Prediction of Heart Disease

Problem 1) Cleaning data and visualization

1) Reading of dataset using read.excel and first 5 rows using head method:

The dataset consists of 270 rows and 13 columns

Problem 1) Cleaning data and visualization

In [93]:
```
# Use pandas.read_excel to read the heart_disease.xlsx file into a dataframe called data

excel = '/Users/karan/Downloads/heart_disease.xlsx'
data=pd.read_excel(excel)
```

In [94]:
```
df1=data   # temporary data frame for analysis
```

In [95]:
```
# Look at the first 5 rows by using head(5)
df1.head()
```

Out[95]:

	age	sex	chest_pain	bp	chol	blood_sugar	ecg	heart_rate	angia	oldpeak	slope	vessel_num	thal	heart_disease
0	70	1	4	130	322	0	2	109	0	2.4	2	3	3	2
1	67	0	3	115	564	0	2	160	0	1.6	2	0	7	1
2	57	1	2	124	261	0	0	141	0	0.3	1	0	7	2
3	64	1	4	128	263	0	0	105	1	0.2	2	1	7	1
4	74	0	2	120	269	0	2	121	1	0.2	1	1	3	1

2) Statistical Analysis of the dataset

In [96]:
```
# Get the statistics of your dataframe
# statistical analysis for numerical variables (mean,median,std,max,min, etc.)
df1.describe()
```

Out[96]:

	age	sex	chest_pain	bp	chol	blood_sugar	ecg	heart_rate	angia	oldpeak	slope	vessel_num	
count	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.000000	270.00000	270.000000	270.000000	270.
mean	54.433333	0.677778	3.174074	131.344444	249.659259	0.148148	1.022222	149.677778	0.329630	1.05000	1.585185	0.670370	4.
std	9.109067	0.468195	0.950090	17.861608	51.686237	0.355906	0.997891	23.165717	0.470952	1.14521	0.614390	0.943896	1.
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.00000	1.000000	0.000000	3.
25%	48.000000	0.000000	3.000000	120.000000	213.000000	0.000000	0.000000	133.000000	0.000000	0.00000	1.000000	0.000000	3.
50%	55.000000	1.000000	3.000000	130.000000	245.000000	0.000000	2.000000	153.500000	0.000000	0.80000	2.000000	0.000000	3.
75%	61.000000	1.000000	4.000000	140.000000	280.000000	0.000000	2.000000	166.000000	1.000000	1.60000	2.000000	1.000000	7.
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.20000	3.000000	3.000000	7.

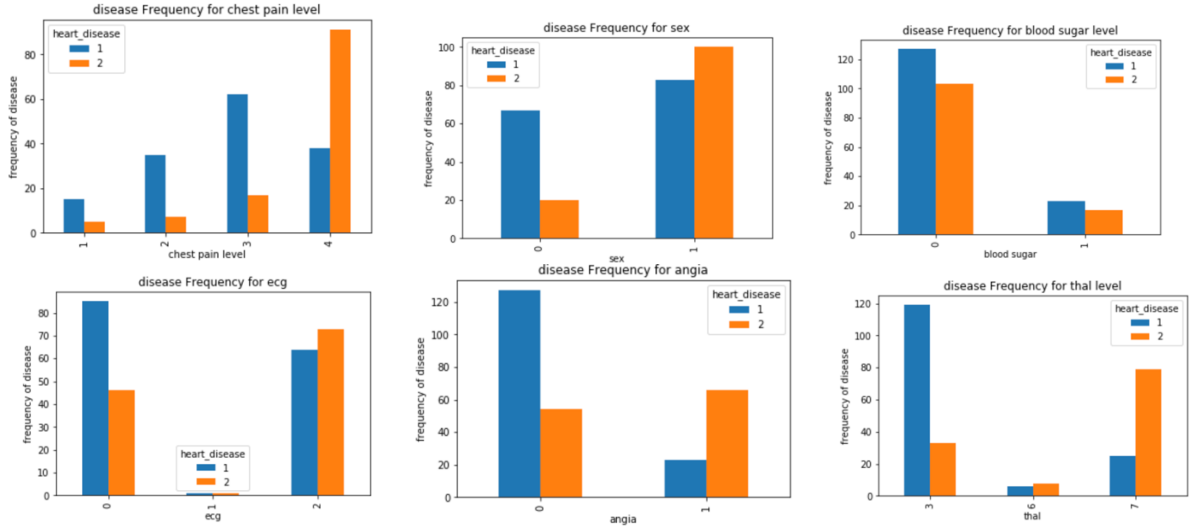
3) To check of there is any missing values or not

```
In [97]: # Are there any columns that contain a missing value? If yes, substitute those with the mean value of each column!
         df1.isna().sum().sort_values(ascending=False)
         # No missing values are found in the dataset #
```

```
Out[97]: heart_disease    0
         thal             0
         vessel_num       0
         slope            0
         oldpeak          0
         angia            0
         heart_rate       0
         ecg              0
         blood_sugar      0
         chol             0
         bp               0
         chest_pain       0
         sex              0
         age              0
         dtype: int64
```

⇨ There is no missing value found in the dataset

4) Data Visualization to find the relationship of heart disease with other variables



Results from Visualization :

⇨ maximum heart disease is present for sex 1

⇨ Most population has blood sugar level 0 out of which more number of people do not have heart disease.

⇨ Most population with ECG 0 do not have heart disease while population with ECG 2 have more heart disease. however, Off the less popuation who has ecg level 1, an equal ratio has heart disease than the ones who dont.

⇨ The dominating angia 0 have more people with no heart disease while out of the population who have angia 1, there are more cases with a heart disease.

⇨ People who have thal level 3 are less likely to have a heart disease but people who have thal 7 are very much more likely to have a heart disease.

⇨ Chest pain level 4 has the maximum presence of heart disease While Chest pain level 3 has the most absence of heart disease

5) To Create Dummy variables for the categorical variables in the dataset

```
# Create a set of dummy variables from the Cat variables
df1_sex = pd.get_dummies(df1['sex'])
df1_sex = df1_sex.rename(columns = {0:"Female", 1:"Male"})
df1_chestpain = pd.get_dummies(df1['chest_pain'])
df1_chestpain = df1_chestpain.rename(columns = {1:"CP1", 2:"CP2", 3:"CP3", 4:"CP4"})
df1_bloodsugar = pd.get_dummies(df1['blood_sugar'])
df1_bloodsugar = df1_bloodsugar.rename( columns = {0:"Blood Sugar Level 1", 1:"Blood Sugar Level 2"})
df1_ecg = pd.get_dummies(df1['ecg'])
df1_ecg = df1_ecg.rename(columns = {0:"ECG 0", 1:"ECG 1", 2:"ECG 2"})
df1_angia = pd.get_dummies(df1['angia'])
df1_angia = df1_angia.rename(columns = {0:"Angia 0 ", 1:"Angia 1"})
df1_thal = pd.get_dummies(df1['thal'])
df1_thal = df1_thal.rename(columns = {3:"Thal 3", 6:"Thal 6", 7:"Thal 7"})
```

```
df1_new = pd.concat([df1, df1_sex,df1_chestpain, df1_bloodsugar,df1_ecg,df1_angia, df1_thal], axis=1)
#del df1_new['chest_pain','blood_sguar','sex','ecg','angia','thal']
df1_new = df1_new.drop(['chest_pain','blood_sugar','sex','ecg','angia','thal'], 1)
```

⇨ Checking the head of the new data frame:

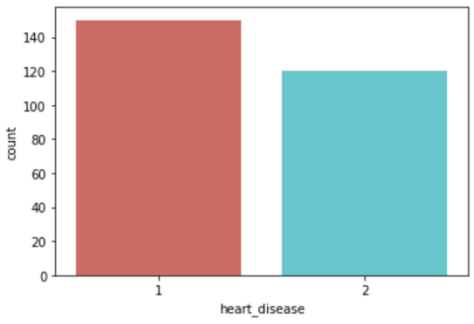
```
df1_new.head()
```

	age	bp	chol	heart_rate	oldpeak	slope	vessel_num	heart_disease	Female	Male	...	Blood Sugar Level 1	Blood Sugar Level 2	ECG 0	ECG 1	ECG 2	Angia 0	Angia 1	Thal 3	Thal 6	Thal 7
0	70	130	322	109	2.4	2	3	2	0	1	...	1	0	0	0	1	1	0	1	0	0
1	67	115	564	160	1.6	2	0	1	1	0	...	1	0	0	0	1	1	0	0	0	1
2	57	124	261	141	0.3	1	0	2	0	1	...	1	0	1	0	0	1	0	0	0	1
3	64	128	263	105	0.2	2	1	1	0	1	...	1	0	1	0	0	0	1	0	0	1
4	74	120	269	121	0.2	1	1	1	1	0	...	1	0	0	0	1	0	1	1	0	0

5 rows × 24 columns

6) What is number of classes disease/not_disease?

⇨ 150 has absence of heart disease while 120 has heart disease



```
In [111]: # Get your class labels in a variable called y, and remove it from your dataframe .
          # Now, data contains your samples and features, and y contains your labels
          y = df1_new['heart_disease']
          df1_new.drop('heart_disease', axis=1, inplace=True)

In [112]: np.random.seed(999)

          # Split the data into a train and validation set. You won't touch the test set until the very end of this program.
          # Perform any analysis (model selection, hyperparameter selection) on your training data (X_train_outer)
          # DO NOT TOUCH X_test UNTIL THE END OF THIS PROGRAM!!!!
          from sklearn.model_selection import train_test_split
          X_train_outer, X_test, y_train_outer, y_test = train_test_split(df1_new, y, test_size=0.2)
          print (X_train_outer.shape, y_train_outer.shape)
          print (X_test.shape, y_test.shape)

          (216, 23) (216,)
          (54, 23) (54,)
```

Splitting the dataset into train and test on 80:20 ratio.

Problem 2) Classification using nearest neighbor classifier, and see overfitting and underfitting by changing the value of n_neighbors (vary it from 1 to the number of samples -- len(X_train_inner))

```
In [140]: # Fit a K-nearest neighbor classifier model to X_train_inner_std data --> Change the value of n_neighbors here and repe
          from sklearn.neighbors import KNeighborsClassifier
          classifier = KNeighborsClassifier(n_neighbors=172)
          classifier.fit(X_train_inner_std, y_train_inner)

Out[140]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=172, p=2,
                               weights='uniform')

In [141]: print(len((X_train_inner)))

          172

In [ ]:

In [142]: # Print accuracy on your validation data (X_val_std)
          classifier.score(X_val_std,y_val)

Out[142]: 0.7045454545454546

In [143]: # Print accuracy on your training data (X_train_inner_std)
          classifier.score(X_train_inner_std,y_train_inner)

Out[143]: 0.5232558139534884
```

What is the accuracy on training and validation when k=1? What about k=len(X_train_inner) ? When underfitting happens when overfitting?

When n_neighbor value is 1
accuracy on training data is 100%
and on validation data is 75%

Hence overfitting is happening when n_neighbor =1

When n_neighbor value is len(X_train_inner)) = > 172
Accuracy on training data is 70.45%
and on validation data is 52.32%

In [140]:
```
# Fit a K-nearest neighbor classifier model to X_train_inner_std data --> Change the value of n_neighbors here and repo
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=172)
classifier.fit(X_train_inner_std, y_train_inner)
```

Out[140]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=172, p=2,
                     weights='uniform')
```

In [141]:
```
print(len((X_train_inner)))
```

```
172
```

In [ ]:
```

```

In [142]:
```
# Print accuracy on your validation data (X_val_std)
classifier.score(X_val_std,y_val)
```

Out[142]:
```
0.7045454545454546
```

In [143]:
```
# Print accuracy on your training data (X_train_inner_std)
classifier.score(X_train_inner_std,y_train_inner)
```

Out[143]:
```
0.5232558139534884
```

Problem 3) So, how to find best k? Perform 10-fold Cross-validation on your train set to find the best value of k for nearest neighbor classifier (you can also find it by trying different values on the hold-out set you defined above, but since the dataset is small, it is better to perform 10-fold cross-validation, why? because you may by chance get a hold-out validation set that works well with k=1! you never know! So, do it on 10 different val sets and average them to make sure you got a good k!

To find the best K we need to use gridsearch and pass values of k from minimum to maximum i.e. 1 to 172 as a parameter and perform 10-cross validation
Average the accuracy result and take the best accuracy

In [118]:
```
# In each fold of your cross validation, your training fold needs to be first standardized (scaled), then an algorithm
# Then the same transformation will be applied to each validation fold
# This is done by pipeline and is extremely useful when you use GridSearchCV or cross_val_score and etc..
from sklearn.pipeline import Pipeline

scaler = StandardScaler()
clf = KNeighborsClassifier()
pipeline = Pipeline([('transformer', scaler), ('estimator', clf)])
```

In [119]:
```
# param_candidate defines the set of hyperparameters that you want your function to analyze
from sklearn.model_selection import GridSearchCV
param_candidate = [{'estimator__n_neighbors': np.arange(1, 172)}]
```

In [120]:
```
# Create a classifier object with GridSearchCV and param_candidate and fit the GridSearchCV object with X_train_outer,
# Read Python's documentation if anything is unclear!
# TO DO
clf = GridSearchCV(pipeline, param_grid=param_candidate, cv=10)

clf.fit(X_train_outer,y_train_outer)
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=Pipeline(memory=None,
                                steps=[('transformer',
                                        StandardScaler(copy=True,
                                                       with_mean=True,
                                                       with_std=True)),
                                       ('estimator',
                                        KNeighborsClassifier(algorithm='auto',
                                                             leaf_size=30,
                                                             metric='minkowski',
                                                             metric_params=None,
                                                             n_jobs=None,
                                                             n_neighbors=5, p=2,
                                                             weights='uniform'))],
                                verbose=False),
             iid='warn', n_jobs=None...
       105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117,
       118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130,
       131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143,
       144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
       157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
       170, 171])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Classifier Best Score :

```
Best score for data1: 0.8379629629629629
```

Best parameters for the model found using grid search:

```
Best k: {'estimator__n_neighbors': 24}
```

Best k value is 24 i.e. accuracy is highest when k is 24.

Problem 4) Perform logistic regression with the default value of C=1, and obtain the cross-validation performance

```
# Do standardization before classification using Pipeline as
# Problem 3. You don't need any param_candidate in this case.
steps = [('scaler', StandardScaler()), ('logclf', LogisticRegression(C=1))]
pipeline = Pipeline(steps)
```

```
# Use the cross_val_score on logistic regression on X_train_outer and y_train_outer with cv=10 and get scores
from sklearn.model_selection import cross_val_score
scores = cross_val_score(pipeline,X_train_outer,y_train_outer,cv=10)
print(scores)
```

```
[0.73913043 0.77272727 0.86363636 0.90909091 1.         0.71428571
 0.66666667 0.80952381 0.9047619  0.85714286]
```

Performed Logistic Regression with C=1 as the parameter, and standard scaling is done
We found that the cross-validation scores when cv=10 varies from .71 to .91
Also, average cross validation score in logistic regression is .8236

Problem 5) Cross-validation performance of which algorithm was better? K-nearest neighbor or logistic regression? You will pick the one that gave you the highest performance (accuracy), and train your model using all training data, test it on your test set (which you never touched up until this point), get your performance, and report it (to me, to the doctor, or the hospital as the generalization performance of your model)

In my model, the **Cross-validation performance** is better for KNN.

```
My logistic regression model gave average accuracy of .8236, However KNN mode
l gave the accuracy of .8379 at k= 24
```

So, I used KNN model on test data set and it predicted with the accuracy of 0 .8518.