สถาบันเทคโนโลยีไทย-ญี่ปุ่น
Thai-Nichi Institute of Technology
泰日工業大学
สร้างนักคิด ผลิตนักปฏิบัติ สร้างนักประดิษฐ์ ผลิตนักบริหาร

DGE 209: Data Analytics

Heart Attack Dataset Analysis and Machine Learning

Prepared for

Dr. San Ratanasanya

Prepared by

Juntorn Thiantanukij (Jamie)

2021610213

March 7, 2022

Contents

Table of Figures

Problem Statement Setup

Cardiovascular diseases (CVDs) are the leading cause of death globally.

According to World Health Organization (WHO) data, an estimated 17.9 million people died from CVDs in 2019, representing 32% of all global deaths. Of these deaths, 85% were due to heart attack and stroke. (Approximately 15 million people died from heart attack and stroke)

The causes of heart attack have various factors. It could be from genetics, diets, lifestyle, others. It varies from age to age, and can happen to every gender.

This project aims to analyze the relationship and trends among those who are likely to have risk and no risk of having a heart attack. And predict the possibility of whether that person is prone to heart attack or not. In this way, hospitals are able to predict whether his/her patient are prone to heart attack or not and prevent before it happens.

Discussion

Data Collection and Preparation

Before importing the dataset to jupyter notebook, necessary libraries will be downloaded. This dataset consists of 14 attributes and 303 rows (patient information).

(Data downloaded from Kaggle, website is located in Appendix)

1. Age – Age of the patient (in years)
2. Sex – Sex of the patient
    a. 0 is female
    b. 1 is male
3. Cp – Chest pain types
    a. 0 is asymptomatic
    b. 1 is typical angina
    c. 2 is atypical angina
    d. 3 is non-anginal pain
4. Trtbps – Resting blood pressure measured in mmHg
5. Chol – Cholesterol measured in mg/dL fetched via BMI Sensor
6. Fbs – Fasting Blood Sugar > 120 mg/dl
    a. 0 is False
    b. 1 is True
7. Restecg – Resting Electrocardiographic Result
    a. 0 is hypertrophy
    b. 1 is normal
    c. 2 is having ST-T wave abnormality
8. Thalachh – Maximum heart rate achieved
9. Exang – Exercise induce angina
    a. 0 is no
    b. 1 is yes
10. Oldpeak – previous peak from ST depression induced by exercise relative to rest
11. Slp – the slope of the peak exercise ST segment
    a. 0 is down sloping
    b. 1 is flat
    c. 2 is upsloping
12. Ca – Number of major vessels (0-4) colored by fluoroscopy
13. Thall – Thallium stress test result
    a. 0 is reversible defect
    b. 1 is fixed defect
    c. 2 is normal
14. Output – the predicted attribute
    a. 0 is less chance of heart attack
    b. 1 is more chance of heart attack

Figure 1 Libraries and Importing Dataset

After importing dataset, the dataset can now be observed. This dataset consists of 303 rows and 14 columns. The first few rows can be observed below in the picture by using .head().

```
1 print("The shape of the dataset is : ", df.shape)
2 df.head()
```

```
The shape of the dataset is :  (303, 14)
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 2 Shape and Head of Dataset

The basic summary, statical data, data types, null values and unique values of this dataset also printed.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trtbps    303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalachh  303 non-null    int64  
 8   exng      303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slp       303 non-null    int64  
 11  caa       303 non-null    int64  
 12  thall     303 non-null    int64  
 13  output    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
1 dict = {}
2 for i in list(df.columns):
3     dict[i] = df[i].value_counts().shape[0]
4
5 pd.DataFrame(dict,index=["unique count"]).transpose()
```

	unique count
age	41
sex	2
cp	4
trtbps	49
chol	152
fbs	2
restecg	3
thalachh	91
exng	2
oldpeak	40
slp	3
caa	5
thall	4
output	2

```
1 df.isnull().sum()
```

```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
oldpeak     0
slp         0
caa         0
thall       0
output      0
dtype: int64
```

```
1 df.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000

Figure 3 df. info, describe and unique values

It can be observed that, this dataset does not have any null values. Most of the attributes of this dataset is int, only Oldpeak is the float datatype.

Next, divided the dataset columns to categorical features, continuous features, and target variable.

The code and statical data are shown below in fig. 4 and fig. 5

```
cat_cols = ['sex','exng','caa','cp','fbs','restecg','slp','thall']
con_cols = ["age","trtbps","chol","thalachh","oldpeak"]
target_col = ["output"]
print("The categorical cols are : ", cat_cols)
print("The continuous cols are : ", con_cols)
print("The target variable is :  ", target_col)
```

```
The categorical cols are :  ['sex', 'exng', 'caa', 'cp', 'fbs', 'restecg', 'slp', 'thall']
The continuous cols are :  ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']
The target variable is :   ['output']
```

Figure 4 Categorical, Continuous and Target Variables

```
df[con_cols].describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
age	303.0	54.366337	9.082101	29.0	47.5	55.0	61.0	77.0
trtbps	303.0	131.623762	17.538143	94.0	120.0	130.0	140.0	200.0
chol	303.0	246.264026	51.830751	126.0	211.0	240.0	274.5	564.0
thalachh	303.0	149.646865	22.905161	71.0	133.5	153.0	166.0	202.0
oldpeak	303.0	1.039604	1.161075	0.0	0.0	0.8	1.6	6.2

```
df[cat_cols].describe().transpose()
```

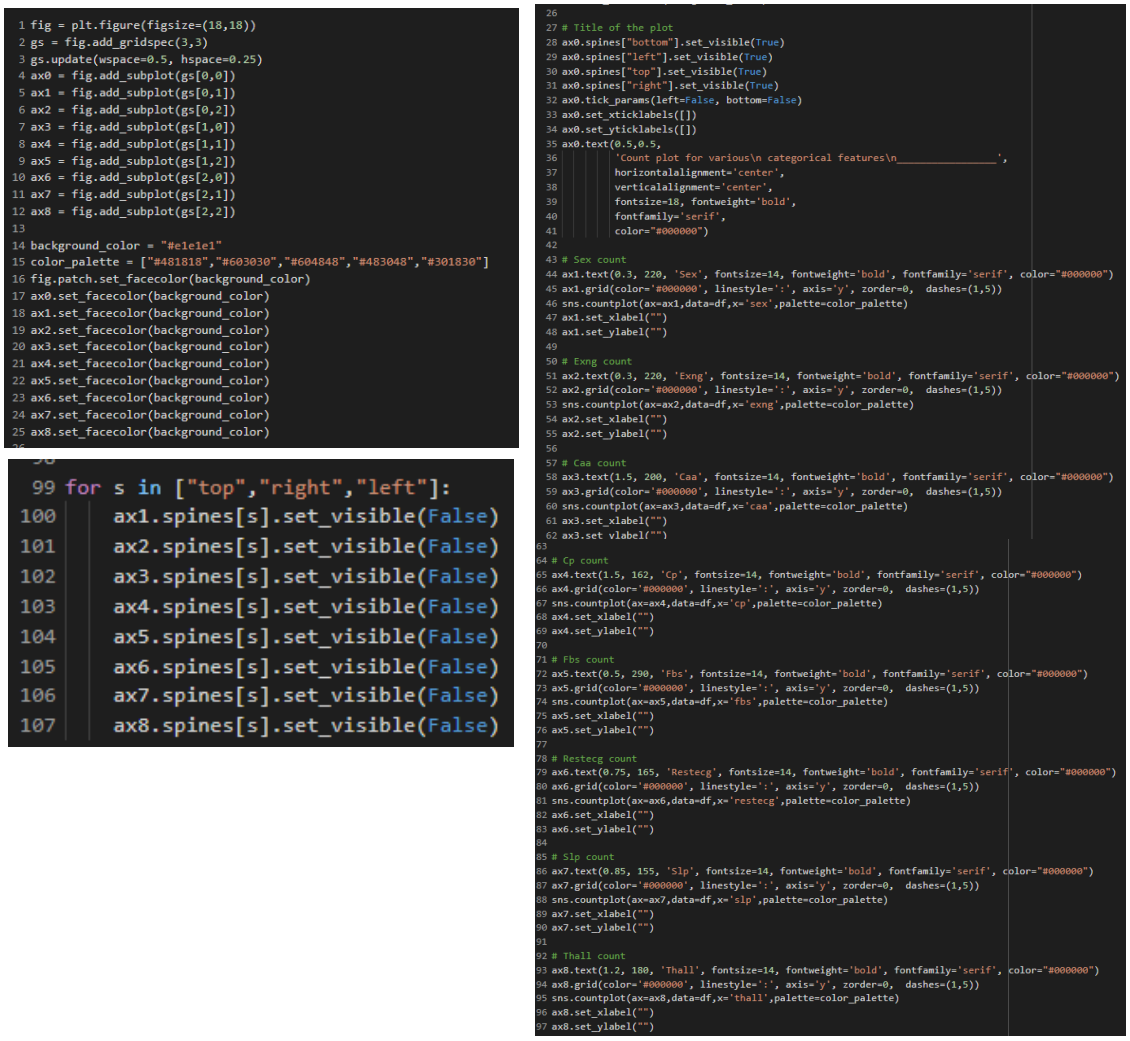
	count	mean	std	min	25%	50%	75%	max
sex	303.0	0.683168	0.466011	0.0	0.0	1.0	1.0	1.0
exng	303.0	0.326733	0.469794	0.0	0.0	0.0	1.0	1.0
caa	303.0	0.729373	1.022606	0.0	0.0	0.0	1.0	4.0
cp	303.0	0.966997	1.032052	0.0	0.0	1.0	2.0	3.0
fbs	303.0	0.148515	0.356198	0.0	0.0	0.0	0.0	1.0
restecg	303.0	0.528053	0.525860	0.0	0.0	1.0	1.0	2.0
slp	303.0	1.399340	0.616226	0.0	1.0	1.0	2.0	2.0
thall	303.0	2.313531	0.612277	0.0	2.0	2.0	3.0	3.0

Figure 5 Statical Information of Categorial and Continuous Features

After previewing all of this information of dataset, the exploratory data analysis process can begin.

Exploratory Data Analysis

In this part, the dataset will be analyzed to observe or investigate, and summarize their main characteristics. In this case, we will perform data analysis to discover patterns of those who are prone to have chances of having a heart attack or not. This report will be using visualization methods by data visualization method, graphs.



```
fig = plt.figure(figsize=(18,18))
gs = fig.add_gridspec(3,3)
gs.update(wspace=0.5, hspace=0.25)
ax0 = fig.add_subplot(gs[0,0])
ax1 = fig.add_subplot(gs[0,1])
ax2 = fig.add_subplot(gs[0,2])
ax3 = fig.add_subplot(gs[1,0])
ax4 = fig.add_subplot(gs[1,1])
ax5 = fig.add_subplot(gs[1,2])
ax6 = fig.add_subplot(gs[2,0])
ax7 = fig.add_subplot(gs[2,1])
ax8 = fig.add_subplot(gs[2,2])

background_color = "#e1e1e1"
color_palette = ["#481818","#603030","#604848","#483048","#301830"]
fig.patch.set_facecolor(background_color)
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)
ax2.set_facecolor(background_color)
ax3.set_facecolor(background_color)
ax4.set_facecolor(background_color)
ax5.set_facecolor(background_color)
ax6.set_facecolor(background_color)
ax7.set_facecolor(background_color)
ax8.set_facecolor(background_color)
```

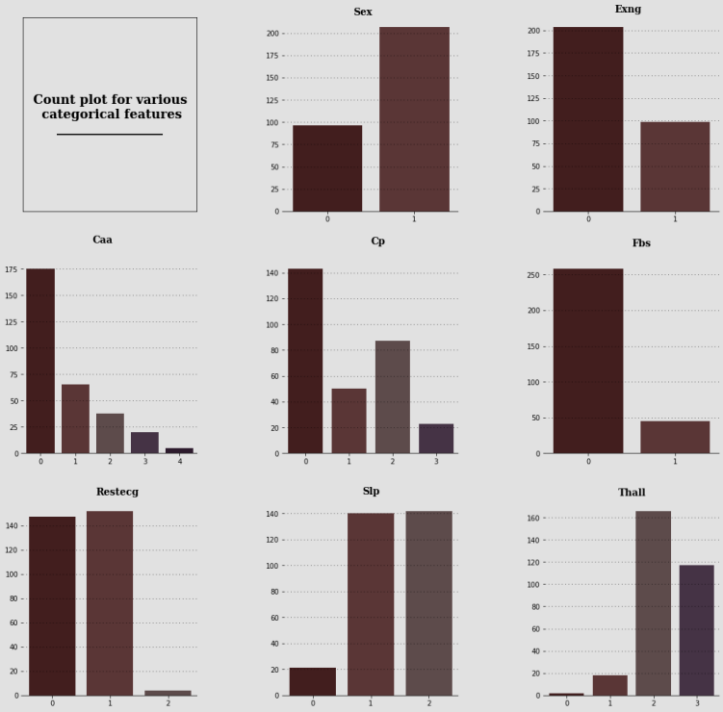
```
for s in ["top","right","left"]:
    ax1.spines[s].set_visible(False)
    ax2.spines[s].set_visible(False)
    ax3.spines[s].set_visible(False)
    ax4.spines[s].set_visible(False)
    ax5.spines[s].set_visible(False)
    ax6.spines[s].set_visible(False)
    ax7.spines[s].set_visible(False)
    ax8.spines[s].set_visible(False)
```

```

# Title of the plot
ax0.spines["bottom"].set_visible(True)
ax0.spines["left"].set_visible(True)
ax0.spines["top"].set_visible(True)
ax0.spines["right"].set_visible(True)
ax0.tick_params(left=False, bottom=False)
ax0.set_xticklabels([])
ax0.set_yticklabels([])
ax0.text(0.5,0.5,
         'Count plot for various\n categorical features\n_____________',
         horizontalalignment='center',
         verticalalignment='center',
         fontsize=18, fontweight='bold',
         fontfamily='serif',
         color="#000000")

# Sex count
ax1.text(0.3, 220, 'Sex', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax1.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax1,data=df,x='sex',palette=color_palette)
ax1.set_xlabel("")
ax1.set_ylabel("")

# Exng count
ax2.text(0.3, 220, 'Exng', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax2.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax2,data=df,x='exng',palette=color_palette)
ax2.set_xlabel("")
ax2.set_ylabel("")

# Caa count
ax3.text(1.5, 200, 'Caa', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax3.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax3,data=df,x='caa',palette=color_palette)
ax3.set_xlabel("")
ax3.set_ylabel("")

# Cp count
ax4.text(1.5, 162, 'Cp', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax4.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax4,data=df,x='cp',palette=color_palette)
ax4.set_xlabel("")
ax4.set_ylabel("")

# Fbs count
ax5.text(0.5, 290, 'Fbs', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax5.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax5,data=df,x='fbs',palette=color_palette)
ax5.set_xlabel("")
ax5.set_ylabel("")

# Restecg count
ax6.text(0.75, 165, 'Restecg', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax6.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax6,data=df,x='restecg',palette=color_palette)
ax6.set_xlabel("")
ax6.set_ylabel("")

# Slp count
ax7.text(0.85, 155, 'Slp', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax7.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax7,data=df,x='slp',palette=color_palette)
ax7.set_xlabel("")
ax7.set_ylabel("")

# Thall count
ax8.text(1.2, 180, 'Thall', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax8.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax8,data=df,x='thall',palette=color_palette)
ax8.set_xlabel("")
ax8.set_ylabel("")
```

Figure 6 Code Block for Categorical Features Graph

The figure above (Fig 6) is a code block that this project uses to present an overview of continuous features in the dataset. This project will put all the graphs in 1 figure since it is more efficient rather than using multiple code blocks. The code includes specifying the figure size, and spaces for each subplot. The color palette and background are also included. The next part is setting the title of the figure and inputting each attribute into a graph and then into a subplot. Lastly, remove the border for each plot for neatness.

Picture shown below (fig 5) is count plot of categorical features of this dataset.



From figure above, it can be concluded that in this dataset:

- Sex – Male (1) is more than Female (0)
- Exng (Exercise induced angina) – No (0) more than yes (1)
- Caa (Number of Major Vessels) – 0 is the highest count and 4 is the lowest count.
- Cp (Chest pain type) – Asymptomatic (0) is the highest whereas non-anginal pain (3) is the lowest.
- Fbs (Fast blooding sugar) – most of the patients have < 120 mg/dl
- Restecg (Resting electrocardiographic results) – Normal (1) is the highest followed by Hypertrophy (0) and ST-T wave abnormality (2) is the lowest.
- Slp (Slope) – Upsloping (2) is the highest followed by flat (1) and down-sloping (0) is the lowest.
- Thall (Thallium stress test result) – type 2 is the highest and type 0 is the lowest.

Next, let's take a look at Continuous features.

```
fig = plt.figure(figsize=(18,18))
gs = fig.add_gridspec(2,3)
gs.update(wspace=0.3, hspace=0.15)
ax0 = fig.add_subplot(gs[0,0])
ax1 = fig.add_subplot(gs[0,1])
ax2 = fig.add_subplot(gs[0,2])
ax3 = fig.add_subplot(gs[1,0])
ax4 = fig.add_subplot(gs[1,1])
ax5 = fig.add_subplot(gs[1,2])

background_color = "#efe8e8"
color_palette = ["#6f936e","#709ca4","#a96969","#abb27f","#7e747c"]
fig.patch.set_facecolor(background_color)
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)
ax2.set_facecolor(background_color)
ax3.set_facecolor(background_color)
ax4.set_facecolor(background_color)
ax5.set_facecolor(background_color)

# Title of the plot
ax0.spines["bottom"].set_visible(True)
ax0.spines["left"].set_visible(True)
ax0.spines["top"].set_visible(True)
ax0.spines["right"].set_visible(True)
ax0.tick_params(left=False, bottom=False)
ax0.set_xticklabels([])
ax0.set_yticklabels([])
ax0.text(0.5,0.5,
         'Boxen plot for various\n continuous features\n_________________',
         horizontalalignment='center',
         verticalalignment='center',
         fontsize=18, fontweight='bold',
         fontfamily='serif',
         color="#000000")
```

```

# Age
ax1.text(-0.05, 84, 'Age', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax1.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.violinplot(ax=ax1,y=df['age'],palette=["#6f936e"],width=0.6)
ax1.set_xlabel("")
ax1.set_ylabel("")

# Trtbps
ax2.text(-0.09, 213, 'Trtbps', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax2.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.violinplot(ax=ax2,y=df['trtbps'],palette=["#709ca4"],width=0.6)
ax2.set_xlabel("")
ax2.set_ylabel("")

# Chol
ax3.text(-0.05, 615, 'Chol', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax3.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.violinplot(ax=ax3,y=df['chol'],palette=["#a96969"],width=0.6)
ax3.set_xlabel("")
ax3.set_ylabel("")

# Thalachh
ax4.text(-0.12, 223.2, 'Thalachh', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax4.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.violinplot(ax=ax4,y=df['thalachh'],palette=["#abb27f"],width=0.6)
ax4.set_xlabel("")
ax4.set_ylabel("")

# oldpeak
ax5.text(-0.12, 7.2, 'Oldpeak', fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax5.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.violinplot(ax=ax5,y=df['oldpeak'],palette=["#7e747c"],width=0.6)
ax5.set_xlabel("")
ax5.set_ylabel("")

for s in ["top","right","left"]:
    ax1.spines[s].set_visible(False)
    ax2.spines[s].set_visible(False)
    ax3.spines[s].set_visible(False)
    ax4.spines[s].set_visible(False)
    ax5.spines[s].set_visible(False)
```

Figure 7 Code Block for Continuous Features

The figure above (Fig 7) is the code for plotting continuous features. For this feature, the violin plot will be used to show the full distribution of each attribute. Similarly, to the previous code block, the size of the whole figure will be set. As well as the subplot space and position. Again, the color palette and background are set, and then the tile of the figure. And lastly, the inputting each attribute in the subplot accordingly. The very last line, the for loop, is to remove the border between subplots.

The result of this code and be seen on the next page with its interpretation.

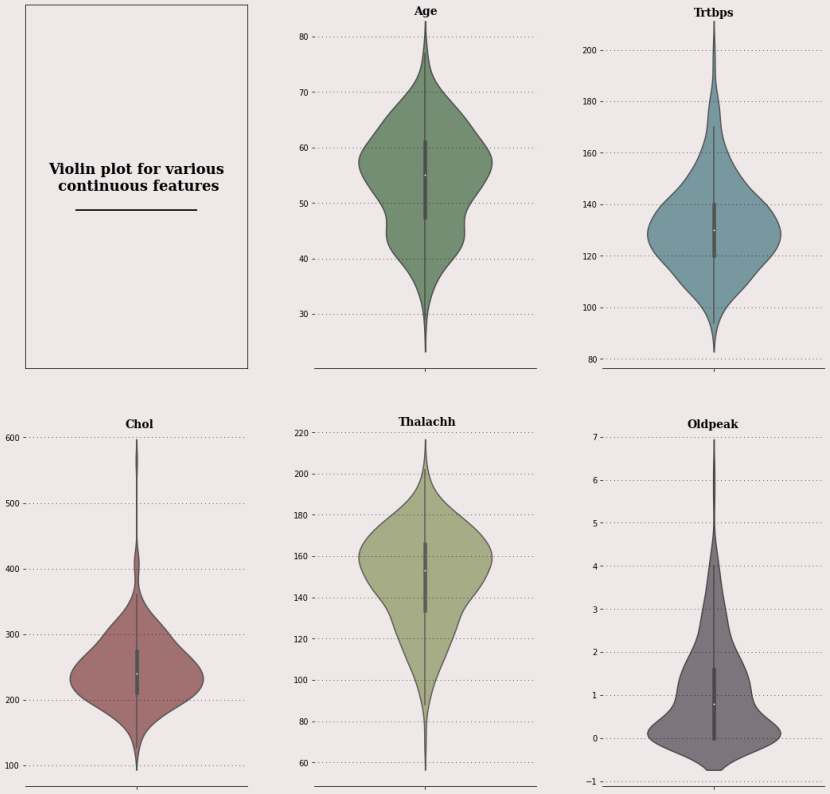


Figure 8 Violin plot for continuous features

From figure above, it can be observed that for continuous features:

- Age - Density distribution is highest for age group around between 50 to 60
- Trtbps (Resting Blood Pressure) – density distribution is highest around 120 to 140
- Chol (Cholesterol) – density distribution is highest between 200 and 300
- Thalachh (Maximum heart rate achieved) – density distribution highest around 135 to 165
- Oldpeak (previous peak from ST Depression) – density distribution highest around 0 to 1.5

It is worth mentioning that the reason this project does not remove outliers is that all of these attributes' values are possible to various values. And these are the data point to help predict heart attack tendency. Removing outliers does not increase the accuracy of the machine learning model too.

And lastly, target variable.

```
fig = plt.figure(figsize=(18,7))
gs = fig.add_gridspec(1,2)
gs.update(wspace=0.3, hspace=0.15)
ax0 = fig.add_subplot(gs[0,0])
ax1 = fig.add_subplot(gs[0,1])

background_color = "#fff2cc"
color_palette = ["#e42f3f","#f6a623","#e073ff","#70f089","#f08970"]
fig.patch.set_facecolor(background_color)
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)

# Title of the plot
ax0.text(0.5,0.5,"Count of the target\n_____",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')

ax0.set_xticklabels([])
ax0.set_yticklabels([])
ax0.tick_params(left=False, bottom=False)

# Target Count
ax1.text(0.35,177,"Output",fontsize=14, fontweight='bold', fontfamily='serif', color="#000000")
ax1.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax1, data=df, x = 'output',palette = color_palette)
ax1.set_xlabel("")
ax1.set_ylabel("")
ax1.set_xticklabels(["Low chances of attack(0)","High chances of attack(1)"])

ax0.spines["top"].set_visible(False)
ax0.spines["left"].set_visible(False)
ax0.spines["bottom"].set_visible(False)
ax0.spines["right"].set_visible(False)
ax1.spines["top"].set_visible(False)
ax1.spines["left"].set_visible(False)
ax1.spines["right"].set_visible(False)
```

Figure 9 Code Block for Target Variable Graph

The code block above (fig 9) is the code for constructing the target variable graph. The method is similar to the previous code. The size of the figure is set as well as the subplot. Then the title and inputting attribute into the graph. And lastly, set the border's visibility to false.

And here's the result:

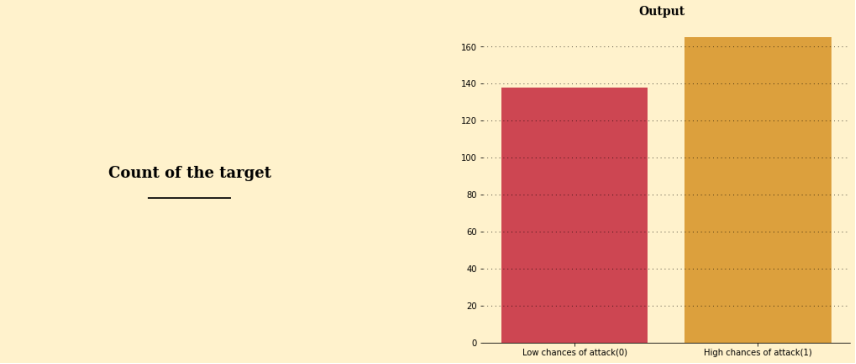


Figure 10 Count of the Target Variable

From figure above, it can be concluded that in this dataset most of the patient have high chance of heart attack.

Next, correlation matrix to see whether there's relationship between variables or not.

The correlation matrix table can be called by using corr() function on the dataset variable as shown below in Fig. 11.

```
df_corr = df.corr().transpose()
df_corr
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
age	1.000000	-0.098447	-0.068653	0.279351	0.213678	0.121308	-0.116211	-0.398522	0.096801	0.210013	-0.168814	0.276326	0.068001	-0.225439
sex	-0.098447	1.000000	-0.049353	-0.056769	-0.197912	0.045032	-0.058196	-0.044020	0.141664	0.096093	-0.030711	0.118261	0.210041	-0.280937
cp	-0.068653	-0.049353	1.000000	0.047608	-0.076904	0.094444	0.044421	0.295762	-0.394280	-0.149230	0.119717	-0.181053	-0.161736	0.433798
trtbps	0.279351	-0.056769	0.047608	1.000000	0.123174	0.177531	-0.114103	-0.046698	0.067616	0.193216	-0.121475	0.101389	0.062210	-0.144931
chol	0.213678	-0.197912	-0.076904	0.123174	1.000000	0.013294	-0.151040	-0.009940	0.067023	0.053952	-0.004038	0.070511	0.098803	-0.085239
fbs	0.121308	0.045032	0.094444	0.177531	0.013294	1.000000	-0.084189	-0.008567	0.025665	0.005747	-0.059894	0.137979	-0.032019	-0.028046
restecg	-0.116211	-0.058196	0.044421	-0.114103	-0.151040	-0.084189	1.000000	0.044123	-0.070733	-0.058770	0.093045	-0.072042	-0.011981	0.137230
thalachh	-0.398522	-0.044020	0.295762	-0.046698	-0.009940	-0.008567	0.044123	1.000000	-0.378812	-0.344187	0.386784	-0.213177	-0.096439	0.421741
exng	0.096801	0.141664	-0.394280	0.067616	0.067023	0.025665	-0.070733	-0.378812	1.000000	0.288223	-0.257748	0.115739	0.206754	-0.436757
oldpeak	0.210013	0.096093	-0.149230	0.193216	0.053952	0.005747	-0.058770	-0.344187	0.288223	1.000000	-0.577537	0.222682	0.210244	-0.430696
slp	-0.168814	-0.030711	0.119717	-0.121475	-0.004038	-0.059894	0.093045	0.386784	-0.257748	-0.577537	1.000000	-0.080155	-0.104764	0.345877
caa	0.276326	0.118261	-0.181053	0.101389	0.070511	0.137979	-0.072042	-0.213177	0.115739	0.222682	-0.080155	1.000000	0.151832	-0.391724
thall	0.068001	0.210041	-0.161736	0.062210	0.098803	-0.032019	-0.011981	-0.096439	0.206754	0.210244	-0.104764	0.151832	1.000000	-0.344029
output	-0.225439	-0.280937	0.433798	-0.144931	-0.085239	-0.028046	0.137230	0.421741	-0.436757	-0.430696	0.345877	-0.391724	-0.344029	1.000000

Figure 11 Correlation Matrix Table

Or put in sns.heatmap (seaborn) for a better visualization. The code is shown below.

```
fig = plt.figure(figsize=(20,16))
gs = fig.add_gridspec(1,1)
gs.update(wspace=0.3, hspace=0.15)
ax0 = fig.add_subplot(gs[0,0])

color_palette = ["#5833ff","#da8829"]
mask = np.triu(np.ones_like(df_corr))
ax0.text(5.5,-0.1,"Correlation Matrix",fontsize=22, fontweight='bold', fontfamily='serif', color="#000000")

sns.heatmap(df_corr, annot=True,cmap='YlGnBu')
plt.show()
```

Figure 12 Full Correlation Matrix

The correlation matrix can show only half of the triangle too, since both sides has the same value.

```
fig = plt.figure(figsize=(14,14))
gs = fig.add_gridspec(1,1)
gs.update(wspace=0.3, hspace=0.15)
ax0 = fig.add_subplot(gs[0,0])

color_palette = ["#5833ff","#da8829"]
mask = np.triu(np.ones_like(df_corr))
ax0.text(5,-0.1,"Correlation Matrix",fontsize=22, fontweight='bold', fontfamily='serif', color="#000000")

sns.heatmap(df_corr, mask=mask, annot=True,cmap='YlGnBu')
plt.show()
```

Figure 13 Half Triangle of Correlation Matrix

The only difference is the mask to show only half of the triangle.

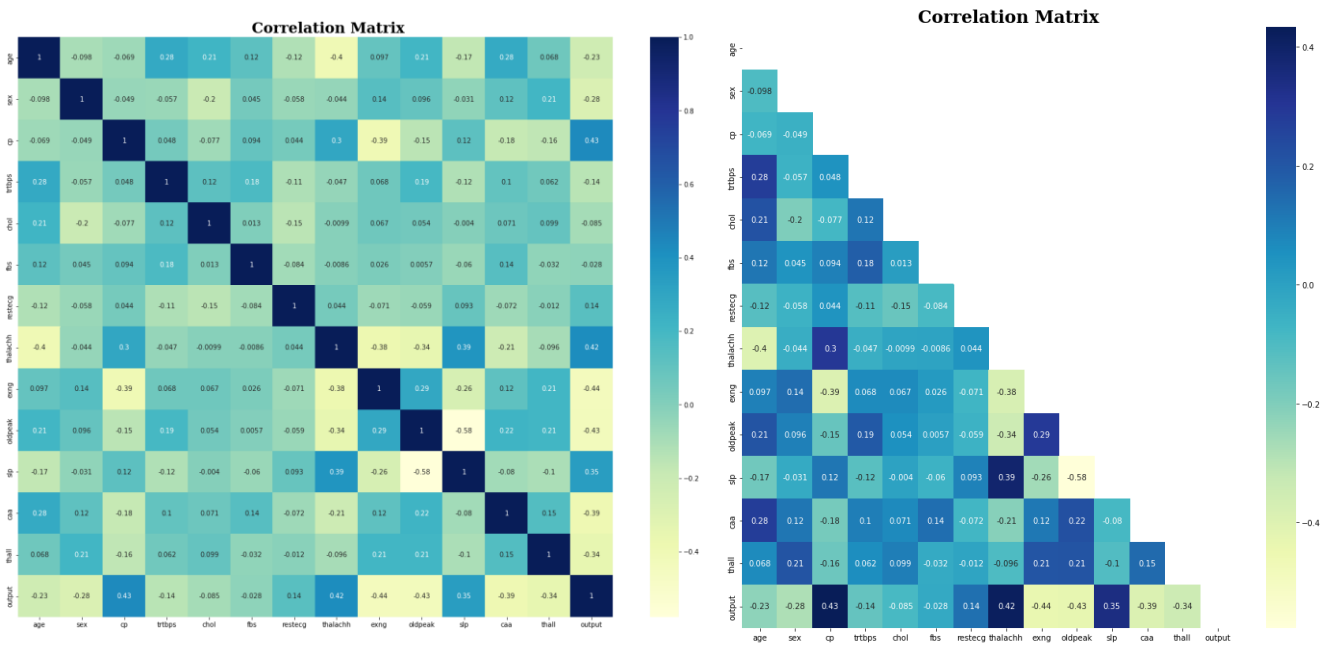


Figure 14 Full and Half Correlation Matrix

The reason to construct correlation matrix is that it can be easily observed which attribute has a positive and negative correlation to one another. For example, slp (slope) and the old peak have quite a strong negative correlation to each other, whereas slp (slope) and thalachh (maximum heart rate achieved) has a positive correlation to each other.

This project also constructs other type of heatmap, the scatterplot heatmap.

```
fig = plt.figure(figsize=(12,12))
corr_mat = df.corr().stack().reset_index(name="correlation")
g = sns.relplot(
    data=corr_mat,
    x="level_0", y="level_1", hue="correlation", size="correlation",
    palette="YlGnBu", hue_norm=(-1, 1), edgecolor=".7",
    height=10, sizes=(50, 250), size_norm=(-.2, .8),
)
g.set(xlabel="features on X", ylabel="featurs on Y", aspect="equal")
g.fig.suptitle('Scatterplot heatmap',fontsize=22, fontweight='bold', fontfamily='serif', color="#000000")
g.despine(left=True, bottom=True)
g.ax.margins(.02)
for label in g.ax.get_xticklabels():
    label.set_rotation(90)
for artist in g.legend.legendHandles:
    artist.set_edgecolor(".7")
plt.show()
```

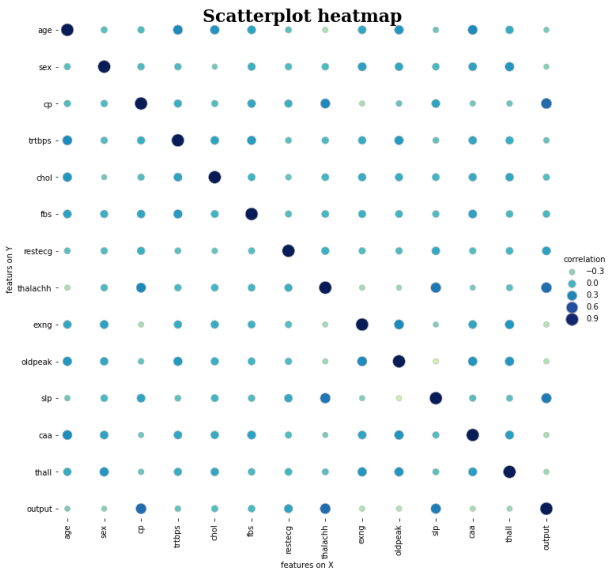


Figure 15 Scatterplot Heatmap and Code

The code above is for setting up title, label and color position and input.

Next, let’s take a look at attribute relationship to target variable.

```
fig = plt.figure(figsize=(18,18))
gs = fig.add_gridspec(5,2)
gs.update(wspace=0.5, hspace=0.5)
ax0 = fig.add_subplot(gs[0,0])
ax1 = fig.add_subplot(gs[0,1])
ax2 = fig.add_subplot(gs[1,0])
ax3 = fig.add_subplot(gs[1,1])
ax4 = fig.add_subplot(gs[2,0])
ax5 = fig.add_subplot(gs[2,1])
ax6 = fig.add_subplot(gs[3,0])
ax7 = fig.add_subplot(gs[3,1])
ax8 = fig.add_subplot(gs[4,0])
ax9 = fig.add_subplot(gs[4,1])

background_color = "#ddd7cb"
color_palette = ["#7dbb95","#e6c672","#8bac9b","#9b8bac","#ac9b8b"]
fig.patch.set_facecolor(background_color)
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)
ax2.set_facecolor(background_color)
ax3.set_facecolor(background_color)
ax4.set_facecolor(background_color)
ax5.set_facecolor(background_color)
ax6.set_facecolor(background_color)
ax7.set_facecolor(background_color)
ax8.set_facecolor(background_color)
ax9.set_facecolor(background_color)
```

Figure 16 Attribute Relation to Target Variable Code Block

```

# Age title
ax0.text(0.5,0.5,"Distribution of age\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax0.spines["bottom"].set_visible(False)
ax0.set_xticklabels([])
ax0.set_yticklabels([])
ax0.tick_params(left=False, bottom=False)

# Age
ax1.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax1, data=df, x='age',hue="output", fill=True,palette=["#7dbb95","#e6c672"], alpha=.5, linewidth=0)
ax1.set_xlabel("")
ax1.set_ylabel("")

# TrTbps title
ax2.text(0.5,0.5,"Distribution of trtbps\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax2.spines["bottom"].set_visible(False)
ax2.set_xticklabels([])
ax2.set_yticklabels([])
ax2.tick_params(left=False, bottom=False)

# TrTbps
ax3.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax3, data=df, x='trtbps',hue="output", fill=True,palette=["#7dbb95","#e6c672"], alpha=.5, linewidth=0)
ax3.set_xlabel("")
ax3.set_ylabel("")

# Chol title
ax4.text(0.5,0.5,"Distribution of chol\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax4.spines["bottom"].set_visible(False)
ax4.set_xticklabels([])
ax4.set_yticklabels([])
ax4.tick_params(left=False, bottom=False)

# Chol
ax5.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax5, data=df, x='chol',hue="output", fill=True,palette=["#7dbb95","#e6c672"], alpha=.5, linewidth=0)
ax5.set_xlabel("")
ax5.set_ylabel("")

# Thalachh title
ax6.text(0.5,0.5,"Distribution of thalachh\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax6.spines["bottom"].set_visible(False)
ax6.set_xticklabels([])
ax6.set_yticklabels([])
ax6.tick_params(left=False, bottom=False)

# Thalachh
ax7.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax7, data=df, x='thalachh',hue="output", fill=True,palette=["#7dbb95","#e6c672"], alpha=.5, linewidth=0)
ax7.set_xlabel("")
ax7.set_ylabel("")


# Oldpeak title
ax8.text(0.5,0.5,"Distribution of oldpeak\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax8.spines["bottom"].set_visible(False)
ax8.set_xticklabels([])
ax8.set_yticklabels([])
ax8.tick_params(left=False, bottom=False)

# Oldpeak
ax9.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax9, data=df, x='oldpeak',hue="output", fill=True,palette=["#7dbb95","#e6c672"], alpha=.5, linewidth=0)
ax9.set_xlabel("")
ax9.set_ylabel("")

for i in ["top","left","right"]:
    ax0.spines[i].set_visible(False)
    ax2.spines[i].set_visible(False)
    ax4.spines[i].set_visible(False)
    ax6.spines[i].set_visible(False)
    ax8.spines[i].set_visible(False)
```

The code in above figure (fig.16) is quite similar to previous graph. The size of the figure, title and subplot position are set. As well as the color palette and inputting each attribute into subplot respectively. In this code block, the continuous features are used. And lastly, as usual, removing the subplot border in for loop. The result and the graphs' meaning can be seen below.

The graph will be presented one by one here.

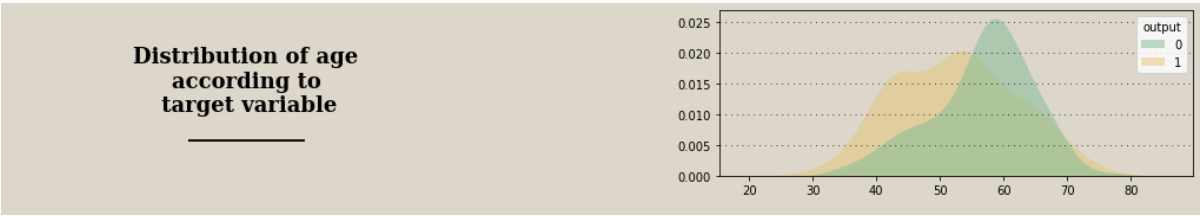


Figure 17 Distribution of Age according to Target Variable

(Fig 17) There's a high density around age 60 that has less chance of having a heart attack. And for those who have a chance of having a heart attack is highest around 40 to 50.

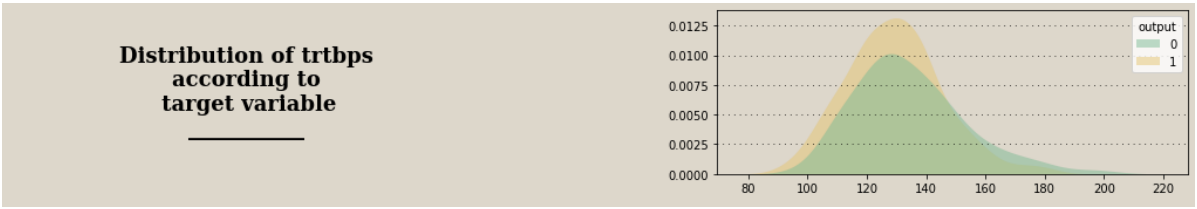


Figure 18 Distribution of TRTBPS according to Target Variable

(Fig 18) For resting blood pressure related to the target variable, there's an overlap between those who have high chances of heart attack and those who are not. However, the density distribution of having a high possibility of having a heart attack is higher at between 120 to 140 mmHG.

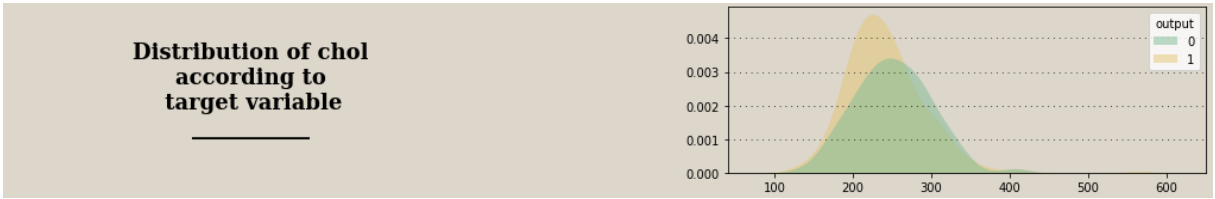


Figure 19 Distribution of Chol According to Target Variable

(Fig 19) Again, there is a slight overlap between the two variables. However, the highest peak for those who have less possibility of having a heart attack is around between 200 and 300 cholesterol levels (mg/dl). Whereas those who have a high chance of having a heart attack, their cholesterol level is around 200 mg/dl.

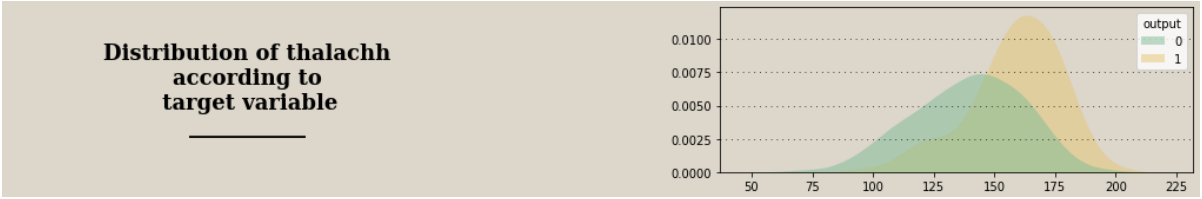


Figure 20 Distribution of Thalachh according to Target Variable

(Fig 20) In regard to the maximum heart rate achieved, the highest density for those who are prone to having a heart attack is around 160 to 175 bpm (beats per minute). Where those who have smaller chances of having a heart attack, their maximum heart rate achieved was around 140 to 150 bpm.

Figure 21 Distribution of Oldpeak according to Target Variable

(Figure 21) And lastly, the Oldpeak relations to target variable. Those who have Oldpeak at 0 are more likely to have a heart attack. There is a slight overlap with a smaller chance of having a heart attack at 0. And those who have Oldpeak more than 1 are more likely not to have a heart attack.

Next, other attribute relationship to target variable.

```python
fig = plt.figure(figsize=(18,20))
gs = fig.add_gridspec(6,2)
gs.update(wspace=0.5, hspace=0.5)
ax0 = fig.add_subplot(gs[0,0])
ax1 = fig.add_subplot(gs[0,1])
ax2 = fig.add_subplot(gs[1,0])
ax3 = fig.add_subplot(gs[1,1])
ax4 = fig.add_subplot(gs[2,0])
ax5 = fig.add_subplot(gs[2,1])
ax6 = fig.add_subplot(gs[3,0])
ax7 = fig.add_subplot(gs[3,1])
ax8 = fig.add_subplot(gs[4,0])
ax9 = fig.add_subplot(gs[4,1])
ax10 = fig.add_subplot(gs[5,0])
ax11 = fig.add_subplot(gs[5,1])

background_color = "#f9f3e4"
color_palette = ["#86876d","#4a6974","#6aac90","#5833ff","#da8829"]
fig.patch.set_facecolor(background_color)
ax0.set_facecolor(background_color)
ax1.set_facecolor(background_color)
ax2.set_facecolor(background_color)
ax3.set_facecolor(background_color)
ax4.set_facecolor(background_color)
ax5.set_facecolor(background_color)
ax6.set_facecolor(background_color)
ax7.set_facecolor(background_color)
ax8.set_facecolor(background_color)
ax9.set_facecolor(background_color)
ax10.set_facecolor(background_color)
ax11.set_facecolor(background_color)
```

```python
# Thall title
ax6.text(0.5,0.5,"Distribution of thall\naccording to\n target variable\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax6.text(1,.5,"Thalium Stress\nTest Result\n0, 1, 2, 3",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )
ax6.spines["bottom"].set_visible(False)
ax6.set_xticklabels([])
ax6.set_yticklabels([])
ax6.tick_params(left=False, bottom=False)

# Thall
ax7.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax7, data=df, x='thall',hue="output", fill=True,palette=["#86876d","#4a6974"], alpha=.5, linewidth=0)
ax7.set_xlabel("")
ax7.set_ylabel("")

# Thalachh title
ax8.text(0.5,0.5,"Boxen plot of\nthalachh according\nto outcome\n_______",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax8.text(1,.5,"Maximum heart\nrate achieved",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )

ax8.spines["bottom"].set_visible(False)
ax8.set_xticklabels([])
ax8.set_yticklabels([])
ax8.tick_params(left=False, bottom=False)

# Thalachh
ax9.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.boxenplot(ax=ax9, data=df,x='output',y='thalachh',palette=["#86876d","#4a6974"])
ax9.set_xlabel("")
ax9.set_ylabel("")

# Exng title
ax10.text(0.5,0.5,"Strip Plot of\nexng vs age\n_______",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax10.text(1,.5,"Exercise induced\nangina\n0 - No\n1 - Yes",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )
ax10.spines["bottom"].set_visible(False)
ax10.set_xticklabels([])
ax10.set_yticklabels([])
ax10.tick_params(left=False, bottom=False)

# Exng
ax11.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.stripplot(ax=ax11, data=df,x='exng',y='age',hue='output',palette=["#86876d","#4a6974"])
ax9.set_xlabel("")
ax9.set_ylabel("")

```

```python

# Cp title
# 0 = Typical Angina, 1 = Atypical Angina, 2 = Non-anginal Pain, 3 = Asymptomatic
ax0.text(0.5,0.5,"Chest pain\ndistribution\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax0.spines["bottom"].set_visible(False)
ax0.set_xticklabels([])
ax0.set_yticklabels([])
ax0.tick_params(left=False, bottom=False)
ax0.text(1,.5,"0 - Asymptomatic\n1 - Typical Angina\n2 - Atypical anginal\n3 - Non-anginal Pain",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )

# Cp
ax1.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax1, data=df, x='cp',hue="output", fill=True,palette=["#86876d","#4a6974"], alpha=.5, linewidth=0)
ax1.set_xlabel("")
ax1.set_ylabel("")

# Caa title
ax2.text(0.5,0.5,"Number of\nmajor vessels\n_________",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax2.text(1,.5,"0 vessels\n1 vessel\n2 vessels\n3 vessels\n4 vessels",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )

ax2.spines["bottom"].set_visible(False)
ax2.set_xticklabels([])
ax2.set_yticklabels([])
ax2.tick_params(left=False, bottom=False)

# Caa
ax3.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.kdeplot(ax=ax3, data=df, x='caa',hue="output", fill=True,palette=["#86876d","#4a6974"], alpha=.5, linewidth=0)
ax3.set_xlabel("")
ax3.set_ylabel("")

# Sex title
ax4.text(0.5,0.5,"Heart Attack\naccording to\nsex\n_____",
        horizontalalignment = 'center',
        verticalalignment = 'center',
        fontsize = 18,
        fontweight='bold',
        fontfamily='serif',
        color='#000000')
ax4.text(1,.5,"0 - Female\n1 - Male",
        horizontalalignment = 'center',
         verticalalignment = 'center',
        fontsize = 14
        )
ax4.spines["bottom"].set_visible(False)
ax4.set_xticklabels([])
ax4.set_yticklabels([])
ax4.tick_params(left=False, bottom=False)

# Sex
ax5.grid(color='#000000', linestyle=':', axis='y', zorder=0,  dashes=(1,5))
sns.countplot(ax=ax5,data=df,x='sex',palette=["#86876d","#4a6974"], hue='output')
ax5.set_xlabel("")
ax5.set_ylabel("")

```

```python
for i in ["top","left","right"]:
    ax0.spines[i].set_visible(False)
    ax2.spines[i].set_visible(False)
    ax4.spines[i].set_visible(False)
    ax6.spines[i].set_visible(False)
    ax8.spines[i].set_visible(False)
    ax10.spines[i].set_visible(False)
```

Figure 22 Code Block for other relations to Target Variable

The code above in Figure 22, uses the same method as other graphs. The size of the figure, subplot, color palette and background, title font and position. As usual, the input is the attribute put accordingly to each subplot. And lastly, same as previously, the for loop to remove the border of each subplot.

The results and interpretation are shown below.

Figures below are other relation to the target variable.

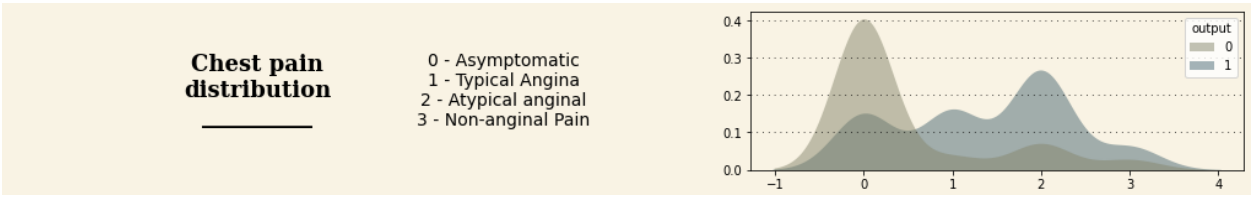


Figure 23 Chest Pain Distribution to Target Variable

(Figure 23) Those who have asymptomatic chest pain, or no sign of chest pain are most likely not going to experience a heart attack. Other types of chest pain have a higher probability of having a heart attack, especially chest pain type 2, atypical anginal pain.

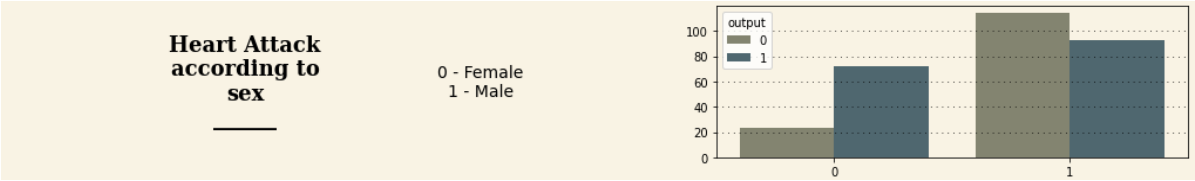


Figure 24 Heart Attack According to Sex

(Figure 24) According to the graph, it can be concluded that females are more prone to a heart attack. Almost double of those who have less chance of having a heart attack. Whereas males are less prone to a heart attack. But there's not much difference compare to those who have high chance experiencing heart attack.

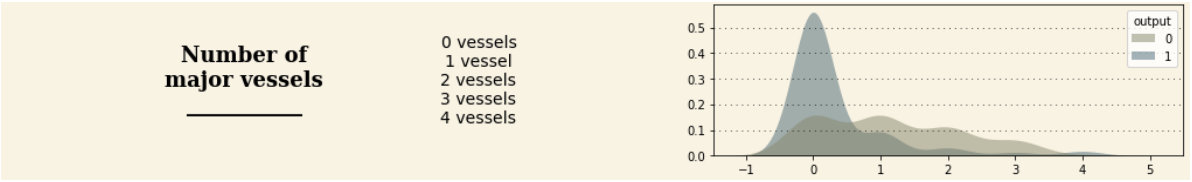


Figure 25 Heart Attack according to No. of Major Vessels

(Figure 25) There is a slight overlap at 0 number of vessels whether the patient has a risk of experiencing a heart attack or not. However, the probability of experiencing a heart attack is higher. And as the number of major vessels increases, the chances of having heart attack decrease.

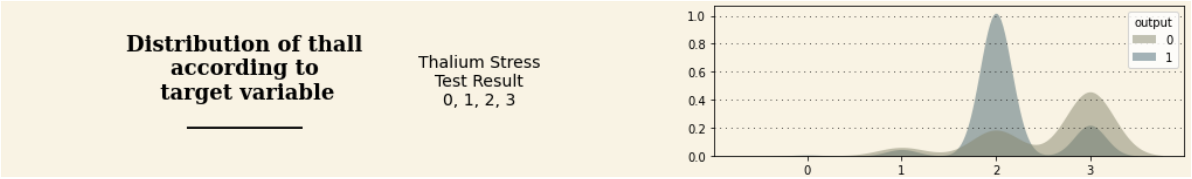


Figure 26 Distribution of thall according to target variable

(Figure 26) Those who have thallium stress test result type 2 are most likely prone to a heart attack. There is a slight overlap in thallium stress test result type 3, but the probability of not having a heart attack tendency is higher.

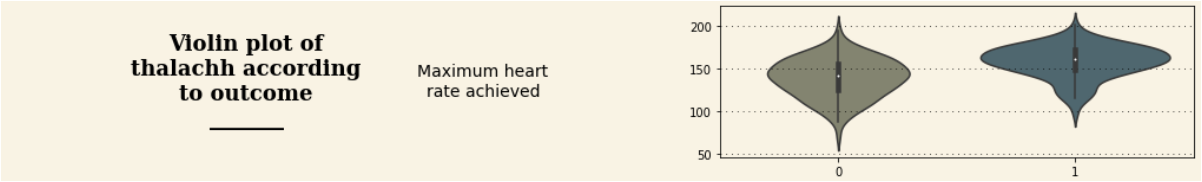


Figure 27 Violin Plot of Thalachh According to Outcome

(Figure 27) The violin plot above shows the distribution of the maximum heart rate achieved. Those who are likely to experience a heart attack have a higher maximum heart at around more than 150 bpm. Whereas those who are not likely to experience heart attack achieved the maximum heart rate at around less than 150 bpm.

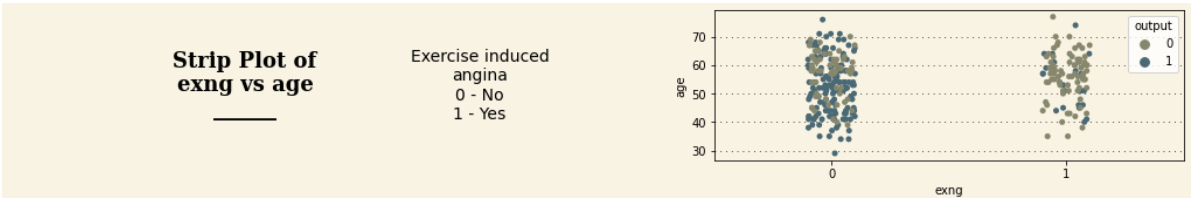


Figure 28 Strip Plot of EXNG vs AGE

(Figure 28) This figure uses Strip Plot because it is a good complement to a boxplot or violin plot in cases where all observations are shown along with some representation of the underlying distribution. According to the graph, if exercise does not induce angina, it is hard to tell whether the person will prone to heart attack or not. Whereas, those who exercise and induced angina does not have high chances of heart attack.

And lastly, the pair plot. The pair plot allows us to see both distribution of single variables and relationships between two variables. The pair plot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column.

The code is shown below.

```
1 sns.pairplot(df,hue='output',palette = ["#de6d6d","#707fbe"])
2 plt.show()
```

Figure 29 Pairplot Code

The seaborn make plotting pair plot easy and it allow us to specify the color palette as well.

The resulted graph is shown below.

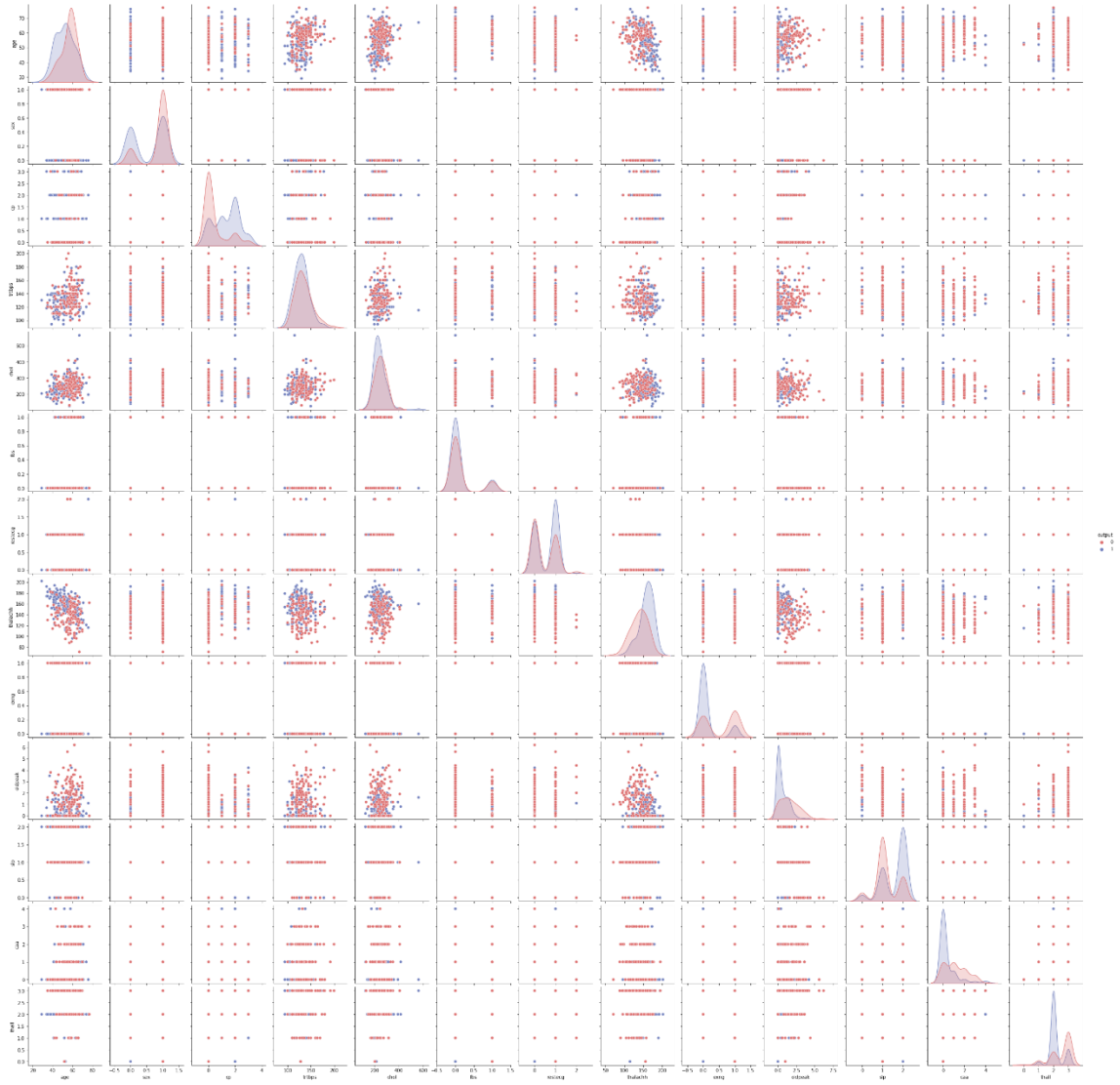


Figure 30 Pairplot

Modeling and Evaluation

In this section, machine learning model will be used to predict the tendency of having heart attack. Multiple models will be used to compare which one give the highest accuracy.

Before implementing machine learning, additional libraries need to be installed as shown below.

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

# Scaling
from sklearn.preprocessing import RobustScaler

# Train Test Split
from sklearn.model_selection import train_test_split

# Models
import torch
import torch.nn as nn
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier

# Metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report, roc_curve

# Cross Validation
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Figure 31 Machine Learning Libraries

Next, making features model ready. By scaling and encoding features as shown below in Fig 32.

```
# creating a copy of df
df1 = df

# define the columns to be encoded and scaled
cat_cols = ['sex','exng','caa','cp','fbs','restecg','slp','thall']
con_cols = ["age","trtbps","chol","thalachh","oldpeak"]

# encoding the categorical columns
df1 = pd.get_dummies(df1, columns = cat_cols, drop_first = True)

# defining the features and target
X = df1.drop(['output'],axis=1)
y = df1[['output']]

# instantiating the scaler
scaler = RobustScaler()

# scaling the continuous featuree
X[con_cols] = scaler.fit_transform(X[con_cols])
print("The first 5 rows of X are")
X.head()
```

```
The first 5 rows of X are
```

	age	trtbps	chol	thalachh	oldpeak	sex_1	exng_1	caa_1	caa_2	caa_3	...	cp_2	cp_3	fbs_1	restecg_1	restecg_2	slp_1	slp_2	thall_1	thall_2	thall_3
0	0.592593	0.75	-0.110236	-0.092308	0.9375	1	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
1	-1.333333	0.00	0.157480	1.046154	1.6875	1	0	0	0	0	...	1	0	0	1	0	0	0	0	1	0
2	-1.037037	0.00	-0.566929	0.584615	0.3750	0	0	0	0	0	...	0	0	0	0	0	0	1	0	1	0
3	0.074074	-0.50	-0.062992	0.769231	0.0000	1	0	0	0	0	...	0	0	0	1	0	0	1	0	1	0
4	0.148148	-0.50	1.795276	0.307692	-0.1250	0	1	0	0	0	...	0	0	0	1	0	0	1	0	1	0

5 rows × 22 columns

Figure 32 Scaling and Encoding Features

After scaling and encoding the features, the data will be split for test and train.

The code below (Fig. 33) splits X and y into random train and test subsets (X_train and X_test for X and y_train and y_test for y). The test_size parameter means that 20% validation data and 80% training data. The random_state parameter is there to sets a seed to the random generator, so that the train-test splits are always deterministic. If there's no seed set, it is different each time.

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 42)
print("The shape of X_train is      ", X_train.shape)
print("The shape of X_test is       ",X_test.shape)
print("The shape of y_train is      ",y_train.shape)
print("The shape of y_test is       ",y_test.shape)
```

```
The shape of X_train is       (242, 22)
The shape of X_test is        (61, 22)
The shape of y_train is       (242, 1)
The shape of y_test is        (61, 1)
```

Figure 33 Split Test and Train

Next, will be implementing various of machine learning model.

The first one is support vector machine or SVM. Support Vector Machine is a linear model for classification and regression problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

```
# instantiating the object and fitting
clf = SVC(kernel='linear', C=1, random_state=42).fit(X_train,y_train)

# predicting the values
y_pred1 = clf.predict(X_test)

# printing the test accuracy
print("The test accuracy score of SVM is ", accuracy_score(y_test, y_pred1)*100, "%")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred1)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The test accuracy score of SVM is  86.88524590163934 %
Confusion Matrix :
 [[26  3]
 [ 5 27]]
```

Figure 34 SVM Model and Confusion Matrix

The code above in Figure 34, uses SVM function to instantiating the object and fitting. Then the variable y_pred1 is used to store the predict values and then printed the accuracy score.

The SVM give us an accuracy score of 86.88%.

After printing the accuracy score, the confusion matrix can be computed too. Confusion matrix is a summary of prediction results on a classification problem. The matrix consists of true positive, false positive, true negative and false negative.

This project also constructs confusion matrix table for a better visualization too.

```
conf_matrix = confusion_matrix(y_test, y_pred1)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

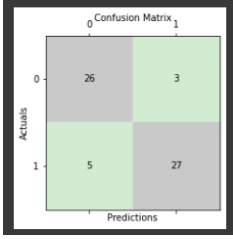


Figure 35 SVM Confusion Matrix Table

The matrix is same as above but just put into table for better visualization. This can be done by using for loop to print the column and row of the table. Then put label and title, lastly plot the table.

Next, we can also do hyperparameter tuning in SVM. It is a process choosing a set of optimal hyperparameters for a model. The process is to choose best C, a hypermeter which is set before the training model and used to control error and Gamma, that is also a hypermeter which used to give curvature weight of the decision boundary. If gamma is too big, it can end up as overfitting.

```
# instantiating the object
svm = SVC()

# setting a grid - not so extensive
parameters = {"C":np.arange(1,10,1),'gamma':[0.00001,0.00005, 0.0001,0.0005,0.001,0.005,0.01,0.05,0.1,0.5,1,5]}

# instantiating the GridSearchCV object
searcher = GridSearchCV(svm, parameters)

# fitting the object
searcher.fit(X_train, y_train)

# the scores
print("The best params are :", searcher.best_params_)
print("The best score is   :", searcher.best_score_)

# predicting the values
y_pred2 = searcher.predict(X_test)

# printing the test accuracy
print("The test accuracy score of SVM after hyper-parameter tuning is ", accuracy_score(y_test, y_pred2)*100, "%")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred2)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The best params are : {'C': 3, 'gamma': 0.1}
The best score is   : 0.8384353741496599
The test accuracy score of SVM after hyper-parameter tuning is  90.1639344262295 %
Confusion Matrix :
 [[26  3]
 [ 3 29]]
```

Figure 36 Hyperparameter Tuning in SVM

The code above will search the best parameters for svm according to the range given in an array. Then used it in a model, and as the result the accuracy score increases to 90%.

```
conf_matrix = confusion_matrix(y_test, y_pred2)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

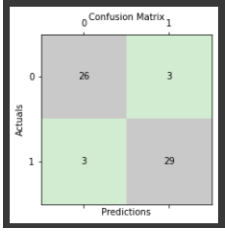


Figure 37 Confusion Matrix of Hyperparameter Tuning in SVM

Same as previous code block for constructing confusion matrix table, the only difference is different variable that carries y_pred for hyperparameter tuning, y_pred2.

For next model, we have Logistic Regression. It's a supervised learning classification algorithm used to predict the probability of a target variable. Unlike linear regression, Logistic regression is used to handle the classification problems and provides a discreet output. Here, the result accuracy score is 90%

The code is shown below.

```
# instantiating the object
logreg = LogisticRegression()

# fitting the object
logreg.fit(X_train, y_train)

# calculating the probabilities
y_pred_proba = logreg.predict_proba(X_test)

# finding the predicted valued
y_pred3 = np.argmax(y_pred_proba,axis=1)

# printing the test accuracy
print("The test accuracy score of Logistic Regression is ", accuracy_score(y_test, y_pred3)*100, "%")
print("")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred3)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The test accuracy score of Logistic Regression is  90.1639344262295 %

Confusion Matrix :
 [[27  2]
 [ 4 28]]
```

Figure 38 Logistic Regression Code

The first step is to instantiating the object, and then fitting the object into the model. After that store the calculated probabilities into a variable that will be used to find the predicted valued. Now we can print the accuracy score of Logistic Regression as well as the confusion matrix.

```
conf_matrix = confusion_matrix(y_test, y_pred3)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

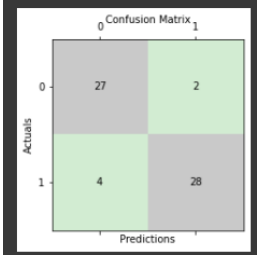


Figure 39 Logistic Regression Matrix Table

Again, same as previous matrix table, everything is the same except y_pred changes to y_pred3 accordingly to Logistic Regression predicted value's variable.

For Logistic Regression, there is a receiver operating characteristic or ROC curve. It is a graph showing the performance of a classification model at all classification thresholds.

```
# calculating the probabilities
y_pred_prob = logreg.predict_proba(X_test)[:,1]

# instantiating the roc_cruve
fpr,tpr,threshols=roc_curve(y_test,y_pred_prob)

# plotting the curve
plt.plot([0,1],[0,1],"k--",'r+')
plt.plot(fpr,tpr,label='Logistic Regression')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Logistric Regression ROC Curve")
plt.show()
```

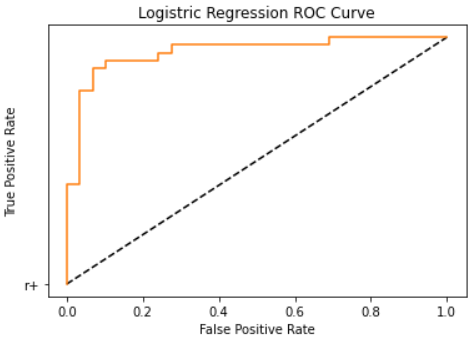


Figure 40 Logistic Regression ROC Curve

Before graphing, we need to calculate the probabilities and then instantiating the ROC Curve. The area under curve (AUC), referred to as index of accuracy(A) or concordance index, is a perfect performance metric for ROC curve. Higher the area under curve, better the prediction power of the model.

In this case, it seems that this model has quite a strong prediction power.

Next, another model called the trees model. Tree-based models use a series of if-then rules to generate predictions. When you get a data point (set of features and values), you use each attribute (a value of a given feature of the data point) to answer a question.

First model is Decision Tree. It uses a flowchart like a tree structure to show the predictions. It starts with a root node and ends with a decision made by leaves. The main advantage of decision tree is it forces the consideration of all possible outcomes of a decision and traces each path to a conclusion.

```
# instantiating the object
dt = DecisionTreeClassifier(random_state = 42)

# fitting the model
dt.fit(X_train, y_train)

# calculating the predictions
y_pred4 = dt.predict(X_test)

# printing the test accuracy
print("The test accuracy score of Decision Tree is ", accuracy_score(y_test, y_pred4)*100, "%")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred4)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The test accuracy score of Decision Tree is  78.68852459016394 %
Confusion Matrix :
 [[25  4]
 [ 9 23]]
```

```
conf_matrix = confusion_matrix(y_test, y_pred4)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

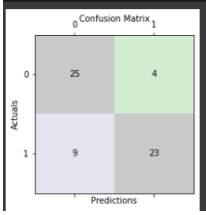


Figure 41 Decision Tree code and Confusion Matrix

Similarly, to previous model first begin by instantiating the object and then fitting the model. After that we can calculate the prediction and print the test accuracy score. The confusion matrix code is the same as other previous model, only change the prediction variable.

The result of test accuracy score is quite low compare to previous model with an accuracy score of 78%

Next, Random Forest Model. The random forest is a classification algorithm consisting of many decisions trees. It creates as many trees on the subset of the data and combines the output of all the trees. In this way it reduces overfitting problem in decision trees. The accuracy score is 78%.

```
# instantiating the object
rf = RandomForestClassifier()

# fitting the model
rf.fit(X_train, y_train)

# calculating the predictions
y_pred5 = dt.predict(X_test)

# printing the test accuracy
print("The test accuracy score of Random Forest is ", accuracy_score(y_test, y_pred5)*100, "%")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred5)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The test accuracy score of Random Forest is  78.68852459016394 %
Confusion Matrix :
 [[25  4]
 [ 9 23]]
```

```
conf_matrix = confusion_matrix(y_test, y_pred5)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

Figure 42 Random Forest Model and Confusion Matrix

Same as other models, instantiating the object, fitting the model, calculating the predictions and then print the accuracy score. Then create a confusion matrix with the prediction value that stored in variable y_pred5. The matrix table is also the same as previously, just change the prediction's variable to be the current model's prediction variable.

Next model, Gradient Boosting Classifier. Gradient boosting classifiers are a collection of machine learning algorithms that combine several weak learning models to generate a powerful predictive model. When doing gradient boosting, decision trees are commonly employed. Gradient boosting, like random forests, is a collection of decision trees. The way trees are formed in the model is one of the two major changes. Random forests construct each tree separately, whereas gradient boosting constructs one tree at a time.

```
# instantiate the classifier
gbt = GradientBoostingClassifier(n_estimators = 300,max_depth=1,subsample=0.8,max_features=0.2,random_state=42)

# fitting the model
gbt.fit(X_train,y_train)

# predicting values
y_pred6 = gbt.predict(X_test)
print("The test accuracy score of Gradient Boosting Classifier is ", accuracy_score(y_test, y_pred6)*100, "%")

#confusion maxtrix
conf = confusion_matrix(y_test, y_pred6)
print ("Confusion Matrix : \n", conf)
print()
print()
```

```
The test accuracy score of Gradient Boosting Classifier is  86.88524590163934 %
Confusion Matrix :
 [[26  3]
 [ 5 27]]
```

```
conf_matrix = confusion_matrix(y_test, y_pred6)

fig, ax = plt.subplots(figsize=(3.5, 3.5))
ax.matshow(conf_matrix, cmap=plt.cm.Accent, alpha=0.35)
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        ax.text(x=j, y=i,s=conf_matrix[i, j], va='center', ha='center', size='medium')

plt.xlabel('Predictions', fontsize=10)
plt.ylabel('Actuals', fontsize=10)
plt.title('Confusion Matrix', fontsize=10)
plt.show()
```

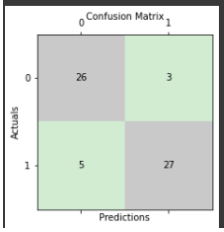


Figure 43 Gradient Boosting Classifier and Confusion Matrix

The coding process is the same as previous model. Starting by instantiating the classier, fitting the model and then predict the values to print the accuracy score. After that construct the confusion matrix based on the predicted value. The accuracy score of this model is 86%.

And lastly, let's compare which model give the highest accuracy score.

```
model = ['SVM', 'Hyper', 'LogRegress', 'DeciTree', 'RandFor', 'GradBoost']
results = [86, 90, 90, 78, 78, 86]
fig = plt.figure(figsize = (9.5, 5))
ax = sns.barplot(model, results, palette ='plasma');
ax.set_xlabel('Model', fontsize=12)
ax.set_ylabel('Accuracy', fontsize=12)
plt.show()
```

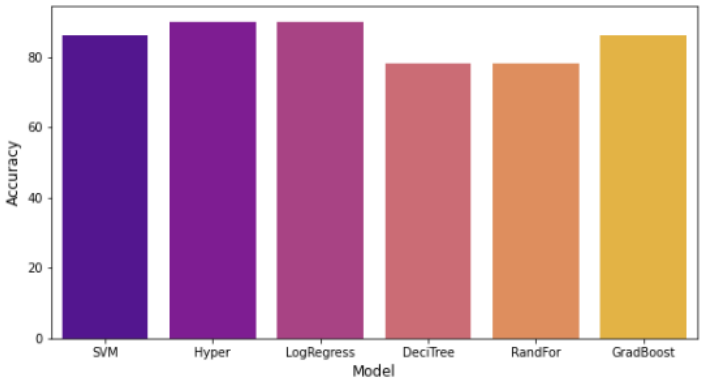


Figure 44 Accuracy Score Comparison

For this dataset, after tuning the hyperparameter in SVM it gives a higher accuracy at 90%. Logistic Regression also gives a high accuracy at 90% too. Whereas the tree models and gradient booster classifier gives a lower prediction accuracy.

Executive Summary

This report had observed an insight of the heart attack dataset. To find the trends and pattern among the patient who likely and not likely going to experience heart attack. This project able to conduct data previewing and cleaning data before performing exploratory data analysis with data visualization method by using graphs. During this process, we able to interpret the graphs and understand the meaning behind the trends of heart attack tendency among patients in this dataset. There is various type of graphs constructed in this process. The main three graphs are categorical features, continuous features and target variable. After that, we try to find the relationship among them to the heart attack probability. With the help of correlation matrix, we able to see which attributes has a positive or negative correlation to each other. And graph the relationship of each feature to the output for better visualization and to discover the pattern of those who are prone to heart attack and those who are not.

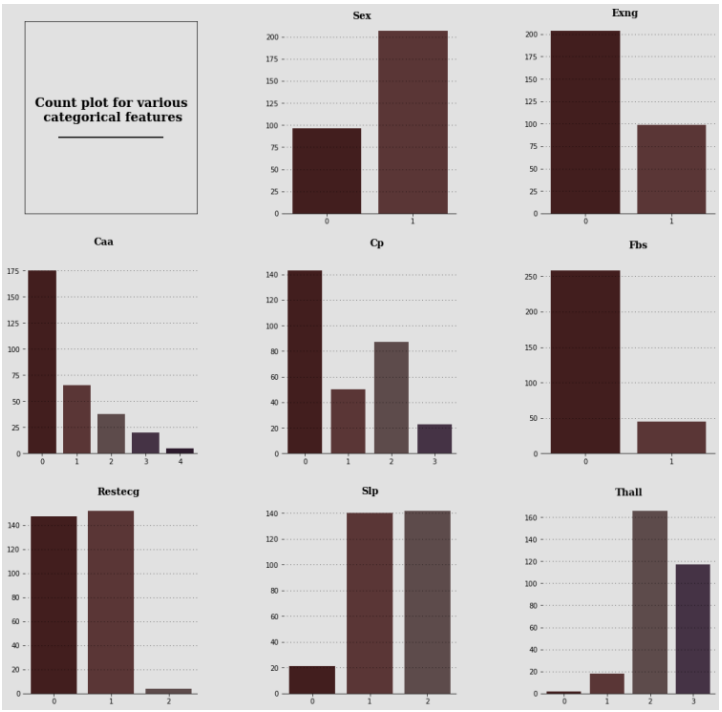
In conclusion of observation from the graph we can interpret information about symptoms related to those who most likely to experience heart attack. For example, Age 40 and 50 has a high chance or quite common to experience heart attack. With resting blood pressure around 120 to 140 mmHG and cholesterol level is around 200 milligrams per deciliter. Maximum heart rate of 160 to 175 bpm, Oldpeak at 0 are more likely to have a heart attack. Gender could be a factor too, females are more prone to a heart attack than males. And if the patient experience chest pain, they are likely to experience heart attack. But if number of vessels is greater than 0 the chance decreases. And Those who have thallium stress test result type 2 are most likely prone to a heart attack. And lastly, if exercise does not induce angina, it is hard to tell whether the person will prone to heart attack or not.

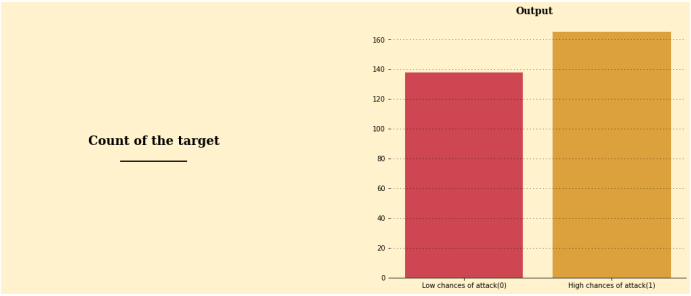
After observing and understanding the trends, we proceed to modeling and evaluating process. This project uses several of model to find highest accuracy among the models. There are support vector machine and the hyperparameter tuning, logistic regression, tree models (decision tree and random forest), and lastly, gradient boosting classifier. Among these 6 models, the tree models give the lowest accuracy of 78%. On the other hand, the hyperparameter tuning in SVM and Logistic Regression Model able to give the highest accuracy at 90%.

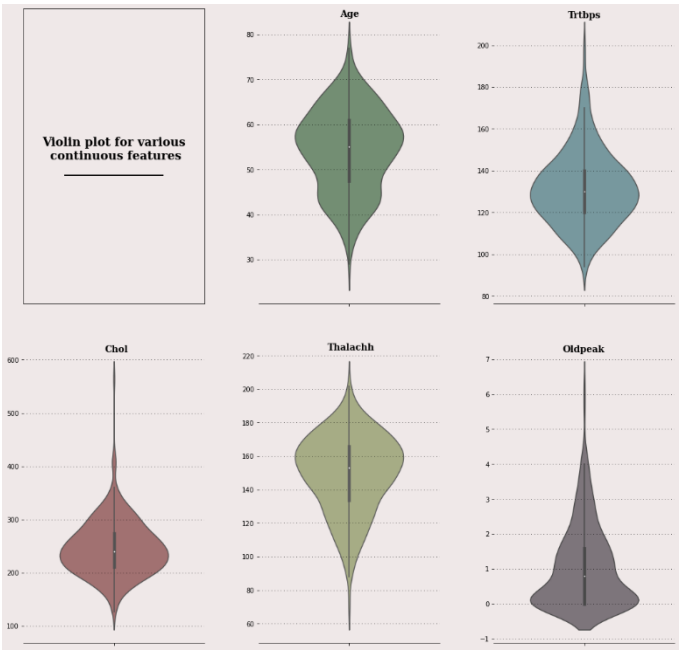
Appendix

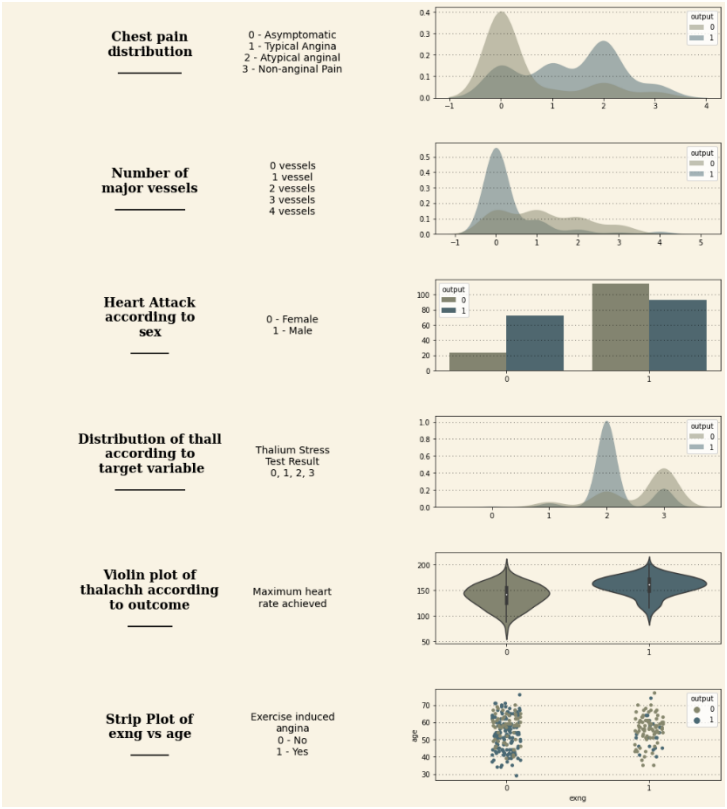
https://www.kaggle.com/rashikrahmanpritom/heart-attack-analysis-prediction-dataset

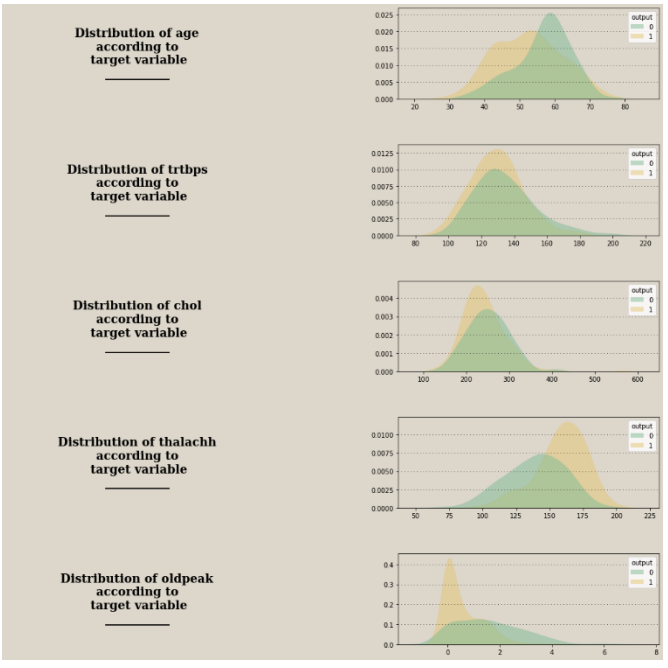
List of Graphs:

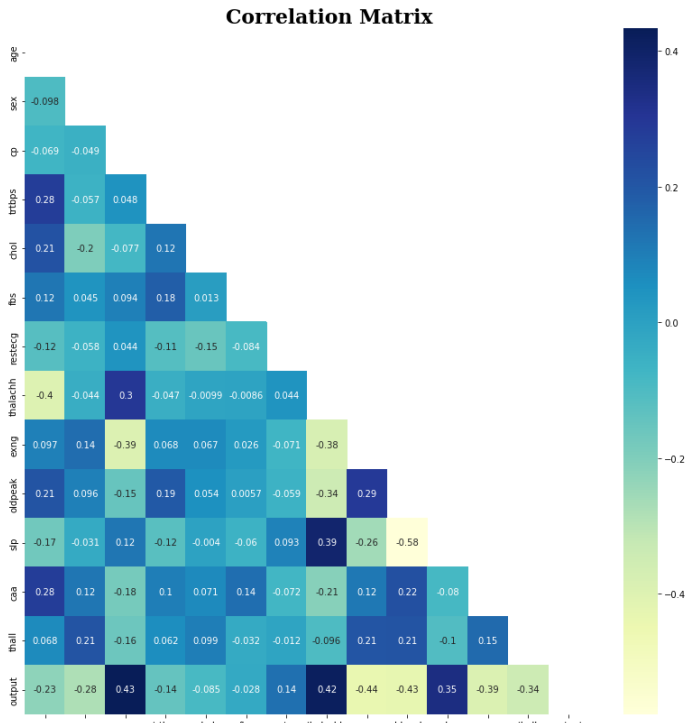











Correlation Matrix
age
sex
cp
trtbps
chol
fbs
restecg
thalachh
exng
oldpeak
slp
caa
thall
output
0.4
0.2
0.0
−0.2
−0.4