# c03_2

July 19, 2022

# 1 CARDIO CATCH DISEASES

## 1.1 0. INTRODUCTION

### 1.1.1 0.1 Planning

**Input**

- Create a tool that increases the diagnostic accuracy;
- Database with patient diagnoses;

**Output**

- Model with stable accuracy for all situationsPerformance of the model;
- Most important patient characteristics;
- Tool that can be easy used by health specialists;

**Tasks**

1. What information is important for predicting heart problems?
    - Is blood pressure the most important parameter?
2. Performance of the model:
    - Can accuracy alone solve this problem?
    - What is the minimum value required for a model used in health problems?
3. Action Plan:
    - User-friendly website to enter patient data and return forecast

### 1.1.2 0.2 Imports

[ ]:
```
import pickle
import numpy    as np
import pandas   as pd
import seaborn  as sns
import sweetviz as sv

from IPython.core.display       import HTML, Image
from typing                     import Union
from sklearn.model_selection    import train_test_split, KFold
from matplotlib                 import pyplot                          as plt
```

```python
from scipy                          import stats

from sklearn.preprocessing          import MinMaxScaler, RobustScaler

from sklearn.ensemble               import RandomForestClassifier,␣
↪ExtraTreesClassifier, AdaBoostClassifier, GradientBoostingClassifier

from yellowbrick.features           import Rank1D
from boruta                         import BorutaPy

from xgboost                        import XGBClassifier
from lightgbm                       import LGBMClassifier
from catboost                       import CatBoostClassifier


from sklearn.feature_selection      import RFE
from sklearn.naive_bayes            import GaussianNB
from sklearn.metrics                import accuracy_score, precision_score,␣
↪recall_score, f1_score, roc_auc_score
from sklearn.neighbors              import KNeighborsClassifier
from sklearn.preprocessing          import MinMaxScaler, RobustScaler
```

### 1.1.3 0.3 Helper Functions

[ ]:
```python
def numerical_metrics(numerical_attributes: Union[int, float]):
    """Shows the main values for descriptive statistics in numerical variables.

    Args:
        numerical_attributes ([float64 and int64]): [Insert all numerical␣
↪attributes in the dataset]

    Returns:
        [dataframe]: [A dataframe with mean, median, std deviation, skewness,␣
↪kurtosis, min, max and range]
    """

    data_mean = pd.DataFrame(numerical_attributes.apply(np.mean)).T
    data_median = pd.DataFrame(numerical_attributes.apply(np.median)).T
    data_std = pd.DataFrame(numerical_attributes.apply(np.std)).T
    data_min = pd.DataFrame(numerical_attributes.apply(min)).T
    data_max = pd.DataFrame(numerical_attributes.apply(max)).T
    data_range = pd.DataFrame(numerical_attributes.apply(lambda x: x.max() - x.
↪min())).T
    data_q1 = pd.DataFrame(numerical_attributes.apply(lambda x: np.quantile(x, .
↪25))).T
```

```
    data_q3 = pd.DataFrame(numerical_attributes.apply(lambda x: np.quantile(x, .
↪75))).T
    data_skew = pd.DataFrame(numerical_attributes.apply(lambda x: x.skew())).T
    data_kurtosis = pd.DataFrame(numerical_attributes.apply(lambda x: x.
↪kurtosis())).T

    num_attributes = pd.
↪concat([data_min,data_max,data_range,data_mean,data_median, data_q1,␣
↪data_q3,data_std,data_skew,data_kurtosis]).T.reset_index()
    num_attributes.columns =␣
↪['Attributes','Min','Max','Range','Mean','Median','Q1','Q3', 'St␣
↪deviation','Skewness','Kurtosis']

    return num_attributes
```

[ ]:
```
def categorical_metrics(data: Union[int, str], col: str):
    """
    Shows the the absolute and percent values in categorical variables.

    Args:
        data ([dataframe]): [Insert all categorical attributes in the dataset]

    Returns:
        [dataframe]: [A dataframe with absolute and percent values]
    """

    return pd.DataFrame({'absolute': data[col].value_counts(), 'percent %':␣
↪data[col].value_counts(normalize = True) * 100 })
```

[ ]:
```
def cat_convert(data: Union[int, float]):
     """
     Revert the Encoding on Categorical Features

     Args:
        data ([dataframe]): [Insert all categorical attributes in the dataset]

     Returns:
        data ([dataframe]): [Categorical Dataframe]
     """

     data['gender'] = data['gender'].apply(lambda x: 'woman' if x == 1 else␣
↪'man')

     data['smoker'] = data['smoker'].apply(lambda x: 'yes' if x == 1 else 'no')
```

```
    data['alcohol_intake'] = data['alcohol_intake'].apply(lambda x: 'yes' if x␣
↪== 1 else 'no')

    data['physical_activity'] = data['physical_activity'].apply(lambda x:␣
↪'yes' if x == 1 else 'no')

    data['cardio_result'] = data['cardio'].apply(lambda x: 'yes' if x == 1␣
↪else 'no')

    data['cholesterol'] = data['cholesterol'].apply(lambda x: 'normal' if x ==␣
↪1 else
                                                      'above normal' if x == 2␣
↪else
                                                      'well above normal')

    data['glucose'] = data['glucose'].apply(lambda x: 'normal' if x == 1 else
                                              'above normal' if x == 2 else
                                              'well above normal')

    return data
```

[ ]:
```
def jupyter_settings():
    %matplotlib inline
    %pylab inline
    plt.style.use('tableau-colorblind10')
    plt.rcParams['figure.figsize'] = [25, 12]
    plt.rcParams['font.size'] = 24
    display(HTML('<style>.container { width:100% !important; }</style>'))
    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
    pd.set_option('display.expand_frame_repr', False)
    sns.set()

jupyter_settings()
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

```
<IPython.core.display.HTML object>
```

[ ]:
```
def multiple_kdeplots(df: Union[int, float, str], rows: int, cols: int):
    """
    Shows a matrix with kdeplots of selected features.

    Args:
        df ([dataframe]): [Insert all categorical attributes in the dataset]
        rows ([int]): [Insert the number of rows of the subplot]
```

```
        cols ([int]): [Insert the number of columns of the subplot]

    Returns:
        [Image]: [A matrix plot with kdeplots]
    """

    for i, col in enumerate(df.columns, 1):
        plt.subplot(rows, cols, i)
        ax = sns.kdeplot(data = df, x = col)
        plt.ylabel('')

    return ax
```

[ ]:
```
def multiple_boxplots(df: Union[int, float, str], rows: int, cols: int):
    """
    Shows a matrix with boxplots of selected features.

    Args:
        df ([dataframe]): [Insert all categorical attributes in the dataset]
        rows ([int]): [Insert the number of rows of the subplot]
        cols ([int]): [Insert the number of columns of the subplot]

    Returns:
        [Image]: [A matrix plot with boxplots]
    """

    for i, col in enumerate(df.columns, 1):
        plt.subplot(rows, cols, i)
        ax = sns.boxplot(data = df, x = col)
        plt.ylabel('')

    return ax
```

[ ]:
```
def correlation_matrix(data: Union[int, float], method: str):
    """Generates a correlation matrix of numerical variables

    Args:
        data ([DataFrame]): [The dataframe of the EDA]
        method ([string]): [The method used, it can be 'pearson', 'kendall' or␣
 ↪'spearman']

    Returns:
        [Image]: [The correlation matrix plot made with seaborn]
    """

    # correlation
    num_attributes = data.select_dtypes(include = ['int64', 'float64'])
```

```
    correlation = num_attributes.corr(method = method)

    # plot
    ax = sns.heatmap(correlation, fmt = '.2f', vmin = -1, vmax = 1, annot =␣
↪True, cmap = 'magma', square = True).set(title = 'Correlation Matrix')

    return ax
```

[ ]:
```
def correlation_ascending(data: Union[int, float, str], col: str, method: str):
    """Generates a correlation matrix of each numerical variables in ascending␣
↪order.

    Args:
        data ([dataFrame]): [The dataframe of the EDA]
        col ([object]): [The column selected]
        method ([string]): [The method used, it can be 'pearson', 'kendall' or␣
↪'spearman']

    Returns:
        [Image]: [The correlation matrix plot made with seaborn]
    """

    # correlation
    num_attributes = data.select_dtypes(include = ['int64', 'float64'])
    correlation = num_attributes.corr(method = method)

    correlation_asc = correlation[col].sort_values(ascending=False).to_frame()
    correlation_asc.columns = ['']
    correlation_asc.drop(col, axis=0, inplace=True)
    plot = sns.heatmap( correlation_asc, annot=True, cmap='rocket').
↪set_title(col);

    return plot
```

[ ]:
```
def corr_cat(data: Union[str, int]):
    """Calculates Correlation Matrix for Categorical Features using Cramer's V

    Args:
        data ([DataFrame]): [The dataframe with all categorical features]


    Returns:
        [Image]: [Correlation Matrix]
    """
```

```
    categorical_attributes = data.select_dtypes(exclude = ['int64', 'float64'])
    cat_attributes_list = categorical_attributes.columns.tolist()

    corr_dict = {}

    for i in range(len(cat_attributes_list)):
        corr_list = []
        for j in range(len(cat_attributes_list)):
            ref = cat_attributes_list[i]
            feat = cat_attributes_list[j]
            cm = pd.crosstab(categorical_attributes[ref],␣
↪categorical_attributes[feat]).to_numpy()
            n = cm.sum()
            r, k = cm.shape
            chi2 = stats.chi2_contingency(cm)[0]
            chi2corr = max(0, chi2 - (k - 1)*(r - 1)/(n - 1))
            kcorr = k - (k - 1)**2/(n - 1)
            rcorr = r - (r - 1)**2/(n - 1)
            corr = np.sqrt((chi2corr/n) / (min(kcorr - 1, rcorr - 1)))
            corr_list.append(corr)

        corr_dict[ref] = corr_list

    corr_df = pd.DataFrame(corr_dict)
    plot = sns.heatmap(corr_df, fmt = '.2f', vmin = -1, vmax = 1, annot = True,␣
↪cmap = 'magma', square = True).set(title = 'Correlation Matrix: Categorical␣
↪Features');

    return plot;
```

[ ]:
```
def plot_with_target(target_x: Union[int, float, str], target_y: Union[int,␣
↪float, str], col: str, position_x: int, position_y: int, label_x: str,␣
↪label_y: str):
    """
    Create some histplots with target feature.

    Args:
        target_x ([dataframe]): [Dataframe with all numerical features and␣
↪positive target]
        target_y ([dataframe]): [Dataframe with all numerical features and␣
↪negative target]
        col ([str]): [Name of the feature of the plot]
        position_x ([int]): [Position in the subplot]
        position_y ([int]): [Position in the subplot]
        label_x ([str]): [Name of the label of negative target]
        label_y ([str]): [Name of the label of positive target]

```

```
    Returns:
        [Image]: [Histplots of all features with target]
    """

    plt.style.use('tableau-colorblind10')
    ax[position_x, position_y].hist(target_x[col], bins = 200, alpha = 0.5,␣
↪label = label_x)
    ax[position_x, position_y].hist(target_y[col], bins = 200, alpha = 0.7,␣
↪label = label_y)
    ax[position_x, position_y].legend()
    ax[position_x, position_y].set_title(col)

    return ax
```

[ ]:
```
def metrics_df(models, target: str, X_train: Union[int, float, str], y_train:␣
↪Union[int, float, str], X_val: Union[int, float, str], y_val: Union[int,␣
↪float, str], verbose = True):
    """Return Metrics of the model

    Args:
        model: [ML model for metrics evaluation]
        target[str]: [Target feature name]
        X_train[dataframe]: [Train variables]
        y_train[list]: [Target feature list]
        X_val[dataframe]: [Validation variables]
        y_val[list]: [Target feature list]
        verbose[bool]: [print training]

    Returns:
        [DataFrame]: [Dataframe with metrics]
    """
    print('Please, wait a moment - Doing ML')
    model_df = []
    i = 1
    j = len(models)

    for model in models:
        model_name = type(model).__name__
        if verbose == True:
            print(f"Training model {i}/{j} -> " + model_name)
        model.fit(X_train, y_train)

        # probabilities prediction
        yhat = model.predict(X_val)

        # accuracy
        accuracy = accuracy_score(y_val, yhat)
```

```python
        # precision
        precision = precision_score(y_val, yhat)

        # recall
        recall = recall_score(y_val, yhat)

        # f1 score
        f1 = f1_score(y_val, yhat)

        # roc auc
        roc_auc = roc_auc_score(y_val, yhat)

        i += 1

        df_result = pd.DataFrame({'Model_Name': model_name,
                                  'Accuracy': accuracy,
                                  'Precision': precision,
                                  'Recall': recall,
                                  'F1-Score': f1,
                                  'ROCAUC': roc_auc}, index = [0])

        model_df.append(df_result)
        final_result = pd.concat(model_df)
    print('Finished, check the results')

    return final_result
```

[ ]:
```python
def cross_validation(models, target: str, X_train: Union[int, float, str],␣
↪y_train: Union[int, float, str], kfold: int = 5, verbose: bool = True):
    """Return CV result

    Args:
        model: [ML model for CV]
        target[str]: [Target feature name]
        X_train[dataframe]: [Train variables]
        y_train[list]: [Target feature list]
        kfold[int]: [Number of data splits]
        verbose[bool]: [print folding]

    Returns:
        [DataFrame]: [Dataframe for CV]
    """
    print('Please, wait a moment, Doing CV')
    folds = KFold(n_splits = kfold, shuffle = True, random_state = 42)
    accuracy_list = []
    precision_list = []
```

```
    recall_list = []
    f1_list = []
    roc_auc_list = []

    model_df = []
    j = 1
    l = len(models)

    for model in models:
        model_name = type(model).__name__
        if verbose == True:
            print(f"Folding model {j}/{l} -> " + model_name)

        for train_cv, val_cv in folds.split(X_train, y_train):
            X_train_fold = X_train.iloc[train_cv]
            y_train_fold = y_train.iloc[train_cv]
            X_val_fold = X_train.iloc[val_cv]
            y_val_fold = y_train.iloc[val_cv]

            # fit model
            model.fit(X_train_fold, y_train_fold)

            # predict probabilities
            yhat = model.predict(X_val_fold)

            data = X_val_fold.copy()
            data[target] = y_val_fold.copy()

            # accuracy
            accuracy = accuracy_score(y_val_fold, yhat)
            accuracy_list.append(accuracy)

            # precision
            precision = precision_score(y_val_fold, yhat)
            precision_list.append(precision)

            # recall
            recall = recall_score(y_val_fold, yhat)
            recall_list.append(recall)

            # f1 score
            f1 = f1_score(y_val_fold, yhat)
            f1_list.append(f1)

            # roc auc
            roc_auc = roc_auc_score(y_val_fold, yhat)
            roc_auc_list.append(roc_auc)
```

```
        df_result = pd.DataFrame({'Model_Name': (model_name),
                                  'Accuracy Mean': np.mean(accuracy_list).
↪round(3),
                                  'Accuracy STD': np.std(accuracy_list).
↪round(3),
                                  'Precision Mean': np.mean(precision_list).
↪round(3),
                                  'Precision STD': np.std(precision_list).
↪round(3),
                                  'Recall Mean': np.mean(recall_list).round(3),
                                  'Recall STD': np.std(recall_list).round(3),
                                  'F1 Score Mean': np.mean(f1_list).round(3),
                                  'F1 Score STD': np.std(f1_list).round(3),
                                  'ROCAUC Mean': np.mean(roc_auc_list).round(3),
                                  'ROCAUC STD': np.std(roc_auc_list).round(3)},␣
↪index = [0])

        j += 1

        model_df.append(df_result)
        cv_result = pd.concat(model_df)
    print('Finished, check the results')

    return cv_result
```

### 1.1.4 0.4 Settings

[ ]:
```
# round
pd.options.display.float_format = '{:.3f}'.format

seed = 42

homepath = '/home/gutto/Repos/Cardio-Catch-Diseases/'
```

### 1.1.5 0.5 Data

This dataset is avaliable here.

There are 3 types of input features:

Objective: factual information;

Examination: results of medical examination;

Subjective: information given by the patient.

**Data fields**

- **Age:** *Objective Feature* | Age in days
- **Height:** *Objective Feature* | Height in cm
- **Weight:** *Objective Feature* | Weight in kg
- **Gender:** *Objective Feature* | Biological gender, can be Male or Female, 1- Woman, 2- Man
- **Systolic blood pressure (ap_hi):** *Examination Feature*
- **Diastolic blood pressure (ap_lo):** *Examination Feature*
- **Cholesterol:** *Examination Feature* | Can be classified as 1: normal, 2: above normal, 3: well above normal
- **Glucose:** *Examination Feature* | Can be classified as 1: normal, 2: above normal, 3: well above normal
- **Smoking:** *Subjective Feature* | Can be smoke or binary
- **Alcohol intake:** *Subjective Feature* | Can be alco or binary
- **Physical activity:** *Subjective Feature* | Can be active or binary
- **Presence or absence of cardiovascular disease:** *Target Variable* | Can be cardio or binary

[ ]:
```
df_raw = pd.read_csv(homepath + '/data/raw/cardio_train.csv', sep = ';')

df_raw.to_pickle(homepath + 'data/processed/df_raw.pkl')
```

## 1.2 1. DATA DESCRIPTION

### 1.2.1 1.1 Dataset First Look

[ ]:
```
df1 = pd.read_pickle(homepath + 'data/processed/df_raw.pkl')
```

[ ]:
```
df1.head().T
```

[ ]:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| id | 0.000 | 1.000 | 2.000 | 3.000 | 4.000 |
| age | 18393.000 | 20228.000 | 18857.000 | 17623.000 | 17474.000 |
| gender | 2.000 | 1.000 | 1.000 | 2.000 | 1.000 |
| height | 168.000 | 156.000 | 165.000 | 169.000 | 156.000 |
| weight | 62.000 | 85.000 | 64.000 | 82.000 | 56.000 |
| ap_hi | 110.000 | 140.000 | 130.000 | 150.000 | 100.000 |
| ap_lo | 80.000 | 90.000 | 70.000 | 100.000 | 60.000 |
| cholesterol | 1.000 | 3.000 | 3.000 | 1.000 | 1.000 |
| gluc | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| smoke | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| alco | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| active | 1.000 | 1.000 | 0.000 | 1.000 | 0.000 |
| cardio | 0.000 | 1.000 | 1.000 | 1.000 | 0.000 |

[ ]:
```
df1.rename(columns = {'gluc': 'glucose', 'alco': 'alcohol_intake', 'smoke':␣
↪'smoker', 'alco': 'alcohol_intake', 'active': 'physical_activity'}, inplace␣
↪= True)
```

### 1.2.2 1.2 Data Dimensions

[ ]:
```
print(f'Number of rows: {df1.shape[0]} \nNumber of columns: {df1.shape[1]}')
```

```
Number of rows: 70000
Number of columns: 13
```

### 1.2.3 1.3 Check Data

[ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 70000 non-null  int64
 1   age                70000 non-null  int64
 2   gender             70000 non-null  int64
 3   height             70000 non-null  int64
 4   weight             70000 non-null  float64
 5   ap_hi              70000 non-null  int64
 6   ap_lo              70000 non-null  int64
 7   cholesterol        70000 non-null  int64
 8   glucose            70000 non-null  int64
 9   smoker             70000 non-null  int64
 10  alcohol_intake     70000 non-null  int64
 11  physical_activity  70000 non-null  int64
 12  cardio             70000 non-null  int64
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

### 1.2.4 1.4 Change Units

[ ]:
```
# Converting Age from days to years
df1['age'] = df1['age'].apply(lambda x: x / 365)
df1['age'] = df1['age'].astype(int)

# Converting Height from cm to m
df1['height'] = df1['height'].apply(lambda x: x / 100)
```

### 1.2.5 1.5 Descriptive Statistics

#### 1.5.1 SweetViz Report

[ ]:
```
feature_config = sv.FeatureConfig(skip = 'id', force_cat = ['gender',␣
↪'cholesterol', 'glucose', 'smoker', 'alcohol_intake', 'physical_activity'])
report_1 = sv.analyze([df1, 'Cardio Catch Diseases'], 'cardio', feature_config)
```

```
report_1.show_notebook(layout = 'vertical', scale = 0.9)
```

```
                                                  |           | [  0%]   00:00 -> (?
↪left)
```

<IPython.core.display.HTML object>

**Report Insights**

*Target*

1. `cardio`
    - the feature is balanced (50/50);
    - highest correlation ratios is `age`(0.24) and `weight`(0.18);
    - `cholesterol` and `glucose` seems to be a little relevant to the target;

*Numerical Features*

2. `age`
    - should be in year, not days;
    - highest correlation ratios are `cardio`(0.24) and `glucose`(0.16);
    - negative skewed curve;
    - negative kurtosis(platykurtic);
3. `height`
    - should be in m, not cm;
    - highest correlation ratios are `weight`(0.29), `gender`(0.5) and `smoker`(0.19);
    - negative skewed curve;
    - positive kurtosis(leptokurtic);
4. `weight`
    - highest correlation ratios are `height`(0.29), `cardio`(0.18) and `gender`(0.16);
    - positive skewed curve;
    - positive kurtosis(leptokurtic);
    - Highest weight is 200kg and min 10 **need attention**;
5. `ap_hi` (systolic blood pressure)
    - no strong correlation to be cited;
    - positive skewed curve;
    - positive kurtosis(leptokurtic);
    - avg and median are ok;
    - Highest weight is 16020 and min -150 **need attention**;
6. `ap_lo` (diastolic blood pressure)
    - no strong correlation to be cited;
    - positive skewed curve;
    - positive kurtosis(leptokurtic);
    - avg and median are ok;
    - Highest weight is 11000 and min -70 **need attention**;

*Categorical Features*

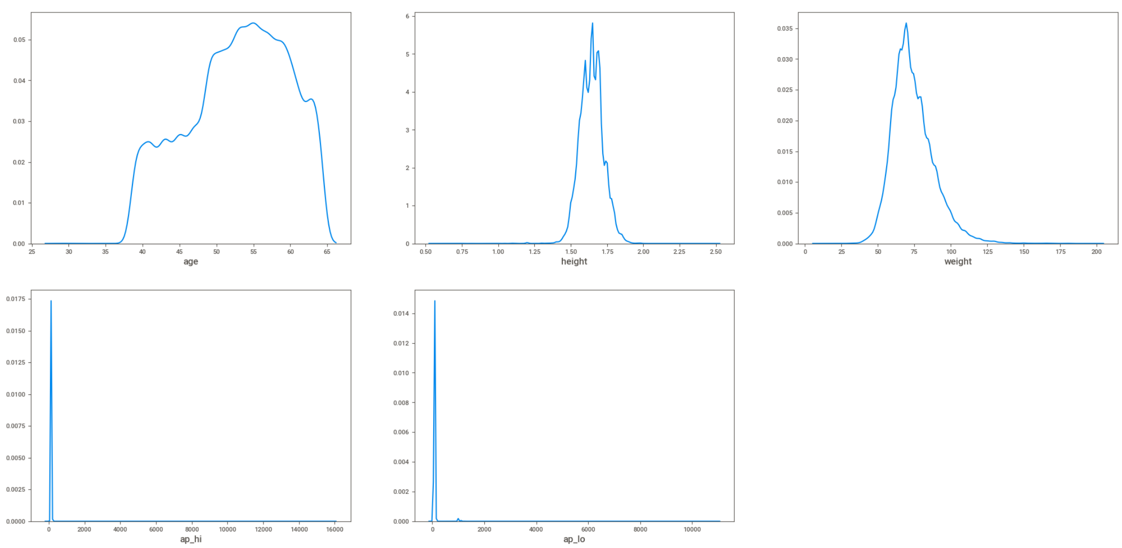
7. `gender`
    - unbalanced (65/35) but seems to be the normal in reality;
    - highest correlation ratios are `smoker`(0.19) and `height`(0.5);

8. `cholesterol`
    - unbalanced (75/14/12) but seems to be the normal in reality;
    - highest correlation ratios are `glucose`(0.19) and `age`(0.16);
9. `glucose`
    - unbalanced (85/8/7) but seems to be the normal in reality;
    - highest correlation ratios are `smoker`(0.14) and `height`(0.12);
10. `smoker`
    - unbalanced (91/9) but seems to be the normal in reality;
    - highest correlation ratios are `gender`(0.19), `smoker`(0.16) and `height`(0.19);
11. `alcohol_intake`
    - unbalanced (95/5) but seems to be the 'normal' in reality;
    - highest correlation ratios are `smoker`(0.16) and `height`(0.09);
12. `physical_activity`
    - unbalanced (80/20) but seems to be the normal in reality;
    - no strong correlation to be cited, this feature can be possibly dropped in the future steps;

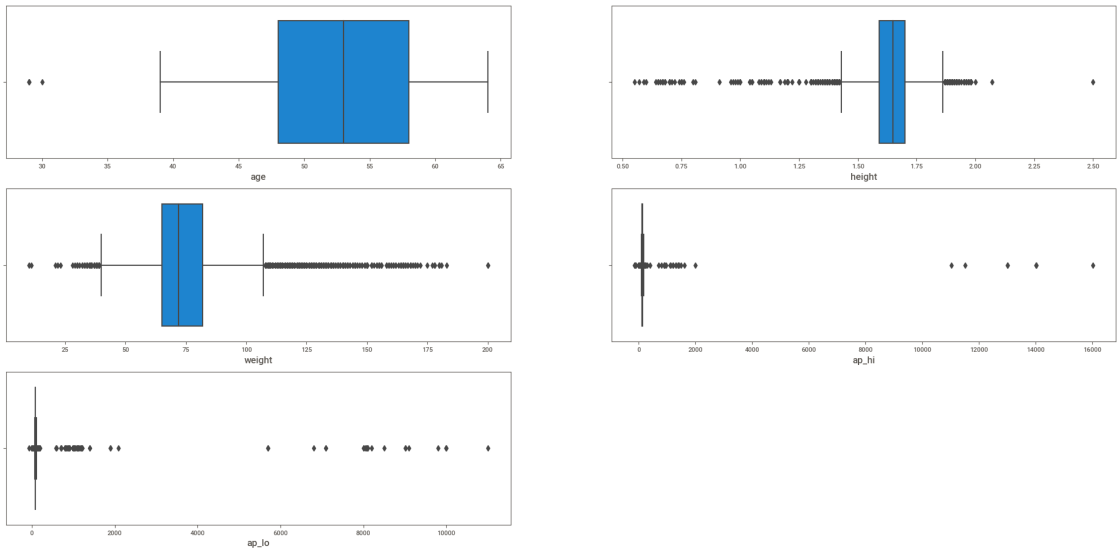
### 1.5.2 Numerical Attributes

[ ]:
```python
num_attributes = df1[['age', 'height', 'weight', 'ap_hi', 'ap_lo']]
```

[ ]:
```python
multiple_kdeplots(num_attributes, 2, 3);
```




[ ]:
```python
multiple_boxplots(num_attributes, 3, 2);
```




To-do: Outliers Study

### 1.5.3 Categorical Attributes

[ ]:
```
df1_cat = df1.copy()

df1_cat = cat_convert(df1_cat)

cat_attributes = df1_cat[['gender', 'cholesterol', 'glucose', 'smoker',␣
↪'alcohol_intake', 'physical_activity', 'cardio', 'cardio_result']]
cat_attributes.head()
```

[ ]:
```
  gender        cholesterol glucose smoker alcohol_intake physical_activity
cardio cardio_result
0    man             normal  normal     no             no               yes
0            no
1  woman  well above normal  normal     no             no               yes
1           yes
2  woman  well above normal  normal     no             no                no
1           yes
3    man             normal  normal     no             no               yes
1           yes
4  woman             normal  normal     no             no                no
0            no
```

[ ]:
```
categorical_metrics(cat_attributes, 'gender')
```

[ ]:
```
        absolute  percent %
woman      45530     65.043
```

```
man       24470     34.957
```

[ ]:
```
categorical_metrics(cat_attributes, 'cholesterol')
```

[ ]:
```
                   absolute  percent %
normal                52385     74.836
above normal           9549     13.641
well above normal      8066     11.523
```

[ ]:
```
categorical_metrics(cat_attributes, 'glucose')
```

[ ]:
```
                   absolute  percent %
normal                59479     84.970
well above normal      5331      7.616
above normal           5190      7.414
```

[ ]:
```
categorical_metrics(cat_attributes, 'smoker')
```

[ ]:
```
     absolute  percent %
no      63831     91.187
yes      6169      8.813
```

[ ]:
```
categorical_metrics(cat_attributes, 'alcohol_intake')
```

[ ]:
```
     absolute  percent %
no      66236     94.623
yes      3764      5.377
```

[ ]:
```
categorical_metrics(cat_attributes, 'physical_activity')
```

[ ]:
```
     absolute  percent %
yes     56261     80.373
no      13739     19.627
```

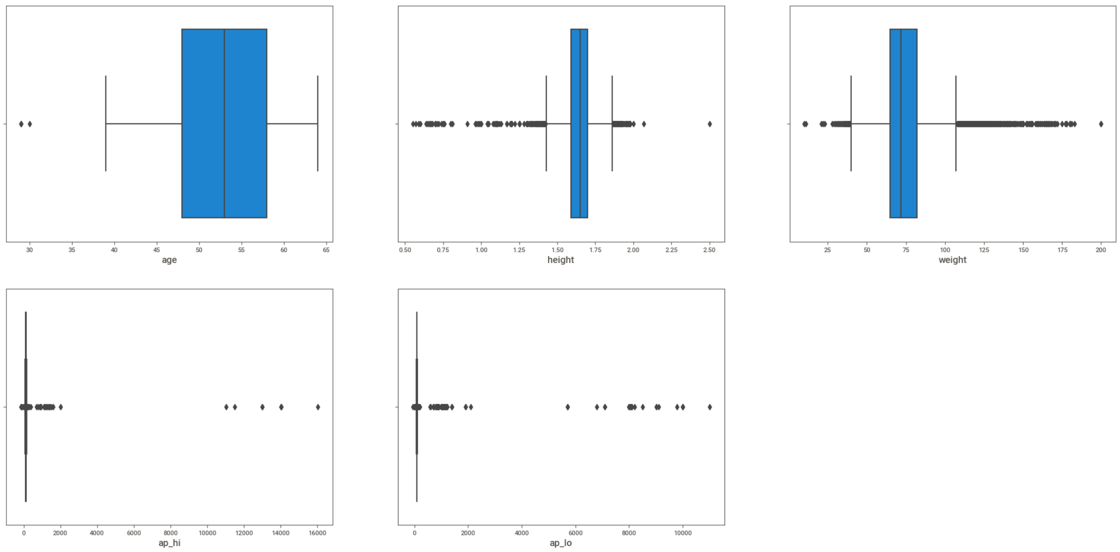
That info is strange, I believed that the percentage of people who exercised would be much lower

[ ]:
```
categorical_metrics(cat_attributes, 'cardio_result')
```

[ ]:
```
     absolute  percent %
no      35021     50.030
yes     34979     49.970
```

### 1.2.6 1.6 Data Filtering

[ ]:
```
multiple_boxplots(df1[['age', 'height', 'weight', 'ap_hi', 'ap_lo']], 2, 3);
```




#### 1.6.1 ap_hi and ap_lo

1. There are a lot of values at pressures above 400, 500, my theory is that this would be a typo where the person who measured would have added an extra 0 unintentionally.
2. Negative pressure values can be another typo, where add one - accidentally

[ ]:
```
df1[df1['ap_hi'] >= 400].head(3)
```

[ ]:
```
        id  age  gender  height  weight  ap_hi  ap_lo  cholesterol  glucose
smoker  alcohol_intake  physical_activity  cardio
1876  2654   41       1   1.600  60.000    902     60            1        1
0               0                 1      0
2014  2845   62       2   1.670  59.000    906      0            1        1
0               0                 1      0
4817  6822   39       1   1.680  63.000    909     60            2        1
0               0                 1      0
```

[ ]:
```
df1[df1['ap_hi'] < 0].head(3)
```

[ ]:
```
          id  age  gender  height  weight  ap_hi  ap_lo  cholesterol  glucose
smoker  alcohol_intake  physical_activity  cardio
4607    6525   41       1   1.650  78.000   -100     80            2        1
0               0                 1      0
16021  22881   60       2   1.610  90.000   -115     70            1        1
0               0                 1      0
20536  29313   42       1   1.530  54.000   -100     70            1        1
0               0                 1      0
```

[ ]:
```
df1[df1['ap_lo'] >= 400].head(3)
```

```
[ ]:       id  age  gender  height  weight  ap_hi  ap_lo  cholesterol  glucose
     smoker  alcohol_intake  physical_activity  cardio
     228  314   47       2   1.830  98.000    160   1100            1        2
     1               0                  1       1
     241  334   60       2   1.570  60.000    160   1000            2        1
     0               0                  0       1
     260  357   49       1   1.500  83.000    140    800            1        1
     0               0                  1       1
```

```
[ ]: df1[df1['ap_lo'] < 0].head(3)
```

```
[ ]:           id  age  gender  height  weight  ap_hi  ap_lo  cholesterol  glucose
     smoker  alcohol_intake  physical_activity  cardio
     60106  85816  61       1   1.670  74.000     15    -70            1        1
     0               0                  1       1
```

```
[ ]: df1['ap_hi'] = df1['ap_hi'].apply(lambda x: (x/10) if (x > 400) else x)
     df1['ap_hi'] = df1['ap_hi'].apply(lambda x: (-x) if (x < 0) else x)
     df1['ap_lo'] = df1['ap_lo'].apply(lambda x: (x/10) if (x > 400) else x)
     df1['ap_lo'] = df1['ap_lo'].apply(lambda x: (-x) if (x < 0) else x)
```

Considering that diastolic pressure values higher than 350 and less than 70 and systolic pressure values higher than 250 and less than 40 are practically impossible even in cases of patients with hypotension or hypertensive crisis, these lines will be removed in this project.

```
[ ]: len(df1[df1['ap_hi'] < 70])
```

```
[ ]: 183
```

```
[ ]: len(df1[df1['ap_hi'] > 350])
```

```
[ ]: 9
```

```
[ ]: len(df1[df1['ap_lo'] > 250])
```

```
[ ]: 24
```

```
[ ]: len(df1[df1['ap_lo'] < 30])
```

```
[ ]: 52
```

```
[ ]: # ap_hi
     df1 = df1[df1['ap_hi'].between(70, 350)]

     # ap_lo
     df1 = df1[df1['ap_lo'].between(40, 250)]
```

#### 1.6.2 Age

[ ]:
```
age_min = df1['age'].min()
age_max = df1['age'].max()

print(f'The youngest client is {age_min} years old and the oldest is {age_max}.
 ↪')
```

```
The youngest client is 29 years old and the oldest is 64.
```

Everything seems right around here

**1.6.3 Height** In general, people with dwarfism are shorter than 1.45 meters for men and 1.40 meters for women. Values less than 1 meter are considered extremely rare, and will be removed from this database as they are possibly typos.

[ ]:
```
df1['height'].min()
```

[ ]:
```
0.55
```

[ ]:
```
df_height = df1.loc[df1['height'] < 1]
len(df_height)
```

[ ]:
```
26
```

[ ]:
```
df_height
```

[ ]:
```
          id  age  gender  height  weight   ap_hi   ap_lo  cholesterol  glucose  \
smoker  alcohol_intake  physical_activity  cardio
224      309   59       2   0.760  55.000 120.000  80.000            1        1
0                0                      1       0
8171   11662   48       2   0.970 170.000 160.000 100.000           1        1
1                0                      1       1
12770  18218   53       1   0.750 168.000 120.000  80.000            1        1
1                0                      1       1
13265  18928   61       2   0.710  68.000 120.000  80.000            3        1
0                0                      1       0
14323  20459   60       1   0.670  57.000 120.000  90.000            1        1
0                0                      1       1
15167  21686   43       1   0.700  68.000 120.000  80.000            1        1
0                0                      0       0
16699  23859   53       2   0.740  98.000 140.000  90.000            1        1
0                0                      1       1
22542  32207   39       1   0.680  65.000 100.000  60.000            1        1
0                0                      0       0
22723  32456   64       1   0.550  81.000 130.000  90.000            1        1
0                0                      1       1
23913  34186   52       1   0.810 156.000 140.000  90.000            1        1
0                0                      1       0
```

```
27384  39156   41        1   0.800 178.000 140.000  90.000            3         3
0               0                  1        1
27603  39462   57        1   0.640  61.000 130.000  70.000            1         1
0               0                  1        0
28737  41075   54        1   0.910  55.000 140.000  90.000            1         1
0               0                  1        1
29157  41661   52        1   0.600  69.000 110.000  70.000            1         1
0               0                  0        0
32098  45832   42        1   0.720  74.000 150.000  90.000            1         1
0               0                  1        1
33607  48009   53        2   0.650  72.000 130.000  80.000            1         1
0               0                  0        0
44490  63545   52        1   0.650  60.000 120.000  80.000            1         1
0               0                  1        0
46319  66161   57        2   0.680  71.000 120.000  80.000            1         1
0               0                  1        0
47352  67631   63        1   0.750  75.000 120.000  80.000            1         1
0               0                  1        0
50789  72476   39        2   0.670  60.000 110.000  80.000            1         1
1               1                  1        0
51459  73386   42        2   0.700  69.000 120.000  80.000            1         1
0               0                  0        0
53344  76116   56        2   0.670  80.000 120.000  80.000            1         1
0               0                  0        1
56022  79917   58        1   0.960  59.000  90.000  60.000            1         1
0               0                  1        1
64115  91523   50        1   0.590  57.600 125.000  67.000            1         1
0               0                  0        0
65302  93223   50        1   0.990  60.000  90.000  60.000            1         1
0               0                  1        0
66643  95141   51        1   0.570  61.000 130.000  90.000            1         1
0               0                  1        1
```

[ ]:
```python
df1.drop(df1[df1['height'] < 1].index, inplace = True)
```

One of the sports with the tallest players is basketball, in which the biggest player who ever played in the NBA, Gheorghe Muresan, was 2.31m tall. Height values above this are very rare and will be removed from df1.

Reference

[ ]:
```python
df1['height'].max()
```

[ ]: 2.5

[ ]:
```python
df1.loc[df1['height'] > 2]
```

[ ]:
```
           id  age  gender  height  weight   ap_hi   ap_lo  cholesterol  glucose  \
smoker  alcohol_intake  physical_activity  cardio
6486     9223   58       1   2.500  86.000 140.000 100.000            3        1
0               0                  1       1
21628   30894   52       2   2.070  78.000 100.000  70.000            1        1
0               1                  1       0
```

```
[ ]: df1.drop(df1[df1['height'] == 2.5].index, inplace = True)
```

### 1.6.4 Weight

```
[ ]: df1['weight'].max()
```

```
[ ]: 200.0
```

```
[ ]: df1['weight'].min()
```

```
[ ]: 10.0
```

Values less than 35 kg look weird in this dataset, so they will be removed

```
[ ]: df_weight = df1.loc[df1['weight'] < 35]
     len(df_weight)
```

```
[ ]: 20
```

```
[ ]: df_weight
```

[ ]:
```
           id  age  gender  height  weight   ap_hi   ap_lo  cholesterol  glucose  \
smoker  alcohol_intake  physical_activity  cardio
3752     5306   42       1   1.200  30.000 110.000  70.000            1        1
0               0                  1       0
14722   21040   62       1   1.430  34.000 100.000  70.000            1        1
0               0                  1       0
16906   24167   47       2   1.700  31.000 150.000  90.000            2        2
0               0                  1       1
18559   26503   49       1   1.600  30.000 120.000  80.000            1        1
0               0                  1       1
22016   31439   42       1   1.460  32.000 100.000  70.000            1        1
0               0                  0       0
26806   38312   63       1   1.570  23.000 110.000  80.000            1        1
0               0                  1       0
29488   42156   55       2   1.770  22.000 120.000  80.000            1        1
1               1                  1       0
33511   47872   57       1   1.530  34.000 110.000  70.000            3        3
0               0                  1       1
33817   48318   59       2   1.780  11.000 130.000  90.000            1        1
0               0                  1       1
```

```
34276  48976   40        2    1.280  28.000 120.000  80.000            1        1
0               0                    1       0
35314  50443   54        1    1.460  32.000 130.000  80.000            1        2
0               0                    0       0
38417  54851   59        1    1.540  32.000 110.000  60.000            1        1
0               0                    1       0
41905  59853   58        1    1.430  30.000 103.000  61.000            2        1
0               0                    1       0
48080  68667   52        1    1.430  33.000 100.000  60.000            1        1
0               0                    1       0
51837  73914   54        2    1.390  34.000 120.000  70.000            1        1
0               0                    1       0
55852  79686   64        1    1.520  34.000 140.000  90.000            1        1
0               0                    1       1
57858  82567   51        2    1.650  10.000 180.000 110.000            2        2
0               0                    1       1
60188  85931   59        1    1.620  21.000 120.000  80.000            2        1
0               0                    1       1
60699  86650   51        1    1.710  29.000 110.000  70.000            2        1
0               0                    1       1
65082  92896   62        1    1.450  33.000 130.000 100.000            2        1
0               0                    1       1
```

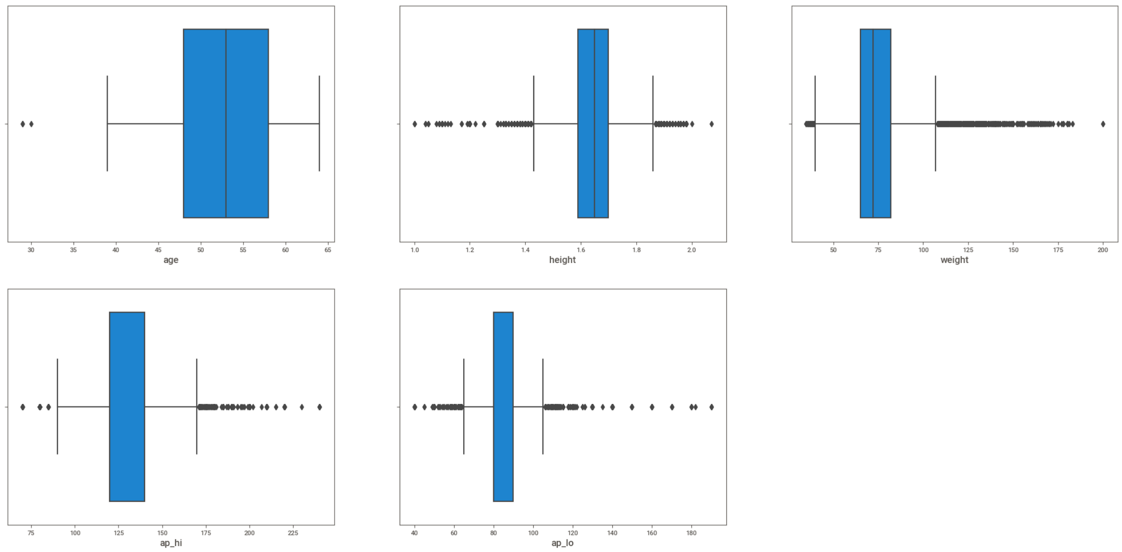
[ ]:
```
df1.drop(df1[df1['weight'] < 35].index, inplace = True)
```

### 1.6.5 After Filtering

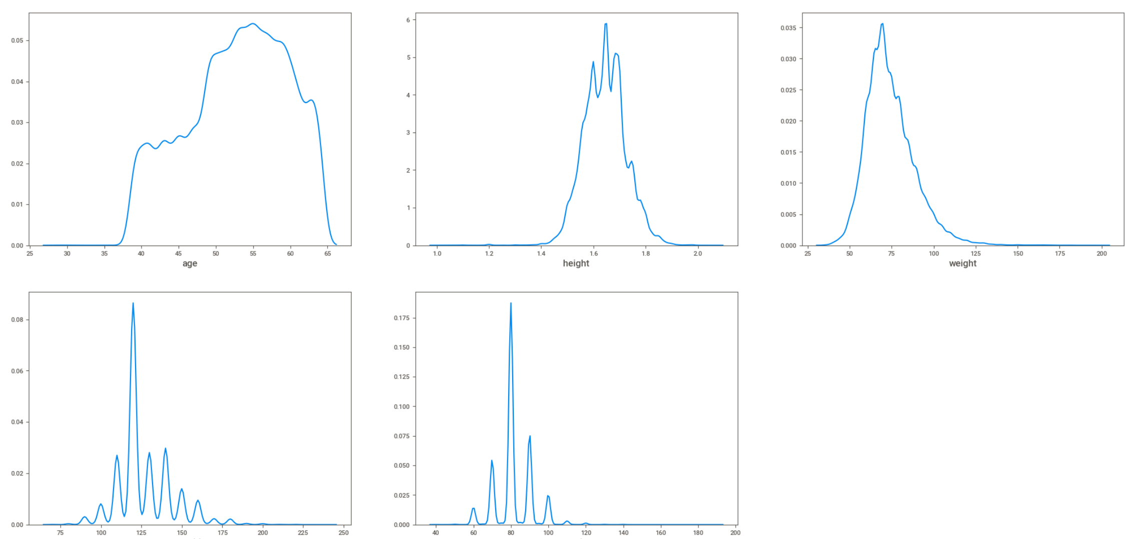
[ ]:
```
rows_lost = 70000 - len(df1)
print(f'In the data filtering, {rows_lost} rows  have been removed')
```

```
In the data filtering, 317 rows  have been removed
```

[ ]:
```
multiple_boxplots(df1[['age', 'height', 'weight', 'ap_hi', 'ap_lo']], 2, 3);
```




```
[ ]: multiple_kdeplots(df1[['age', 'height', 'weight', 'ap_hi', 'ap_lo']], 2, 3);
```




### 1.2.7 1.7 Save State

```
[ ]: df1.to_pickle(homepath + 'data/processed/df1.pkl')
```

## 1.3 2. FEATURE ENGINEERING

### 1.3.1 2.1 Load Checkpoint

[ ]:
```
df2 = pd.read_pickle(homepath + 'data/processed/df1.pkl')
```

### 1.3.2 2.2 Systolic and Diastolic Pressure

I noticed something interesting in the descriptive statistics, **some diastolic pressure values are higher than systolic pressure values**. My theory is: In the construction of this database someone inserted the values of these two columns swapped because the value of the systolic pressure is, by definition, greater than the diastolic pressure.

[ ]:
```
df2['pulse_pressure_range'] = df2['ap_hi'] - df2['ap_lo']

df2['systolic_pressure'] = df2[['ap_hi', 'ap_lo', 'pulse_pressure_range']].
 ↪apply(lambda x: x['ap_hi'] if x['pulse_pressure_range'] >= 0
                                                                        ␣
 ↪            else (x['ap_hi'] + (x['pulse_pressure_range']*(-1))), axis = 1)


df2['diastolic_pressure'] = df2[['ap_hi', 'ap_lo', 'pulse_pressure_range']].
 ↪apply(lambda x: x['ap_lo'] if x['pulse_pressure_range'] >= 0
                                                                        ␣
 ↪            else (x['ap_lo'] + (x['pulse_pressure_range'])), axis = 1)

df2['pulse_pressure_range'] = df2['pulse_pressure_range'].apply(lambda x: x if␣
 ↪x >= 0 else -x)
```

### 1.3.3 2.3 Blood Pressure

According to the American Heart Association, the ideal blood pressure range is 120/80 mm Hg. A person's blood pressure is expressed in two values – 120 and 80 in the previous case. The first value is the systolic blood pressure, while the value after the slash '/' symbol is the diastolic blood pressure.

- **Systolic blood pressure**: This unit indicates how much pressure blood exerts on the arterial walls when the heart beats at the time of measurement. This is when the heart pumps blood out of the heart and circulates it to various organs in the body.
- **Diastolic blood pressure**: This unit indicates how much pressure is exerted on the arterial walls when the heart rests between two beats. This is the period during which the heart opens its chamber to fill with blood.

In general, systolic blood pressure receives more medical attention. It is also an important risk factor for cardiovascular disease in older people. It is widely observed that systolic blood pressure increases steadily with age due to the increased stiffness of large arteries and plaque formation in the blood vessels. Under normal circumstances, blood pressure approaching 300 is hazardous. In various health forums, individuals have reported having experienced blood pressure above 250. Most of these individuals have also reported suffering from extreme medical conditions, such as a heavy buzzing in the ears, uncontrollably intense headaches, dizziness, and even loss of consciousness.

Reference

A sudden fall in blood pressure can be dangerous. A change of just 20 mm Hg — a drop from 110 systolic to 90 mm Hg systolic, for example, can cause dizziness and fainting when the brain fails to receive enough blood. And big drops, such as those caused by uncontrolled bleeding, severe infections or allergic reactions, can be life-threatening.

Reference

Blood pressure can be classified into four categories based on the readings from a sphygmomanometer:

- **Hypotension**: Systolic pressure reading lower than 90 and diastolic lower than 60;
- **Normal**: Systolic pressure reading between 90-120 and diastolic pressure reading between 60-80 is considered normal;
- **Pre-high blood pressure**: Systolic pressure reading between 120-140 and diastolic pressure reading between 80-90 is considered a slightly elevated level of blood pressure;
- **High blood pressure**: Systolic pressure reading between 140-180 and diastolic pressure reading between 90-100 is considered to be a high blood pressure condition;
- **Hypertensive crisis**: If one's systolic pressure exceeds 180 or diastolic pressure crosses 100, it is a stage that requires immediate medical attention.

[ ]:
```
df2['blood_pressure'] = df2.apply(lambda x:  'hypotension' if␣
↪(x['systolic_pressure'] < 90) and (x['diastolic_pressure'] < 60)
                                         else 'normal' if␣
↪(x['systolic_pressure'] >= 90 and x['systolic_pressure'] <= 120) and␣
↪(x['diastolic_pressure'] >= 60 and x['diastolic_pressure'] <= 80)
                                         else 'prehigh_blood_pressure' if␣
↪(x['systolic_pressure'] > 120 and x['systolic_pressure'] <= 140) or␣
↪(x['diastolic_pressure'] > 80 and x['diastolic_pressure'] <= 90)
                                         else 'high_blood_pressure' if␣
↪(x['systolic_pressure'] > 140 and x['systolic_pressure'] <= 180) or␣
↪(x['diastolic_pressure'] > 90 and x['diastolic_pressure'] <= 100)
                                         else 'hypertensive_crisis' if␣
↪(x['systolic_pressure'] > 180) and (x['diastolic_pressure'] > 100)
                                         else 'need_an_analysis', axis = 1)
```

[ ]:
```
categorical_metrics(df2, 'blood_pressure')
```

[ ]:

| | absolute | percent % |
|---|---|---|
| normal | 38785 | 55.659 |
| prehigh_blood_pressure | 24750 | 35.518 |
| high_blood_pressure | 5823 | 8.356 |
| hypertensive_crisis | 157 | 0.225 |
| need_an_analysis | 155 | 0.222 |
| hypotension | 13 | 0.019 |

### 1.3.4 2.3 Pulse Pressure

A normal pulse pressure range is between 40 and 60 mm Hg, values below 40 are considered low and above 60 high. Low pulse pressure can indicate decreased cardiac output. It's often observed in people with heart failure.As people age, it's common for their pulse pressure measurement to increase. This can be due to high blood pressure or atherosclerosis, fatty deposits that build up on your arteries. Additionally, iron deficiency anemia and hyperthyroidism can lead to an increase in pulse pressure.

Reference

```
[ ]: df2['pulse_pressure'] = df2['pulse_pressure_range'].apply(lambda x: 'low' if (x␣
     ↪>= 0) and (x < 40)
                                                              else 'normal' if (x␣
     ↪>= 40) and (x < 60)
                                                              else 'high')
```

```
[ ]: categorical_metrics(df2, 'pulse_pressure')
```

```
[ ]:         absolute  percent %
     normal     50728     72.798
     high       11692     16.779
     low         7263     10.423
```

### 1.3.5 2.4 BMI and Body Mass

The body mass index (BMI) is a measure that uses your height and weight to work out if your weight is healthy, it can be calculated with person's weight in kilograms divided by the square of height in meters. A high or low BMI may be an indicator of poor diet, varying activity levels or high stress but normal bmi, alone, doesn't mean healthy.

The BMI result will fit into one of 5 bands:

| Underweight | Normal | Overweight | Obese | Extremely Obese |
|---|---|---|---|---|
| Under 18.5 | Between 18.5 and 24.9 | Between 25 and 29.9 | Between 30 and 39.9 | 40 or over |

Health problems associated with a BMI in the obesity include:

- type 2 diabetes;
- stroke;
- heart disease;
- high blood pressure.

Health problems associated with a BMI in the underweight:

- weakened immuned system;
- anaemia;
- palpitations.

Reference

[ ]:
```
# BMI
df2['bmi'] = round(df2['weight']/(df2['height']**2), 1)

df2['body_mass'] = df2['bmi'].apply(lambda x: 'underweight' if (x < 18.5)
                                    else  'normal' if (x >= 18.5) and (x <␣
 ↪25)
                                    else  'overweight' if (x >= 25) and (x␣
 ↪< 30)
                                    else  'obese' if (x >= 30) and (x < 40)
                                    else 'extremely_obese')
```

[ ]:
```
categorical_metrics(df2, 'body_mass')
```

[ ]:

| | absolute | percent % |
|---|---|---|
| overweight | 25451 | 36.524 |
| normal | 25100 | 36.020 |
| obese | 16656 | 23.903 |
| extremely_obese | 1869 | 2.682 |
| underweight | 607 | 0.871 |

### 1.3.6 2.5 Period of Life

Reference

[ ]:
```
age_min = df2['age'].min()
age_max = df2['age'].max()

print(f'The smallest age in this dataset is {age_min} years.\nThe highest age␣
 ↪in this dataset is {age_max} years.')
```

```
The smallest age in this dataset is 29 years.
The highest age in this dataset is 64 years.
```

[ ]:
```
df2['period_of_life'] = df2['age'].apply(lambda x: 'early_adulthood' if (x >=␣
 ↪29) and (x < 40)
                                        else   'middle_adulthood')
```

[ ]:
```
categorical_metrics(df2, 'period_of_life')
```

[ ]:

| | absolute | percent % |
|---|---|---|
| middle_adulthood | 67905 | 97.448 |
| early_adulthood | 1778 | 2.552 |

### 1.3.7 2.5 Drop Columns

[ ]:
```
df2.drop(columns = ['ap_hi', 'ap_lo', 'pulse_pressure_range'], inplace = True)
```

### 1.3.8 2.6 Save State

[ ]:
```
df2.to_pickle(homepath + 'data/processed/df2.pkl')
```

## 1.4 3. EXPLORATORY DATA ANALYSIS

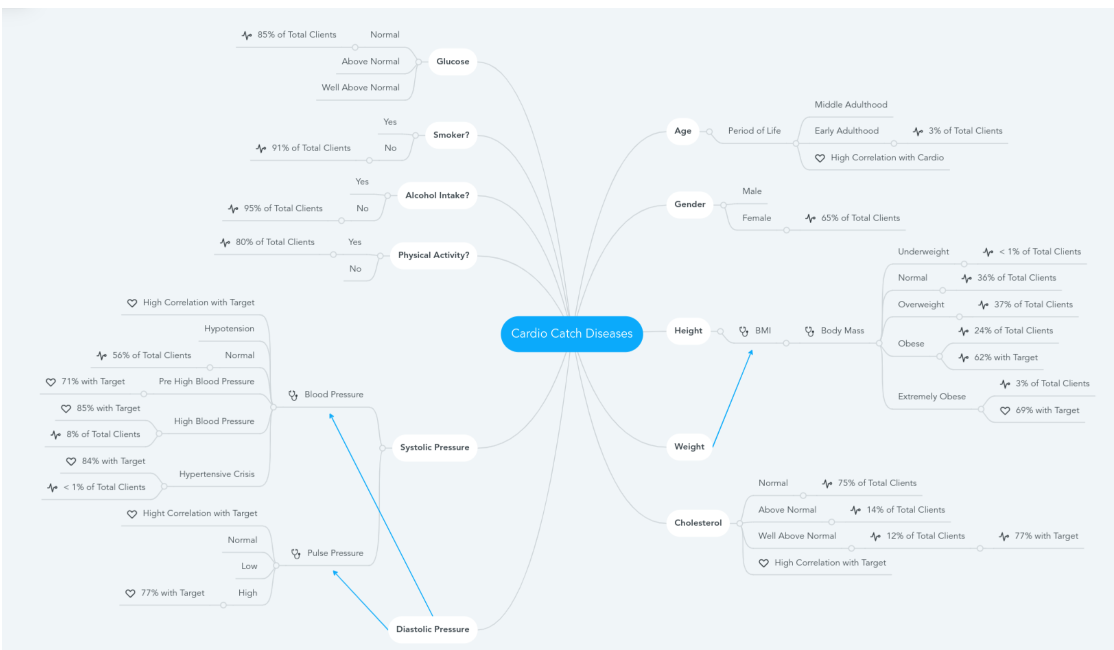
### 1.4.1 3.1 Load Checkpoint

[ ]:
```
df3 = pd.read_pickle(homepath + 'data/processed/df2.pkl')
```

### 1.4.2 3.2 Mind Map

[ ]:
```
Image(homepath + 'reports/figures/mindmap.png')
```

[ ]:




### 1.4.3 3.3 Dataframe for EDA

[ ]:
```
df3 = cat_convert(df3)
df3.drop(columns = 'id', inplace = True)
```

[ ]:
```
df3.head()
```

```
[ ]:    age gender  height  weight        cholesterol glucose smoker alcohol_intake
    physical_activity  cardio  systolic_pressure  diastolic_pressure
    blood_pressure pulse_pressure    bmi    body_mass    period_of_life cardio_result
    0   50    man   1.680  62.000              normal  normal     no             no
    yes       0            110.000               80.000                 normal
    low 22.000      normal  middle_adulthood            no
    1   55   woman   1.560  85.000  well above normal  normal     no             no
    yes       1            140.000               90.000  prehigh_blood_pressure
    normal 34.900       obese  middle_adulthood            yes
    2   51   woman   1.650  64.000  well above normal  normal     no             no
    no       1            130.000               70.000  prehigh_blood_pressure
    high 23.500      normal  middle_adulthood            yes
    3   48    man   1.690  82.000              normal  normal     no             no
    yes       1            150.000              100.000     high_blood_pressure
    normal 28.700  overweight  middle_adulthood            yes
    4   47   woman   1.560  56.000              normal  normal     no             no
    no       0            100.000               60.000                 normal
    normal 23.000      normal  middle_adulthood            no
```

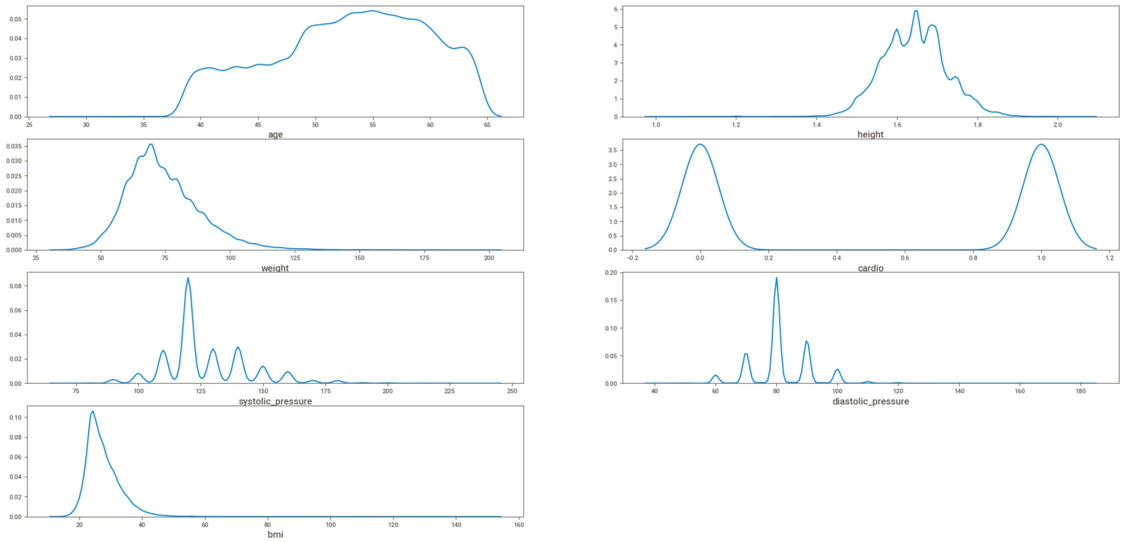
### 1.4.4 3.4 EDA Report

```
[ ]: feature_config = sv.FeatureConfig(skip = 'id', force_cat = ['gender',␣
      ↪'cholesterol', 'glucose', 'smoker', 'alcohol_intake', 'physical_activity',␣
      ↪'blood_pressure', 'body_mass', 'period_of_life'])
     report_2 = sv.analyze([df2, 'Cardio Catch Diseases'], 'cardio', feature_config)

     report_2.show_notebook(layout = 'vertical', scale = 0.9)
```

```
                                              |          | [  0%]   00:00 -> (?␣
    ↪left)

<IPython.core.display.HTML object>
```

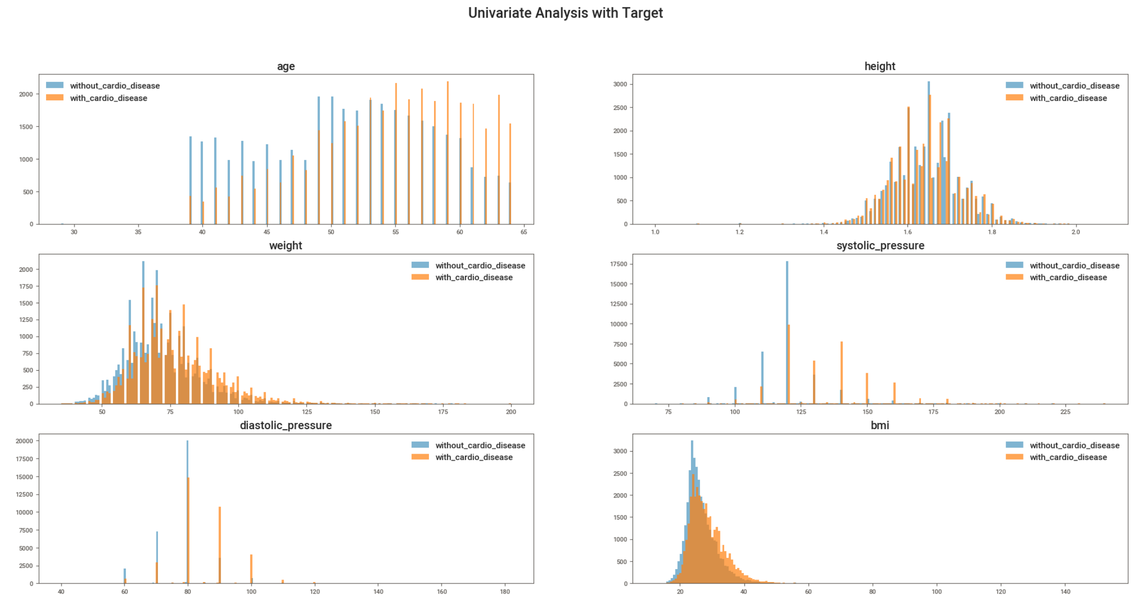
### 1.4.5 3.5 Univariate Analysis

```
[ ]: univariate_numerical = df3.select_dtypes(include = ['int64', 'float64'])
     multiple_kdeplots(univariate_numerical, 4, 2);
```




[ ]:
```python
# creating two datasets, one with cardio disease and other without hue
with_cardio_disease = univariate_numerical[univariate_numerical['cardio'] == 1]
without_cardio_disease = univariate_numerical[univariate_numerical['cardio'] == 0]

# creating subplots
fig, ax = plt.subplots(3, 2)

plot_with_target(without_cardio_disease, with_cardio_disease,
                 'age', 0, 0, 'without_cardio_disease', 'with_cardio_disease');

plot_with_target(without_cardio_disease, with_cardio_disease,
                 'height', 0, 1, 'without_cardio_disease', 'with_cardio_disease');

plot_with_target(without_cardio_disease, with_cardio_disease,
                 'weight', 1, 0, 'without_cardio_disease', 'with_cardio_disease');

plot_with_target(without_cardio_disease, with_cardio_disease,
                 'systolic_pressure', 1, 1, 'without_cardio_disease', 'with_cardio_disease');

plot_with_target(without_cardio_disease, with_cardio_disease,
                 'diastolic_pressure', 2, 0, 'without_cardio_disease', 'with_cardio_disease');

plot_with_target(without_cardio_disease, with_cardio_disease,
```

```
                    'bmi', 2, 1, 'without_cardio_disease', 'with_cardio_disease');
fig.suptitle('Univariate Analysis with Target', fontsize = 18);
```

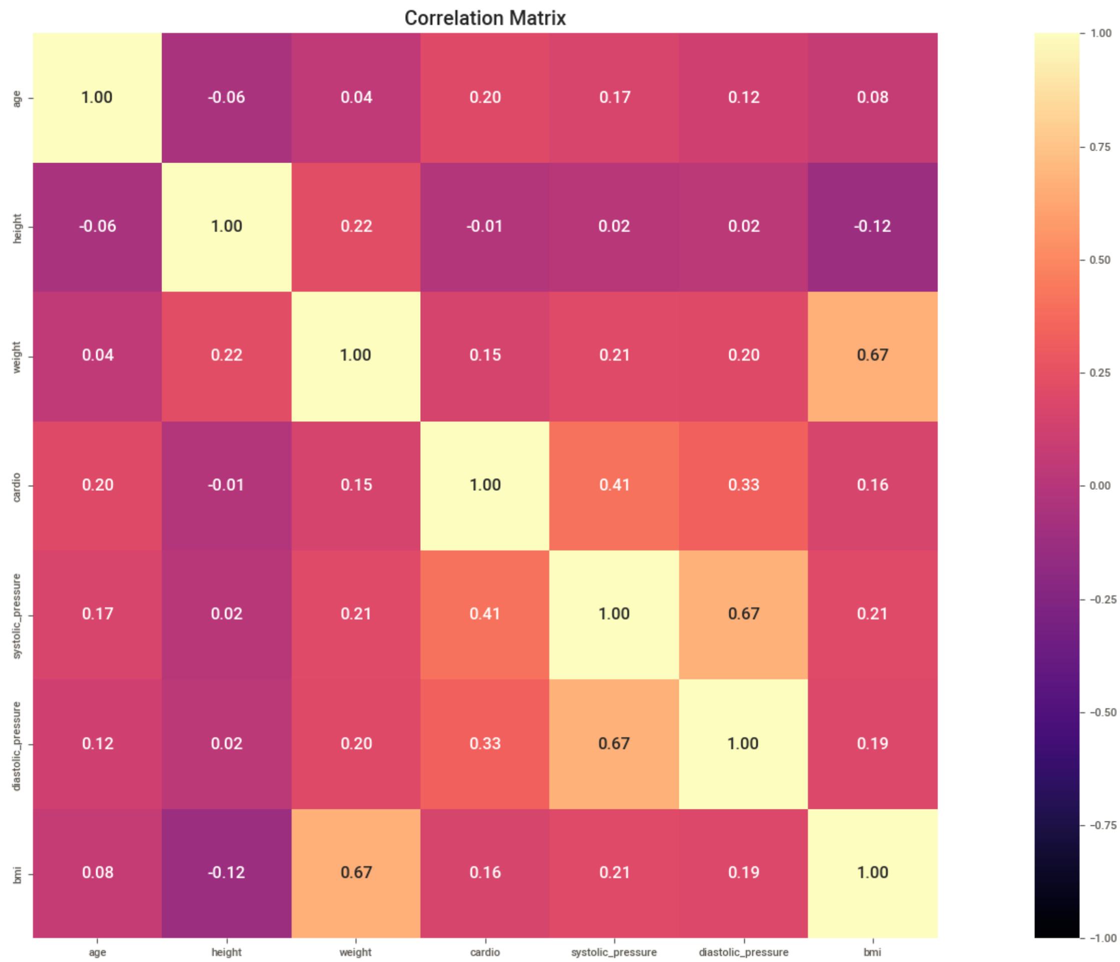



### 1.4.6 3.6 Bivariate Analysis

### 1.4.7 3.7 Multivariate Analysis

#### 3.7.1 Numerical Features

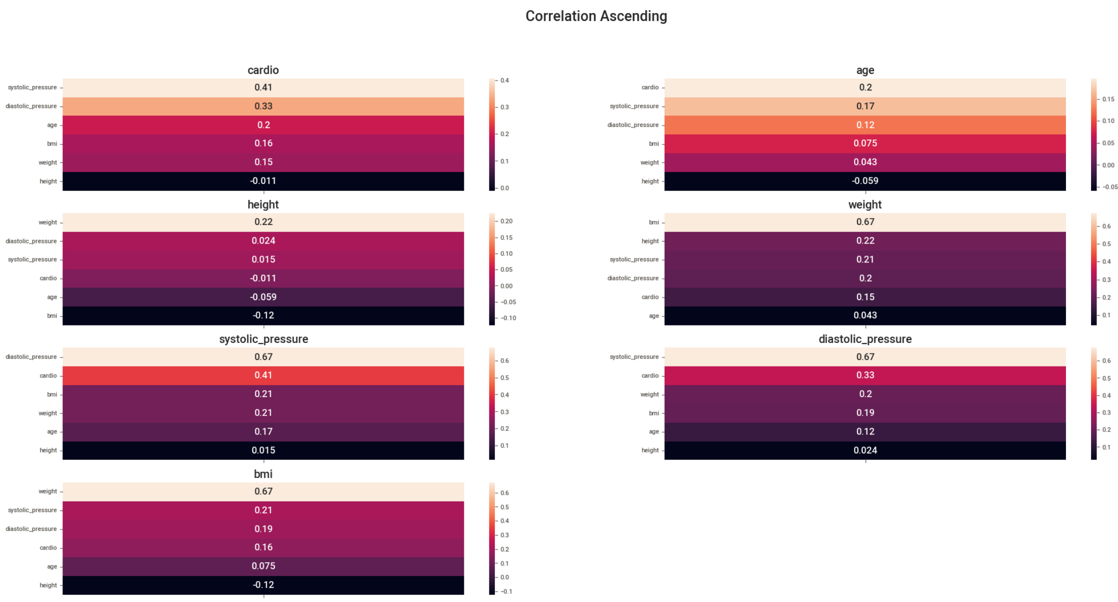
[ ]:
```
correlation_matrix(df3, 'kendall');
```




[ ]:

```
# creating subplots
fig, ax = plt.subplots()

plt.subplot(4, 2, 1)
correlation_ascending(df3, 'cardio', 'kendall');

plt.subplot(4, 2, 2)
correlation_ascending(df3, 'age', 'kendall');

plt.subplot(4, 2, 3)
correlation_ascending(df3, 'height', 'kendall');

plt.subplot(4, 2, 4)
correlation_ascending(df3, 'weight', 'kendall');

plt.subplot(4, 2, 5)
correlation_ascending(df3, 'systolic_pressure', 'kendall');
```

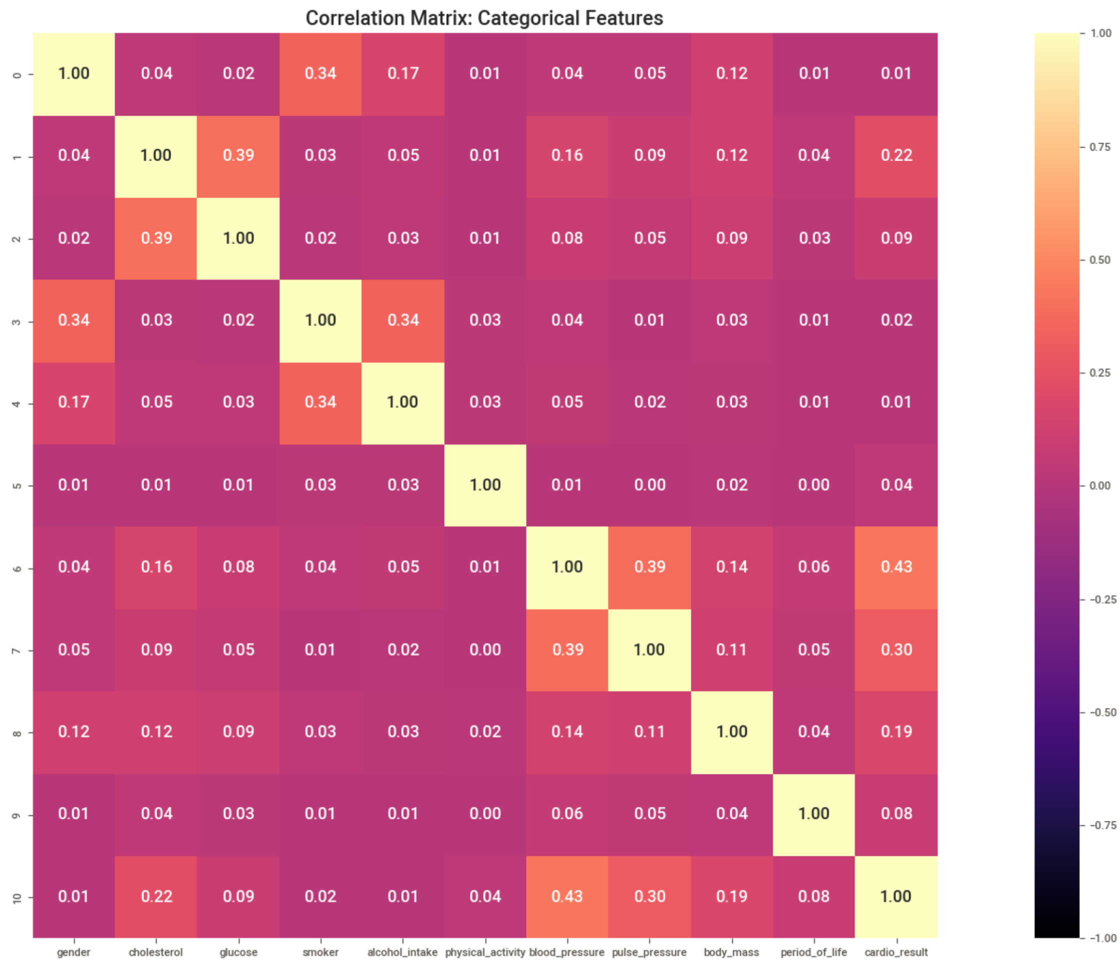
```
plt.subplot(4, 2, 6)
correlation_ascending(df3, 'diastolic_pressure', 'kendall');

plt.subplot(4, 2, 7)
correlation_ascending(df3, 'bmi', 'kendall');
fig.suptitle('Correlation Ascending', fontsize = 18);
```




### 3.7.2 Categorical Features

[ ]:
```
corr_cat(df3)
```

[ ]: [Text(0.5, 1.0, 'Correlation Matrix: Categorical Features')]




### 1.4.8 3.8 Conclusion

## 1.5 4. DATA PREPARATION

### 1.5.1 4.1 Load Checkpoint

```
[ ]: df4 = pd.read_pickle(homepath + 'data/processed/df2.pkl')
```

### 1.5.2 4.2 Split Data

```
[ ]: # Train -> 70%
     # Test -> 20%
     # Validation -> 10%
     x_train, x_test = train_test_split(df4, test_size = 0.2, random_state = seed)

     # if train = 80% x dataset, then val = 0.1/0.8 = 0.125
     # 0.125 x 0.8 = 0.1
     x_train, x_val = train_test_split(df4, test_size = 0.125, random_state = seed)
```

### 1.5.3 4.3 Encoding

```
[ ]: # Ordinal Encoding
     bp_encoding = {'need_an_analysis': 0, 'hypothension': 1, 'normal': 2,␣
      ↪'prehigh_blood_pressure': 3, 'high_blood_pressure': 4, 'hypertensive_crisis':
      ↪ 5}
     pp_encoding = {'measurement_error': 0, 'low': 1, 'normal': 2, 'high': 3}
     bm_encoding = {'underweight': 1, 'normal': 2, 'overweight': 3, 'obese': 4,␣
      ↪'extremely_obese': 5}
     pl_encoding = {'early_adulthood': 1, 'middle_adulthood': 2}

     # One Hot Encoding
     x_train, x_test, x_val = [pd.get_dummies(dataframe, prefix = ['gender'],␣
      ↪columns = ['gender']) for dataframe in [x_train, x_test, x_val]]



     for df in [x_train, x_test, x_val]:
         df['blood_pressure'] = df['blood_pressure'].map(bp_encoding)
         df['pulse_pressure'] = df['pulse_pressure'].map(pp_encoding)
         df['body_mass'] = df['body_mass'].map(bm_encoding)
         df['period_of_life'] = df['period_of_life'].map(pl_encoding)
```

### 1.5.4 4.4 Rescaling

```
[ ]: num_attributes = df4[['age', 'height', 'weight', 'systolic_pressure',␣
      ↪'diastolic_pressure', 'bmi']]
```

```
[ ]: multiple_kdeplots(num_attributes, 3, 2);
```

```
[ ]: multiple_boxplots(num_attributes, 3, 2);
```

1. Without Outlier and With Normal Distribution: Standard Scaler (Mean and std deviation)
2. With Outliers and Normal Distribution: Robust Scaler (Quartile)
3. Without Normal Distribution: MinMax Scaler

```
[ ]: # scaler
mms = MinMaxScaler()
rs = RobustScaler()

min_max_scaler = ['age', 'height', 'systolic_pressure', 'diastolic_pressure',␣
 ↪'bmi']
robust_scaler = ['weight']

for col in min_max_scaler:
    pickle.dump(mms, open(homepath + f'src/features/{col}_scaler.pkl', 'wb'))
    for dataframe in [x_train, x_test, x_val]:
        dataframe[[col]] = mms.fit_transform(dataframe[[col]].values)

for col in robust_scaler:
    pickle.dump(rs, open(homepath + f'src/features/{col}_scaler.pkl', 'wb'))
    for dataframe in [x_train, x_test, x_val]:
        dataframe[[col]] = rs.fit_transform(dataframe[[col]].values)
```

### 1.5.5 4.5 Split X and y

```
[ ]: # target: cardio
y_train, y_test, y_val = [dataframe['cardio'] for dataframe in [x_train,␣
 ↪x_test, x_val]]
ids_train, ids_test, ids_val = [dataframe['id'] for dataframe in [x_train,␣
 ↪x_test, x_val]]
X_train, X_test, X_val = [dataframe.drop(columns = ['cardio', 'id']) for␣
 ↪dataframe in [x_train, x_test, x_val]]
```

### 1.5.6 4.6 Feature Importance

#### 4.6.1 Feature Importance with ExtraTrees

```
[ ]: et_importance = ExtraTreesClassifier(n_estimators = 100, random_state = seed,␣
 ↪n_jobs = 6)
et_importance.fit(X_train, y_train)

# feature importance dataframe
feature_selection = pd.DataFrame({'feature': X_train.
 ↪columns,'feature_importance': et_importance.feature_importances_})\
                                .sort_values('feature_importance', ascending =␣
 ↪False).reset_index(drop = True)

# plot feature importance
ax = sns.barplot(x = 'feature_importance', y = 'feature', data =␣
 ↪feature_selection, orient = 'h')
ax.set_title('Feature Importance with ExtraTrees');
```

### 4.6.2 Feature Importance with RandomForest

```
[ ]: rf_importance = RandomForestClassifier(n_estimators = 100, random_state = seed,␣
      ↪n_jobs = 6)
     rf_importance.fit(X_train, y_train)

     # feature importance dataframe
     feature_selection = pd.DataFrame({'feature': X_train.
      ↪columns,'feature_importance': rf_importance.feature_importances_})\
                                      .sort_values('feature_importance', ascending =␣
      ↪False).reset_index(drop = True)

     # plot feature importance
     ax = sns.barplot(x = 'feature_importance', y = 'feature', data =␣
      ↪feature_selection, orient = 'h')
     ax.set_title('Feature Importance with RandomForest');
```

### 4.6.3 Rank Features

```
[ ]: visualizer_1d = Rank1D(algorithm = 'shapiro', size = (1024, 768))
     visualizer_1d.fit(X_train, y_train)
     visualizer_1d.transform(X_train)
     visualizer_1d.finalize();
```

To-do:

- p-value alert

### 4.6.4 Boruta

```
[ ]: rf_boruta = RandomForestClassifier(n_estimators = 500, n_jobs = 6, random_state␣
      ↪= seed)

     boruta = BorutaPy(rf_boruta, n_estimators = 'auto', random_state = seed).
      ↪fit(X_train.values, y_train.values)
```

```
[ ]: cols_selected = boruta.support_.tolist()

     cols_selected_boruta = X_train.iloc[:, cols_selected].columns.to_list()

     cols_not_selected_boruta = list(np.setdiff1d(X_train.columns,␣
      ↪cols_selected_boruta))
```

```
[ ]: cols_selected_boruta
```

```
[ ]: cols_not_selected_boruta
```

### 1.5.7 4.7 Save State

```
[ ]: X_train.to_pickle(homepath + 'data/processed/X_train.pkl')
X_test.to_pickle(homepath + 'data/processed/X_test.pkl')
X_val.to_pickle(homepath + 'data/processed/X_val.pkl')
y_train.to_pickle(homepath + 'data/processed/y_train.pkl')
y_test.to_pickle(homepath + 'data/processed/y_test.pkl')
y_val.to_pickle(homepath + 'data/processed/y_val.pkl')
ids_train.to_pickle(homepath + 'data/processed/ids_train.pkl')
ids_test.to_pickle(homepath + 'data/processed/ids_test.pkl')
ids_val.to_pickle(homepath + 'data/processed/ids_val.pkl')
```

## 1.6 5. MACHINE LEARNING MODELLING

```
[ ]: X_train = pd.read_pickle(homepath + 'data/processed/X_train.pkl')
X_val = pd.read_pickle(homepath + 'data/processed/X_val.pkl')
y_train = pd.read_pickle(homepath + 'data/processed/y_train.pkl')
y_val = pd.read_pickle(homepath + 'data/processed/y_val.pkl')
```

```
[ ]: models = [
          RandomForestClassifier(random_state = seed, n_jobs = 7),
          ExtraTreesClassifier(random_state = seed, n_jobs = 7),
          GaussianNB(),
          XGBClassifier(random_state = seed, n_jobs = 7, eval_metric =␣
 ↪'logloss', use_label_encoder = False),
          CatBoostClassifier(random_state = seed, verbose = False),
          AdaBoostClassifier(random_state = seed),
          LGBMClassifier(random_state = seed, n_jobs = 7),
          KNeighborsClassifier(n_neighbors = 7, n_jobs = 7),
          GradientBoostingClassifier(random_state = seed)
         ]
```

```
[ ]: metrics_result = metrics_df(models, 'cardio', X_train, y_train, X_val, y_val)
```

```
[ ]: metrics_result
```

```
[ ]: cv_result = cross_validation(models, 'cardio', X_train, y_train)
```

```
[ ]: cv_result
```

## 1.7 6. HYPERPARAMETER FINE TUNNING

## 1.8 7. BUSINESS TRANSLATION