In Summary;

This project focuses on predicting the likelihood of a heart attack based on individual demographic and clinical attributes using Python. I began by importing the necessary libraries and dataset, followed by exploring the data for initial insights. A categorical feature analysis was conducted for attributes such as sex, chest pain type (cp), fasting blood sugar (fbs), resting electrocardiographic results (restecg), exercise-induced angina (exng), slope of the peak exercise ST segment (slp), number of major vessels (caa), and thalassemia (thall), along with the output variable indicating the occurrence of a heart attack. Simultaneously, I performed numeric feature analysis and studied the dataset's correlations, identifying key relationships between features and the target variable. Standardization was applied to numerical data to ensure consistency and improve model performance.

To better understand the data, I created visualizations such as box plots, swarm plots, and categorical plots, which highlighted trends and revealed potential outliers. Outlier detection allowed me to refine the dataset for cleaner analysis. The modeling phase included encoding categorical columns, scaling numerical features, and splitting the dataset into training and testing subsets. I employed logistic regression for predictive modeling, evaluated its performance using the ROC Curve, and applied hyperparameter tuning to optimize the model while preventing overfitting or memorization. This comprehensive approach combined exploratory analysis, visualization, and machine learning techniques to develop a robust predictive model for heart attack risk assessment.

* Import the necessary libraries.

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, roc_curve


import warnings

warnings.filterwarnings('ignore')
```

* Import the data.

```
data =
pd.read_csv('C:\\Users\\Asus\\OneDrive\\Masaüstü\\website\\python\\PREDICTION
\\hearth_attack_prediction\\heart.csv')
```

## Data Content

* Data Content¶

* 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

* alue 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

## Read and Analyse the Data

* We dont know gender 0 if it is male or female.

```
data.head()
```

* Describe the data.

* If fbs > 120 = 1, if <120 = 0, it is about blood level whit emtpy stomacth.

* Mean hbs close to 0, rather than 1, it means that it is < 120.

* Output mean closer to 1, close to heart attack.

```
data.describe()
```

* Info about the data.

* No missing values.

```
data.info()
```

* You can not educate you value with missing values.

```
data.isnull().sum()
```

* People by sex.

```
data['sex'].value_counts().shape[0]
```

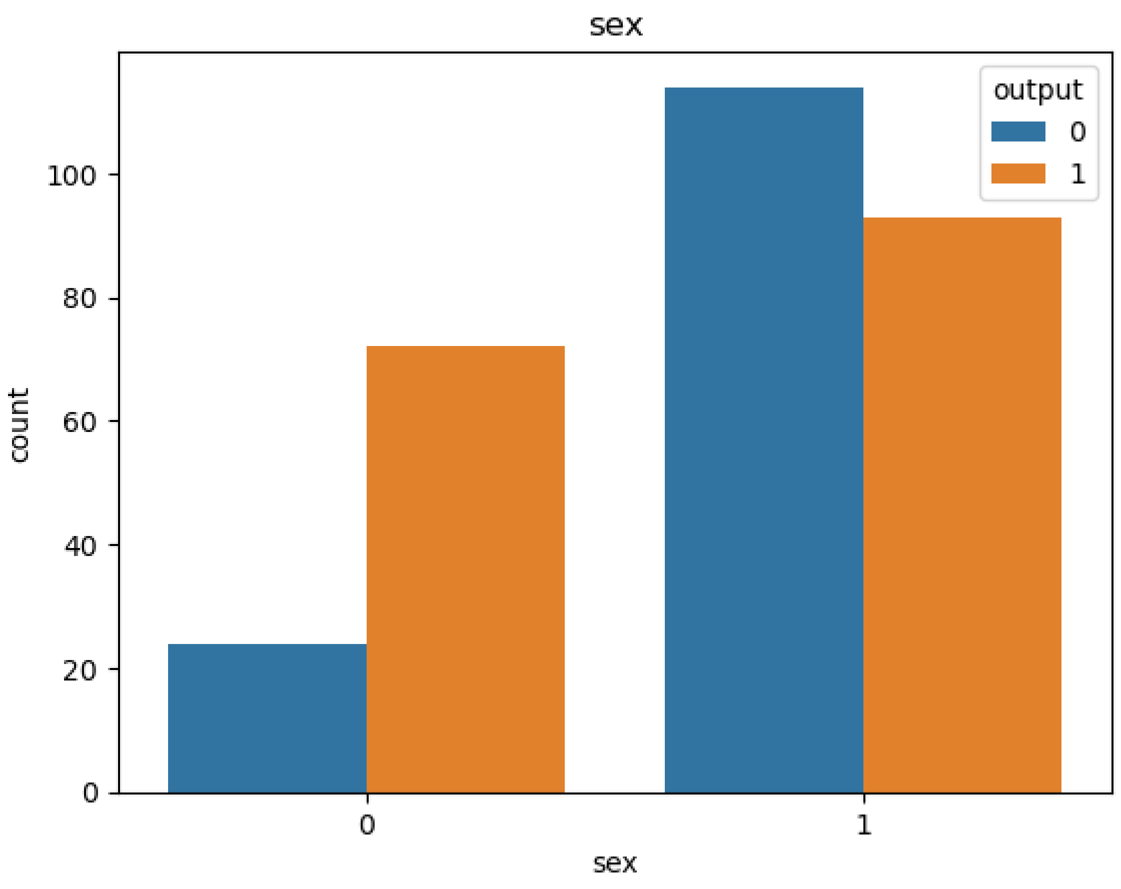
## Unique Value Analysis

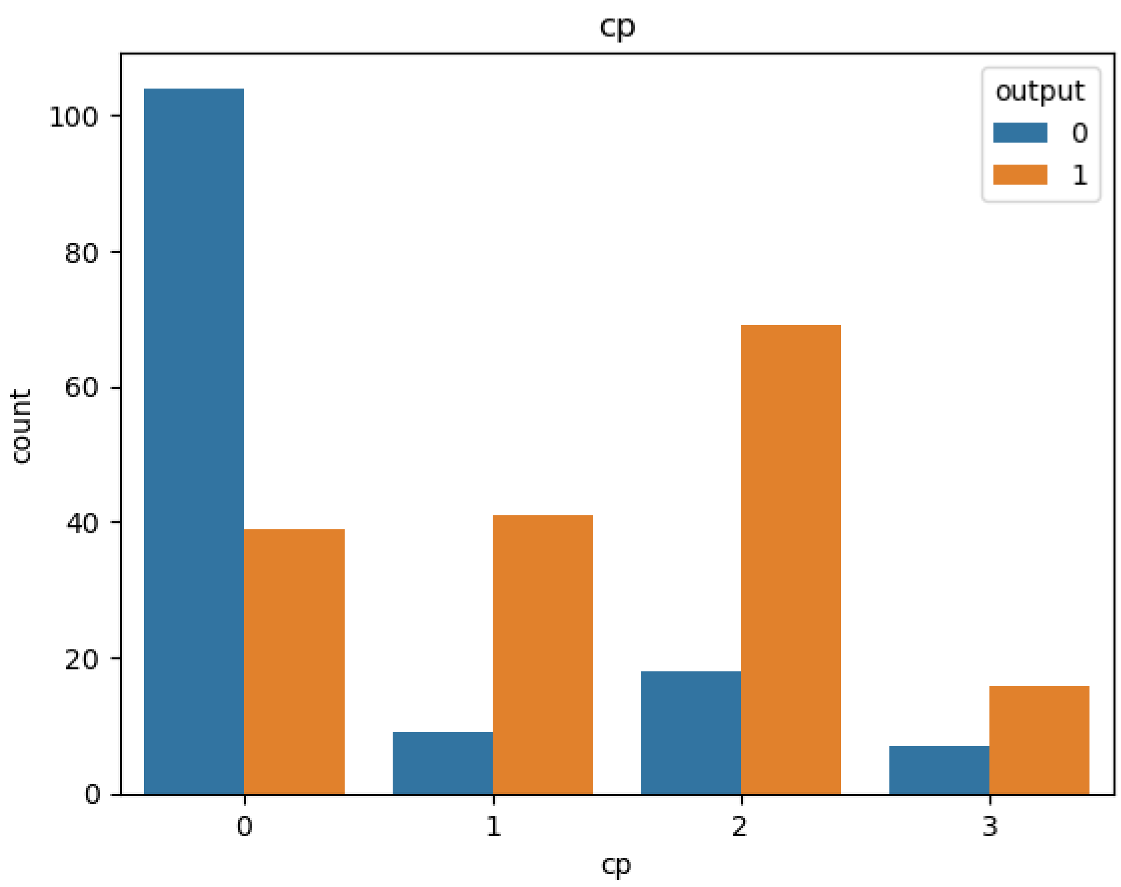
* Cathegorical and numeric values are different for prediction.

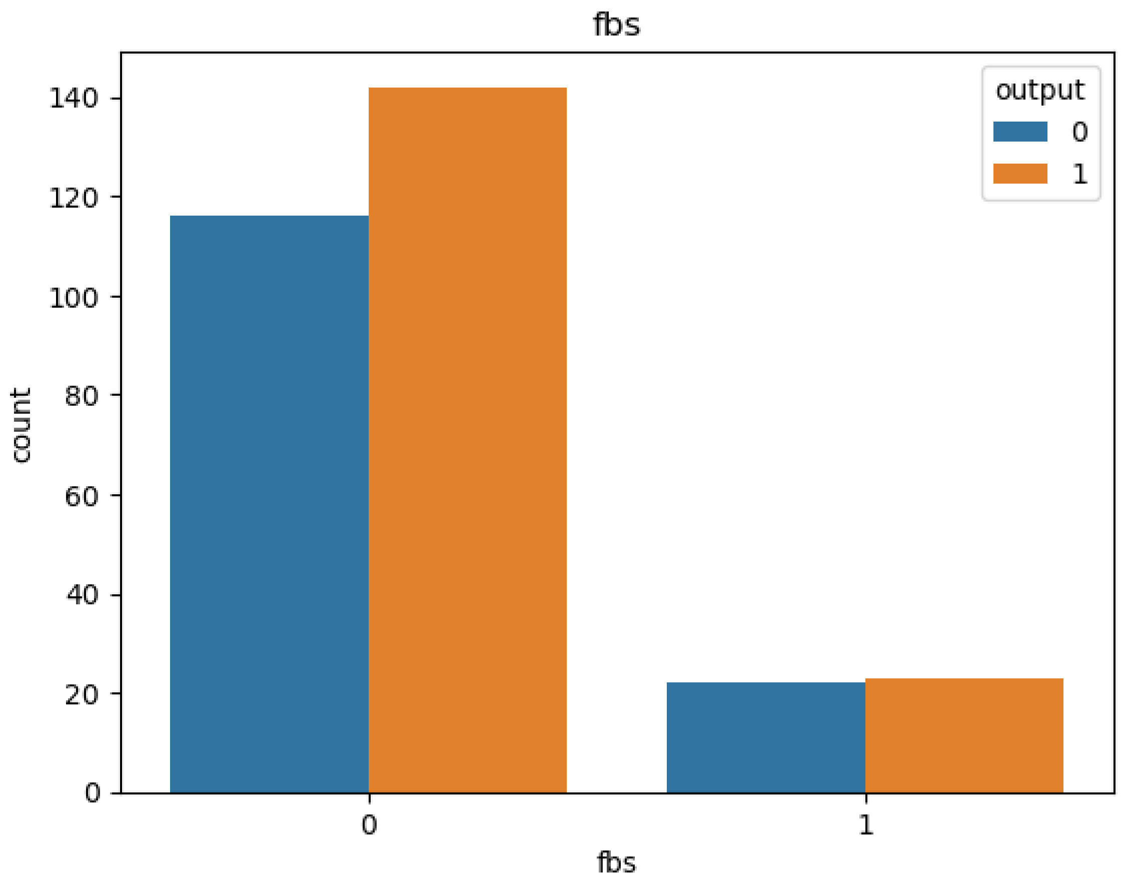
```
for i in list(data.columns):
   print('{} -- {}'.format(i,data[i].value_counts().shape[0]))
```

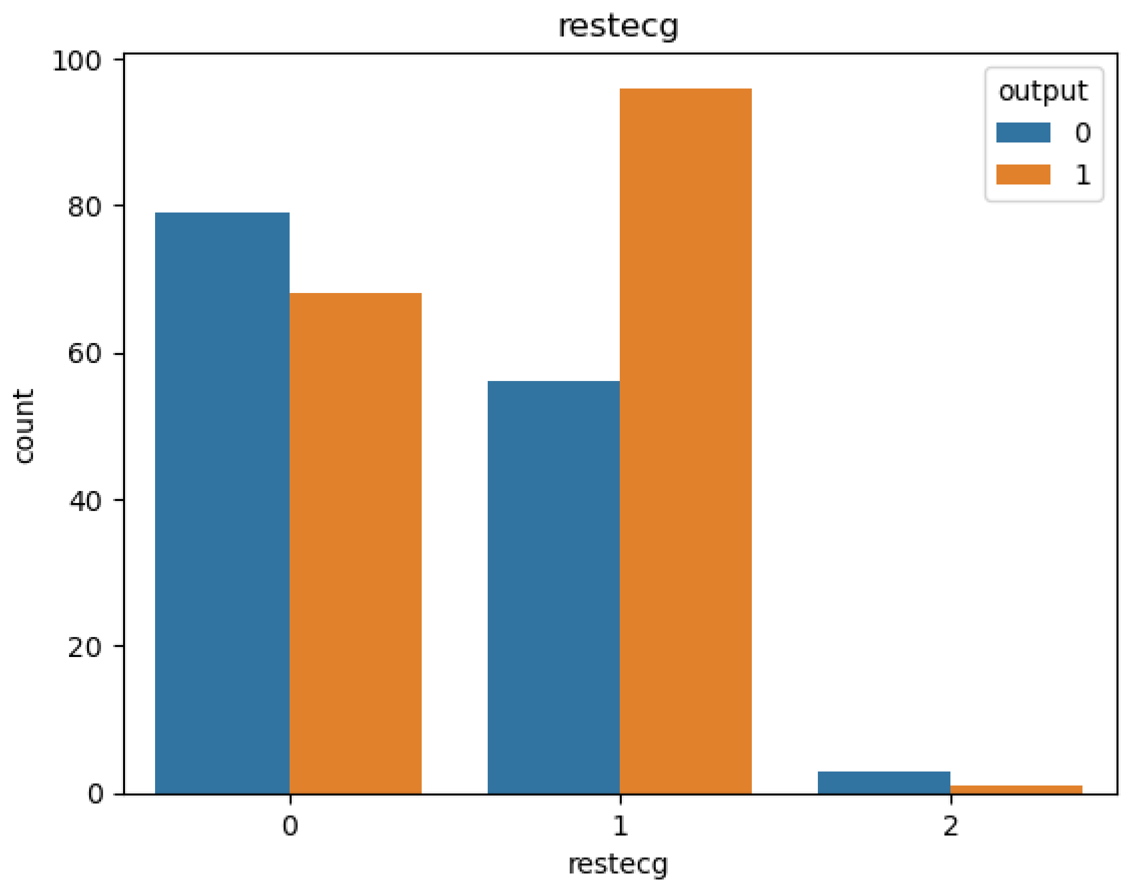
## Cathegorical Feature Analysis

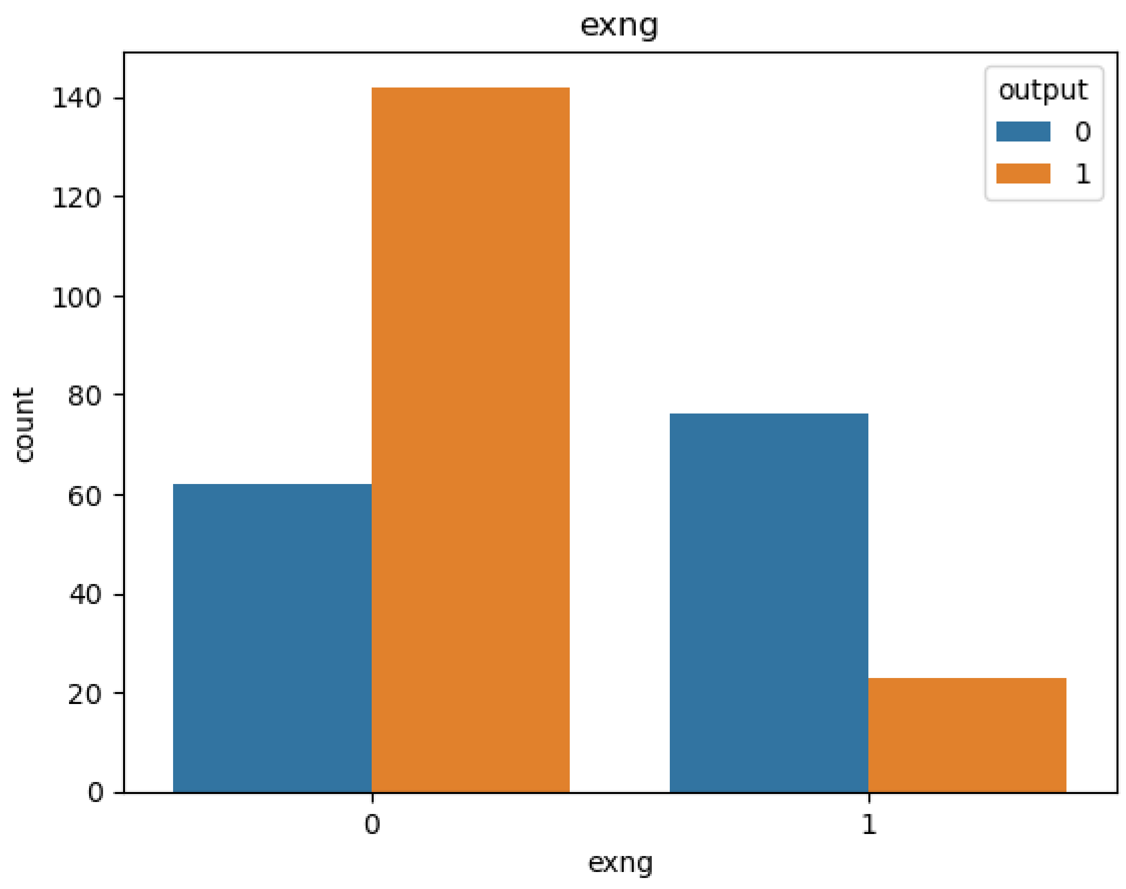
```
cathegorical_list = ['sex', 'cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall', 'output']
data_cathergoric = data.loc[:, cathegorical_list]
for i in cathegorical_list:
   plt.figure()
   sns.countplot(x=i, data = data_cathergoric, hue= 'output')
   plt.title(i)
```

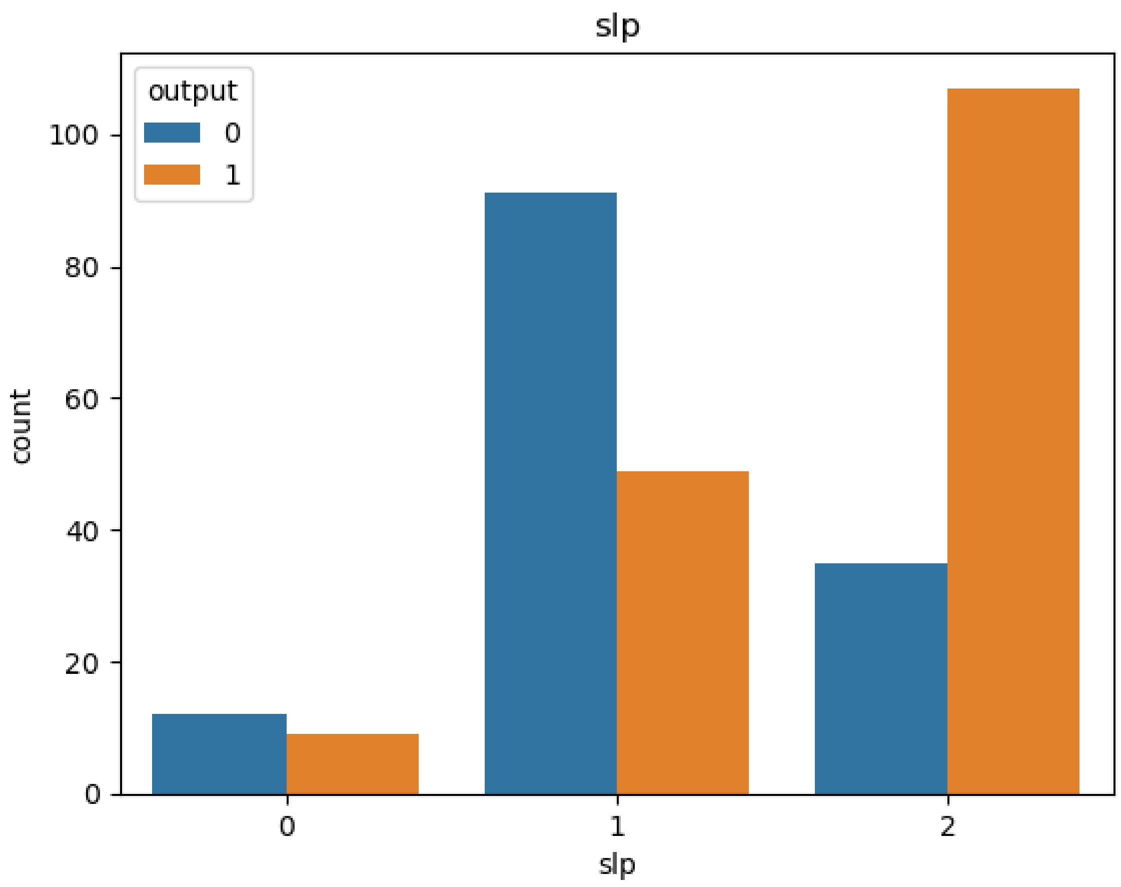


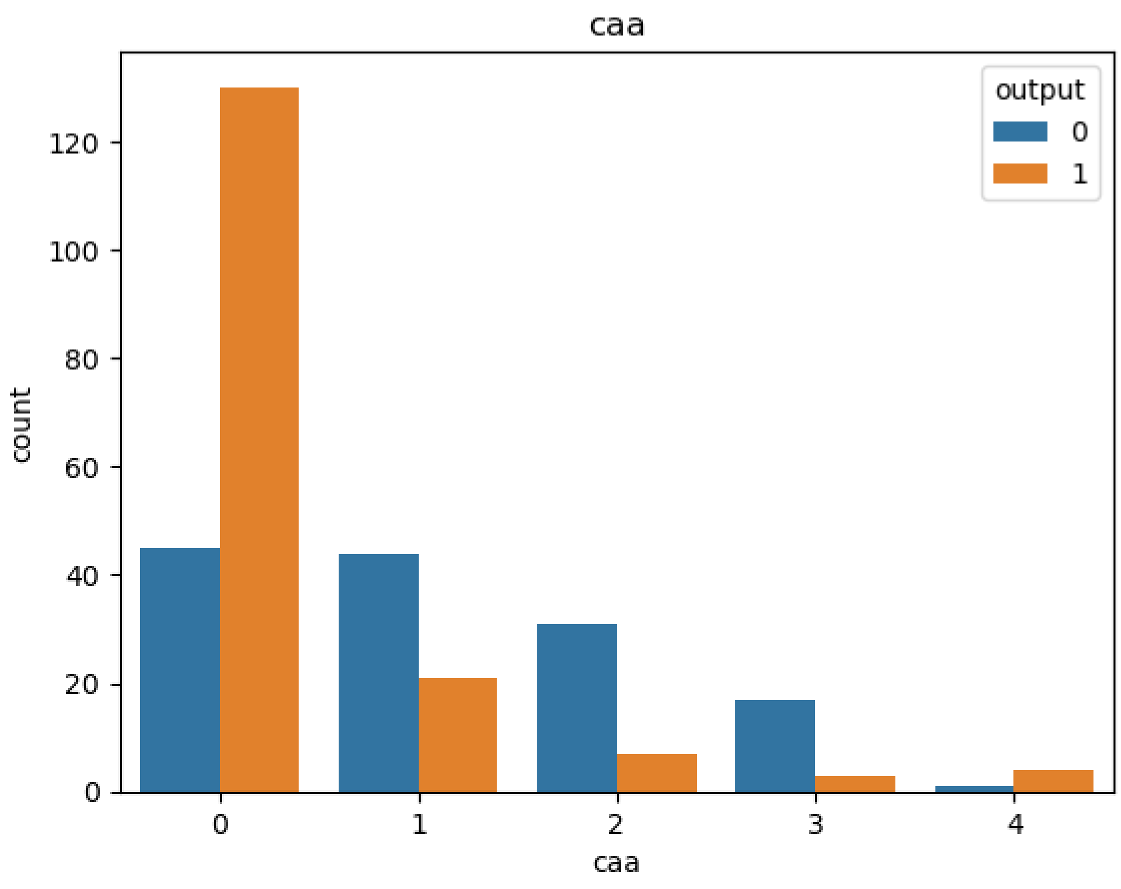

cp
output
0
1
count
cp
0
1
2
3
0
20
40
60
80
100

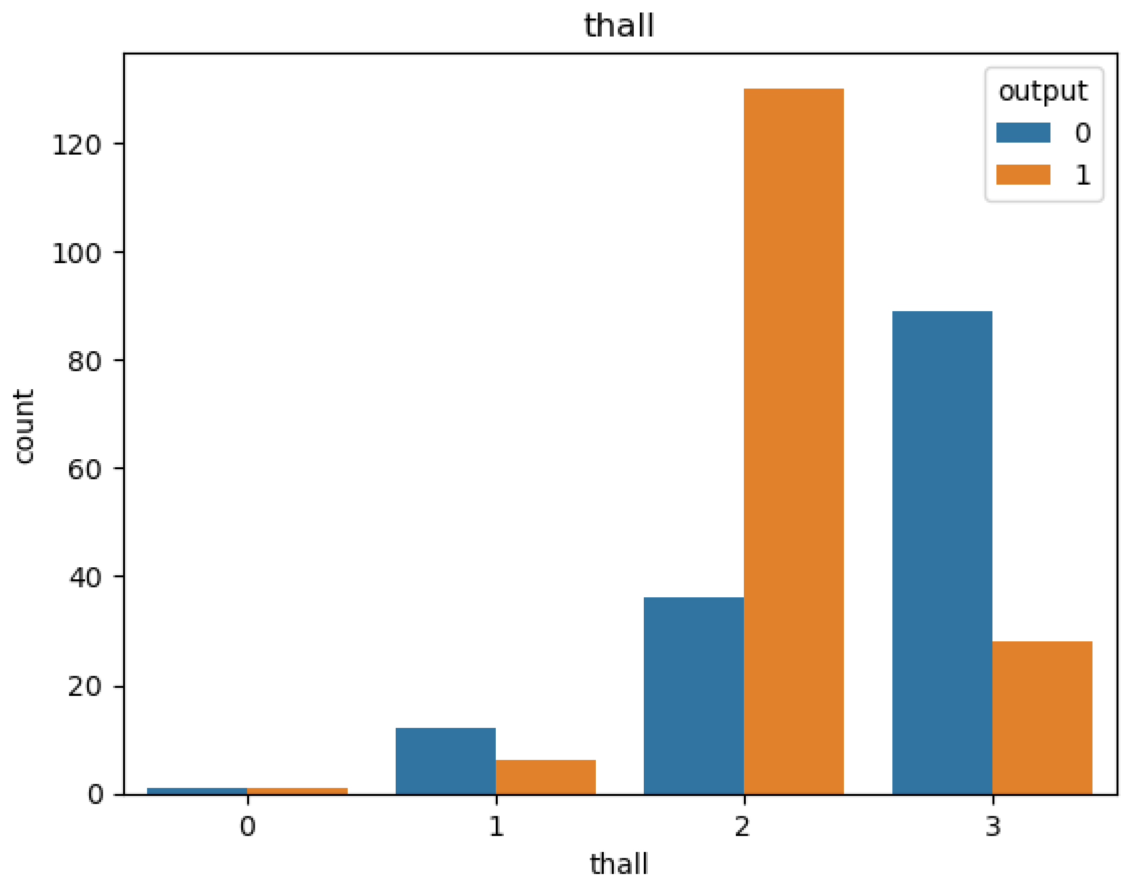

fbs
output
0
1
count
fbs
0
1
0
20
40
60
80
100
120
140

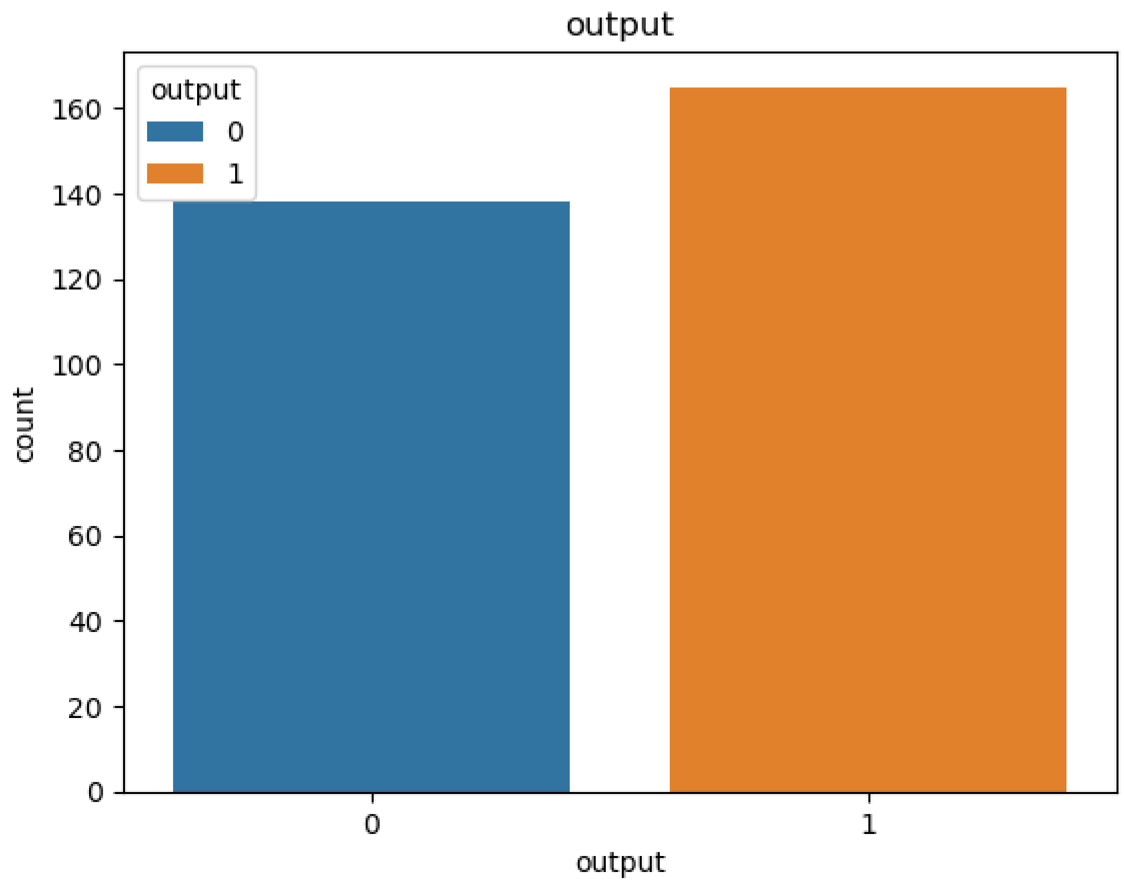

restecg
output
0
1
count
0
20
40
60
80
100
0
1
2
restecg


exng
output
0
1
count
0
20
40
60
80
100
120
140
0
1
exng


slp
output
0
1
count
100
80
60
40
20
0
0
1
2
slp


caa
output
0
1
count
120
100
80
60
40
20
0
0
1
2
3
4
caa


thall
output
0
1
count
120
100
80
60
40
20
0
0
1
2
3
thall


output
output
0
1
count
160
140
120
100
80
60
40
20
0
0
1
output

* sex = gender 0 and 1, 0 = no heart attack possibility, 1 = high possibility.

* 0 gender = higher possib of hearth atack (hpoha) rather then 1.

* cp = if they have 1 2 3 symptoms, they have higher possib of h.a.

* exng = if you have exng, less poha.

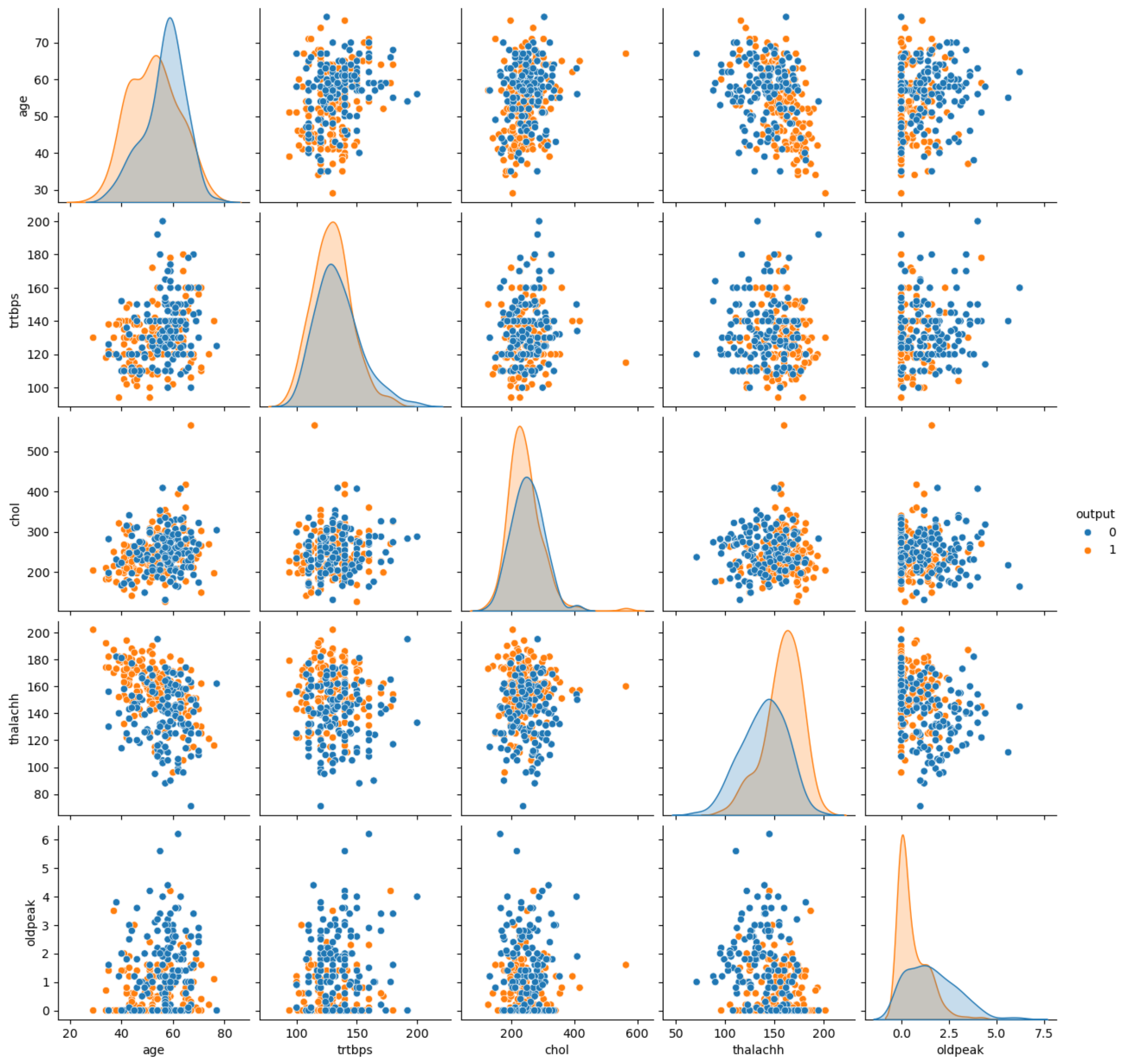
* for coa, if you are in 0 group, hpoha.

* thall = if 2, hpoha.

## Numeric Feature Analysis

* Bivariate data analysis with scatter plot.

```
numeric_list = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak', 'output']

data_numeric = data.loc[:, numeric_list]

sns.pairplot(data_numeric, hue = 'output', diag_kind = 'kde')

plt.show()
```



* for chol = too many outliers

* thalachh = negative skewness

* oldpeak = positive skewness

* pos and neg skewness produce otliers.

## Look at correlation.

* Age and thalachh = negative corr, age decrease, thalchh increase. age decrease hpoha.

* Other features; circle type = no correlation.

## Standardization

* to make standarize the data.

```
scaler = StandardScaler()

scaler

scaled_array = scaler.fit_transform(data[numeric_list[: -1]])

scaled_array
```

* std = 1, standarization is succesfull.
* Mean is too low.

```
pd.DataFrame(scaled_array).describe()
```

## Box Plot Analysis

```
data_dummy = pd.DataFrame(scaled_array, columns = numeric_list[:-1])

data_dummy.head()
```

* Concat both dataframe.

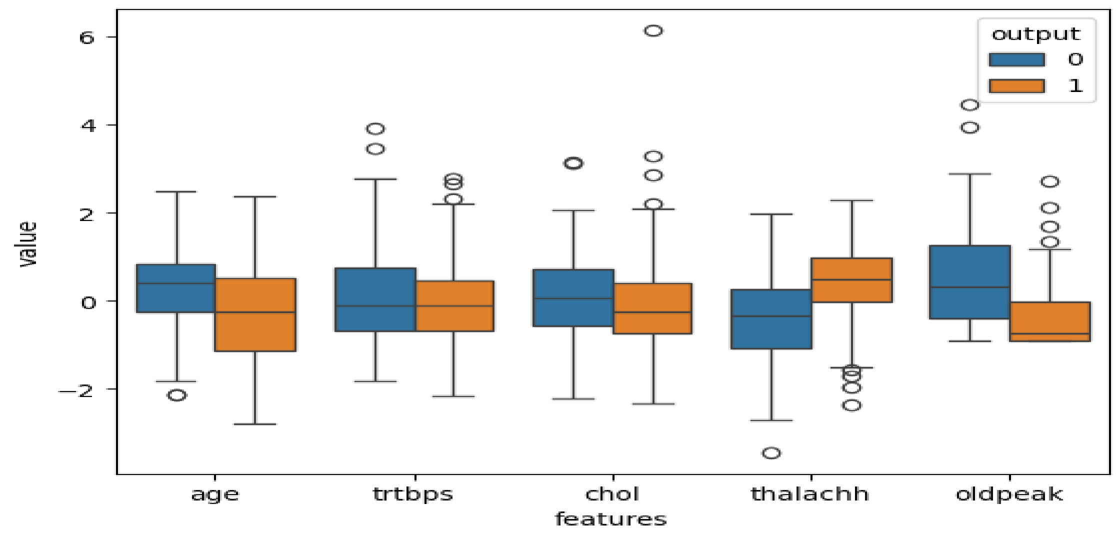
```
data_dummy = pd.concat([data_dummy, data.loc[:, 'output']], axis = 1)

data_dummy.head()
```

* It gives a list of data_dummy.

```
data_melted = pd.melt(data_dummy, id_vars='output', var_name='features',
value_name='value')

data_melted.head(20)
```

* Box plot.

```
plt.figure()

sns.boxplot(x = 'features', y = 'value', hue = 'output', data = data_melted)

plt.show()
```



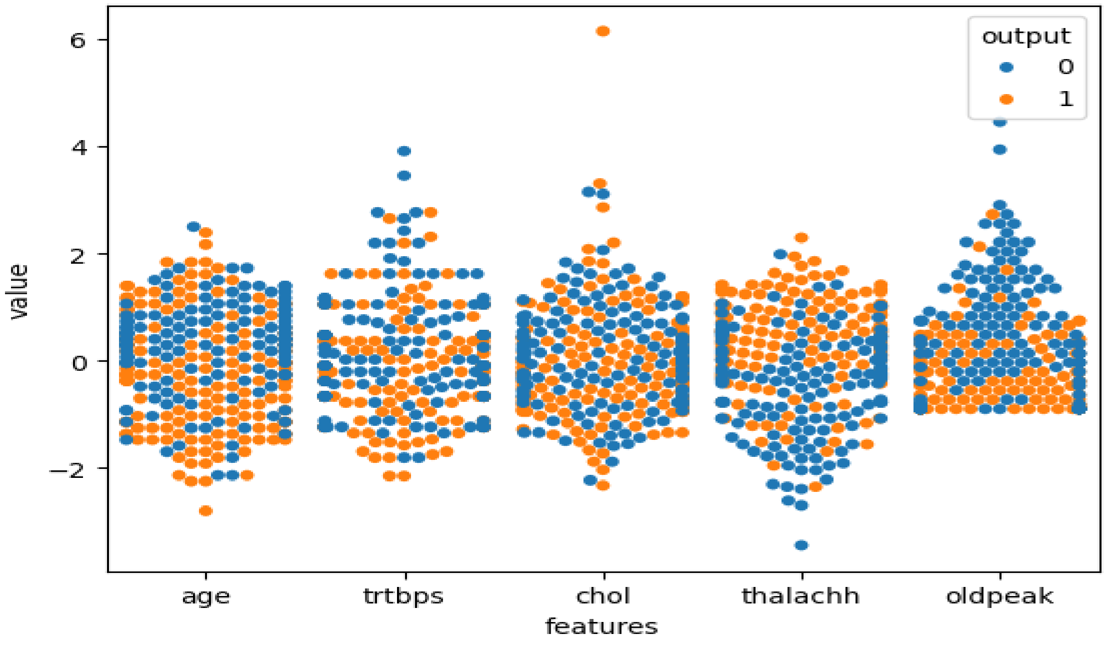
* Go to numeric value analysis and compare.

* Chol, someone has a high rate. In NVA you can see that there is someone who has a huge chol.

## Swarm Plot Analysis

* To see based on individual results.

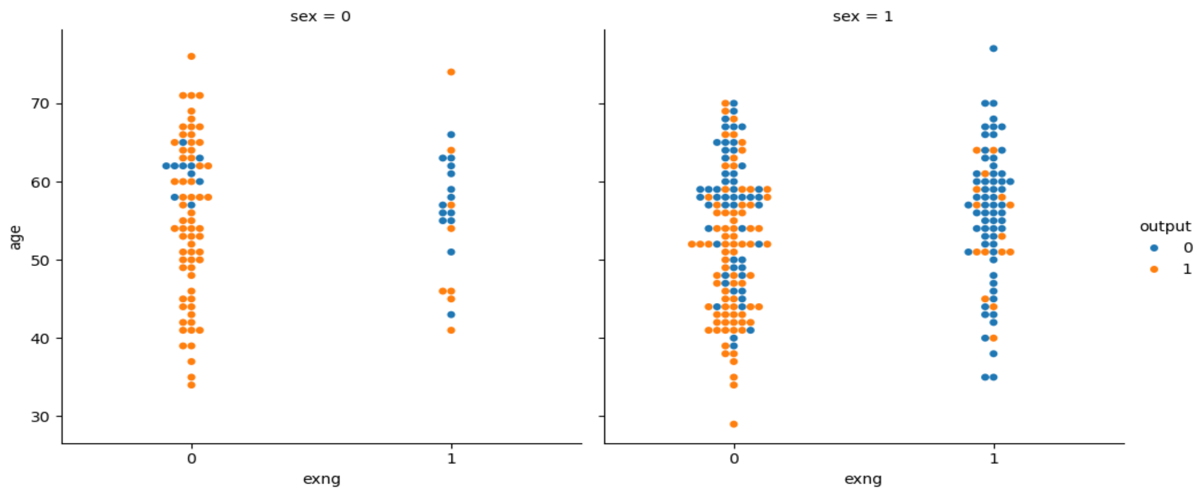
```
plt.figure()
sns.swarmplot(x = 'features', y = 'value', hue = 'output', data = data_melted)
plt.show()
```



* for thalachh oranges are above, and oldpeak oranges are below.

## Cat Plot Analysis

```
sns.catplot(x = 'exng', y = 'age', hue = 'output', col = 'sex', kind = 'swarm', data = data)
plt.show()
```

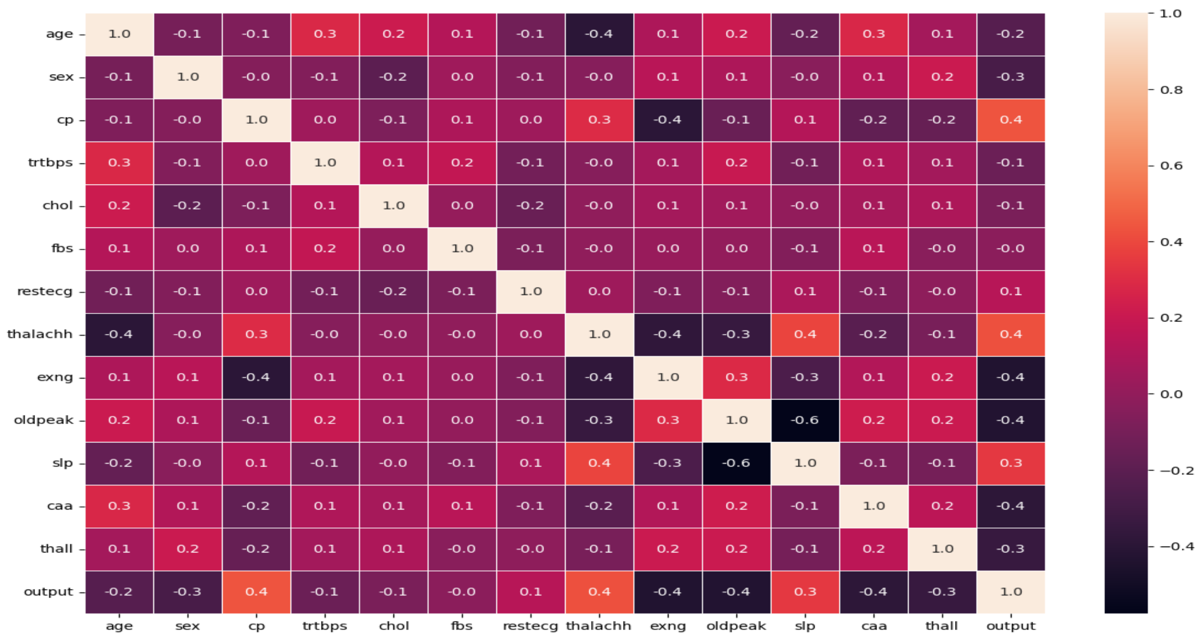


## CORRELATION ANALYSIS

```
data.corr()
```

* if the corr = 1, lineer relations, corr = 0 no relations, corr =-1 negative relations.

```
plt.figure(figsize =(14, 10))
sns.heatmap(data.corr(), annot=True, fmt='.1f', linewidths=0.7)
plt.show()
```



* darker color = unlinear relations

```
## Outlier Detection
        numeric_list
        numeric_list2 = ['age', 'trtbps', 'chol', 'thalachh', 'oldpeak']
        data_numeric2 = data.loc[:, numeric_list2]
        data_numeric2
        data.describe()
        for i in numeric_list2:
          Q1 = np.percentile(data.loc[:, i], 25)
          Q3 = np.percentile(data.loc[:, i], 75)

          IQR = Q3 - Q1
          print('Old shape: ', data.loc[:, i].shape)

  # upper bound
         upper = np.where(data.loc[:, i] >= (Q3 + 2.5*IQR))

  # lower bound
        lower = np.where(data.loc[:, i] <= (Q1 - 2.5*IQR))

        print("{} -- {}".format(upper,lower))
          try:
            data.drop(upper[0], inplace = True)
          except: print('KeyError: {} not found in axis'.format(upper[0]))
          try:
            data.drop(lower[0], inplace = True)
          except: print('KeyError: {} not found in axis'.format(lower[0]))

        print('New shape: ', data.shape)
        data
```

# we dropped outliers.

## MODELLING

```
data2 = data.copy()
```

### Encoding Categorical Columns

* We need to transform all cathegorical values to numeric values.

* Dont do that: red:1, yellow:2, green:3

* Do that:

```
*       red     1 1 0 0 0
*       yellow  0 0 1 0 1
*       green   0 0 0 1 0
```

#cathegorical_list = ['sex', 'cp', 'fbs', 'restecg', 'exng', 'slp', 'caa', 'thall', 'output']

```
data2 = pd.get_dummies(data2, columns = cathegorical_list[:-1], drop_first=True)
```

# Convert boolean values to integers

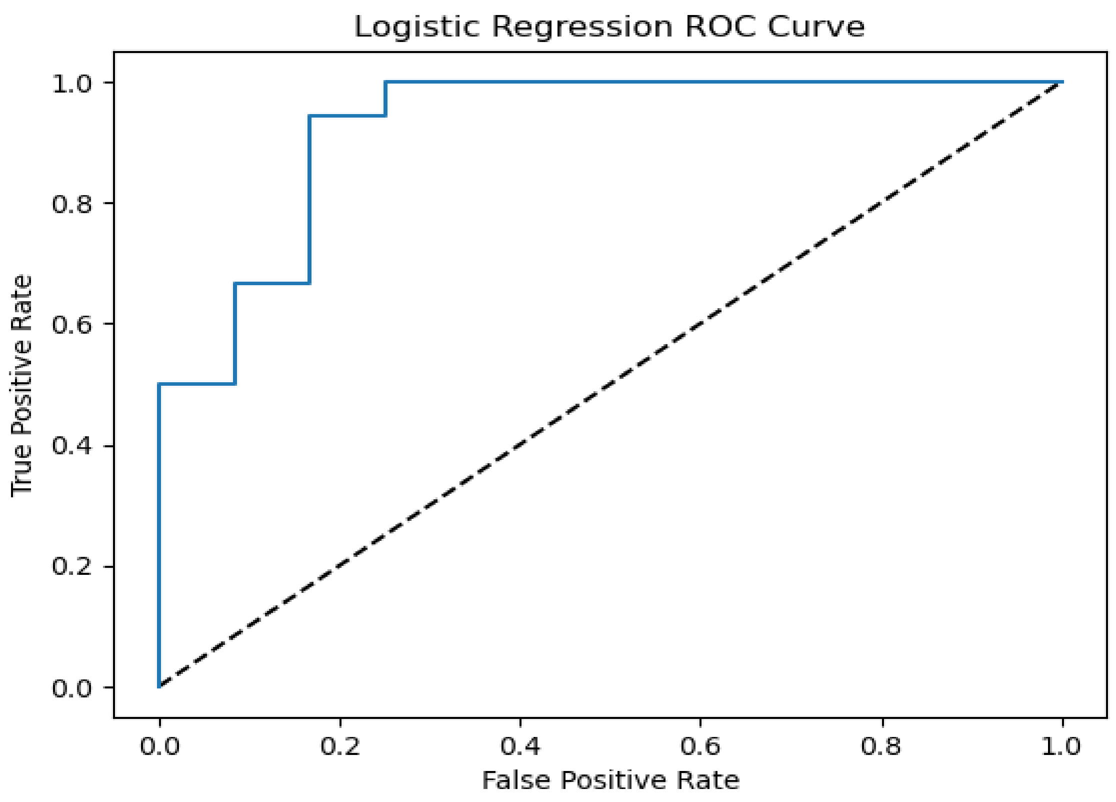
```
data2 = data2.astype(int)
data2.head()
x = data2.drop(['output'], axis = 1)
y = data2[['output']]
```

### SCALING

```
x[numeric_list[:-1]] = scaler.fit_transform(x[numeric_list[:-1]])
x.head()
```

```
### TRAIN/TEST SPLIT
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.1, random_state=3)

# 10% test
        print('x_train: {}'.format(x_train.shape))
        print('x_test: {}'.format(x_test.shape))
        print('y_train: {}'.format(y_train.shape))
        print('y_test: {}'.format(y_test.shape))

### LOGISTIC REGRESSION
        logreg = LogisticRegression()
        logreg

# Training
        logreg.fit(x_train, y_train)
        x_test

# Calculate probabilities
        y_pred_prob = logreg.predict_proba(x_test)
        y_pred_prob
# the sum of first column value and second column value shoul equal to 1.
# for first, output value 95% = 0 and 5% = 1
# transfer all to 0 and 1.
        y_pred = np.argmax(y_pred_prob, axis = 1)
        y_pred
        dummy_ = pd.DataFrame(y_pred_prob)
        dummy_['y_pred'] = y_pred
        dummy_.head()
        print('Test accuracy: {}'.format(accuracy_score(y_pred, y_test)))
```

```
# ROC CURVE
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob[:,1])
        y_pred_prob
        plt.plot([0,1], [0,1], 'k--')
        plt.plot(fpr, tpr, label = 'Logistic Regression')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Logistic Regression ROC Curve')
        plt.show()
```



* It close to left top corner, accuracy is 90%.

* Look more to details about ROC Curve.

### Logistic Regression Hyperparameter Tuning

* To protect overfitting there is some running behind 'regularization techniques'. These techniques are the parameters of logistic regression and called 'hyperparameter'. It prevent to memorisation.

```
lr = LogisticRegression()

lr
```

# Technique - 1

```
penalty = ['l1', 'l2']

parameters = {'penalty': penalty}

from sklearn.model_selection import GridSearchCV

lr_searcher = GridSearchCV(lr, parameters)

lr_searcher.fit(x_train, y_train)

print('Best Parameters: ', lr_searcher.best_params_)

y_pred = lr_searcher.predict(x_test)

print('Test Accuracy:{}'.format(accuracy_score(y_pred, y_test)))
```