```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

```
df = pd.read_csv('processed.cleveland.data',names=  ['age','sex','chest_pain','blood pressure','serum_cholestoral','fasting_blood_sugar',\
              'electrocardiographic','max_heart_rate','induced_angina','ST_depression','slope','vessels','thal','diagnosis'])
```

```
df
```

	age	sex	chest_pain	blood pressure	serum_cholestoral	fasting_blood_sugar	electrocardiographic	max
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	
...	...	...	...	...	...	...	...	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	
302	38.0	1.0	3.0	138.0	175.0	0.0	0.0	

303 rows × 14 columns

```
df.describe()
```

	age	sex	chest_pain	blood pressure	serum_cholestoral	fasting_blood_sugar	electroca
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	
mean	54.438944	0.679868	3.158416	131.689769	246.693069	0.148515	
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	
50%	56.000000	1.000000	3.000000	130.000000	241.000000	0.000000	
75%	61.000000	1.000000	4.000000	140.000000	275.000000	0.000000	
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	

```
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    float64
 1   sex                   303 non-null    float64
 2   chest_pain            303 non-null    float64
 3   blood pressure        303 non-null    float64
 4   serum_cholestoral     303 non-null    float64
 5   fasting_blood_sugar   303 non-null    float64
 6   electrocardiographic  303 non-null    float64
 7   max_heart_rate        303 non-null    float64
 8   induced_angina        303 non-null    float64
 9   ST_depression         303 non-null    float64
 10  slope                 303 non-null    float64
 11  vessels               303 non-null    object 
 12  thal                  303 non-null    object 
 13  diagnosis             303 non-null    int64  
dtypes: float64(11), int64(1), object(2)
memory usage: 33.3+ KB
```

checking for null

```
# checking for null
df.isna().sum()
```

```
age                   0
sex                   0
chest_pain            0
blood pressure        0
serum_cholestoral     0
fasting_blood_sugar   0
electrocardiographic  0
max_heart_rate        0
induced_angina        0
ST_depression         0
slope                 0
vessels               0
thal                  0
diagnosis             0
dtype: int64
```

vessel and thal variables have some missing values shown as '?'

```
## vessel and thal variables have some missing values shown as '?'
```

```
(df == '?').sum()
```

```
age                    0
sex                    0
chest_pain             0
blood pressure         0
serum_cholestoral      0
fasting_blood_sugar    0
electrocardiographic   0
max_heart_rate         0
induced_angina         0
ST_depression          0
slope                  0
vessels                4
thal                   2
diagnosis              0
dtype: int64
```

```
df['vessels'].value_counts()
```

```
0.0    176
1.0     65
2.0     38
3.0     20
?        4
Name: vessels, dtype: int64
```

```
df['thal'].value_counts()
```

```
3.0    166
7.0    117
6.0     18
?        2
Name: thal, dtype: int64
```

removing the '?' misssing values

```
df[df['vessels'] == '?']['vessels']
```

```
166    ?
192    ?
287    ?
302    ?
Name: vessels, dtype: object
```

```
df[df['thal'] == '?']['thal']
```

```
87     ?
266    ?
Name: thal, dtype: object
```

```
missing_vessels = df[df['vessels'] == '?']['vessels'].index
missing_thals = df[df['thal'] == '?']['thal'].index
```

```
df.drop(missing_vessels,inplace=True)
df.drop(missing_thals,inplace=True)
```

```
## After removing 6 missing values
df
```

	age	sex	chest_pain	blood pressure	serum_cholestoral	fasting_blood_sugar	electrocardiographic	max
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	
...	...	...	...	...	...	...	...	
297	57.0	0.0	4.0	140.0	241.0	0.0	0.0	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	

297 rows × 14 columns

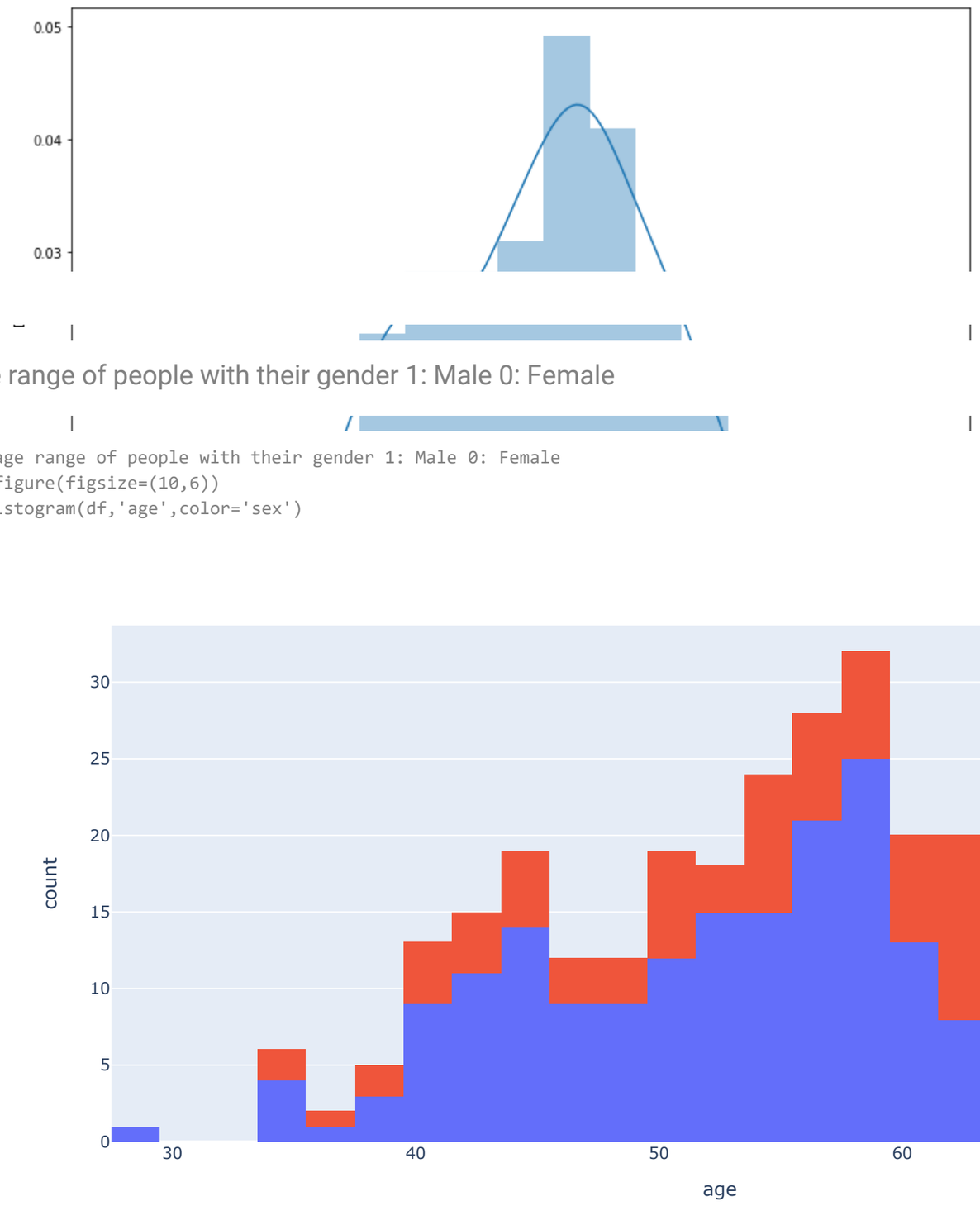
Checking for age distribution

```
## Checking for age distribution
plt.figure(figsize=(12,8))
sns.distplot(df['age'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to

<matplotlib.axes._subplots.AxesSubplot at 0x7f9865d68850>
```



Age range of people with their gender 1: Male 0: Female

```
### age range of people with their gender 1: Male 0: Female
plt.figure(figsize=(10,6))
px.histogram(df,'age',color='sex')
```



```
<Figure size 720x432 with 0 Axes>
```

Target distribution

```
### Target distribution
df['diagnosis'].value_counts()
```

```
0    160
1     54
3     35
2     35
4     13
Name: diagnosis, dtype: int64
```

```
# ''' As given in the Dataset :The "goal" field refers to the presence of heart disease
#      in the patient.  It is integer valued from 0 (no presence) to 4.
#      Experiments with the Cleveland database have concentrated on simply
#      attempting to distinguish presence (values 1,2,3,4) from absence (value
#      0).  '''
```

```
df['Target'] = df['diagnosis'].apply(lambda x : 1 if x >= 1 else 0)
```

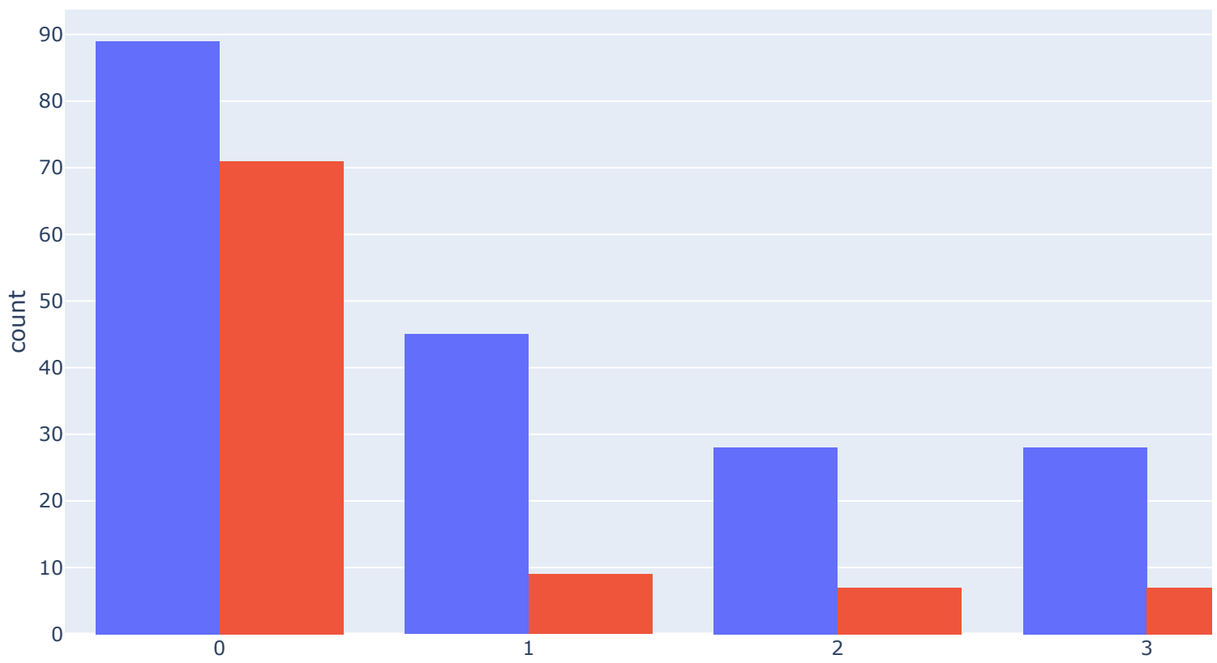
```
df
```

	age	sex	chest_pain	blood pressure	serum_cholestoral	fasting_blood_sugar	electrocardiographic	max
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	
...	...	...	...	...	...	...	...	
297	57.0	0.0	4.0	140.0	241.0	0.0	0.0	
298	45.0	1.0	1.0	110.0	264.0	0.0	0.0	
299	68.0	1.0	4.0	144.0	193.0	1.0	0.0	
300	57.0	1.0	4.0	130.0	131.0	0.0	0.0	
301	57.0	0.0	2.0	130.0	236.0	0.0	2.0	

297 rows × 15 columns

Distribution of Diagnosis to sex

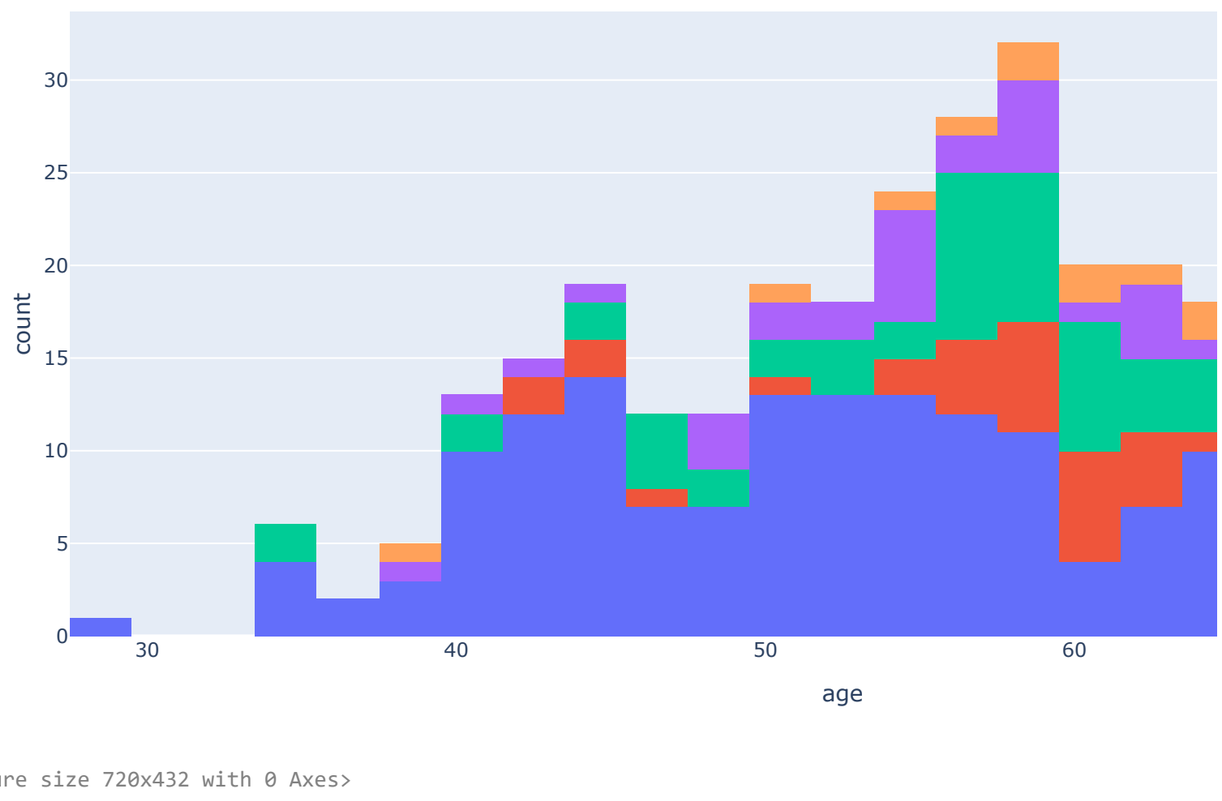
```
#### Distribution of Diagnosis to sex
plt.figure(figsize=(10,6))
px.histogram(df,x='diagnosis',color='sex',barmode="group")
```




```
<Figure size 720x432 with 0 Axes>
```

Severity of Diagnosis according to Age

```
plt.figure(figsize=(10,6))
px.histogram(df,'age',color='diagnosis')
```




```
<Figure size 720x432 with 0 Axes>
```

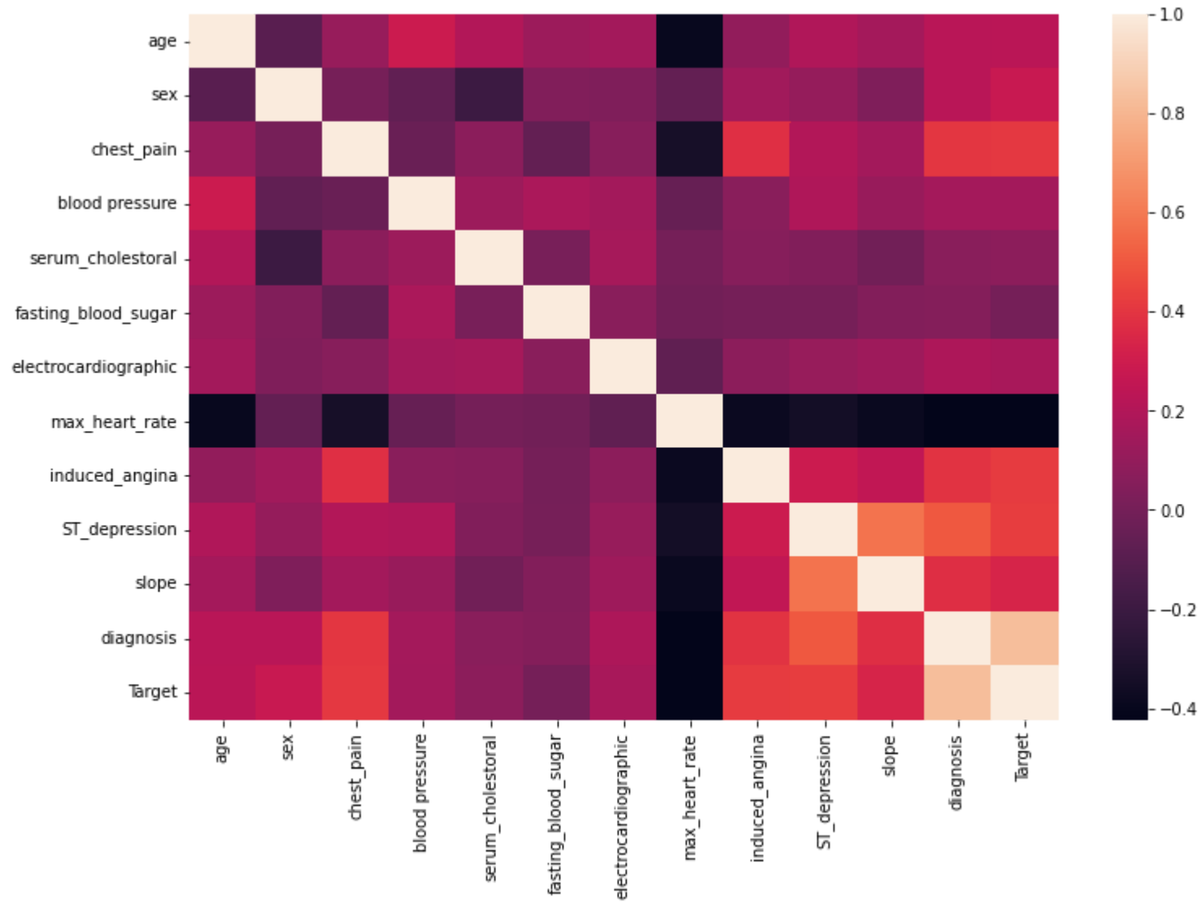
Finding correlation of variables

```
### Finding correlation of variables
df.corr()
```

	age	sex	chest_pain	blood pressure	serum_cholestoral	fasting_blood_sugar
age	1.000000	-0.092399	0.110471	0.290476	0.202644	0.132062
sex	-0.092399	1.000000	0.008908	-0.066340	-0.198089	0.038850
chest_pain	0.110471	0.008908	1.000000	-0.036980	0.072088	-0.057663
blood pressure	0.290476	-0.066340	-0.036980	1.000000	0.131536	0.180860
serum_cholestoral	0.202644	-0.198089	0.072088	0.131536	1.000000	0.012708
fasting_blood_sugar	0.132062	0.038850	-0.057663	0.180860	0.012708	1.000000
electrocardiographic	0.149917	0.033897	0.063905	0.149242	0.165046	0.068831
max_heart_rate	-0.394563	-0.060496	-0.339308	-0.049108	-0.000075	-0.007842
induced_angina	0.096489	0.143581	0.377525	0.066691	0.059339	-0.000893
ST_depression	0.197123	0.106567	0.203244	0.191243	0.038596	0.008311
slope	0.159405	0.033345	0.151079	0.121172	-0.009215	0.047819
diagnosis	0.222156	0.226797	0.404248	0.159620	0.066448	0.049040
Target	0.227075	0.278467	0.408945	0.153490	0.080285	0.003167

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f984fffbc50>
```



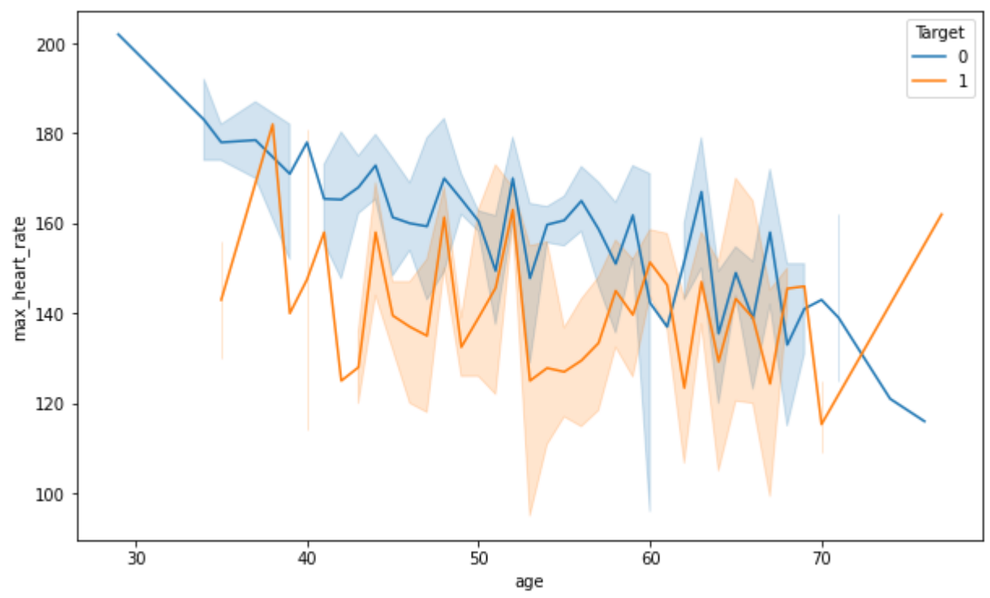
Lineplot of age vs max heart rate

```
plt.figure(figsize=(10,6))
sns.lineplot(df['age'],df['max_heart_rate'],hue=df['Target'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argume

<matplotlib.axes._subplots.AxesSubplot at 0x7f98500443d0>
```




```
df['diagnosis']
```

```
0      0
1      2
2      1
3      0
4      0
      ..
297    1
298    1
299    2
300    3
301    1
Name: diagnosis, Length: 297, dtype: int64
```

Train test split and Feature Scaling

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


X= df.drop(['diagnosis','Target'],axis=1)
y = df['Target']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


sc = StandardScaler()
```

Model Classification

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,log_loss
```

Decision Tree

```
### Taking average of 10 training examples
dt_avg = []
for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    clf1 = DecisionTreeClassifier()
    dt_model = clf1.fit(X_train,y_train)
    dt_pred = dt_model.predict(X_test)
    dt_avg.append(accuracy_score(y_test,dt_pred))

print('Average Decision Tree Test accuracy:',sum(dt_avg)/10)
```

```
Average Decision Tree Test accuracy: 0.75
```

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,log_loss
print(classification_report(y_test,dt_pred))

print(confusion_matrix(y_test,dt_pred))
```

```
                  precision    recall  f1-score   support

               0       0.78      0.81      0.79        36
               1       0.70      0.67      0.68        24

        accuracy                           0.75        60
       macro avg       0.74      0.74      0.74        60
    weighted avg       0.75      0.75      0.75        60

    [[29  7]
     [ 8 16]]
```

```
print('Log loss for Decision Tree model:',log_loss(y_test,dt_pred))
```

```
Log loss for Decision Tree model: 8.634787385094524
```

Logistic Regression

```
### Taking average of 10 training examples
lr_avg = []
for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    clf2 = LogisticRegression(multi_class= "multinomial", solver='newton-cg')
    lr_model = clf2.fit(X_train,y_train)
    lr_pred = lr_model.predict(X_test)
    lr_avg.append(accuracy_score(y_test,lr_pred))

print('Average Logistic Regression Test accuracy:',sum(lr_avg)/10)
```

```
Average Logistic Regression Test accuracy: 0.845
```

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,log_loss
print(classification_report(y_test,lr_pred))

print(confusion_matrix(y_test,lr_pred))
```

```
                  precision    recall  f1-score   support

               0       0.94      0.94      0.94        32
               1       0.93      0.93      0.93        28

        accuracy                           0.93        60
       macro avg       0.93      0.93      0.93        60
    weighted avg       0.93      0.93      0.93        60

    [[30  2]
     [ 2 26]]
```

```
print('Log loss for logistic Regression model:',log_loss(y_test,lr_pred))
```

```
Log loss for logistic Regression model: 2.3026117462417193
```

Random Forest classifier

```
### Taking average of 10 training examples
rf_avg = []
for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    clf3 = RandomForestClassifier()
    rf_model = clf3.fit(X_train,y_train)
    rf_pred = rf_model.predict(X_test)
    rf_avg.append(accuracy_score(y_test,rf_pred))

print('Average Random Forest Test accuracy:',sum(rf_avg)/10)
```

```
Average Random Forest Test accuracy: 0.8133333333333332
```

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,log_loss
print(classification_report(y_test,rf_pred))

print(confusion_matrix(y_test,rf_pred))
```

```
                  precision    recall  f1-score   support

               0       0.78      0.89      0.83        36
```

```
                1       0.79      0.62      0.70        24

         accuracy                           0.78        60
        macro avg       0.78      0.76      0.76        60
     weighted avg       0.78      0.78      0.78        60

     [[32  4]
      [ 9 15]]
```

```
print('Log loss for Random Forest model:',log_loss(y_test,rf_pred))
```

```
     Log loss for Random Forest model: 7.483454858725995
```

KNN Classifier

```
### Taking average of 10 training examples
knn_avg = []
for i in range(10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
    clf4 = KNeighborsClassifier()
    knn_model = clf4.fit(X_train,y_train)
    knn_pred = knn_model.predict(X_test)
    knn_avg.append(accuracy_score(y_test,knn_pred))
```

```
print('Average K Nearest Neighbors Test accuracy:',sum(knn_avg)/10)
```

```
     Average K Nearest Neighbors Test accuracy: 0.8283333333333334
```

```
print('Log loss for KNN model:',log_loss(y_test,knn_pred))
```

```
     Log loss for KNN model: 5.18086976573195
```

Neural network model

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
clf5 = MLPClassifier(solver='lbfgs', alpha=1e-08,hidden_layer_sizes=(60,),activation='logistic', random_state=(2),)
nn_model = clf5.fit(X_train,y_train)
nn_pred = nn_model.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix,accuracy_score,log_loss
print(classification_report(y_test,nn_pred))

print(confusion_matrix(y_test,nn_pred))
```

```
                  precision    recall  f1-score   support

               0       0.76      0.68      0.72        28
               1       0.74      0.81      0.78        32

         accuracy                           0.75        60
        macro avg       0.75      0.75      0.75        60
     weighted avg       0.75      0.75      0.75        60

     [[19  9]
      [ 6 26]]
```

```
print('Log loss for NN model:',log_loss(y_test,nn_pred))
```

```
     Log loss for NN model: 8.6348140383422
```

Lowest log loss calculated for Logistic Regression model is 2.3026117462417193