Random Forest Classifier

Red wine quality

S.P.P.M.Sudasinghe
S14510/2018s16976

Contents

Dataset 2
Problem introduction 2
My approach 2
Importing libraries and dataset 3
Libraries 3
Dataset 3
Cleaning data 4
Rename column names 4
Remove duplicates 4
Data analysis and Visualization 5
Number of rows and columns 5
Data types 5
Missing values 6
Correlation matrix 6
Number of observations in each quality group 7
Bar plot of wine quality 7
Testing and training datasets 9
Defining the training and testing sets 9
Apply random forest classifier 9
Accuracy 10
Confusion matrix 10
Building a prediction system 12
Further Improvements 12
Conclusion 13
References 13
Link for the complete code 13

Dataset

Red Wine Quality Dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv

Data set information: https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality.names

The following are explanations of the columns (variables):

Attribute Information:

For more information, read [Cortez et al., 2009].
Input variables (based on physicochemical tests):
1 - fixed acidity
2 - volatile acidity
3 - citric acid
4 - residual sugar
5 - chlorides
6 - free sulfur dioxide
7 - total sulfur dioxide
8 - density
9 - pH
10 - sulphates
11 - alcohol
Output variable (based on sensory data):
12 - quality (score between 0 and 10)

There are 12 columns which represent 11 features and the one target variable ‘quality’.

Problem introduction

The problem is how to predict red wine quality is bad or good based on its other attributes using past data.

My approach

Since this is a supervised machine learning problem try to create a suitable model to predict the wine quality (good/bad) using a random forest classifier.

Importing libraries and dataset

Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sn
```

- Pandas is a package for data analysis & manipulation
- Numpy is a package for scientific computing & multidimensional arrays
- Matplotlip is a package for creating visualizations
- Sklearn is a machine learning library
- Seaborn is a data visualization library based on matplotlib

Dataset

The following code loads the CSV data separate with ';' and display the structure of the data set.

```
df = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv" ,sep=';')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

- It shows that, the data containing in the 1st row is repeated again in the 5th row. Then there can be more duplicated data rows like this in the dataset.
- Also spaces are included in most of the variable names which doesn't allow to use.
- So in the cleaning data part these points should be considered.

Cleaning data

Normally about 80% of the time spent in data analysis is for cleaning data. That's why here it's very important. As mentioned before there are spaces between some column names.

Replace those spaces by underscore('_').

Rename column names

```
df.columns = df.columns.str.replace(' ', '_')
df.head()
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

And also as mentioned above need to remove duplicate rows as well. Otherwise it makes wrong conclusions & can have a large effect on final output.

Remove duplicates

1st check the number of duplicate data rows:

```
print('Number of duplicated rows =', df.duplicated().sum())
```

```
Number of duplicated rows = 240
```

There are 240 duplicate rows in this dataset. So the next step would be removing these rows

```
df.drop_duplicates(inplace=True)
df.reset_index(drop=True, inplace=True)
print('Number of rows after removing duplicates =', len(df))
```

```
Number of rows after removing duplicates = 1359
```

Now there are only 1359 unique data rows in this data set.

Data analysis and Visualization

Number of rows and columns

Check how many columns and rows are there?

```
df.shape
```

```
(1359, 13)
```

There are 1359 rows (without duplicates) and 13 columns.

Data types

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1359 entries, 0 to 1358
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         1359 non-null   float64
 1   volatile_acidity      1359 non-null   float64
 2   citric_acid           1359 non-null   float64
 3   residual_sugar        1359 non-null   float64
 4   chlorides             1359 non-null   float64
 5   free_sulfur_dioxide   1359 non-null   float64
 6   total_sulfur_dioxide  1359 non-null   float64
 7   density               1359 non-null   float64
 8   pH                    1359 non-null   float64
 9   sulphates             1359 non-null   float64
 10  alcohol               1359 non-null   float64
 11  quality               1359 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 127.5 KB
```

- Response variable (quality) is the only integer variable and others are float type variables.
- It takes 127.5KB memory.

Missing values

Check are there any missing values in the dataset

```
print('Number of missing values:',df.isna().sum().sum())
```

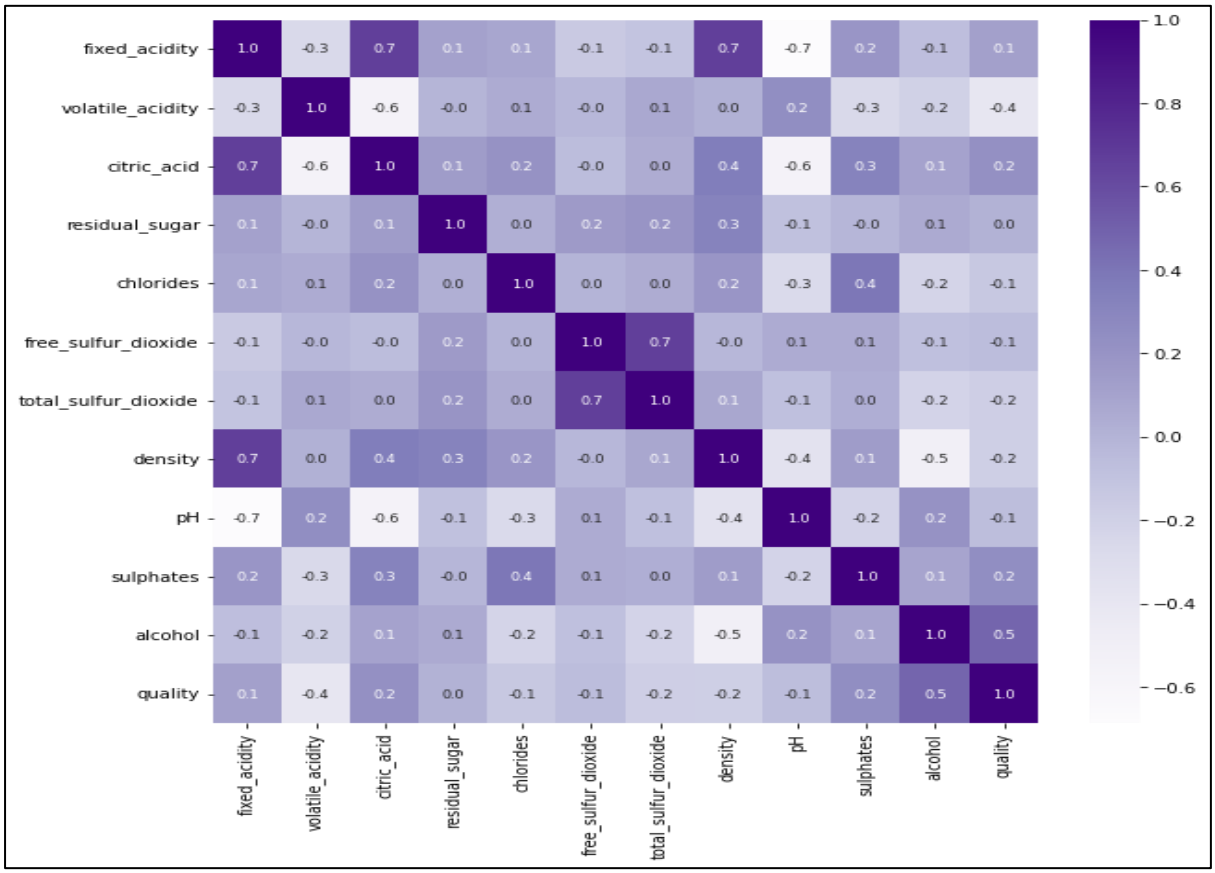
```
Number of missing values: 0
```

- No missing values in the dataset. So no need of an imputation or any other technique.
- Using is.na() function with sum() we can obtain the number of missing values for entire data set.

Correlation matrix

Draw a heat map to understand the correlation between the variables.

```
plt.figure(figsize=(10,10))
sn.heatmap(df.corr(), cbar=True, square=True, fmt='.1f',annot=True,annot_kws={'size':8}, cmap='Purples')
```



Dark colour means the positive correlations between 2 variables and light colour means the negative correlation between variables. This heat map shows that there is a significant relationship between the two variables quality and volatile acidity. It's a negative correlation. And also there is a positive significant correlation between quality and alcohol. Therefore volatile acidity and alcohol may be more important when creating a model to predict wine quality.

Number of observations in each quality group

```
df.quality.value_counts()
```

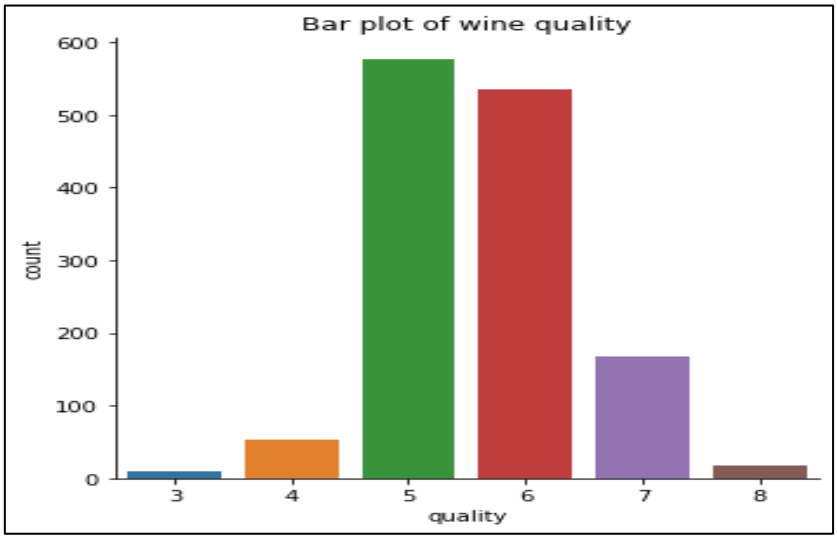
```
5    577
6    535
7    167
4     53
8     17
3     10
Name: quality, dtype: int64
```

We can notice there is a data imbalance here. Using a graphical method this can be visualized clearly.

Bar plot of wine quality

Since in this problem quality labels can be considered as categorical variables we can plot a bar plot to visualize data.

```
plt.figure(figsize = (4,4))
sn.catplot(x='quality', data=df, kind='count')
plt.show()
```



Quality level score is in between 3 and 8. It doesn't mean much and also data are extremely imbalanced. Bar sizes of quality score 5 and 6 are extremely larger than others. To overcome this problems split this in to 2 groups and try to balance it. When the red wine quality is less than or equal to 5 consider the quality as 'Bad' and recode it as '0'. When the red wine quality is greater than 5 consider the quality as 'Good' and recode it as '1'.

```
df['quality']=pd.cut(df['quality'],bins=[0,5.5,10], labels=[0,1])
df.head()
```

Check the new quality labels:

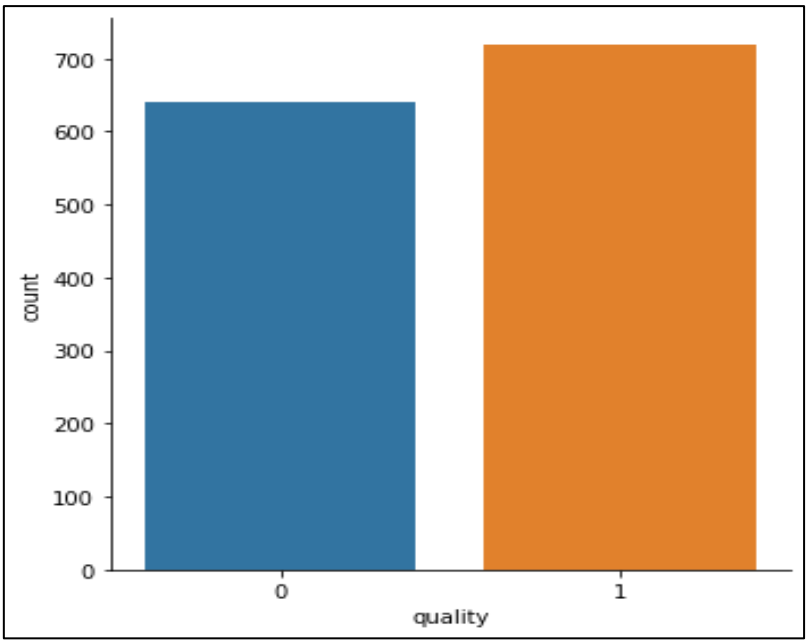
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

Check the counts for new labels:

```
df.quality.value_counts()
```

```
1    719
0    640
Name: quality, dtype: int64
```

- There are 719 good quality wines and 640 bad quality wines.
- According to the new quality labels, data set is quite balanced. Following bar plot also visually represent that.



Testing and training datasets

Defining the training and testing sets

```
X_train, X_test, Y_train, Y_test = train_test_split(df.drop(['quality'],axis='columns'),df.quality,test_size=0.2)
```

- Data frame already has the response variable ‘quality’. To take X we need to drop the response variable and Y only contain the response variable.
- To Split the dataset in to training and testing, we need to specify those ratios. According to the above code test size is taken as 0.2. It means 20% of total data is test data and other 80% are training. This code creates above mentioned 4 variables: X_train, X_test, Y_train and Y_test.

Apply random forest classifier

Train classifier

```
model = RandomForestClassifier()
model.fit(X_train,Y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

- *‘from sklearn.ensemble import RandomForestClassifier’* applies in here and the term ensemble is used when using multiple algorithms to predict an outcome. In here we use multiple decision trees for the final predictions.
- There are so many parameters here and the random forest classifier use *‘gini impurity’* as its attribute selection measure. The parameter *‘n_estimators’* is for the number of random trees. In this case there are 100 random trees. Final outcome is based on these 100 random trees.
- This is the training step and after this step model will be trained.

Accuracy

After training, check the accuracy using actual and predicted values.

```
print('Accuracy:',model.score(X_test,Y_test)*100,'%')
```

```
Accuracy: 84.19117647058823 %
```

When we use 100 random trees accuracy is 84.19%

Confusion matrix

- Confusion matrix is another way to describe the performance of a classification model.
- It is a table that summarizes prediction and true results for classification problems.
- Plot truth (Y_test) on one axis & the prediction (Y_predicted) on the other axis.

Obtain prediction values:

```
Y_predicted=model.predict(X_test)
```

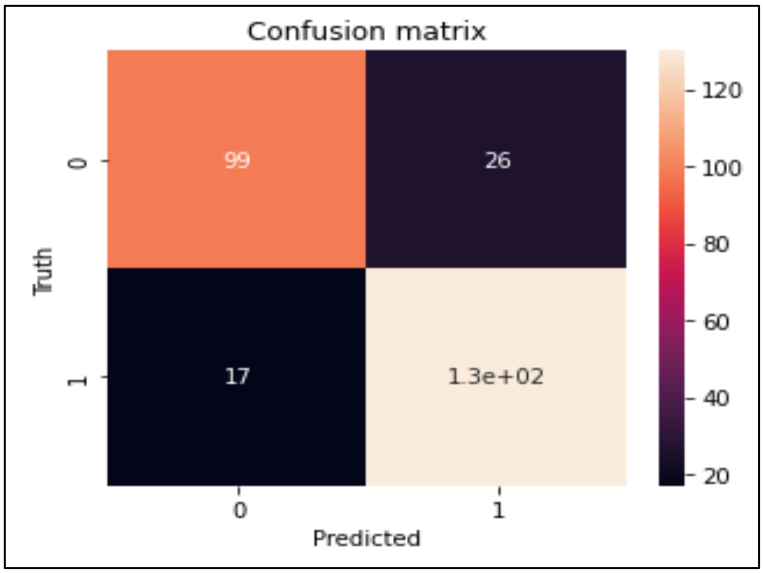
Create confusion matrix:

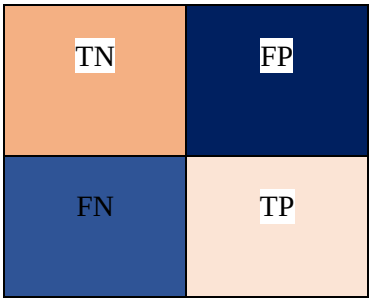
```
cm=confusion_matrix(Y_test,Y_predicted)
cm
```

```
array([[ 99,  26],
       [ 17, 130]])
```

To visualize the above array in a more meaningful manner we use the following code:

```
plt.figure(figsize=(5,4))
sn.heatmap(cm,annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```





- TN- True Negative: represents data points that have a bad quality label that has been correctly classified as bad quality by the model
- FP- False Positive: represents data points that have a bad quality label but that has been predicted as good quality by the model
- FN- False Negative: represents data points that have a good quality label that has been incorrectly predicted as bad quality by the algorithm
- TP- True positive: represents the data points that have a good quality label which was correctly predicted by the model

Above confusion matrix tells that out of the 147 good quality labels, 130 are correctly classified and 17 are incorrectly classified as bad quality. Also, it shows that out of 125 bad quality labels, 99 are correctly classified and 26 are incorrectly classified as good quality.

TP and TN indicate the data points that our model has correctly classified. So by using the following equation we can obtain the same accuracy value:

$$Accuracy = \frac{Number\ of\ correctly\ calssified\ samples}{Total\ number\ of\ samples}$$

$$Accuracy = \left(\frac{TP + TN}{TN + FP + FN + TP}\right) \times 100\%$$

By substituting values to this equation:

$$Accuracy = \frac{130 + 99}{99 + 26 + 17 + 130} \times 100\%$$

$$Accuracy = 84.19117647\%$$

Building a prediction system

```
fixed_acidity =7.4
volatile_acidity= 0.7
citric_acid =0
residual_sugar= 1.9
chlorides =0.076
free_sulfur_dioxide= 11
total_sulfur_dioxide= 34
density= 0.9978
pH= 3.51
sulphates= 0.56
alcohol= 9.4
prediction=model.predict([[fixed_acidity ,volatile_acidity ,citric_acid ,residual_sugar ,chlorides ,free_sulfur_dioxide ,total_sulfur_dioxide ,density ,pH ,
print(prediction)
if(prediction == 1):
  print('A good quality wine')
else:
  print('A bad quality wine')
```

```
[0]
A bad quality wine
```

If the quality label is 1 then the quality is good. If the label is 0 then the quality of the wine is bad. According to the entered values here the code gives this is a bad quality wine. So this can be used as a prediction system for future activities.

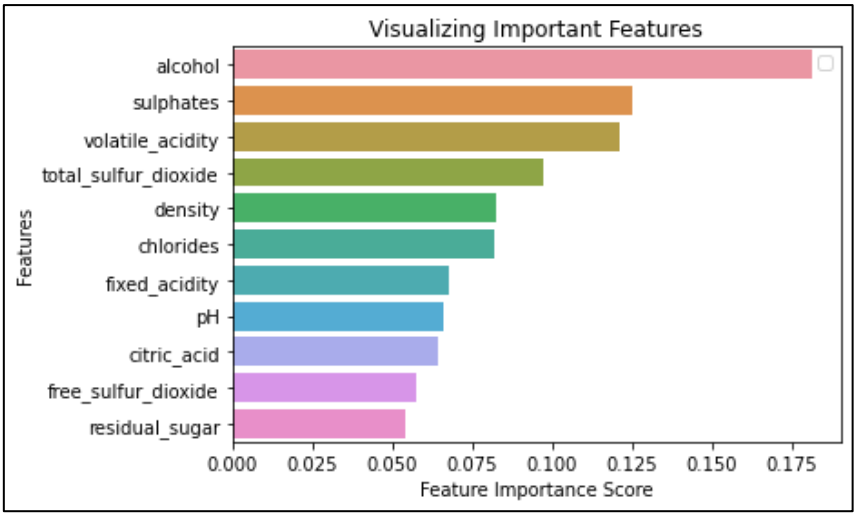
Further Improvements

- Visualize feature importance as below and remove the unimportant variables after a proper investigation and generate a new model.

```
import pandas as pd
feature_imp = pd.Series(model.feature_importances_, index=df.columns[:11]).sort_values(ascending=False)
feature_imp
```

```
alcohol                0.181335
sulphates              0.125270
volatile_acidity       0.121302
total_sulfur_dioxide   0.097266
density                0.082389
chlorides              0.082150
fixed_acidity          0.067834
pH                     0.066294
citric_acid            0.064121
free_sulfur_dioxide    0.057717
residual_sugar         0.054321
dtype: float64
```

```
sn.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()
```



We can clearly notice that alcohol, sulphates & volatile_acidity are very important variables in this case and free_sulfur_dioxide & residual_sugar are not very important.

- By changing/tuning the parameters in the random forest classifier you can obtain a more accurate model than this.
  Ex: To obtain a good accuracy value you can tune the parameter 'number of trees' (n_estimators)
- Define 'Bad' and 'Good' quality labels in another way & try to maximize the accuracy.

Conclusion

The random forest is a flexible machine learning algorithm and as mentioned earlier it can be used for ensemble learning as well. Not only that but also it can be used to make more accurate predictions. Anyone can improve the prediction model and can obtain a higher accuracy level than what obtained here by using many techniques as mentioned in the previous part under the heading 'Further Improvements'.

References

- https://www.datacamp.com/community/tutorials/random-forests-classifier-python
- https://www.youtube.com/watch?v=ok2s1vV9XW0
- https://towardsdatascience.com/the-confusion-matrix-unveiled-2d030136be40

Link for the complete code

- https://colab.research.google.com/drive/1vIa0kRSs6US_0hpYyBfCfLZtdu3cYpH-?usp=sharing