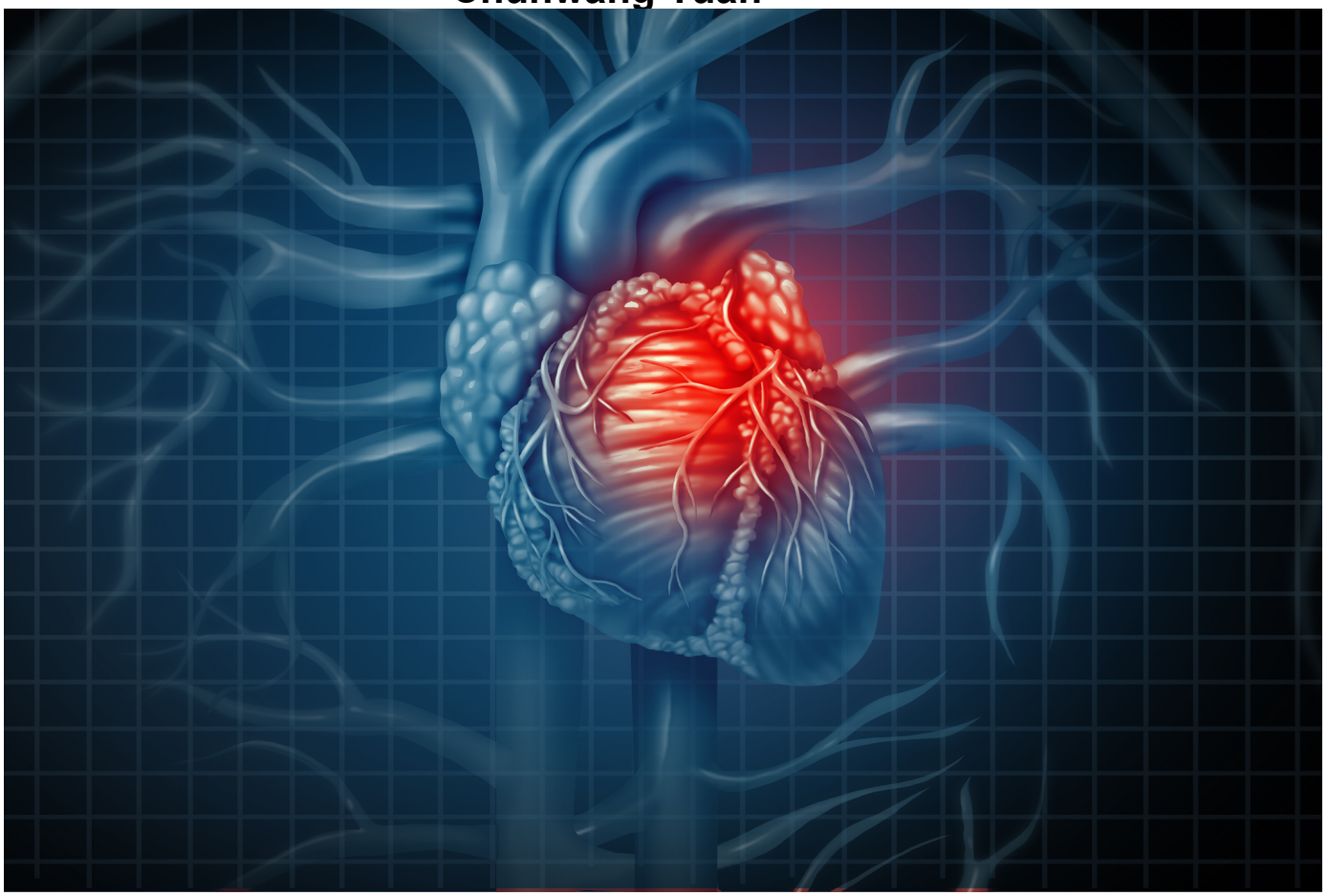
Heart Failure Prediction

Chunwang Yuan



Introduction

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Four out of 5CVD deaths are due to heart attacks and strokes. Heart failure is a common event caused by CVDs and this dataset contains 11 features that can be used to predict a possible heart disease.

People with cardiovascular disease or who are at high cardiovascular risk need early detection and management wherein a machine learning model can be of great help.

Data

Heart Failure Prediction Dataset

DATA DICTONARY

1 **Age**: Age of the patient [years]
2 **Sex**: Sex of the patient [M: Male, F: Female]
3 **ChestPainType**: [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**: 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]

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

What We need to do

- Based on the data and data dictionary, We have a classification problem.

- We wil make classification on the target variable **Heart Disease**
- And we need to build a model to get best calssification possible on the target variable.

Exploratory Data Analysis

- Let's import the libraries

In [1]:

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

from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

import statsmodels.api as sm
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier

import xgboost as xgb
from xgboost import XGBClassifier,plot_importance
from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.compose import make_column_transformer
from sklearn.metrics import accuracy_score,classification_report

import plotly.offline

import plotly
import plotly.express as px
import plotly.graph_objs as go
import plotly.offline as py
from plotly.offline import iplot
from plotly.subplots import make_subplots
import plotly.figure_factory as ff


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

In [2]:

```
df = pd.read_csv('heart.csv')
df.tail()
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG
913	45	M	TA	110	264	0	Norma
914	68	M	ASY	144	193	1	Norma
915	57	M	ASY	130	131	0	Norma
916	57	F	ATA	130	236	0	LVH
917	38	M	NAP	138	175	0	Norma

In [3]:

```
for column in df.columns:
    if df[column].dtype == type(object):
        le = sklearn.preprocessing.LabelEncoder()
        df[column] = le.fit_transform(df[column])
df.tail()
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG
913	45	1	3	110	264	0	1
914	68	1	0	144	193	1	1
915	57	1	0	130	131	0	1
916	57	0	1	130	236	0	0
917	38	1	2	138	175	0	1

In [4]:

```
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    int32  
 2   ChestPainType   918 non-null    int32  
 3   RestingBP       918 non-null    int64  
 4   Cholesterol     918 non-null    int64  
 5   FastingBS       918 non-null    int64  
 6   RestingECG      918 non-null    int32  
 7   MaxHR           918 non-null    int64  
 8   ExerciseAngina  918 non-null    int32  
 9   Oldpeak         918 non-null    float64
 10  ST_Slope        918 non-null    int32  
 11  HeartDisease    918 non-null    int64  
dtypes: float64(1), int32(5), int64(6)
memory usage: 68.3 KB
```

- Overall data types seem ok.

In [5]:

```
df.duplicated().sum()
```

Out[5]:

```
0
```

- No duplicates

In [6]:

```
def missing (df):
    missing_number = df.isnull().sum().sort_values(ascending=False)
    missing_percent = (df.isnull().sum()/df.isnull().count()).sort_values(asce
    missing_values = pd.concat([missing_number, missing_percent], axis=1, keys
    return missing_values

missing(df)
```

Out[6]:

	Missing_Number	Missing_Percent
HeartDisease	0	0.0
ST_Slope	0	0.0
Oldpeak	0	0.0
ExerciseAngina	0	0.0
MaxHR	0	0.0
RestingECG	0	0.0
FastingBS	0	0.0
Cholesterol	0	0.0
RestingBP	0	0.0
ChestPainType	0	0.0
Sex	0	0.0
Age	0	0.0

- No missing values.

Target Variable

In [7]:

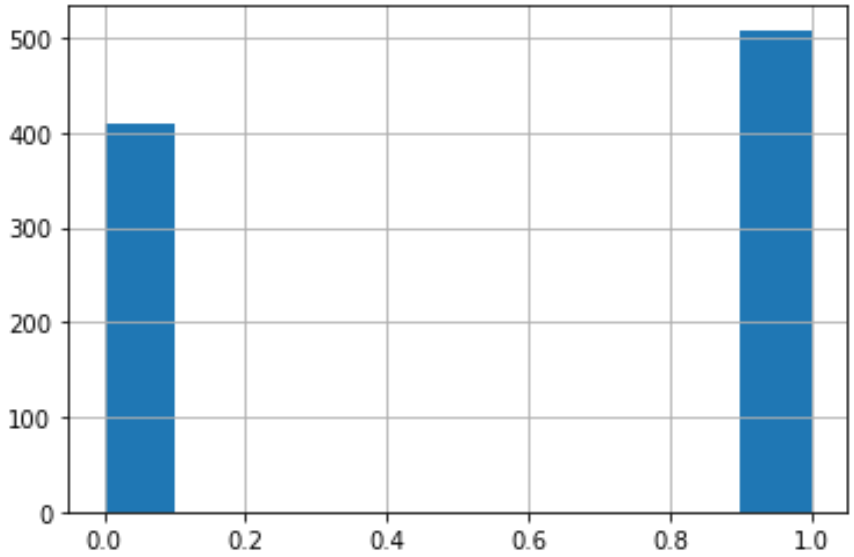
```
X = df.drop(columns = 'HeartDisease')
y = df['HeartDisease']
```

In [8]:

```
y.hist()
```

Out[8]:

```
<AxesSubplot:>
```



- Almost 500 patient had a heart disease.
- Almost 400 patient didn't have a heart disease.
- There is a little imblanace but nothing in the disturbing level.

Features

In [9]:

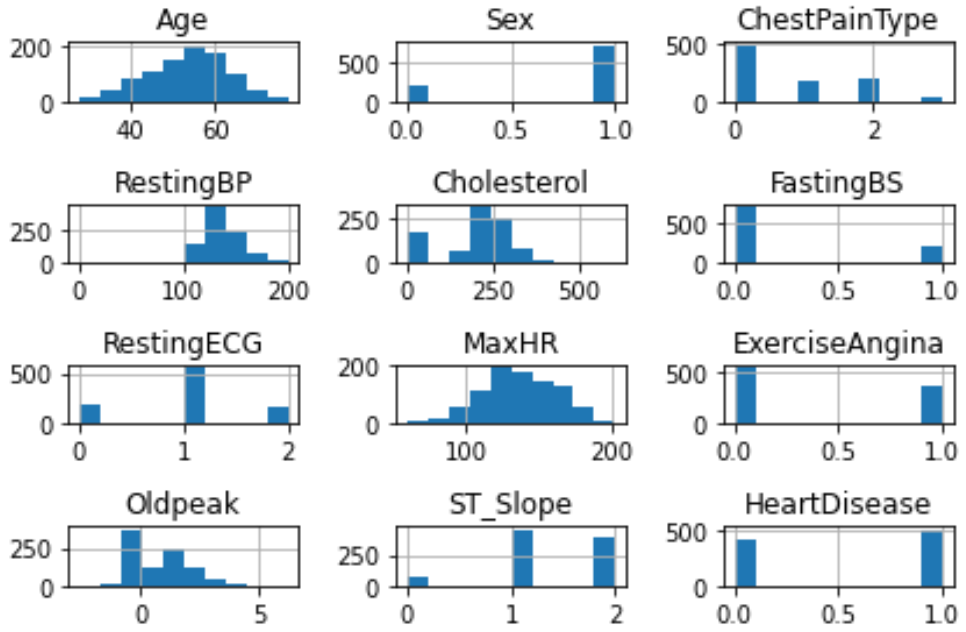
```
df.describe()
```

Out[9]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	Fastin
count	918.000000	918.000000	918.000000	918.000000	918.000000	918.000
mean	53.510893	0.789760	0.781046	132.396514	198.799564	0.233
std	9.432617	0.407701	0.956519	18.514154	109.384145	0.423
min	28.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	47.000000	1.000000	0.000000	120.000000	173.250000	0.000
50%	54.000000	1.000000	0.000000	130.000000	223.000000	0.000
75%	60.000000	1.000000	2.000000	140.000000	267.000000	0.000
max	77.000000	1.000000	3.000000	200.000000	603.000000	1.000

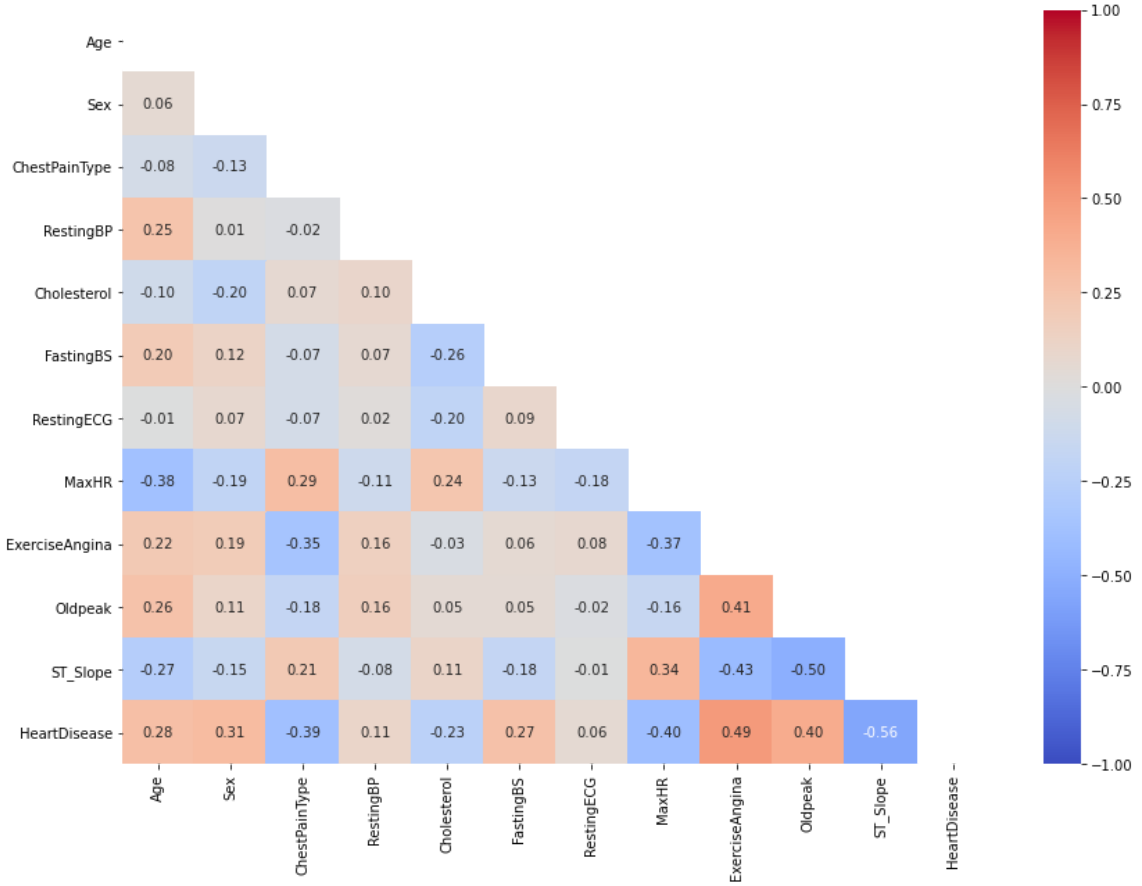
In [10]:

```
hists = df.hist()
plt.tight_layout()
plt.show()
```



In [11]:

```
matrix = np.triu(df.corr())
fig, ax = plt.subplots(figsize=(14,10))
sns.heatmap (df.corr(), annot=True, fmt= '.2f', vmin=-1, vmax=1, center=0, cma
```



- Based on the matrix, we can observe relatively strong positive correlation between the features(ExerciseAngina, Oldpeak,Sex) and HeartDisease.
- We can observe relatively strong negative correlation between the features(ST_Slope, MaxHR,ChestPainType) and HeartDisease.

Gender and Heart Disease

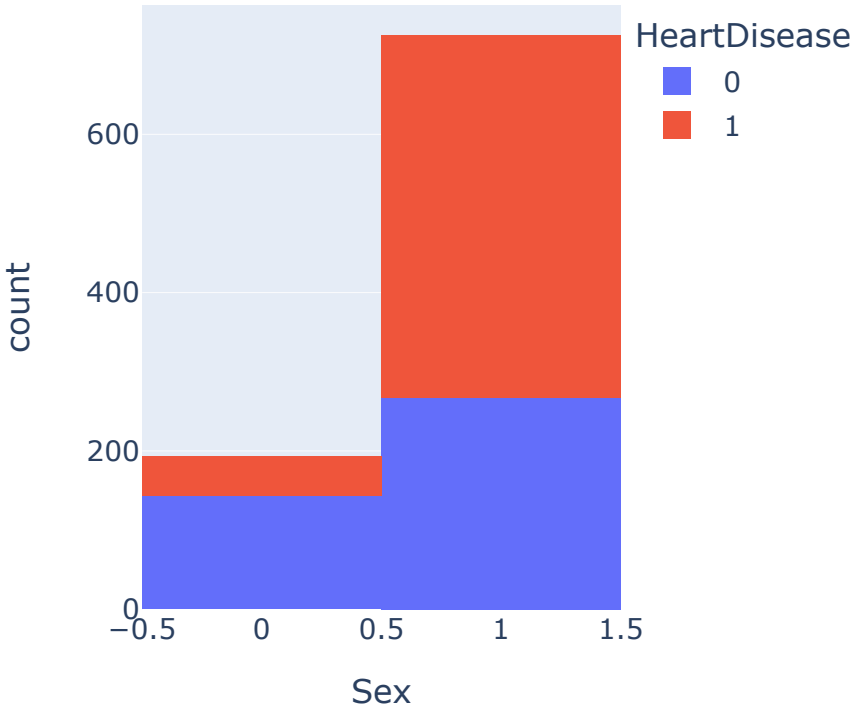
In [12]:

```
print (f'A female person has a probability of {round(df[df["Sex"]==0]["HeartDi

print()

print (f'A male person has a probability of {round(df[df["Sex"]==1]["HeartDise

print()
```

```
A female person has a probability of 25.91 % have a HeartDisease

A male person has a probability of 63.17 % have a HeartDisease
```

In [13]:

```
fig = px.histogram(df, x="Sex", color="HeartDisease",width=400, height=400)
fig.show()
```

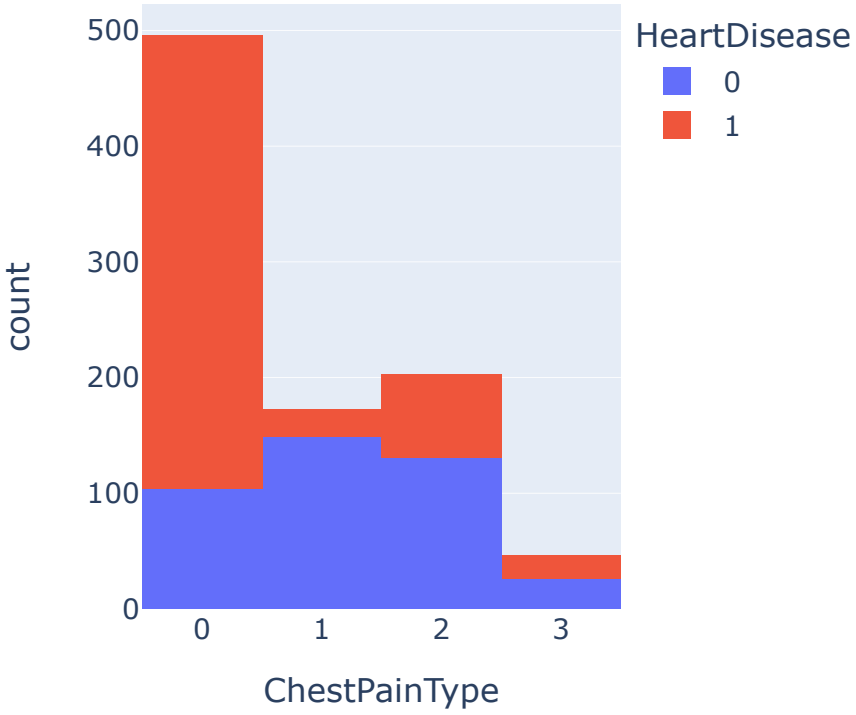


- Men are almost 2.5 times more likely have a heart disease than women.

Chest Pain Type and Heart Disease

In [14]:

```
fig = px.histogram(df, x="ChestPainType", color="HeartDisease",width=400, heig
fig.show()
```

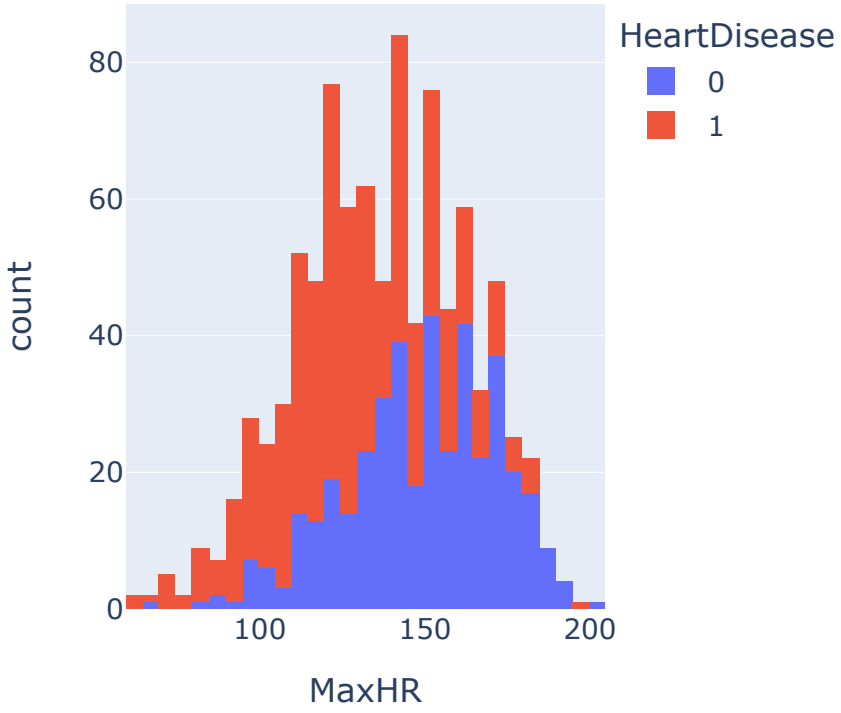


- We can observe clear differences among the chest pain type.
- By checking the original dataset, 0 of ChestPainType is ASY.
- Person with ASY: Asymptomatic chest pain has almost 6 times more likely have a heart disease than person with ATA Atypical Angina chest pain.

MaxHR and Heart Disease

In [15]:

```
fig = px.histogram(df, x="MaxHR", color="HeartDisease",width=400, height=400)
fig.show()
```

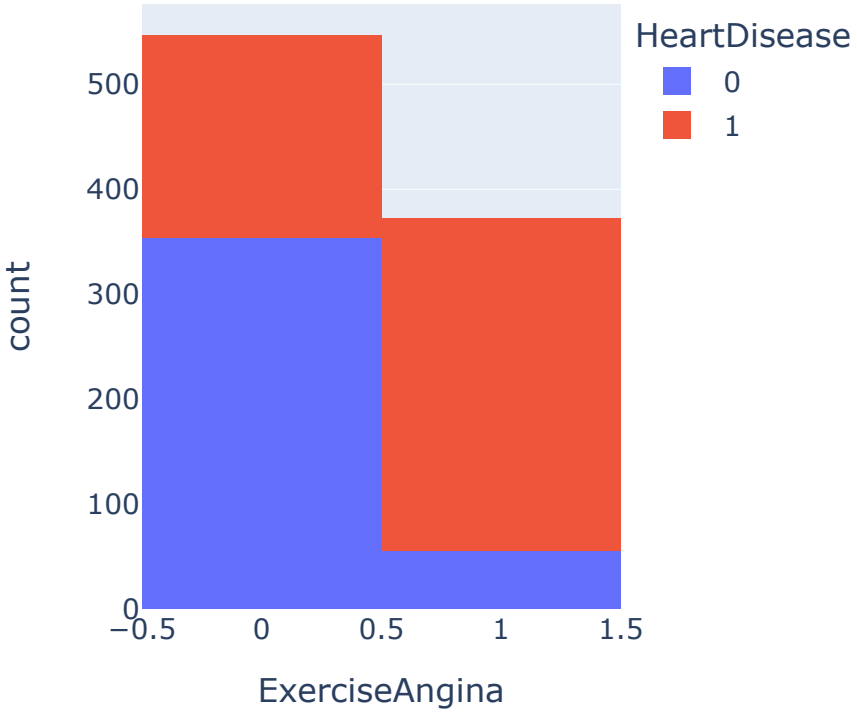


- Mean of Maximum heart rate on having HeartDisease is less than the mean of without HeartDisease.

ExerciseAngina and Heart Disease

In [16]:

```
fig = px.histogram(df, x="ExerciseAngina", color="HeartDisease",width=400, hei
fig.show()
```

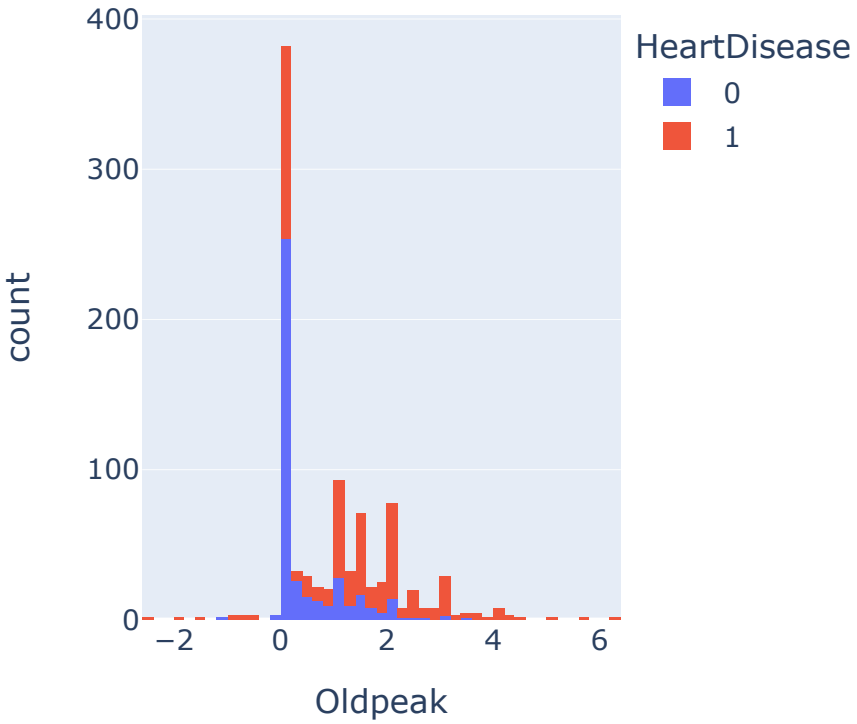


- ExerciseAngina: exercise-induced angina with 'Yes' almost 2.4 times more likley have a heart disaese than exercise-induced angina with 'No'

Oldpeak and Heart Disease

In [17]:

```
fig = px.histogram(df, x="Oldpeak", color="HeartDisease",width=400, height=400
fig.show()
```

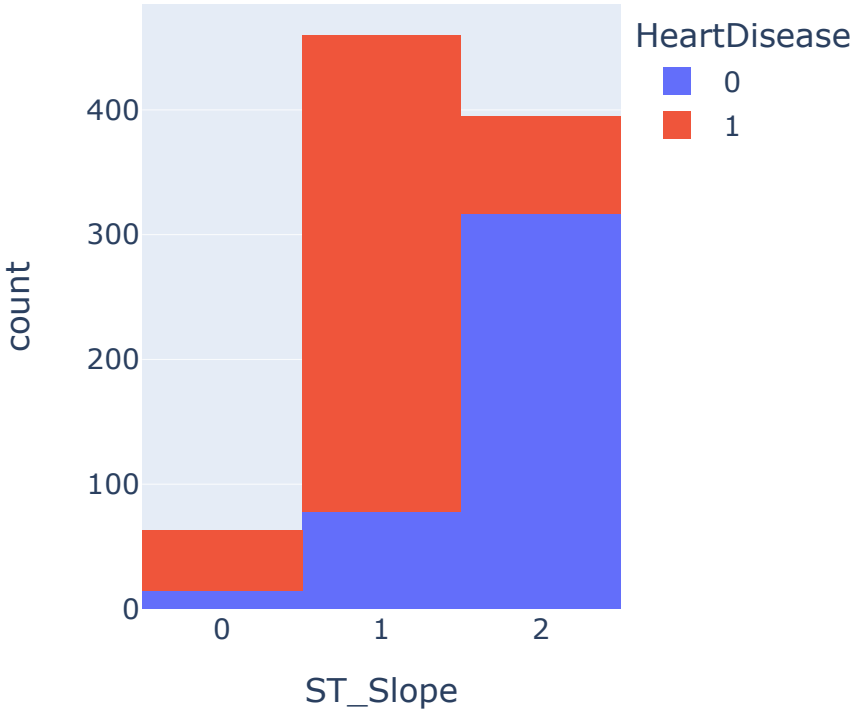


- Oldpeak: Oldpeak around 1~4 almost several times more likley have a heart disaese than other value.

ST_Slope and Heart Disease

In [18]:

```
fig = px.histogram(df, x="ST_Slope", color="HeartDisease",width=400, height=40
fig.show()
```



- ST_Slope: the slope of the peak exercise ST segment has differences.
- ST_Slope Up significantly less likely has heart disease than the other two segment.

Overall Insights from the Exploratory Data Analysis

- Target variable has close to balanced data.
- Oldpeak (depression related number) has a positive correlation with the heart disease.
- Maximum heart rate has negative correlation with the heart disease.
- Interestingly cholesterol has negative correlation with the heart disease.
- Based on the gender; Men are almost 2.44 times more likely have a heart disease than women.
- We can observe clear differences among the chest pain type.
- Person with ASY: Asymptomatic chest pain has almost 6 times more likely have a heart disease than person with ATA Atypical Angina chest pain.
- RestingECG: resting electrocardiogram results don't differ much.
- Person with ST: having ST-T wave abnormality is more likely have a heart disease than the others.

- ExerciseAngina: exercise-induced angina with 'Yes' almost 2.4 times more likley have a heart disaese than exercise-induced angina with 'No'
- ST_Slope: the slope of the peak exercise ST segment has differences.
- ST_Slope Up significantly less likely has heart disease than the other two segment.

MODEL SELECTION

- We'll use Logistic Regression, Random Forest, Support Vector Machine, KNeighbors, Decision Tree, Ada Boost, Bagging, Gradient Boosting and XGBoost models with scaler.
- And then we will use the top 3 best performance models to bulid a custom ensemble model in two layers with superlearner.
- Finally we will compare those models

Model Creation and Comparison

In [84]:

```
X = pd.DataFrame(StandardScaler().fit_transform(X), columns=X.columns)
```

In [85]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, rand
```

In [86]:

```
%%time
LR = LogisticRegression(solver='lbfgs', max_iter=10000, random_state=555)
RF = RandomForestClassifier(n_estimators = 100, random_state=555)
SVM = SVC(random_state=0, probability=True)
KNC = KNeighborsClassifier()
DTC = DecisionTreeClassifier()
ABC = AdaBoostClassifier(n_estimators = 100)
BC = BaggingClassifier(n_estimators = 100)
GBC = GradientBoostingClassifier(n_estimators = 100)
clf_XGB = XGBClassifier(n_estimators = 100, seed=555, use_label_encoder=False,
clfs = []
accs = {}
print('5-fold cross validation:\n')
for clf, label in zip([LR, RF, SVM, KNC, DTC, ABC, BC, GBC, clf_XGB],
                      ['Logistic Regression',
                       'Random Forest',
                       'Support Vector Machine',
                       'KNeighbors',
                       'Decision Tree',
                       'Ada Boost',
                       'Bagging',
                       'Gradient Boosting',
                       'XGBoost']):
    scores = sklearn.model_selection.cross_val_score(clf, X_train, y_train, cv
    print("Train CV Accuracy: %0.3f (+/- %0.3f) [%s]" % (scores.mean(), scores
    md = clf.fit(X_train, y_train)
    clfs.append(md)
    accuracy = sklearn.metrics.accuracy_score(clf.predict(X_test), y_test)
    accs[label] = accuracy
    print("Test Accuracy: %0.4f " % (accuracy))
```

```
5-fold cross validation:

Train CV Accuracy: 0.855 (+/- 0.017) [Logistic Regression]
Test Accuracy: 0.8348
Train CV Accuracy: 0.863 (+/- 0.017) [Random Forest]
Test Accuracy: 0.8652
Train CV Accuracy: 0.874 (+/- 0.020) [Support Vector Machine]
Test Accuracy: 0.8739
Train CV Accuracy: 0.847 (+/- 0.024) [KNeighbors]
Test Accuracy: 0.8696
Train CV Accuracy: 0.807 (+/- 0.020) [Decision Tree]
Test Accuracy: 0.8261
Train CV Accuracy: 0.844 (+/- 0.033) [Ada Boost]
Test Accuracy: 0.8565
Train CV Accuracy: 0.844 (+/- 0.040) [Bagging]
Test Accuracy: 0.8435
Train CV Accuracy: 0.862 (+/- 0.025) [Gradient Boosting]
Test Accuracy: 0.8652
Train CV Accuracy: 0.853 (+/- 0.021) [XGBoost]
```

```
Test Accuracy: 0.8826
Wall time: 18.2 s
```

- As the accuracy result shows, XGBoost model has the best performance on testing accuracy at 0.8826, following by Random Forest, Support Vector Machine, KNeighbors Ada Boost and Gradient Boosting models around 0.86. The Decision Tree model has the worst performance at 0.80.

Bulid a custom ensemble (superlearner) with best three of models

- The best three models above are XGBoost, Support Vector Machine and KNeighbors.

In [87]:

```
from mlens.ensemble import SuperLearner
from mlens.model_selection import Evaluator
from mlens.metrics import make_scorer

from sklearn.metrics import accuracy_score
```

In [88]:

```
ensemble = SuperLearner(scorer=accuracy_score, random_state=555, verbose=2)
ensemble.add([KNC, SVM,clf_XGB])
ensemble.add_meta(GradientBoostingClassifier())
ensemble.fit(X_train, y_train)
```

```
Fitting 2 layers
Processing layer-1             done | 00:00:00
Processing layer-2             done | 00:00:00
Fit complete                        | 00:00:01
```

Out[88]:

```
SuperLearner(array_check=None, backend=None, folds=2,
       layers=[Layer(backend='threading', dtype=<class 'numpy.fl
oat32'>, n_jobs=-1,
   name='layer-1', propagate_features=None, raise_on_exception=T
rue,
   random_state=4782, shuffle=False,
   stack=[Group(backend='threading', dtype=<class 'numpy.float3
2'>,
   indexer=FoldIndex(X=None, folds=2, raise_on_ex...4AFDC0>)],
   n_jobs=-1, name='group-37', raise_on_exception=True, transfor
mers=[])],
   verbose=1)],
       model_selection=False, n_jobs=None, raise_on_exception=Tr
ue,
       random_state=555, sample_size=20,
       scorer=<function accuracy_score at 0x0000020F044AFDC0>,
       shuffle=False, verbose=2)
```

In [67]:

```
print ("Accuracy - Train : ", sklearn.metrics.accuracy_score(ensemble.predict(
print ("Accuracy - Test : ", sklearn.metrics.accuracy_score(ensemble.predict(X
en_acc = accuracy_score(ensemble.predict(X_test), y_test)
```

```
Predicting 2 layers
Processing layer-1             done | 00:00:00
Processing layer-2             done | 00:00:00
Predict complete                    | 00:00:00
Accuracy - Train :  0.9127906976744186

Predicting 2 layers
Processing layer-1             done | 00:00:00
Processing layer-2             done | 00:00:00
Predict complete                    | 00:00:00
Accuracy - Test :  0.8782608695652174

Predicting 2 layers
Processing layer-1             done | 00:00:00
Processing layer-2             done | 00:00:00
Predict complete                    | 00:00:00
```

- Unfortunately, the ensemble of three models didn't perform better.

Neural Networks

- Beacuse the ensemble model didn't have better accuracy score, let's see the Neural Networks' performance.

In [91]:

```
nn_X_train = np.reshape(X_train.values, (X_train.shape[0], X_train.shape[1], 1
nn_X_test = np.reshape(X_test.values, (X_test.shape[0], X_test.shape[1], 1))
nn_X_train.shape
```

Out[91]:

```
(688, 11, 1)
```

In [102]:

```
import tensorflow as tf

# create model
mdl_k = tf.keras.Sequential([tf.keras.layers.Input(shape=(nn_X_train.shape[1],
mdl_k.add(tf.keras.layers.Dense(12, input_dim=3, activation='relu', kernel_ini
mdl_k.add(tf.keras.layers.Dropout(0.5))
mdl_k.add(tf.keras.layers.Dense(8, activation='sigmoid'))

mdl_k.add(tf.keras.layers.LSTM(64))
mdl_k.add(tf.keras.layers.LeakyReLU(alpha=0.2))
mdl_k.add(tf.keras.layers.Dense(1, activation='relu'))

# Compile model
mdl_k.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=1e-
```

In [103]:

```
mdl_k.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 11, 12)            24

 dropout_1 (Dropout)         (None, 11, 12)            0

 dense_4 (Dense)             (None, 11, 8)             104

 lstm_1 (LSTM)               (None, 64)                18688

 leaky_re_lu_1 (LeakyReLU)   (None, 64)                0

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 18,881
Trainable params: 18,881
Non-trainable params: 0
_________________________________________________________________
```

In [104]:

```
val_dataset = tf.data.Dataset.from_tensor_slices((nn_X_test, y_test)).batch(ba
```

In [105]:

```
%%time
history = mdl_k.fit(
    nn_X_train,
    y_train,
    epochs=100,
    batch_size=128,
    validation_data=val_dataset,
    use_multiprocessing=True
)
```

```
Epoch 1/100
6/6 [==============================] - 6s 259ms/step - loss:
0.3403 - accuracy: 0.4433 - val_loss: 0.2485 - val_accuracy:
0.5522
Epoch 2/100
6/6 [==============================] - 0s 39ms/step - loss:
0.2647 - accuracy: 0.5538 - val_loss: 0.2560 - val_accuracy:
0.5522
Epoch 3/100
6/6 [==============================] - 0s 36ms/step - loss:
0.2487 - accuracy: 0.5538 - val_loss: 0.2464 - val_accuracy:
0.5522
Epoch 4/100
6/6 [==============================] - 0s 37ms/step - loss:
0.2501 - accuracy: 0.4884 - val_loss: 0.2481 - val_accuracy:
0.4870
Epoch 5/100
6/6 [==============================] - 0s 36ms/step - loss:
0.2480 - accuracy: 0.5276 - val_loss: 0.2435 - val_accuracy:
```

In [106]:

```
mdl_k.evaluate(nn_X_test, y_test)
```

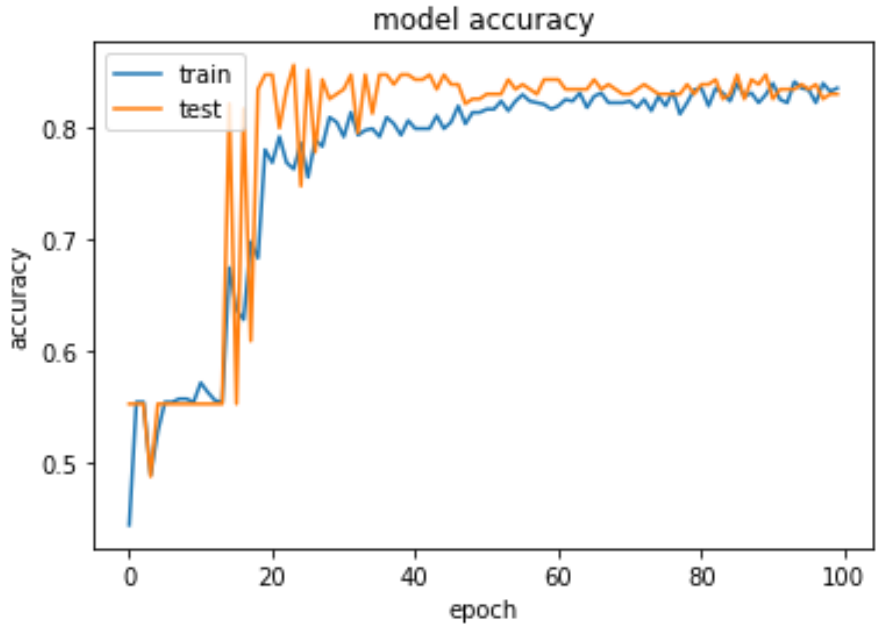
```
8/8 [==============================] - 1s 11ms/step - loss: 0.12
69 - accuracy: 0.8304
```

Out[106]:

```
[0.1269000768661499, 0.8304347991943359]
```

In [107]:

```
import matplotlib.pyplot as plt
%matplotlib inline
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

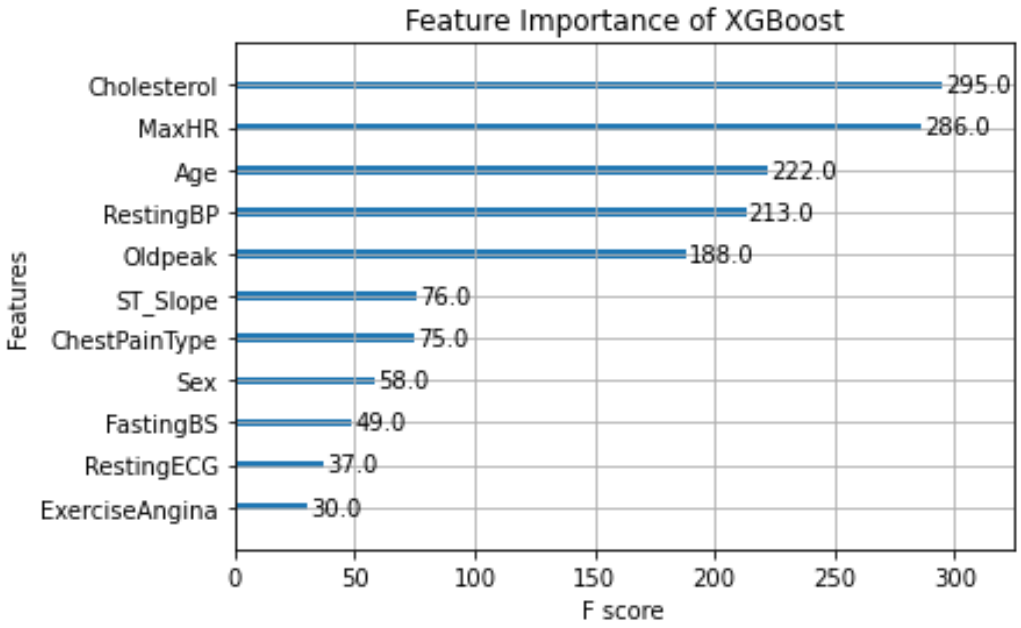


- Under current parameters, the result turns out that the neural networks didn't perform well.

Feature Importance

In [90]:

```
feature_importance = plot_importance(clf_XGB.fit(X_train, y_train), title = 'F
```



- The plot of Feature Importance of XGBoost model illustrated that the Cholesterol and MaxHR have a strong impact on Heart Disease, following by Age, RestingBP and Oldpeak.

Conclusion

- We have developed model to classifiy heart disease cases.
- By comparing the results of models above, we can conclude that the XGBoost model is the best model for this dataset and got a good accuracy of prediction at 0.88.

THANK YOU 🤘