Project Report

Cardiovascular Disease Detection **Aishwarya Paruchuri**

Problem Statement:

Health issues are a major concern in the 21st century. The aim of this project is to predict whether a person has cardiovascular disease or not using features like Systolic blood pressure, Diastolic blood pressure, height, weight and 8 other features.

Data Description:

The dataset has 1000 records(including cardio-train and cardio-validation) and 13 features(including target variable) with missing values. The available features are as follows:

1. **Id| int:**
   Unique id of each individual
2. **age | int (days):**
   Age of person in days
3. **height | int (cm) :**
   Height of person in cm
4. **weight | float (kg) :**
   Weight of person in kg
5. **gender |categorical:**
   Gender as male or female
6. **ap_hi | int:**
   Systolic blood pressure of a person
7. **ap_lo | int**:
   Diastolic blood pressure of a person
8. **cholesterol | Normal, Above normal,H i g h :**
   Cholesterol of a person as Normal, Above normal and high
9. **gluc | Normal, Above normal,H i g h :**
   Glucose level as Normal, Above normal and high
10. **smoke | binary:**
    Does the person smoke or not
11. **alco | binary:**
    Does the person has alcohol intake or not
12. **active | binary:**
    Does the person indulge in some physical activity or not
13. **cardio | binary:**
    Presence or absence of cardiovascular disease

The dataset also contains missing values which will be discussed later.

Data Analysis and Visualization

Training data:

Categorical features: gender, cholesterol, gluc, smoke, alco, active
Continuous features: id, age, height, weight, ap_hi, ap_lo

The dataset has missing values, Let's see how many are there in each column.

figure 1: Null values

```
cardio.isnull().sum()
```

```
id                0
age             165
gender          171
height          302
weight          164
ap_hi           153
ap_lo           168
cholesterol     167
gluc            167
smoke           174
alco            165
active          157
cardio            0
dtype: int64
```

Table 1: Basic information about missing data

We can notice that height has the most missing values with the lowest in ap_hi.
Description of data frame to get a better understanding of mean and standard deviations.

```
cardio.describe()
```

	id	age	height	weight	ap_hi	ap_lo	smoke	alco	active	cardio
count	500.000000	335.000000	198.000000	336.000000	347.000000	332.000000	326.000000	335.000000	343.000000	500.000000
mean	50279.916000	19490.886567	163.934343	74.347321	128.685879	90.060241	0.092025	0.065672	0.813411	0.502000
std	29913.623631	2466.702487	8.258559	14.335964	18.490176	87.396945	0.289505	0.248078	0.390150	0.500497
min	38.000000	14334.000000	120.000000	45.000000	12.000000	60.000000	0.000000	0.000000	0.000000	0.000000
25%	23446.500000	17988.500000	159.250000	65.000000	120.000000	80.000000	0.000000	0.000000	1.000000	0.000000
50%	51913.500000	19719.000000	165.000000	72.000000	120.000000	80.000000	0.000000	0.000000	1.000000	1.000000
75%	78656.000000	21597.500000	168.000000	82.000000	140.000000	90.000000	0.000000	0.000000	1.000000	1.000000
max	99662.000000	23479.000000	187.000000	155.000000	190.000000	1000.000000	1.000000	1.000000	1.000000	1.000000

Table 2: Basic information about data

Oldest and youngest patient:

```
cardio.loc[cardio['age'].idxmax()]
```

```
id              64953
age                64
gender            Men
height            NaN
weight             69
ap_hi             140
ap_lo             100
cholesterol    Normal
gluc              NaN
smoke             NaN
alco                0
active            NaN
cardio              1
Name: 11, dtype: object
```

Table3.a: Oldest figure

```
cardio.loc[cardio['age'].idxmin()]
```

```
id              44000
age                39
gender            NaN
height            163
weight            NaN
ap_hi             130
ap_lo             NaN
cholesterol    Normal
gluc              NaN
smoke               0
alco                0
active            NaN
cardio              0
Name: 147, dtype: object
```

Table3.b: Youngest person

This gives us a better understanding of the range of age values.

Data Visualization:

Let us analyse different features and check how they are related to our target variable.

1. Relationship between the cardio and ages:

figure 4: Age vs Cardio kde plot

In the above figure, I've used seaborn kdeplot to understand the relationship between age and cardio. The blue shade shows density of ages who do not have cardio-vascular disease whereas red plot shows density of ages that have cardiovascular diseases.

From the plot, it is clearly visible that people with ages between 55-65 tested positive cardio-vascular disease. Also, people with 45-55 have been tested-negative for cardio-vascular disease. This gives us an understanding that as you get older, there is a higher chance you might have cardio-vascular disease.

Here is a correlation matrix to get a linear relationship between age and cardio.

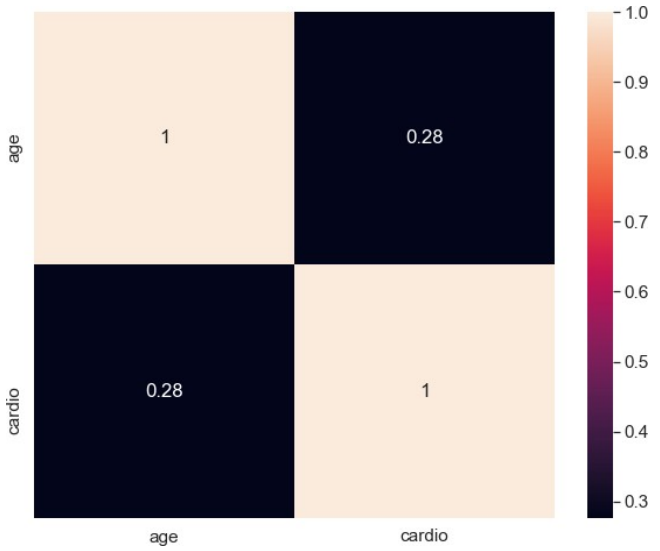


figure 5: Age vs Cardio heatmap

We can see that age and cardio have a positive relationship i.e if there are chances of people with older ages to have cardio-vascular disease.

2. Age groups with lowest rate of cardiovascular disease

a) Survival ratio (cardio=0/(cardio=0+cardio=1))

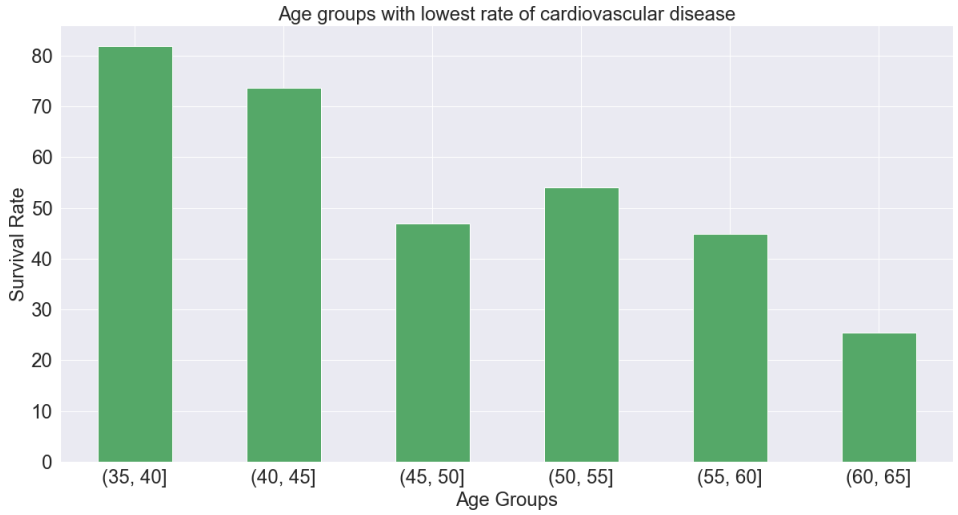
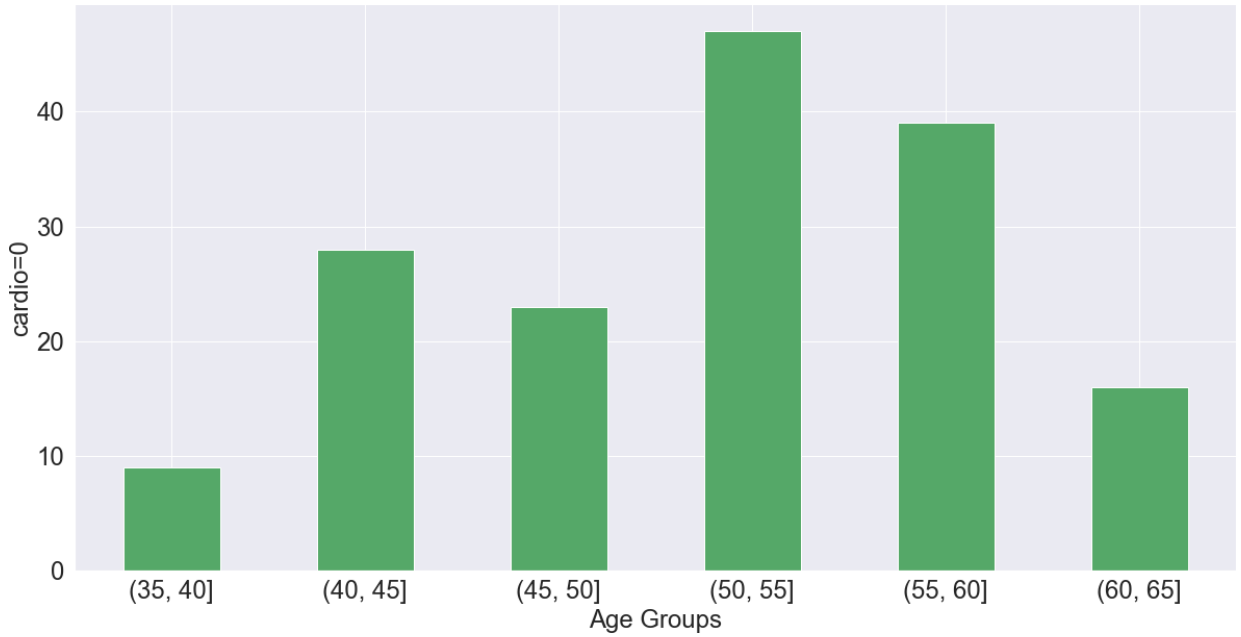


figure 6: Age group with lowest rate of cardio disease

The figure below gives us an idea about what are the ages in which people will have the highest rate of survival or lowest rate of having cardiovascular disease. We see that as the age of a person increases, the chances of not having cardiovascular disease decreases.

b) Plot just when cardio=0



3. Attributes that can have a role in prediction:

Mapping:

To get the attributes that can have a higher impact on prediction, it is important to map the categorical data into numerical categories. So let's do that.

Using pandas map method, I've mapped gender, cholesterol and gluc into the following numerical categorical values:

1. **gender**: 'Men':1,'Women':0
2. **cholesterol:**'Normal':0, 'Above Normal' :1,'High':2
3. **gluc**: 'Normal':0, 'Above Normal' : 1,'High':2

a) Systolic blood pressure vs Cardio

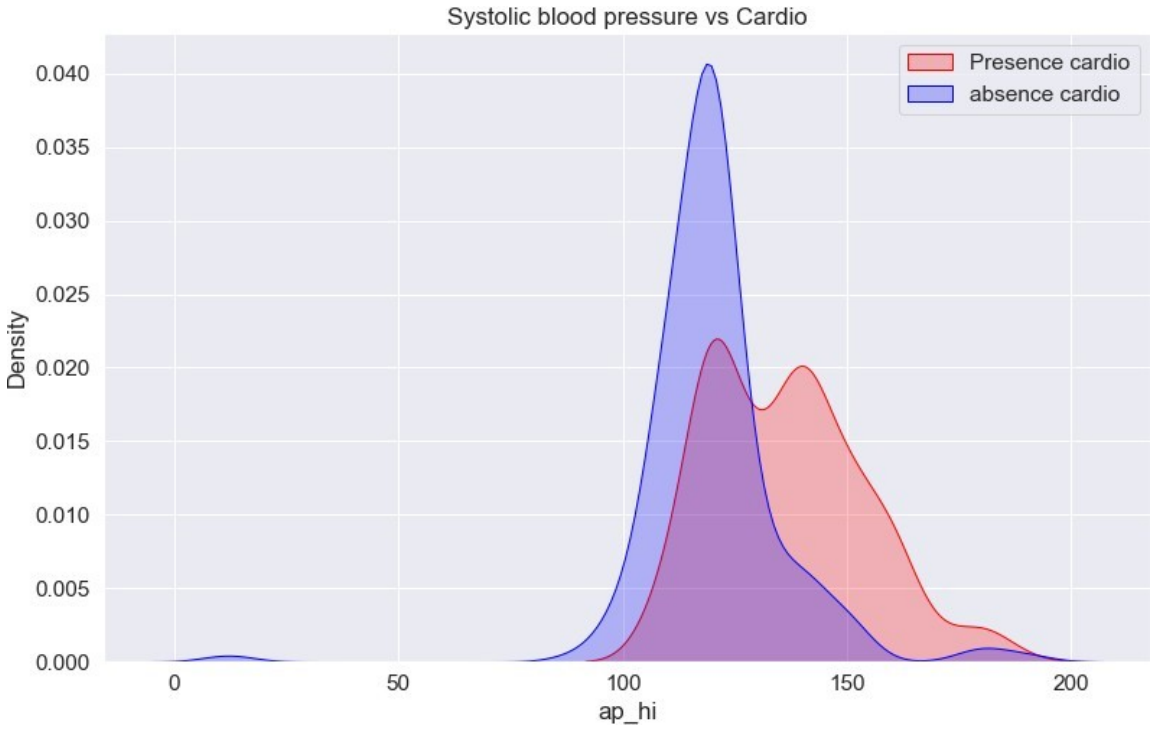


figure 7: Systolic blood pressure vs Cardio kde plot

It can be seen that some people with cardiovascular disease have a higher systolic blood pressure.

b) cholesterol vs cardio

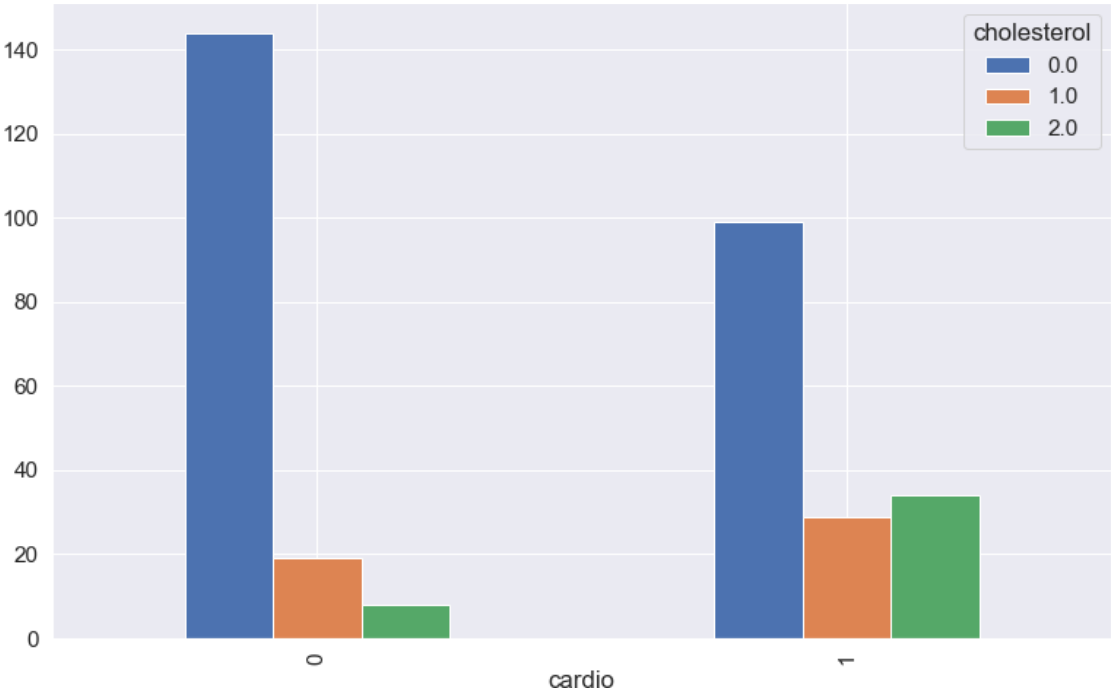


figure 8: cholesterol vs cardio bar plot

The number of people with cardiovascular disease have more above normal and high cholesterol.

c) glucose vs cardio

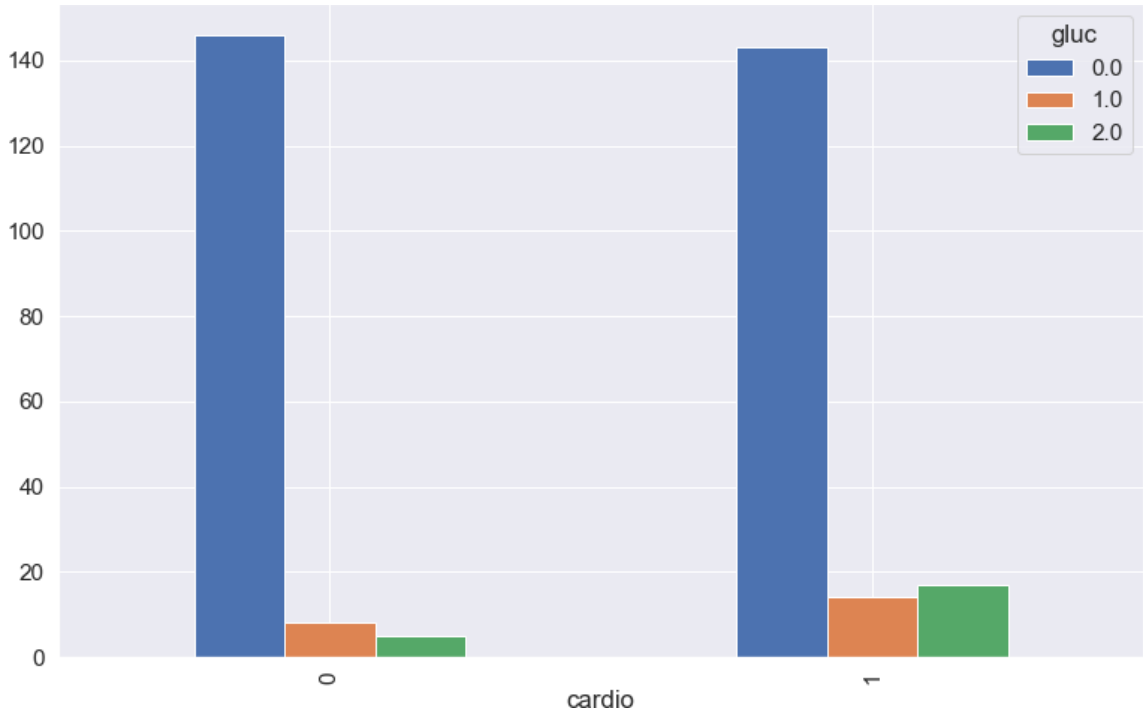


figure 9: glucose vs cardio bar plot

The number of people whose glucose level is above normal and higher is much more for positive cardiovascular people rather than those who don't have cardiovascular disease.

4. More visuals on data distributions

Heatmap:

Heatmap gives the correlation between different attributes and can help us understand if a particular feature will have an impact on the target variable. The correlation values close to 1 or -1 mean there is a correlation between two variables and correlation value close to 0 means no correlation.

Plotted using seaborn heatma

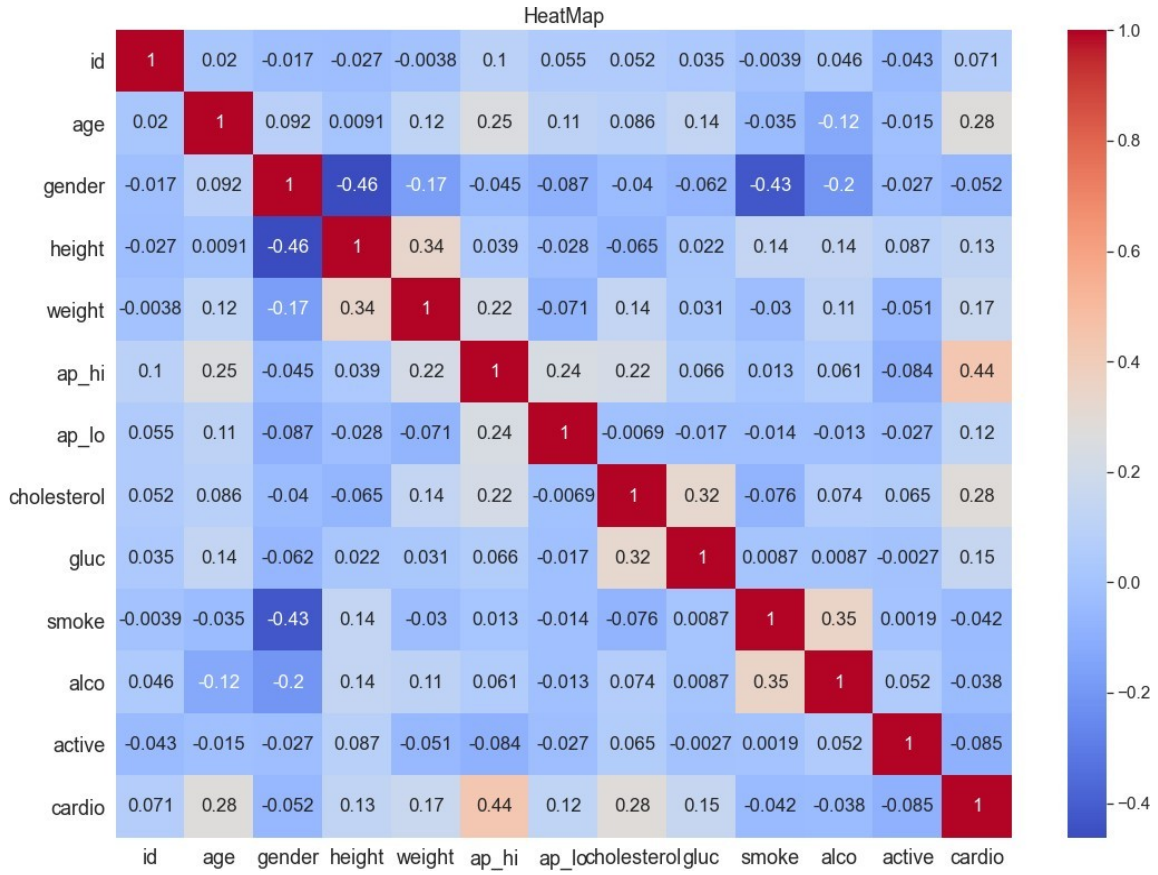


figure 10: Heatmap of training data

age, weight, ap_hi, ap_lo, cholesterol, gluc are the features that have a correlation value with cardio more than 0.1. These features can have a higher impact on prediction.

5. ScatterMatrix

Scatter matrix gives scatter plots of all the features with respect to each other. The diagonal plots give a histogram for that particular feature.

In the plot below, we can see that there isn't much change of weight with respect to age. People generally don't gain much weight as they get older.

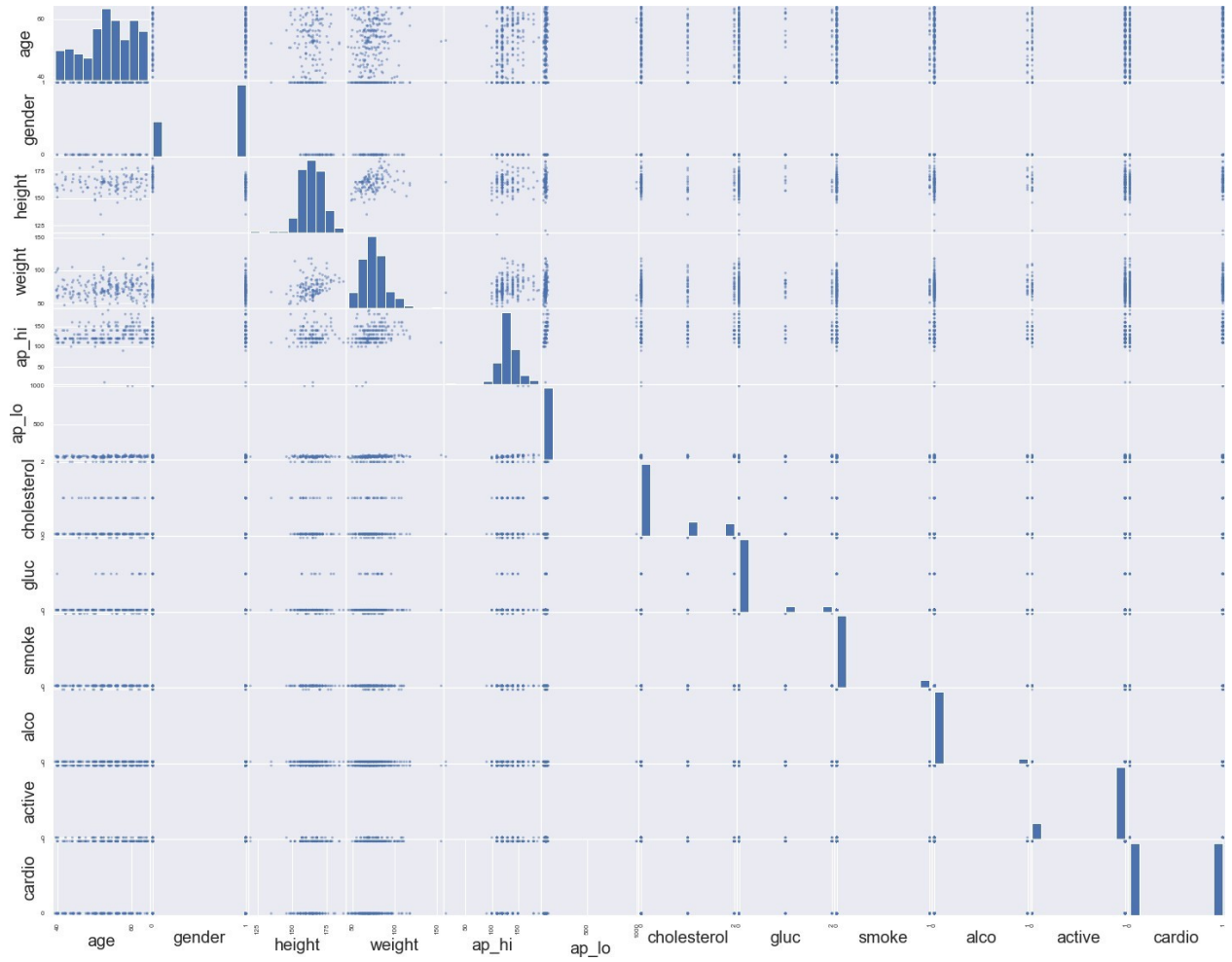


figure 11: Scatter plot of data

6. PerColumnDistribution:

Using pandas histogram, I've plotted the distribution of values for each variable. Most people in the data have heights between 160-175, we have more males in the dataset, most of them have normal glucose level and also normal cholesterol.

Histogram gives us a better understanding of the count of records for a particular value of a given feature.

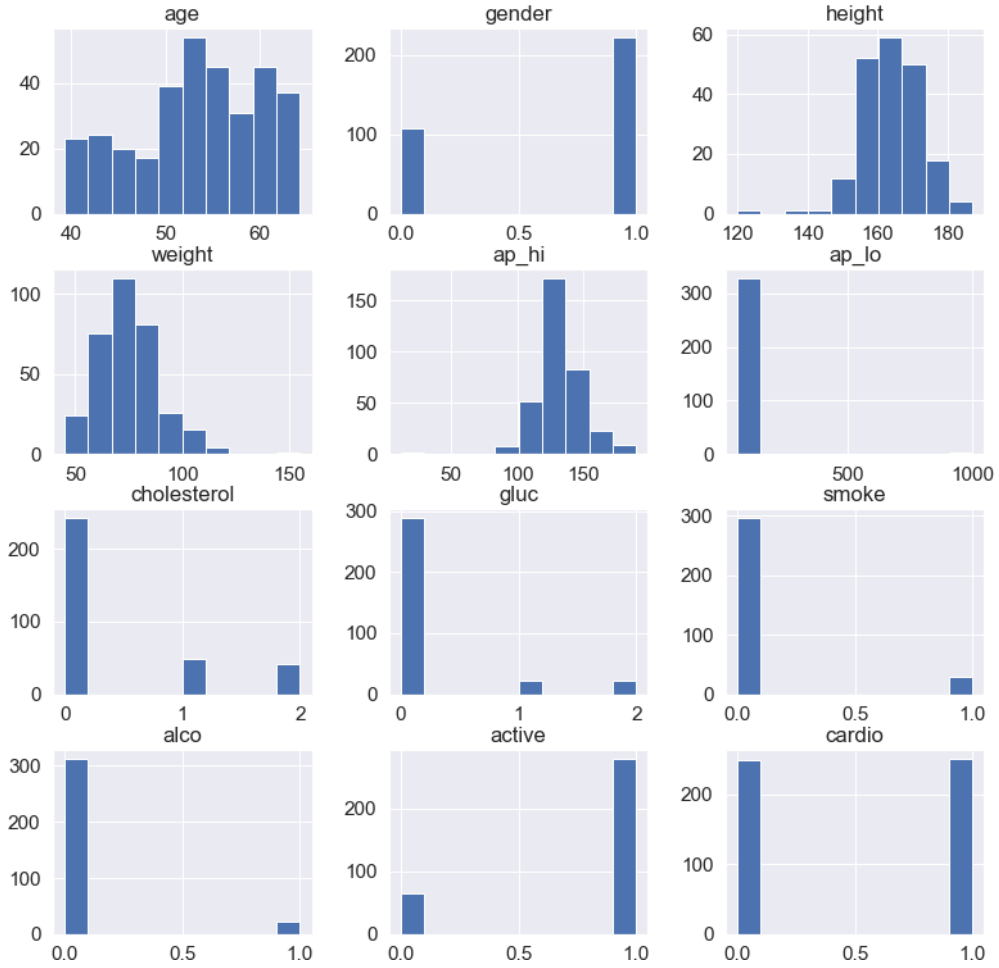


figure 12:per column distribution

Finding Missing Values

The dataset has missing values, here is the count:

```
cardio.isnull().sum()
```

```
id              0
age           165
gender        171
height        302
weight        164
ap_hi         153
ap_lo         168
cholesterol   167
gluc          167
smoke         174
alco          165
active        157
cardio          0
dtype: int64
```

figure 13:missing values in each column

Plotting a heatmap for missing values:

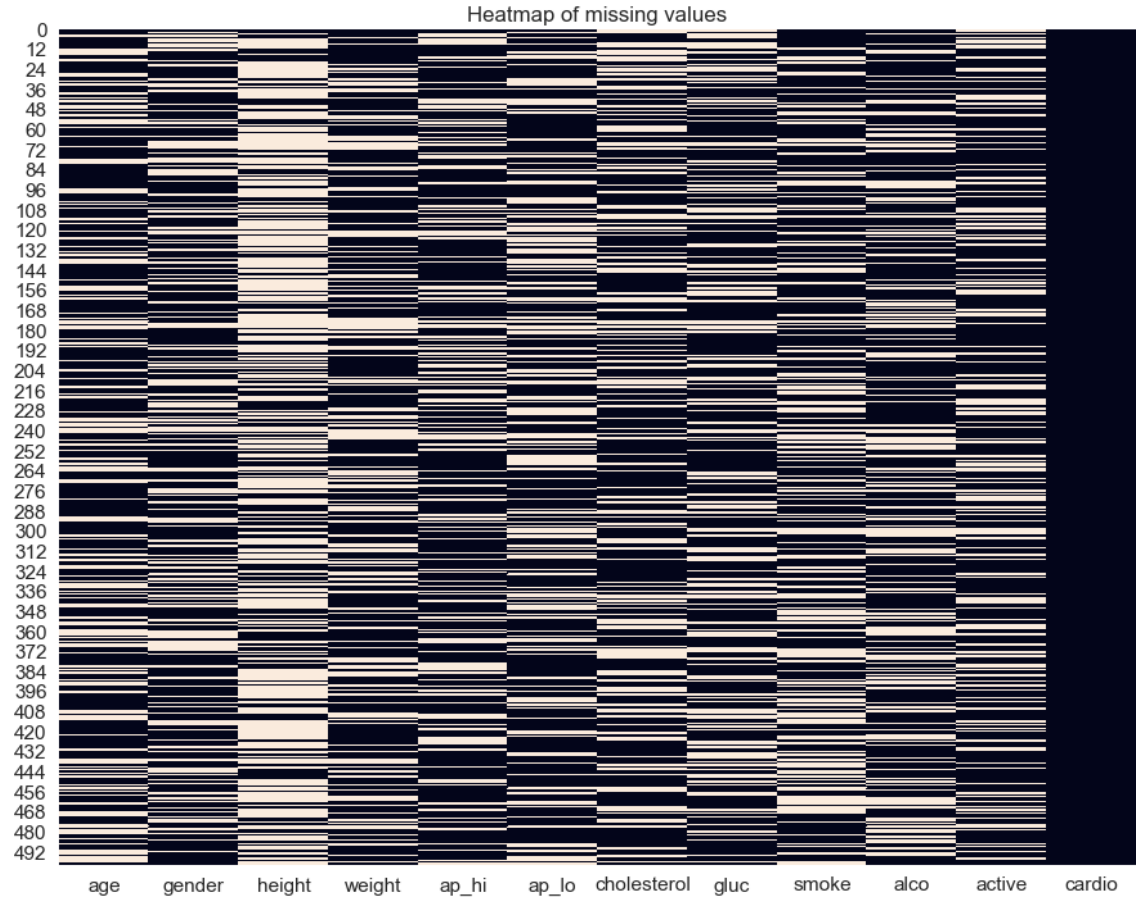


figure 14:Missing values heatmap(1)

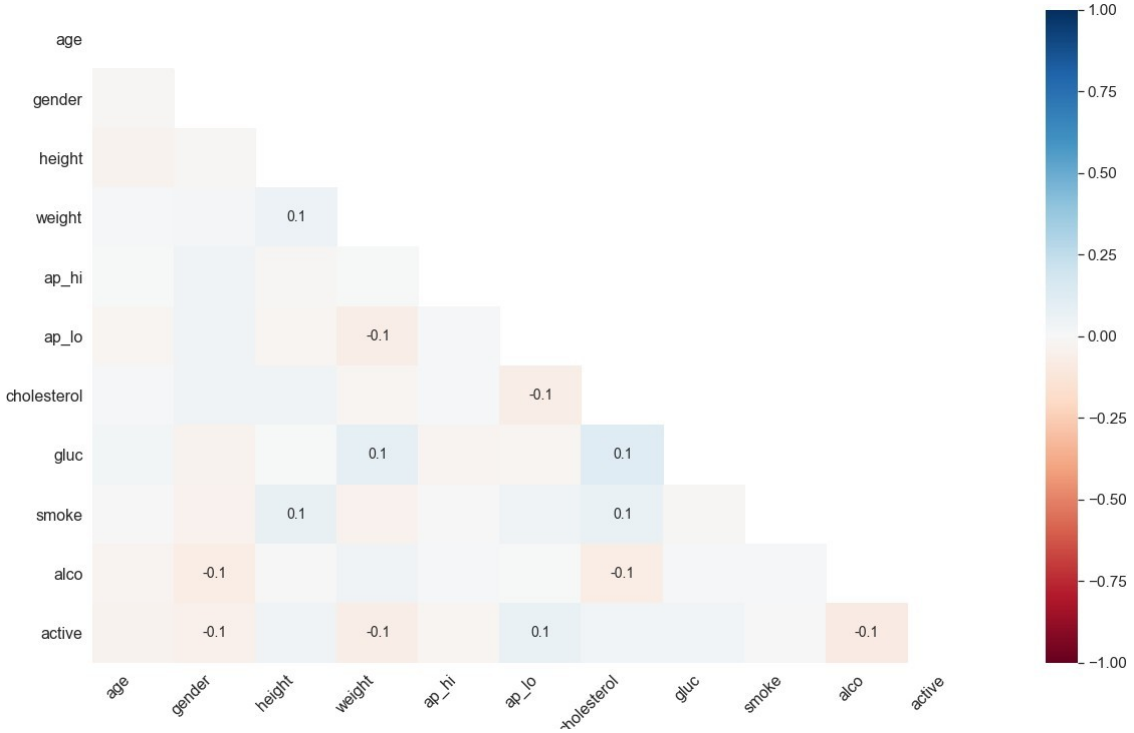


figure 15:Missing values heatmap(2)

In figure 15, it is clearly visible that missing values are missing at random as the correlation matrix gives values very close to 0.

Techniques to fill missing values:

1. **Drop all Nanvalues**
2. **Replace all Nan values byzero**
3. **Replace all Nan values with max**
4. **Replace Nan values with mean**
5. **Replace Nan values with median**
6. **Replace Categorical values with mode and Continuous values with mean and median: (best estimator)**
7. **Miss Forest missingpy:**

Q. How to estimate which strategy has worked the best ?
We can say that a strategy has worked best when the difference of values between means and standard deviations before and after imputing missing values in the least.

Suppose a dataframe computed 'data_mean' has missing values filled with mean and 'data' is our original dataframe. So, (data_mean.mean()-data.mean()) will give the difference of mean between the missing value technique and original data for each feature.
Summation over that will give the sum of means for all features.

```
#difference of zero and original
zero_m=(cardio_impute_zeros.mean()-cardio.mean()).sum()
zero_s=(cardio_impute_zeros.std()-cardio.std()).sum()
print("Difference of mean ",zero_m)
print("Difference of std ",zero_s)
```

```
Difference of mean  -207.8079386640284
Difference of std  178.26065641004848
```

```
#difference of dropped and original
drop_m=(cardio_drop.mean()-cardio_copy.mean()).sum()
drop_s=(cardio_drop.std()-cardio_copy.std()).sum()
print("Difference of mean ",drop_m)
print("Difference of std ",drop_s)
```

```
Difference of mean  -7.444032667967421
Difference of std  -85.36919172035404
```

```
#difference of mean and original
mean_m=(cardio_impute_mean.mean()-cardio_copy.mean()).sum()
mean_s=(cardio_impute_mean.std()-cardio_copy.std()).sum()
print("Difference of mean ",mean_m)
print("Difference of std ",mean_s)
```

```
Difference of mean  8.168465903679589e-13
Difference of std  -26.682991460802253
```

```
#difference of median and original
median_m=(cardio_impute_median.mean()-cardio_copy.mean()).sum()
median_s=(cardio_impute_median.std()-cardio_copy.std()).sum()
print("Difference of mean ",median_m)
print("Difference of std ",median_s)
```

```
Difference of mean  -6.044918382253137
Difference of std  -25.83845538113237
```

```
#difference of mean for missforest and original
forest_m=round((df_forest.mean()-cardio_copy.mean()).sum(),2)
forest_s=round((df_forest.std()-cardio_copy.std()).sum(),2)
print("Difference of mean ",forest_m)
print("Difference of std ",forest_s)
```

```
Difference of mean  1.04
Difference of std  595.24
```

```
#difference of mean for mean-median-mode and original
mean_mode_m=(cardio.mean()-cardio_copy.mean()).sum()
mean_mode_s=(cardio.std()-cardio_copy.std()).sum()
print("Difference of mean ",mean_mode_m)
print("Difference of std ",mean_mode_s)
```

```
Difference of mean  -3.5889797182247194
Difference of std  -26.358758876297145
```

We observe that imputing by mean,median and mode gives least differenec with respect to mean and standard deviations of the original data

Feature Engineering:

Dropped features:

I've dropped the column 'id' as it is unique to every data entry and will not affect the decision making process

Working on Test dataset(cardio-validation.csv):

The testing data needs to be refined so that the distribution is the same as our training dataset. So I have done the following computations:

1. Dropped 'id' column
2. Converting age from days to years
3. Mapping categorical string features into categorical numerical
4. Filling Missing values using Mean-Median-Mode imputation

Train-Test data:

Now we have the perfect data we want to use for our model.

Training data:

1. x_train, y_train: Extracted from Training data frame with 'cardio' feature as the target variable and other features as training variables and scaled x_train using sklearnStandardscaler

Validation data:

1. x_test y_test: Extracted from Validation data frame with 'cardio' feature as the target variable and other features as training variables and scaled x_test using sklearnStandardscaler

Models:

I tried using 3 models: Logistic regression, SVM and Random forest. Out of which random forest gave the highest accuracy 73.2%

SVM MODEL:

First I fit the model in the training data alone and tried finding best values for the parameters C and Kernel wrt cardio-validation dataset and got an accuracy of 73%.

```
from sklearn.metrics import f1_score
from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
print("f1_score of svm model wrt train data:",f1_score(y_test, y_pred))
```

```
Accuracy: 0.73
              precision    recall  f1-score   support

           0       0.83      0.70      0.76       307
           1       0.62      0.78      0.69       193

    accuracy                           0.73       500
   macro avg       0.73      0.74      0.73       500
weighted avg       0.75      0.73      0.73       500

f1_score of svm model wrt train data: 0.6896551724137931
```

With the best parameters at hand, I fit the model on train+validation data and got an accuracy of about 72%

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
print("f1_score of svm model wrt train data:",f1_score(y_test, y_pred))
```

```
Accuracy: 0.72
              precision    recall  f1-score   support

           0       0.84      0.68      0.75       155
           1       0.60      0.79      0.68        95

    accuracy                           0.72       250
   macro avg       0.72      0.73      0.72       250
weighted avg       0.75      0.72      0.72       250

f1_score of svm model wrt train data: 0.6818181818181819
```

LOGISTIC REGRESSION:

First I fit the model in the training data alone and tried finding best values for the parameters C wrt cardio-validation dataset and got an accuracy of 73%.

```
# instantiate the model (using the default parameters)
logreg = LogisticRegression(C = 10,max_iter = 2000)
# fit the model with data
logreg.fit(X_train,y_train)
y_pred=logreg.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.73
```

With the best parameters at hand, I fit the model on train+validation data and got an accuracy of about 72.8%

```
# instantiate the model (using the default parameters)
logreg = LogisticRegression(C= 10,max_iter=2000)
# fit the model with data
logreg.fit(X_train,y_train)
y_pred_task1 = logreg.predict(X_test)
y_test_task1 = y_test.copy()
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_task1))
```

```
Accuracy: 0.728
```

LOGISTIC REGRESSION(own implementation):

```
class LogisticRegression_new:
    def __init__(self, lr=0.001, n_iterations=1000):
        self.lr=lr
        self.n_iterations=n_iterations
        self.weights= None
        self.bias = None

    def fit(self,X,y):
        m, n = X.shape
        self.weights = np.zeros(n)
        self.bias=0

        # gradient function
        for _ in range (self.n_iterations):
            linear_model=np.dot(X, self.weights) + self.bias
            y_pred=self.sigmoid_function(linear_model)

            dw = (1/m)*np.dot(X.T, (y_pred-y))
            db = (1/m)*np.sum(y_pred - y)

            self.weights -= self.lr*dw
            self.bias -= self.lr*db

        # value prediction
    def predict(self,X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_pred = self.sigmoid_function(linear_model)
        y_pred_class = [1 if i>0.5 else 0 for i in y_pred]
        return y_pred_class

    def sigmoid_function(self,x):
        return 1 / (1 + np.exp(-x))
```

Got 71.4% accuracy wrt train and validation data

```
reg_lr= LogisticRegression_new( lr= 0.0001, n_iterations=1000)
reg_lr.fit(X_train, y_train)
pred = reg_lr.predict(X_test)
print("Accuracy score :", np.sum(y_test==pred) / len(y_test))
```

```
Accuracy score : 0.714
```

RANDOM FOREST(Best model):

First I fit the model in the training data alone and tried finding best values for the parameters n_estimators , max_depth, min_samples_split wrt cardio-validation dataset and got an accuracy of 76%.

```
model_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=5,min_samples_split=2,n_jobs=-1,
model_rf.fit(X_train,y_train)
y_pred_test = model_rf.predict(X_test)
acc = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test)
print("Random Forest Accuracy = %.3f, F1-score = %.3f"% (acc, f1))
from sklearn.metrics import classification_report
print(classification_report(y_pred_test,y_test))
```

```
Random Forest Accuracy = 0.768, F1-score = 0.747
              precision    recall  f1-score   support

           0       0.83      0.75      0.79       284
           1       0.71      0.79      0.75       216

    accuracy                           0.77       500
   macro avg       0.77      0.77      0.77       500
weighted avg       0.77      0.77      0.77       500
```

With the best parameters at hand, I fit the model on train+validation data and got an accuracy of about 73.2%

```
model_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_depth=5,min_samples_split=2,n_jobs=-1,
model_rf.fit(X_train,y_train)
y_pred = model_rf.predict(X_test)
acc = accuracy_score(y_test_task1, y_pred)
f1 = f1_score(y_test_task1, y_pred)
print("Random Forest Accuracy = %.3f, F1-score = %.3f"% (acc, f1))
from sklearn.metrics import classification_report
print(classification_report(y_pred_task1,y_test_task1))
```

```
Random Forest Accuracy = 0.732, F1-score = 0.707
              precision    recall  f1-score   support

           0       0.84      0.69      0.76       153
           1       0.62      0.79      0.69        97

    accuracy                           0.73       250
   macro avg       0.73      0.74      0.72       250
weighted avg       0.75      0.73      0.73       250
```

Kaggle Upload:

The prediction that needs to be uploaded on kaggle is on a different dataset which is ‘cardio-test.csv’. In order to predict on this data, there are some changes that need to be done:

1. Dropping ‘id’
2. Converting age from days to years
3. Label encoding all string categorical values
4. Standardizing the data

The dataset has 250 data records without the cardio column. The predictions are made using the Random Forest model and are uploaded to kaggle.

Task 2:

Check the predictions of our model on cardio-complete dataset which doesn’t have any missing values:

Data PreProcessing:

1. Dropping column ‘id’
2. Label encoder for gender, cholesterol and gluc
3. Changing ‘age’ from days to years
4. Split the data into 75:25 training testing

logistic Regression:

Logistic regression is applied using HyperParameter tuning and the best parameters were C = 0.01 The data is fitted on these parameters and an accuracy score was generated as: **73.2%** before selecting a threshold.

```
# instantiate the model (using the default parameters)
logreg = LogisticRegression(C = 0.01,max_iter = 2000)
# fit the model with data
logreg.fit(X_train,y_train)
y_pred_task2 = logreg.predict(X_test)
y_test_task2 = y_test.copy()
print("Accuracy:",metrics.accuracy_score(y_test_task2, y_pred_task2))
f1 = f1_score(y_test_task2, y_pred_task2)
print("Logistic regression Accuracy = %.3f, F1-score = %.3f"% (acc, f1))
from sklearn.metrics import classification_report
print(classification_report(y_pred_test,y_test))
```

```
Accuracy: 0.732
Logistic regression Accuracy = 0.732, F1-score = 0.727
              precision    recall  f1-score   support

           0       0.76      0.71      0.74       132
           1       0.70      0.75      0.73       118

    accuracy                           0.73       250
   macro avg       0.73      0.73      0.73       250
weighted avg       0.73      0.73      0.73       250
```

Comparing Task 2 and Task 1 prediction scores:

	f1	Precision	Recall	Accuracy
Task 1	0.693694	0.793814	0.616000	0.728
Task 2	0.726531	0.754237	0.700787	0.732

Table 5: Task 2 and Task 1 comparison(1)

The table above shows the precision, recall, f1, accuracy scores on the testing datasets for task 1 and task 2. Task 2 has performed slightly better than Task 1.

Here are confusion matrix for task 1 and task 2:

Confusion matrix for Task 1

Predicted	0	1
Actual		
0	105	20
1	48	77

Confusion matrix for Task 2(cardio-complete)

Predicted	0	1
Actual		
0	94	29
1	38	89

Task 3:

This task aims to understand the concept of overfitting and underfitting.

Here I've transformed the training features from task 1 into second degree polynomial and have compared training and cross-val scores before and after adding polynomial features. I've used Logistic regression here as our aim is to understand the concept of Bias and Variance and not some complex mode.

Here is the comparison:

```
Before feature transform:
Train score:  0.706
Cross-val score:  0.69

After degree 2 feature transform:
Train score:  0.756
Cross-val score:  0.682

After degree 3 feature transform:
Train score:  0.786
Cross-val score:  0.6779999999999999
```

From the above result we can see that the training accuracy increases and testing accuracy decreases after adding polynomial features. Clear case of *Overfitting*. This means that the model is trying to fit the data very well but can't generalize the model.

Visualization:

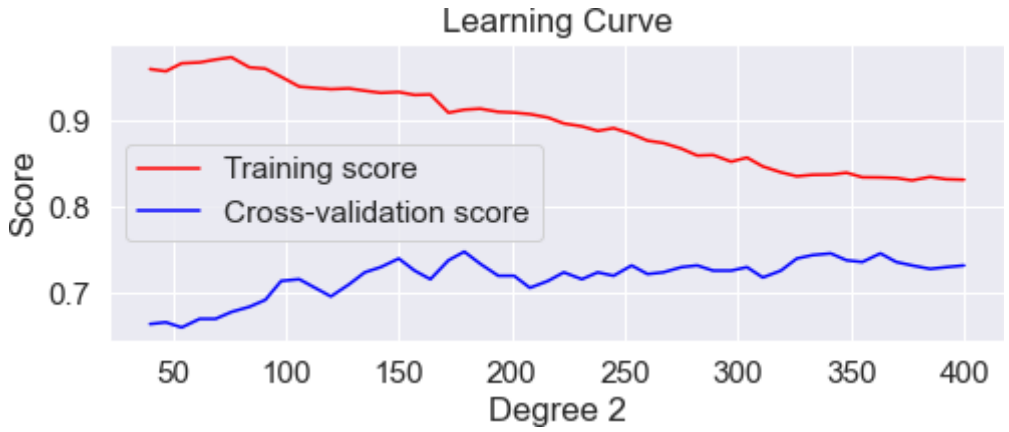


figure 16: Degree 2polynomial

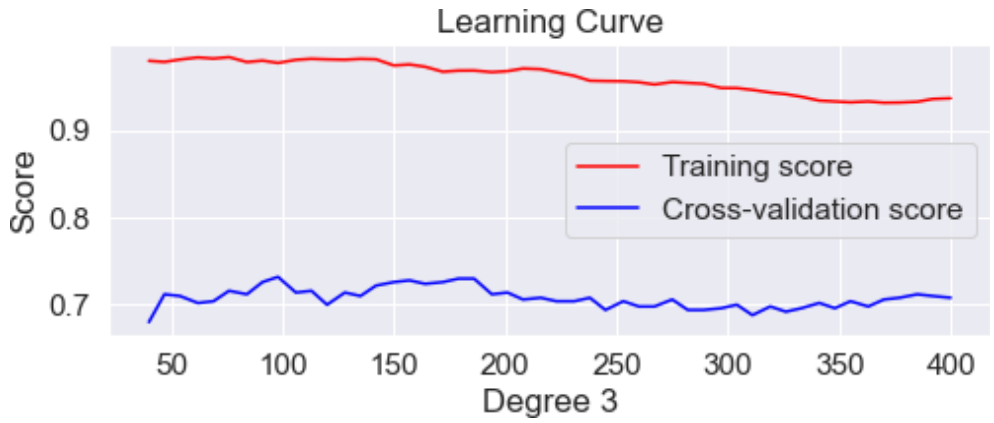


figure 17: Degree 3polynomia

As we can see from the graph that Training score is higher than cross-validation score which means that there is high variance. We can also notice that as we increase the degree of polynomial features the model overfits the data even more.

Conclusion:

Task 1 is mostly data visualization and prediction on kaggle(cardio-test) dataset using the best model which was Random forest. We used cardio-train and cardio-validation to predict on the test dataset. Missing values were filled using Mean-Median-Mode.

Task 2 was testing the model used in Task 1 on the cardio-complete dataset. The given dataset didn't have any missing values. Results of task 1 and task 2 were compared.

Task 3 gave the understanding of overfitting the data when complex features are added. Training and cross validation scores were plotted for polynomial features with degree 2 and 3.