

Photo by [rawpixel](https://unsplash.com/@rawpixel?utm_source=medium&utm_medium=referral" \t "_blank) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Machine Learning is used across many spheres around the world. The healthcare industry is no exception. Machine Learning can play an essential role in predicting presence/absence of Locomotor disorders, Heart diseases and more. Such information, if predicted well in advance, can provide important insights to doctors who can then adapt their diagnosis and treatment per patient basis.

In this article, I’ll discuss a project where I worked on predicting potential Heart Diseases in people using Machine Learning algorithms. The algorithms included K Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier and Random Forest Classifier. The dataset has been taken from [Kaggle](https://www.kaggle.com/ronitf/heart-disease-uci" \t "_blank). My complete project is available at [Heart Disease Prediction](https://github.com/kb22/Heart-Disease-Prediction).

**Import libraries**

I imported several libraries for the project:

1. **numpy**: To work with arrays
2. **pandas**: To work with csv files and dataframes
3. **matplotlib**: To create charts using pyplot, define parameters using rcParams and color them with cm.rainbow
4. **warnings**: To ignore all warnings which might be showing up in the notebook due to past/future depreciation of a feature
5. **train\_test\_split**: To split the dataset into training and testing data
6. **StandardScaler**: To scale all the features, so that the Machine Learning model better adapts to the dataset

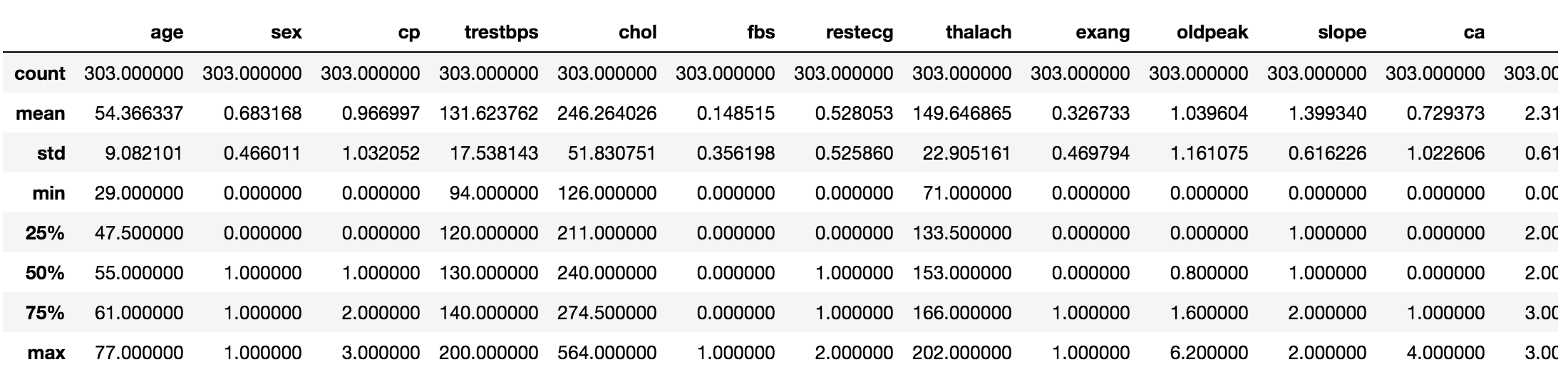
Next, I imported all the necessary Machine Learning algorithms.

**Import dataset**

After downloading the dataset from Kaggle, I saved it to my working directory with the name dataset.csv. Next, I used read\_csv() to read the dataset and save it to the dataset variable.

Before any analysis, I just wanted to take a look at the data. So, I used the info() method.

As you can see from the output above, there are a total of 13 features and 1 target variable. Also, there are no missing values so we don’t need to take care of any null values. Next, I used describe() method.



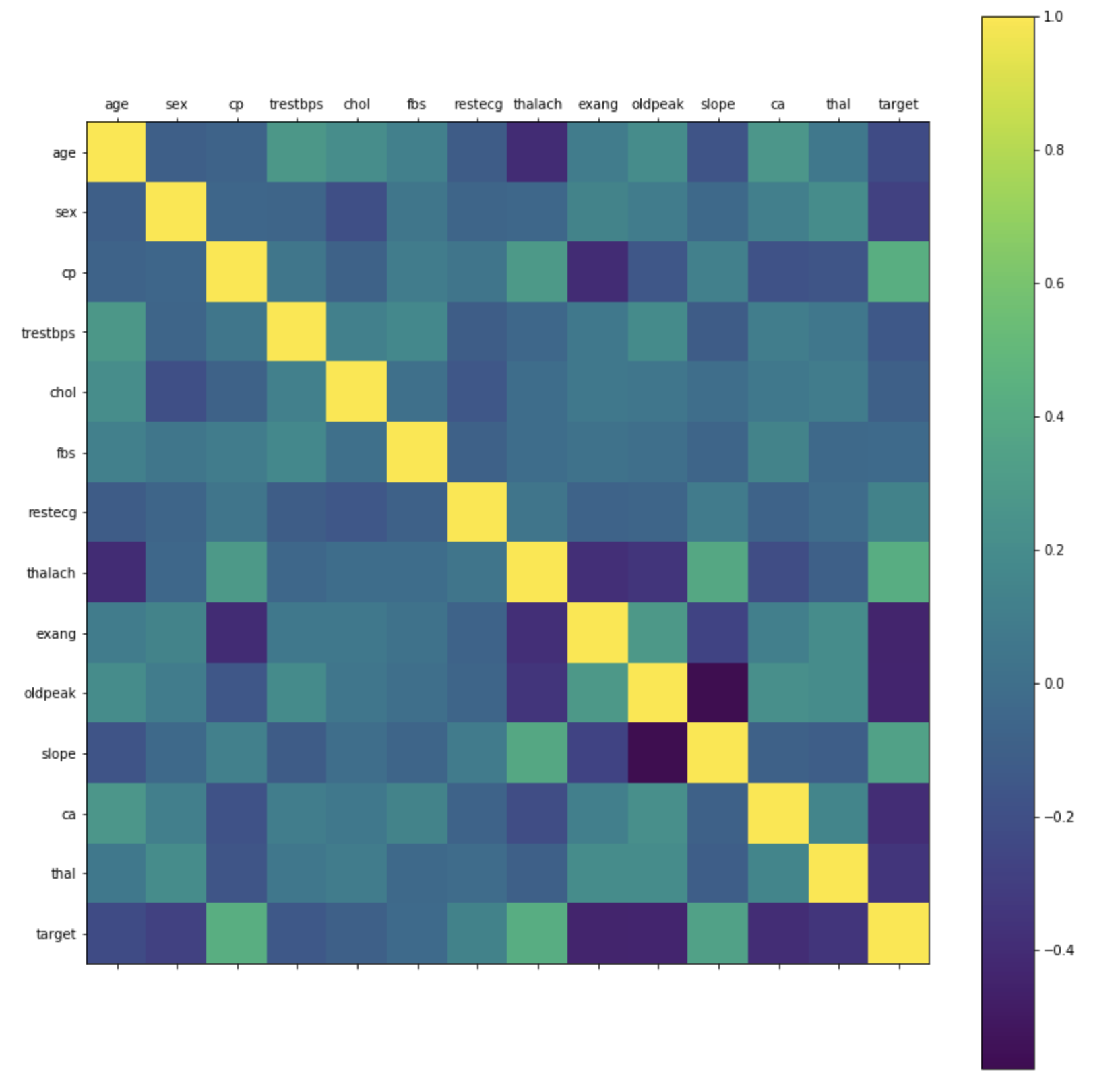
dataset.describe()

The method revealed that the range of each variable is different. The maximum value of age is 77 but for chol it is 564. Thus, feature scaling must be performed on the dataset.

**Understanding the data**

**Correlation Matrix**

To begin with, let’s see the correlation matrix of features and try to analyse it. The figure size is defined to 12 x 8 by using rcParams. Then, I used pyplot to show the correlation matrix. Using xticks and yticks, I’ve added names to the correlation matrix. colorbar() shows the colorbar for the matrix.

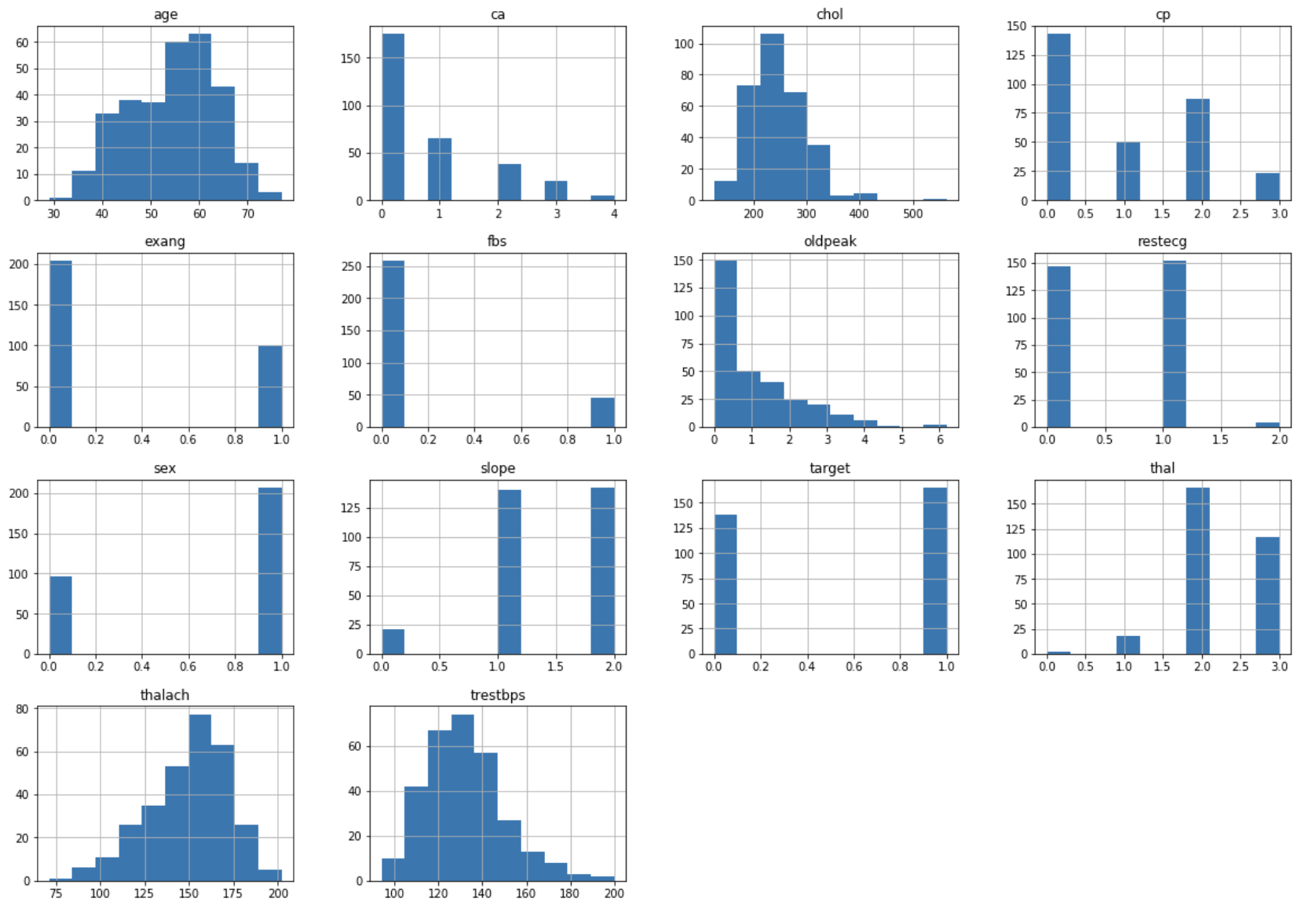


Correlation Matrix

It’s easy to see that there is no single feature that has a very high correlation with our target value. Also, some of the features have a negative correlation with the target value and some have positive.

**Histogram**

The best part about this type of plot is that it just takes a single command to draw the plots and it provides so much information in return. Just use dataset.hist().



dataset.hist()

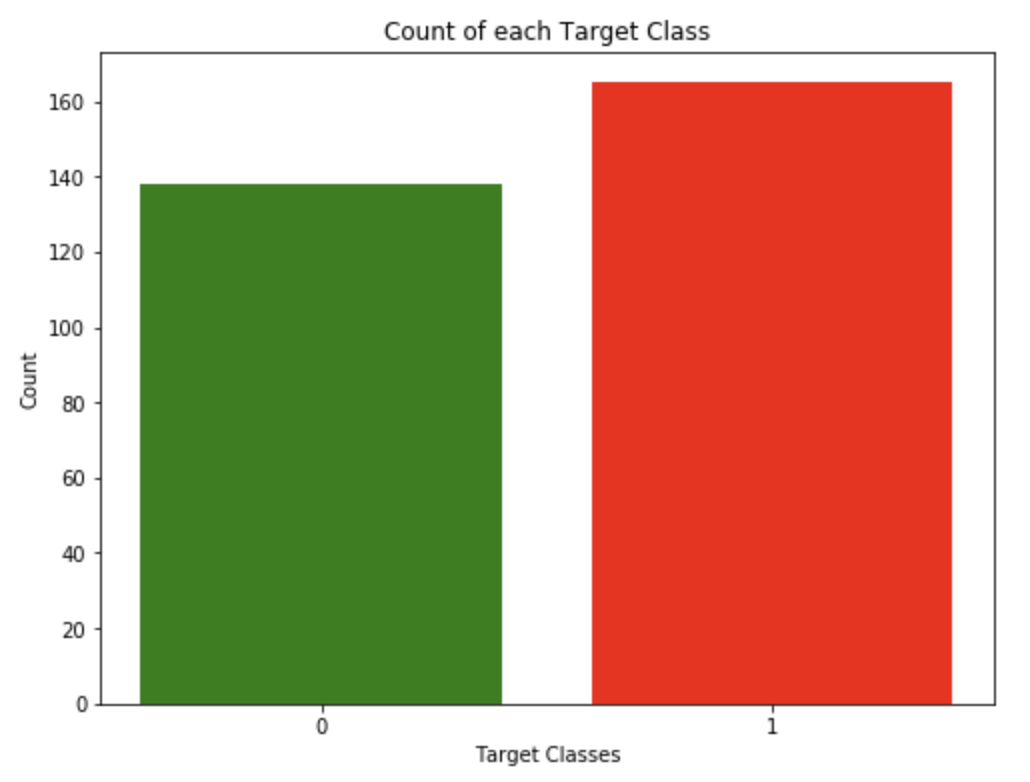
Let’s take a look at the plots. It shows how each feature and label is distributed along different ranges, which further confirms the need for scaling. Next, wherever you see discrete bars, it basically means that each of these is actually a categorical variable. We will need to handle these categorical variables before applying Machine Learning. Our target labels have two classes, 0 for no disease and 1 for disease.

**Bar Plot for Target Class**

It’s really essential that the dataset we are working on should be approximately balanced. An extremely imbalanced dataset can render the whole model training useless and thus, will be of no use. Let’s understand it with an example.

*Let’s say we have a dataset of 100 people with 99 non-patients and 1 patient. Without even training and learning anything, the model can always say that any new person would be a non-patient and have an accuracy of 99%. However, as we are more interested in identifying the 1 person who is a patient, we need balanced datasets so that our model actually learns.*

For x-axis I used the unique() values from the target column and then set their name using xticks. For y-axis, I used value\_count() to get the values for each class. I colored the bars as green and red.



From the plot, we can see that the classes are almost balanced and we are good to proceed with data processing.

**Data Processing**

To work with categorical variables, we should break each categorical column into dummy columns with 1s and 0s.

Let’s say we have a column Gender, with values 1 for Male and 0 for Female. It needs to be converted into two columns with the value 1 where the column would be true and 0 where it will be false. Take a look at the Gist below.

To get this done, we use the get\_dummies() method from pandas. Next, we need to scale the dataset for which we will use the StandardScaler. The fit\_transform() method of the scaler scales the data and we update the columns.

The dataset is now ready. We can begin with training our models.

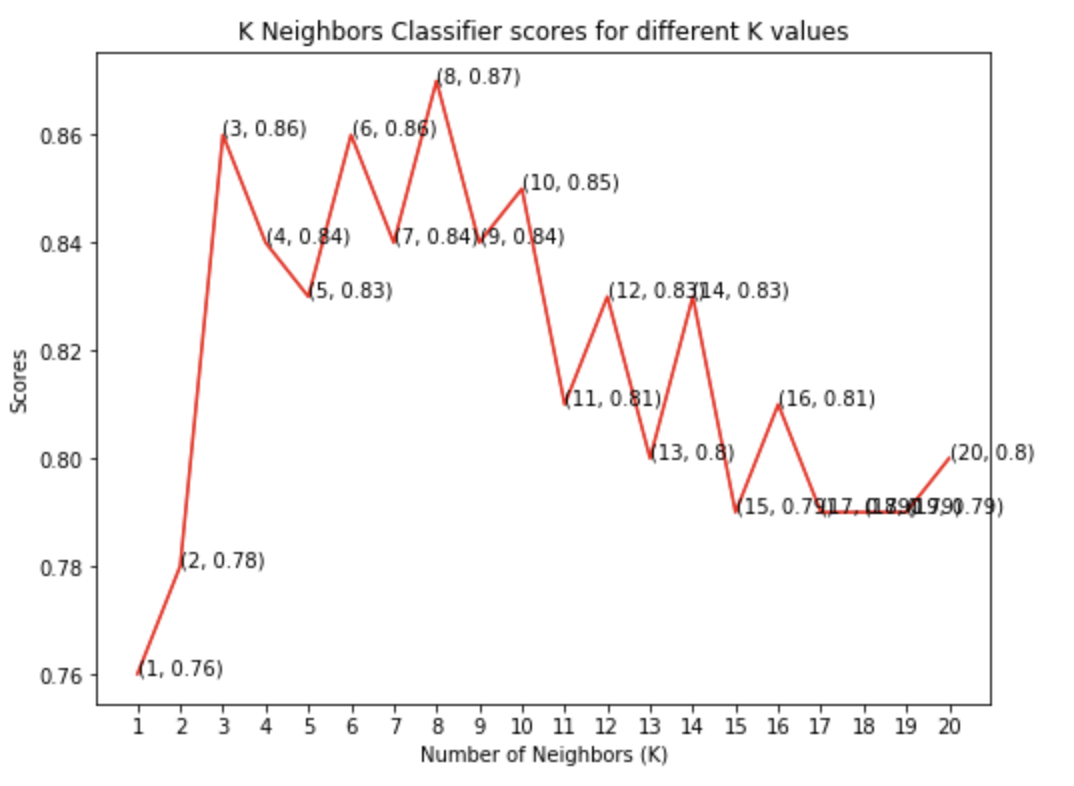
**Machine Learning**

In this project, I took 4 algorithms and varied their various parameters and compared the final models. I split the dataset into 67% training data and 33% testing data.

**K Neighbors Classifier**

This classifier looks for the classes of K nearest neighbors of a given data point and based on the majority class, it assigns a class to this data point. However, the number of neighbors can be varied. I varied them from 1 to 20 neighbors and calculated the test score in each case.

Then, I plot a line graph of the number of neighbors and the test score achieved in each case.

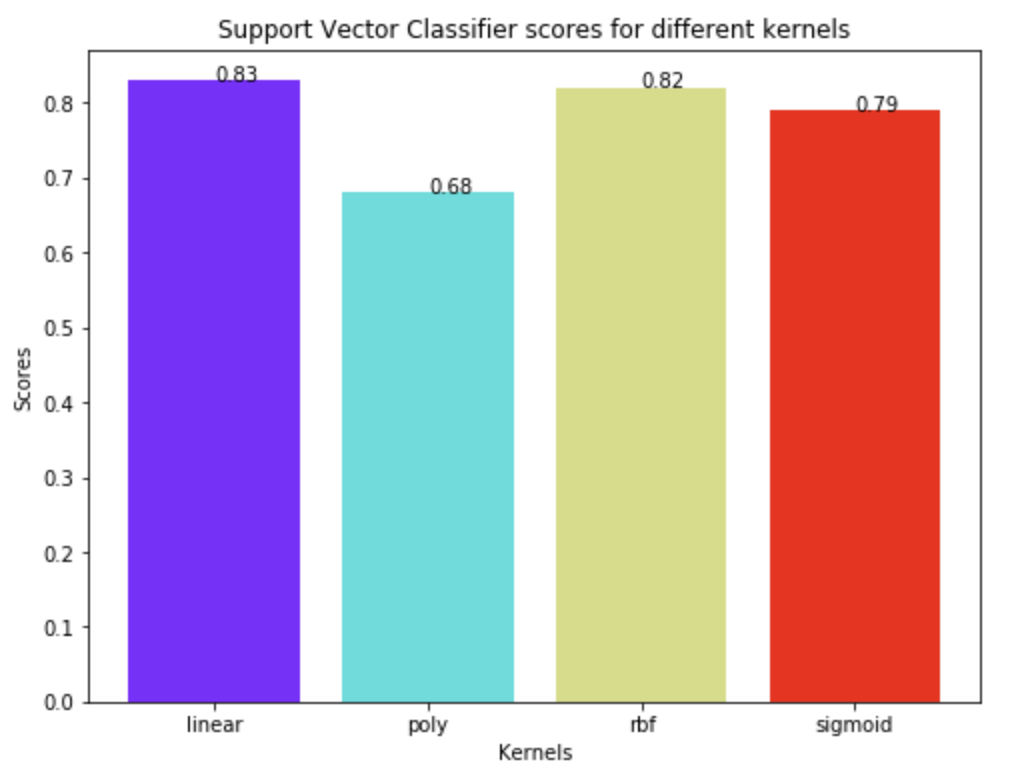


As you can see, we achieved the maximum score of 87% when the number of neighbors was chosen to be 8.

**Support Vector Classifier**

This classifier aims at forming a hyperplane that can separate the classes as much as possible by adjusting the distance between the data points and the hyperplane. There are several kernels based on which the hyperplane is decided. I tried four kernels namely, *linear*, *poly*, *rbf*, and *sigmoid*.

Once I had the scores for each, I used the rainbow method to select different colors for each bar and plot a bar graph of the scores achieved by each.

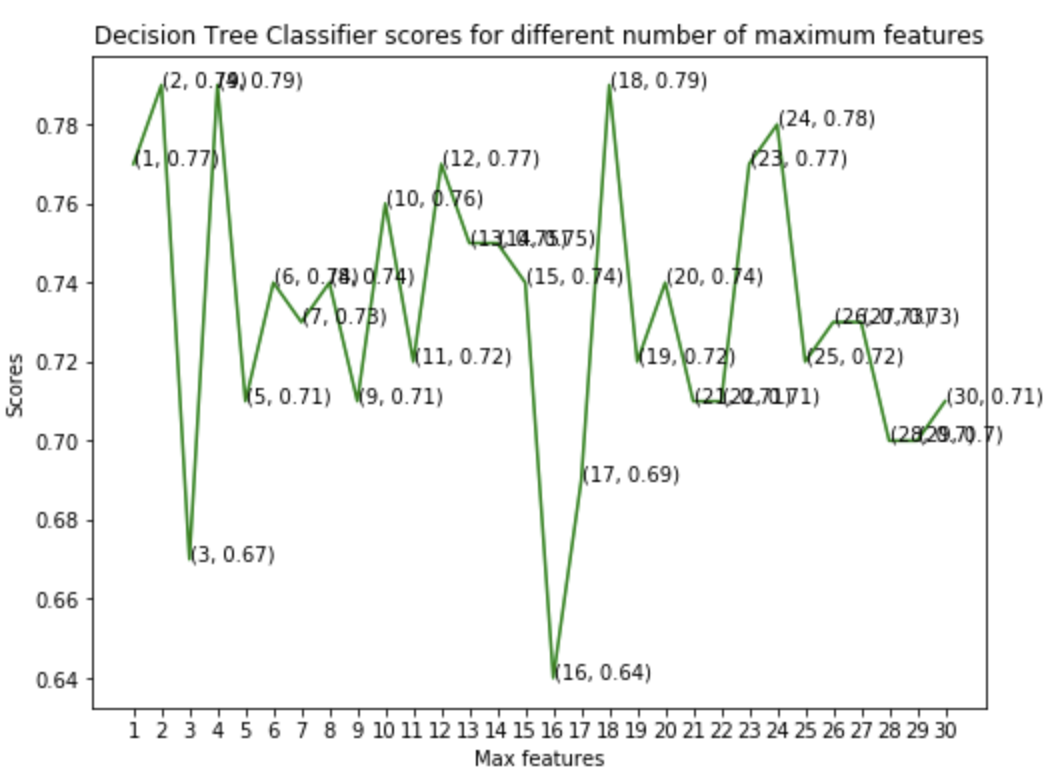


As can be seen from the plot above, the linear kernel performed the best for this dataset and achieved a score of 83%.

**Decision Tree Classifier**

This classifier creates a decision tree based on which, it assigns the class values to each data point. Here, we can vary the maximum number of features to be considered while creating the model. I range features from 1 to 30 (the total features in the dataset after dummy columns were added).

Once we have the scores, we can then plot a line graph and see the effect of the number of features on the model scores.

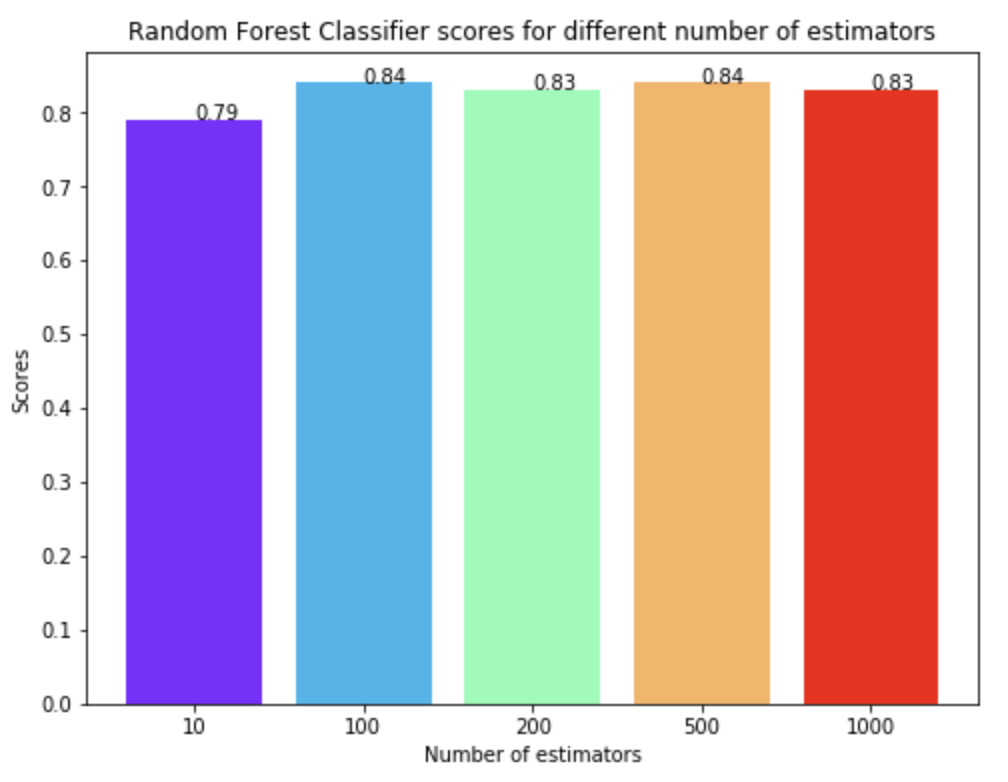


From the line graph above, we can clearly see that the maximum score is 79% and is achieved for maximum features being selected to be either 2, 4 or 18.

**Random Forest Classifier**

This classifier takes the concept of decision trees to the next level. It creates a forest of trees where each tree is formed by a random selection of features from the total features. Here, we can vary the number of trees that will be used to predict the class. I calculate test scores over 10, 100, 200, 500 and 1000 trees.

Next, I plot these scores across a bar graph to see which gave the best results. You may notice that I did not directly set the X values as the array [10, 100, 200, 500, 1000]. It will show a continuous plot from 10 to 1000, which would be impossible to decipher. So, to solve this issue, I first used the X values as [1, 2, 3, 4, 5]. Then, I renamed them using xticks.



Taking a look at the bar graph, we can see that the maximum score of 84% was achieved for both 100 and 500 trees.

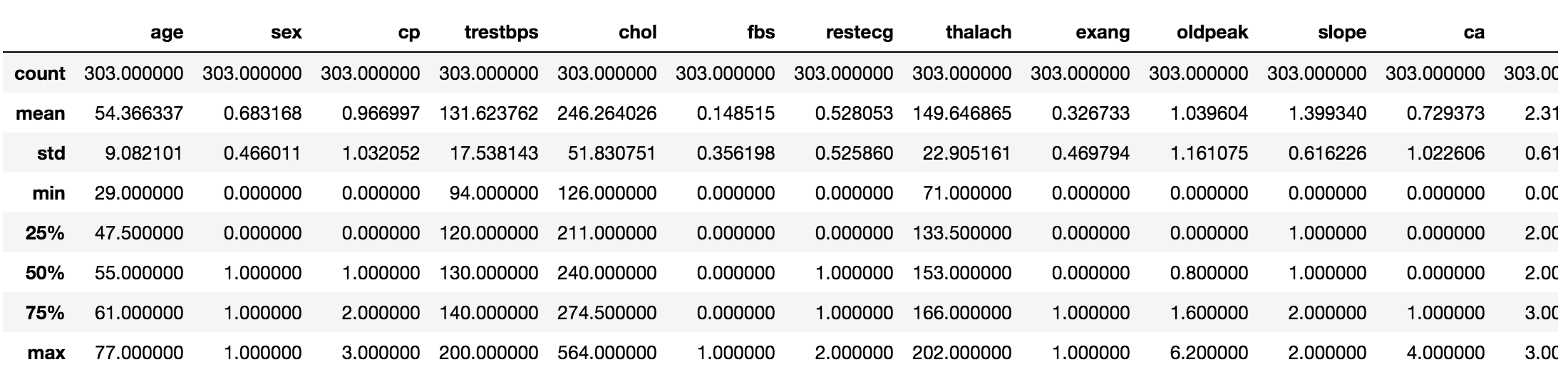
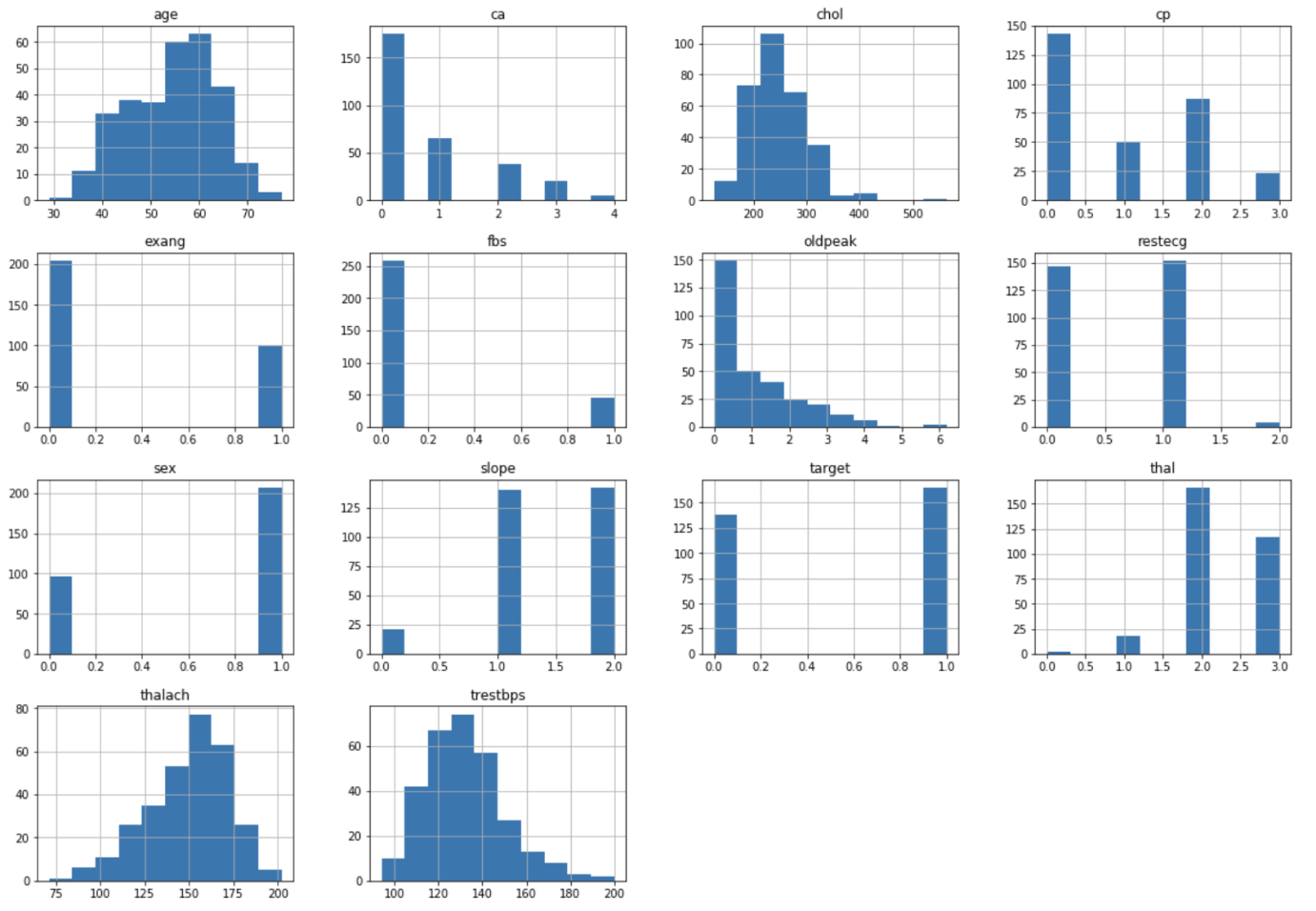
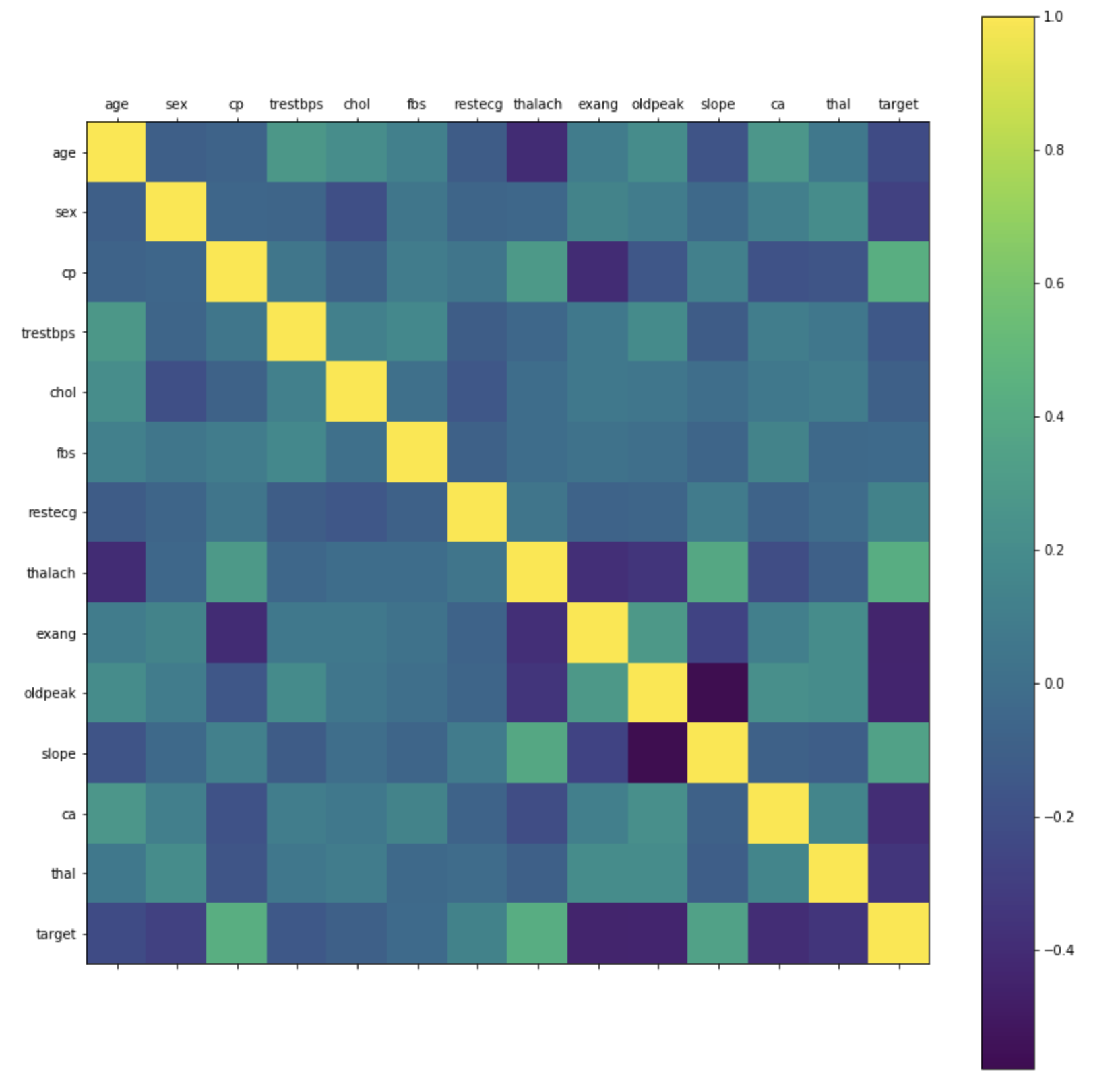
**Conclusion**

The project involved analysis of the heart disease patient dataset with proper data processing. Then, 4 models were trained and tested with maximum scores as follows:

1. K Neighbors Classifier: 87%
2. Support Vector Classifier: 83%
3. Decision Tree Classifier: 79%
4. Random Forest Classifier: 84%

**K Neighbors Classifier scored the best score of 87% with 8 neighbors.**

Thank you for reading! Feel free to share your thoughts and ideas.



dataset.describe()

The method revealed that the range of each variable is different. The maximum value of age is 77 but for chol it is 564. Thus, feature scaling must be performed on the dataset.

# Understanding the data

## Correlation Matrix

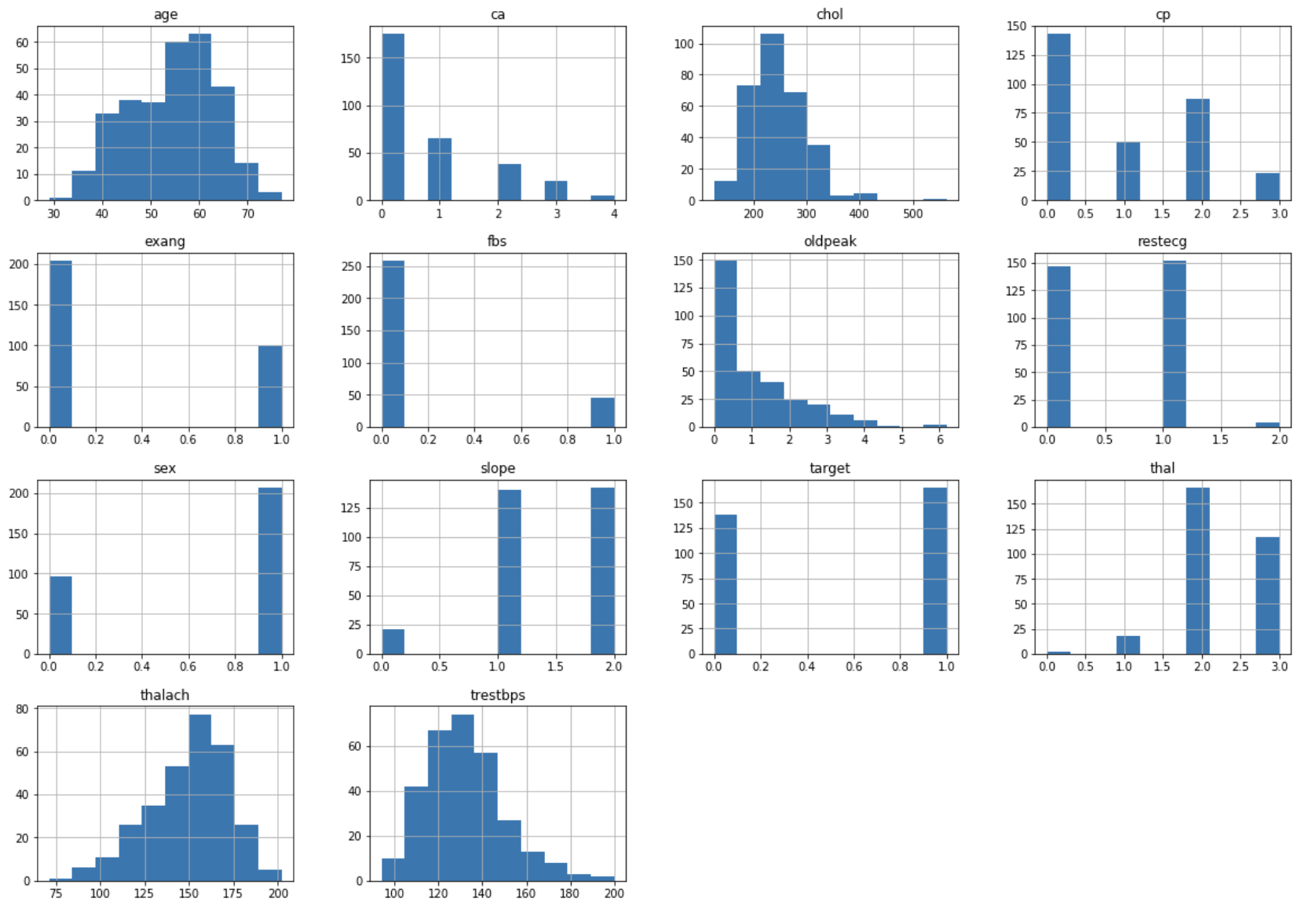
To begin with, let’s see the correlation matrix of features and try to analyse it. The figure size is defined to 12 x 8 by using rcParams. Then, I used pyplot to show the correlation matrix. Using xticks and yticks, I’ve added names to the correlation matrix. colorbar() shows the colorbar for the matrix.

Correlation Matrix

It’s easy to see that there is no single feature that has a very high correlation with our target value. Also, some of the features have a negative correlation with the target value and some have positive.

## Histogram

The best part about this type of plot is that it just takes a single command to draw the plots and it provides so much information in return. Just use dataset.hist().



dataset.hist()

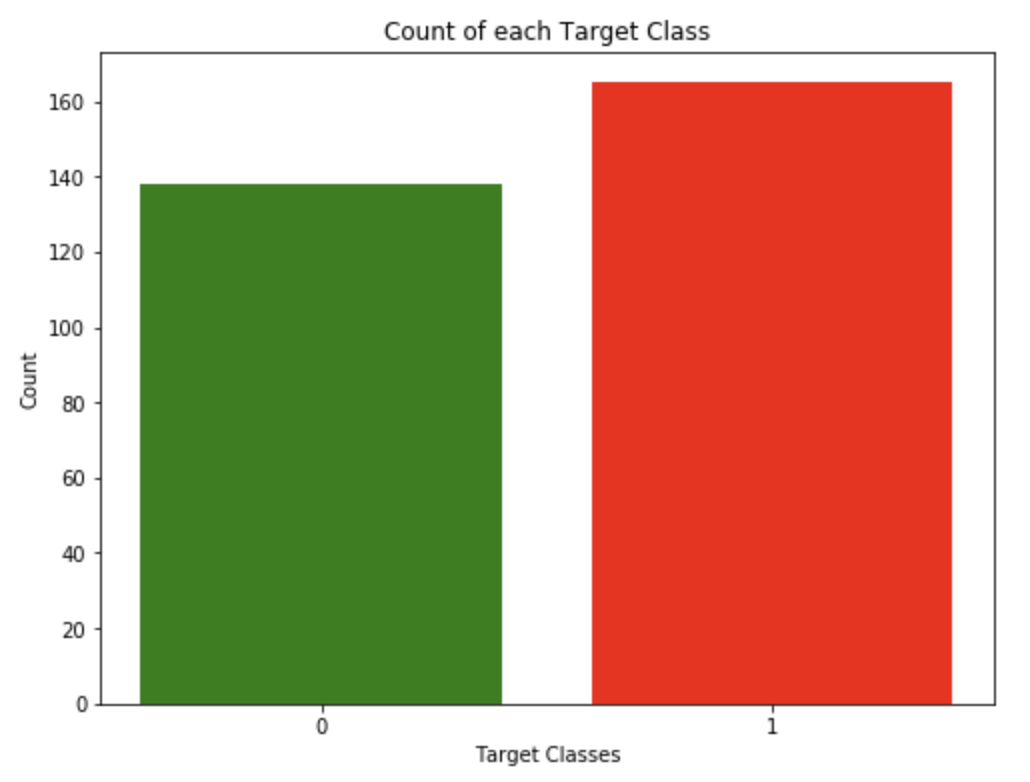
Let’s take a look at the plots. It shows how each feature and label is distributed along different ranges, which further confirms the need for scaling. Next, wherever you see discrete bars, it basically means that each of these is actually a categorical variable. We will need to handle these categorical variables before applying Machine Learning. Our target labels have two classes, 0 for no disease and 1 for disease.

## Bar Plot for Target Class

It’s really essential that the dataset we are working on should be approximately balanced. An extremely imbalanced dataset can render the whole model training useless and thus, will be of no use. Let’s understand it with an example.

*Let’s say we have a dataset of 100 people with 99 non-patients and 1 patient. Without even training and learning anything, the model can always say that any new person would be a non-patient and have an accuracy of 99%. However, as we are more interested in identifying the 1 person who is a patient, we need balanced datasets so that our model actually learns.*

For x-axis I used the unique() values from the target column and then set their name using xticks. For y-axis, I used value\_count() to get the values for each class. I colored the bars as green and red.



From the plot, we can see that the classes are almost balanced and we are good to proceed with data processing.

# Data Processing

To work with categorical variables, we should break each categorical column into dummy columns with 1s and 0s.

Let’s say we have a column Gender, with values 1 for Male and 0 for Female. It needs to be converted into two columns with the value 1 where the column would be true and 0 where it will be false. Take a look at the Gist below.

To get this done, we use the get\_dummies() method from pandas. Next, we need to scale the dataset for which we will use the StandardScaler. The fit\_transform() method of the scaler scales the data and we update the columns.

The dataset is now ready. We can begin with training our models.

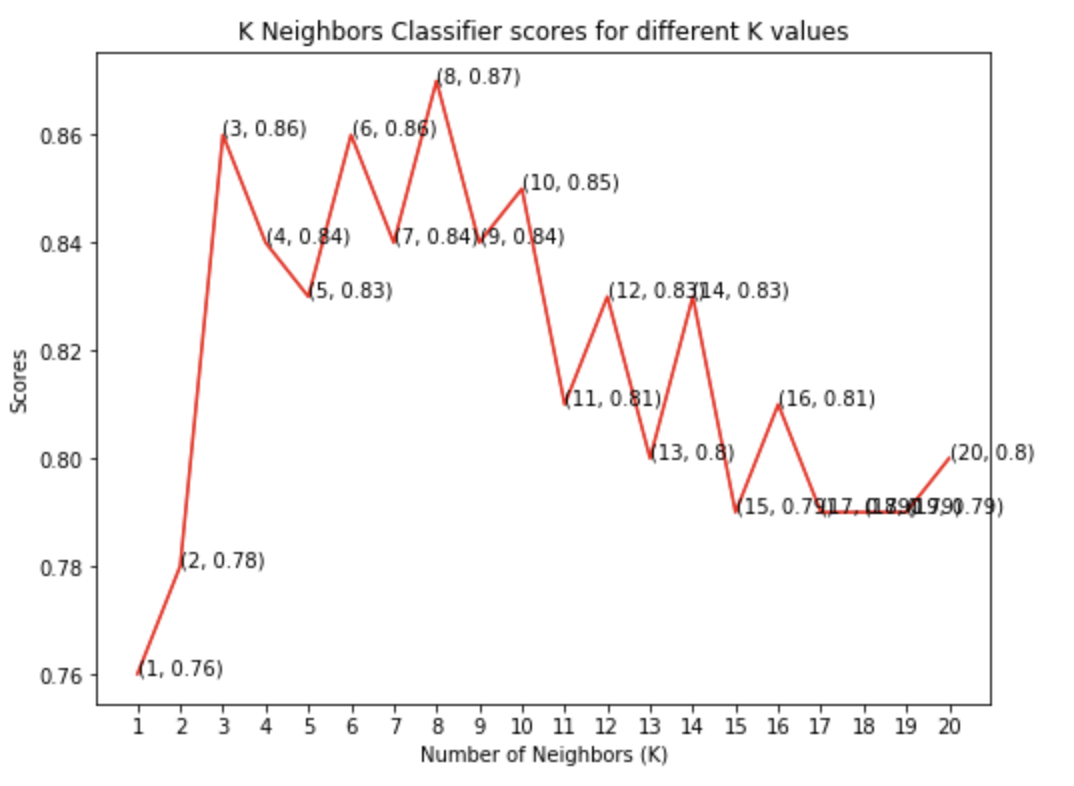
# Machine Learning

In this project, I took 4 algorithms and varied their various parameters and compared the final models. I split the dataset into 67% training data and 33% testing data.

## K Neighbors Classifier

This classifier looks for the classes of K nearest neighbors of a given data point and based on the majority class, it assigns a class to this data point. However, the number of neighbors can be varied. I varied them from 1 to 20 neighbors and calculated the test score in each case.

Then, I plot a line graph of the number of neighbors and the test score achieved in each case.

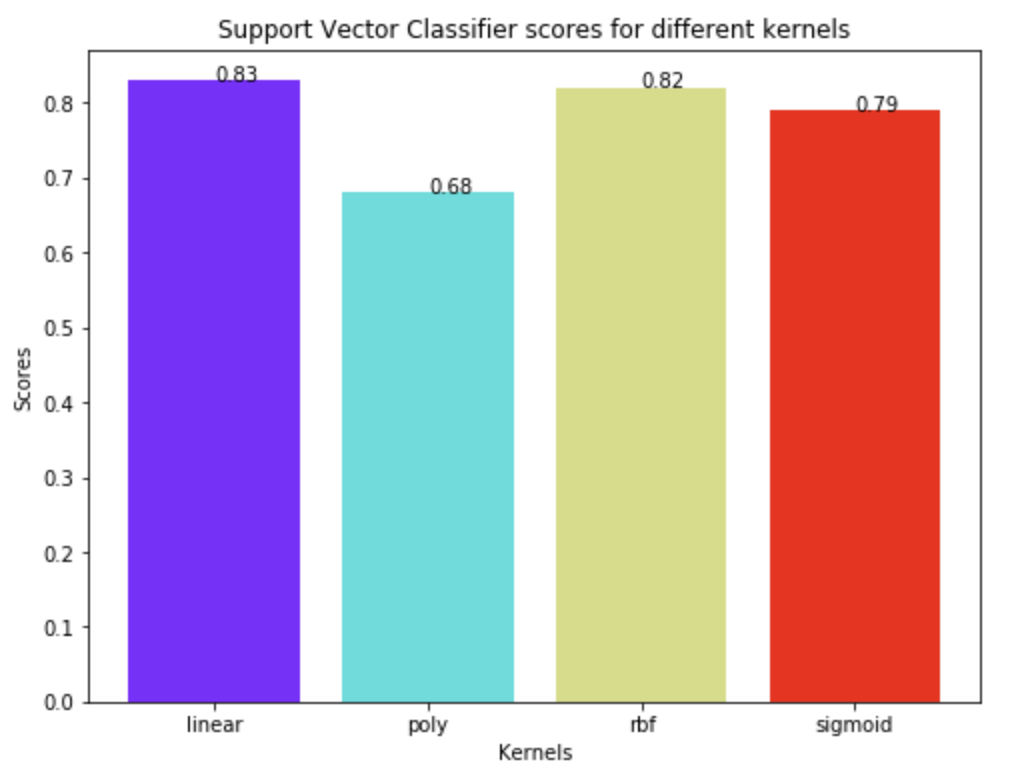


As you can see, we achieved the maximum score of 87% when the number of neighbors was chosen to be 8.

## Support Vector Classifier

This classifier aims at forming a hyperplane that can separate the classes as much as possible by adjusting the distance between the data points and the hyperplane. There are several kernels based on which the hyperplane is decided. I tried four kernels namely, linear, poly, rbf, and sigmoid.

Once I had the scores for each, I used the rainbow method to select different colors for each bar and plot a bar graph of the scores achieved by each.

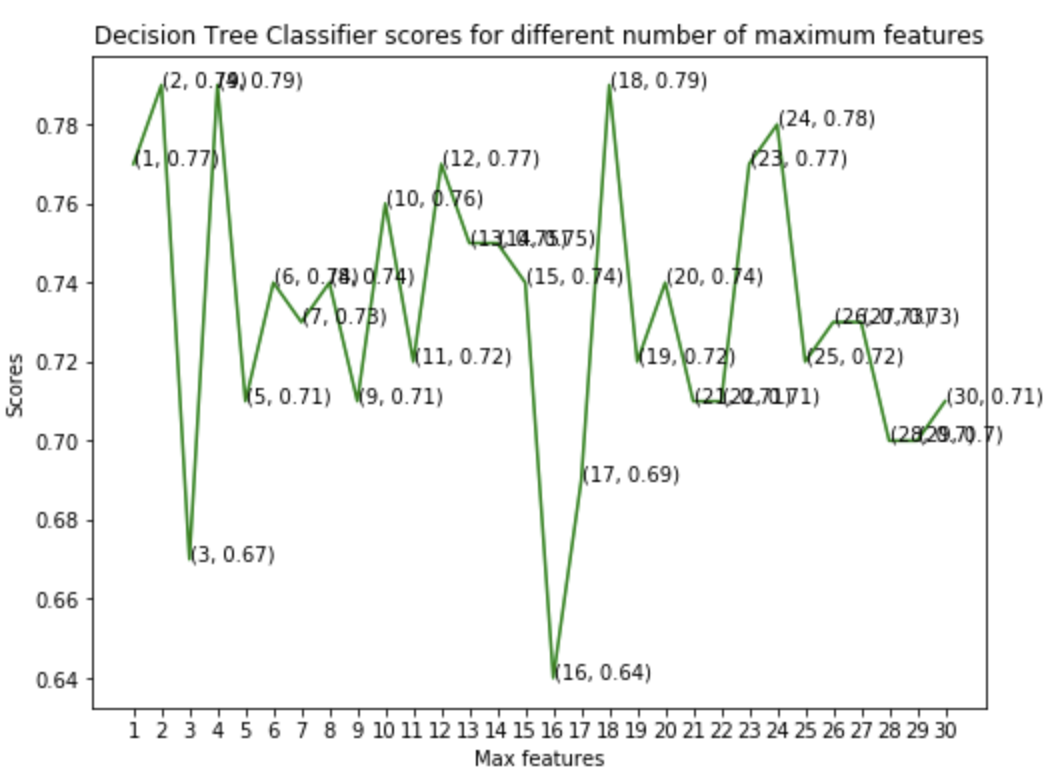


As can be seen from the plot above, the linear kernel performed the best for this dataset and achieved a score of 83%.

## Decision Tree Classifier

This classifier creates a decision tree based on which, it assigns the class values to each data point. Here, we can vary the maximum number of features to be considered while creating the model. I range features from 1 to 30 (the total features in the dataset after dummy columns were added).

Once we have the scores, we can then plot a line graph and see the effect of the number of features on the model scores.

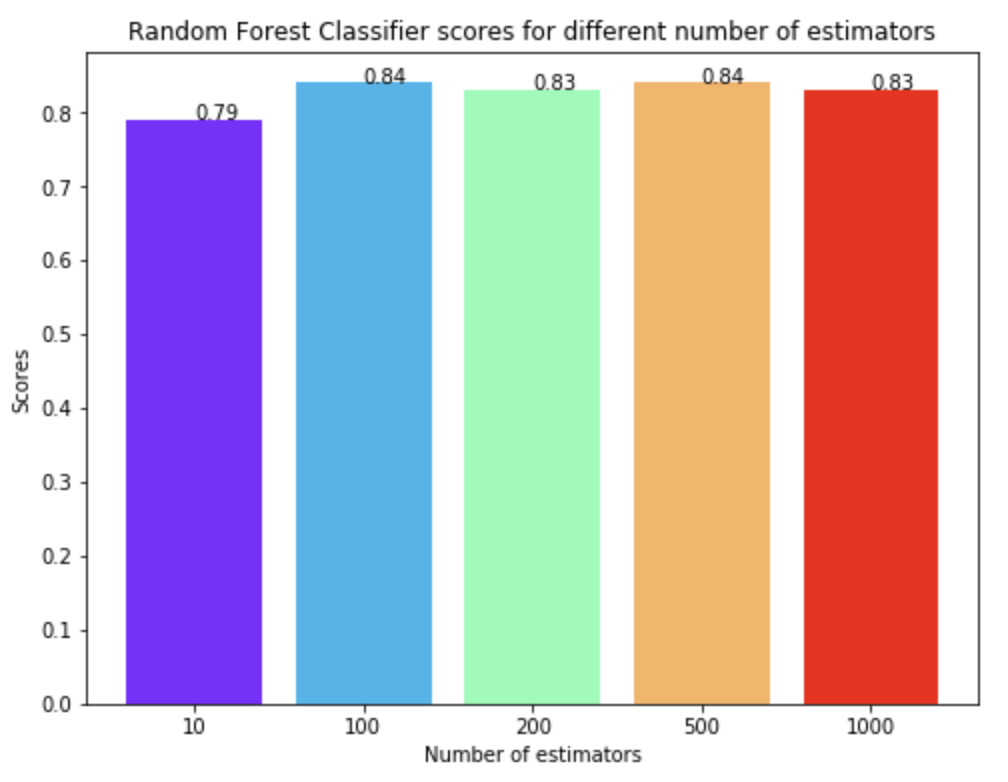


From the line graph above, we can clearly see that the maximum score is 79% and is achieved for maximum features being selected to be either 2, 4 or 18.

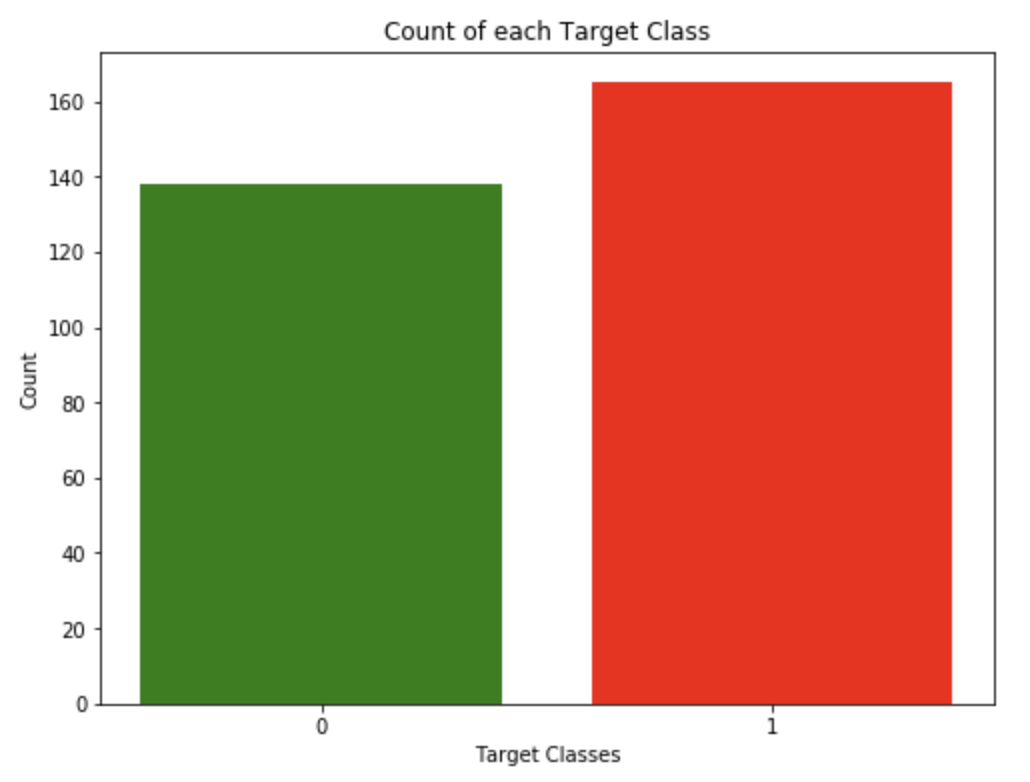
## Random Forest Classifier

This classifier takes the concept of decision trees to the next level. It creates a forest of trees where each tree is formed by a random selection of features from the total features. Here, we can vary the number of trees that will be used to predict the class. I calculate test scores over 10, 100, 200, 500 and 1000 trees.

Next, I plot these scores across a bar graph to see which gave the best results. You may notice that I did not directly set the X values as the array [10, 100, 200, 500, 1000]. It will show a continuous plot from 10 to 1000, which would be impossible to decipher. So, to solve this issue, I first used the X values as [1, 2, 3, 4, 5]. Then, I renamed them using xticks.



Taking a look at the bar graph, we can see that the maximum score of 84% was achieved for both 100 and 500 trees.



Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](https://medium.com/about?autoplay=1&source=post_page-----36f00f3edb2c----------------------)

#### [Make Medium yours](https://medium.com/topics?source=post_page-----36f00f3edb2c----------------------)

Follow all the topics you care about, and we’ll deliver the best stories for you to your homepage and inbox. [Explore](https://medium.com/topics?source=post_page-----36f00f3edb2c----------------------)

#### [Become a member](https://medium.com/membership?source=post_page-----36f00f3edb2c----------------------)

Get unlimited access to the best stories on Medium — and support writers while you’re at it. Just $5/month. [Upgrade](https://medium.com/membership?source=post_page-----36f00f3edb2c----------------------)

[About](https://medium.com/about?autoplay=1&source=post_page-----36f00f3edb2c----------------------)[Help](https://help.medium.com/?source=post_page-----36f00f3edb2c----------------------)[Legal](https://medium.com/policy/9db0094a1e0f?source=post_page-----36f00f3edb2c----------------------)