HealthMonitoringSystem

May 30, 2019

0.0.1 Heart Monitoring System

Input Attributes The training dataset contains 14 attributes. The attributes from 0 to 12 are input to the system and the 13th attribute is the output that ranges from 0 to 4 which is used to train the system. The input to the given system is as follows:

a.Age: The input is given as the age in a number of years of the patient.

b.Sex: For female input value is 0, male input value is 1.

c.Chest Pain Type: i.Value 1- Typical Angina: Typical Angina is the substernal chest pain provoked by physical or emotional stress which was relieved by rest or nitro-glycerine. ii.Value 2- Atypical Angina: Atypical Angina is the chest pain which includes ambiguous chest discomfort and shortness of breath along with typical angina. iii.Value 3- Non-angina Pain: Chest pain with a duration of over 30 mins or less than 5 seconds and can be relive easily. iv.Value 4- Asymptomatic: Asymptomatic chest pain does not cause or exhibit the symptoms of the disease.

d.Resting Blood Pressure: It is the blood pressure value for a person normally without any vigorous activities. It is measured as 120 mm Hg systolic, which is the pressure when heartbeats and over 80 mm Hg diastolic, which is the pressure when the heart relaxes.

e.Serum Cholesterol in mg/dL: Total serum cholesterol consists of HDL cholesterol also known as good cholesterol, LDL Cholesterol also known as bad cholesterol, and triglycerides. Total serum cholesterol above 200 mg/dL raises the risk of heart disease.

f.Fasting Blood Sugar: It is the level of glucose after fasting for 8 hours. It should be less than 100 mg/dL. The input value should be 1 for FBS greater than 120 mg/dL or value should be 0 for FBS less than 120 mg/dL.

g.Resting Electrocardiograph Result: Resting electrocardiogram measures the electrical activity of the heart. Value 0 denotes normal, value 1 denotes ST and T wave abnormality, and value 2 denotes probable or definite left ventricular hypertrophy which shows the enlargement or thickening of walls of heart's main pumping chamber.

h.Maximum Heart Rate Achieved: The maximum heart rate of the person is measured while performing rigorous activities like running, biking, etc. The maximum heart rate dangerous for a person depends on their age.

i.Exercise-Induced Angina: A chest discomfort during activity is exercise-induced angina. Value is either 1 or 0 depending on whether a person has exercise-induced angina or not respectively.

j.ST depression induced by exercise relative to rest: The ST segment lying very low below baseline in electrocardiogram during exercise to the ratio of during rest.

k.The slope of the pick exercise ST segment: It can be observed from electrocardiograph results. i.Value 1-Upsloping ii.Value 2- Flat iii.Value 3-Downsloping

l.The number of major vessels colored by fluoroscopy: Fluoroscopy is an X-ray movie in which the body part and its motion can be seen clearly.

m.Thal: Thal or Thalassemia is an abnormal production of hemoglobin. i.Value 3-Normal ii.Value 6-Fixed defect iii.Value 7-Reversible defect

In [47]:
```
# data analysis, splitting and wrangling
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# visualization
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
import itertools
from sklearn.model_selection import  cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
# column names in accordance with feature information
col_names = ['age','sex','chest_pain','blood_pressure','serum_cholestoral','fasting_bl
             'max_heart_rate','induced_angina','ST_depression','slope','no_of_vessels

# read the file
df = pd.read_csv("processed.cleveland.data.csv", names=col_names, header=None, na_valu

print("Number of records: {}\nNumber of variables: {}".format(df.shape[0], df.shape[1]

# display the first 5 lines
df.head()
```

```
Number of records: 303
Number of variables: 14
```

Out[48]:

	age	sex	chest_pain	blood_pressure	serum_cholestoral	\
0	63.0	1.0	1.0	145.0	233.0	
1	67.0	1.0	4.0	160.0	286.0	
2	67.0	1.0	4.0	120.0	229.0	
3	37.0	1.0	3.0	130.0	250.0	
4	41.0	0.0	2.0	130.0	204.0	

```
   fasting_blood_sugar  electrocardiographic  max_heart_rate  induced_angina  \
0                  1.0                   2.0           150.0             0.0
1                  0.0                   2.0           108.0             1.0
2                  0.0                   2.0           129.0             1.0
3                  0.0                   0.0           187.0             0.0
4                  0.0                   2.0           172.0             0.0

   ST_depression  slope  no_of_vessels  thal  diagnosis
0            2.3    3.0            0.0   6.0          0
1            1.5    2.0            3.0   3.0          2
2            2.6    2.0            2.0   7.0          1
3            3.5    3.0            0.0   3.0          0
4            1.4    1.0            0.0   3.0          0
```

In [49]:
```
# extract numeric columns and find categorical ones
numeric_columns = ['serum_cholestoral', 'max_heart_rate', 'age', 'blood_pressure', 'S
categorical_columns = [c for c in df.columns if c not in numeric_columns]
print(categorical_columns)
```

```
['sex', 'chest_pain', 'fasting_blood_sugar', 'electrocardiographic', 'induced_angina', 'slope'
```

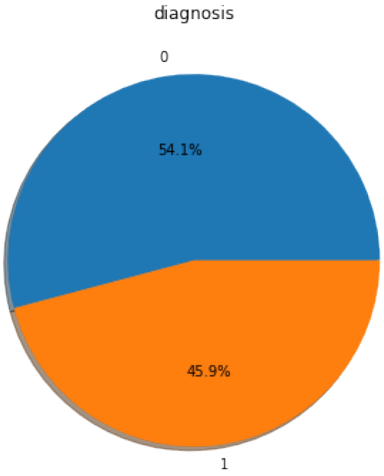
In [50]:
```
# count values of explained variable
df.diagnosis.value_counts()
```

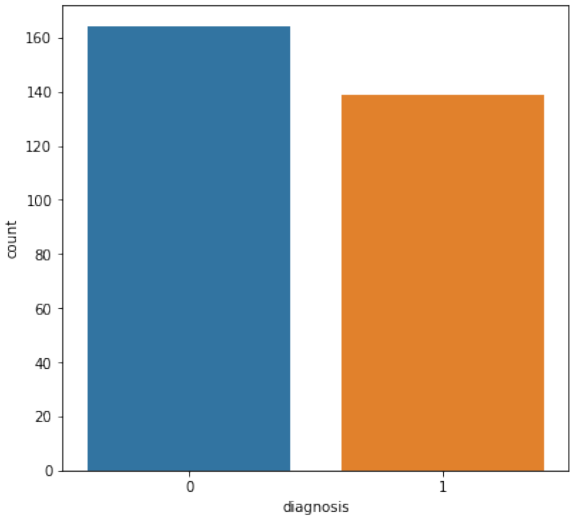
Out[50]:
```
0    164
1     55
2     36
3     35
4     13
Name: diagnosis, dtype: int64
```

In [51]:
```
# create a boolean vector and map it with corresponding values (True=1, False=0)
df.diagnosis = (df.diagnosis != 0).astype(int)
df.diagnosis.value_counts()
```

Out[51]:
```
0    164
1    139
Name: diagnosis, dtype: int64
```

In [52]:
```
# create two plots side by side
f, ax = plt.subplots(1,2,figsize=(14,6))
df['diagnosis'].value_counts().plot.pie(autopct='%1.1f%%', ax=ax[0], shadow=True)
ax[0].set_title('diagnosis')
ax[0].set_ylabel('')
sns.countplot('diagnosis', data=df, ax=ax[1])
plt.show()
```





Now the distribution of target value is almost equal, so using standard metrics in further machine learning modelling like accuracy and AUC is justified.

0.0.2 Numeric features

There are 5 numeric columns, so let's take care of them first. Outliers occurrence in the dataset may be a result of wrong input and create undesired noise, thus our role is to evaluate their substance. A data point is considered as an outlier when it falls outside 3 standard deviations.

```
In [53]: # view of descriptive statistics
         df[numeric_columns].describe()
```

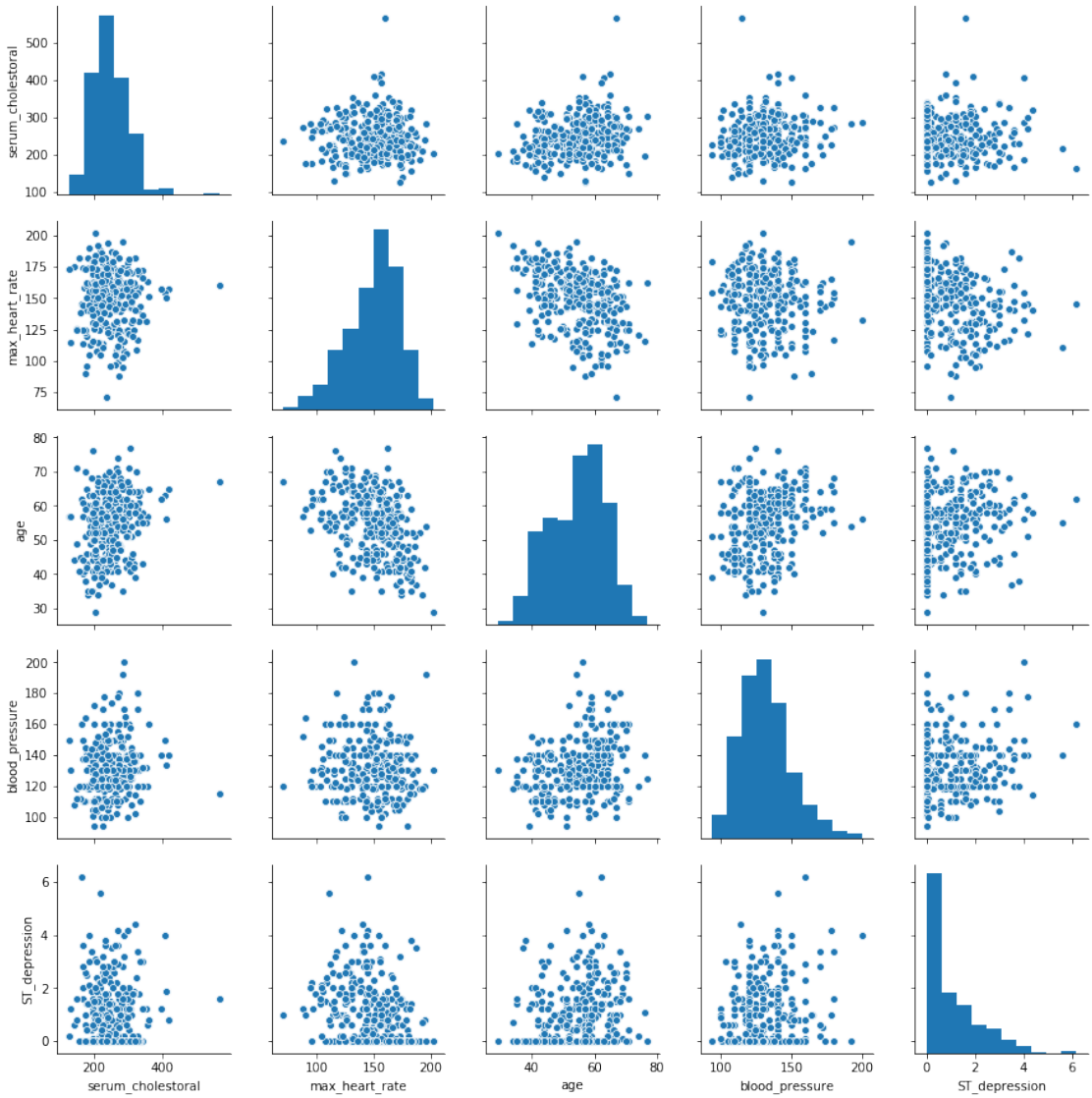
```
Out[53]:        serum_cholestoral  max_heart_rate         age  blood_pressure  \
         count         303.000000      303.000000  303.000000      303.000000
         mean          246.693069      149.607261   54.438944      131.689769
         std            51.776918       22.875003    9.038662       17.599748
         min           126.000000       71.000000   29.000000       94.000000
         25%           211.000000      133.500000   48.000000      120.000000
         50%           241.000000      153.000000   56.000000      130.000000
         75%           275.000000      166.000000   61.000000      140.000000
         max           564.000000      202.000000   77.000000      200.000000

                ST_depression
         count     303.000000
         mean        1.039604
         std         1.161075
         min         0.000000
         25%         0.000000
         50%         0.800000
         75%         1.600000
         max         6.200000
```

All extreme (min/max) values could occur in a real clinical scenario, hence the decision to keep them as they are.

We can gain some intuition about relationships amongst numeric features by plotting each pair in a scattered form. To do this efficiently, pairplot method from Seaborn library comes in handy.

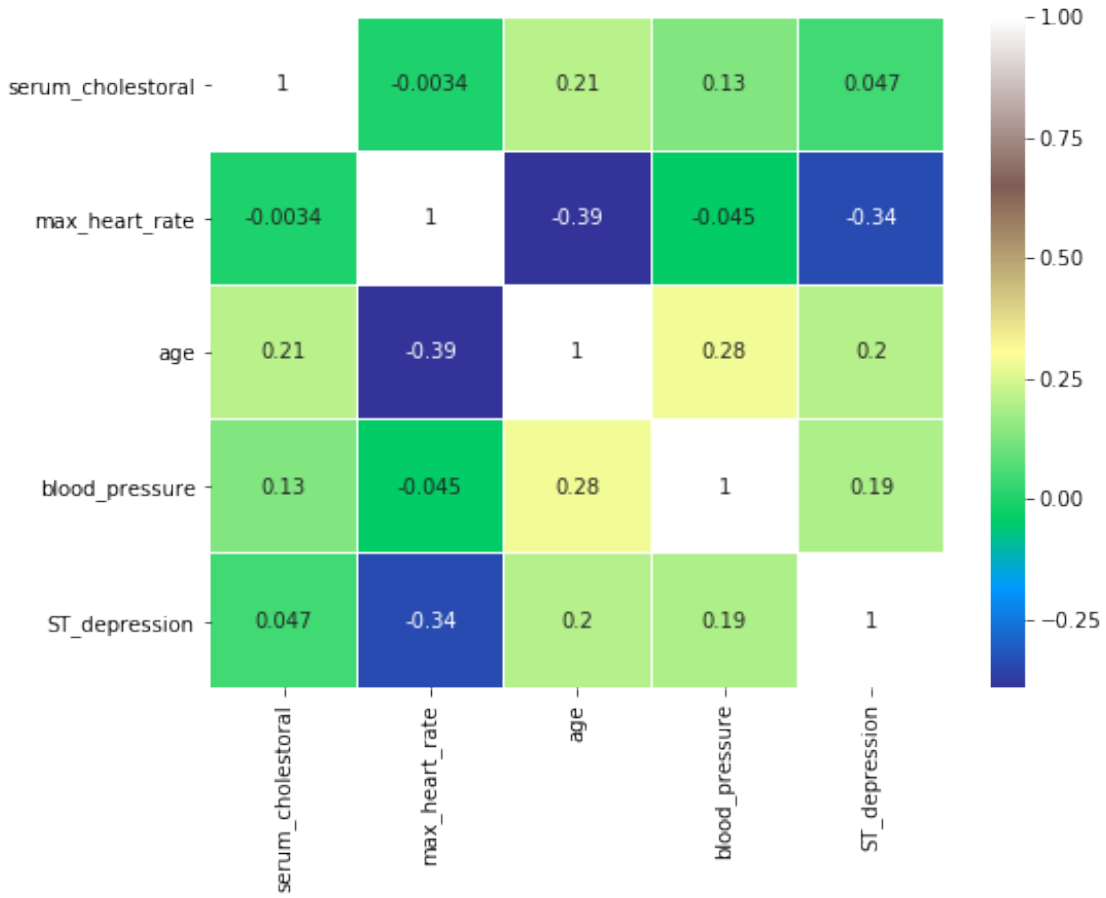
```
In [54]: # create a pairplot
         sns.pairplot(df[numeric_columns])
         plt.show()
```



Seeing above plots, I infer that none of displayed value pairs is having an explicitly high correlation, so there is no necessity to ditch any feature at this stage. I also notice a negative correlation between 'age' and 'max_heart_rate' and positive correlation between 'age' and 'blood_pressure', what is intuitive.

A correlation matrix will make us sure whether the above is correct.

```
In [55]: # create a correlation heatmap
         sns.heatmap(df[numeric_columns].corr(),annot=True, cmap='terrain', linewidths=0.1)
         fig=plt.gcf()
         fig.set_size_inches(8,6)
         plt.show()
```

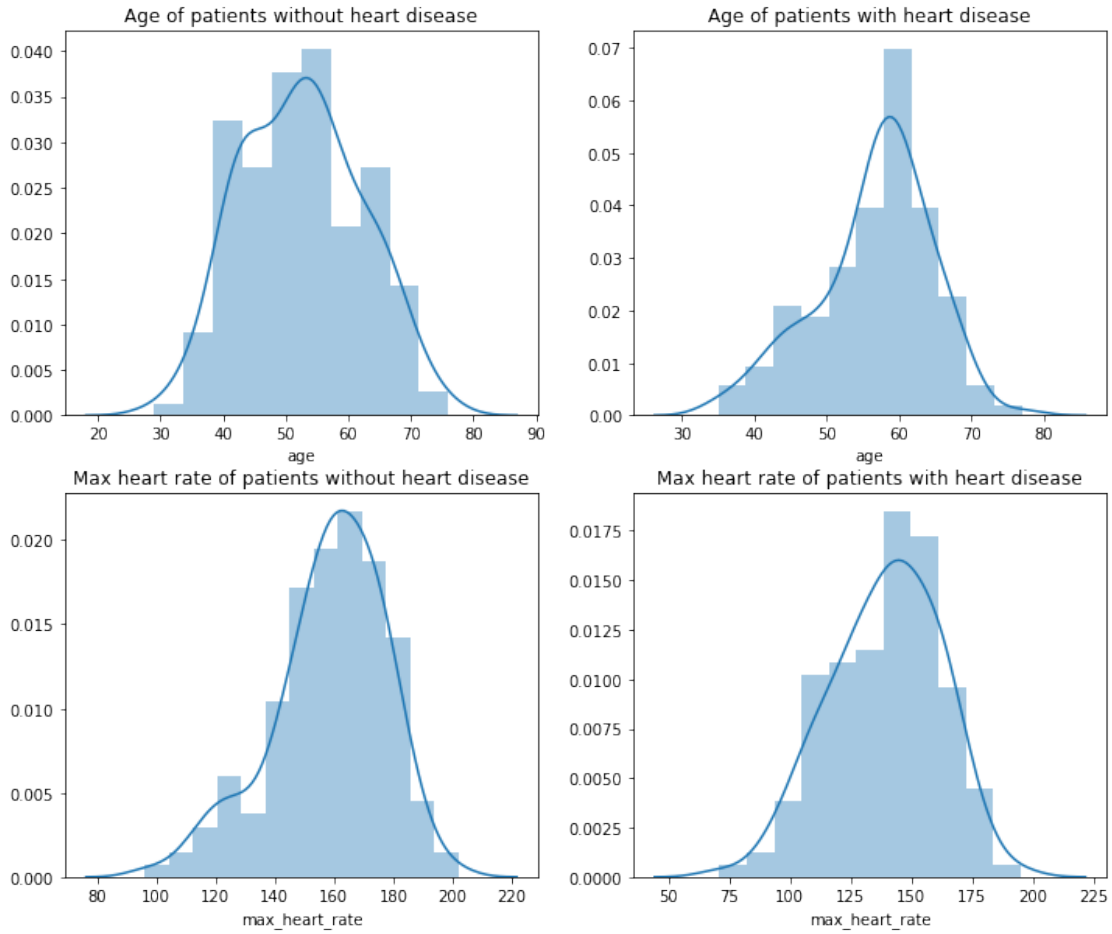


Apart from two aforementioned relations, there is one more important dependency: 'max_heart_rate' and 'ST_depression'. The conclusion comes up that both features, 'age' and 'max_heart_rate', will play an important role in predicting heart disease. Let's take a look at their distributions.

```
In [56]: # create four distplots
         plt.figure(figsize=(12,10))
         plt.subplot(221)
         sns.distplot(df[df['diagnosis']==0].age)
         plt.title('Age of patients without heart disease')
         plt.subplot(222)
         sns.distplot(df[df['diagnosis']==1].age)
         plt.title('Age of patients with heart disease')
         plt.subplot(223)
```

```
sns.distplot(df[df['diagnosis']==0].max_heart_rate)
plt.title('Max heart rate of patients without heart disease')
plt.subplot(224)
sns.distplot(df[df['diagnosis']==1].max_heart_rate)
plt.title('Max heart rate of patients with heart disease')
plt.show()
```

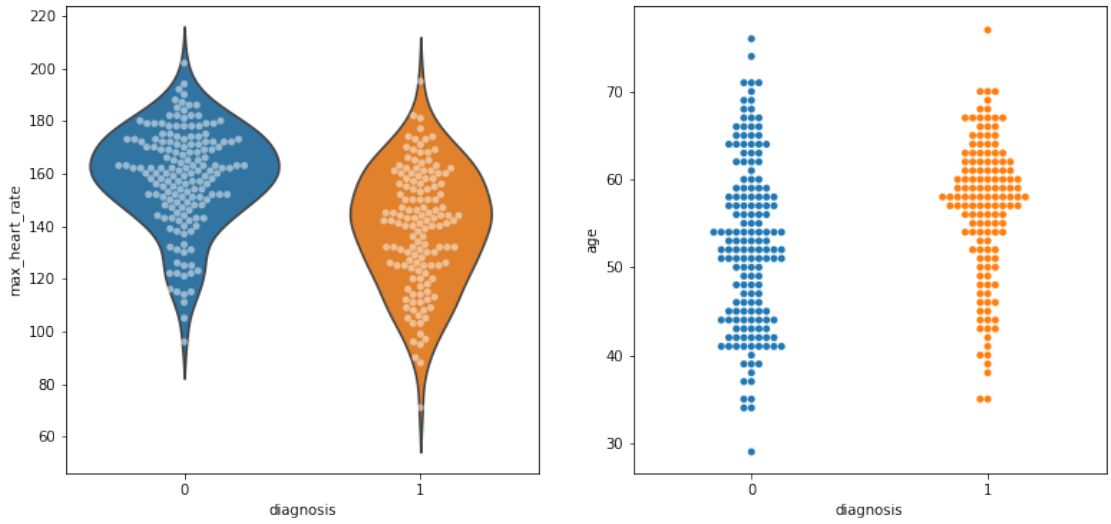
```
/anaconda3/lib/python3.7/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a non-tu
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



Age distribution of healthy patients is much wider than ill people. The latter are at the highest risk in their sixties. Max heart rate distribution does not differ as much, but the risk peaks when max_heart_rate value is between 150 and 170. Higher values are more common for well patients. Below graphs will provide us with another perspective.

```
In [57]: # create swarmplot inside the violinplot
         plt.figure(figsize=(13,6))
         plt.subplot(121)
```

```
sns.violinplot(x="diagnosis", y="max_heart_rate", data=df, inner=None)
sns.swarmplot(x='diagnosis', y='max_heart_rate', data=df, color='w', alpha=0.5)
plt.subplot(122)
sns.swarmplot(x='diagnosis', y='age', data=df)
plt.show()
```



0.0.3 Categorical features

Let's take a closer look at categorical variables and see how they impact our target.

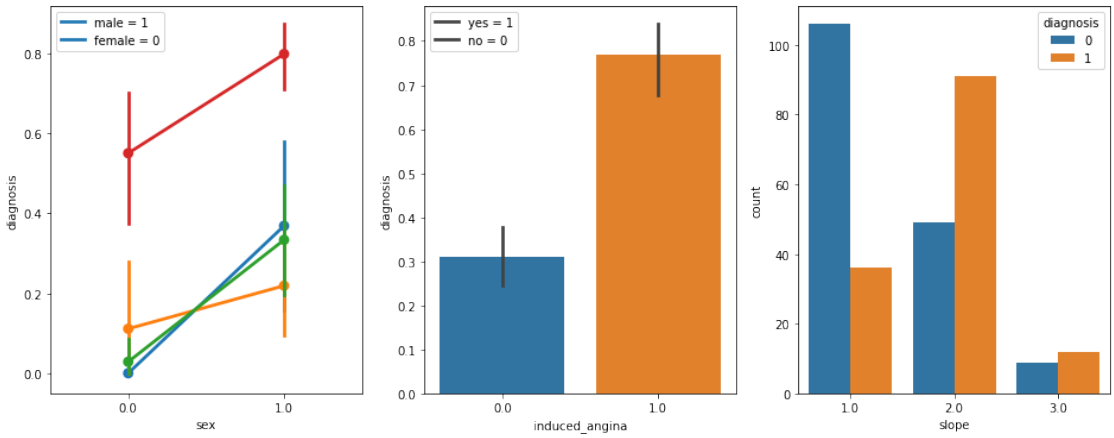
```
In [58]: # count ill vs healthy people grouped by sex
         df.groupby(['sex','diagnosis'])['diagnosis'].count()

Out[58]: sex  diagnosis
         0.0  0            72
              1            25
         1.0  0            92
              1           114
         Name: diagnosis, dtype: int64

In [59]: # average number of diagnosed people grouped by number of blood vessels detected by f
         df[['no_of_vessels','diagnosis']].groupby('no_of_vessels').mean()

Out[59]:                diagnosis
         no_of_vessels
         0.0             0.261364
         1.0             0.676923
         2.0             0.815789
         3.0             0.850000
```

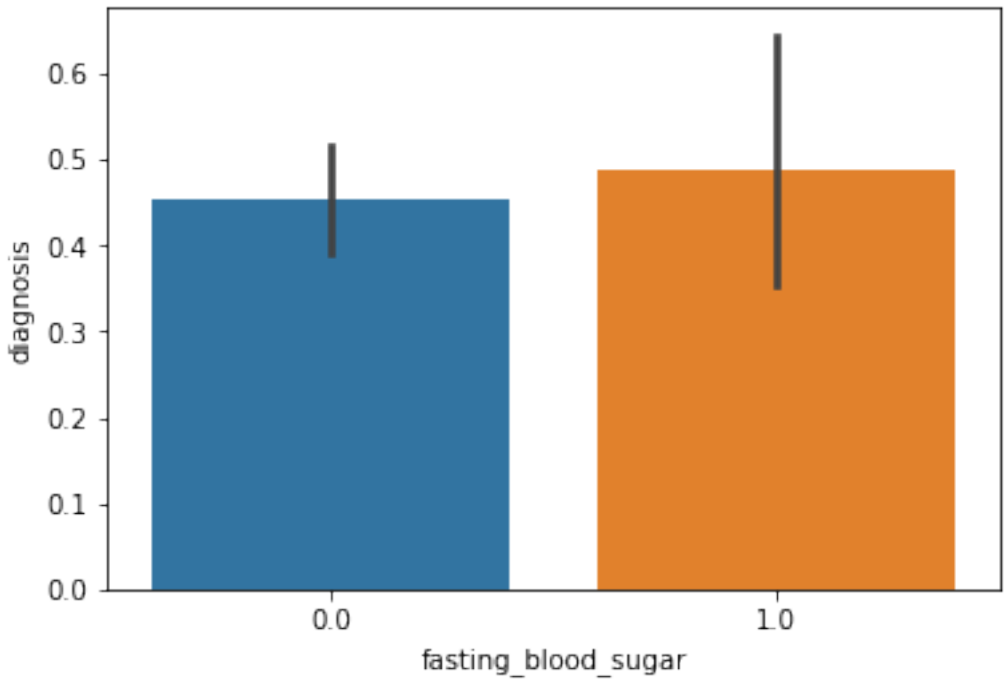
```
In [60]: # create pairplot and two barplots
         plt.figure(figsize=(16,6))
         plt.subplot(131)
         sns.pointplot(x="sex", y="diagnosis", hue='chest_pain', data=df)
         plt.legend(['male = 1', 'female = 0'])
         plt.subplot(132)
         sns.barplot(x="induced_angina", y="diagnosis", data=df)
         plt.legend(['yes = 1', 'no = 0'])
         plt.subplot(133)
         sns.countplot(x="slope", hue='diagnosis', data=df)
         plt.show()
```



Observations: Men are much more prone to get a heart disease than women. The higher number of vessels detected through fluoroscopy, the higher risk of disease. While soft chest pain may be a bad symptom of approaching problems with heart (especially in case of men), strong pain is a serious warning! Risk of getting heart disease might be even 3x higher for someone who experienced exercise-induced angina. The flat slope (value=2) and downslope (value=3) of the peak exercise indicates a high risk of getting disease

```
In [61]: # create a barplot
         sns.barplot(x="fasting_blood_sugar", y="diagnosis", data=df)

Out[61]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e18ec18>
```



Almost even distribution suggests 'fasting blood sugar' to be a very weak feature for our prediction, therefore it could be excluded from our model. It is not likely that leaving this variable will improve our model accuracy, yet it shouldn't make it worse too. I decide to keep this variable as it is and confirm my hypothesis through checking feature importance of a few models.

0.0.4 Data Preparation

In order to make our dataset compatible with machine learning algorithms contained in Sci-kit Learn library, first of all, we need to handle all missing data.

There are many options we could consider when replacing a missing value, for example:

A constant value that has meaning within the domain, such as 0, distinct from all other values A value from another randomly selected record A mean, median or mode value for the column A value estimated by another predictive model

0.0.5 3.Data Preparation

In order to make our dataset compatible with machine learning algorithms contained in Sci-kit Learn library, first of all, we need to handle all missing data.

There are many options we could consider when replacing a missing value, for example:

A constant value that has meaning within the domain, such as 0, distinct from all other values A value from another randomly selected record A mean, median or mode value for the column A value estimated by another predictive model

0.0.6 Data Preparation

In order to make our dataset compatible with machine learning algorithms contained in Sci-kit Learn library, first of all, we need to handle all missing data.

There are many options we could consider when replacing a missing value, for example:

A constant value that has meaning within the domain, such as 0, distinct from all other values A value from another randomly selected record A mean, median or mode value for the column A value estimated by another predictive model

```
In [62]: # show columns having missing values
         df.isnull().sum()
```

```
Out[62]: age                    0
         sex                    0
         chest_pain             0
         blood_pressure         0
         serum_cholestoral      0
         fasting_blood_sugar    0
         electrocardiographic   0
         max_heart_rate         0
         induced_angina         0
         ST_depression          0
         slope                  0
         no_of_vessels          4
         thal                   2
         diagnosis              0
         dtype: int64
```

```
In [63]: # fill missing values with mode
         df['no_of_vessels'].fillna(df['no_of_vessels'].mode()[0], inplace=True)
         df['thal'].fillna(df['thal'].mode()[0], inplace=True)
```

Having clean data, a label can be separated from the data frame. It is also a good moment to split our data train and test sets. I will allocate 30% of the entire data to test set, which is typically considered as a standard split for this size of dataset.

```
In [64]: # extract the target variable
         X, y = df.iloc[:, :-1], df.iloc[:, -1]
         print(X.shape)
         print(y.shape)
```

```
(303, 13)
(303,)
```

```
In [65]: # split the data
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
         print ("train_set_x shape: " + str(X_train.shape))
         print ("train_set_y shape: " + str(y_train.shape))
         print ("test_set_x shape: " + str(X_test.shape))
         print ("test_set_y shape: " + str(y_test.shape))
```

```
train_set_x shape: (212, 13)
train_set_y shape: (212,)
test_set_x shape: (91, 13)
test_set_y shape: (91,)
```

Data needs to be normalized or standardized before applying to machine learning algorithms. Standardization scales the data and gives information on how many standard deviations the data is placed from its mean value. Effectively, the mean of the data (ţ) is 0 and the standard deviation () is 1.

```
In [66]: # scale feature matrices
         scaler = StandardScaler()
         X_train = scaler.fit_transform(X_train)
         X_test = scaler.transform(X_test)
```

0.0.7 Modelling and predicting with Machine Learning

The main goal of the entire project is to predict heart disease occurrence with the highest accuracy. In order to achieve this, we will test several classification algorithms. This section includes all results obtained from the study and introduces the best performer according to accuracy metric. I have chosen several algorithms typical for solving supervised learning problems throughout classification methods.

First of all, let's equip ourselves with a handy tool that benefits from the cohesion of SciKit Learn library and formulate a general function for training our models. The reason for displaying accuracy on both, train and test sets, is to allow us to evaluate whether the model overfits or underfits the data (so-called bias/variance tradeoff).

```
In [67]: def train_model(X_train, y_train, X_test, y_test, classifier, **kwargs):

             """
             Fit the chosen model and print out the score.

             """

             # instantiate model
             model = classifier(**kwargs)

             # train model
             model.fit(X_train,y_train)

             # check accuracy and print out the results
             fit_accuracy = model.score(X_train, y_train)
             test_accuracy = model.score(X_test, y_test)

             print(f"Train accuracy: {fit_accuracy:0.2%}")
             print(f"Test accuracy: {test_accuracy:0.2%}")

             return model
```

0.0.8 K-Nearest Neighbours (KNN)

K-Nearest Neighbors algorithm is a non-parametric method used for classification and regression. The principle behind nearest neighbour methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these.

```
In [68]: # KNN
         model = train_model(X_train, y_train, X_test, y_test, KNeighborsClassifier)
```

```
Train accuracy: 88.21%
Test accuracy: 86.81%
```

Despite its simplicity, the result is very promising. Let's see if KNN can perform even better by trying different 'n_neighbours' inputs.

```
In [69]: # Seek optimal 'n_neighbours' parameter
         for i in range(1,10):
             print("n_neigbors = "+str(i))
             train_model(X_train, y_train, X_test, y_test, KNeighborsClassifier, n_neighbors=i)
```

```
n_neigbors = 1
Train accuracy: 100.00%
Test accuracy: 74.73%
n_neigbors = 2
Train accuracy: 87.74%
Test accuracy: 79.12%
n_neigbors = 3
Train accuracy: 90.57%
Test accuracy: 83.52%
n_neigbors = 4
Train accuracy: 87.74%
Test accuracy: 84.62%
n_neigbors = 5
Train accuracy: 88.21%
Test accuracy: 86.81%
n_neigbors = 6
Train accuracy: 85.38%
Test accuracy: 86.81%
n_neigbors = 7
Train accuracy: 87.26%
Test accuracy: 86.81%
n_neigbors = 8
Train accuracy: 85.38%
Test accuracy: 85.71%
n_neigbors = 9
Train accuracy: 86.32%
Test accuracy: 85.71%
```

It turns out that default value of n_neighbours (5) is optimal.

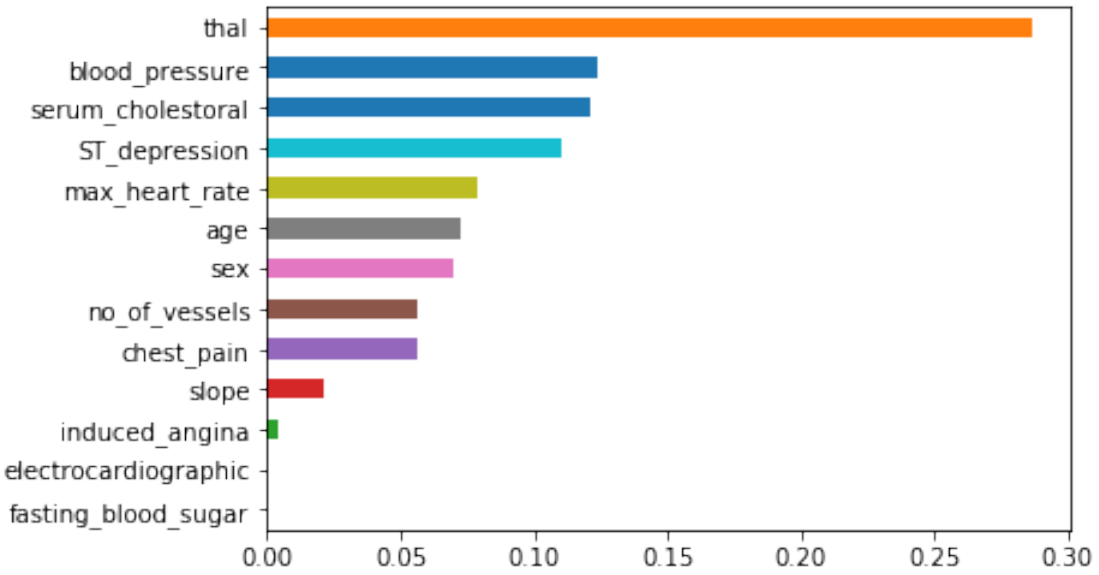
Decision Trees DT algorithm creates a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. It is simple to understand and interpret and it's possible to visualize how important a particular feature was for our tree.

```
In [70]: # Decision Tree
         model = train_model(X_train, y_train, X_test, y_test, DecisionTreeClassifier, random_s

         # plot feature importances
         pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh
```

```
Train accuracy: 100.00%
Test accuracy: 75.82%
```

```
Out[70]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e334438>
```



Variable 'thal' turns out to be a significantly important feature. Remember my hypothesis that 'fasting_blood_sugar" is a very weak feature? Above graph confirms this clearly. Decision tree model learns the train set perfectly, and at the same time is entirely overfitting the data, what results in poor prediction. Other values of 'max_depth' parameter need to be tried out.

```
In [71]: # Check optimal 'max_depth' parameter
         for i in range(1,8):
             print("max_depth = "+str(i))
             train_model(X_train, y_train, X_test, y_test, DecisionTreeClassifier, max_depth=i
```

```
max_depth = 1
Train accuracy: 76.89%
```

```
Test accuracy: 74.73%
max_depth = 2
Train accuracy: 78.30%
Test accuracy: 72.53%
max_depth = 3
Train accuracy: 87.74%
Test accuracy: 76.92%
max_depth = 4
Train accuracy: 91.98%
Test accuracy: 78.02%
max_depth = 5
Train accuracy: 94.81%
Test accuracy: 78.02%
max_depth = 6
Train accuracy: 97.17%
Test accuracy: 79.12%
max_depth = 7
Train accuracy: 97.64%
Test accuracy: 75.82%
```

With max_depth set as 6, the score went to almost 80%. By now, KNN outperforms Decision Tree.

0.0.9 Logistic Regression

Logistic regression is a basic technique in statistical analysis that attempts to predict a data value based on prior observations. A logistic regression algorithm looks at the relationship between a dependent variable and one or more dependent variables.

```
In [72]: # Logistic Regression
         model = train_model(X_train, y_train, X_test, y_test, LogisticRegression)
```

```
Train accuracy: 85.85%
Test accuracy: 85.71%
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
```

Negligible difference between train and test score tells us as that model performs at the optimal level. Although the result itself is slightly lower than KNN, yet is still satisfactory.

0.0.10 Gaussian Naive Bayes

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

```
In [73]: #Gaussian Naive Bayes
         model = train_model(X_train, y_train, X_test, y_test, GaussianNB)
```

```
Train accuracy: 85.38%
Test accuracy: 86.81%
```

This model produced the same result as leading KNN algorithm. While it slightly underfits the data, this model doesn't offer any hyperparameters for tuning and improve overall performance.

0.0.11 Support Vector Machines

Support Vector Machines are perhaps one of the most popular machine learning algorithms. They are the go-to method for a high-performing algorithm with a little tuning. At first, let's try it on default settings.

```
In [74]: # Support Vector Machines
         model = train_model(X_train, y_train, X_test, y_test, SVC)
```

```
Train accuracy: 92.92%
Test accuracy: 82.42%
```

The above numbers are not remarkable by any means. I will adjust two parameters, "C", and 'kernel' to take full advantage of SVM power.

```
In [75]: # tuned SVM
         model = train_model(X_train, y_train, X_test, y_test, SVC, C=0.05, kernel='linear')
```

```
Train accuracy: 84.91%
Test accuracy: 87.91%
```

With impressive accuracy of almost 88%, Support Vector Machines are taking the lead!

0.0.12 Neural Network

Neural Network are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input

```
In [76]: model = train_model(X_train, y_train, X_test, y_test, MLPClassifier)
```

```
Train accuracy: 89.15%
Test accuracy: 84.62%
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)
```

0.0.13 Random Forests

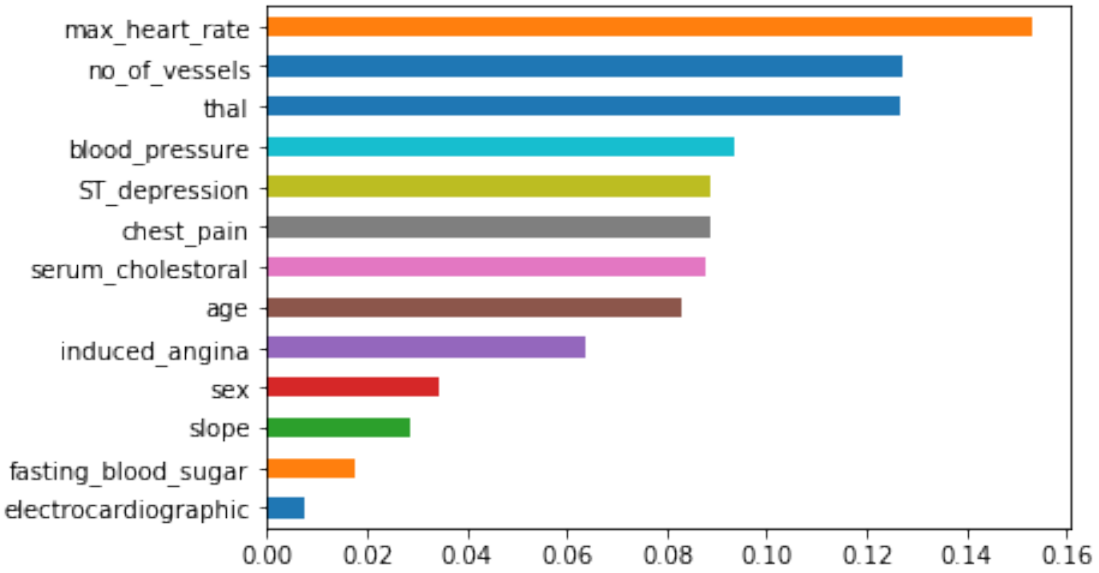
Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

```
In [77]: # Random Forests
         model = train_model(X_train, y_train, X_test, y_test, RandomForestClassifier, random_s
         pd.Series(model.feature_importances_,X.columns).sort_values(ascending=True).plot.barh
```

```
Train accuracy: 99.06%
Test accuracy: 83.52%
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The defau
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e4689b0>
```



This result did not meet our expectations. Altering 'n_estimators' parameter will surely pull out more of this strong algorithm.

```
In [78]: # tuned Random Forests
         model = train_model(X_train, y_train, X_test, y_test, RandomForestClassifier, n_estima
```

```
Train accuracy: 100.00%
Test accuracy: 87.91%
```

While it is typical for Random Forests to perfectly learn and fit into training data, the test accuracy achieved outstanding 89%!

0.0.14 Conclusion

The goal of the project was to compare different machine learning algorithms and predict if a certain person, given various personal characteristics and symptoms, will get heart disease or not. Here are the final results.

In [79]:
```
# initialize an empty list
accuracy = []

# list of algorithms names
classifiers = ['KNN', 'Decision Trees', 'Logistic Regression', 'Naive Bayes', 'SVM','

# list of algorithms with parameters
models = [KNeighborsClassifier(n_neighbors=5), DecisionTreeClassifier(max_depth=6, ra
        GaussianNB(), SVC(C=0.05, kernel='linear'), MLPClassifier(),RandomForestClass

# loop through algorithms and append the score into the list
for i in models:
    model = i
    model.fit(X_train, y_train)
    score = model.score(X_test, y_test)
    accuracy.append(score)
```

```
/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/anaconda3/lib/python3.7/site-packages/sklearn/neural_network/multilayer_perceptron.py:562: Co
  % self.max_iter, ConvergenceWarning)
```

In [93]:
```
# create a dataframe from accuracy results
summary = pd.DataFrame({'ALGORITHM':'    ','ACCURACY':accuracy}, index=classifiers)
print(summary)
```

```
                    ALGORITHM  ACCURACY
KNN                           0.868132
Decision Trees                0.791209
Logistic Regression           0.857143
Naive Bayes                   0.868132
SVM                           0.879121
Neural Network                0.835165
Random Forests                0.879121
```