CONTRIBUTION REPORT

SHAMRA MARZOOK
E/16/232
GROUP 2
16/10/2021

CHRONIC KIDNEY DISEASE ANALYSIS

INTRODUCTION

Chronic kidney disease, also known as chronic renal disease is a major medical problem that damage the kidneys which results in a gradual loss of kidney function and decrease their ability to keep one healthy.
Early detection and treatment can often keep chronic kidney disease from getting worse and delay the process, preventing it from being fatal.

Machine Learning is a rising field concerned with the study of huge and multiple variable data. It has a huge impact on health sector through an effective analysis of various chronic diseases for more accurate diagnosis and successful treatment.

This work train and test the given factors which causes Chronic Kidney Disease and with the best performing machine learning model it can effortlessly predict the result of the patient with much higher accuracy because traditional methods can be faulty at certain times. Prediction in traditional method is done by doctors, prevention of human errors is not possible.

This work predominantly focused on, prediction of chronic kidney disease in patients using classification techniques of data mining. Thereby, down-staging it to stages that are more amenable to curative treatment.

First the dataset was obtained, and the data-preprocessing was done. Then feature selection and model evaluation and classification was done to obtain favorable results. My contribution to the project included both the data pre-processing as well as the model evaluation.

DATA PREPROCESSING

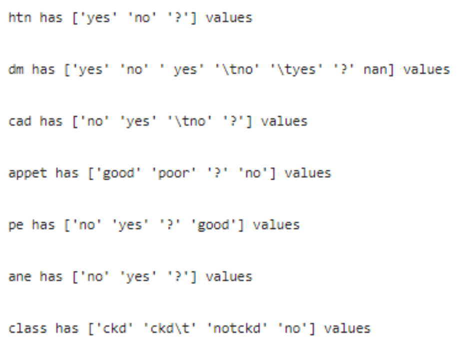
Each features were examined and data preprocessing was done to refine the dataset. It consisted of both numerical and categorical data types and it was decided to pre-process the two types separately. And my task included pre-processing of categorical data and numerical data as well as.

When the dataset was examined, it was seen that the dataset consisted of attributes where some of the data included incorrect characters in both numerical and categorical data types.
For example:

su has ['0' '3' '4' '1' ***'?'*** *'2' '5'] values*
cad has ['no' 'yes' ***'\tno' '?'****] values*

su data has '?' and cad data has **'\tno' '?'**.

These types of data had to be refined and preprocessed. In order to do so, the following steps were done.


```
htn has ['yes' 'no' '?'] values

dm has ['yes' 'no' ' yes' '\tno' '\tyes' '?' nan] values

cad has ['no' 'yes' '\tno' '?'] values

appet has ['good' 'poor' '?' 'no'] values

pe has ['no' 'yes' '?' 'good'] values

ane has ['no' 'yes' '?'] values

class has ['ckd' 'ckd\t' 'notckd' 'no'] values
```

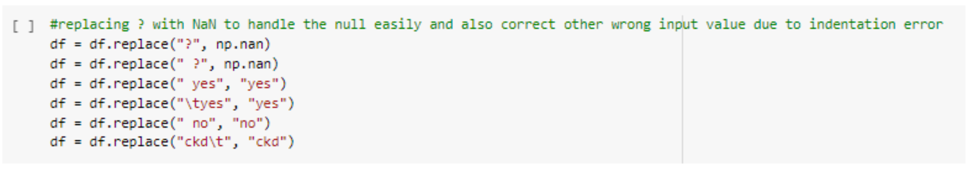

```
[ ]  #replacing ? with NaN to handle the null easily and also correct other wrong input value due to indentation error
     df = df.replace("?", np.nan)
     df = df.replace(" ?", np.nan)
     df = df.replace(" yes", "yes")
     df = df.replace("\tyes", "yes")
     df = df.replace(" no", "no")
     df = df.replace("ckd\t", "ckd")
```

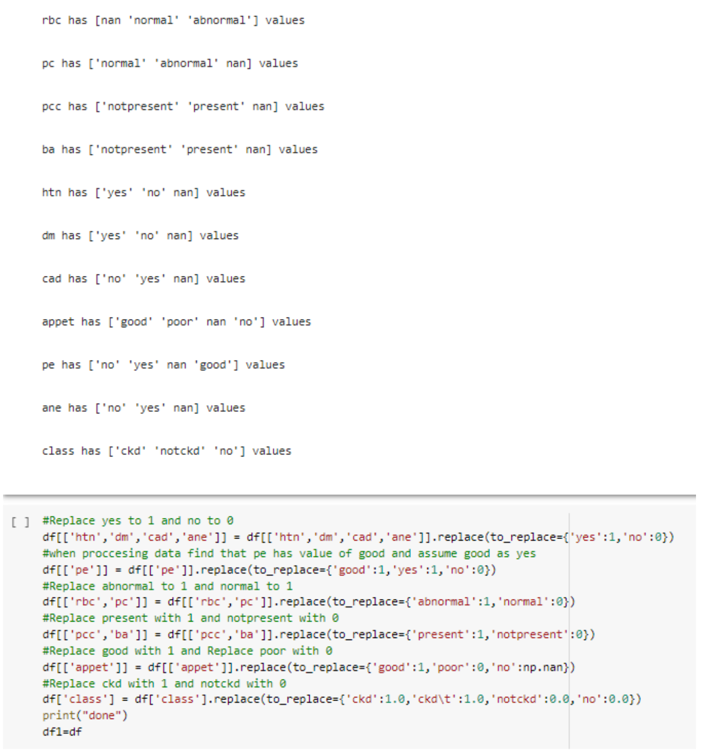
Figure 1: Handling incorrect characters and indentation errors in the dataset

Categorical data were in the form of “Yes”, “No”, “Normal” "Abnormal", “Present”, “Not present”,etc.
For example:

*rbc has [nan '**normal**' '**abnormal**'] values*
*pcc has ['**notpresent**' '**present**' nan] values*
*htn has ['**yes**' '**no**' nan] values*

They were replaced by values with binary input, where 1 is for true and 0 is for false values, so that a refined dataset is obtained.

The following were the steps that were carried out to achieve this.



```
rbc has [nan 'normal' 'abnormal'] values

pc has ['normal' 'abnormal' nan] values

pcc has ['notpresent' 'present' nan] values

ba has ['notpresent' 'present' nan] values

htn has ['yes' 'no' nan] values

dm has ['yes' 'no' nan] values

cad has ['no' 'yes' nan] values

appet has ['good' 'poor' nan 'no'] values

pe has ['no' 'yes' nan 'good'] values

ane has ['no' 'yes' nan] values

class has ['ckd' 'notckd' 'no'] values
```

```
[ ]  #Replace yes to 1 and no to 0
     df[['htn','dm','cad','ane']] = df[['htn','dm','cad','ane']].replace(to_replace={'yes':1,'no':0})
     #when proccesing data find that pe has value of good and assume good as yes
     df[['pe']] = df[['pe']].replace(to_replace={'good':1,'yes':1,'no':0})
     #Replace abnormal to 1 and normal to 1
     df[['rbc','pc']] = df[['rbc','pc']].replace(to_replace={'abnormal':1,'normal':0})
     #Replace present with 1 and notpresent with 0
     df[['pcc','ba']] = df[['pcc','ba']].replace(to_replace={'present':1,'notpresent':0})
     #Replace good with 1 and Replace poor with 0
     df[['appet']] = df[['appet']].replace(to_replace={'good':1,'poor':0,'no':np.nan})
     #Replace ckd with 1 and notckd with 0
     df['class'] = df['class'].replace(to_replace={'ckd':1.0,'ckd\t':1.0,'notckd':0.0,'no':0.0})
     print("done")
     df1=df
```

Figure 2: Encoding of data

Next the dataset was examined for null values, and the graph in figure shows a summary of the null values in each features.

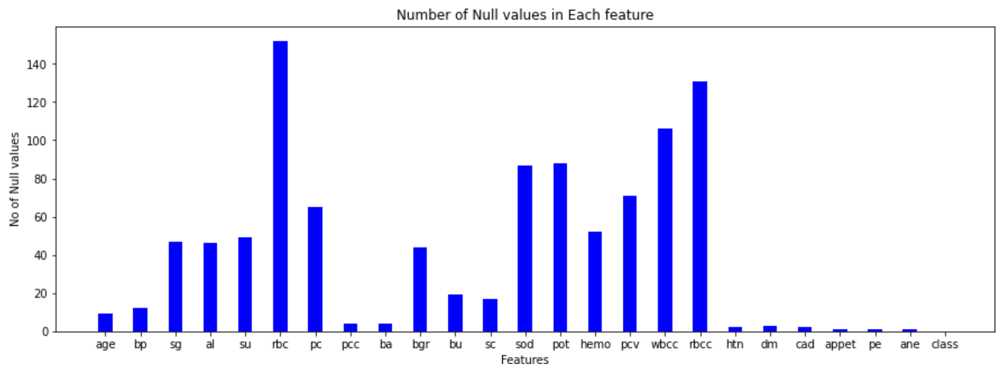


Figure 3: Null values of dataset

When the figure () is observed, it can be seen got that some features have more than 50 null values. Therefore it was required to handle them as well. It was unable to drop the features with the null values because the dataset consisted of 400 instances and if the features are dropped, the dataset reduced to a number around 200. Therefore in order to handle the null values in both numerical and categorical data types, mean value method, random value method and most frequent value methods were adopted (The handling of null values were carried out but the rest of the team members).

MODEL EVALUATION

The dataset was split as training and testing datasets. When splitting 20% was used as testing data and the rest 80% as training data. Scikit-learn provides a wide range of machine learning classifiers which have a unified/consistent interface for fitting, predicting accuracy, etc.

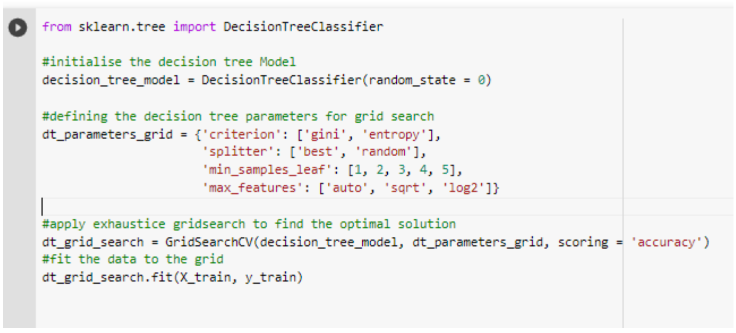
For the model evaluation 4 classifiers were used. They are K-Nearest Neighbors Classifier, Support Vector Classifier, Decision Tree Classifier and Random Forest classifier. The confusion matrix and the classification report for each algorithm was examined. A confusion matrix is a technique for summarizing the performance of a classification algorithm.

Under model evaluation, my task included the training, testing and evaluation of the Decision Tree Classifier & Random Forest classifier.

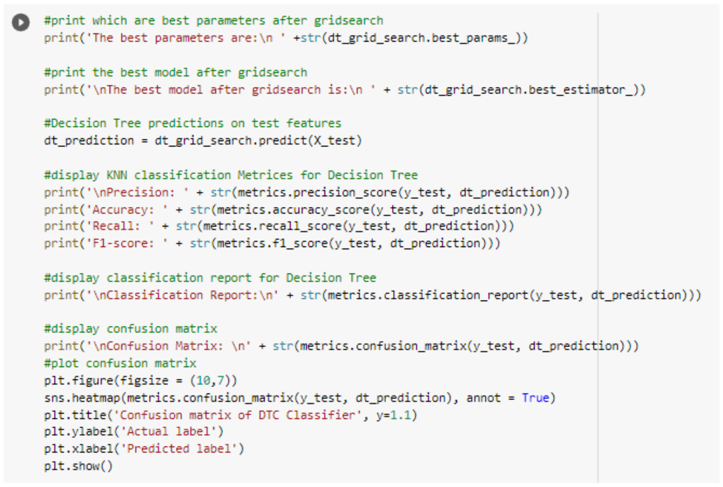
Decision Tree Classifier

A decision tree is a flowchart-like tree structure where an internal node represents features the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in recursively manner call recursive partitioning.

For the training and evaluation of the decision tree classifier the following steps were carried out.



```python
from sklearn.tree import DecisionTreeClassifier

#initialise the decision tree Model
decision_tree_model = DecisionTreeClassifier(random_state = 0)

#defining the decision tree parameters for grid search
dt_parameters_grid = {'criterion': ['gini', 'entropy'],
                      'splitter': ['best', 'random'],
                      'min_samples_leaf': [1, 2, 3, 4, 5],
                      'max_features': ['auto', 'sqrt', 'log2']}

#apply exhaustice gridsearch to find the optimal solution
dt_grid_search = GridSearchCV(decision_tree_model, dt_parameters_grid, scoring = 'accuracy')
#fit the data to the grid
dt_grid_search.fit(X_train, y_train)
```

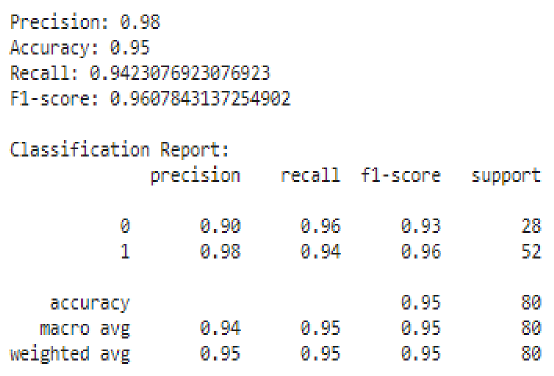


```python
#print which are best parameters after gridsearch
print('The best parameters are:\n ' +str(dt_grid_search.best_params_))

#print the best model after gridsearch
print('\nThe best model after gridsearch is:\n ' + str(dt_grid_search.best_estimator_))

#Decision Tree predictions on test features
dt_prediction = dt_grid_search.predict(X_test)

#display KNN classification Metrices for Decision Tree
print('\nPrecision: ' + str(metrics.precision_score(y_test, dt_prediction)))
print('Accuracy: ' + str(metrics.accuracy_score(y_test, dt_prediction)))
print('Recall: ' + str(metrics.recall_score(y_test, dt_prediction)))
print('F1-score: ' + str(metrics.f1_score(y_test, dt_prediction)))

#display classification report for Decision Tree
print('\nClassification Report:\n' + str(metrics.classification_report(y_test, dt_prediction)))

#display confusion matrix
print('\nConfusion Matrix: \n' + str(metrics.confusion_matrix(y_test, dt_prediction)))
#plot confusion matrix
plt.figure(figsize = (10,7))
sns.heatmap(metrics.confusion_matrix(y_test, dt_prediction), annot = True)
plt.title('Confusion matrix of DTC Classifier', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```

Figure 4: Decision Tree Classifier

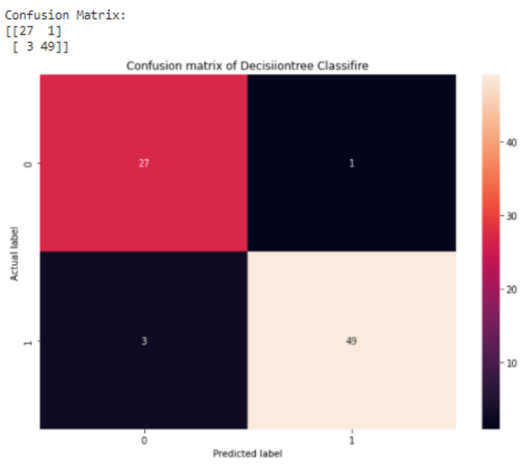
First the model was initialized and the decision tree parameters for grid search were defined. To find the optimal solution, exhaustice grid search was done and the data was fit to the grid.

Decision Tree predictions on test features was done. And the classification metrices like precision, accuracy, recall and F1-score were shown. Then the confusion matrix and the classification report was examined.



```
Precision: 0.98
Accuracy: 0.95
Recall: 0.9423076923076923
F1-score: 0.9607843137254902

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93        28
           1       0.98      0.94      0.96        52

    accuracy                           0.95        80
   macro avg       0.94      0.95      0.95        80
weighted avg       0.95      0.95      0.95        80
```

Figure 5: Classification report for DTC



```
Confusion Matrix:
[[27  1]
 [ 3 49]]
```

Figure 6: Confusion matrix for DTC

As shown in figure (), the accuracy of the model is 0.95 and the precision is 0.98. Here CKD (chronic kidney disease) is consider as 1 and non-CKD (non- chronic kidney disease) as 0. The classification report shows how the metrices vary for the CKD and non-CKD.

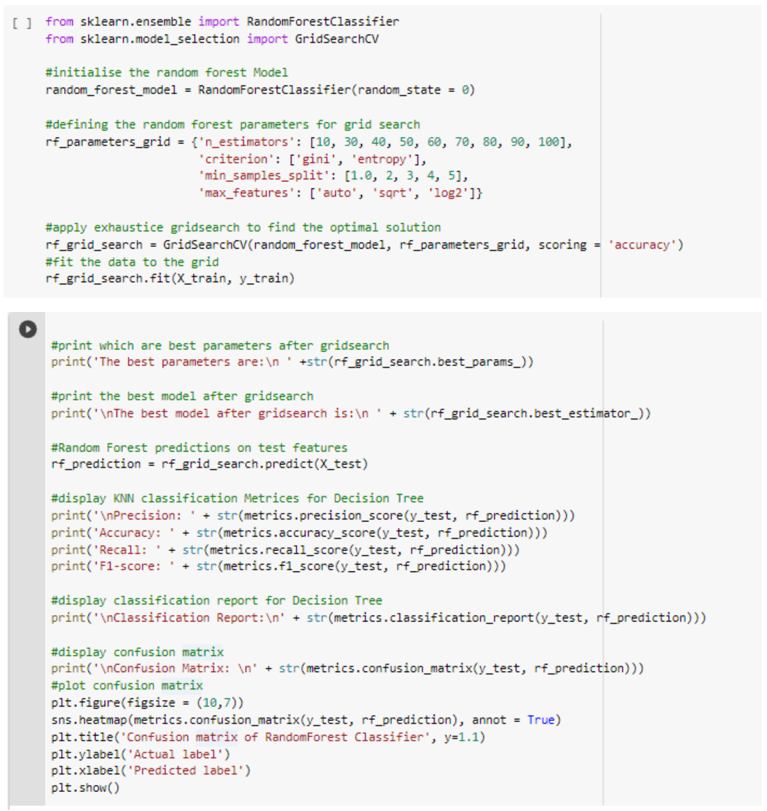
By considering the confusion matrix it's clear that the prediction is very much accurate. Out of the 52 patients who has CKD, only 3 are predicted as non-CKD. Out of the 28 non-CKD patients, only one was wrongly predicted to have CKD.

Since the overall accuracy is 95%, this classifier is a good module. But 3 CKD patients are predicted as non-CKD, which might not be a good evaluation. Therefore the random forest classifier is used.

Random Forest Classifier

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use algorithm. A forest is comprised of trees. It is said that the more trees it has, the more robust a forest is. Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It also provides a pretty good indicator of the feature importance.

For the training and evaluation of the random forest classifier the following steps were carried out. They are similar to the procedure as in decision.

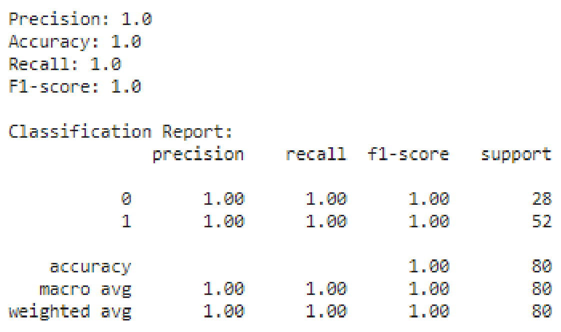


```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.model_selection import GridSearchCV

    #initialise the random forest Model
    random_forest_model = RandomForestClassifier(random_state = 0)

    #defining the random forest parameters for grid search
    rf_parameters_grid = {'n_estimators': [10, 30, 40, 50, 60, 70, 80, 90, 100],
                          'criterion': ['gini', 'entropy'],
                          'min_samples_split': [1.0, 2, 3, 4, 5],
                          'max_features': ['auto', 'sqrt', 'log2']}

    #apply exhaustice gridsearch to find the optimal solution
    rf_grid_search = GridSearchCV(random_forest_model, rf_parameters_grid, scoring = 'accuracy')
    #fit the data to the grid
    rf_grid_search.fit(X_train, y_train)
```

```
#print which are best parameters after gridsearch
print('The best parameters are:\n ' +str(rf_grid_search.best_params_))

#print the best model after gridsearch
print('\nThe best model after gridsearch is:\n ' + str(rf_grid_search.best_estimator_))

#Random Forest predictions on test features
rf_prediction = rf_grid_search.predict(X_test)

#display KNN classification Metrices for Decision Tree
print('\nPrecision: ' + str(metrics.precision_score(y_test, rf_prediction)))
print('Accuracy: ' + str(metrics.accuracy_score(y_test, rf_prediction)))
print('Recall: ' + str(metrics.recall_score(y_test, rf_prediction)))
print('F1-score: ' + str(metrics.f1_score(y_test, rf_prediction)))

#display classification report for Decision Tree
print('\nClassification Report:\n' + str(metrics.classification_report(y_test, rf_prediction)))

#display confusion matrix
print('\nConfusion Matrix: \n' + str(metrics.confusion_matrix(y_test, rf_prediction)))
#plot confusion matrix
plt.figure(figsize = (10,7))
sns.heatmap(metrics.confusion_matrix(y_test, rf_prediction), annot = True)
plt.title('Confusion matrix of RandomForest Classifier', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
```

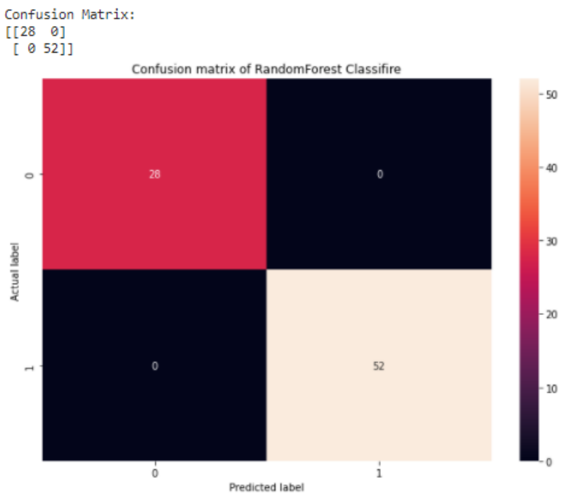
Figure 7: Random Forest Classifier

The classification report and the confusion matrix is given below.



```
Precision: 1.0
Accuracy: 1.0
Recall: 1.0
F1-score: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       1.00      1.00      1.00        52

    accuracy                           1.00        80
   macro avg       1.00      1.00      1.00        80
weighted avg       1.00      1.00      1.00        80
```

Figure 8: Classification report for RF



```
Confusion Matrix:
[[28  0]
 [ 0 52]]
```

Figure 9 : Confusion matrix for RF

As shown in figure (), the accuracy of the model is 100% and the precision is 100%. Here also CKD is consider as 1 and non-CKD as 0. In the confusion matrix also it's clear that the prediction is 100% accurate. All the 52 patients who has CKD, are predicted as CKD correctly, and all the 28 non-CKD patients are also correctly predicted as non-CKD.

Since the overall accuracy is 95%, this classifier is a good module. But 3 CKD patients are predicted as non-CKD, which might not be a good evaluation. Therefore the random forest classifier is used.

This module has the best classification and is very much suitable for the medical prediction because, prediction of a disease is very critical and thus it should be 100% accurate.

DECISION ON MODEL SELECTION

The figure () gives the comparison of the 4 classifiers on the basis of accuracy for the prediction of CKD in patients.

Accuracy from KNN: 78.75 %
Accuracy from DTC: 95%

Accuracy from RF: 100%
Accuracy from SVC: 97.5%

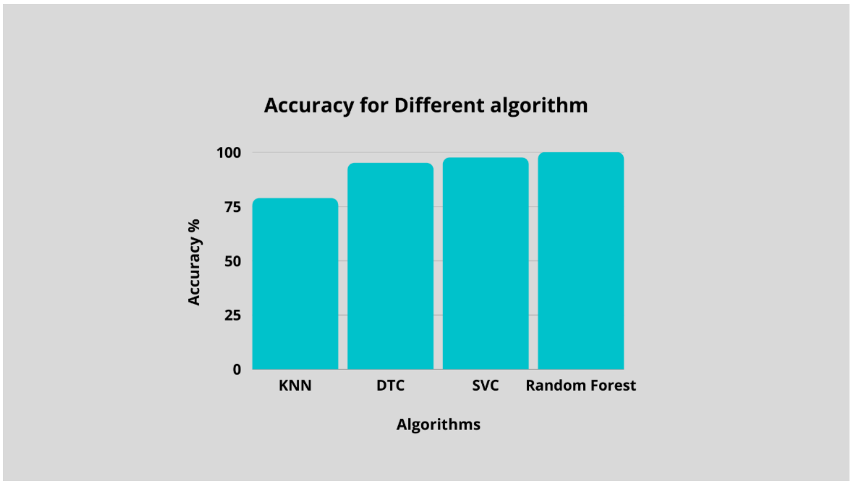


Figure 10: Accuracy for different algorithms

As you can see in the graphs, random forest, decision tree classifiers and support vector machine classifier produce very good results. But the best accuracy is of Random forest which is 100%. Therefore random forest algorithm is used as the model training algorithm. It is the best algorithm for the prediction of chronic kidney disease in patients