In [1]:
```
!pip install -q hvplot
```

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import hvplot.pandas
from scipy import stats

%matplotlib inline
sns.set_style("whitegrid")
plt.style.use("fivethirtyeight")
```

In [3]:
```
data = pd.read_csv("heart.csv")
data.head()
```

Out[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach
0	63	1	3	145	233	1	0	150
1	37	1	2	130	250	0	1	187
2	41	0	1	130	204	0	0	172
3	56	1	1	120	236	0	1	178
4	57	0	0	120	354	0	1	163

In [4]:
```
data.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    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [5]:
```
data.shape
```

Out[5]: (303, 14)

In [6]:
```
pd.set_option("display.float", "{:.2f}".forma
t)
data.describe()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs
count	303.00	303.00	303.00	303.00	303.00	303.(
mean	54.37	0.68	0.97	131.62	246.26	0.15
std	9.08	0.47	1.03	17.54	51.83	0.36
min	29.00	0.00	0.00	94.00	126.00	0.00
25%	47.50	0.00	0.00	120.00	211.00	0.00
50%	55.00	1.00	1.00	130.00	240.00	0.00
75%	61.00	1.00	2.00	140.00	274.50	0.00
max	77.00	1.00	3.00	200.00	564.00	1.00

In [7]:
```
data.target.value_counts()
```

Out[7]:
```
1    165
0    138
Name: target, dtype: int64
```

In [8]:
```
data.target.value_counts().hvplot.bar(
    title="Heart Disease Count", xlabel='Hear
t Disease', ylabel='Count',
    width=500, height=350
)
```

Out[8]:

In [9]:
```
# Checking for missing values
data.isna().sum()
```

Out[9]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [10]:
```
categorical_val = []
```

```
continous_val = []
for column in data.columns:
    if len(data[column].unique()) <= 10:
        categorical_val.append(column)
    else:
        continous_val.append(column)
```

In [11]:
```
categorical_val
```

Out[11]:
```
['sex', 'cp', 'fbs', 'restecg', 'exang', 'slo
pe', 'ca', 'thal', 'target']
```

In [12]:
```
have_disease = data.loc[data['target']==1, 's
ex'].value_counts().hvplot.bar(alpha=0.4)
no_disease = data.loc[data['target']==0, 'se
x'].value_counts().hvplot.bar(alpha=0.4)

(no_disease * have_disease).opts(
    title="Heart Disease by Sex", xlabel='Se
x', ylabel='Count',
    width=500, height=450, legend_cols=2, leg
end_position='top_right'
)
```

Out[12]:

In [13]:
```
have_disease = data.loc[data['target']==1, 'c
p'].value_counts().hvplot.bar(alpha=0.4)
no_disease = data.loc[data['target']==0, 'cp'
].value_counts().hvplot.bar(alpha=0.4)

(no_disease * have_disease).opts(
    title="Heart Disease by Chest Pain Type",
xlabel='Chest Pain Type', ylabel='Count',
    width=500, height=450, legend_cols=2, leg
end_position='top_right'
)
```

Out[13]:

In [14]:
```
have_disease = data.loc[data['target']==1, 'f
bs'].value_counts().hvplot.bar(alpha=0.4)
no_disease = data.loc[data['target']==0, 'fb
s'].value_counts().hvplot.bar(alpha=0.4)

(no_disease * have_disease).opts(
    title="Heart Disease by fasting blood sug
ar", xlabel='fasting blood sugar > 120 mg/dl
 (1 = true; 0 = false)',
    ylabel='Count', width=500, height=450, le
gend_cols=2, legend_position='top_right'
)
```

Out[14]:

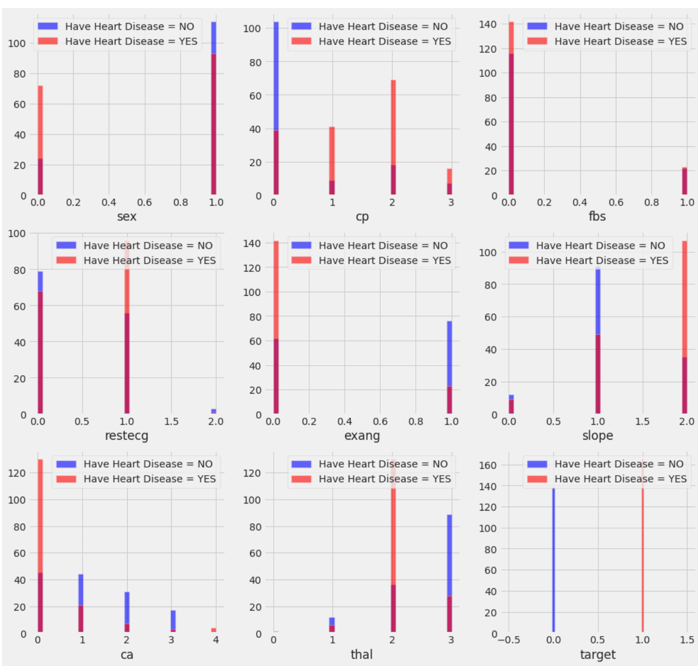
In [15]:
```
have_disease = data.loc[data['target']==1, 'r
estecg'].value_counts().hvplot.bar(alpha=0.4)
```

```
no_disease = data.loc[data['target']==0, 'res
tecg'].value_counts().hvplot.bar(alpha=0.4)

(no_disease * have_disease).opts(
    title="Heart Disease by resting electroca
rdiographic results", xlabel='resting electro
cardiographic results',
    ylabel='Count', width=500, height=450, le
gend_cols=2, legend_position='top_right'
)
```

Out[15]:

In [16]:

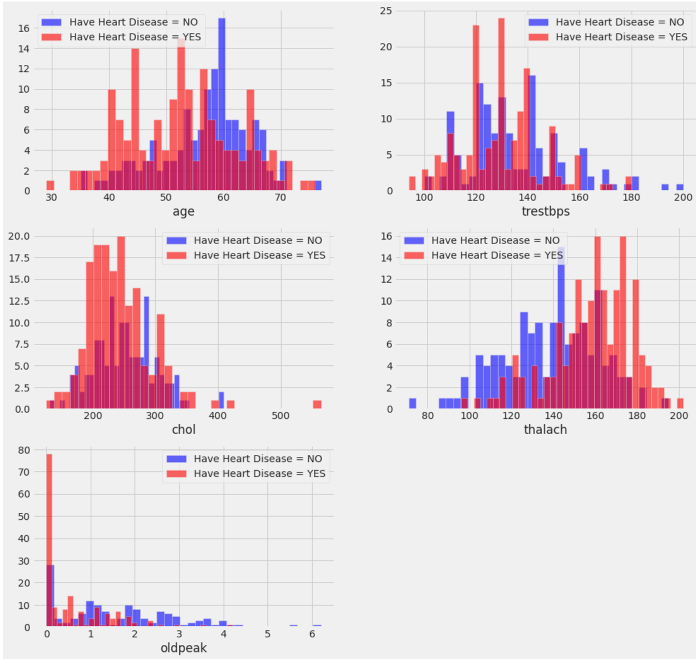
```
plt.figure(figsize=(15, 15))

for i, column in enumerate(categorical_val, 1
):
    plt.subplot(3, 3, i)
    data[data["target"] == 0][column].hist(bi
ns=35, color='blue', label='Have Heart Diseas
e = NO', alpha=0.6)
    data[data["target"] == 1][column].hist(bi
ns=35, color='red', label='Have Heart Disease
= YES', alpha=0.6)
    plt.legend()
    plt.xlabel(column)
```



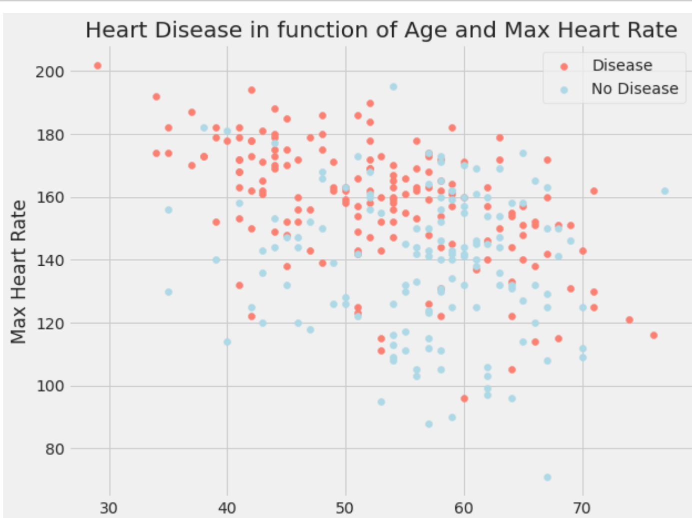
In [17]:

```
plt.figure(figsize=(15, 15))

for i, column in enumerate(continous_val, 1):
    plt.subplot(3, 2, i)
    data[data["target"] == 0][column].hist(bi
ns=35, color='blue', label='Have Heart Diseas
e = NO', alpha=0.6)
    data[data["target"] == 1][column].hist(bi
ns=35, color='red', label='Have Heart Disease
= YES', alpha=0.6)
    plt.legend()
```

```
    plt.legend()
    plt.xlabel(column)
```

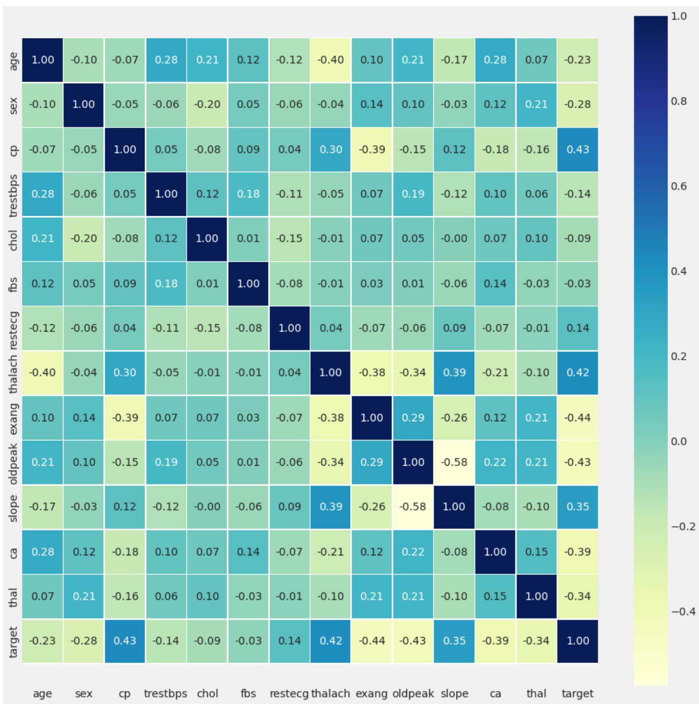


In [18]:
```
# Create another figure
plt.figure(figsize=(9, 7))

# Scatter with postivie examples
plt.scatter(data.age[data.target==1],
            data.thalach[data.target==1],
            c="salmon")

# Scatter with negative examples
plt.scatter(data.age[data.target==0],
            data.thalach[data.target==0],
            c="lightblue")

# Add some helpful info
plt.title("Heart Disease in function of Age a
nd Max Heart Rate")
plt.xlabel("Age")
plt.ylabel("Max Heart Rate")
plt.legend(["Disease", "No Disease"]);
```



In [19]:
```
# Let's make our correlation matrix a little
 prettier
corr_matrix = data.corr()
fig, ax = plt.subplots(figsize=(15, 15))
ax = sns.heatmap(corr_matrix,
                 annot=True,
                 linewidths=0.5,
                 fmt=".2f",
                 cmap="YlGnBu");
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[19]: (14.5, -0.5)



In [20]:
```
categorical_val.remove('target')
dataset = pd.get_dummies(data, columns = cate
gorical_val)
```

In [21]:
```
dataset.head()
```

Out[21]:

	age	trestbps	chol	thalach	oldpeak	target	sex_
0	63	145	233	150	2.30	1	0
1	37	130	250	187	3.50	1	0
2	41	130	204	172	1.40	1	1
3	56	120	236	178	0.80	1	0
4	57	120	354	163	0.60	1	1

5 rows × 31 columns

In [22]:
```
print(data.columns)
```

```
In [22]: print(data.columns)
print(dataset.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'cho
l', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'th
al', 'target'],
      dtype='object')
Index(['age', 'trestbps', 'chol', 'thalach',
'oldpeak', 'target', 'sex_0',
       'sex_1', 'cp_0', 'cp_1', 'cp_2', 'cp_
3', 'fbs_0', 'fbs_1', 'restecg_0',
       'restecg_1', 'restecg_2', 'exang_0',
'exang_1', 'slope_0', 'slope_1',
       'slope_2', 'ca_0', 'ca_1', 'ca_2', 'ca
_3', 'ca_4', 'thal_0', 'thal_1',
       'thal_2', 'thal_3'],
      dtype='object')
```

In [23]:

```
from sklearn.preprocessing import StandardSca
ler

s_sc = StandardScaler()
col_to_scale = ['age', 'trestbps', 'chol', 't
halach', 'oldpeak']
dataset[col_to_scale] = s_sc.fit_transform(da
taset[col_to_scale])
```

In [24]:

```
dataset.head()
```

Out[24]:

	age	trestbps	chol	thalach	oldpeak	target	se
0	0.95	0.76	-0.26	0.02	1.09	1	0
1	-1.92	-0.09	0.07	1.63	2.12	1	0
2	-1.47	-0.09	-0.82	0.98	0.31	1	1
3	0.18	-0.66	-0.20	1.24	-0.21	1	0
4	0.29	-0.66	2.08	0.58	-0.38	1	1

5 rows × 31 columns

In [25]:

```
from sklearn.metrics import accuracy_score, c
onfusion_matrix, classification_report

def print_score(clf, X_train, y_train, X_test
, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classificat
ion_report(y_train, pred, output_dict=True))
        print("Train Result:\n")
        print(f"Accuracy Score: {accuracy_sco
re(y_train, pred) * 100:.2f}%")
        print("")
        print(f"CLASSIFICATION REPORT:\n{clf
```

```
report}")
        print("")
        print(f"Confusion Matrix: \n {confusi
on_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classificat
ion_report(y_test, pred, output_dict=True))
        print("Test Result:\n")
        print(f"Accuracy Score: {accuracy_sco
re(y_test, pred) * 100:.2f}%")
        print("")
        print(f"CLASSIFICATION REPORT:\n{clf_
report}")
        print("")
        print(f"Confusion Matrix: \n {confusi
on_matrix(y_test, pred)}\n")
```

In [26]:
```
from sklearn.model_selection import train_tes
t_split

X = dataset.drop('target', axis=1)
y = dataset.target

X_train, X_test, y_train, y_test = train_test
_split(X, y, test_size=0.3, random_state=42)
```

1.Logistic Regression

In [27]:
```
from sklearn.linear_model import LogisticRegr
ession

lr_clf = LogisticRegression(solver='liblinea
r')
lr_clf.fit(X_train, y_train)

print_score(lr_clf, X_train, y_train, X_test,
y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test,
y_test, train=False)
```

```
Train Result:

Accuracy Score: 86.79%

CLASSIFICATION REPORT:
              0      1  accuracy  macro avg
weighted avg
precision  0.88   0.86      0.87       0.87
0.87
recall     0.82   0.90      0.87       0.86
0.87
f1-score   0.85   0.88      0.87       0.87
0.87
```

```
support   97.00 115.00      0.87     212.00
212.00

Confusion Matrix:
 [[ 80  17]
 [ 11 104]]

Test Result:

Accuracy Score: 86.81%

CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  w
eighted avg
precision  0.87  0.87      0.87       0.87
0.87
recall     0.83  0.90      0.87       0.86
0.87
f1-score   0.85  0.88      0.87       0.87
0.87
support   41.00 50.00      0.87      91.00
91.00

Confusion Matrix:
 [[34  7]
 [ 5 45]]
```

In [28]:

```
test_score = accuracy_score(y_test, lr_clf.pr
edict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.
predict(X_train)) * 100

results_df = pd.DataFrame(data=[["Logistic Re
gression", train_score, test_score]],
                          columns=['Model',
'Training Accuracy %', 'Testing Accuracy %'])
results_df
```

Out[28]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81

2.Random Forest

In [29]:

```
from sklearn.ensemble import RandomForestClas
sifier
from sklearn.model_selection import Randomize
dSearchCV

rf_clf = RandomForestClassifier(n_estimators=
1000, random_state=42)
```

```
rf_clf.fit(X_train, y_train)

print_score(rf_clf, X_train, y_train, X_test,
y_test, train=True)
print_score(rf_clf, X_train, y_train, X_test,
y_test, train=False)
```

```
Train Result:

Accuracy Score: 100.00%

CLASSIFICATION REPORT:
              0      1  accuracy  macro avg
weighted avg
precision  1.00   1.00      1.00       1.00
1.00
recall     1.00   1.00      1.00       1.00
1.00
f1-score   1.00   1.00      1.00       1.00
1.00
support   97.00 115.00      1.00     212.00
212.00

Confusion Matrix:
 [[ 97   0]
 [  0 115]]

Test Result:

Accuracy Score: 82.42%

CLASSIFICATION REPORT:
              0     1  accuracy  macro avg  w
eighted avg
precision  0.80  0.84      0.82       0.82
0.82
recall     0.80  0.84      0.82       0.82
0.82
f1-score   0.80  0.84      0.82       0.82
0.82
support   41.00 50.00      0.82      91.00
91.00

Confusion Matrix:
 [[33  8]
 [ 8 42]]
```

In [30]:

```
test_score = accuracy_score(y_test, rf_clf.pr
edict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.
predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Random Fo
rest Classifier", train_score, test_score]],
                          columns=['Model',
'Training Accuracy %', 'Testing Accuracy %'])
results_df = results_df.append(results_df_2,
```

```
ignore_index=True)
results_df
```

Out[30]:

	Model	Training Accuracy %	Testing Accuracy %
0	Logistic Regression	86.79	86.81
1	Random Forest Classifier	100.00	82.42

Accuracy of Logistic Regression

In [31]:

```
test_score = accuracy_score(y_test, lr_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, lr_clf.predict(X_train)) * 100

tuning_results_df = pd.DataFrame(data=[["Tuned Logistic Regression", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df
```

Out[31]:

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	86.79	86.81

Accuracy of Random Forest

In [32]:

```
test_score = accuracy_score(y_test, rf_clf.predict(X_test)) * 100
train_score = accuracy_score(y_train, rf_clf.predict(X_train)) * 100

results_df_2 = pd.DataFrame(data=[["Tuned Random Forest Classifier", train_score, test_score]],
                          columns=['Model', 'Training Accuracy %', 'Testing Accuracy %'])
tuning_results_df = tuning_results_df.append(results_df_2, ignore_index=True)
tuning_results_df
```

Out[32]:

		Training	Testing

	Model	Training Accuracy %	Testing Accuracy %
0	Tuned Logistic Regression	86.79	86.81
1	Tuned Random Forest Classifier	100.00	82.42

Random forest Feature

In [33]:
```
def feature_imp(df, model):
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importan
ces_
    return fi.sort_values(by="importance", as
cending=False)
```

In [34]:
```
feature_imp(X, rf_clf).plot(kind='barh', figs
ize=(12,7), legend=False)
```

Out[34]: <AxesSubplot:>