Department of Physics

Master's degree of Artificial Intelligence andVirtual Reality

Faculté des Sciences
كلية العلوم

Project report:

Realized by:

Zeghli Fatima Zahra
Ouafa Ait Ouamer
Bourass Wiame

Supervised by:

Pr. Kaicer mohammed

Titled:

« Predicting heart disease using machinelearning »

College Academic Year: 2022/2023

Introduction

Objectif

The objective of using machine learning for statistics in heart disease is to develop models that can predict the likelihood of an individual developing heart disease based on various statistical features. These features may include demographic characteristics, medical history, lifestyle factors, and other variables that are known to be associated with heart disease.

The goal of such machine learning models is to provide doctors, healthcare providers, and individuals with a tool that can help them better understand their risk of developing heart disease, and take steps to prevent or manage it. For example, a machine learning model that predicts the likelihood of an individual developing heart disease based on their age, cholesterol levels, and blood pressure could be used to help identify individuals who are at high risk of developing heart disease, and recommend preventive measures such as lifestyle changes or medication.

In addition to predicting the risk of heart disease, machine learning models can also be used to analyze large datasets of heart disease patients and identify patterns or trends that may be useful in understanding the disease and developing new treatments. For example, a machine learning model could be used to identify common risk factors or patterns of progression in heart disease, which could help inform the development of new therapies or prevention strategies.

Development tools

Google Collab :

Google Colab is a free online platform that allows users to run and execute Jupyter notebooks in the cloud. It is particularly useful for machine learning because it provides access to powerful computing resources, such as GPUs, which can be used to trainmachine learning models. It also includes various libraries and tools that are commonly used in machine learning, such as TensorFlow and PyTorch.

Figure 1.1: Logo de Google Collab

One of the main benefits of using Colab for machine learning is that it allows users to experiment with different models and parameters quickly and easily, without having to set up their own local development environment. It is particularly useful for those who do not have access to high-performance computers or do not want to spend the time and resources to set up their own machine learning environment.

In addition to running Jupyter notebooks, Colab also provides a variety of other tools and resources for machine learning, such as integration with Google Drive for storing and sharing data, and the ability to import and export data from popular cloud storage platforms like Google Cloud Storage

Programming language used

Python:

Python is a popular choice for machine learning because it has a number of featuresthat make it well-suited for this task. Some of the main reasons why Python is used in machine learning include:

1. It has a wide range of libraries and tools specifically designed for machine learning, such as NumPy, pandas, and scikit-learn, which make it easy to perform tasks such as loading and manipulating data, building and training models, and evaluating their performance.
2. It is easy to learn and use, especially for those who are new to programming or machine learning. Its simple syntax and clear structure make it a great choice for experimenting and prototyping ideas.
3. It is a general-purpose language, which means that it can be used for tasks beyond machine learning, such as web development, data analysis, and more.

4. It has good support for parallel and distributed computing, which can be useful for training large machine learning models or working with large datasets.

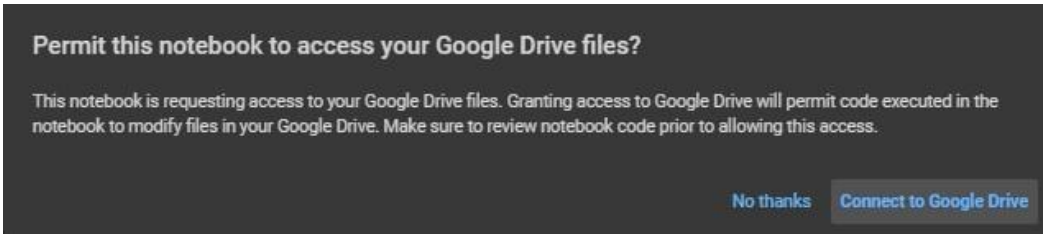
I. Data preparation

1) Accessing Google Drive from Google Collab

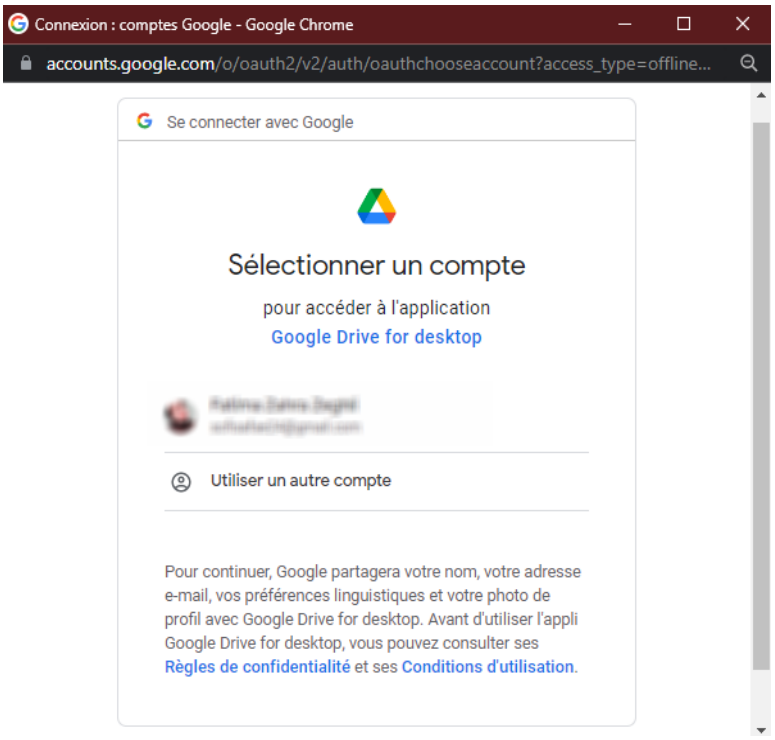
We use the drive module from google.colab to mount our entire Google Drive to Colab by executing thebelow code which will provide us with an authentication link.

```
from google.colab import drive

drive.mount('/content/drive')
```

Then we connect to google Drive



And choose the Google account whose Drive we want to mount



Once the Drive is mounted, we will access the ***'.csv'*** file that we deposed in this directory ***'/content/drive/My Drive/datasetMathAI/heart.csv'*** after importing ***pandas*** library to read the .csv file

```
Mounted at /content/drive
```

```
import pandas as pd

df = pd.read_csv('/content/drive/My Drive/datasetMathAI/heart.csv'
```

2) Introduction to our Dataset

Before proceeding with our data, it is always better to have a quick test on the dateset to see if we have the right type of data so we can establish a well built and homogeneous program, so by that we display the first five rows of the data frame to closely investigate the quality of the first few entries of the object df:

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

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

Column labels of the Data Frame df:

```
df.columns
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

The shape of df to get the number of rows and columns of our Data Frame:

```
df.shape

print("No of rows ",df.shape[0])

print("No of columns ",df.shape[1])
```

```
No of rows  1025
No of columns  14
```

3) Attribute specification:

Num of Attribute	Attribute Abbreviation	Attribute Description	Attribute's Interval
1	age	The person's age in years	[29,77]
2	sex	The person's sex (1 = male, 0 = female)	[0|1]
3	cp	chest pain type (4 values) — Value 0: asymptomatic — Value 1: atypical angina — Value 2: non-anginal pain — Value 3: typical angina	[0|1|2|3]
4	trestbps	The person's resting blood pressure (mm Hg on admission to the hospital)	[94,200]
5	chol	The person's cholesterol measurement in mg/dl	[126,564]
6	fbs	The person's fasting blood sugar (> 120 mg/dl, 1 = true; 0 = false)	[0|1]
7	restech	resting electrocardiographic results — Value 0: showing probable or definite left ventricular hypertrophy by Estes' criteria — Value 1: normal — Value 2: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)	[0|1|2]
8	thalach	The person's maximum heart rate achieved	[71,202]
9	exang	Exercise induced angina (1 = yes; 0 = no)	[0|1]
10	oldpeak	ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot. See more here)	[0.0,6.2]
11	slope	the slope of the peak exercise ST segment — 0: downsloping; 1: flat; 2: upsloping	[0|1|2]
12	ca	The number of major vessels (0–3)	[0|1|2|3]
13	thal	A blood disorder called thalassemia Value 0: NULL Value 1: fixed defect (no blood flow in some part of the heart) Value 2: normal blood flow Value 3: reversible defect (a blood flow is observed but it is not normal)	[0|1|2|3]
14	target	Heart disease (1 = no, 0= yes)	[0|1]

II. Data cleaning

1) Check data type

The variables types are categorical(qualitatives) and continuous(quantitatives) for a more optimized code we separate the list of features into two lists: listVarQuant and listVarQualt.

```
listVar = ['age','sex','cp','fbs','restecg','exang','slope','ca','thal','trestbps','chol',
'thalach','oldpeak']

listVarQuant = ['age','trestbps','chol','thalach','oldpeak']

listVarQualt = ['sex','cp','fbs','restecg','exang','slope','ca','thal']
```

Get data type of each column

```
df.types
```

```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

Count number of distinct elements on columns to look for inappropriate data

```
df.nunique()
```

```
age          39
sex           2
cp            4
trestbps     40
chol        115
fbs           2
restecg       3
thalach      76
exang         2
oldpeak      32
slope         3
ca            4
thal          3
target        2
dtype: int64
```

!'ca' with 4 distinct elements and 'thal' 3 distinct elements which contradicts feature description

2) Remove rows with defective data

```
df['ca'].unique()
```

```
array([2, 0, 1, 3, 4])
```

```
df['thal'].unique()
```

```
array([3, 2, 1, 0])
```

Now we affect the non-defective data after slicing of arrows with flawed values

```
df = df[ (df['ca'] != 4) & (df['thal'] != 3)]
```

3) Check for missing Values

```
df.isnull().sum() != 3)]
```

```
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
```

4) Check for duplicate rows

```
df.duplicated().sum()
```

```
421
```

Dropping duplicates so there's no biased values and we are left with distinct rows

```
df = df.drop_duplicates()
```

III. Statistics summary

Descriptive statistics summarizes or describes the characteristics of a data set. Descriptive statistics consists of three basic categories of measures: measures of central tendency, measures of variability (or spread), and frequency distribution.

Using describe for numerical continuous data to get a descriptive statistics summary that includes

→ count - The number of not-empty values.

→ mean - The average (mean) value.

→ std - The standard deviation.

→ min - the minimum value.

→ 25% - The 25% percentile*.

→ 50% - The 50% percentile*.

→ 75% - The 75% percentile*.

→ max - the maximum value.

```
df[listVarQuant].describe()
```

	age	trestbps	chol	thalach	oldpeak
count	183.000000	183.000000	183.000000	183.000000	183.000000
mean	53.748634	129.972678	244.683060	153.256831	0.774317
std	9.586631	16.293473	48.248192	22.768897	0.975776
min	29.000000	94.000000	141.000000	71.000000	0.000000
25%	45.500000	120.000000	209.500000	142.000000	0.000000
50%	54.000000	130.000000	240.000000	158.000000	0.400000
75%	61.000000	140.000000	273.000000	170.000000	1.400000
max	77.000000	180.000000	417.000000	202.000000	4.400000

IV. Sampling(échantillonnage)

Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points to identify patterns and trends in the larger data set being examined. To work with a small, manageable amount of data about a statistical population to build and run analytical models more quickly, while still producing accurate findings.

1) Random sampling

In this example we will choose two samples selected randomly from our dataset called sample1 and sample2

```
sample1=df.sample(n = 100, replace = False)

sample2=df.sample(n = 100, replace = False)
```

2) Variance

The term variance refers to a statistical measurement of the spread between numbers in a data set. More specifically, variance measures how far each number in the set is from the mean (average), and thus from every other number in the set.

For the calculation to be more accurate it is better to separate the variance of the population and the variance of sample

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

S^2	=	sample variance
x_i	=	the value of the one observation
$\bar{x}$	=	the mean value of all observations
n	=	the number of observations

```
def sampleVariance(data):
  n = len(data)
  variance = []
  for feature in listVarQuant :
    mean = stats.mean(data[feature])
    deviations = [(x - mean) ** 2 for x in data[feature]]
    variance.append(abs(sum(deviations) / n-1))
  return variance
```

$$\sigma^2 = \frac{\sum(xi - \bar{x})^2}{N}$$

σ^2	=	population variance
x_i	=	the value of the one observation
$\bar{x}$	=	the mean value of all observations
N	=	the number of observations

```
def populationVariance(data):
  N = len(data)
  variance = []
  for feature in listVarQuant :
    mean = stats.mean(data[feature])
```

```
    deviations = [(x - mean) ** 2 for x in data[feature]]
    variance.append(sum(deviations) / N)
  return variance
```

```
varianceFrame = pd.DataFrame([populationVariance(df),sampleVariance(sample1),
                              sampleVariance(sample2)],
    columns = ['age','trestbps','chol','thalach','oldpeak'],
    index = ['original dataset','sample1','sample2']
)
varianceFrame
```

	age	trestbps	chol	thalach	oldpeak
original dataset	91.401296	264.026576	2315.167309	515.589776	0.946936
sample1	90.753100	263.981600	2256.969100	512.589900	0.224544
sample2	76.505100	286.021100	2490.863600	423.541100	0.207800

3) Standard deviation

Standard deviation is a statistic that measures the dispersion of a dataset relative to its mean and is calculated as the square root of the variance.

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

σ	=	population standard deviation
N	=	the size of the population
x_i	=	each value from the population
μ	=	the population mean

If the data points are further from the mean, there is a higher deviation within the data set; thus, the more spread out the data, the higher the standard deviation.

If the data behaves in a normal curve, then 68% of the data points will fall within one standard deviation of the average, or mean, data point.

Larger variances cause more data points to fall outside the standard deviation., smaller variances result in more data that is close to average.

The variance calculated above will be employed primarily to take the square root of its value, which indicates the standard deviation of the data.

```
def stddeviation(Variance):
  stddev = []
  for feature in range(len(Variance)) :
    stddev.append(math.sqrt(Variance[feature]))
  return stddev
```

```
stddev0=stddeviation(populationVariance(df))
stddev1=stddeviation(sampleVariance(sample1))
stddev2=stddeviation(sampleVariance(sample2))
stddeviation = pd.DataFrame([stddev0,stddev1,stddev2],
    columns = ['age','trestbps','chol','thalach','oldpeak'],
    index = ['original dataset','sample1','sample2']
)
stddeviation
```

	age	trestbps	chol	thalach	oldpeak
original dataset	9.560402	16.248895	48.116186	22.706602	0.973106
sample1	9.526442	16.247511	47.507569	22.640448	0.473861
sample2	8.746719	16.912158	49.908552	20.580114	0.455851

4) Approximation error

The approximation error in a data value is **the discrepancy between an exact value and some approximation to it**.

$$\delta = \left| \frac{v_A - v_E}{v_E} \right| \cdot 100\%$$

δ	=	percent error
v_A	=	actual value observed
v_E	=	expected value

Using this formula to measure the accuracy with which our sample distribution represents a population by using standard deviation.

```
def approxError(population,sample):
  approxError = []
  sumError = 0
  for i in range(len(sample)) :
    sumError += abs(population[i] - sample[i])/population[i]
    approxError.append(str(round(sumError,7))+'%')
  approxError.append(str(round(sumError/5,7))+'%')
  return approxError
```

```
approxError1 = approxError(stddev0,stddev1)
approxError2 =  approxError(stddev0,stddev2)
approxError = pd.DataFrame([approxError1,approxError2],
    columns = ['age','trestbps','chol','thalach','oldpeak','general error'],
    index = ['sample1','sample2']
)
approxError
```

	age	trestbps	chol	thalach	oldpeak	general error
sample1	0.0035522%	0.0036374%	0.0162863%	0.0191997%	0.5322429%	0.1064486%
sample2	0.0851097%	0.1259287%	0.1631795%	0.2568302%	0.788381%	0.1576762%

After executing our code for multiple times with different random samples sample1 and sample2, we find sample1 with the lowest error approximation possible with `0.1064486%` percentile error, which means it is the best presentation of the general population with a close standard deviation value to the population.

5) Choosing presentation

Save the new data frame sample1 into a csv file and save it into a drive folder 'drive/My Drive/datasetMathAI/' so the data is not overlapped.

```
sample1.to_csv('sample1.csv')

!cp sample1.csv "drive/My Drive/datasetMathAI/"
```

And now we can retrieve it from drive

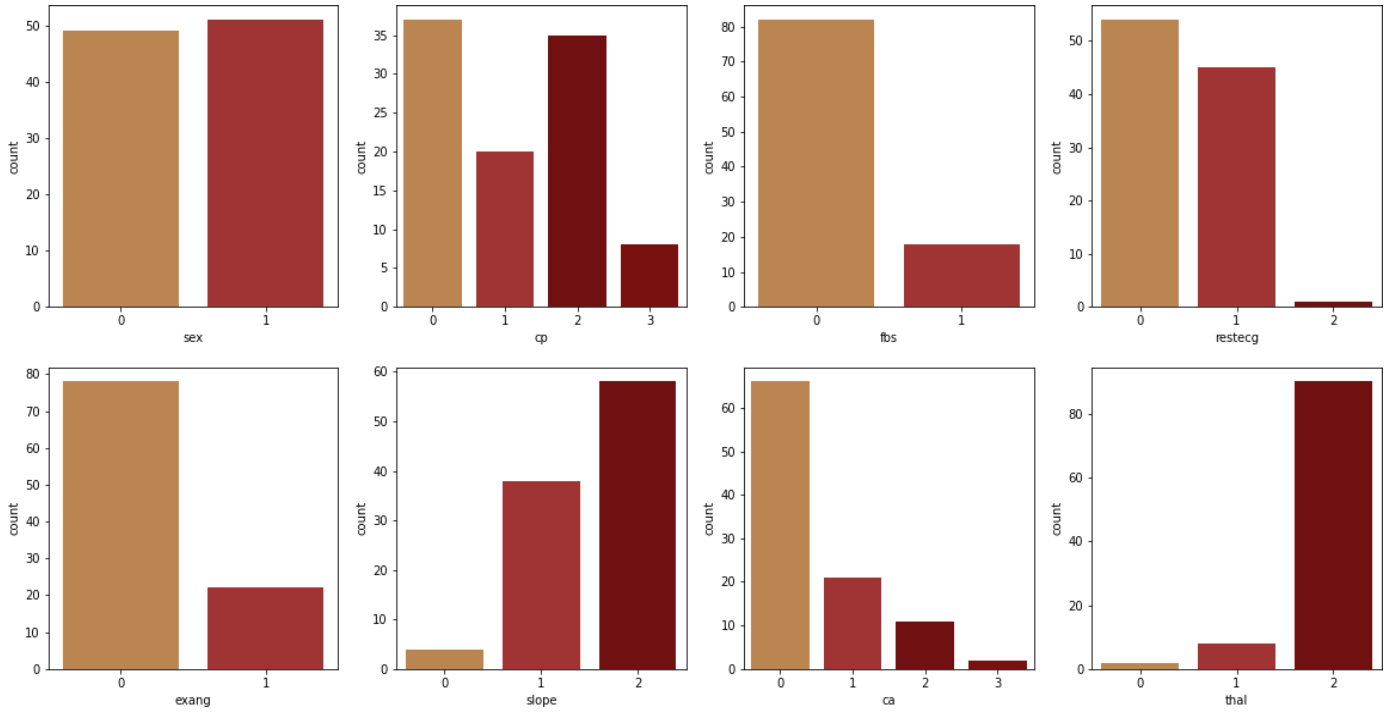
```
sample1=pd.read_csv('/content/drive/My Drive/datasetMathAI/sample1.csv')

sample1=sample1.drop('Unnamed: 0', axis=1)
```

V. Data Visualization

1) countplot

```
import seaborn as sns

import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots(2,4,figsize=(20,10))

l=c=d=0

for l in range(2):

  for c in range(4):

    sns.countplot(ax=ax[l,c],x=listVarQualt[d], data=sample1,palette=['peru',"firebrick","
maroon","darkred"])

    d=d+1
```



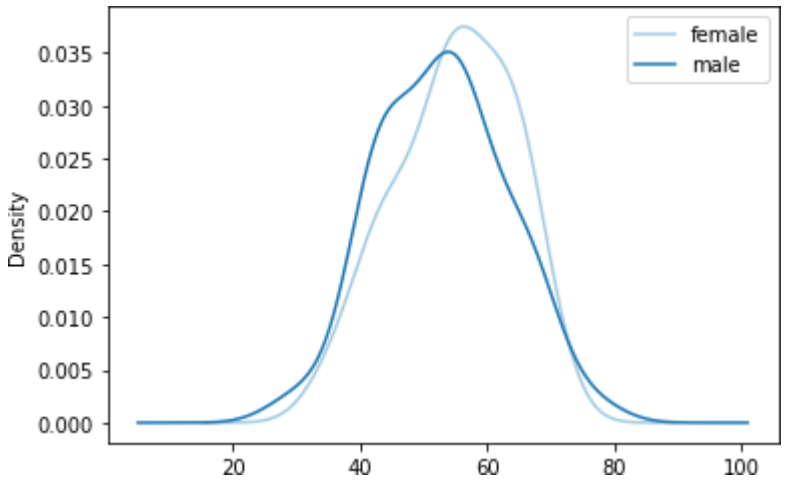
Showing the counts of each observations in each categorical bin** using bars as it is easier to see occurrences of each numerical value.
There's an almost equal spread of both genders so both males and females have an equal chance of a fair study.

- Binning is a technique for reducing the cardinality of continuous and discrete data. Binning groups related values together in bins** to reduce the number of distinct values.

2) plot

Let's plot a more detailed overview of the variation of gender depending on age.

```
sample1.groupby('sex').age.plot(kind='kde')

sample1.legend(['female','male'])
```



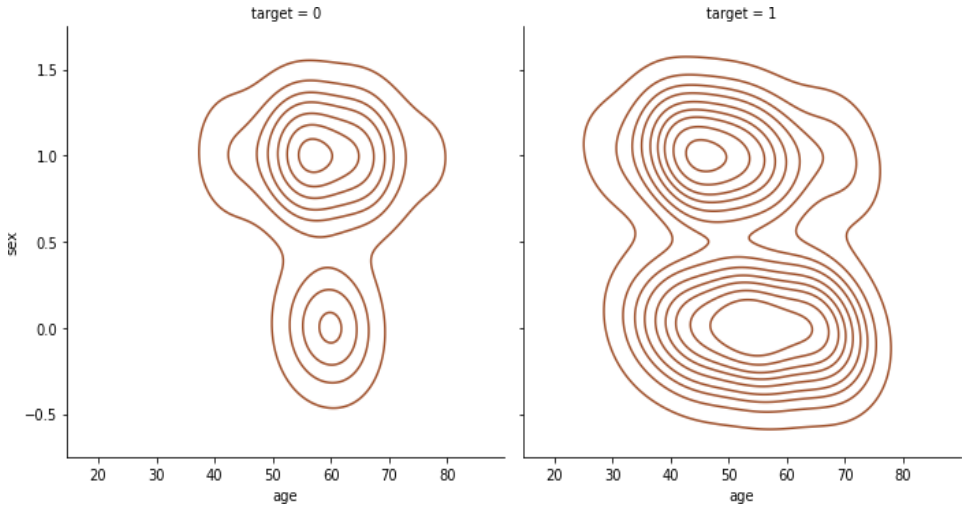
As the last conclusion we had, we see a small variation of density between 30 and 50 with more males and vise-versa for females between 60 and 70.

Then could it be possible that a specific gender is more likely to be diagnosed by a heart disease?

3) displot

Plotting a trio-var plot by age,sex and target.

```
sns.displot(data=sample1, x='age',y='sex',col='target', kind="kde",color='sienna')
```

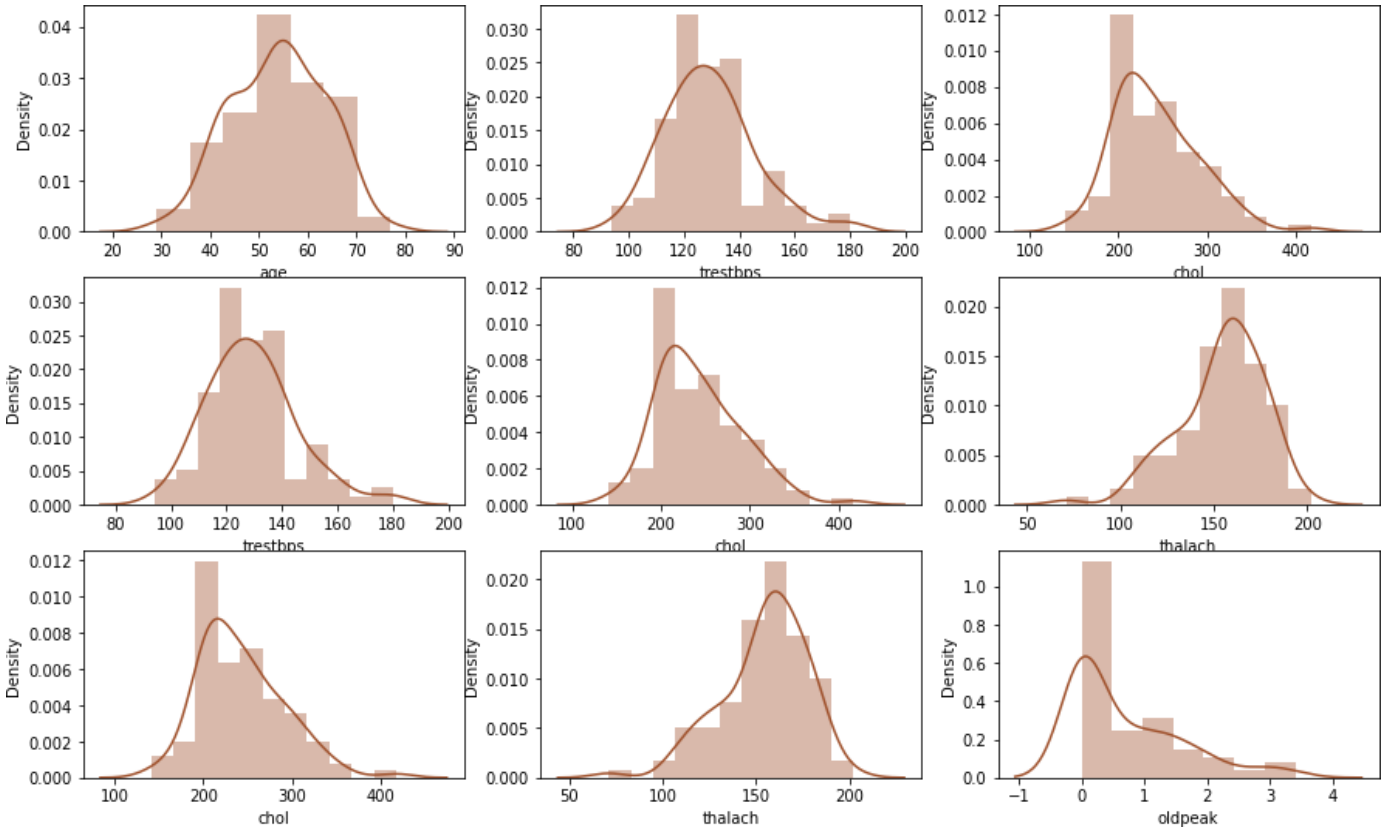


As the plot shows from our data it shows that men are more likely to have a heart disease with more dispersion going from their 30's to 80's opposed to females that have a smaller range and far less heart disease cases.

4) Normal distribution

Distribution of each quantative feature in relation with population

```
rows, cols = 3,3

fig, axarr = plt.subplots(rows, cols, figsize=(cols*5, rows*3))

for row in range(rows):

    for col in range(cols):

        ax = axarr[row, col]

        sns.distplot(sample1[listVarQuant[row+col]], ax=ax,color='sienna')
```



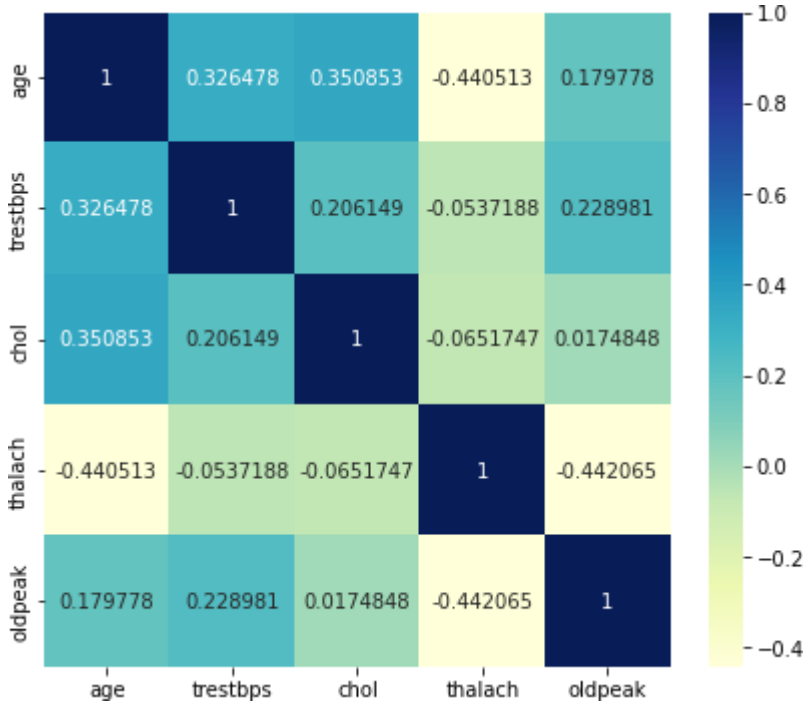
- normal distribution for: age, trestbps and almost for chol
- oldpeak is left-skewed
- thalac is right-skewed

5) The Pearson correlation

The Pearson correlation coefficient (r) is the most common way of measuring a linear correlation. It is a number between –1 and 1 that measures the strength and direction of the relationship between two variables.

Pearson correlation coefficient (r)	**Correlation type**	**Interpretation**
Between 0 and 1	Positive correlation	When one variable changes, the other variable changes in the **same direction**.
0	No correlation	There is **no relationship** between the variables.
Between 0 and –1	Negative correlation	When one variable changes, the other variable changes in the **opposite direction**.

```
#correlation coefficient of Pearson(r)

fig = plt.subplots(figsize=(7,6))

sns.heatmap(sample1[listVarQuant].corr(), annot=True, fmt='g', xticklabels=listVarQuant, yticklabels=listVarQuant, cmap='YlGnBu')

plt.show()
```

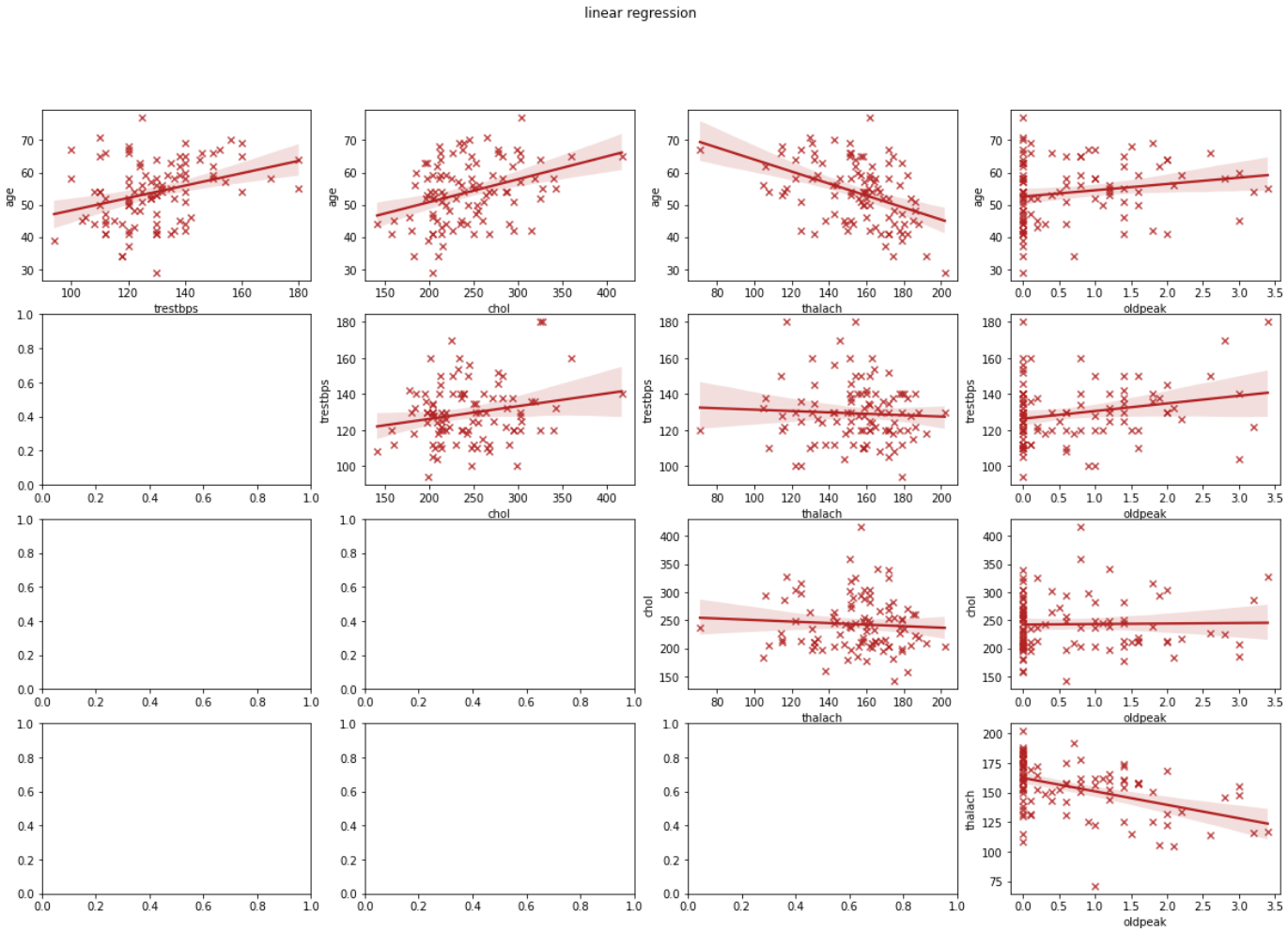


Moderate negative correlation between:

- Thalach and age
- Trestbps and age
- Thalach and oldpeak
- Age and chol

6) Linear regression

```
datareg=sample1

fig, ax = plt.subplots(4,4,figsize=(20,13))

fig.suptitle('linear regression')

sns.regplot(ax=ax[0,0],y='age', x='trestbps', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[0,1],y='age', x='chol', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[0,2],y='age', x='thalach', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[0,3],y='age', x='oldpeak', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[1,1],y='trestbps', x='chol', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[1,2],y='trestbps', x='thalach', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[1,3],y='trestbps', x='oldpeak', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[2,2],y='chol', x='thalach', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[2,3],y='chol', x='oldpeak', data=datareg,color='firebrick',marker='x')

sns.regplot(ax=ax[3,3],y='thalach', x='oldpeak', data=datareg,color='firebrick',marker='x')
```



VX. Anova

Analysis of variance (ANOVA) is a statistical technique that is used to check if the means of two or more groups are significantly different from each other. ANOVA checks the impact of one or more factors by comparing the means of different samples

1) Hypotheses of ANOVA

All hypothesis testing is done under the assumption the null hypothesis is true

Ho: The (population) means of all groups under consideration are equal.

Ha: The (pop.) means are not all equal.

2) Assumptions of ANOVA

Like so many of our inference procedures, ANOVA has some underlying assumptions which should be in place in order to make the results of calculations completely trustworthy. They include:

(i) Subjects are chosen via a simple random sample.

(ii) Within each group/population, the response variable is normally distributed.

(iii) While the population means may be different from one group to the next, the population standard deviation is the same for all groups.

3) The source (of variability) column

Source	SS	df	MS	F
Model/Group	SSG	$k-1$	$\text{MSG} = \frac{SSG}{k-1}$	$\frac{MSG}{MSE}$
Residual/Error	SSE	$n-k$	$\text{MSE} = \frac{SSE}{n-k}$	
Total	SST	$n-1$		

• ***SS=Sum of Squares (sum of squared deviations):***

SST measures variation of the data around the overall mean $\bar{x}$

SSG measures variation of the group means around the overall mean

SSE measures the variation of each observation around its group mean $\bar{x}_i$

• ***Degrees of freedom***

k – 1 for SSG, since it measures the variation of the k group means about the overall mean

n – k for SSE, since it measures the variation of the n observations about k group means

n – 1 for SST, since it measures the variation of all n observations about the overall mean

• ***MS = Mean Square = SS/df***

• ***The F statistic = MSG/MSE***

$$F = \frac{MS_B}{MS_W} = \frac{\frac{SS_B}{DoF_B}}{\frac{SS_W}{DoF_W}}$$

```
n = len(sample1)
```

```
k = len(sample1['target'].unique())

samplesize_target0 = len(sample1[sample1.target == 0]['age'])

samplesize_target1 = len(sample1[sample1.target == 1]['age'])

# Calculate Means

mean_target0 = sample1[sample1.target == 0]['age'].mean()

mean_target1 = sample1[sample1.target == 1]['age'].mean()

mean_overall = sample1['age'].mean()

# Calculate Variances

var_target0 = sample1[sample1.target == 0]['age'].var()

var_target1 = sample1[sample1.target == 1]['age'].var()

# the sum of the squared deviations of the sample averages from the overall average, weighted by the size
 of the samples

SSG = samplesize_target0 * (mean_overall - mean_target0)**2 + \

      samplesize_target1 *(mean_overall - mean_target1)**2

#the sum of the squared deviations within samples from the sample averages

SSE = (samplesize_target0 - 1) * var_target0 + \

      (samplesize_target1 - 1) * var_target1

#total squared deviation (from the average)

SST = SSE + SSG

# Degrees of Freedom

dof_SSG = k - 1

dof_SSE = n - k

dof_SST = n - 1

#Mean square

MSG = (SSG/dof_SSG)

MSE = (SSE/dof_SSE)

# F Stastics

f_stat = MSG / MSE

tableAnova_agetarget = pd.DataFrame([[round(SSG,4),dof_SSG,round(MSG,4),f_stat],[SSE,dof_SSE,MSE,''],[SST
,n-1,'',''']],

    columns = ['SS','df','MS','f-statistic'],

    index = ['Group','Error','Total']

)

tableAnova_agetarget
```

	SS	df	MS	f-statistic
Group	870.020000	1	870.02	10.265982
Error	8305.290021	98	84.747857	
Total	9175.310000	99		

To have a more accurate p-value it is better to use automated calculation than an F-table of critical values

- Enter values for degrees of freedom (v_1 and v_2).
- Enter a value for one, and only one, of the other textboxes.
- Click **Calculate** to compute a value for the last textbox.

Degrees of freedom (v_1)	1
Degrees of freedom (v_2)	98
f Statistic (f)	10.265982
Probability: P(F≤10.265982)	0.99817
Probability: P(F≥10.265982)	0.00183

Calculate

P-value is 0.00183($p < 0.05$);means we can reject the null hypothesis Ho.

X. Pca

Published in 1901 by Karl Pearson, principal component analysis (PCA) is a multivariate approach for studying the multicollinearity of a set of distinct continuous or ordinal variables $\mathbf{X} = (X^1, \cdots, X^p)$

The objective of PCA is to synthesize information by reducing the number of dimensions in order to have a visual and simple reading of the interactions occurring between the various variables of the dataset.

In the case of continuous variables, we will favor its centered-reduced (or normalized) version which has the immense advantage of delimiting the projection inside the trigonometric circle.

Steps:

First we should standardize the columns that we want to work with

It is important to center and reduce our variables to reduce the scale effect because they are not calculated on the same basis. from which we will calculate eigenvalues and eigenvectors.

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

```
cols_to_scale=['age','trestbps','chol','thalach','oldpeak']
```

```
from sklearn.preprocessing import StandardScaler
scaler_data=StandardScaler().fit_transform(df[cols_to_scale])
```

```
scaler_data
```

```
array([[ 0.85158732,  0.49721821,  1.0307007 , -2.02734839,  1.10174021],
       [ 0.43417305, -1.84662083,  0.07721504, -1.33302934,  0.19990919],
       [ 0.43417305, -0.98310118,  1.52817148, -0.55192042,  3.60682636],
       ...,
       [ 0.53852661,  0.62057815, -0.48243958,  0.48955815, -0.80212527],
       [-0.71371622, -1.22982108,  0.63686967, -1.5066091 ,  0.19990919],
       [-0.40065551, -1.22982108,  0.20158274,  0.27258345, -0.80212527]])
```

Second, the correlation coefficient matrix

$$R = \frac{1}{2}.Z.Z'$$

```
cov_matrix=np.matmul(scaler_data.T,scaler_data)
cov_matrix
```

```
array([[ 604.        ,  165.14874392,  147.3625425 , -284.53538424,
         114.4105589 ],
       [ 165.14874392,  604.        ,  103.87317566,  -40.97983571,
          93.59932444],
       [ 147.3625425 ,  103.87317566,  604.        ,    7.57254023,
          13.48017191],
       [-284.53538424,  -40.97983571,    7.57254023,  604.        ,
        -201.09755039],
       [ 114.4105589 ,   93.59932444,   13.48017191, -201.09755039,
         604.        ]])
```

Next, to determine the principal components of variables, we have to find eigen values and eigen vectors for the same. Let R be the quare matrix. A non-zero vector v is an eigenvector of Z if

$Z \lambda = \lambda V$

The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a PCA : The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

```
from scipy.linalg import eigh
values, vectors=eigh(cov_matrix)
values
```

```
array([ 255.10068972,  457.74420209,  516.07507744,  684.52492693,
       1106.55510382])
```

```
vectors
```

```
array([[-0.58889262, -0.65593162, -0.08647058, -0.33999061, -0.31605128],
       [-0.36688982,  0.22619603, -0.43074563,  0.68300924, -0.40272049],
       [-0.25183408,  0.2323679 , -0.6949874 , -0.33414638,  0.53658501],
       [ 0.52773389, -0.64328473, -0.44057329,  0.30419767,  0.14505393],
       [-0.42032591, -0.22534191,  0.36037305,  0.46229291,  0.65495319]])
```

Identify principal components

```
#prendre 3 variables importante depuis les 5
pca=PCA(n_components=3)
pca.fit(scaler_data)
pca_data=pca.transform(scaler_data)
```

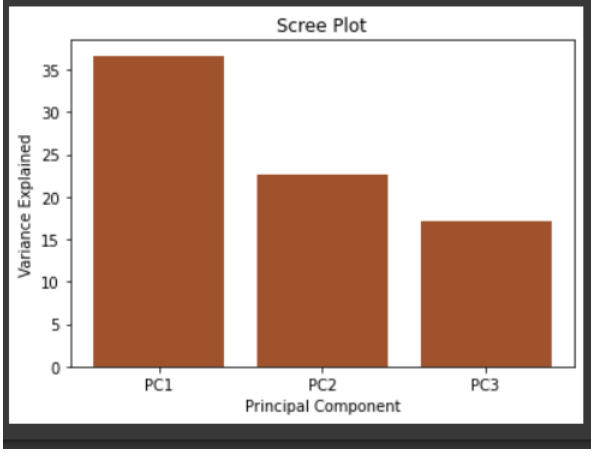
```
pca_data
```

```
array([[ 2.47647377, -0.28609722, -0.40171994],
       [ 0.38513207, -1.36355619, -1.74776279],
       [ 2.08714684, -0.86682783,  0.16980243],
       ...,
       [-0.17218704,  0.48333918,  0.18007768],
       [ 0.16799014, -0.88465274, -1.17602042],
       [-1.11739207, -0.0151302 , -1.05901577]])
```

```
#features importance
per_var=np.round(pca.explained_variance_ratio_*100, decimals=2)
per_var
```

```
array([36.64, 22.67, 17.09])
```

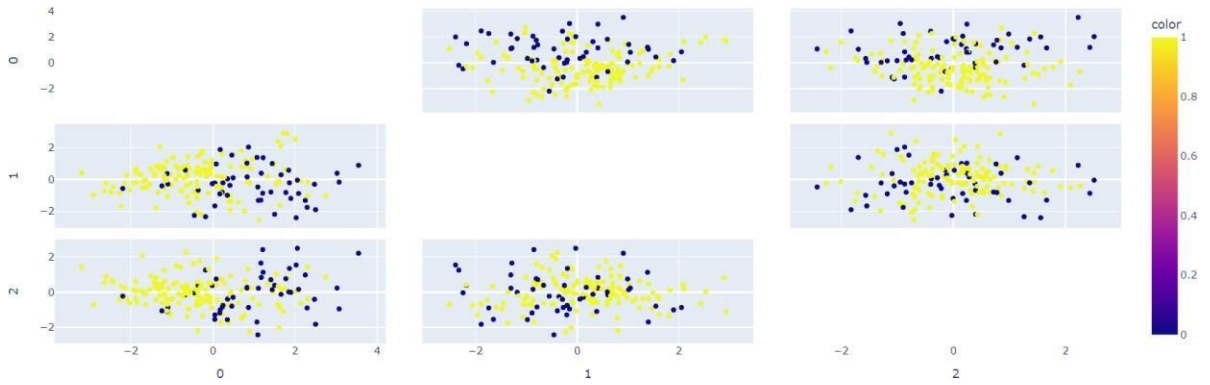
```
labels=['PC'+str(x) for x in range(1,len(per_var)+1)]
plt.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=labels,color='sienna' )
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.show()
```



This PCs conserve about 76.4% of informations

Scatter matrix records the relationship between variables, which is important to find a dimension of maximum variance.

```
import plotly.express as px

#df = px.data.df()
fig = px.scatter_matrix(
    pca_data,
    labels=per_var,
    dimensions=range(3),
    color=df["target"]
)
fig.update_traces(diagonal_visible=False)
fig.show()
```




```
pcs = pd.DataFrame(list(zip(pca_data[:,0], pca_data[:,1], pca_data[:,2], df['target'])),columns =['pc1', 'pc2', 'pc3', 'target'])
fig = px.scatter_3d(pcs, x='pc1', y='pc2', z='pc3',color='target')
fig.show()
```