Data Mining - 20595

Maman 21

Yehonatan Simian
206584021

Question 1

Problem definition and data preparation stages are described below.

Stage 1 - Data mining goals

The goal of this work is to predict, in a somewhat reliable way, the existence **chronic kidney disease** (ckd) in a given subject, out of given set of categories. Meaning being able to receive data divided to specific categories, such as subject's age and blood pressure, and decide whether it has a chronic kidney disease or not.
The assumptions are very minimal: we assume that the is *some* correlation between the categories in the data-set and the existence of ckd.

- We *do not* assume that the categories are independent. This will be reflected in the KDD steps.
- We *do not* assume that the data-set is 100% correct. This will be reflected in the KDD steps (correcting data, removing categories) and when choosing the classification method(s).

I did not understand the question about **the abstractions** I'm using, but if the answer is Python (as an abstraction of the algorithms) - so be it.

Stage 2 - Data definition

The data is taken from UCI Maching Learning Repository, whose source is a Apollo Hospitals in India. It contains 400 records of patients with data of 25 categories of information such as blood tests results and medical conditions.
The categories / attributes are described in `Table 1` below. A word about terminology: the *categories* I'm referring to are the columns of the data-set, e.g. Age, Hypertension etc. The *attributes* I'm referring to are the categories' attributes, e.g. type, range and maybe more. I've added a short description to non-trivial categories, such as Specific Gravity and Hypertension, so I'd have a better idea about the data I'm dealing with.

Some observations that can be made immediately by looking at the table below:

- The **measure units** are **irrelevant** because we are not doctors, we are data-scientists; We care more about the data in terms of numbers, categories, mean and variance rather than measure units, as for they grant us no useful information *per se*.
- We can notice that most of the categories can be classified as below:
  - **Blood related attributes**: Blood Pressure; Specific Gravity; Albumin; Sugar; Red Blood Cells; Pus Cell; Pus Cell clumps; Bacteria; Blood Glucose Random; Blood Urea; Serum Creatinine; Sodium; Potassium; Hemoglobin; Packed Cell Volume; White Blood Cell Count; Red Blood Cell Count.
  - **Illnesses / Conditions**: Hypertension; Diabetes Mellitus; Coronary Artery Disease; Pedal Edema; Anemia.
  - **Others**: Age; Appetite; Class.
- As I said, I've added a short description to non-trivial categories in order to learn about my data's nature; Note this quote I ran into while searching for the meaning of **Serum Creatinine**:

  "Your serum creatinine [...] tells how well your kidneys are working. When your kidneys are not working well, your serum creatinine level goes up."
  ~American Kidney Fund.

  This information might affect the importance of Serum Creatinine in the final decision tree.

One observation that cannot be derived from the table below: the are some **defects** in the given data-set; Some data is missing or corrupted. More about handling these defects in the KDD steps.

Abbr	Category[: Description]	Type	Range	Comments
age	Age	numerical	1 - 99	increments of 1
bp	Blood Pressure	numerical	10 - 200	increments of 10
sg	Specific Gravity: measure of density of blood	numerical	1.005 - 1.025	increments of 0.005
al	Albumin: blood protein	nominal	0 - 5	increments of 1
su	Sugar	nominal	0 - 5	increments of 1
rbc	Red Blood Cells	nominal	normal,abnormal	
pc	Pus Cell	nominal	normal,abnormal	
pcc	Pus Cell Clumps	nominal	present,notpresent	
ba	Bacteria	nominal	present,notpresent	
bgr	Blood Glucose Random	numerical	1 - 500	increments of 1
bu	Blood Urea	numerical	1 - 400	increments of 1
sc	Serum Creatinine: kidneys-related product	numerical	0 - 100	increments of 0.1
sod	Sodium	numerical	1 - 200	increments of 1
pot	Potassium	numerical	1 - 50	increments of 0.1
hemo	Hemoglobin	numerical	1 - 20	increments of 0.1
pcv	Packed Cell Volume	numerical	1 - 100	increments of 1
wc	White Blood Cell Count	numerical	1,000 - 30,000	increments of 100
rc	Red Blood Cell Count	numerical	2 - 9	increments of 0.1
htn	Hypertension: high blood pressure	nominal	yes,no	
dm	Diabetes Mellitus: glucose-related disease	nominal	yes,no	
cad	Coronary Artery Disease: bad blood supply to the heart	nominal	yes,no	
appet	Appetite	nominal	good,poor	
pedal	Pedal Edema: build-up of fluid in the body's tissue	nominal	yes,no	
ane	Anemia: not enough healthy red blood cells	nominal	yes,no	
class	Class	nominal	ckd,notckd	

Table 1: Table of data categories and attributes.

Stage 3 - KDD steps description

The KDD steps are:

1. **Goals Description** - Described in `Stage 1`.
2. **Data Integration** - This includes the data gathering described in `Stage 2`. I hereby mention that one and only source of data is used in this data mining exercise, thus no integration between different sources is required. Furthermore, the data I'll be working on was received in a `.arff` file. I have converted it to a `.csv` file, and will be working on it as my data-set.
3. **Data Cleaning** - This shall be a major step in the mining, as for there is much data to be cleaned, such as:
   - Missing values, represented as `?` in the data-set;
   - Incorrect data, probably the cause of a human mistake (mistyping);
   - Non-independent categories (at least at first sight);
   - One unnecessary categories that might sabotage the results if not removed;

   More about the cleaning process in `Stage 5`.
4. **Data Transformation** - There are a few transformations that need to be made:
   - As mentioned before, transforming the `.arff` file to a `.csv` file;
   - In contrast to Maman11, I *will not* binarize the data; After examining the data-set, some defects have been found, such as columns that contain " yes" instead of "yes". However, willing that the final decision tree will be more readable to humans, I will leave binary text data as is, e.g. "normal" and "abnormal", as for "0" and "1" might be confusing to human readers, and a fortiori non-binary data such as Appetite.
   - I will use Python to trimming and lower-casing the data in order to fix defects such as the one mentioned above;
   - I will also use Python to normalize and discretize the data. Discretization (binning) is essential, especially with low-ranged numerical categories such as Age and Blood Urea;

   More about the transformation process in `Stage 5`.
   Note: data reduction is unnecessary as for the relatively low amount of records in the data-set.
5. **Means Selection** - For the transformations and the execution of the data-mining algorithm themselves, I will use Python, and specifically the libraries **Pandas** and **Sklearn**, as for they are well equipped with data-mining tools and algorithms. Some other libraries that worth mentioning and might be used as necessary are Numpy, Scipy, and Graphviz. Moreover, training models must be selected as well. All described within `Stage 4`.

Stage 3 continued on next page...

6. **Models Execution** - After cleaning the data and choosing models, the algorithms themselves must be run. Fortunately, Python's library Sklearn has every algorithm that we need, So this part should be quick and easy peasy lemon squeezy.
   The execution should be made on ”training data” and ”testing data” separately. However, since the data-set is quite small, I will use a techniques such as **ensemble learning** and **cross-validation** in order to enhance the performance of the models, and use the same data-set for the training as well as the testing.

7. **Results Reviewing** - After choosing models and training them on the data-set, I will use Python again to view the results graphically and maybe with charts as well. I hereby remind, for the uninformed reader, that the main result should be the decision tree - a model that can classify, hopefully reliably, the existence of ckd based on data given in the other categories (non necessarily all of them).

8. **Conclusions Drawing** - After drawing graphs, we need to draw conclusions as well. We'll see if and how much are the different decision trees differ, and how high are their accuracy and precision. We'll try to understand the reasons for these results, if they are good enough to use them in real life to predict ckd, and if not - what other preparation stages could be made in order to receive more reliable results.

Stage 4 - Data mining alternatives overview

Four data mining alternatives algorithms are over-viewed below.

- **Linear regression.** This method is based on the dependency of categories (i.e. the existence of ckd is linearly depended on the other categories). It it often used to fit a predictive model to an observed data set of values, if the goal is error reduction, or quantify the strength of the relationship between the response and the explanatory variable - if the goal is to explain variation in the response variable.
  Pros: Easy to implement and to understand.
  Cons: Based on linear dependency of the different categories, which we cannot rely on in our specific scenario.

- **ID3.** A greedy algorithm that builds a decision tree, and splits the data-set's records for each split in the tree, based on a given criteria (Information Gain). Being based on information gain (IG), the data must be discretized, otherwise the entropy won't provide us much information.
  Pros: Easy to implement and to understand. Higher accuracy for correctly classified data.
  Cons: Slow, data must be discretized, can overfit very easily.

- **C4.5.** This algorithm tries to improvise, adapt and overcome the cons of ID3's overfitting problem. This is done using Gain Ratio criteria instead of IG. Besides that, similar to ID3.
  Pros: Faster than ID3, no need to discretize, higher precision and accuracy than ID3 for incorrectly classified data.
  Cons: The splits in the tree are less balanced.

- **Cart.** This algorithm can handle both nominal and numeric data attributes. It is based on Gini Index as criteria, and can deal with missing values (tho will be unnecessary in our scenario). It also uses post pruning, in contrast to the other mention decision trees, which generally reduces overfitting.
  Pros: Fits our data well, have generally high precision and accuracy. Generally reduces overfitting.
  Cons: Average run-time.

Stage 5 - Data preparation steps description

The data preparation steps are described below:

1. Transferring the `.arff` file to `.csv` that I can work with using Python;
2. Editing the `csv` file manually using MS Excel: record #370 has an offset the 6 last values in the row; I have moved them manually to their correct position (one cell left).
3. Changing the one and only ”no” value of appetite to ”poor”, as for this must be a typo.
4. Removing the first column of the data sheet (id);
5. Removing `Blood Pressure` category; This category is strongly related to `Hypertension`. However, high pressure value do not seem to correlate with the hypertension ones. This can be a cause of hypertension being a long-term medical condition, whereas high blood-pressure is a short-term that can change within minutes. We try to predict ckd, which is a long-term medical condition (a disease), therefore hypertension seems like a more relevant classifier, and blood-pressure can be abandoned this time.
6. I have decided *not* to delete `Red Blood Cells` from the data-set; At first it seemed like a duplication of data, along with the `Red Blood Cell Count` category. But after reading a bit online, it seems that the former is *qualitative* while the latter is *quantitative*. Therefore they are different and may be independent, and therefore non of them have to be removed.
7. I wanted to give `Serum Creatinine` a higher influence on the data mining process, as for the quote that I presented in `Stage 2`. After thinking a while I have decided that if this relation actually exists, it shall be reflected in the results, and I might sabotage the results if interfered with the data; Therefore no change in the data is made regarding this category.
8. From now on, the data will be handled using Python; The integration to Python was quick thanks to Pandas’ read_excel method.
9. Lower-casing the entire data (although no such mistakes have been noticed by me);
10. Trimming the entire data value from unnecessary spaces (some have been noticed by me);
11. Handling records with data to far from the mean (our-of-range); It’s hard to decide a range by hand, but after investigating and observing highly-unusual data values, I’ve chosen the following ranges as thresholds:
    - Blood Glucose Random: 50 - 500 instead of 1 - 500;
    - Serum Creatinine: 0 - 20 instead of 0 - 100;
    - Sodium: 100 - 200 instead of 1 - 200;
    - Potassium: 1 - 10 instead of 1 - 50;

Stage 5 continued on next page. . .

- While Blood Cell Count: 2,500 - 25,000 instead of 1,000 - 30,000;

After this step, 1 + 4 + 1 + 2 + 2 = 10 values have been replaced with ? marks.

12. Missing values will be handled as follows: Since there are $25-2 = 23$ categories, the threshold to rows deletion should be $\sqrt{23} \approx 5$ missing values. Hence, rows with 5 missing values (? values) seems like a good threshold; However, this means deletion of 94 rows, almost a quarter of our data-set (which was not big in the beginning). I want to go on the safe side, therefore **no rows will be deleted**. The missing values will be **replaced with the mean** of the category, or for nominal categories, that would be the **mode** (most occurred value).
Note: perhaps a better solution would be to use variance to see which missing values can be replaced with the mean, and which categories has too high variance. However, this would required choosing more arbitrary thresholds and data manipulation, which I'm trying to reduce as for data-set's low size.

13. Discretazation (binning) is unnecessary; Not manually anyway. This action would require more arbitrary binning thresholds. However, I've searched and found out that Sklearn's DecisionTreeClassifier already does implicit-binning (to be more precise: ordinalizing) by itself as needed, based on pre-calculated thresholds. Thus, I shall let the algorithm decide it by itself.

14. Normalization is unnecessary for similar reason as above;

The last step of the data preparation stage is providing a graphic view of the fixed data-set, before taken to execution. In the next page, charts of all categories will be shown. Some observations and questions might arise when looking at the charts, such as:

- Is the correlation of Blood Glucose Random, Blood Urea and Serum Creatinine a coincidence?
- Is the correlation of Hemoglobin and Packed Cell Volume a coincidence?
- Should have I deleted the White Blood Cell Count and Red Blood Cell Count? This question raises because of the influence of the mean value, assigned to missing values, on the distribution of data (the column with the mean value is the highest by far).
- Aren't these graphs look awesome af?

The answer to the first two questions might be easily a solid "no"; Correlation does not necessarily imply causation, and those categories might just distribute the same way by nature (geometric distribution for the former, normal distribution for the latter).
Note: I have converted the "good/poor" values of Appetite to "yes/no" values in order to fit the chart of binary categories.

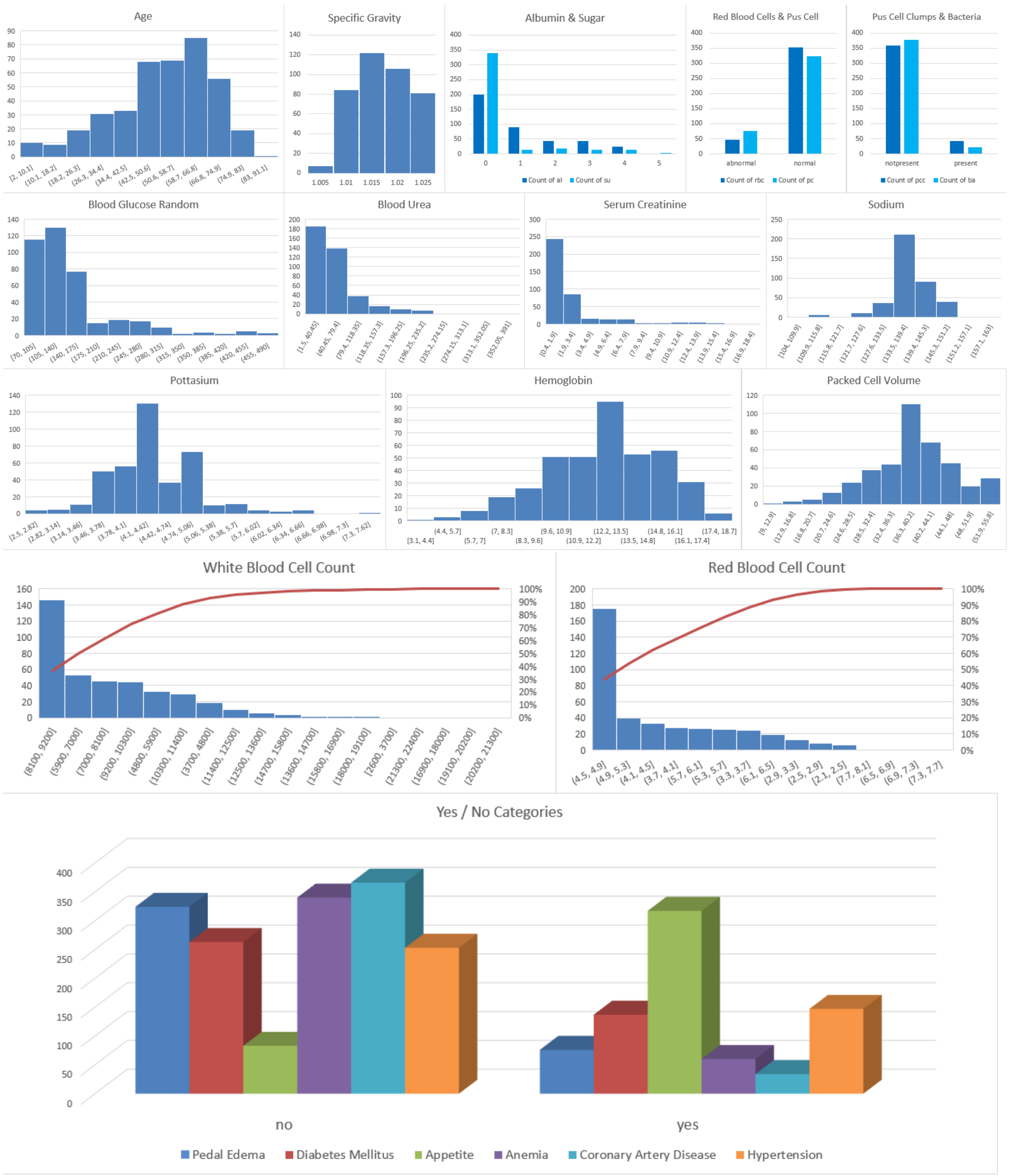


Figure 1: Charts of the prepared data

Question 2

Data classification stages are described below.

Stage 1 - Choosing and describing two classification methods

So. . . which data mining alternatives should we choose?!
I hereby remind that I am required to choose only 2 methods. Also, prediction of ckd is our goal, which means that we want a method that fits our data attributes well, and has high accuracy and precision as well. Note that algorithm's speed is irrelevant at all when trying to predict a disease. Linear regression seemes like the least relevant method to our scenario, so I won't choose it. C4.5 seems like a better version of ID3 to our data types. In `Stage 1`, I emphasized that I *do not* assume that the data-set is 100% correct (especially after cleaning and seeing how many defect values exist). Hence, a method with higher accuracy for incorrectly classified data is better for our scenario. Therefore, I will choose C4.5 over ID3 as the first classification method. The second method shall be Cart, as for is has many pros and no relevant cons at all.

After choosing classification methods, let's not forget the existence of **bagging** and **random forests**; Those two methods are ensemble learning techniques, used to improve the performance of the chosen models, and reduce overfitting. Thus, I want to use at least one of them on the chosen models. Since I am required to choose only 2 models, and I use only one ensemble algorithm that'll be applied to both models, hence differences in the results are more likely to be caused by the nature of the algorithm themselves and not the ensemble learning technique. I will arbitrarily choose random forests as my ensemble technique.

Another topic before heading to the winners of the contest: **cross-validation**. This technique is used to evaluate the performance of a decision tree model by partitioning the data-set into multiple subsets, where one subset is used for testing and the remaining subsets are used for training the model. I will use Sklearn automatic cross-validation tools because it is faster than splitting the data myself, and has more features than I can provide.

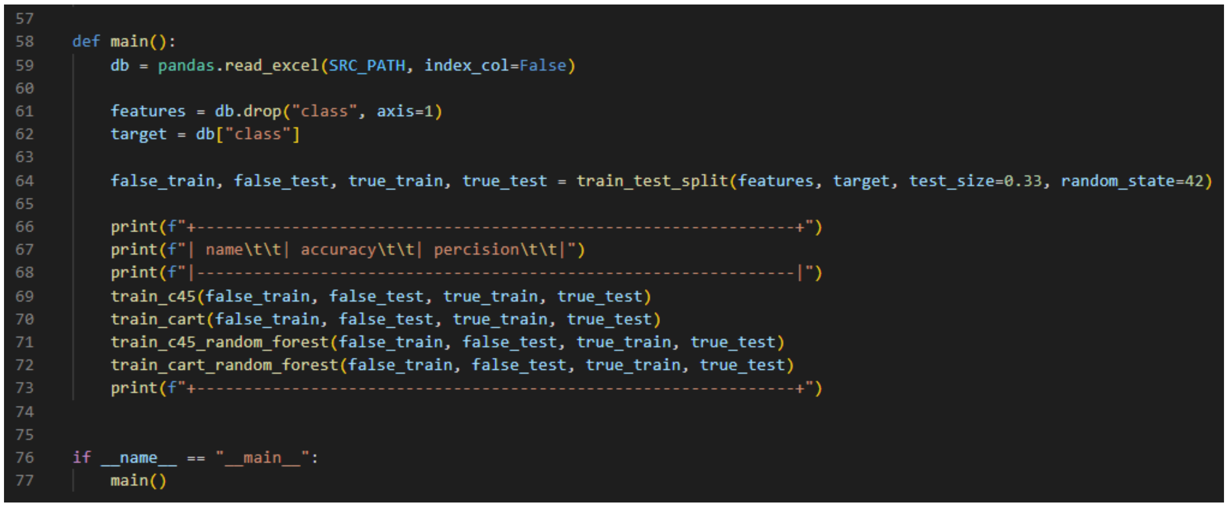
To summarize: The data mining alternative I have chosen are **C4.5** and **Cart**, along with **random forests** and **cross-validation** enhancements for better results.

Stage 2 - Describing the classification methods' steps

The steps for both methods are pretty similar:

- **Step #1:** Load the data-base to Python using Pandas.
- **Step #2:** Use Sklearn's train_test_split method for cross-validation.
  *Note: in hindsight, it appears that this method is **not** cross-validation. I could have used KFold instead, but this work is getting too large anyway, so I gave up on it.*
- **Step #3:** Use Sklearn's RandomForestCalssifier method to build the tree(s).
- **Step #4:** Analyze the accuracy and precision scores.
- **Step #5:** Use Graphviz to visualize the results.

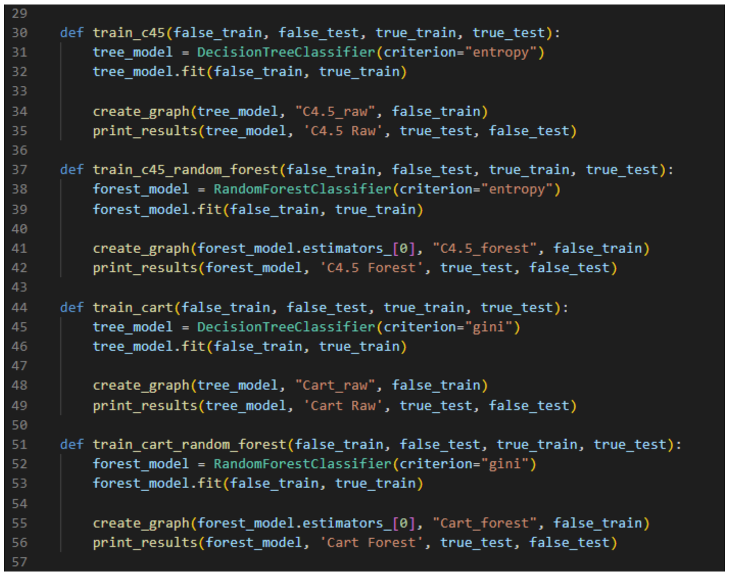
Here's some python code behind it:



```python

def main():
    db = pandas.read_excel(SRC_PATH, index_col=False)

    features = db.drop("class", axis=1)
    target = db["class"]

    false_train, false_test, true_train, true_test = train_test_split(features, target, test_size=0.33, random_state=42)

    print(f"+---------------------------------------------------------------+")
    print(f"| name\t\t| accuracy\t\t| percision\t\t|")
    print(f"|---------------------------------------------------------------|")
    train_c45(false_train, false_test, true_train, true_test)
    train_cart(false_train, false_test, true_train, true_test)
    train_c45_random_forest(false_train, false_test, true_train, true_test)
    train_cart_random_forest(false_train, false_test, true_train, true_test)
    print(f"+---------------------------------------------------------------+")


if __name__ == "__main__":
    main()
```

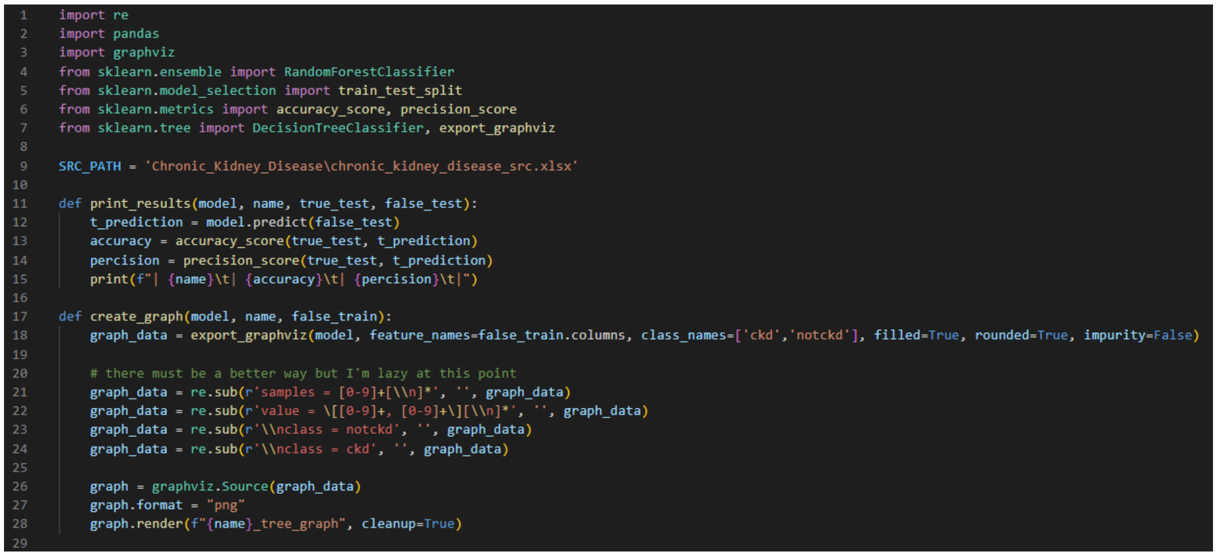
Figure 2: Classification methods code overview (Steps #1 and #2)

As can be seen below, the code for training the different models is very similar.



```python

def train_c45(false_train, false_test, true_train, true_test):
    tree_model = DecisionTreeClassifier(criterion="entropy")
    tree_model.fit(false_train, true_train)

    create_graph(tree_model, "C4.5_raw", false_train)
    print_results(tree_model, 'C4.5 Raw', true_test, false_test)

def train_c45_random_forest(false_train, false_test, true_train, true_test):
    forest_model = RandomForestClassifier(criterion="entropy")
    forest_model.fit(false_train, true_train)

    create_graph(forest_model.estimators_[0], "C4.5_forest", false_train)
    print_results(forest_model, 'C4.5 Forest', true_test, false_test)

def train_cart(false_train, false_test, true_train, true_test):
    tree_model = DecisionTreeClassifier(criterion="gini")
    tree_model.fit(false_train, true_train)

    create_graph(tree_model, "Cart_raw", false_train)
    print_results(tree_model, 'Cart Raw', true_test, false_test)

def train_cart_random_forest(false_train, false_test, true_train, true_test):
    forest_model = RandomForestClassifier(criterion="gini")
    forest_model.fit(false_train, true_train)

    create_graph(forest_model.estimators_[0], "Cart_forest", false_train)
    print_results(forest_model, 'Cart Forest', true_test, false_test)

```

Figure 3: Training models code (Step #3)



```python
import re
import pandas
import graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score
from sklearn.tree import DecisionTreeClassifier, export_graphviz

SRC_PATH = 'Chronic_Kidney_Disease\chronic_kidney_disease_src.xlsx'

def print_results(model, name, true_test, false_test):
    t_prediction = model.predict(false_test)
    accuracy = accuracy_score(true_test, t_prediction)
    percision = precision_score(true_test, t_prediction)
    print(f"| {name}\t| {accuracy}\t| {percision}\t|")

def create_graph(model, name, false_train):
    graph_data = export_graphviz(model, feature_names=false_train.columns, class_names=['ckd','notckd'], filled=True, rounded=True, impurity=False)

    # there must be a better way but I'm lazy at this point
    graph_data = re.sub(r'samples = [0-9]+[\\n]*', '', graph_data)
    graph_data = re.sub(r'value = \[[0-9]+, [0-9]+\][\\n]*', '', graph_data)
    graph_data = re.sub(r'\\nclass = notckd', '', graph_data)
    graph_data = re.sub(r'\\nclass = ckd', '', graph_data)

    graph = graphviz.Source(graph_data)
    graph.format = "png"
    graph.render(f"{name}_tree_graph", cleanup=True)

```

Figure 4: Analysis and visualization code (Steps #4 and #5)

Stage 3 - Reporting the classification methods' results

After viewing the code responsible for the results, it's time to show the results themselves!
Note: the following trees are one sample each of the corresponding random forest.

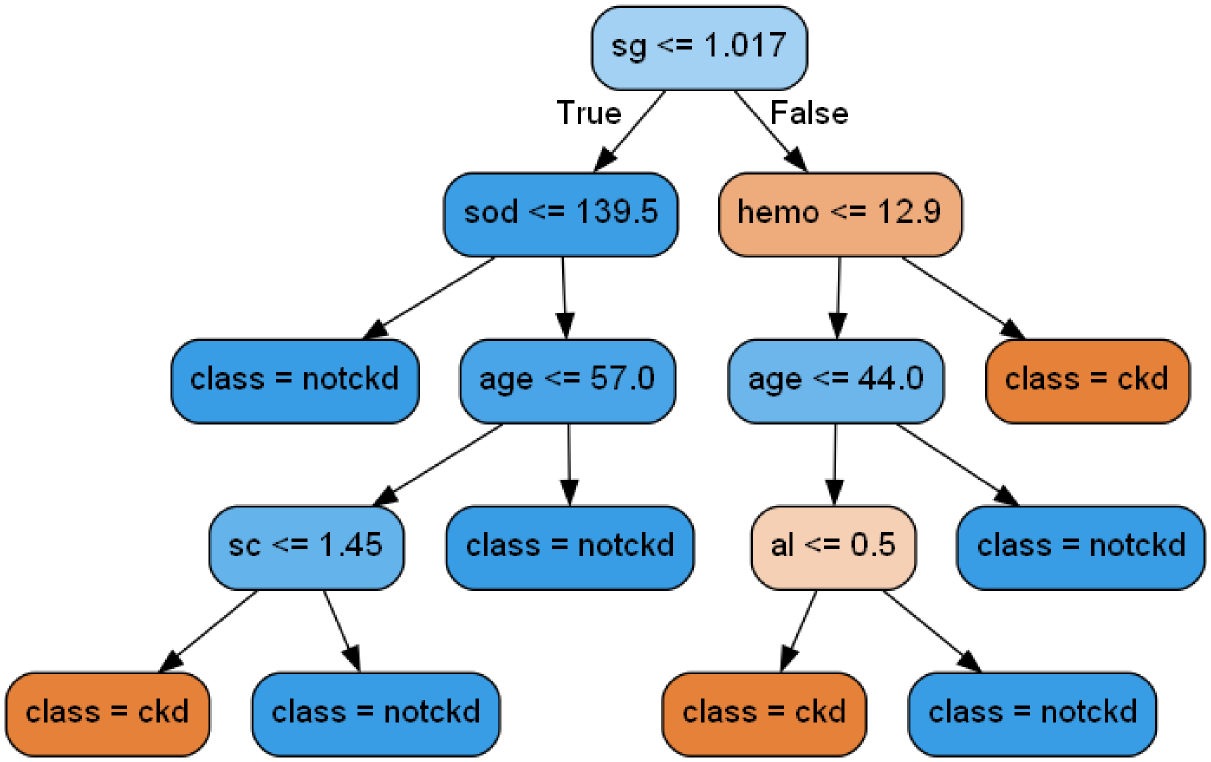


Figure 5: C4.5 decision tree

Stage 3 continued on next page. . .

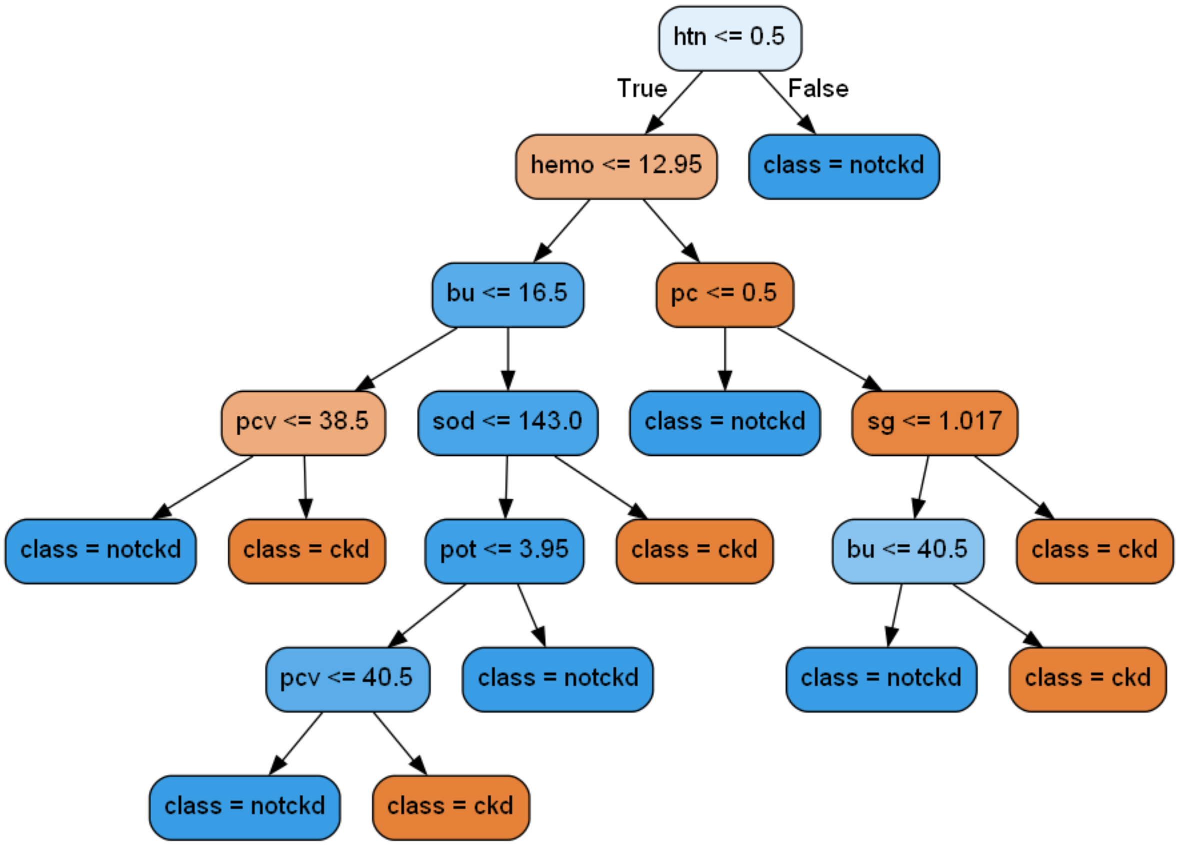
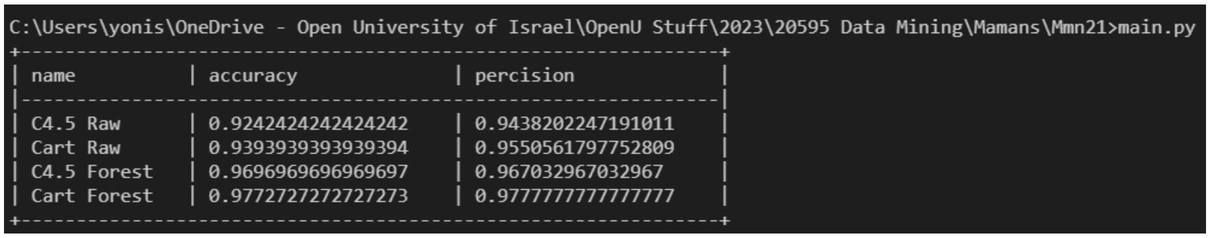


Figure 6: Cart decision tree

Stage 4 - Evaluating the classification methods' accuracy

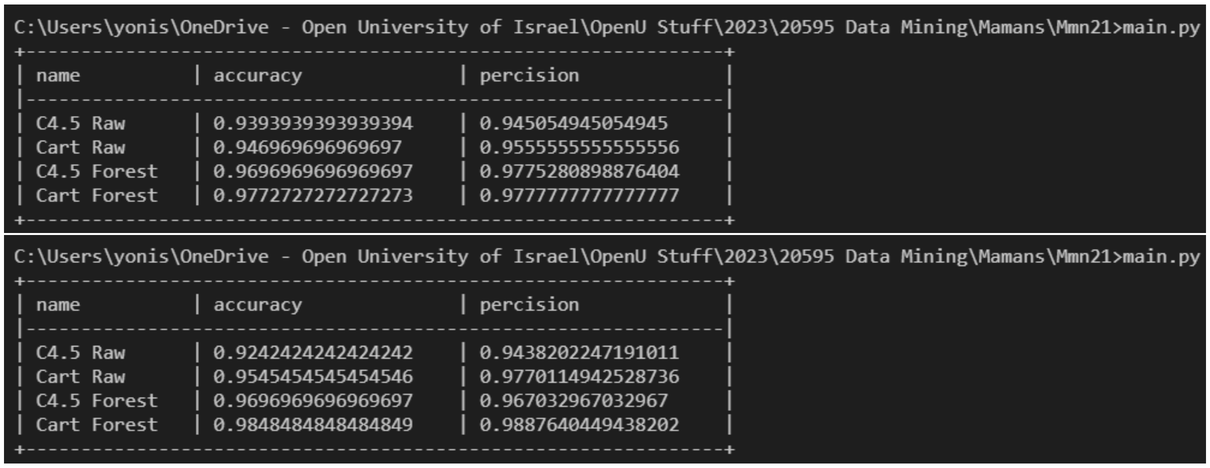
Out of pure curiosity (and because it's a change of merely few lines in Python), I've ran the algorithms without random forests as well, to see if my claims about it's accuracy and precision were accurate and precise. The results were not surprising:



```
C:\Users\yonis\OneDrive - Open University of Israel\OpenU Stuff\2023\20595 Data Mining\Mamans\Mmn21>main.py
+--------------------------------------------------------------+
| name          | accuracy               | percision           |
|--------------------------------------------------------------|
| C4.5 Raw      | 0.9242424242424242     | 0.9438202247191011  |
| Cart Raw      | 0.9393939393939394     | 0.9550561797752809  |
| C4.5 Forest   | 0.9696969696969697     | 0.967032967032967   |
| Cart Forest   | 0.9772727272727273     | 0.9777777777777777  |
+--------------------------------------------------------------+
```

Figure 7: Accuracy and precision evaluations

More executions show similar results:



```
C:\Users\yonis\OneDrive - Open University of Israel\OpenU Stuff\2023\20595 Data Mining\Mamans\Mmn21>main.py
+--------------------------------------------------------------+
| name          | accuracy               | percision           |
|--------------------------------------------------------------|
| C4.5 Raw      | 0.9393939393939394     | 0.945054945054945   |
| Cart Raw      | 0.946969696969697      | 0.9555555555555556  |
| C4.5 Forest   | 0.9696969696969697     | 0.9775280898876404  |
| Cart Forest   | 0.9772727272727273     | 0.9777777777777777  |
+--------------------------------------------------------------+
```

```
C:\Users\yonis\OneDrive - Open University of Israel\OpenU Stuff\2023\20595 Data Mining\Mamans\Mmn21>main.py
+--------------------------------------------------------------+
| name          | accuracy               | percision           |
|--------------------------------------------------------------|
| C4.5 Raw      | 0.9242424242424242     | 0.9438202247191011  |
| Cart Raw      | 0.9545454545454546     | 0.9770114942528736  |
| C4.5 Forest   | 0.9696969696969697     | 0.967032967032967   |
| Cart Forest   | 0.9848484848484849     | 0.9887640449438202  |
+--------------------------------------------------------------+
```

Figure 8: More evaluations...

As can be seen, the accuracy and ~~percision~~ precision scores are both very high. In addition, the claims that random forests yield higher scores were right, at least in this specific case. Furthermore, it seems like the Cart algorithm have yielded better results then C4.5. More about comparisons and conclusion in the next and final step!

Stage 5 - Analyzing the classification methods' results

So, what did we see here?

- The trees looked a little bit different, and different executions have yielded different trees;
- Hemoglobin was a second split in both trees, but besides that the first split categories where quite different;
- The accuracy and precision scores were very high, no lower than 0.96 in the forests!
- Cart algorithm have yielded better results than C4.5;
- Both algorithms ran within a short amount of time;
- Serum Creatinine had lower impact than I thought it would have;

Conclusions:

- The reason for differentiation between the trees might be the data-set's small size; Overall, the trees were all pretty accurate despite the differences, so maybe a small-sized data yield different yet accurate results (at least accurate to the small size of data tested on it).
- Using C4.5 and Cart was a good choice, I've also tested ID3 and it had lower scores;
- Cart was a little bit better for this task, so it's good to choose an algorithm that fits the given data-types well;
- Not provide larger weight to Serum Creatinine was a good choice, even though it (semantically) had a potential to be a large impact on the results;
- Runtime does not matter when running on small data-sets;
- MS Excel is pretty darn awesome.

Suggestions for improvements:

- A large data-set is definitely essential for data-mining, especially when the results are as important as prediction of diseases.
- I tried to plan ahead before actually doing any work, but it appeared to me that the more I work on it, the more I understand my task. E.g. Producing the charts *before* planning what to do with the data could be effective, and even producing charts of the *raw* data could help me figure what's the best way to handle the missing data.
- Me being me, I can't finish a work without some memes. I've had fun, now let's enjoy!

Stage 5 continued on next page. . .

"Insanity is doing the same thing over & over again & expecting different results."
BIG DATA?
YEAH I WAS INTO DATA BEFORE IT WAS BIG
*MACHINE LEARNING:
PyTorch
Data Scientist
pandas
numpy
sklearn
matplotlib
DATA

Figure 9: Data science memes