Heart Attack Analysis & Prediction

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & R
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve

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

```
/kaggle/input/heart-attack-analysis-prediction-dataset/o2Saturation.csv
/kaggle/input/heart-attack-analysis-prediction-dataset/heart.csv
```

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
  - 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

Read and Analyse Data

- In this section, we read heart.csv

```
# read data
df = pd.read_csv("/kaggle/input/heart-attack-analysis-prediction-dataset/heart.csv")
```

```
df.head()
```

	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

```
# describe basic statistics of data
df.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	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	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	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	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	

```
# information about data frame
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
```

Missing Value Analysis

```
# missing value
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
```

Unique Value Analysis

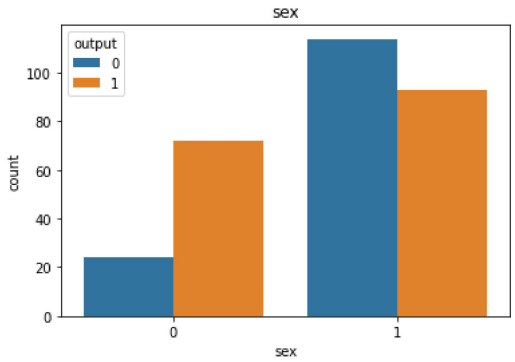
```
for i in list(df.columns):
    print("{} -- {}".format(i, df[i].value_counts().shape[0]))
```

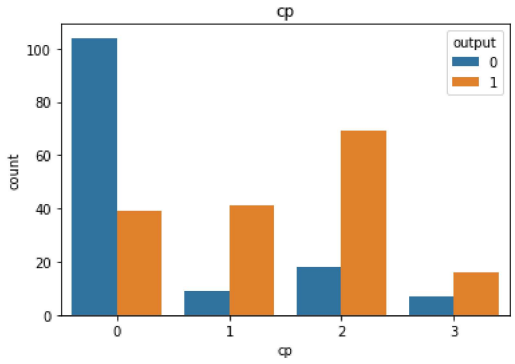
```
 age -- 41
 sex -- 2
 cp -- 4
 trtbps -- 49
 chol -- 152
 fbs -- 2
 restecg -- 3
 thalachh -- 91
 exng -- 2
 oldpeak -- 40
 slp -- 3
 caa -- 5
 thall -- 4
 output -- 2
```

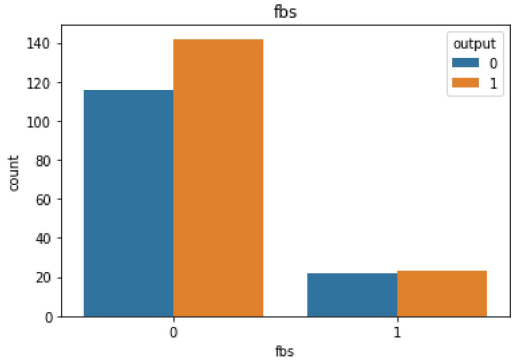
Categorical Feature Analysis

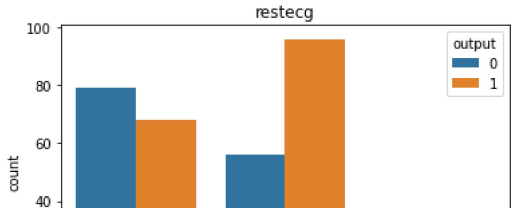
```
categorical_list = ["sex", "cp","fbs","restecg","exng","slp","caa","thall","output"]
```

```
df_categoric = df.loc[:, categorical_list]
for i in categorical_list:
    plt.figure()
    sns.countplot(x = i, data = df_categoric, hue = "output")
    plt.title(i)
```


sex
output
0
1
count
sex


cp
output
0
1
count
cp


fbs
output
0
1
count
fbs

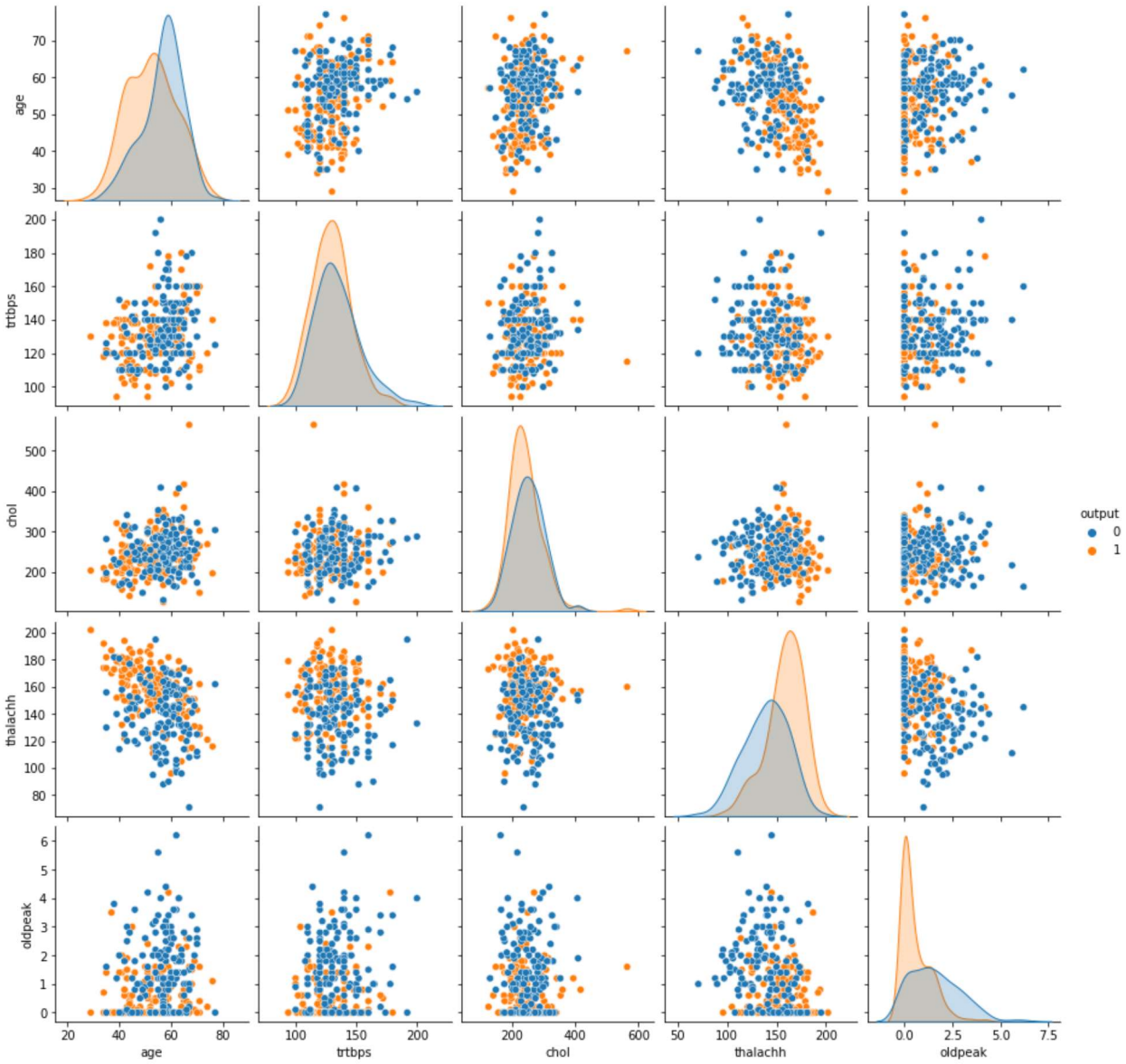

restecg
output
0
1
count

Numeric Feature Analysis

- Bivariate data analysis with scatter plot

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


df_numeric = df.loc[:, numeric_list]
sns.pairplot(df_numeric, hue = "output", diag_kind = "kde")
plt.show()
```



Standardization

```
scaler = StandardScaler()
scaler
```

```
StandardScaler()
```

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

```
scaled_array
```

```
array([[ 0.9521966 ,  0.76395577, -0.25633371,  0.01544279,  1.08733806],
       [-1.91531289, -0.09273778,  0.07219949,  1.63347147,  2.12257273],
       [-1.47415758, -0.09273778, -0.81677269,  0.97751389,  0.31091206],
       ...,
       [ 1.50364073,  0.70684287, -1.029353  , -0.37813176,  2.03630317],
       [ 0.29046364, -0.09273778, -2.2275329 , -1.51512489,  0.13837295],
       [ 0.29046364, -0.09273778, -0.19835726,  1.0649749 , -0.89686172]])
```

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

Box Plot Analysis

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

	age	trtbps	chol	thalachh	oldpeak
0	0.952197	0.763956	-0.256334	0.015443	1.087338
1	-1.915313	-0.092738	0.072199	1.633471	2.122573
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705
4	0.290464	-0.663867	2.082050	0.583939	-0.379244

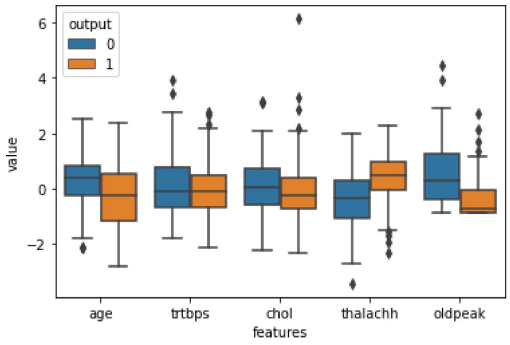
```
df_dummy = pd.concat([df_dummy, df.loc[:, "output"]], axis = 1)
df_dummy.head()
```

	age	trtbps	chol	thalachh	oldpeak	output
0	0.952197	0.763956	-0.256334	0.015443	1.087338	1
1	-1.915313	-0.092738	0.072199	1.633471	2.122573	1
2	-1.474158	-0.092738	-0.816773	0.977514	0.310912	1
3	0.180175	-0.663867	-0.198357	1.239897	-0.206705	1
4	0.290464	-0.663867	2.082050	0.583939	-0.379244	1

```
data_melted = pd.melt(df_dummy, id_vars = "output", var_name = "features", value_name = "value")
data_melted.head(20)
```

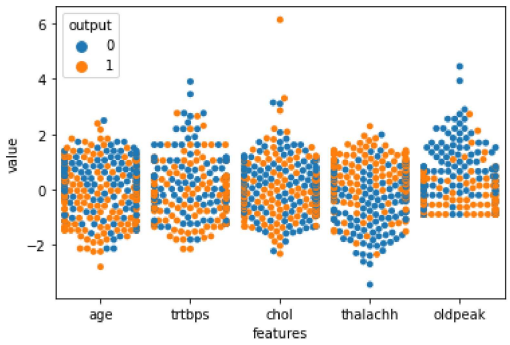
	output	features	value
0	1	age	0.952197
1	1	age	-1.915313
2	1	age	-1.474158
3	1	age	0.180175
4	1	age	0.290464
5	1	age	0.290464
6	1	age	0.180175
7	1	age	-1.143291
8	1	age	-0.260980
9	1	age	0.290464
10	1	age	-0.040403
11	1	age	-0.702136
12	1	age	-0.591847
13	1	age	1.062485
14	1	age	0.400752
15	1	age	-0.481558
16	1	age	0.400752
17	1	age	1.283063
18	1	age	-1.253580
19	1	age	1.613930

```
# box plot
plt.figure()
sns.boxplot(x = "features", y = "value", hue = "output", data= data_melted)
plt.show()
```



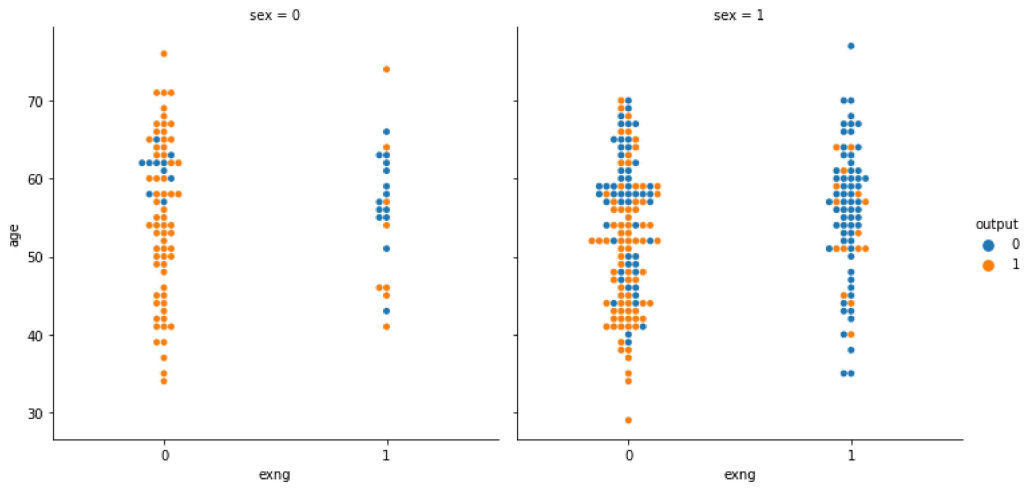
Swarm Plot Analysis

```
# swarm plot
plt.figure()
sns.swarmplot(x = "features", y = "value", hue = "output", data= data_melted)
plt.show()
```



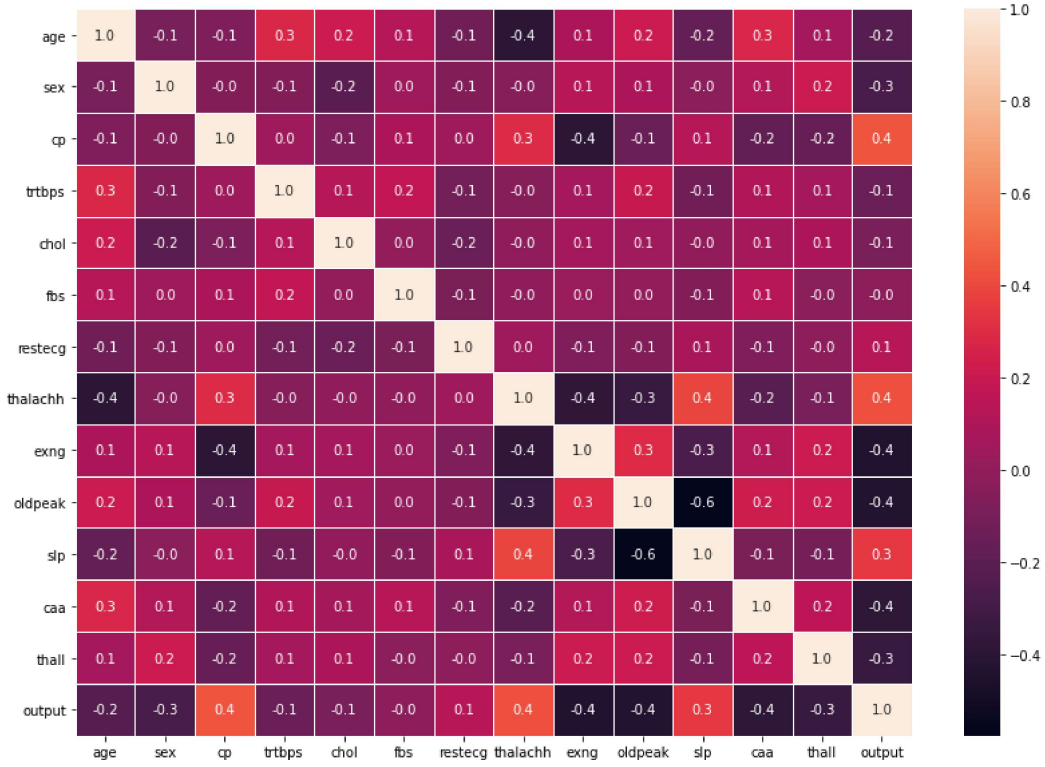
Cat Plot Analysis

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



Correlation Analysis

```
plt.figure(figsize = (14,10))
sns.heatmap(df.corr(), annot = True, fmt = ".1f", linewidths = .7)
plt.show()
```



Outlier Detection

- Outliers can disrupt ML process.
- Box-Plot-Diagram-to-identify-Outliers-figure-1.png

```
numeric_list = ["age", "trtbps","chol","thalachh","oldpeak"]
df_numeric = df.loc[:, numeric_list]
df_numeric.head()
```

	age	trtbps	chol	thalachh	oldpeak
0	63	145	233	150	2.3
1	37	130	250	187	3.5
2	41	130	204	172	1.4
3	56	120	236	178	0.8
4	57	120	354	163	0.6

```
df.describe()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	30
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	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	
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	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	
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	

```
# outlier detection
for i in numeric_list:

    # IQR
    Q1 = np.percentile(df.loc[:, i],25)
    Q3 = np.percentile(df.loc[:, i],75)

    IQR = Q3 - Q1

    print("Old shape: ", df.loc[:, i].shape)

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

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

    print("{} -- {}".format(upper, lower))

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

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

    print("New shape: ", df.shape)
```

```
 Old shape:  (303,)
 (array([], dtype=int64),) -- (array([], dtype=int64),)
 New shape:  (303, 14)
 Old shape:  (303,)
 (array([223, 248]),) -- (array([], dtype=int64),)
 New shape:  (301, 14)
 Old shape:  (301,)
 (array([85]),) -- (array([], dtype=int64),)
 New shape:  (300, 14)
 Old shape:  (300,)
 (array([], dtype=int64),) -- (array([], dtype=int64),)
 New shape:  (300, 14)
 Old shape:  (300,)
 (array([203, 220]),) -- (array([], dtype=int64),)
 New shape:  (298, 14)
```

Modelling

```
df1 = df.copy()
```

Encoding Categorical Columns

```
df1 = pd.get_dummies(df1, columns = categorical_list[:-1], drop_first = True)
df1.head()
```

	age	trtbps	chol	thalachh	oldpeak	output	sex_1	cp_1	cp_2	cp_3	...	exng_1	slp_1	slp_2	caa_1	caa_2	caa_3	caa_4	thall_1
0	63	145	233	150	2.3	1	1	0	0	1	...	0	0	0	0	0	0	0	1
1	37	130	250	187	3.5	1	1	0	1	0	...	0	0	0	0	0	0	0	0
2	41	130	204	172	1.4	1	0	1	0	0	...	0	0	1	0	0	0	0	0
3	56	120	236	178	0.8	1	1	1	0	0	...	0	0	1	0	0	0	0	0
4	57	120	354	163	0.6	1	0	0	0	0	...	1	0	1	0	0	0	0	0

5 rows × 23 columns

```
#
X = df1.drop(["output"], axis = 1)
y = df1[["output"]]
```

Scaling

```
scaler = StandardScaler()
scaler
```

```
StandardScaler()
```

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

	age	trtbps	chol	thalachh	oldpeak	sex_1	cp_1	cp_2	cp_3	fbs_1	...	exng_1	slp_1	slp_2	caa_1	caa_2	caa_3	caa_4
0	0.965901	0.845093	-0.236684	0.021855	2.3	1	0	0	1	1	...	0	0	0	0	0	0	0
1	-1.902555	-0.061886	0.119326	1.639116	3.5	1	0	1	0	0	...	0	0	0	0	0	0	0
2	-1.461254	-0.061886	-0.843995	0.983470	1.4	0	1	0	0	0	...	0	0	1	0	0	0	0
3	0.193624	-0.666538	-0.173859	1.245729	0.8	1	1	0	0	0	...	0	0	1	0	0	0	0
4	0.303950	-0.666538	2.297269	0.590082	0.6	0	0	0	0	0	...	1	0	1	0	0	0	0

5 rows × 22 columns

Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 3)
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))
```

```
X_train: (268, 22)
X_test: (30, 22)
y_train: (268, 1)
y_test: (30, 1)
```

Logistic Regression

```
logreg = LogisticRegression()
logreg
```

```
LogisticRegression()
```

```
# fitting = training
logreg.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# calculate probabilities
y_pred_prob = logreg.predict_proba(X_test)
y_pred_prob
```

```
array([[0.94252608, 0.05747392],
       [0.06987766, 0.93012234],
       [0.11254982, 0.88745018],
       [0.47977949, 0.52022051],
       [0.08754891, 0.91245109],
       [0.01966602, 0.98033398],
       [0.01313076, 0.98686924],
       [0.25608762, 0.74391238],
       [0.93025651, 0.06974349],
       [0.04680322, 0.95319678],
       [0.95711862, 0.04288138],
       [0.01124265, 0.98875735],
       [0.41858447, 0.58141553],
       [0.6057325 , 0.3942675 ],
       [0.02733085, 0.97266915],
       [0.0261538 , 0.9738462 ],
       [0.84053044, 0.15946956],
       [0.03593571, 0.96406429],
       [0.86161159, 0.13838841],
```