Introduction

Many years ago, was so difficult to diagnose a heart issue. Directly, all depend on the qualification of the doctor and his state. But sometimes the best one could not diagnose it and many people live and never know that they may be able to have problems with the heart, but when it happened it might be so late, and using way by taking medicine may never help.

Humanity's body has many problems which may lead to heart disease. Many factors may affect each other, for example, increased cholesterol can make narrow veins and blood pressure will be increased. In this case, the heart will be overloaded and finally could bring to the worst consequences.

Today humanity has a lot of different technologies which can help to save lives before then it would be late and one of these is machine learning.

Description of dataset

The database was downloaded from:
http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/
For prediction, we use the dataset called "processed.cleveland.data"

This dataset has 14 attributes. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0). The names and social security numbers of the patients were recently removed from the database, replaced with dummy values. One file has been "processed", that one containing the Cleveland database. All four unprocessed files also exist in this directory. **It was mentioned in the description file "heart-disease.names"**

Attributes:

1 **age**	*(age in years)*
2 **sex**	*(sex 1 = male, 0 = female)*
3 **cp**	*(chest pain type, 1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic)*
4 **trestbps**	*(resting blood pressure (in mm Hg on admission to the hospital))*
5 **chol**	*(serum cholestoral in mg/dl)*
6 **fbs**	*(fasting blood sugar > 120 mg/dl, 1 = true; 0 = false)*
7 **restecg**	*(resting electrocardiographic results, 0: normal, 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), 2: showing probable or definite left ventricular hypertrophy by Estes' criteria)*
8 **thalach**	*(maximum heart rate achieved)*
9 **exang**	*(exercise induced angina, 1 = yes, 0 = no)*
10 **oldpeak**	*(ST depression induced by exercise relative to rest)*
11 **slope**	*(the slope of the peak exercise ST segment, 1 - upsloping, 2 - flat, 3 - downsloping)*
12 **ca**	*(number of major vessels (0-3) colored by flourosopy)*
13 **thal**	*(3 - normal, 6 - fixed defect, 7 - reversable defect)*
14 **target**	*(diagnosis of heart disease, angiographic disease status, 0: < 50% diameter narrowing, 1: > 50% diameter narrowing (in any major vessel: attributes 59 through 68 are vessels))*

Heart disease diagnostic and prediction

For our research, we use Anaconda Navigator with installed JupyterLab.

Import and reading dataset

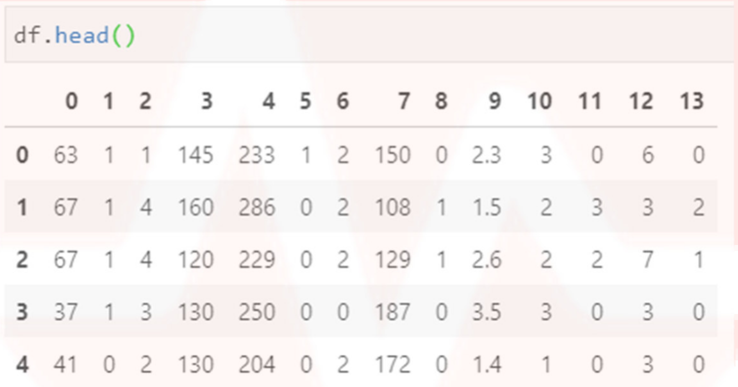
After creating a new file in JupyterLab and opening it we import need modules for importing a dataset:

```
#Importing needed modules
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Next, we used the panda's function to import data from "processed.cleveland.data":

```
#Reading from TXT file by read_csv
df = pd.read_csv(r"C:\processed.cleveland.data", header = None)
```

Then we check imported data:



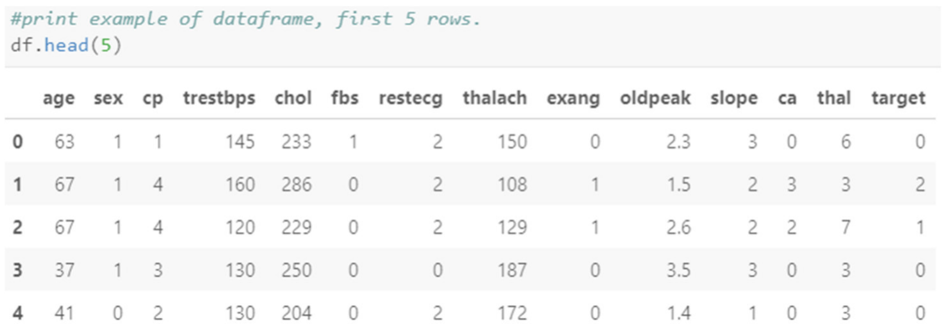
```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

The above image demonstrates that this data does not have any column name. Next, we should assign a name for each column:

```
#Naming dataframe's columns
df.columns = ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg',
                         'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target']
```

Checking naming of dataframe:



```
#print example of dataframe, first 5 rows.
df.head(5)
```

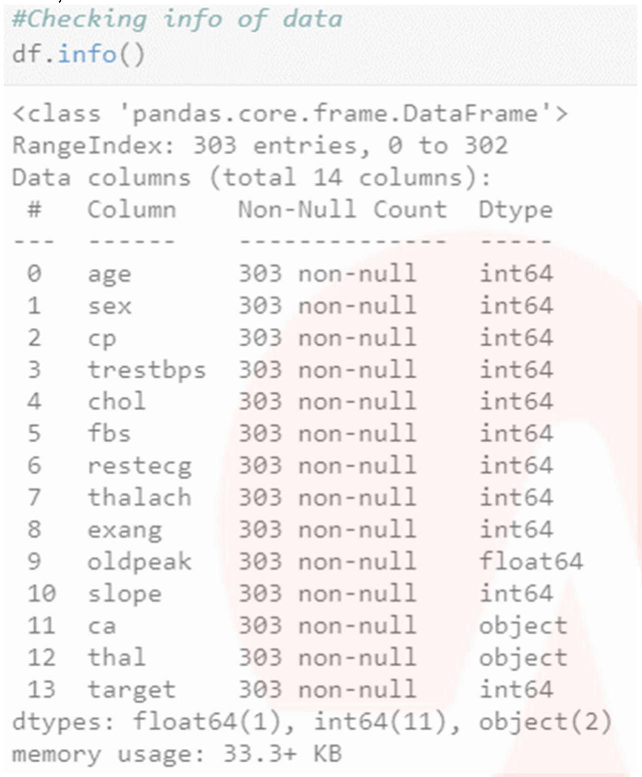
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

Now data little more readable.

Preparing and cleaning dataset

For the next stage, we should check all information about data. For example, we must understand which data type is each column and why. Also, it is important too to check missed values that could distract our prediction.

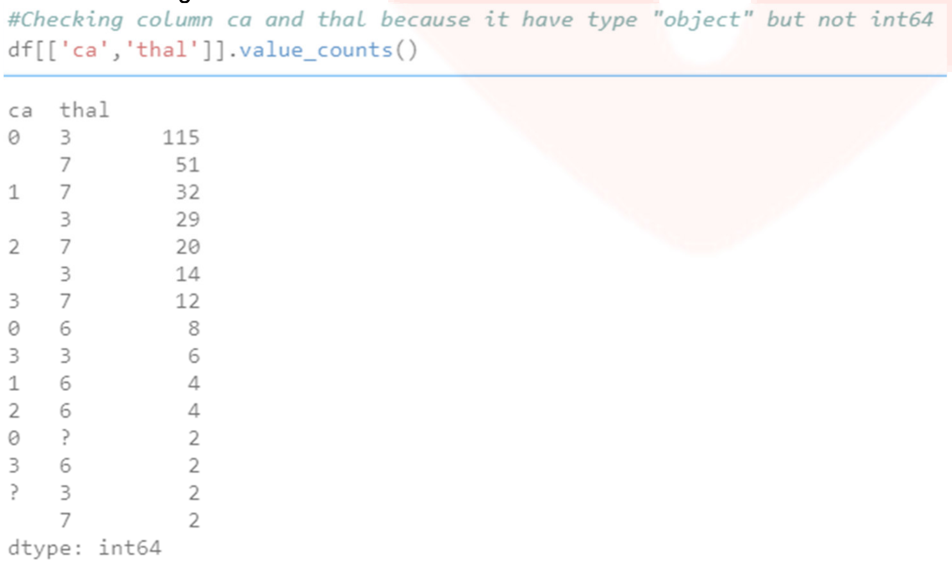
Now, lets look at the DataFrame information for understanding about datatype:



```
#Checking info of data
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    object
 12  thal      303 non-null    object
 13  target    303 non-null    int64
dtypes: float64(1), int64(11), object(2)
memory usage: 33.3+ KB
```

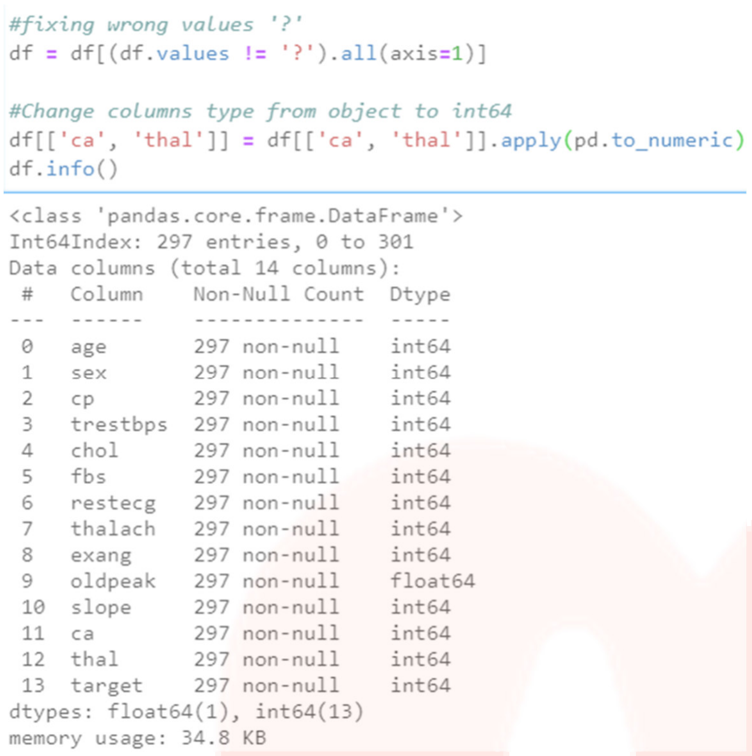
As we see, almost all columns have datatype "int64", but the columns "ca" and "thal" have datatype "object", let's look at data in strange columns:



```
#Checking column ca and thal because it have type "object" but not int64
df[['ca','thal']].value_counts()
```

```
ca  thal
0   3      115
    7       51
1   7       32
    3       29
2   7       20
    3       14
3   7       12
0   6        8
3   3        6
1   6        4
2   6        4
0   ?        2
3   6        2
?   3        2
    7        2
dtype: int64
```

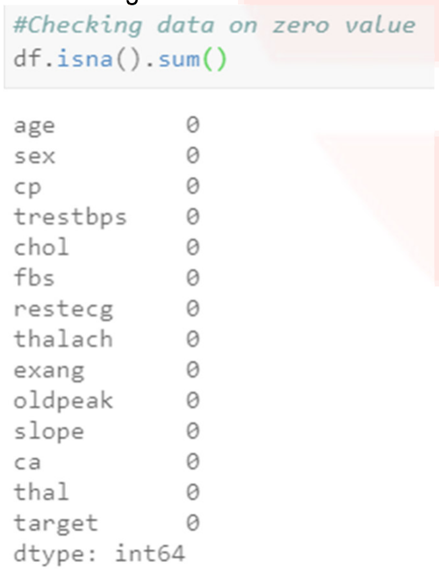
As we see, we have several characters "Question mark", we have it only 3 and 2 in "ca" and "thal" columns respectively. It is easier to remove these 5 rows and finally we will be able to fix datatype of columns:



```
#fixing wrong values '?'
df = df[(df.values != '?').all(axis=1)]

#Change columns type from object to int64
df[['ca', 'thal']] = df[['ca', 'thal']].apply(pd.to_numeric)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 297 entries, 0 to 301
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       297 non-null    int64
 1   sex       297 non-null    int64
 2   cp        297 non-null    int64
 3   trestbps  297 non-null    int64
 4   chol      297 non-null    int64
 5   fbs       297 non-null    int64
 6   restecg   297 non-null    int64
 7   thalach   297 non-null    int64
 8   exang     297 non-null    int64
 9   oldpeak   297 non-null    float64
 10  slope     297 non-null    int64
 11  ca        297 non-null    int64
 12  thal      297 non-null    int64
 13  target    297 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 34.8 KB
```

Now our dataset looks better. The next stage is checking and fixing NULL (NaN) values because some function could work not right:

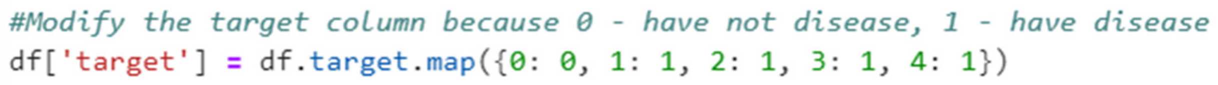


```
#Checking data on zero value
df.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

As we see, this DataFrame does not have any problems with null or NaN values.

In the dataset, we can see that the last column "target" has several values from 0 to 3. From description we can see that 0: < 50% diameter narrowing, 1: > 50% diameter narrowing (in any major vessel: attributes 59 through 68 are vessels. We decided that disease is or is not and for better understanding, we should fix values in this column like 0 is not have disease and 1, 2, 3 have heart disease. For this, we will use DataFrame map value:



```
#Modify the target column because 0 - have not disease, 1 - have disease
df['target'] = df.target.map({0: 0, 1: 1, 2: 1, 3: 1, 4: 1})
```

The last things that we can do with the DataFrame are looking through the description:

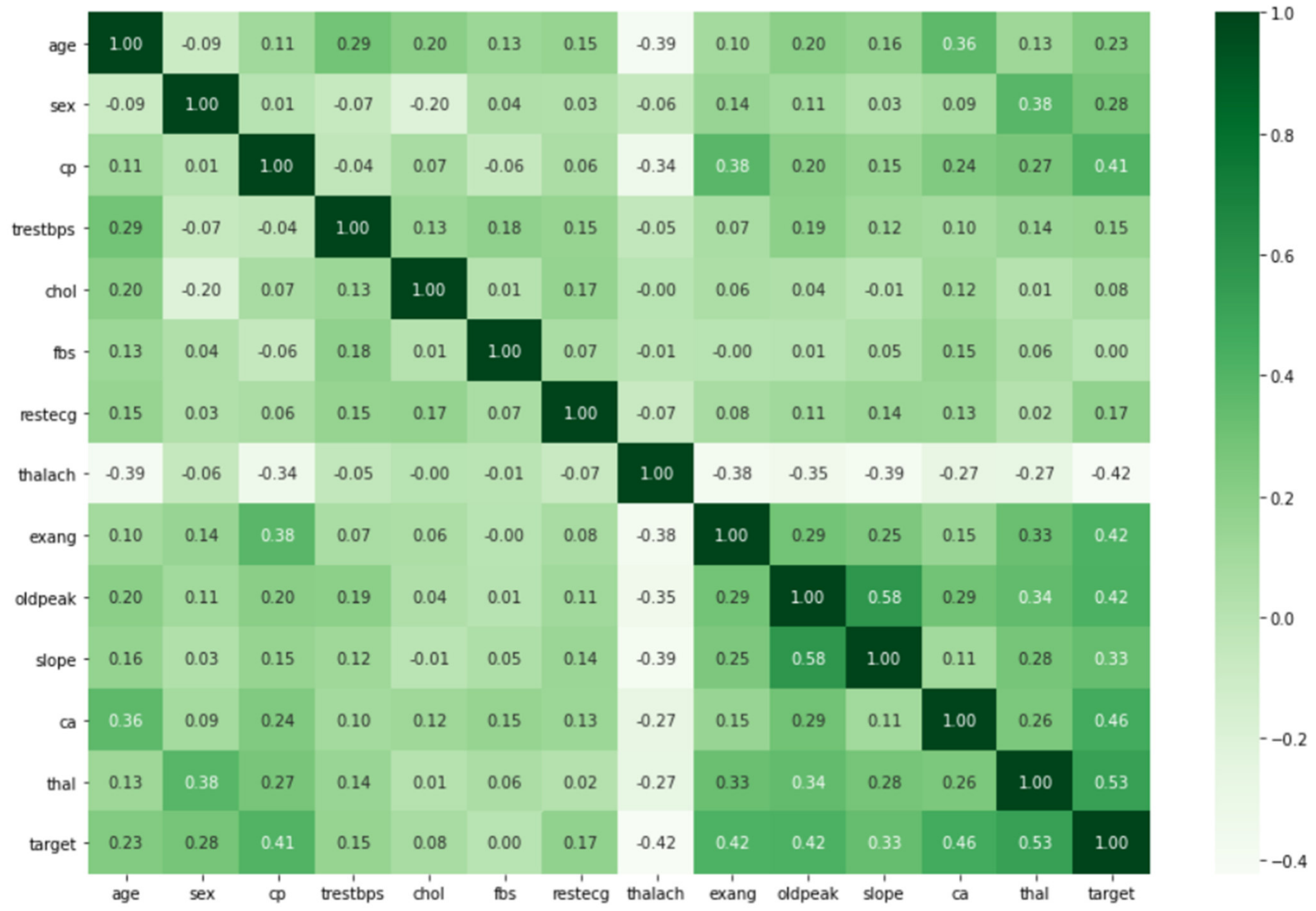
```
##Description dataset
df.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000	297.000000
mean	54.542088	0.676768	3.158249	131.693603	247.350168	0.144781	0.996633	149.599327	0.326599	1.055556	1.602694	0.676768	4.730640	0.461279
std	9.049736	0.468500	0.964859	17.762806	51.997583	0.352474	0.994914	22.941562	0.469761	1.166123	0.618187	0.938965	1.938629	0.499340
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000	0.000000	3.000000	0.000000
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.000000	0.000000	0.000000	1.000000	0.000000	3.000000	0.000000
50%	56.000000	1.000000	3.000000	130.000000	243.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000	0.000000	3.000000	0.000000
75%	61.000000	1.000000	4.000000	140.000000	276.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000	1.000000	7.000000	1.000000
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000	3.000000	7.000000	1.000000

Data analysis

Python has a lot of tools for data analysis, but more informative, as we think, is a correlation matrix built with heatmap:

```
#Visualize correlation matrix by matrix with heatmap
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True,cmap="Greens",fmt='.2f')
```



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
age	1.00	-0.09	0.11	0.29	0.20	0.13	0.15	-0.39	0.10	0.20	0.16	0.36	0.13	0.23
sex	-0.09	1.00	0.01	-0.07	-0.20	0.04	0.03	-0.06	0.14	0.11	0.03	0.09	0.38	0.28
cp	0.11	0.01	1.00	-0.04	0.07	-0.06	0.06	-0.34	0.38	0.20	0.15	0.24	0.27	0.41
trestbps	0.29	-0.07	-0.04	1.00	0.13	0.18	0.15	-0.05	0.07	0.19	0.12	0.10	0.14	0.15
chol	0.20	-0.20	0.07	0.13	1.00	0.01	0.17	-0.00	0.06	0.04	-0.01	0.12	0.01	0.08
fbs	0.13	0.04	-0.06	0.18	0.01	1.00	0.07	-0.01	-0.00	0.01	0.05	0.15	0.06	0.00
restecg	0.15	0.03	0.06	0.15	0.17	0.07	1.00	-0.07	0.08	0.11	0.14	0.13	0.02	0.17
thalach	-0.39	-0.06	-0.34	-0.05	-0.00	-0.01	-0.07	1.00	-0.38	-0.35	-0.39	-0.27	-0.27	-0.42
exang	0.10	0.14	0.38	0.07	0.06	-0.00	0.08	-0.38	1.00	0.29	0.25	0.15	0.33	0.42
oldpeak	0.20	0.11	0.20	0.19	0.04	0.01	0.11	-0.35	0.29	1.00	0.58	0.29	0.34	0.42
slope	0.16	0.03	0.15	0.12	-0.01	0.05	0.14	-0.39	0.25	0.58	1.00	0.11	0.28	0.33
ca	0.36	0.09	0.24	0.10	0.12	0.15	0.13	-0.27	0.15	0.29	0.11	1.00	0.26	0.46
thal	0.13	0.38	0.27	0.14	0.01	0.06	0.02	-0.27	0.33	0.34	0.28	0.26	1.00	0.53
target	0.23	0.28	0.41	0.15	0.08	0.00	0.17	-0.42	0.42	0.42	0.33	0.46	0.53	1.00

As we can see, the highest correlation to "target" have next columns: "thal", "ca", "oldpeak", "exang", "cp" and "slope". We think that using these columns should give to us a good prediction.

Data visualization

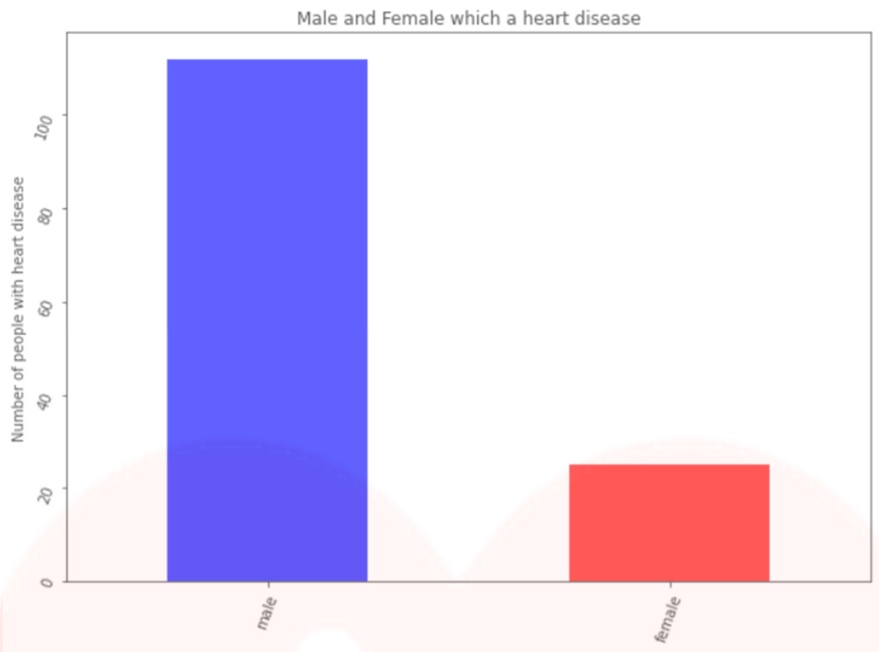
For better understanding, we should copy to the new DataFrame and rename some data. Why we did not it before because some functions work only with numeric and give mistake if in the table be some value like string or character. For example. in column "sex" we have values 0 and 1, when we will build a chart without renaming values it will be not so useful but if we change 0 to female and 1 to male it will be more understandable.

Write several functions and apply them for changing the data:

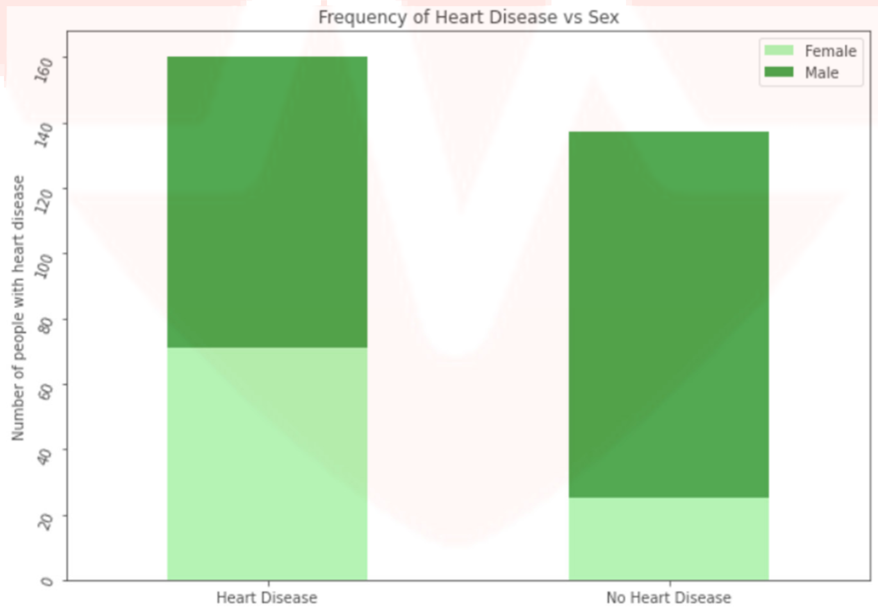
```
#Copy several columns to new DataFrame
df_v01 = df[['sex','cp','thal','target']]

#Create several functions for mapping new values
def mapSex(sex):
    if sex == 0:
        return 'female'
    else:
        return 'male'
def mapDisease(target):
    if target == 0:
        return 'Heart Disease'
    else:
        return 'No Heart Disease'
def mapThal(thal):
    if thal == 3:
        return 'Normal'
    elif thal == 6:
        return 'Fixed Defect'
    elif thal == 7:
        return 'Reversable Defect'
def mapCp(cp):
    if cp == 1:
        return 'Typical Angina'
    elif cp == 2:
        return 'Atypical Angina'
    elif cp == 3:
        return 'Non-Anginal Pain'
    elif cp == 4:
        return 'Asymptomatic'

#Apply these functions
df_v01.update(df_v01.loc[:,'sex'].apply(mapSex))
df_v01.update(df_v01.loc[:,'cp'].apply(mapCp))
df_v01.update(df_v01.loc[:,'thal'].apply(mapThal))
df_v01.update(df_v01.loc[:,'target'].apply(mapDisease))
```

Now we can build a diagram and look at how many have heart disease have female and male:

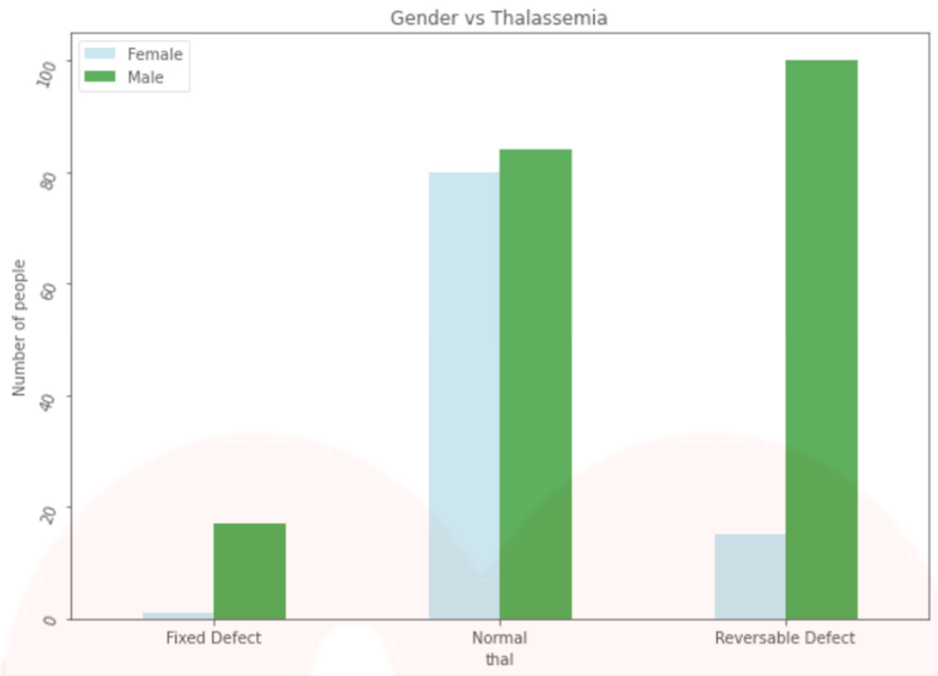
```
#Build bar-chart showing number of people with disease
df_v01.sex[df.target==1].value_counts().plot(kind='bar',rot=0,figsize=(10,7),color=['blue','red'])
plt.title("Male and Female which a heart disease");
plt.ylabel("Number of people with heart disease")
plt.xlabel("")
plt.xticks(rotation=70);
plt.yticks(rotation=70);
```



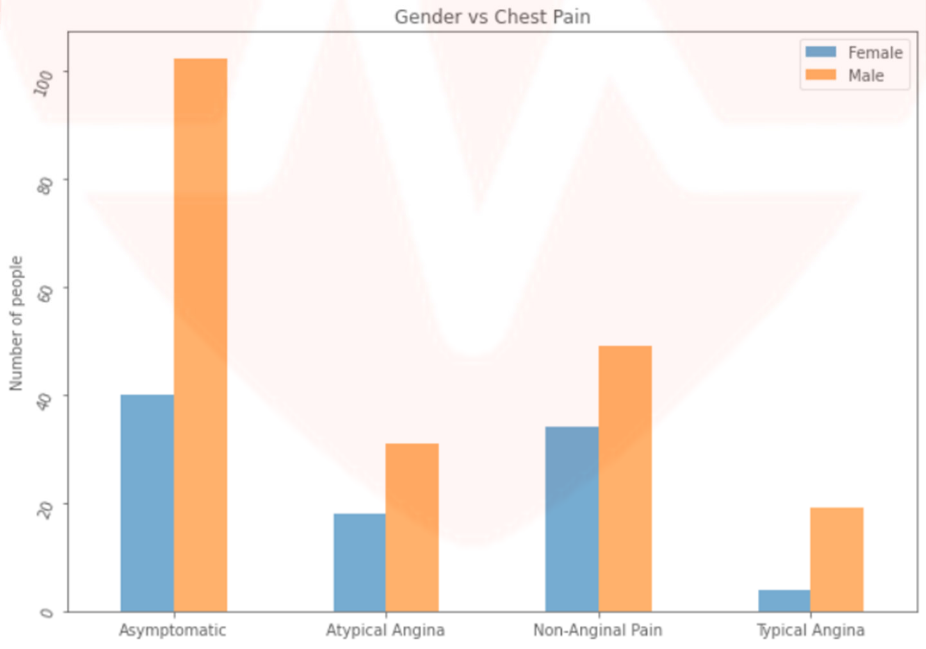
The image above shows that the dataset has more men with heart disease than women.
Use this chart we can visualize any data from the dataset. Let look next the chart like "Frequency of Heart Disease vs Sex":



The image above shows that males have more frequent heart disease then females, maybe because females break male's hearts, but for checking this need additional attributes. Let's compare gender vs thalassemia:



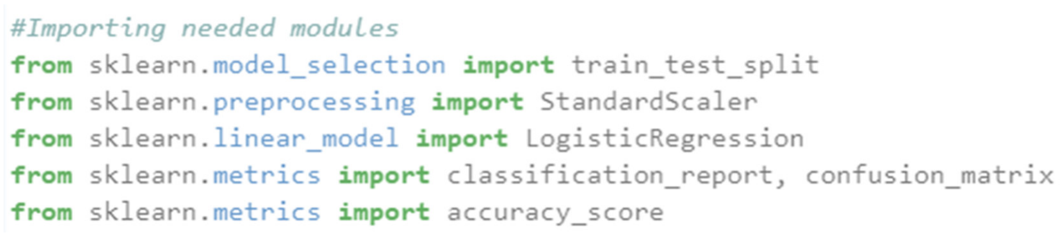
The chart above has so clear information that men have problems and with thalassemia too. Let's do last a chart for demonstrating chest pain and who have oftener men or women?



As we see that men have more frequent problems with health than women. Python has a lot of other charts and also we can create many other comparisons.

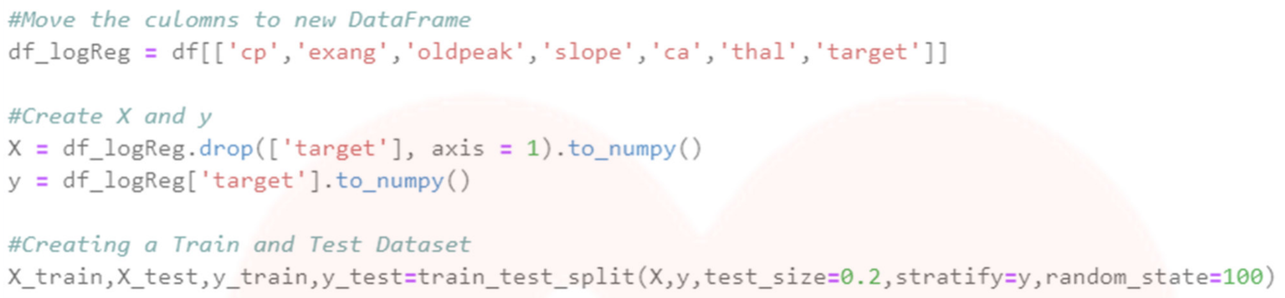
Data preprocessing

The next stage is preparing data for machine learning. We must perform the next stages:
Firstly, we should import needed modules:



```
#Importing needed modules
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

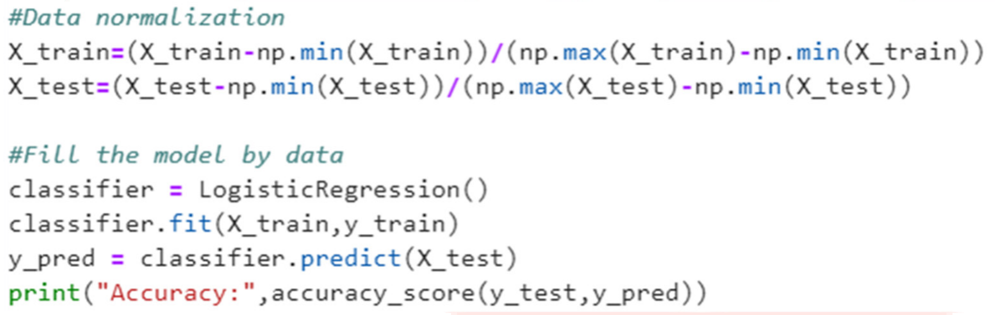
Next, we have to pick out need columns relay on the suggestion of a correlation matrix and prepare data train and test data:



```
#Move the culomns to new DataFrame
df_logReg = df[['cp','exang','oldpeak','slope','ca','thal','target']]

#Create X and y
X = df_logReg.drop(['target'], axis = 1).to_numpy()
y = df_logReg['target'].to_numpy()

#Creating a Train and Test Dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,stratify=y,random_state=100)
```

Logistic regression

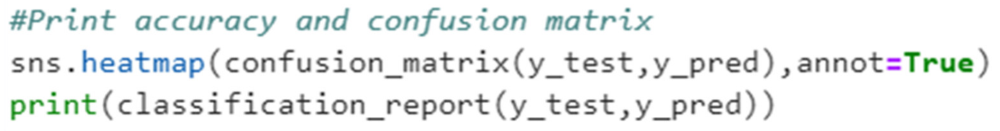
Now we can try to predict by line regression, but first, it should be better to normalize data by formula (X-Xmin)/(Xmax-Xmin):



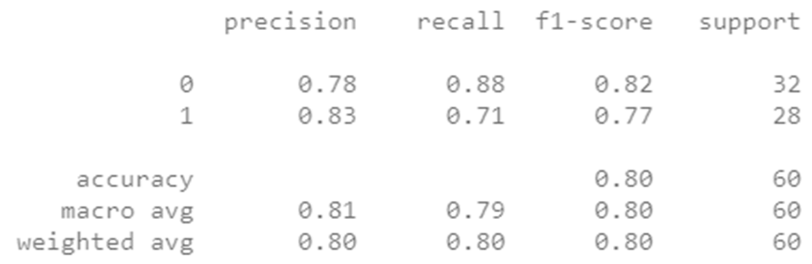
```
#Data normalization
X_train=(X_train-np.min(X_train))/(np.max(X_train)-np.min(X_train))
X_test=(X_test-np.min(X_test))/(np.max(X_test)-np.min(X_test))

#Fill the model by data
classifier = LogisticRegression()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
```

This prediction about 0.8 accuracies.

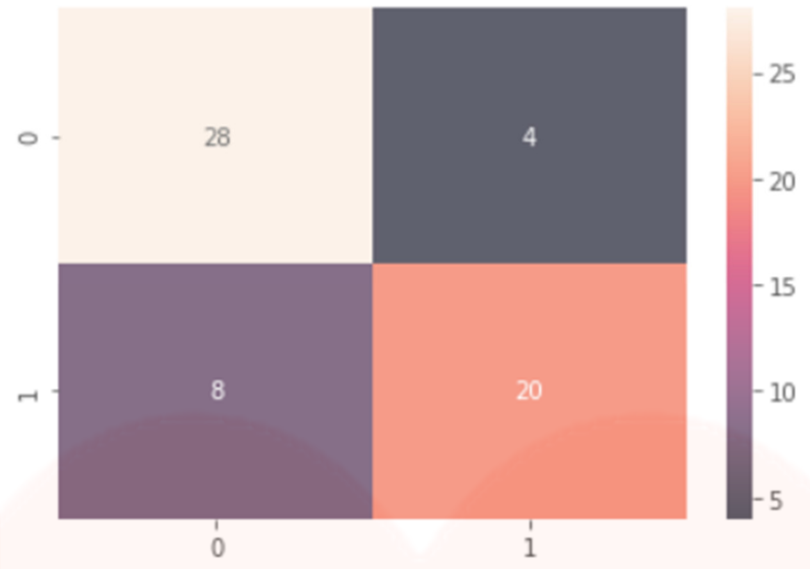
Now we can print accuracy scores by classification report and print confusion matrix for understanding true and mistakes:



```
#Print accuracy and confusion matrix
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
print(classification_report(y_test,y_pred))
```



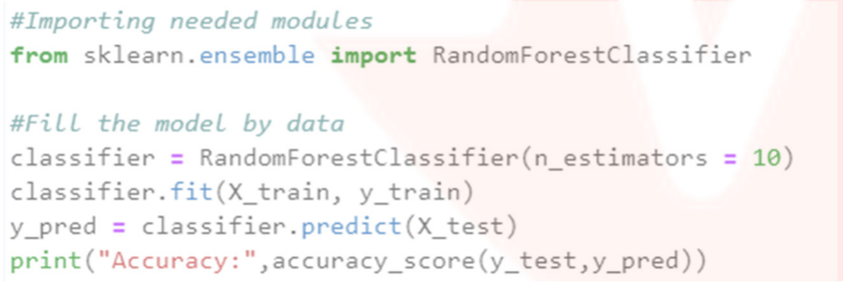
```
              precision    recall  f1-score   support

           0       0.78      0.88      0.82        32
           1       0.83      0.71      0.77        28

    accuracy                           0.80        60
   macro avg       0.81      0.79      0.80        60
weighted avg       0.80      0.80      0.80        60
```



Count trust: 28 + 20 / 60 = 80
Count mistakes: 8 + 4 / 60 = 0.2

Now we can try different classifications like a Random Forest Classification, a Decision Tree, a GaussianNB. and we can understand which works better with our dataset.

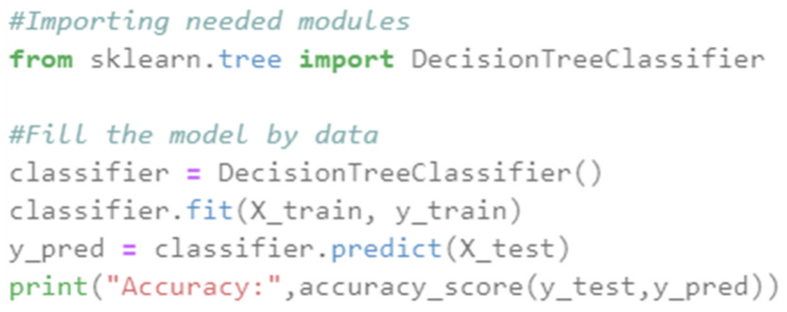
Random Forest Classification



```
#Importing needed modules
from sklearn.ensemble import RandomForestClassifier

#Fill the model by data
classifier = RandomForestClassifier(n_estimators = 10)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
```

Accuracy: 0.7833333333333333

Decision Tree



```
#Importing needed modules
from sklearn.tree import DecisionTreeClassifier

#Fill the model by data
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
```

Accuracy: 0.7166666666666667

GausianNB

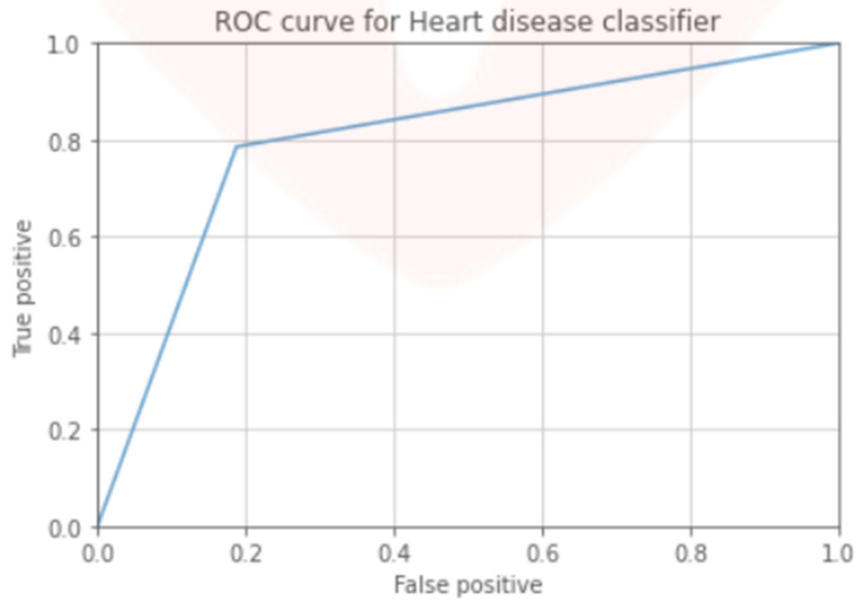
```
#Importing needed modules
from sklearn.naive_bayes import GaussianNB

#Fill the model by data
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
```

```
Accuracy: 0.8333333333333334
```

Visualization prediction

Finally, it is the last stage. It would be better to visualize our prediction by ROC curve. For this stage, we should import the needed library and fit the parameters of our prediction:

```
#Importing needed modules
from sklearn.metrics import roc_curve

#Build ROC curve for Heart disease classifier
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.plot(fpr,tpr)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.title('ROC curve for Heart disease classifier')
plt.xlabel('False positive')
plt.ylabel('True positive')
plt.grid(True)
```



Conclusion

We imported data from the dataset, checked it, cleaned it, made some analysis, use different classifications, and finally visualize it. We can see that GaussianNB gives more accuracy than others. Random forest classification every time gives different accuracy from 0.71-0.83. I understand that it is only the start of practice and for deep understanding, it will take a lot of time in the future, but if in future it can save lives, I think it is so useful and people must improve it every time.

Content

Sources

http://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/

Janosi A., Steinbrunn W., Pfisterer M. & Detrano R (n.d.). *Heart Disease Data Set*. https://archive.ics.uci.edu/ml/datasets/heart+Disease

Vestur C. (2017, jan 29). *Building a performing Machine Learning model from A to Z* https://www.slideshare.net/CharlesVestur/building-a-performing-machine-learning-model-from-a-to-z