Project Name: Heart Attack Risk Predictor

• Project is to predict wheatear person is having heart attack or not by using automated machine learning techniques.

• In Project ML algorithms uses for understanding data & to use Auto ML Techniques, EVAL ML to make work simpler

In this project we will Make an app which will help us predict the risk of a Heart Attack a person have.

We will do use various Algorithms to predict the result and see which one suits best and then we will use Auto ML Library EVAL ML to predict the results.

We will do the following things:

- Data Analysis
- Feature Engineering
- Satandardization
- Model Building
- Predictions

Let us import the necessary liabraries and read our DataSet

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Let us import our Data Set

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=Tru
```

```
df= pd.read_csv("/content/drive/MyDrive/heart.csv")
```

```
df= df.drop(['oldpeak','slp','thall'],axis=1)
```

```
df.head()
```

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

Data Analysis

Understanding our DataSet:

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 0: typical angina
- Value 1: atypical angina
- Value 2: non-anginal pain
- Value 3: 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

```
df.shape
```

```
(303, 11)
```

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

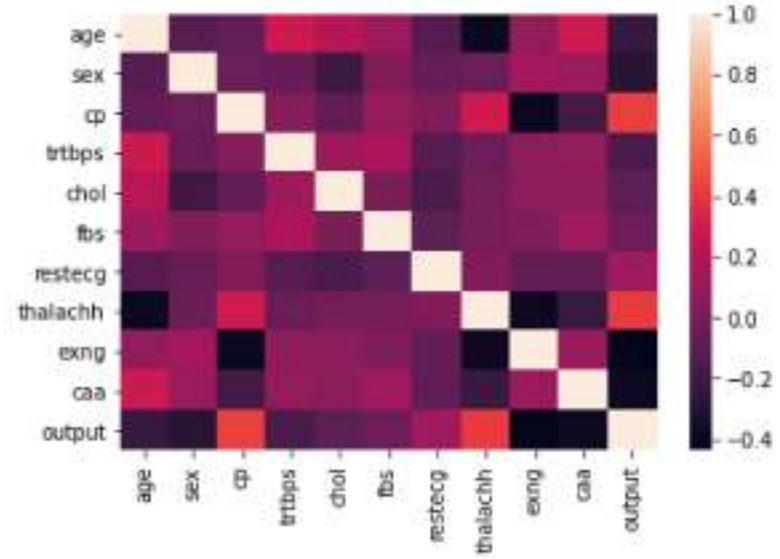
```
age         0
sex         0
cp          0
trtbps      0
chol        0
fbs         0
restecg     0
thalachh    0
exng        0
caa         0
output      0
dtype: int64
```

- As we can see there are no null values in our Data Set

```
df.corr()
```

```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1620b9a50>
```

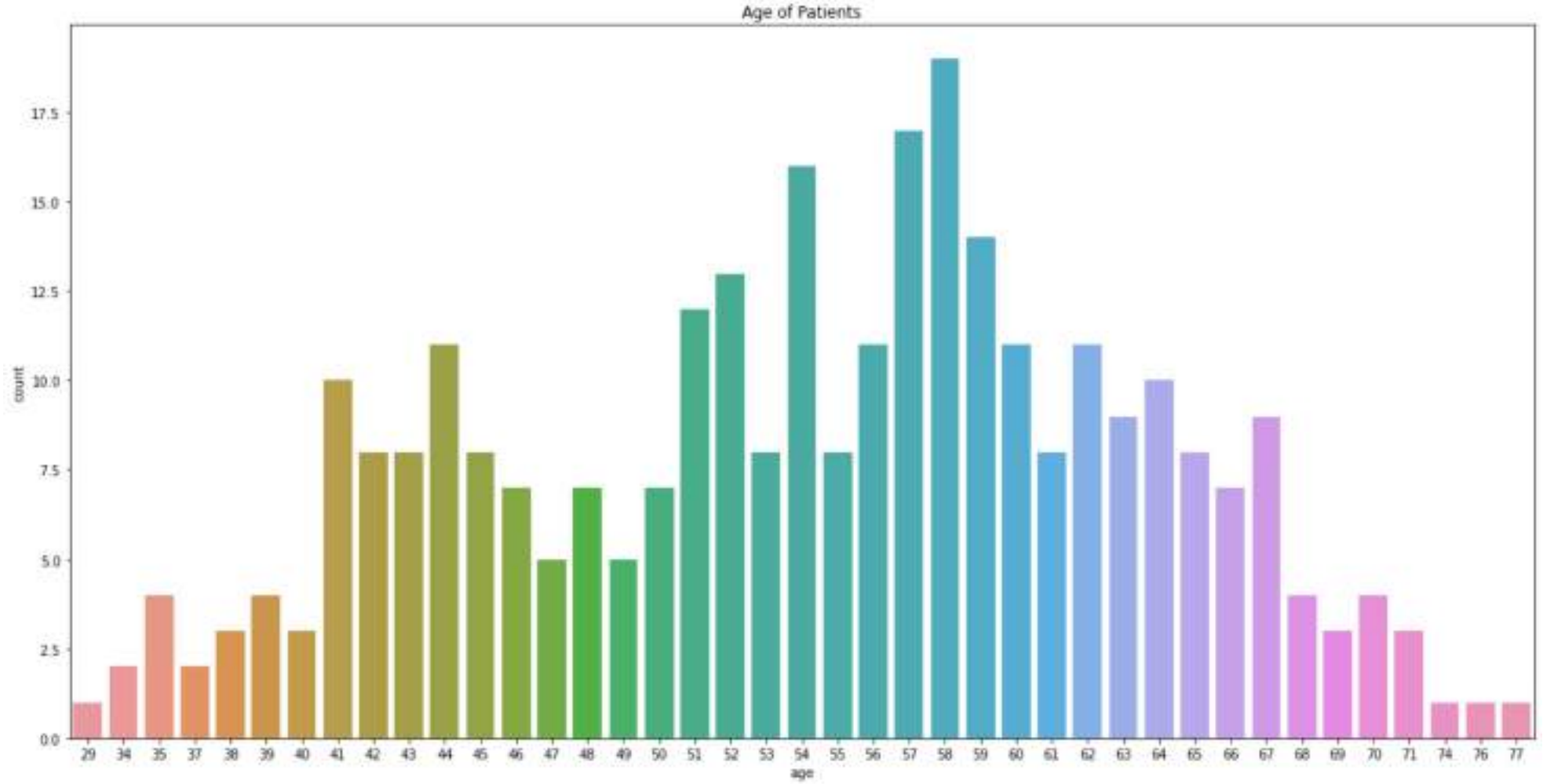


As we can see our variables are not highly correlated to each other

- We will do Uni and Bi variate analysis on our Features

```
plt.figure(figsize=(20, 10))
plt.title("Age of Patients")
plt.xlabel("Age")
sns.countplot(x='age',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd162022a90>
```

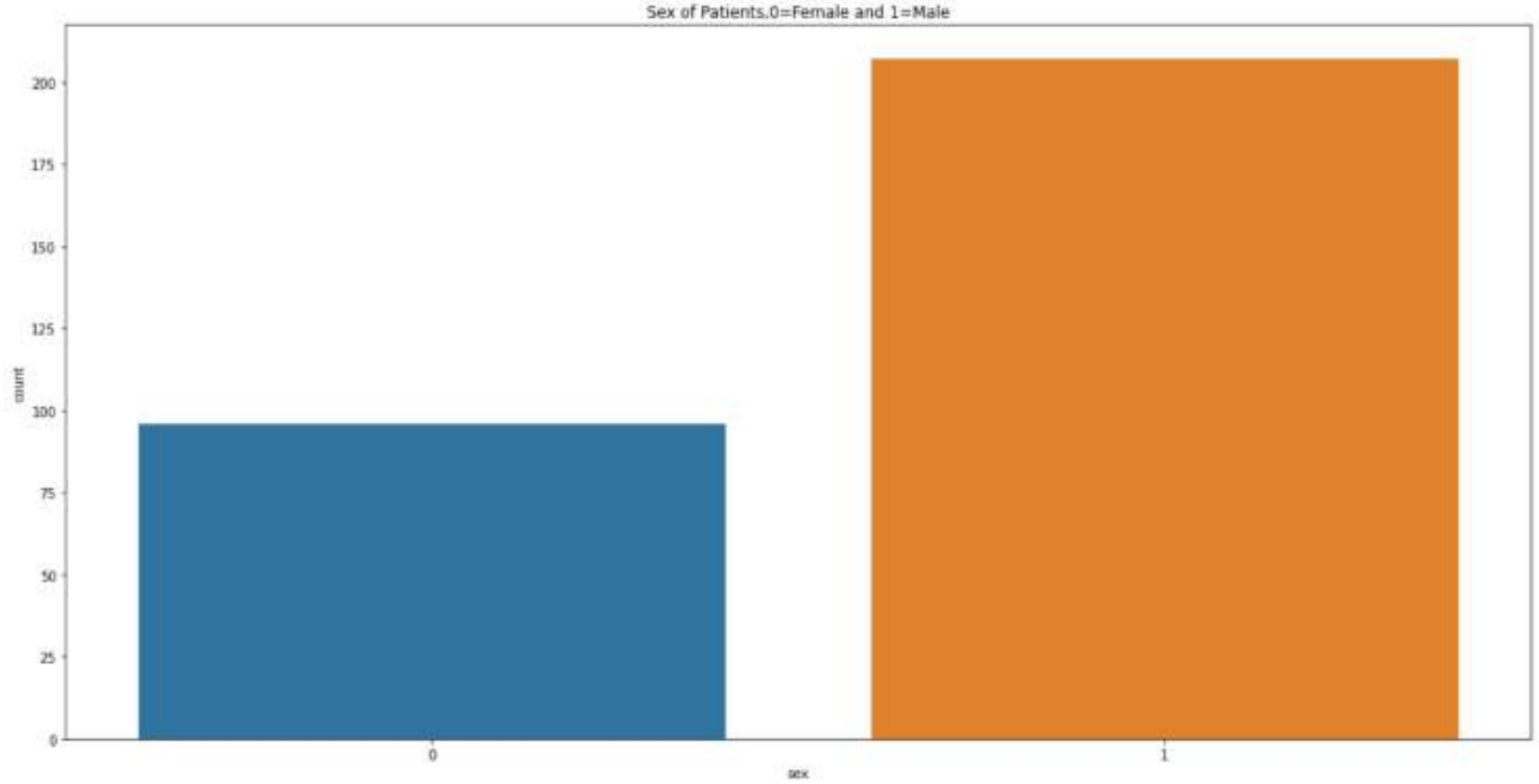


- As we can see the Patients are of Age Group 51-67years in majority

```
plt.figure(figsize=(20, 10))
plt.title("Sex of Patients,0=Female and 1=Male")
```

```
sns.countplot(x='sex',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd161fda710>
```




```
cp_data= df['cp'].value_counts().reset_index()
cp_data['index'][3]= 'asymptomatic'
```

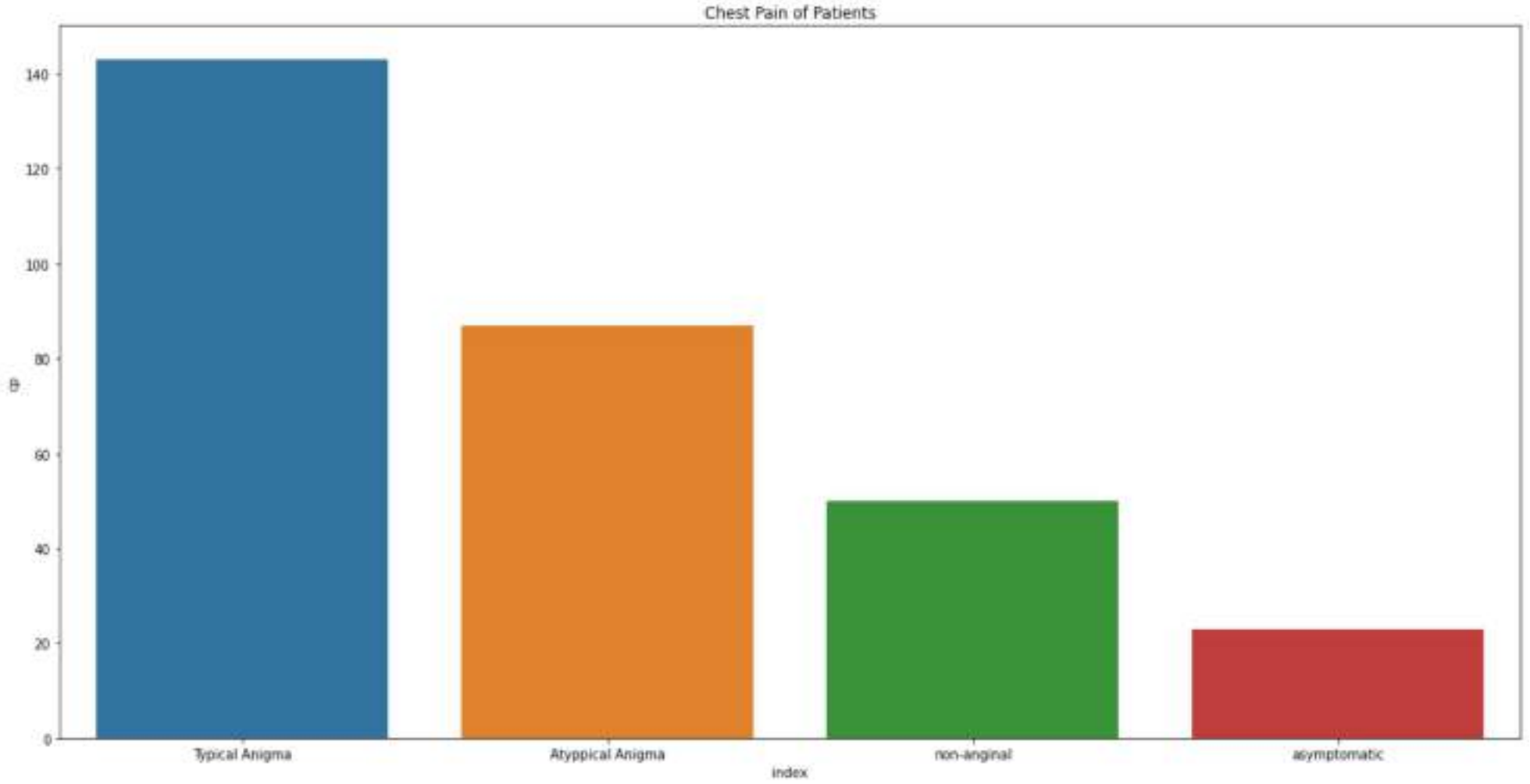
```
cp_data['index'][2]= 'non-anginal'
cp_data['index'][1]= 'Atyppical Anigma'
cp_data['index'][0]= 'Typical Anigma'
cp_data
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  after removing the cwd from sys.path.
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  """
```

	index	cp
0	Typical Anigma	143
1	Atyppical Anigma	87
2	non-anginal	50
3	asymptomatic	23

```
plt.figure(figsize=(20, 10))
plt.title("Chest Pain of Patients")

sns.barplot(x=cp_data['index'],y= cp_data['cp'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd161e5dbd0>
```



- We have seen how the the Chest Pain Category is distributed

```
ecg_data= df['restecg'].value_counts().reset_index()
ecg_data['index'][0]= 'normal'
ecg_data['index'][1]= 'having ST-T wave abnormality'
```

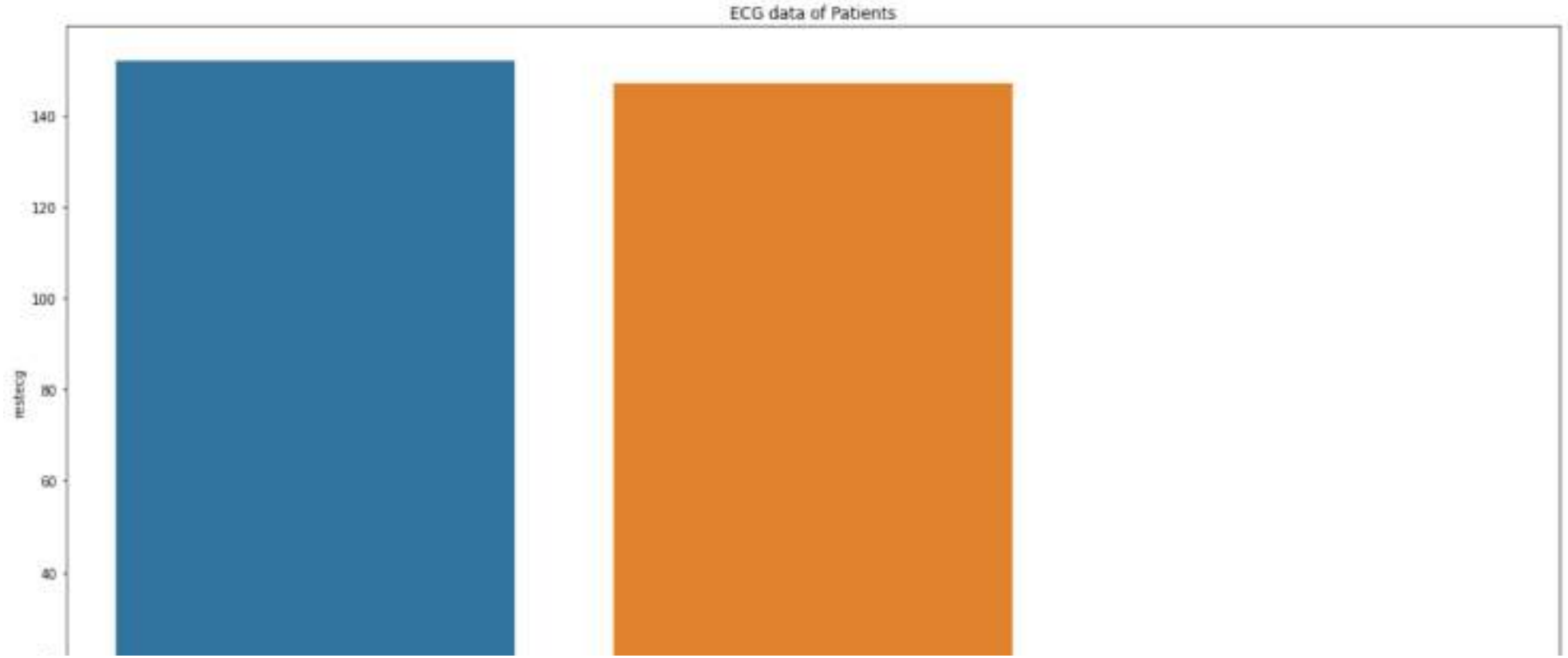
```
ecg_data['index'][2]= 'showing probable or definite left ventricular hypertrophy by Estes'

ecg_data
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  This is separate from the ipykernel package so we can avoid doing imports until
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
  after removing the cwd from sys.path.
```

	index	restecg
0	normal	152
1	having ST-T wave abnormality	147
2	showing probable or definite left ventricular ...	4

```
plt.figure(figsize=(20, 10))
plt.title("ECG data of Patients")

sns.barplot(x=ecg_data['index'],y= ecg_data['restecg'])
```

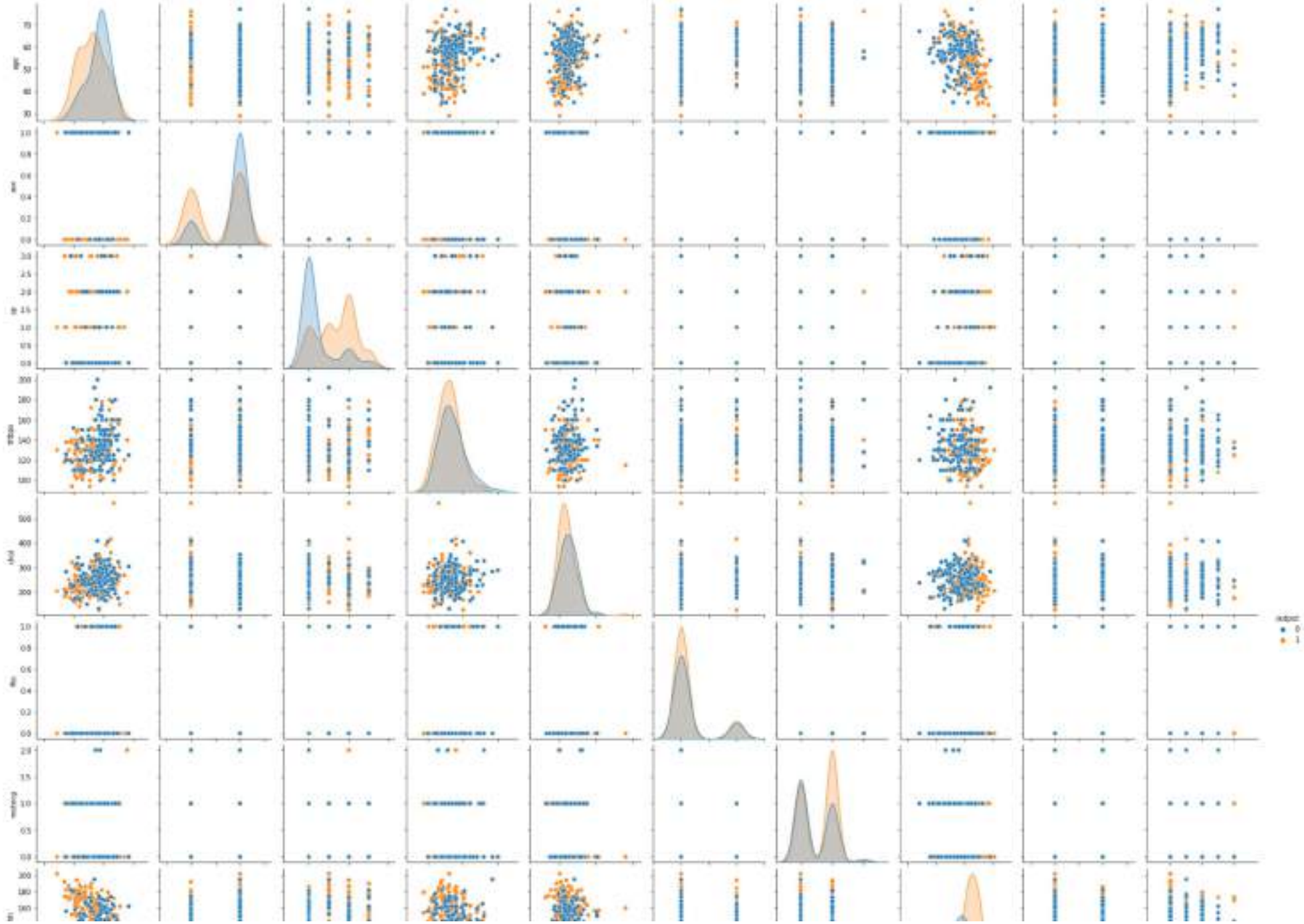
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd161dd7cd0>
```

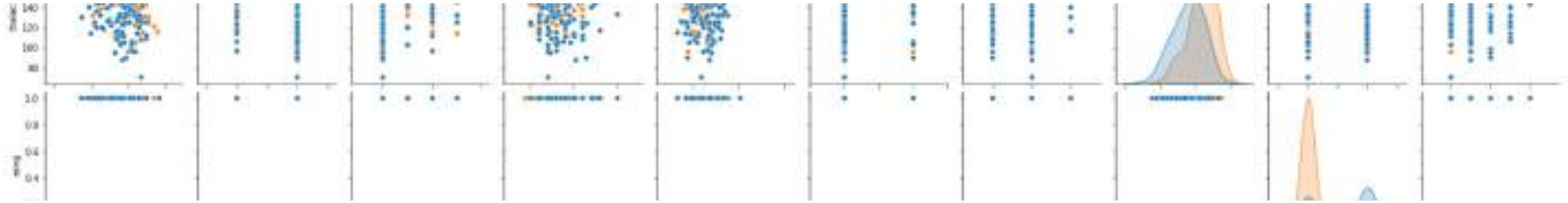


This is our ECG Data

```
sns.pairplot(df,hue='output',data=df)
```

<seaborn.axisgrid.PairGrid at 0x7fd161d4c850>

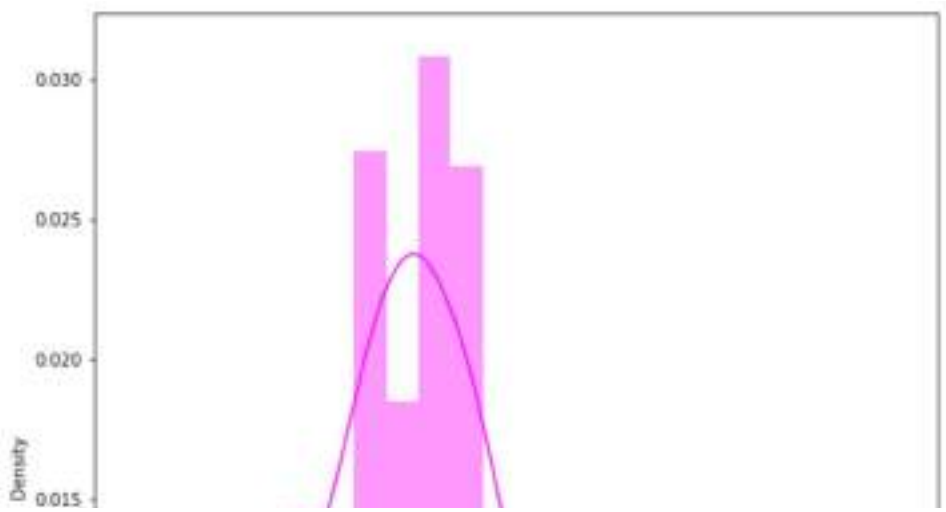


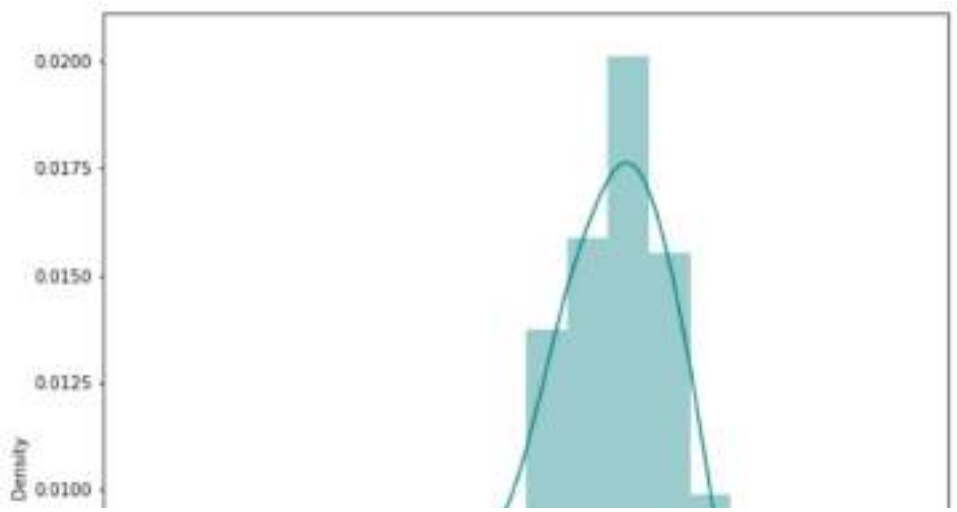


Let us see for our Continuous Variable

```
plt.figure(figsize=(20,10))
plt.subplot(1,2,1)
sns.distplot(df['trtbps'], kde=True, color = 'magenta')
plt.xlabel("Resting Blood Pressure (mmHg)")
plt.subplot(1,2,2)
sns.distplot(df['thalachh'], kde=True, color = 'teal')
plt.xlabel("Maximum Heart Rate Achieved (bpm)")
```

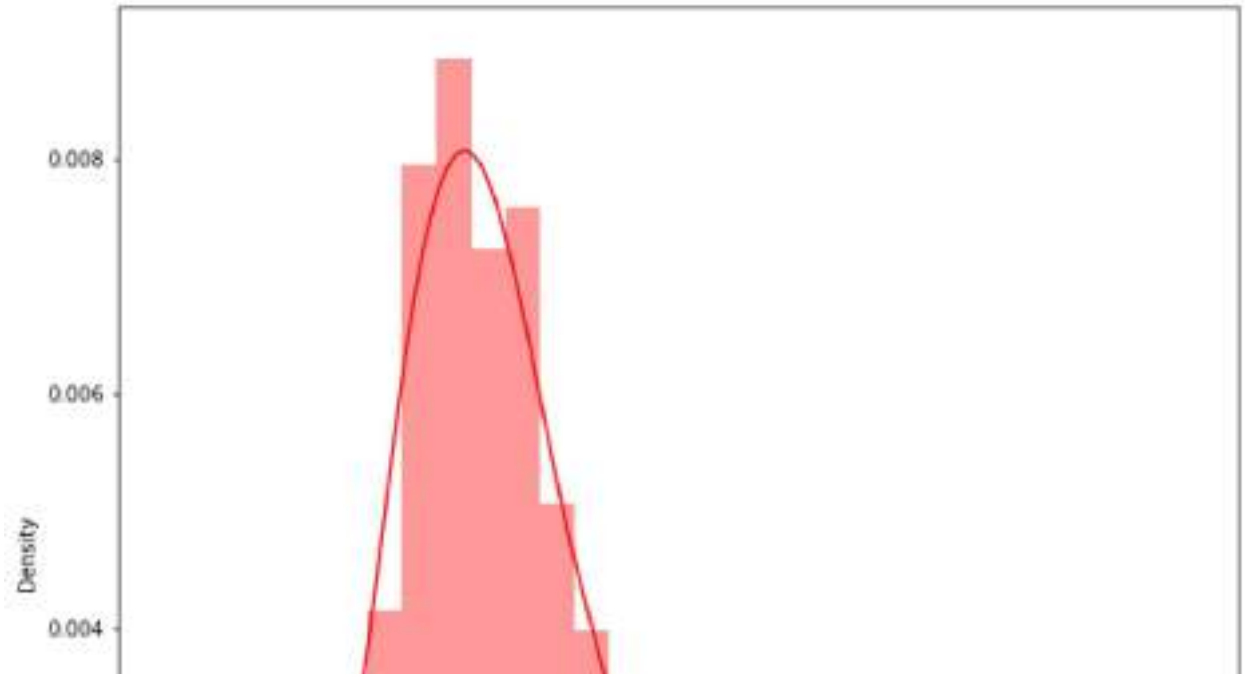
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated function and wi
  warnings.warn(msg, FutureWarning)
Text(0.5, 0, 'Maximum Heart Rate Achieved (bpm)')
```






```
plt.figure(figsize=(10,10))
sns.distplot(df['chol'], kde=True, color = 'red')
plt.xlabel("Cholestrol")
```

```
C:\Users\khank\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function an
  warnings.warn(msg, FutureWarning)
Text(0.5, 0, 'Cholestrol')
```



- We have done the Analysis of the data now let's have a look at out data

```
df.head()
```

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

Let us do Standardisation

```
from sklearn.preprocessing import StandardScaler

scale=StandardScaler()

scale.fit(df)
```

```
StandardScaler()
```

```
df= scale.transform(df)

df=pd.DataFrame(df,columns=['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'caa', 'output'])

df.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	caa	output
0	0.952197	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832	0.015443	-0.696631	-0.714429	0.914529
1	-1.915313	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962	1.633471	-0.696631	-0.714429	0.914529
2	-1.474158	-1.468418	0.032031	-0.092738	-0.816773	-0.417635	-1.005832	0.977514	-0.696631	-0.714429	0.914529
3	0.180175	0.681005	0.032031	-0.663867	-0.198357	-0.417635	0.898962	1.239897	-0.696631	-0.714429	0.914529
4	0.290464	-1.468418	-0.938515	-0.663867	2.082050	-0.417635	0.898962	0.583939	1.435481	-0.714429	0.914529

We can insert this data into our ML Models

We will use the following models for our predictions :

- Logistic Regression
- Decision Tree
- Random Forest
- K Nearest Neighbour
- SVM

Then we will use the ensembling techniques

Let us split our data

```
x= df.iloc[:,:-1]
x
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	caa
0	0.952197	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832	0.015443	-0.696631	-0.714429
1	-1.915313	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962	1.633471	-0.696631	-0.714429

```
y= df.iloc[:,-1:]
y
```

	output
0	0.914529
1	0.914529
2	0.914529
3	0.914529
4	0.914529
...	...
298	-1.093459
299	-1.093459
300	-1.093459
301	-1.093459
302	-1.093459

303 rows × 1 columns

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=101)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.preprocessing import LabelEncoder
```

```
lbl= LabelEncoder()
```

```
encoded_y= lbl.fit_transform(y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:251: DataConversionWarning: A column-vector y was passed
  y = column_or_1d(y, warn=True)
```

```
logreg= LogisticRegression()
```

```
logreg = LogisticRegression()
logreg.fit(x_train, encoded_y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
Y_pred1
```

```
array([0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 0])
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


encoded_ytest= lbl.fit_transform(y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:251: DataConversionWarning: A column-vector y was passed
  y = column_or_1d(y, warn=True)
```

```
Y_pred1 = logreg.predict(x_test)
lr_conf_matrix = confusion_matrix(encoded_ytest,Y_pred1 )
lr_acc_score = accuracy_score(encoded_ytest, Y_pred1)


lr_conf_matrix
```

```
array([[35,  9],
       [ 4, 43]])
```

```
print(lr_acc_score*100,"%")
```

```
85.71428571428571 %
```

As we see the Logistic Regression Model have a 85% accuracy

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier


tree= DecisionTreeClassifier()
```

```
tree.fit(x_train,encoded_y)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
ypred2=tree.predict(x_test)
```

```
encoded_ytest= lbl.fit_transform(y_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/preprocessing/_label.py:251: DataConversionWarning: A column-vector y was passed
  y = column_or_1d(y, warn=True)
```

```
tree_conf_matrix = confusion_matrix(encoded_ytest,ypred2 )
tree_acc_score = accuracy_score(encoded_ytest, ypred2)
```

```
tree_conf_matrix
```

```
array([[26, 18],
       [ 9, 38]])
```

```
print(tree_acc_score*100,"%")
```

```
70.32967032967034 %
```

As we see our Decision Tree Model does not perform well as it gives a score of only 69%

Random Forest

```
from sklearn.ensemble import RandomForestClassifier


rf= RandomForestClassifier()


rf.fit(x_train,encoded_y)

     RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                            criterion='gini', max_depth=None, max_features='auto',
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_jobs=None, oob_score=False, random_state=None,
                            verbose=0, warm_start=False)


ypred3 = rf.predict(x_test)


rf_conf_matrix = confusion_matrix(encoded_ytest,ypred3 )
rf_acc_score = accuracy_score(encoded_ytest, ypred3)


rf_conf_matrix

     array([[30, 14],
            [ 8, 39]])


print(rf_acc_score*100,"%")

     75.82417582417582 %
```

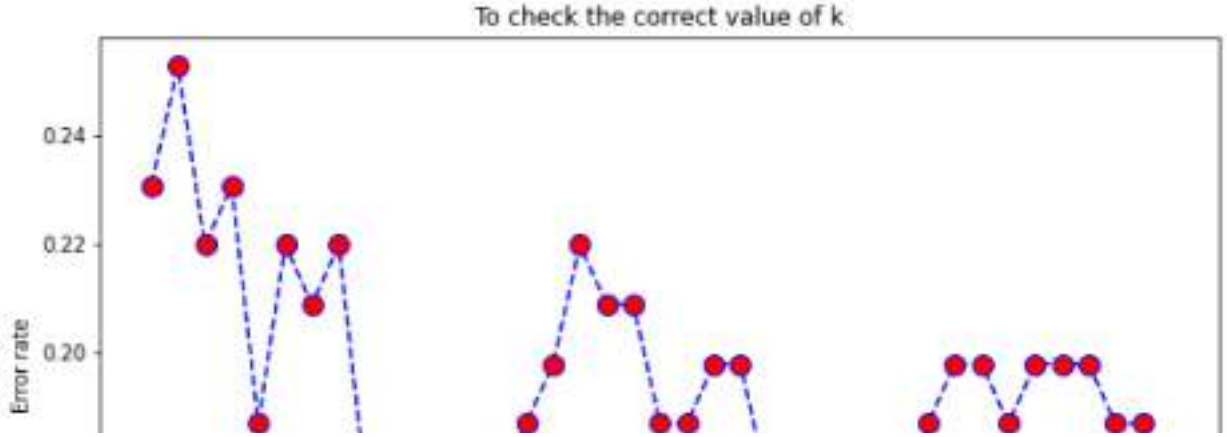
RF also gives us an accuracy of around 80%

K Nearest Neighbour

We have to select what k we will use for the maximum accuracy

Let's write a function for it

```
from sklearn.neighbors import KNeighborsClassifier


error_rate= []
for i in range(1,40):
    knn= KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train,encoded_y)
    pred= knn.predict(x_test)
    error_rate.append(np.mean(pred != encoded_ytest))


plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.xlabel('K Vlaue')
plt.ylabel('Error rate')
plt.title('To check the correct value of k')
plt.show()
```



- As we see from the graph we should select K= 12 as it gives the best error rate

```
knn= KNeighborsClassifier(n_neighbors=12)
knn.fit(x_train,encoded_y)
ypred4= knn.predict(x_test)
```

```
knn_conf_matrix = confusion_matrix(encoded_ytest,ypred4 )
knn_acc_score = accuracy_score(encoded_ytest, ypred4)
```

```
knn_conf_matrix
```

```
array([[35,  9],
       [ 5, 42]])
```

```
print(knn_acc_score*100,"%")
```

```
84.61538461538461 %
```

As we see KNN gives us an accuracy of around 85% which is good

Support Vector Machine(SVM)

```
from sklearn import svm
```

```
svm= svm.SVC()
```

```
svm.fit(x_train,encoded_y)
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
ypred5= svm.predict(x_test)
```

```
svm_conf_matrix = confusion_matrix(encoded_ytest,ypred5)
svm_acc_score = accuracy_score(encoded_ytest, ypred5)
```

```
svm_conf_matrix
```

```
array([[34, 10],
       [ 8, 39]])
```

```
print(svm_acc_score*100,"%")
```

```
80.21978021978022 %
```

We get an accuracy of 80% in SVM

Let us see our model accuracy in Table form

```
model_acc= pd.DataFrame({'Model' : ['Logistic Regression','Decision Tree','Random Forest','K Nearest Neighbor','SVM'],'Accuracy' : [l
```

```
model_acc = model_acc.sort_values(by=['Accuracy'],ascending=False)
```

```
model_acc
```

	Model	Accuracy
0	Logistic Regression	85.714286
3	K Nearest Neighbor	84.615385
4	SVM	80.219780
2	Random Forest	75.824176
1	Decision Tree	70.329670

Let us use one more Techniques known as Adaboost, this is a Boosting technique which uses multiple models for better accuracy.

Adaboost Classifier

Let us first use some random parameters for training the model without Hypertuning.

```
from sklearn.ensemble import AdaBoostClassifier
```

```
adab= AdaBoostClassifier(base_estimator=svm,n_estimators=100,algorithm='SAMME',learning_rate=0.01,random_state=0)
```

```
adab.fit(x_train,encoded_y)
```

```
AdaBoostClassifier(algorithm='SAMME',
                   base_estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                                      class_weight=None, coef0=0.0,
                                      decision_function_shape='ovr', degree=3,
                                      gamma='scale', kernel='rbf', max_iter=-1,
                                      probability=False, random_state=None,
                                      shrinking=True, tol=0.001,
                                      verbose=False),
                   learning_rate=0.01, n_estimators=100, random_state=0)
```

```
ypred6=adab.predict(x_test)
```

```
adab_conf_matrix = confusion_matrix(encoded_ytest,ypred6)
adab_acc_score = accuracy_score(encoded_ytest, ypred6)
```

```
adab_conf_matrix
```

```
array([[ 0, 44],
       [ 0, 47]])
```

```
print(adab_acc_score*100,"%")
```

```
51.64835164835166 %
```

```
adab.score(x_train,encoded_y)
```

```
0.5566037735849056
```

```
adab.score(x_test,encoded_ytest)
```

```
0.5164835164835165
```

As we see our model has performed very poorly with just 50% accuracy

We will use Grid Seach CV for HyperParameter Tuning

Grid Search CV

Let us try Grid Search CV for our top 3 performing Algorithms for HyperParameter tuning

```
from sklearn.model_selection import GridSearchCV
```

```
model_acc
```

	Model	Accuracy
0	Logistic Regression	85.714286
3	K Nearest Neighbor	84.615385
4	SVM	80.219780
2	Random Forest	75.824176
1	Decision Tree	70.329670

Logistic Regression

```
param_grid= {

    'solver': ['newton-cg', 'lbfgs', 'liblinear','sag', 'saga'],
```

```
    'penalty' : ['none', 'l1', 'l2', 'elasticnet'],
    'C' : [100, 10, 1.0, 0.1, 0.01]

}


grid1= GridSearchCV(LogisticRegression(),param_grid)


grid1.fit(x_train,encoded_y)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:536: FitFailedWarning: Estimator fit failed. Th
ValueError: penalty='none' is not supported for the liblinear solver

  FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
```

```
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:1505: UserWarning: Setting penalty='none' will ignor
  "Setting penalty='none' will ignore the C and l1_ratio "
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:536: FitFailedWarning: Estimator fit failed. Th
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got l1 penalty.

  FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:536: FitFailedWarning: Estimator fit failed. Th
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_validation.py:536: FitFailedWarning: Estimator fit failed. Th
ValueError: Solver sag supports only 'l2' or 'none' penalties, got l1 penalty.

  FitFailedWarning)
```

```
grid1.best_params_
```

```
{'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}
```

Let us apply these para in our Model

```
logreg1= LogisticRegression(C=0.01,penalty='l2',solver='liblinear')
```

```
logreg1.fit(x_train,encoded_y)
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
logreg_pred= logreg1.predict(x_test)
```

```
logreg_pred_conf_matrix = confusion_matrix(encoded_ytest,logreg_pred)
logreg_pred_acc_score = accuracy_score(encoded_ytest, logreg_pred)
```

```
logreg_pred_conf_matrix
```

```
array([[33, 11],
       [ 6, 41]])
```

```
print(logreg_pred_acc_score*100,"%")
```

```
81.31868131868131 %
```

We got an accuracy of 81%

KNN

```
n_neighbors = range(1, 21, 2)
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan', 'minkowski']
```

```
grid = dict(n_neighbors=n_neighbors,weights=weights,metric=metric)
```

```
from sklearn.model_selection import RepeatedStratifiedKFold


cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)


grid_search = GridSearchCV(estimator=knn, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)


grid_search.fit(x_train,encoded_y)
```

```
     GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10, random_state=1),
                  error_score=0,
                  estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                                 metric='minkowski',
                                                 metric_params=None, n_jobs=None,
                                                 n_neighbors=12, p=2,
                                                 weights='uniform'),
                  iid='deprecated', n_jobs=-1,
                  param_grid={'metric': ['euclidean', 'manhattan', 'minkowski'],
                              'n_neighbors': range(1, 21, 2),
                              'weights': ['uniform', 'distance']},
                  pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                  scoring='accuracy', verbose=0)
```

```
grid_search.best_params_
```

```
     {'metric': 'manhattan', 'n_neighbors': 11, 'weights': 'distance'}
```

Let's apply

```
knn= KNeighborsClassifier(n_neighbors=12,metric='manhattan',weights='distance')
knn.fit(x_train,encoded_y)
knn_pred= knn.predict(x_test)
```

```
knn_pred_conf_matrix = confusion_matrix(encoded_ytest,knn_pred)
knn_pred_acc_score = accuracy_score(encoded_ytest, knn_pred)
```

```
knn_pred_conf_matrix
```

```
array([[33, 11],
       [ 5, 42]])
```

```
print(knn_pred_acc_score*100,"%")
```

```
82.41758241758241 %
```

We have an Accuracy of 82.5%

SVM

```
kernel = ['poly', 'rbf', 'sigmoid']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
```

```
grid = dict(kernel=kernel,C=C,gamma=gamma)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=svm, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
```

```
grid_search.fit(x_train,encoded_y)
```

```
GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=10, random_state=1),
             error_score=0,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
```

```
                                   decision_function_shape='ovr', degree=3,
                                   gamma='scale', kernel='rbf', max_iter=-1,
                                   probability=False, random_state=None, shrinking=True,
                                   tol=0.001, verbose=False),
                 iid='deprecated', n_jobs=-1,
                 param_grid={'C': [50, 10, 1.0, 0.1, 0.01], 'gamma': ['scale'],
                             'kernel': ['poly', 'rbf', 'sigmoid']},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                 scoring='accuracy', verbose=0)
```

```
grid_search.best_params_
```

```
{'C': 0.1, 'gamma': 'scale', 'kernel': 'sigmoid'}
```

Let us apply these

```
from sklearn.svm import SVC
```

```
svc= SVC(C= 0.1, gamma= 'scale',kernel= 'sigmoid')
```

```
svc.fit(x_train,encoded_y)
```

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
svm_pred= svc.predict(x_test)
```

```
svm_pred_conf_matrix = confusion_matrix(encoded_ytest,svm_pred)
svm_pred_acc_score = accuracy_score(encoded_ytest, svm_pred)
```

```
svm_pred_conf_matrix
```

```
array([[32, 12],
       [ 5, 42]])
```

```
print(svm_pred_acc_score*100,"%")
```

```
81.31868131868131 %
```

Accuracy is 81%

Final Verdict

After comparing all the models the best performing model is :

Logistic Regression with no Hyperparameter tuning

```
logreg= LogisticRegression()
logreg = LogisticRegression()
logreg.fit(x_train, encoded_y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
Y_pred1
```

```
array([0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
```

```
       1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 0])
```

```
lr_conf_matrix
```

```
array([[35,  9],
       [ 4, 43]])
```

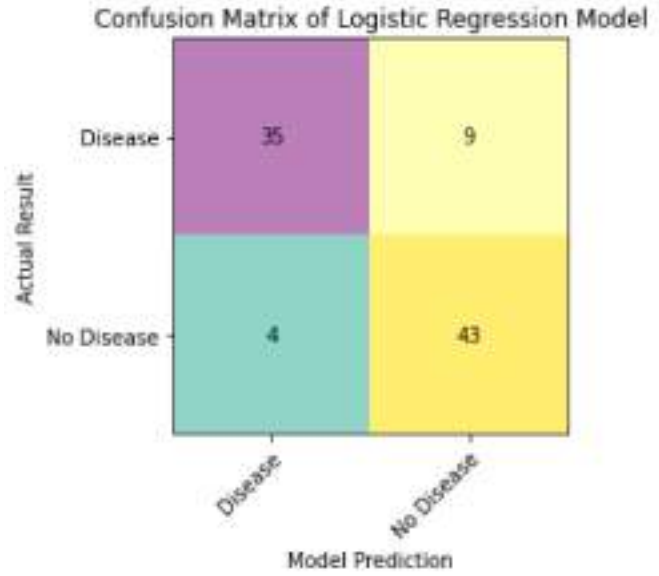
```
print(lr_acc_score*100,"%")
```

```
85.71428571428571 %
```

Let us build a proper confusion matrix for our model

```
# Confusion Matrix of  Model enlarged
options = ["Disease", 'No Disease']

fig, ax = plt.subplots()
im = ax.imshow(lr_conf_matrix, cmap= 'Set3', interpolation='nearest')

# We want to show all ticks...
ax.set_xticks(np.arange(len(options)))
ax.set_yticks(np.arange(len(options)))
# ... and label them with the respective list entries
ax.set_xticklabels(options)
ax.set_yticklabels(options)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(options)):
```

```
    for j in range(len(options)):
        text = ax.text(j, i, lr_conf_matrix[i, j],
                       ha="center", va="center", color="black")

ax.set_title("Confusion Matrix of Logistic Regression Model")
fig.tight_layout()
plt.xlabel('Model Prediction')
plt.ylabel('Actual Result')
plt.show()
print("ACCURACY of our model is ",lr_acc_score*100,"%")
```




```
ACCURACY of our model is  85.71428571428571 %
```

We have succesfully made our model which predicts weather a person is having a risk of Heart Disease or not with 85.7% accuracy

```
import pickle
```

```
pickle.dump(logreg,open('heart.pkl','wb'))
```

Using Auto ML

EVAL ML :

EvalML is an open-source AutoML library written in python that automates a large part of the machine learning process and we can easily evaluate which machine learning pipeline works better for the given set of data.

Installing Eval ML

```
!pip install evalml
```

```
Collecting evalml
  Downloading evalml-0.30.2-py3-none-any.whl (6.3 MB)
     |████████████████████████████████| 6.3 MB 3.8 MB/s
Requirement already satisfied: seaborn>=0.11.1 in /usr/local/lib/python3.7/dist-packages (from evalml) (0.11.1)
Collecting cloudpickle>=1.5.0
  Downloading cloudpickle-1.6.0-py3-none-any.whl (23 kB)
Collecting colorama>=0.4.4
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting category-encoders>=2.2.2
  Downloading category_encoders-2.2.2-py2.py3-none-any.whl (80 kB)
     |████████████████████████████████| 80 kB 6.3 MB/s
Collecting imbalanced-learn>=0.8.0
  Downloading imbalanced_learn-0.8.0-py3-none-any.whl (206 kB)
     |████████████████████████████████| 206 kB 62.6 MB/s
Collecting pandas>=1.2.5
  Downloading pandas-1.3.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.3 MB)
     |████████████████████████████████| 11.3 MB 35.4 MB/s
Collecting pmdarima==1.8.0
  Downloading pmdarima-1.8.0-cp37-cp37m-manylinux1_x86_64.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 40.1 MB/s
Collecting lightgbm>=2.3.1
  Downloading lightgbm-3.2.1-py3-none-manylinux1_x86_64.whl (2.0 MB)
     |████████████████████████████████| 2.0 MB 48.1 MB/s
Collecting requirements-parser>=0.2.0
  Downloading requirements-parser-0.2.0.tar.gz (6.3 kB)
Collecting matplotlib>=3.3.3
  Downloading matplotlib-3.4.3-cp37-cp37m-manylinux1_x86_64.whl (10.3 MB)
     |████████████████████████████████| 10.3 MB 31.2 MB/s
Requirement already satisfied: click>=7.1.2 in /usr/local/lib/python3.7/dist-packages (from evalml) (7.1.2)
Collecting networkx<2.6,>=2.5
  Downloading networkx-2.5.1-py3-none-any.whl (1.6 MB)
     |████████████████████████████████| 1.6 MB 58.3 MB/s
Collecting sktime>=0.7.0
  Downloading sktime-0.7.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (5.8 MB)
     |████████████████████████████████| 5.8 MB 26.7 MB/s
Requirement already satisfied: ipywidgets>=7.5 in /usr/local/lib/python3.7/dist-packages (from evalml) (7.6.3)
Collecting graphviz>=0.13
  Downloading graphviz-0.17-py3-none-any.whl (18 kB)
Collecting plotly>=5.0.0
  Downloading plotly-5.2.1-py2.py3-none-any.whl (21.8 MB)
     |████████████████████████████████| 21.8 MB 1.3 MB/s
```

```
Collecting nlp-primitives>=1.1.0
  Downloading nlp_primitives-1.1.0-py3-none-any.whl (18.0 MB)
     |████████████████████████████████| 18.0 MB 116 kB/s
Collecting scipy>=1.5.0
  Downloading scipy-1.7.1-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.whl (28.5 MB)
     |████████████████████████████████| 28.5 MB 50 kB/s
Collecting woodwork==0.5.1
  Downloading woodwork-0.5.1-py3-none-any.whl (135 kB)
     |████████████████████████████████| 135 kB 62.8 MB/s
Collecting shap>=0.36.0
  Downloading shap-0.39.0.tar.gz (356 kB)
     |████████████████████████████████| 356 kB 59.2 MB/s
Collecting xgboost>=1.4.2
  Downloading xgboost-1.4.2-py3-none-manylinux2010_x86_64.whl (166.7 MB)
```

Let us load our DataSet.

```
Requirement already satisfied: dask>=2.12.0 in /usr/local/lib/python3.7/dist-packages (from evalml) (2.12.0)
```

```
df= pd.read_csv("/content/drive/MyDrive/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

Let us split our Data Set into Dependent i.e our Targer variable and independent variable

```
x= df.iloc[:,:-1]
```

```
x
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	caa
0	0.952197	0.681005	1.973123	0.763956	-0.256334	2.394438	-1.005832	0.015443	-0.696631	-0.714429
1	-1.915313	0.681005	1.002577	-0.092738	0.072199	-0.417635	0.898962	1.633471	-0.696631	-0.714429
2	-1.474158	-1.468418	0.032031	-0.092738	-0.816773	-0.417635	-1.005832	0.977514	-0.696631	-0.714429
3	0.180175	0.681005	0.032031	-0.663867	-0.198357	-0.417635	0.898962	1.239897	-0.696631	-0.714429
4	0.290464	-1.468418	-0.938515	-0.663867	2.082050	-0.417635	0.898962	0.583939	1.435481	-0.714429
...	...	...	...	...	...	...	...	...	...	...
298	0.290464	-1.468418	-0.938515	0.478391	-0.101730	-0.417635	0.898962	-1.165281	1.435481	-0.714429
299	-1.033002	0.681005	1.973123	-1.234996	0.342756	-0.417635	0.898962	-0.771706	-0.696631	-0.714429
300	1.503641	0.681005	-0.938515	0.706843	-1.029353	2.394438	0.898962	-0.378132	-0.696631	1.244593
301	0.290464	0.681005	-0.938515	-0.092738	-2.227533	-0.417635	0.898962	-1.515125	1.435481	0.265082
302	0.290464	-1.468418	0.032031	-0.092738	-0.198357	-0.417635	-1.005832	1.064975	-0.696631	0.265082

303 rows × 10 columns

```
y= df.iloc[:,-1:]
y= lbl.fit_transform(y)
y
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:63: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav

array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
```

```
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
|████████████████████████████████| 679 kB 49.8 MB/s
```

Importing Eval ML Library

```
Downloading distributed-2021.1.1-py3-none-any.whl (672 kB)
```

```
import evalml
```

```
|████████████████████████████████| 671 kB 46.9 MB/s
```

Eval ML Library will do all the pre processing techniques for us and split the data for us

```
Downloading distributed-2.30.1-py3-none-any.whl (656 kB)
```

```
X_train, X_test, y_train, y_test = evalml.preprocessing.split_data(x, y, problem_type='binary')
```

```
Requirement already satisfied: nbformat>=4.2.0 in /usr/local/lib/python3.7/dist-packages (from ipywidgets>=7.5->evalml) (5.1.3)
```

There are different problem type parameters in Eval ML, we have a Binary type problem here, that's why we are using Binary as a input

```
Requirement already satisfied: traitlets>=4.3.1 in /usr/local/lib/python3.7/dist-packages (from ipywidgets>=7.5->evalml) (5.0.5
```

```
evalml.problem_types.ProblemTypes.all_problem_types
```

```
[<ProblemTypes.BINARY: 'binary'>,
 <ProblemTypes.MULTICLASS: 'multiclass'>,
 <ProblemTypes.REGRESSION: 'regression'>,
 <ProblemTypes.TIME_SERIES_REGRESSION: 'time series regression'>,
 <ProblemTypes.TIME_SERIES_BINARY: 'time series binary'>,
 <ProblemTypes.TIME_SERIES_MULTICLASS: 'time series multiclass'>]
```

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.3.3->evalml) (0.10.0)
```

Running the Auto ML to select best Algorithm

```
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.3.3->evalml) (7.1.2)
```

```
from evalml.automl import AutoMLSearch
automl = AutoMLSearch(X_train=X_train, y_train=y_train, problem_type='binary')
automl.search()
```

```
Using default limit of max_batches=1.

Generating pipelines to search over...
8 pipelines ready for search.

*****************************
* Beginning pipeline search *
*****************************

Optimizing for Log Loss Binary.
Lower score is better.

Using SequentialEngine to train and score pipelines.
Searching up to 1 batches for a total of 9 pipelines.
Allowed model families: random_forest, extra_trees, decision_tree, lightgbm, linear_model, catboost, xgboost

Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
Mode Baseline Binary Classification Pipeline:
        Starting cross validation
        Finished cross validation - mean Log Loss Binary: 15.699
```

As we see from the above output thge Auto ML Classifier has given us the best fit Algorithm which is Extra Trees Classifier with Imputer We can also commpare the rest of the models

```
Elastic Net Classifier w/ Imputer + Standard Scaler:
```

```
automl.rankings
```

	id	pipeline_name	search_order	mean_cv_score	standard_deviation_cv_score	validation_score	percent_better_than_baseline
0	7	Extra Trees Classifier w/ Imputer	7	0.441986	0.025672	0.456773	97.184590
1	3	Random Forest Classifier w/ Imputer	3	0.451508	0.021426	0.466939	97.123934
2	5	Logistic Regression Classifier w/ Imputer + St...	5	0.488080	0.031375	0.520345	96.890973
3	1	Elastic Net Classifier w/ Imputer +	1	0.488306	0.030127	0.519350	96.889536

```
automl.best_pipeline
```

```
pipeline = BinaryClassificationPipeline(component_graph={'Imputer': ['Imputer', 'X', 'y'], 'Extra Trees Classifier': ['Extra
Trees Classifier', 'Imputer.x', 'y']}, parameters={'Imputer':{'categorical_impute_strategy': 'most_frequent',
'numeric_impute_strategy': 'mean', 'categorical_fill_value': None, 'numeric_fill_value': None}, 'Extra Trees Classifier':
{'n_estimators': 100, 'max_features': 'auto', 'max_depth': 6, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0,
'n_jobs': -1}}, random_seed=0)
```

```
best_pipeline=automl.best_pipeline
```

```
Successfully uninstalled lightgbm-2.2.3
```

We can have a Detailed description of our Best Selected Model

```
Uninstalling imbalanced-learn-0.4.3:
```

```
automl.describe_pipeline(automl.rankings.iloc[0]["id"])
```

```
**********************************
* Extra Trees Classifier w/ Imputer *
**********************************
```

```
Problem Type: binary
Model Family: Extra Trees

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
         * numeric_fill_value : None
2. Extra Trees Classifier
         * n_estimators : 100
         * max_features : auto
         * max_depth : 6
         * min_samples_split : 2
         * min_weight_fraction_leaf : 0.0
         * n_jobs : -1

Training
========
Training for binary problems.
Total training time (including CV): 2.1 seconds

Cross Validation
----------------
             Log Loss Binary  MCC Binary  Gini   AUC  Precision    F1  Balanced Accuracy Binary  Accuracy Binary # Training # V
0                      0.457       0.584 0.785 0.893      0.857 0.738                     0.779            0.790        161
1                      0.412       0.676 0.806 0.903      0.833 0.822                     0.837            0.840        161
2                      0.457       0.591 0.732 0.866      0.913 0.712                     0.769            0.787        162
mean                   0.442       0.617 0.774 0.887      0.868 0.757                     0.795            0.806          -
std                    0.026       0.051 0.038 0.019      0.041 0.057                     0.037            0.029          -
coef of var            0.058       0.083 0.049 0.021      0.047 0.076                     0.046            0.036          -
```

```
best_pipeline.score(X_test, y_test, objectives=["auc","f1","Precision","Recall"])

OrderedDict([('AUC', 0.8852813852813852),
             ('F1', 0.7812499999999999),
```

```
                ('Precision', 0.8064516129032258),
                ('Recall', 0.7575757575757576)])
```

Now if we want to build our Model for a specific objective we can do that

```
automl_auc = AutoMLSearch(X_train=X_train, y_train=y_train,
                          problem_type='binary',
                          objective='auc',
                          additional_objectives=['f1', 'precision'],
                          max_batches=1,
                          optimize_thresholds=True)

automl_auc.search()
```

```
    Generating pipelines to search over...
    8 pipelines ready for search.

    ****************************
    * Beginning pipeline search *
    ****************************

    Optimizing for AUC.
    Greater score is better.

    Using SequentialEngine to train and score pipelines.
    Searching up to 1 batches for a total of 9 pipelines.
    Allowed model families: random_forest, extra_trees, decision_tree, lightgbm, linear_model, catboost, xgboost

    Evaluating Baseline Pipeline: Mode Baseline Binary Classification Pipeline
    Mode Baseline Binary Classification Pipeline:
            Starting cross validation
            Finished cross validation - mean AUC: 0.500

    ****************************
    * Evaluating Batch Number 1 *
    ****************************

    Elastic Net Classifier w/ Imputer + Standard Scaler:
            Starting cross validation
            Finished cross validation - mean AUC: 0.847
    Decision Tree Classifier w/ Imputer:
            Starting cross validation
            Finished cross validation - mean AUC: 0.723
    Random Forest Classifier w/ Imputer:
            Starting cross validation
            Finished cross validation - mean AUC: 0.874
    LightGBM Classifier w/ Imputer:
```

```
automl_auc.rankings
```

	id	pipeline_name	search_order	mean_cv_score	standard_deviation_cv_score	validation_score	percent_better_than_baseline
0	7	Extra Trees Classifier w/ Imputer	7	0.887205	0.018958	0.892506	38.720539
1	3	Random Forest Classifier w/ Imputer	3	0.873658	0.013643	0.869779	37.365775
2	6	XGBoost Classifier w/ Imputer	6	0.849162	0.027477	0.818182	34.916166
3	5	Logistic Regression Classifier w/ Imputer + St...	5	0.848007	0.017890	0.842752	34.800710

Best pipeline: Extra Trees Classifier w/ Imputer

```
automl_auc.describe_pipeline(automl_auc.rankings.iloc[0]["id"])
```

```
***********************************
* Extra Trees Classifier w/ Imputer *
***********************************

Problem Type: binary
Model Family: Extra Trees

Pipeline Steps
==============
1. Imputer
         * categorical_impute_strategy : most_frequent
         * numeric_impute_strategy : mean
         * categorical_fill_value : None
```

```
            * numeric_fill_value : None
     2. Extra Trees Classifier
            * n_estimators : 100
            * max_features : auto
            * max_depth : 6
            * min_samples_split : 2
            * min_weight_fraction_leaf : 0.0
            * n_jobs : -1

     Training
     ========
     Training for binary problems.
     Total training time (including CV): 2.1 seconds

     Cross Validation
     ----------------
                   AUC    F1  Precision # Training # Validation
     0           0.893 0.738      0.857        161           81
     1           0.903 0.822      0.833        161           81
     2           0.866 0.712      0.913        162           80
     mean        0.887 0.757      0.868          -            -
     std         0.019 0.057      0.041          -            -
     coef of var 0.021 0.076      0.047          -            -


best_pipeline_auc = automl_auc.best_pipeline




# get the score on holdout data
best_pipeline_auc.score(X_test, y_test,  objectives=["auc"])

     OrderedDict([('AUC', 0.8852813852813852)])
```

We got an 88.5 % AUC Score which is the highest of all

Save the model

```
best_pipeline.save("model.pkl")
```

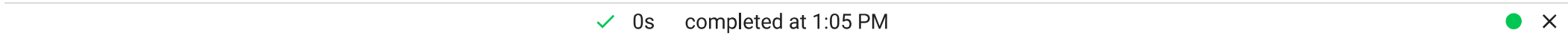
Loading our Model

```
final_model=automl.load('model.pkl')
```

```
final_model.predict_proba(X_test)
```

	0	1
0	0.468324	0.531676
1	0.093848	0.906152
2	0.383646	0.616354
3	0.107272	0.892728
4	0.141027	0.858973
...	...	...
56	0.268136	0.731864
57	0.846652	0.153348
58	0.861607	0.138393
59	0.739515	0.260485
60	0.878833	0.121167

61 rows × 2 columns


0s
completed at 1:05 PM