Cardiovascular Disease

Salma Al-Arfaj

Data points are trying to tell you something about reality, be careful not to trust outliers !!

Salma Alarfaj ☺

Loading Dataset and inspecting Data frame

```
# Reading the Dataset >>> Add your path of the file
df = pd.read_csv('cardio_train.csv', sep = ';')
```

```
#df = pd.read_csv('../input/cardiovascular-disease-dataset/cardio_train.csv',sep=';')
df.head()
```

	id	age	gender	height	weight	ap_hi	ap_lo	cholesterol	gluc	smoke	alco	active	cardio
0	0	18393	2	168	62.0	110	80	1	1	0	0	1	0
1	1	20228	1	156	85.0	140	90	3	1	0	0	1	1
2	2	18857	1	165	64.0	130	70	3	1	0	0	0	1
3	3	17623	2	169	82.0	150	100	1	1	0	0	1	1
4	4	17474	1	156	56.0	100	60	1	1	0	0	0	0

```
df.shape
```

```
(70000, 13)
```

```
df.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   gluc         70000 non-null  int64  
 9   smoke        70000 non-null  int64  
 10  alco         70000 non-null  int64  
 11  active       70000 non-null  int64  
 12  cardio       70000 non-null  int64  
dtypes: float64(1), int64(12)
memory usage: 6.9 MB
```

Preparation and Feature Generation

```python
# dropping id column
df.drop(columns=['id'], inplace=True)
```

```python
# changing the age column into year we will divide the age by 365.25 and round them
df['age'] = df['age'].map(lambda x: round(x/365.25))
```

(1)

```python
def age_groups (age):
        if (age >=30 and a
            return 'from_3
        elif (age>=35 and
            return 'from_3
        elif (age>=40 and
            return 'from_4
        elif (age>=45 and
            return 'from_4
        elif (age>=50 and
            return 'from_5
        elif (age>=55 and
            return 'from_5
        else:
            return 'from_60_to_65'
```

(2)

```python
def bp_cat
    if x<=
        re
    elif x
        re
    elif x
        re
    elif x
        re
    elif x
        re
    else:
        re
```

(3)

```python
def bmi_groups(bmi):
    if   bmi <18.5:
        return "Underweight"
    elif bmi >= 18.5 and bmi < 25 :
        return "Normal weight"
    elif bmi >= 25 and bmi < 30:
        return "Pre-obesity"
    elif bmi >= 30 and bmi < 35:
        return "Obesity Class 1 (Moderately obese)"
    elif bmi >= 35 and bmi < 40:
        return "Obesity Class 2 (Severely obese)"
    elif bmi >= 40:
        return "Obesity Class 3 (Very severely obese)"
```

Impossible values and Data Cleaning

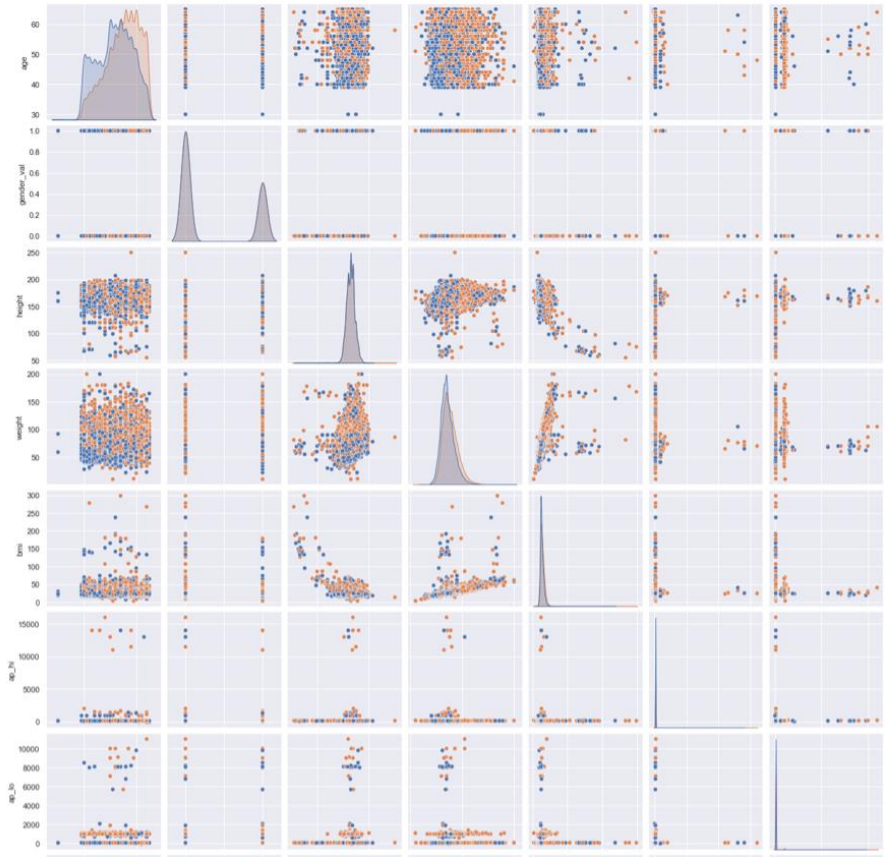
```
#checking for duplicates
df.duplicated() #.sum()
```

```
0         False
1         False
2         False
3         False
4         False
          ...
69995     False
69996     False
69997     False
69998     False
69999     False
Length: 70000, dtype: bool
```

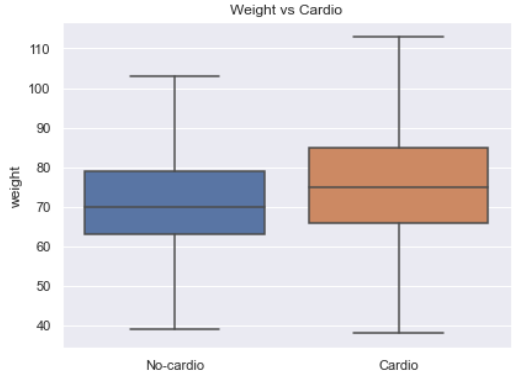
```
# Statistic Overview after correcting for the Age column horizontally
df.describe().T
```

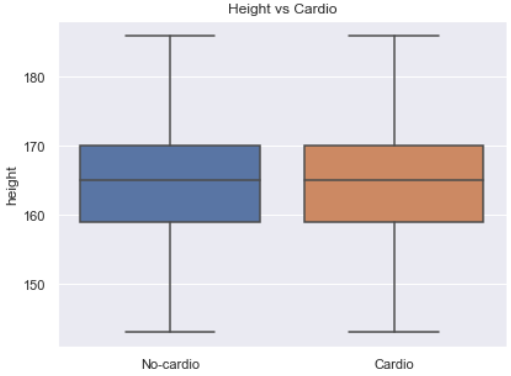
	count	mean	std	min	25%	50%	75%	max
age	70000.0	53.303157	6.760171	30.00	48.00	54.000	58.00	65.00
gender_val	70000.0	0.349571	0.476838	0.00	0.00	0.000	1.00	1.00
height	70000.0	164.359229	8.210126	55.00	159.00	165.000	170.00	250.00
weight	70000.0	74.205690	14.395757	10.00	65.00	72.000	82.00	200.00
bmi	70000.0	27.556545	6.091405	3.47	23.88	26.375	30.22	298.67
ap_hi	70000.0	128.817286	154.011419	-150.00	120.00	120.000	140.00	16020.00
ap_lo	70000.0	96.630414	188.472530	-70.00	80.00	80.000	90.00	11000.00
cholesterol	70000.0	1.366871	0.680250	1.00	1.00	1.000	2.00	3.00
gluc	70000.0	1.226457	0.572270	1.00	1.00	1.000	1.00	3.00
smoke	70000.0	0.088129	0.283484	0.00	0.00	0.000	0.00	1.00
alco	70000.0	0.053771	0.225568	0.00	0.00	0.000	0.00	1.00
active	70000.0	0.803729	0.397179	0.00	1.00	1.000	1.00	1.00
cardio	70000.0	0.499700	0.500003	0.00	0.00	0.000	1.00	1.00

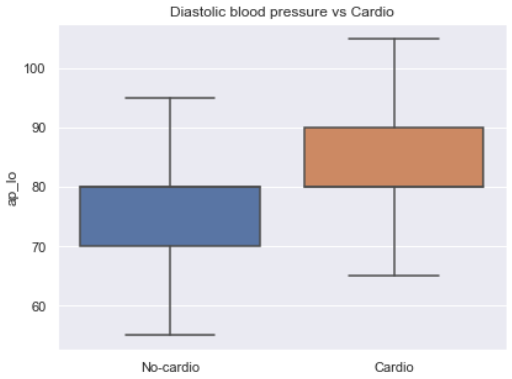
we visualize our findings via plots

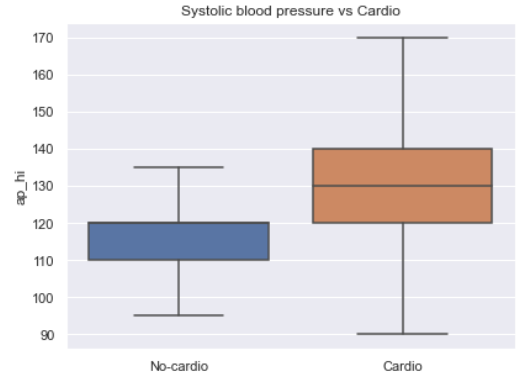


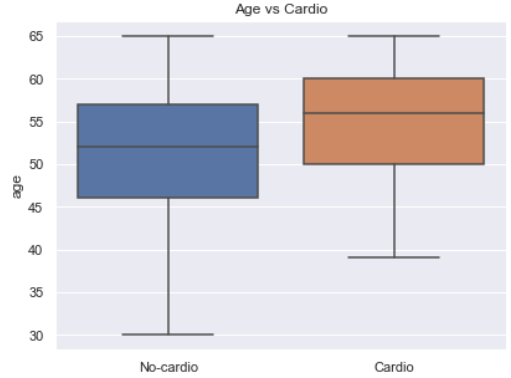
Box Plots

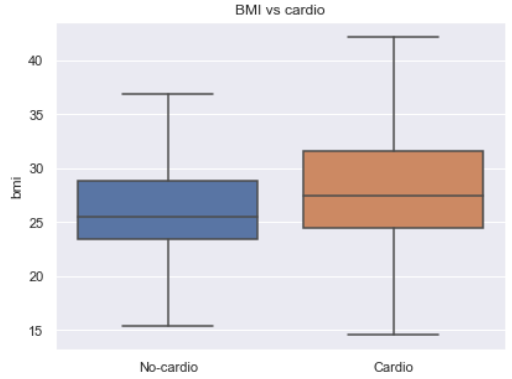

Weight vs Cardio
weight
110
100
90
80
70
60
50
40
No-cardio
Cardio


Height vs Cardio
height
180
170
160
150
No-cardio
Cardio


Diastolic blood pressure vs Cardio
ap_lo
100
90
80
70
60
No-cardio
Cardio


Systolic blood pressure vs Cardio
ap_hi
170
160
150
140
130
120
110
100
90
No-cardio
Cardio


Age vs Cardio
age
65
60
55
50
45
40
35
30
No-cardio
Cardio


BMI vs cardio
bmi
40
35
30
25
20
15
No-cardio
Cardio

Drop Outliers and Impossible Values

```
# inspecting the zero values in the denominator of bmi ap_lo
mask0 = (df.ap_lo == 0)
df[mask0]
```

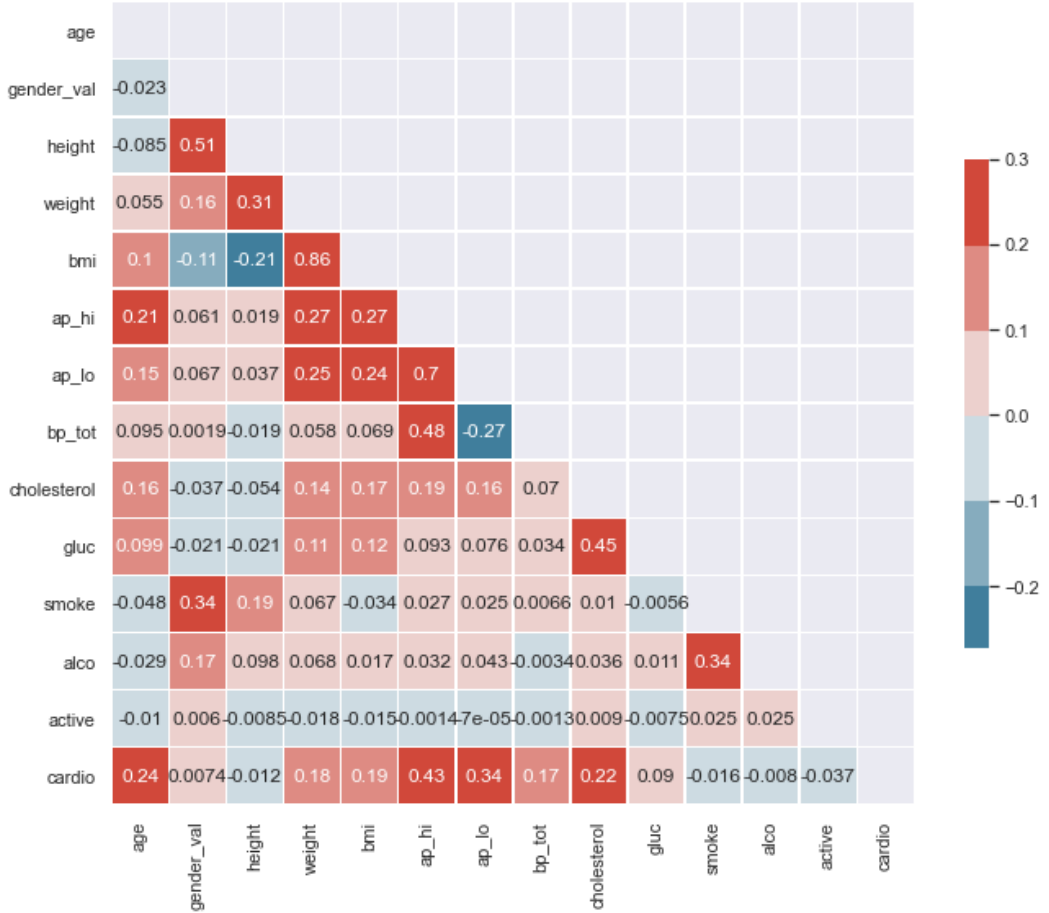
```
# diving for more information about (min, max) values of weight
```

```
#Dropping Outliers and Impossible data points
df.drop(df.query('bmi >60 or bmi <15').index, axis=0, inplace=True)
df.drop(df.query('ap_hi > 220 or ap_lo >180 or ap_hi<40 or ap_lo<40').index,
axis=0, inplace=True)
df.drop(df.query('height < 100 or height > 200').index, axis=0, inplace=True)
df.drop(df.query('weight < 30 or weight > 200').index, axis=0, inplace=True)
```

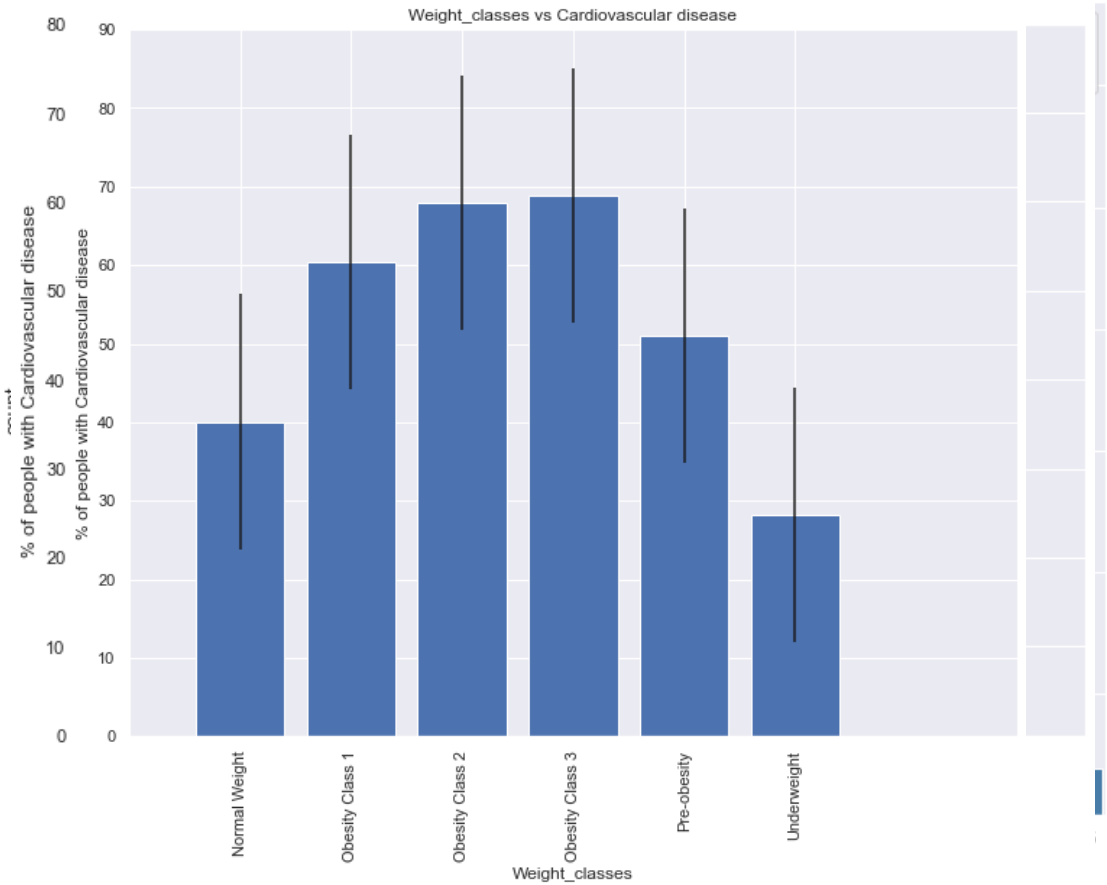
```
df[mask3].count() # we find 31 data point
```

```
# diving for more information about (min, max) values of ap_hi reference (min= 50, max= 210)
mask12 = ((df.bmi < 15) | (df.bmi > 60))
df[mask12].count() # we find 93 data point , 26 of them are men
```

Descriptive Analysis


age
gender_val -0.023
height -0.085 0.51
weight 0.055 0.16 0.31
bmi 0.1 -0.11 -0.21 0.86
ap_hi 0.21 0.061 0.019 0.27 0.27
ap_lo 0.15 0.067 0.037 0.25 0.24 0.7
bp_tot 0.095 0.0019 -0.019 0.058 0.069 0.48 -0.27
cholesterol 0.16 -0.037 -0.054 0.14 0.17 0.19 0.16 0.07
gluc 0.099 -0.021 -0.021 0.11 0.12 0.093 0.076 0.034 0.45
smoke -0.048 0.34 0.19 0.067 -0.034 0.027 0.025 0.0066 0.01 -0.0056
alco -0.029 0.17 0.098 0.068 0.017 0.032 0.043 -0.0034 0.036 0.011 0.34
active -0.01 0.006 -0.0085 -0.018 -0.015 -0.0014 -7e-05 -0.0013 0.009 -0.0075 0.025 0.025
cardio 0.24 0.0074 -0.012 0.18 0.19 0.43 0.34 0.17 0.22 0.09 -0.016 -0.008 -0.037
age
gender_val
height
weight
bmi
ap_hi
ap_lo
bp_tot
cholesterol
gluc
smoke
alco
active
cardio
0.3
0.2
0.1
0.0
−0.1
−0.2

- How many person in this dataset have Cardiovascular disease?
- Does smoking increase the risk of Cardiovascular disease?
- Does cholesterol correlate with Cardiovascular disease?
- Does a particular gender have a higher risk of Cardiovascular disease?
- What is the effect of alcohol on Cardiovascular disease?
- What is the effect of glucose levels on Cardiovascular disease?
- Does Physical activity have an effect against Cardiovascular disease?
- What age-groups commonly carry cardiovascular diseases?
- Does high number of ap_hi and ap_lo affect Cardiovascular disease?
- Does obesity affect Cardiovascular disease?


Weight_classes vs Cardiovascular disease
% of people with Cardiovascular disease
% of people with Cardiovascular disease
0
10
20
30
40
50
60
70
80
0
10
20
30
40
50
60
70
80
90
Normal Weight
Obesity Class 1
Obesity Class 2
Obesity Class 3
Pre-obesity
Underweight
Weight_classes

Model Generation and Selection

```
X = df.drop(['cardio', 'age_groups', 'gender' , 'weight_classes' , 'bp_categ'
, 'bp_tot'], axis=1)
y = df['cardio']
```

```
# we define a function that split the dataframe into training and testing dat
asets,
# and return clssification report
def test_model (model, X, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
,
                                                        random_state=123, str
atify=y)
    model.fit(x_train, y_train)
    y_pred= model.predict(x_test)

    print (pd.DataFrame(classification_report(y_test, y_pred, output_dict = T
rue)))
```

K-Nearest Neighbors

```
knn = KNeighborsClassifier (n_neighbors=9)
test_model (knn, X, y)
```

```
              precision    recall  f1-score   support

           0       0.70      0.74      0.72      6938
           1       0.72      0.67      0.69      6797

    accuracy                           0.71     13735
   macro avg       0.71      0.71      0.70     13735
weighted avg       0.71      0.71      0.71     13735
```

Logistic Regression Model

```
logreg = LogisticRegression (penalty='l2')
test_model (logreg, X, y)
```

	precision	recall	f1-score	support
0	0.70	0.78	0.74	6938
1	0.74	0.67	0.70	6797
accuracy			0.72	13735
macro avg	0.72	0.72	0.72	13735
weighted avg	0.72	0.72	0.72	13735

Decision Tree Model

```
decisiontree = DecisionTreeClassifier(max_depth=4)
test_model (decisiontree, X, y)
```

	precision	recall	f1-score	support
0	0.72	0.73	0.73	6938
1	0.72	0.72	0.72	6797
accuracy			0.72	13735
macro avg	0.72	0.72	0.72	13735
weighted avg	0.72	0.72	0.72	13735

Random Forest Model

```
randomforest = RandomForestClassifier(n_estimators=1000, max_depth=10,
                                      min_samples_leaf=15, min_samples_split=
4)
test_model (randomforest, X, y)
```

	precision	recall	f1-score	support
0	0.72	0.78	0.75	6938
1	0.75	0.69	0.72	6797
accuracy			0.73	13735
macro avg	0.73	0.73	0.73	13735
weighted avg	0.73	0.73	0.73	13735

Thank You