1.1 Data description

The main Data file: (Given by the kaggle)

Age : Age of the patient

Sex : Sex of the patient

exang: exercise induced angina (1 = yes; 0 = no)

ca: number of major vessels (0-3)

cp : Chest Pain type chest pain type

- *Value 1: typical angina*
- *Value 2: atypical angina*
- *Value 3: non-anginal pain*
- *Value 4: asymptomatic*

trtbps : resting blood pressure (in mm Hg)

chol : cholestoral in mg/dl fetched via BMI sensor

fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

rest_ecg : resting electrocardiographic results

- *Value 0: normal*
- *Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)*
- *Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria*

thalach : maximum heart rate achieved

target : 0= less chance of heart attack 1= more chance of heart attack

1.2 Packages used

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import missingno
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

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

- *Seaborn, Matplotlib, Missingno, Plotly for ploting graph*
- *Mlxtend for association rule mining*
- *Sklearn for clustering and classification machine learning*

1.3 Data Import

There are two data set we used, first is the original data set given:

```
data.head()
```

✓ 0.7s

	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

Another one is the data file assigned the meaning of each categorical data to original meaning by spreadsheet, eg. *Sex = 1: Boys* , *Cp = 1 : atypical angina etc.*

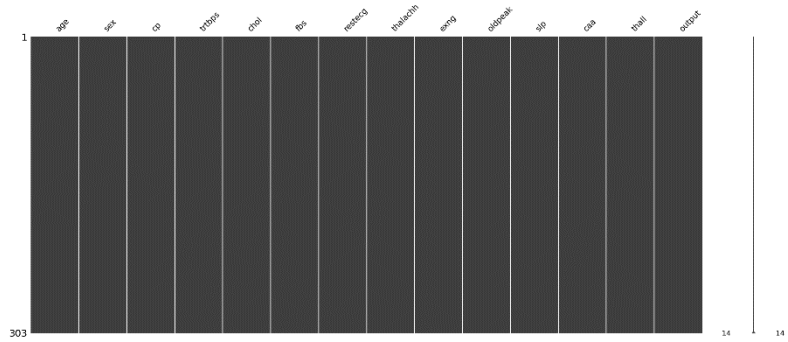
This data prepares for later association rule mining.

```
ap.head()
```

✓ 0.3s

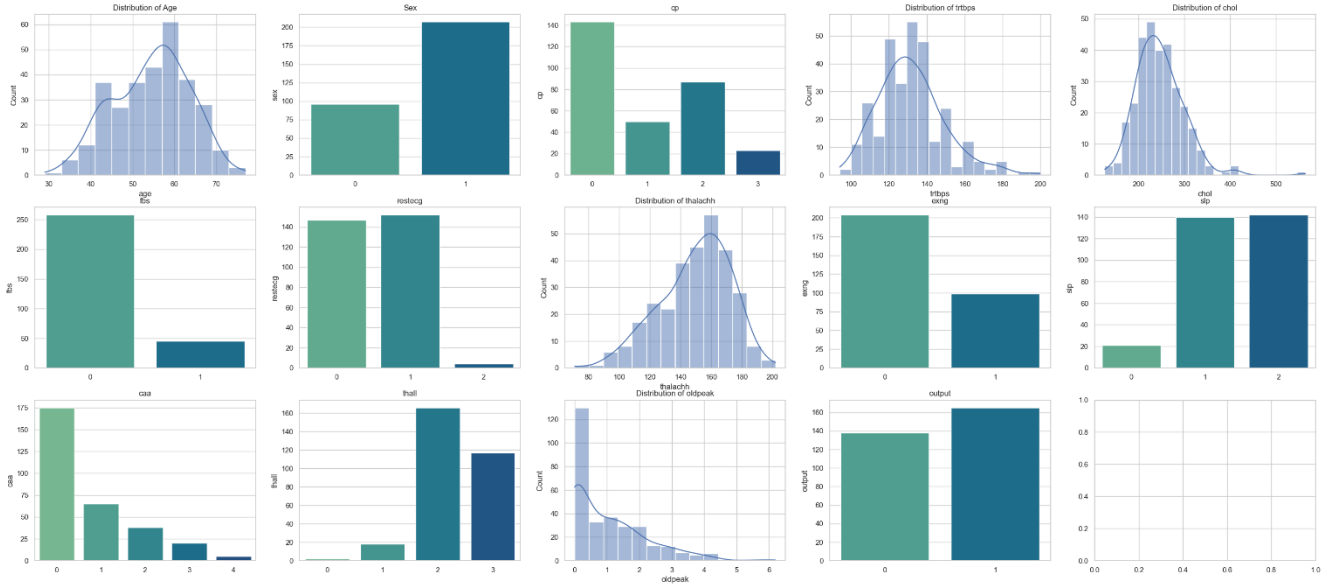
	Sex	Cp	Fbs	Restecg	Exng	Caa	Output
0	Boys	asymptomatic	fasting blood sugar	normal	No exercise induced angina	0 major vessels	more chance of heart attack
1	Boys	non-anginal pain	No fasting blood sugar	having ST-T wave abnormality	No exercise induced angina	0 major vessels	more chance of heart attack
2	Girls	atypical angina	No fasting blood sugar	normal	No exercise induced angina	0 major vessels	more chance of heart attack
3	Boys	atypical angina	No fasting blood sugar	having ST-T wave abnormality	No exercise induced angina	0 major vessels	more chance of heart attack
4	Girls	typical angina	No fasting blood sugar	having ST-T wave abnormality	exercise induced angina	0 major vessels	more chance of heart attack

2.1 Exploratory Data Analysis (EDA)



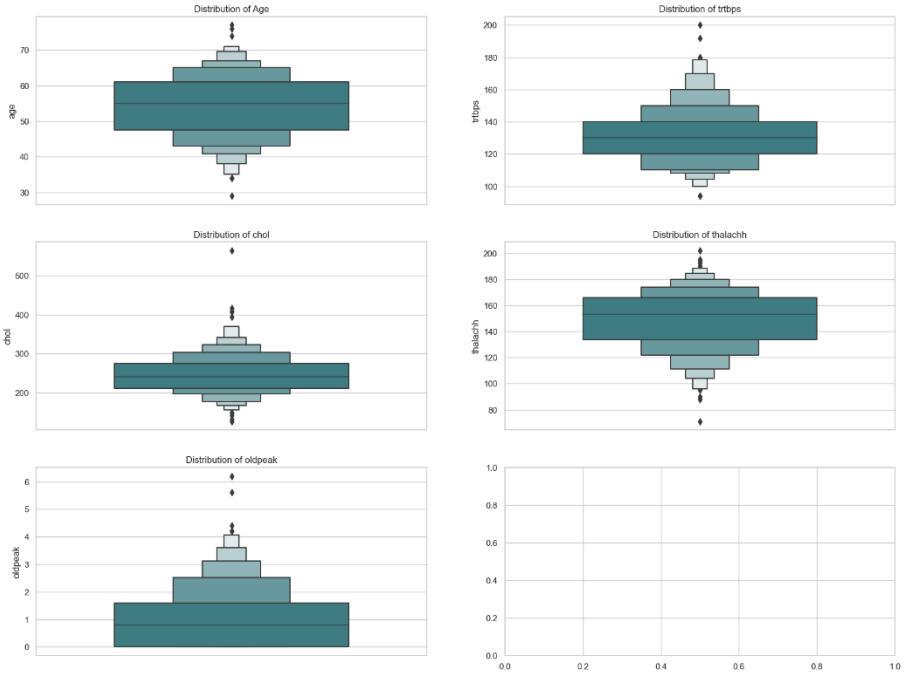

```
data.describe()
✓ 0.6s
```

	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
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

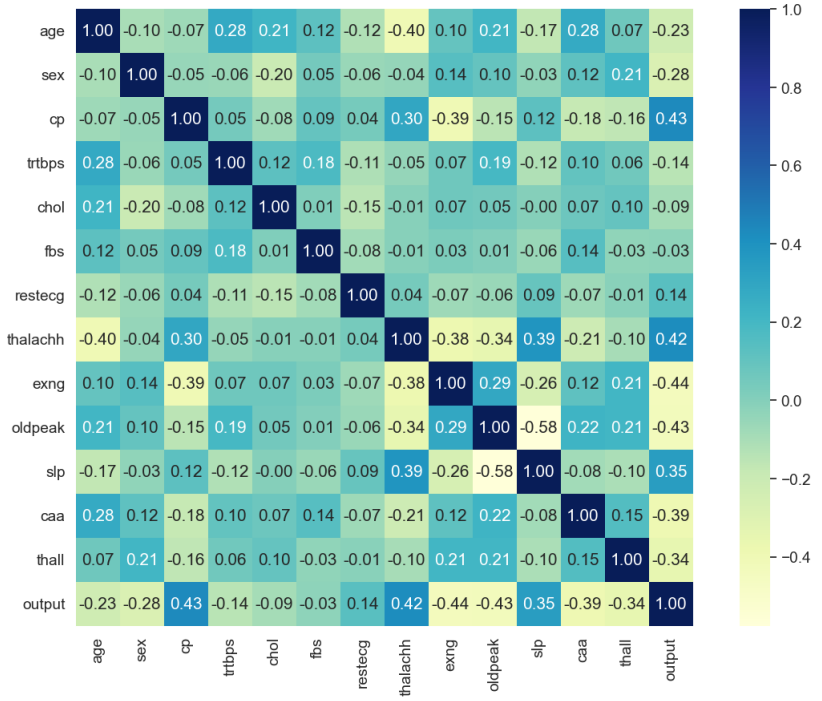


- Using missingno plot, we can see that the data is no missing value.
- Briefly visualize all the data (excluding target) we have:
  - Age in our data lightly normal distributed
  - Most of our data belonging with sex 1 or having typical angina *(cp=0)*
  - Resting blood pressure*(Trtbps)* and cholestoral *(chol)* are right skewed
  - Most of the people having rested electrocardiographic results(*restecg*)
  - Maximum heart rate (*thalachh*) achieved are left skewed
  - More people exercise induced angina(*exang)*
  - Less people having *slp = 0*
  - Most of the people having 0 number of major vessels*(caa)*
  - More people having *thall = 2 or 3*
  - *Oldpeak* data right skewed

2.1 Exploratory Data Analysis (EDA).cont



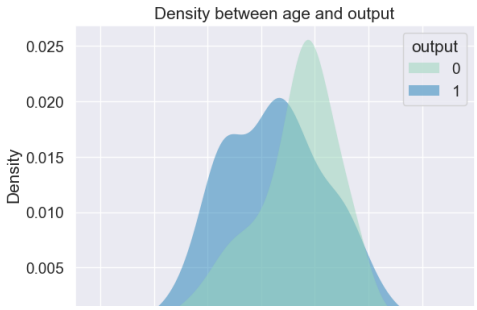
- All the continuous data*(age, trtbps, chol, thalachh, oldpeak)* having outliers



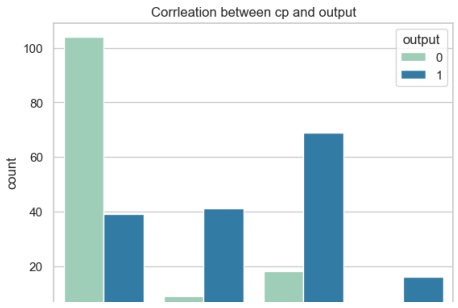
For the target output:

- *Cp, thalachh and slp* are most positive correlated with the target outcome
- *Exng and oldpeak are most negative correlated with the targe outcome*

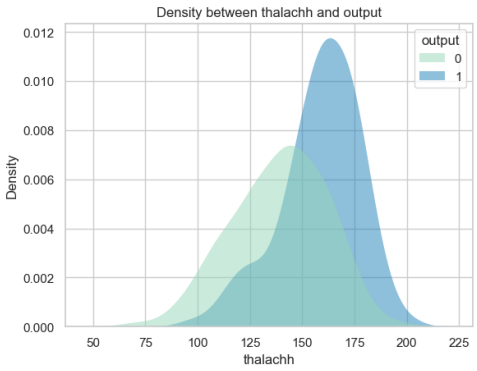
2.1 Exploratory Data Analysis (EDA).cont



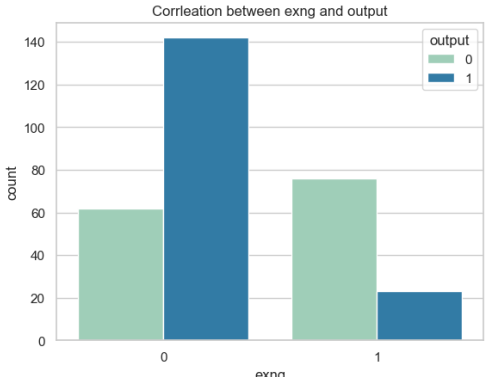
Age is not strong correlated with target



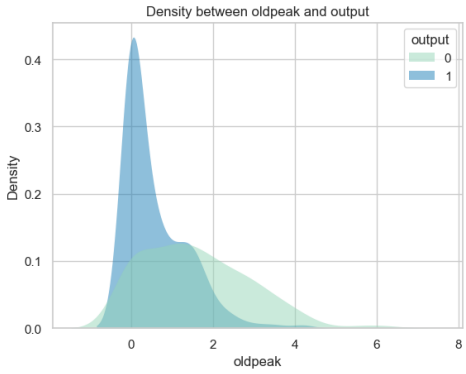
Having *cp* = 0 : less opportunity of target



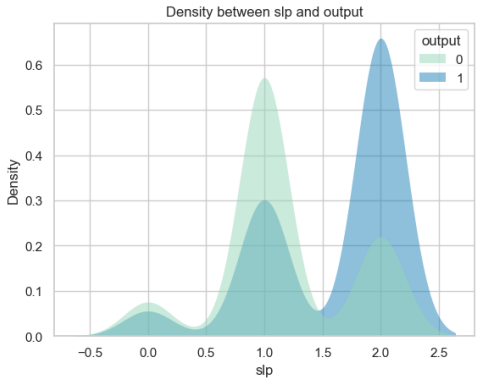
Having higher *thalachh* : less opportunity of target



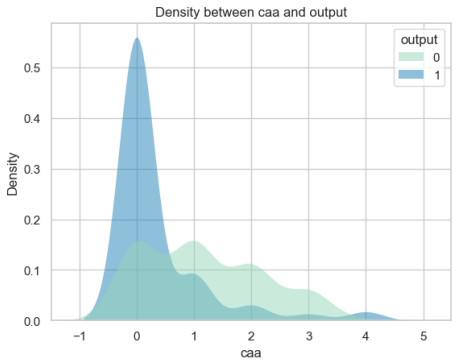
Having *exng* = 1 : higher opportunity of target



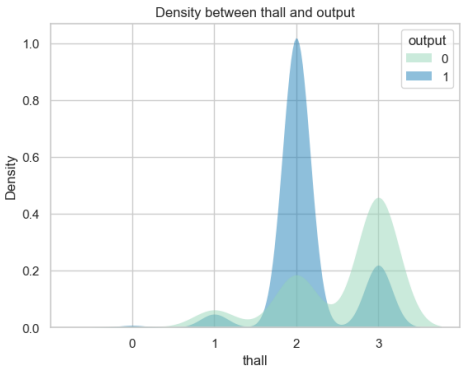
Having *oldpeak* = 0 : higher opportunity of target



Having *slp* = 2 : higher opportunity of target



Having *caa* = 0 : higher opportunity of target



Having *thall* = 2 : higher opportunity of target

2.2 Exploratory Data Analysis (EDA) Conclusion:

1. All the continuous data*(age, trtbps, chol, thalachh, oldpeak)* having outliers
2. It is evident that younger people having higher chances of heart attack according to the distribution plot of age with respect to output.
3. If Chest Pain type is typical angina *(cp=0)* not likely having heart attack.
4. People having higher maximum heart rate *(thalachh*) more likely having heart attack.
5. People exercise not induced angina more likely having heart attack.
6. People having 0 number of major vessels more likely having heart attack.

3. Association rule mining

Change dataframe for Apriori Algorithm by transaction encoder:

```
ap_df = ap.T.reset_index(drop=True).T
ap_df = ap_df.to_numpy()
te = TransactionEncoder()
te_ary = te.fit(ap_df).transform(ap_df)
apap_df = pd.DataFrame(te_ary, columns=te.columns_)
```

Setting the minimum support belong = 20% :

```
frequent_itemsets = apriori(apap_df, min_support=0.2, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets[(frequent_itemsets['length']==2)&(frequent_itemsets['support']>=0.4)]
```

✓ 0.8s

	support	itemsets	length
15	0.432343	(0 major vessels, No exercise induced angina)	2
16	0.511551	(0 major vessels, No fasting blood sugar)	2
18	0.429043	(0 major vessels, more chance of heart attack)	2
21	0.429043	(No exercise induced angina, Boys)	2
22	0.574257	(Boys, No fasting blood sugar)	2
32	0.577558	(No exercise induced angina, No fasting blood ...	2
35	0.468647	(No exercise induced angina, more chance of he...	2
40	0.438944	(No fasting blood sugar, having ST-T wave abno...	2
42	0.468647	(No fasting blood sugar, more chance of heart ...	2
45	0.412541	(No fasting blood sugar, typical angina)	2

```
frequent_itemsets = apriori(apap_df, min_support=0.2, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets[(frequent_itemsets['length']==2)&(frequent_itemsets['support']>=0.5)]
```

✓ 0.8s

	support	itemsets	length
16	0.511551	(0 major vessels, No fasting blood sugar)	2
22	0.574257	(Boys, No fasting blood sugar)	2
32	0.577558	(No exercise induced angina, No fasting blood ...	2

```
frequent_itemsets = apriori(apap_df, min_support=0.2, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x: len(x))
frequent_itemsets[(frequent_itemsets['length']==2)&(frequent_itemsets['support']>=0.55)]
```

✓ 0.9s

	support	itemsets	length
22	0.574257	(Boys, No fasting blood sugar)	2
32	0.577558	(No exercise induced angina, No fasting blood ...	2

Frequent item set 1:

Exercise induced angina(exng=0) -> No fasting blood sugar(fbs=0)

Support = 0.577558

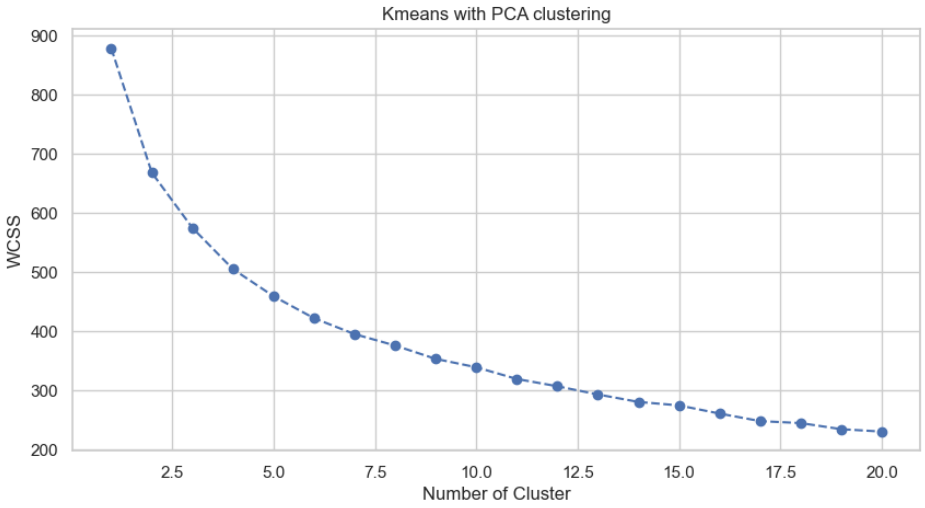
Frequent item set 2:

Boys (sex=1) -> No fasting blood sugar(fbs=0)

Support = 0.574257

4.1 Clustering K-means with PCA

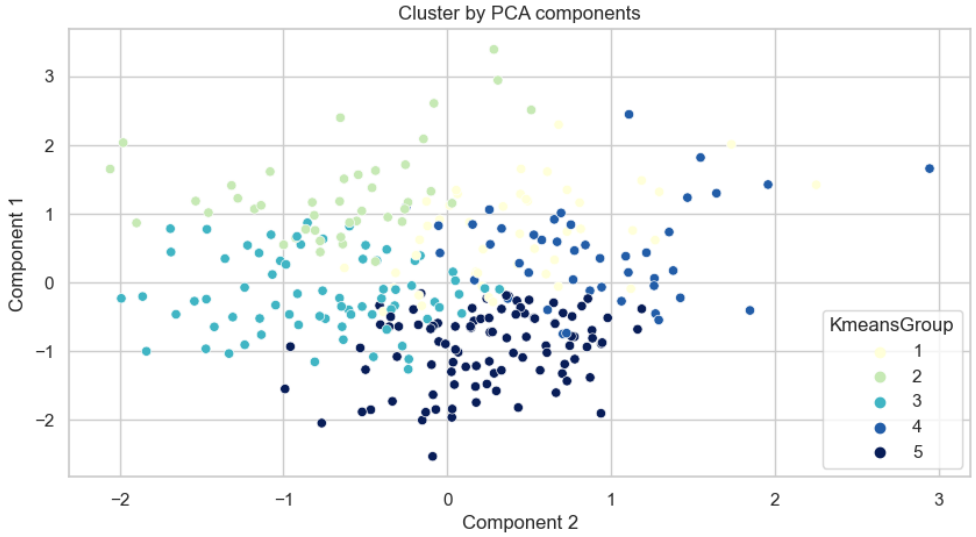
```
kmeans_data = data[['age','trtbps','chol','thalachh','oldpeak']]
scaler_st = RobustScaler()
X_std = scaler_st.fit_transform(kmeans_data)
pca = PCA()
pca.fit(X_std)
scores_pca=pca.transform(X_std)
wcss=[]
for i in range(1,21):
    kmeans_pca = KMeans(n_clusters= i, init='k-means++',random_state=42)
    kmeans_pca.fit(scores_pca)
    wcss.append(kmeans_pca.inertia_)
plt.figure(figsize=(10,5))
plt.plot(range(1,21),wcss,marker='o',linestyle ='--')
plt.xlabel('Number of Cluster')
plt.ylabel('WCSS')
plt.title('Kmeans with PCA clustering')
plt.show()
```



- K-means only work more well with continuous data, so I only select those continuous data from the data set *(age, trtbps, chol, thalachh, oldpeal).*
- Our continuous data not likely normally distributed, some of them are left and right skewed, so I use Robust scaler instead of Minmax or Standard scaler.
- Principal component analysis (PCA) is also needed before compute the K-means, due to our data having more than 2 columns.
  After PCA we can use the Elbow method. Elbow method calculated the sum of squared distance between different cluster number. If the line chart resembles an arm, then the “elbow point” indicate that the underlying model best fits at that point. In the graph right hand corner, 5 cluster is quite suitable for our data.

4.1 Clustering K-means with PCA. cont

```
kmeans_pca = KMeans(n_clusters=5, init='k-means++',random_state=42)
kmeans_pca.fit(scores_pca)
pcadf = pd.DataFrame(scores_pca)
df_x_pca_kmeans = pd.concat([data.reset_index(drop=True),pcadf],axis=1)
df_x_pca_kmeans.columns.values[-5:]=['Component 1','Component 2','Component 3','Component 4','Component 5']
df_x_pca_kmeans['Segment K-means PCA'] = kmeans_pca.labels_
df_x_pca_kmeans['KmeansGroup'] = df_x_pca_kmeans['Segment K-means PCA']
df_x_pca_kmeans['KmeansGroup'] = df_x_pca_kmeans['KmeansGroup'] +1
x_axis = df_x_pca_kmeans['Component 2']
y_axis = df_x_pca_kmeans['Component 1']
plt.figure(figsize=(10,5))
hue_order=[1,2,3,4,5]
sns.scatterplot(x=x_axis,y=y_axis, hue = df_x_pca_kmeans['KmeansGroup'],hue_order=hue_order,palette='YlGnBu')
plt.title('Cluster by PCA components')
plt.show()
```



- Then we use 5 cluster to do the K-means,
  and plot in 2D graph by PCA. We can see that it is slightly well distribute 5 groups.

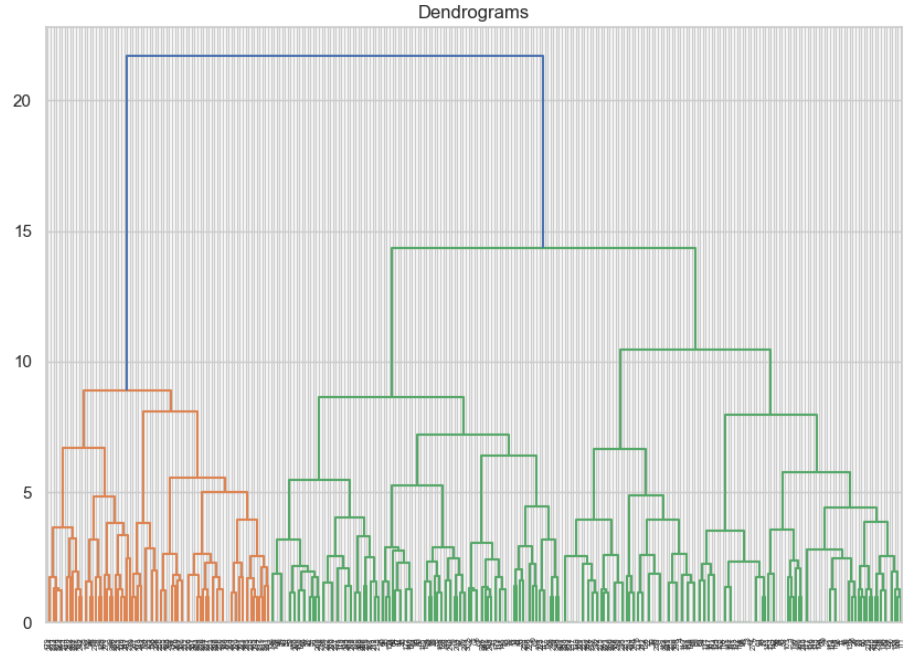
Here are these 5 groups mean with different continuous data:

```
kmeans_data['KmeansGroup'] = df_x_pca_kmeans['KmeansGroup']
kmeans_data['output'] = data['output']
kmeans_data.groupby(['KmeansGroup']).mean()
```

KmeansGroup	age	trtbps	chol	thalachh	oldpeak	output
1	59.574468	153.340426	241.468085	154.744681	0.565957	0.595745
2	61.046512	146.279070	243.209302	131.790698	2.888372	0.139535
3	57.732394	118.802817	226.788732	130.859155	1.183099	0.422535
4	56.613636	131.022727	327.886364	153.931818	0.984091	0.522727
5	45.489796	124.336735	227.367347	166.724490	0.376531	0.795918

4.2 Hierarchical clustering

```
temp1 = pd.get_dummies(data['cp'])
temp1.columns = ['typical angina', 'atypical angina', 'non-anginal pain','asymptomatic']
temp2 = pd.get_dummies(data['restecg'])
temp2.columns = ['normal', 'having ST-T wave abnormality','showing probable or definite left ventricular hypertrophy by Estes']
data = pd.concat([data,temp1,temp2],axis=1)
data = data.drop(columns=['cp','restecg'],axis=1)
data = data[['age', 'sex', 'trtbps', 'chol', 'fbs', 'thalachh', 'exng', 'oldpeak','slp', 'caa', 'thall','typical angina', 'atypical angina','non-anginal pain', 'asymptomatic', 'norm
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
hier_data = data[['sex','fbs','exng','slp','caa','thall','typical angina', 'atypical angina', 'non-anginal pain','asymptomatic','normal', 'having ST-T wave abnormality','showing pro
dend = shc.dendrogram(shc.linkage(hier_data, method='ward'))
```



- Before we do the clustering, we can one hot code those categorical data, it will be more efficient for hierarchical clustering and later prediction part.
  (Change dummy variables for *cp* and *restecg*)
- Here used ward to be the linkage method or called minimal increase of sum-of-squares (MISSQ). After observation, 3 cluster will be well to cluster the data.

Here are these 3 groups mode with different categorical data:

```
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
hier_data['HierarchyGroup'] = cluster.fit_predict(hier_data)
hier_data['HierarchyGroup'] = hier_data['HierarchyGroup'] + 1
hier_data['output'] = data['output']
hier_data.groupby(['HierarchyGroup']).agg(pd.Series.mode)
```

✓ 0.9s Python

HierarchyGroup	sex	fbs	exng	slp	caa	thall	typical angina	atypical angina	non-anginal pain	asymptomatic	normal	having ST-T wave abnormality	showing probable or definite left ventricular hypertrophy by Estes	output
1	1	0	0	2	0	2	0	0	0	0	0	1	0	1
2	1	0	1	1	2	3	1	0	0	0	1	0	0	0
3	1	0	0	1	0	2	0	0	0	0	1	0	0	1

5. Data Preprocessing

```
X=data.iloc[:,:-1]
y=data['output']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42)

scaler = MinMaxScaler()
X_train = pd.DataFrame(
    scaler.fit_transform(X_train),
    columns = X_train.columns
)

X_test = pd.DataFrame(
    scaler.transform(X_test),
    columns = X_test.columns
)
```

- We already one hot code those categorical data in Hierarchical clustering part.
- We need to split our data into training set and testing set for the prediction.
- After we split the data, this time we use Minmax scaler to normalize our data, it will not that fluctuate comparing with Standard Scaler or Robust scaler for the not normally distributed data.

6.0 Prediction Background

Reason using logistic regression:

Logistic regression can handle continuous data well.
Easier to implement and very efficient to train
Less inclined to over-fitting

Reason using support vector machine:

SVM works well with a clear margin of separation between classes.
More effective in high dimensional data

Reason using gradient boosting:

Gradient boosting often works great with categorical and numerical values.
Provides predictive accuracy comparing with other models
Tuning parameter makes it very flexible

- We use grid search and “f1” score which is used to determine the binary classification score to find the best parameter of different models
- Cross Validation 5 times

6.1 Logistic Regression

```
logr = LogisticRegression()
param_grid_logr = [
    {'penalty':['l1','l2','elasticnet','none'],
    'dual':[True,False],
    'fit_intercept':[True,False],
    'C':[0.001, 0.01, 0.05, 0.1],
    'class_weight':[dict,'balanced'],
    'solver':['lbfgs','newton-cg','liblinear','sag','saga'],
    'max_iter':[100,500,1000],
    'multi_class':['auto','ovr','multinomial'],
    'warm_start':[True,False]
    }
]
param_logr = GridSearchCV(logr, param_grid = param_grid_logr, cv = 5, verbose=True,scoring="f1")
best_param_logr = param_logr.fit(X_train,y_train)
best_param_logr.best_params_
```

```
{'C': 0.1,
 'class_weight': dict,
 'dual': False,
 'fit_intercept': True,
 'max_iter': 100,
 'multi_class': 'auto',
 'penalty': 'l2',
 'solver': 'lbfgs',
 'warm_start': True}
```

```
logr = LogisticRegression(C = 0.1 , class_weight = 'dict', dual = False, fit_intercept = True, max_iter = 100
logr.fit(X_train,y_train)
logr_Prediction = logr.predict(X_test)
```

6.2 Support Vector Machine

```
svm = SVC()
param_grid_svm = [
    {
    'kernel':['linear', 'rbf', 'sigmoid'],
    'gamma':['scale', 'auto'],
    'class_weight':['dict' , 'balanced'],
    'decision_function_shape':['ovo', 'ovr'],
    'probability':[True,False],
    'shrinking':[True,False],
    'verbose':[True,False]
    }
]
param_svm = GridSearchCV(svm, param_grid = param_grid_svm, cv = 5, verbose=True,scoring="f1")
best_param_svm = param_svm.fit(X_train,y_train)
best_param_svm.best_params_
```

```
{'class_weight': 'balanced',
 'decision_function_shape': 'ovo',
 'gamma': 'scale',
 'kernel': 'rbf',
 'probability': True,
 'shrinking': True,
 'verbose': True}
```

```
svm = SVC(kernel='rbf',class_weight='balanced',decision_function_shape='ovo',gamma='scale',probabi
svm.fit(X_train, y_train)
svm_Prediction = svm.predict(X_test)
```

6.3 Gradient Boosting

```
gb = GradientBoostingClassifier()
param_grid_gb = [
    {
    'loss':['log_loss', 'deviance', 'exponential'],
    'learning_rate':[0.1,0.01,0.001],
    'n_estimators':[100,200,400,600,800],
    'subsample':[0.0,0.5,1],
    'criterion':['friedman_mse', 'squared_error', 'mse'],
    'max_features':['auto', 'sqrt', 'log2'],
    'init':['estimator','zero'],

    }
]
param_gb = GridSearchCV(gb, param_grid = param_grid_gb, cv = 5, verbose=True,scoring="f1")
best_param_gb = param_gb.fit(X_train,y_train)
best_param_gb.best_params_
```

```
{'criterion': 'squared_error',
 'init': 'zero',
 'learning_rate': 0.01,
 'loss': 'deviance',
 'max_features': 'sqrt',
 'n_estimators': 600,
 'subsample': 0.5}
```

```
gb_Model = GradientBoostingClassifier(criterion = 'squared_error',init='zero',learning_rate=0.0
gb_Model.fit(X_train, y_train)
gb_Prediction = gb_Model.predict(X_test)
```

6.4 Conclusion

```
print("The accuracy score of logisitc regression is:",accuracy_score(y_test,logr_Prediction))
print("The accuracy score of gradient boosting is:",accuracy_score(y_test,gb_Prediction))
print("The accuracy score of support vector machine is:",accuracy_score(y_test,svm_Prediction))
```

```
The accuracy score of logisitc regression is: 0.7741935483870968
The accuracy score of gradient boosting is: 0.8064516129032258
The accuracy score of support vector machine is: 0.8064516129032258
```

Support vector machine and Gradient boosting also work well to predict the outcome.