EDA, Feature Engineering, Hypothesis Testing, and Classification on Heart Failure Prediction Dataset

Introduction

What are cardiovascular diseases?

Cardiovascular diseases (CVDs) are a group of disorders of the heart and blood vessels. They include:

- coronary heart disease – a disease of the blood vessels supplying the heart muscle;
- cerebrovascular disease – a disease of the blood vessels supplying the brain;
- peripheral arterial disease – a disease of blood vessels supplying the arms and legs;
- rheumatic heart disease – damage to the heart muscle and heart valves from rheumatic fever, caused by streptococcal bacteria;
- congenital heart disease – birth defects that affect the normal development and functioning of the heart caused by malformations of the heart structure from birth; and deep vein thrombosis and pulmonary embolism – blood clots in the leg veins, which can dislodge and move to the heart and lungs.

Heart attacks and strokes are usually acute events and are mainly caused by a blockage that prevents blood from flowing to the heart or brain. The most common reason for this is a build-up of fatty deposits on the inner walls of the blood vessels that supply the heart or brain. Strokes can be caused by bleeding from a blood vessel in the brain or from blood clots. (source))

Dataset Overview

Feature Information

- Age: age of the patient [years]
- Sex: sex of the patient [M: Male, F: Female]
- ChestPainType: chest pain type [TA: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- RestingBP: resting blood pressure [mm Hg]
- Cholesterol: serum cholesterol [mm/dl]
- FastingBS: fasting blood sugar [1: if FastingBS > 120 mg/dl, 0: otherwise]
- 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]
- MaxHR: maximum heart rate achieved [Numeric value between 60 and 202]
- ExerciseAngina: exercise-induced angina [Y: Yes, N: No]
- Oldpeak: oldpeak = ST [Numeric value measured in depression]
- ST_Slope: the slope of the peak exercise ST segment [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: output class [1: heart disease, 0: Normal]

Source

Dataset was taken from Heart Failure Prediction Dataset

Section 1: Setup, Load, and Clean

In [ ]:
```
import os
data_path = ['data']
```

In [ ]:
```
## Import neccessary libraries to load data
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
## Load in the Dataset
filepath = os.sep.join(
    data_path + ['heart.csv'])
df = pd.read_csv(filepath)
df.head()
```

Out[ ]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAn
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	

In [ ]:
```
## Examine the information from the data
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 [ ]:
```
## Check for null value
df.isnull().sum()
```

Out[ ]:
```
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
dtype: int64
```

In [ ]:
```
## Create range section in describe table
heart_df = df.copy()
stat_df = heart_df.describe()
stat_df.loc['range'] = stat_df.loc['max'] - stat_df.loc['min']
stat_df.T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max	range
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0	49.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0	142.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2	8.8
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0	1.0

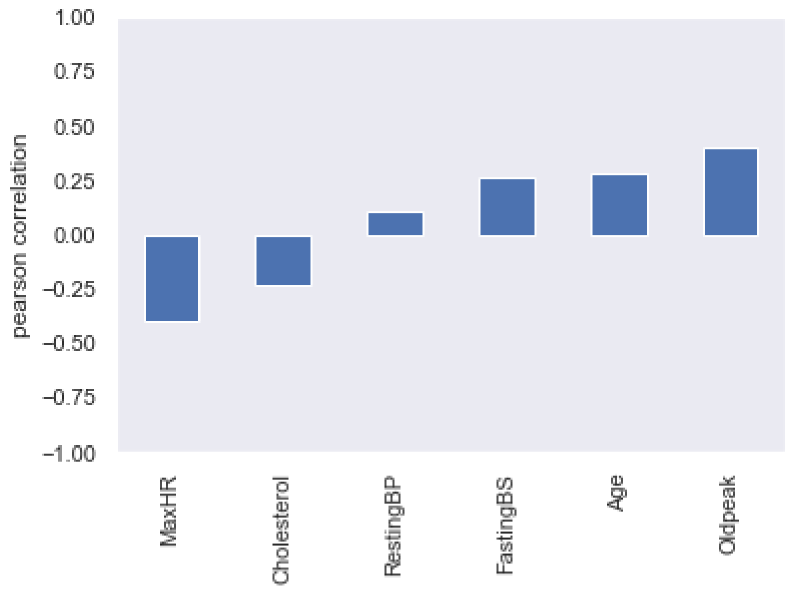
In [ ]:
```
## Import neccessary libraries to visualize data
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_theme(style="dark")
```

In [ ]:
```
## Check correlation between each features to target
y = (heart_df['HeartDisease'] == 1).astype(int)
fields = list(heart_df.columns[:-1])  # everything except "HeartDisease"
correlations = heart_df[fields].corrwith(y)
correlations.sort_values(inplace=True)
correlations
```

Out[ ]:
```
MaxHR         -0.400421
Cholesterol   -0.232741
RestingBP      0.107589
FastingBS      0.267291
Age            0.282039
Oldpeak        0.403951
dtype: float64
```

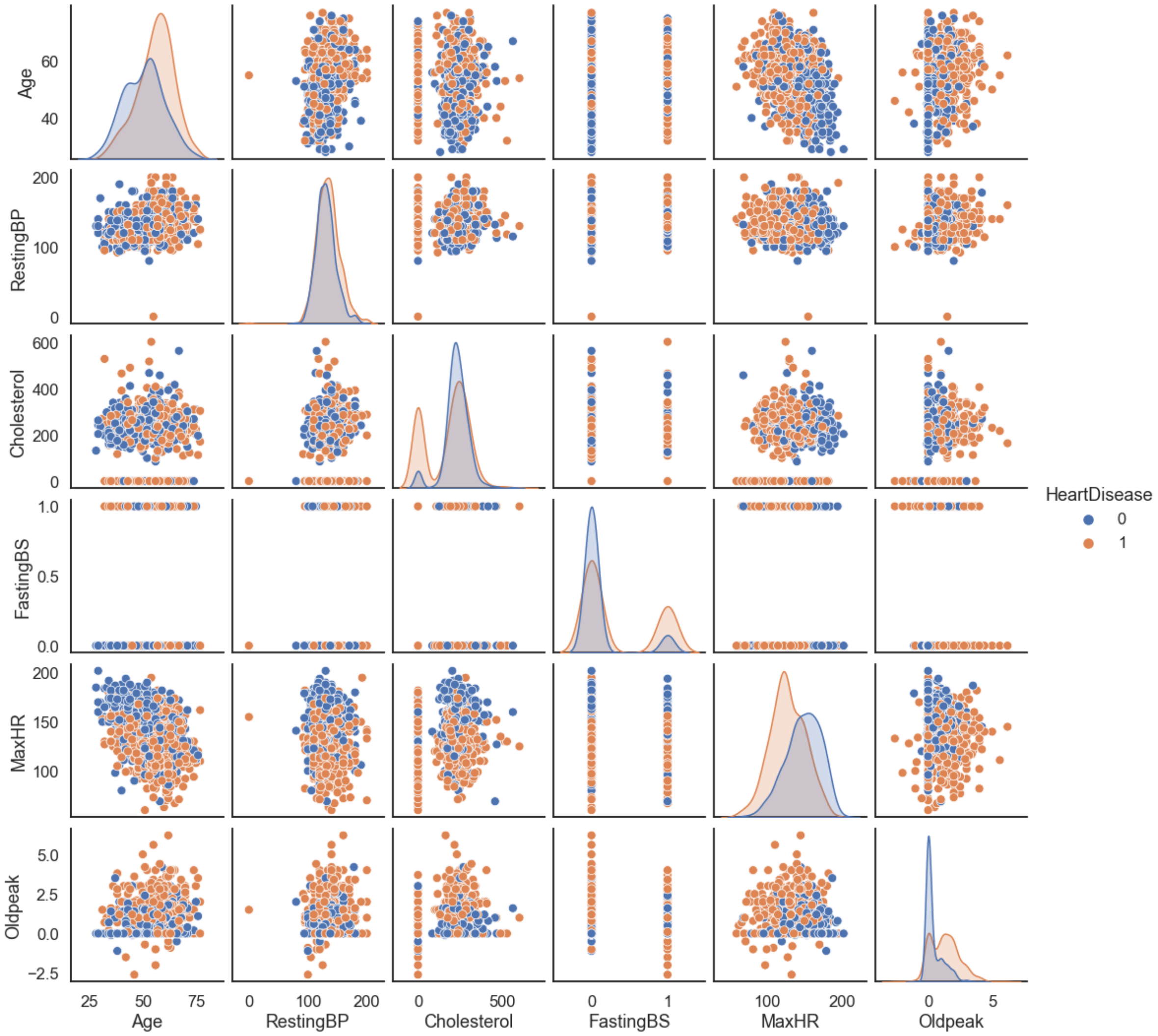
In [ ]:
```
## Graph the correlation chart
ax = correlations.plot(kind='bar')
ax.set(ylim=[-1, 1], ylabel='pearson correlation')
```

Out[ ]: [(-1.0, 1.0), Text(0, 0.5, 'pearson correlation')]



In [ ]:

```
sns.set_context('talk')
sns.set_style('white')
sns.pairplot(heart_df, hue='HeartDisease')
plt.show()
```



First Section Short Recap/Conclusion:

- 'Oldpeak' and 'MaxHR' was highly correlated with 'HeartDisease'.

Section 2: Simple Exploratory Data Analysis (EDA)

In [ ]:
```
## check for unique variables on each features
heart_df.nunique()
```

Out[ ]:
```
Age                50
Sex                 2
ChestPainType       4
RestingBP          67
Cholesterol       222
FastingBS           2
RestingECG          3
MaxHR             119
ExerciseAngina      2
Oldpeak            53
ST_Slope            3
HeartDisease        2
dtype: int64
```

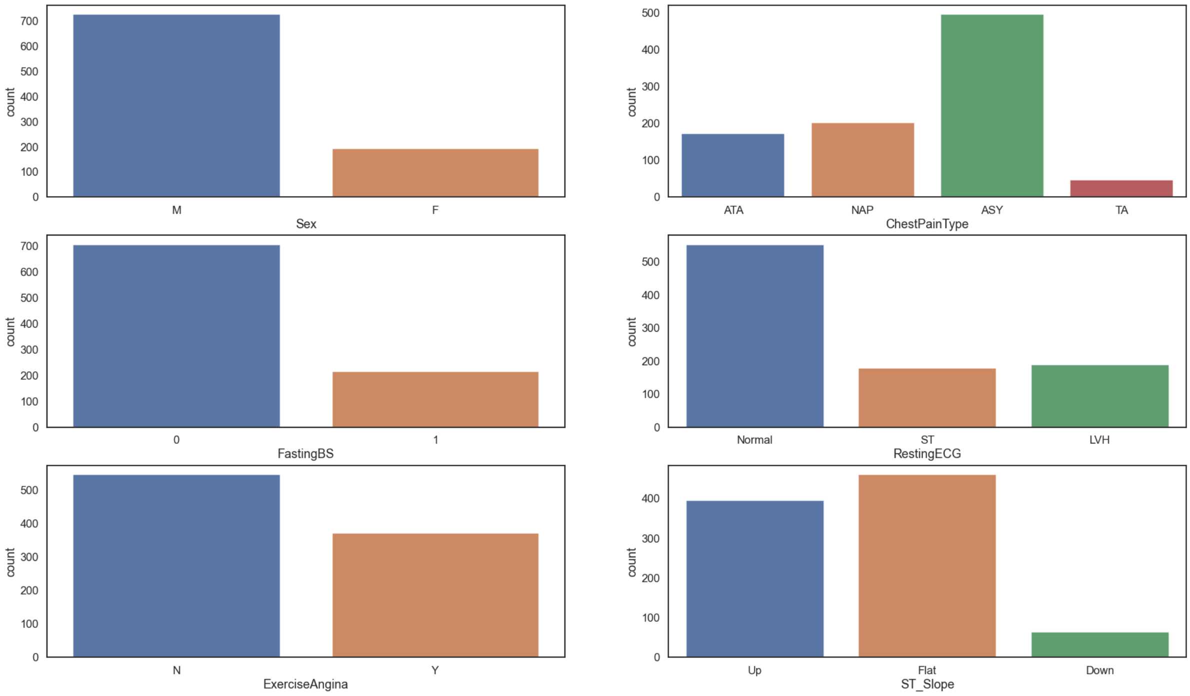
In [ ]:
```
## Take every fatures that have less than 5 uniques variable as categorical
## Take every fatures that have more than 4 uniques variable as numerical
## Excluding the HeartDisease columns
categorical_data_columns = heart_df.drop(
    columns='HeartDisease').dtypes[heart_df.nunique() < 5].index.tolist()
numerical_data_columns = heart_df.drop(
    columns='HeartDisease').dtypes[heart_df.nunique() > 4].index.tolist()
```

In [ ]:
```
## Checkout for each variables
print("Categorical Features: ", categorical_data_columns)
print("Numerical Features: ", numerical_data_columns)
```

```
Categorical Features:  ['Sex', 'ChestPainType', 'FastingBS', 'RestingECG', 'Exerci
seAngina', 'ST_Slope']
Numerical Features:  ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
```

Categorical Data Columns EDA

In [ ]:
```
## Visualize distributon for every categorical columns
plt.figure(figsize=(30, 30))
i = 1
for column in categorical_data_columns:
    plt.subplot(5, 2, i)
    sns.countplot(x=heart_df[column])
    i += 1
plt.show()
```



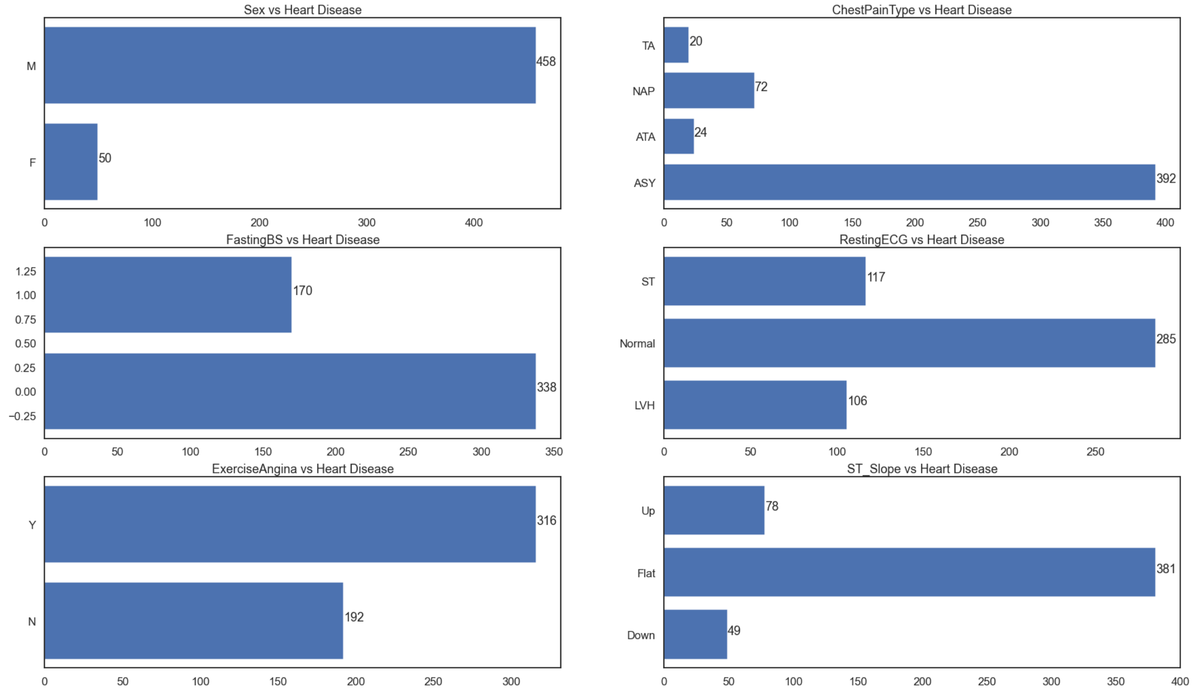
In [ ]:

```
## Calculate proportion of dominant classes in each category

num_rows = heart_df[categorical_data_columns].shape[0]
for i, cat in enumerate(heart_df[categorical_data_columns].columns):
    val_counts = heart_df[cat].value_counts()
    dominant_frac = val_counts.iloc[0] / num_rows
    print(
        f'`{val_counts.index[0]}` alone contributes to {round(dominant_frac * 100,
```

```
`M` alone contributes to 78.98% of Sex
`ASY` alone contributes to 54.03% of ChestPainType
`0` alone contributes to 76.69% of FastingBS
`Normal` alone contributes to 60.13% of RestingECG
`N` alone contributes to 59.59% of ExerciseAngina
`Flat` alone contributes to 50.11% of ST_Slope
```

In [ ]:

```
## Visualize HeartDisease count for every categorical columns

plt.figure(figsize=(30, 30))
i = 1
for column in categorical_data_columns:
    type_count = heart_df.groupby(column)["HeartDisease"].sum()
    plt.subplot(5, 2, i)
    x = type_count.index
    y = type_count.values
    plt.barh(x, y)
    plt.title(f"{column} vs Heart Disease")
    for index, value in enumerate(y):
        plt.text(value, index,
                 value)
    i += 1
plt.show()
```



In [ ]:

```
## Calculate the percentage of HeartDisease
tuples = []
values = []
for x in categorical_data_columns:
    value_distribution = heart_df[x].value_counts().sort_index()
    stroke_count = heart_df.groupby(x)["HeartDisease"].sum().sort_index()
    for y in range(len(value_distribution)):
        ratio = round(
            (stroke_count.iloc[y] / value_distribution.iloc[y]) * 100, 2)
        #print("{}: {} percentage on stroke is {}%".format(x, stroke_count.index[y
        ratio = str(ratio) + "%"
        tuples.append((x, stroke_count.index[y]))
        values.append(ratio)
```

In [ ]:

```
## Displaying the percentage in a table form
multicol1 = pd.MultiIndex.from_tuples(tuples)
df_percentage = pd.DataFrame([values],
                             index=['HeartDisease Percentage'],
                             columns=multicol1)

df_percentage.T
```

Out[ ]:

		HeartDisease Percentage
Sex	**F**	25.91%
	M	63.17%
ChestPainType	**ASY**	79.03%
	ATA	13.87%
	NAP	35.47%
	TA	43.48%
FastingBS	**0**	48.01%
	1	79.44%
RestingECG	**LVH**	56.38%
	Normal	51.63%
	ST	65.73%
ExerciseAngina	**N**	35.1%
	Y	85.18%
ST_Slope	**Down**	77.78%
	Flat	82.83%
	Up	19.75%

Categorical Data Columns EDA Short Recap/Conclusion:

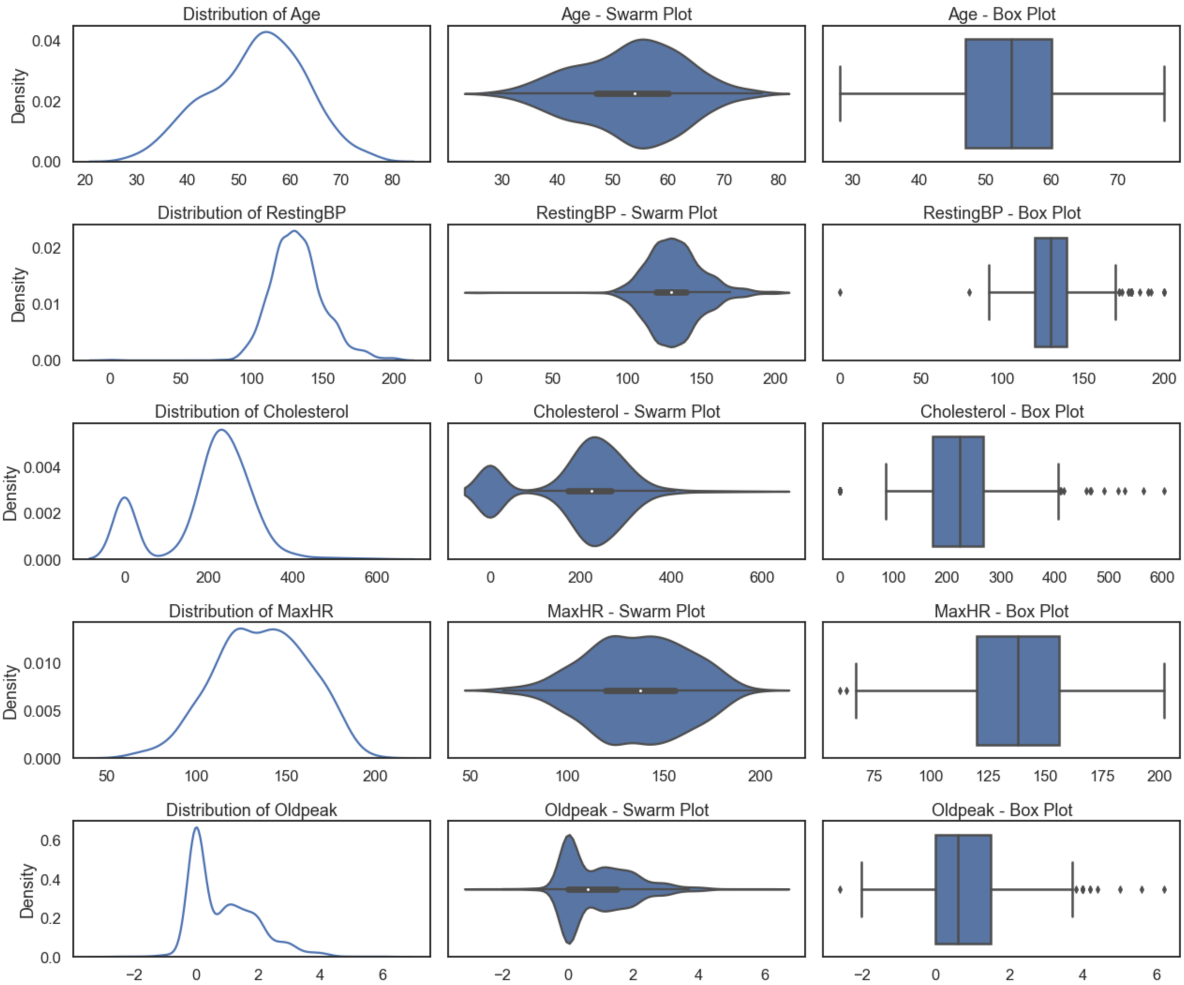
- 'Male', 'Asymptomatic', 'Fasting Blood Sugar > 120 mg/dl', 'Having ST-T Wave Abnormality', 'Exercise-Induced Angina', and 'Flat Slope of the Peak Exercise ST Segment', have a high influence on causes of heart disease.

Numerical Data Columns EDA

In [ ]:

```
## Visualize distribution on numerical features
rows = len(numerical_data_columns)
cols = 3

fig = plt.figure(1, (18, rows*3))

i = 0
for feature in numerical_data_columns:

    i += 1
    ax1 = plt.subplot(rows, cols, i)
    sns.kdeplot(data=heart_df, x=feature)
    ax1.set_xlabel(None)
    ax1.set_title(f'Distribution of {feature}')
    plt.tight_layout()

    i += 1
    ax2 = plt.subplot(rows, cols, i)
    sns.violinplot(data=heart_df, x=feature)
    ax2.set_xlabel(None)
    ax2.set_title(f'{feature} - Swarm Plot')
    plt.tight_layout()
```

```
    i += 1
    ax3 = plt.subplot(rows, cols, i)
    sns.boxplot(data=heart_df, x=feature, orient='h', linewidth=2.5)
    ax3.set_xlabel(None)
    ax3.set_title(f'{feature} - Box Plot')
    plt.tight_layout()

plt.show()
```



In [ ]:

```
## Find outliers using Tukey's method
def tukey_outliers(x):
    ## Tukey outliers are based on the boundaries defined by quantiles and IQR
    q1 = np.percentile(x, 25)
    q3 = np.percentile(x, 75)

    iqr = q3 - q1

    lower_boundary = q1 - (iqr * 1.5)
    upper_boundary = q3 + (iqr * 1.5)

    outliers = x[(x < lower_boundary) | (x > upper_boundary)]
    return outliers
```

In [ ]:

```
## Calculate the tukey outliers

outlier_dict = {}
for num_feature in numerical_data_columns:
    outliers = tukey_outliers(heart_df[num_feature])
    if len(outliers):
        print(f"-> {num_feature} has {len(outliers)} tukey outliers")
```

```
        outlier_dict[num_feature] = outliers
    else:
        print(f"-> {num_feature} doesn't have any tukey outliers.")
        outlier_dict[num_feature] = None
```

```
-> Age doesn't have any tukey outliers.
-> RestingBP has 28 tukey outliers
-> Cholesterol has 183 tukey outliers
-> MaxHR has 2 tukey outliers
-> Oldpeak has 16 tukey outliers
```

In [ ]:

```
## Show the percentage of outliers

for x in numerical_data_columns:
    if x == 'Age':
        continue
    outliers = heart_df.loc[outlier_dict[x].index]
    print("{} has {}% of outliers".format(
        x, round(len(outliers)/len(heart_df) * 100, 2)))
```

```
RestingBP has 3.05% of outliers
Cholesterol has 19.93% of outliers
MaxHR has 0.22% of outliers
Oldpeak has 1.74% of outliers
```

In [ ]:

```
skew_limit = 0  # define a limit above which we will log transform
skew_vals = heart_df[numerical_data_columns].skew()
```

In [ ]:

```
skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0: 'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))

skew_cols
```

Out[ ]:

	Skew
Oldpeak	1.022872
RestingBP	0.179839
MaxHR	-0.144359
Age	-0.195933
Cholesterol	-0.610086

Numerical Data Columns EDA Short Recap/Conclusion:

- 'Cholesterol' has the highest percentage of outliers with '19.93%'.
- 'Oldpeak' is the most skewed data with '1.022872' of skewness.

Section 3: Feature Engineering

In [ ]:

```
heart_df_oe = heart_df.copy()
```

In [ ]:

```
## Create ordinal endocing mapper
```

```python
sex_mapper = {
    'F':0,
    'M':1
    }
heart_df_oe['Sex'] = df['Sex'].replace(sex_mapper)

cpt_mapper = {
    'ASY': 3,
    'ATA': 0,
    'NAP': 1,
    'TA': 2
}
heart_df_oe['ChestPainType'] = df['ChestPainType'].replace(cpt_mapper)

re_mapper = {
    'LVH':1,
    'Normal':0,
    'ST':2
    }
heart_df_oe['RestingECG'] = df['RestingECG'].replace(re_mapper)

ea_mapper = {
    'N':0,
    'Y':1
    }
heart_df_oe['ExerciseAngina'] = df['ExerciseAngina'].replace(ea_mapper)

sts_mapper = {
    'Down':1,
    'Flat':2,
    'Up':0
    }
heart_df_oe['ST_Slope'] = df['ST_Slope'].replace(sts_mapper)
```

In [ ]:

```python
heart_df_oe.head()
```

Out[ ]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAn
0	40	1	0	140	289	0	0	172	
1	49	0	1	160	180	0	0	156	
2	37	1	0	130	283	0	2	98	
3	48	0	3	138	214	0	0	108	
4	54	1	1	150	195	0	0	122	

In [ ]:

```python
## Check correlation between each features to target
y = (heart_df_oe['HeartDisease'] == 1).astype(int)
fields = list(heart_df_oe.columns[:-1])  # everything except "HeartDisease"
correlations = heart_df_oe[fields].corrwith(y)
correlations.sort_values(inplace=True)
correlations
```

Out[ ]:

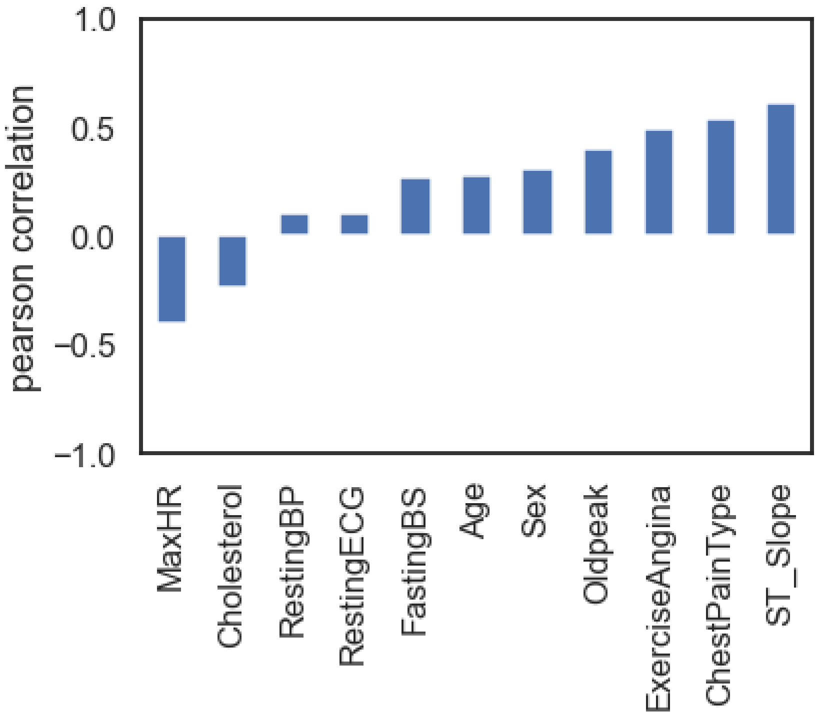
```
MaxHR            -0.400421
Cholesterol      -0.232741
RestingBP         0.107589
RestingECG        0.107628
FastingBS         0.267291
Age               0.282039
Sex               0.305445
Oldpeak           0.403951
ExerciseAngina    0.494282
ChestPainType     0.536974
ST_Slope          0.607921
dtype: float64
```

In [ ]:

```python
## Graph the correlation chart
ax = correlations.plot(kind='bar')
ax.set(ylim=[-1, 1], ylabel='pearson correlation')
```

Out[ ]:

```
[(-1.0, 1.0), Text(0, 0.5, 'pearson correlation')]
```



Feature Encoding Short Recap/Conclusion:

- After Ordinal Encoding, 'ST_Slope' and 'ChestPainType' was highly correlated with 'HeartDisease'.
- Before Ordinal Encoding, turns out 'Oldpeak' wasn't the highest corellated feature, so we gonna try to do some transformation into this features (due to a high skew value)
- We won't drop those high-percentage outliers on 'Cholesterol' because we're planning to use Normalization - Standarization Scalling on later section.

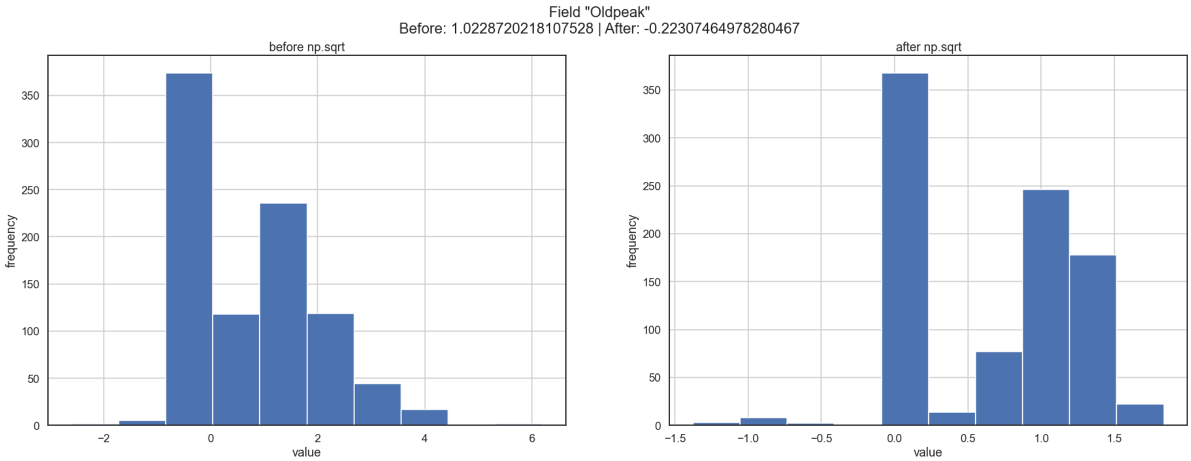
In [ ]:

```python
heart_df_oe_ft = heart_df_oe.copy()
```

In [ ]:

```python
## Create before-after tansformation graph
skew_features = ['Oldpeak']
for field in skew_features:
    # Create two "subplots" and a "figure" using matplotlib
    fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(30, 10))
```

```
    # Create a histogram on the "ax_before" subplot
    heart_df_oe_ft[field].hist(ax=ax_before)

    after_skew = np.cbrt(heart_df_oe_ft[field])

    # Apply a log transformation (numpy syntax) to this column
    after_skew.hist(ax=ax_after)

    # Formatting of titles etc. for each subplot
    ax_before.set(title='before np.sqrt', ylabel='frequency', xlabel='value')
    ax_after.set(title='after np.sqrt', ylabel='frequency', xlabel='value')
    fig.suptitle('Field "{}"\nBefore: {} | After: {}\n'.format(
        field, heart_df_oe_ft[field].skew(), after_skew.skew()))
```



In [ ]:

```
## Apply transformation to the feature
heart_df_oe_ft['Oldpeak'] = np.cbrt(heart_df_oe_ft['Oldpeak'])
```

In [ ]:

```
heart_df_oe_ft.head()
```

Out[ ]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAn
0	40	1	0	140	289	0	0	172	
1	49	0	1	160	180	0	0	156	
2	37	1	0	130	283	0	2	98	
3	48	0	3	138	214	0	0	108	
4	54	1	1	150	195	0	0	122	

Feature Transformation Short Recap/Conclusion:

- Because there's negative value on skewed features ('Oldpeak'), we gonna use Cube Root as Feature Transformation approach.
- After Feature Encoding with Cube Root method, 'Oldpeak' skewness decrease from '1.02' to '0.22'.

Section 4: Classification

In [ ]:

```
heart_df['HeartDisease'].value_counts()
```

Out[ ]:
```
1    508
0    410
Name: HeartDisease, dtype: int64
```

The 'HeartDisease' count kind of slightly imbalanced, so we gonna focus more on Precision - Recall with 'F1 Score'

In [ ]:
```
## Split the Training and Test set with 'StratifiedShuffleSplit'
## It would be better to split-out this way for imbalanced dataset

from sklearn.model_selection import StratifiedShuffleSplit

feature_cols = [x for x in heart_df_oe_ft.columns if x != 'HeartDisease']

## Get the split indexes
strat_shuf_split = StratifiedShuffleSplit(n_splits=4,
                                          test_size=0.333,
                                          random_state=42)

train_idx, test_idx = next(strat_shuf_split.split(
    heart_df_oe_ft[feature_cols], heart_df_oe_ft['HeartDisease']))

## Create the dataframes
X_train = heart_df_oe_ft.loc[train_idx, feature_cols]
y_train = heart_df_oe_ft.loc[train_idx, 'HeartDisease']

X_test = heart_df_oe_ft.loc[test_idx, feature_cols]
y_test = heart_df_oe_ft.loc[test_idx, 'HeartDisease']
```

In [ ]:
```
## Import neccessary libraries for modelling

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, \

from sklearn.preprocessing import MinMaxScaler, StandardScaler, FunctionTransforme
from sklearn.pipeline import make_pipeline, Pipeline
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.metrics import roc_curve, precision_recall_curve, confusion_matrix, r

import datetime
```

In [ ]:
```
## Create list of scaler

dummyScaller = FunctionTransformer()

scalers = [
    (dummyScaller, "NoScaling"),
    (MinMaxScaler(), "MinMaxScaler"),
    (StandardScaler(), "StandardScaler")
]
```

In [ ]:
```
## Create list of models

models = [
    RandomForestClassifier(random_state=42),
    GradientBoostingClassifier(random_state=42),
    SVC(random_state=42),
```

```
    KNeighborsClassifier(),
    LogisticRegression(random_state=42, solver='liblinear')
]
```

In [ ]:

```
## Create list (dictionaries) for 'GridsearchCV'

search_space_dict = {}

search_space_dict['RandomForestClassifier'] = {
        'randomforestclassifier__n_estimators': range(10, 300, 50),
        'randomforestclassifier__max_depth': range(0, 100, 10),
        'randomforestclassifier__max_features': ["auto", "sqrt", "log2"],
        'randomforestclassifier__criterion': ['gini', 'entropy']
    }

search_space_dict['GradientBoostingClassifier'] = {
        'gradientboostingclassifier__n_estimators': range(10, 300, 50),
        'gradientboostingclassifier__learning_rate': [0.1, 0.01, 0.001, 0.0001],
        'gradientboostingclassifier__max_features': [1, 2, 3, 4],
        'gradientboostingclassifier__subsample': [1.0, 0.5]
    }

search_space_dict['SVC'] = {
        'svc__gamma': [0.1, 1, 10],
        'svc__C': [0.1, 1, 10],
        'svc__kernel': ['linear', 'rbf']
    }

search_space_dict['KNeighborsClassifier'] = {
        'kneighborsclassifier__weights': ['uniform', 'distance'],
        'kneighborsclassifier__n_neighbors': range(1, 50, 2)
    }

search_space_dict['LogisticRegression'] = {
        'logisticregression__penalty': ['l1', 'l2', 'elasticnet'],
        'logisticregression__C': [0.01, 0.1, 1, 10, 100]
    }
```

In [ ]:

```
## Create pipeline matrix

pipelines_matrix = {}

for scaler, scaler_desc in scalers:
    pipelines_matrix[scaler_desc] = {}
    print(scaler_desc)
    for model in models:
        print("            ", model.__class__.__name__)
        pipelines_matrix[scaler_desc][model.__class__.__name__] = make_pipeline(
            scaler, model)
```

```
NoScaling
             RandomForestClassifier
             GradientBoostingClassifier
             SVC
             KNeighborsClassifier
             LogisticRegression
MinMaxScaler
             RandomForestClassifier
             GradientBoostingClassifier
             SVC
             KNeighborsClassifier
             LogisticRegression
StandardScaler
             RandomForestClassifier
             GradientBoostingClassifier
             SVC
             KNeighborsClassifier
             LogisticRegression
```

In [ ]:

```
## Create a function for performing cross validation of all algorithms
## Fuction will return a dataframe with the result from each pipeline

def cross_validator(X_train, y_train, pipelines_matrix):
    i = 0
    for scaler in pipelines_matrix:
        print("----------------------", scaler)
        for model in pipelines_matrix[scaler]:
            i += 1
            print("     +++++++", model)
            startT = datetime.datetime.now()

            pipeline = pipelines_matrix[scaler][model]

            search_space = search_space_dict[model]
            clf = GridSearchCV(pipeline,
                               search_space,
                               scoring='f1',
                               cv = 4,
                               n_jobs=-1)
            clf.fit(X_train, y_train)

            print("          f1-scores: ", clf.best_score_)

            headers = ['scaler', 'model',
                       'f1', 'best_params']
            dfResultsTemp = pd.DataFrame(columns=headers)
            dfResultsTemp.loc[0] = [
                scaler, model, clf.best_score_, clf.best_params_]

            print("             exec time:", datetime.datetime.now() -
                  startT, datetime.datetime.now())

            if i == 1:
                data_concat = dfResultsTemp.copy()
            else:
                data_concat = pd.concat([data_concat, dfResultsTemp])

    return data_concat
```

In [ ]:

```
## Run the function

grid_search_df = cross_validator(X_train, y_train, pipelines_matrix)
```

```
---------------------- NoScaling
     +++++++ RandomForestClassifier
          f1-scores:  0.8788514944844867
             exec time: 0:00:22.346816 2022-01-26 00:54:05.537119
     +++++++ GradientBoostingClassifier
          f1-scores:  0.8843207891723295
             exec time: 0:00:05.244787 2022-01-26 00:54:10.782035
     +++++++ SVC
          f1-scores:  0.8735797133365211
             exec time: 0:00:25.667875 2022-01-26 00:54:36.450191
     +++++++ KNeighborsClassifier
          f1-scores:  0.7339716036042038
             exec time: 0:00:00.202430 2022-01-26 00:54:36.652869
     +++++++ LogisticRegression
          f1-scores:  0.8710166857990979
             exec time: 0:00:00.063671 2022-01-26 00:54:36.716746
---------------------- MinMaxScaler
     +++++++ RandomForestClassifier
          f1-scores:  0.8788514944844867
             exec time: 0:00:22.647035 2022-01-26 00:54:59.364002
     +++++++ GradientBoostingClassifier
          f1-scores:  0.8843207891723295
             exec time: 0:00:05.436807 2022-01-26 00:55:04.801025
     +++++++ SVC
          f1-scores:  0.882691527612817
             exec time: 0:00:00.115919 2022-01-26 00:55:04.917157
     +++++++ KNeighborsClassifier
          f1-scores:  0.8787522968881352
             exec time: 0:00:00.211279 2022-01-26 00:55:05.128697
     +++++++ LogisticRegression
          f1-scores:  0.8710623742321698
             exec time: 0:00:00.059883 2022-01-26 00:55:05.188811
---------------------- StandardScaler
     +++++++ RandomForestClassifier
          f1-scores:  0.8788514944844867
             exec time: 0:00:22.861337 2022-01-26 00:55:28.050370
     +++++++ GradientBoostingClassifier
          f1-scores:  0.8830829297169877
             exec time: 0:00:05.419634 2022-01-26 00:55:33.470217
     +++++++ SVC
          f1-scores:  0.8805231575388777
             exec time: 0:00:00.154154 2022-01-26 00:55:33.624588
     +++++++ KNeighborsClassifier
          f1-scores:  0.8861955559141458
             exec time: 0:00:00.199574 2022-01-26 00:55:33.824371
     +++++++ LogisticRegression
          f1-scores:  0.8713864394170808
             exec time: 0:00:00.052167 2022-01-26 00:55:33.876755
```

In [ ]:

```
## Display the result

grid_search_df.sort_values(by=['f1'], ascending=False, ignore_index=True)
```

Out[ ]:

	scaler	model	f1	best_params
0	StandardScaler	KNeighborsClassifier	0.886196	{'kneighborsclassifier__n_neighbors': 15, 'kne...
1	NoScaling	GradientBoostingClassifier	0.884321	{'gradientboostingclassifier__learning_rate': ...
2	MinMaxScaler	GradientBoostingClassifier	0.884321	{'gradientboostingclassifier__learning_rate': ...
3	StandardScaler	GradientBoostingClassifier	0.883083	{'gradientboostingclassifier__learning_rate': ...
4	MinMaxScaler	SVC	0.882692	{'svc__C': 1, 'svc__gamma': 1, 'svc__kernel': ...
5	StandardScaler	SVC	0.880523	{'svc__C': 1, 'svc__gamma': 0.1, 'svc__kernel'...
6	NoScaling	RandomForestClassifier	0.878851	{'randomforestclassifier__criterion': 'entropy...
7	MinMaxScaler	RandomForestClassifier	0.878851	{'randomforestclassifier__criterion': 'entropy...
8	StandardScaler	RandomForestClassifier	0.878851	{'randomforestclassifier__criterion': 'entropy...
9	MinMaxScaler	KNeighborsClassifier	0.878752	{'kneighborsclassifier__n_neighbors': 15, 'kne...
10	NoScaling	SVC	0.873580	{'svc__C': 1, 'svc__gamma': 0.1, 'svc__kernel'...
11	StandardScaler	LogisticRegression	0.871386	{'logisticregression__C': 0.1, 'logisticregres...
12	MinMaxScaler	LogisticRegression	0.871062	{'logisticregression__C': 0.1, 'logisticregres...
13	NoScaling	LogisticRegression	0.871017	{'logisticregression__C': 1, 'logisticregressi...
14	NoScaling	KNeighborsClassifier	0.733972	{'kneighborsclassifier__n_neighbors': 39, 'kne...

In [ ]:

```
## Create a csv file for future evaluation

grid_search_df.sort_values(
    by=['f1'], ascending=False, ignore_index=True).to_csv('model_evaluation_heart_
```

Grid Search CV Evaluation Short Recap/Conclusion:

- 'KNeighborsClassifier' with 'StandardScaler' have the highest F1 score with '0.886196'.
- 'GradientBoostingClassifier' following on number 2 with identical F1 Weighted score (between 'NoScalling' and 'MinMaxScaler') with '0.884321'.
- We gonna take 'GradientBoostingClassifier' and 'KNeighborsClassifier' for futher analysis.

In [ ]:

```
## Recreate model list with top models only

top_models = [
    ['K Neighbors Classifier', KNeighborsClassifier(n_neighbors=15, weights='unifo
    ['Gradient Boosting Classifier', GradientBoostingClassifier(learning_rate=0.01
                                                                random_state=42, s
]
```

In [ ]:

```
##  Create a report dataframe for Classification Reports and Confussion Matrix
```

```python
def report_dataframe(m):
    pipeline = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('model', m[1])])

    pipeline.fit(X_train, y_train)

    y_pred = pipeline.predict(X_test)

    print(m[0] + " with StandardScaler Reports: ")

    cr = classification_report(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print(cr)
```

In [ ]:

```python
for m in range(len(top_models)):
    report_dataframe(top_models[m])
```

```
K Neighbors Classifier with StandardScaler Reports:
[[117  20]
 [ 15 154]]
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       137
           1       0.89      0.91      0.90       169

    accuracy                           0.89       306
   macro avg       0.89      0.88      0.88       306
weighted avg       0.89      0.89      0.89       306

Gradient Boosting Classifier with StandardScaler Reports:
[[117  20]
 [ 11 158]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       137
           1       0.89      0.93      0.91       169

    accuracy                           0.90       306
   macro avg       0.90      0.89      0.90       306
weighted avg       0.90      0.90      0.90       306
```

It might be nice to create a 'VotingClassifier' between those top models, most likely will improve the final model as well.

In [ ]:

```python
## Create a Voting Classifier pipeline
## Next steps is to fit, get Classification Report, and Confussion Matrix

voting_classifier_pipeline = Pipeline(steps=[
    ('data_scaling', StandardScaler()),
    ('model', VotingClassifier(top_models, voting='soft'))])
```

In [ ]:

```python
voting_classifier_pipeline.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('data_scaling', StandardScaler()),
                ('model',
                 VotingClassifier(estimators=[['K Neighbors Classifier',
                                               KNeighborsClassifier(n_neighbors=1
5)],
                                              ['Gradient Boosting Classifier',
                                               GradientBoostingClassifier(learning
_rate=0.01,
                                                                          max_feat
ures=3,
                                                                          n_estima
tors=110,
                                                                          random_s
tate=42,
                                                                          subsampl
e=0.5)]],
                                  voting='soft'))])
```

In [ ]:

```python
y_pred = voting_classifier_pipeline.predict(X_test)

print("Voting Classifier" + " with " + "StandardScaler" + " Reports: ")

cr = classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
print(cm)
print(cr)
```

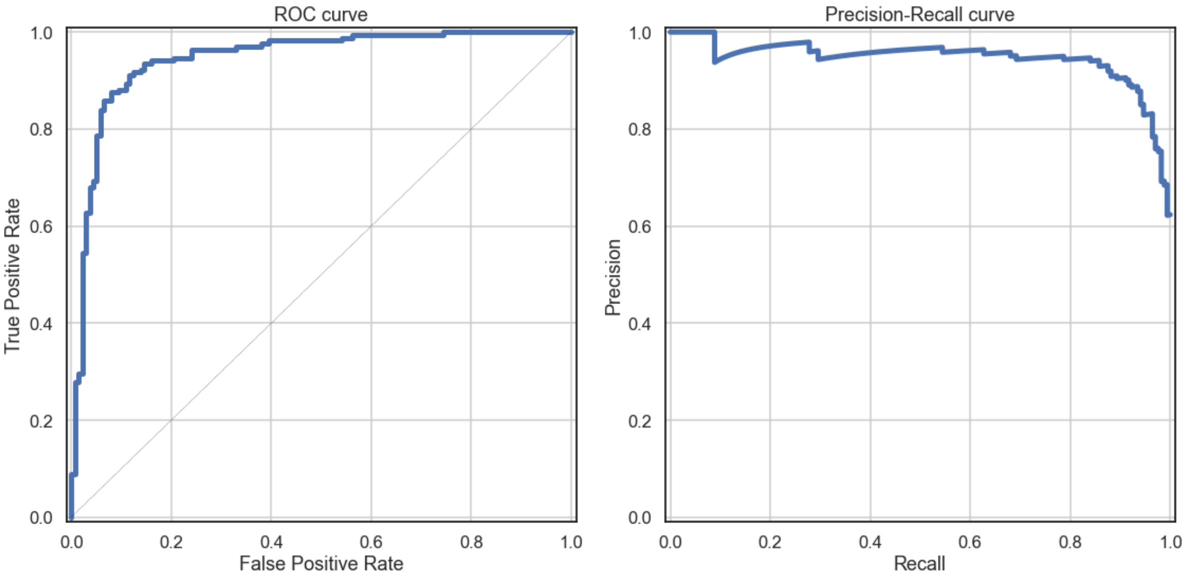
```
Voting Classifier with StandardScaler Reports:
[[117  20]
 [ 12 157]]
              precision    recall  f1-score   support

           0       0.91      0.85      0.88       137
           1       0.89      0.93      0.91       169

    accuracy                           0.90       306
   macro avg       0.90      0.89      0.89       306
weighted avg       0.90      0.90      0.90       306
```

Next steps is to create a graph for ROC and Precision - Recall curve and Feature Importances

In [ ]:

```python
## Create a graph for ROC and Precision - Recall curve

sns.set_context('talk')

fig, axList = plt.subplots(ncols=2)
fig.set_size_inches(16, 8)

# Get the probabilities for each of the two categories
y_prob = voting_classifier_pipeline.predict_proba(X_test)

# Plot the ROC-AUC curve
ax = axList[0]

fpr, tpr, thresholds = roc_curve(y_test, y_prob[:, 1])
ax.plot(fpr, tpr, linewidth=5)
# It is customary to draw a diagonal dotted line in ROC plots.
# This is to indicate completely random prediction. Deviation from this
# dotted line towards the upper left corner signifies the power of the model.
ax.plot([0, 1], [0, 1], ls='--', color='black', lw=.3)
ax.set(xlabel='False Positive Rate',
       ylabel='True Positive Rate',
```

```
        xlim=[-.01, 1.01], ylim=[-.01, 1.01],
        title='ROC curve')
ax.grid(True)

# Plot the precision-recall curve
ax = axList[1]

precision, recall, _ = precision_recall_curve(y_test, y_prob[:, 1])
ax.plot(recall, precision, linewidth=5)
ax.set(xlabel='Recall', ylabel='Precision',
        xlim=[-.01, 1.01], ylim=[-.01, 1.01],
        title='Precision-Recall curve')
ax.grid(True)

plt.tight_layout()
```

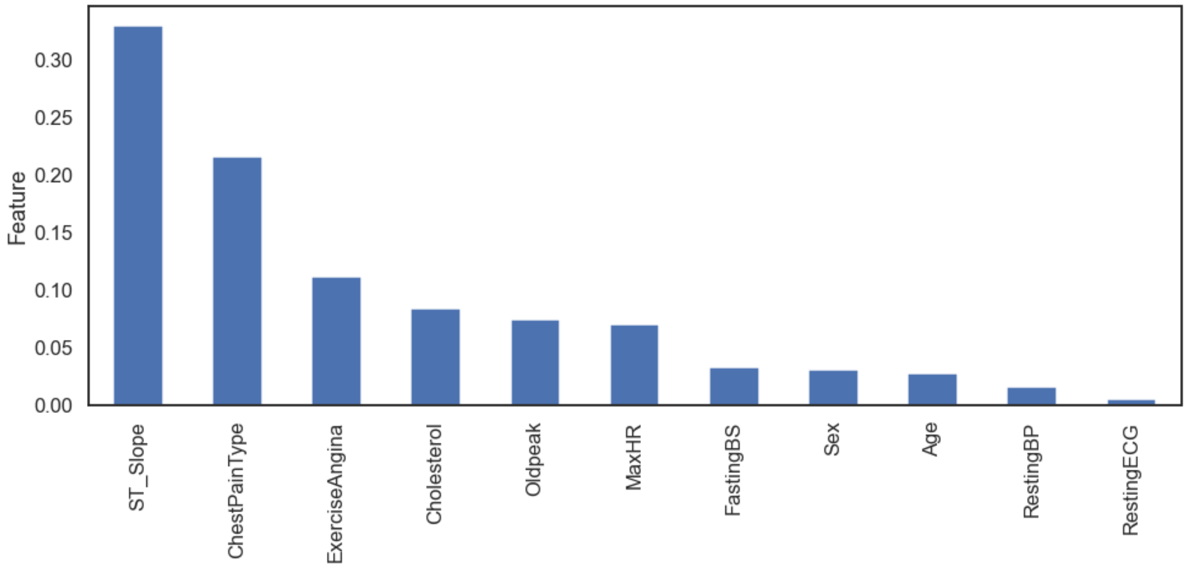


In [ ]:

```
## Create a graph for Feature Importance (Gradient Boosting Classifier)

feature_imp = pd.Series(voting_classifier_pipeline.named_steps['model'].named_esti
                        index=feature_cols).sort_values(ascending=False)

ax = feature_imp.plot(kind='bar', figsize=(16, 6))
ax.set(ylabel='Relative Importance')
ax.set(ylabel='Feature')
```

Out[ ]:

```
[Text(0, 0.5, 'Feature')]
```



Final Model Evaluation Short Recap/Conclusion:

- There's no improvement with 'VotingClassifier'.
- ROC - AUC and Precision - Recall curve seems pretty well.
- As shown from previous pearson correlation, 'ST_slope' and 'ChestPainType' was proven to be highly impactful to model estimation.

2022 | Dimas Adrian Mukti / @berodimas