SC1015
Mini Project

Heart Disease Prediction

A140 Team 8

Aaron Jerome Lim Li Yang (U2221616A)
Brendan Yap Ming Thye (U2221347F)
Chia Jia En (U2221939E)

Practical Motivation

- Heart disease is the **leading cause of death worldwide**, and early detection is key to effective treatment and prevention
- In Singapore, heart disease accounted for 32% of all deaths in 2021
- By accurately predicting the likelihood of developing heart disease, early intervention can be possible

Singapore Statistics

In Singapore, 21 people die from cardiovascular disease (heart diseases and stroke) every day. Cardiovascular disease accounted for 32% of all deaths in 2021. This means that almost 1 out of 3 deaths in Singapore is due to heart diseases or stroke.

DEATHS FROM CARDIOVASCULAR DISEASE

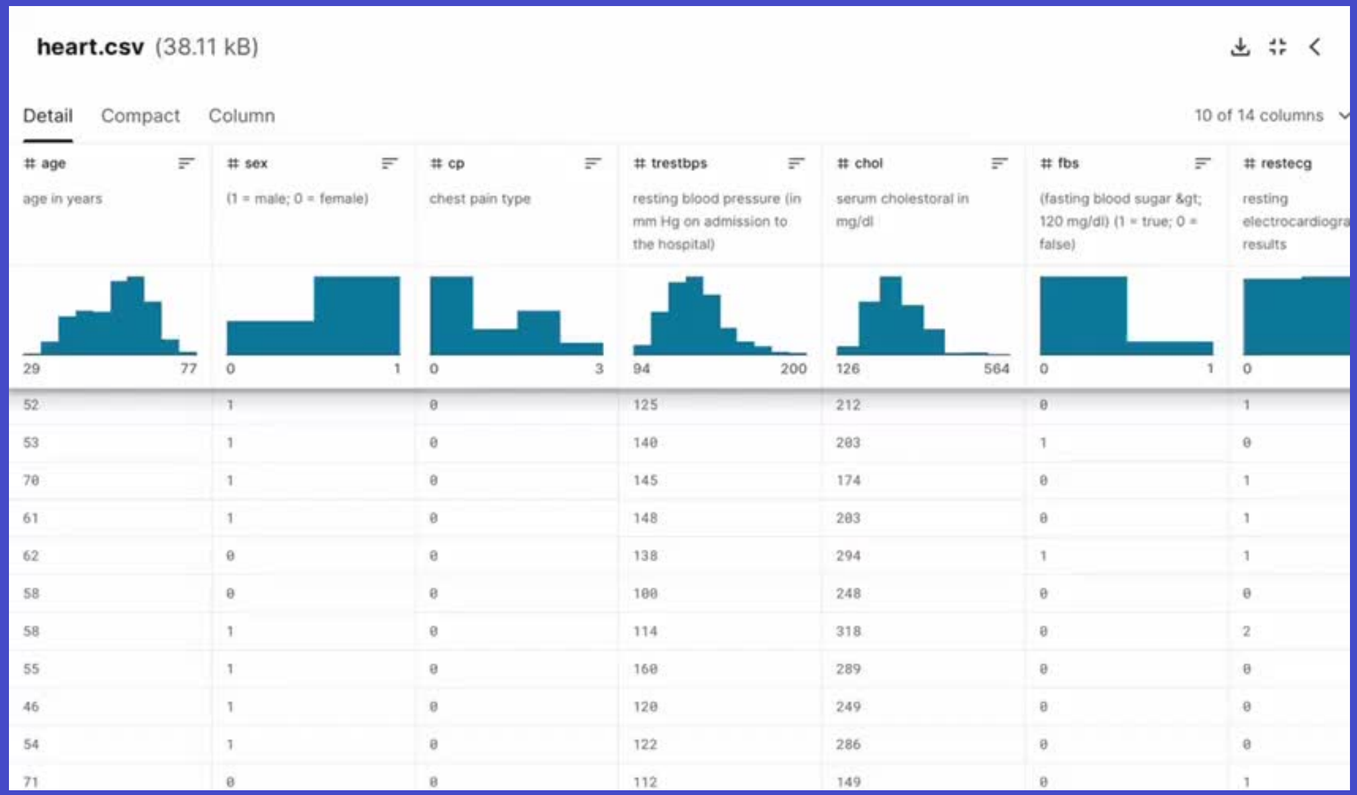
	2021	2020	2019
Total No. of Deaths	24,292	22,054	21,446
Ischaemic Heart Diseases	20.1%	20.5%	18.8%
Cerebrovascular Diseases (including stroke)	6.1%	6.0%	5.8%
Hypertensive Diseases (including hypertensive heart disease)	3.4%	2.9%	2.6%
Other Heart Diseases	2.3%	2.1%	2.0%
Atherosclerosis	0.2%	0.2%	0.1%
Total % of Deaths from Cardiovascular Disease	32.0%	31.7%	29.3%
Total No. of Deaths from Cardiovascular Disease	7,762	6,990	6,291

Source: Ministry of Health

The Dataset

UCI Heart Disease Dataset

- Multivariate: 14 Attributes
- Categorical and Numerical
- Widely used in ML research
- High revelance in the medical space



Heart Disease Dataset

Public Health Dataset

kaggle.com

Problem Definition

Given various health parameters are we able to accurately predict whether a patient is likely or not to develop heart disease?

Problem Formulation

Binary Classification

Preliminary Exploration

Quick Observations

- Response variable: *target*
- Predictors: *remaining 13 features*
- No null values at first glance

```
# shape of dataset
print('Data type: ', type(df))
print('Data dims: ', df.shape)
```

```
Data type:  <class 'pandas.core.frame.DataFrame'
Data dims:  (1025, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1025 non-null   int64  
 1   sex       1025 non-null   int64  
 2   cp        1025 non-null   int64  
 3   trestbps  1025 non-null   int64  
 4   chol      1025 non-null   int64  
 5   fbs       1025 non-null   int64  
 6   restecg   1025 non-null   int64  
 7   thalach   1025 non-null   int64  
 8   exang     1025 non-null   int64  
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64  
 11  ca        1025 non-null   int64  
 12  thal      1025 non-null   int64  
 13  target    1025 non-null   int64  
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
```

Description of Variables

1. **age**: age in years
2. **sex**: sex
   - 0: female
   - 1: male
3. **cp**: chest pain type
   - 0: asymptomatic
   - 1: typical angina
   - 2: atypical angina
   - 3: non-anginal pain
4. **trestbps**: resting blood pressure (in mm Hg on admission to the hospital)
5. **chol**: serum cholestoral in mg/dl
6. **fbs**: (fasting blood sugar > 120 mg/dl)
   - 1: true
   - 0: false
7. **restecg**: resting electrocardiographic results
   - 0: normal
   - 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
   - 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. **thalach**: maximum heart rate achieved
9. **exang**: exercise induced angina
   - 1: yes
   - 0: no
10. **oldpeak**: ST depression induced by exercise relative to rest
11. **slope**: the slope of the peak exercise ST segment
    - 1: upsloping
    - 2: flat
    - 3: downsloping
12. **ca**: number of major vessels (0-3) colored by flourosopy
13. **thal**: Thalassemia
    - 1 = normal
    - 2 = fixed defect
    - 3 = reversable defect
14. **target**: diagnosis of heart disease
    - 0: < 50% diameter narrowing
    - 1: > 50% diameter narrowing

In [8]:
```
# use a dictionary to map numeric values to actual definition, for each categorical variable
dict1 = {'sex': {0: 'Female', 1: 'Male'},
         'cp': {0: 'Asymptomatic', 1: 'Typical', 2: 'Atypical', 3: 'Non-Anginal'},
         'fbs': {0: 'False', 1: 'True'},
         'restecg': {0: 'Normal', 1: 'ST-T Wave Abnormality', 2: 'Probable/Definite LVH'},
         'exang': {0: 'No', 1: 'Yes'},
         'slope': {1: 'Upsloping', 2: 'Flat', 3: 'Downsloping'},
         'ca': {0: 0, 1: 1, 2: 2, 3: 3},
         'thal': {1: 'Normal', 2: 'Fixed', 3: 'Reversible'},
         'target': {0: 'No', 1: 'Yes'}}

# create temporary dataframe for EDA
df_new = df.copy()
for i in dict1:
    df_new = df_new.replace({i: dict1[i]})
```

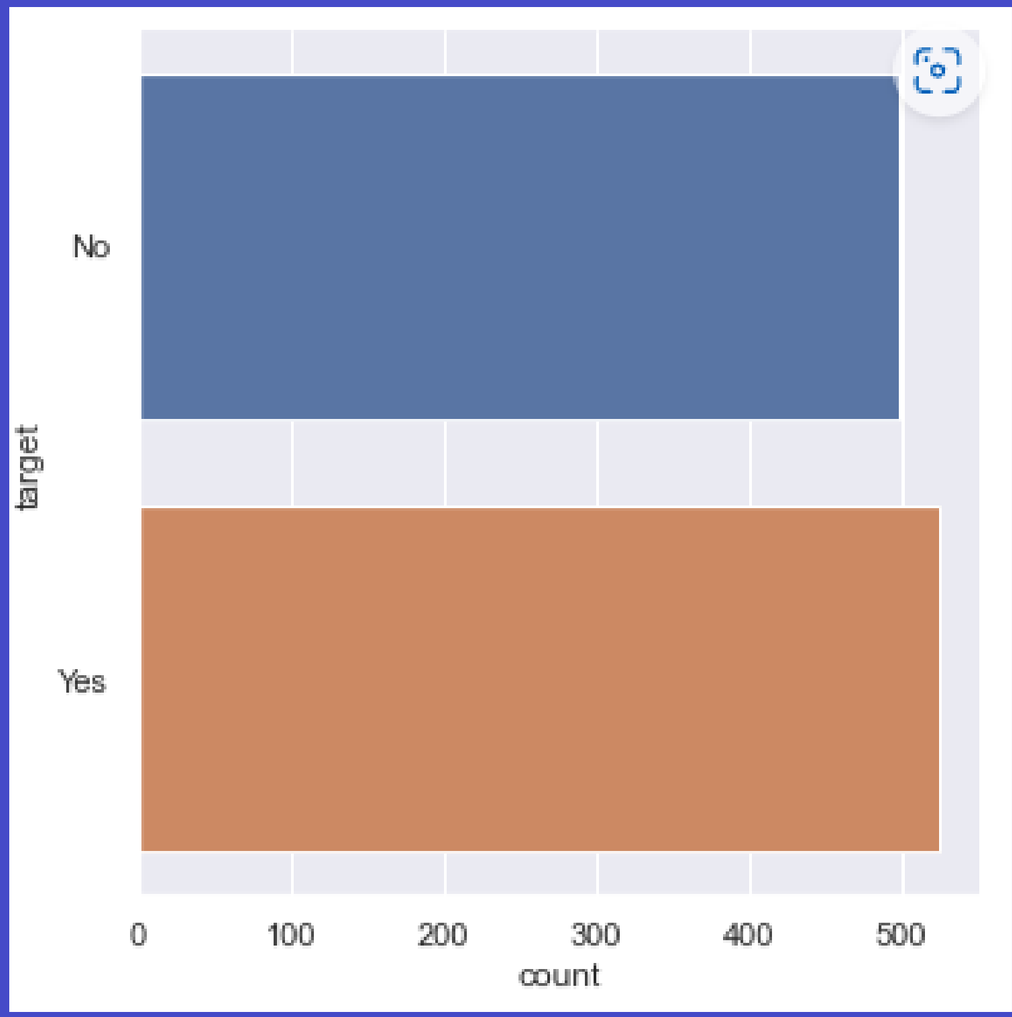
In [9]:
```
df_new.head()
```

Out[9]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	Male	Asymptomatic	125	212	False	ST-T Wave Abnormality	168	No	1.0	Flat	2	Reversible	No
1	53	Male	Asymptomatic	140	203	True	Normal	155	Yes	3.1	0	0	Reversible	No
2	70	Male	Asymptomatic	145	174	False	ST-T Wave Abnormality	125	Yes	2.6	0	0	Reversible	No
3	61	Male	Asymptomatic	148	203	False	ST-T Wave Abnormality	161	No	0.0	Flat	1	Reversible	No
4	62	Female	Asymptomatic	138	294	True	ST-T Wave Abnormality	106	No	1.9	Upsloping	3	Fixed	No

Univariate Analysis

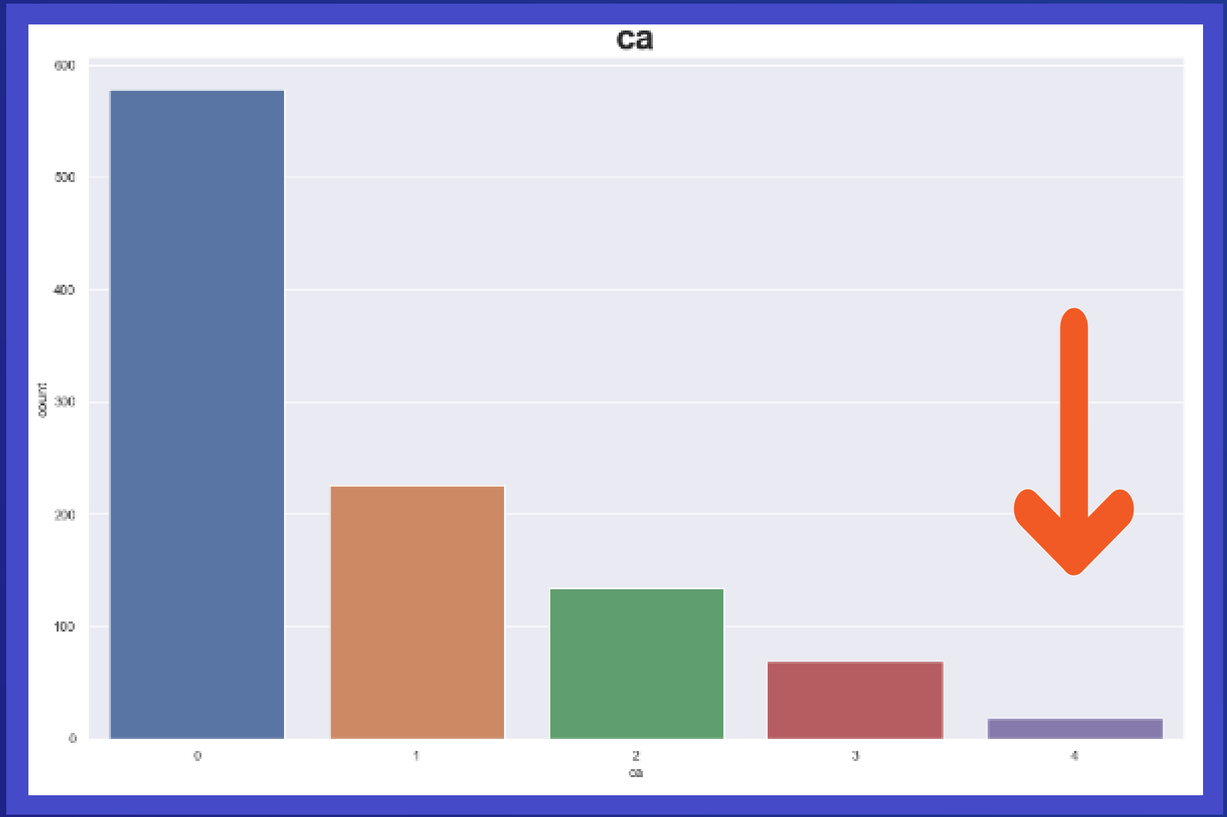
Response Variable (Target)

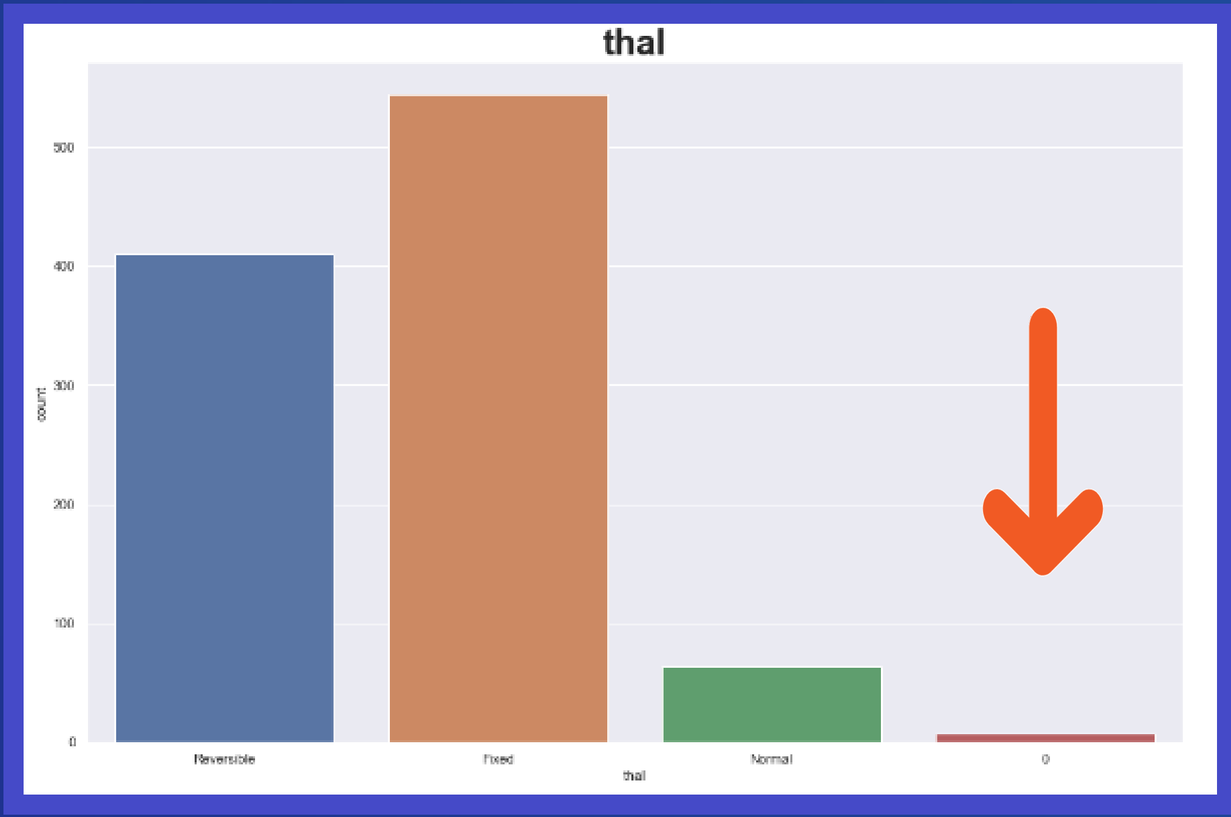


Univariate Analysis

Categorical Variables

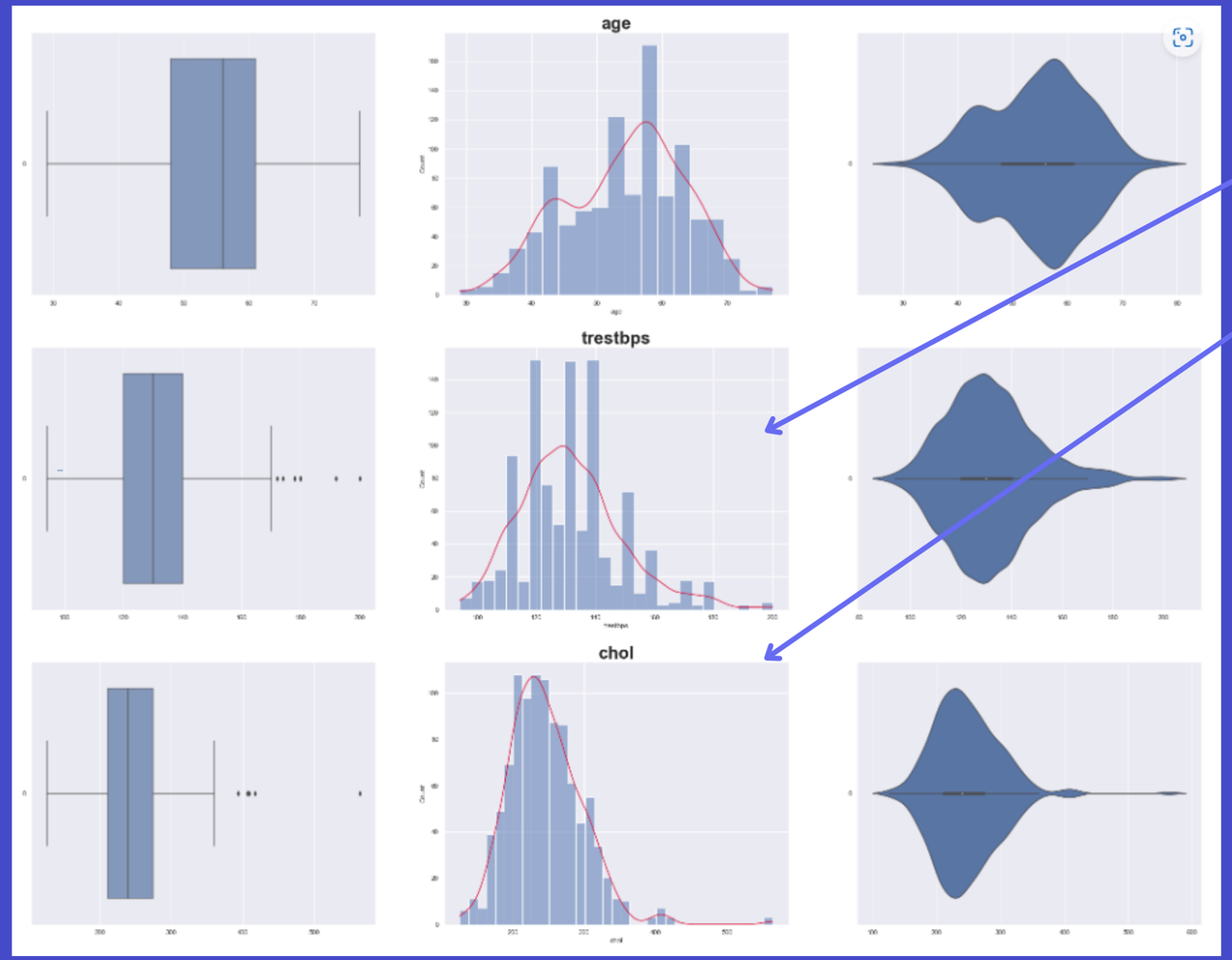
Undefined classes? -> Data cleaning required





Univariate Analysis

Continuous Variables



Moderately Skewed:

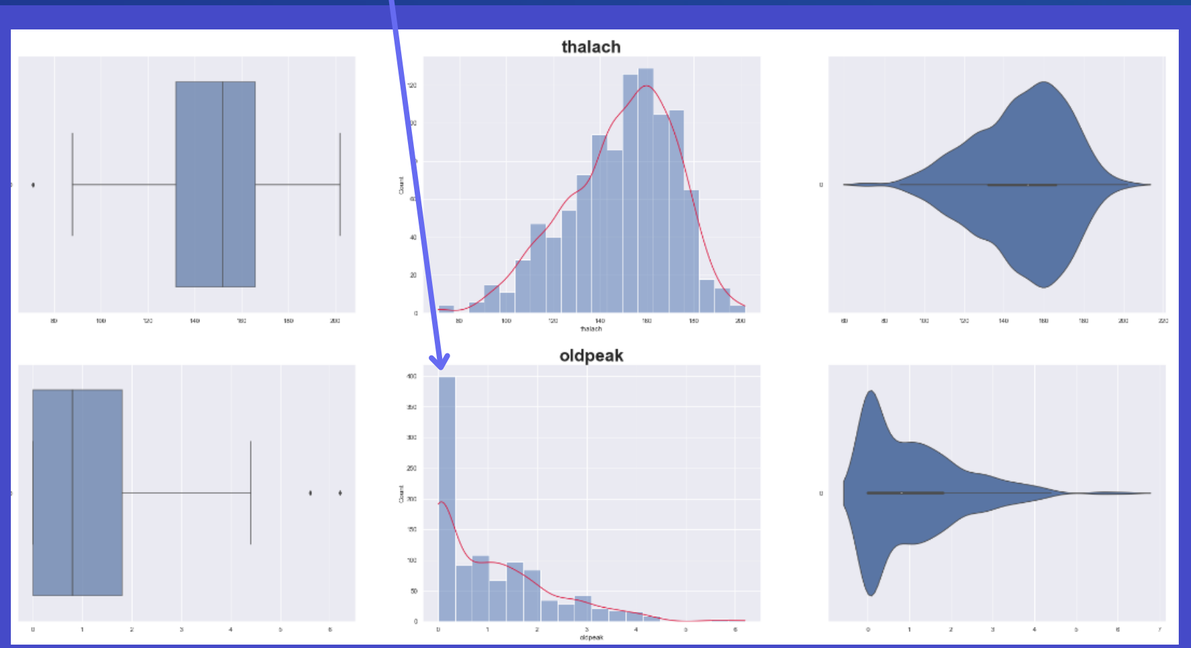
- **trestbps**

Highly Skewed:

- **chol**
- **oldpeak**

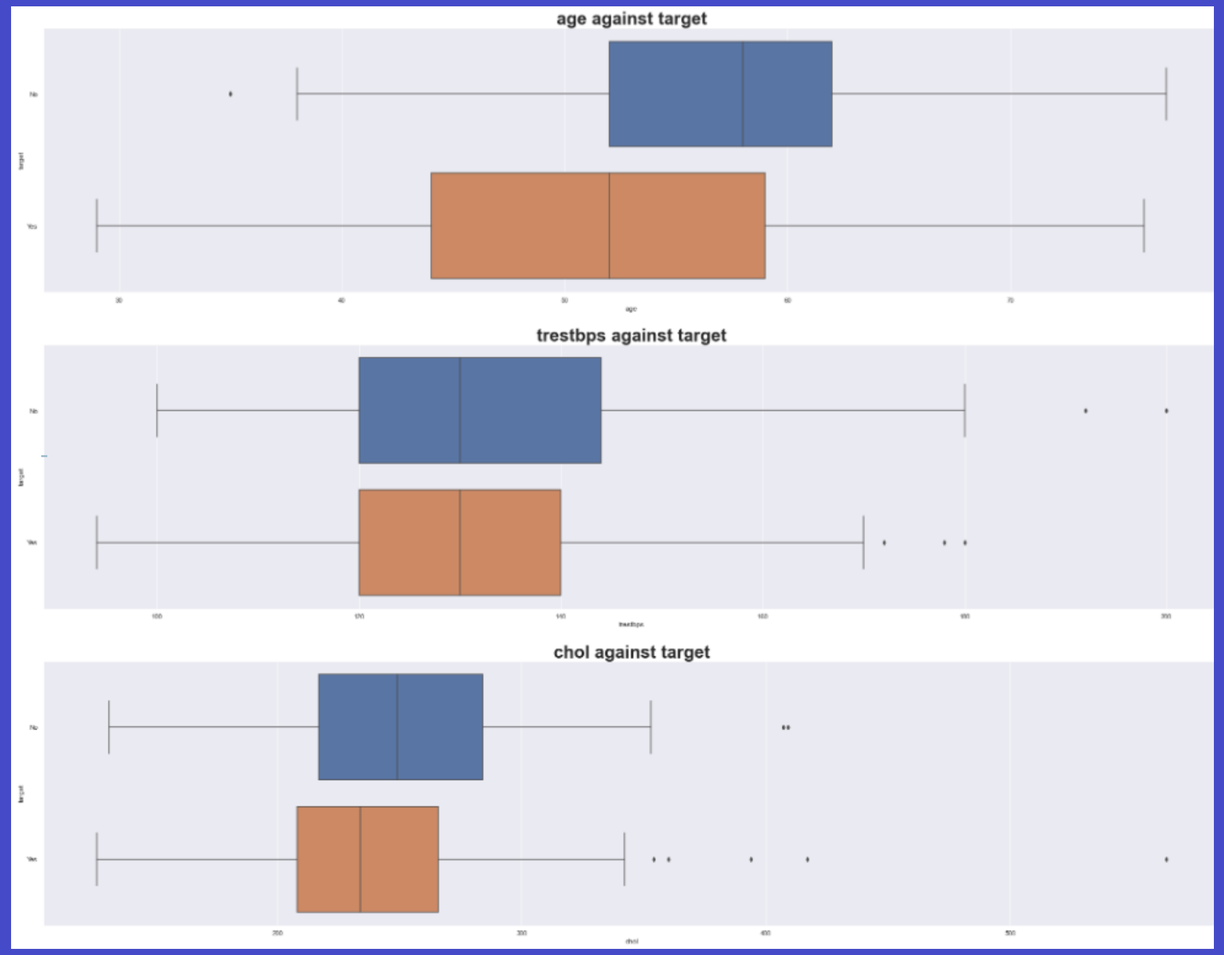
```
print("Skewness:")
df_new[cont].skew(axis=0)
```

```
Skewness:

age         -0.248866
trestbps     0.739768
chol         1.074073
thalach     -0.513777
oldpeak      1.210899
dtype: float64
```

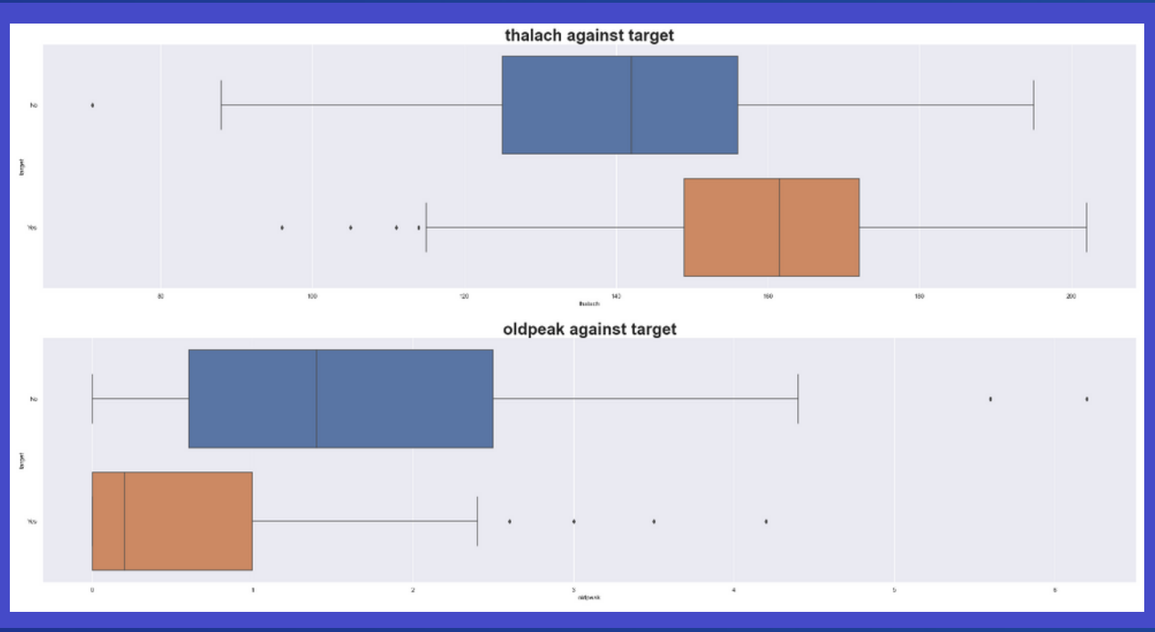


Multivariate Analysis

Continuous Variables against Response



What do the data distributions tell us?

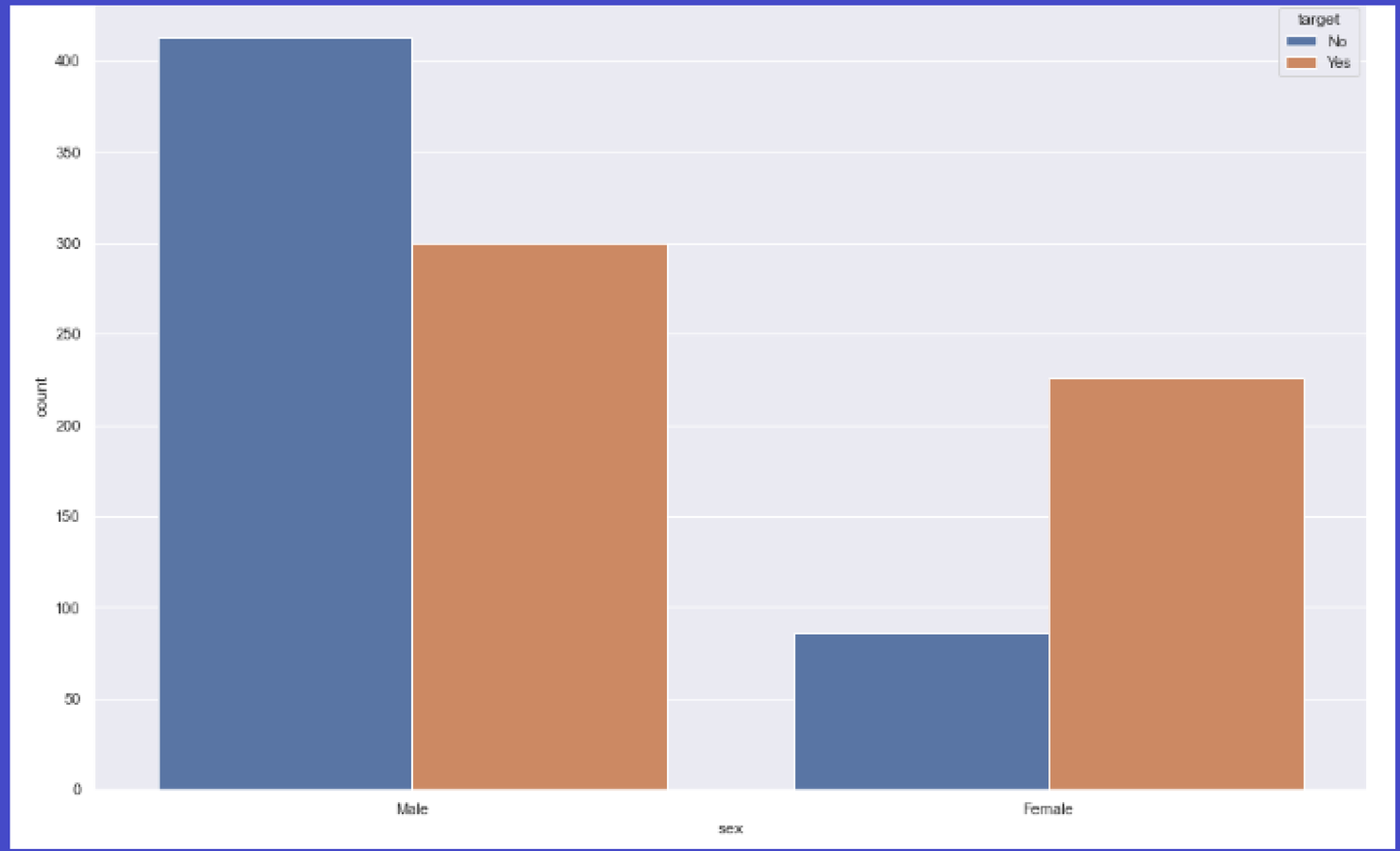


Multivariate Analysis

Categorical Variables against Response

Sex against Target

- Higher ratio with *no* heart disease in *males*
- Higher ratio with heart disease in *females*

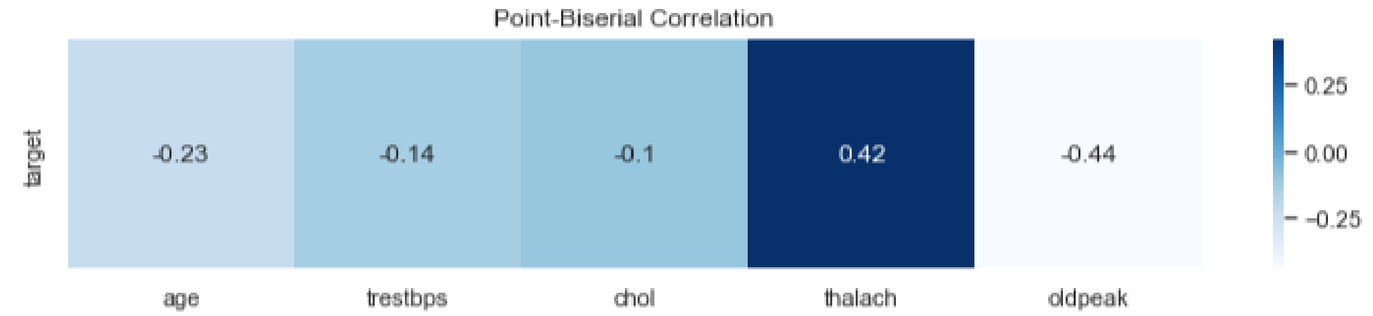


Multivariate Analysis

Point Biserial Correlation Matrix

- *thalach* is moderately positively-correlated
- *oldpeak* is moderately negatively-correlated
- other variables are only mildy correlated with target

```python
from scipy.stats import pointbiserialr as pbs

pbs_results = []

# Point biserial correlation
for feature in cont:
    pbs_results.append(pbs(df[feature], df['target'])[0])

# Plot heatmap
fig = plt.figure(figsize=(12,2))
sb.heatmap([pbs_results], annot=True, cmap='Blues',
           xticklabels=cont, yticklabels=['target'])
plt.title("Point-Biserial Correlation")
plt.show()
```



Data Cleaning ('thal')

Undefined 'thal' value: 0

- We regard as MCAR value

Data imputation needed

- Sci-Kit Learn's ***KNNImputer*** utilised

```
df['thal'].value_counts()
```

```
2    544
3    410
1     64
0      7
Name: thal, dtype: int64
```

*from variable description on kaggle

thal

1 = normal; 2 = fixed defect; 3 = reversable defect

```
# initialize KNNImputer
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=5, weights='uniform', metric='nan_euclidean')

# convert `thal = 0` values to NaN
df['thal'] = df['thal'].convert_dtypes()
df.loc[df['thal'] == 0, 'thal'] = np.NaN

# use KNNImputer
imputed = imputer.fit_transform(df)
df_imputed = pd.DataFrame(imputed, columns=df.columns)
df_imputed['thal'] = np.round(df_imputed['thal'])
df_imputed = df_imputed.astype(df.dtypes.to_dict()) # retain data types from original dataset
```

```
df_imputed['thal'].value_counts()
```

```
2    551
3    410
1     64
Name: thal, dtype: Int64
```

Data Cleaning ('ca')

Undefined 'ca' value: 4

- ca = 4 has some meaning that we do not know
- Does not represent missing data

Further analysis of dataset

- There are 18 instances of ca = 4 ; 1% of total samples
- All ca = 4 is linked with a male patient

```
df_imputed['ca'].value_counts()

0    578
1    226
2    134
3     69
4     18
Name: ca, dtype: int64
```

*from variable description on kaggle

 # ca

number of major vessels (0-3) colored by flourosopy

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
38	1	2	138	175	0	1	173	0	0	2	4	2	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
52	1	2	138	223	0	1	169	0	0	2	4	2	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
52	1	2	138	223	0	1	169	0	0	2	4	2	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
43	1	0	132	247	1	0	143	1	0.1	1	4	3	0
52	1	2	138	223	0	1	169	0	0	2	4	2	1
43	1	0	132	247	1	0	143	1	0.1	1	4	3	0
38	1	2	138	175	0	1	173	0	0	2	4	2	1
58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
58	1	1	125	220	0	1	144	0	0.4	1	4	3	1
38	1	2	138	175	0	1	173	0	0	2	4	2	1
43	1	0	132	247	1	0	143	1	0.1	1	4	3	0

Data Cleaning ('ca')

Data imputation *unsuitable* ✗

Removal of samples *more suitable* ✓

```
df_imputed = df_imputed[df_imputed.ca != 4]
df_imputed['ca'].value_counts()
```

```
0    578
1    226
2    134
3     69
Name: ca, dtype: int64
```

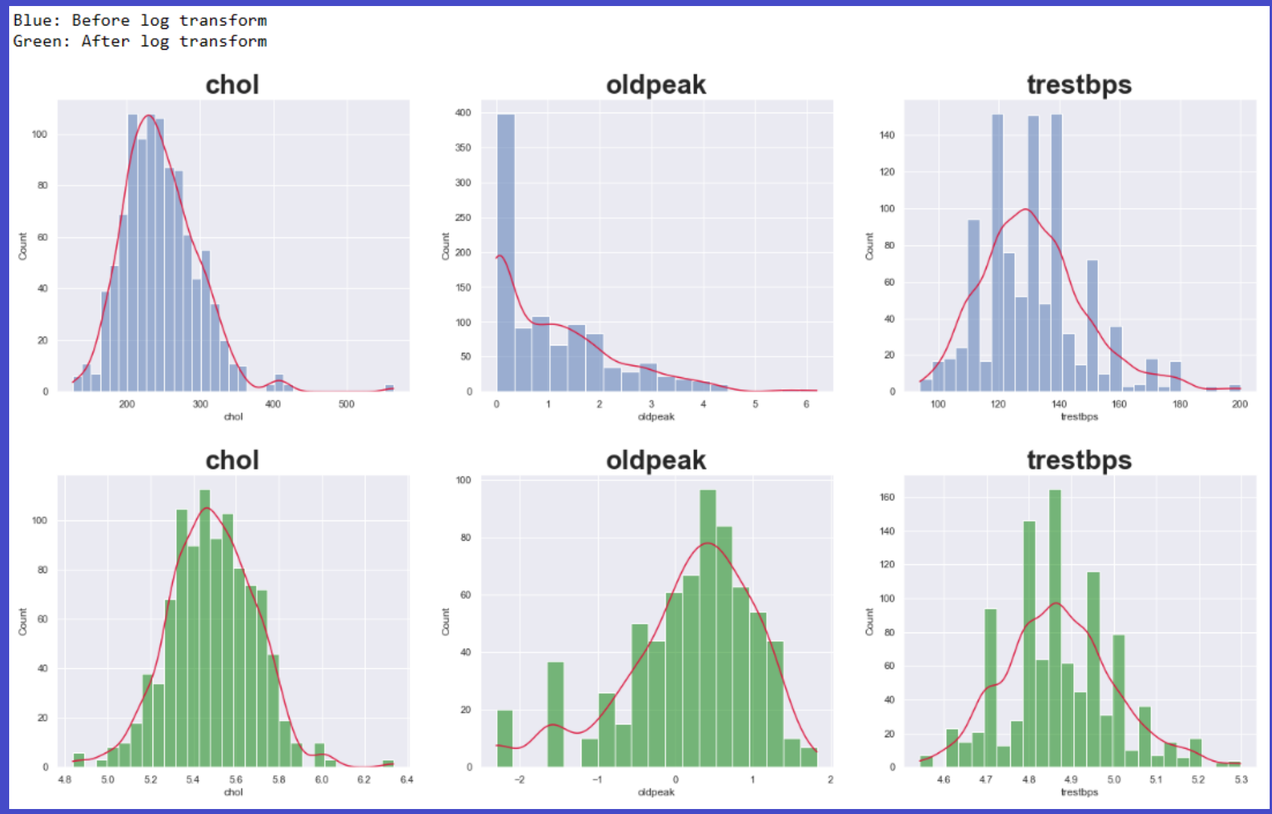
Data Normalisation

Transforming positively skewed predictors identified earlier using log transformation

- 'chol'
- 'oldpeak'
- 'trestbps'

Predictors now have a more normal distribution

```
f, axes = plt.subplots(1, 3, figsize=(24,6))
print("Blue: Before log transform")
print("Green: After log transform")

count = 0
for feature in skew_list:
    g = sb.histplot(data=df_new[feature], ax=axes[count], kde=True)
    g.set_title(feature, fontdict={'fontsize':30, 'fontweight':'bold'})
    g.lines[0].set_color('crimson')
    count+=1

# after normalization
f, axes = plt.subplots(1, 3, figsize=(24,6))

count = 0
for feature in skew_list:
    g = sb.histplot(data=df_trans[feature], ax=axes[count], kde=True, color='green')
    g.set_title(feature, fontdict={'fontsize':30, 'fontweight':'bold'})
    g.lines[0].set_color('crimson')
    count+=1
```



Data Standardisation and One-Hot Encoding

```
from sklearn.preprocessing import StandardScaler

# initialize scaler
scaler = StandardScaler()

# scale and encode columns
scaled_cols = scaler.fit_transform(df_imputed[cont])
ohe_cols = pd.get_dummies(df_imputed[cat], columns = cat)

# convert scaled columns to dataframe
scaled_cols = pd.DataFrame(scaled_cols, index=df_imputed[cont].index, columns=df_imputed[cont].columns)

# concatenate columns back together
df_processed = pd.concat([scaled_cols, ohe_cols, df_imputed['target'].astype('uint8')], axis=1)

df_processed.head()
```

	age	trestbps	chol	thalach	oldpeak	sex_0	sex_1	cp_0	cp_1	cp_2	...	slope_1	slope_2	ca_0	ca_1	ca_2	ca_3	thal_1	thal_2	thal_3
0	-0.287864	-0.372147	-0.673204	0.827753	-0.075393	0	1	1	0	0	...	0	1	0	0	1	0	0	0	1
1	-0.176799	0.477888	-0.847645	0.264253	1.707976	0	1	1	0	0	...	0	0	1	0	0	0	0	0	1
2	1.711300	0.761232	-1.409733	-1.036134	1.283364	0	1	1	0	0	...	0	0	1	0	0	0	0	0	1
3	0.711718	0.931239	-0.847645	0.524330	-0.924616	0	1	1	0	0	...	0	1	0	1	0	0	0	0	1
4	0.822783	0.364550	0.916148	-1.859712	0.688908	1	0	1	0	0	...	1	0	0	0	0	1	0	1	0

5 rows × 29 columns

```
df_processed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1007 entries, 0 to 1024
Data columns (total 29 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   age         1007 non-null   float64
 1   trestbps    1007 non-null   float64
 2   chol        1007 non-null   float64
 3   thalach     1007 non-null   float64
 4   oldpeak     1007 non-null   float64
 5   sex_0       1007 non-null   uint8  
 6   sex_1       1007 non-null   uint8  
 7   cp_0        1007 non-null   uint8  
 8   cp_1        1007 non-null   uint8  
 9   cp_2        1007 non-null   uint8  
 10  cp_3        1007 non-null   uint8  
 11  fbs_0       1007 non-null   uint8  
 12  fbs_1       1007 non-null   uint8  
 13  restecg_0   1007 non-null   uint8  
 14  restecg_1   1007 non-null   uint8  
 15  restecg_2   1007 non-null   uint8  
 16  exang_0     1007 non-null   uint8  
 17  exang_1     1007 non-null   uint8  
 18  slope_0     1007 non-null   uint8  
 19  slope_1     1007 non-null   uint8  
 20  slope_2     1007 non-null   uint8  
 21  ca_0        1007 non-null   uint8  
 22  ca_1        1007 non-null   uint8  
 23  ca_2        1007 non-null   uint8  
 24  ca_3        1007 non-null   uint8  
 25  thal_1      1007 non-null   uint8  
 26  thal_2      1007 non-null   uint8  
 27  thal_3      1007 non-null   uint8  
 28  target      1007 non-null   uint8  
dtypes: float64(5), uint8(24)
memory usage: 70.8 KB
```

Feature Selection

Random Forest Classifier

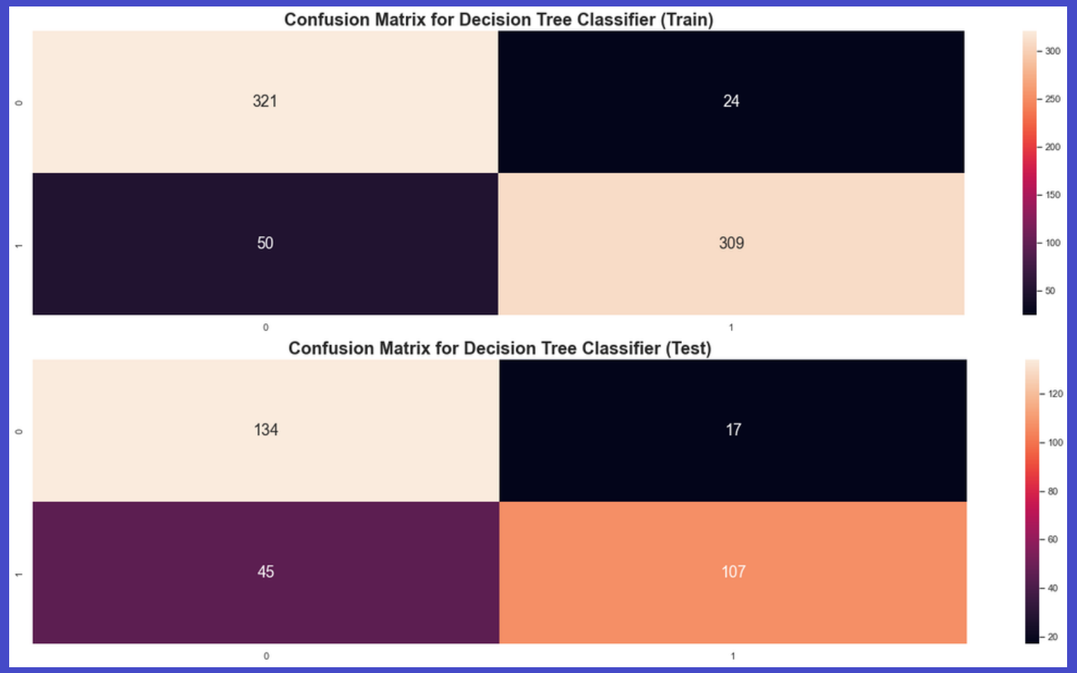
```python
from sklearn.ensemble import RandomForestClassifier

# split data into X and y
y = pd.DataFrame(df_processed['target'])
X = pd.DataFrame(df_processed.drop('target', axis = 1))

# initialize RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=200, random_state=0)
rfc.fit(X, y.values.ravel())

# create list of column names
df_labels = list(df_processed.columns.values)

# get importances of features
imptance = rfc.feature_importances_
imptance_sorted = imptance.argsort()[::-1]

# print each feature from most important to least important
for i in imptance_sorted:
    print("{} - {:.2f}%".format(df_labels[i], (imptance[i] * 100.0)))
```

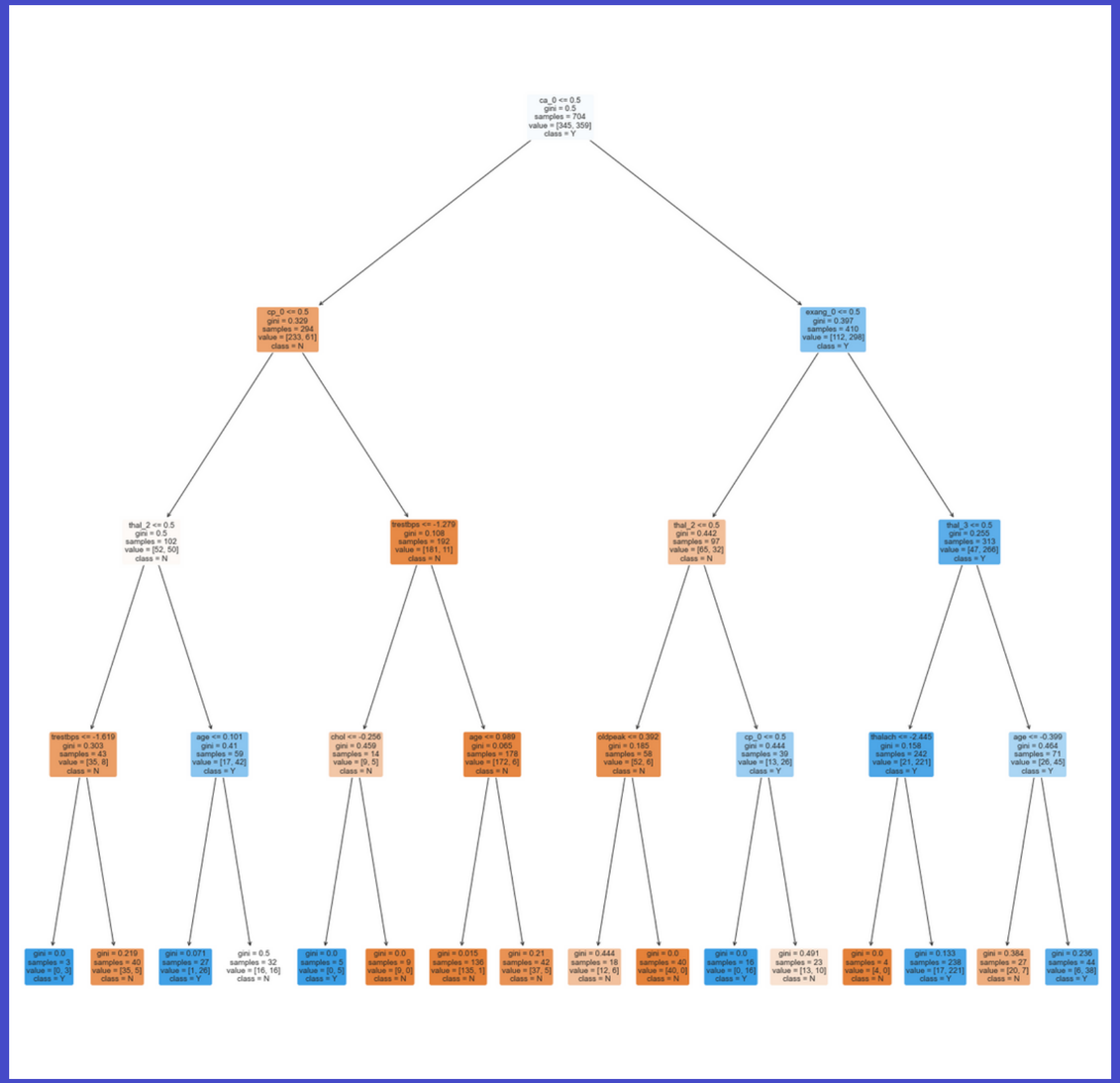
```
ca_0 - 10.19%
oldpeak - 9.01%
cp_0 - 9.01%
thalach - 8.88%
age - 7.64%
thal_2 - 7.28%
chol - 6.71%
thal_3 - 6.50%
trestbps - 6.01%
exang_0 - 3.89%
exang_1 - 3.50%
slope_2 - 2.54%
sex_1 - 2.24%
slope_1 - 2.02%
cp_2 - 1.95%
sex_0 - 1.87%
ca_1 - 1.63%
restecg_0 - 1.31%
restecg_1 - 1.22%
cp_3 - 1.14%
ca_2 - 1.02%
cp_1 - 0.94%
fbs_0 - 0.83%
ca_3 - 0.81%
fbs_1 - 0.78%
slope_0 - 0.52%
thal_1 - 0.48%
restecg_2 - 0.07%
```

Preliminary Model

Decision Tree Classifier

- Classification accuracy (Train) : 0.89
- Classification accuracy (Test) : 0.79
- TPR / FPR (Train) : 0.861 / 0.070
- TPR / FPR (Test) : 0.704 / 0.113





Model Selection

K-Fold Cross Validation

```python
# list of models' names
names = [
    "Logistic Regression",
    "SGD",
    "Ridge",
    "K-Nearest Neighbors",
    "Linear SVC",
    "RBF SVC",
    "Random Forest",
    "Ada Boost",
    "Gradient Boosting",
    "Bagging",
    "Naive Bayes",
    "XG Boost",
    "Decision Tree"
]

# list of classifier models
models = [
    LogisticRegression(),
    SGDClassifier(),
    RidgeClassifier(),
    KNeighborsClassifier(),
    SVC(kernel="linear"),
    SVC(),
    RandomForestClassifier(),
    AdaBoostClassifier(),
    GradientBoostingClassifier(),
    BaggingClassifier(),
    GaussianNB(),
    XGBClassifier(),
    DecisionTreeClassifier()
]

name = 0
scoring = 'f1'
cm_dict = {}

for model in models:
    kfold = KFold(n_splits = 10) # value with low bias and modest variance
    result = cross_val_score(model, X_train, y_train.values.ravel(), cv=kfold, scoring=scoring)
    cm_dict[names[name]] = [result.mean(), result.std()]
    #print('%s:\t %f (%f)' % (names[name], result.mean(), result.std()))
    name+=1

print("Mean Accuracy of Models")
for k,v in sorted(cm_dict.items(), key=lambda p:p[1][0], reverse=True):
    print('%f: %s' % (v[0], k))
```

```python
# Import essential models and functions from sklearn
from sklearn.model_selection import cross_val_score, GridSearchCV, KFold
from sklearn.metrics import auc, confusion_matrix, classification_report, accuracy_score, mean_absolute_error

# classifier models that can be applied to binary classification
from sklearn.linear_model import LogisticRegression, SGDClassifier, RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
```

```
Mean Accuracy of Models
0.978056: Random Forest
0.963302: XG Boost
0.957205: Decision Tree
0.953705: Bagging
0.953686: Gradient Boosting
0.927590: RBF SVC
0.901465: Ada Boost
0.876725: Logistic Regression
0.872117: Ridge
0.868320: Linear SVC
0.844533: Naive Bayes
0.842023: K-Nearest Neighbors
0.823904: SGD
```

Top 3 Models

1. Random Forest
2. Bagging
3. XG Boost

Hyperparameter Tuning

GridSearch

Random Tree Classifier

```
rfc = RandomForestClassifier()

grid_params = {
    'max_depth': range(4, 20, 2),     # keep <10 so that trees don't overfit
    'n_estimators': range(100, 501, 100),
    'max_features': range(4, 28, 4),
    'max_samples': [x / 20 for x in range(1, 11)]
}

gs_rfc = GridSearchCV(rfc, grid_params, cv=5, n_jobs=-1, scoring='f1')
gs_rfc.fit(X_train, y_train.values.ravel())
```

```
F1-Score before GridSearchCV:
0.9537054
F1-Score after GridSearchCV:
0.9688628
% increase in F1-Score:
1.589%
```

Bagging

```
bg = BaggingClassifier()

grid_params = {
    'n_estimators': range(100, 501, 100),
    'max_features': range(4, 28, 4),
    'max_samples': [x / 20 for x in range(1, 11)]
}

gs_bg = GridSearchCV(bg, grid_params, cv=5, n_jobs=-1, scoring='f1')
gs_bg.fit(X_train, y_train.values.ravel())
```

```
F1-Score before GridSearchCV:
0.9537054
F1-Score after GridSearchCV:
0.9688628
% increase in F1-Score:
1.589%
```

XG Boost

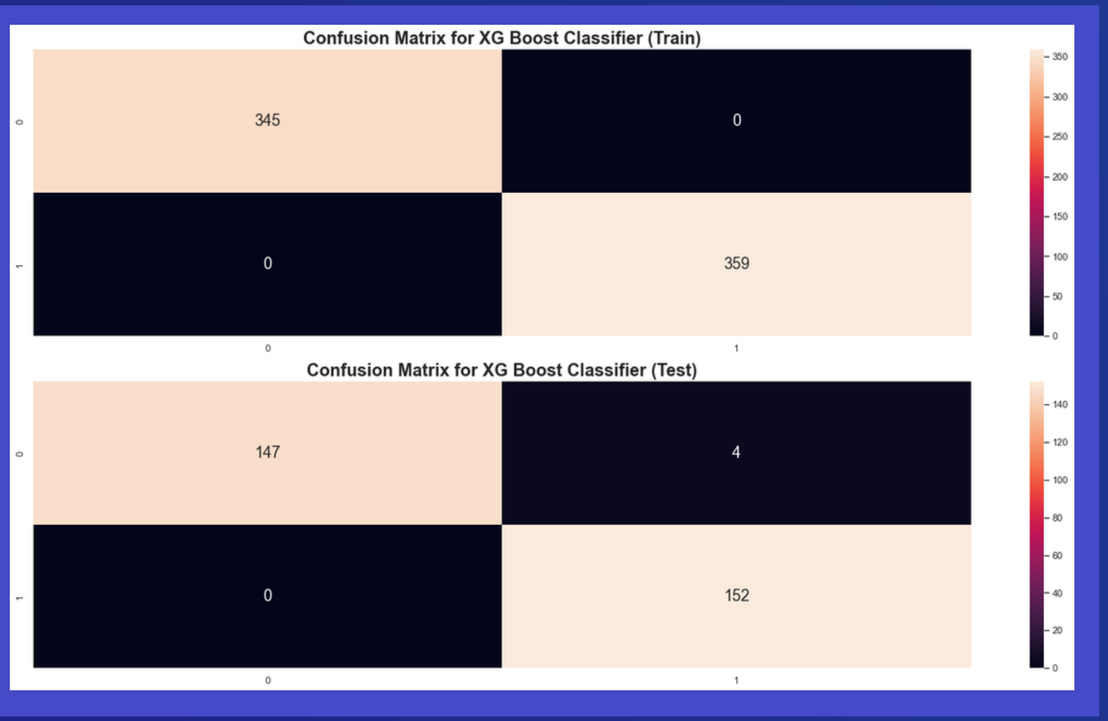
```
# XG Boost Classifier
grid_params = {
    'max_depth': range(4, 11, 2),
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.5, 0.7, 0.9, 1.0],
    'learning_rate': [0.05, 0.12, 0.18, 0.24, 0.3],
    'n_estimators': range(100, 1001, 150),
    'gamma': [0, 1, 5, 10]
}

gs_xg = GridSearchCV(xgb, grid_params, cv=5, n_jobs=-1, scoring='f1')
gs_xg.fit(X_train, y_train)
```

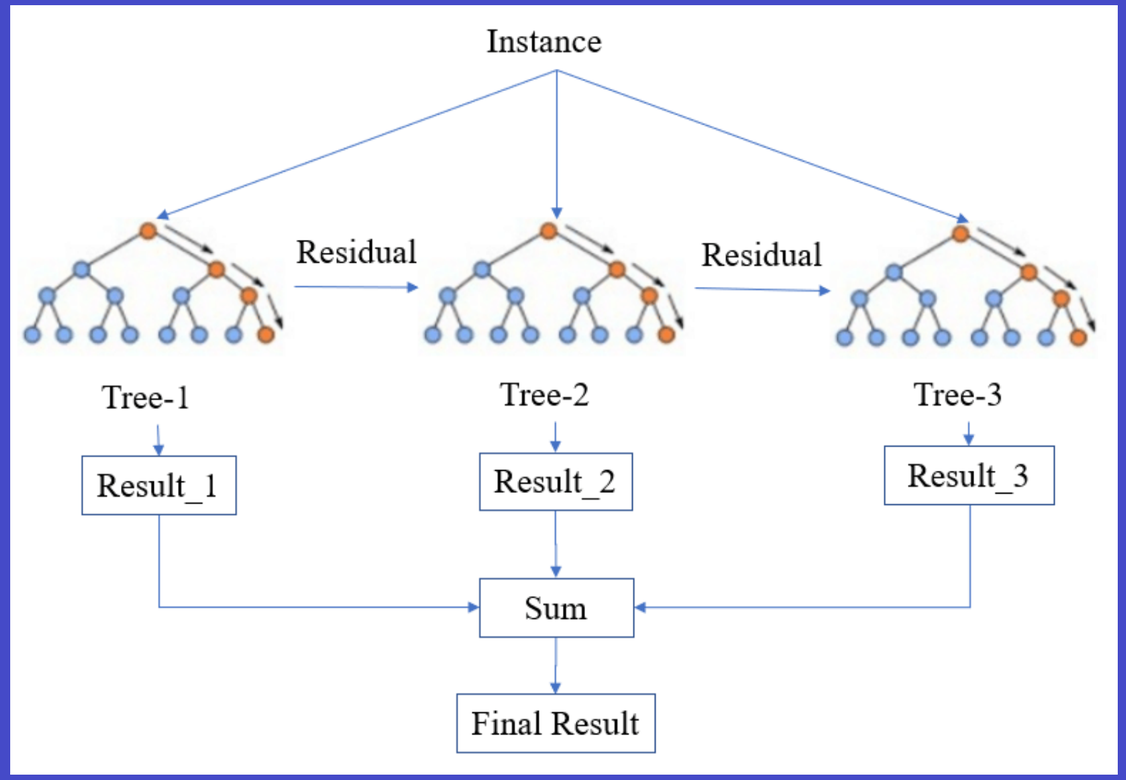
```
F1-Score before GridSearchCV:
0.9633016
F1-Score after GridSearchCV:
0.9749278
% increase in F1-Score:
1.207%
```

Final Prediction Model

XG Boost

- Classification accuracy (Train) : 1.000
- Classification accuracy (Test) : 0.987
- TPR / FPR (Train) : 1.000 / 0.000
- TPR / FPR (Test) : 1.000 / 0.026





Data-driven Insights

Assessing Model Performance

- At first glance, our prediction model appears to be *overfitting* based on its *perfect accuracy* on the *training* dataset
- However, it maintains a high prediction accuracy on unseen data (*test* dataset)
- Thus, it proves that our model is reliable and can be used to make accurate predictions on general data

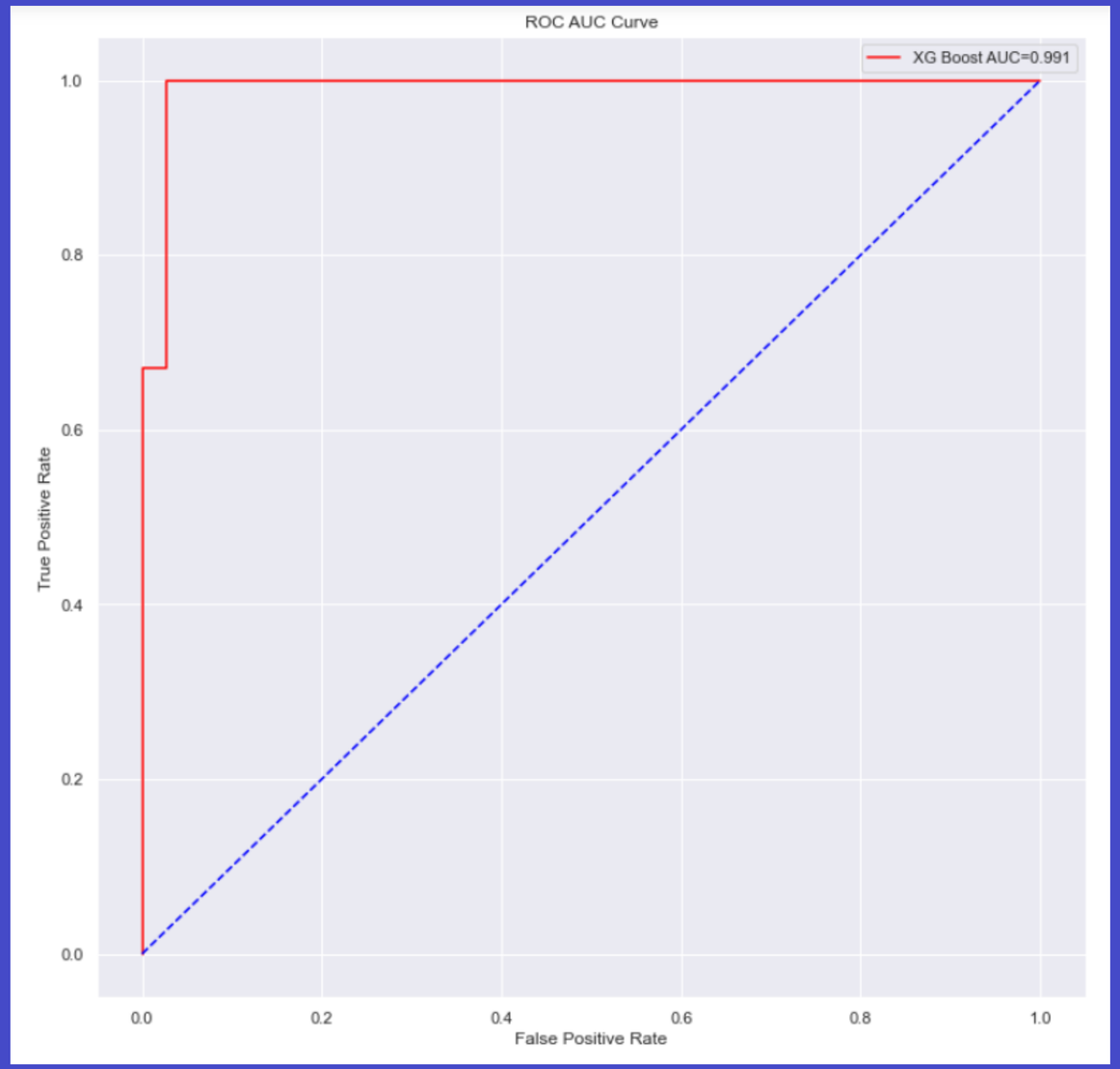
- Classification accuracy (Train) : 1.000
- Classification accuracy (Test) : 0.987
- TPR / FPR (Train) : 1.000 / 0.000
- TPR / FPR (Test) : 1.000 / 0.026

Data-driven Insights

ROC AUC Curve

- Receiver Operator Characteristic (ROC); Area Under Curve (AUC)
- The closer to 1, the better the model is at distinguishing the classes
- AUC Score: 0.991

```python
from sklearn.metrics import roc_curve, roc_auc_score

# predict probabilities for XG Boost
xg_pred_prob = gs_xg.predict_proba(X_test)

# ROC curve
fprxg, tprxg, _ = roc_curve(y_test, xg_pred_prob[:,1])

# AUC score
aucxg = roc_auc_score(y_test, xg_pred_prob[:,1])

# Constructing the ROC AUC plot
f = plt.figure(figsize=(12,12))
plt.plot(fprxg, tprxg, color='red', label=f'XG Boost AUC={aucxg:.3f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='blue') # ROC curve for TPR = FPR
plt.title("ROC AUC Curve")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='best')
plt.show()
```



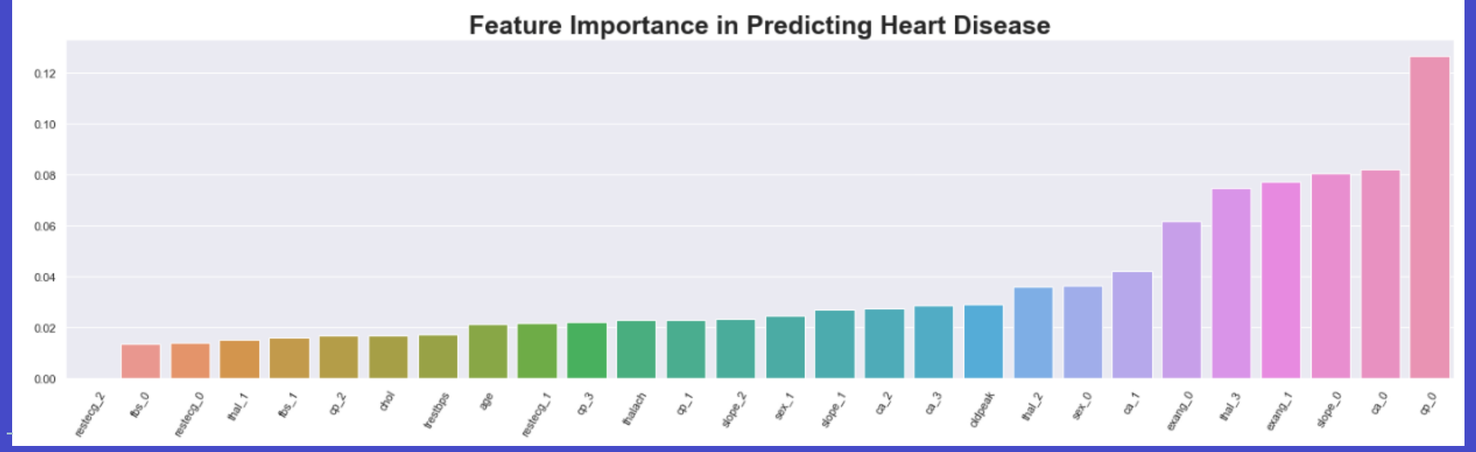
Data-driven Insights

Insights of Predictors Contribution

- Through analysis of our model, we are able to gain insights into the contributions of each feature towards heart disease

```
# get importances of features
imptance = gs_xg.best_estimator_.feature_importances_

keys = list(X.columns.values)
FI_dict = dict(zip(keys, imptance))
FI_dict = dict(sorted(FI_dict.items(), key=lambda item: item[1]))

# plot feature importances
f = plt.figure(figsize=(24,6))
g = sb.barplot(x=list(FI_dict.keys()), y=list(FI_dict.values()))
g.set_xticklabels(g.get_xticklabels(), rotation=60)
g.set_title('Feature Importance in Predicting Heart Disease', fontdict={'fontsize':25, 'fontweight':'bold'})
```

```
Text(0.5, 1.0, 'Feature Importance in Predicting Heart Disease')
```



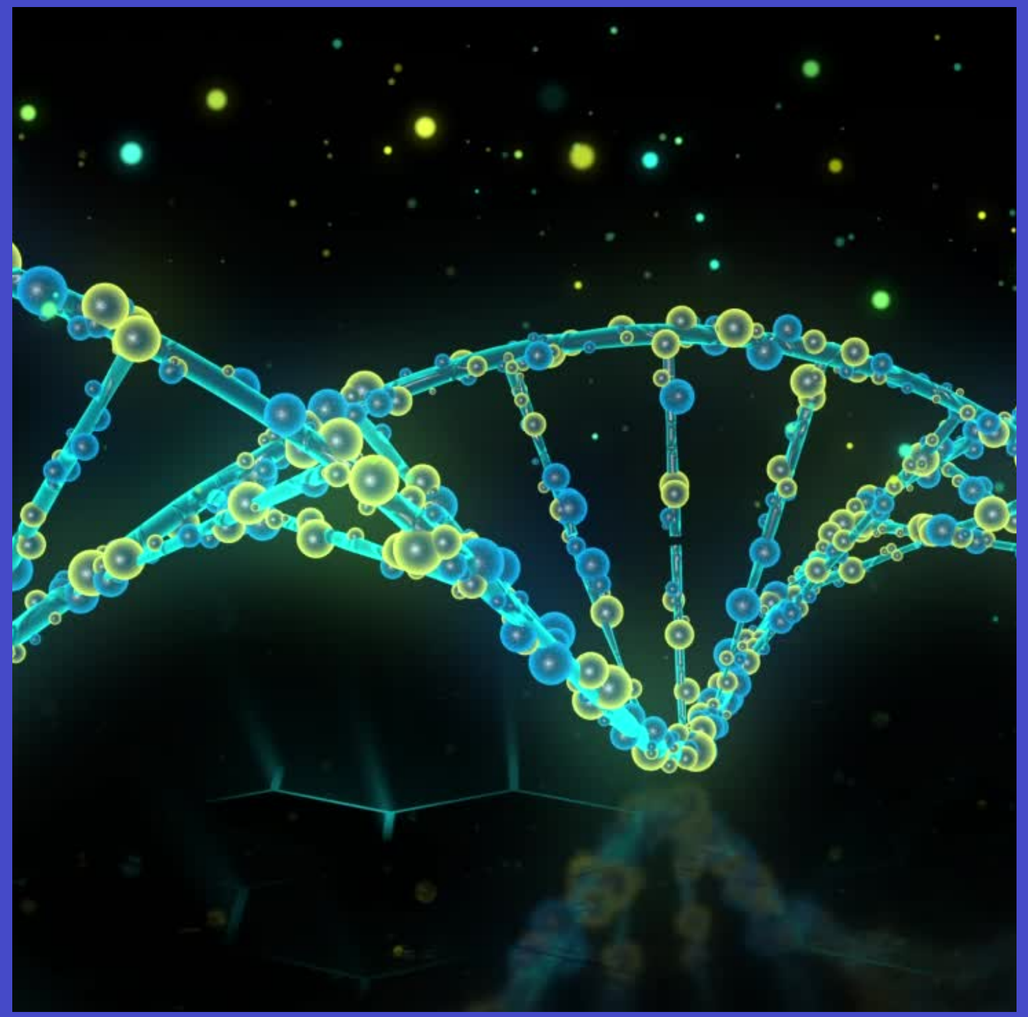
Our condensed findings are...

Outcomes

- Accurately predict the presence of heart disease
- Can be used as a prediction model for heart disease
- Potentially save lives and/or reduce healthcare costs
- The insights from our model allows researchers to develop better strategies for prevention and treatment

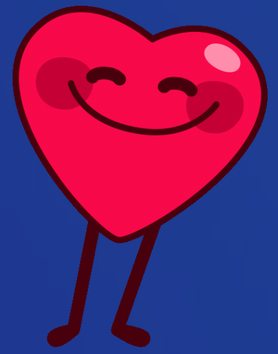
Recommendations

- Dataset collected in 1988, might be dated
- Medical advancements and adoption of healthier lifestyles since then may have strengthened resilience against heart disease
- However, human biological factors are relatively stable over time; more stagnant than human behaviour
- Data is still reliable



Recommendations

- Data does not account for lifestyle practices or habits
- These factors play a part in the development of heart disease
- More well-rounded and effective if these are present during prediction



THANK YOU!