ITDAA Project – Heart Disease Prediction App

SHIVAN PRAMLALL

EDUV4945866@VOSSIE.NET

Table of Contents

Question 2 1

Question 3 4

Question 4 6

Link for Heart Disease Prediction App 9

Project code 10

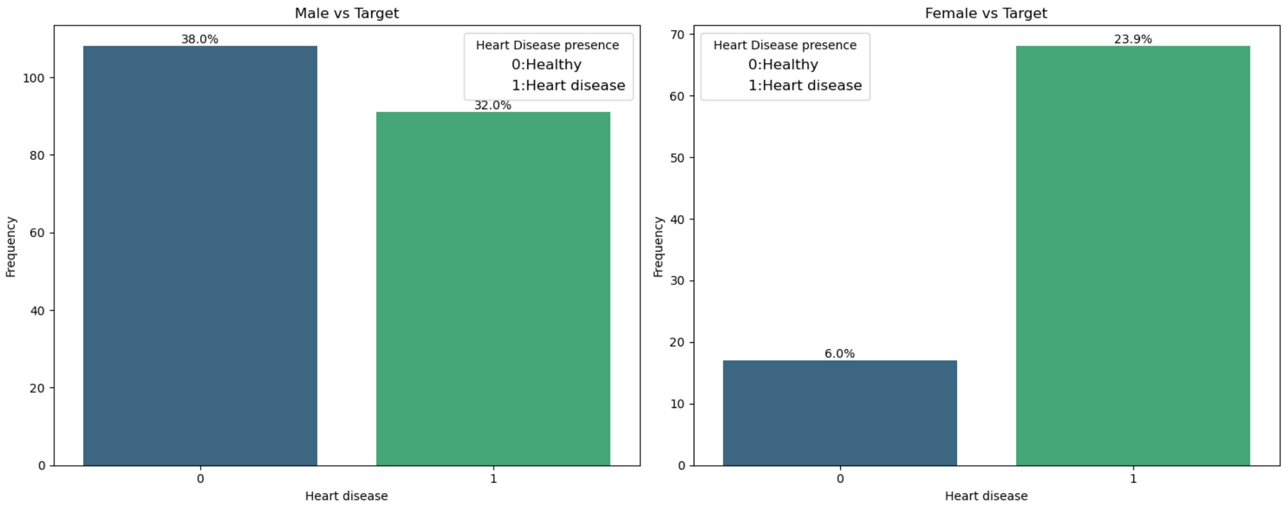
Streamlit web interface code 10

Project Code.. **Error! Bookmark not defined.**

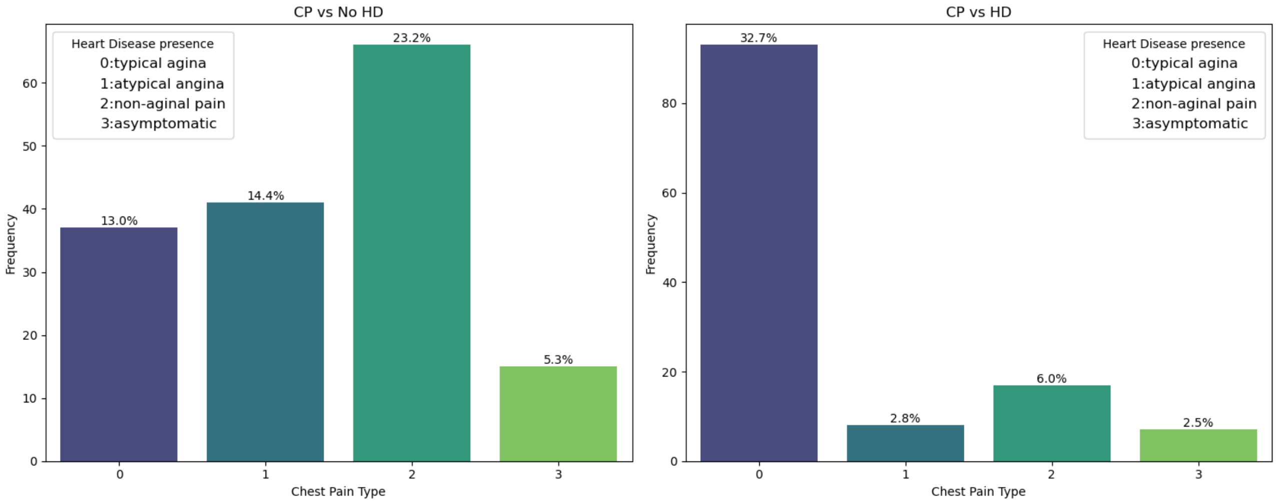
Question 2

2.1.b

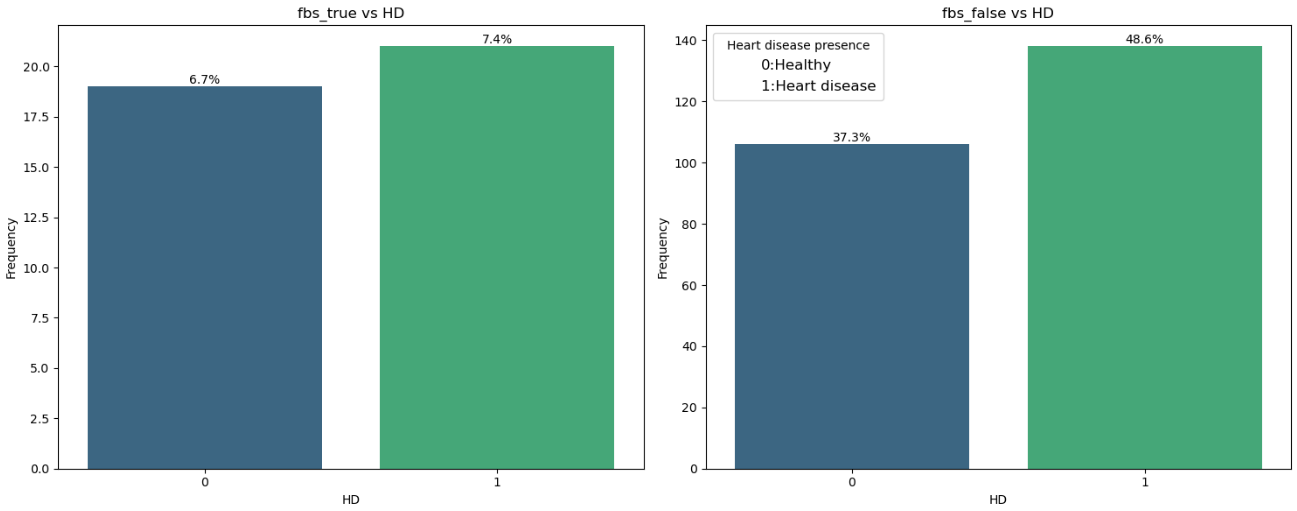
Sex vs Target



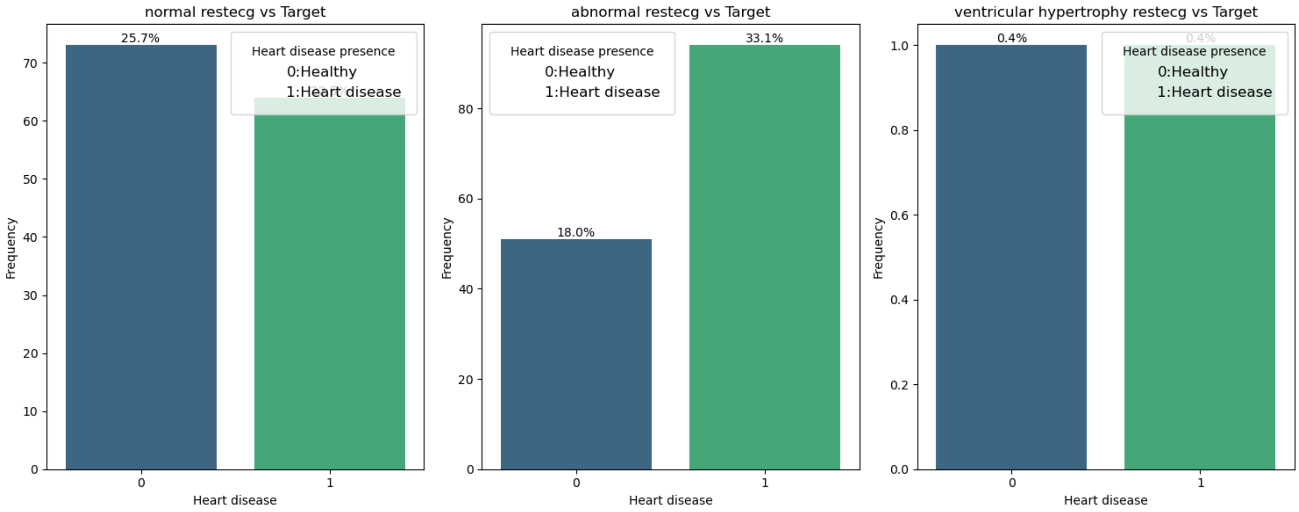
Cp vs Target



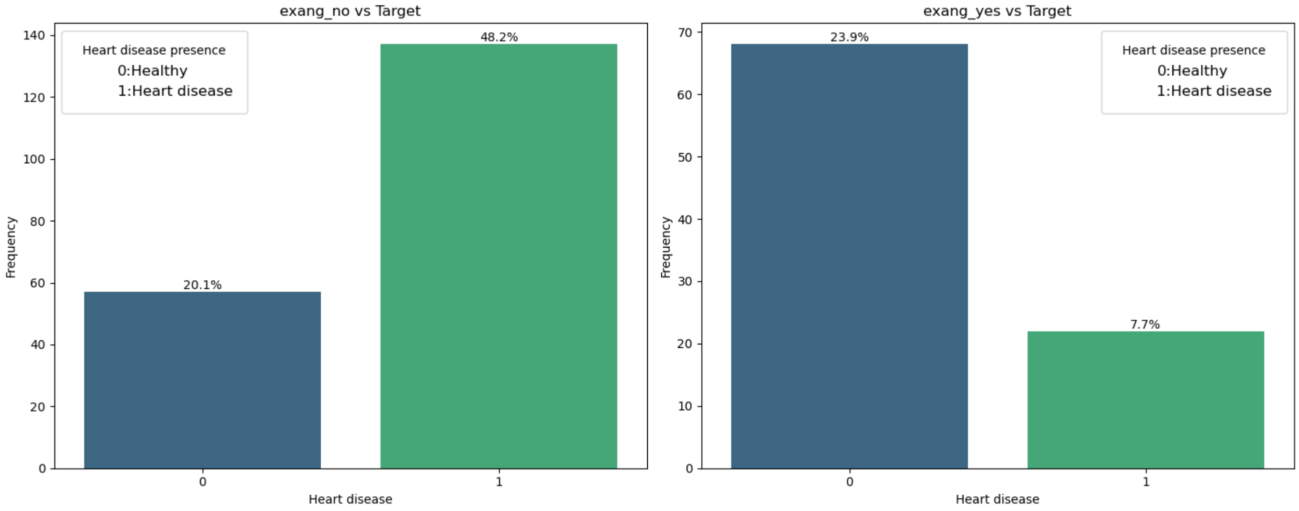
Fbs Vs Target



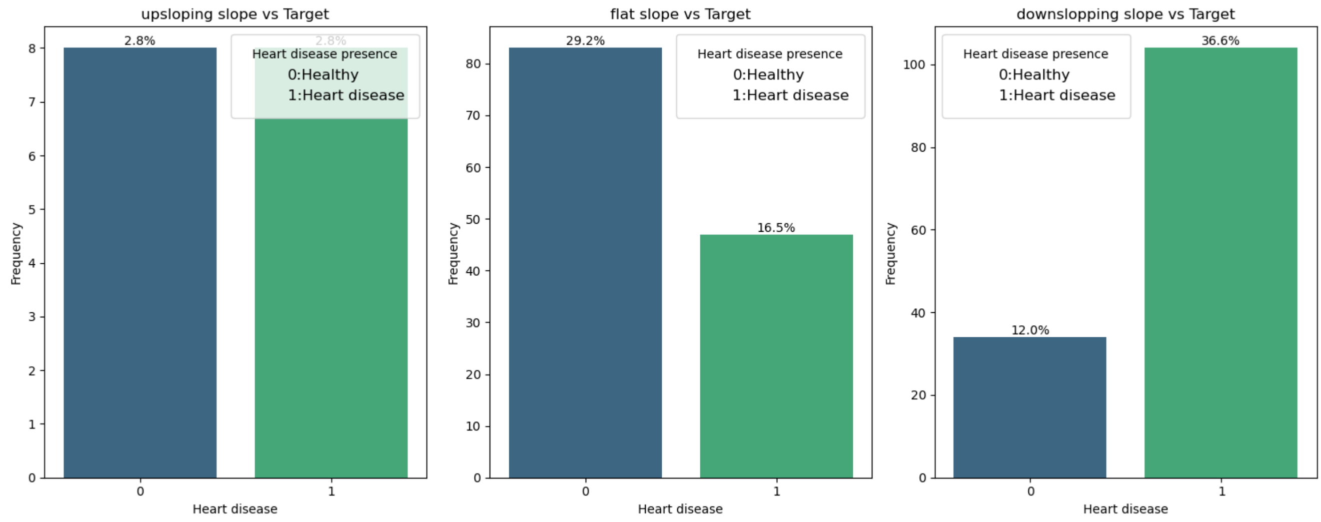
Restecg Vs Target



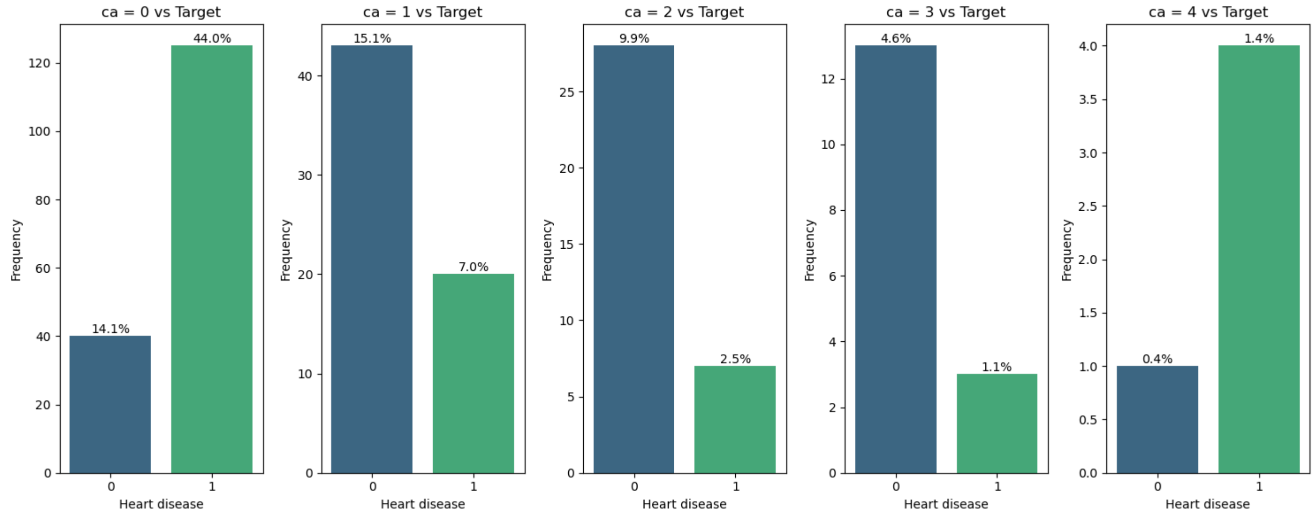
Exang Vs Target



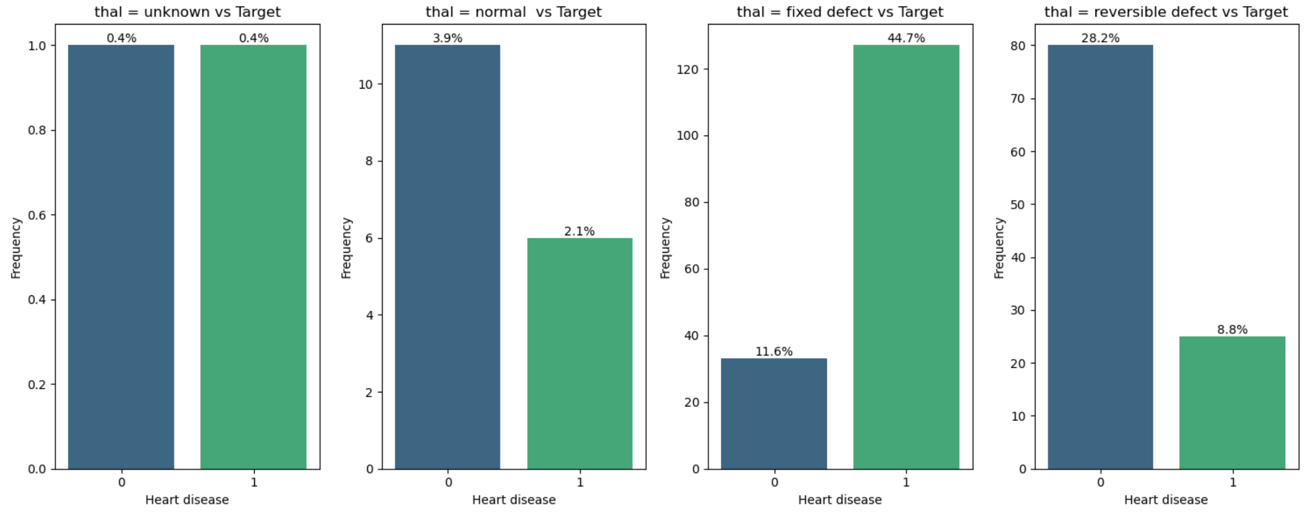
Slope Vs Target



Ca Vs Target

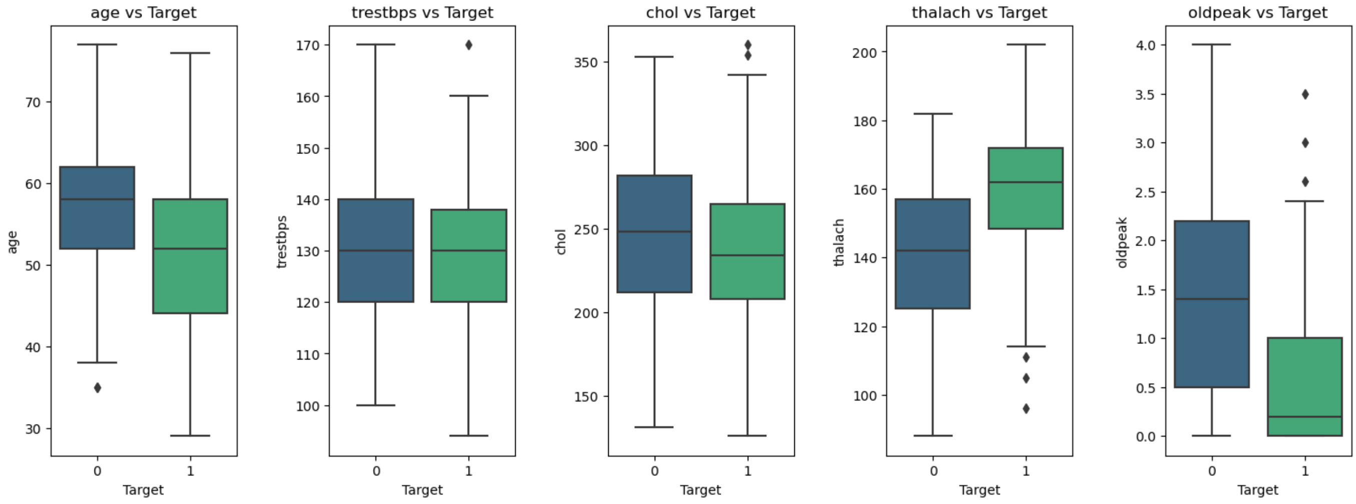


Thal Vs Target



2.1.c

Numerical variables Vs Target



Question 3

3.2

Logistic Regression

Logistic Regression is a statistical method used for binary classification. It models the probability that a given input belongs to a certain category.

Advantages:

- Logistic Regression performs well when the dataset is linearly separable.
- Logistic Regression not only gives a measure of how relevant a predictor (coefficient size) is, but also its direction of association (positive or negative).

Disadvantages:

- The main limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable. Most of the time data would be a jumbled mess.

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise it may lead to overfit.

Random Forest

Random Forest is an ensemble learning method that builds a multitude of decision trees at training time and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

Advantages:

- Random Forest algorithm is less prone to overfitting than Decision Tree and other algorithms.
- Random Forest algorithm outputs the importance of features which is a very useful.

Disadvantages:

- Random Forest algorithm may change considerably by a small change in the data.
- Random Forest algorithm computations may go far more complex compared to other algorithms.

Support Vector Machine (SVM)

SVM is a supervised learning model used for classification and regression tasks. It finds the hyperplane that best separates the classes in feature space.

Advantages:

- SVM is effective in cases where the number of dimensions is greater than the number of samples.
- SVM is relatively memory efficient.

Disadvantages:

- SVM algorithm is not suitable for large data sets.
- SVM does not perform very well when the data set has more noise i.e. target classes are overlapping.

Model performance.

Model	**Accuracy**
Logistic Regression	72%

Random Forest	74%
Support Vector Machines (SVM)	74%

Question 4

1. The Functionality of Your Application

Objectives: The Heart Disease Prediction App is a web interface that allows for doctors to input information about their patients and to find out if that patient has heart disease or not.

Implement Features: Doctors will input the following information about their clients:

- Age
- Sex
- Chest Pain Type
- Resting blood pressure (mm Hg)
- Serum cholestoral (mg/dl)
- Fasting blood sugar > 120 mg/dl
- Resting electrocardiographic results
- Max heart rate achieved
- Exercise included angina
- ST depression included by exercise relative to rest
- Slope of the peak exercise ST segment
- Number of major vessels coloured by fluoroscopy
- Status of the heart

The application uses a Random Forest model to predict whether a patient has heart disease or not.

Interactivity: The application makes use of Streamlit’s widgets like sliders, buttons, and number inputs.

2. The Usability

User Experience (UX): The application has been designed with the end-user in mind. It is intuitive and easy to navigate.

Instructions: Doctors can input the patient information in the sidebar on the left. The sidebar can be expanded/closed by clicking on the arrow on the top left corner on the screen.

Once all information has been selected, click on the “Submit” button. This will submit all inputs into the model.

Feedback: After the user has clicked on the “Submit” button, the app will output the prediction probability of whether a patient is health or has heart disease.

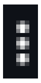
3. The Design of the Application

Layout: A clean, logical layout. User inputs all the info in the sidebar. This avoids the user to scroll back and forth to keep on changing patient information. The selected info along with prediction probability will be pasted, avoiding any scrolling which can be h hassle.

Visual Appeal: A consistent colour scheme, typography, and styling was used.

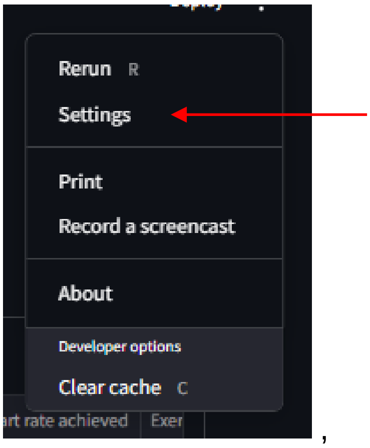
Users can change to their desired theme by:

1. clicking on the three dots on the top right corner,



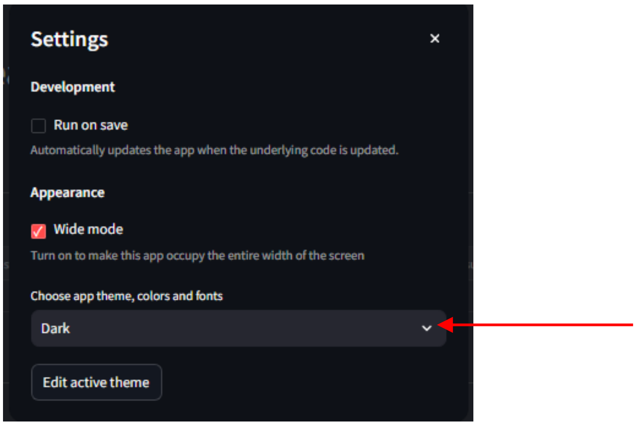
,

2. click on settings,



,

3. click on “Choose app themes, colours and fonts”.



Settings ×

Development

□ Run on save

Automatically updates the app when the underlying code is updated.

Appearance

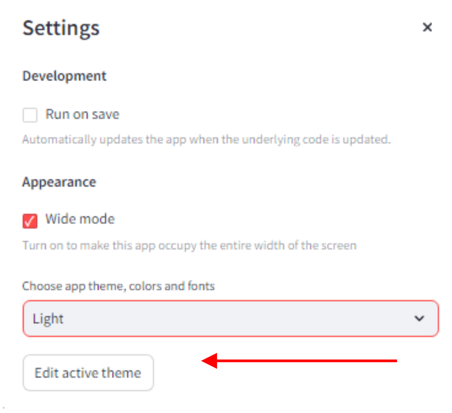
☑ Wide mode

Turn on to make this app occupy the entire width of the screen

Choose app theme, colors and fonts

Dark

4. click on the “Edit active theme” button.



Settings

Development

□ Run on save

Automatically updates the app when the underlying code is updated.

Appearance

☑ Wide mode

Turn on to make this app occupy the entire width of the screen

Choose app theme, colors and fonts

Light

4. The Code Design/Structure

Modular Code: The following functions were defined:

create_model(dataset) - creates random forest regression model.

predict(model,input_features - create model.

prediction_probablity(model,input_features - calculating prediction prediction_probablity of model.

submitted = form.form_submit_button("Submit") - "Submit" button that will submit all info selected to the model.

5. Your Code and Application Documentation

Comments: Comments are used to explain complex code segments.

6. Error-Handling

The application makes uses of Streamlit’s widgets like sliders, number inputs which have a minimum values option. Hence user cannot input negative values.

The application also uses radio buttons that will only allow the user to select the values in the radio button. This prevents user from making typo errors or choosing incorrect values if they were to type.

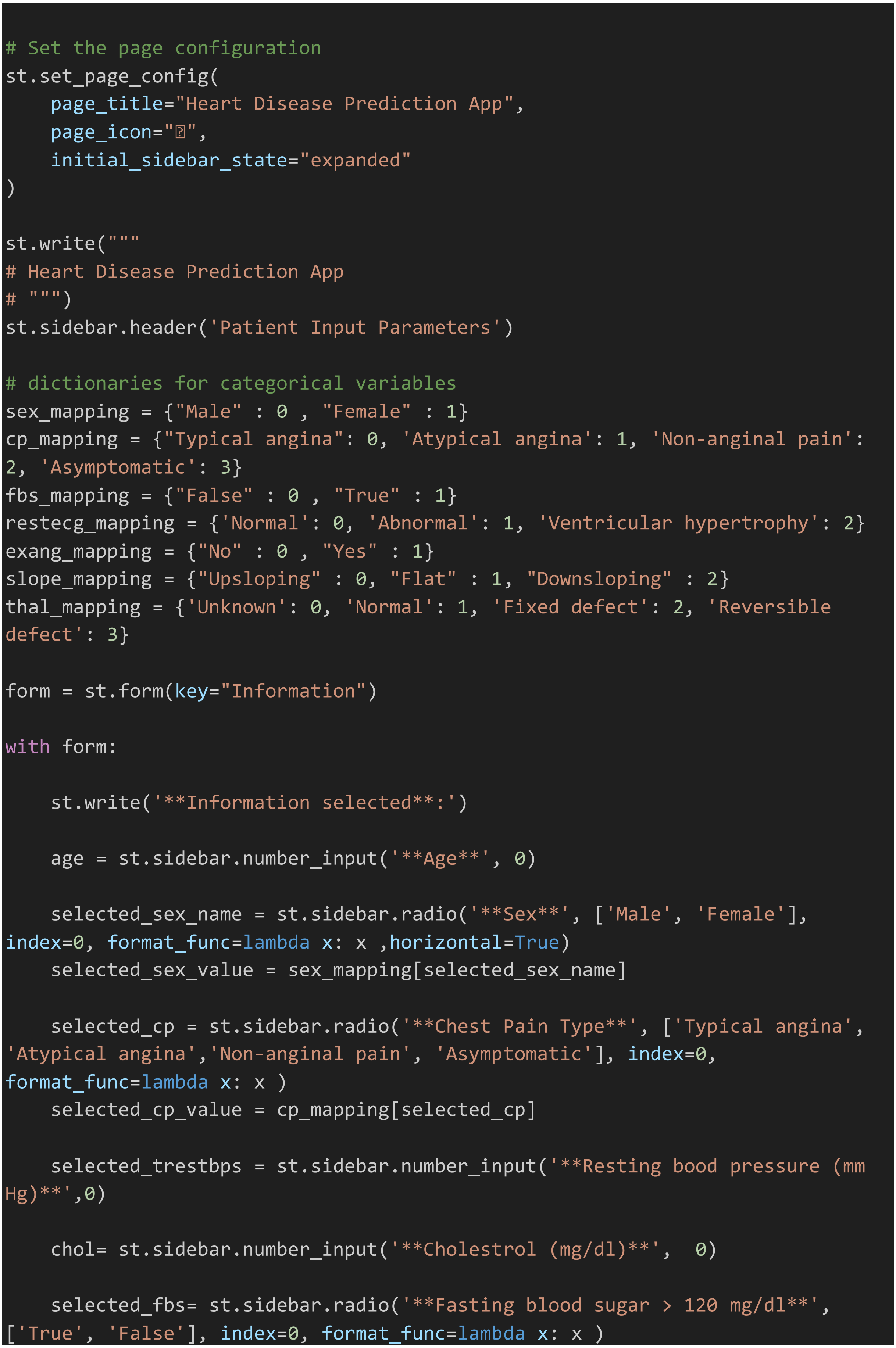
Link for Heart Disease Prediction App

Click the link to view the application.

Project code

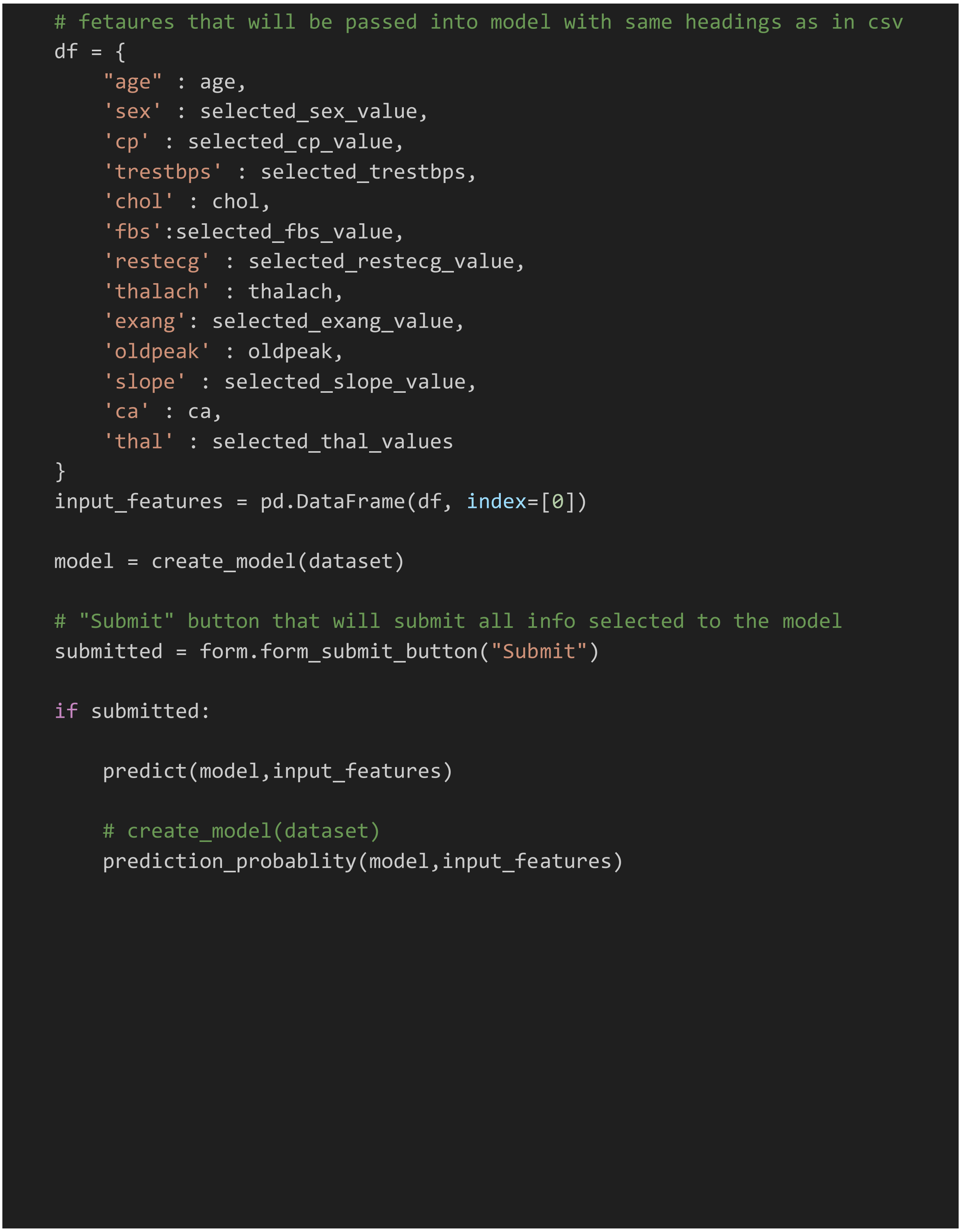
Streamlit web interface code

```
import pandas as pd
import streamlit as st
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

dataset =
pd.read_csv(r"https://raw.githubusercontent.com/ShivanPramlall/ITDAA-
Project/main/heart%20(1).csv",sep=";")

# creates logistics regression model
def create_model(dataset):

    # Remove outliers
    for i in [i for i in dataset.columns]:
        if dataset[i].nunique()>=12:
            Q1 = dataset[i].quantile(0.25)
            Q3 = dataset[i].quantile(0.75)
            IQR = Q3 - Q1
            dataset = dataset[dataset[i] <= (Q3+(1.5*IQR))]
            dataset = dataset[dataset[i] >= (Q1-(1.5*IQR))]

    X = dataset.iloc[:,:-1] # Using all column except for the last column as X
    y = dataset.iloc[:,-1] # Selecting the last column as Y

    # train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=16)

    # Train the Logistic Regression model
    model = RandomForestClassifier(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    return model

# create model
def predict(model,input_features):
    # Make predictions
    prediction = model.predict(input_features)

# calculating prediction prediction_probablity of model
def prediction_probablity(model,input_features):
    st.subheader('Prediction Probability')
    st.write('**0 : Patient is healthy \t 1 : Patient has heart disease**')
    prediction_probability = model.predict_proba(input_features)
    st.write(prediction_probability)
```



```
# Set the page configuration
st.set_page_config(
    page_title="Heart Disease Prediction App",
    page_icon="",
    initial_sidebar_state="expanded"
)

st.write("""
# Heart Disease Prediction App
# """)
st.sidebar.header('Patient Input Parameters')

# dictionaries for categorical variables
sex_mapping = {"Male" : 0 , "Female" : 1}
cp_mapping = {"Typical angina": 0, 'Atypical angina': 1, 'Non-anginal pain':
2, 'Asymptomatic': 3}
fbs_mapping = {"False" : 0 , "True" : 1}
restecg_mapping = {'Normal': 0, 'Abnormal': 1, 'Ventricular hypertrophy': 2}
exang_mapping = {"No" : 0 , "Yes" : 1}
slope_mapping = {"Upsloping" : 0, "Flat" : 1, "Downsloping" : 2}
thal_mapping = {'Unknown': 0, 'Normal': 1, 'Fixed defect': 2, 'Reversible
defect': 3}

form = st.form(key="Information")

with form:

    st.write('**Information selected**:')

    age = st.sidebar.number_input('**Age**', 0)

    selected_sex_name = st.sidebar.radio('**Sex**', ['Male', 'Female'],
index=0, format_func=lambda x: x ,horizontal=True)
    selected_sex_value = sex_mapping[selected_sex_name]

    selected_cp = st.sidebar.radio('**Chest Pain Type**', ['Typical angina',
'Atypical angina','Non-anginal pain', 'Asymptomatic'], index=0,
format_func=lambda x: x )
    selected_cp_value = cp_mapping[selected_cp]

    selected_trestbps = st.sidebar.number_input('**Resting bood pressure (mm
Hg)**',0)

    chol= st.sidebar.number_input('**Cholestrol (mg/dl)**',  0)

    selected_fbs= st.sidebar.radio('**Fasting blood sugar > 120 mg/dl**',
['True', 'False'], index=0, format_func=lambda x: x )
```

```
    selected_fbs_value = fbs_mapping[selected_fbs]

    selected_restecg = st.sidebar.radio('**Resting electrocardiographic
results**', ['Normal', 'Abnormal','Ventricular hypertrophy'], index=0,
format_func=lambda x: x)
    selected_restecg_value  = restecg_mapping[selected_restecg]

    thalach= st.sidebar.number_input('**Max heart rate achieved**',  0)

    selected_exang= st.sidebar.radio('**Exercise included angina**', ['Yes',
'No'], index=0, format_func=lambda x: x ,horizontal=True)
    selected_exang_value = exang_mapping[selected_exang]

    oldpeak= st.sidebar.number_input('**ST depression included by exercise
relative to rest**', 0)

    selected_slope= st.sidebar.radio('**Slope of the peak exercise ST
segment**', ['Upsloping', 'Flat', 'Downsloping'], index=0, format_func=lambda
x: x)
    selected_slope_value = slope_mapping[selected_slope]

    ca= st.sidebar.slider('**Number of major vessls coloured by
fluoroscopy**', 0,4)

    selected_thal= st.sidebar.radio('**Status of the heart**', ['Unknown',
'Normal', 'Fixed defect','Reversible defect'], index=0, format_func=lambda x:
x)
    selected_thal_values = thal_mapping[selected_thal]

    # features displyed in a dataframe
    data = {
        "Age" : age,
        'Sex' : selected_sex_value,
        'Chest Pain Type' : selected_cp_value,
        'Resting blood pressure (mm Hg)' : selected_trestbps,
        'Cholestoral (mg/dl)' : chol,
        'Fasting blood sugar > 120 mg/dl':selected_fbs_value,
        'Resting electrocardiographic results' : selected_restecg_value,
        'Max heart rate achieved' : thalach,
        'Exercise included angina': selected_exang_value,
        'ST depression included by exercise relative to rest' : oldpeak,
        'Slope of the peak exercise ST segment' : selected_slope_value,
        'Number of major vessls coloured by fluoroscopy' : ca,
        'Status of the heart' : selected_thal_values
    }
    features = pd.DataFrame(data, index=[0])
    st.write(features)
```



```
    # fetaures that will be passed into model with same headings as in csv
    df = {
        "age" : age,
        'sex' : selected_sex_value,
        'cp' : selected_cp_value,
        'trestbps' : selected_trestbps,
        'chol' : chol,
        'fbs':selected_fbs_value,
        'restecg' : selected_restecg_value,
        'thalach' : thalach,
        'exang': selected_exang_value,
        'oldpeak' : oldpeak,
        'slope' : selected_slope_value,
        'ca' : ca,
        'thal' : selected_thal_values
    }
    input_features = pd.DataFrame(df, index=[0])

    model = create_model(dataset)

    # "Submit" button that will submit all info selected to the model
    submitted = form.form_submit_button("Submit")

    if submitted:

        predict(model,input_features)

        # create_model(dataset)
        prediction_probablity(model,input_features)
```

project-code

June 24, 2024

[3]:

```
import pandas as pd
import sqlite3
import chardet
import seaborn as sns
import numpy as np
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
```

[4]:

```
# Read the CSV file into a pandas DataFrame
df = pd.read_csv(r'C:\Users\Shivan\OneDrive\Desktop\ITDAA4-12 - Data Mining and␣
 ↪Data Administration module\Project\heart (1).csv'
                ,sep=";")
df
```

[4]:

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   …  …   ..      …   …  …      …      …    …      …
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     …   ..  …     …
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
```

```
301       1    1      3        0
302       1    1      2        0

[303 rows x 14 columns]
```

[5]:
```
df2 = df.copy()

# Remove outliers
for i in [i for i in df2.columns]:
    if df2[i].nunique()>=12:
        Q1 = df2[i].quantile(0.25)
        Q3 = df2[i].quantile(0.75)
        IQR = Q3 - Q1
        df2 = df2[df2[i] <= (Q3+(1.5*IQR))]
        df2 = df2[df2[i] >= (Q1-(1.5*IQR))]

df2 = df2.reset_index(drop=True)

print('Before removal of outliers, The dataset had {} samples.'.format(df.
 ↪shape[0]))
print('After removal of outliers values, The dataset now has {} samples.'.
 ↪format(df2.shape[0]))
```

```
Before removal of outliers, The dataset had 303 samples.
After removal of outliers values, The dataset now has 284 samples.
```

[6]:
```
df = df2
df.shape
```

[6]:
```
(284, 14)
```

1 Question 1

[7]:
```
# Connect to SQLite database
conn = sqlite3.connect('Heart_Disease_Database.db')  # Creates or connects to a␣
 ↪database named database.db

# Upload DataFrame to SQLite database
df.to_sql('Heart disease', conn, if_exists='replace', index=False)

# Commit changes and close connection
conn.commit()
# conn.close()

print("CSV data has been successfully uploaded as a database .")
```

```
CSV data has been successfully uploaded as a database .
```

[8]:
```
conn = sqlite3.connect('Heart_Disease_Database.db')
```

2 Question 2

## 2.1.a

[17]:
```
# check for null values
df.isnull().sum()
```

[17]:
```
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
```

[9]:
```
# checking the unique values to ensure that there arent any foreign values, eg␣
 ↪a puncutation mark when expecting interger
# values

for col in df.columns:
    unique_values = df[col].unique()
    print(f"{col} : {unique_values}")
```

```
age : [63 37 41 56 57 44 54 48 49 64 58 50 66 43 69 59 42 61 40 71 51 53 65 46
 45 39 52 47 62 34 35 29 55 60 68 67 74 76 70 38 77]
sex : [1 0]
cp : [3 2 1 0]
trestbps : [145 130 120 140 150 110 135 160 105 125 142 155 104 138 128 108 134
122
 115 118 100 124  94 112 102 152 101 132 148 129 136 126 106 156 170 146
 117 165 144 123 154 164]
chol : [233 250 204 236 354 192 294 263 168 239 275 266 211 283 219 340 226 247
 234 243 199 302 212 175 197 198 177 273 213 304 232 269 360 308 245 208
 264 321 325 235 257 216 256 231 141 252 201 222 260 182 303 265 309 186
 203 183 220 209 258 227 261 221 205 240 318 298 277 214 248 255 207 223
 288 160 315 246 244 195 196 254 126 313 262 215 193 271 268 267 210 295
 306 178 242 180 228 149 278 253 342 157 286 229 284 224 206 167 230 335
 276 353 225 330 290 172 305 188 282 185 326 270 307 249 341 174 281 289
 322 299 300 293 274 184 259 200 218 319 166 311 169 187 176 241 131]
fbs : [1 0]
restecg : [0 1 2]
thalach : [150 187 172 178 163 148 153 173 174 160 139 171 144 162 158 114 151
161
 179 137 157 123 152 168 140 188 125 170 165 142 180 143 182 156 115 149
 146 175 186 185 159 130 190 132 147 154 202 166 164 184 122 169 138 111
 194 131 133 155 167 192 121  96 126 105 181 116 108 129 120 112 128 109
 113  99 177 141 136  97 127 103 124 145  88 106  95 118 134  90]
exang : [0 1]
oldpeak : [2.3 3.5 1.4 0.8 0.6 0.4 1.3 0.  1.6 1.2 0.2 1.8 1.  2.6 1.5 0.5 3.
2.4
 0.1 1.9 1.1 2.  0.7 0.3 0.9 3.6 3.1 3.2 2.5 2.2 2.8 3.4 2.9 2.1 3.8 4. ]
slope : [0 2 1]
ca : [0 2 1 3 4]
thal : [1 2 3 0]
target : [1 0]
```

2.1 2.2.b

2.1.1 categorical variables = [‘sex’, ‘cp’, ‘fbs’,‘restecg’, ‘exang’,‘slope’, ‘ca’, ‘thal’]

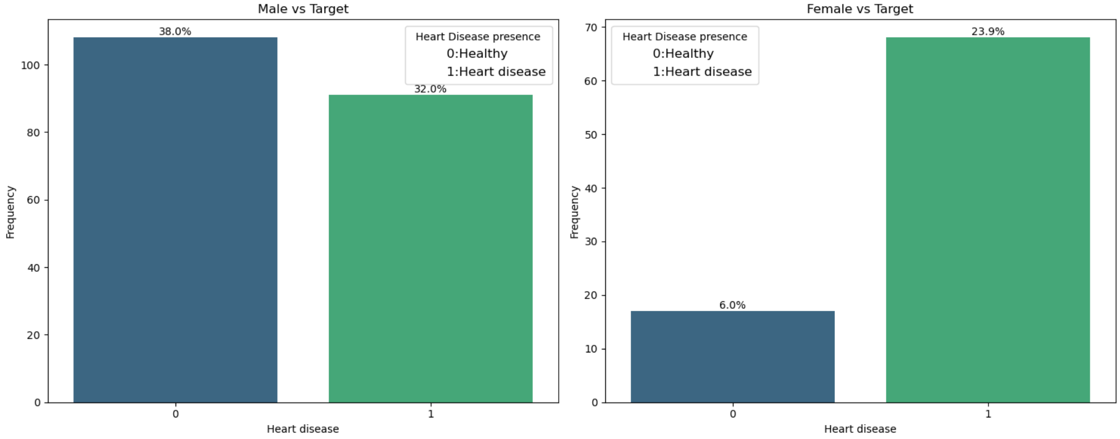
[19]:

```python
#  fucntion to add percentage on plots
def add_percentage_labels(ax):
    total = len(df)
    for p in ax.patches:
        percentage = '{:.1f}%'.format(100 * p.get_height() / total)
        x = p.get_x() + p.get_width() / 2
        y = p.get_height()
        ax.annotate(percentage, (x, y), ha='center', va='bottom')
```

[20]:

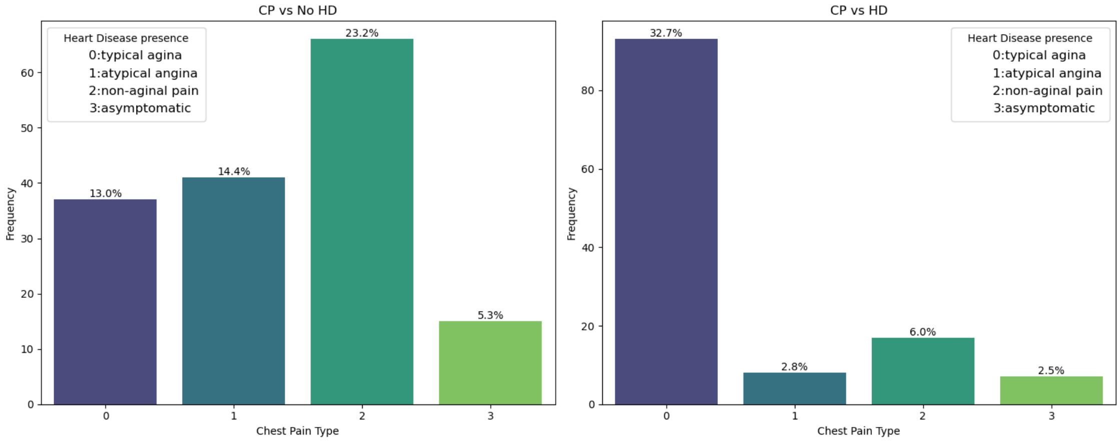
```python
query = "SELECT sex, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

# Separate the data by sex
male_data = data2[data2['sex'] == 1]
female_data = data2[data2['sex'] == 0]

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot count plot for males
sns.countplot(x='target', data=male_data, palette='viridis', ax=axes[0])
axes[0].set_title('Male vs Target')
axes[0].set_xlabel('Heart disease')
```

```
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for females
sns.countplot(x='target', data=female_data, palette='viridis', ax=axes[1])
axes[1].set_title('Female vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

option1 = mpatches.Patch(label='0:Healthy ', facecolor='none')
option2 = mpatches.Patch(label='1:Heart disease', facecolor='none')
axes[0].legend(handles=[option1, option2], borderpad=0.5, fontsize=12, title =␣
 ↪"Heart Disease presence")
axes[1].legend(handles=[option1, option2], borderpad=0.5, fontsize=12,title =␣
 ↪"Heart Disease presence")

# Adjust layout
plt.tight_layout()
plt.show()
```



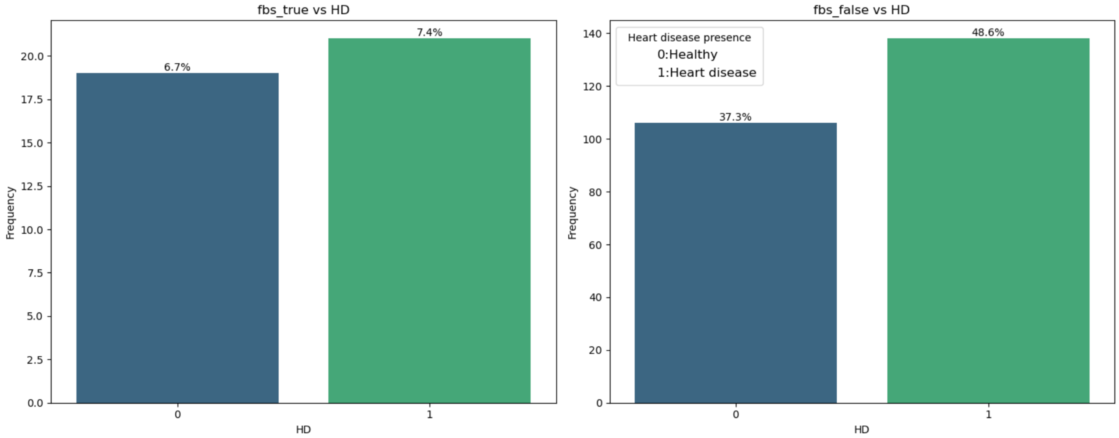
[21]:
```
query = "SELECT cp, target FROM 'Heart disease'"
data1 = pd.read_sql_query(query, conn)

no_HD = data1[data1['target'] == 1]
is_HD = data1[data1['target'] == 0]

# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot count plot for "Chest pain for woman"
```

```
sns.countplot(x='cp', data=no_HD, palette='viridis', ax=axes[0])
axes[0].set_title('CP vs No HD')
axes[0].set_xlabel('Chest Pain Type')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for "Chest pain for men"
sns.countplot(x='cp', data=is_HD, palette='viridis', ax=axes[1])
axes[1].set_title('CP vs HD')
axes[1].set_xlabel('Chest Pain Type')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Creating legend
option0= mpatches.Patch(label='0:typical agina', facecolor = 'none')
option1 = mpatches.Patch(label='1:atypical angina',facecolor = 'none')
option2 = mpatches.Patch(label='2:non-aginal pain',facecolor = 'none')
option3 = mpatches.Patch(label='3:asymptomatic',facecolor = 'none')
axes[0].legend(handles=[option0,option1, option2,option3], borderpad=0.5,␣
 ↪fontsize=12, title = "Heart Disease presence")
axes[1].legend(handles=[option0,option1, option2,option3], borderpad=0.5,␣
 ↪fontsize=12, title = "Heart Disease presence")

plt.tight_layout()
plt.show()
```



[22]:
```
query = "SELECT fbs, target FROM 'Heart disease'"
data1 = pd.read_sql_query(query, conn)

fbs_yes = data1[data1['fbs'] == 1]
fbs_no = data1[data1['fbs'] == 0]
```

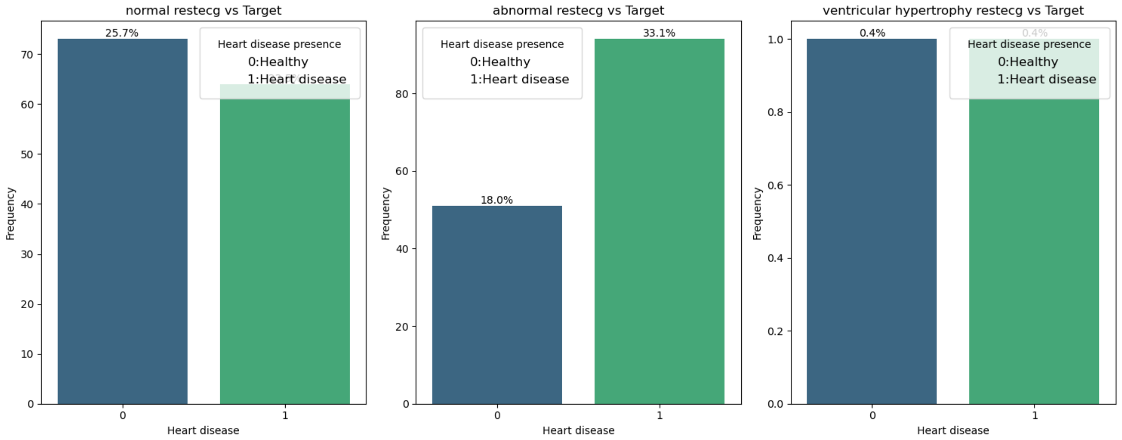
```python
# Create a figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot count plot for fbs = "true"
sns.countplot(x='target', data=fbs_yes, palette='viridis', ax=axes[0])
axes[0].set_title('fbs_true vs HD')
axes[0].set_xlabel('HD')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for fbs ="false"
sns.countplot(x='target', data=fbs_no, palette='viridis', ax=axes[1])
axes[1].set_title('fbs_false vs HD')
axes[1].set_xlabel('HD')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Creating legend
option0= mpatches.Patch(label='0:Healthy', facecolor = 'none')
option1 = mpatches.Patch(label='1:Heart disease',facecolor = 'none')
axes[1].legend(handles=[option0,option1], borderpad=0.5, fontsize = 12, title =␣
 ↪"Heart disease presence")

plt.tight_layout()
plt.show()
```



[23]:

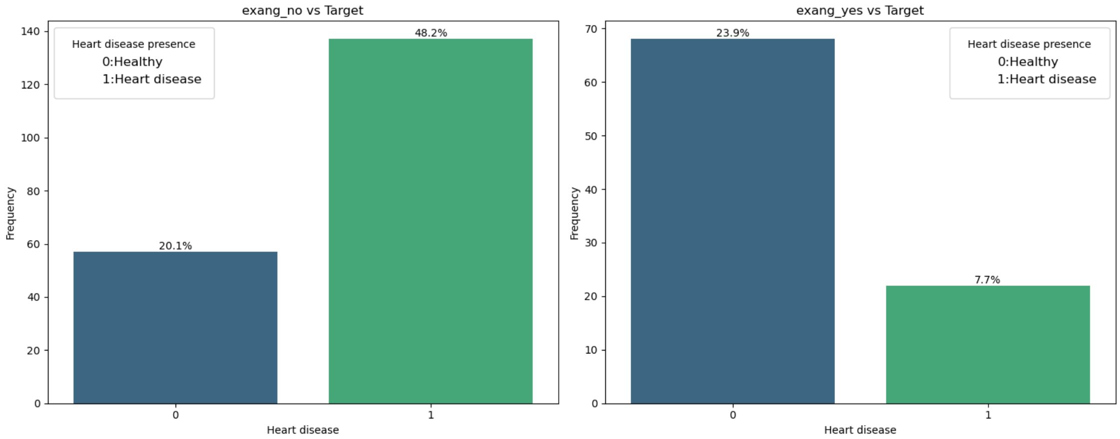
```python
query = "SELECT restecg, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

restecg_0 = data2[data2['restecg'] == 0]
```

```
restecg_1 = data2[data2['restecg'] == 1]
restecg_2 = data2[data2['restecg'] == 2]

# Create a figure with three subplots (1 row, 3 columns)
fig, axes = plt.subplots(1, 3, figsize=(15, 6))

# Plot count plot for normal restecg
sns.countplot(x='target', data=restecg_0, palette='viridis', ax=axes[0])
axes[0].set_title('normal restecg vs Target')
axes[0].set_xlabel('Heart disease')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for abnormal restecg
sns.countplot(x='target', data=restecg_1, palette='viridis', ax=axes[1])
axes[1].set_title('abnormal restecg vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Plot count plot for Ventricular hypertrophy restecg
sns.countplot(x='target', data=restecg_2, palette='viridis', ax=axes[2])
axes[2].set_title('ventricular hypertrophy restecg vs Target')
axes[2].set_xlabel('Heart disease')
axes[2].set_ylabel('Frequency')
add_percentage_labels(axes[2])

option1 = mpatches.Patch(label='0:normal ', facecolor='none')
option2 = mpatches.Patch(label='1:abnormal', facecolor='none')
option3 = mpatches.Patch(label='2:hypertrophy', facecolor='none')

option0= mpatches.Patch(label='0:Healthy', facecolor = 'none')
option1 = mpatches.Patch(label='1:Heart disease',facecolor = 'none')
axes[0].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title =␣
 ↪"Heart disease presence")
axes[1].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title =␣
 ↪"Heart disease presence")
axes[2].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title =␣
 ↪"Heart disease presence")

plt.tight_layout()
plt.show()
```



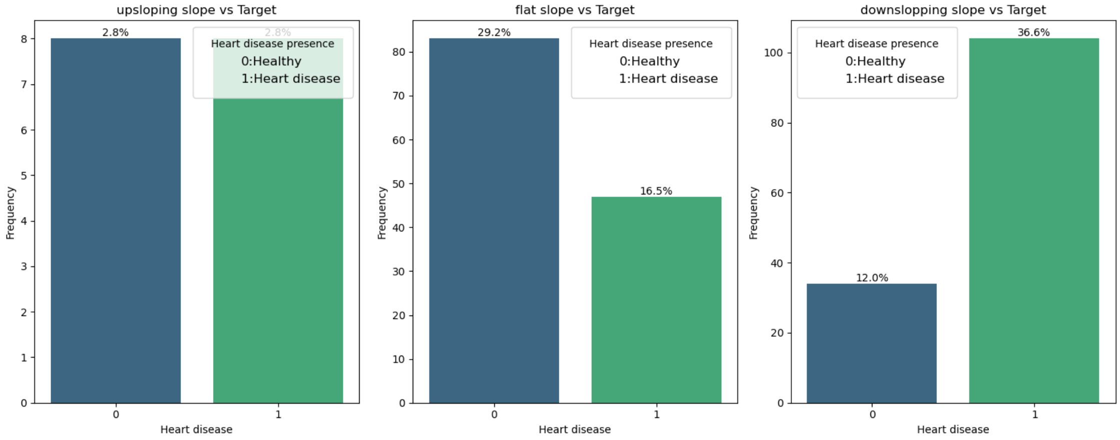
[24]:

```
query = "SELECT exang, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

exang_yes = data2[data2['exang'] == 1]
exang_no = data2[data2['exang'] == 0]

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Plot count plot for exang_no
sns.countplot(x='target', data=exang_no, palette='viridis', ax=axes[0])
axes[0].set_title('exang_no vs Target')
axes[0].set_xlabel('Heart disease')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for exang_yes
sns.countplot(x='target', data=exang_yes, palette='viridis', ax=axes[1])
axes[1].set_title('exang_yes vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

option0= mpatches.Patch(label='0:Healthy', facecolor = 'none')
option1 = mpatches.Patch(label='1:Heart disease',facecolor = 'none')
axes[0].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title =␣
 ↪"Heart disease presence")
axes[1].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title =␣
 ↪"Heart disease presence")

plt.tight_layout()
plt.show()
```



[25]:
```
query = "SELECT slope, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

slope_0 = data2[data2['slope'] == 0]
slope_1 = data2[data2['slope'] == 1]
slope_2 = data2[data2['slope'] == 2]

# Create a figure with three subplots (1 row, 3 columns)
fig, axes = plt.subplots(1, 3, figsize=(15, 6))

# Plot count plot for upsloping slope
sns.countplot(x='target', data=slope_0, palette='viridis', ax=axes[0])
axes[0].set_title('upsloping slope vs Target')
axes[0].set_xlabel('Heart disease')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for flat slope
sns.countplot(x='target', data=slope_1, palette='viridis', ax=axes[1])
axes[1].set_title('flat slope vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Plot count plot for downslopping slope
sns.countplot(x='target', data=slope_2, palette='viridis', ax=axes[2])
axes[2].set_title('downslopping slope vs Target')
axes[2].set_xlabel('Heart disease')
axes[2].set_ylabel('Frequency')
```

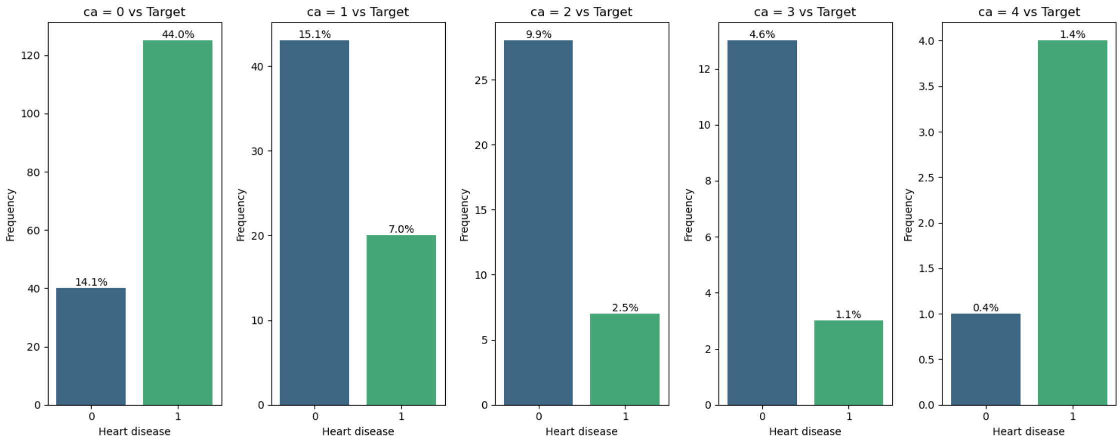
```
add_percentage_labels(axes[2])

option0= mpatches.Patch(label='0:Healthy', facecolor = 'none')
option1 = mpatches.Patch(label='1:Heart disease',facecolor = 'none')
axes[0].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title = "Heart disease presence")
axes[1].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title = "Heart disease presence")
axes[2].legend(handles=[option0,option1], borderpad=1, fontsize = 12, title = "Heart disease presence")

plt.tight_layout()
plt.show()
```



[26]:

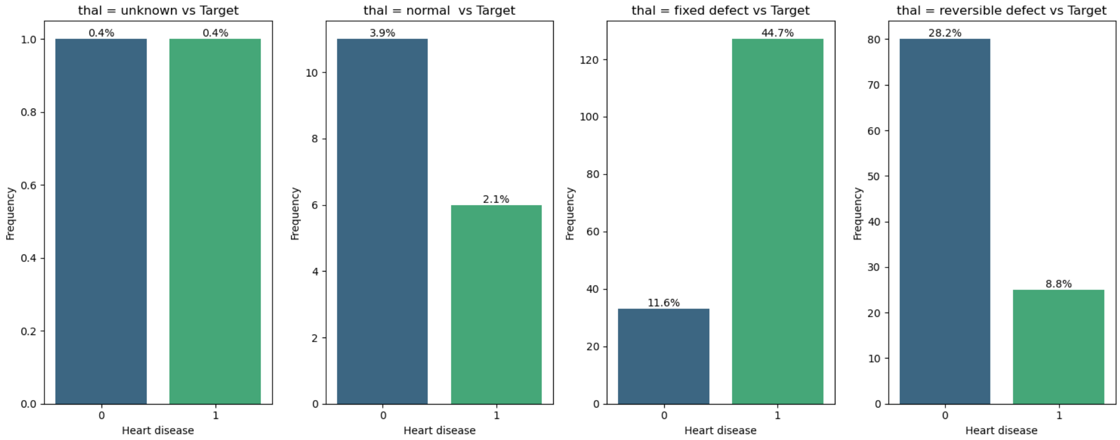
```
query = "SELECT ca, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

ca_0 = data2[data2['ca'] == 0]
ca_1 = data2[data2['ca'] == 1]
ca_2 = data2[data2['ca'] == 2]
ca_3 = data2[data2['ca'] == 3]
ca_4 = data2[data2['ca'] == 4]

# Create a figure with five subplots (1 row, 5 columns)
fig, axes = plt.subplots(1, 5, figsize=(15, 6))

# Plot count plot for upsloping ca=0
sns.countplot(x='target', data=ca_0, palette='viridis', ax=axes[0])
axes[0].set_title('ca = 0 vs Target')
axes[0].set_xlabel('Heart disease')
axes[0].set_ylabel('Frequency')
```

```
add_percentage_labels(axes[0])

# Plot count plot for flat ca=1
sns.countplot(x='target', data=ca_1, palette='viridis', ax=axes[1])
axes[1].set_title('ca = 1 vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Plot count plot for ca=2
sns.countplot(x='target', data=ca_2, palette='viridis', ax=axes[2])
axes[2].set_title('ca = 2 vs Target')
axes[2].set_xlabel('Heart disease')
axes[2].set_ylabel('Frequency')
add_percentage_labels(axes[2])

# Plot count plot for ca=3
sns.countplot(x='target', data=ca_3, palette='viridis', ax=axes[3])
axes[3].set_title('ca = 3 vs Target')
axes[3].set_xlabel('Heart disease')
axes[3].set_ylabel('Frequency')
add_percentage_labels(axes[3])

# Plot count plot for ca=4
sns.countplot(x='target', data=ca_4, palette='viridis', ax=axes[4])
axes[4].set_title('ca = 4 vs Target')
axes[4].set_xlabel('Heart disease')
axes[4].set_ylabel('Frequency')
add_percentage_labels(axes[4])

plt.tight_layout()
plt.show()
```



[27]:

```python
query = "SELECT thal, target FROM 'Heart disease'"
data2 = pd.read_sql_query(query, conn)

thal_0 = data2[data2['thal'] == 0]
thal_1 = data2[data2['thal'] == 1]
thal_2 = data2[data2['thal'] == 2]
thal_3 = data2[data2['thal'] == 3]

# Create a figure with two subplots (1 row, 2 columns)
fig, axes = plt.subplots(1, 4, figsize=(15, 6))

# Plot count plot for thal=unknown
sns.countplot(x='target', data=thal_0, palette='viridis', ax=axes[0])
axes[0].set_title('thal = unknown vs Target')
axes[0].set_xlabel('Heart disease')
axes[0].set_ylabel('Frequency')
add_percentage_labels(axes[0])

# Plot count plot for thal= normal
sns.countplot(x='target', data=thal_1, palette='viridis', ax=axes[1])
axes[1].set_title('thal = normal  vs Target')
axes[1].set_xlabel('Heart disease')
axes[1].set_ylabel('Frequency')
add_percentage_labels(axes[1])

# Plot count plot for thal=abnormal
sns.countplot(x='target', data=thal_2, palette='viridis', ax=axes[2])
axes[2].set_title('thal = fixed defect vs Target')
axes[2].set_xlabel('Heart disease')
axes[2].set_ylabel('Frequency')
add_percentage_labels(axes[2])

# Plot count plot for thal=reversible defect
sns.countplot(x='target', data=thal_3, palette='viridis', ax=axes[3])
axes[3].set_title('thal = reversible defect vs Target')
axes[3].set_xlabel('Heart disease')
axes[3].set_ylabel('Frequency')
add_percentage_labels(axes[3])

plt.tight_layout()
plt.show()
```

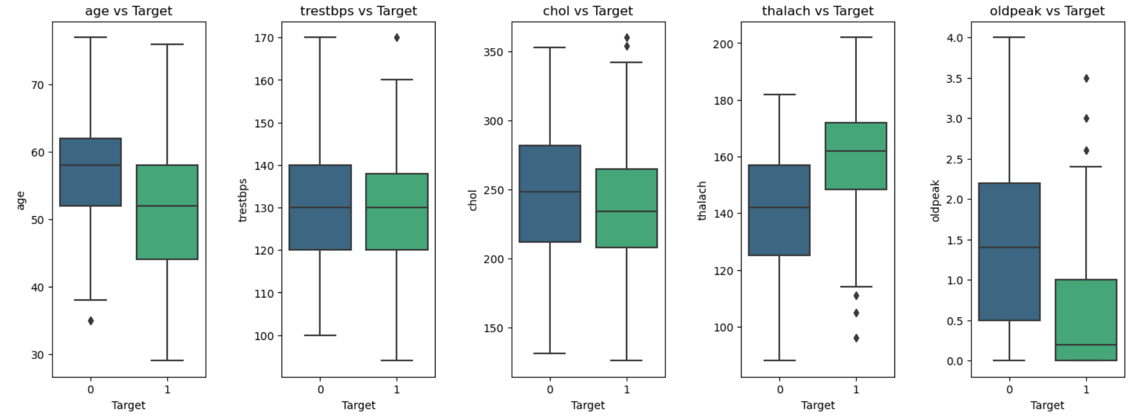


2.2 2.1.c

2.2.1 numerical variables = [‘age’ , ‘trestbps’, ‘chol’, ‘thalach’, ‘oldpeak’]

[28]:

```
numerical_variables =  ['age' , 'trestbps', 'chol', 'thalach', 'oldpeak', ]

fig, axes = plt.subplots(1, 5, figsize=(15, 6))

query = "SELECT age,trestbps, chol, thalach, oldpeak, target FROM 'Heart␣
 ↪disease'"
data2 = pd.read_sql_query(query, conn)

for i in range(5):
    sns.boxplot(x='target', y= numerical_variables[i], data=data2,␣
 ↪palette='viridis',ax=axes[i])
    axes[i].set_title( f'{numerical_variables[i]} vs Target')
    axes[i].set_xlabel('Target')
    axes[i].set_ylabel(f'{numerical_variables[i]}')

plt.tight_layout(pad=3.0)
```



3 Question 3

3.1 3.1

[12]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,␣
  ↪confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

query = "SELECT age, sex, cp, trestbps, chol, fbs,restecg, thalach, exang,␣
  ↪oldpeak, slope, ca, thal  FROM 'Heart disease'"
X = pd.read_sql_query(query, conn)

query = "SELECT target FROM 'Heart disease'"
y = pd.read_sql_query(query, conn)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=16)


# Create a StandardScaler object
scaler = StandardScaler()

# Fit the scaler on the training data and transform
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train.shape,y_train.shape,X_test.shape,y_test.shape
```

[12]:
```
((227, 13), (227, 1), (57, 13), (57, 1))
```

3.2 3.2

3.3 Logistic regression (72% accuracy)

[23]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Define the logistic regression model
model_lr = LogisticRegression()

# Train the model
model_lr.fit(X_train, y_train)

# Make predictions
y_pred = model_lr.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

```
Accuracy:  0.7192982456140351
```

```
C:\Users\Shivan\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

3.4 Randam Forest (74% accuracy)

[24]:

```python
from sklearn.ensemble import RandomForestClassifier

# Initialize Random Forest classifier
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model
random_forest.fit(X_train, y_train)

# Make predictions
y_pred = random_forest.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

```
C:\Users\Shivan\AppData\Local\Temp\ipykernel_6512\849478621.py:7:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
```

```
ravel().
  random_forest.fit(X_train, y_train)
```

```
Accuracy:  0.7368421052631579
```

3.5 SVM model (74% accuracy)

[25]:

```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

```
Accuracy:  0.7368421052631579
```

```
C:\Users\Shivan\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1143:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[27]:

```
conn.close()
```

References

- https://reintech.io/blog/connect-to-database-with-python
- https://docs.python.org/3/library/sqlite3.html
- https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/
- https://www.geeksforgeeks.org/box-plot-in-python-using-matplotlib/
- https://www.geeksforgeeks.org/countplot-using-seaborn-in-python/
- https://www.datacamp.com/tutorial/understanding-logistic-regression-python
- https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
- https://www.linkedin.com/pulse/basics-decision-tree-python-omkar-sutar/
- https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107
- https://dhirajkumarblog.medium.com/random-forest-algorithm-advantages-and-disadvantages-1ed22650c84f
- https://medium.com/@akshayjain_757396/advantages-and-disadvantages-of-logistic-regression-in-machine-learning-a6a247e42b20
- https://medium.com/@benjaminkipkem/building-a-machine-learning-model-with-streamlit-8e28ea8516a0
- https://www.youtube.com/watch?v=8M20LyCZDOY&list=PLtqF5YXg7GLmCvTswG32NqQypOuYkPRUE&index=2&t=353s&pp=iAQB
- https://www.youtube.com/watch?v=ZZ4B0QUHuNc&list=PLtqF5YXg7GLmCvTswG32NqQypOuYkPRUE&index=1&t=300s&pp=iAQB
- https://www.youtube.com/watch?v=Ebb4gUI2IpQ&pp=ygUJc3RyZWFtbGl0
- https://www.youtube.com/watch?v=xl0N7tHiwlw&t=2606s&pp=ygUJc3RyZWFtbGl0