Intro

In this notebook I'll apply different EDA (Exploratory Data Analysis) techniques on the Heart Failure Prediction data.

The goal is to predict if a patient will have a Heart Failure or not the using LightGBM Classifier to predict based on given medical features.

Attribute Information

Variable	Definition	Key
Age	age of the patient	years
Sex	sex of the patient	M = Male, F = Female
ChestPainType	chest pain type	TA = Typical Angina, ATA = Atypical Angina, NAP = Non-Anginal Pain, ASY = Asymptomatic
RestingBP	resting blood pressure	mm Hg
Cholesterol	serum cholesterol	mm/dl
FastingBS	fasting blood sugar	1 = if FastingBS > 120 mg/dl, 0 = otherwise
RestingECG	resting electrocardiogram results	Normal = Normal, ST = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH = showing probable or definite left ventricular hypertrophy by Estes' criteria
MaxHR	maximum heart rate achieved	Numeric value between 60 and 202
ExerciseAngina	exercise-induced angina	Y = Yes, N = No
Oldpeak	oldpeak = ST	Numeric value measured in depression
ST_Slope	the slope of the peak exercise ST segment	Up = upsloping, Flat = flat, Down = downsloping
HeartDisease	output class	1 = heart disease, 0 = Normal

Import Libaries

```
import sys
import math
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# import plotly
import plotly.express as px
# import plotly.offline
# import cufflinks as cf
# cf.go_offline()
# cf.set_config_file(offline=False, world_readable=True)
# import ipywidgets as widgets
# from ipywidgets import interact

sns.set_style('whitegrid')
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
print(f'Python  version: {sys.version}')
print(f'pandas  version: {pd.__version__}')
print(f'numpy   version: {np.__version__}')
print(f'seaborn version: {sns.__version__}')
```

```
Python  version: 3.7.12 (default, Oct 12 2021, 03:36:26)
[GCC 8.3.0]
pandas  version: 1.2.5
numpy   version: 1.19.5
seaborn version: 0.11.2
```

```
Timestamp('2021-11-16 11:44:47.221313')
```

Load data

```
url = 'https://raw.githubusercontent.com/GuySuphakit/Heart-Failure-Prediction/main/heart.csv'
temp_df = pd.read_csv(url)
temp_df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
0	40	M	ATA	140	289	0	Normal	172
1	49	F	NAP	160	180	0	Normal	156
2	37	M	ATA	130	283	0	ST	98
3	48	F	ASY	138	214	0	Normal	108
4	54	M	NAP	150	195	0	Normal	122

Getting to know the data

In this section, we'll take a quick look at the data, to see how many row are there, and whther there are any missing values or not, to decie what kind of preprocessing will be needed.

```
temp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

The dataset consists of 918 samples and 12 columns : 8 predictors and one target variable and there are no missing values (which is a very good thing!) but looking at the dtypes. It seems that all discrete columns are in object and continuous in int64.

Data cleaning

As stated in the previous section, only few cleaning will be perform, mainly:

- Convert discrete columns to category
- If there is any duplicate data, drop it.

```
def read_data():
    url = 'https://raw.githubusercontent.com/GuySuphakit/Heart-Failure-Prediction/main/heart.csv'
    df = pd.read_csv(url)
    return df

# def change_lebel_of_HeartDisease(df):
#     # change the lebel of HeartDisease because it has caused some ambiguity.
#     df.HeartDisease = df.HeartDisease.replace({0: 'Normal', 1: 'Heart Disease'})
#     return df

def convert_FastingBS_to_category(df):
    df.FastingBS = df.FastingBS.astype('category')
    return df

def convert_obj_columns_to_category(df):
    for c in df.columns:
        col_type = df[c].dtype
        if col_type == 'object' or col_type.name == 'category':
            df[c] = df[c].astype('category')
    return df

def drop_duplicate(df):
    df = df.drop_duplicates()
    return df
```

```
df = (read_data()
      .pipe(convert_FastingBS_to_category)
      .pipe(convert_obj_columns_to_category)
      .pipe(drop_duplicate))

df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
0	40	M	ATA	140	289	0	Normal	172
1	49	F	NAP	160	180	0	Normal	156
2	37	M	ATA	130	283	0	ST	98
3	48	F	ASY	138	214	0	Normal	108
4	54	M	NAP	150	195	0	Normal	122

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             918 non-null    int64
 1   Sex             918 non-null    category
 2   ChestPainType   918 non-null    category
 3   RestingBP       918 non-null    int64
 4   Cholesterol     918 non-null    int64
 5   FastingBS       918 non-null    category
 6   RestingECG      918 non-null    category
 7   MaxHR           918 non-null    int64
 8   ExerciseAngina  918 non-null    category
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    category
 11  HeartDisease    918 non-null    int64
dtypes: category(6), float64(1), int64(5)
memory usage: 56.4 KB
```

```
def print_category_columns(df):
    for c in df.columns:
        col_type = df[c].dtype
        if col_type.name == 'category':
            print(f'{c:15}: {list(enumerate(df[c].cat.categories))}')
            print('-' * 60)

print_category_columns(df)
```

```
Sex             : [(0, 'F'), (1, 'M')]
------------------------------------------------------------
```

```
ChestPainType  : [(0, 'ASY'), (1, 'ATA'), (2, 'NAP'), (3, 'TA')]
------------------------------------------------------------
FastingBS      : [(0, 0), (1, 1)]
------------------------------------------------------------
RestingECG     : [(0, 'LVH'), (1, 'Normal'), (2, 'ST')]
------------------------------------------------------------
ExerciseAngina : [(0, 'N'), (1, 'Y')]
------------------------------------------------------------
ST_Slope       : [(0, 'Down'), (1, 'Flat'), (2, 'Up')]
------------------------------------------------------------
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype   
---  ------          --------------  -----   
 0   Age             918 non-null    int64   
 1   Sex             918 non-null    category
 2   ChestPainType   918 non-null    category
 3   RestingBP       918 non-null    int64   
 4   Cholesterol     918 non-null    int64   
 5   FastingBS       918 non-null    category
 6   RestingECG      918 non-null    category
 7   MaxHR           918 non-null    int64   
 8   ExerciseAngina  918 non-null    category
 9   Oldpeak         918 non-null    float64 
 10  ST_Slope        918 non-null    category
 11  HeartDisease    918 non-null    int64   
dtypes: category(6), float64(1), int64(5)
memory usage: 56.4 KB
```

We cleaned the data with a clean code!

Exploratory Data Analysis (EDA)

In this section, we'll explore the data visually and summarize it using descriptive statistic methods.

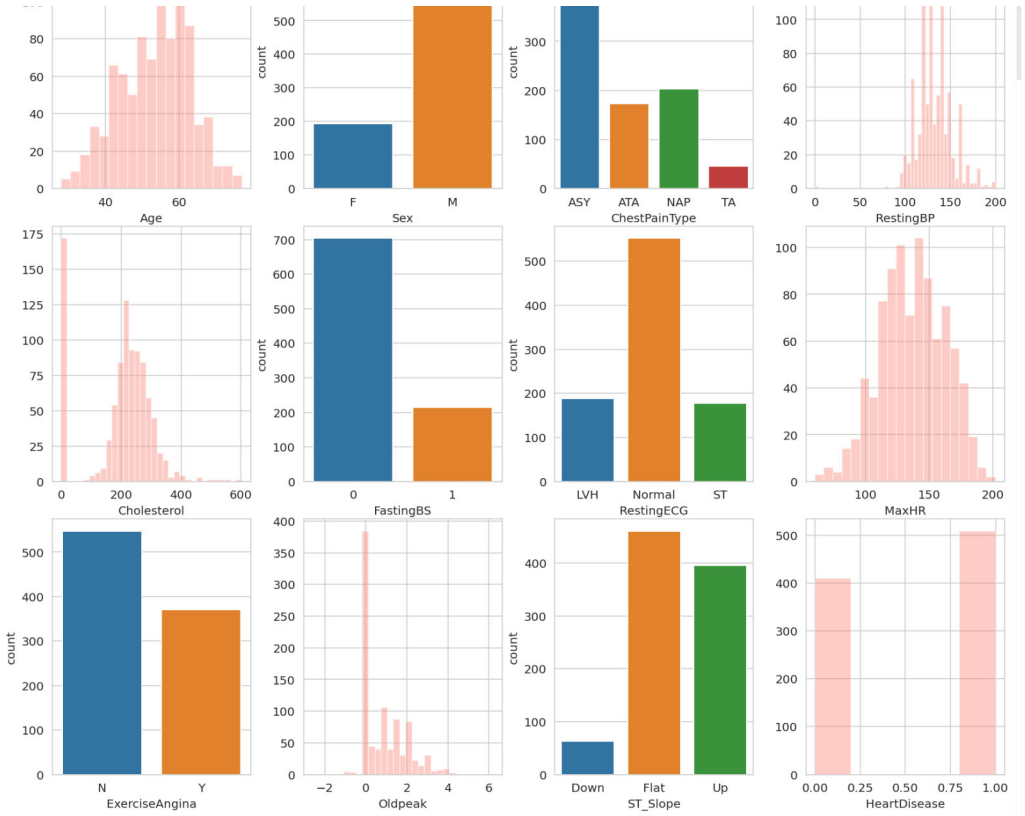
To keep things simpler, we'll divide this section into three subsections:

1. Univariate analysis: in this section we'll focus only at one variable at a time, and study the variable descriptive statistics with some charts like: Bar chart, Line chart, Histogram, Boxplot, etc ..., and how the variable is distributed, and if there is any skewness in the distribution.
2. Bivariate analysis: in this section we'll study the relation between two variables, and present different statistics such as Correlation, Covariance, and will use some other charts like: scatterplot, and will make use of the hue parameter of the previous charts.
3. Multivariate analysis: in this section we'll study the relation between three or more variables, and will use additional type of charts, such as parplot.

Overall

Let's briefly see overall image of our dataset :

```
def plot_mn(df, cols, n_rows:int=1, kind:str='boxplot', color='salmon'):
    """
    plot boxplot, violin, hist in m (rows) by n (columns)
    >>> plot_mn(df, ['Calories', 'Fat'], 2, 'hist')
    """
    n=len(cols)
    n_cols=math.ceil(n / n_rows)
    fig, ax = plt.subplots(n_rows, n_cols, figsize=(n_cols*3, n_rows*3.5))
    ax=ax.ravel()

    fig.tight_layout()
    for i, c in enumerate(cols):
        col_type = df[c].dtype
        if col_type.name == 'category':
            sns.countplot(data=df, x=c, ax=ax[i])
        else:
            if kind.lower()=='boxplot':
                sns.boxplot(data=df[[c]], ax=ax[i], color=color)
            if kind.lower()=='boxen':
```




```
df.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

```
df.describe().T.round(1)
```

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.5	9.4	28.0	47.0	54.0	60.0	77.0
RestingBP	918.0	132.4	18.5	0.0	120.0	130.0	140.0	200.0
Cholesterol	918.0	198.8	109.4	0.0	173.2	223.0	267.0	603.0
MaxHR	918.0	136.8	25.5	60.0	120.0	138.0	156.0	202.0
Oldpeak	918.0	0.9	1.1	-2.6	0.0	0.6	1.5	6.2
HeartDisease	918.0	0.6	0.5	0.0	0.0	1.0	1.0	1.0

```
df.describe(include='category').T
```

	count	unique	top	freq
Sex	918	2	M	725
ChestPainType	918	4	ASY	496
FastingBS	918	2	0	704
RestingECG	918	3	Normal	552
ExerciseAngina	918	2	N	547
ST_Slope	918	3	Flat	460

```
numerical= df.select_dtypes('int64').columns

categorical = df.select_dtypes('category').columns

print(f'Numerical Columns:  {df[numerical].columns}')
print('-'* 100)
print(f'Categorical Columns: {df[categorical].columns}')
```

```
Numerical Columns:  Index(['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'HeartDisease'], dtype='object')
----------------------------------------------------------------------------------------------------
Categorical Columns: Index(['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina',
       'ST_Slope'],
      dtype='object')
```

```
df.nunique().sort_values(ascending=False)
```

```
Cholesterol       222
MaxHR             119
RestingBP          67
Oldpeak            53
Age                50
ChestPainType       4
RestingECG          3
ST_Slope            3
Sex                 2
FastingBS           2
ExerciseAngina      2
HeartDisease        2
dtype: int64
```

Univariate Analysis

Here in this section, will perform analysis on each variable individually, but according to the variable type different methods and visualization will be used, main types of variables:

- Numerical
  - Continuous: continuous variables are continous measurements.
  - Discrete: discrete variables represent counts.
- Categorical
  - Nominal: nominal variable has a finite set of possible values, which don't have any ordereing relation among them.
  - Ordinal: in contrast to Nominal variable, ordinal varible defines an ordering relation between the values.
  - Binary: binary variables are a special case of nominal variables, but they only have two possible values.

Types of varibles

- Continuous: Age, RestingBP, Cholesterol, Oldpeak and MaxHR are continuous variables.
- Norminal: ChestPainType and RestingECG is nominal variables.
- Binary: Sex , ExerciseAngina, FastingBS and ST_Slope are a binary variable.

Age

The Age is a Continuous variable.

```
df["Age"].describe().round(1)
```

```
count    918.0
mean      53.5
std        9.4
min       28.0
25%       47.0
50%       54.0
75%       60.0
max       77.0
Name: Age, dtype: float64
```

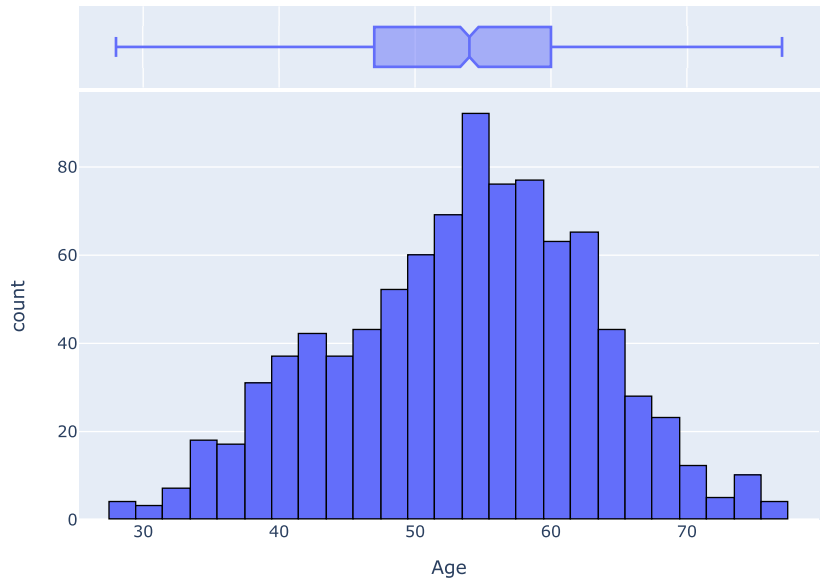
```
df.Age.mode()[0]
```

```
54
```

```
print(stats.skew(df.Age))
```

```
-0.19561273124487544
```

```
fig = px.histogram(df, x='Age', marginal='box')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



We can conclude from this chats following:

- The Age is very close to a normal distribution, with a smll negative skewness (left skewed)
- The most common scores are between 47 and 60
- The average age is 53.5 with a standard deviation of 9.4
- There are no outliers

Sex

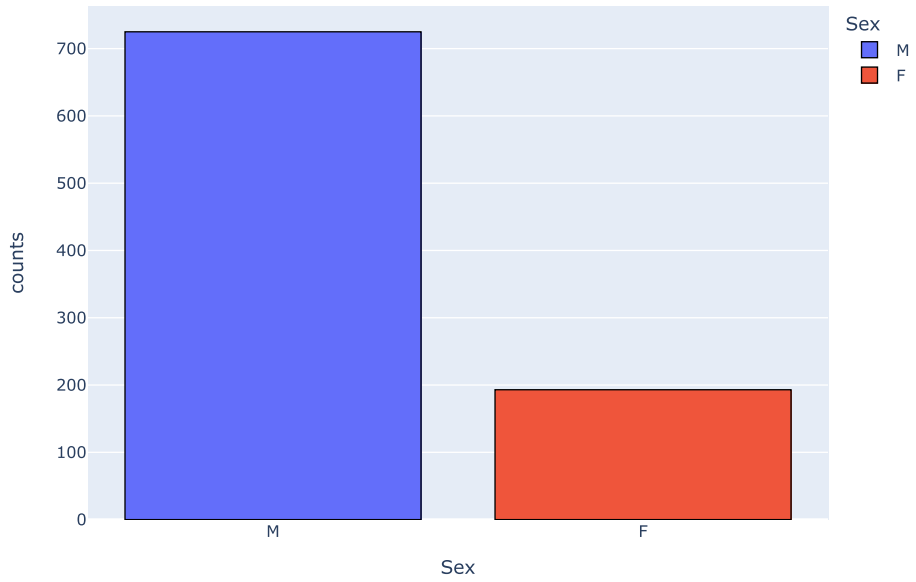
The Sex variable indicates whether the patient' sex is male or female, so it's a binary variable.

```
df.Sex.value_counts()
```

```
M    725
F    193
Name: Sex, dtype: int64
```

```
temp_df = df.groupby(by="Sex", as_index=False).agg(
    counts=pd.NamedAgg(column="Sex", aggfunc="count")).sort_values(by="counts", ascending=False)
```

```
fig = px.bar(temp_df,
             x='Sex',
             y='counts',
             color='Sex',
             color_continuous_scale=px.colors.qualitative.D3)

fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



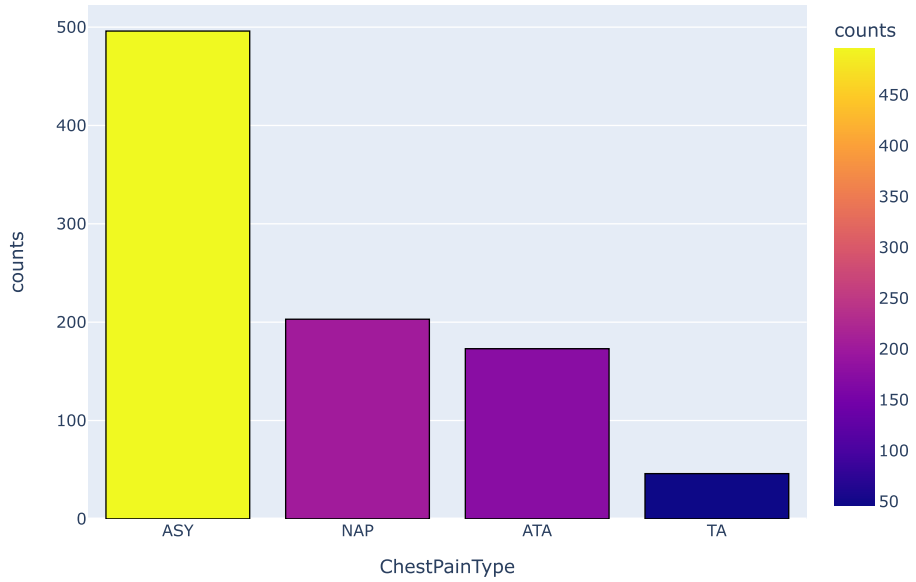
ChestPainType

The ChestPainType is nominal variables.

```
df.ChestPainType.value_counts()
```

```
ASY    496
NAP    203
ATA    173
TA      46
Name: ChestPainType, dtype: int64
```

```
temp_df = df.groupby(by="ChestPainType", as_index=False).agg(
    counts=pd.NamedAgg(column="ChestPainType", aggfunc="count")).sort_values(by="counts", ascending=

fig = px.bar(temp_df,
             x='ChestPainType',
             y='counts',
             color='counts',
             color_discrete_sequence = px.colors.qualitative.D3)
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



Most of ChestPainType is ASY.

- TA = Typical Angina
- ATA = Atypical Angina
- NAP = Non-Anginal Pain
- ASY = Asymptomatic

RestingBP

The RestingBP(resting blood pressure) is continous variable.

```
df.RestingBP.describe().round()
```

```
count    918.0
mean     132.0
std       19.0
min        0.0
25%      120.0
50%      130.0
75%      140.0
max      200.0
Name: RestingBP, dtype: float64
```

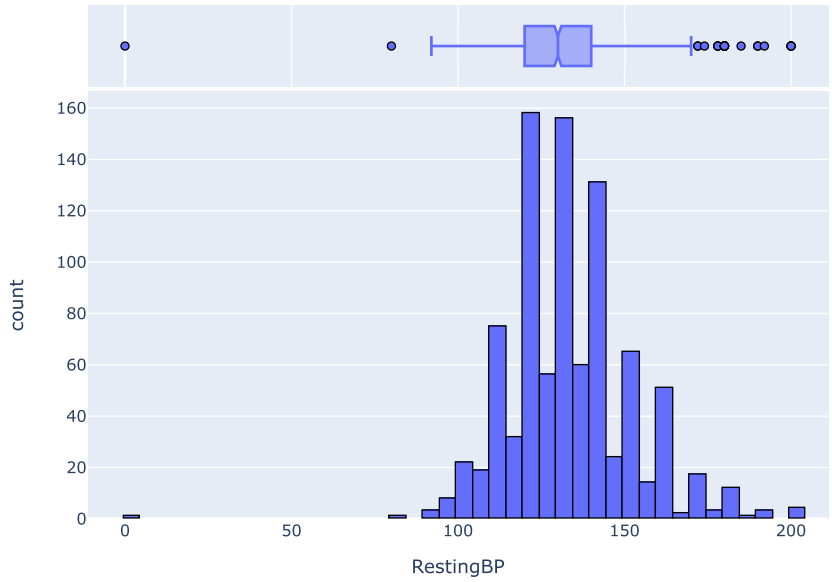
```
df.RestingBP.mode()[0]
```

```
120
```

```
print(stats.skew(df.RestingBP))
```

```
0.17954532149156327
```

```
fig = px.histogram(df, x='RestingBP', marginal='box')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```




```
df[df['RestingBP'] == 0]
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
449	55	M	NAP	0	0	0	Normal	155

The plot shows that:

- most patient have resting blood pressure between 120 and 140
- The distribution is moderately skewd to the right with a positive skew value 0.18
- There is 1 record with RestingBP = 0, which is not normal.

Cholesterol

The Cholesterol is continuous varible.

```
df.Cholesterol.describe().round()
```

```
count    918.0
mean     199.0
std      109.0
min        0.0
25%      173.0
50%      223.0
75%      267.0
max      603.0
Name: Cholesterol, dtype: float64
```

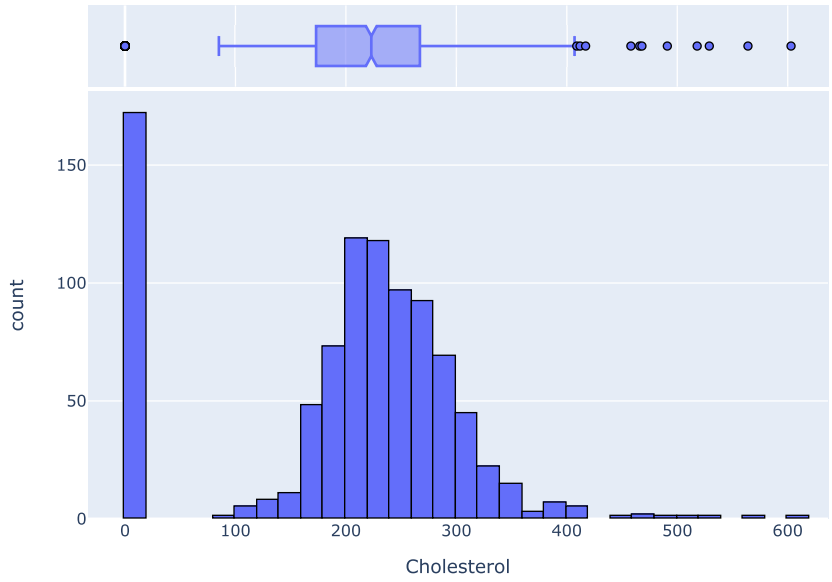
```
df.Cholesterol.mode()[0]
```

```
0
```

```
print(stats.skew(df.Cholesterol))
```

```
-0.6090891046626045
```

```
fig = px.histogram(df, x='Cholesterol', marginal='box')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```




```
df[df['Cholesterol'] == 0].head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR
293	65	M	ASY	115	0	0	Normal	93
294	32	M	TA	95	0	1	Normal	127
295	61	M	ASY	105	0	1	Normal	110
296	50	M	ASY	145	0	1	Normal	139
297	57	M	ASY	110	0	1	ST	131

```
df[df['Cholesterol'] >= 500]['Cholesterol']
```

```
30     518
76     529
149    603
616    564
Name: Cholesterol, dtype: int64
```

There are many patients who are not supposed to have Choresterol = 0, which is impossible and the outliers are above value 458 since such high Cholesterol is only seen in a small number of people. We require more processing to analyse data statistics.

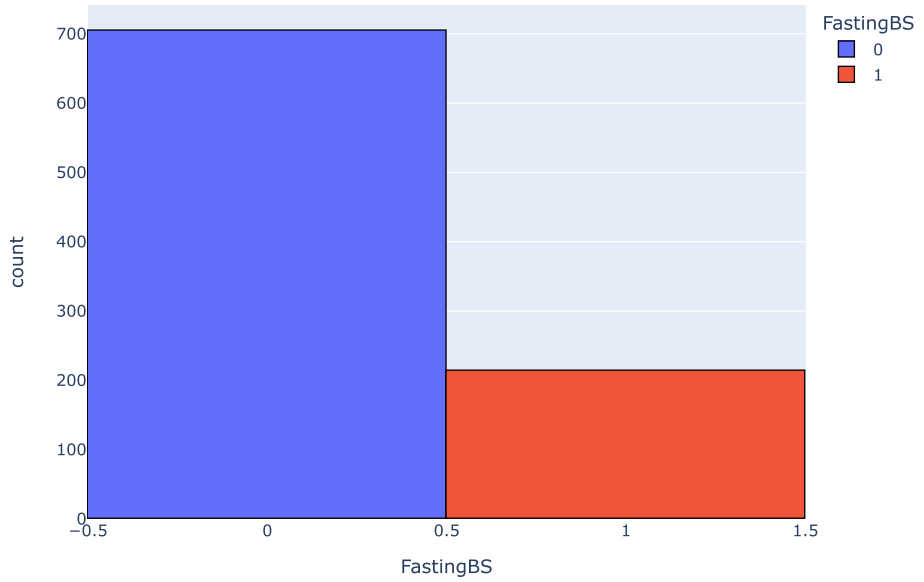
FastingBS

The FastingBS(fasting blood sugar) is binary variable since 1 = if FastingBS > 120 mg/dl, 0 = otherwise.

```
df.FastingBS.value_counts()
```

```
0    704
1    214
Name: FastingBS, dtype: int64
```

```
fig = px.histogram(df, x='FastingBS', color='FastingBS')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



RestingECG

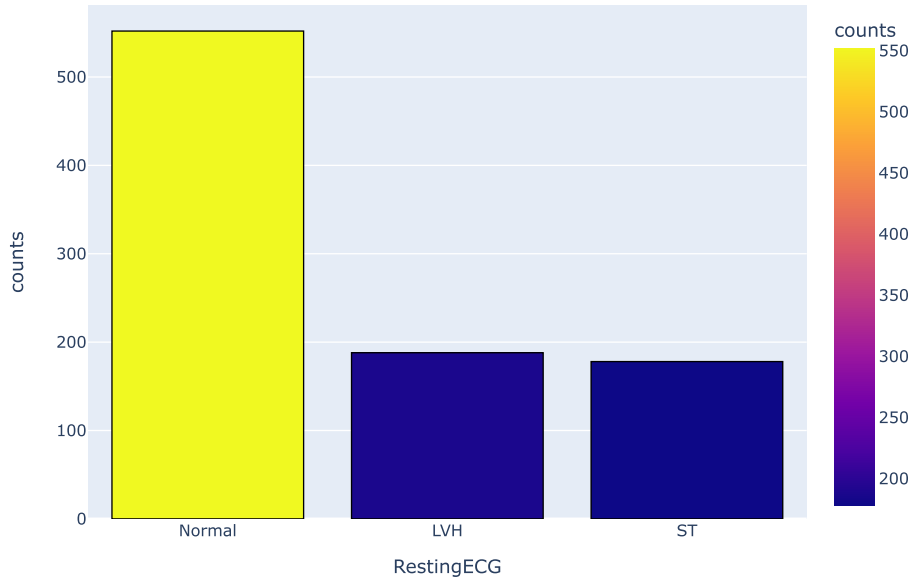
RestingECG stands for resting electrocardiogram results and it have Normal, ST and LVH. so, this is an norminal variable.

```
df["RestingECG"].value_counts()
```

```
Normal    552
LVH       188
ST        178
Name: RestingECG, dtype: int64
```

```
temp_df = df.groupby(by="RestingECG", as_index=False).agg(
    counts=pd.NamedAgg(column="RestingECG", aggfunc="count")).sort_values(by="counts", ascending=Fals
```

```
fig = px.bar(temp_df,
             x='RestingECG',
             y='counts',
             color='counts',
             color_discrete_sequence = px.colors.qualitative.D3)
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



MaxHR

It's Continuos variable.

```
df.MaxHR.describe().round()
```

```
count    918.0
mean     137.0
std       25.0
min       60.0
25%      120.0
50%      138.0
75%      156.0
max      202.0
Name: MaxHR, dtype: float64
```

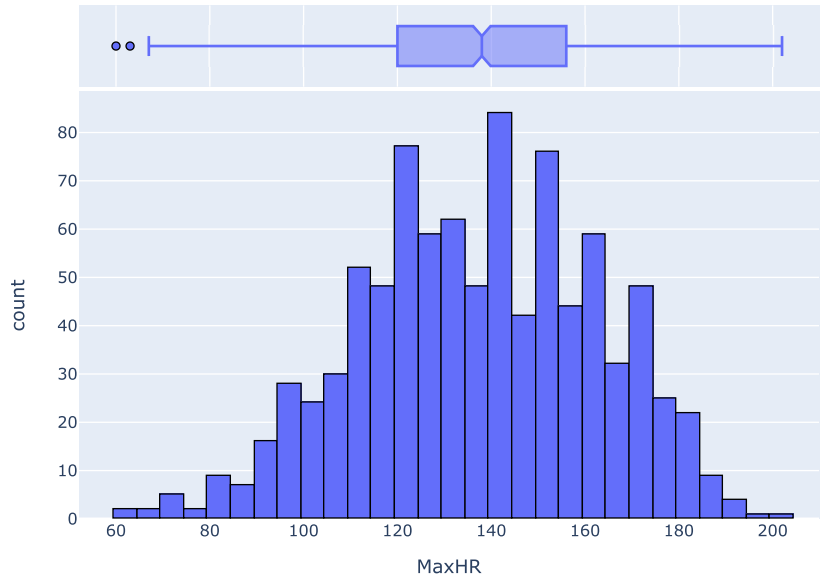
```
df.MaxHR.mode()[0]
```

```
150
```

```
df.MaxHR.skew()
```

```
-0.14435941846180994
```

```
fig = px.histogram(df, x='MaxHR', marginal='box')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



This variable looking like a normal distribution, with a small negative skewness.

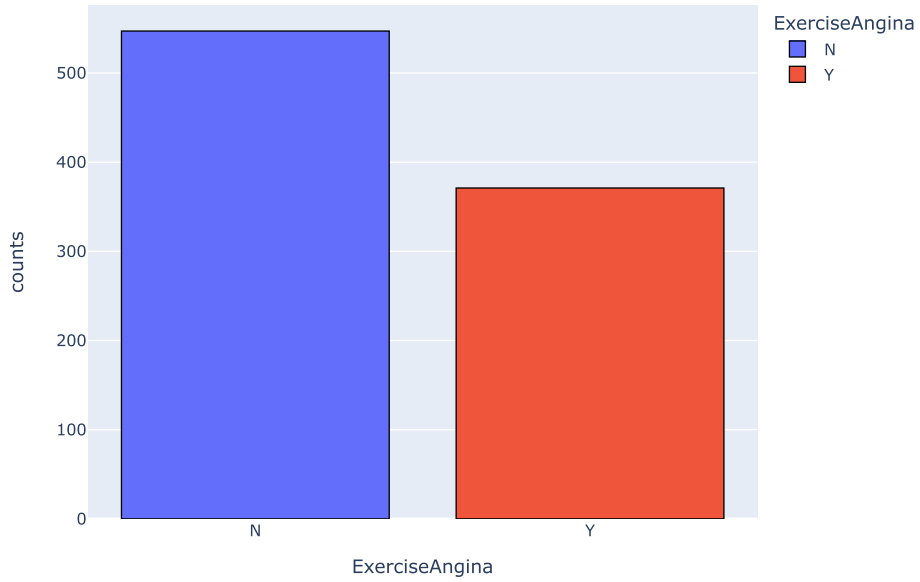
ExerciseAngina

It's binary variable.

```
df.ExerciseAngina.value_counts()
```

```
N    547
Y    371
Name: ExerciseAngina, dtype: int64
```

```
temp_df = df.groupby(by="ExerciseAngina", as_index=False).agg(
    counts=pd.NamedAgg(column="ExerciseAngina", aggfunc="count")).sort_values(by="counts", ascending=

fig = px.bar(temp_df,
             x='ExerciseAngina',
             y='counts',
             color='ExerciseAngina',
             color_continuous_scale=px.colors.qualitative.D3)

fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



Oldpeak

It's Continuos variable.

```
df.Oldpeak.describe().round()
```

```
count    918.0
mean       1.0
std        1.0
min       -3.0
25%        0.0
50%        1.0
75%        2.0
max        6.0
Name: Oldpeak, dtype: float64
```

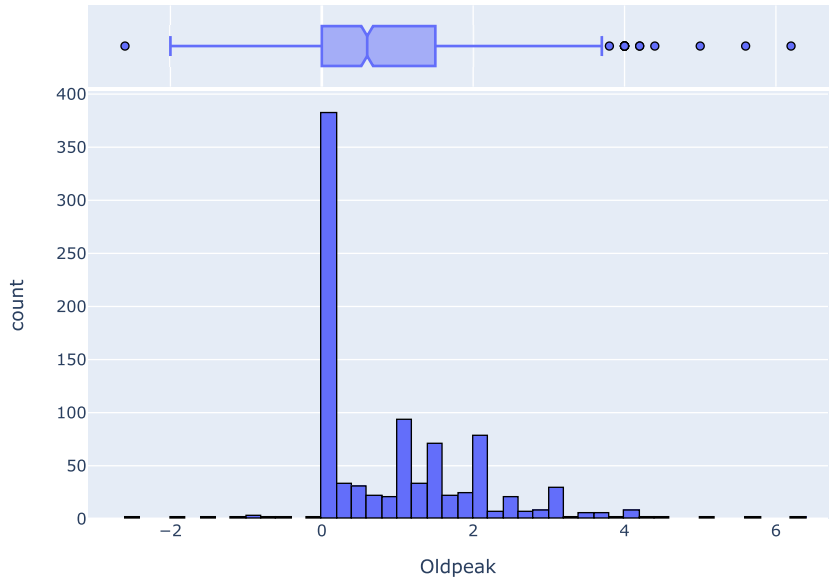
```
df.Oldpeak.mode()[0]
```

```
0.0
```

```
df.Oldpeak.skew()
```

```
1.0228720218107528
```

```
fig = px.histogram(df, x='Oldpeak', marginal='box')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



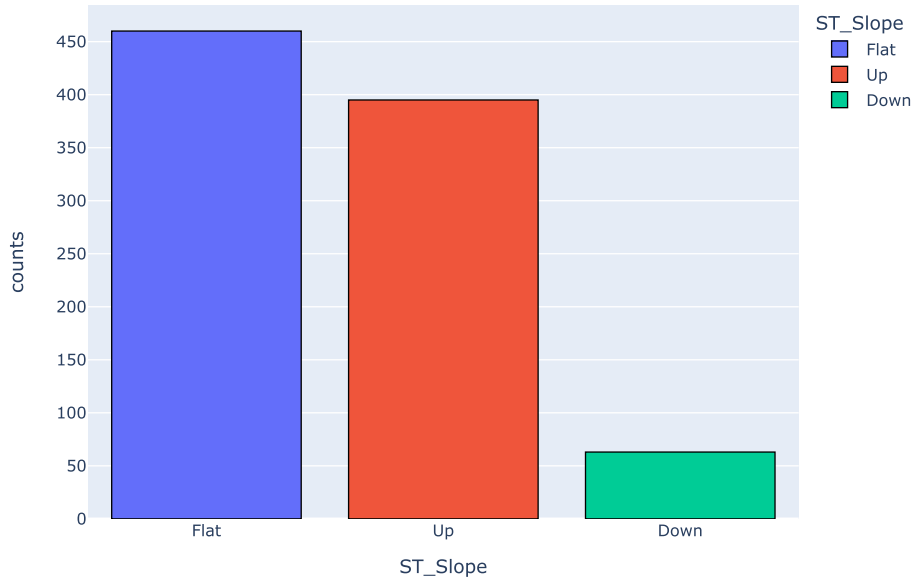
ST_Slope

It's binary variable.

```
df.ST_Slope.value_counts()
```

```
Flat    460
Up      395
Down     63
Name: ST_Slope, dtype: int64
```

```
temp_df = df.groupby(by="ST_Slope", as_index=False).agg(
    counts=pd.NamedAgg(column="ST_Slope", aggfunc="count")).sort_values(by="counts", ascending=False)

fig = px.bar(temp_df,
             x='ST_Slope',
             y='counts',
             color='ST_Slope',
             color_continuous_scale=px.colors.qualitative.D3)

fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



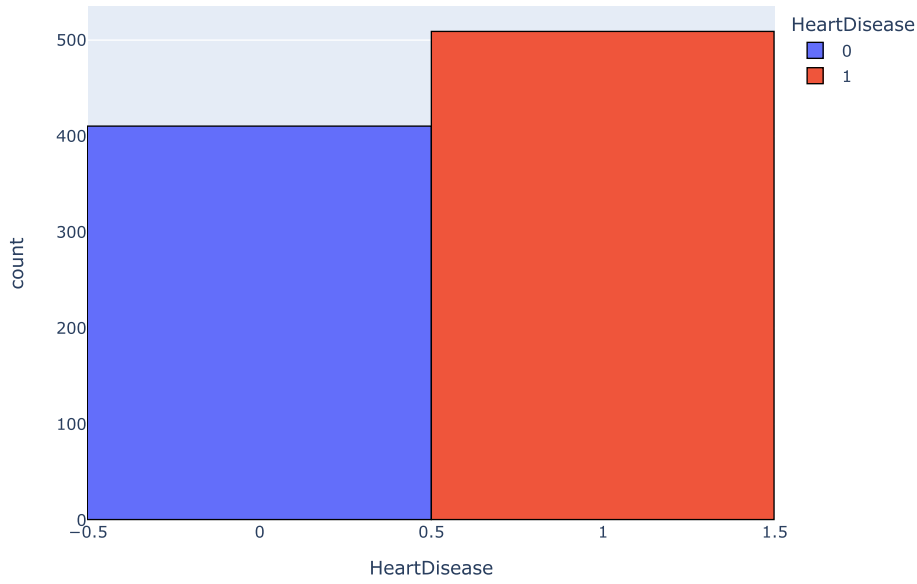
HeartDisease

It's binary variable.

```
df.HeartDisease.value_counts()
```

```
1    508
0    410
Name: HeartDisease, dtype: int64
```

```
fig = px.histogram(df, x='HeartDisease', color='HeartDisease')
fig.update_traces(marker_line_width=1, marker_line_color="black")
fig.show()
```



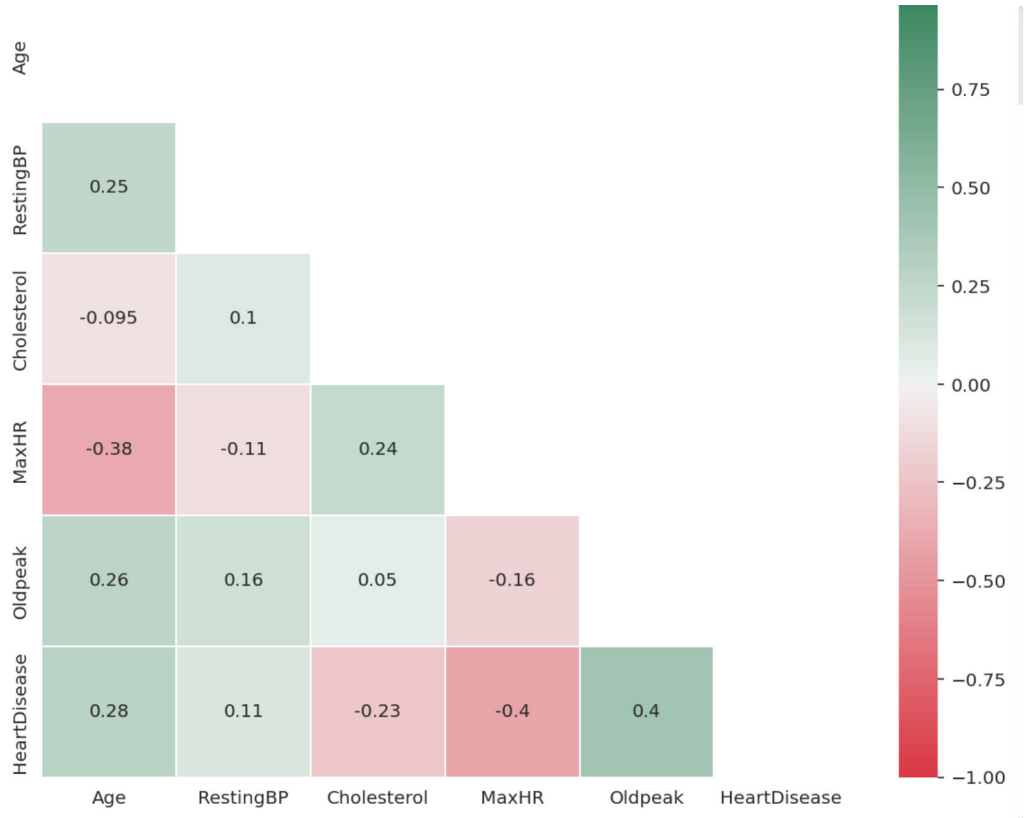
Bivariate analysis

In this section, we'll focus on studying the relationship between two different variables, to answer different question, like

- What is the relation between variable x and variable y? is it linear or non-linear?
- In case of a linear relation, is positive linear relation or negative linear relation? and how strong is the relation?
- How the distribution for two variables changes?

Correlation Matrix

```
dcorr=df[df.columns].corr()
# dcorr

mask = np.zeros_like(dcorr)
# mask.shape
mask[np.triu_indices_from(mask)] = True

fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(dcorr, cmap=sns.diverging_palette(10, 145, n=100),
            vmin=-1, vmax=1, center=0, linewidths=1, annot=True, mask=mask, ax=ax).set_title("Correla
```



Scatter plot

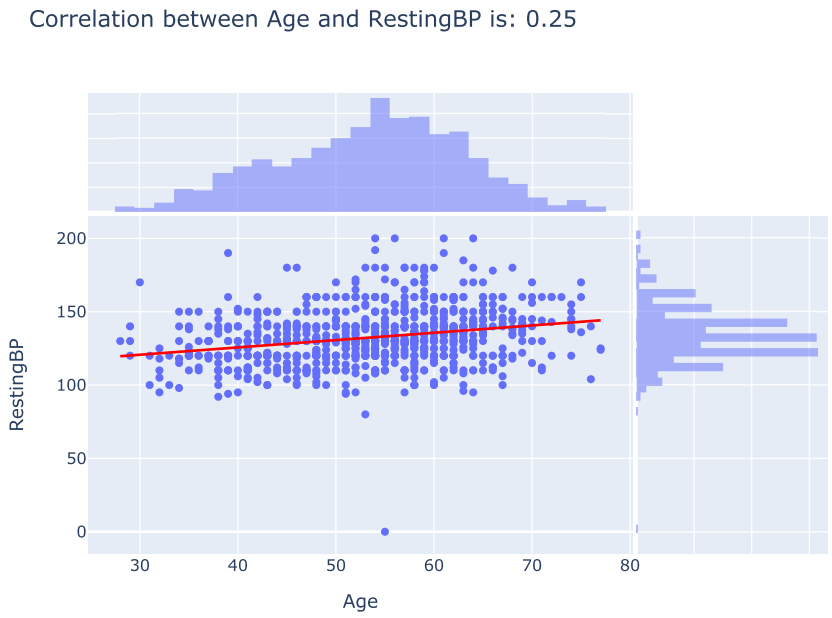
Let's show the scatter for each two variables at a time:

```
!pip install statsmodels
```

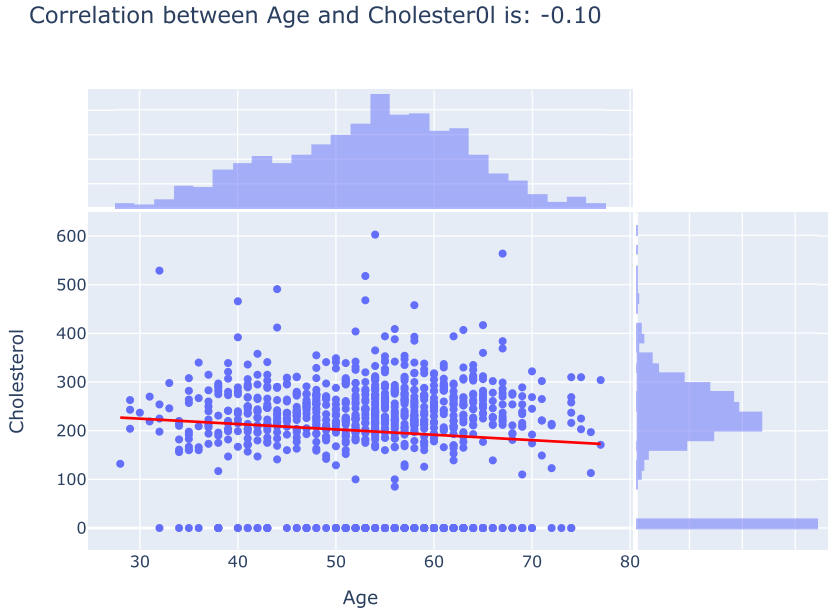
```
Requirement already satisfied: statsmodels in /root/venv/lib/python3.7/site-packages (0.13.1)
Requirement already satisfied: patsy>=0.5.2 in /root/venv/lib/python3.7/site-packages (from statsmodels) (0.5.2)
Requirement already satisfied: numpy>=1.17 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from statsmodels
Requirement already satisfied: scipy>=1.3 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from statsmodels)
Requirement already satisfied: pandas>=0.25 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from statsmodel
Requirement already satisfied: six in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (from patsy>=0.5.2->
Requirement already satisfied: pytz>=2017.3 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from pandas>=0.
Requirement already satisfied: python-dateutil>=2.7.3 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (
WARNING: You are using pip version 20.1.1; however, version 21.3.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.
```

```
corr_value = df["Age"].corr(df["RestingBP"])

fig = px.scatter(
    data_frame=df,
    x="Age",
    y="RestingBP",
```

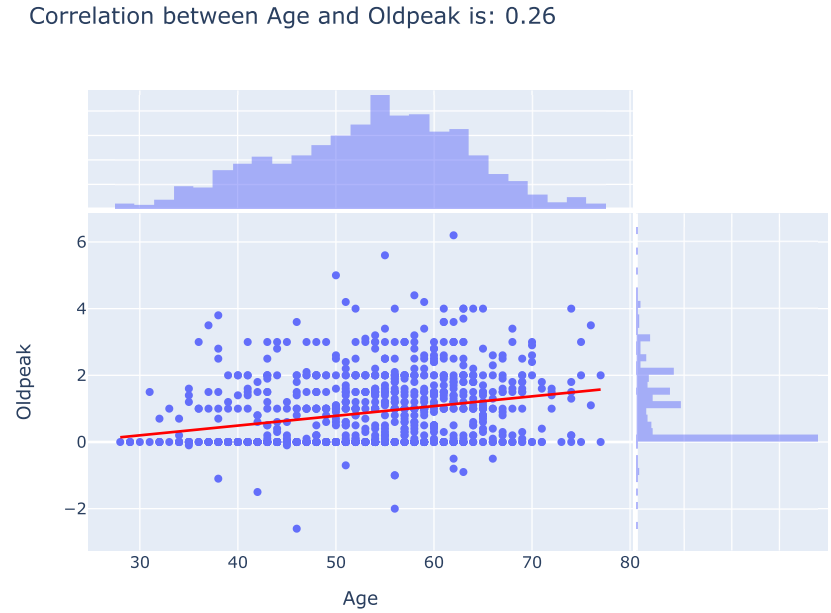
```
    marginal_x="histogram",
    marginal_y="histogram",
    trendline="ols",
    trendline_color_override="red",
    title f"Correlation between Age and RestingBP is: {corr value: 2f}"
```




```
corr_value = df["Age"].corr(df["Cholesterol"])

fig = px.scatter(
    data_frame=df,
    x="Age",
    y="Cholesterol",
    marginal_x="histogram",
    marginal_y="histogram",
    trendline="ols",
    trendline_color_override="red",
    title=f"Correlation between Age and Cholester0l is: {corr_value:.2f}",
)

fig.show()
```

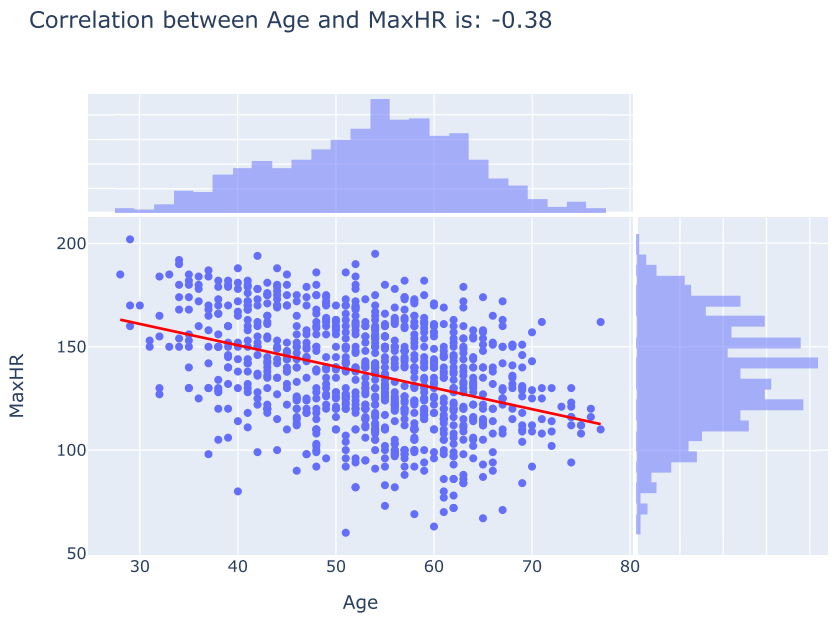



```
corr_value = df["Age"].corr(df["Oldpeak"])

fig = px.scatter(
    data_frame=df,
    x="Age",
    y="Oldpeak",
    marginal_x="histogram",
    marginal_y="histogram",
    trendline="ols",
    trendline_color_override="red",
    title=f"Correlation between Age and Oldpeak is: {corr_value:.2f}",
)

fig.show()
```




```
corr_value = df["Age"].corr(df["MaxHR"])

fig = px.scatter(
    data_frame=df,
    x="Age",
    y="MaxHR",
    marginal_x="histogram",
```

```
    marginal_y="histogram",
    trendline="ols",
    trendline_color_override="red",
       l  f"     l      b            d          {       l     f}"
```



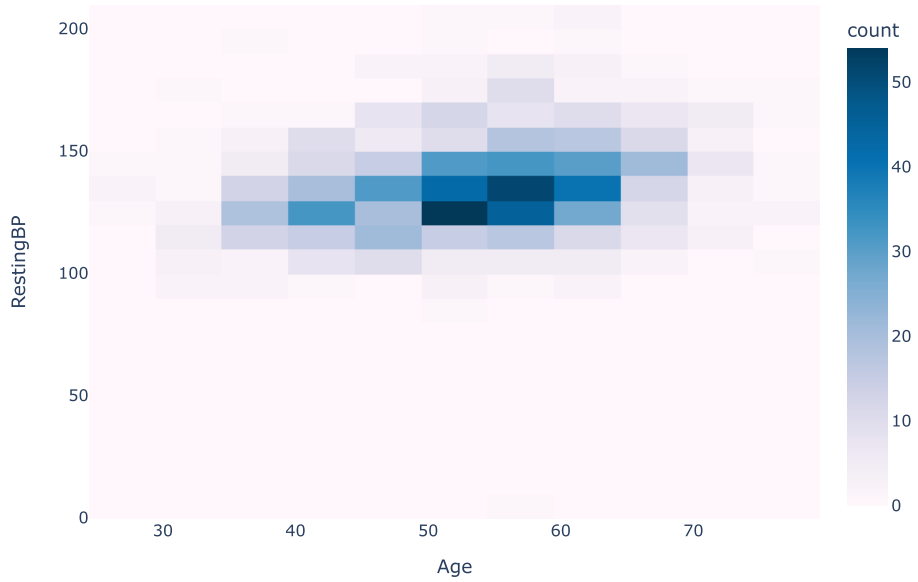
Bivariate distributions

Another way to study the relation between two variables is with 2D Histograms (distribution).

Just like the distributions we used in the Univariate Analysis section, we can show the distribution for two variables x and y, which would give us better insights on how much the values from the two variables overlap, and show cluster regions in the 2D space.

Compared to scatter plots, 2D histograms are better at handling large amounts of data, as they use rectangular bins, and count the number of points withing each bin.

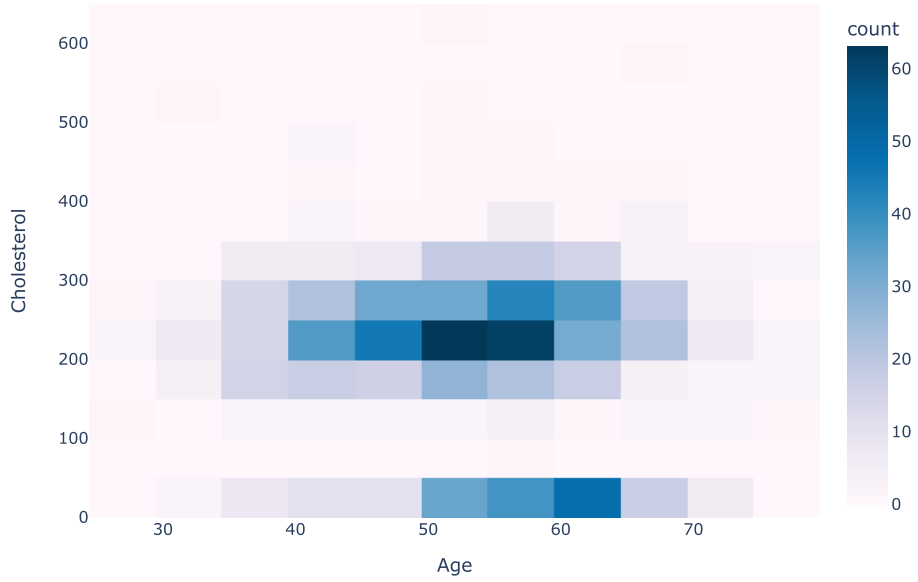
```
fig = px.density_heatmap(
    data_frame=df, x="Age", y="RestingBP", color_continuous_scale="PuBu"
)

fig.show()
```



We can see from this chart some clusters.

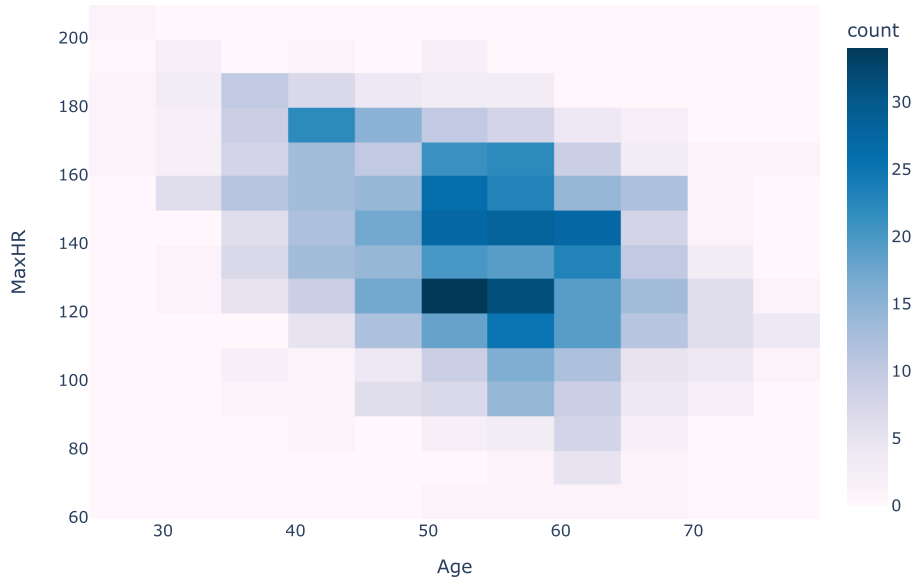
For example, there are two clusters of Patient who age between 50 and 54 years old and the other clusters of Patient who age between 55 and 59 years old. These two clusters account for about 100 patients. (which is about 10% of the total dataset)

```
fig = px.density_heatmap(
    data_frame=df, x="Age", y="Cholesterol", color_continuous_scale="PuBu"
)

fig.show()
```



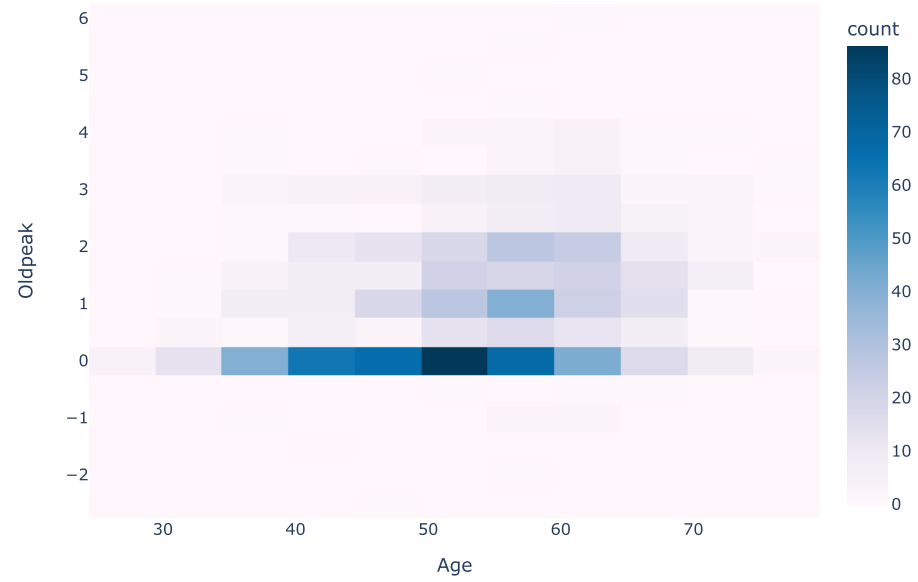
This chart shows that patients who age above 50 has a higher Cholesterol level than patients who age below 50.

```
fig = px.density_heatmap(
    data_frame=df, x="Age", y="MaxHR", color_continuous_scale="PuBu"
)

fig.show()
```



This chart shows that MaxHR ia lower when the patient is older. This is a positive linear relation.

```
fig = px.density_heatmap(
    data_frame=df, x="Age", y="Oldpeak", color_continuous_scale="PuBu"
)

fig.show()
```



This chart shows that almost of the patients has 0.1 - 0.2 oldpeak.

Multivariate analysis

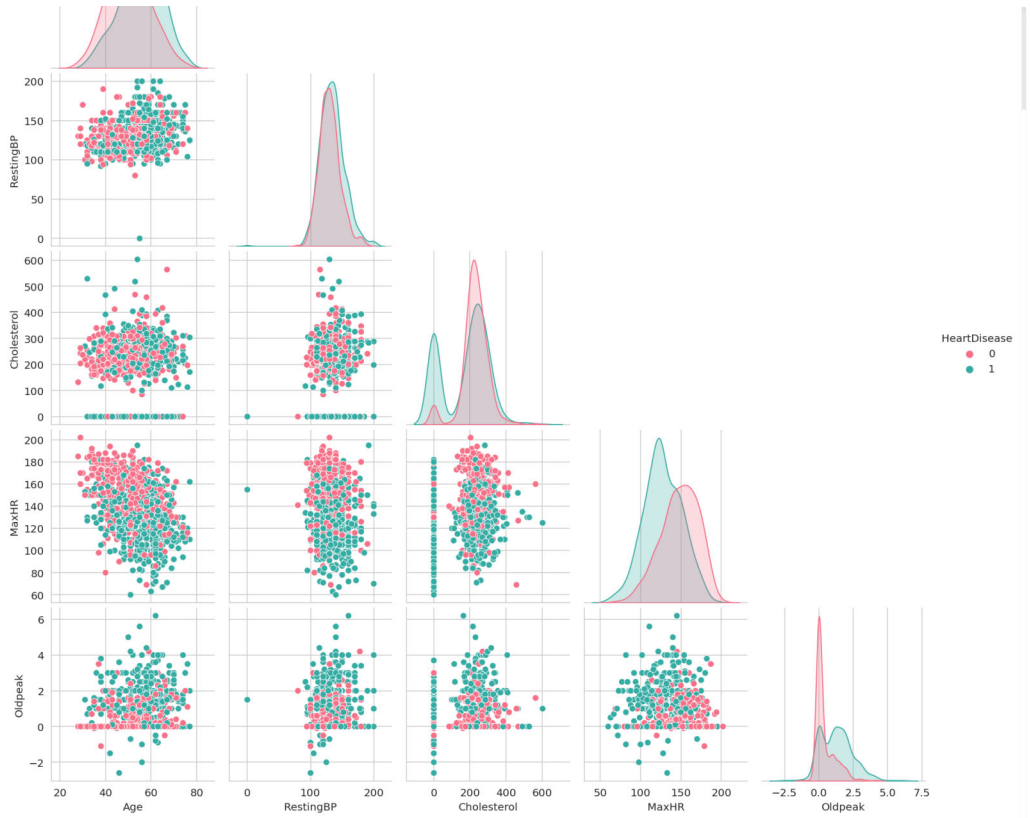
So far, all the plots we used before were used either to explore one variable, or to show the relation between a pair of variables.

However, we are often interested in answering the question: How does the relation between two variables changes as a function of a third variable?

In this section, we'll focus on answering these kinds of questions, where we'll use similar plots to the ones we used before, with conditioning on other variable.

Pairplot

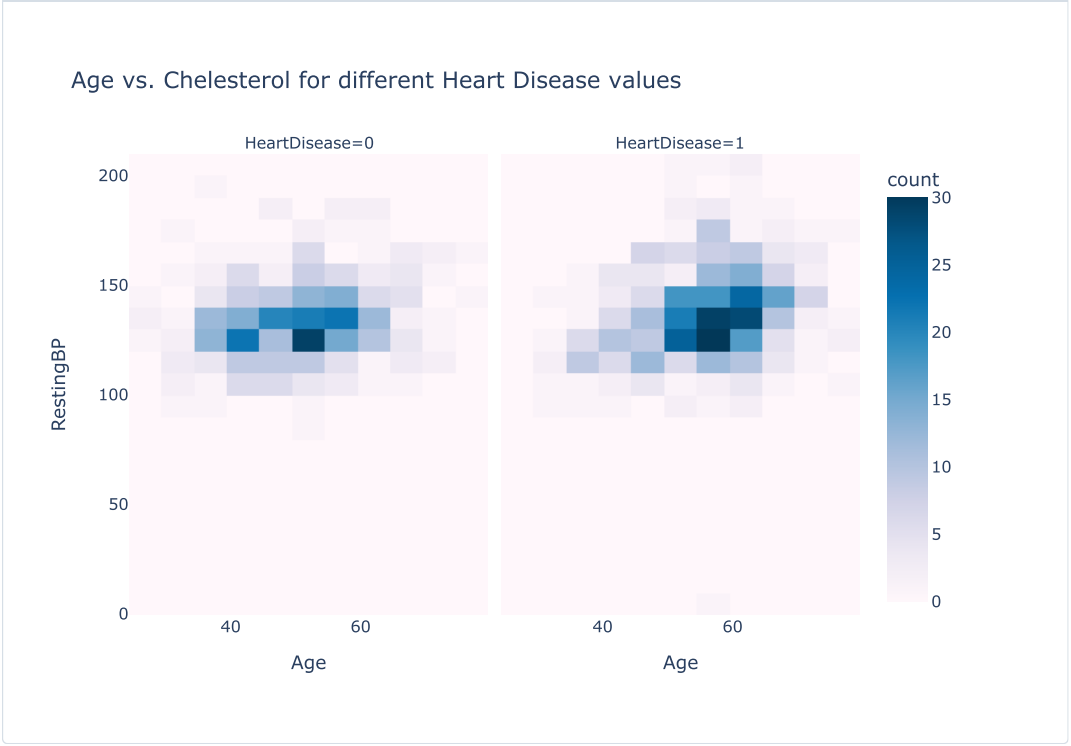
```
sns.pairplot(df, hue="HeartDisease", palette="husl", corner=True);
```



Bivariate distribution with HeartDisease

Age vs RestingBP

```
px.density_heatmap(
    data_frame=df,
    x="Age",
    y="RestingBP",
    color_continuous_scale="PuBu",
    facet_col="HeartDisease",
    title="Age vs. Chelesterol for different Heart Disease values",
)
```



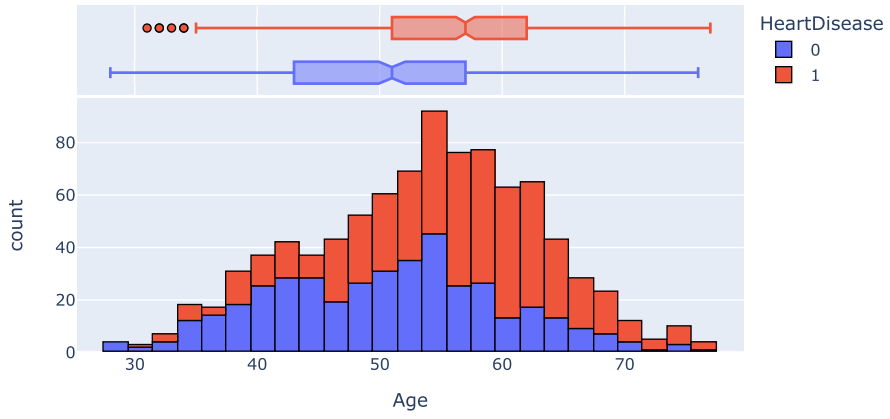
Age vs Cholesterol

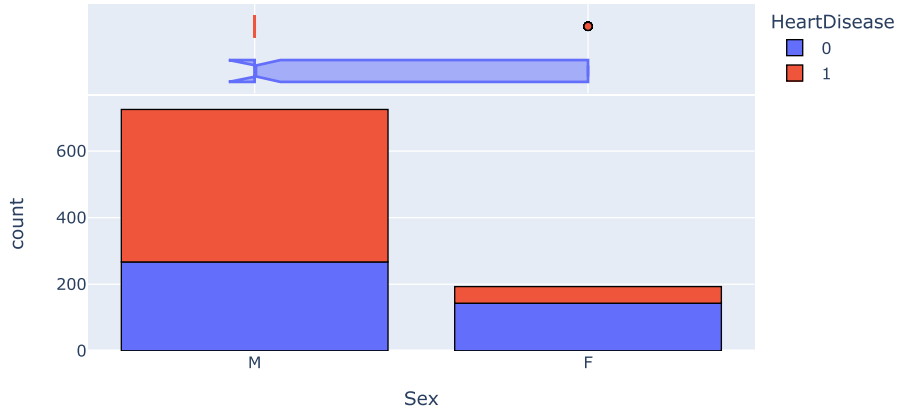
```
px.density_heatmap(
    data_frame=df,
    x="Age",
    y="Cholesterol",
    color_continuous_scale="PuBu",
    facet_col="HeartDisease",
    title="Age vs. Chelesterol for different Heart Disease values",
)
```

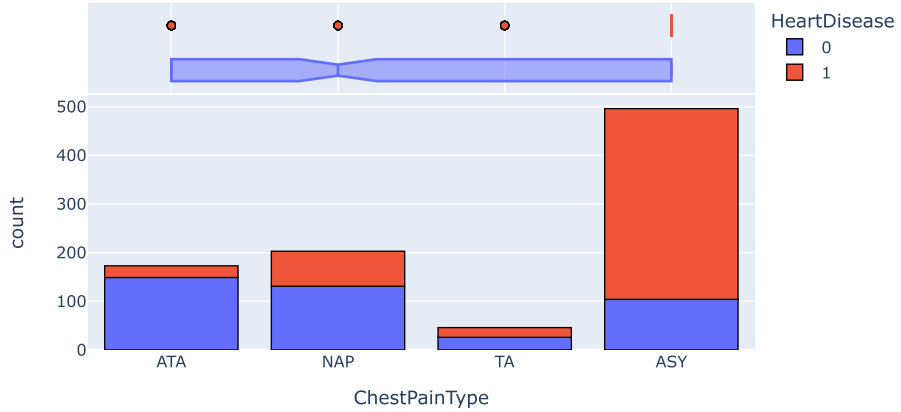
Age vs MaxHR

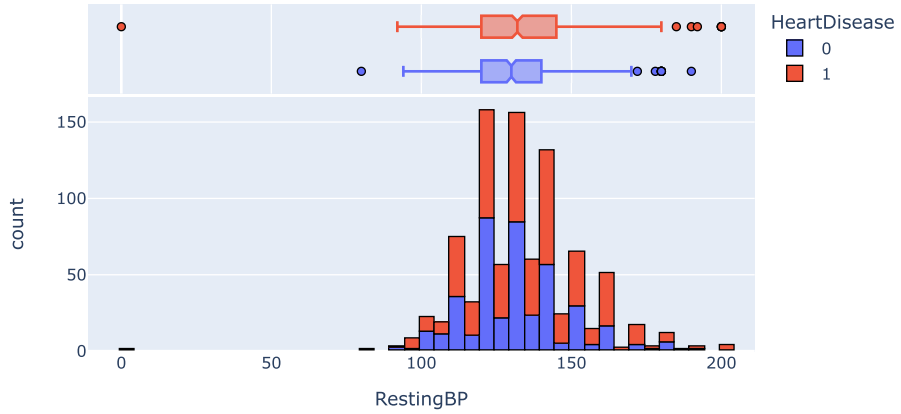
```
px.density_heatmap(
    data_frame=df,
    x="Age",
    y="MaxHR",
    color_continuous_scale="PuBu",
    facet_col="HeartDisease",
```

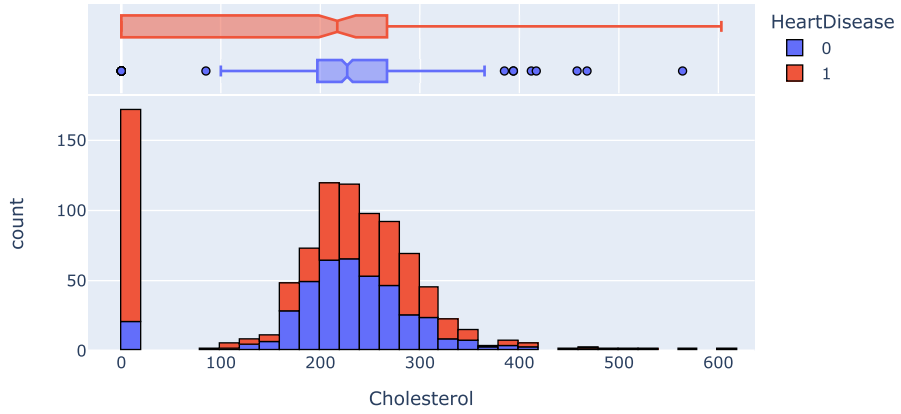

```
def num_plot(df, col):
    fig = px.histogram(df, x=col, color="HeartDisease",
                       marginal="box")
    fig.update_layout(height=400, width=700, showlegend=True)
    fig.update_traces(marker_line_width=1,marker_line_color="black")
    fig.show()

cols= df.columns
for col in cols:
    num_plot(df, col)
```

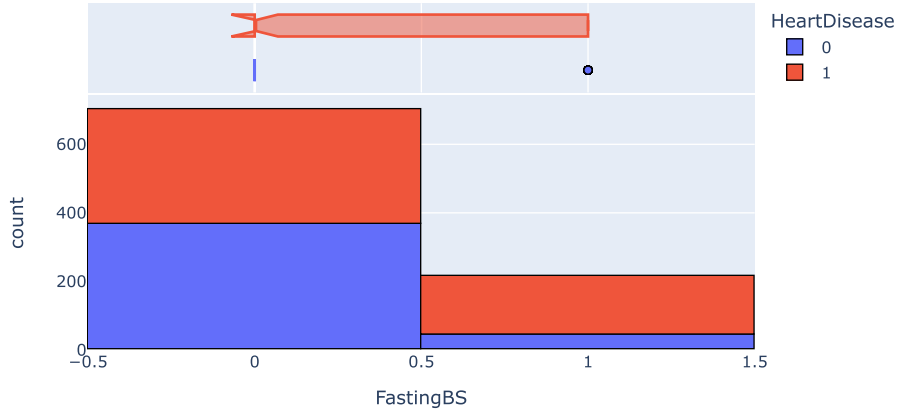


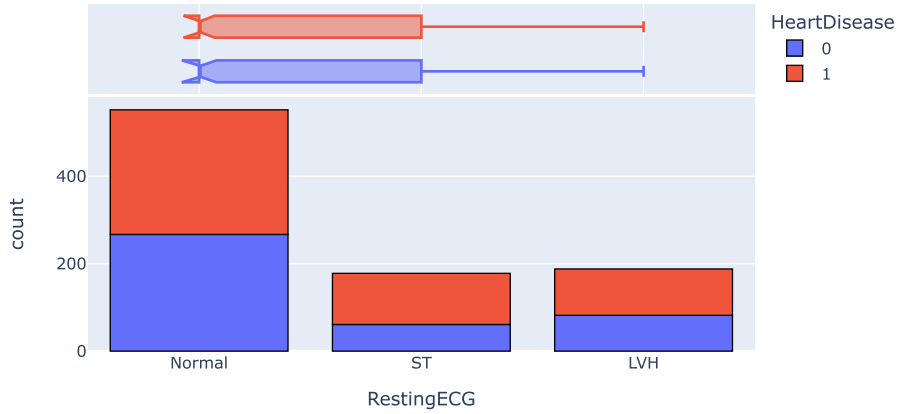

HeartDisease
0
1
count
600
400
200
0
M
F
Sex

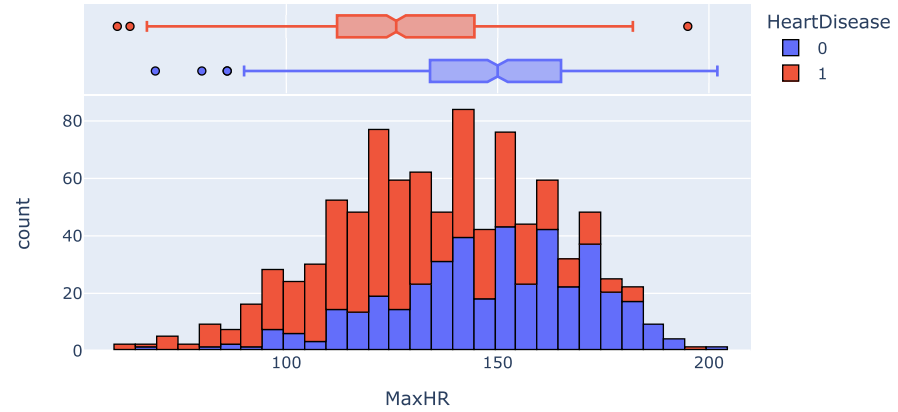

HeartDisease
0
1
count
500
400
300
200
100
0
ATA
NAP
TA
ASY
ChestPainType

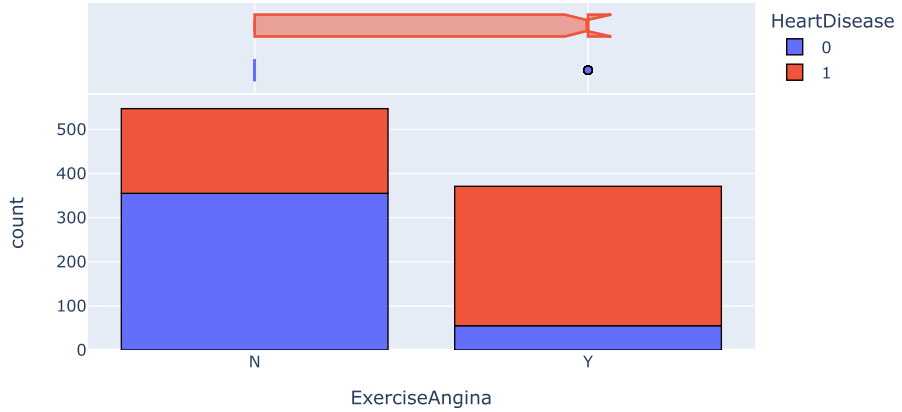

HeartDisease
0
1
count
150
100
50
0
0
50
100
150
200
RestingBP

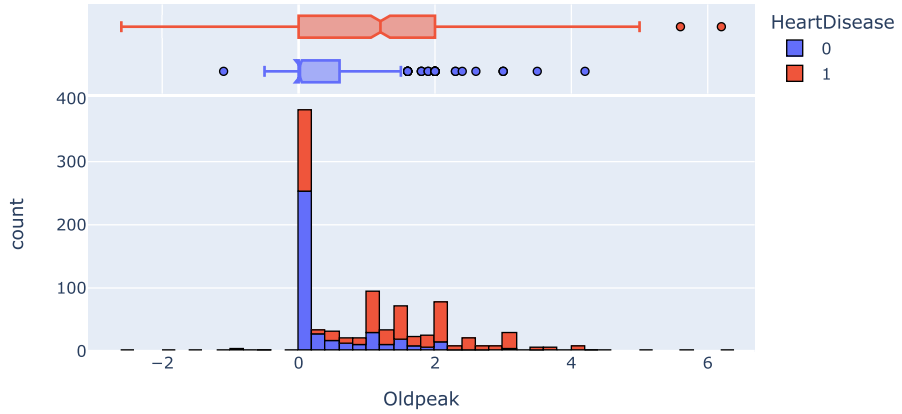

HeartDisease
0
1
count
150
100
50
0
0
100
200
300
400
500
600
Cholesterol

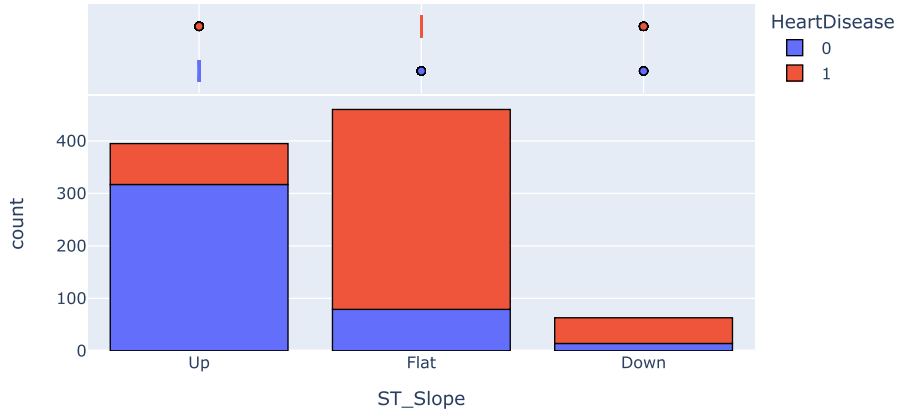

HeartDisease
0
1
count
600
400
200
0
−0.5
0
0.5
1
1.5
FastingBS

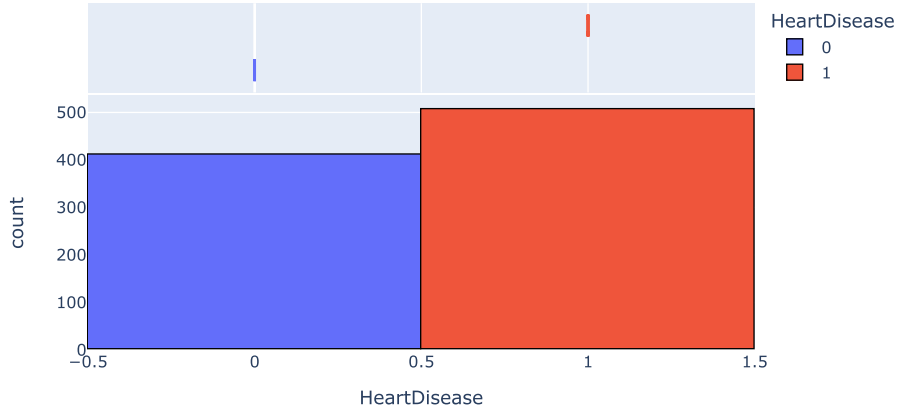

HeartDisease
0
1
count
400
200
0
Normal
ST
LVH
RestingECG


HeartDisease
0
1
count
0
20
40
60
80
100
150
200
MaxHR


HeartDisease
0
1
count
0
100
200
300
400
500
N
Y
ExerciseAngina


HeartDisease
0
1
count
0
100
200
300
400
−2
0
2
4
6
Oldpeak





We can see some different between Normal and HeartDisease.

Model

Data Preprocessing

After we see all the chart and analysis, we'll start some preprocessing the data before move to the model section.

```
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, ShuffleSplit, cross_va
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classification_report, accuracy
```

```
feature_cols = ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope']
target_cols = 'HeartDisease'

X = df[feature_cols]
y = df[target_cols]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42, stratify=
```

```
def data_preprocessing(X_train):
    ''' replace outliers with mean to make the distribution more normal. '''
    mean_chol = X_train[X_train.Cholesterol > 0]['Cholesterol'].mean()
    mean_rest = X_train[X_train.RestingBP > 0]['Cholesterol'].mean()

    X_train['Cholesterol'].replace(to_replace = [0,X_train[X_train['Cholesterol'] >= 500]['Cholester
    X_train['RestingBP'].replace(to_replace = [0,244.635389], value = mean_rest, inplace = True)

    return X_train
```

```
df.columns
```

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

```
y_train.value_counts()
```

```
1    432
0    348
Name: HeartDisease, dtype: int64
```


AutoML with FLAML Library

FLAML is a Python library (https://github.com/microsoft/FLAML) designed to automatically produce accurate machine learning models with low computational cost. It is fast and cheap. The simple and lightweight design makes it easy to use and extend, such as adding new learners. FLAML can

- serve as an economical AutoML engine,
- be used as a fast hyperparameter tuning tool, or
- be embedded in self-tuning software that requires low latency & resource in repetitive tuning tasks.

In this notebook, we use one real data example (binary classification) to showcase how to use FLAML library.

FLAML requires `Python>=3.6`. To run this notebook example, please install flaml with the `notebook` option:

```
pip install flaml[notebook]
```

```
!pip install flaml
```

```
Collecting flaml
  Downloading FLAML-0.7.1-py3-none-any.whl (160 kB)
     |████████████████████████████████| 160 kB 30.9 MB/s
Collecting xgboost<=1.3.3,>=0.90
  Downloading xgboost-1.3.3-py3-none-manylinux2010_x86_64.whl (157.5 MB)
     |████████████████████████████████| 157.5 MB 112 kB/s
Requirement already satisfied: NumPy>=1.16.2 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from flaml) (1
Requirement already satisfied: scipy>=1.4.1 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from flaml) (1.
Requirement already satisfied: pandas>=1.1.4 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from flaml) (1
Collecting lightgbm>=2.3.1
  Downloading lightgbm-3.3.1-py3-none-manylinux1_x86_64.whl (2.0 MB)
     |████████████████████████████████| 2.0 MB 42.8 MB/s
Requirement already satisfied: scikit-learn>=0.24 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from flam
Requirement already satisfied: pytz>=2017.3 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from pandas>=1.
Requirement already satisfied: python-dateutil>=2.7.3 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (
Requirement already satisfied: wheel in /usr/local/lib/python3.7/site-packages (from lightgbm>=2.3.1->flaml) (0.37.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from sc
Requirement already satisfied: joblib>=0.11 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from scikit-lea
Requirement already satisfied: six>=1.5 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (from python-da
Installing collected packages: xgboost, lightgbm, flaml
Successfully installed flaml-0.7.1 lightgbm-3.3.1 xgboost-1.3.3
WARNING: You are using pip version 20.1.1; however, version 21.3.1 is available.
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.
```

```
from flaml import AutoML
automl = AutoML()
```

```
settings = {
    "time_budget": 180,  # total running time in seconds
```

```
    "metric": 'accuracy',  # can be: 'r2', 'rmse', 'mae', 'mse', 'accuracy', 'roc_auc', 'roc_auc_ovr
                           # 'roc_auc_ovo', 'log_loss', 'mape', 'f1', 'ap', 'ndcg', 'micro_f1', 'mac
    "task": 'classification',  # task type
    "log_file_name": 'airlines_experiment.log',  # flaml log file
}
```

```
automl.fit(X_train=X_train, y_train=y_train, **settings)
```

```
[flaml.automl: 11-16 11:48:05] {1826} INFO - iteration 86, current learner rf
[flaml.automl: 11-16 11:48:05] {2029} INFO -  at 68.0s, estimator rf's best error=0.1372,       best estimator rf's
[flaml.automl: 11-16 11:48:05] {1826} INFO - iteration 87, current learner rf
[flaml.automl: 11-16 11:48:07] {2029} INFO -  at 69.7s, estimator rf's best error=0.1372,       best estimator rf's
[flaml.automl: 11-16 11:48:07] {1826} INFO - iteration 88, current learner rf
[flaml.automl: 11-16 11:48:08] {2029} INFO -  at 71.3s, estimator rf's best error=0.1372,       best estimator rf's
[flaml.automl: 11-16 11:48:08] {1826} INFO - iteration 89, current learner lrl1
[flaml.automl: 11-16 11:48:09] {2029} INFO -  at 72.0s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:09] {1826} INFO - iteration 90, current learner lrl1
[flaml.automl: 11-16 11:48:10] {2029} INFO -  at 72.7s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:10] {1826} INFO - iteration 91, current learner lrl1
[flaml.automl: 11-16 11:48:10] {2029} INFO -  at 73.4s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:10] {1826} INFO - iteration 92, current learner lrl1
[flaml.automl: 11-16 11:48:11] {2029} INFO -  at 74.2s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:11] {1826} INFO - iteration 93, current learner lrl1
[flaml.automl: 11-16 11:48:12] {2029} INFO -  at 74.8s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:12] {1826} INFO - iteration 94, current learner lrl1
[flaml.automl: 11-16 11:48:12] {2029} INFO -  at 75.6s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:12] {1826} INFO - iteration 95, current learner lrl1
[flaml.automl: 11-16 11:48:13] {2029} INFO -  at 76.3s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:13] {1826} INFO - iteration 96, current learner lrl1
[flaml.automl: 11-16 11:48:14] {2029} INFO -  at 77.0s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:14] {1826} INFO - iteration 97, current learner lrl1
[flaml.automl: 11-16 11:48:14] {2029} INFO -  at 77.7s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:14] {1826} INFO - iteration 98, current learner lrl1
[flaml.automl: 11-16 11:48:15] {2029} INFO -  at 78.4s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:15] {1826} INFO - iteration 99, current learner lrl1
[flaml.automl: 11-16 11:48:16] {2029} INFO -  at 79.1s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:16] {1826} INFO - iteration 100, current learner lrl1
[flaml.automl: 11-16 11:48:17] {2029} INFO -  at 79.8s, estimator lrl1's best error=0.2756,     best estimator rf's
[flaml.automl: 11-16 11:48:17] {1826} INFO - iteration 101, current learner lrl1
```

```
''' retrieve best config and best learner'''
print('Best ML leaner:', automl.best_estimator)
print('Best hyperparmeter config:', automl.best_config)
print('Best accuracy on validation data: {0:.4g}'.format(1-automl.best_loss))
print('Training duration of best run: {0:.4g} s'.format(automl.best_config_train_time))
```

```
Best ML leaner: lgbm
Best hyperparmeter config: {'n_estimators': 17.0, 'num_leaves': 4.0, 'min_child_samples': 25.0, 'learning_rate': 0.38
Best accuracy on validation data: 0.8692
Training duration of best run: 0.1457 s
```

```
automl.model
```

```
LGBMClassifier(colsample_bytree=0.9425456251062123,
               learning_rate=0.38406305271476093, max_bin=255,
               min_child_samples=25, n_estimators=17, num_leaves=4,
               objective='binary', reg_alpha=0.6913795549330475,
               reg_lambda=0.12357355094487746, subsample=0.7898698216956215)
```

```
''' compute predictions of testing dataset '''
y_pred = automl.model.predict(X_test)
print('Predicted labels', y_pred)
print('True labels', y_test)
y_pred_proba = automl.model.predict_proba(X_test)[:,1]
```

```
Predicted labels [0 0 1 1 0 0 0 1 0 0 1 0 1 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0
 0 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 1
 0 1 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 1 0]
True labels 29     0
615    1
383    1
```

```
783    0
684    1
      ..
238    1
242    1
234    0
605    0
492    1
Name: HeartDisease, Length: 138, dtype: int64
```

```
''' compute different metric values on testing dataset'''
from flaml.ml import sklearn_metric_loss_score
print('accuracy', '=', 1 - sklearn_metric_loss_score('accuracy', y_pred, y_test))
print('roc_auc', '=', 1 - sklearn_metric_loss_score('roc_auc', y_pred_proba, y_test))
print('log_loss', '=', sklearn_metric_loss_score('log_loss', y_pred_proba, y_test))
print('Training duration of best run: {0:.4g} s'.format(automl.best_config_train_time))
```

```
accuracy = 0.34782608695652173
roc_auc = 0.24469439728353148
log_loss = 0.9483898773683159
Training duration of best run: 0.1457 s
```

confusion matrix

$$\begin{bmatrix} C_{0,0} & C_{0,1} \\ C_{1,0} & C_{1,1} \end{bmatrix} = \begin{bmatrix} tn & fp \\ fn & tp \end{bmatrix}$$

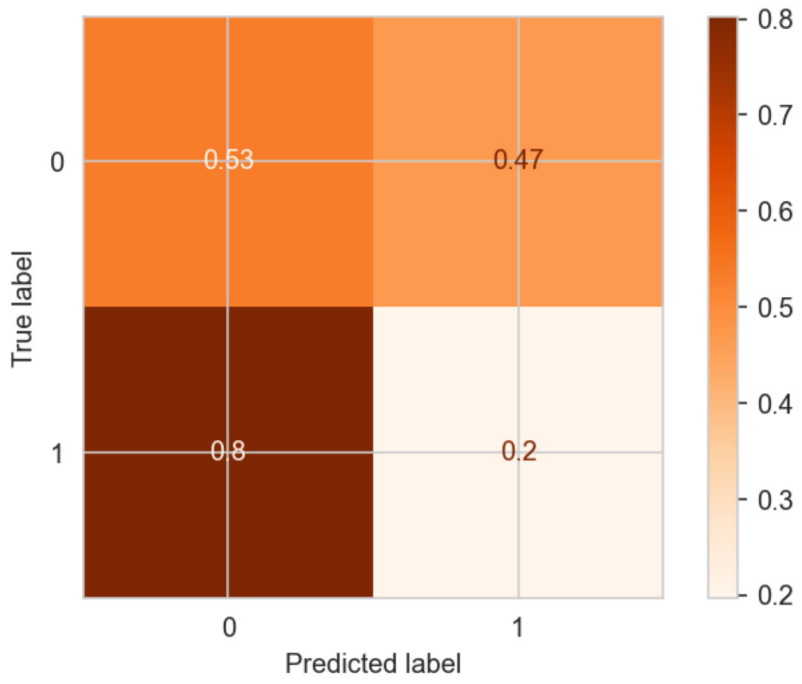
	predicted false	predicted true
actual false	tn	fp
actual true	fn	tp

```
cm = confusion_matrix(y_test, automl.model.predict(X_test))
cm
```

```
array([[33, 29],
       [61, 15]])
```

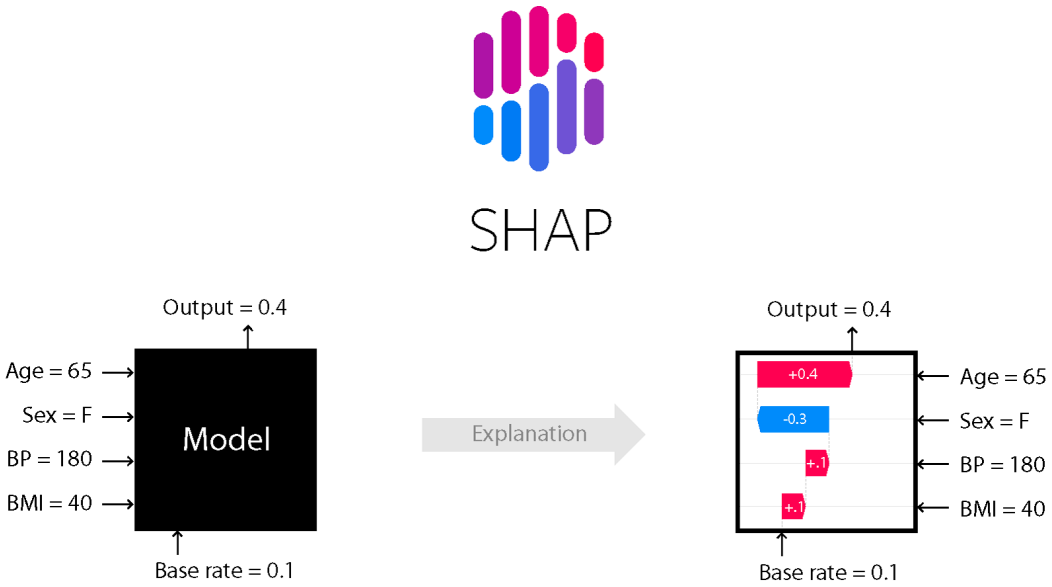
```
plot_confusion_matrix(automl.model, X_test, y_test, cmap='Oranges', normalize='true')
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fc3dce0fee0>
```



SHAP

> SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.



- doc: https://shap.readthedocs.io/en/latest/
- installation:
  - `pip install shap` or
  - `conda install -c conda-forge shap`

```
import shap
print(f'shap version {shap.__version__}')
# load JS visualization code to notebook
shap.initjs()
```

```
shap version 0.39.0
```

```
def case_detail(case_data):
    '''
    format obj returned from shap.force_plot()
    '''
    de=pd.DataFrame(case_data.data['features'])
    fcols=[]
    for i in case_data.data['features'].keys():
        fcols.append(case_data.data['featureNames'][i])
    de.columns=fcols
    return de

def individual_case_plot(explainer, X, case_index, verbose=False):
    """
    >>> individual_case_plot(explainer, X_train, 1)
    """
    shap_values = explainer.shap_values(X.iloc[[case_index]])
    g=shap.force_plot(explainer.expected_value, shap_values=shap_values, features=X.iloc[case_index,
    if verbose:
        pprint(g.__dict__)
    return g
```

```
# explain the model's predictions using SHAP
explainer = shap.TreeExplainer(automl.model)
shap_values = explainer.shap_values(X_test)
```

```
X
```

```
shap_values[:3]
```

```
shap.summary_plot(shap_values, X_train, plot_type="bar")
```

```
feature_cols
```

```
dshap=pd.DataFrame(shap_values, columns=feature_cols)
dshap
```