DSC680 Project1: Heart Health Prediction

Assignment 4.1

Name: Madhuri Basava

Date: 06/28/2023

In [1]:
```
# Import the necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Set the style of matplotlib
%matplotlib inline
plt.style.use('fivethirtyeight')
```

In [2]:
```
# Load the heart health dataset into the data frame
heart_health_df = pd.read_csv('heart_statlog_cleveland_hungary_final.csv')
heart_health_df
```

Out[2]:

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	ST slope	targ
0	40	1	2	140	289	0	0	172	0	0.0	1	
1	49	0	3	160	180	0	0	156	0	1.0	2	
2	37	1	2	130	283	0	1	98	0	0.0	1	
3	48	0	4	138	214	0	0	108	1	1.5	2	
4	54	1	3	150	195	0	0	122	0	0.0	1	
...	...	...	...	...	...	...	...	...	...	...	...	
1185	45	1	1	110	264	0	0	132	0	1.2	2	
1186	68	1	4	144	193	1	0	141	0	3.4	2	
1187	57	1	4	130	131	0	0	115	1	1.2	2	
1188	57	0	2	130	236	0	2	174	0	0.0	2	
1189	38	1	3	138	175	0	0	173	0	0.0	1	

1190 rows × 12 columns

In [3]:

```
# Describe the dataset
heart_health_df.describe()
```

Out[3]:

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg
count	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000	1190.000000
mean	53.720168	0.763866	3.232773	132.153782	210.363866	0.213445	0.698319
std	9.358203	0.424884	0.935480	18.368823	101.420489	0.409912	0.870359
min	28.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
25%	47.000000	1.000000	3.000000	120.000000	188.000000	0.000000	0.000000
50%	54.000000	1.000000	4.000000	130.000000	229.000000	0.000000	0.000000
75%	60.000000	1.000000	4.000000	140.000000	269.750000	0.000000	2.000000
max	77.000000	1.000000	4.000000	200.000000	603.000000	1.000000	2.000000

In [4]:

```
# Display the information to understand the dataset
heart_health_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1190 entries, 0 to 1189
Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   age                  1190 non-null   int64  
 1   sex                  1190 non-null   int64  
 2   chest pain type      1190 non-null   int64  
 3   resting bp s         1190 non-null   int64  
 4   cholesterol          1190 non-null   int64  
 5   fasting blood sugar  1190 non-null   int64  
 6   resting ecg          1190 non-null   int64  
 7   max heart rate       1190 non-null   int64  
 8   exercise angina      1190 non-null   int64  
 9   oldpeak              1190 non-null   float64
 10  ST slope             1190 non-null   int64  
 11  target               1190 non-null   int64  
dtypes: float64(1), int64(11)
memory usage: 111.7 KB
```

In [5]:

```
# Check for any missing values
heart_health_df.isna().sum()
```

Out[5]:

```
age                    0
sex                    0
chest pain type        0
resting bp s           0
cholesterol            0
fasting blood sugar    0
resting ecg            0
max heart rate         0
exercise angina        0
oldpeak                0
ST slope               0
target                 0
dtype: int64
```

There are no missing values in the dataset

In [6]:

```
# Check if there are duplicate rows in the data set
heart_health_df.duplicated().sum()
```

Out[6]:

```
272
```

There are 272 duplicates in the dataset. So we remove them.

In [7]:

```
# Removing the duplicate rows

heart_health_df.drop_duplicates(inplace=True)
heart_health_df.duplicated().sum()
```

Out[7]:

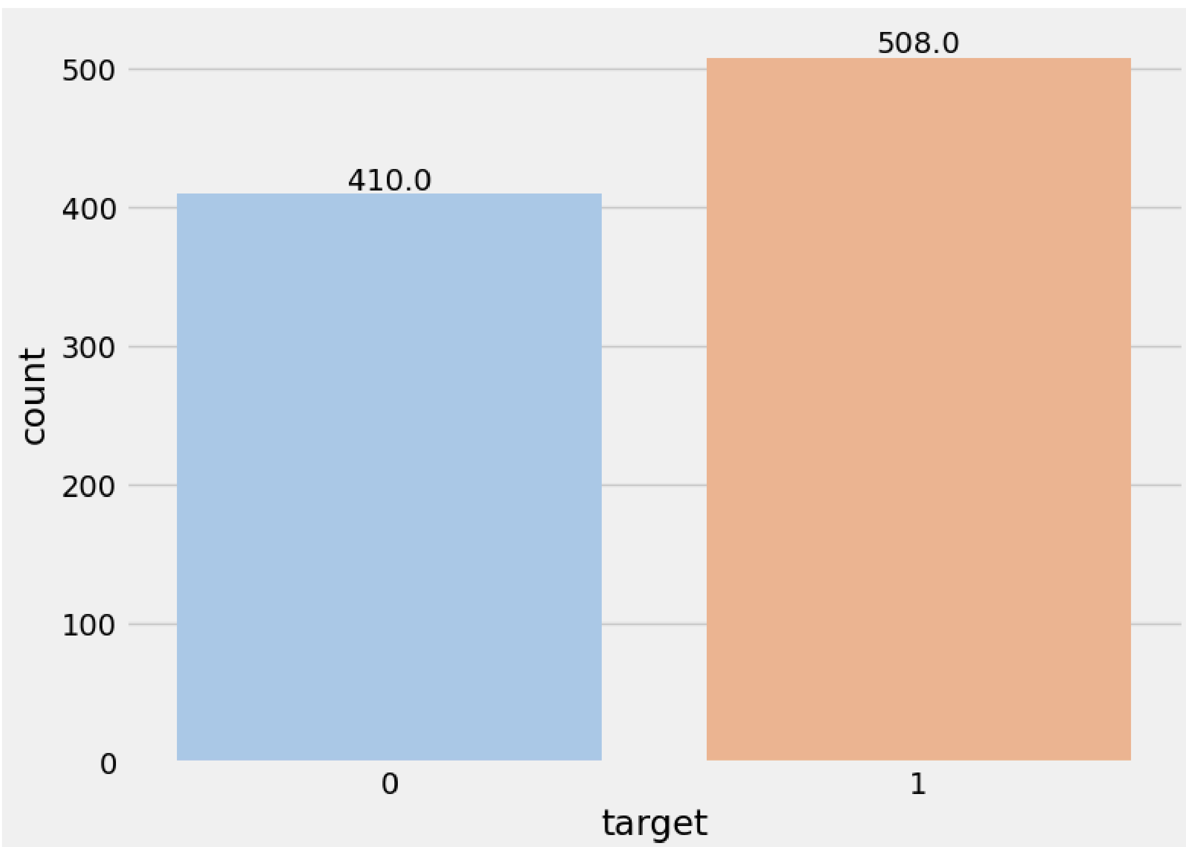
```
0
```

Exploratory Data Analysis

In [8]:
```python
# Print the number of unique values for each column

for col in heart_health_df.columns:
    print(f'{col} has {heart_health_df[col].nunique()} values')
```

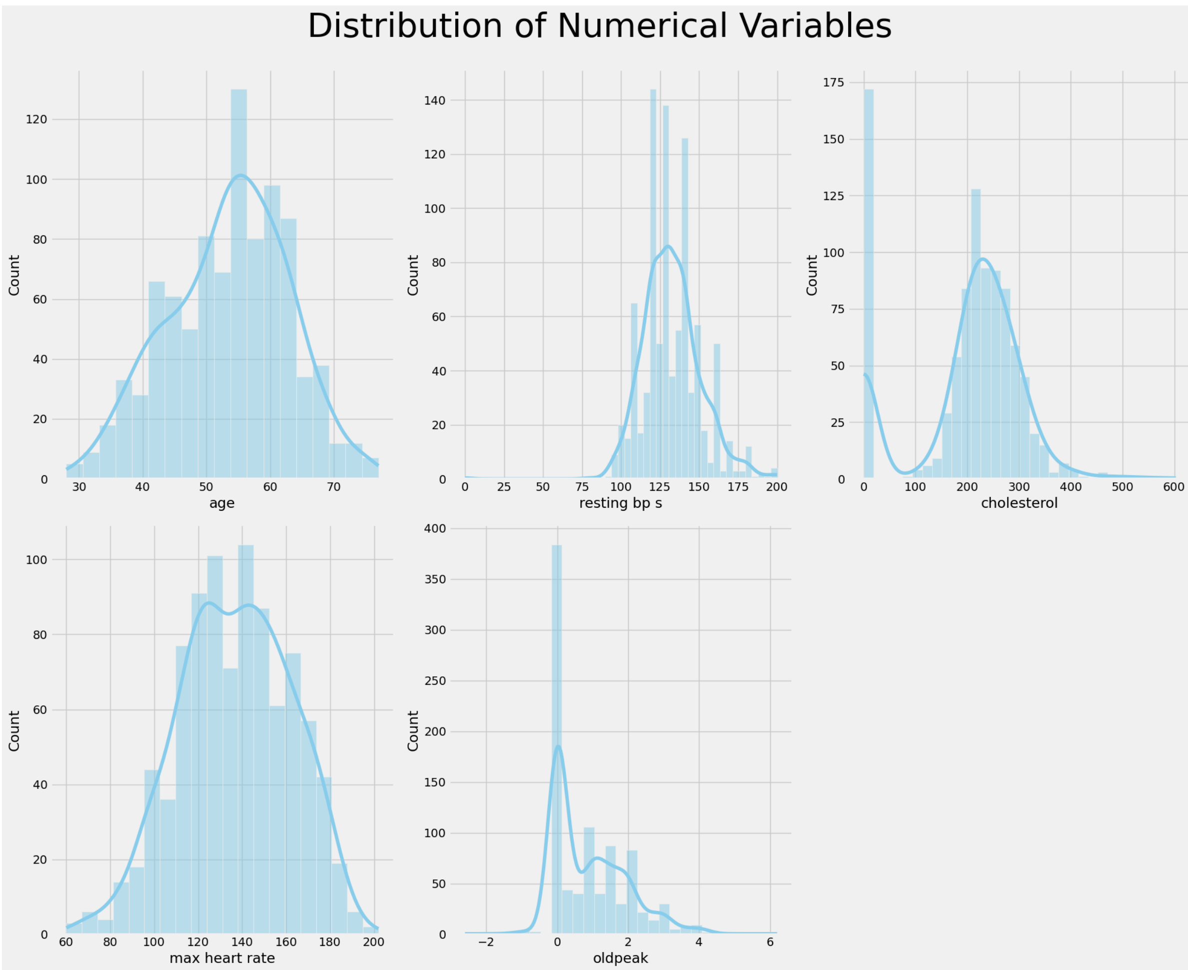
```
age has 50 values
sex has 2 values
chest pain type has 4 values
resting bp s has 67 values
cholesterol has 222 values
fasting blood sugar has 2 values
resting ecg has 3 values
max heart rate has 119 values
exercise angina has 2 values
oldpeak has 53 values
ST slope has 4 values
target has 2 values
```

In [9]:
```python
# Target distribution

# Set the figure size and create a count plot
plt.figure(figsize=(8, 6))
ax = sns.countplot(x=heart_health_df['target'], palette='pastel')

# Add labels to each bar in the plot
for p in ax.patches:
    ax.text(p.get_x()+p.get_width()/2, p.get_height()+3, f'{p.get_height()}', ha="c

plt.show()
```



From the above graph, we can infer that the dataset in balanced.

In [10]:
```
# Distribution of numerical columns

plt.figure(figsize=(20, 16))
plotnumber = 1

for col in heart_health_df.columns:
    # Check if the number of unique values is less than 5
    if heart_health_df[col].nunique() > 5:
        plt.subplot(2, 3, plotnumber)
        sns.histplot(heart_health_df[col], kde=True, color='skyblue')
        plt.xlabel(col)

        plotnumber += 1

plt.suptitle('Distribution of Numerical Variables', fontsize=40, y=1)
plt.tight_layout()
plt.show()
```



From the above graphs, we can infer that the cholesterol has zero values which does not makes sense.So we will remove those rows.

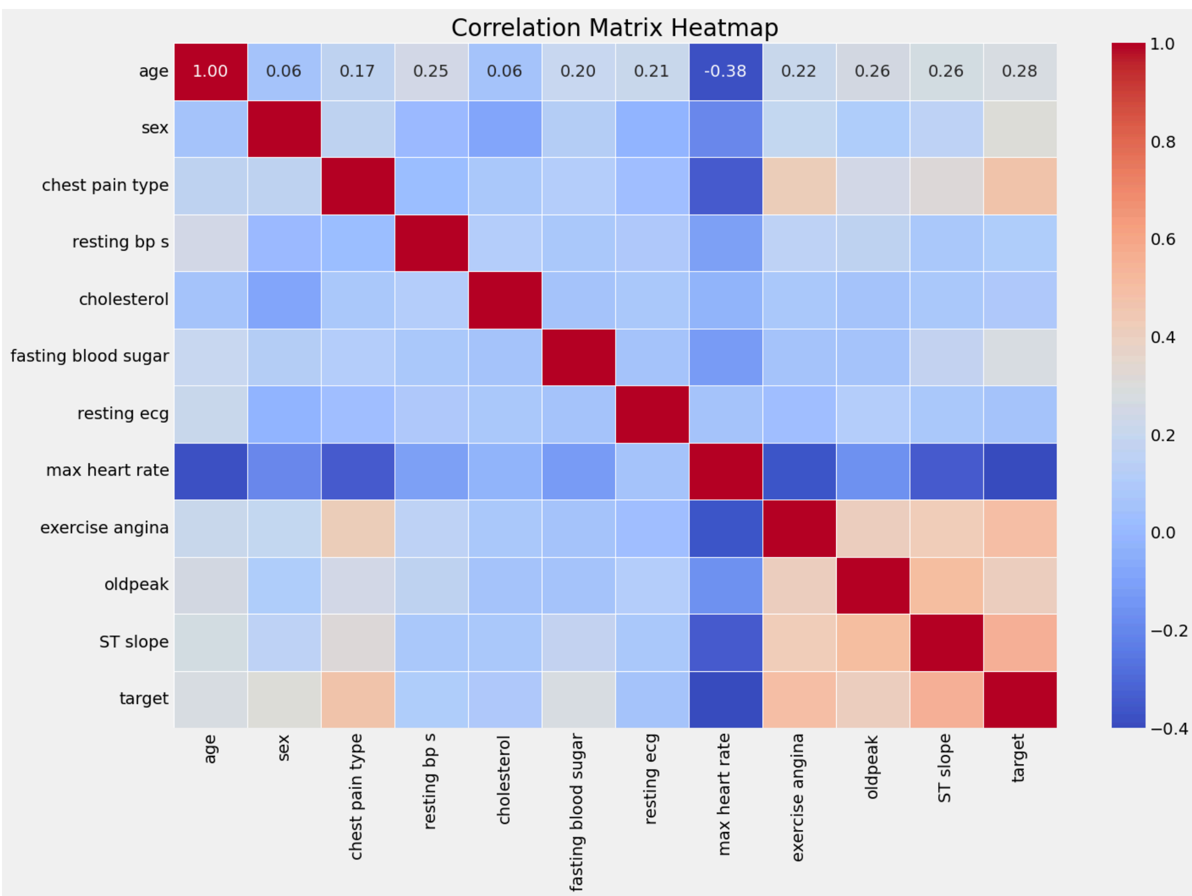
In [11]:
```
heart_health_df['cholesterol'] = heart_health_df['cholesterol'].replace(0, np.nan)
heart_health_df['cholesterol'].isnull().sum()
```

Out[11]: 172

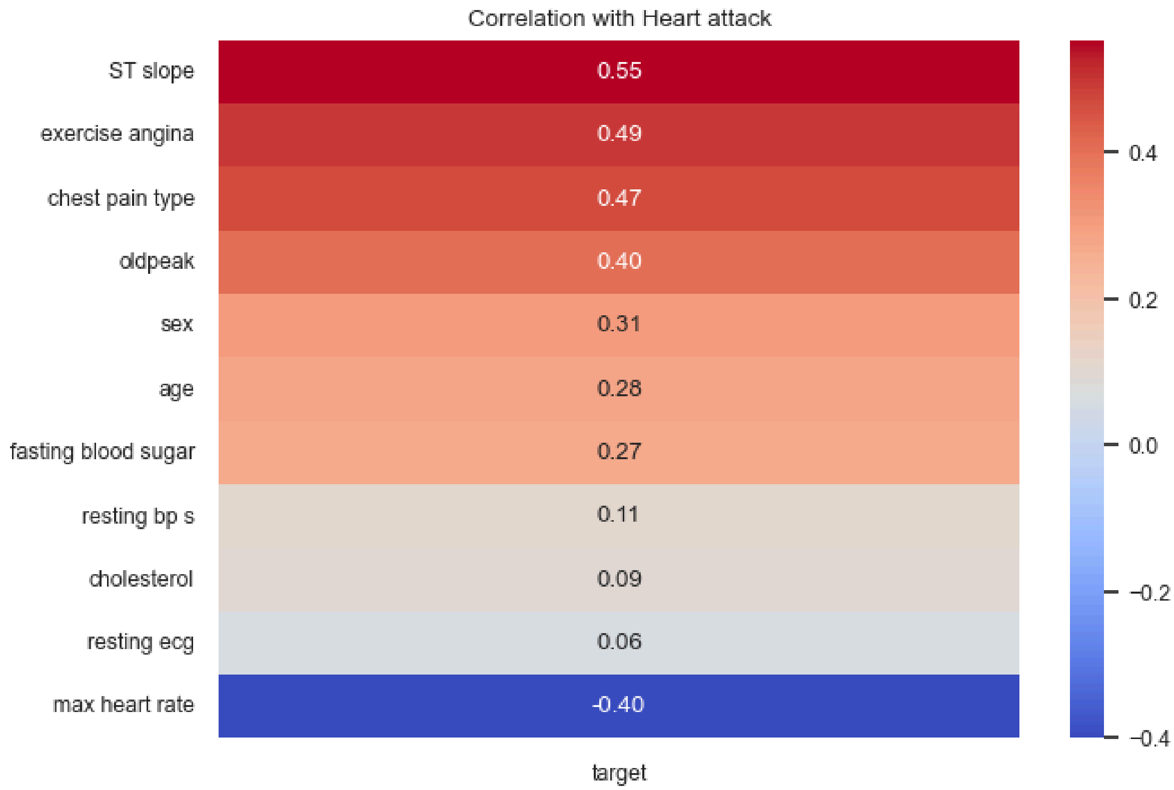
In [12]:
```
from sklearn.impute import KNNImputer

# Initialize the KNN imputer
knn_imputer = KNNImputer(n_neighbors=5)
# Impute missing values using the KNN method
heart_health_df = pd.DataFrame(knn_imputer.fit_transform(heart_health_df), columns=

heart_health_df['cholesterol'].isnull().sum()
```

Out[12]: 0

In [13]:
```
# Correlation matrix
#Graph I.

correlation_matrix = heart_health_df.corr()
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5, fmt='.
plt.title("Correlation Matrix Heatmap")
plt.show()

corr = heart_health_df.corr()
target_corr = corr['target'].drop('target')

# Sort correlation values in descending order
target_corr_sorted = target_corr.sort_values(ascending=False)
```



In [14]:
```
#Graph II
# Create a heatmap of the correlations with the target column
sns.set(font_scale=0.8)
sns.set_style("white")
sns.set_palette("PuBuGn_d")
sns.heatmap(target_corr_sorted.to_frame(), cmap="coolwarm", annot=True, fmt='.2f')
plt.title('Correlation with Heart attack')
plt.show()
```



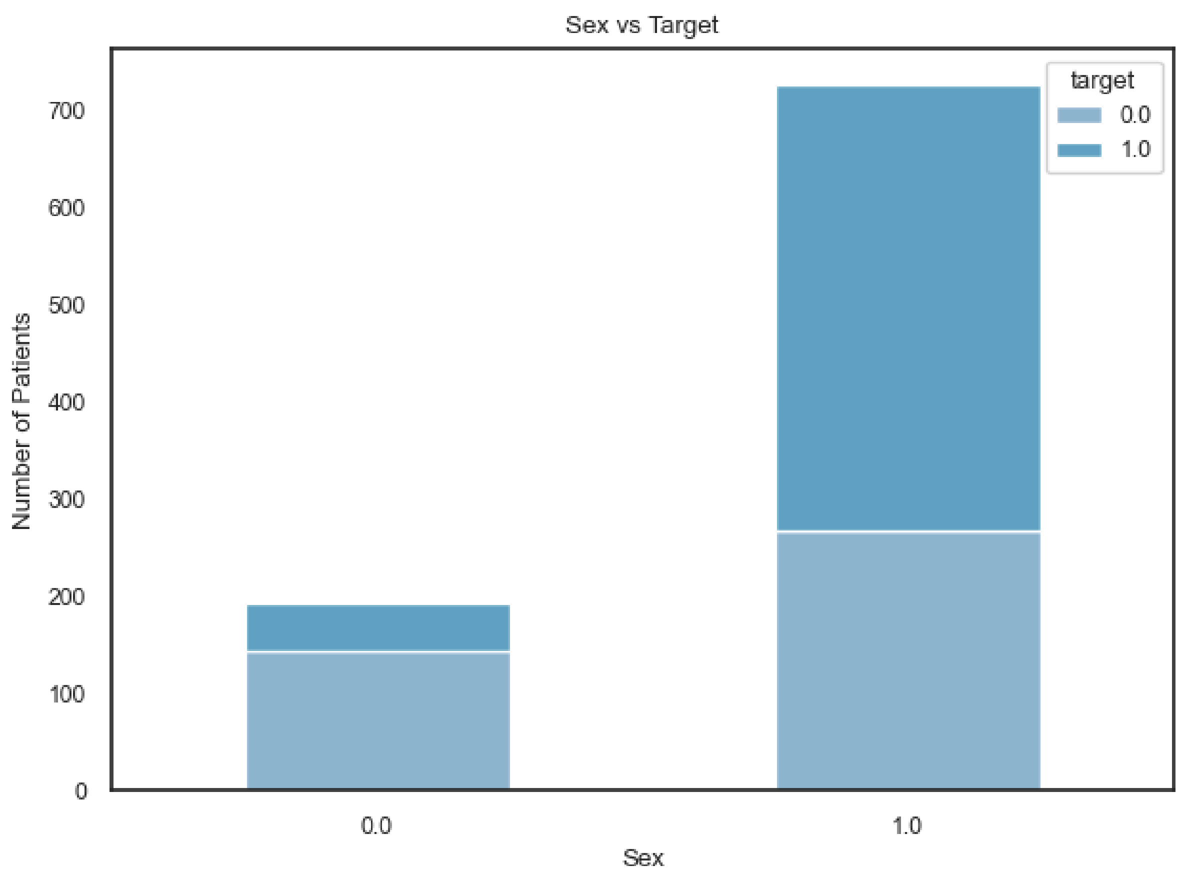
Intepret the Results

Correlations

1. Downsloping Peak Exercise ST Segment (slp_downsloping, 0.55): The presence of a downsloping peak exercise ST segment in an ECG report is associated with a higher probability of heart disease. This feature might be indicative of ischemia, which is reduced blood flow to the heart.
2. Exercise-Induced Angina (exng, 0.49): Exercise-induced angina is associated with a lower likelihood of heart disease. Similar to typical angina, individuals experiencing chest pain during exercise are more likely to seek early medical intervention, reducing the risk of advanced heart disease.
3. Non-anginal Chest Pain (cp_non-anginal pain, 0.47): Non-anginal chest pain also shows a significant positive correlation with heart disease. This type of chest pain is often mistaken for indigestion or muscle pain, possibly leading to delayed diagnosis and treatment.

4. ST Depression Induced by Exercise Relative to Rest (oldpeak, 0.40): ST depression induced by exercise, a sign of possible heart stress, shows a negative correlation with heart disease. This could suggest effective treatment and management of patients with this symptom, decreasing the likelihood of severe heart disease.
5. Sex (sex_male, 0.31): Males in this dataset are more likely to have heart disease compared to Females.
6. Maximum Heart Rate Achieved (-0.40): A high maximum heart rate achieved during testing is associated with a higher likelihood of heart disease. A high heart rate during exercise could reflect an underlying stress on the heart, which might indicate some form of cardiovascular disease.

In [15]:
```
sex_target = heart_health_df.groupby(['sex', 'target']).size().unstack().fillna(0)
sex_target.plot(kind='bar', stacked=True)
plt.title('Sex vs Target')
plt.xlabel('Sex')
plt.ylabel('Number of Patients')
plt.xticks(rotation=0)
plt.show()
```



In the above graph, 1-Male and 0-Female. Men are generally at higher risk of heart disease than women. However, after menopause, a woman's risk increases to almost match that of a man's.

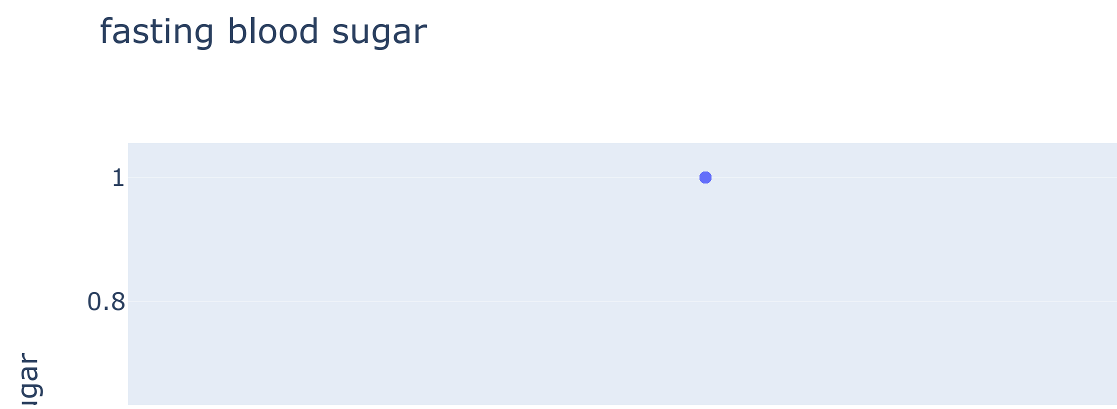
In [16]:

```
# Box plot for cholesterol and target
fig3 = px.box(heart_health_df, x='target', y='cholesterol', title='cholesterol')
fig3.show()
```

High levels of (LDL low-density lipoprotein)cholesterol are associated with an increased risk of heart disease, while high levels of HDL (high-density lipoprotein) cholesterol are protective. Cholesterol can build up in the walls of arteries, leading to atherosclerosis.

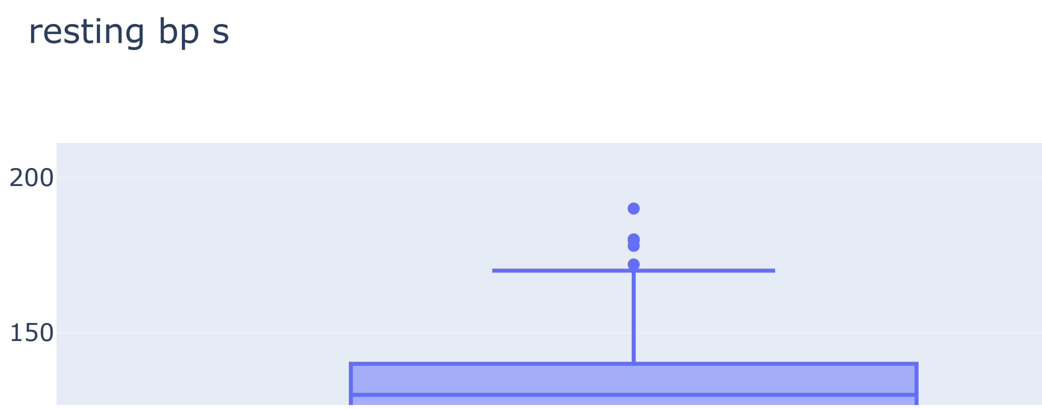
In [17]:

```
# Box plot for fasting blood sugar and target
fig4 = px.box(heart_health_df, x='target', y='fasting blood sugar', title='fasting
fig4.show()
```



High fasting blood sugar levels (prediabetes or diabetes) can contribute to narrowing of the arteries and increase the risk of heart disease. A fasting blood sugar level less than 100 mg/dL is considered normal. 100-125 mg/dL is considered prediabetes, and 126 mg/dL or higher on two separate tests means you have diabetes.

In [18]:
```
# Box plot for resting bp s and target
fig5 = px.box(heart_health_df, x='target', y='resting bp s', title='resting bp s')
fig5.show()
```



Hypertension (high blood pressure) damages the arteries and makes them more susceptible to plaque buildup, increasing the risk of heart disease and stroke.

Feature Engineering

In [19]:
```
# Create descriptive statistical features
# Basic statistics: Providing the model with simple statistical descriptions of the
heart_health_df['sum'] = heart_health_df[heart_health_df.columns].sum(axis=1)   # S
heart_health_df['std'] = heart_health_df[heart_health_df.columns].std(axis=1)   # S
heart_health_df['mean'] = heart_health_df[heart_health_df.columns].mean(axis=1) # M
heart_health_df['max'] = heart_health_df[heart_health_df.columns].max(axis=1)   # M
heart_health_df['min'] = heart_health_df[heart_health_df.columns].min(axis=1)   # M
heart_health_df['mode'] = heart_health_df[heart_health_df.columns].mode(axis=1)[0]
heart_health_df['median'] = heart_health_df[heart_health_df.columns].median(axis=1)
heart_health_df['q_25th'] = heart_health_df[heart_health_df.columns].quantile(0.25,
heart_health_df['q_75th'] = heart_health_df[heart_health_df.columns].quantile(0.75,
heart_health_df['skew'] = heart_health_df[heart_health_df.columns].skew(axis=1) # S
heart_health_df['kurt'] = heart_health_df[heart_health_df.columns].kurt(axis=1) # K
heart_health_df['range'] = heart_health_df[heart_health_df.columns].max(axis=1) - h

heart_health_df
```

Out[19]:

	age	sex	chest pain type	resting bp s	cholesterol	fasting blood sugar	resting ecg	max heart rate	exercise angina	oldpeak	...	me
0	40.0	1.0	2.0	140.0	289.0	0.0	0.0	172.0	0.0	0.0	...	105.5429
1	49.0	0.0	3.0	160.0	180.0	0.0	0.0	156.0	0.0	1.0	...	90.0697
2	37.0	1.0	2.0	130.0	283.0	0.0	1.0	98.0	0.0	0.0	...	90.6584
3	48.0	0.0	4.0	138.0	214.0	0.0	0.0	108.0	1.0	1.5	...	84.5278
4	54.0	1.0	3.0	150.0	195.0	0.0	0.0	122.0	0.0	0.0	...	85.8547
...	...	...	...	...	...	...	...	...	...	...	...	
913	45.0	1.0	1.0	110.0	264.0	0.0	0.0	132.0	0.0	1.2	...	91.1946
914	68.0	1.0	4.0	144.0	193.0	1.0	0.0	141.0	0.0	3.4	...	91.0349
915	57.0	1.0	4.0	130.0	131.0	0.0	0.0	115.0	1.0	1.2	...	72.2201
916	57.0	0.0	2.0	130.0	236.0	0.0	2.0	174.0	0.0	0.0	...	98.6351
917	38.0	1.0	3.0	138.0	175.0	0.0	0.0	173.0	0.0	0.0	...	86.3803

918 rows × 24 columns

Modelling

In [20]:

```
# Prepare data
# Define independent variables
ind_col = [col for col in heart_health_df.columns if col!='target']
# Define dependent variable
dep_col = 'target'

X = heart_health_df[ind_col]
y = heart_health_df[dep_col]

# For later use in feature importance plotting
dataframe = heart_health_df[ind_col]
```

In [21]:

```
from sklearn.preprocessing import StandardScaler

# Features features
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [22]:

```
from sklearn.model_selection import train_test_split

# Divide the data set into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

In [23]:

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```python
def evaluate_model(classifier, model_name, X_train, y_train, X_test, y_test):
    """
    Train, predict, and evaluate a classifier.

    Parameters:
        classifier: The machine learning classifier to train and evaluate.
        model_name: A string representing the name of the model for display.
        X_train, y_train: Training data and labels.
        X_test, y_test: Testing data and labels.

    Returns:
        A dictionary with confusion matrix, accuracy, precision, recall, and F1 sco
    """
    # Train the classifier
    classifier.fit(X_train, y_train)

    # Make predictions
    y_train_pred = classifier.predict(X_train)
    y_test_pred = classifier.predict(X_test)

    # Evaluate the model
    train_accuracy = accuracy_score(y_train, y_train_pred)
    accuracy = accuracy_score(y_test, y_test_pred)
    precision = precision_score(y_test, y_test_pred, average='macro')
    recall = recall_score(y_test, y_test_pred, average='macro')
    f1 = f1_score(y_test, y_test_pred, average='macro')
    conf_matrix = confusion_matrix(y_test, y_test_pred)
    class_report = classification_report(y_test, y_test_pred)

    # Print the evaluation metrics
    print(f"Training Accuracy of {model_name}: {train_accuracy:.5f}\n")
    print(f"Confusion Matrix:\n{conf_matrix}\n")
    print(f"Test Accuracy of {model_name}: {accuracy:.5f}")
    print(f"Test Precision of {model_name}: {precision:.5f}")
    print(f"Test Recall of {model_name}: {recall:.5f}")
    print(f"Test F1 Score of {model_name}: {f1:.5f}\n")
    print(f"Classification Report:\n{class_report}")

    # Return the metrics as a dictionary
    return {
        "Model Name": model_name,
        "Training Accuracy": train_accuracy,
        "Accuracy": accuracy,
        "Precision": precision,
        "Recall": recall,
        "F1 Score": f1,
        "Confusion Matrix": conf_matrix,
        "Classification Report": class_report
    }
```

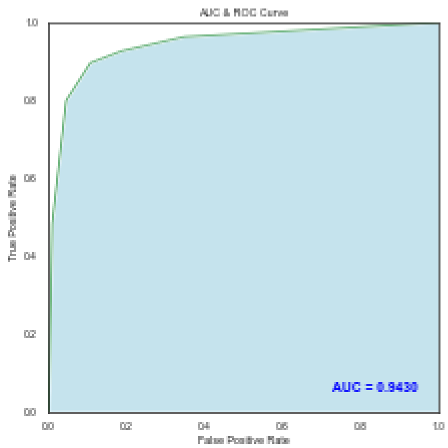
Here we are evaluating 3 models (K-nearest Neighbors,Naive Bayes model and Logistic regression)

1) K-Nearest Neighbors Model

In [24]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
model_name = "K-Nearest Neighbors"
knn_results = evaluate_model(knn, model_name, X_train, y_train, X_test, y_test)
```

```
Training Accuracy of K-Nearest Neighbors: 0.89097

Confusion Matrix:
[[100  12]
 [ 17 147]]

Test Accuracy of K-Nearest Neighbors: 0.89493
Test Precision of K-Nearest Neighbors: 0.88961
Test Recall of K-Nearest Neighbors: 0.89460
Test F1 Score of K-Nearest Neighbors: 0.89179

Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      0.89      0.87       112
         1.0       0.92      0.90      0.91       164

    accuracy                           0.89       276
   macro avg       0.89      0.89      0.89       276
weighted avg       0.90      0.89      0.90       276
```

In [25]:
```
from sklearn import metrics
from sklearn import metrics
y_pred = knn.predict_proba(X_test)[:, 1]
auc = metrics.roc_auc_score(y_test, y_pred)

false_positive_rate, true_positive_rate, thresolds = metrics.roc_curve(y_test, y_pr

plt.figure(figsize=(8, 6), dpi=40)
plt.rcParams["axes.grid"] = False
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(false_positive_rate, true_positive_rate, 'g')
plt.fill_between(false_positive_rate, true_positive_rate, facecolor='lightblue', al
plt.text(0.95, 0.05, 'AUC = %0.4f' % auc, ha='right', fontsize=12, weight='bold', c
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")

plt.show()
```



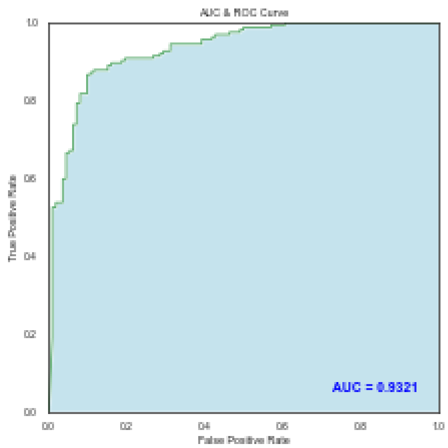
2) Naive Bayes Model

In [26]:
```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()
model_name = "Gaussian Naive Bayes"
gnb_results = evaluate_model(gnb, model_name, X_train, y_train, X_test, y_test)
```

```
Training Accuracy of Gaussian Naive Bayes: 0.84268

Confusion Matrix:
[[104   8]
 [ 34 130]]

Test Accuracy of Gaussian Naive Bayes: 0.84783
Test Precision of Gaussian Naive Bayes: 0.84783
Test Recall of Gaussian Naive Bayes: 0.86063
Test F1 Score of Gaussian Naive Bayes: 0.84646

Classification Report:
              precision    recall  f1-score   support

         0.0       0.75      0.93      0.83       112
         1.0       0.94      0.79      0.86       164

    accuracy                           0.85       276
   macro avg       0.85      0.86      0.85       276
weighted avg       0.87      0.85      0.85       276
```

In [27]:
```
y_pred = gnb.predict_proba(X_test)[:, 1]
auc = metrics.roc_auc_score(y_test, y_pred)

false_positive_rate, true_positive_rate, thresolds = metrics.roc_curve(y_test, y_pr

plt.figure(figsize=(8, 6), dpi=40)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(false_positive_rate, true_positive_rate, 'g')
```

```
plt.fill_between(false_positive_rate, true_positive_rate, facecolor='lightblue', al
plt.text(0.95, 0.05, 'AUC = %0.4f' % auc, ha='right', fontsize=12, weight='bold', c
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```



3) Logistic Regression

In [28]:
```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
model_name = "Logistic Regression"
logreg_results = evaluate_model(logreg, model_name, X_train, y_train, X_test, y_tes
```

```
Training Accuracy of Logistic Regression: 0.91589

Confusion Matrix:
[[107   5]
 [ 14 150]]

Test Accuracy of Logistic Regression: 0.93116
Test Precision of Logistic Regression: 0.92602
Test Recall of Logistic Regression: 0.93500
Test F1 Score of Logistic Regression: 0.92945

Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.96      0.92       112
         1.0       0.97      0.91      0.94       164

    accuracy                           0.93       276
   macro avg       0.93      0.93      0.93       276
weighted avg       0.93      0.93      0.93       276
```

In [29]:
```
y_pred = logreg.predict_proba(X_test)[:, 1]
auc = metrics.roc_auc_score(y_test, y_pred)

false_positive_rate, true_positive_rate, thresolds = metrics.roc_curve(y_test, y_pr
```

```python
plt.figure(figsize=(8, 6), dpi=40)
plt.axis('scaled')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.title("AUC & ROC Curve")
plt.plot(false_positive_rate, true_positive_rate, 'g')
plt.fill_between(false_positive_rate, true_positive_rate, facecolor='lightblue', al
plt.text(0.95, 0.05, 'AUC = %0.4f' % auc, ha='right', fontsize=12, weight='bold', c
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```