NAME : Vikash Shakya

REG. NO. : 23122143

Section : B

Model building and EDA to predict a possible heart attack

In [ ]:

```
#importing libraries
import numpy as np
import pandas as pd

import warnings as warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
data = pd.read_csv("heart.csv")

print("HEART DISEASE DATASET : ")
data
```

```
HEART DISEASE DATASET :
```

Out[ ]:

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

303 rows × 14 columns

In [ ]:
```
#array of all columns
print(data.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

Description of columns

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

In [ ]:
```
# checking total no. of rows and columns present in original datset
data.shape
```

Out[ ]:
```
(303, 14)
```

Data Cleaning

In [ ]:
```
# getting information about total non-null values and datatypes
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [ ]:
```
# to get percentage of null values present in all columns respectively
round(data.isnull().sum()*100/len(data),2)
```

```
Out[ ]: age         0.0
        sex         0.0
        cp          0.0
        trestbps    0.0
        chol        0.0
        fbs         0.0
        restecg     0.0
        thalach     0.0
        exang       0.0
        oldpeak     0.0
        slope       0.0
        ca          0.0
        thal        0.0
        target      0.0
        dtype: float64
```

- **No null values present in dataset. We are good to go!**

Data Analysis

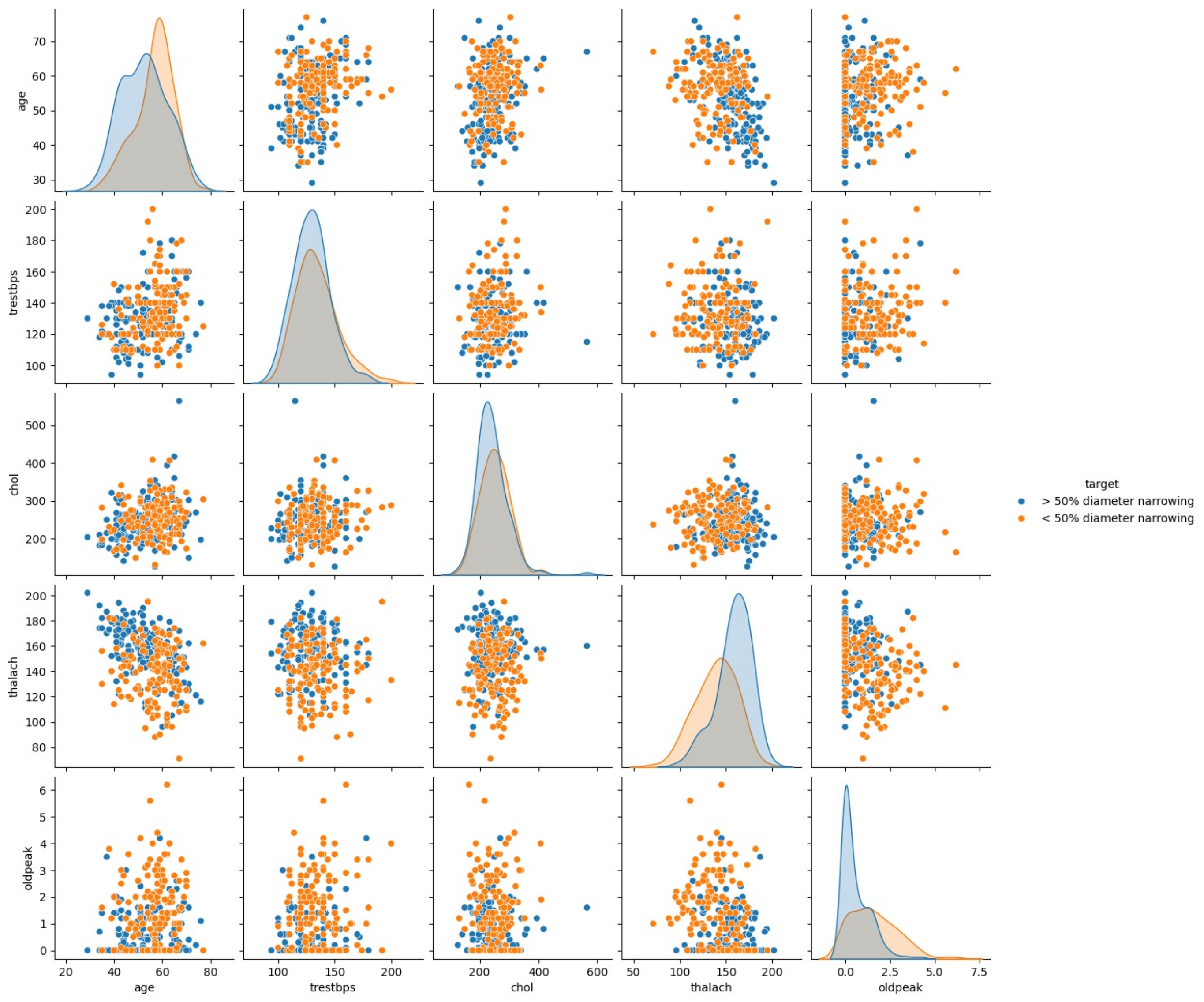
```
In [ ]: # firstly we will make a copy of our original dataset to perform further actions
        data_to_use = data.copy()
```

Visualising Numeric Variables

Let's make a pairplot of all the numeric variables

- Pair plot is used to understand the best set of features to explain a relationship between two variables or to form the most separated clusters.

```
In [ ]: # Visualizing the
        cont_data = data_to_use[["age","trestbps","chol","thalach","oldpeak","target"]].copy()
        cont_data['target'] = cont_data['target'].replace({0: '< 50% diameter narrowing', 1: '> 50% diameter narrowing'

        sns.pairplot(cont_data ,hue = "target" )
        plt.show()
```



Let's make a heatmap of all the numeric variables

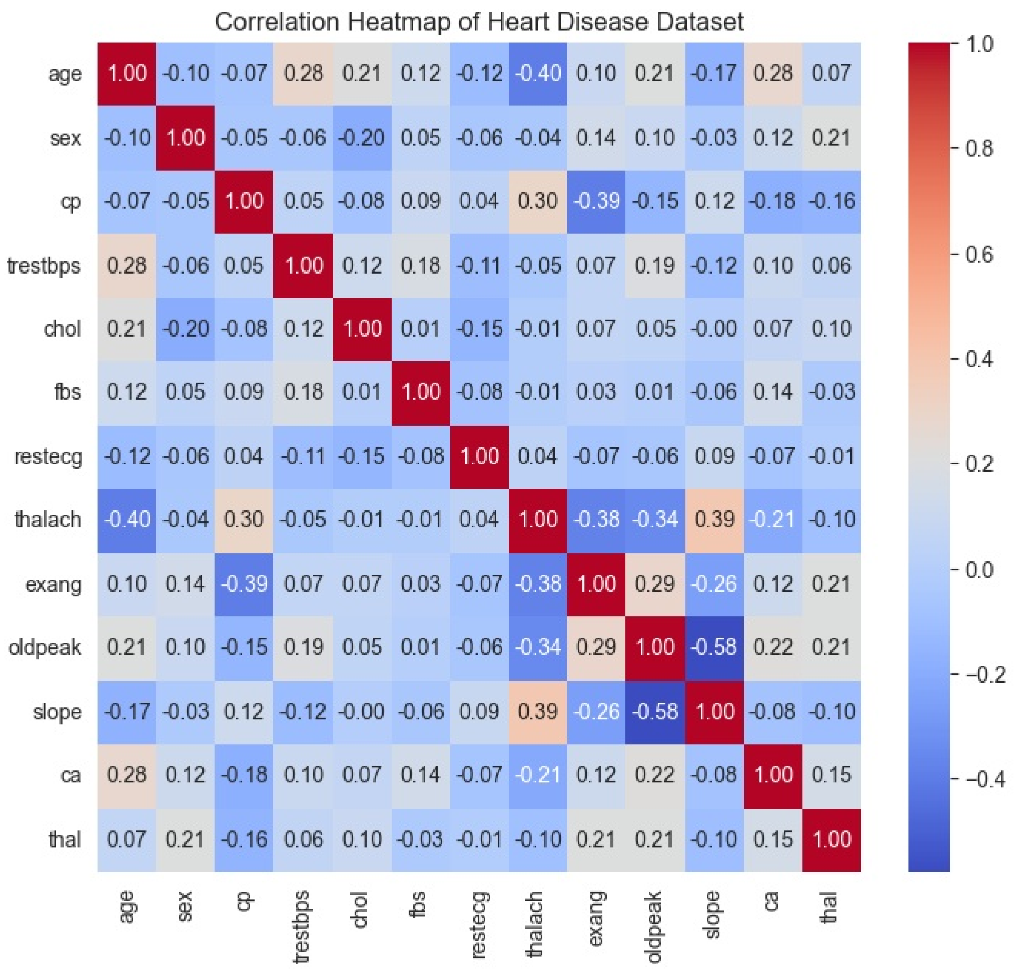
- Heatmap is also used to understand the best set of features to explain a relationship between two variables .
- It is usually not used that much in drawing any final conclusions but a great way to visualize the relation between features by looking at the different color representation of them.
- The heat map shows the relative intensity of values captured by your eye tracker by assigning each value a color representation.

```
In [ ]: df2 = data_to_use[['age', 'sex','cp','trestbps','chol','fbs','restecg','thalach','exang','oldpeak','slope','ca'
        print(df2)
```

```
     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
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[303 rows x 13 columns]
```

In [ ]:
```python
corr_matrix = df2.corr()
```

In [ ]:
```python
plt.figure(figsize=(8, 7))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title("Correlation Heatmap of Heart Disease Dataset")
plt.show()
```

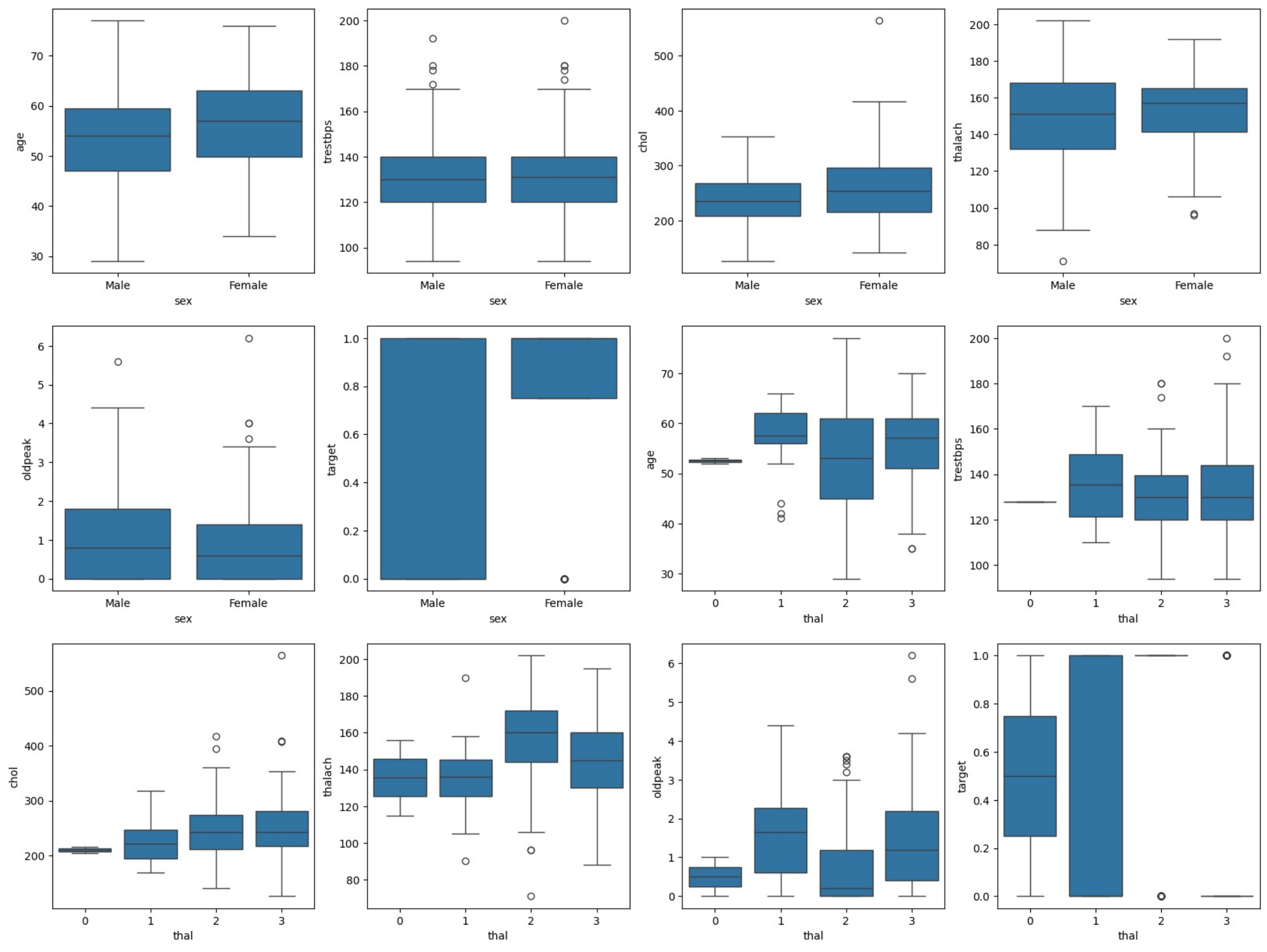


Visualising Categorical Variables

As you might have noticed, there are a few categorical variables as well. Let's make a boxplot for some of these variables.

- Outliers(for a normal distribution).
- “Minimum” and “Maximum” of different categories in a boxplot.

In [ ]:
```python
categ_data_to_use_in_boxplot = data_to_use[['sex','thal']].copy()
categ_data_list = categ_data_to_use_in_boxplot.columns.values.tolist()
cont_data_list = cont_data.columns.values.tolist()
data_modified = data.copy()
data_modified['sex'] = data_modified['sex'].replace({0: 'Female', 1: 'Male'})


plt.figure(figsize=(20,15))

a = 1
for i in enumerate(categ_data_list):
    for j in enumerate(cont_data_list):
        plt.subplot(3,4,a)
        sns.boxplot(x=i[1], y= j[1], data=data_modified)
        a +=1

plt.subplots_adjust(hspace = 0.2)

plt.show()
```



Observations:

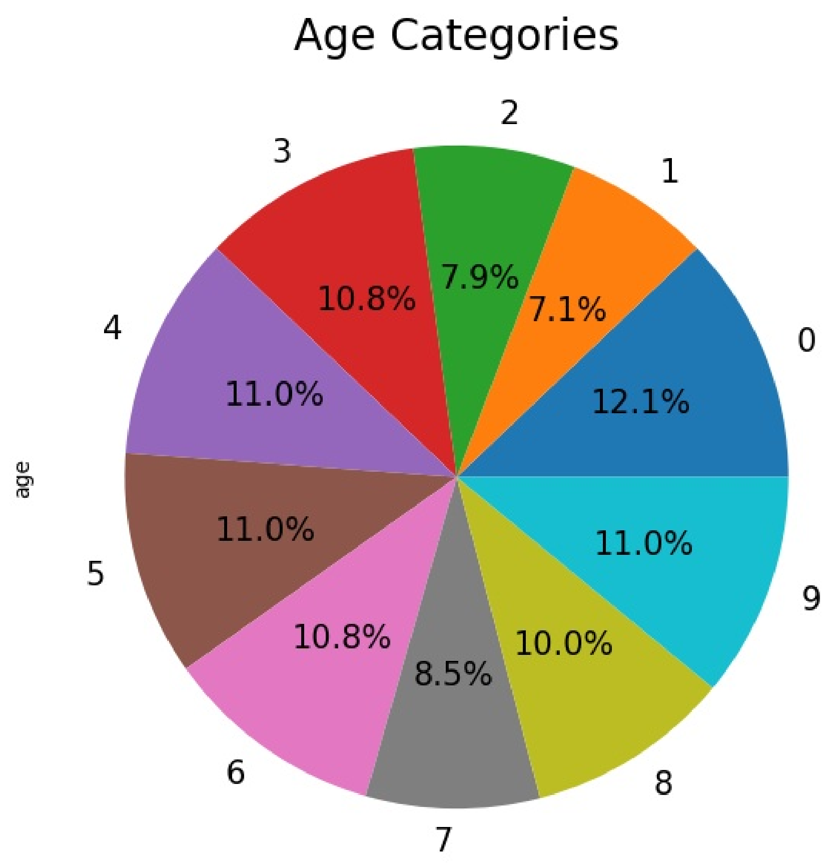
By observing boxplots of sex against different features we get,

- Maximum and minimum of age in male is greater than and less than maximum and minimum of age in female respectively.It means that this dataset contains more younger and old males than females of respective category.
- Resting Blood Pressure is same in both sexes but also both male and female data of resting blood pressure contains outliers.
- **Cholestrol level** is slightly **more in female** compared to male.
- Inter Quartile Range of Maximum heart rate achieved for **male is between 140 and 170** whereas for **female it is between 140 and 160**.

By observing boxplots of thal(a blood disorder called thalassemia) against different features we get to know that

- Thal **value of 2**(**normal blood flow**) has lowest minimum and highest maximum which denotes its relationship against age.It has biggest IQR when compared to other thal values quartiles. Most common Thal value for all age groups is thal value 2(normal blood flow).
- Thal **value of 1**(**fixed defect** (no blood flow in some part of the heart)) against age is between **50 and 70 years old.**
- Thal **value of 3**(**reversible defect** (a blood flow is observed but it is not normal)) against age have its median at **58 years** old approximately.
- In **thal vs chol** plot we can see that the medians of cholestrol levels are increasing as thal values are increasing which shows that cholestrol level and thalessemia are positively correlated.
- In **thal vs thallach** plot we can see that **median of thal value 2 is greatest** which denotes that even after having normal blood flow the range of maximum heart reate achieved is greater than other thal values.

In [ ]:
```
plt.figure(figsize=(7,7))
df = data.age.value_counts().to_frame()
textprops = {"fontsize":15}
df2.head(10).age.plot(kind='pie',autopct='%1.1f%%',textprops =textprops )

plt.title("Age Categories",fontsize = 20)
plt.show()
```



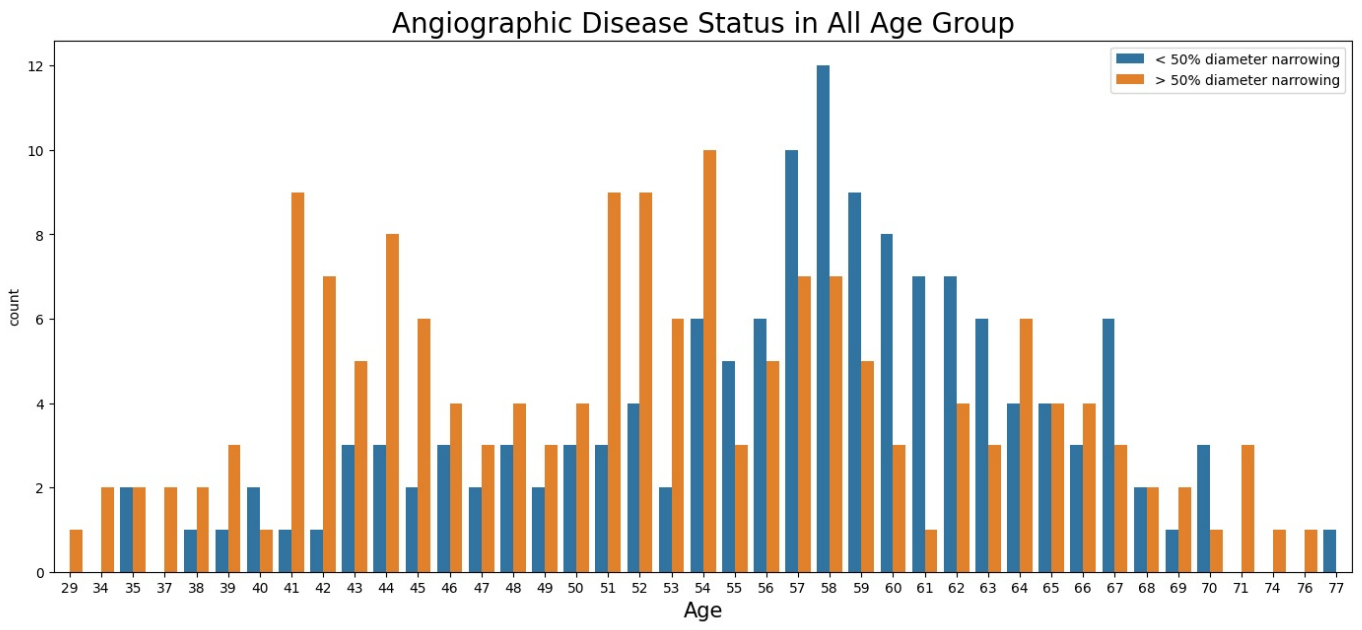
Observations:

- Most count of records of age in the dataset is **58 years old** with **14.1%**.
- We can see from the pie plot that the person present in the dataset mostly belongs to age group between **50 - 60 years old.**
- **However it is a very small dataset with very limited amount of observations so we can't make age a deciding factor.**

Let's visualize more about some important relation between features and target variable

Histogram plot

```
In [ ]: plt.figure(figsize=(17,7))
        ax = sns.countplot(x="age", hue="target", data=data_to_use)
        plt.title("Angiographic Disease Status in All Age Group",fontsize = 20)
        plt.legend(["< 50% diameter narrowing","> 50% diameter narrowing"],loc='upper right')
        plt.xlabel("Age",fontsize = 15)
        plt.show()
```

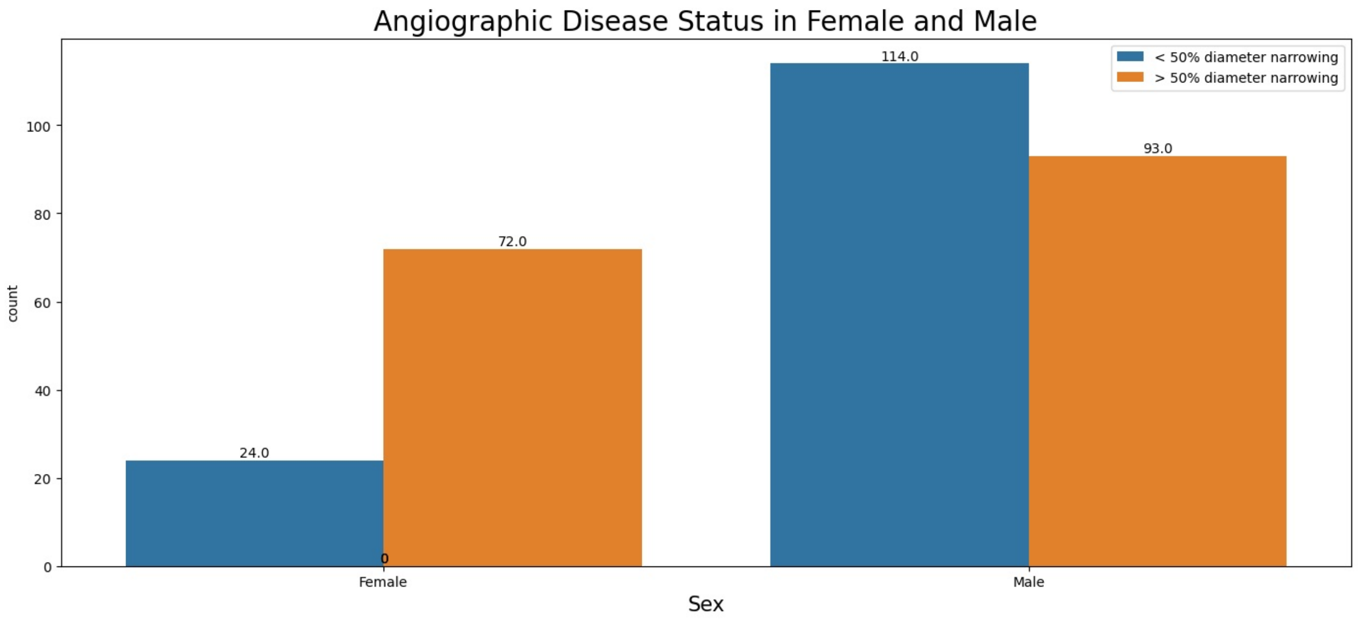


Observations:

According to dataset if we try to draw some conclusions on the basis of this plot then we conclude that,

- The orange bars shows that the person is at higher risk getting a heart attack and the blue bars shows that person is at less risk.
- The **age group 51-60 years old** is at more risk.
- Due to lack of more observations and small dataset we can see that according to chart, young people are at more risk than old people which is not true.

In [ ]:
```
x = ["Female","Male"]
values = range(len(x))

plt.figure(figsize=(17,7))
ax = sns.countplot(x="sex", hue="target", data=data_to_use)
for rect in ax.patches:
    ax.text (rect.get_x() + rect.get_width()  / 2,rect.get_height()+ 0.75,rect.get_height(),horizontalalignment=
plt.title("Angiographic Disease Status in Female and Male",fontsize = 20)
plt.legend(["< 50% diameter narrowing","> 50% diameter narrowing"],loc='upper right')
plt.xlabel("Sex",fontsize = 15)
plt.xticks(values,x)
plt.show()
```



Observations

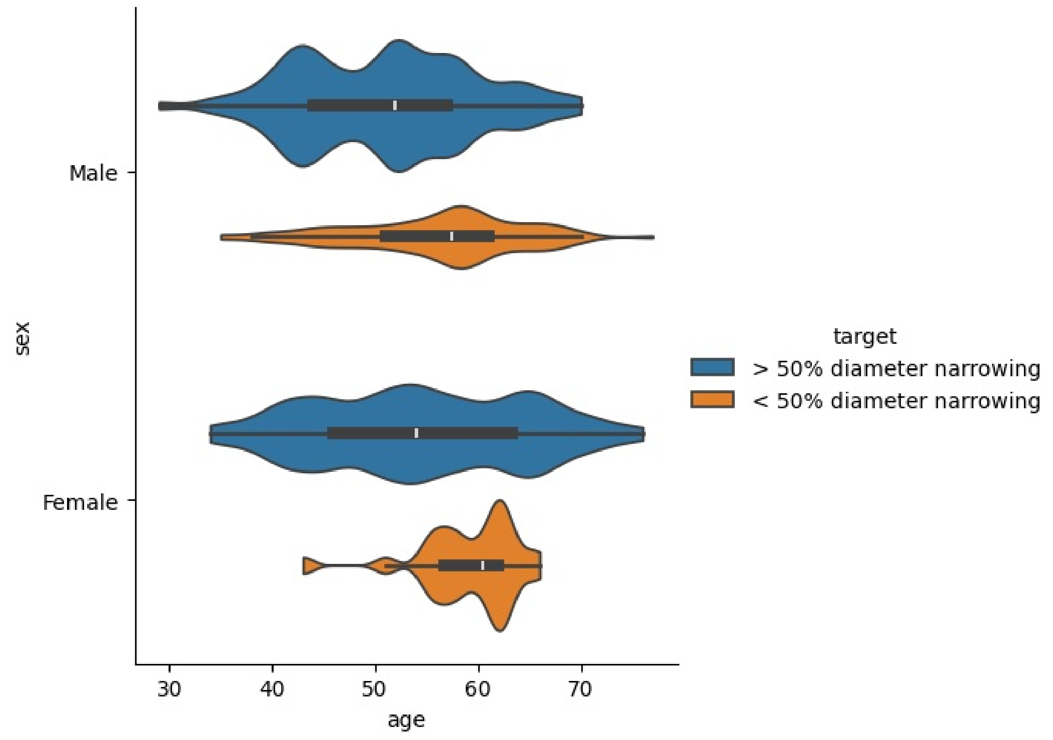
- *Such difference in male and female plot is due to less records of female present in dataset.*
- Ratio of more susceptible to a possible heart attack to less susceptible in **female** is **3:1**.
- Ratio of more susceptible to a possible heart attack to less susceptible in **male** is almost **3:4**.

Violin plot used for categorical data

Analyzing relationship between age and sex with target variable in single plot

In [ ]:
```
data_modified['target'] = data_modified['target'].replace({0: '< 50% diameter narrowing', 1: '> 50% diameter na
plt.figure(figsize=(10,7))
sns.catplot(x="age", y="sex",
            hue="target",
            data=data_modified,
            orient="h", height=5, aspect=1, palette="tab10",
            kind="violin", dodge=True, cut=0, bw=.2)

plt.show()
```

```
<Figure size 1000x700 with 0 Axes>
```

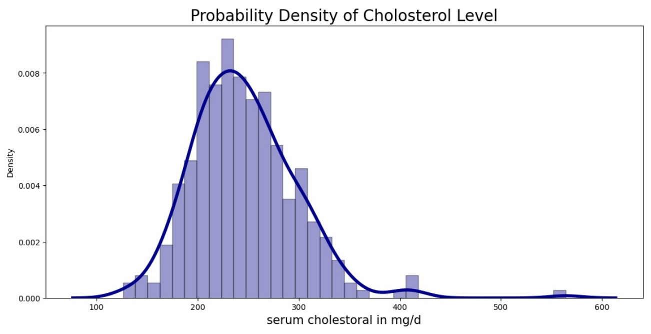


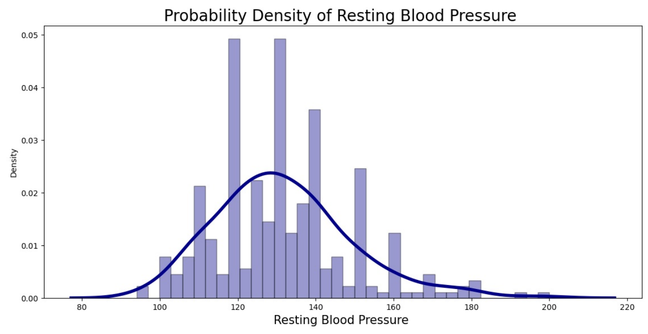
Observations:

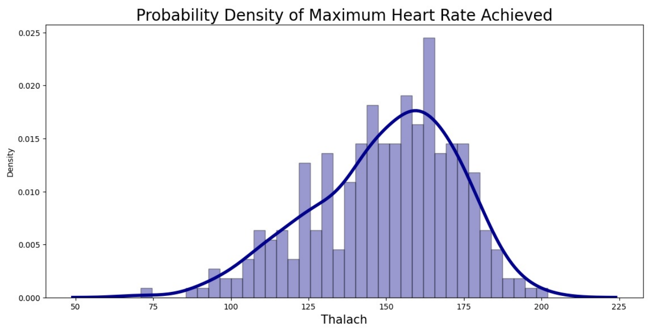
- In plot of female, where target: > 50% diameter narrowing (more susceptible to a heart attack), the range of age is greater and evenly distributed with flatter peaks.
- Whereas in plot of female, where target: < 50% diameter narrowing (less susceptible to a heart attack), the range of age is shorter and has higher peaks at certain age groups near 55 years and 62 years.
- We can see there are very less outliers in dataset.

Probablity Density Plot of Some More Features

In [ ]:
```
plt.figure(figsize = (30,15))
a = 1
for i in ["chol","trestbps","thalach"]:

    plt.subplot(2,2,a)
    sns.distplot(data_to_use[i],hist=True, kde=True,
                 bins=int(180/5), color = 'darkblue',
                 hist_kws={'edgecolor':'black'},
                 kde_kws={'linewidth': 4})
    if i=="chol":

        plt.title("Probability Density of Cholosterol Level",fontsize = 20)
        plt.xlabel("serum cholestoral in mg/d",fontsize = 15)

    elif i == "thalach":
        plt.title("Probability Density of Maximum Heart Rate Achieved",fontsize = 20)
        plt.xlabel("Thalach",fontsize = 15)

    else:
        plt.title("Probability Density of Resting Blood Pressure ",fontsize = 20)
        plt.xlabel("Resting Blood Pressure ",fontsize = 15)
    a+=1

plt.subplots_adjust(hspace = 0.4)
plt.show()
```







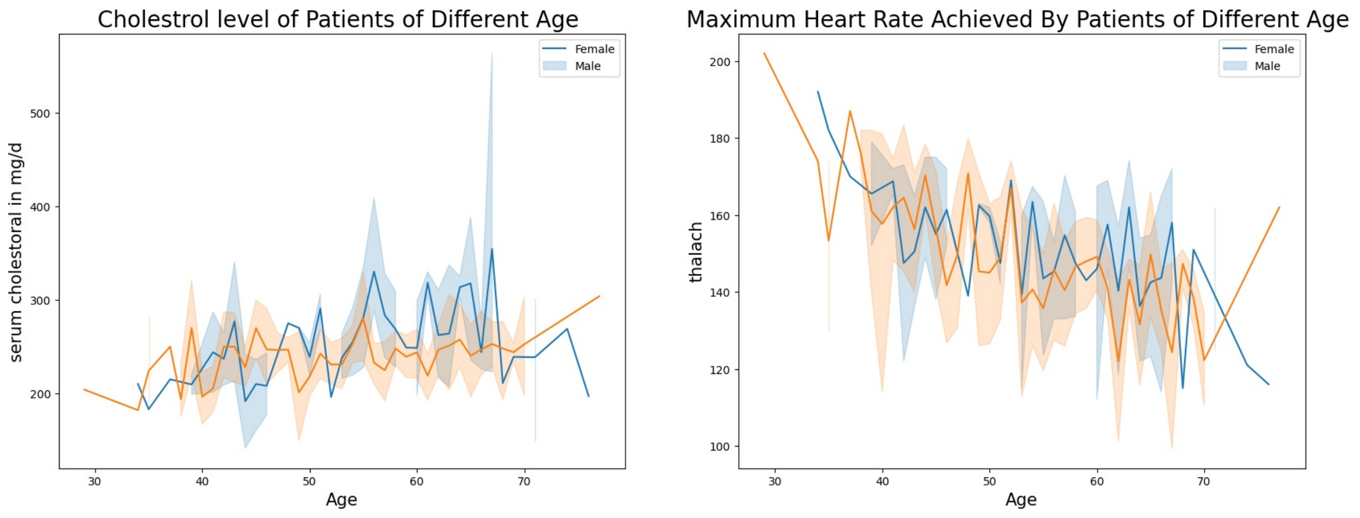
Observations:

- Most common cholesterol level present in dataset is between **200 mg/d to 240 mg/d** with maximum value of **230 mg/d** which is considered **borderline to moderately elevated**.
- Few records of Thalach are at near **400 mg/d** and **550mg/d** which is very high and risky.
- Resting blood pressure at **120** and **125** have highest probablity density of **0.05**.
- Probablity density of **Thalach(maximum heart rate achieved)** is mostly evenly distributed between **140** and **175**.
- Highest probablity of **Thalach(maximum heart rate achieved)** is nearly at **165**.

Line plot of cholosterols level and Thalach in male and female with their ages

- Serum cholesterol levels helps in figuring out risk for developing heart disease.
- The maximum heart rate greater than (220-present age) is considerd hazardous.

In [ ]:
```python
plt.figure(figsize = (20,7))
a = 1
y = ["chol","thalach"]
for feature in y:
    plt.subplot(1,2,a)
    sns.lineplot(data=data_to_use, x="age", y=feature,hue = "sex")
    plt.xlabel("Age",fontsize = 15)
    plt.legend(["Female", "Male"], loc ="upper right")

    if feature=="chol":
        plt.title("Cholestrol level of Patients of Different Age ",fontsize = 20)
        plt.ylabel("serum cholestoral in mg/d",fontsize = 15)
    else:
        plt.title("Maximum Heart Rate Achieved By Patients of Different Age ",fontsize = 20)
        plt.ylabel("thalach",fontsize = 15)
    a+=1
plt.show()
```



Observations:

- Serum cholestrol level in Female is higher than Male between the age of 52 years to 68 years.
- From the thalach plot we can see that as the age is increasing the chart is showing downward trend directing towards the fact that maximum heart rate achieved decreases as the person grows older.

Data preparation

```
In [ ]: X = data_to_use.drop("target",axis =1)
        y = data_to_use["target"]

        datalist = data.columns.values.tolist()
        datalist.remove("target")
```

Model Building - Logistic Regression

Before applying Logistic Regression ,

- We will split the data into train-test set.
- Then data will be scaled.
- For continuos data Normalization technique (MinMaxscaler) is used.
- for categorical data Mean encoding technique is used.

Splitting the dataset into train and test dataset

```
In [ ]: from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import MinMaxScaler

        X_train, X_test, y_train, y_test= train_test_split(X,y, test_size= 0.25, random_state=120)
```

Scaling of Train Data

```
In [ ]: # Target Mean Encoding for categorical columns in train data

        for i in ['age','cp','restecg','slope','ca','thal'] :
            Mean_encoded_variable = data_to_use.groupby([i])['target'].mean().to_dict()

            X_train[i] =  X_train[i].map(Mean_encoded_variable)

        # Normalization for continuos column in train data

        scaler = MinMaxScaler()
        num_vars = ['trestbps', 'chol', 'thalach', 'oldpeak']

        # Fit on object

        X_train[num_vars] = scaler.fit_transform(X_train[num_vars])
```

Training of the Model using Logistic Regression

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

        model= LogisticRegression()

        model.fit(X_train, y_train)
        trainscore =  model.score(X_train,y_train)
```

Scaling of Test Data and Applying Test Data on Model

```
In [ ]: # Target Mean Encoding on categorical columns in test data
        for i in ['age','cp','restecg','slope','ca','thal'] :
            Mean_encoded_variable1 = data_to_use.groupby([i])['target'].mean().to_dict()

            X_test[i] =  X_test[i].map(Mean_encoded_variable1)

        # Normalization for continuos columns in test data

        X_test[num_vars] = scaler.transform(X_test[num_vars])

        testscore =  model.score(X_test,y_test)
```

Test Score and Train Score

In [ ]:
```
print("test score: {} \ntrain score: {}".format(testscore*100,trainscore*100),'\n')

y_pred =  model.predict(X_test)
```

```
test score: 88.1578947368421
train score: 84.14096916299559
```

Confusion Matrix

In [ ]:
```
from sklearn.metrics import confusion_matrix

print("Confusion Matrix : \n",confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix :
 [[29  4]
 [ 5 38]]
```

Different Classification Metrics

In [ ]:
```
from sklearn.metrics import classification_report,accuracy_score,f1_score,precision_score,recall_score,roc_curv

print(' f1 score: ',f1_score(y_test, y_pred)*100,'\n')
print(' Accuracy: ',accuracy_score(y_test, y_pred)*100,'\n')
print(' precision score: ',precision_score(y_test, y_pred)*100,'\n')
print(' recall score: ',recall_score(y_test, y_pred)*100,'\n')
print("Classification report: \n",classification_report(y_test, y_pred))
```

```
 f1 score:  89.41176470588236

 Accuracy:  88.1578947368421

 precision score:  90.47619047619048

 recall score:  88.37209302325581

Classification report:
               precision    recall  f1-score   support

           0       0.85      0.88      0.87        33
           1       0.90      0.88      0.89        43

    accuracy                           0.88        76
   macro avg       0.88      0.88      0.88        76
weighted avg       0.88      0.88      0.88        76
```

- Accuracy is used when the True Positives and True negatives are more important while F1-score is used when the False Negatives and False Positives are crucial.
- Accuracy can be used when the class distribution is similar while F1-score is a better metric when there are imbalanced classes as in the above case.
- In most real-life classification problems, imbalanced class distribution exists and thus **F1-score is a better metric** to evaluate our model on.
- In our model, most important metric will be **F1-score**. As it is used when the False Negatives are crucial.

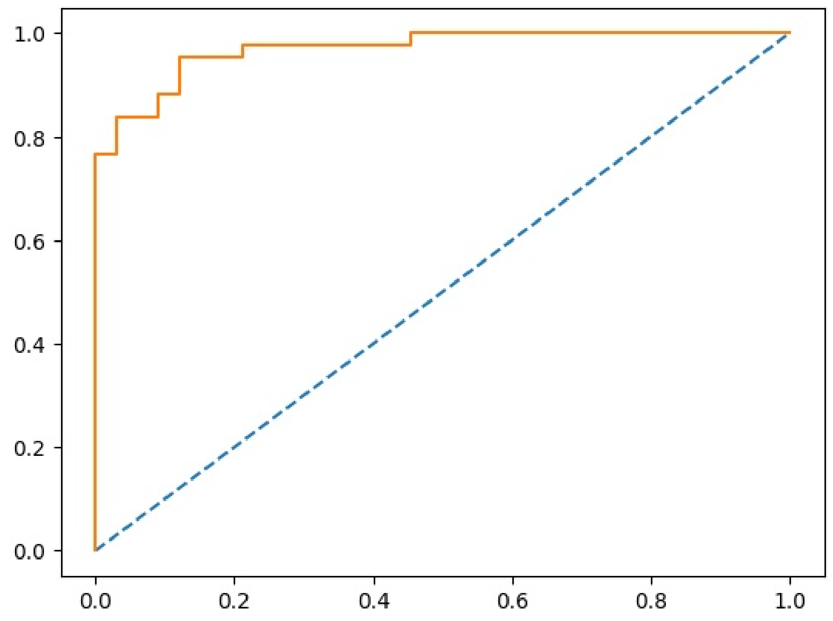
In [ ]:
```
probabilityValues = model.predict_proba(X_test)[:,1]
auc = roc_auc_score(y_test, y_pred)
print("AUC Score: ",auc*100)
```

```
AUC Score:  88.12544045102186
```

ROC Curve

In [ ]:
```
fpr,tpr, threshold =  roc_curve(y_test,probabilityValues)
plt.plot([0,1],[0,1], linestyle = '--')
plt.plot(fpr,tpr)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x18b43733050>]
```



- K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters.
- It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
- t is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd
```

- The k-means clustering algorithm mainly performs two tasks:
-
  - Determines the best value for K center points or centroids by an iterative process.
-
  - Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

- The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready

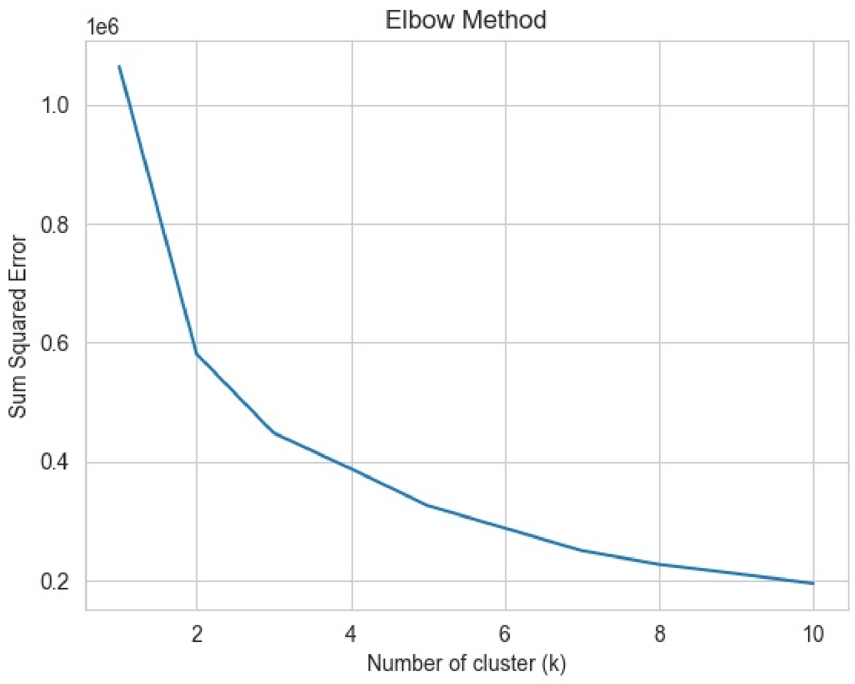
In [ ]:
```
X = data_to_use.drop("age",axis =1)
y = data_to_use["age"]
```

In [ ]:
```
#Find optimum number of cluster
sse = [] #SUM OF SQUARED ERROR
for k in range(1,11):
        km = KMeans(n_clusters=k, random_state=2)
        km.fit(X)
        sse.append(km.inertia_)
```

- Elbow Method

Finding the ideal number of groups to divide the data into is a basic stage in any unsupervised algorithm. One of the most common techniques for figuring out this ideal value of k is the elbow approach.

In [ ]:
```
sns.set_style("whitegrid")
g=sns.lineplot(x=range(1,11), y=sse)

g.set(xlabel ="Number of cluster (k)",
        ylabel = "Sum Squared Error",
        title ='Elbow Method')

plt.show()
```



- From the above graph, we can observe that at k=2 and k=3 elbow-like situation. So, we are considering K=3

- Build the Kmeans clustering model

In [ ]:
```
kmeans = KMeans(n_clusters = 3, random_state = 2)  # Choose the optimal number of clusters
kmeans.fit(X)
```

Out[ ]:
```
KMeans
KMeans(n_clusters=3, random_state=2)
```

- Find the cluster center

In [ ]:
```
kmeans.cluster_centers_
```

Out[ ]:
```
array([[5.24590164e-01, 7.04918033e-01, 1.37147541e+02, 3.21540984e+02,
        1.63934426e-01, 4.91803279e-01, 1.50213115e+02, 3.77049180e-01,
        1.15573770e+00, 1.44262295e+00, 9.67213115e-01, 2.37704918e+00,
        4.59016393e-01],
       [7.23809524e-01, 1.02857143e+00, 1.27866667e+02, 1.96171429e+02,
        1.33333333e-01, 6.47619048e-01, 1.50076190e+02, 2.85714286e-01,
        1.00476190e+00, 1.43809524e+00, 6.09523810e-01, 2.20000000e+00,
        6.38095238e-01],
       [7.22627737e-01, 1.03649635e+00, 1.32043796e+02, 2.51138686e+02,
        1.53284672e-01, 4.52554745e-01, 1.49065693e+02, 3.35766423e-01,
        1.01459854e+00, 1.35036496e+00, 7.15328467e-01, 2.37226277e+00,
        5.10948905e-01]])
```

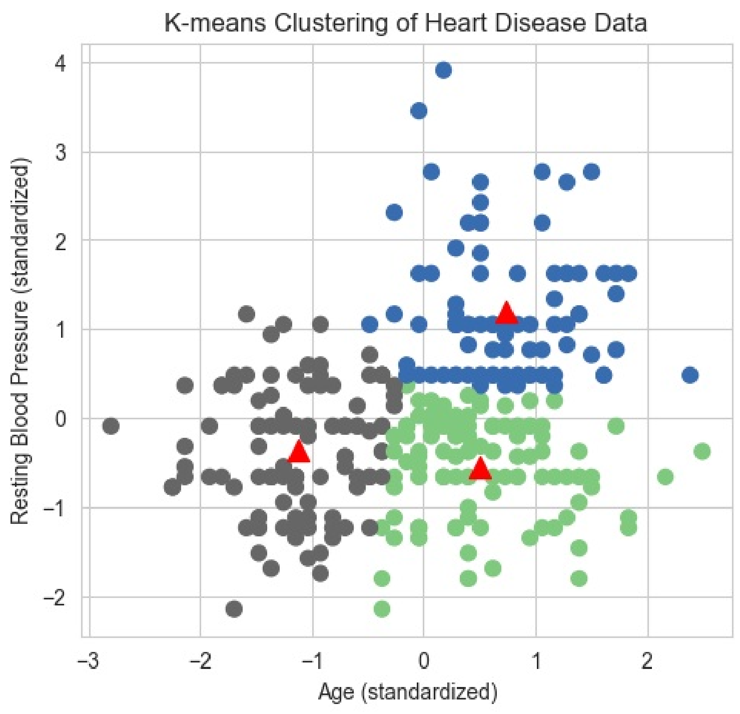
- Predict the cluster group:

In [ ]:
```
pred = kmeans.fit_predict(X)
pred
```

```
Out[ ]: array([2, 2, 1, 2, 0, 1, 0, 2, 1, 1, 2, 2, 2, 1, 2, 1, 0, 2, 2, 2, 2, 2,
               2, 2, 1, 0, 1, 1, 0, 1, 1, 1, 1, 2, 1, 1, 0, 2, 2, 0, 0, 2, 1, 2,
               0, 0, 2, 2, 1, 2, 2, 0, 2, 1, 2, 1, 1, 2, 1, 0, 2, 0, 1, 1, 1, 1,
               1, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 0, 0, 0, 2, 0, 2, 1,
               1, 2, 2, 1, 1, 0, 1, 2, 0, 2, 0, 2, 2, 2, 1, 2, 1, 1, 2, 2, 2, 2,
               0, 1, 0, 1, 2, 1, 1, 1, 1, 2, 0, 2, 2, 2, 1, 1, 1, 2, 1, 2, 1, 2,
               0, 2, 0, 2, 1, 1, 1, 2, 0, 0, 1, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 2,
               1, 1, 2, 1, 1, 1, 2, 0, 1, 1, 1, 0, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1,
               2, 0, 1, 2, 0, 2, 0, 2, 2, 0, 2, 2, 2, 1, 0, 1, 1, 2, 1, 0, 2, 2,
               2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 1, 0, 2, 0, 2, 0, 2, 2,
               0, 1, 2, 0, 2, 1, 2, 1, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 2, 2, 2,
               1, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 0, 2, 0, 2, 1, 2, 2, 2, 2, 2, 2,
               1, 1, 0, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2, 0, 1, 0, 1, 1, 2, 1, 0,
               1, 2, 0, 1, 1, 0, 2, 1, 1, 1, 1, 1, 2, 2, 1, 1, 2])
```

- Plot the cluster center with data points

```
In [ ]: # Plotting
plt.figure(figsize=(12, 5))

# Scatter plot for the clusters
plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], c=pred, cmap=cm.Accent, s=50)
plt.grid(True)

# Plotting cluster centers
for center in kmeans.cluster_centers_:
    plt.scatter(center[0], center[1], marker='^', c='red', s=100)

plt.xlabel("Age (standardized)")
plt.ylabel("Resting Blood Pressure (standardized)")
plt.title("K-means Clustering of Heart Disease Data")
plt.show()
```



- The plot display Age vs.Resting Blood Pressure with data points colored by clusters, and red markers indicate K-means cluster centers.