DSTI

S21: Applied MSc in Data Science & Artificial Intelligence

Python Lab Project: Stroke Prediction Model

Instructor: Hanna Abi Akl

Student: Constant Patrice A. Kodja Adjovi

Period: Mars 2024 The goal of this project is to predict whether a patient is likely to get as **stroke** based on 10 input parameters: **gender, age, hypertension, heart disease, ever married, work type, residence_type, average glucose level, body mass index** and **smoking status**.

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import recall_score, precision_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import make_pipeline
from sklearn.feature_selection import SelectKBest,chi2,f_classif
from scipy.stats import chi2_contingency
import statsmodels.api
from imblearn.over_sampling import SMOTE
```

```
raw_data = pd.read_csv("datasets/stroke_data.csv",sep=",",index_col="id")
```

```
df_PrePro = raw_data.copy()  # Copy for data preprocessing
```

Data Exploration Analysis

```
df_PrePro.head()
```

```
       gender   age  hypertension  heart_disease ever_married      work_type  \
id
9046     Male  67.0             0              1          Yes        Private
51676  Female  61.0             0              0          Yes  Self-employed
31112    Male  80.0             0              1          Yes        Private
```

```
60182  Female  49.0             0              0          Yes        Private
1665   Female  79.0             1              0          Yes  Self-employed

      Residence_type  avg_glucose_level   bmi   smoking_status  stroke
id
9046           Urban             228.69  36.6  formerly smoked       1
51676          Rural             202.21   NaN     never smoked       1
31112          Rural             105.92  32.5     never smoked       1
60182          Urban             171.23  34.4           smokes       1
1665           Rural             174.12  24.0     never smoked       1
```

```
df_PrePro.tail()
```

```
       gender   age  hypertension  heart_disease ever_married      work_type  \
id
18234  Female  80.0             1              0          Yes        Private
44873  Female  81.0             0              0          Yes  Self-employed
19723  Female  35.0             0              0          Yes  Self-employed
37544    Male  51.0             0              0          Yes        Private
44679  Female  44.0             0              0          Yes       Govt_job

      Residence_type  avg_glucose_level   bmi   smoking_status  stroke
id
18234          Urban              83.75   NaN     never smoked       0
44873          Urban             125.20  40.0     never smoked       0
19723          Rural              82.99  30.6     never smoked       0
37544          Rural             166.29  25.6  formerly smoked       0
44679          Urban              85.28  26.2          Unknown       0
```

The dataset has the two types of variables: the **numerical** and the **categorical**.
The categorical variables are from **ordinal** and **nominal type**.

I used the observations ID is used as rows identifier.

```
df_PrePro.describe()
```

```
                age  hypertension  heart_disease  avg_glucose_level  \
count  5110.000000   5110.000000    5110.000000        5110.000000
mean     43.226614      0.097456       0.054012         106.147677
std      22.612647      0.296607       0.226063          45.283560
min       0.080000      0.000000       0.000000          55.120000
25%      25.000000      0.000000       0.000000          77.245000
50%      45.000000      0.000000       0.000000          91.885000
75%      61.000000      0.000000       0.000000         114.090000
max      82.000000      1.000000       1.000000         271.740000

               bmi       stroke
count  4909.000000  5110.000000
mean     28.893237     0.048728
```

```
std       7.854067     0.215320
min      10.300000     0.000000
25%      23.500000     0.000000
50%      28.100000     0.000000
75%      33.100000     0.000000
max      97.600000     1.000000
```

The dataset has one response variable **stroke** which is a categorical and 10 explanatory variables as features. Based on the above structure there are 7 **categorical features** and 3 **numerical features**.

Within the categorical features there are 5 nominal variables that's why they are not displayed in the above describe table.

Missing data: The feature **bmi** has missing values (NaN); that's why it does not have a total of 5110 observations. The feature structure according to the "stroke" will be checked in order to see if the observations or row liked to those NaN should deleted or imputed with a kind of 'bmi' mean.

Bias hypothesis: The dataset seems unbalanced by comparing the means and the maximum of features: Stroke, hypertension, heart-disease and bmi have their means far from O.5. This is due to the size of the small observations (less than 10,000). This will likely push the model to predict between the large size values.

Outlier hypothesis: Numerical features like **BMI** and **Average Glucose Level** seem to have outliers. Their **means** are close to the **min value** than to the **max**. With this the means would seem incorrect and this could present **data imbalance** for the model which will likely prefer predict more for one side than the other. **age** feature looks more balanced.

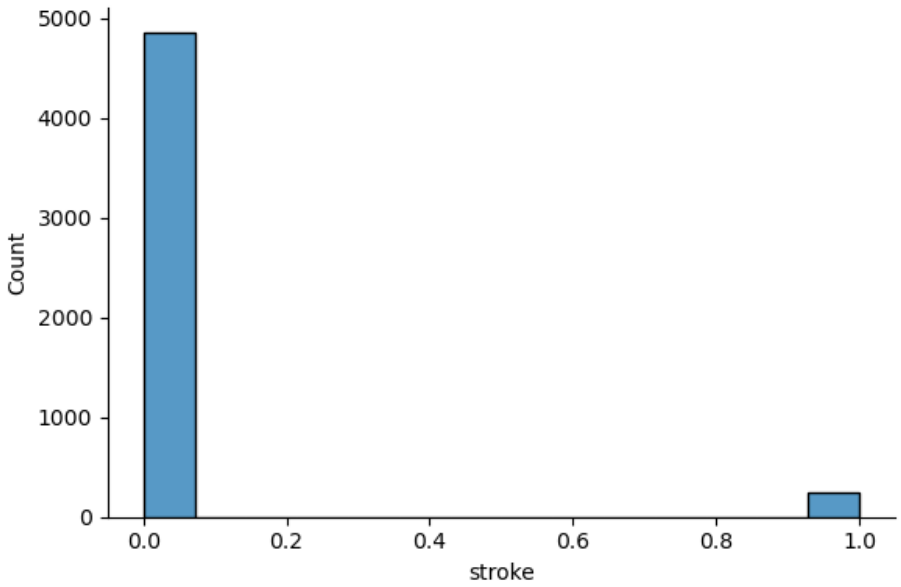
Target variable "stroke" data analysis

```
df_PrePro.stroke.value_counts()
```

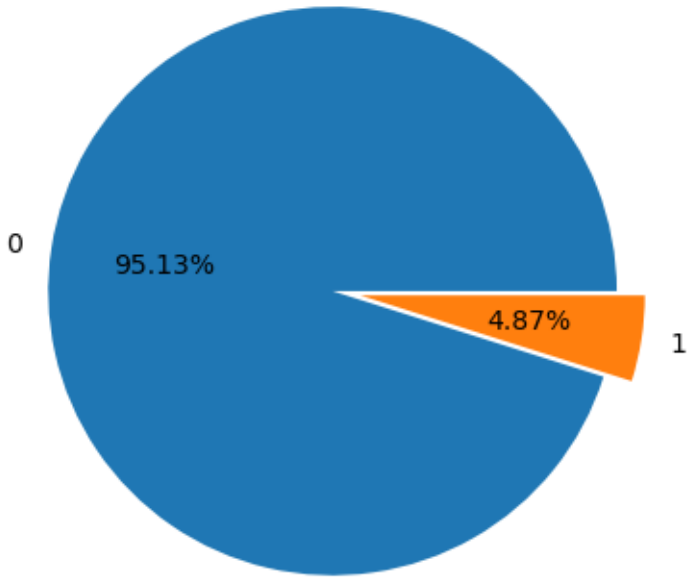
```
stroke
0    4861
1     249
Name: count, dtype: int64
```

```
sns.displot(df_PrePro.stroke,height=4, aspect=1.5)
```

```
<seaborn.axisgrid.FacetGrid at 0x213a7bd02b0>
```




```
plt.pie(df_PrePro.stroke.value_counts(),labels=df_PrePro.stroke.value_counts().index,autopct
plt.show()
```



The response variable **stroke** is hugely unbalanced and will really push the model to more predict unstroke cases.

Explanatory variables data analysis

Gender data analysis The study text indicates that there is **Other** as gender. We will check the number of observations concerned

```
df_PrePro.gender.value_counts()
```

```
gender
Female    2994
Male      2115
Other        1
Name: count, dtype: int64
```

```
#As only one within 5110 observations which is concerned we cannot remove it without a negat

indexOther = df_PrePro[(df_PrePro['gender'] == "Other")].index
df_PrePro.drop(indexOther , inplace=True)
```

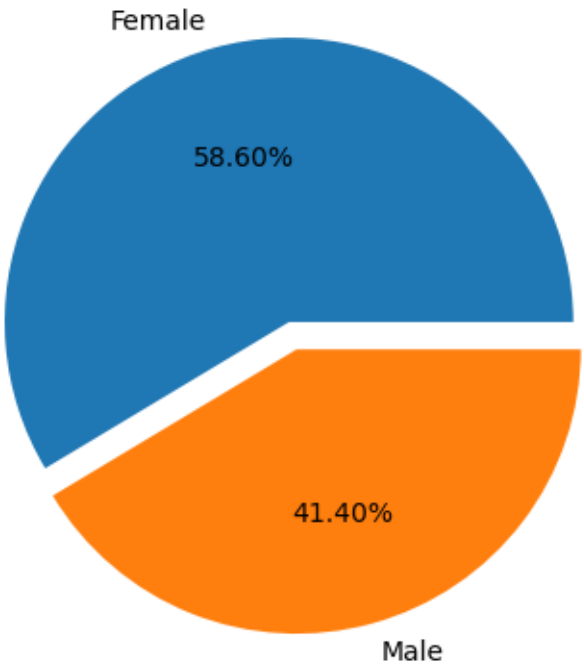
```
df_PrePro.gender.describe()
```

```
count       5109
unique         2
top       Female
freq        2994
Name: gender, dtype: object
```

```
df_PrePro.gender.value_counts()
```

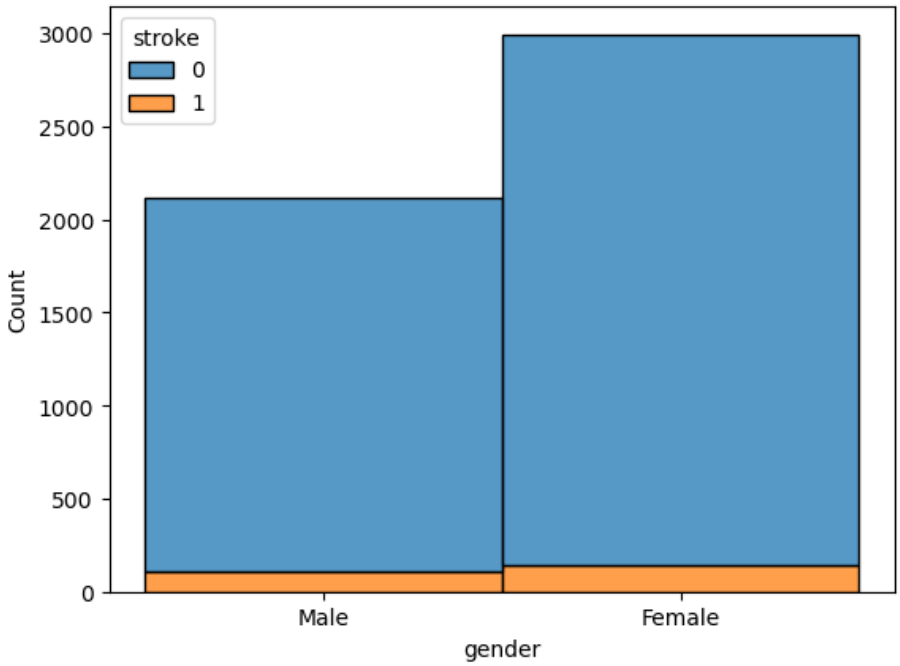
```
gender
Female    2994
Male      2115
Name: count, dtype: int64
```

```
plt.pie(df_PrePro.gender.value_counts(),labels=df_PrePro.gender.value_counts().index,autopct
plt.show()
```




```
sns.histplot(x=('gender'),hue=('stroke'),multiple="stack",data=df_PrePro)
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
((df_PrePro.loc[:,["stroke","gender"]]).value_counts())/len(df_PrePro)*100
```

```
stroke  gender
0       Female    55.842631
        Male      39.283617
1       Female     2.759836
        Male       2.113917
Name: count, dtype: float64
```

Females seem to slightly have more stroke than males: **2.76% for Female** and **2.11% for Male**

BMI data analysis

```
df_PrePro.isna().sum()
```

```
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
```

```
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

```
#There is a confimation of 4% of missing value inside the column bmi  (201).
print(round(201/5110*100),'%')
```

```
4 %
```

```
df_PrePro.loc[df_PrePro.bmi.isna(),["stroke","age","bmi"]].groupby("stroke").count()
```

```
        age  bmi
stroke
0       161    0
1        40    0
```

BMI has 40 missing values within the 249 stroked people so we won't delete them. We notice above that within the 201 bmi missing values we have 161 for unstroked observations and 40 for the other. We could replace NaN values by the bmi feature average but we will try to do it according to stroke groups means. So the 161 will be imputed with the unstroked mean and stroke one for the 40.

```
indexNaUnstroke= df_PrePro.loc[df_PrePro.bmi.isna(),:][(df_PrePro['stroke'] == 0)].index
indexNaStroke= df_PrePro.loc[df_PrePro.bmi.isna(),:][(df_PrePro['stroke'] == 1)].index
print(len(indexNaStroke),'and',len(indexNaUnstroke))
```

```
40 and 161
```

```
C:\Users\P.Kodja\AppData\Local\Temp\ipykernel_33700\2926388912.py:1: UserWarning: Boolean Se
  indexNaUnstroke= df_PrePro.loc[df_PrePro.bmi.isna(),:][(df_PrePro['stroke'] == 0)].index
C:\Users\P.Kodja\AppData\Local\Temp\ipykernel_33700\2926388912.py:2: UserWarning: Boolean Se
  indexNaStroke= df_PrePro.loc[df_PrePro.bmi.isna(),:][(df_PrePro['stroke'] == 1)].index
```

```
bmiUnstrokeMean=round(df_PrePro.loc[(df_PrePro['stroke'] == 0)].bmi.mean(),1)
bmiStrokeMean=round(df_PrePro.loc[(df_PrePro['stroke'] == 1)].bmi.mean(),1)
print(bmiStrokeMean,'and',bmiUnstrokeMean)
```

```
30.5 and 28.8
```

```
df_PrePro.loc[indexNaUnstroke,"bmi"] = bmiUnstrokeMean
df_PrePro.loc[indexNaStroke,"bmi"] = bmiStrokeMean
```

```
#df_PrePro.bmi.isna().index
df_PrePro.isna().sum()
```

```
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
```

```
Residence_type       0
avg_glucose_level    0
bmi                  0
smoking_status       0
stroke               0
dtype: int64
```

```
df_PrePro.describe()
```

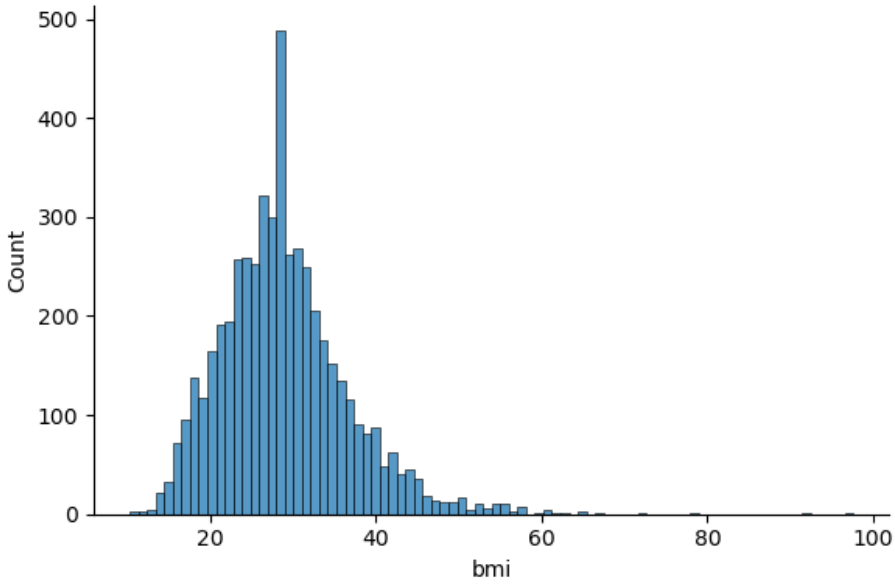
```
                age  hypertension  heart_disease  avg_glucose_level  \
count  5109.000000   5109.000000    5109.000000        5109.000000
mean     43.229986      0.097475       0.054022         106.140399
std      22.613575      0.296633       0.226084          45.285004
min       0.080000      0.000000       0.000000          55.120000
25%      25.000000      0.000000       0.000000          77.240000
50%      45.000000      0.000000       0.000000          91.880000
75%      61.000000      0.000000       0.000000         114.090000
max      82.000000      1.000000       1.000000         271.740000

                bmi       stroke
count  5109.000000  5109.000000
mean     28.904150     0.048738
std       7.699558     0.215340
min      10.300000     0.000000
25%      23.800000     0.000000
50%      28.400000     0.000000
75%      32.800000     0.000000
max      97.600000     1.000000
```

There is not a significant difference between **bmi** means before (**28.89456**) and after (**28.904150**)

Numeric features outliers checking

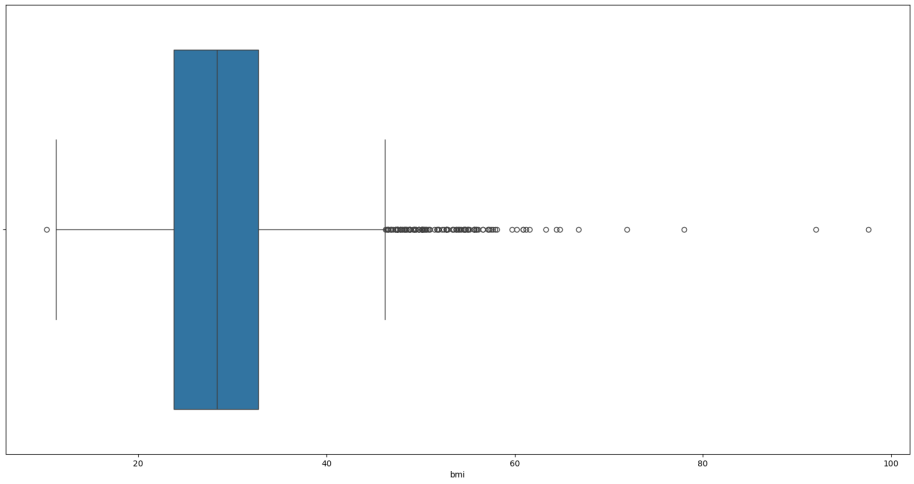
```
sns.displot(df_PrePro.bmi,height=4, aspect=1.5)
```

```
<seaborn.axisgrid.FacetGrid at 0x213aaa2e4d0>
```




```
plt.figure(figsize=(20,10))
sns.boxplot(x=df_PrePro.bmi)
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\categorical.py:632: F
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```



BMI feature seems to have **outlier values** after 80. This has negative impact on the mean and the median use would be better. As the the outliers frequencies are low, the model is likely to not predict the related observation in that range based on their limited number. This introduces data **imbalance** which could be

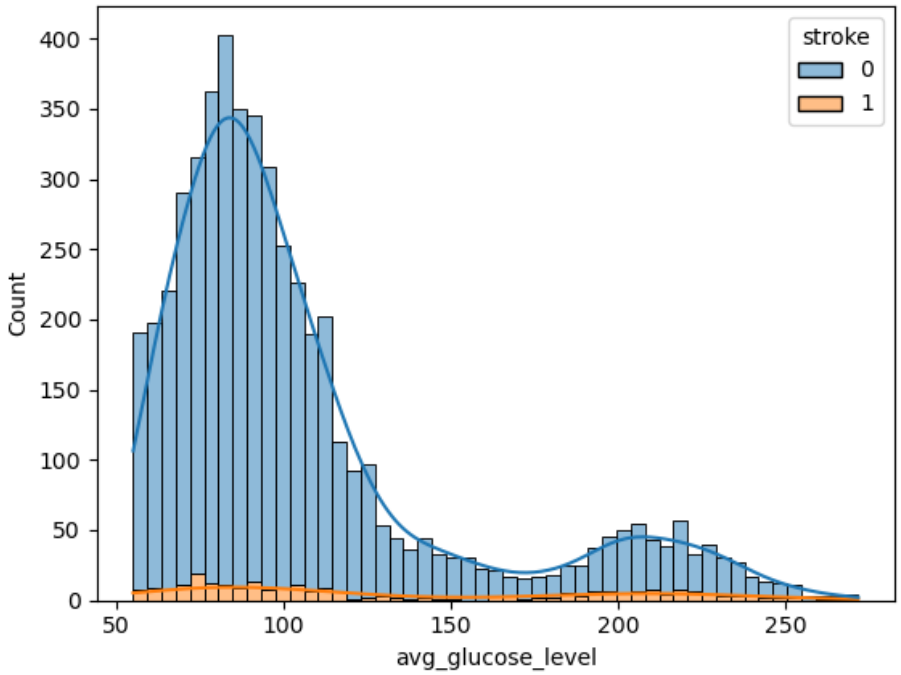
resolved with more observations that we do not have. Outliers could also be cut to ajust data range but cautious must be taken and further anakyses must be conducted first.

---

avg_glucose_level data analysis

```
#sns.displot(df_PrePro.avg_glucose_level,height=4, aspect=1.5)
sns.histplot(x=('avg_glucose_level'),hue=('stroke'),multiple="stack",kde=True,data=df_PrePro
plt.savefig("images/avg_glucose_level.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
df_PrePro.avg_glucose_level.describe()
```

```
count    5109.000000
mean      106.140399
std        45.285004
```

```
min         55.120000
25%         77.240000
50%         91.880000
75%        114.090000
max        271.740000
Name: avg_glucose_level, dtype: float64
```

Above figures confirm the graph results. The **median (91.88)** is inferier to the **mean (106.14)** proving that the data has **right skewness** indicating outliers presence as the median is close to the **min 55.12** and far from the **max 271.74**. This could be the consequence of the small data size (number of observations). At this stage, data is unbalanced and will impact negatively the model training and prediction.

Besides, the feature seems to be a **bimodal** column even though the second mode frequence is less than 100, the model can take it into account during the training steps and the prediction of typical observation (typically concentrated around the mean or the median) will be negatively impacted if the model is accustomed to outliers.

Based on the feature structure and limited data size, it would be better to choose adequate statistic model which handles outliers.

The stroke seems following the same trend like the feature data.

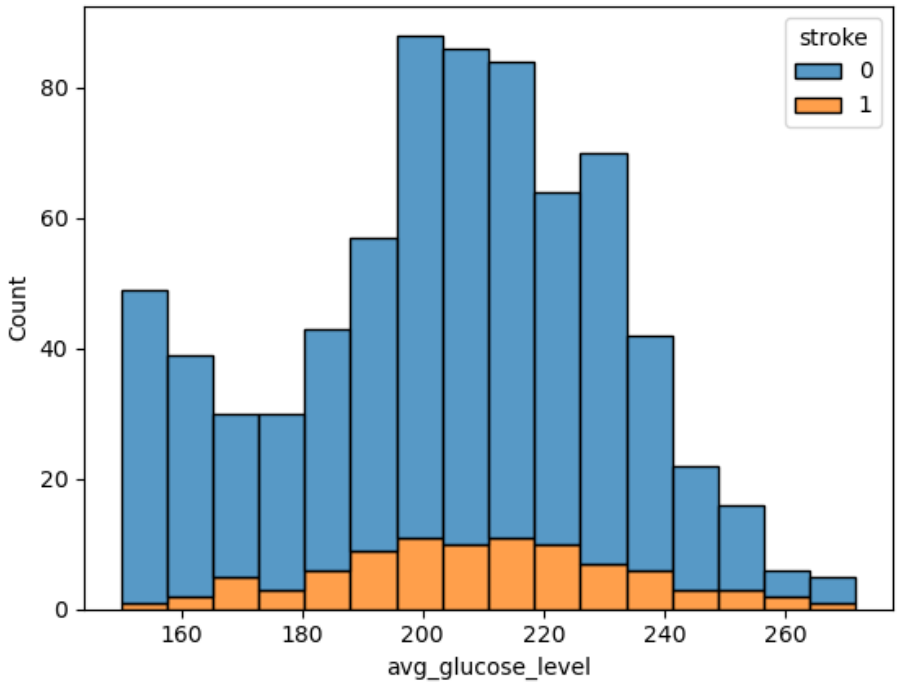
```
print("# avg glucose level for fasting blood in [70, 100] (normal case) :",
      df_PrePro.loc[((df_PrePro.avg_glucose_level > 70) & (df_PrePro.avg_glucose_level < 100
print("# Maximum of avg glucose level for all must be < 125 (over all or general case): ",
      df_PrePro.loc[df_PrePro.avg_glucose_level <= 125,:].avg_glucose_level.count())
print("# avg glucose level > 125 (Consider dangerous) :",
      df_PrePro.loc[df_PrePro.avg_glucose_level > 125,:].avg_glucose_level.count())
```

```
# avg glucose level for fasting blood in [70, 100] (normal case) : 2375
# Maximum of avg glucose level for all must be < 125 (over all or general case):  4110
# avg glucose level > 125 (Consider dangerous) : 999
```

The above graph proves the bimodal characteristic of the feature data with the strocke display. We notice a concentration around the first median 91.88 which decreases and seem close to zero around 150 ($<$ 125 the maximum tolorated)

```
sns.histplot(x=('avg_glucose_level'),hue=('stroke'),multiple="stack",data=df_PrePro[df_PrePr
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
df_PrePro[df_PrePro.avg_glucose_level >=150.0].avg_glucose_level.describe()
```

```
count    731.000000
mean     203.828577
std       26.774624
min      150.000000
25%      186.495000
50%      205.840000
75%      222.490000
max      271.740000
Name: avg_glucose_level, dtype: float64
```

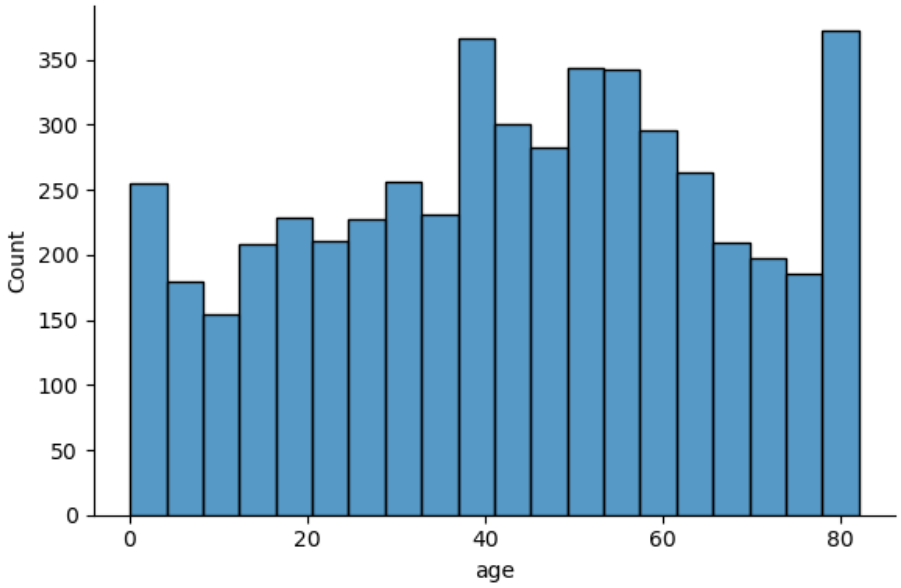
This part of date seems having almost normal distribution charactéristic could be separated from the main part into another explanatory variable. But these values are extremes ones (>125) and their number is not sufficient enough to be a separate variable.

Maybe one of solution would be to remove those observations in order to a balanced normal data for models training as we cannot provide more observation data.

Age data analysis

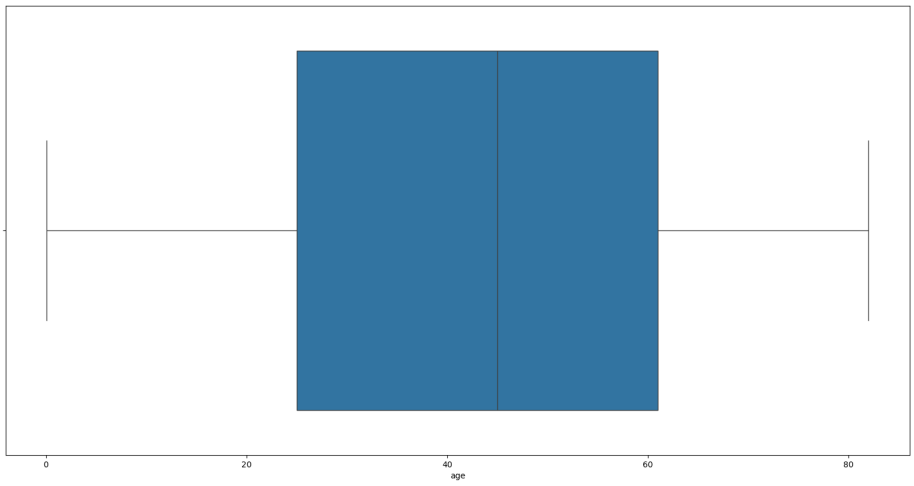
```
sns.displot(df_PrePro.age,height=4, aspect=1.5)
```

```
<seaborn.axisgrid.FacetGrid at 0x213ab021de0>
```




```
plt.figure(figsize=(20,10))
sns.boxplot(x=df_PrePro.age)
plt.show()
```

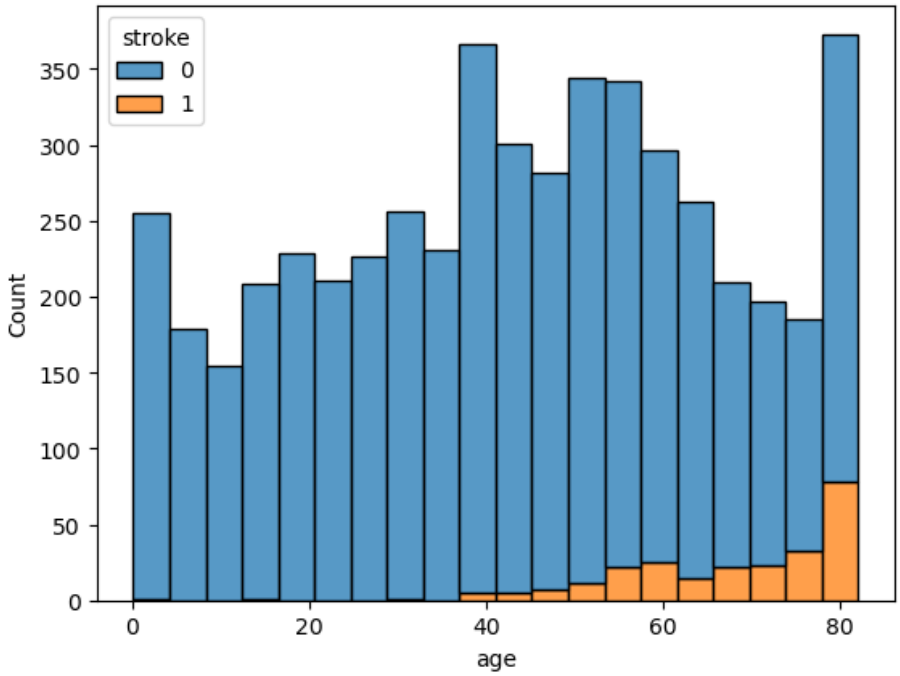
```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\categorical.py:632: F
  positions = grouped.grouper.result_index.to_numpy(dtype=float)
```




```
sns.histplot(x=('age'),hue=('stroke'),multiple="stack",data=df_PrePro)
```

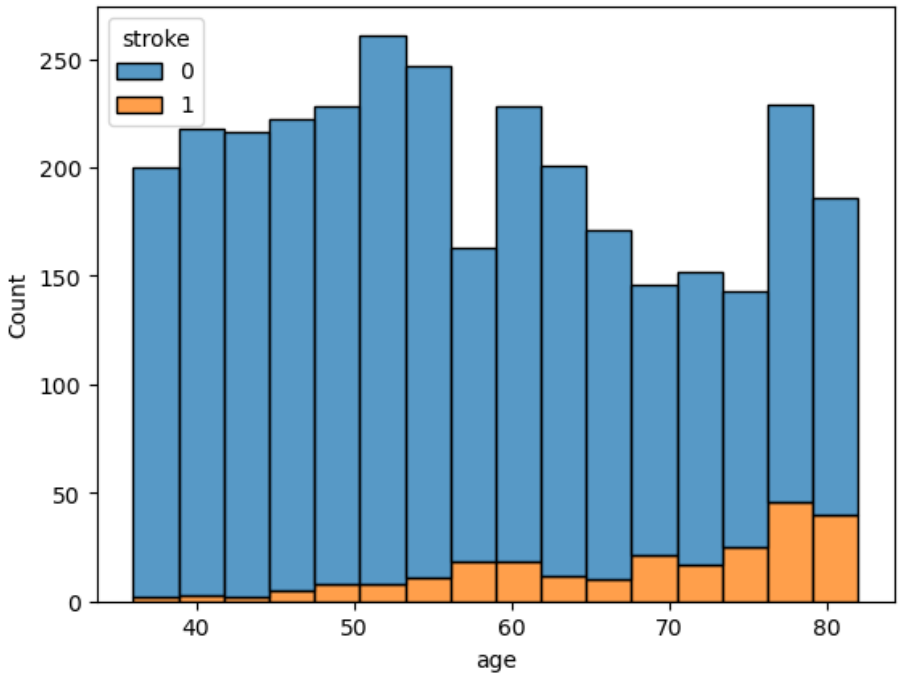
```
plt.savefig("images/age.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
sns.histplot(x=('age'),hue=('stroke'),multiple="stack",data=df_PrePro[df_PrePro.age >=35.1])
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
df_PrePro.age[df_PrePro.age >=35.1].min(), df_PrePro.age[df_PrePro.age >=35.1].max()
```

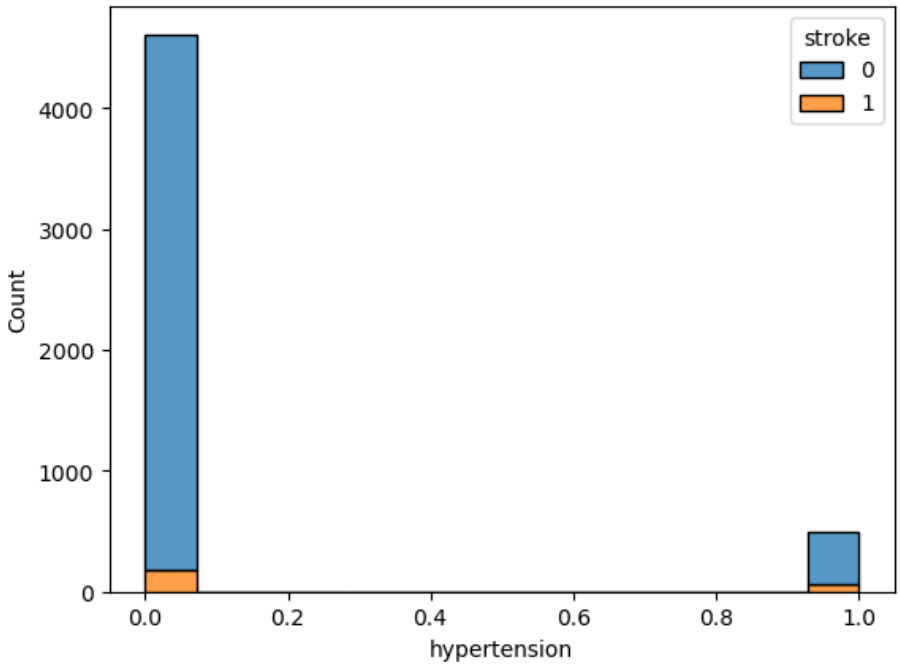
```
(36.0, 82.0)
```

Age feature seems a balanced data without outlier values. The stroke seems to happen between ages 36 to 82 and seem to have increase trend as ages are increasing.

Hypertension data analysis

```
sns.histplot(x=('hypertension'),hue=('stroke'),multiple="stack",data=df_PrePro)
plt.savefig("images/hypertension.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
df_PrePro.hypertension.value_counts()
```

```
hypertension
0    4611
1     498
Name: count, dtype: int64
```

```
((df_PrePro.loc[:,["stroke","hypertension"]]).value_counts())/len(df_PrePro)*100
```

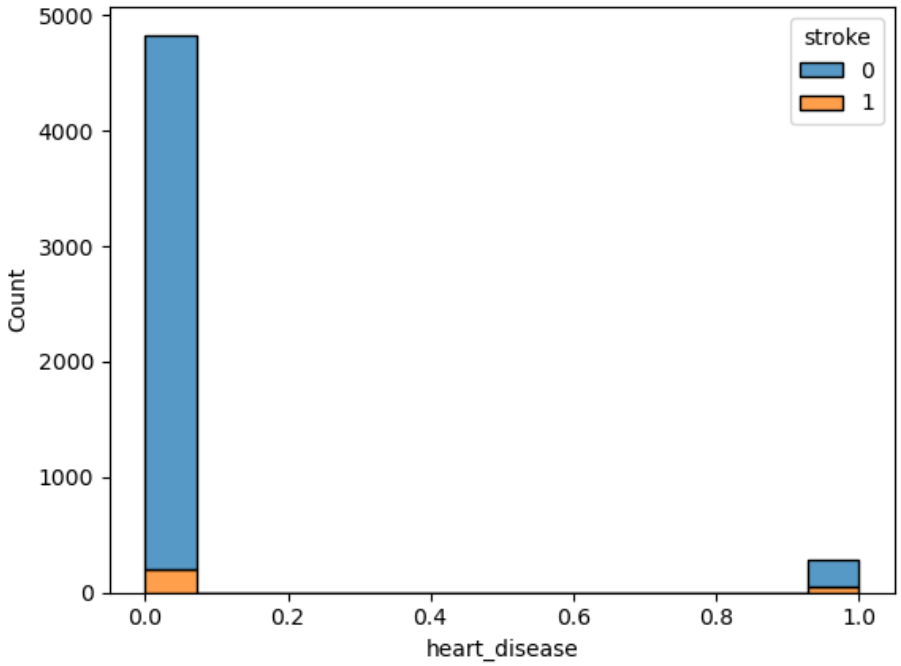
```
stroke  hypertension
0       0               86.670581
        1                8.455666
1       0                3.581914
        1                1.291838
Name: count, dtype: float64
```

Unbalanced data: large number of non-hypertension people than the hypertension people. According to the data, it seems to have **more stroke within non-hypertension people**. The lack of sufficient observations could lead to this as those who have the stroke also have the hypertension disease.

Heart disease data analysis

```
sns.histplot(x=('heart_disease'),hue=('stroke'),multiple="stack",data=df_PrePro)
plt.savefig("images/heart_disease.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```



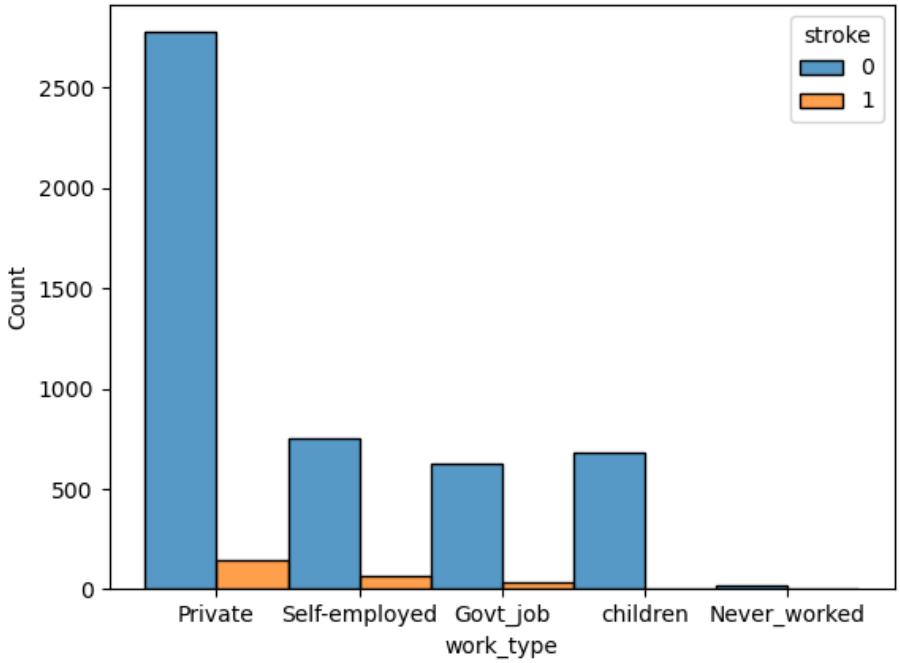
Seems almost the same thing like for hypertension: more strokes happen within people who do not have heart disease. The lack of sufficient observations could lead to this as those who have the stroke also have the heart disease.

Work Type data analysis

```
sns.histplot(x=('work_type'),hue=('stroke'),multiple="dodge",data=df_PrePro)    #multiple="s
plt.savefig("images/work_type.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
```

```
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
(df_PrePro.loc[:,["stroke","work_type"]]).value_counts()
```

```
stroke  work_type
0       Private          2775
        Self-employed     754
        children          685
        Govt_job          624
1       Private           149
        Self-employed      65
        Govt_job           33
0       Never_worked       22
1       children            2
Name: count, dtype: int64
```

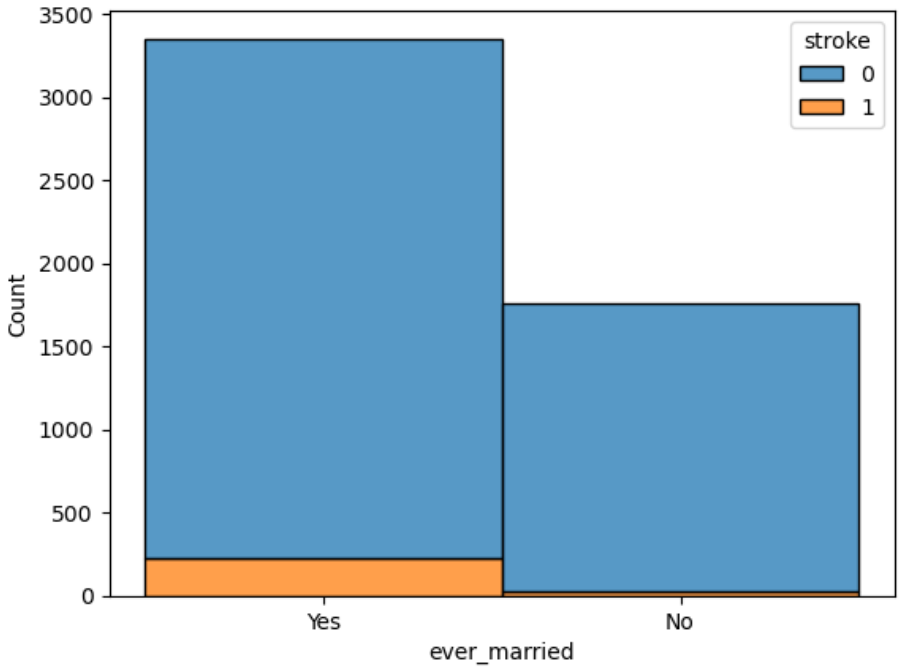
The above result is sorted according to the frequencies. Private, self-employed and Government job people used to have strock with the highest part for the **private sector** and then follow **Self-employed people**. This is proved that those who face more economical challenges used to have more stroke within them Conversely, people working for the government have fewer strokes than the private sector as a whole.

People who have never worked and children do not have strokes or are less likely to experience them. This feature seems to have an impact on the target variable.

Ever Married data analysis

```
sns.histplot(x=('ever_married'),hue=('stroke'),multiple="stack",data=df_PrePro)
plt.savefig("images/ever_married.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
(df_PrePro.loc[:,["stroke","ever_married"]]).value_counts()
```

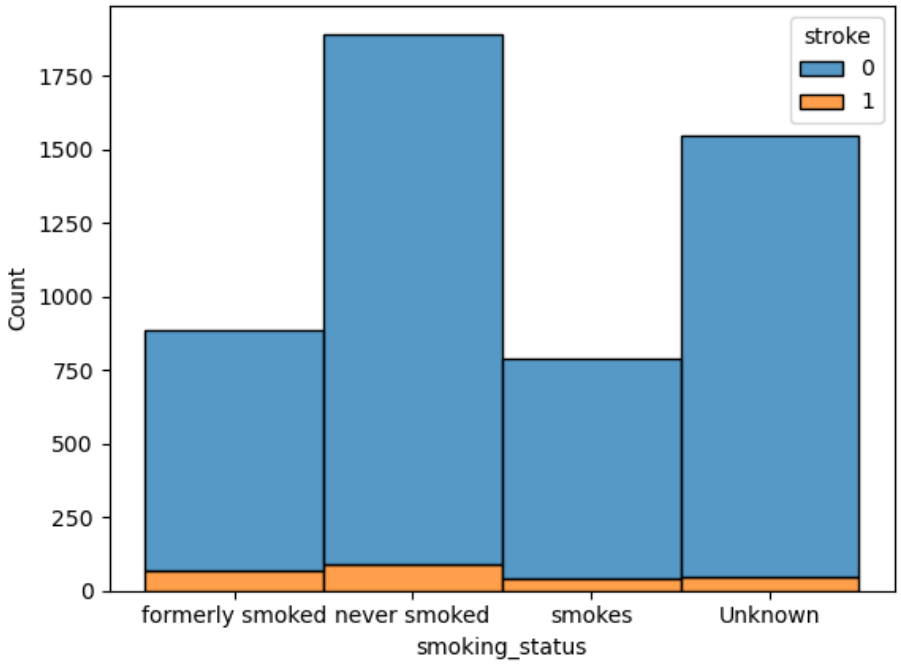
```
stroke  ever_married
0       Yes             3133
        No              1727
1       Yes              220
        No                29
Name: count, dtype: int64
```

Married people seem to more have stroke than the others who have less stroke.
There is data imbalance between the two classes

Smoking status data analysis

```
sns.histplot(x=('smoking_status'),hue=('stroke'),multiple="stack",data=df_PrePro)
plt.savefig("images/smoking_status.png")
plt.show()
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\seaborn\_base.py:948: FutureW
  data_subset = grouped_data.get_group(pd_key)
```




```
(df_PrePro.loc[:,["stroke","smoking_status"]]).value_counts()
```

```
stroke  smoking_status
0       never smoked       1802
        Unknown            1497
        formerly smoked     814
        smokes              747
1       never smoked         90
```

```
        formerly smoked      70
        Unknown              47
        smokes               42
Name: count, dtype: int64
```

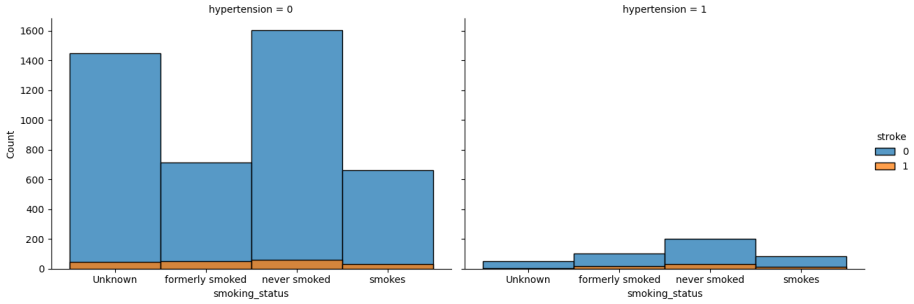
Here people who never smoked are in a high number within the dataset and they are the ones having more stroke than those who have formerly smoked which number is inferior to never smoked people. This could may be caused by the limited number of observation in formerly smoked class. Same for the smoked class which has less stroked people than never smoked class.

Multiple features comparing graphs

Hypertension, smoking_status and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension",hue="stroke", height=4.5, aspect=1.4)
g.map(sns.histplot, "smoking_status")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x213abbb7220>
```

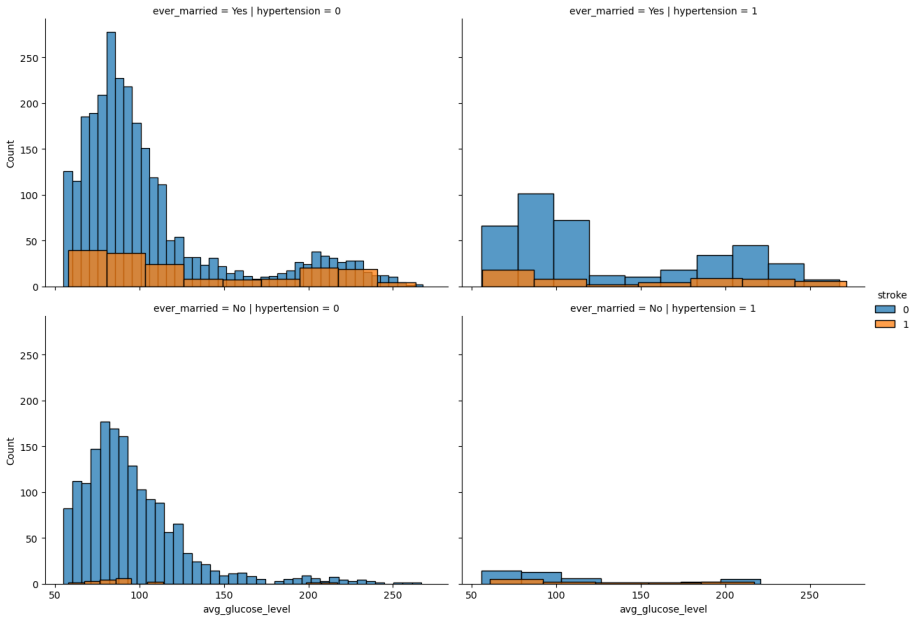


People who have never smoked and those who have stopped smoking face stroke issue and the risk exist if they have hypertension.

Hypertension, avg_glucose_level, ever_married and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension",row="ever_married",hue="stroke", height=4.5,
g.map(sns.histplot, "avg_glucose_level")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x213ac47d180>
```



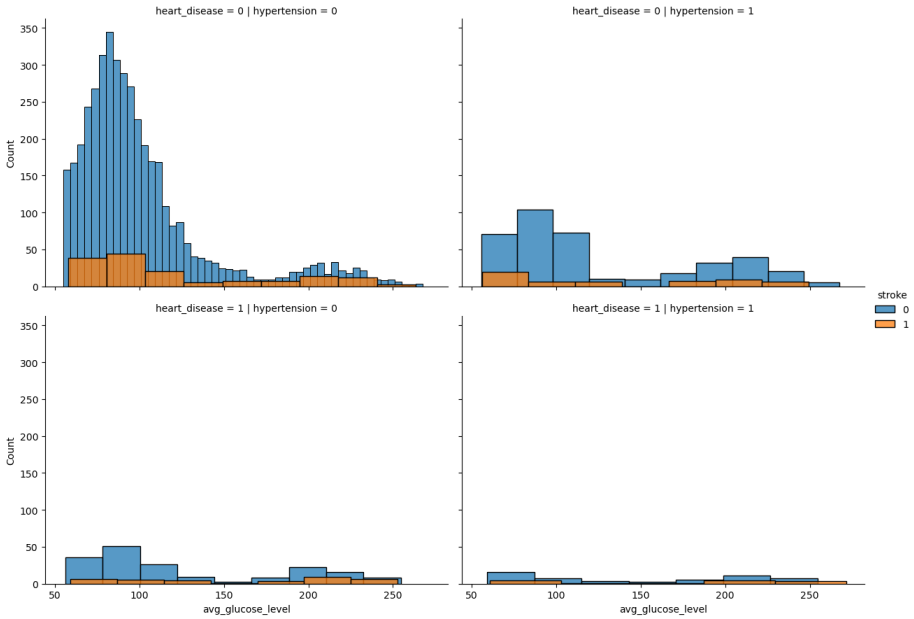
Married people with high average glucose level than normal have high stroke rate. The risk seems less for the same people with hypertension

---

Hypertension, avg_glucose_level, heart_disease and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension",row="heart_disease",hue="stroke", height=4.5
g.map(sns.histplot, "avg_glucose_level")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x213ac681ed0>
```

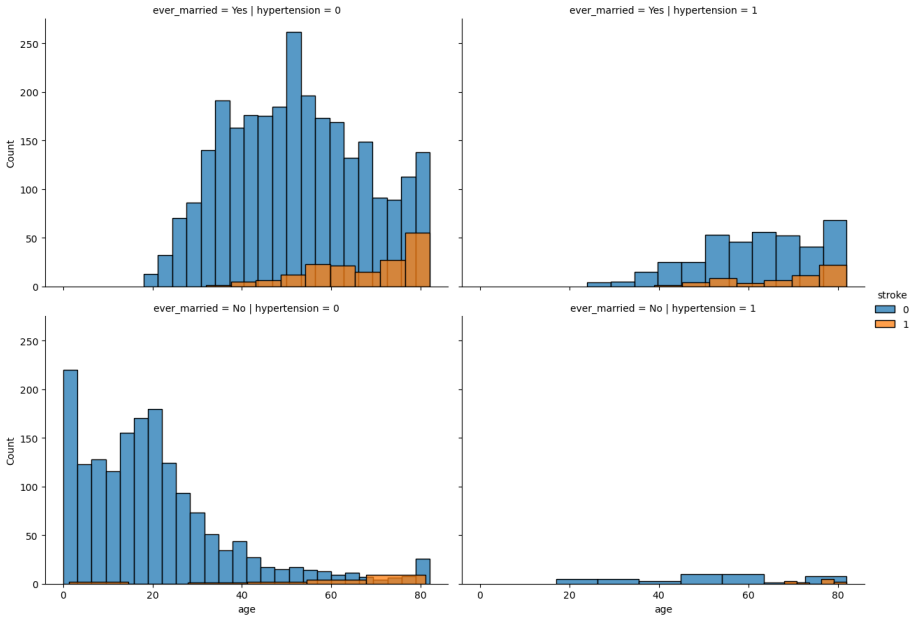


As we discussed earlier, people with average glucose levels between 70 and 125 have a high rate of stroke, regardless of heart disease or hypertension. Within them, those with hypertension also a bit more chance to face stroke.

Hypertension, age, ever_married and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension",row="ever_married",hue="stroke", height=4.5,
g.map(sns.histplot, "age")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x213ac879750>
```

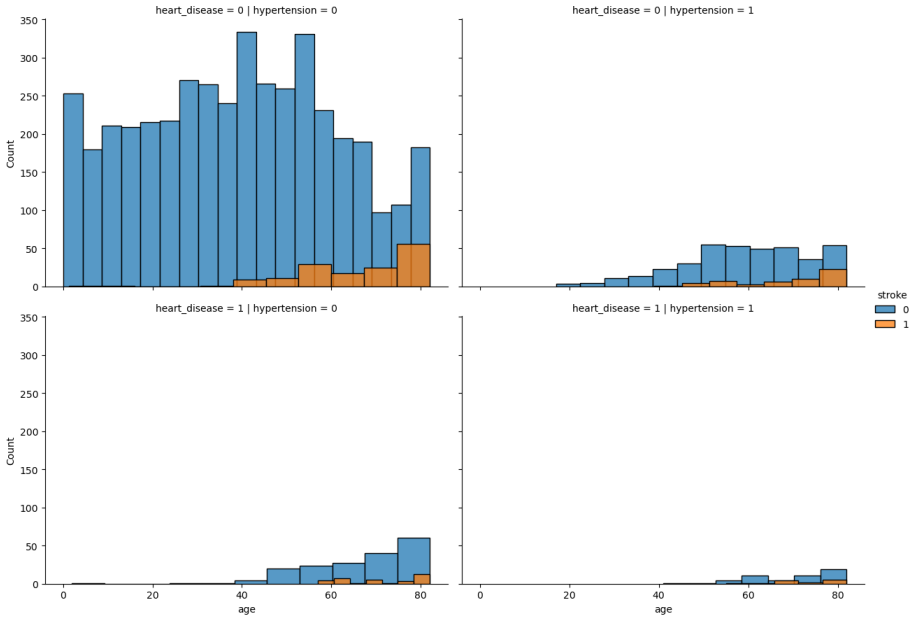


In most cases old people have more chances to have a stroke. Those within them who are married seems the most impacted according to the current data. Those who are old, married and have hypertension have stroke.

Hypertension, age, heart_disease and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension",row="heart_disease",hue="stroke", height=4.5
g.map(sns.histplot, "age")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x213aefe3250>
```



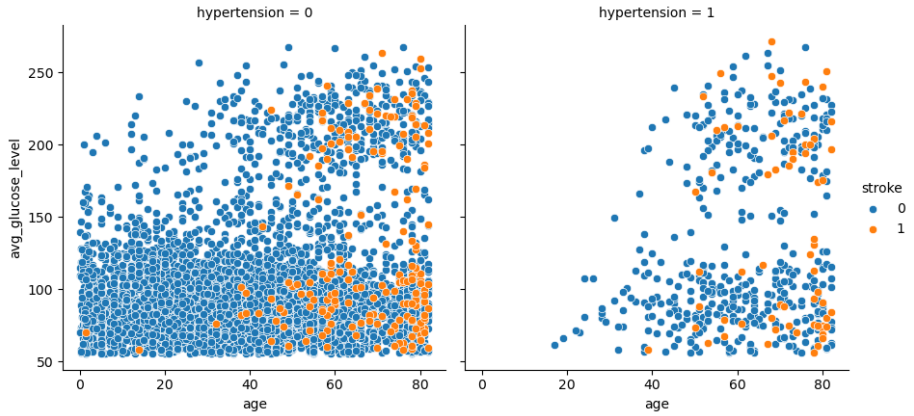
Old people with heart_disease face less strok than those without a heart_disease

---

Hypertension, age, avg_glucose_level and stroke

```
g = sns.FacetGrid(df_PrePro, col="hypertension", hue="stroke", height=4.5, aspect=1)
g.map_dataframe(sns.scatterplot, x="age", y="avg_glucose_level")
g.add_legend()
```

```
<seaborn.axisgrid.FacetGrid at 0x2139bcf84c0>
```



Most people seem to have an average blood sugar below 125, regardless of age,

but strokes occur among them in people over 50 years old. The data imbalance does not allow to really notice the hypertension impact on those people but we have the same trend for old people with hypertension.

Feature Engineering and variables selection

Features data transformation and features correlation will studied here in order to select relevant variable for the classification model

1-Data transformation

They will be two data transformations for the **categorical variables**: one just for feature correlation study with pandas corr() function which needs numerical values and later will be feature transformation for the model use.

```
df_transf= df_PrePro.copy() # Temporary dataframe for feature transformation for correlatio
```

```
labelClassEncod = {} # A dictionnary variable to keep the label order after transform use
```

```
# CONVERSION STRING TO INTEGER
# A function is designed in order to avoid codes repeating.
# LabelEncoder() of sklearn.preprocessing is used
# "featureColumn": DataFrame column which is nominal categorical variable, "featureLabelLis
```

```
def labelEncodeFunc(featureColumn,featureLabelList):
    global labelClassEncod
    if isinstance((df_transf.loc[(df_transf.loc[:,featureColumn]).index[0],featureColumn]),s
        encodeMethod=LabelEncoder()
        encodeMethod.fit(featureLabelList) # Gender column labels

        # Column "featureColumn" values order saving in the Dictionnary
        labelClassEncod[featureColumn]=list(encodeMethod.classes_)

        #The transformation use on the column "featureColumn"
        featureTransformed = encodeMethod.transform(df_transf.loc[:,featureColumn])
        df_transf.loc[:,featureColumn]=featureTransformed

        return list(encodeMethod.classes_), df_transf.loc[:,featureColumn]
    else :
        # labelClassEncod[featureColumn] will display the former "featureColumn" transforma
        return labelClassEncod[featureColumn], df_transf.loc[:,featureColumn]
```

```
# Gender column transformation in 0 (Female) and 1 (Male)
labelEncodeFunc("gender",["Female","Male"])
```

```
(['Female', 'Male'],
```

```
id
9046     1
51676    0
31112    1
60182    0
1665     0
        ..
18234    0
44873    0
19723    0
37544    1
44679    0
Name: gender, Length: 5109, dtype: object)
```

```
# ever_married column transformation in 0 (No) and 1 (Yes)
labelEncodeFunc("ever_married",["No","Yes"])
```

```
(['No', 'Yes'],
id
9046     1
51676    1
31112    1
60182    1
1665     1
        ..
18234    1
44873    1
19723    1
37544    1
44679    1
Name: ever_married, Length: 5109, dtype: object)
```

```
# work_type column transformation in 0,1,2,3,4 in the order 'Govt_job', 'Never_worked', 'Pr
labelEncodeFunc("work_type",["Private","Self-employed","children","Govt_job","Never_worked"]
```

```
(['Govt_job', 'Never_worked', 'Private', 'Self-employed', 'children'],
id
9046     2
51676    3
31112    2
60182    2
1665     3
        ..
18234    2
44873    3
19723    3
37544    2
44679    0
```

```
 Name: work_type, Length: 5109, dtype: object)
```

```
# Residence_type column transformation in 0 ('Rural') and 1 (Urban')
labelEncodeFunc("Residence_type",["Rural","Urban"])
```

```
(['Rural', 'Urban'],
 id
 9046     1
 51676    0
 31112    0
 60182    1
 1665     0
         ..
 18234    1
 44873    1
 19723    0
 37544    0
 44679    1
 Name: Residence_type, Length: 5109, dtype: object)
```

```
# smoking_status column transformation in the following order 'Unknown', 'formerly smoked',
labelEncodeFunc("smoking_status",["never smoked","formerly smoked","smokes","Unknown"])
```

```
(['Unknown', 'formerly smoked', 'never smoked', 'smokes'],
 id
 9046     1
 51676    2
 31112    2
 60182    3
 1665     2
         ..
 18234    2
 44873    2
 19723    2
 37544    1
 44679    0
 Name: smoking_status, Length: 5109, dtype: object)
```

```
#All label order printing
labelClassEncod
```

```
{'gender': ['Female', 'Male'],
 'ever_married': ['No', 'Yes'],
 'work_type': ['Govt_job',
  'Never_worked',
  'Private',
  'Self-employed',
  'children'],
 'Residence_type': ['Rural', 'Urban'],
```

```
 'smoking_status': ['Unknown', 'formerly smoked', 'never smoked', 'smokes']}
```

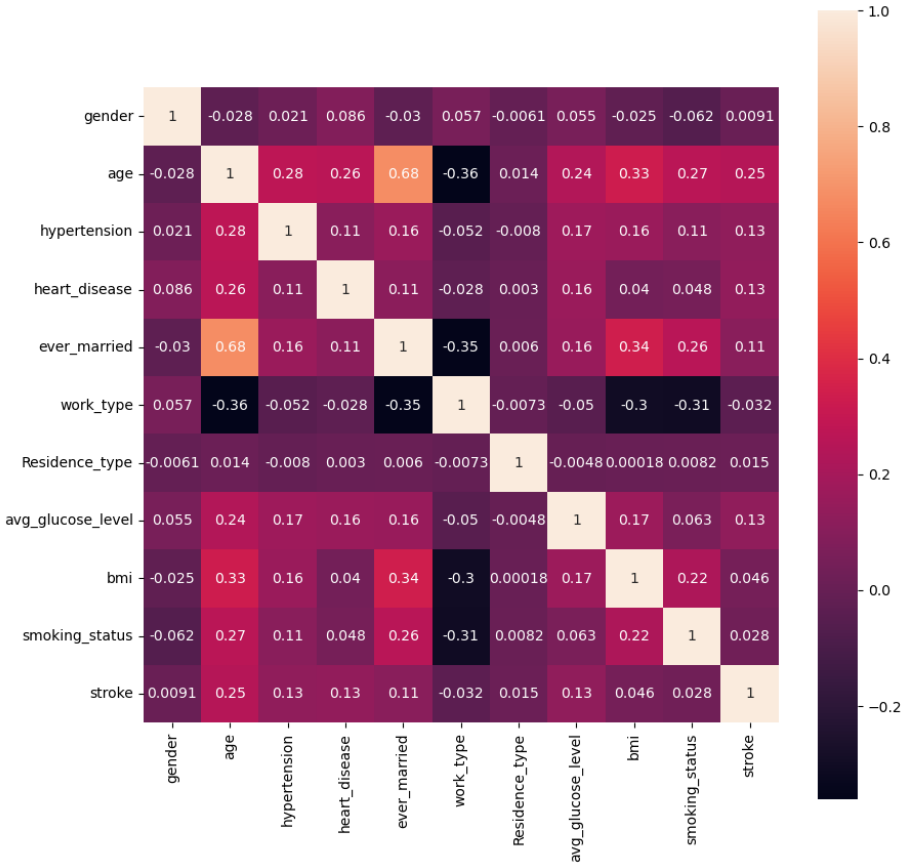
```
df_transf.head()
```

```
      gender   age  hypertension  heart_disease ever_married work_type  \
id
9046       1  67.0             0              1            1         2
51676      0  61.0             0              0            1         3
31112      1  80.0             0              1            1         2
60182      0  49.0             0              0            1         2
1665       0  79.0             1              0            1         3

      Residence_type  avg_glucose_level   bmi smoking_status  stroke
id
9046               1             228.69  36.6              1       1
51676              0             202.21  30.5              2       1
31112              0             105.92  32.5              2       1
60182              1             171.23  34.4              3       1
1665               0             174.12  24.0              2       1
```

2-Feature correlation study and variable selection

The above dataframe is arrange in numeric values to use Pearson statistic test with pandas **df_transf.corr()** below

```
#needs seaborn version 0.13.0
plt.figure(figsize=(10,10))
sns.heatmap(df_transf.corr(),annot=True,square=True)
plt.savefig("images/corr_plot.png")
plt.show()
```



Correlation hypotheses from the above results

- **age**, **hypertension**, **heart-disease**, **ever__married** and **avg__glucose__level** seem strongly correlated with the target variable **stroke**
- Between **stroke** and explanatory variables: only **"gender" and "Residence__type"** seem not correlated to response variable stroke. only **"work__type"** is negatively correlated to stroke
- Between **explanatory variables**: **"age" and "work__type"** seem negatively correlated. **"age"** and **"ever__married"** seem **highly** positively correlated

Correlations confirmation cheking in three steps:

- Correlation between numerical variable with **Pearson statistic test** with pd.corr()
- Correlation between categorical variables with **Chi2 statistic test with chi2__contingency() function**

- Correlation between numerical and categorical variables with **ANOVA test** through **"SelectKBest"** class.

1-Correlation between numerical variable with **Pearson statistic test** with pd.corr()

```
df_numeric = df_transf[['age', 'avg_glucose_level', 'bmi']]
df_numeric.corr()
```

	age	avg_glucose_level	bmi
age	1.000000	0.238323	0.327495
avg_glucose_level	0.238323	1.000000	0.169268
bmi	0.327495	0.169268	1.000000

There is no significant correlation between numerical variables even though age and bmi seems more correlated than the others

2-Correlation between categorical variables and the target variable **"stroke"** with **Chi2 statistic test with chi2_contingency() function**

```
def chi2CorrTest(feat1,feat2):     # e.g: df["stroke"],df["gender"]
    """
    Function of correlation computation between two categorical variables
    """
    table=pd.crosstab(feat1,feat2)
    corr_table=chi2_contingency(table)
    #print(corr_table[0])
    print("Chi2 Test P_value:",corr_table[1])
    print(corr_table[2])
    if (corr_table[1] < 0.05):
        print("These features are '\033[94m correlated' as ''P_value'' < 0.05")
    else:
        print("These features are not correlated as ''P_value'' >= 0.05")
    #return

chi2CorrTest(df_transf["gender"],df_transf["stroke"])
```

```
Chi2 Test P_value: 0.5598277580669416
1
These features are not correlated as ''P_value'' >= 0.05
```

```
chi2CorrTest(df_transf["hypertension"],df_transf["stroke"])
```

```
Chi2 Test P_value: 1.688936253410575e-19
1
These features are ' correlated' as ''P_value'' < 0.05
```

```
chi2CorrTest(df_transf["heart_disease"],df_transf["stroke"])
```

```
Chi2 Test P_value: 2.120831133146208e-21
1
These features are ' correlated' as ''P_value'' < 0.05
```

```
chi2CorrTest(df_transf["ever_married"],df_transf["stroke"])
```

```
Chi2 Test P_value: 1.6862856191673454e-14
1
These features are ' correlated' as ''P_value'' < 0.05
```

```
chi2CorrTest(df_transf["work_type"],df_transf["stroke"])
```

```
Chi2 Test P_value: 5.40903546949726e-10
4
These features are ' correlated' as ''P_value'' < 0.05
```

```
chi2CorrTest(df_transf["smoking_status"],df_transf["stroke"])
```

```
Chi2 Test P_value: 2.0077041756108317e-06
3
These features are ' correlated' as ''P_value'' < 0.05
```

```
chi2CorrTest(df_transf["Residence_type"],df_transf["stroke"])
```

```
Chi2 Test P_value: 0.29982523877153633
1
These features are not correlated as ''P_value'' >= 0.05
```

3-Correlation between numerical and categorical variables with **ANOVA test**

ANOVA test with formula.api.ols().fit() and api.stats.anova_lm()

This is to test correlation between categorical features **work_type**, **ever_married** and the numerical feature **age**

```
#import statsmodels.api
resulta=statsmodels.formula.api.ols('age~work_type',data=df_transf).fit()
corrAgeWorkType=statsmodels.api.stats.anova_lm(resulta)
```

```
corrAgeWorkType
```

	df	sum_sq	mean_sq	F	PR(>F)
work_type	4.0	1.215328e+06	303831.964265	1110.246464	0.0
Residual	5104.0	1.396769e+06	273.661727	NaN	NaN

```
#import statsmodels.api
resulta=statsmodels.formula.api.ols('age~ever_married',data=df_transf).fit()
corrAgeWorkType=statsmodels.api.stats.anova_lm(resulta)
```

```
corrAgeWorkType
```

	df	sum_sq	mean_sq	F	PR(>F)
ever_married	1.0	1.204583e+06	1.204583e+06	4370.69022	0.0
Residual	5107.0	1.407514e+06	2.756048e+02	NaN	NaN

Conclusion: above P-values ($0.0 < 0.05$) confirm correlations between "age" and "ever_married" and "work_type"; but as pd.corr() has previously proved

the correlation coefficient of "age" and "ever_married" is high, implying a strong relationship between both. So to avoid data redundancy a choice might be made between "age" and "ever_married".

Let check below SelectKBest class result and see

```
def kbest_Feature_Selection_clf(data_frame, target, k=5):
    """
    Selecting K-Best features for classification
    """
    feat_selector = SelectKBest(f_classif, k=k)
    _ = feat_selector.fit(data_frame.drop(target, axis=1), data_frame[target])

    feat_scores = pd.DataFrame()
    feat_scores["F Score"] = feat_selector.scores_
    feat_scores["P Value"] = feat_selector.pvalues_
    feat_scores["Support"] = feat_selector.get_support()
    feat_scores["Attribute"] = data_frame.drop(target, axis=1).columns

    return feat_scores

df_FeatSelect = df_transf # DataFrame saving for temporary operation
kbest_features = kbest_Feature_Selection_clf(df_FeatSelect, "stroke", k=5)

kbest_features = kbest_features.sort_values(["F Score", "P Value"], ascending=[False, False]
kbest_features
```

	F Score	P Value	Support	Attribute
1	326.799849	7.435469e-71	True	age
3	94.666779	3.506802e-22	True	heart_disease
7	90.550026	2.705303e-21	True	avg_glucose_level
2	84.919947	4.441473e-20	True	hypertension
4	60.609558	8.367747e-15	True	ever_married
8	10.928987	9.532482e-04	False	bmi
5	5.341306	2.086549e-02	False	work_type
9	4.037898	4.454172e-02	False	smoking_status
6	1.213760	2.706407e-01	False	Residence_type
0	0.421144	5.163959e-01	False	gender

Above the 5 first features have strongest F-Scores and weakest P-Values. These results were also confirmed by the previous correlation studies. We can also add the following 3-bmi, work_type and smoking_status- but to avoid overfitting and data redundancy some will be chosen.

Final feature selection conclusion: For this study 2 models will be tested fondamentally based on the use of pd.corr() and SelectKBest():

- Model1: Stroke~age,hypertension,heart_disease,avg_glucose_level (By prefering **age** and leaving out **ever_married** strongly correlated)
- Model2 Stroke~age,hypertension,heart_disease,avg_glucose_level,ever_married (adding **ever_married** to **Model1**)

The best will be selected.

```
df_proces = df_transf.copy()
```

```
df_proces
```

```
      gender   age  hypertension  heart_disease ever_married work_type  \
id
9046       1  67.0             0              1            1         2
51676      0  61.0             0              0            1         3
31112      1  80.0             0              1            1         2
60182      0  49.0             0              0            1         2
1665       0  79.0             1              0            1         3
...      ...   ...           ...            ...          ...       ...
18234      0  80.0             1              0            1         2
44873      0  81.0             0              0            1         3
19723      0  35.0             0              0            1         3
37544      1  51.0             0              0            1         2
44679      0  44.0             0              0            1         0

      Residence_type  avg_glucose_level   bmi smoking_status  stroke
id
9046               1             228.69  36.6              1       1
51676              0             202.21  30.5              2       1
31112              0             105.92  32.5              2       1
60182              1             171.23  34.4              3       1
1665               0             174.12  24.0              2       1
...              ...                ...   ...            ...     ...
18234              1              83.75  28.8              2       0
44873              1             125.20  40.0              2       0
19723              0              82.99  30.6              2       0
37544              0             166.29  25.6              1       0
44679              1              85.28  26.2              0       0

[5109 rows x 11 columns]
```

```
df_PrePro_copy= df_PrePro.copy()
```

```
df_PrePro_copy
```

```
       gender   age  hypertension  heart_disease ever_married      work_type  \
id
9046     Male  67.0             0              1          Yes        Private
51676  Female  61.0             0              0          Yes  Self-employed
```

```
31112    Male  80.0             0              1          Yes        Private
60182  Female  49.0             0              0          Yes        Private
1665   Female  79.0             1              0          Yes  Self-employed
...       ...   ...           ...            ...          ...            ...
18234  Female  80.0             1              0          Yes        Private
44873  Female  81.0             0              0          Yes  Self-employed
19723  Female  35.0             0              0          Yes  Self-employed
37544    Male  51.0             0              0          Yes        Private
44679  Female  44.0             0              0          Yes       Govt_job

      Residence_type  avg_glucose_level   bmi   smoking_status  stroke
id
9046           Urban             228.69  36.6  formerly smoked       1
51676          Rural             202.21  30.5     never smoked       1
31112          Rural             105.92  32.5     never smoked       1
60182          Urban             171.23  34.4           smokes       1
1665           Rural             174.12  24.0     never smoked       1
...              ...                ...   ...              ...     ...
18234          Urban              83.75  28.8     never smoked       0
44873          Urban             125.20  40.0     never smoked       0
19723          Rural              82.99  30.6     never smoked       0
37544          Rural             166.29  25.6  formerly smoked       0
44679          Urban              85.28  26.2          Unknown       0

[5109 rows x 11 columns]
```

MODELS

```
#Data splitting
df_train, df_test = train_test_split(df_proces,test_size = 0.2)
```

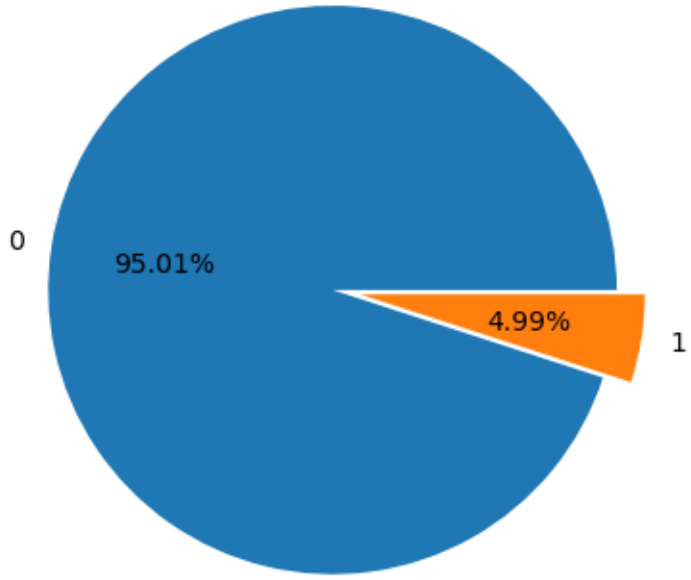
```
len(df_proces)
```

```
5109
```

```
len(df_train)
```

```
4087
```

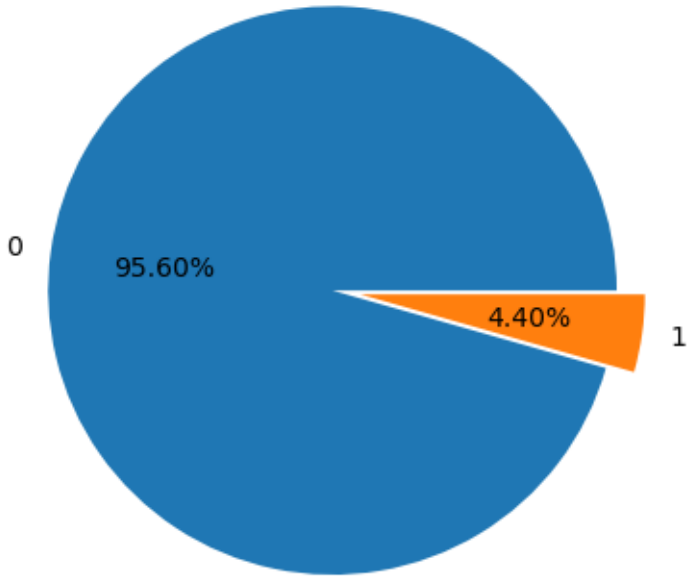
```
plt.pie(df_train.stroke.value_counts(),labels=df_train.stroke.value_counts().index,autopct='
plt.show()
```




```
len(df_test)
```

```
1022
```

```
plt.pie(df_test.stroke.value_counts(),labels=df_test.stroke.value_counts().index,autopct='%1
plt.show()
```



In both datasets (Train and Test) there is a same target variable class repartition proportion 95% for unstroke and 5% for stroke; strong imbalance for the target variable "stroke" delicate for the model training.

```
df_train
```

```
      gender   age  hypertension  heart_disease ever_married work_type  \
id
4699       1  50.0             0              0            0         0
24885      1  79.0             0              1            1         3
24892      1  64.0             0              0            1         2
68275      1  52.0             0              0            1         2
59359      1  79.0             0              0            1         3
...      ...   ...           ...            ...          ...       ...
27799      1  72.0             0              0            1         2
7298       0  56.0             0              0            1         3
70031      0  71.0             1              0            1         2
49057      0  32.0             0              0            0         2
8085       1  18.0             0              0            0         2

      Residence_type  avg_glucose_level   bmi smoking_status  stroke
id
```

```
4699              0             121.17  25.5             1       0
24885             1              88.83  40.3             3       0
24892             0              97.08  31.7             0       0
68275             1             247.69  35.1             0       0
59359             1             105.93  25.2             2       0
...             ...                ...   ...           ...     ...
27799             0             209.26  38.1             1       0
7298              0              70.23  35.5             2       0
70031             0             195.25  33.3             2       0
49057             0              67.92  22.8             3       0
8085              0             143.45  32.0             3       0

[4087 rows x 11 columns]
```

```
print(df_proces.gender.mean())
print(df_train.gender.mean())
print(df_test.gender.mean())
```

```
0.41397533763945976
0.40787863958894055
0.4383561643835616
```

```
print(df_proces.stroke.mean())
print(df_train.stroke.mean())
print(df_test.stroke.mean())
```

```
0.04873752201996477
0.04991436261316369
0.04403131115459882
```

Mean seems the same in the same column across the three datasets. The model training result can be applied on global dataset later.

Features extraction for models use

```
df_train.columns
```

```
Index(['gender', 'age', 'hypertension', 'heart_disease', 'ever_married',
       'work_type', 'Residence_type', 'avg_glucose_level', 'bmi',
       'smoking_status', 'stroke'],
      dtype='object')
```

```
# get the values of the columns for the training data
X_train = df_train.loc[:,['age','hypertension','heart_disease','avg_glucose_level']].values
y_train = df_train.stroke.values

# get the values of the columns for the test data
X_test = df_test.loc[:,['age','hypertension','heart_disease','avg_glucose_level']].values
y_test = df_test.stroke.values
```

Logistic Regression

The first model use

Model1: Stroke~age,hypertension,heart_disease,avg_glucose_level (By prefering **age** and leaving out **ever_married** strongly correlated)

```
# here we initialize the model
lr_model = LogisticRegression(random_state=0,max_iter=1000)
```

```
# here we train the model on the training data
lr_model.fit(X=X_train, y=y_train)
```

```
LogisticRegression(max_iter=1000, random_state=0)
```

```
y_test_predicted = lr_model.predict(X_test)
```

```
y_test
```

```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

```
y_test_predicted
```

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

```
cf = pd.DataFrame(
    columns=["y_test_0","y_test_1"],index=["y_pred_0","y_pred_1"]
)
```

```
cf.loc[:,:] = confusion_matrix(y_true= y_test,y_pred= y_test_predicted)
```

```
cf
```

```
         y_test_0 y_test_1
y_pred_0      977        0
y_pred_1       45        0
```

This result is the consequence of high data imbalance forcing the model to more predict unstroke cases.

```
recall_score(y_true=y_test, y_pred=y_test_predicted)
```

```
0.0
```

```
precision_score(y_true=y_test, y_pred=y_test_predicted)
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\sklearn\metrics\_classificati
  _warn_prf(average, modifier, msg_start, len(result))
```

```
0.0
```

```
report =classification_report(y_true=y_test, y_pred=y_test_predicted)
```

```
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\sklearn\metrics\_classificati
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\sklearn\metrics\_classificati
```

```
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\envs\env_strokepred\lib\site-packages\sklearn\metrics\_classificati
  _warn_prf(average, modifier, msg_start, len(result))
```

```
print(report)
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       977
           1       0.00      0.00      0.00        45

    accuracy                           0.96      1022
   macro avg       0.48      0.50      0.49      1022
weighted avg       0.91      0.96      0.93      1022
```

This result is the consequence of high data imbalance forcing the model to more predict unstroke cases.

We will use the Logistic Regression parameter 'balance' (**class_weight='balanced'**) for the model training:

```
# here we initialize the model with
lr_model = LogisticRegression(random_state=0,max_iter=1000,class_weight='balanced')
```

```
# here we train the model on the training data
lr_model.fit(X=X_train, y=y_train)
```

```
LogisticRegression(class_weight='balanced', max_iter=1000, random_state=0)
```

```
y_test_predicted = lr_model.predict(X_test)
```

```
cf = pd.DataFrame(
    columns=["y_test_0","y_test_1"],index=["y_pred_0","y_pred_1"]
)
```

```
cf.loc[:,:] = confusion_matrix(y_true= y_test,y_pred= y_test_predicted)
```

```
cf
```

```
         y_test_0 y_test_1
y_pred_0      729      248
y_pred_1        8       37
```

The Logistic Regression model seems well performing with **74% (723/975*100) for True negative and 83% (39/47*100) for True positive**. But There high 2nd errors with 26% false negative (252/975*100)

```
report =classification_report(y_true=y_test, y_pred=y_test_predicted)
```

```
print(report)
```

```
              precision    recall  f1-score   support
```

```
              0       0.99      0.75      0.85       977
              1       0.13      0.82      0.22        45

    accuracy                            0.75      1022
   macro avg       0.56      0.78      0.54      1022
weighted avg       0.95      0.75      0.82      1022
```

Data imbalance impact looks high in the above figures:

- **Recall** looks good for both classes (Unstroke and stroke) but
- **Precision** and **f1 score** are very weak for class 1 only 23% far from 1.0

We will try to balance the data with **SMOTE** and compare the results

SMOTE use

```
smote=SMOTE()
train_x,train_y=smote.fit_resample(X_train,y_train)
test_x,test_y=smote.fit_resample(X_test,y_test)
```

```
print(train_x.shape,train_y.shape,test_x.shape,test_y.shape)
```

```
(7766, 4) (7766,) (1954, 4) (1954,)
```

```
# here we initialize the model
lr_model = LogisticRegression(random_state=0,max_iter=1000)
```

```
# here we train the model on the training data
lr_model.fit(X=train_x, y=train_y)
```

```
LogisticRegression(max_iter=1000, random_state=0)
```

```
test_predicted_y = lr_model.predict(test_x)
```

```
cfsmote = pd.DataFrame(
    columns=["y_test_0","y_test_1"],index=["y_pred_0","y_pred_1"]
)
```

```
cfsmote.loc[:,:] = confusion_matrix(y_true= test_y,y_pred= test_predicted_y)
```

```
cfsmote
```

```
         y_test_0 y_test_1
y_pred_0      734      243
y_pred_1      166      811
```

```
cfsmote/len(test_y)
```

```
          y_test_0  y_test_1
y_pred_0   0.37564   0.12436
y_pred_1  0.084954  0.415046
```

```
recall_score(y_true=test_y, y_pred=test_predicted_y)
```

```
0.8300921187308086
```

```
precision_score(y_true=test_y, y_pred=test_predicted_y)
```

```
0.7694497153700189
```

```
report_smote =classification_report(y_true=test_y, y_pred=test_predicted_y)
print(report_smote)
```

```
              precision    recall  f1-score   support

           0       0.82      0.75      0.78       977
           1       0.77      0.83      0.80       977

    accuracy                           0.79      1954
   macro avg       0.79      0.79      0.79      1954
weighted avg       0.79      0.79      0.79      1954
```

With the SMOTE use, f1 scores are both acceptable for both classes. The trade-off between Precision and Recall seems fine.

This prove the SMOTE action seems benefit for the model training and will improve the model prediction capacity.

The second model use

Model2: Stroke~age,hypertension,heart_disease,avg_glucose_level,ever_married (adding **ever_married** to **Model1**)

```
# get the values of the columns for the training data
X_train = df_train.loc[:,['age','hypertension','heart_disease','avg_glucose_level', 'ever_ma
y_train = df_train.stroke.values

# get the values of the columns for the test data
X_test = df_test.loc[:,['age','hypertension','heart_disease','avg_glucose_level', 'ever_marr
y_test = df_test.stroke.values

# here we initialize the model
lr_model = LogisticRegression(random_state=0,max_iter=1000,class_weight='balanced')

# here we train the model on the training data
lr_model.fit(X=X_train, y=y_train)
```

```
LogisticRegression(class_weight='balanced', max_iter=1000, random_state=0)
```

```
y_test_predicted = lr_model.predict(X_test)
```

```
cf = pd.DataFrame(
    columns=["y_test_0","y_test_1"],index=["y_pred_0","y_pred_1"]
)
```

```
cf.loc[:,:] = confusion_matrix(y_true= y_test,y_pred= y_test_predicted)
```

```
cf
```

```
         y_test_0 y_test_1
y_pred_0      730      247
y_pred_1        9       36
```

Not significant performance increase even though false negative has decreased from 252 to 248.

```
report =classification_report(y_true=y_test, y_pred=y_test_predicted)
```

```
print(report)
```

```
              precision    recall  f1-score   support

           0       0.99      0.75      0.85       977
           1       0.13      0.80      0.22        45

    accuracy                           0.75      1022
   macro avg       0.56      0.77      0.54      1022
weighted avg       0.95      0.75      0.82      1022
```

Conclusion:* the feature **'ever_married'** does not have more impact the model performance than the first one. This was anticipated as **age** is correlated to **ever-married**. **The first model without ever_married feature will be sufficient as model**

OTHER CLASSIFIERS USE

SMOTE datasets will be used: test_x, train_y, test_x and test_y

MLP Classifier

```
# try a new classifier: Multi-Layer Perceptron classifier
nn_model = MLPClassifier(hidden_layer_sizes=(20,10),max_iter=1000)
```

```
#nn_model.fit(X=X_train,y=y_train)
nn_model.fit(X=train_x,y=train_y)
```

```
MLPClassifier(hidden_layer_sizes=(20, 10), max_iter=1000)
```

```
y_test_predicted_nn = nn_model.predict(test_x)
```

```
report_nn = classification_report(y_pred=y_test_predicted_nn,y_true=test_y)
```

Decision Tree Classifier

```
# Try a Decision Tree classifier
dt_model = DecisionTreeClassifier()

dt_model.fit(X=train_x,y=train_y)

DecisionTreeClassifier()

y_test_predicted_dt = dt_model.predict(test_x)

report_dt = classification_report(y_pred=y_test_predicted_dt,y_true=test_y)
```

Random Forest Classifier

```
# Try an ensemble classifier: Random Forest
rf_model = RandomForestClassifier()

rf_model.fit(X=train_x,y=train_y)

RandomForestClassifier()

y_test_predicted_rf = rf_model.predict(test_x)

report_rf = classification_report(y_pred=y_test_predicted_rf,y_true=test_y)
```

CLASSIFIERS REPORTS SUMMARY

```
print("Report of logistic regression")

print(report_smote)
```

```
Report of logistic regression
              precision    recall  f1-score   support

           0       0.82      0.75      0.78       977
           1       0.77      0.83      0.80       977

    accuracy                           0.79      1954
   macro avg       0.79      0.79      0.79      1954
weighted avg       0.79      0.79      0.79      1954
```

```
print("Report of MLP model")
print(report_nn)
```

```
Report of MLP model
              precision    recall  f1-score   support

           0       0.82      0.79      0.81       977
           1       0.80      0.83      0.81       977
```

```
    accuracy                           0.81      1954
   macro avg       0.81      0.81      0.81      1954
weighted avg       0.81      0.81      0.81      1954
```

```
print("Report of Decision Tree classifier model")
print(report_dt)
```

```
Report of Decision Tree classifier model
              precision    recall  f1-score   support

           0       0.77      0.92      0.84       977
           1       0.90      0.73      0.81       977

    accuracy                           0.82      1954
   macro avg       0.84      0.82      0.82      1954
weighted avg       0.84      0.82      0.82      1954
```

```
print("Report of Random Forest model")
print(report_rf)
```

```
Report of Random Forest model
              precision    recall  f1-score   support

           0       0.75      0.95      0.84       977
           1       0.93      0.69      0.79       977

    accuracy                           0.82      1954
   macro avg       0.84      0.82      0.81      1954
weighted avg       0.84      0.82      0.81      1954
```

Conclusion **Decision Tree classifier model** and **Random Forest model** seem better with **f1-score** macro average equal to 83% with the best trade-off between Recall-score and Precision score. Those models know more how to handle outliers. **MLP model** comes next but **logistic regression** has accpetable result regarding the small size of the data.