In [26]:
```python
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, accuracy_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.preprocessing import OrdinalEncoder
from sklearn.ensemble import RandomForestClassifier
```

In [12]:
```python
#Creating a dataframe from the data
heart_data = pd.read_csv(r"C:\Users\alexa\OneDrive\Documents\heart\heart.csv")
```

In [13]:
```python
#Looking at the different types of data stored in each column to determine if there are any categoricals that w
heart_data.dtypes
```

Out[13]:
```
Age                 int64
Sex                object
ChestPainType      object
RestingBP           int64
Cholesterol         int64
FastingBS           int64
RestingECG         object
MaxHR               int64
ExerciseAngina     object
Oldpeak           float64
ST_Slope           object
HeartDisease        int64
dtype: object
```

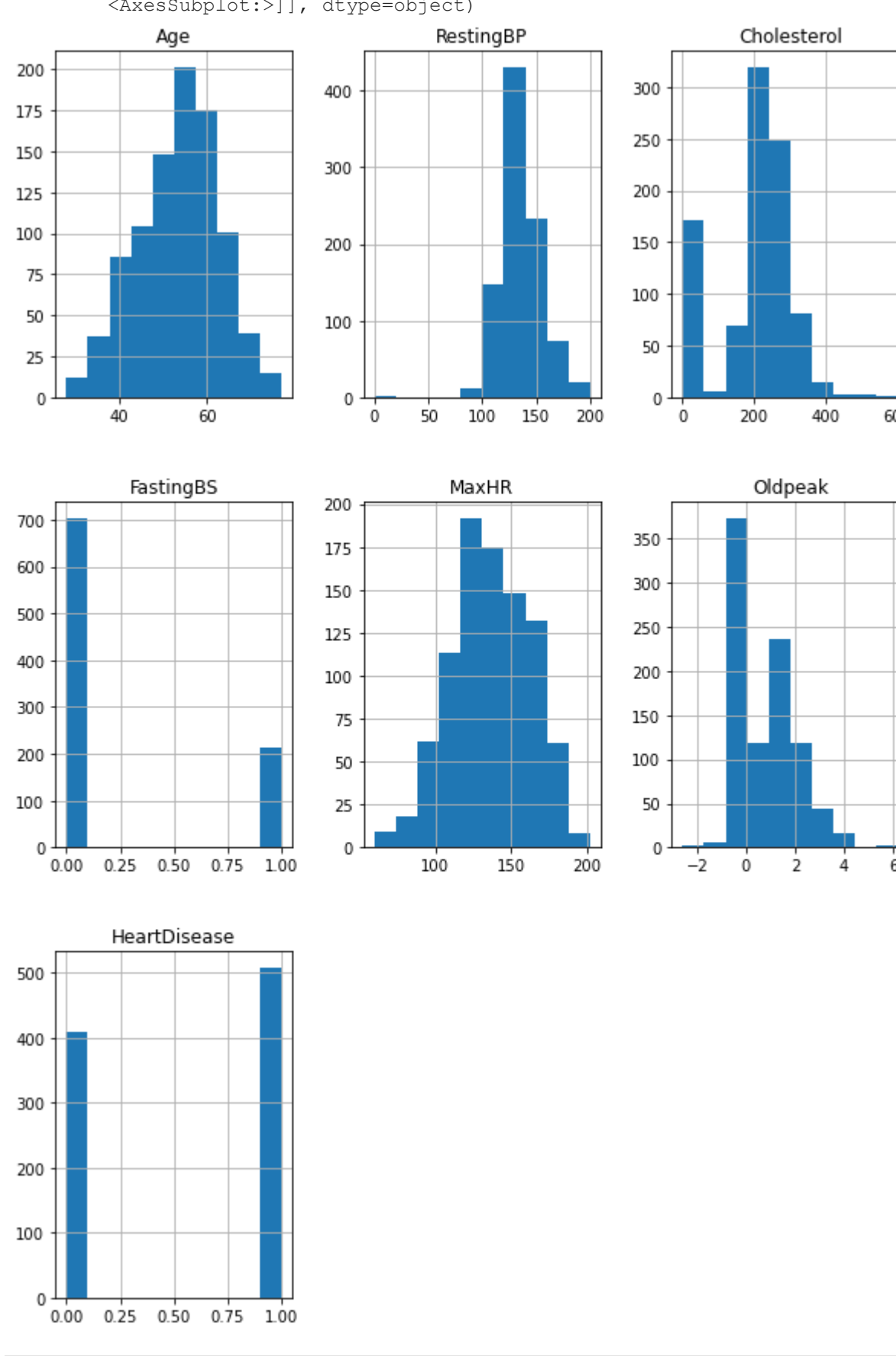
In [14]:
```python
#A quick view of the data
heart_data.head()
```

Out[14]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

In [15]:
```python
#Making a histogram that shows the distribution of the data
heart_data.hist(figsize = (10,15))
```

Out[15]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'RestingBP'}>,
        <AxesSubplot:title={'center':'Cholesterol'}>],
       [<AxesSubplot:title={'center':'FastingBS'}>,
        <AxesSubplot:title={'center':'MaxHR'}>,
        <AxesSubplot:title={'center':'Oldpeak'}>],
       [<AxesSubplot:title={'center':'HeartDisease'}>, <AxesSubplot:>,
        <AxesSubplot:>]], dtype=object)
```

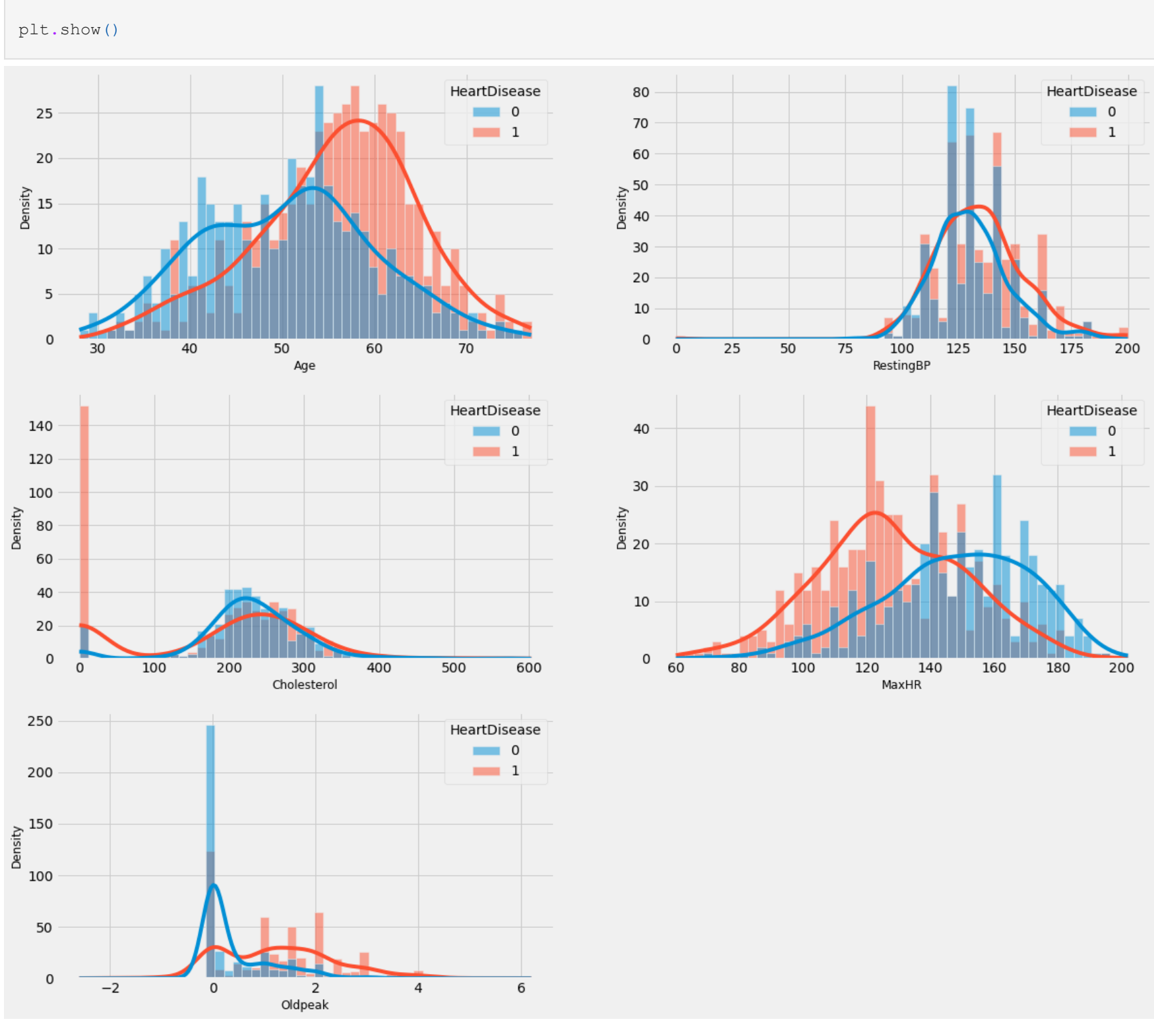


In [16]:
```python
#Setting up features to be plotted in the next code block
discrette_feature = [i for i in heart_data.columns if heart_data[i].nunique() < 10]
continuous_feature = [i for i in heart_data.columns if heart_data[i].nunique() > 10]
print(f'discrette feature: {discrette_feature}')
print(f'continuous feature: {continuous_feature}')
```

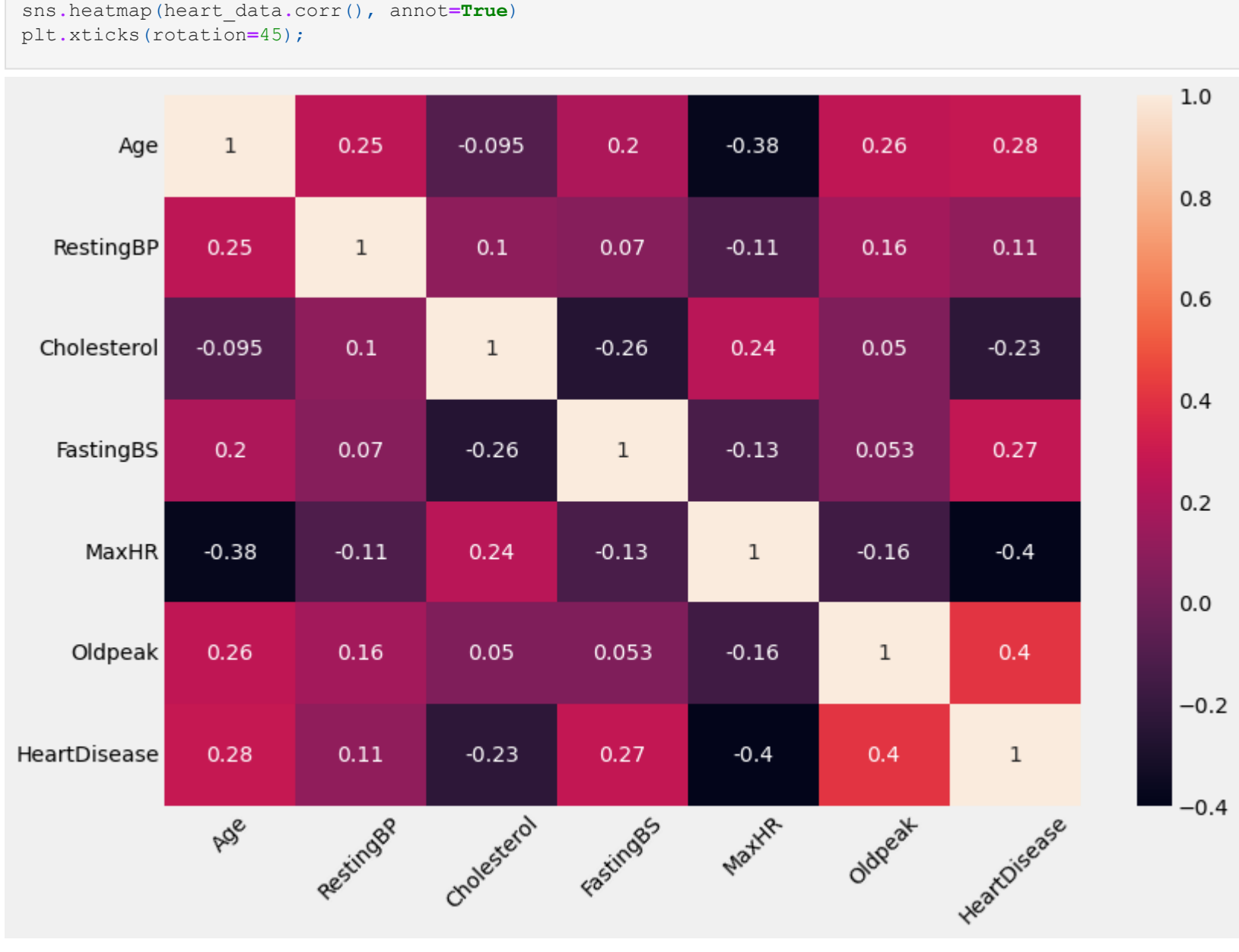
```
discrette feature: ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'ExerciseAngina', 'ST_Slope', 'HeartDise
ase']
continuous feature: ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
```

In [17]:
```python
#Potting continuous features of the dataset as histograms showing distributions for both those who had heart fa
plt.style.use('fivethirtyeight')
i = 1
plt.figure(figsize = (18,16))
for feature in continuous_feature:
    plt.subplot(3, 2, i)
    sns.histplot(x=heart_data[feature],kde=True,bins = 50, hue = heart_data.HeartDisease)
    plt.xlabel(feature,size = 12)
    plt.ylabel("Density",size = 12)
    i += 1

plt.show()
```



In [25]:
```python
#Creating a visualization of the correlation between variables in the data set
plt.figure(figsize=(12, 8))
sns.heatmap(heart_data.corr(), annot=True)
plt.xticks(rotation=45);
```



From this visualization we can see that the variables most strongly correlated to heart disease are old peak and max heart rate.

In [18]:
```python
#Setting the target for prediction and features that will be used to train the network
y = heart_data['HeartDisease']
X = heart_data.drop(['HeartDisease'], axis=1)

train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.8, test_size=0.2)
```

In [19]:
```python
#Using ordinal encoder to convert categoricals to numerical values for network
s = (heart_data.dtypes == 'object')
object_cols = list(s[s].index)

label_train_X = train_X.copy()
label_test_X = test_X.copy()

ordinal_encoder = OrdinalEncoder()
label_train_X[object_cols] = ordinal_encoder.fit_transform(train_X[object_cols])
label_test_X[object_cols] = ordinal_encoder.transform(label_test_X[object_cols])
```

In [20]:
```python
#Checking that encoding worked
label_train_X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 734 entries, 498 to 790
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Age             734 non-null    int64  
 1   Sex             734 non-null    float64
 2   ChestPainType   734 non-null    float64
 3   RestingBP       734 non-null    int64  
 4   Cholesterol     734 non-null    int64  
 5   FastingBS       734 non-null    int64  
 6   RestingECG      734 non-null    float64
 7   MaxHR           734 non-null    int64  
 8   ExerciseAngina  734 non-null    float64
 9   Oldpeak         734 non-null    float64
 10  ST_Slope        734 non-null    float64
dtypes: float64(6), int64(5)
memory usage: 68.8 KB
```

In [21]:
```python
#Scaling the data for use in the network
scaler = StandardScaler()

label_train_X = scaler.fit_transform(label_train_X)
label_test_X = scaler.transform(label_test_X)
```

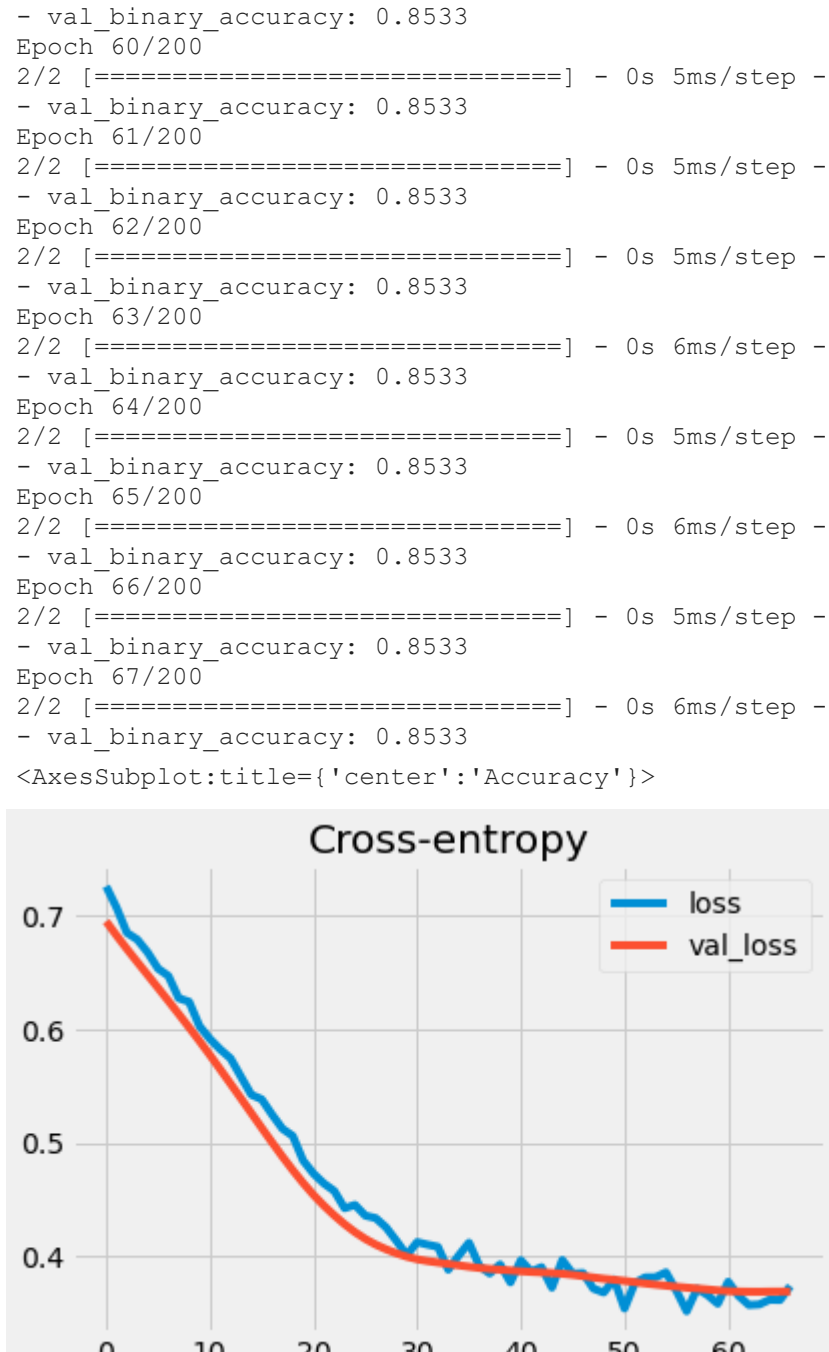
In [22]:
```python
#Building the structure of the model
model = keras.Sequential([
    layers.Dense(32, activation = 'swish', input_shape = [11]),
    layers.Dropout(.3),
    layers.Dense(32, activation = 'swish'),
    layers.Dropout(.3),
    layers.Dense(32, activation = 'swish'),
    layers.Dropout(.3),
    layers.Dense(1, activation = 'sigmoid'),
])
```

In [23]:
```python
#Adding methods for optimizing and evaluating the model
model.compile(
    optimizer = 'adam',
    loss = 'binary_crossentropy',
    metrics = ['binary_accuracy']
)
```

In [24]:
```python
#Adding a callback that will stop the model at best values.
early_stopping = keras.callbacks.EarlyStopping(
    patience=5,
    min_delta=0.001,
    restore_best_weights=True,
)
#Training the model
history = model.fit(
    label_train_X, train_y,
    validation_data=(label_test_X, test_y),
    batch_size=576,
    epochs=200,
    callbacks=[early_stopping],
)
#Plotting the learning curves
history_df = pd.DataFrame(history.history)
history_df.loc[:, ['loss', 'val_loss']].plot(title="Cross-entropy")
history_df.loc[:, ['binary_accuracy', 'val_binary_accuracy']].plot(title="Accuracy")
```

```
Epoch 1/200
2/2 [==============================] - 0s 68ms/step - loss: 0.7262 - binary_accuracy: 0.4237 - val_loss: 0.6953
- val_binary_accuracy: 0.5326
Epoch 2/200
2/2 [==============================] - 0s 6ms/step - loss: 0.7075 - binary_accuracy: 0.4837 - val_loss: 0.6829
- val_binary_accuracy: 0.6250
Epoch 3/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6853 - binary_accuracy: 0.5463 - val_loss: 0.6710
- val_binary_accuracy: 0.7228
Epoch 4/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6793 - binary_accuracy: 0.5749 - val_loss: 0.6595
- val_binary_accuracy: 0.7663
Epoch 5/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6680 - binary_accuracy: 0.6213 - val_loss: 0.6481
- val_binary_accuracy: 0.7935
Epoch 6/200
2/2 [==============================] - 0s 5ms/step - loss: 0.6537 - binary_accuracy: 0.6431 - val_loss: 0.6367
- val_binary_accuracy: 0.7989
Epoch 7/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6472 - binary_accuracy: 0.6662 - val_loss: 0.6252
- val_binary_accuracy: 0.8098
Epoch 8/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6276 - binary_accuracy: 0.7057 - val_loss: 0.6136
- val_binary_accuracy: 0.8152
Epoch 9/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6245 - binary_accuracy: 0.7003 - val_loss: 0.6018
- val_binary_accuracy: 0.8098
Epoch 10/200
2/2 [==============================] - 0s 6ms/step - loss: 0.6031 - binary_accuracy: 0.7343 - val_loss: 0.5899
- val_binary_accuracy: 0.8098
Epoch 11/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5918 - binary_accuracy: 0.7398 - val_loss: 0.5775
- val_binary_accuracy: 0.8152
Epoch 12/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5825 - binary_accuracy: 0.7520 - val_loss: 0.5649
- val_binary_accuracy: 0.8152
Epoch 13/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5745 - binary_accuracy: 0.7779 - val_loss: 0.5521
- val_binary_accuracy: 0.8152
Epoch 14/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5584 - binary_accuracy: 0.7766 - val_loss: 0.5393
- val_binary_accuracy: 0.8152
Epoch 15/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5425 - binary_accuracy: 0.7997 - val_loss: 0.5263
- val_binary_accuracy: 0.8207
Epoch 16/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5385 - binary_accuracy: 0.7752 - val_loss: 0.5133
- val_binary_accuracy: 0.8207
Epoch 17/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5250 - binary_accuracy: 0.7916 - val_loss: 0.5005
- val_binary_accuracy: 0.8207
Epoch 18/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5126 - binary_accuracy: 0.7929 - val_loss: 0.4880
- val_binary_accuracy: 0.8207
Epoch 19/200
2/2 [==============================] - 0s 6ms/step - loss: 0.5061 - binary_accuracy: 0.8202 - val_loss: 0.4760
- val_binary_accuracy: 0.8207
Epoch 20/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4844 - binary_accuracy: 0.8202 - val_loss: 0.4647
- val_binary_accuracy: 0.8207
Epoch 21/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4729 - binary_accuracy: 0.8215 - val_loss: 0.4542
- val_binary_accuracy: 0.8261
Epoch 22/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4643 - binary_accuracy: 0.8052 - val_loss: 0.4447
- val_binary_accuracy: 0.8315
Epoch 23/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4577 - binary_accuracy: 0.8161 - val_loss: 0.4361
- val_binary_accuracy: 0.8261
Epoch 24/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4424 - binary_accuracy: 0.8174 - val_loss: 0.4283
- val_binary_accuracy: 0.8261
Epoch 25/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4454 - binary_accuracy: 0.8188 - val_loss: 0.4213
- val_binary_accuracy: 0.8261
Epoch 26/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4359 - binary_accuracy: 0.8174 - val_loss: 0.4153
- val_binary_accuracy: 0.8261
Epoch 27/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4337 - binary_accuracy: 0.8134 - val_loss: 0.4104
- val_binary_accuracy: 0.8261
Epoch 28/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4255 - binary_accuracy: 0.8283 - val_loss: 0.4062
- val_binary_accuracy: 0.8261
Epoch 29/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4137 - binary_accuracy: 0.8161 - val_loss: 0.4026
- val_binary_accuracy: 0.8315
Epoch 30/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4011 - binary_accuracy: 0.8365 - val_loss: 0.3997
- val_binary_accuracy: 0.8370
Epoch 31/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4125 - binary_accuracy: 0.8379 - val_loss: 0.3974
- val_binary_accuracy: 0.8370
Epoch 32/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4103 - binary_accuracy: 0.8338 - val_loss: 0.3958
- val_binary_accuracy: 0.8478
Epoch 33/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4084 - binary_accuracy: 0.8256 - val_loss: 0.3945
- val_binary_accuracy: 0.8478
Epoch 34/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3882 - binary_accuracy: 0.8501 - val_loss: 0.3933
- val_binary_accuracy: 0.8533
Epoch 35/200
2/2 [==============================] - 0s 7ms/step - loss: 0.4006 - binary_accuracy: 0.8365 - val_loss: 0.3921
- val_binary_accuracy: 0.8533
Epoch 36/200
2/2 [==============================] - 0s 6ms/step - loss: 0.4120 - binary_accuracy: 0.8338 - val_loss: 0.3908
- val_binary_accuracy: 0.8533
Epoch 37/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3915 - binary_accuracy: 0.8392 - val_loss: 0.3897
- val_binary_accuracy: 0.8533
Epoch 38/200
2/2 [==============================] - 0s 7ms/step - loss: 0.3849 - binary_accuracy: 0.8379 - val_loss: 0.3889
- val_binary_accuracy: 0.8533
Epoch 39/200
2/2 [==============================] - 0s 7ms/step - loss: 0.3929 - binary_accuracy: 0.8256 - val_loss: 0.3882
- val_binary_accuracy: 0.8533
Epoch 40/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3770 - binary_accuracy: 0.8365 - val_loss: 0.3875
- val_binary_accuracy: 0.8533
Epoch 41/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3966 - binary_accuracy: 0.8433 - val_loss: 0.3868
- val_binary_accuracy: 0.8533
Epoch 42/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3866 - binary_accuracy: 0.8433 - val_loss: 0.3863
- val_binary_accuracy: 0.8478
Epoch 43/200
2/2 [==============================] - 0s 7ms/step - loss: 0.3905 - binary_accuracy: 0.8488 - val_loss: 0.3858
- val_binary_accuracy: 0.8424
Epoch 44/200
2/2 [==============================] - 0s 10ms/step - loss: 0.3724 - binary_accuracy: 0.8420 - val_loss: 0.3853
- val_binary_accuracy: 0.8424
Epoch 45/200
2/2 [==============================] - 0s 8ms/step - loss: 0.3968 - binary_accuracy: 0.8420 - val_loss: 0.3847
- val_binary_accuracy: 0.8424
Epoch 46/200
2/2 [==============================] - 0s 7ms/step - loss: 0.3848 - binary_accuracy: 0.8433 - val_loss: 0.3837
- val_binary_accuracy: 0.8424
Epoch 47/200
2/2 [==============================] - 0s 8ms/step - loss: 0.3853 - binary_accuracy: 0.8338 - val_loss: 0.3826
- val_binary_accuracy: 0.8424
Epoch 48/200
2/2 [==============================] - 0s 8ms/step - loss: 0.3715 - binary_accuracy: 0.8447 - val_loss: 0.3814
- val_binary_accuracy: 0.8478
Epoch 49/200
2/2 [==============================] - 0s 9ms/step - loss: 0.3681 - binary_accuracy: 0.8420 - val_loss: 0.3802
- val_binary_accuracy: 0.8478
Epoch 50/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3797 - binary_accuracy: 0.8529 - val_loss: 0.3793
- val_binary_accuracy: 0.8478
Epoch 51/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3540 - binary_accuracy: 0.8501 - val_loss: 0.3784
- val_binary_accuracy: 0.8533
Epoch 52/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3766 - binary_accuracy: 0.8433 - val_loss: 0.3773
- val_binary_accuracy: 0.8533
Epoch 53/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3815 - binary_accuracy: 0.8447 - val_loss: 0.3761
- val_binary_accuracy: 0.8478
Epoch 54/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3816 - binary_accuracy: 0.8460 - val_loss: 0.3751
- val_binary_accuracy: 0.8478
Epoch 55/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3859 - binary_accuracy: 0.8447 - val_loss: 0.3741
- val_binary_accuracy: 0.8533
Epoch 56/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3698 - binary_accuracy: 0.8447 - val_loss: 0.3732
- val_binary_accuracy: 0.8533
Epoch 57/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3516 - binary_accuracy: 0.8488 - val_loss: 0.3724
- val_binary_accuracy: 0.8533
Epoch 58/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3715 - binary_accuracy: 0.8460 - val_loss: 0.3716
- val_binary_accuracy: 0.8533
Epoch 59/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3659 - binary_accuracy: 0.8610 - val_loss: 0.3708
- val_binary_accuracy: 0.8533
Epoch 60/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3582 - binary_accuracy: 0.8460 - val_loss: 0.3700
- val_binary_accuracy: 0.8533
Epoch 61/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3776 - binary_accuracy: 0.8406 - val_loss: 0.3693
- val_binary_accuracy: 0.8533
Epoch 62/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3652 - binary_accuracy: 0.8474 - val_loss: 0.3690
- val_binary_accuracy: 0.8533
Epoch 63/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3571 - binary_accuracy: 0.8488 - val_loss: 0.3688
- val_binary_accuracy: 0.8533
Epoch 64/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3576 - binary_accuracy: 0.8597 - val_loss: 0.3688
- val_binary_accuracy: 0.8533
Epoch 65/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3617 - binary_accuracy: 0.8529 - val_loss: 0.3689
- val_binary_accuracy: 0.8533
Epoch 66/200
2/2 [==============================] - 0s 5ms/step - loss: 0.3615 - binary_accuracy: 0.8597 - val_loss: 0.3690
- val_binary_accuracy: 0.8533
Epoch 67/200
2/2 [==============================] - 0s 6ms/step - loss: 0.3738 - binary_accuracy: 0.8542 - val_loss: 0.3689
- val_binary_accuracy: 0.8533
```

Out[24]:
```
<AxesSubplot:title={'center':'Accuracy'}>
```



The network predicts for heart failure with an accuracy of around 85%

In [27]:
```python
#Creating a random forest classifier to compare to the neural network
RFC = RandomForestClassifier(random_state = 1)
RFC.fit(label_train_X, train_y)
RFC_predictions = RFC.predict(label_test_X)
RFC_acc = accuracy_score(test_y, RFC_predictions)
RFC_recall = recall_score(test_y, RFC_predictions)
print(RFC_acc)
print(RFC_recall)
```

```
0.875
0.9207920792079208
```

The random forest classifier appears to have a higher accuracy than the network at 87.5% accuracy. Although the difference is small the random forrest was easier to setup and required less computation time as well which are both important factors.