Hillary Gesel
Shengqi Ye
Pronata Datta
Shun An Chang

Heart Disease

Introduction

Cardiovascular Disease (CVD) or heart disease is the leading cause of morbidity and mortality in the United States, according to the Centers for Disease Control and Prevention

We wanted to evaluate and analyze several pathological parameters that are commonly used to diagnose heart disease. Heart disease is a condition that is developed over time and leads to a significant amount of medical expenses and long term

Using the database found in Kaggle, we performed analysis on 11 parameters that can be used to predict a possible heart disease.

Dataset - Kaggle

https://www.kaggle.com/fedesoriano/heart-failure-prediction

Attribute Information

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]

Data From Kaggle

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	exerciseangina	Oldpeak	ST_Slope	HeartDisease
0	40.0	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0.0
1	49.0	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1.0
2	37.0	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0.0
3	48.0	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1.0
4	54.0	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45.0	M	TA	110.0	264.0	0.0	Normal	132.0	N	1.2	Flat	1.0
914	68.0	M	ASY	144.0	193.0	1.0	Normal	141.0	N	3.4	Flat	1.0
915	57.0	M	ASY	130.0	131.0	0.0	Normal	115.0	Y	1.2	Flat	1.0
916	57.0	F	ATA	130.0	236.0	0.0	LVH	174.0	N	0.0	Flat	1.0
917	38.0	M	NAP	138.0	175.0	0.0	Normal	173.0	N	0.0	Up	0.0

918 rows × 12 columns

	SexID	Sex
0	0	F
1	1	M

	exerciseanginaID	exerciseangina
0	0	N
1	1	Y

```
con = sqlite3.connect("HeartDB.db") # change to 'sqlite:/
cur = con.cursor()
cur.execute("drop table if exists Sex")
cur.execute("""CREATE TABLE Sex (
            SexID integer PRIMARY KEY,

            Sex TEXT
            );""") # use your column names here

sql_statement = 'SELECT DISTINCT Sex FROM Heart'
cur.execute(sql_statement)
Sex_fetched = cur.fetchall()
touple_to_list = []
# print(patient_info)
sex_list = [1,0]
for ele in Sex_fetched:
  for i in ele:
    touple_to_list.append(i)

sex_data = list(zip(sex_list,touple_to_list))
cur.executemany("""INSERT INTO Sex(SexID, Sex)
                 VALUES (?,?);""", sex_data)
con.commit()

sql_statement = "select * from Sex;"
Sex = pd.read_sql_query(sql_statement, con)
display(Sex)

Sex_dic = {}
for i in range(2):
  Sex_dic[Sex.SexID[i]] = Sex.Sex[i]
print(Sex_dic)
```

```
#con = sqlite3.connect("HeartDB.db") # change to 'sqlite:///your_filename.db'
#cur = con.cursor()
cur.execute("drop table if exists ExerciseAngina")
cur.execute("""CREATE TABLE ExerciseAngina (
            exerciseanginaID integer PRIMARY KEY,
            exerciseangina TEXT
            );""") # use your column names here
sql_statement = 'SELECT DISTINCT ExerciseAngina FROM Heart order by exerciseangina;'
cur.execute(sql_statement)
exerciseangina_fetched = cur.fetchall()

# print(patient_info)

cur.executemany("""INSERT INTO ExerciseAngina(exerciseangina)
                 VALUES (?);""", exerciseangina_fetched)

cur.execute("""UPDATE ExerciseAngina SET ExerciseAnginaID = 0 where ExerciseAnginaID = 1;""")
cur.execute("""UPDATE ExerciseAngina SET ExerciseAnginaID = 1 where ExerciseAnginaID = 2;""")
con.commit()

sql_statement = "select * from ExerciseAngina;"
ExerciseAngina = pd.read_sql_query(sql_statement, con)
display(ExerciseAngina)

ExerciseAngina_dic = {}
for i in range(2):
  ExerciseAngina_dic[ExerciseAngina.exerciseanginaID[i]] = ExerciseAngina.exerciseangina[i]
print(ExerciseAngina_dic)
```

	ChestPainTypeID	ChestPainType
0	0	ASY
1	1	ATA
2	2	NAP
3	3	TA

	RestingECGID	RestingECG
0	0	LVH
1	1	Normal
2	2	ST

```
    );""") # use your column names here

sql_statement = 'SELECT DISTINCT ChestPainType FROM Heart'
cur.execute(sql_statement)
chestpaintype_fetched = cur.fetchall()

# print(patient_info)

cur.executemany("""INSERT INTO ChestPainType(ChestPainType)
                VALUES (?);""", chestpaintype_fetched)

cur.execute("""UPDATE ChestPainType SET ChestPainTypeID = 0 where ChestPainTypeID = 3;""")
cur.execute("""UPDATE ChestPainType SET ChestPainTypeID = 3 where ChestPainTypeID = 4;""")

con.commit()

sql_statement = "select * from ChestPainType;"
ChestPainType = pd.read_sql_query(sql_statement, con)
display(ChestPainType)

ChestPainType_dic = {}
for i in range(4):
  ChestPainType_dic[ChestPainType.ChestPainTypeID[i]] = ChestPainType.ChestPainType[i]
print(ChestPainType_dic)
```

```
#con = sqlite3.connect("HeartDB.db") # change to 'sqlite:///your_filename.db'
#cur = con.cursor()
cur.execute("drop table if exists RestingECG")
cur.execute("""CREATE TABLE RestingECG (
            RestingECGID integer PRIMARY KEY,
            RestingECG TEXT
            );""") # use your column names here

sql_statement = 'SELECT DISTINCT RestingECG FROM Heart'
cur.execute(sql_statement)
restingecg_fetched = cur.fetchall()

# print(patient_info)

cur.executemany("""INSERT INTO RestingECG(RestingECG)
                VALUES (?);""", restingecg_fetched)

cur.execute("""UPDATE RestingECG SET RestingECGID = 0 where RestingECGID = 3;""")

con.commit()

# ST_slope_dic = {}
# for i in range(3):
#   ST_slope_dic[ST_Slope.ST_SlopeID[i]] = ST_Slope.ST_Slope[i]

sql_statement = "select * from RestingECG;"
RestingECG = pd.read_sql_query(sql_statement, con)
display(RestingECG)

# RestingECG_dic = {}
# for i in range(3):
#   RestingECG_dic[RestingECG.RestingECGID[i]] = RestingECG.RestingECG[i]
# print(RestingECG_dic)
```

	ST_SlopeID	ST_Slope
0	0	Down
1	1	Flat
2	2	Up

```
#con = sqlite3.connect("HeartDB.db") # change to 'sqlite:///your_filename.db'
#cur = con.cursor()
cur.execute("drop table if exists ST_Slope")
cur.execute("""CREATE TABLE ST_Slope (
            ST_SlopeID integer PRIMARY KEY,
            ST_Slope TEXT
            );""") # use your column names here

sql_statement = 'SELECT DISTINCT ST_Slope FROM Heart'
cur.execute(sql_statement)
st_slope_fetched = cur.fetchall()

# print(patient_info)

cur.executemany("""INSERT INTO ST_Slope(ST_Slope)
                VALUES (?);""", st_slope_fetched)


cur.execute("""UPDATE ST_Slope SET ST_SlopeID = 0 where ST_SlopeID = 3;""")
cur.execute("""UPDATE ST_Slope SET ST_SlopeID = 3 where ST_SlopeID = 1;""")
cur.execute("""UPDATE ST_Slope SET ST_SlopeID = 1 where ST_SlopeID = 2;""")
cur.execute("""UPDATE ST_Slope SET ST_SlopeID = 2 where ST_SlopeID = 3;""")


con.commit()

sql_statement = "select * from ST_Slope;"
ST_Slope = pd.read_sql_query(sql_statement, con)
display(ST_Slope)
ST_slope_dic = {}
for i in range(3):
  ST_slope_dic[ST_Slope.ST_SlopeID[i]] = ST_Slope.ST_Slope[i]

print(ST_slope_dic)
```

```
join_sql = """select
          h.Age,
          s.Sex,
          c.ChestPainType,
          h.RestingBP,
          h.Cholesterol,
          h.FastingBS,
          r.RestingECG,
          h.MaxHR,
          e.ExerciseAngina,
          h.Oldpeak,
          ss.ST_Slope,
          h.HeartDisease
          from map_heart h
          join
          Sex s
          on h.Sex=s.SexID
          join
          ExerciseAngina e
          on h.exerciseangina=e.exerciseanginaID
          join
          ChestPainType c
          on h.ChestPainType=c.ChestPainTypeID
          join
          RestingECG r
          on h.RestingECG=r.RestingECGID
          join
          ST_Slope ss
          on h.ST_Slope=ss.ST_SlopeID"""

heart_information = pd.read_sql_query(join_sql, con)
```

Dataset with Foreign Keys to Normalized Tables

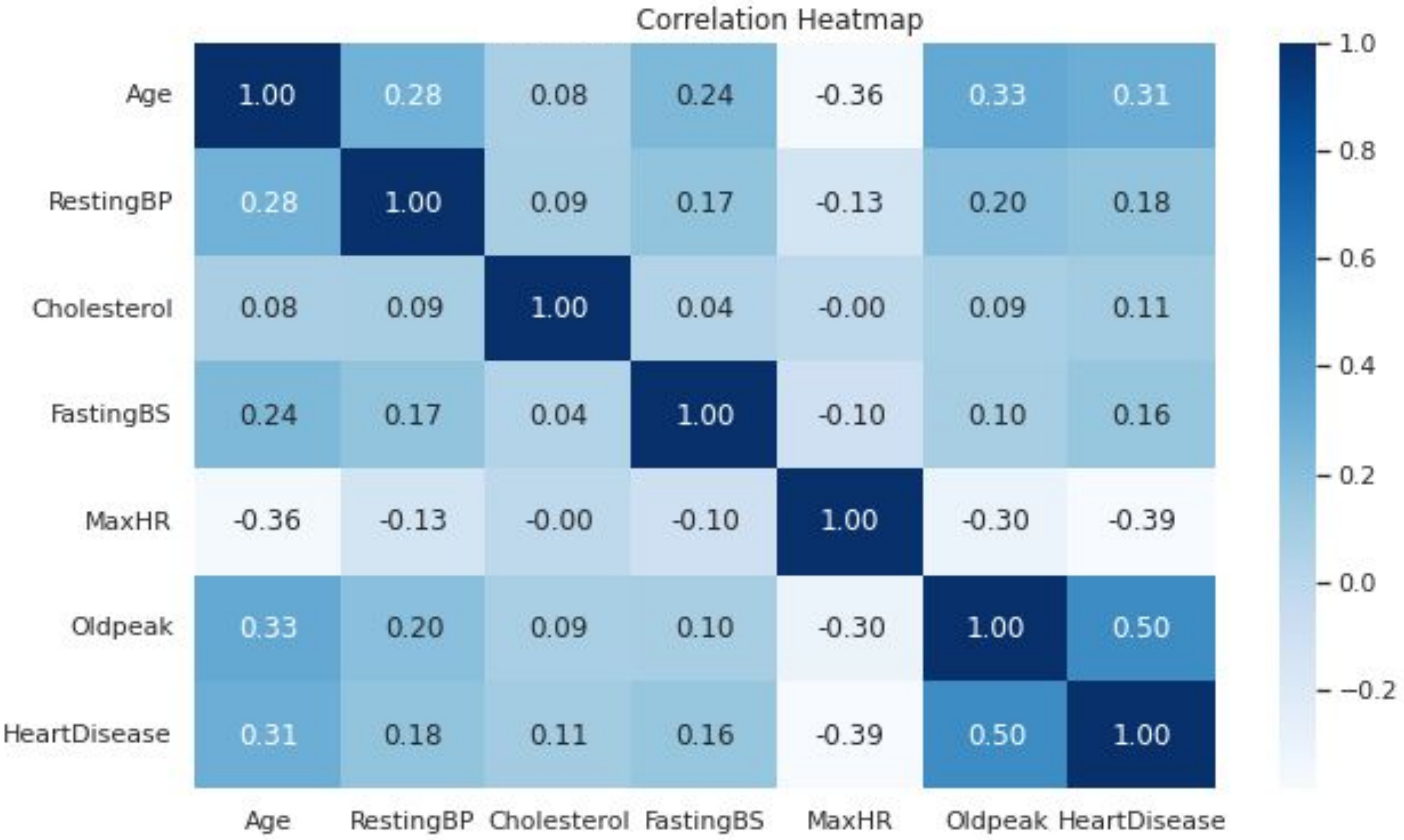
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40.0	1	1	140.0	289.0	0.0	1	172.0	0	0.0	2	0.0
1	49.0	0	2	160.0	180.0	0.0	1	156.0	0	1.0	1	1.0
2	37.0	1	1	130.0	283.0	0.0	2	98.0	0	0.0	2	0.0
3	48.0	0	0	138.0	214.0	0.0	1	108.0	1	1.5	1	1.0
4	54.0	1	2	150.0	195.0	0.0	1	122.0	0	0.0	2	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45.0	1	3	110.0	264.0	0.0	1	132.0	0	1.2	1	1.0
914	68.0	1	0	144.0	193.0	1.0	1	141.0	0	3.4	1	1.0
915	57.0	1	0	130.0	131.0	0.0	1	115.0	1	1.2	1	1.0
916	57.0	0	1	130.0	236.0	0.0	0	174.0	0	0.0	1	1.0
917	38.0	1	2	138.0	175.0	0.0	1	173.0	0	0.0	2	0.0

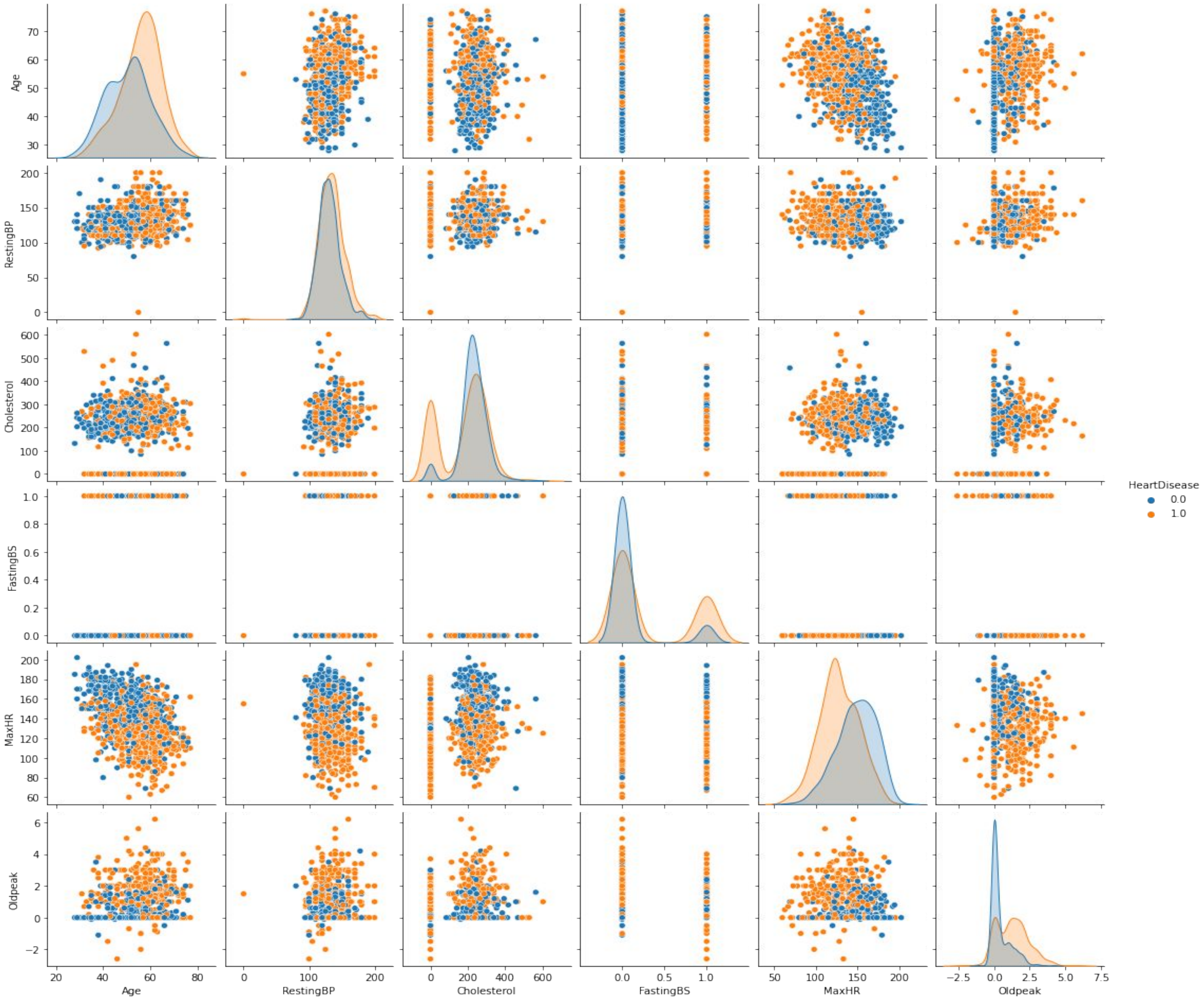
918 rows × 12 columns

Data From Kaggle

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	exerciseangina	Oldpeak	ST_Slope	HeartDisease
0	40.0	M	ATA	140.0	289.0	0.0	Normal	172.0	N	0.0	Up	0.0
1	49.0	F	NAP	160.0	180.0	0.0	Normal	156.0	N	1.0	Flat	1.0
2	37.0	M	ATA	130.0	283.0	0.0	ST	98.0	N	0.0	Up	0.0
3	48.0	F	ASY	138.0	214.0	0.0	Normal	108.0	Y	1.5	Flat	1.0
4	54.0	M	NAP	150.0	195.0	0.0	Normal	122.0	N	0.0	Up	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
913	45.0	M	TA	110.0	264.0	0.0	Normal	132.0	N	1.2	Flat	1.0
914	68.0	M	ASY	144.0	193.0	1.0	Normal	141.0	N	3.4	Flat	1.0
915	57.0	M	ASY	130.0	131.0	0.0	Normal	115.0	Y	1.2	Flat	1.0
916	57.0	F	ATA	130.0	236.0	0.0	LVH	174.0	N	0.0	Flat	1.0
917	38.0	M	NAP	138.0	175.0	0.0	Normal	173.0	N	0.0	Up	0.0

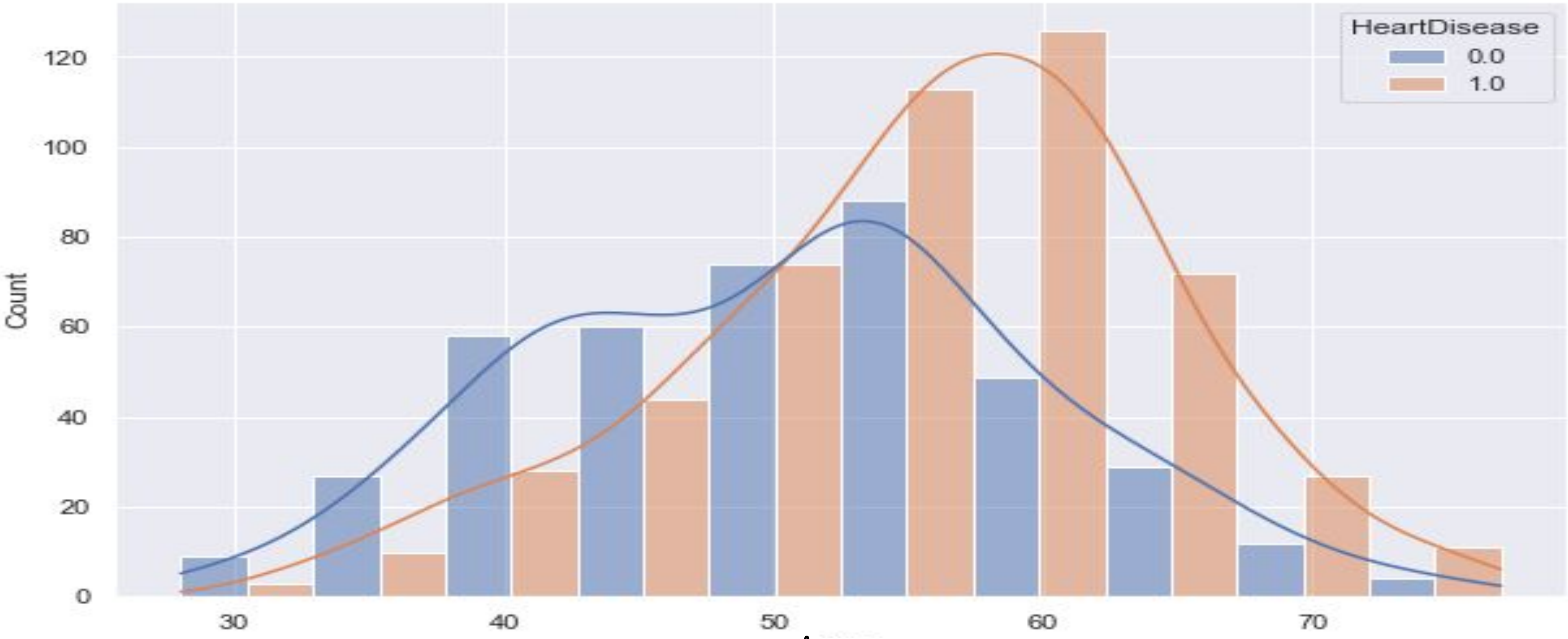
918 rows × 12 columns


Correlation Heatmap
Age
RestingBP
Cholesterol
FastingBS
MaxHR
Oldpeak
HeartDisease
1.00 0.28 0.08 0.24 -0.36 0.33 0.31
0.28 1.00 0.09 0.17 -0.13 0.20 0.18
0.08 0.09 1.00 0.04 -0.00 0.09 0.11
0.24 0.17 0.04 1.00 -0.10 0.10 0.16
-0.36 -0.13 -0.00 -0.10 1.00 -0.30 -0.39
0.33 0.20 0.09 0.10 -0.30 1.00 0.50
0.31 0.18 0.11 0.16 -0.39 0.50 1.00
1.0
0.8
0.6
0.4
0.2
0.0
−0.2



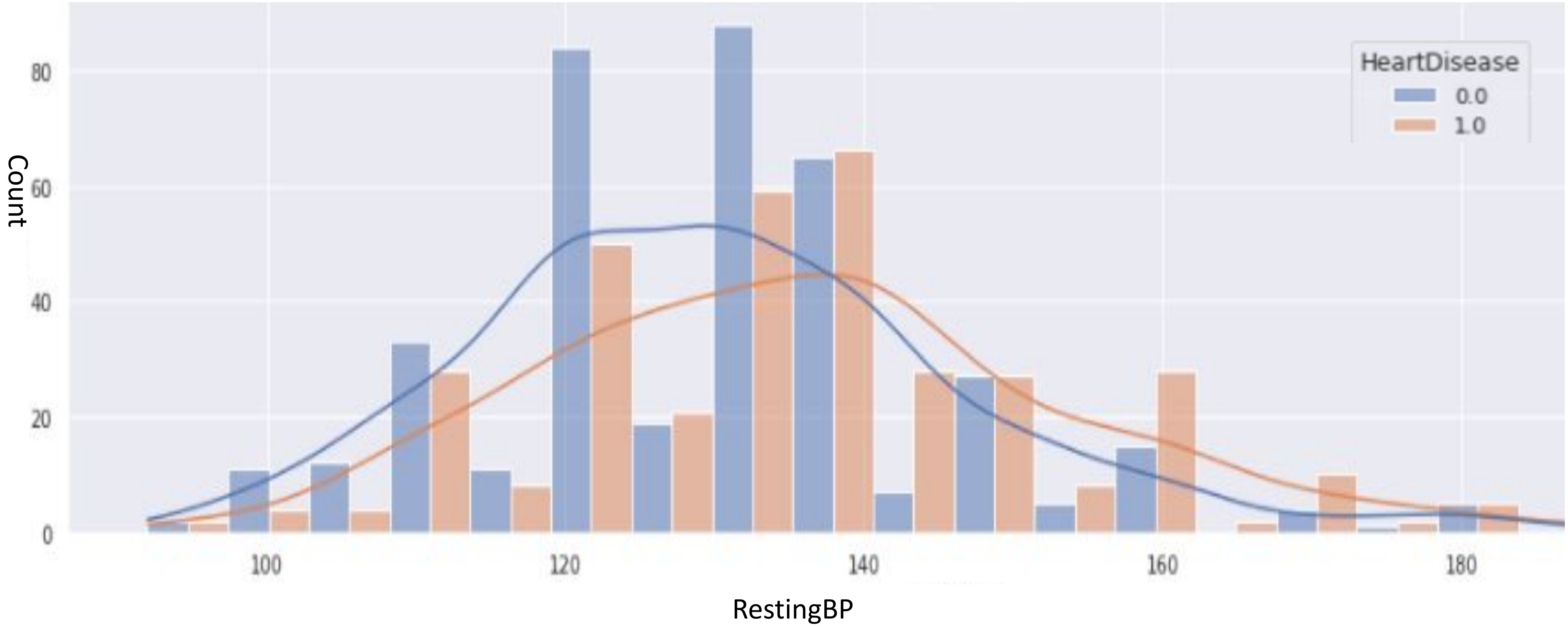
Age
RestingBP
Cholesterol
FastingBS
MaxHR
Oldpeak
HeartDisease
0.0
1.0

Age hist plot

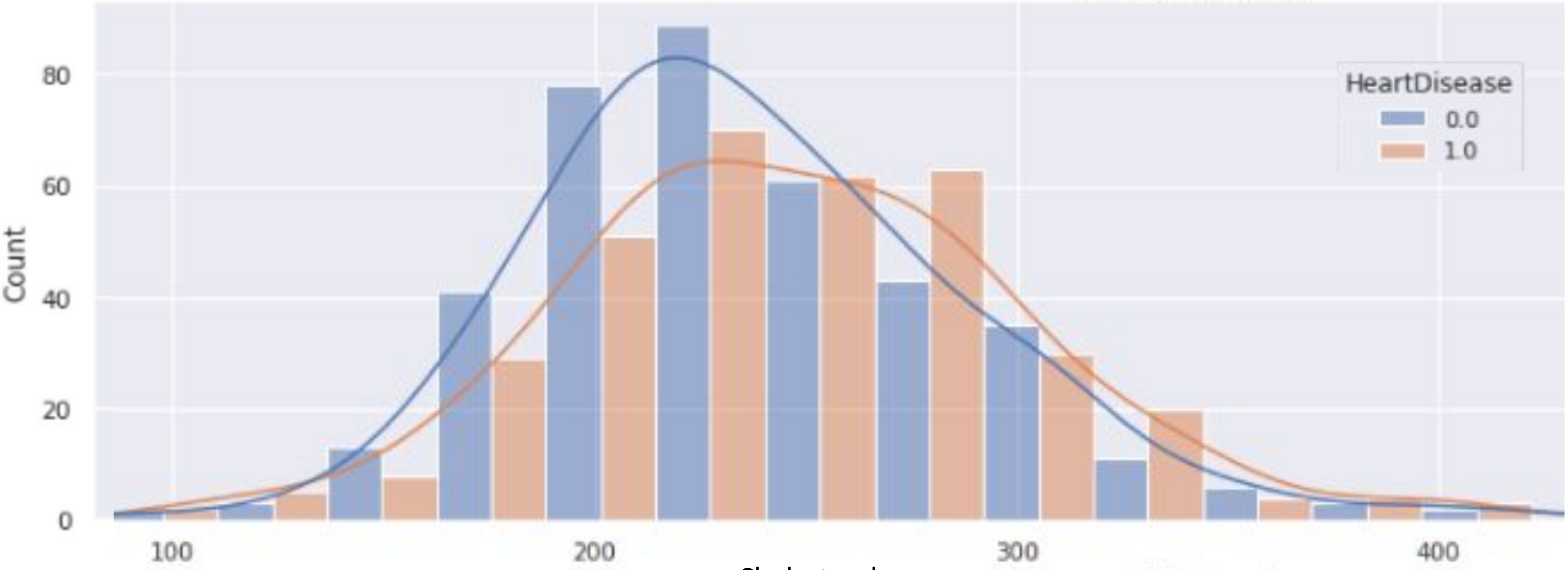


Age

RestingBP hist plot

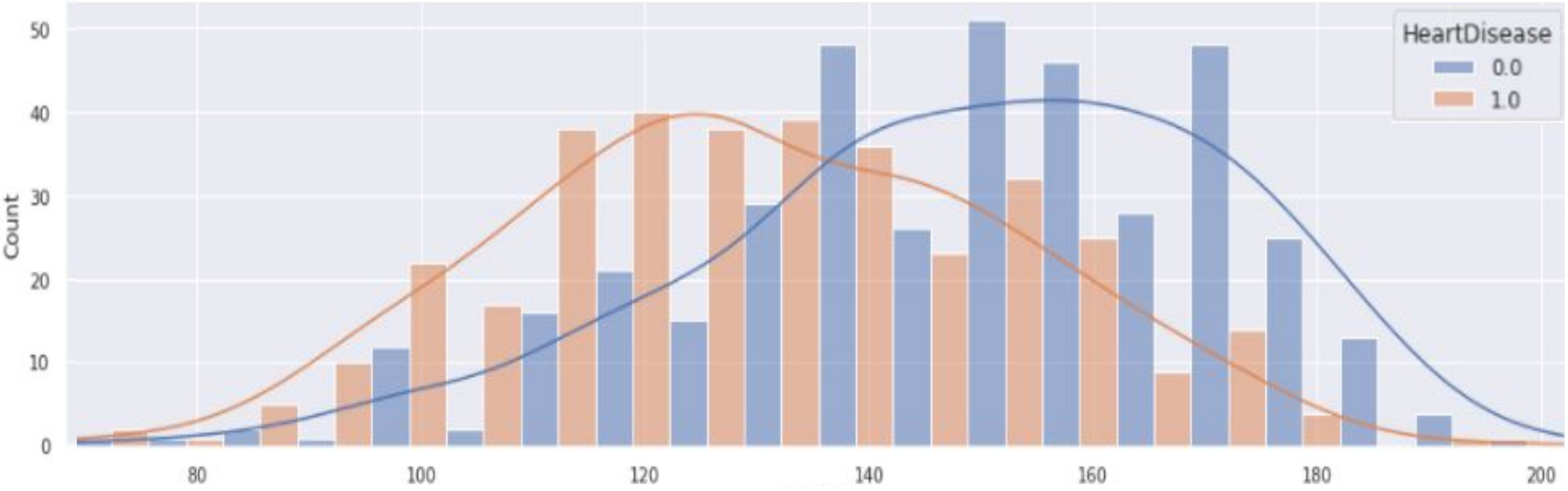


Cholesterol hist plot



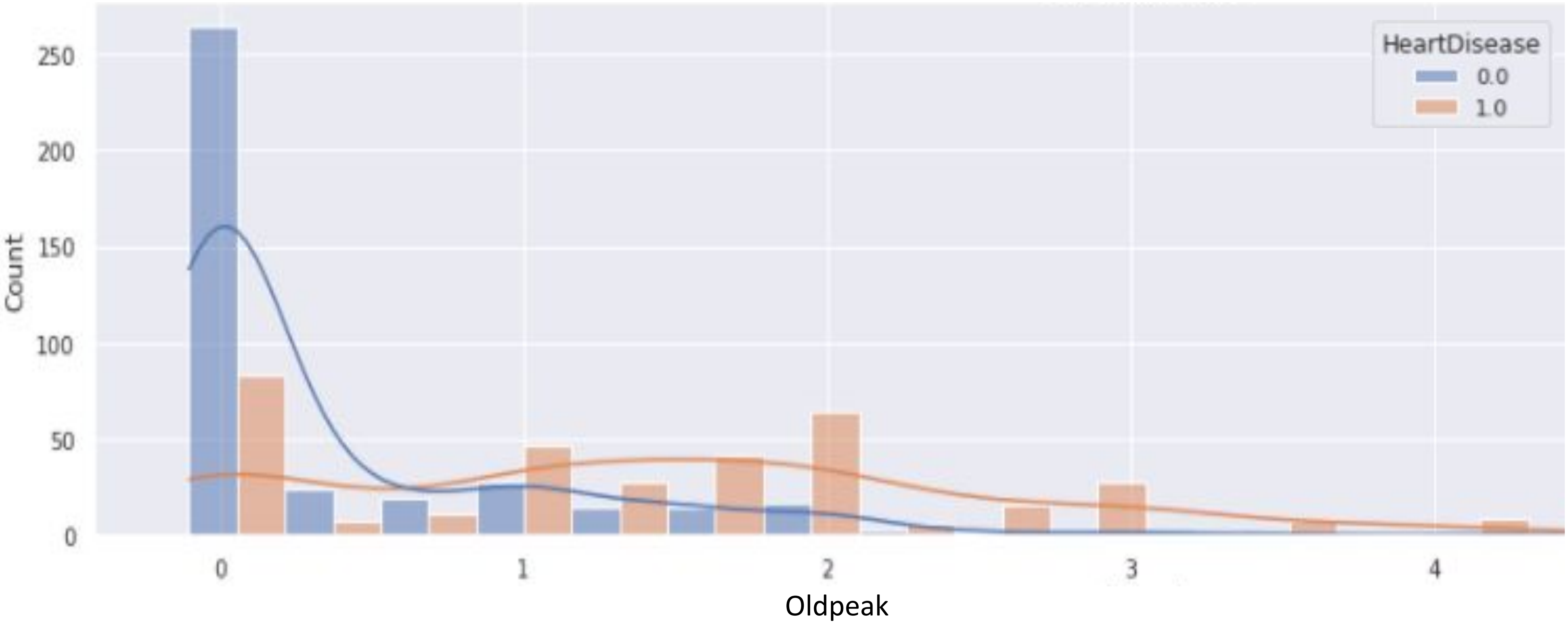
Cholesterol

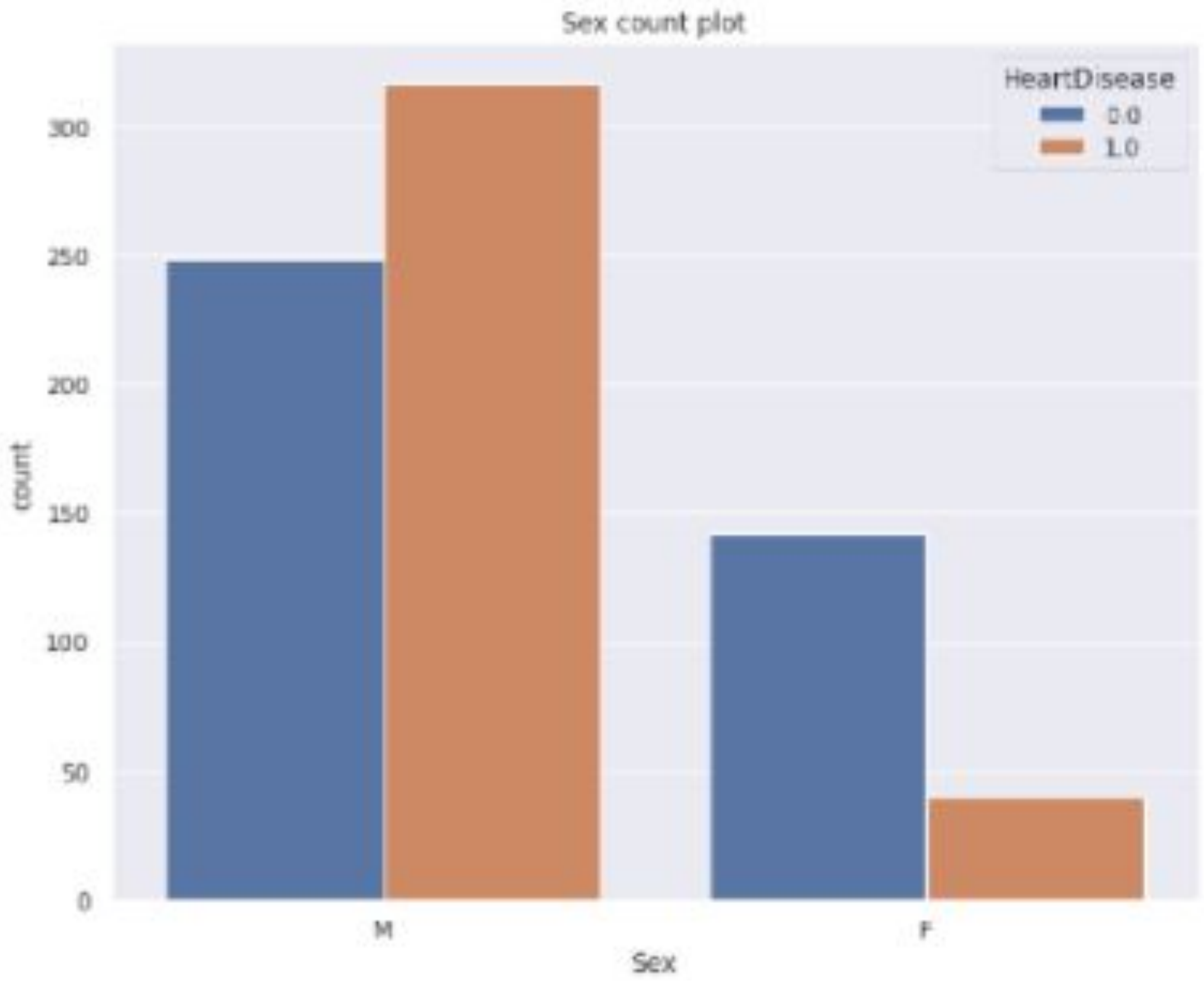
MaxHR hist plot

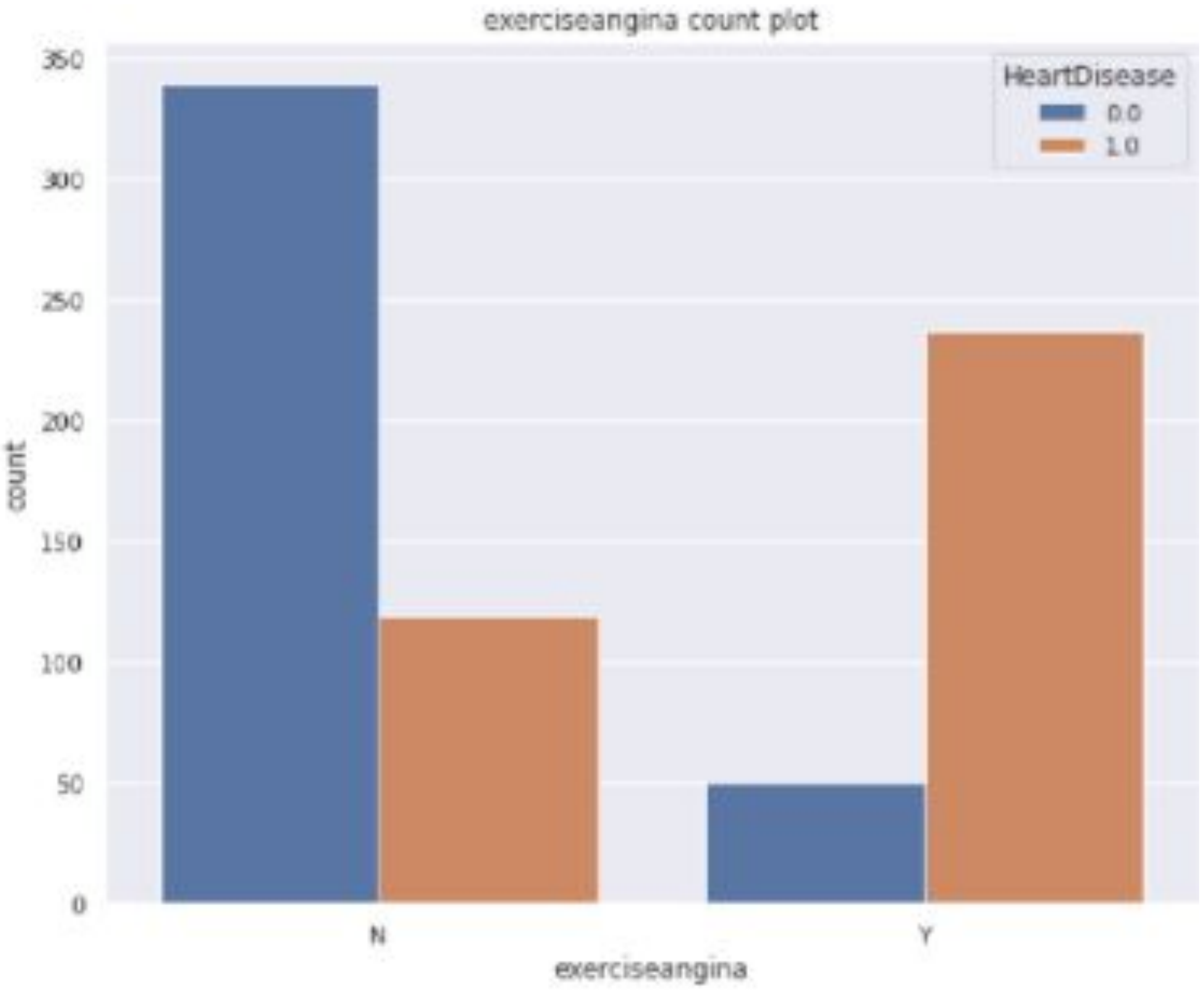


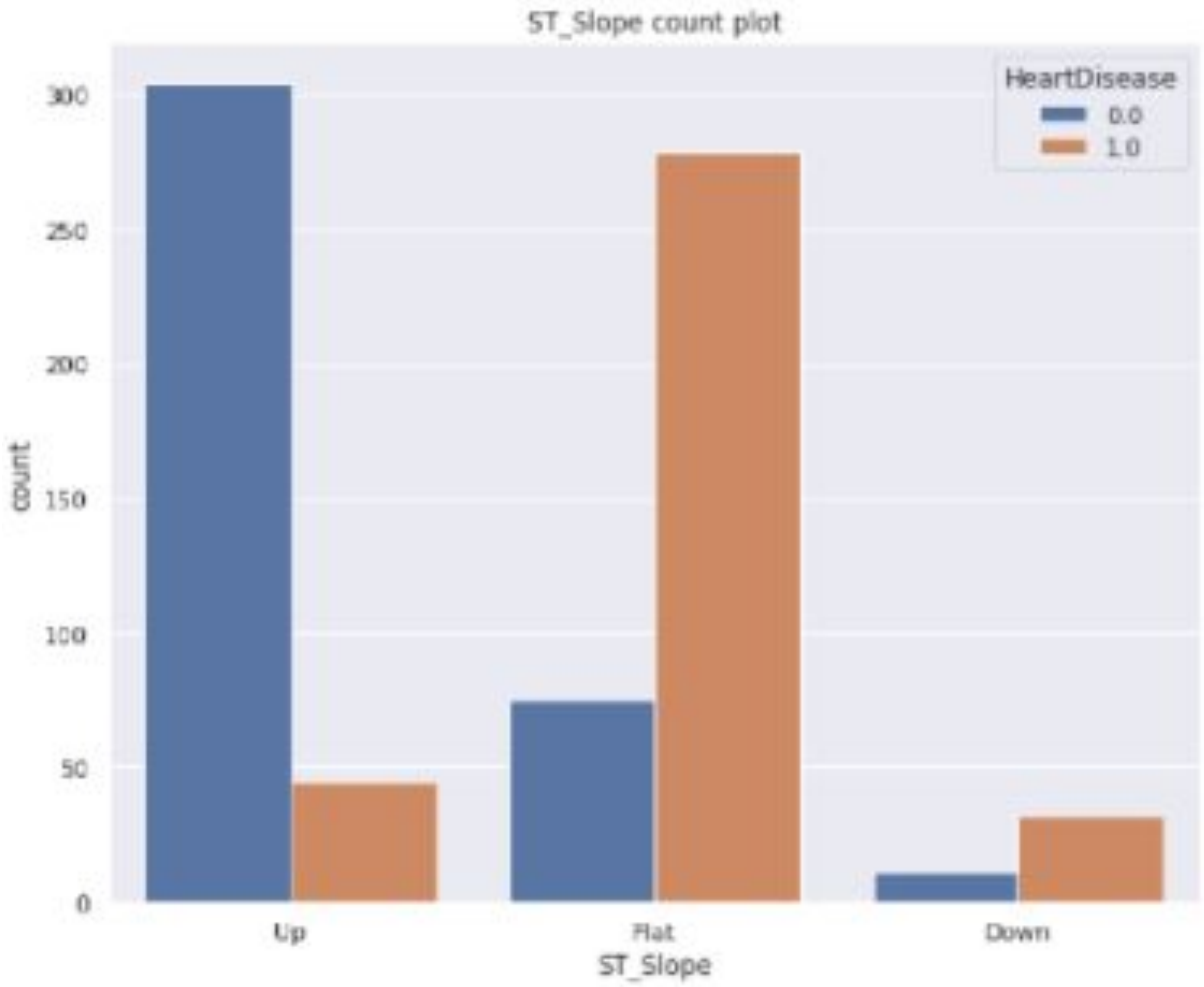
MaxHR

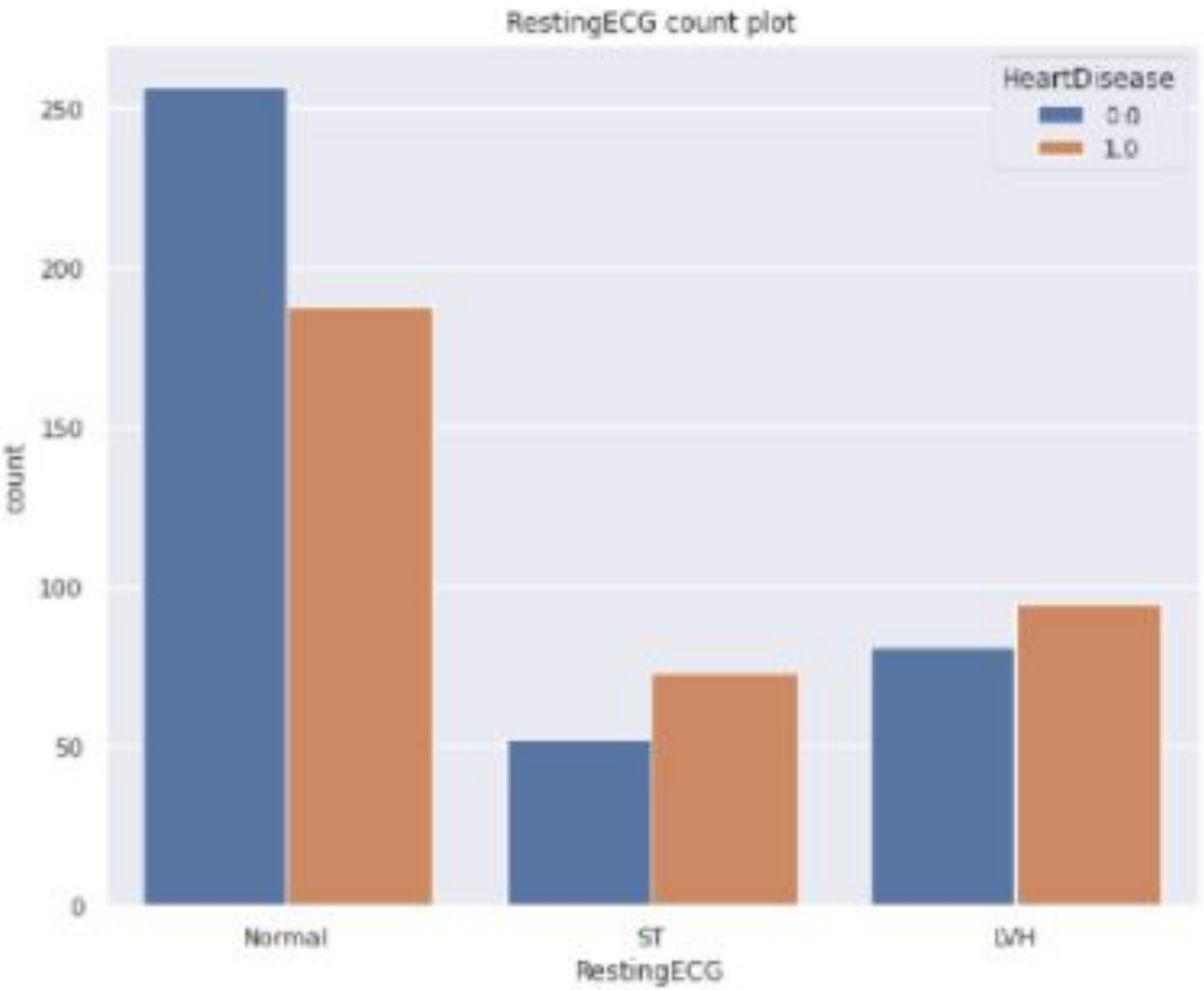
Oldpeak hist plot



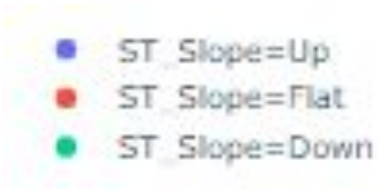

Sex count plot
HeartDisease
0.0
1.0
count
0
50
100
150
200
250
300
M
F
Sex

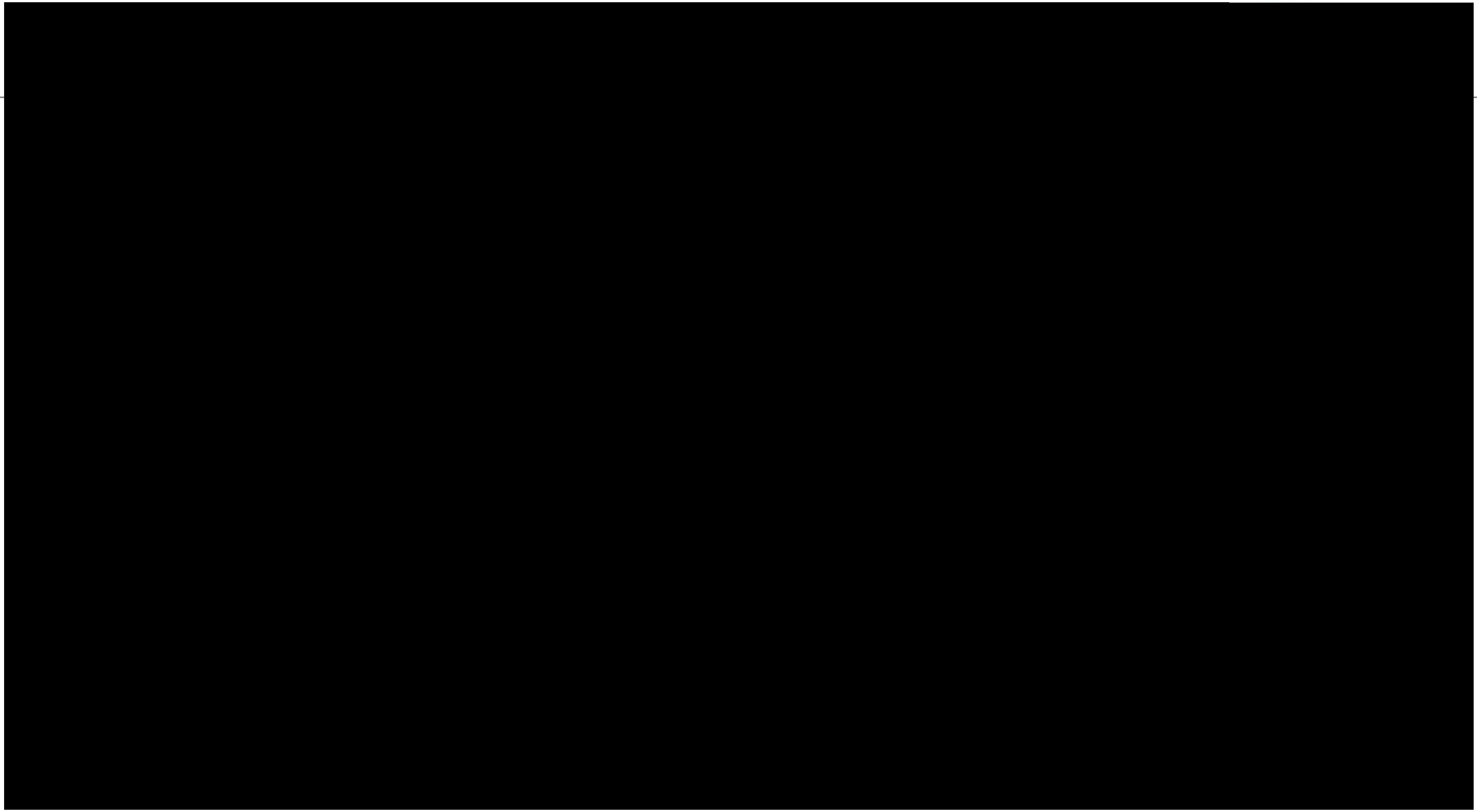

exerciseangina count plot
HeartDisease
0.0
1.0
count
0
50
100
150
200
250
300
350
N
Y
exerciseangina


ST_Slope count plot
HeartDisease
0.0
1.0
count
300
250
200
150
100
50
0
Up
Flat
Down
ST_Slope


RestingECG count plot
HeartDisease
0.0
1.0
count
250
200
150
100
50
0
Normal
ST
LVH
RestingECG

Plotly Interactive Graph





```
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, OrdinalEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
new_heart_data = heart["exerciseangina"].replace("N",0)
new_heart_data = heart["exerciseangina"].replace("Y",1)

new_heart_data = heart.replace("M",0)
new_heart_data = heart.replace("F",1,)


heart_data_with_dummies = pd.get_dummies(new_heart_data, drop_first=True)

y_1 = heart_data_with_dummies["HeartDisease"]
x_1 = heart_data_with_dummies.drop("HeartDisease", axis = 1)
```

Categorical Encoding

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	Sex_M	ChestPainType_ATA	ChestPainType_NAP	ChestPainType_TA	RestingECG_Normal	RestingECG_ST	exerciseangina_Y	ST_Slope_Flat	ST_Slope_Up
0	40.0	140.0	289.0	0.0	172.0	0.0	1	1	0	0	1	0	0	0	1
1	49.0	160.0	180.0	0.0	156.0	1.0	0	0	1	0	1	0	0	1	0
2	37.0	130.0	283.0	0.0	98.0	0.0	1	1	0	0	0	1	0	0	1
3	48.0	138.0	214.0	0.0	108.0	1.5	0	0	0	0	1	0	1	1	0
4	54.0	150.0	195.0	0.0	122.0	0.0	1	0	1	0	1	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
913	45.0	110.0	264.0	0.0	132.0	1.2	1	0	0	1	1	0	0	1	0
914	68.0	144.0	193.0	1.0	141.0	3.4	1	0	0	0	1	0	0	1	0
915	57.0	130.0	131.0	0.0	115.0	1.2	1	0	0	0	1	0	1	1	0
916	57.0	130.0	236.0	0.0	174.0	0.0	0	1	0	0	0	0	0	1	0
917	38.0	138.0	175.0	0.0	173.0	0.0	1	0	1	0	1	0	0	0	1

Data Scaling

```
scaler = StandardScaler()
scaler.fit(x_1)
scaled_inputs = scaler.transform(x_1)
scaled_inputs
```

```
array([[-1.35607325,  0.40398044,  0.7504942 , ..., -0.79074163,
        -0.95029534,  1.06655324],
       [-0.40865641,  1.5619801 , -1.09340492, ..., -0.79074163,
         1.05230444, -0.9375997 ],
       [-1.67187886, -0.17501939,  0.64899516, ..., -0.79074163,
        -0.95029534,  1.06655324],
       ...,
       [ 0.43349189, -0.17501939, -1.92231369, ...,  1.26463557,
         1.05230444, -0.9375997 ],
       [ 0.43349189, -0.17501939, -0.1460806 , ..., -0.79074163,
         1.05230444, -0.9375997 ],
       [-1.56661032,  0.28818048, -1.17798745, ..., -0.79074163,
        -0.95029534,  1.06655324]])
```

```
import numpy as np
from sklearn.model_selection import train_test_split
```

```
# split data
x_train, x_test, y_train, y_test, = train_test_split(scaled_inputs, y_1, random_state=42)
```

Logistic Regression for Classification

```
#######################################Logistic Regression###############################################
from sklearn.linear_model import LogisticRegression
```

```
lg_model = LogisticRegression()
clf = lg_model.fit(x_train, y_train)
```

```
x_test_prob = clf.predict_proba(x_test)
```

```
clf.score(x_test, y_test)
```

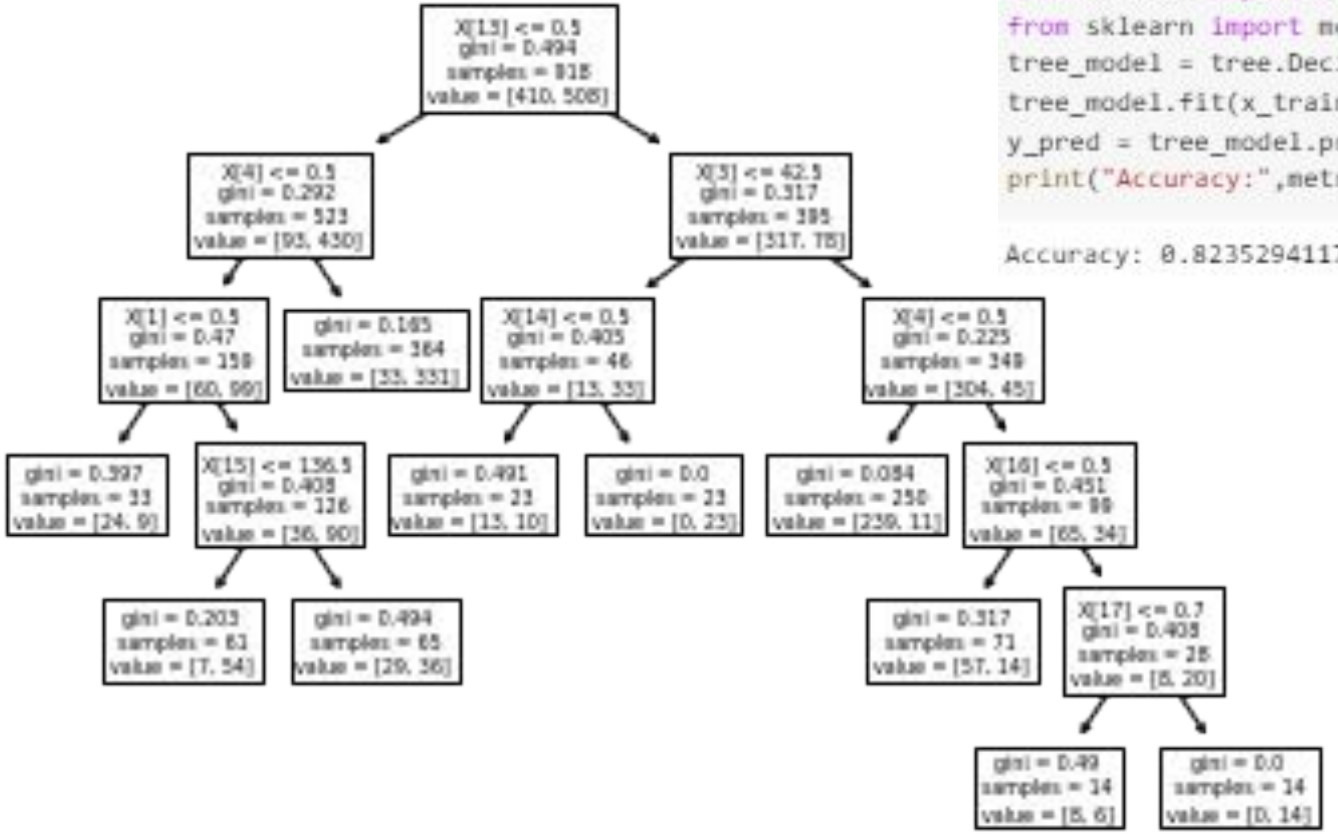
```
0.8716577540106952
```

Decision Trees

```python
heartX = heart[['Age', 'Sex', 'RestingBP', 'Cholesterol', 'ASY','ATA','NAP','TA','LVH','Normal','ST','Down','Flat','Up',
              'FastingBS', 'MaxHR', 'ExerciseAngina', 'Oldpeak']]
heartY = heart[['HeartDisease']]
```

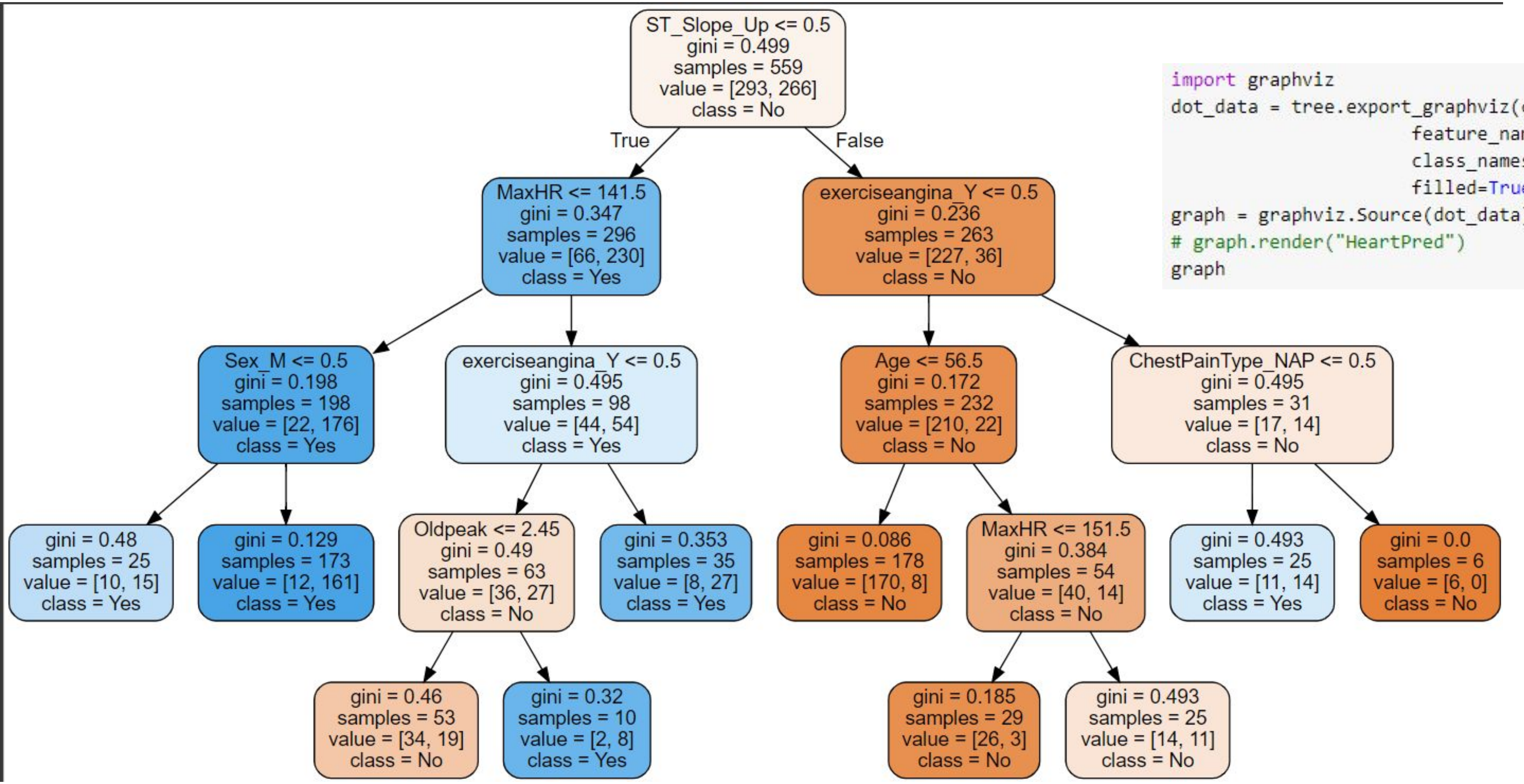
```python
from sklearn import tree
clf = tree.DecisionTreeClassifier(max_leaf_nodes = 10, min_samples_leaf = 5, max_depth= 5)
clf = clf.fit(heartX, heartY)
```

```python
tree.plot_tree(clf)
```



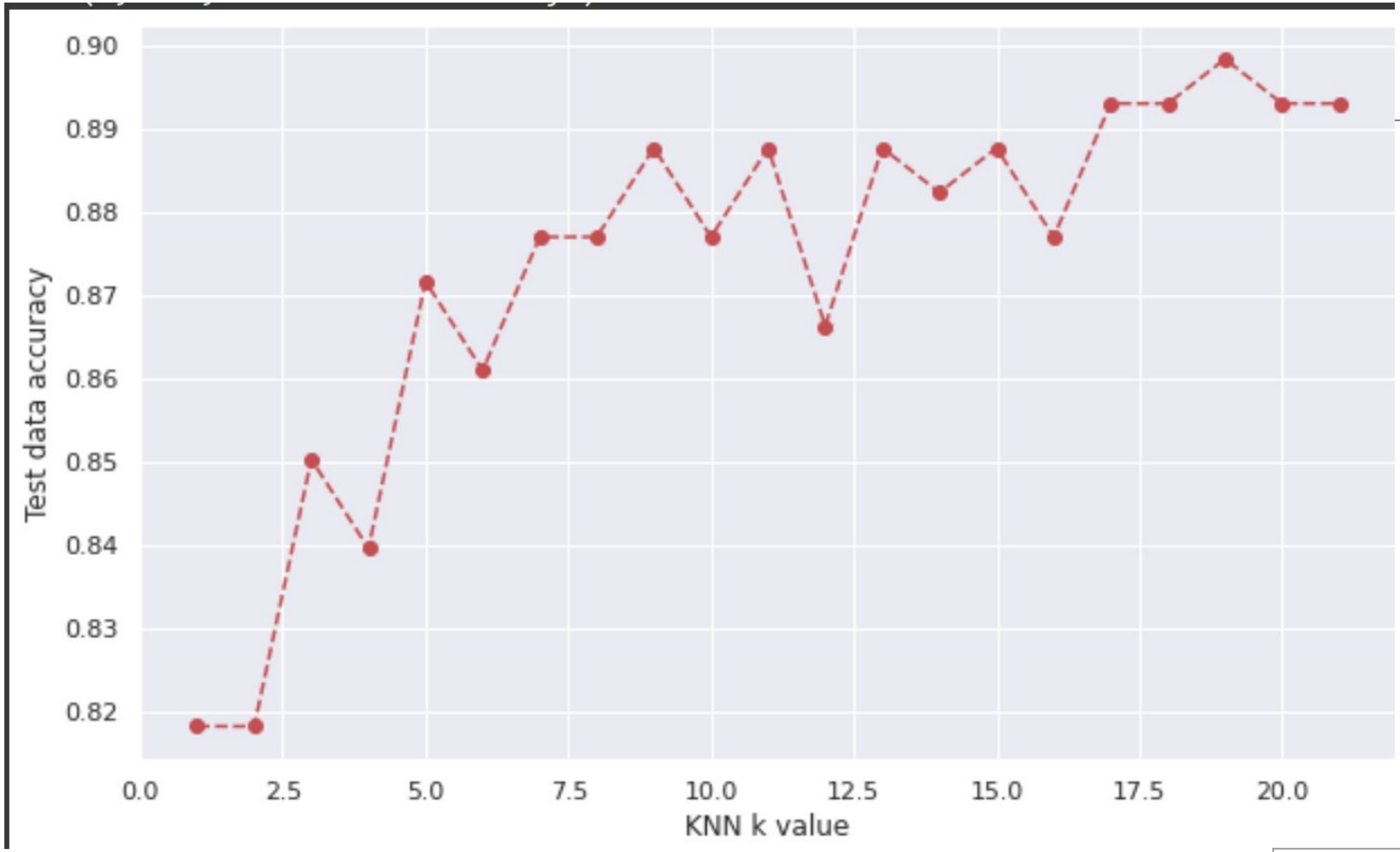

```python
from sklearn import tree
from sklearn import metrics
tree_model = tree.DecisionTreeClassifier(max_leaf_nodes = 10, min_samples_leaf = 5, max_depth= 5)
tree_model.fit(x_train, y_train)
y_pred = tree_model.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8235294117647058
```




```python
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None,
                    feature_names=heartX.columns,
                    class_names=['No','Yes'],
                    filled=True, rounded=True)
graph = graphviz.Source(dot_data)
# graph.render("HeartPred")
graph
```

KNN for Classification




```python
k_range = list(range(1,22,1))

score_list = []
score_dict = {}
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
for k in k_range:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    score_dict[k] = metrics.accuracy_score(y_test, y_pred)
    score_list.append(metrics.accuracy_score(y_test, y_pred))


print(score_list)

plt.plot(k_range, score_list,'--ro', label='Accuracy')
plt.xlabel("KNN k value")
plt.ylabel("Test data accuracy")
```

KNN	89.83% (k = 19)

Support Vector Machine (Linear Kernel)

```python
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix

#Create a svm Classifier
svm_linear_model = svm.SVC(C = 1, kernel='linear') # Linear Kernel

svm_linear_model.fit(x_train, y_train)

#Predict the response for test dataset
y_pred = svm_linear_model.predict(x_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.8235294117647058
              precision    recall  f1-score   support

         0.0       0.84      0.82      0.83        98
         1.0       0.80      0.83      0.82        89

    accuracy                           0.82       187
   macro avg       0.82      0.82      0.82       187
weighted avg       0.82      0.82      0.82       187
```

Conclusion

Based on the predictors used and the models we created, our methods for predicting heart disease were quite accurate. The most accurate model was KNN at a k value of 19.

Interestingly KNN for all k values was more accurate than the decision tree. Future analysis including family history, environmental factors, including smoking, secondhand smoke exposure, diet and alcoholism should be included.

Analysis Model	Best Accuracy
KNN	89.83% (k = 19)
Logistic Regression	87.16%
Decision Trees	82.3%
Support Vector Machine (Linear Kernel)	82.35%