol', axis=1)
cardio_data = cardio_data.drop('gluc', axis=1)
```

Figure 44: Dataframe Transformation From Pandas to Pyspark

```
import findspark
findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
import pyspark
from pyspark.sql import *
spark = SparkSession.builder.appName('logistic_regression_adv').getOrCreate()

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import MultilayerPerceptronClassifier
```

```
cardio_data_pyspark = spark.createDataFrame(cardio_data)
```

Figure 45: Dataframe Transformation From Pandas to Pyspark

```
print(type(cardio_data))
print(type(cardio_data_pyspark))
print(cardio_data_pyspark.columns)
cardio_data_pyspark.printSchema()
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pyspark.sql.dataframe.DataFrame'>
['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active', 'cardio', 'new_cholesterol', 'new_gluc']
root
 |-- age: double (nullable = true)
 |-- gender: string (nullable = true)
 |-- height: double (nullable = true)
 |-- weight: double (nullable = true)
 |-- ap_hi: double (nullable = true)
 |-- ap_lo: double (nullable = true)
 |-- smoke: boolean (nullable = true)
 |-- alco: boolean (nullable = true)
 |-- active: boolean (nullable = true)
 |-- cardio: long (nullable = true)
 |-- new_cholesterol: string (nullable = true)
 |-- new_gluc: string (nullable = true)
```

Figure 46: Dataframe Transformation From Pandas to Pyspark

After covert to pyspark dataframe, I should deal with the string feature and combine all features to one vector since it is required by pyspark. As shown in Figure 47 and 48, In Figure 47, I should convert string feature to indexer first, then I convert the indexer to encoder(vector). After that, all of my features are vector and I use assembler to assemble them together to build model. Figure 48 shows the coder transform feature by transformer. Figure 49 shows the transformation result.

```
from pyspark.ml.feature import VectorAssembler,VectorIndexer,OneHotEncoder,StringIndexer
```

```
gender_indexer = StringIndexer(inputCol='gender',outputCol='genderIndex')
gender_encoder = OneHotEncoder(inputCol='genderIndex',outputCol='genderVec')
cholesterol_index = StringIndexer(inputCol='new_cholesterol', outputCol='cholesterolIndex')
cholesterol_encoder = OneHotEncoder(inputCol='cholesterolIndex', outputCol='cholesterolVec')
gluc_indexer = StringIndexer(inputCol='new_gluc', outputCol='glucIndex')
gluc_encoder = OneHotEncoder(inputCol='glucIndex', outputCol='glucVec')
```

```
assembler = VectorAssembler(inputCols=['age',
                                       'genderVec',
                                       'height',
                                       'weight',
                                       'ap_hi',
                                       'ap_lo',
                                       'smoke',
                                       'alco',
                                       'active',
                                       'cholesterolVec',
                                       'glucVec'
], outputCol='features')
```

Figure 47: Convert String Feature to Vector

```
cardio_data_pyspark = gender_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gender_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = cholesterol_index.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = cholesterol_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = gluc_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gluc_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = assembler.transform(cardio_data_pyspark)
```

Figure 48: Convert String Feature to Vector

```
cardio_data_pyspark.columns
```

```
['age',
 'gender',
 'height',
 'weight',
 'ap_hi',
 'ap_lo',
 'smoke',
 'alco',
 'active',
 'cardio',
 'new_cholesterol',
 'new_gluc',
 'genderIndex',
 'genderVec',
 'cholesterolIndex',
 'cholesterolVec',
 'glucIndex',
 'glucVec',
 'features']
```

Figure 49: Convert String Feature to Vector

7.2.1 - Logistic Regression

7.2.1.1 - Logistic Regression Parameters Setting

As shown in Figure 50, I build 3 logistic model. For logistic model 1 and logistic model 2, I'm trying to discover how the regularization parameters affect the model performance. For logistic model 1 and logistic model 3, I'm trying to discover how the number of iteration affect the model performance. The rest of parameters I will leave them default.

```
Logistic_model_1 = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
Logistic_model_2 = LogisticRegression(regParam=1, maxIter=100, labelCol='cardio')
Logistic_model_3 = LogisticRegression(regParam=0, maxIter=200, labelCol='cardio')
```

Figure 50: Parameters Setting for Logistic Regression Model

7.2.1.2 - Logistic Regression Running

```
Logistic_model_1 = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
Logistic_model_2 = LogisticRegression(regParam=1, maxIter=100, labelCol='cardio')
Logistic_model_3 = LogisticRegression(regParam=0, maxIter=200, labelCol='cardio')

l1 = Logistic_model_1.fit(train_data)
l2 = Logistic_model_2.fit(train_data)
l3 = Logistic_model_3.fit(train_data)

pred1 = l1.transform(test_data)
pred2 = l2.transform(test_data)
pred3 = l3.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Logistic Regression Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Logistic Regression Model 2: " + str(AUC2))
AUC3 = evaluator.evaluate(pred3)
print("Logistic Regression Model 3: " + str(AUC3))
```

```
Logistic Regression Model 1: 0.72235186249218
Logistic Regression Model 2: 0.7147894749812257
Logistic Regression Model 3: 0.72235186249218
```

Figure 51: Analysis of Logistic Regression

7.2.1.3 - Logistic Regression Evaluation

As shown in Figure 51, I can see that the model with regularization parameter 0 has higher AUC than the model with regularization parameter 1. The max number of iteration doe not affect the model performance. The AUC is only a simple evaluation for the model, I I will do more evaluation in the next section.

7.2.2 - C5.0 Decision Tree Algorithm

7.2.2.1 - C5.0 Decision Tree Parameters Setting

As shown in Figure 52, I build 2 decision tree classifier model. I'm trying to find how the impurity function affect the model performance. The rest of parameters I will leave them default.

```
decision_tree_model_1 = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
decision_tree_model_2 = DecisionTreeClassifier(impurity='entropy', labelCol='cardio')
```

Figure 52: Parameters Setting for Decision Tree Model

7.2.2.2 - C5.0 Decision Tree Running

```
decision_tree_model_1 = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
decision_tree_model_2 = DecisionTreeClassifier(impurity='entropy', labelCol='cardio')

d1 = decision_tree_model_1.fit(train_data)
d2 = decision_tree_model_2.fit(train_data)

pred1 = d1.transform(test_data)
pred2 = d2.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Decision Tree Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Decision Tree Model 2: " + str(AUC2))
```

```
Decision Tree Model 1: 0.7286841002958867
Decision Tree Model 2: 0.7286841002958867
```

Figure 53: Analysis of C5.0 Decision Tree

7.2.2.3 - C5.0 Decision Tree Evaluation

As shown in Figure 53, the impurity function does not affect the performance of decision tree model.

7.2.3 - Random Forest

7.2.3.1 - Random Forest Parameters Setting

As shown in Figure 54, I build 3 random forest classifier model. For random forest model 1 and random forest model 2, I'm trying to find how the number of trees affect the model performance. For random forest model 1 and random forest model 3, I'm trying to find how the impurity function affect the model performance. The rest of parameters I will leave them default.

```
random_forest_model_1 = RandomForestClassifier(numTrees=5, impurity='gini', labelCol='cardio')
random_forest_model_2 = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
random_forest_model_3 = RandomForestClassifier(numTrees=5, impurity='entropy', labelCol='cardio')
```

Figure 54: Parameters Setting for Random Forest Model

7.2.3.2 - Random Forest Model Running

```
random_forest_model_1 = RandomForestClassifier(numTrees=5, impurity='gini', labelCol='cardio')
random_forest_model_2 = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
random_forest_model_3 = RandomForestClassifier(numTrees=5, impurity='entropy', labelCol='cardio')

clf1 = random_forest_model_1.fit(train_data)
clf2 = random_forest_model_2.fit(train_data)
clf3 = random_forest_model_3.fit(train_data)

pred1 = clf1.transform(test_data)
pred2 = clf2.transform(test_data)
pred3 = clf3.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Random Forest Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Random Forest Model 2: " + str(AUC2))
AUC3 = evaluator.evaluate(pred3)
print("Random Forest Model 3: " + str(AUC3))
```

```
Random Forest Model 1: 0.7196959263015215
Random Forest Model 2: 0.7235781568883579
Random Forest Model 3: 0.7218680230446664
```

Figure 55: Analysis of Random Forest

7.2.3.3 - Random Forest Evaluation

As shown in Figure 55, the random forest model with higher number of trees has higher AUC than the random forest model with lower number of trees. The random forest model with gini impurity function has lower AUC than the random forest model with entropy impurity function. The AUC evaluation is only a simple evaluation for the model, I will do more evaluation in the next section.

7.3 Search for Patterns

Algorithm	**Model 1**	**Model 2**	**Model 3**
Logistic Regression	72.24%	71.48%	72.24%
C5.0 DS	72.87%	72.87%	
Random Forest	71.97%	72.36%	72.19%

Table 3: Accuracy of Algorithms Comparison

By the given table, I can see that the performance of three algorithms does not perform quite different. Even the decision tree has the highest AUC, the logistic regression model also perform very well. It is very good result because logistic regression is good for explain the relation between features and response variable. Even the decision tree has the highest AUC, it takes more time to train and does not perform much better than Logistic Regression. Therefore, I will focus on **Logistic**

Regression model in the rest of research since it gives me good accuracy and provides good explanation for relation between features and response variable. I will also use C5.0 and Random Forest to find the important features.

8 Interpretation

8.1 Study and discuss the mined patterns

In this research, I have collected a qualified dataset. Then I choose the useful features and remove the irrelevant features. After selecting useful features, I clean the data, remove outliers and extreme values. After that, I regroup and balance the unbalanced categories and remove the old categories. Therefore, I can make sure that the dataset used to build model and mine patterns is qualified and prepared.

Since our mining goal is a classification, I use four classification algorithms the same dataset: **Logistic Regression**, **C5.0 Decision Tree**, and **Random Forest** to mine the pattern. Then I can compare the pattern mined by these algorithms. As discussed previously, the **Logistic Regression** has the second highest accuracy and good at explaining the relation between features and response variable, therefore, I will focus on it at section 8.3. The **C5.0 Decision Tree** and **Random Forest** model will be discarded because it take long time to train and does not provide a good enough accuracy.

I should discuss whether the mined patterns are meaningful, and whether the result is acceptable. If the mined patterns are not meaningful or the result is not acceptable, I should find the reason and optimize the model or discard the model. If the mined patterns are meaningful and the result is acceptable, I should find the explanation for the model and show the details.

8.2 Visualize the data, results, models, and patterns

8.2.1 - Visualize Data Distribution

```
In [2]: cardio_data.describe()
```

Out[2]:

	age	height	weight	ap_hi	ap_lo
count	68764.000000	68764.000000	68764.000000	68764.000000	68764.000000
mean	19464.414199	164.361177	74.120405	126.601652	81.377320
std	2468.214027	8.185037	14.330192	16.720722	9.667527
min	10798.000000	55.000000	11.000000	60.000000	40.000000
25%	17657.000000	159.000000	65.000000	120.000000	80.000000
50%	19701.000000	165.000000	72.000000	120.000000	80.000000
75%	21324.000000	170.000000	82.000000	140.000000	90.000000
max	23713.000000	250.000000	200.000000	220.000000	190.000000

Figure 56: Numeric Features After Preprocessing

```
In [4]: plt.subplot(221)
        plt.hist(cardio_data.gender)
        plt.title('Gender Distribution')
        plt.subplot(222)
        plt.hist(cardio_data.new_cholesterol)
        plt.title('New Cholesterol Distribution')
        plt.subplot(223)
        plt.hist(cardio_data.new_gluc)
        plt.title('New Gluc Distribution')

Out[4]: Text(0.5, 1.0, 'New Gluc Distribution')
```

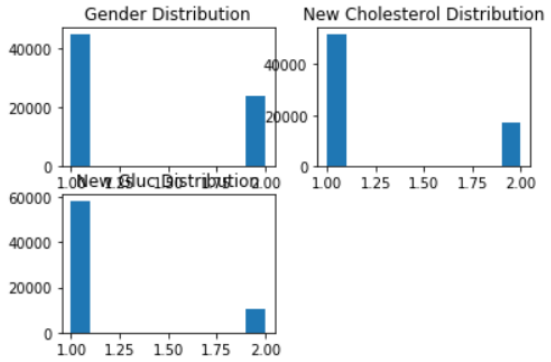


Figure 57: Categorical Features After Preprocessing

In Figure 56, 57, I can see that after preprocessing, the missing value is removed. Outliers and extreme values of numeric features are removed. Imbalanced categorical features are reclassified and much better than before. Therefore, I can conclude that our model is representative.

8.2.2 - Visualize Logistic Regression Model Result

In this section, I will visualize the Logistic Regression model with regularization parameter 0. Figure 58 shows the coefficients of each feature and which can also used to represent the feature importance.

```
Logistic_model = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
clf = Logistic_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.coefficients
```

```
['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active', 'cardio', 'new_cholesterol', 'new_gluc', 'genderIndex',
'genderVec', 'cholesterolIndex', 'cholesterolVec', 'glucIndex', 'glucVec', 'features']

DenseVector([0.0001, 0.0329, -0.0038, 0.0113, 0.0536, 0.0171, -0.1415, -0.2434, -0.2236, -0.6206, 0.0651])
```

Figure 58: Summary of Logistic Regression Model

8.2.3 - Visualize C5.0 Decision Tree Model Result

In this section, I will visualize the Decision Tree Model with entropy criterion function. As shown in Figure 59, it is the features importance of decision tree.

```
decision_tree_model = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
clf = decision_tree_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.featureImportances
```

```
['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active', 'cardio', 'new_cholesterol', 'new_gluc', 'genderIndex',
'genderVec', 'cholesterolIndex', 'cholesterolVec', 'glucIndex', 'glucVec', 'features']

SparseVector(11, {0: 0.1225, 1: 0.0003, 2: 0.0006, 3: 0.0059, 4: 0.8116, 5: 0.0041, 6: 0.002, 8: 0.0016, 9: 0.0503, 10: 0.0011})
```

Figure 59: Summary of Decision Model Model

8.2.4 - Visualize Random Forest Result

Since the Random Forest generates lots of decision trees I cannot directly get a viewer for the decision tree. Figure 60 shows that importance of each feature in random forest.

```
random_forest_model = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
clf = random_forest_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.featureImportances
```

```
['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active', 'cardio', 'new_cholesterol', 'new_gluc', 'genderIndex',
'genderVec', 'cholesterolIndex', 'cholesterolVec', 'glucIndex', 'glucVec', 'features']

SparseVector(11, {0: 0.1108, 1: 0.0002, 2: 0.0022, 3: 0.0179, 4: 0.5545, 5: 0.2338, 6: 0.0013, 7: 0.0005, 8: 0.0011, 9: 0.0743, 10: 0.0035})
```

Figure 60: Summary of Random Forest Model

8.3 Interpret the results, models, and patterns

In this section, I will only interpret **Logistic Regression** model. I won't interpret **Random Forest** because the random forest based on the decision tree. The random forest finds the average results of **n** decision trees where n is predefined hyperparameter. I won't interpret **Decision Tree** since it does not perform very well either.

8.3.1 - Interpretation for Logistic Regression

```
Logistic_model = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
clf = Logistic_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.coefficients
```

```
['age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'smoke', 'alco', 'active', 'cardio', 'new_cholesterol', 'new_gluc', 'genderIndex',
'genderVec', 'cholesterolIndex', 'cholesterolVec', 'glucIndex', 'glucVec', 'features']
```

```
DenseVector([0.0001, 0.0329, -0.0038, 0.0113, 0.0536, 0.0171, -0.1415, -0.2434, -0.2236, -0.6206, 0.0651])
```

Figure 61: Summary of Logistic Regression Model

Figure 61 shows the summary of how each features affect the probability of getting cardiovascular disease, I will discuss them in detail. The most important thing should be mentioned before all interpretation is that the logistic regression does not predict the probability directly, instead, it predicts odds. **odds = p / (1 - p)**. Therefore, in the rest of interpretation, all the changes caused by features will affect on odds but not probability. However, it is easily to get probability back by inverse the formula.

8.3.1.1 - Age

From the summary table and Figure 62, I can see that as the age increase, the odds of getting cardiovascular is also increase. The summary shows that for each day increase, the odd of getting cardiovascular disease will increase by 0.0001.

```
In [9]: plt.scatter(cardio_data.age, cardio_data.cardio)
Out[9]: <matplotlib.collections.PathCollection at 0x1f5491ef748>
```

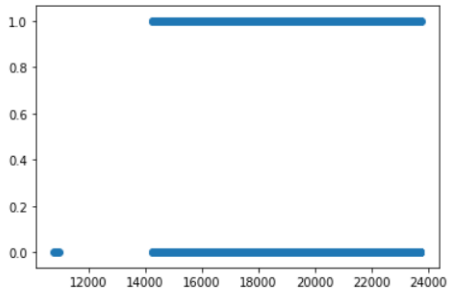


Figure 62: Age vs Cardio

8.3.1.2 - gender

From the summary table and Figure 63, I can see that female (gender 1) has higher odds to get cardiovascular disease than male(gender 2). It is a interesting finding, we should do some further research on that. The female has 0.0329 odds higher than male.

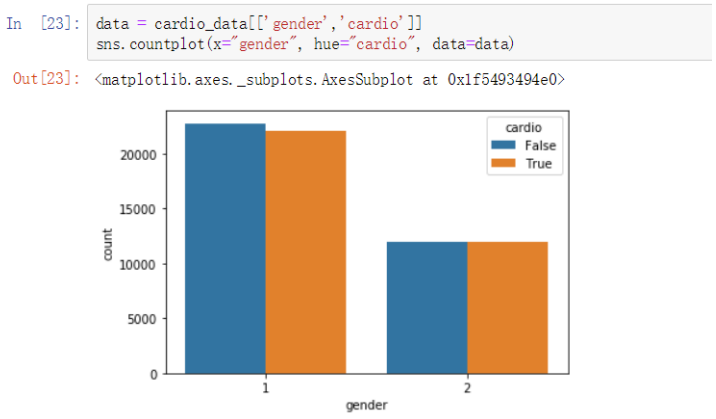


Figure 63: Gender vs Cardio

8.3.1.3 - height

From the summary table and Figure 64, I can see that higher people has lower odds to get cardiovascular disease than lower people. For each 1 cm increase in people height, the odds to get cardiovascular disease will decrease by 0.0038. That's an interesting finding, we should do some further research on that.

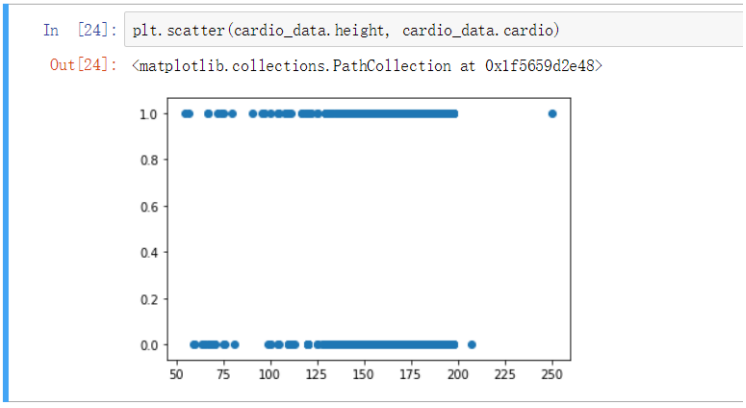


Figure 64: Height vs Cardio

8.3.1.4 - weight

From the summary table and Figure 65, I can see that as the weight increase, the odds of getting cardiovascular disease will also increase. For each killogram increased in weights, the odds of getting cardiovascular disease will increase by 0.0113.

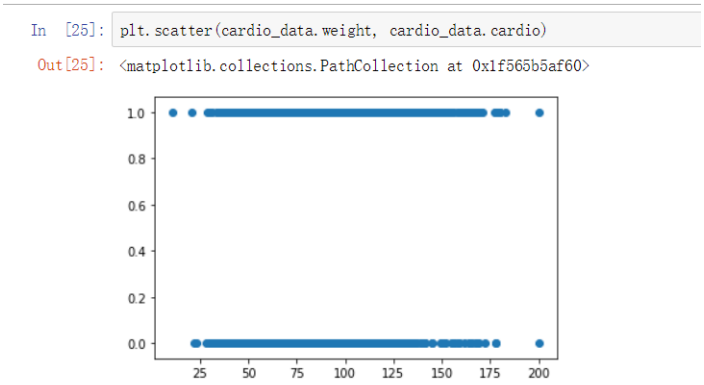


Figure 65: Weight vs Cardio

8.3.1.5 - ap_hi

From the summary table and Figure 66, I can see that as the systolic blood pressure increase, the odds of getting cardiovascular disease will also increase. For each unit increased in the systolic blood pressure, the odds of getting cardiovascular disease increase by 0.0536.

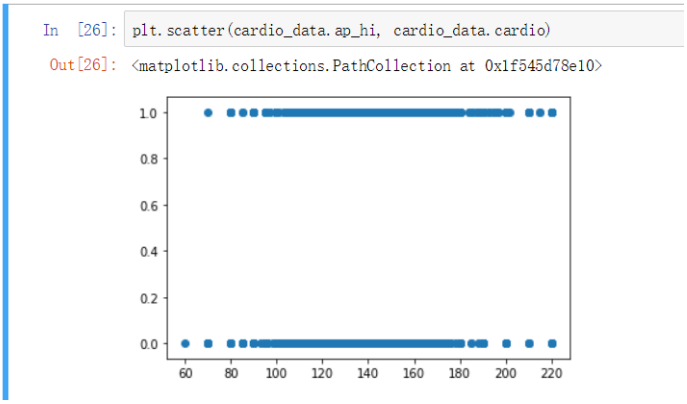


Figure 66: ap_hi vs Cardio

8.3.1.6 - ap_lo

From the summary and Figure 67, I can see that as the diastolic blood pressure increase, the odds of getting cardiovascular disease will also increase. For each unit increased in the diastolic blood pressure increase, the odds of getting cardiovascular disease increase by 0.0171.

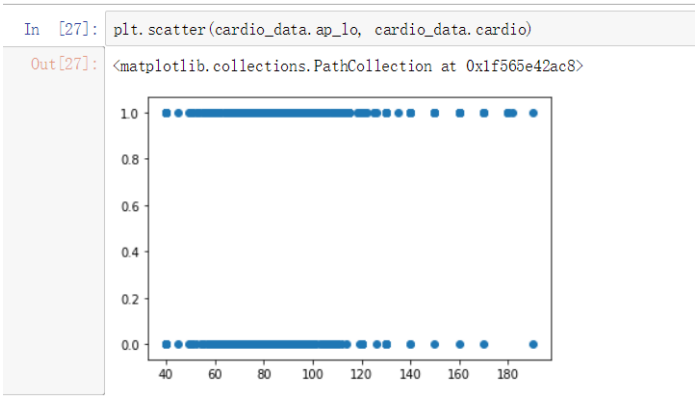


Figure 67: ap_lo vs Cardio

8.3.1.7 - smoke

From the summary and Figure 68, I can see that the people who does not smoke has lower odds of getting cardiovascular disease than people who smoke. The people who does not smoke has 0.1415 lower odds of getting cardiovascular disease than people who smoke.

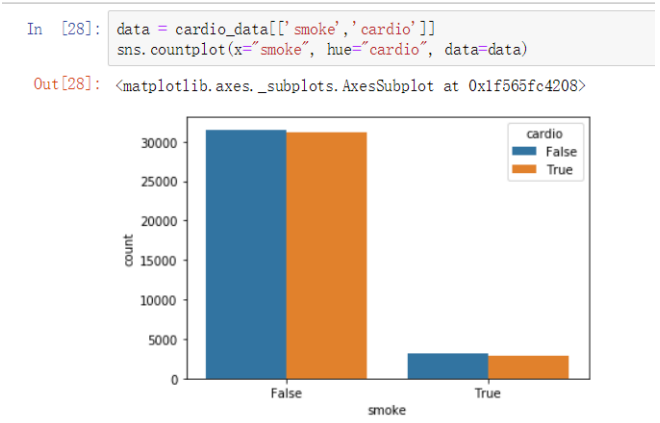


Figure 68: smoke vs Cardio

8.3.1.8 - alco

From the summary, I can see that the people who does not drink alcohol has lower odds of getting cardiovascular disease than people who drink alcohol. The people who does not drink alcohol has 0.2434 lower odds of getting cardiovascular disease than people who drink alcohol.

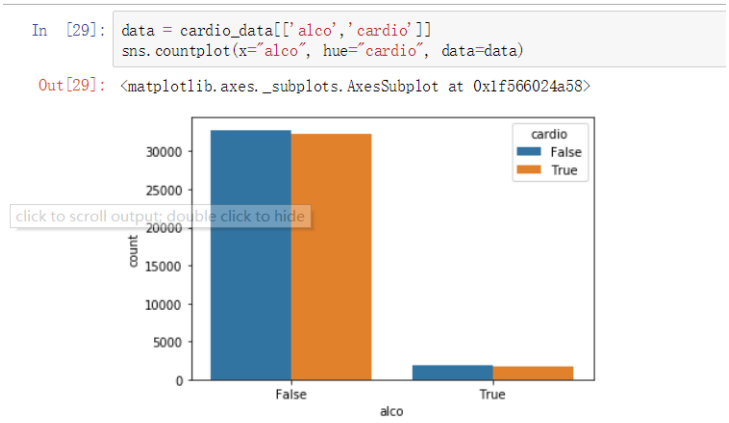


Figure 69: alco vs Cardio

8.3.1.9 - active

From the summary, I can see that the people who does activity has lower odds of getting cardiovascular disease than people who does not do activity. The people who does activity has 0.2236 lower odds of getting cardiovascular disease than people who does not do activity.

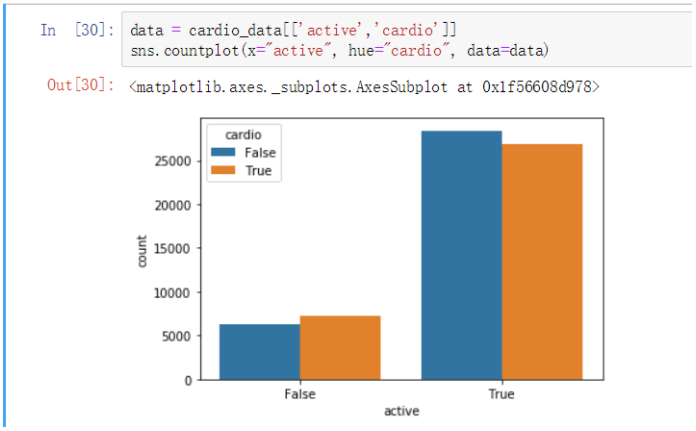


Figure 70: active vs Cardio

8.3.1.10 - new_cholesterol

From the summary, I can see that the people with normal cholesterol has lower odds of getting cardiovascular disease than the person with abnormal(high) cholesterol. The person with normal cholesterol has 0.6206 odds of getting cardiovascular disease lower than the person with abnormal(high) cholesterol.

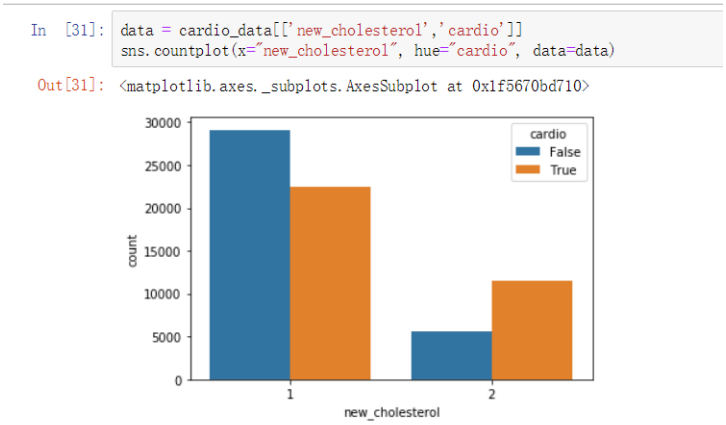


Figure 71: new_cholesterol vs Cardio

8.3.1.11 - new_gluc

From the summary, I can see that the person with normal glucose has higher odds of getting cardiovascular disease than the person with abnormal(high) glucose. That's another unusual finding, it may need further research. The person with normal glucose has 0.0651 higher odds of getting cardiovascular disease than the person with abnormal(high) glucose.

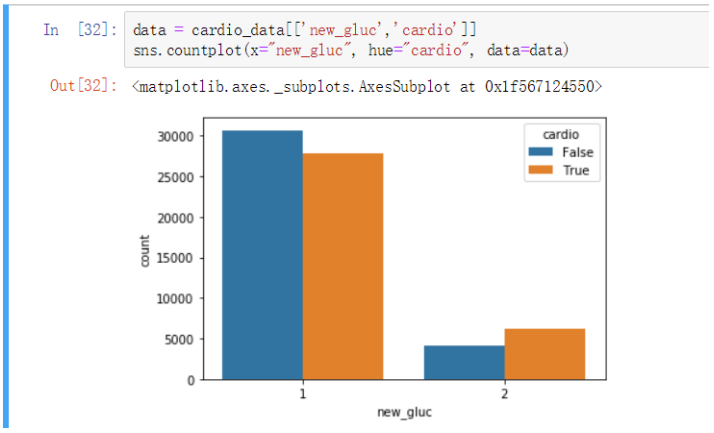


Figure 72: new_gluc vs Cardio

8.4 Assess and evaluate results, models, and patterns

8.4.1 - Assessment for Explanation

From the above discussion, we can see that the increase in **age**, **weight**, **ap_hi**, and **ap_lo** will increase the probability of getting cardiovascular disease. We also find that the person with lower cholesterol will have lower probability of getting cardiovascular disease and the smoker has higher probability of getting cardiovascular disease. They are in line with common sense. We can conclude that these relations explanation are reliable.

However, we find some uncommon things: the female has higher probability of getting cardiovascular disease than male. The person with physical activity has higher probability of getting cardiovascular disease than the person without physical activity and the person with lower glucose has higher probability of getting cardiovascular disease than the person with higher glucose. All of them need further research by the professional scientist.

8.4.2 - Assessment for Prediction

For prediction, the **Logistic Regression** algorithm gets 72.24% prediction accuracy which is quite high. However, only has the accuracy is not enough. Since my data mining goal is classification, I should also a calculate precision, recall, and f1 score.

In pattern recognition, information retrieval and binary classification, precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances. Both

precision and recall are therefore based on an understanding and measure of relevance.

F1 score (also F-score or F-measure) is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct positive results divided by the number of all positive results returned by the classifier, and r is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive). The F1 score is the harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 73: Introduction to TP,TN,FP,FN

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

Figure 74: Calculation for Precision

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

Figure 75: Calculation for Recall

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

Figure 76: Calculation for F1 Score

Important: Since the pyspark only has limited number of evaluation functions, I only use the pyspark to evaluate the AUC. After that, I retrieve the prediction results from model and convert them into numpy array, then I use sklearn to evaluate it.

In Figure 77, it shows that the precision is 0.749, recall is 0.68 and f1 score is 0.712. It is not very high but good enough. Therefore, we can use this model to predict whether a person will get cardiovascular disease by provided features.

```
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np

Logistic_model = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
clf = Logistic_model.fit(train_data)
pred = clf.transform(test_data)

prediction = np.array(pred.select('prediction').collect())
label = np.array(pred.select('cardio').collect())

print("Precision: " + str(precision_score(label, prediction)))
print("Recall: " + str(recall_score(label, prediction)))
print("F1 Score: " + str(f1_score(label, prediction)))
```

```
Precision: 0.7492990268843807
Recall: 0.6788702928870293
F1 Score: 0.7123480987847903
```

Figure 77: Evaluation Results

8.4.3 - Assessment for Feature Importance

ap_hi, **age**, **cholesterol**, and **ap_lo** are most important features in both logistic regression model and random forest model which means we should focus on the study on how to reduce the risk of getting cardiovascular disease by take care of these features. Obviously, blood pressure is the key of keys.

8.5 Iterate prior steps(1 - 7) as required

8.5.1 Iteration for Step 1 - Business Understanding

In this step, we are aiming to find a interesting topic and the background related to it. After some research, I find that I'm interested in health area. In this step, I also construct the business goal and data mining goal in this step. One important thing should be noticed is that I should connect the data mining goal to business goal in this step.

8.5.2 Iteration for Step 2 - Data Understanding

Important: Since the pyspark dataframe is hard to manipulate, we choose pandas to describe and visualize our data, then convert it back to the pyspark dataframe and build the pyspark model. The pyspark data frame and pandas data frame can easily convert to each other by the following code:

```
[1]: import findspark
     import pandas as pd
     findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
     import pyspark
     from pyspark.sql import *
     spark = SparkSession.builder.appName('dataframe_transformation').getOrCreate()

[4]: # Read Data into PySprak Data Frame
     original_df_in_pyspark = spark.read.csv('cardio_train_new.csv')

[6]: # Convert PySpark Dataframe to Pandas Dataframe
     df_in_pandas = original_df_in_pyspark.toPandas()

[7]: # Convert Pandas Dataframe to PySpark Dataframe
     df_in_pyspark = spark.createDataFrame(df_in_pandas)
```

Figure 78: DataFrame Transformation

In this step, I collect the qualified data from Kaggle which is a open source platform for machine learning and data mining. After finding the data set, I checked the format for each feature in the dataset. Then I explore the data features and find that some of the data may not useful and may be removed for further research. I also verified the data quality and find that there is not missing value for this dataset. However, there are outliers and extreme values which need be processed in the later research.

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [ ]: cardio_data = pd.read_csv('cardio_train_new.csv')
        cardio_data

In [ ]: cardio_data.dtypes

In [ ]: plt.hist(cardio_data.cardio)

In [ ]: print("age")
        print("Count: " + str(len(cardio_data.age)))
        print("Min: " + str(np.nanmin(cardio_data.age)))
        print("Max: " + str(np.nanmax(cardio_data.age)))
        print("Range: " + str(np.nanmax(cardio_data.age) - np.nanmin(cardio_data.age)))
        print("Variance: " + str(np.nanvar(cardio_data.age)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.age)))

In [ ]: print("height")
        print("Count: " + str(len(cardio_data.height)))
        print("Min: " + str(np.nanmin(cardio_data.height)))
        print("Max: " + str(np.nanmax(cardio_data.height)))
        print("Range: " + str(np.nanmax(cardio_data.height) - np.nanmin(cardio_data.height)))
        print("Variance: " + str(np.nanvar(cardio_data.height)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.height)))
```

Figure 79: Expore Data - 1

```
In [ ]: print("weight")
        print("Count: " + str(len(cardio_data.weight)))
        print("Min: " + str(np.nanmin(cardio_data.weight)))
        print("Max: " + str(np.nanmax(cardio_data.weight)))
        print("Range: " + str(np.nanmax(cardio_data.weight) - np.nanmin(cardio_data.weight)))
        print("Variance: " + str(np.nanvar(cardio_data.weight)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.weight)))

In [ ]: print("ap_hi")
        print("Count: " + str(len(cardio_data.ap_hi)))
        print("Min: " + str(np.nanmin(cardio_data.ap_hi)))
        print("Max: " + str(np.nanmax(cardio_data.ap_hi)))
        print("Range: " + str(np.nanmax(cardio_data.ap_hi) - np.nanmin(cardio_data.ap_hi)))
        print("Variance: " + str(np.nanvar(cardio_data.ap_hi)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.ap_hi)))

In [ ]: print("ap_lo")
        print("Count: " + str(len(cardio_data.ap_lo)))
        print("Min: " + str(np.nanmin(cardio_data.ap_lo)))
        print("Max: " + str(np.nanmax(cardio_data.ap_lo)))
        print("Range: " + str(np.nanmax(cardio_data.ap_lo) - np.nanmin(cardio_data.ap_lo)))
        print("Variance: " + str(np.nanvar(cardio_data.ap_lo)))
        print("Standard Deviation: " + str(np.nanstd(cardio_data.ap_lo)))

In [ ]: plt.hist(cardio_data.smoke)

In [ ]: plt.hist(cardio_data.alco)

In [ ]: plt.hist(cardio_data.cholesterol)
```

Figure 80: Expore Data - 2

```
In [ ]: plt.hist(cardio_data.gluc)
In [ ]: plt.hist(cardio_data.active)
In [ ]: plt.hist(cardio_data.gender)
In [ ]: cardio_data.describe()
In [ ]: cardio_data.info()
In [ ]: cardio_data.isnull()
```

Figure 81: Expore Data - 3

8.5.3 Iteration for Step 3 - Data Preparation

In this step,I integrate two files into one and remove the **id** feature since this is a unique identifier and irrelevant to the response variable. Then I remove the data with outliers or extreme values since these data does not make any sense(the blood pressure of human won't exceed 220 or below 40). I didn't construct any new features since I'm not professional scientist in medical area. I also drop the missing value. Then I change the data type to the correct type firstly since the initial data types in the dataset are all continuous.

```
In [ ]: file_1_path = "cardio_train_1.csv"
file_2_path = "cardio_train_2.csv"

data_list = []
with open('cardio_train_1.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    next(csv_reader)
    for row in csv_reader:
        temp_row = row[0].split(';')
        data_list.append(temp_row)
with open('cardio_train_2.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    for row in csv_reader:
        temp_row = row[0].split(';')
        data_list.append(temp_row)

with open ('cardio_train_new.csv', mode='w', newline="") as csv_file:
    csv_writer = csv.writer(csv_file)
    csv_writer.writerow(['id', 'age', 'gender', 'height', 'weight', 'ap_hi', 'ap_lo', 'cholesterol', 'gluc', 'smoke', 'alco', 'active', 'cardi
    for row in data_list:
        id = row[0]
        age = row[1]
        gender = row[2]
        height = row[3]
        weight = row[4]
        ap_hi = row[5]
        ap_lo = row[6]
        cholesterol = row[7]
        gluc = row[8]
        smoke = row[9]
        alco = row[10]
        active = row[11]
        cardio = row[12]
        csv_writer.writerow([id, age, gender, height, weight, ap_hi, ap_lo, cholesterol, gluc, smoke, alco, active, cardio])
```

Figure 82: File Integration

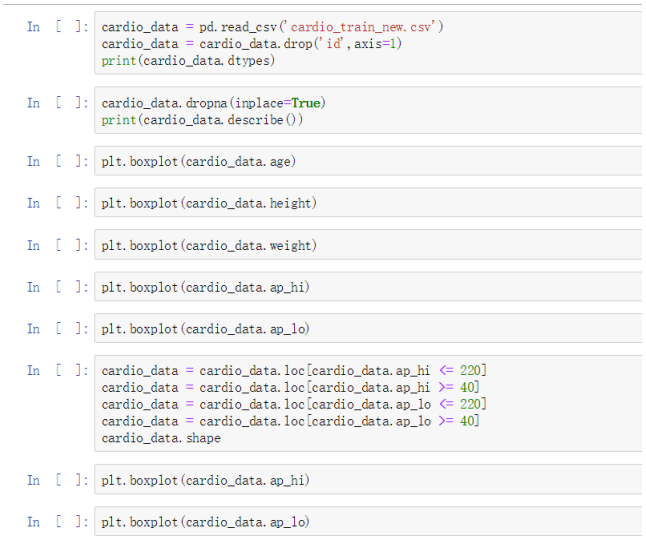


Figure 83: Remove id, Identify and Remove Outliers and Extreme Values

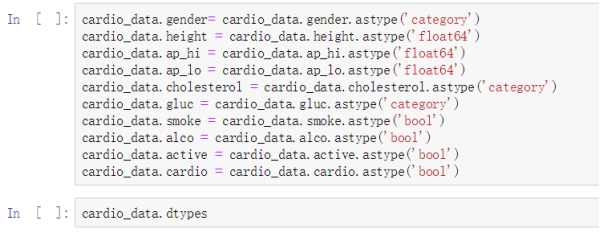


Figure 84: Modify Data Types

8.5.4 Iteration for Step 4 - Data Transformation

In this step, I regroup **cholesterol** and **gluc** features because they have three groups but except the group 1(normal group), other groups have very low proportion. So I regroup group 2 and group 3 to a new group - abnormal group. I didn't project any data since for this research, the **logistic regression** will auto log the response variable and the **decision tree** does not any projection of the data. I also use **feature selection** and **correlation matrix** to evaluate the feature importance. Since I only have 13 features in this dataset, I decide not remove any features.

```
In [ ]: cholesterol = cardio_data.cholesterol
        new_cholesterol = []
        for c in cholesterol:
            if c == 1:
                new_cholesterol.append(1)
            else:
                new_cholesterol.append(2)
        cardio_data['new_cholesterol'] = new_cholesterol
        cardio_data.new_cholesterol = cardio_data.new_cholesterol.astype('category')
        gluc = cardio_data.gluc
        new_gluc = []
        for g in gluc:
            if g == 1:
                new_gluc.append(1)
            else:
                new_gluc.append(2)
        cardio_data['new_gluc'] = new_gluc
        cardio_data.new_gluc = cardio_data.new_gluc.astype('category')

In [ ]: plt.hist(cardio_data.new_cholesterol)

In [ ]: plt.hist(cardio_data.new_gluc)
```

Figure 85: Reclassify Category

```
In [ ]: from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import f_classif
        bestfeatures = SelectKBest(score_func=f_classif , k=10)
        X = cardio_data.drop('cardio', axis=1)
        Y = cardio_data.cardio
        bestfeatures.fit(X, Y)
        scores = pd.DataFrame(bestfeatures.scores_)
        columns = pd.DataFrame(X.columns)
        feature_scores = pd.concat([columns, scores], axis=1)
        print(feature_scores)

In [ ]: import seaborn as sns
        plt.figure(figsize=(14,14))
        sns.heatmap(cardio_data.corr(),
                    vmin=-1,
                    cmap='coolwarm',
                    annot=True);
```

Figure 86: Reclassify Category

8.5.5 Iteration for Step 5 - Data Mining Methods Selection

In this step, I choose classification as the data mining methods of the whole research because the type of response variable is binary. Regression methods are used for continuous response variable and clustering methods are used for unlabeled response variable.

8.5.6 Iteration for Step 6 - Data Mining Algorithms Selection

In this step, I discuss the advantages and disadvantages for most of the classification algorithms provided by SPSS modeller. After discussion, I choose **Logistic Regression**, **C5.0 Decision Tree**, **Random Forest**, and **Neural Network** in this research. I use logistic Regression because it is easy to interpret the relation between features and response variable which is one of the two mining goals(**explanation**). Then I choose the C5.0 decision tree and random forest because these algorithms are

build for classification and always get a good prediction accuracy which is another mining goal (**prediction**). After choosing the algorithms, I set the algorithms' parameters based and ready to move forward. For the parameters of the algorithm, I set different parameters and compare how the parameters affect the model performance.

```
[1]: # Must be included at the beginning of each new notebook. Remember to change the app name.
     import findspark
     findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
     import pyspark
     from pyspark.sql import *
     spark = SparkSession.builder.appName('Model_Parameters_Chosen').getOrCreate()

[2]: from pyspark.ml.classification import LogisticRegression
     from pyspark.ml.classification import DecisionTreeClassifier
     from pyspark.ml.classification import RandomForestClassifier

[7]: Logistic_model_1 = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
     Logistic_model_2 = LogisticRegression(regParam=1, maxIter=100, labelCol='cardio')
     Logistic_model_3 = LogisticRegression(regParam=0, maxIter=200, labelCol='cardio')

[9]: decision_tree_model_1 = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
     decision_tree_model_2 = DecisionTreeClassifier(impurity='entropy', labelCol='cardio')

[10]: random_forest_model_1 = RandomForestClassifier(numTrees=5, impurity='gini', labelCol='cardio')
      random_forest_model_2 = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
      random_forest_model_3 = RandomForestClassifier(numTrees=5, impurity='entropy', labelCol='cardio')
```

Figure 87: Parameter Setting for Different Model

8.5.7 Iteration for Step 7

Important: Since the pyspark only has limited number of evaluation functions, I only use the pyspark to evaluate the AUC. After that, I retrieve the prediction results from model and convert them into numpy array, then I use sklearn to evaluate it.

In this step, I will actually build the model and mining the data. Firstly, I should convert the dataframe from pandas back to pyspark. Then I split the data into training set (80%) and test set (20%) by Pareto principle [19]. Then I use the training set to generate the model and use the test set to validate the model. After mining the dataset, I find that there exists relation between features and response variable by logistic regression model. I also get a good prediction accuracy from the logistic regression model. After all, I made a conclusion for the mining pattern and mining result and made a comparison between each model.

```
from pyspark.ml.feature import VectorAssembler,VectorIndexer,OneHotEncoder,StringIndexer
```

```
gender_indexer = StringIndexer(inputCol='gender',outputCol='genderIndex')
gender_encoder = OneHotEncoder(inputCol='genderIndex',outputCol='genderVec')
cholesterol_index = StringIndexer(inputCol='new_cholesterol', outputCol='cholesterolIndex')
cholesterol_encoder = OneHotEncoder(inputCol='cholesterolIndex', outputCol='cholesterolVec')
gluc_indexer = StringIndexer(inputCol='new_gluc', outputCol='glucIndex')
gluc_encoder = OneHotEncoder(inputCol='glucIndex', outputCol='glucVec')
```

```
assembler = VectorAssembler(inputCols=['age',
                                       'genderVec',
                                       'height',
                                       'weight',
                                       'ap_hi',
                                       'ap_lo',
                                       'smoke',
                                       'alco',
                                       'active',
                                       'cholesterolVec',
                                       'glucVec'
], outputCol='features')
```

Figure 88: Transform Data Frame

```
cardio_data_pyspark = gender_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gender_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = cholesterol_index.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = cholesterol_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = gluc_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gluc_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = assembler.transform(cardio_data_pyspark)
```

```
cardio_data_pyspark.columns
```

Figure 89: Transform Data Frame

```
Logistic_model_1 = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
Logistic_model_2 = LogisticRegression(regParam=1, maxIter=100, labelCol='cardio')
Logistic_model_3 = LogisticRegression(regParam=0, maxIter=200, labelCol='cardio')

l1 = Logistic_model_1.fit(train_data)
l2 = Logistic_model_2.fit(train_data)
l3 = Logistic_model_3.fit(train_data)

pred1 = l1.transform(test_data)
pred2 = l2.transform(test_data)
pred3 = l3.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Logistic Regression Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Logistic Regression Model 2: " + str(AUC2))
AUC3 = evaluator.evaluate(pred3)
print("Logistic Regression Model 3: " + str(AUC3))
```

Figure 90: Evaluation of Logistic Regression Model

```
decision_tree_model_1 = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
decision_tree_model_2 = DecisionTreeClassifier(impurity='entropy', labelCol='cardio')

d1 = decision_tree_model_1.fit(train_data)
d2 = decision_tree_model_2.fit(train_data)

pred1 = d1.transform(test_data)
pred2 = d2.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Decision Tree Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Decision Tree Model 2: " + str(AUC2))
```

Figure 91: Evaluation of Decision Tree Model

```
random_forest_model_1 = RandomForestClassifier(numTrees=5, impurity='gini', labelCol='cardio')
random_forest_model_2 = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
random_forest_model_3 = RandomForestClassifier(numTrees=5, impurity='entropy', labelCol='cardio')

clf1 = random_forest_model_1.fit(train_data)
clf2 = random_forest_model_2.fit(train_data)
clf3 = random_forest_model_3.fit(train_data)

pred1 = clf1.transform(test_data)
pred2 = clf2.transform(test_data)
pred3 = clf3.transform(test_data)

AUC1 = evaluator.evaluate(pred1)
print("Random Forest Model 1: " + str(AUC1))
AUC2 = evaluator.evaluate(pred2)
print("Random Forest Model 2: " + str(AUC2))
AUC3 = evaluator.evaluate(pred3)
print("Random Forest Model 3: " + str(AUC3))
```

Figure 92: Evaluation of Random Forest Model

8.5.8 Iteration for Step 8

In this section, I visualize the models' result and further discussed their evaluation results. At the end, I use precision, recall, f1 to evaluate the best model (Logistic Regression).

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns

pd.set_option('display.max_columns',15)
cardio_data = pd.read_csv('cardio_train_new.csv')
cardio_data = cardio_data.drop('id',axis=1)
cardio_data.dropna(inplace=True)

cardio_data = cardio_data.loc[cardio_data.ap_hi <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_hi >= 40]
cardio_data = cardio_data.loc[cardio_data.ap_lo <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_lo >= 40]

cardio_data.gender= cardio_data.gender.astype('category')
cardio_data.height = cardio_data.height.astype('float64')
cardio_data.ap_hi = cardio_data.ap_hi.astype('float64')
cardio_data.ap_lo = cardio_data.ap_lo.astype('float64')
cardio_data.cholesterol = cardio_data.cholesterol.astype('category')
cardio_data.gluc = cardio_data.gluc.astype('category')
cardio_data.smoke = cardio_data.smoke.astype('bool')
cardio_data.alco = cardio_data.alco.astype('bool')
cardio_data.active = cardio_data.active.astype('bool')
cardio_data.cardio = cardio_data.cardio.astype('bool')

cholesterol = cardio_data.cholesterol
new_cholesterol = []
for c in cholesterol:
    if c == 1:
        new_cholesterol.append(1)
    else:
        new_cholesterol.append(2)
cardio_data['new_cholesterol'] = new_cholesterol
cardio_data.new_cholesterol = cardio_data.new_cholesterol.astype('category')

gluc = cardio_data.gluc
new_gluc = []
for g in gluc:
    if g == 1:
        new_gluc.append(1)
    else:
        new_gluc.append(2)
cardio_data['new_gluc'] = new_gluc
cardio_data.new_gluc = cardio_data.new_gluc.astype('category')

cardio_data = cardio_data.drop('cholesterol', axis=1)
cardio_data = cardio_data.drop('gluc', axis=1)
```

Figure 93: Visualize and Further Evaluate Model

```
cardio_data.describe()
```

```
plt.subplot(221)
plt.hist(cardio_data.gender)
plt.title('Gender Distribution')
plt.subplot(222)
plt.hist(cardio_data.new_cholesterol)
plt.title('New Cholesterol Distribution')
plt.subplot(223)
plt.hist(cardio_data.new_gluc)
plt.title('New Gluc Distribution')
```

Figure 94: Visualize and Further Evaluate Model

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score

cardio_data = pd.read_csv('cardio_train_new.csv')
cardio_data = cardio_data.drop('id',axis=1)
cardio_data.dropna(inplace=True)

cardio_data = cardio_data.loc[cardio_data.ap_hi <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_hi >= 40]
cardio_data = cardio_data.loc[cardio_data.ap_lo <= 220]
cardio_data = cardio_data.loc[cardio_data.ap_lo >= 40]

cardio_data.gender= cardio_data.gender.astype('str')
cardio_data.height = cardio_data.height.astype('float64')
cardio_data.ap_hi = cardio_data.ap_hi.astype('float64')
cardio_data.ap_lo = cardio_data.ap_lo.astype('float64')
cardio_data.cholesterol = cardio_data.cholesterol.astype('category')
cardio_data.gluc = cardio_data.gluc.astype('category')
cardio_data.smoke = cardio_data.smoke.astype('bool')
cardio_data.alco = cardio_data.alco.astype('bool')
cardio_data.active = cardio_data.active.astype('bool')
cardio_data.cardio = cardio_data.cardio.astype('int64')

cholesterol = cardio_data.cholesterol
new_cholesterol = []
for c in cholesterol:
    if c == 1:
        new_cholesterol.append(1)
    else:
        new_cholesterol.append(2)
cardio_data['new_cholesterol'] = new_cholesterol
cardio_data.new_cholesterol = cardio_data.new_cholesterol.astype('str')

gluc = cardio_data.gluc
new_gluc = []
for g in gluc:
    if g == 1:
        new_gluc.append(1)
```

Figure 95: Visualize and Further Evaluate Model

```
    else:
        new_gluc.append(2)
cardio_data['new_gluc'] = new_gluc
cardio_data.new_gluc = cardio_data.new_gluc.astype('str')

cardio_data = cardio_data.drop('cholesterol', axis=1)
cardio_data = cardio_data.drop('gluc', axis=1)

import findspark
findspark.init('/home/ubuntu/spark-2.1.1-bin-hadoop2.7')
import pyspark
from pyspark.sql import *
spark = SparkSession.builder.appName('logistic_regression_adv').getOrCreate()

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import MultilayerPerceptronClassifier

cardio_data_pyspark = spark.createDataFrame(cardio_data)

from pyspark.ml.feature import VectorAssembler,VectorIndexer,OneHotEncoder,StringIndexer

gender_indexer = StringIndexer(inputCol='gender',outputCol='genderIndex')
gender_encoder = OneHotEncoder(inputCol='genderIndex',outputCol='genderVec')
cholesterol_index = StringIndexer(inputCol='new_cholesterol', outputCol='cholesterolIndex')
cholesterol_encoder = OneHotEncoder(inputCol='cholesterolIndex', outputCol='cholesterolVec')
gluc_indexer = StringIndexer(inputCol='new_gluc', outputCol='glucIndex')
gluc_encoder = OneHotEncoder(inputCol='glucIndex', outputCol='glucVec')

assembler = VectorAssembler(inputCols=['age',
                                       'genderVec',
                                       'height',
                                       'weight',
                                       'ap_hi',
                                       'ap_lo',
                                       'smoke',
                                       'alco',
                                       'active',
                                       'cholesterolVec',
                                       'glucVec'
], outputCol='features')

cardio_data_pyspark = gender_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gender_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = cholesterol_index.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = cholesterol_encoder.transform(cardio_data_pyspark)
```

Figure 96: Visualize and Further Evaluate Model

```
cardio_data_pyspark = gluc_indexer.fit(cardio_data_pyspark).transform(cardio_data_pyspark)
cardio_data_pyspark = gluc_encoder.transform(cardio_data_pyspark)

cardio_data_pyspark = assembler.transform(cardio_data_pyspark)

train_data, test_data = cardio_data_pyspark.randomSplit([0.8,.2])
```

```
Logistic_model = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
clf = Logistic_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.coefficients
```

```
decision_tree_model = DecisionTreeClassifier(impurity='gini', labelCol='cardio')
clf = decision_tree_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.featureImportances
```

```
random_forest_model = RandomForestClassifier(numTrees=10, impurity='gini', labelCol='cardio')
clf = random_forest_model.fit(train_data)
```

```
print(cardio_data_pyspark.columns)
clf.featureImportances
```

```
plt.scatter(cardio_data.age, cardio_data.cardio)
```

```
data = cardio_data[['gender','cardio']]
sns.countplot(x="gender", hue="cardio", data=data)
```

```
plt.scatter(cardio_data.height, cardio_data.cardio)
```

```
plt.scatter(cardio_data.weight, cardio_data.cardio)
```

```
plt.scatter(cardio_data.ap_hi, cardio_data.cardio)
```

```
plt.scatter(cardio_data.ap_lo, cardio_data.cardio)
```

```
data = cardio_data[['smoke','cardio']]
sns.countplot(x="smoke", hue="cardio", data=data)
```

Figure 97: Visualize and Further Evaluate Model

```
data = cardio_data[['alco','cardio']]
sns.countplot(x="alco", hue="cardio", data=data)
```

```
data = cardio_data[['active','cardio']]
sns.countplot(x="active", hue="cardio", data=data)
```

```
data = cardio_data[['new_cholesterol','cardio']]
sns.countplot(x="new_cholesterol", hue="cardio", data=data)
```

```
data = cardio_data[['new_gluc','cardio']]
sns.countplot(x="new_gluc", hue="cardio", data=data)
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np

Logistic_model = LogisticRegression(regParam=0, maxIter=100, labelCol='cardio')
clf = Logistic_model.fit(train_data)
pred = clf.transform(test_data)

prediction = np.array(pred.select('prediction').collect())
label = np.array(pred.select('cardio').collect())

print("Precision: " + str(precision_score(label, prediction)))
print("Recall: " + str(recall_score(label, prediction)))
print("F1 Score: " + str(f1_score(label, prediction)))
```

```
Precision: 0.7492990268843807
Recall: 0.6788702928870293
F1 Score: 0.7123480987847903
```

Figure 98: Visualize and Further Evaluate Model

9 Version Control - Github

As shown in Figure 98 and 99, since the work is done in AWS server remotely, it is better to use github control the version avoid any potential problem such as server down or file lost. The Github link for this iteration 4 is :

https://github.com/VirtueZhao/INFOSYS_722_Iteration4_BDAS

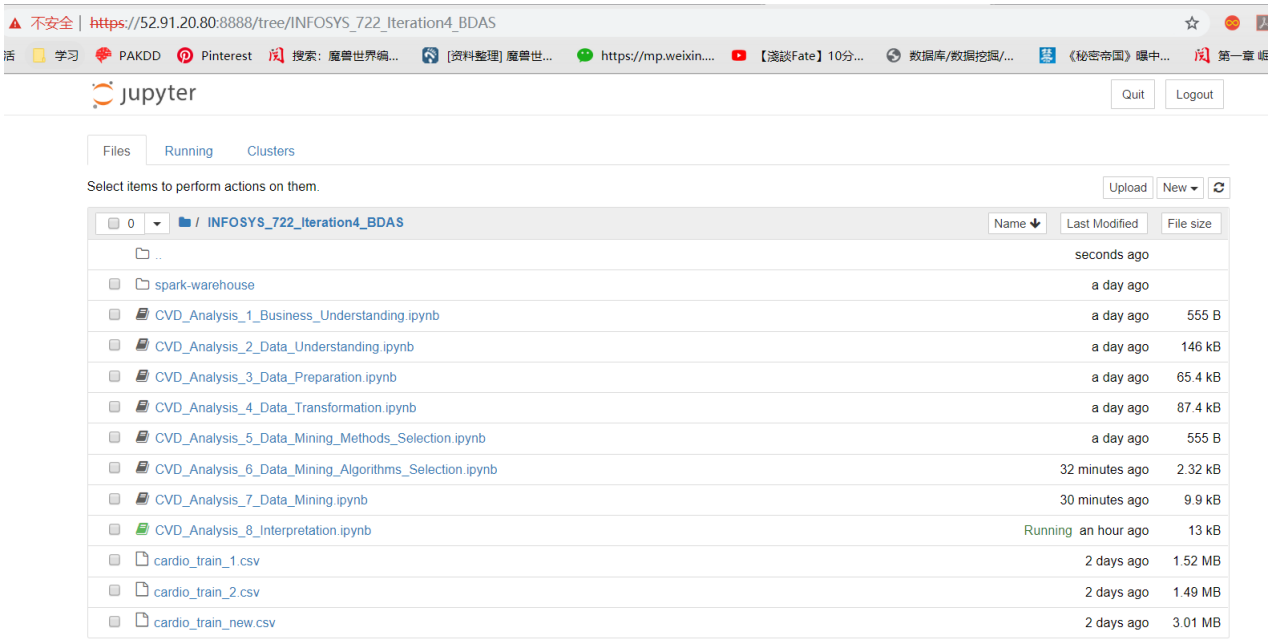


Figure 99: AWS Server

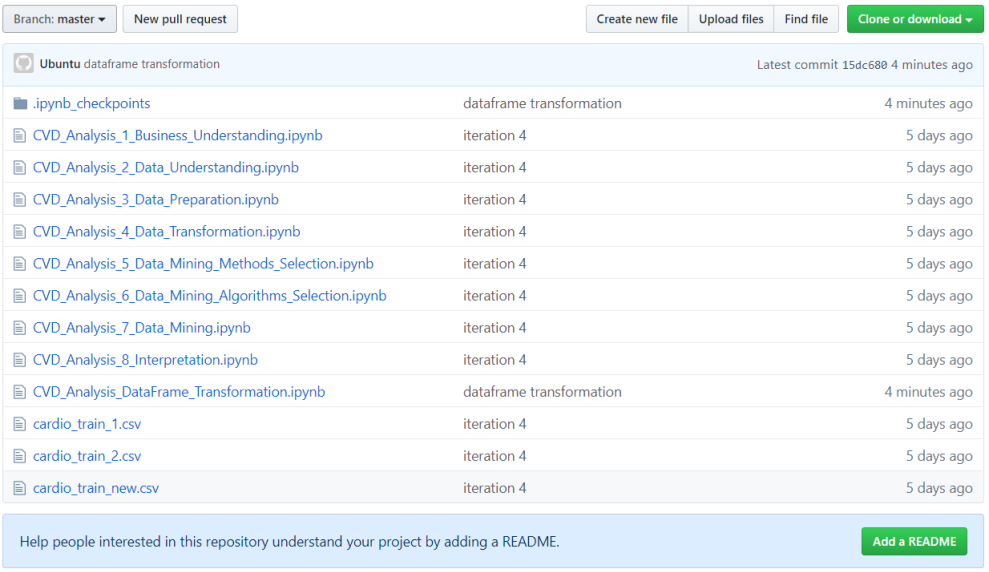


Figure 100: Github

10 Disclaimer

"I acknowledge that the submitted work is my own original work in accordance with the University of Auckland guidelines and policies on academic integrity and copyright. (See: https://www.auckland.ac.nz/en/students/forms-policies-and-guidelines/student-policies-and-guidelines/academic-integrity-copyright.html).

I also acknowledge that I have appropriate permission to use the data that I have utilised in this project. (For example, if the data belongs to an organisation and the data has not been published in the public domain then the data must be approved by the rights holder.) This includes permission to upload the data file to Canvas. The University of Auckland bears no responsibility for the student's misuse of data."

Bibliography

[1] IBM Knowledge Center. Ibm spss modeler crisp-dm guide, 2017.

[2] Valentin Fuster. Cardiovascular disease and the un millennium development goals: a serious concern, 2006.

[3] Valentin Fuster and Rajesh Vedanthan. Cardiovascular disease and the un millennium development goals: time to move forward, 2008.

[4] IBM. Bayesian model, 2017.

[5] IBM. C5.0 model, 2017.

[6] IBM. C&r tree model, 2017.

[7] IBM. Decision tree model, 2017.

[8] IBM. Generalized linear model, 2017.

[9] IBM. Handling outliers and extreme values, 2017.

[10] IBM. Linear model, 2017.

[11] IBM. Logistic model, 2017.

[12] IBM. Neural network model, 2017.

[13] IBM. Statistical model, 2017.

[14] IBM. Svm model, 2017.

[15] Vivek Kashid. Definition of different type data mining methods, 2018.

[16] Microsoft. Split dataset into training set and testing set, 2018.

[17] United Nations. Un 17 sustainable development: Health and well-being, 2019.

[18] World Health Organization. Cardiovascular diseases, 2019.

[19] Vilfredo Pareto. 1896, 2018.

[20] Peter M Rothwell. Limitations of the usual blood-pressure hypothesis and importance of variability, instability, and episodic hypertension. *The Lancet*, 375(9718):938–948, 2010.

[21] Sklearn. Decision tree model, 2019.

[22] Sklearn. Sklearn type of models, 2019.

[23] Sklearn. Svm model sklearn, 2019.