In [1]:

```
import pandas as pd
```

In [2]:

```
import numpy as np
```

In [98]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [7]:

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

In [9]:

```
df.head()
```

Out[9]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Exer
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

1. Age: age of the patient [years]
2. Sex: sex of the patient [M: Male, F: Female]
3. ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
4. RestingBP: resting blood pressure [mm Hg]
5. Cholesterol: serum cholesterol [mm/dl]
6. FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
7. RestingECG: resting electrocardiogram results [Normal: Normal, ST: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV), LVH: showing probable or definite left ventricular hypertrophy by Estes' criteria]
8. MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
9. ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
10. Oldpeak: oldpeak = ST [Numeric value measured in depression]
11. ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
12. HeartDisease: output class [1: heart disease, 0: Normal]

In [11]:

```
#Basic info to see the type of info we have
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             918 non-null    int64  
 1   Sex             918 non-null    object 
 2   ChestPainType   918 non-null    object 
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    object 
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    object 
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    object 
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```

In [16]:

```
#Doing a describe operaration 0on our dataset to see the range and width of the data we
df.describe()
```

Out[16]:

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisea
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.0000
mean	53.510893	132.396514	198.799564	0.233115	136.809368	0.887364	0.5533
std	9.432617	18.514154	109.384145	0.423046	25.460334	1.066570	0.4974
min	28.000000	0.000000	0.000000	0.000000	60.000000	-2.600000	0.0000
25%	47.000000	120.000000	173.250000	0.000000	120.000000	0.000000	0.0000
50%	54.000000	130.000000	223.000000	0.000000	138.000000	0.600000	1.0000
75%	60.000000	140.000000	267.000000	0.000000	156.000000	1.500000	1.0000
max	77.000000	200.000000	603.000000	1.000000	202.000000	6.200000	1.0000

In [23]:

```
#Checking for any null values: -
for column in df.columns:
  print(f"{column} :: {len(df[df[column].isnull() == True][column])}")
```

```
Age :: 0
Sex :: 0
ChestPainType :: 0
RestingBP :: 0
Cholesterol :: 0
FastingBS :: 0
RestingECG :: 0
MaxHR :: 0
ExerciseAngina :: 0
Oldpeak :: 0
ST_Slope :: 0
HeartDisease :: 0
```

In [45]:

```
#Checking the data for any inconsistencies: -
for column in df.columns:
  print(f"COLUMN : {column}")
  print(df[column].value_counts())
  print("########################################")
```

```
COLUMN : Age
54    51
58    42
55    41
56    38
57    38
52    36
51    35
59    35
62    35
53    33
60    32
48    31
61    31
63    30
50    25
46    24
41    24
43    24
```

#There are no inconsistencies in data

In [46]:

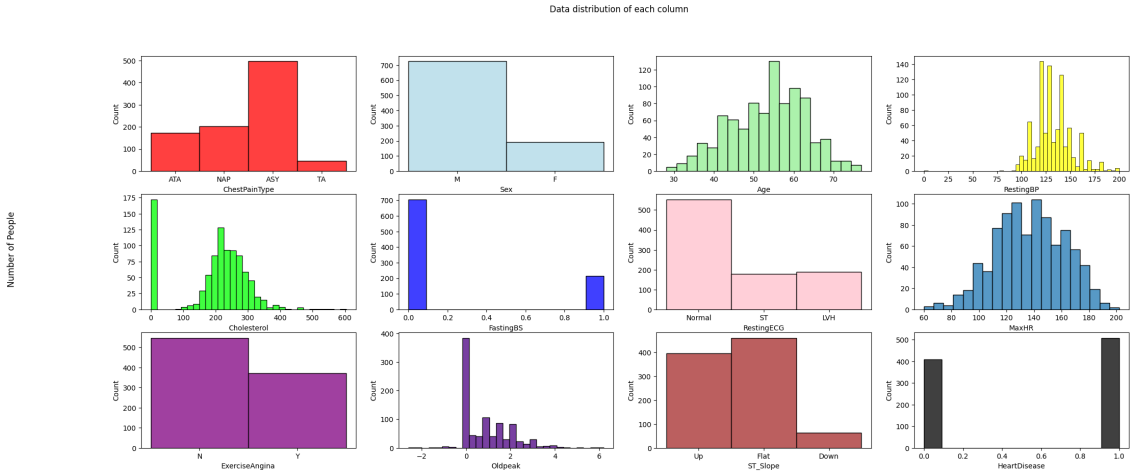
```
df.columns
```

Out[46]:

```
Index(['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'Fastin
gBS',
       'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST_Slope',
       'HeartDisease'],
      dtype='object')
```

In [105]:

```
fig, axs = plt.subplots(3,4,figsize=(25,10))
fig.supylabel('Number of People')
fig.suptitle('Data distribution of each column')
sns.histplot( df['ChestPainType'],ax = axs[0,0],color='red')
sns.histplot( df['Sex'],ax = axs[0,1],color='lightblue')
sns.histplot( df['Age'],ax = axs[0,2],color='lightgreen')
sns.histplot( df['RestingBP'],ax = axs[0,3],color='yellow')
sns.histplot( df['Cholesterol'],ax = axs[1,0],color='lime')
sns.histplot( df['FastingBS'],ax = axs[1,1],color='blue')
sns.histplot( df['RestingECG'],ax = axs[1,2],color='pink')
sns.histplot( df['MaxHR'],ax = axs[1,3])
sns.histplot( df['ExerciseAngina'],ax = axs[2,0],color='purple')
sns.histplot( df['Oldpeak'],ax = axs[2,1],color='indigo')
sns.histplot( df['ST_Slope'],ax = axs[2,2],color='brown')
sns.histplot( df['HeartDisease'],ax = axs[2,3],color='black')
plt.show()
```



In [146]:

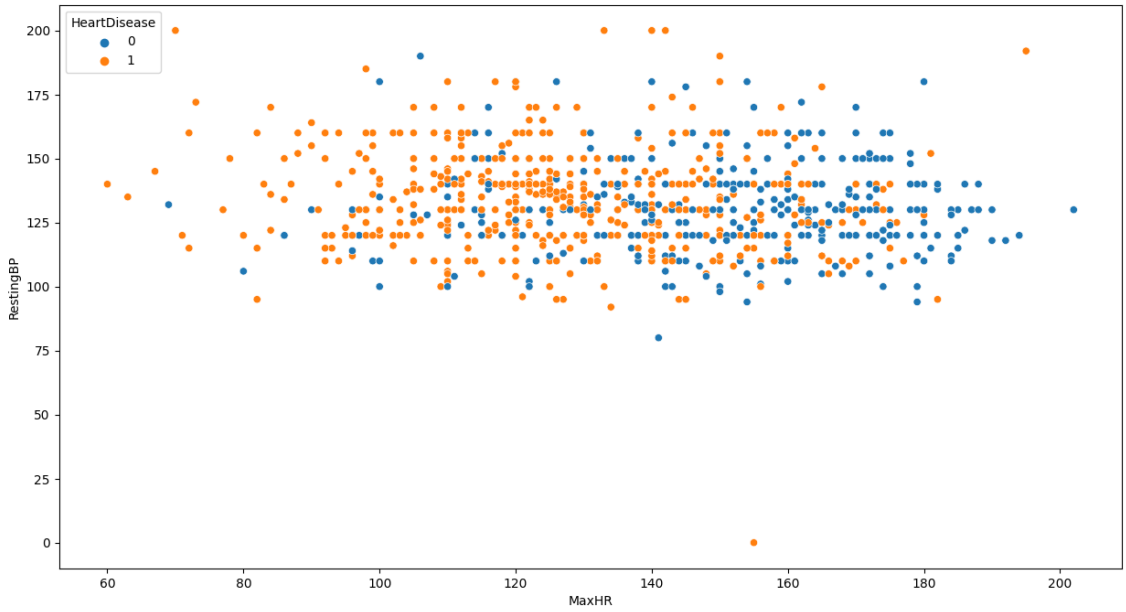
```
#Here we can see there are some 0 values in cholestrol, I am going to impute these valu
MEDIAN = df['Cholesterol'].median()
for i in range(len(df['Cholesterol'])):
  if df['Cholesterol'][i] == 0:
    df['Cholesterol'][i] = MEDIAN
```

```
<ipython-input-146-a09b22e15527>: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-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  df['Cholesterol'][i] = MEDIAN
```

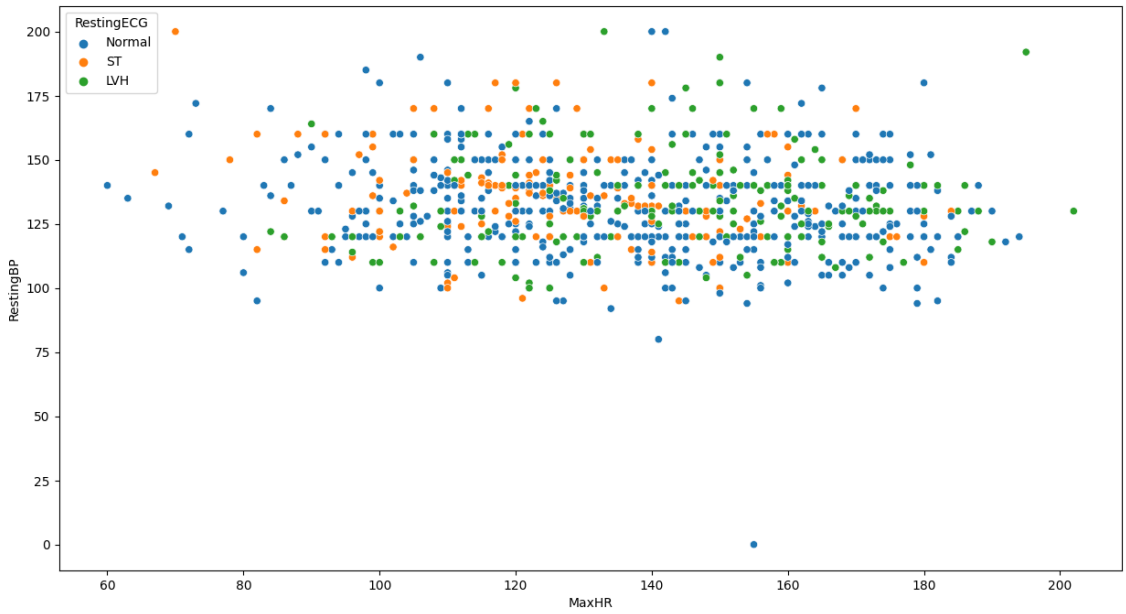
In [148]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df, x='MaxHR', y = 'RestingBP',hue=df['HeartDisease'])
plt.show()
```



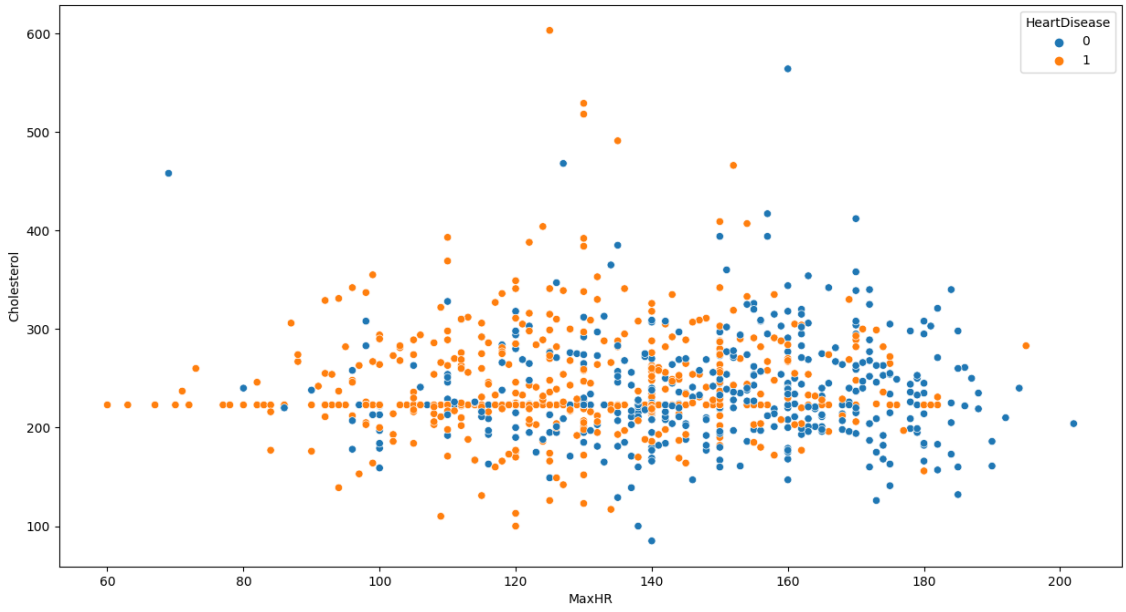
In [149]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df, x='MaxHR', y = 'RestingBP',hue=df['RestingECG'])
plt.show()
```



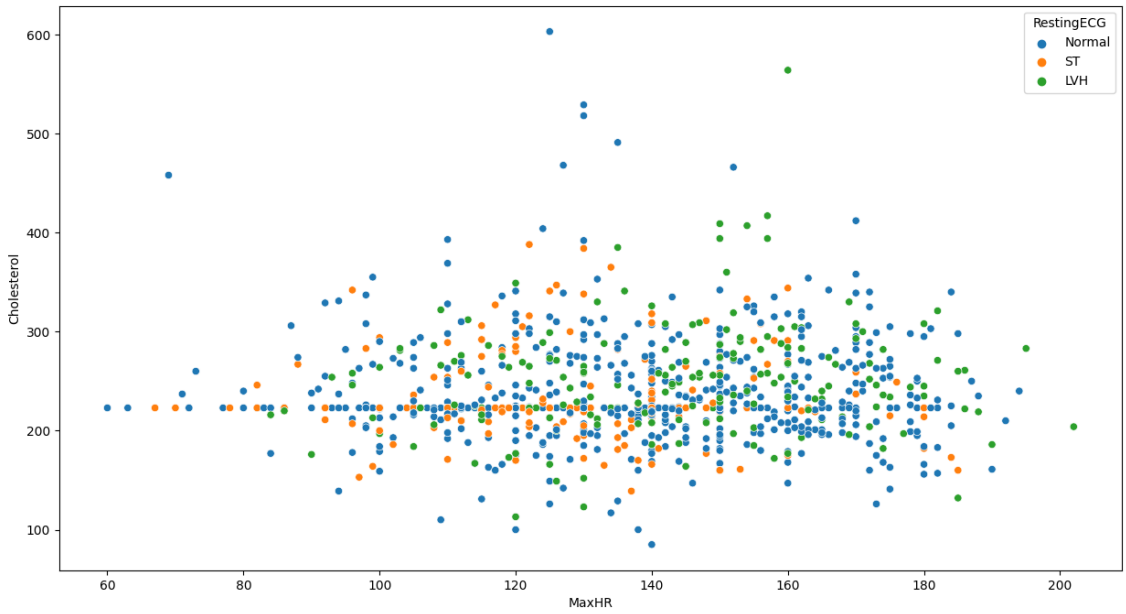
In [150]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df, x='MaxHR', y = 'Cholesterol',hue=df['HeartDisease'])
plt.show()
```



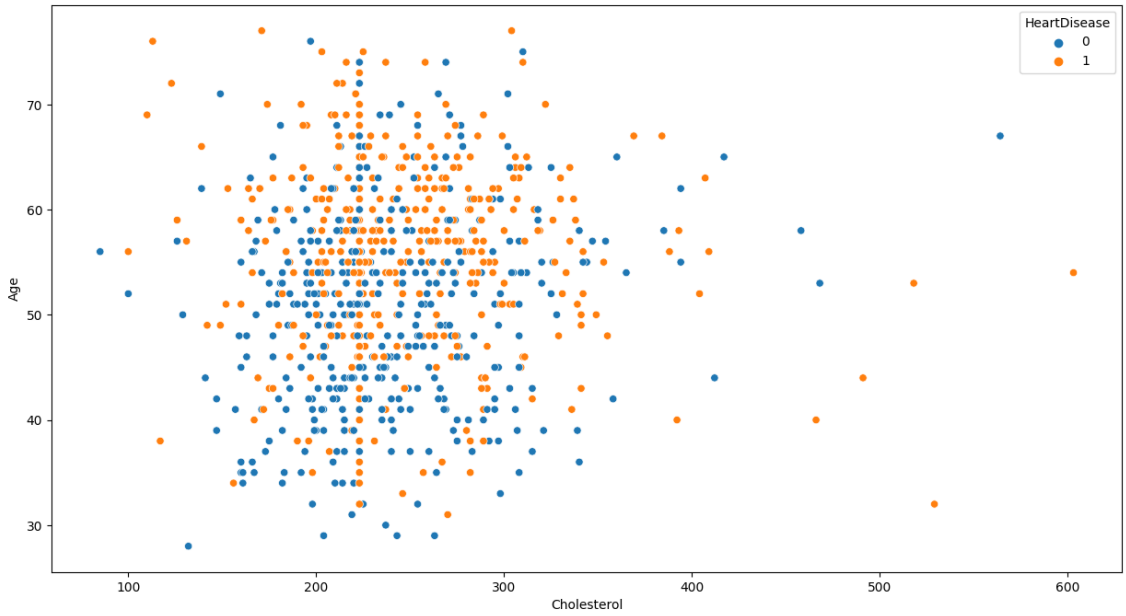
In [151]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df, x='MaxHR', y = 'Cholesterol',hue=df['RestingECG'])
plt.show()
```



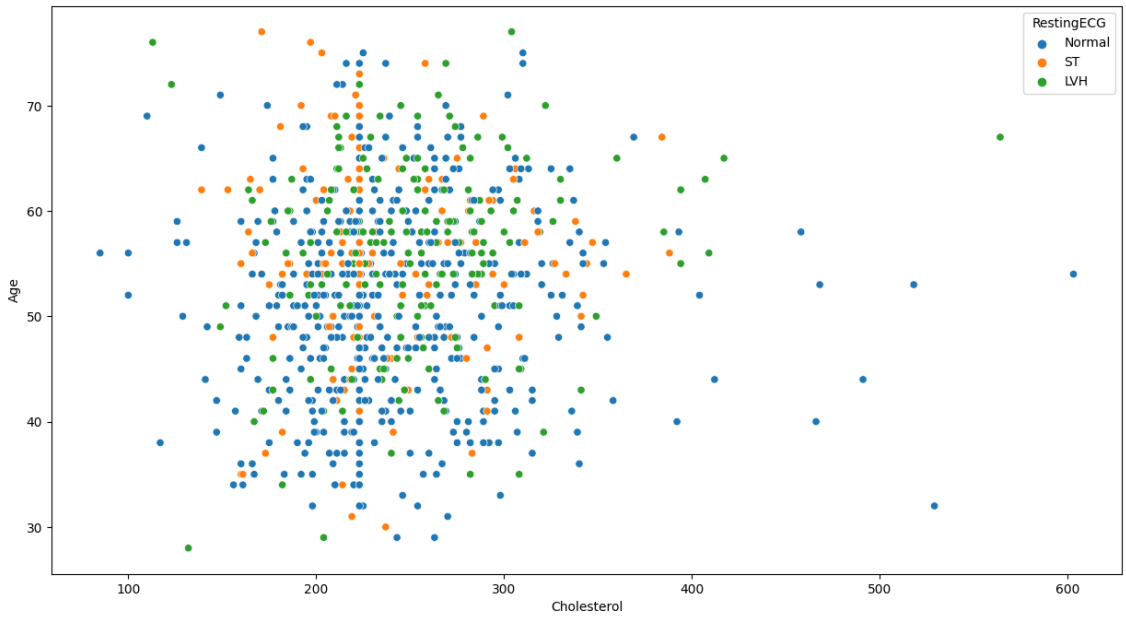
In [156]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df,  x = 'Cholesterol',y='Age',hue=df['HeartDisease'])
plt.show()
```



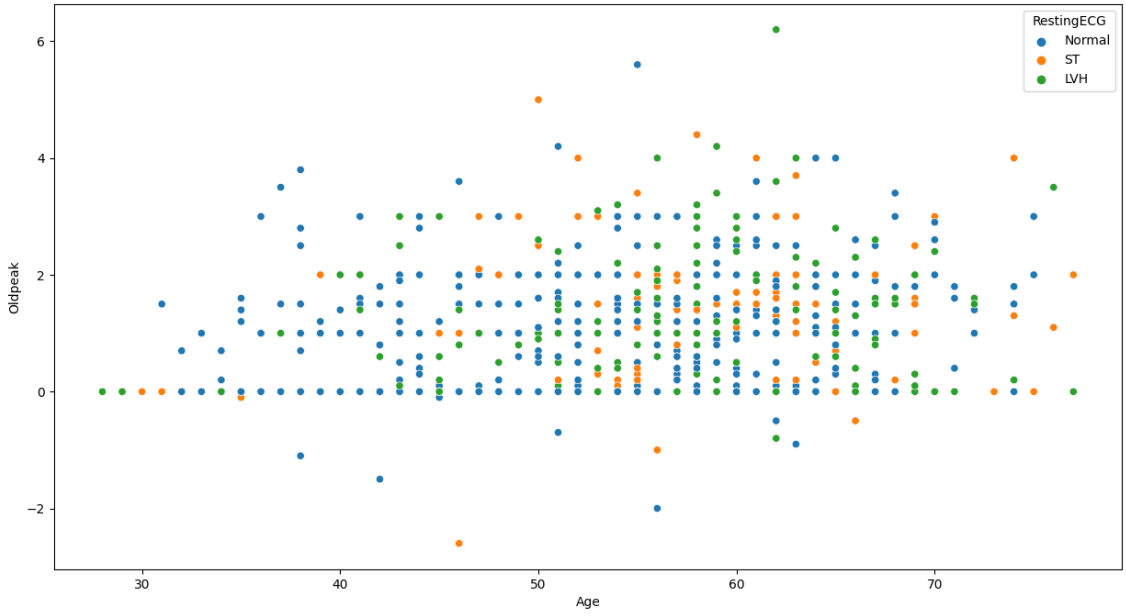
In [158]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df,  x = 'Cholesterol',y='Age',hue=df['RestingECG'])
plt.show()
```



In [162]:

```
plt.figure(figsize=(15,8))
sns.scatterplot(data = df,  x = 'Age',y='Oldpeak',hue=df['RestingECG'])
plt.show()
```

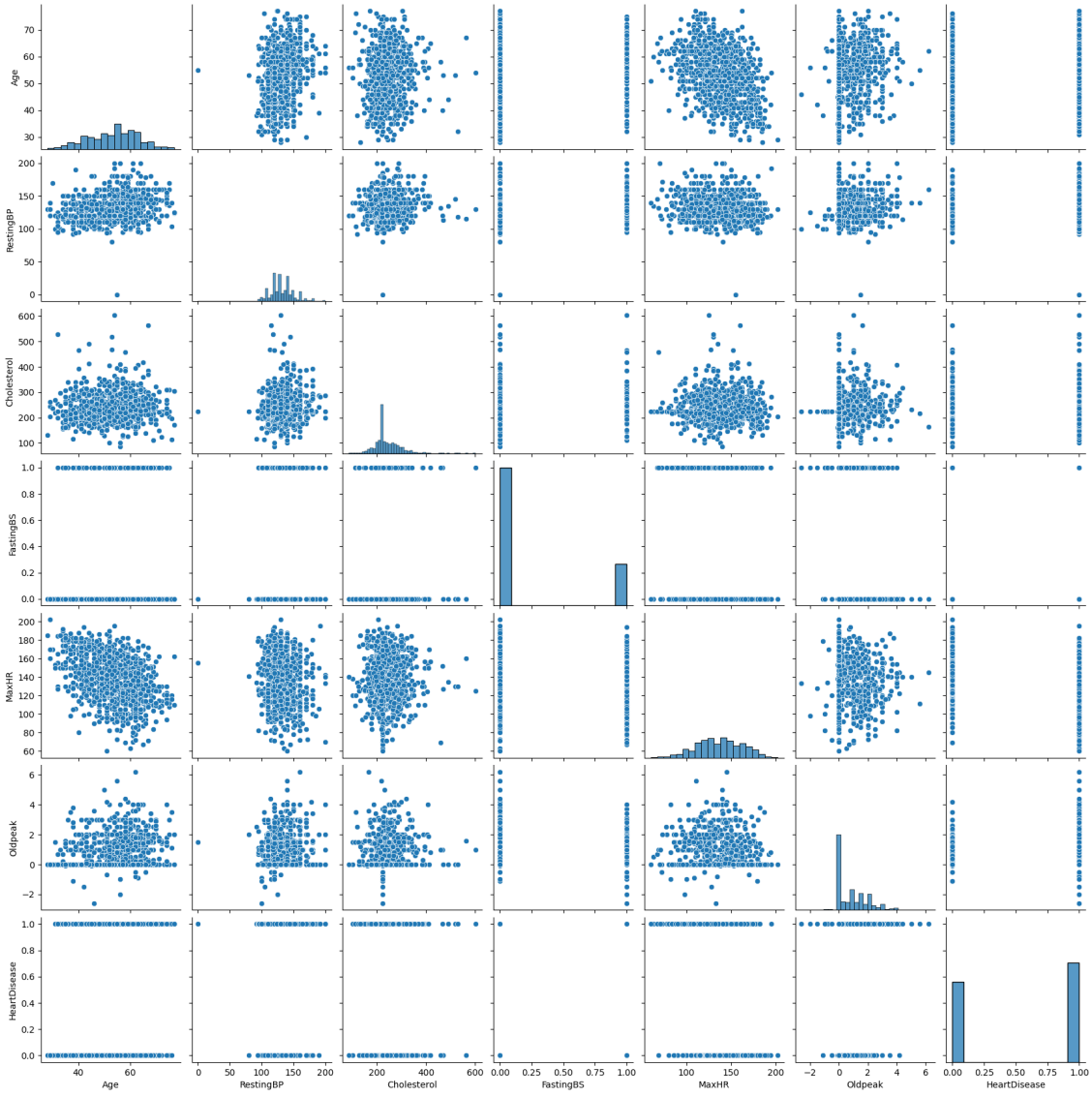


In [163]:

```
sns.pairplot(df)
```

Out[163]:

```
<seaborn.axisgrid.PairGrid at 0x7bde63acf820>
```



In [169]:

```
#Label Encoding The rest. :)
obj_columns = []
for column in df.columns:
  if type(df[column][0]) == type('this is a string'):
    obj_columns.append(column)
    print(column)
```

```
Sex
ChestPainType
RestingECG
ExerciseAngina
ST_Slope
```

In [170]:

```
from sklearn.preprocessing import LabelEncoder

le_sex = LabelEncoder()
le_chest_pain_type = LabelEncoder()
le_resting_ecg = LabelEncoder()
le_exercise_angina = LabelEncoder()
le_st_slope = LabelEncoder()
```

In [172]:

```
le_sex.fit(df['Sex'])
le_chest_pain_type.fit(df['ChestPainType'])
le_resting_ecg.fit(df['RestingECG'])
le_exercise_angina.fit(df['ExerciseAngina'])
le_st_slope.fit(df['ST_Slope'])
```

Out[172]:

LabelEncoder()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [173]:

```
sex = le_sex.transform(df['Sex'])
chest_pain_type = le_chest_pain_type.transform(df['ChestPainType'])
resting_ecg = le_resting_ecg.transform(df['RestingECG'])
excersice_angina = le_exercise_angina.transform(df['ExerciseAngina'])
st_slope = le_st_slope.transform(df['ST_Slope'])
```

In [175]:

```
print(le_sex.classes_)
print(le_chest_pain_type.classes_)
print(le_resting_ecg.classes_)
print(le_exercise_angina.classes_)
print(le_st_slope.classes_)
```

```
['F' 'M']
['ASY' 'ATA' 'NAP' 'TA']
['LVH' 'Normal' 'ST']
['N' 'Y']
['Down' 'Flat' 'Up']
```

In [176]:

```

```

In [177]:

```
new_df = df.copy()
```

In [190]:

```
cols = list(df.columns)
for column_no in range(len(cols)):
  cols[column_no] = cols[column_no][0:2].lower()+cols[column_no][2:]
```

In [191]:

```
cols
```

Out[191]:

```
['age',
 'sex',
 'chestPainType',
 'restingBP',
 'cholesterol',
 'fastingBS',
 'restingECG',
 'maxHR',
 'exerciseAngina',
 'oldpeak',
 'st_Slope',
 'heartDisease']
```

In [192]:

```
new_df.columns = cols
```

In [196]:

```
encoded_cols = [sex,chest_pain_type,resting_ecg,excersice_angina,st_slope]
```

In [197]:

```
for i in encoded_cols:
  print(len(i))
```

```
918
918
918
918
918
```

In [201]:

```
obj_columns = []
for column in new_df.columns:
  if type(new_df[column][0]) == type('this is a string'):
    obj_columns.append(column)
    print(column)
```

```
sex
chestPainType
restingECG
exerciseAngina
st_Slope
```

In [202]:

```
for i,j in zip(obj_columns,encoded_cols):
  new_df[i] = j
```

In [203]:

```
new_df
```

Out[203]:

	age	sex	chestPainType	restingBP	cholesterol	fastingBS	restingECG	maxHR	exerc
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	
...	...	...	...	...	...	...	...	...	
913	45	1	3	110	264	0	1	132	
914	68	1	0	144	193	1	1	141	
915	57	1	0	130	131	0	1	115	
916	57	0	1	130	236	0	0	174	
917	38	1	2	138	175	0	1	173	

918 rows × 12 columns

In [204]:

```
new_df.describe()
```

Out[204]:

	age	sex	chestPainType	restingBP	cholesterol	fastingBS	restingE
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000000	918.000
mean	53.510893	0.789760	0.781046	132.396514	240.581699	0.233115	0.989
std	9.432617	0.407701	0.956519	18.514154	53.982967	0.423046	0.631
min	28.000000	0.000000	0.000000	0.000000	85.000000	0.000000	0.000
25%	47.000000	1.000000	0.000000	120.000000	214.000000	0.000000	1.000
50%	54.000000	1.000000	0.000000	130.000000	223.000000	0.000000	1.000
75%	60.000000	1.000000	2.000000	140.000000	267.000000	0.000000	1.000
max	77.000000	1.000000	3.000000	200.000000	603.000000	1.000000	2.000

Preparing for model building now: -

In [212]:

```
X = new_df.copy()
X.drop(['heartDisease'],axis=1,inplace=True)
y = new_df.copy()['heartDisease']
```

In [213]:

```
X.head()
```

Out[213]:

	age	sex	chestPainType	restingBP	cholesterol	fastingBS	restingECG	maxHR	exercis
0	40	1	1	140	289	0	1	172	
1	49	0	2	160	180	0	1	156	
2	37	1	1	130	283	0	2	98	
3	48	0	0	138	214	0	1	108	
4	54	1	2	150	195	0	1	122	

In [217]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=7,shuff
```

In [224]:

```
from sklearn.pipeline import Pipeline,make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import MinMaxScaler
```

In [233]:

```
scaler = MinMaxScaler()
scaling_cols = ['age','restingBP','cholestrol','maxHR','oldpeak']
```

In [234]:

```
#integrating scaler into pipeline
scaling_pipe = make_pipeline(scaler)
```

In [241]:

```
# making the column transformer and integrating the pipeline into it.
transformer = ColumnTransformer(
    transformers=[('MinMaxScaler',scaling_pipe,scaling_cols)]
)
```

In [242]:

```
from sklearn.ensemble import RandomForestClassifier
#Initializing the model:-
rfc = RandomForestClassifier(n_estimators=500,max_depth=200)
```

In [243]:

```
main_pipe = make_pipeline(transformer,rfc)
```

Dropping the pipeline idea for a bit.

In [246]:

```
scaler_age = MinMaxScaler()
scaler_resting_bp = MinMaxScaler()
scaler_cholesterol = MinMaxScaler()
scaler_max_hr = MinMaxScaler()
scaler_oldpeak = MinMaxScaler()
```

In [248]:

```
X['age']  = scaler_age.fit_transform(np.array(X['age']).reshape(-1,1))
X['restingBP']  = scaler_resting_bp.fit_transform(np.array(X['restingBP']).reshape(-1,1
X['cholesterol']  = scaler_cholesterol.fit_transform(np.array(X['cholesterol']).reshape
X['maxHR']  = scaler_max_hr.fit_transform(np.array(X['maxHR']).reshape(-1,1))
X['oldpeak']  = scaler_oldpeak.fit_transform(np.array(X['oldpeak']).reshape(-1,1))
```

In [249]:

```
X.head()
```

Out[249]:

	age	sex	chestPainType	restingBP	cholesterol	fastingBS	restingECG	maxHR	e
0	0.244898	1	1	0.70	0.393822	0	1	0.788732	
1	0.428571	0	2	0.80	0.183398	0	1	0.676056	
2	0.183673	1	1	0.65	0.382239	0	2	0.267606	
3	0.408163	0	0	0.69	0.249035	0	1	0.338028	
4	0.530612	1	2	0.75	0.212355	0	1	0.436620	

In [250]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=7,shuff
```

In [272]:

```
rfc.fit(X_train,y_train)
```

Out[272]:

```
RandomForestClassifier(max_depth=200, n_estimators=500)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [273]:

```
rfc.score(X_test,y_test)
```

Out[273]:

```
0.907608695652174
```

In [274]:

```
from sklearn.metrics import precision_score,recall_score,accuracy_score
```

In [275]:

```
precision_score(y_test,rfc.predict(X_test))
```

Out[275]:

```
0.8969072164948454
```

In [276]:

```
recall_score(y_test,rfc.predict(X_test))
```

Out[276]:

```
0.925531914893617
```

In [277]:

```
accuracy_score(y_test,rfc.predict(X_test))
```

Out[277]:

```
0.907608695652174
```

Please do upvote if you like the analysis :)

In [ ]: