END OF TERM REPORT

CHRONIC KINDEY DISEASE CLASSIFICATION

MACHINE LEARNING PROJECT 4DS1

From 27th October to 15th December

Realized by:

Yasmine Daly

Moudhaffer Bouallegui

Molka Louati

Oussama Sellami

Ramy Ben Hassine

Table of contents:

Abbreviations

CRISP-DM	**Cross-Industry Standard Process for Data Mining**
KNN	**K-Nearest Neighbour**
SVM	**Support Vector Machine**
CKD	**Chronic Kidney Disease**
CFS	**Correlation-Based Feature Selection**
RFE	**Recursive Feature Elimination**
RFECV	**Recursive Feature Elimination Cross Validation**

General introduction

Chronic kidney disease (CKD) is a global health issue with a high rate of morbidity and mortality and a high rate of disease progression. Because there are no visible symptoms in the early stages of CKD, patients frequently go unnoticed. The early detection of CKD allows patients to receive timely treatment, slowing the disease's progression. Due to its rapid recognition performance and accuracy, machine learning models can effectively assist physicians in achieving this goal. We propose a machine learning methodology for the CKD diagnosis in this paper. This information was completely anonymized. As a reference, the CRISP-DM model (Cross industry standard process for data mining) was used in this study.

The remainder of this report is structured as follows: The first chapter will be presenting the first step which is business understanding, a study of the overall field, a description of the proposed methodology and its goals. Chapter 2 will present the second step, Data understanding. Along those chapters 3 and 4 will describe the third step data preparation for both of the articles. Then, chapters 5 and 6 will contain a detailed explanation of the modelling phase for both of the articles. Followed by Evaluation and Deployment chapters. Finally, we conclude with the enhancements chapter that describes our added value to this study.

A general conclusion summarises the work done and presents possible perspectives.

1. Business understanding

1.1. Introduction

During this chapter, we will present the host company and its areas of activity. In a later section, we are going to talk about the general context of the project, study the existing situation and its downsides which lead us to describe the proposed solution and its objective and then, we will present the requirements of the project and the analysis of the solution. Finally, we decide on the methodology of work to be adopted to complete our project.

1.2. Overview

Chronic kidney disease (CKD) is one of the leading causes of death in recent years, according to a report by the Global Burden of Disease. One in every seven persons has CKD, one of the undiscovered illnesses that have the greatest influence on patients' quality of life and increase the chance of death significantly. The general system of social security in health (SGSSS) has taken chronic kidney disease (CKD) into account, as a high-cost pathology for generating a powerful economic impact on the finances of the system, causing a dramatic effect on the quality of life of the patient and their family, including employment repercussions. To reduce the high mortality of CKD, research should be deepened and directed to the initial stages of the disease, analysing its risk group, with the help of laboratory tests, seeking that patients do not reach the final stages such as dialysis, transplantation, or death. Through automatic learning, the aim is to find a valuable contribution so that an early classification of the disease can be carried out in its initial stages through the results of clinical laboratories, taking advantage of the great potential of automatic learning in the analysis and classification of the data. It is necessary that the technical help tools that are based on data can support the decision-making process in the initial diagnoses quickly, with high precision, and at low cost. With them, the time required for diagnosis is reduced, allowing the patient to receive treatment for the disease before it progresses to a stage of no return.

1.3. Objective

The major objective is to diagnose CKD at the early stages with the least possible tests and cost and with a high accuracy rate. This project also aims to effectively handle the missing values, present in the CKD data set with mean for the numerical variables and with mode for the categorical variables. Feature selection is also performed with the help of information gain to find the most important features that play a vital role in

detecting CKD. Various machine learning algorithms are applied and analysed to detect CKD and the best one with the best performance and accuracy rate is found.

1.4. Proposed methodology

The proposed methodology involves using the CRISP-DM methodology. In fact, it is an agile and iterative method. Each iteration brings additional business knowledge which makes it easier to tackle the next iteration.

The figure below shows the 6 phases of the life cycle process model.

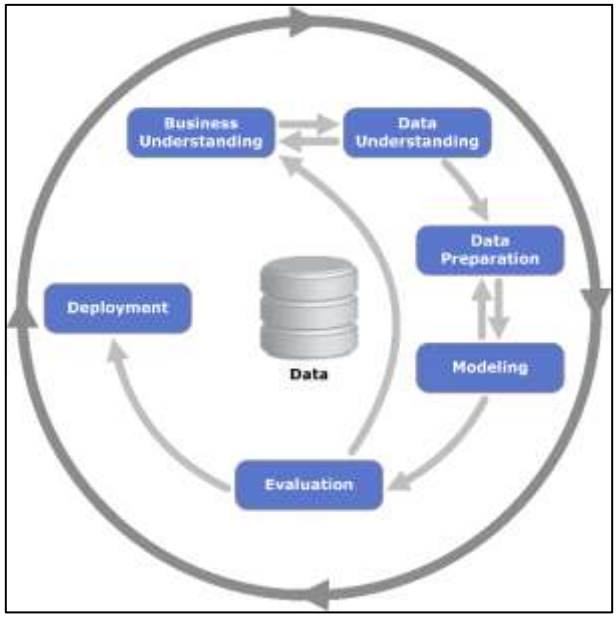


Figure 1: CRISP-DM life cycle

1.5. Summary

This chapter presented the first step of the CRISP-DM method, which is the business understanding. Also, it presented an overview for the overall business, the objective of this study and finally the proposed methodology.

2. Data understanding

2.1. Introduction

After having a business understanding, in this chapter we will focus on the second phase which is data understanding in which we will find out what data we have, what is our data and if our data is of quality.

2.2. Data

This study uses data from the UCI repository. The data collected from the Apollo Hospital, India by b. Jerlin Rubini. The number of instances in the dataset is 400, 250 instances were chronic kidney disease (ckd) and 150 instances were non chronic kidney disease (notckd). There were 25 attributes, one of them was class attribute.

2.3. Attributes

The dataset contains 25 features; 24 with our target value which is the diagnosis.

The attributes were abbreviated on the dataset as follows:

In this section, we will present features of our dataset, what they stand for as well as their types.

1.Age

age in years

2.Blood Pressure

bp in mm/Hg

The diastolic pressure is the pressure in the arteries when the heart rests between beats. This is the time when the heart fills with blood and gets oxygen. This is what your diastolic blood pressure number means:

Normal: Lower than 80

Stage 1 hypertension: 80-89

Stage 2 hypertension: 90 or more

Hypertensive crisis: 120 or more.

3.Specific Gravity

sg - (1.005,1.010,1.015,1.020,1.025)

Urine specific gravity is a laboratory test that shows the concentration of all chemical particles in the urine.

4.Albumin

al - (0,1,2,3,4,5)

The albumin urine test measures the amount of albumin in a urine sample. Albuminuria is a sign of kidney disease and means that you have too much albumin in your urine. Albumin is a protein found in the blood. A healthy kidney doesn't let albumin pass from the blood into the urine. A damaged kidney lets some albumin pass into the urine. The less albumin in your urine, the better.

5.Sugar

su - (0,1,2,3,4,5)

The glucose urine test measures the amount of sugar (glucose) in a urine sample. The presence of glucose in the urine is called glycosuria or glucosuria.

Glucose is not usually found in urine. If it is, further testing is needed. Normal glucose range in urine: 0 to 0.8 mmol/l (0 to 15 mg/dL)

6.Red Blood Cells

rbc - (normal,abnormal)

Red blood cells are one of the major components of blood, along with white blood cells and platelets. Red blood cells help carry oxygen throughout the body. A high red blood cell count means the number of red blood cells in your bloodstream is higher than normal. Normal red blood cell counts are:

For men, 4.7 to 6.1 million red blood cells per microliter of blood

For women, 4.2 to 5.4 million red blood cells per microliter of blood

For children, 4.0 to 5.5 million red blood cells per microliter of blood

7.Pus Cell

pc - (normal,abnormal)

8.Pus Cell clumps

pcc - (present,notpresent)

Presence of pus cells in urine defined as pyuria is an important accompaniment of bacteriuria which may be asymptomatic or can indicate toward underlying urinary tract infectio**n.**

9.Bacteria

ba - (present,notpresent)

Presence of bacteria in urine.

10.Blood Glucose Random

bgr in mgs/dl

Random glucose testing is a blood test done at a random moment of the day to check glucose (sugar) levels. values of 200 mg/dL or above can indicate diabetes.

11.Blood Urea

bu in mgs/dl

A blood urea nitrogen (BUN) test measures the amount of nitrogen in your blood that comes from the waste product urea. Urea is made when protein is broken down in your body. Urea is made in the liver and passed out of your body in the urine. A BUN test is done to see how well your kidneys are working. In general, around 6 to 24 mg/dL (2.1 to 8.5 mmol/L) is considered normal.

12.Serum Creatinine

sc in mgs/dl

Creatinine is a waste product that comes from the normal wear and tear on muscles of the body. Everyone has creatinine in their bloodstream. The normal level of creatinine depends on your age, race, gender, and body size. In general , a normal result is 0.7 to 1.3 mg/dL for men and 0.6 to 1.1 mg/dL for women.

13.Sodium

sod in mEq/L

A normal blood sodium level is between 135 and 145 milliequivalents per liter (mEq/L). Hyponatremia occurs when the sodium in your blood falls below 135 mEq/L.

14.Potassium

pot in mEq/L

Normally, your blood potassium level is 3.6 to 5.2 millimoles per liter (mmol/L). A very low potassium level (less than 2.5 mmol/L) can be life-threatening and requires urgent medical attention.

15.Hemoglobin

hemo in gms

A haemoglobin test measures the amount of haemoglobin in your blood. Haemoglobin is a protein in your red blood cells that

carries oxygen to your body's organs and tissues and transports carbon dioxide from your organs and tissues back to your lungs. An Hb value less than 5.0 g/dL can lead to heart failure and death. The normal range for haemoglobin is:

For men, 13.5 to 17.5 grams per deciliter

For women, 12.0 to 15.5 grams per deciliter

Anything below that is considered a form of anemia.

16.Packed Cell Volume

The packed cell volume (PCV) is a measurement of the proportion of blood that is made up of cells. The value is expressed as a percentage or fraction of cells in blood. For example, a PCV of 40% means that there are 40 millilitres of cells in 100 millilitres of blood. Critical value are <18% and >55% (for adults)

17.White Blood Cell Count

wc in cells/cumm

White blood cell count varies from person to person , the normal range is usually between 4,000 and 11,000 white blood cells per microlitre of blood. Anything below 4,000 is typically considered to be a low white blood cell count.

18.Red Blood Cell Count

rc in millions/cmm

Red blood cells (RBC) in humans deliver oxygen to the body tissues. The average RBC count is 4.7 to 6.1 million cells per microliter for men and 4.2 to 5.4 million cells per microliter for women. A low RBC, also called anemia, is a common complication of CKD. Meanwhile, the data needs to be preprocessed to make it suitable for machine learning.

19.Hypertension

htn - (yes,no)

20.Diabetes Mellitus

dm - (yes,no)

21.Coronary Artery Disease

cad - (yes,no)

The coronary arteries supply blood, oxygen and nutrients to your heart.

Coronary artery disease develops when the coronary arteries become damaged or diseased. The damage may be caused by various factors (Smoking / High blood pressure /High cholesterol /Diabetes /sedentary lifestyle...)

22.Appetite

appet - (good, poor)

Appetite is the desire to eat food, sometimes due to hunger. A poor appetite is when your desire to eat is reduced. The medical term for a loss of appetite is anorexia. Any illness can reduce appetite. If the illness is treatable, the appetite should return when the condition is cured.

23.Pedal Edema

pe - (yes, no)

Pedal edema causes an abnormal accumulation of fluid in the ankles, feet, and lower legs causing swelling of the feet and ankles. Two mechanisms can cause edema of the feet. Two of the most common causes are being overweight and standing or sitting for long periods.

24.Anemia

ane - (yes, no)

25.Class

class - (ckd,notckd)

the table below shows a summary on all features and their types

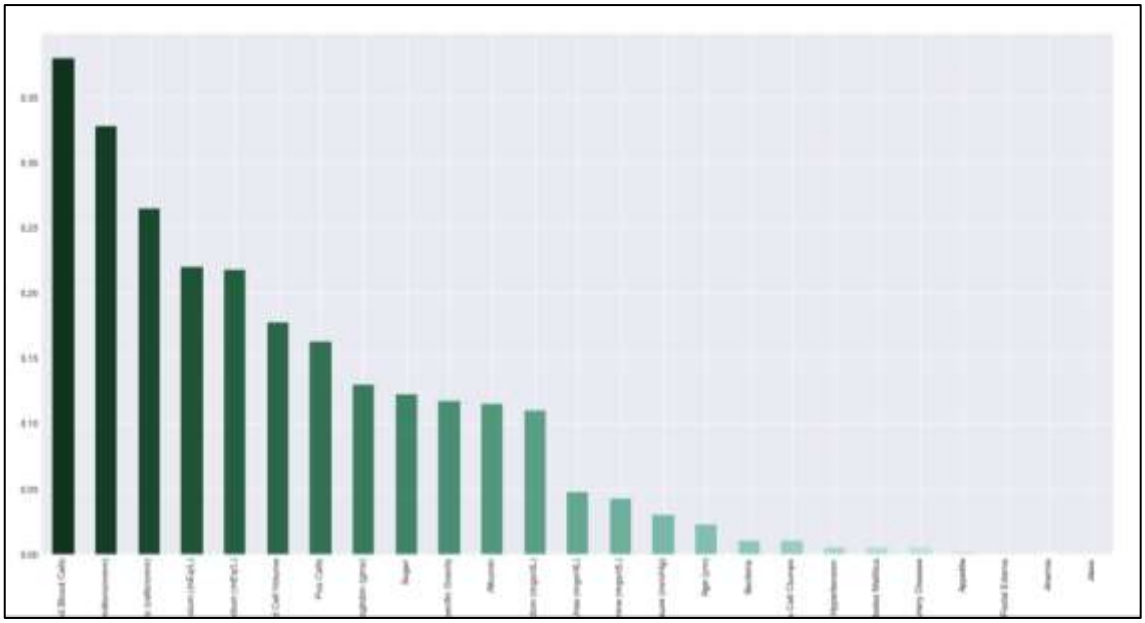
Features	Description	Type
Age	Age	Numerical
Bp	Blood pressure	Numerical
Sg	Specific gravity	Nominal
Al	Albumin	Nominal
Su	Sugar	Nominal
Rbc	Red blood cells	Nominal
Pc	Pus cells	Nominal
Pcc	Pus cells clumps	Nominal
Ba	Bacteria	Nominal
Bgr	Blood glucose random	Numerical
Bu	Blood urea	Numerical
Sc	Serum creatinine	Numerical
Sod	Sodium	Numerical

Pot	Potassium	Numerical
Hemo	Haemoglobin	Numerical
Pcv	Packed cell volume	Numerical
Wc	White blood cell count	Numerical
Rc	Red blood cell count	Numerical
Htn	Hypertension	Nominal
Dm	Diabetes mellitus	Nominal
Cad	Coronary artery disease	Nominal
Appet	Appetite	Nominal
Pe	Pedal edema	Nominal
Ane	Anaemia	Nominal
Class	Class	Nominal

Missing values:

The plot below shows the rate of missing values for each feature.

This rate is very high. These values will be imputed in the next chapter.

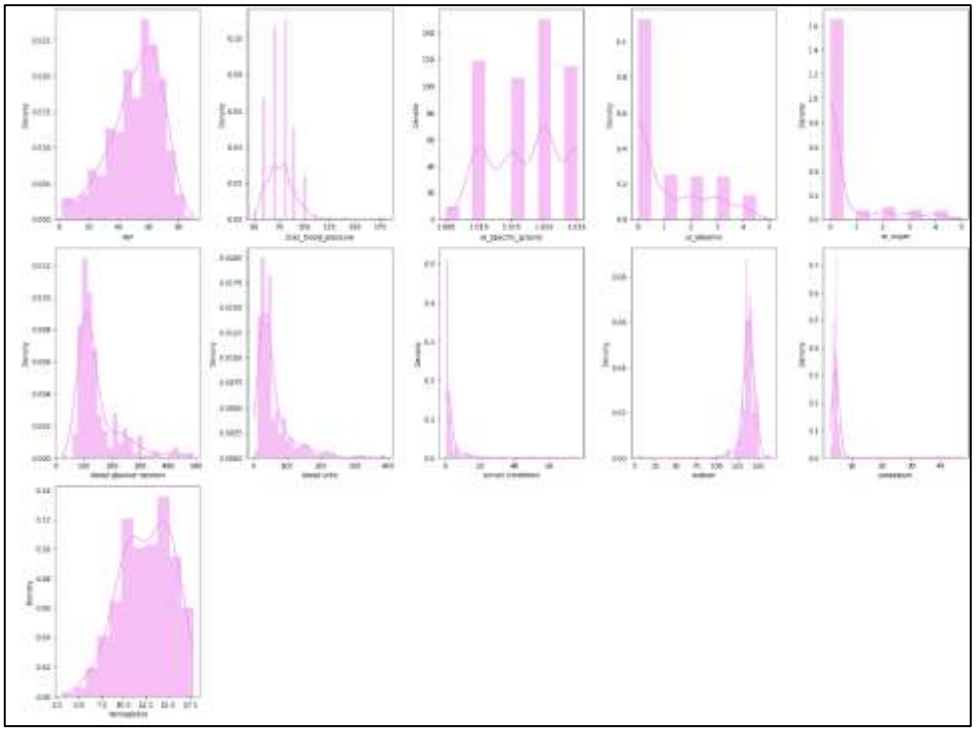


The plot below shows the distribution of the numerical features of our dataset.

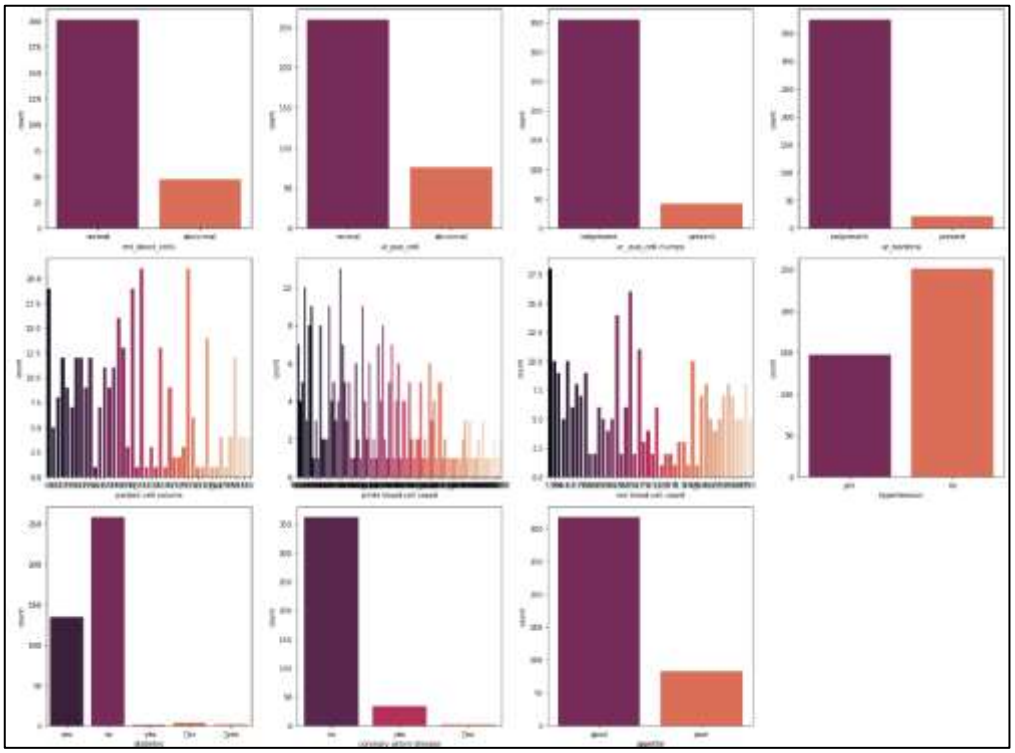
Some features have a high missing value rate which impacts their distributions.

Almost all of the features are skewed. Some others are very skewed.

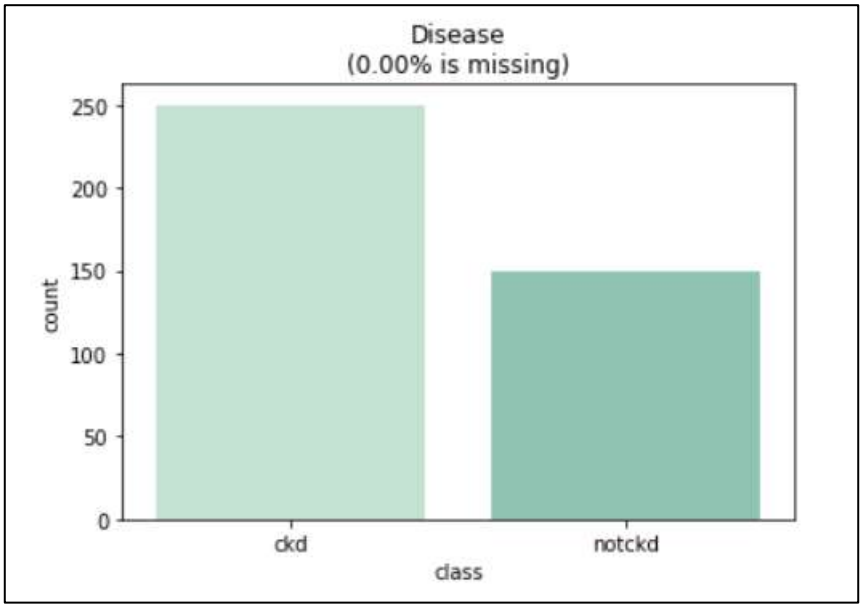
Numerical features:



Categorical features



For the class column, the plot shown below shows it is imbalanced with zero missing values.



2.4. Summary

To summarise, we described the business that our study revolves around, the main objective and the proposed methodology.

3. Data preparation for Article 1

3.1. Introduction

After finishing the data understanding, this chapter includes data preparation for the first article. In fact, data preparation is the process of cleaning and transforming raw data prior to processing and analysis. It is an important step prior to processing and often involves reformatting data, making corrections to data and the combining of data sets to enrich data.

3.2. Data cleaning

We have noticed that some values are mistyped in the dataset using the command unique.

3.3. Data transformation

For ease purposes, we changed the features' abbreviations to their full names.

3.4. Imputing missing values

Dropping observations in a field that is as sensitive as the medical field can be considered as forbidden. Everything counts.

So, in order to impute the missing values as mentioned in both of the articles. numerical data will be replaced with the mean and the categorical data will be replaced with mode

3.5. Data encoding

In order to prepare our data for modelling, we encoded the dataset using the OrdinalEncoder.

```
from sklearn.preprocessing import OrdinalEncoder
def encode(column):
    ord_enc = OrdinalEncoder()
    data[column] = ord_enc.fit_transform(data[[column]])
    data[column] = data[column].astype(int)
[encode(col) for col in data.columns if (data[col].dtype == 'object')]
```

3.6. Feature selection

Recursive Feature Elimination with Cross Validation:

Recursive Feature Elimination, Cross-Validated (RFECV) feature selection. RFECV Selects the best subset of features for the supplied estimator by removing 0 to N features (where N is the number of features) using recursive feature elimination, then selecting the best subset based on the cross-validation score of the model. Recursive feature elimination eliminates n features from a model by fitting the model multiple times and at each step, removing the weakest features, determined by either the **coef_** or **feature_importances_** attribute of the fitted model.

We can get the best parameters for RFECV using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
rfcc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=rfcc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X, y)
```

Now, we can see what the best parameters are:

```
cv_rfc.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 4,
 'max_features': 'auto',
 'n_estimators': 200}
```

Finally, we can use RFECV to get the most significant features according to our wrapper method.

Here we used a RandomForestClassifier as an estimator for our Feature Selection.

```
from sklearn.feature_selection import RFECV

rfecv = RFECV(estimator=RandomForestClassifier())
model = RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=4, max_features='auto', random_state=42)
pipeline = Pipeline(steps=[('s',rfecv),('m',model)])
# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
# summarize all features
rfecv.fit(X, y)
feat = []
for i in range(X.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfecv.support_[i], rfecv.ranking_[i]))
    if rfe.support_[i]:
        feat.append(data.columns[i])
```

Then, we will display all the features and their importance ranks as well as whether they were selected or not.

```
Accuracy: 0.984 (0.027)
Column: 0, Selected False, Rank: 9.000
Column: 1, Selected False, Rank: 8.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected True, Rank: 1.000
Column: 4, Selected False, Rank: 7.000
Column: 5, Selected False, Rank: 15.000
Column: 6, Selected False, Rank: 14.000
Column: 7, Selected False, Rank: 18.000
Column: 8, Selected False, Rank: 16.000
Column: 9, Selected False, Rank: 2.000
Column: 10, Selected False, Rank: 6.000
Column: 11, Selected True, Rank: 1.000
Column: 12, Selected False, Rank: 5.000
Column: 13, Selected False, Rank: 13.000
Column: 14, Selected True, Rank: 1.000
Column: 15, Selected True, Rank: 1.000
Column: 16, Selected False, Rank: 10.000
Column: 17, Selected True, Rank: 1.000
Column: 18, Selected False, Rank: 3.000
Column: 19, Selected False, Rank: 4.000
Column: 20, Selected False, Rank: 19.000
Column: 21, Selected False, Rank: 11.000
Column: 22, Selected False, Rank: 12.000
Column: 23, Selected False, Rank: 17.000
```

3.7. Conclusion

In this chapter, we presented the data preparation phase of the first article. In which we cleaned the data, transformed it and encoded it. Finally, we presented the RFECV feature selection method.

4. Data preparation for Article 2

4.1. Introduction

After finishing the data understanding step, this chapter includes data preparation for the second article. This step is very crucial. A New York Times article reported that data scientists spend between 50% and 80% of their time in data preparation.

4.2. Data cleaning

We have noticed that some values are mistyped in the dataset using the command unique.

As for the missing values imputation, we did the same as the first article, numerical features were imputed by the mean and the categoricals were replaced with mode.

4.3. Data transformation

For ease purposes, we changed the features' abbreviations to their full names.

4.4. Correlation-Based Feature Selection

For this section, we based our selection on the already executed correlation-work.

4.5. Data transformation

Dropping observations in a field that is as sensitive as the medical field can be considered as forbidden. Everything counts.

So, in order to impute the missing values as mentioned in both of the articles. numerical data will be replaced with the mean and the categorical data will be replaced with mode.

4.6. Data encoding

In order to prepare our data for modelling, we encoded the dataset using the OrdinalEncoder.

```
from sklearn.preprocessing import OrdinalEncoder
def encode(column):
    ord_enc = OrdinalEncoder()
    data[column] = ord_enc.fit_transform(data[[column]])
    data[column] = data[column].astype(int)
[encode(col) for col in data.columns if (data[col].dtype == 'object')]
```

4.7. Feature selection

Correlation-based Feature Selection (CFS) is suitable to be applied to multivariate data. CFS works by calculating the interaction between features. CFS evaluates a subset of features taking into account predictive capabilities of each level of redundancy among features and those features.

We analysed the features using the Pearson correlation matrix.

The figure below shows the features according to their correlation indexes. According to it, we will remove Red Blood Cell Count, Ur_pus_Cell, Ur_specific_Gravity, Whot Blood Cell Count, Red_blood_cells and Potassium.

```python
# Pearson correlation coefficient
corr = data.corr()["class"].sort_values(ascending=False)[1:]

# absolute for positive values
abs_corr = abs(corr)

# random threshold for features to keep
relevant_features = abs_corr[abs_corr>0.29]
type(relevant_features)
list2 = relevant_features.index.tolist()
relevant_features
```

```
hemoglobin              0.729628
ur_specific_gravity     0.698957
packed cell volume      0.690060
red blood cell count    0.590913
ur_pus_cell             0.375154
sodium                  0.342288
Dias_blood_pressure     0.290600
serum creatinine        0.294079
anemia                  0.325396
ur_sugar                0.327812
blood urea              0.372033
pedal edema             0.375154
appetite                0.393341
blood glucose random    0.401374
diabetes                0.559060
hypertension            0.590438
ur_albumin              0.599238
Name: class, dtype: float64
```

4.7. Conclusion

In this chapter, we presented the data preparation phase of the second article. In which we cleaned the data, transformed it and encoded it. Finally, we presented the RFECV feature selection method.

5. Modelling for Article 1

5.1. Introduction

During this chapter, we will focus on the fourth step, Data modelling. After all the cleaning, formatting and feature selection, we will now feed the data to the model. For

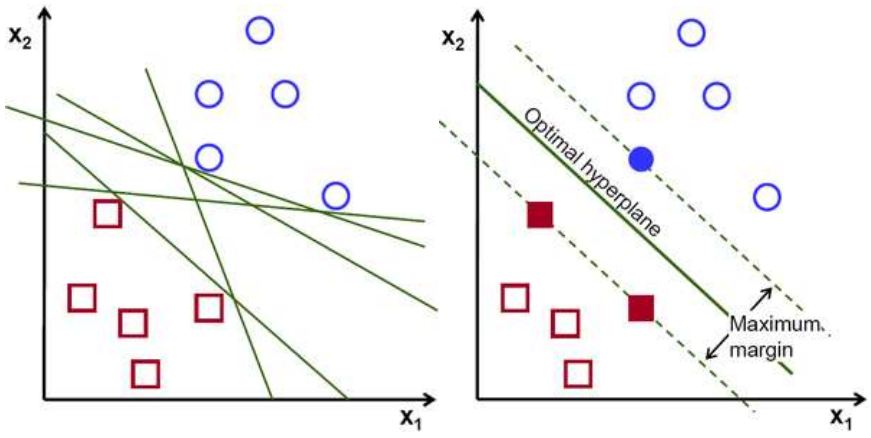
the first article, the models that will be used in this part are K-Nearest Neighbour (KNN), Decision Tree, Random Forest and Support Vector Machine (SVM).

5.2. Modelling with all features

5.2.1. Support Vector Machine

A support vector machine (SVM) is a type of machine learning algorithm that performs supervised learning for classification or regression of data groups. In AI and machine learning, supervised learning systems provide both input and desired output data, which are labelled for classification.

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.



To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximising the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Fitting a model:

We can perform this using Scikit-Learn as so:

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc, precision_score, recall_score

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(rfecv_df, y, test_size=0.2)

# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)

# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

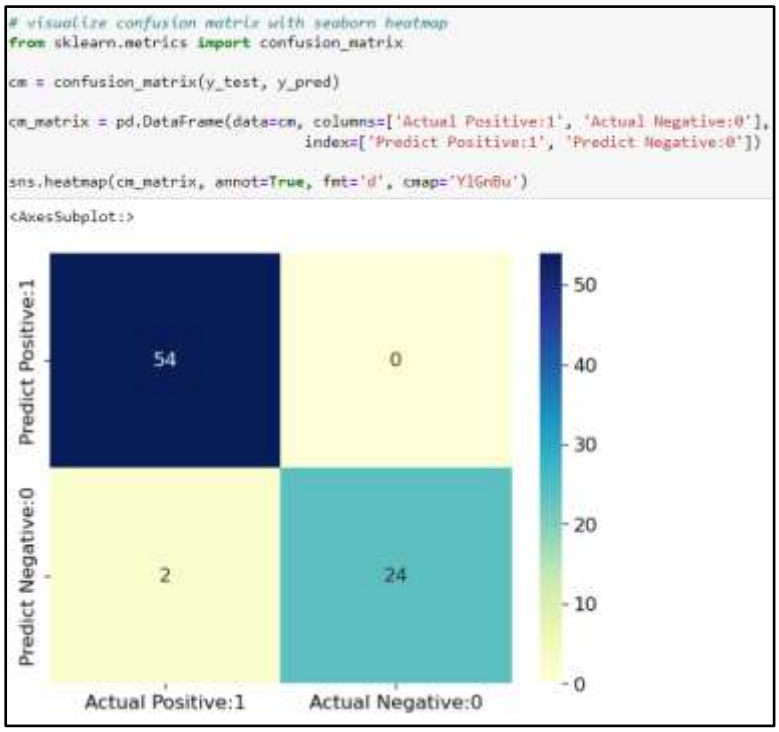
Output:

Model accuracy score with linear kernel and C=1000.0 : 0.975

We can use a heatmap to visualise the confusion matrix:

```
# visualize confusion matrix with seaborn heatmap
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<AxesSubplot:>
```



The confusion matrix shows 54 + 25 = 79 correct predictions and 0 + 1 = 1 incorrect predictions.

True Positives (Actual Positive:1 and Predict Positive:1) - 54

True Negatives (Actual Negative:0 and Predict Negative:0) - 25

False Positives (Actual Negative:0 but Predict Positive:1) - 0 (Type I error)

False Negatives (Actual Positive:1 but Predict Negative:0) - 1 (Type II error)

Classification report:

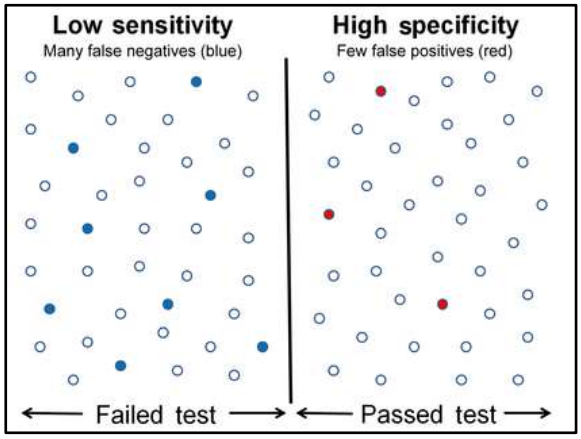
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	54
1	1.00	0.92	0.96	26
accuracy			0.97	80
macro avg	0.98	0.96	0.97	80
weighted avg	0.98	0.97	0.97	80

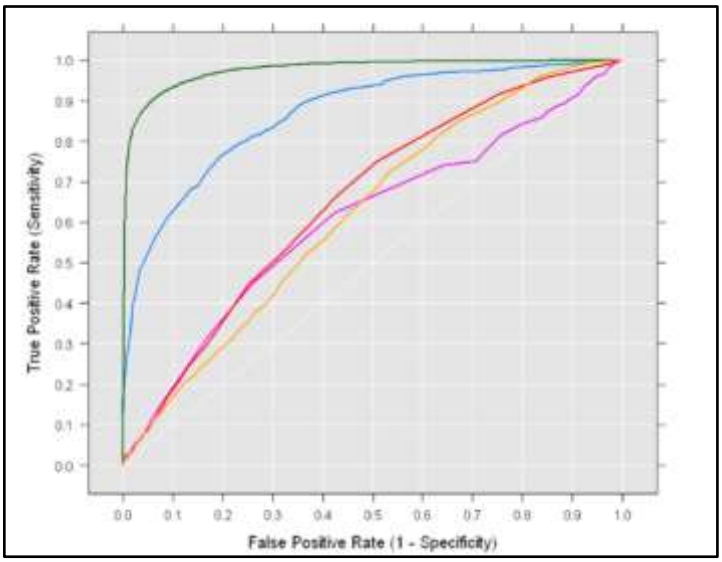
- Precision: High precision relates to low false positives rate, a 0.96 rate is excellent.
- Recall:. A recall should ideally be 1 but anything more than 0.9 is very good.
- F-Measure: this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. An F1 score above 0.9 is excellent.

ROC Curve:

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR).



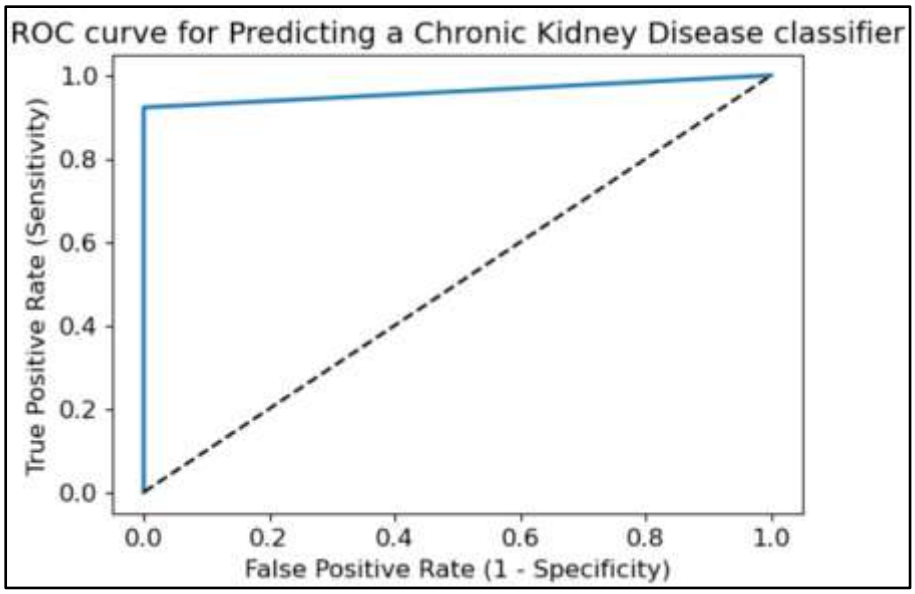
Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR).



We can plot a roc curve using the Scikit-Learn class roc_curve

```
# plot ROC Curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0,1], [0,1], 'k--' )
plt.rcParams['font.size'] = 12
plt.title('ROC curve for Predicting a Chronic Kidney Disease classifier')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.show()
```

As we can see in the figure below, our model is performing very well.



ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. We can compute this measurement as the figure below shows:

```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9615
```

Cross validation score:

Cross-Validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to learn or train a model and the other used to validate the model.

> *A 10-fold cross-validation, in particular, the most commonly used error-estimation method in machine learning, can easily break down in the case of class imbalances, even if the skew is less extreme than the one previously considered.*

Implementing the concept of **stratified sampling** in cross-validation ensures the training and test sets have the same proportion of the feature of interest as in the original dataset. Doing this with the target variable ensures that the cross-validation result is a close approximation of generalisation error.

We can perform this as the figure below shows, we get a 0.9758 which is an excellent result:

```
from sklearn.model_selection import KFold

#train model with cv of 12
# svm_folds=KFold(n_splits=12,shuffle=True,random_state=100)
svm_cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
svm_cv_scores = cross_val_score(linear_svc1000, rfecv_df, y, cv=svm_cv)
#print each cv score (accuracy) and average them
print(svm_cv_scores)
print('cv_scores mean:{}'.format(np.mean(svm_cv_scores)))
```

```
[1.    1.    0.95  0.975 0.975 1.    0.925 0.925 1.    1.    0.975 1.
 0.975 1.    0.975 0.925 0.975 0.975 1.    0.975 0.95  0.925 1.    1.
 1.    1.    0.975 0.925 0.975 1.   ]
cv_scores mean:0.9758333333333334
```

5.2.2. K-Nearest Neighbour

The k-nearest neighbour's algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

kNN has many advantages, such that it is a good accuracy generator, also has a relatively fast training time compared to other classifiers, and can be appropriately handled in an overfitting case.

For the following execution, the euclidean method will be used as a metric for the k-nearest neighbour's classifier.

And for that, the training and testing results achieved 96% and 97% respectively.

```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3, metric = 'euclidean')
classifier.fit(X_ktrain, y_ktrain)

# Predicting the Test set results
y_kpred = classifier.predict(X_ktest)
```

```
y_kpred_train = kclassifier.predict(X_ktrain)

print('Training set score: {:.4f}'.format(kclassifier.score(X_ktrain, y_ktrain)))

print('Test set score: {:.4f}'.format(kclassifier.score(X_ktest, y_ktest)))
```

```
Training set score: 0.9600
Test set score: 0.9700
```

As for the overall performance:

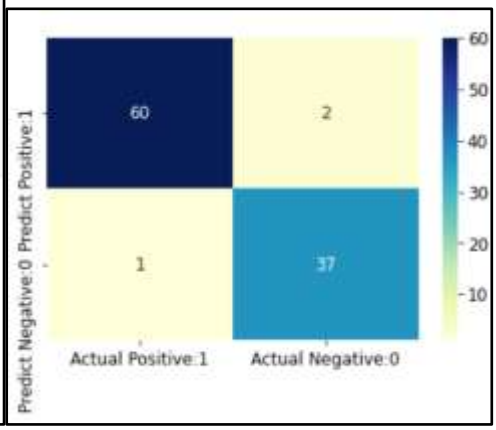
```
print("K-Nearest Neighbours Model Accuracy:", accuracy_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Model Precision:", precision_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Model Recall:", recall_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Model F-measure:", f1_score(y_ktest, y_kpred))
```

```
K-Nearest Neighbours Model Accuracy: 0.97
K-Nearest Neighbours Model Precision: 0.9487179487179487
K-Nearest Neighbours Model Recall: 0.9736842105263158
K-Nearest Neighbours Model F-measure: 0.9610389610389611
```

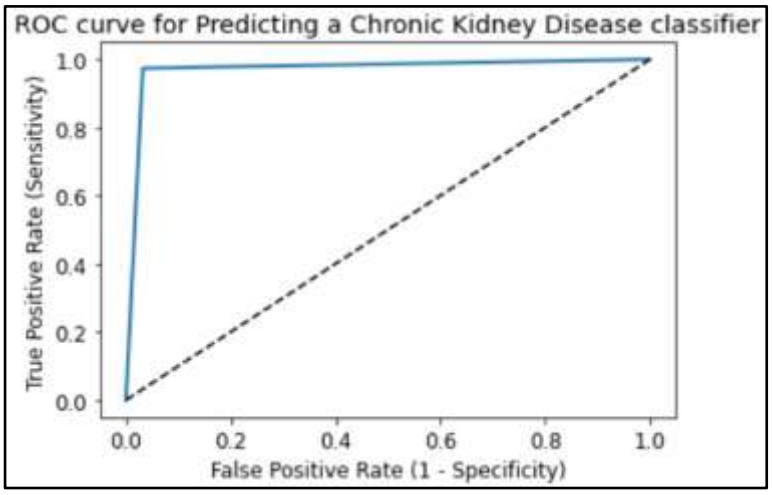
Hence, we proceed to the confusion matrix to define the performance of the classification algorithm. Yet, for this case, we need to visualise and summarise the performance of the k-nearest neighbour's classifier and its impact on the previous steps.

```
from sklearn.metrics import confusion_matrix

kcm = confusion_matrix(y_ktest, y_kpred)

# Print the Confusion Matrix and slice it into four pieces
print('Confusion matrix\n\n', kcm)

print('\nTrue Positives(TP) = ', kcm[0,0])

print('\nTrue Negatives(TN) = ', kcm[1,1])

print('\nFalse Positives(FP) = ', kcm[0,1])

print('\nFalse Negatives(FN) = ', kcm[1,0])
```

```
Confusion matrix

 [[60  2]
 [ 1 37]]

True Positives(TP) =  60

True Negatives(TN) =  37

False Positives(FP) =  2

False Negatives(FN) =  1
```



As a conclusion, we plot the results for predicting the Chronic Kidney Disease using KNN classifier.



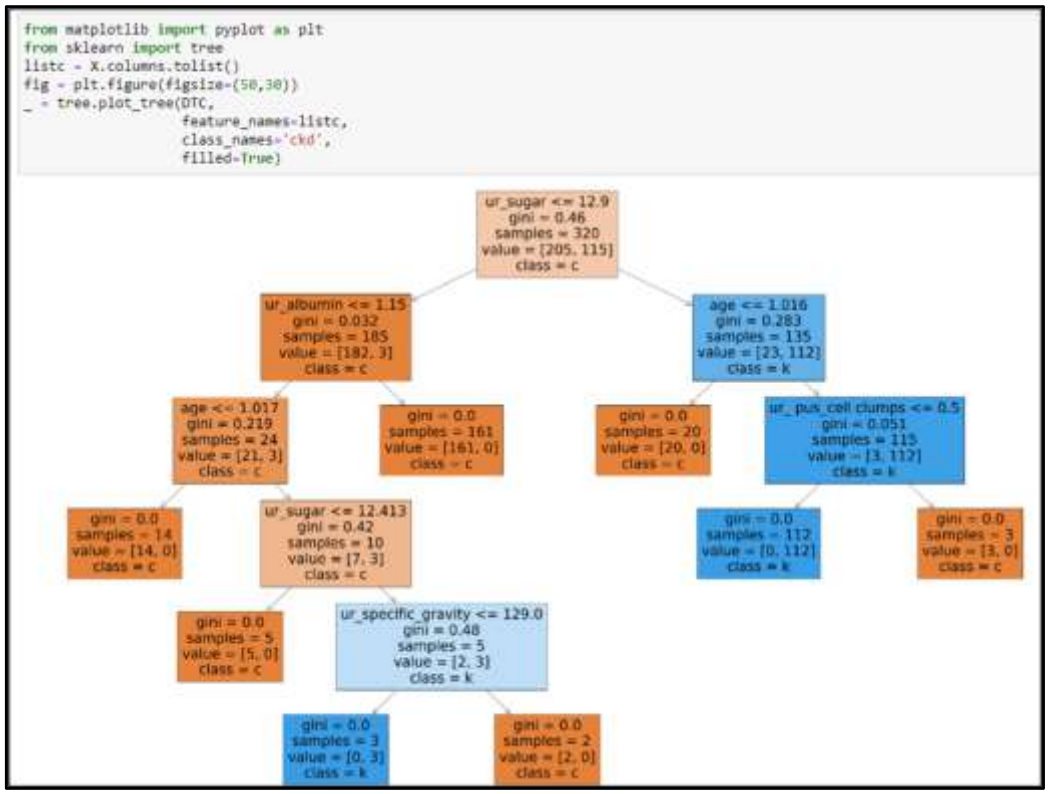
Moreover, to ensure more realistic results and a better predictive model, we set sail towards the k-fold cross validation as it is a resampling procedure used to evaluate machine learning models.

5.2.3. Decision Tree

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

```
from matplotlib import pyplot as plt
from sklearn import tree
listc = X.columns.tolist()
fig = plt.figure(figsize=(50,30))
_ = tree.plot_tree(DTC,
                   feature_names=listc,
                   class_names='ckd',
                   filled=True)
```




```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred = dtree.predict(X_test)
print(" accuracy: {0:.5f}".format(accuracy_score(y_test, y_pred)))
print("precision: {0:.5f}".format(precision_score(y_test, y_pred)))
print("   recall: {0:.5f}".format(recall_score(y_test, y_pred)))
print(" f1 score: {0:.5f}".format(f1_score(y_test, y_pred)))
```

```
 accuracy: 0.94949
precision: 0.96364
   recall: 0.94643
 f1 score: 0.95495
```

5.2.4. Random Forest

This algorithm creates a vast number of decision trees that work together. In this method, decision trees serve as pillars. The term “random forest” refers to a collection of decision trees whose nodes are established at the preprocessing stage. The best feature is chosen from a random selection of features after numerous trees have been constructed. Another idea produced by the decision tree algorithm is generating a decision tree. As a result, a random forest comprises these trees, which are used to categorise new objects from a vector of inputs. Each decision tree constructed is used to classify data. If votes are assigned to that class, the random forest will select the classification with the most votes from all the trees in the forest.

After splitting our data that was selected with RFECV into train and test sets, we applied the algorithm in question, as shown in the figure below with its metrics.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
rfc=RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=4, max_features='auto', random_state=42)
rfc.fit(x_rftrain, y_rftrain)
y_rfpred=rfc.predict(x_rftest)
RandomForestClf_accuracy_score = metrics.accuracy_score(y_rftest, y_rfpred)
print("Random Forest Classifier Model Accuracy:", accuracy_score(y_rftest, y_rfpred))

print("Random Forest Classifier Model Precision:", precision_score(y_rftest, y_rfpred))

print("Random Forest Classifier Model Recall:", recall_score(y_rftest, y_rfpred))

print("Random Forest Classifier Model F-measure:", f1_score(y_rftest, y_rfpred))

scores = cross_val_score(rfc, x_rftrain, y_rftrain,cv=10)
print("Mean cross-validation score: %.2f" % scores.mean())
```

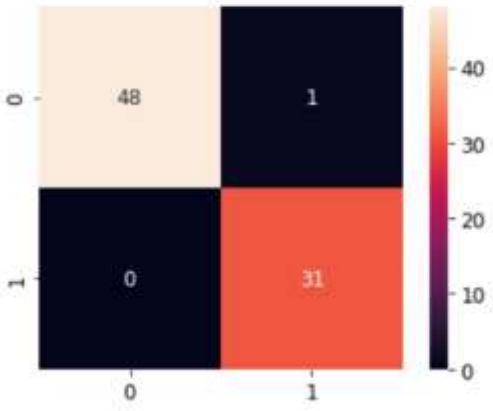
```
Random Forest Classifier Model Accuracy: 0.9875
Random Forest Classifier Model Precision: 0.96875
Random Forest Classifier Model Recall: 1.0
Random Forest Classifier Model F-measure: 0.9841269841269841
Mean cross-validation score: 0.99
```

The figure shown below presents the heatmap of our algorithm.

It shows 48 for True Positive (TP), 31 for True Negative (TN), only one False Positive (FP) and none for False Negative (FN).

```
sns.heatmap(cm, square=True , annot=True)
```

```
<AxesSubplot:>
```



After that, we can see the classification report of the random forest algorithm.

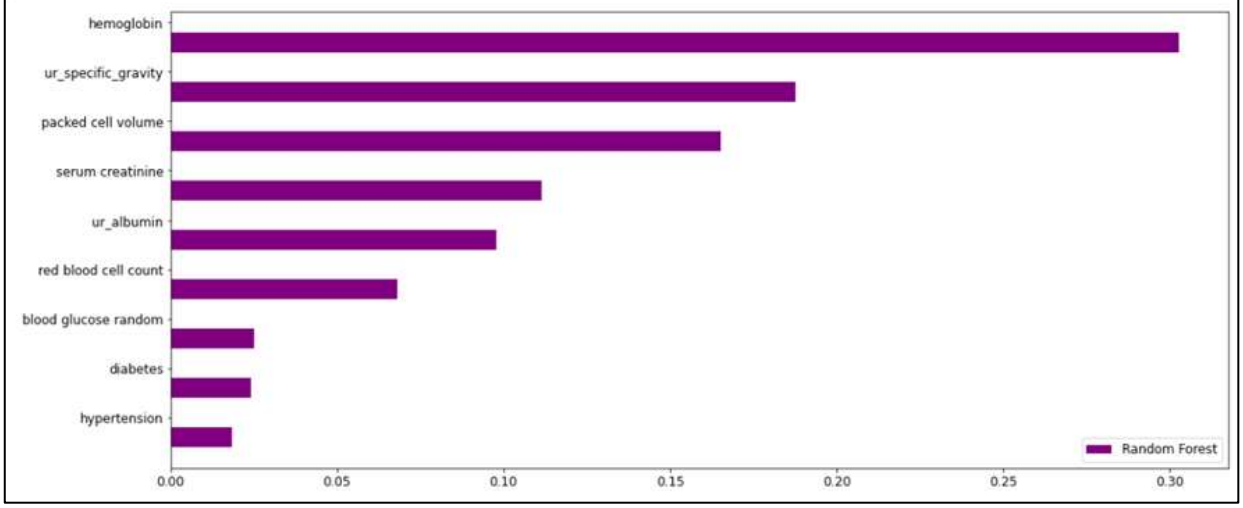
It shows very good results, it is basically the best model in this article.

```
print(classification_report(y_rftest, y_rfpred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        49
           1       0.97      1.00      0.98        31

    accuracy                           0.99        80
   macro avg       0.98      0.99      0.99        80
weighted avg       0.99      0.99      0.99        80
```

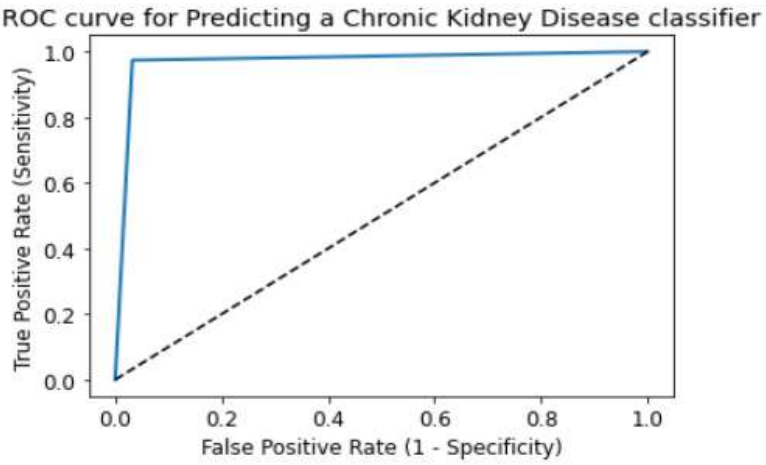
The RFE method is used to select the most significant features by finding high correlation between specific features and target (labels). The figure below shows the most significant features according to RFE. The feature haemoglobin has the highest importance followed by ur_specific_gravity and packed_cell_volume for the top three important features.

```
feature_importance=pd.DataFrame({'rfc':rfc.feature_importances_},index=rfecv_df.columns)
feature_importance.sort_values(by='rfc',ascending=True,inplace=True)

index = np.arange(len(feature_importance))
fig, ax = plt.subplots(figsize=(18,8))
rfc_feature=ax.barh(index,feature_importance['rfc'],0.4,color='purple',label='Random Forest')
ax.set(yticks=index+0.4,yticklabels=feature_importance.index)

ax.legend()
plt.show()
```



The figure below shows the ROC Curve for the Random Forest algorithm.

```
ffpr, ttpr, tthresholds = roc_curve(y_rftest, y_rfpred)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Predicting a Chronic Kidney Disease classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```



5.3. Conclusion

In this chapter, we presented the fourth step of the CRISP-DM method, modelling for the first article. We presented 4 models, KNN, SVM, Decision Tree as well as Randm Forest.

6. Modelling for Article 2

6.1. Introduction

During this chapter, we will focus on the fourth step, Data modelling. After all the cleaning, formatting and feature selection, we will now feed the data to the model. For the second article, the models that will be used in this part are K-Nearest Neighbour (KNN), Decision Tree and Random Forest.

6.2. Base Modeling Without Feature selection

In this subsection, we will present our base modelling without the feature selection method.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc, precision_score, recall_score

classifiers = ['LinearSVM', 'Naive_Bayes', 'KNeighbors']

models = [SVC(kernel='linear', C=1000.0),
          GaussianNB(),
          KNeighborsClassifier(n_neighbors=3, metric = 'euclidean')]

def split(df,label):
    X_tr, X_te, Y_tr, Y_te = train_test_split(df, label, test_size=0.2, random_state=42)
    return X_tr, X_te, Y_tr, Y_te

def acc_score(df,label):
    Score = pd.DataFrame({"Classifier":classifiers})
    j = 0
    acc = []
    prec = []
    rec = []
    f_score = []
    X_train,X_test,Y_train,Y_test = split(df,label)
    for i in models:
        model = i
        model.fit(X_train,Y_train)
        predictions = model.predict(X_test)
        acc.append(accuracy_score(Y_test,predictions))
        prec.append(precision_score(Y_test, predictions))
        rec.append(recall_score(Y_test, predictions))
        f_score.append(f1_score(Y_test, predictions))
        j = j+1
    Score["Accuracy"] = acc
    Score["Precision"] = prec
    Score["Recall"] = rec
    Score["F-Measure"] = f_score
    Score.sort_values(by="Accuracy", ascending=False,inplace = True)
    Score.reset_index(drop=True, inplace=True)
    return Score
```

Classification report:

```
score1 = acc_score(X,y)
score1
```

	Classifier	Accuracy	Precision	Recall	F-Measure
0	Naive_Bayes	0.9875	0.965517	1.000000	0.982456
1	LinearSVM	0.9750	0.964286	0.964286	0.964286
2	KNeighbors	0.7500	0.595238	0.892857	0.714286

6.3. Modeling with CFS

6.3.1. Support Vector Machine

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(cor_df, y, test_size=0.2)

# instantiate classifier with linear kernel and C=1000.0
linear_svc1000=SVC(kernel='linear', C=1000.0)


# fit classifier to training set
linear_svc1000.fit(X_train, y_train)


# make predictions on test set
y_pred=linear_svc1000.predict(X_test)


# compute and print accuracy score
print('Model accuracy score with linear kernel and C=1000.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score with linear kernel and C=1000.0 : 0.9625
```

Classification report:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        42
           1       0.95      0.97      0.96        38

    accuracy                           0.96        80
   macro avg       0.96      0.96      0.96        80
weighted avg       0.96      0.96      0.96        80
```

Check for overfitting and underfitting:

```
y_pred_train = linear_svc1000.predict(X_train)

print('Training set score: {:.4f}'.format(linear_svc1000.score(X_train, y_train)))

print('Test set score: {:.4f}'.format(linear_svc1000.score(X_test, y_test)))
```

```
Training set score: 0.9719
Test set score: 0.9625
```

The training-set accuracy score is 0.9719 while the test-set accuracy is 0.9625. These two values are quite comparable. So,there is no sign of overfitting.

Compare model accuracy with null accuracy:

So, the model accuracy is 0.9625. But, we cannot say that our model is very good based on the above accuracy. We must compare it with the null accuracy. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

Hence, we should first check the class distribution in the test set.

```
# check class distribution in test set
y_test.value_counts()
```

```
0    42
1    38
Name: class, dtype: int64
```

We can see that the occurrences of the most frequent class is 42. So, we can calculate null accuracy by dividing 42 by total number of occurrences.

```
# check null accuracy score

null_accuracy = (42/(42+38))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

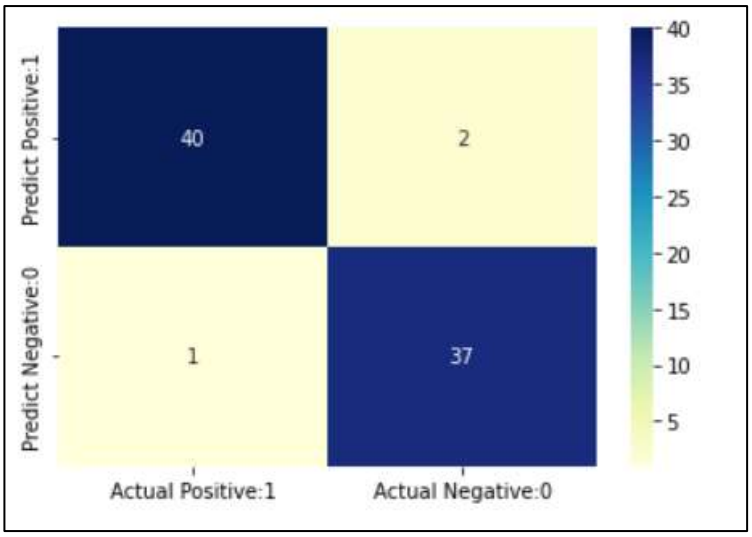
```
Null accuracy score: 0.5250
```

We can see that our model accuracy score is 0.9625 but null accuracy score is 0.5250. So, we can conclude that our SVM Classification model is doing a very good job in predicting the class labels.

Confusion matrix:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[40  2]
 [ 1 37]]

True Positives(TP) =  40

True Negatives(TN) =  37

False Positives(FP) =  2

False Negatives(FN) =  1
```



The confusion matrix shows 40 + 37 = 77 correct predictions and 2 + 1 = 3 incorrect predictions.

In this case, we have:

True Positives (Actual Positive:1 and Predict Positive:1) - 40

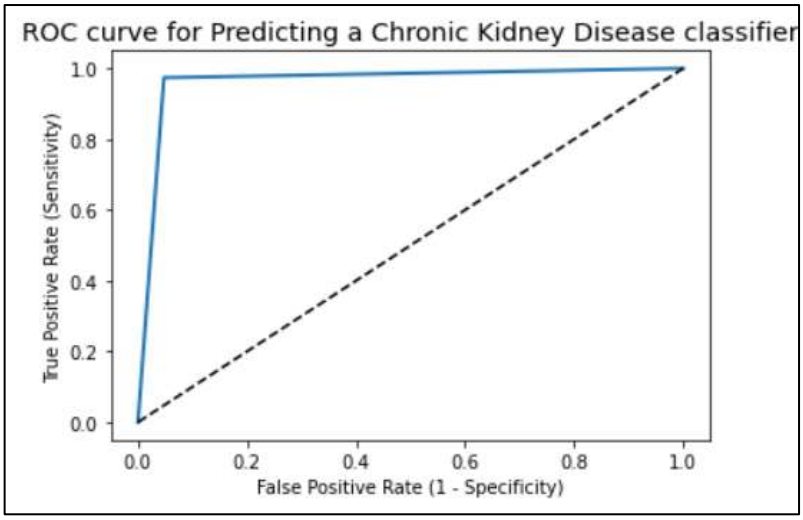
True Negatives (Actual Negative:0 and Predict Negative:0) - 37

False Positives (Actual Negative:0 but Predict Positive:1) - 2 (Type I error)

False Negatives (Actual Positive:1 but Predict Negative:0) - 1 (Type II error)

ROC curve:

```
# plot ROC Curve

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred)

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Predicting a Chronic Kidney Disease classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```




```
# compute ROC AUC

from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9630
```

6.3.2. K-Nearest Neighbour

The k-nearest neighbour's algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

kNN has many advantages, such that it is a good accuracy generator, also has a relatively fast training time compared to other classifiers, and can be appropriately handled in an overfitting case.

For the following execution, the euclidean method will be used as a metric for the k-nearest neighbour's classifier.

And for that, the training and testing results achieved 89% and 85% respectively.

```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 3, metric = 'euclidean')
classifier.fit(X_ktrain, y_ktrain)

# Predicting the Test set results
y_kpred = classifier.predict(X_ktest)
```

```
y_kpred_train = classifier.predict(X_ktrain)

print('Training set score: {:.4f}'.format(classifier.score(X_ktrain, y_ktrain)))

print('Test set score: {:.4f}'.format(classifier.score(X_ktest, y_ktest)))
```

```
Training set score: 0.8938
Test set score: 0.8500
```

As for the overall performance:

Precision: Quantifies the number of positive class predictions that actually belong to the positive class.

Recall: Quantifies the number of positive class predictions made out of all positive examples in the dataset.

F-Measure: Provides a single score that balances both the concerns of precision and recall in one number.

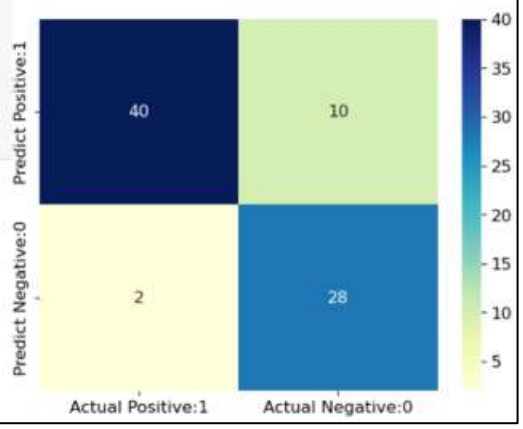
```
print("K-Nearest Neighbours Classifier Model Accuracy:", accuracy_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Classifier Model Precision:", precision_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Classifier Model Recall:", recall_score(y_ktest, y_kpred))

print("K-Nearest Neighbours Classifier Model F-measure:", f1_score(y_ktest, y_kpred))
```

```
K-Nearest Neighbours Classifier Model Accuracy: 0.85
K-Nearest Neighbours Classifier Model Precision: 0.7368421052631579
K-Nearest Neighbours Classifier Model Recall: 0.9333333333333333
K-Nearest Neighbours Classifier Model F-measure: 0.8235294117647058
```

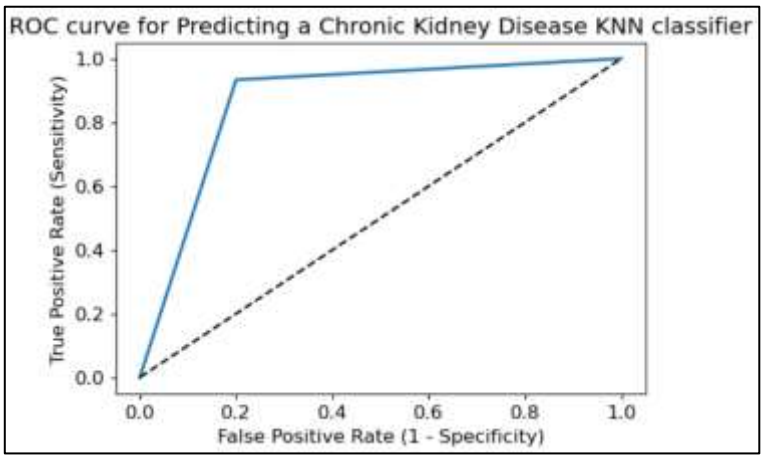
Hence, we proceed to the confusion matrix to define the performance of the classification algorithm. Yet, for this case, we need to visualise and summarise the performance of the k-nearest neighbour's classifier and its impact on the previous steps.

```
from sklearn.metrics import confusion_matrix

kcm = confusion_matrix(y_ktest, y_kpred)

# Print the Confusion Matrix and slice it into four pieces
print('Confusion matrix\n\n', kcm)

print('\nTrue Positives(TP) = ', kcm[0,0])

print('\nTrue Negatives(TN) = ', kcm[1,1])

print('\nFalse Positives(FP) = ', kcm[0,1])

print('\nFalse Negatives(FN) = ', kcm[1,0])
```

```
Confusion matrix

 [[40 10]
 [ 2 28]]

True Positives(TP) =  40

True Negatives(TN) =  28

False Positives(FP) =  10

False Negatives(FN) =  2
```



As a conclusion, we plot the results for predicting the Chronic Kidney Disease using KNN classifier.



Moreover, to ensure more realistic results and a better predictive model, we set sail towards the k-fold cross validation as it is a resampling procedure used to evaluate machine learning models.

```
from sklearn.model_selection import KFold

#create a new KNN model, metric = 'euclidean'
knn_cv = KNeighborsClassifier(n_neighbors=3, metric = 'euclidean')
#train model with cv of 10
folds=KFold(n_splits=10,shuffle=True,random_state=100)
cv_scores = cross_val_score(knn_cv, cor_df, y, cv=folds)
#print each cv score (accuracy) and average them
print(cv_scores)
print('cv_scores mean:{}'.format(np.mean(cv_scores)))
```

```
[0.825 0.775 0.775 0.775 0.725 0.725 0.675 0.775 0.825 0.775]
cv_scores mean:0.7650000000000001
```

6.3.3. Naïve Bayes

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is, the presence of one particular feature does not affect the other. Hence it is called naive.

```
In [50]: from sklearn.naive_bayes import GaussianNB

         # Splitting data
         X_nbtrain, X_nbtest, y_nbtrain, y_nbtest = train_test_split(cor_df, y, test_size=0.2)

         # Instantiating a Naive Bayes Classifier
         classifier = GaussianNB()

         # Fitting Naive Bayes to the Training set
         classifier.fit(X_nbtrain, y_nbtrain)

         # make predictions on test set
         y_nbpred = classifier.predict(X_nbtest)
         y_nbpred

Out[50]: array([0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1,
                0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
                0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0])
```

Classification report:

```
print("Naive Bayes Classifier Model Accuracy:", accuracy_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model Precision:", precision_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model Recall:", recall_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model F-measure:", f1_score(y_nbtest, y_nbpred))
```

```
Naive Bayes Classifier Model Accuracy: 0.95
Naive Bayes Classifier Model Precision: 0.8888888888888888
Naive Bayes Classifier Model Recall: 1.0
Naive Bayes Classifier Model F-measure: 0.9411764705882353
```

Check for overfitting and underfitting:

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(classifier.score(X_nbtrain, y_nbtrain)))

print('Test set score: {:.4f}'.format(classifier.score(X_nbtest, y_nbtest)))
```

```
Training set score: 0.9594
Test set score: 0.9500
```

The training-set accuracy score is 0.9594 while the test-set accuracy is 0.9500. These two values are quite comparable. So,there is no sign of overfitting.

Compare model accuracy with null accuracy:

So, the model accuracy is 0.9500. But, we cannot say that our model is very good based on the above accuracy. We must compare it with the null accuracy. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

Hence, we should first check the class distribution in the test set.

```
# check class distribution in test set

y_nbtest.value_counts()
```

```
0    48
1    32
Name: class, dtype: int64
```

We can see that the occurrences of the most frequent class is 48. So, we can calculate null accuracy by dividing 48 by total number of occurrences.

```
# check null accuracy score

null_accuracy = (48/(48+32))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.6000
```

We can see that our model accuracy score is 0.9500 but null accuracy score is 0.6000. So, we can conclude that our Gaussian Naive Bayes Classification model is doing a very good job in predicting the class labels.

Now, based on the above analysis we can conclude that our classification model accuracy is very good.

But, it does not give the underlying distribution of values. Also, it does not tell anything about the type of errors our classifier is making.

We have another tool called **Confusion matrix** that comes to our rescue.

Confusion matrix:

A confusion matrix will give us a clear picture of classification model performance and the types of errors produced by the model. Four types of outcomes are possible while evaluating a classification model performance. These four outcomes are described below:-

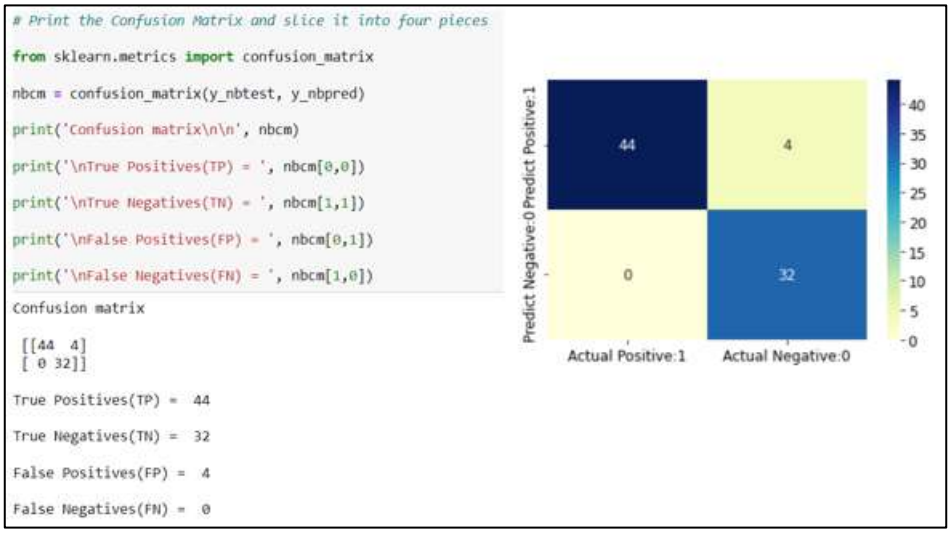
True Positives (TP) – True Positives occur when we predict an observation belongs to a certain class and the observation actually belongs to that class.

True Negatives (TN) – True Negatives occur when we predict an observation does not belong to a certain class and the observation actually does not belong to that class.

False Positives (FP) – False Positives occur when we predict an observation belongs to a certain class but the observation actually does not belong to that class. This type of error is called Type I error.

False Negatives (FN) – False Negatives occur when we predict an observation does not belong to a certain class but the observation actually belongs to that class. This is a very serious error and it is called Type II error.

These four outcomes are summarised in a confusion matrix given below.



```
# Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix
nbcm = confusion_matrix(y_nbtest, y_nbpred)
print('Confusion matrix\n\n', nbcm)
print('\nTrue Positives(TP) = ', nbcm[0,0])
print('\nTrue Negatives(TN) = ', nbcm[1,1])
print('\nFalse Positives(FP) = ', nbcm[0,1])
print('\nFalse Negatives(FN) = ', nbcm[1,0])
```

```
Confusion matrix

 [[44  4]
 [ 0 32]]

True Positives(TP) =  44

True Negatives(TN) =  32

False Positives(FP) =  4

False Negatives(FN) =  0
```


The confusion matrix shows 44 + 32 = 76 correct predictions and 4 + 0 = 4 incorrect predictions.

In this case, we have:

True Positives (Actual Positive:1 and Predict Positive:1) - 44

True Negatives (Actual Negative:0 and Predict Negative:0) - 32

False Positives (Actual Negative:0 but Predict Positive:1) - 4 (Type I error)

False Negatives (Actual Positive:1 but Predict Negative:0) - 0 (Type II error)

ROC curve:

ROC Curve Another tool to measure the classification model performance visually is ROC Curve.

The ROC Curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold levels.

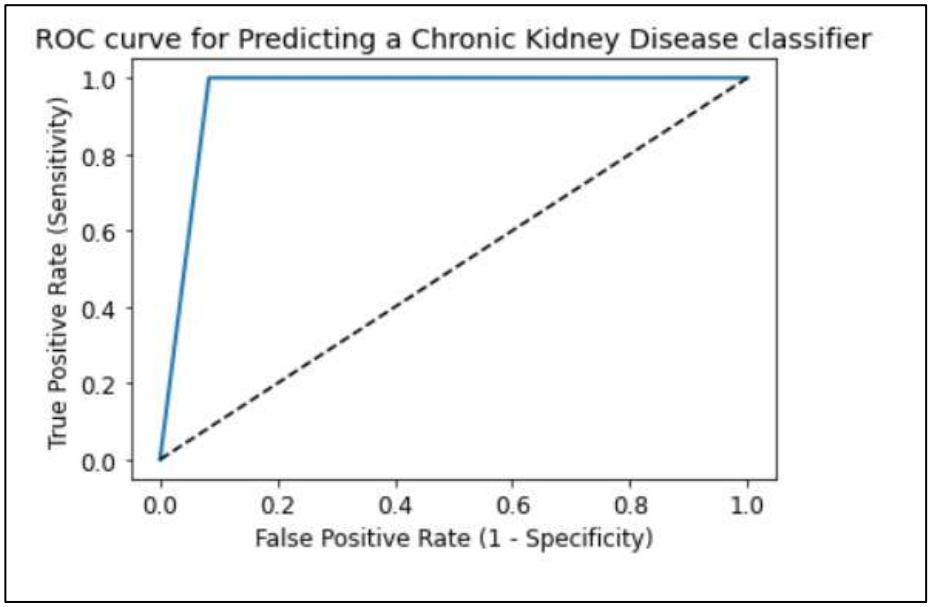
True Positive Rate (TPR) is also called Recall. It is defined as the ratio of TP to (TP + FN).

False Positive Rate (FPR) is defined as the ratio of FP to (FP + TN).

In the ROC Curve, we will focus on the TPR (True Positive Rate) and FPR (False Positive Rate) of a single point. This will give us the general performance of the ROC curve which consists of the TPR and FPR at various threshold levels. So, an ROC Curve plots TPR vs FPR at different classification threshold levels. If we lower the threshold levels, it may result in more items being classified as positive. It will increase both True

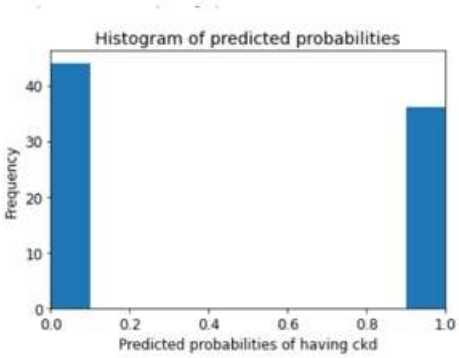
```
fpr, tpr, thresholds = roc_curve(y_nbtest, y_nbpred)# Plot heatmap

plt.figure(figsize=(6,4))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for Predicting a Chronic Kidney Disease classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```

Positives (TP) and False Positives (FP)



Histogram of predicted probabilities:

```
# store the predicted probabilities for class 1

y_nbpred1 = classifier.predict_proba(X_nbtest)[:, 1]

# plot histogram of predicted probabilities
# adjust the font size
plt.rcParams['font.size'] = 12


# plot histogram with 10 bins
plt.hist(y_nbpred1, bins = 10)


# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities')


# set the x-axis limit
plt.xlim(0,1)


# set the title
plt.xlabel('Predicted probabilities of having ckd')
plt.ylabel('Frequency')
```



Observations:

We can see that the above histogram is highly positively skewed.

The first column tells us that there are approximately 43 observations with probability between 0.0 and 0.2 who have ckd.

6.4. Comparison with Adaboost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import roc_curve, auc, precision_score, recall_score

classifiers = ['LinearSVM', 'Naive_Bayes', 'KNeighbors', 'SVM_AdaBoost', 'NB_AdaBoost']

models = [SVC(kernel='linear', C=1000.0),
          GaussianNB(),
          KNeighborsClassifier(n_neighbors=3, metric = 'euclidean'),
          AdaBoostClassifier(base_estimator=SVC(kernel='linear', C=1000.0), algorithm='SAMME',n_estimators=100,random_state=100),
          AdaBoostClassifier(base_estimator=GaussianNB(), algorithm='SAMME',n_estimators=100,random_state=100)]

def split(df,label):
    X_tr, X_te, Y_tr, Y_te = train_test_split(df, label, test_size=0.2, random_state=42)
    return X_tr, X_te, Y_tr, Y_te

def acc_score(df,label):
    Score = pd.DataFrame({"Classifier":classifiers})
    j = 0
    acc = []
    prec = []
    rec = []
    f_score = []
    X_train,X_test,Y_train,Y_test = split(df,label)
    for i in models:
        model = i
        model.fit(X_train,Y_train)
        predictions = model.predict(X_test)
        acc.append(accuracy_score(Y_test,predictions))
        prec.append(precision_score(Y_test, predictions))
        rec.append(recall_score(Y_test, predictions))
        f_score.append(f1_score(Y_test, predictions))
        j = j+1
    Score["Accuracy"] = acc
    Score["Precision"] = prec
    Score["Recall"] = rec
    Score["F-Measure"] = f_score
    Score.sort_values(by="Accuracy", ascending=False,inplace = True)
    Score.reset_index(drop=True, inplace=True)
    return Score
```

Classification report:

```
score1 = acc_score(cor_df,y)
score1
```

	Classifier	Accuracy	Precision	Recall	F-Measure
0	Naive_Bayes	0.9875	0.965517	1.000000	0.982456
1	NB_AdaBoost	0.9875	0.965517	1.000000	0.982456
2	SVM_AdaBoost	0.9750	0.964286	0.964286	0.964286
3	LinearSVM	0.9625	0.931034	0.964286	0.947368
4	KNeighbors	0.7625	0.604651	0.928571	0.732394

6.5. Conclusion

In this chapter, we presented the modelling phase for the second article. In which features were selected using a Correlation-based Feature Selection (CFS) and AdaBoost was used for ensemble learning to improve the detection of CKD. KNearest Neighbour algorithm (kNN), Naive Bayes and Support Vector Machine (SVM) was used as base classifier

7. Evaluation

7.1. Introduction

In this chapter, we will focus on the fifth phase which is the evaluation phase. After choosing and running the models, we need to evaluate the results to determine if there is some reason why these models are not accurate and also justify why they are appropriate to use. This section involves models' comparison of both articles.

7.2. Comparison

7.2.1. Models comparison for article 1

The table shown below shows the algorithms comparison of the first articles based on the following metrics, which are accuracy, precision, recall and f1-score.

	SVM	KNN	Decision Tree	Random Forest
Accuracy	0.98	0.97	0.94	0.985
Precision	0.98	0.948	0.96	0.985
Recall	0.96	0.97	0.94	0.99
F1-score	0.95	0.96	0.95	0.985

7.2.2. Compare algorithms with our data preparation enhancements.

Classifier	Accuracy	Precision	Recall	F1-score
Linear SVM	1	1	1	1
Random Forest	1	1	1	1
Decision Tree	0.985	0.965	1	0.982
Naive Bayes	0.875	0.965	1	0.982
SVM AdaBoost	0.9875	1	0.9642	0.981
Naive Bayes_AdaBoost	0.9875	0.965	1	0.98
KNN	0.975	0.933	1	0.965

7.2.3. Compare algorithms with our data augmentation enhancements.

Classifier	Accuracy	Precision	Recall	F-Measure
Random Forest	1.00	1.000000	1.000000	1.000000
Decision Tree	0.99	0.981818	1.000000	0.990826
Linear SVM	0.98	0.964286	1.000000	0.981818
SVM_AdaBoost	0.97	0.947368	1.000000	0.972973
Naive_Bayes	0.95	0.915254	1.000000	0.955752
NB_Adaboost	0.95	0.915254	1.000000	0.955752
KNeighboors	0.84	0.796875	0.944444	0.864407

7.2. Conclusion

During this chapter, we presented the most important phase which is the evaluation phase. We evaluated all models of articles 1 and 2. This detailed evaluation will be based mainly on the feature selection method and on our enhanced data preparation.

8. Deployment

8.1. Introduction

A model is not particularly useful unless the customer can utilise it . Machine learning model deployment is the process of placing a finished machine learning model into a live environment where it can be used for its intended purpose.

During this chapter, the deployment phase will be presented for our best model extracted from the evaluation phase which is Naive Bayes.

8.2. Deployment phase

We can use **Streamlit**, an open source app framework in Python language. It helps us create web apps for data science and machine learning in a short time. It is compatible with major Python libraries such as scikit-learn, Keras, PyTorch, SymPy, NumPy, pandas, Matplotlib etc.

Let’s first start by exporting our model as the figure below shows:

```
import joblib

joblib.dump(classifier, 'NBpipe.joblib')
```

We then read that model like so:

```
import streamlit as st
from utils import columns
import numpy as np
import pandas as pd
import joblib

model = joblib.load('NBpipe.joblib')
```

Finally, by running the Streamlit app we get the interface shown below where a visitor can fill out the form with real medical details and get an accurate estimate of whether they have chronic kidney disease or not.

Do you have chronic kidney disease?

Specific Gravity

Albumin

Blood Glucose

Serum Creatinine

Hemoglobin

Packed Cell Volume

Red Blood Cell Count

Do you have hypertension ?

Yes

Yes No

Do you have diabetes ?

Yes

Yes No

Predict

8.3. Conclusion

This chapter presented the project architecture as well as a detailed explanation of the solution's code step by step along with screenshots in the annex A. Finally, the overall output was displayed after testing the code to show the extracted text from an invoice.

9. Enhancements

9.1. Introduction

In this chapter, we will be outlining the various improvements we performed by comparison to both articles. While mirroring both articles is great for learning and growing our knowledge, it is imperative that we also critically evaluate scientific publications to establish three important components of the learning process:

Grasping why certain methods are used and in which context.

Building an elastic imagination for data understanding

Applying new methods through independent search

This is why we have dedicated this chapter to showcase our own innovative ideas and how we applied concepts that were not considered in the two scientific publications.

9.2. Data preparation

Blood pressure:

First of all, we start with the blood pressure features which are Dias blood pressure, hypertension, and coronary artery disease as well as age.

Here we can see that the average diastolic blood pressure for people with hypertension is 81.31 and without hypertension is 73.65.

```
#the average diastolic blood pressure for people with and without hypertension
data[['Dias_blood_pressure','hypertension']].groupby(['hypertension'], as_index = False).mean()
```

	hypertension	Dias_blood_pressure
0	no	73.651452
1	yes	81.310345

For people who have hypertension, the average blood pressure is equal to 81.31, and the median blood pressure is equal to 80.

For people who don't have hypertension, average blood pressure equals 73.65, and median blood pressure equals 70.

So, for treating missing values for the feature blood pressure, we will proceed as follows:

If the person has hypertension the missing value will be replaced by the average blood pressure of people that have hypertension.

Otherwise, it will be replaced by the average blood pressure of people that don't.

For the feature hypertension as you can see here, we only have two missing values.

lood_pressure	ur_specific_gravity	ur_albumin	ur_sugar	red_blood_cells	ur_pus_cell	ur_ pus_cell clumps	ur_bacteria	blood glucose random
70.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	70.0
60.0	1.025	0.0	0.0	normal	normal	notpresent	notpresent	116.0

These two patients don't have high blood pressure or suffer from any illness.

They are in a healthy situation. So, we will assign them the value 'no' to the hypertension feature.

Diabetes and blood glucose:

Diabetes and blood glucose are moderately Correlated with a coefficient of 0,53. Therefore, they are considered to be good predictors of each other.

```
data['blood glucose random'].corr(data['diabetes'])
```

```
0.5275677530947323
```

Knowing that blood glucose over 140 mg/dL is strong evidence for the diagnosis of diabetes, we can replace the missing values of sick people (diabetes = 1) with the mean blood glucose of sick people. And respectively we can replace the missing values of healthy people (diabetes = 0) with the mean blood glucose of healthy people.

```
data['blood glucose random'].loc[(bgr_null & (data.diabetes == 0))] = data['blood glucose random'].loc[(bgr_null & (data.diabetes
data['blood glucose random'].loc[(bgr_null & (data.diabetes == 1))] = data['blood glucose random'].loc[(bgr_null & (data.diabetes
data['blood glucose random'].isnull().sum()
```

```
0
```

ur_sugar:

Now, we pass to the ur_sugar feature, the urine sugar blood level.

As we can see here, we have 49 missing values for this feature, which represent 12.125%.

```
data['ur_sugar'].isnull().sum()
```

```
49
```

We should treat it carefully. In order to replace the missing values for this feature, we will try to extract as much information as possible from other features such as diabetes and blood glucose random. As we can see in the figure below, we extracted these three features that have missing values in ur_sugar feature.

```
data[['ur_sugar','blood glucose random','diabetes']].loc[(data['ur_sugar'].isnull())]
```

	ur_sugar	blood glucose random	diabetes
13	NaN	98.0	1
17	NaN	114.0	0
21	NaN	205.0	1
30	NaN	93.0	1
37	NaN	137.0	1
50	NaN	91.0	1
57	NaN	93.0	1
59	NaN	205.0	1
78	NaN	158.0	0
81	NaN	360.0	1
82	NaN	104.0	0
85	NaN	205.0	1
86	NaN	415.0	1

In the figure below, we have sorted the features that have a high blood glucose random.

	ur_sugar	blood glucose random	diabetes
88	0.0	251.0	1
198	2.0	252.0	1
58	0.0	253.0	1
212	4.0	253.0	1
210	2.0	255.0	1
140	4.0	256.0	0
175	0.0	261.0	0
18	3.0	263.0	1
27	4.0	264.0	1
137	0.0	268.0	1
231	NaN	269.0	1

After describing the feature ur_sugar, we have found that 75% of patients have the value zero and they come from different categories..

```
data['ur_sugar'].describe()

count    351.000000
mean       0.450142
std        1.099191
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        5.000000
Name: ur_sugar, dtype: float64
```

So, for this feature all missing values will be replaced by zero.

Red Blood Cells (RBC):

Upon observing the normal as well as the abnormal red blood cells, we found that there is a clear separation between those two categories of red blood cells (normal/abnormal). In a way that:

75% of the patients who have normal red blood cells have a red blood cell count above 4.5

75% of the patients who have abnormal red blood cells have a red blood cell count below 4.2

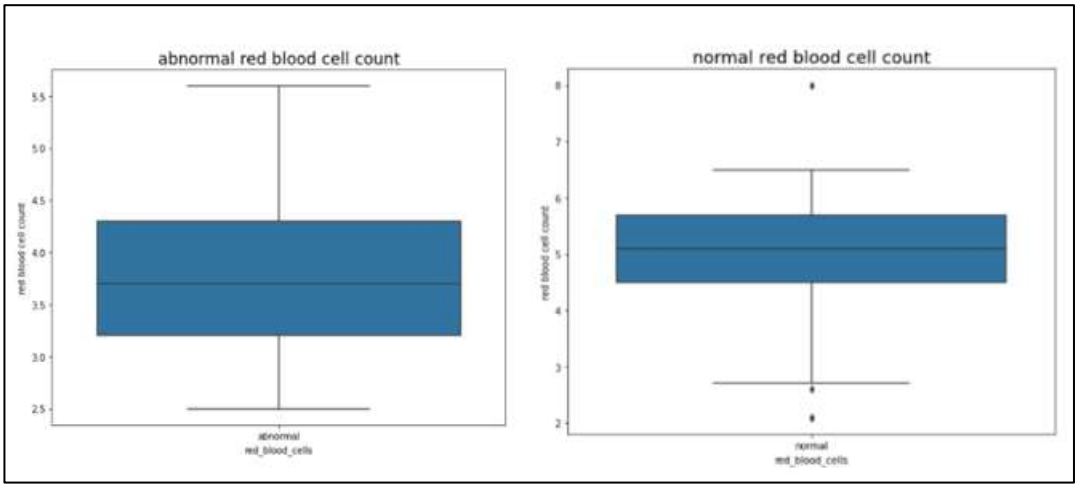
However, it is important to note that the normal values of red blood cells are as follows:

Adult males: 4.7 to 7.2 million/microliter

Adult females: 4.2 to 5.0 million/microliter

Children: 3.8 to 5.5 million/microliter

As we don't have the sex of the patient, the healthy range of RBC count should be from 4.8 to 7.2 Million/microliter for adults above 16 years old, and from 3.8 to 5.5 million/microliter for kids. Anything below the normal values will be considered abnormal red blood cells.



Digging deeper into the dataset, we concluded that out of 16 children in this data, 6 of them have normal red blood cell count, while 2 have abnormal red blood cells and 8 have missing values.

It is important to note that all the kids have red blood cell counts within the normal range of 3.8 to 5.5 million/microliter.

As a result, the missing values in the red blood cell column will be filled with 'normal' for patients under 16 years old (children).

```
data['red_blood_cells'].loc[(data['age'] < 16)] =
data['red_blood_cells'].loc[(data['age'] < 16)].fillna('normal')
```

```
ata['red_blood_cells'].loc[(data['red_blood_cells'].isnull())&(data['red blood cell count'] < 4.5)] =data['red_blood_cells'].loc[
   (data['red_blood_cells'].isnull())&(data['red blood cell count'] < 4.5)].fillna('abnormal')
ata['red_blood_cells'].loc[(data['red_blood_cells'].isnull())&(data['red blood cell count'] >= 4.5)] =data['red_blood_cells'].loc
   (data['red_blood_cells'].isnull())&(data['red blood cell count'] >= 4.5)].fillna('normal')
```

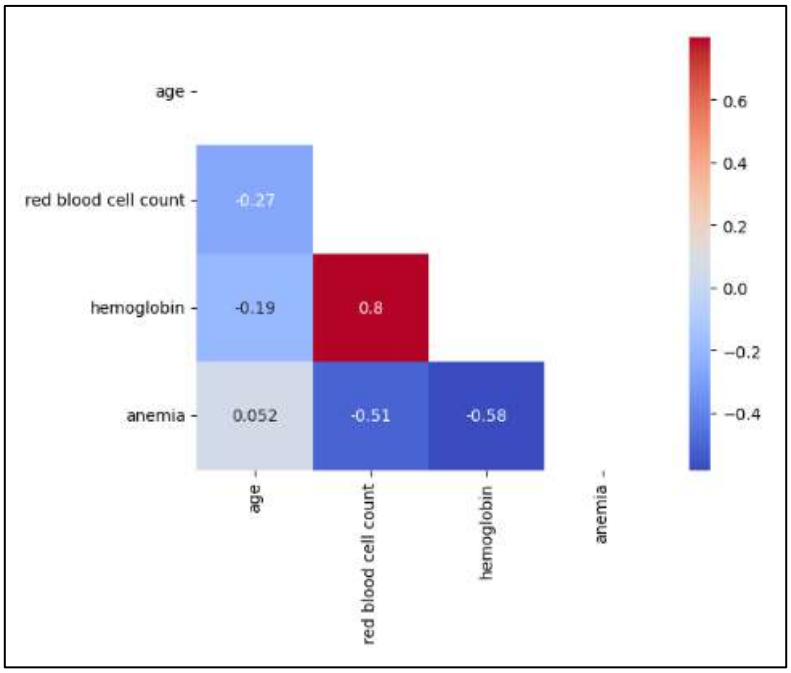
Haemoglobin & Red Blood Cell Count:

As we advance in the analysis of the dataset's features and their correlations with one another, we proceed to test the correlation between age, red blood cell count, haemoglobin, and anemia.

```
#the correlation between ('age','red_blood_cells','red blood cell count','hemoglobin','anemia')
hematology = data[['age','red_blood_cells','red blood cell count','hemoglobin','anemia']].corr()
hematology
```

	age	red blood cell count	hemoglobin	anemia
age	1.000000	-0.268896	-0.192928	0.051753
red blood cell count	-0.268896	1.000000	0.798880	-0.508560
hemoglobin	-0.192928	0.798880	1.000000	-0.583289
anemia	0.051753	-0.508560	-0.583289	1.000000

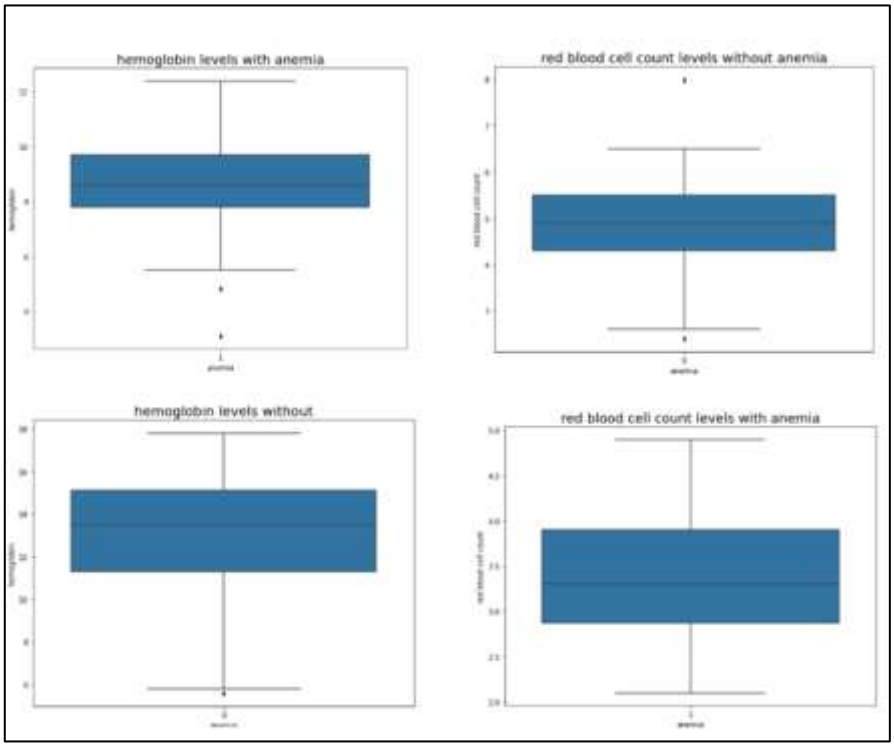
According to the heatmap of the correlation matrix shown below,



There is a strong positive correlation between the red blood cell count and haemoglobin.

There is a moderate positive correlation between haemoglobin and anemia.

There is a moderate negative correlation between anemia and red blood cell count.



After several analyses and boxplots, we noticed the existence of a clear separation in haemoglobin values between people who have anemia and people who do not. And for more specifications

```
#hemoglobin level and red blood cell count without anemia
data[['anemia','hemoglobin','red blood cell count']]
.loc[(data['anemia'] == 0)&(data['hemoglobin'].notnull())].describe()
```

	anemia	hemoglobin	red blood cell count
count	274.0	274.000000	274.000000
mean	0.0	13.452096	4.919069
std	0.0	2.362541	0.816425
min	0.0	5.600000	2.400000
25%	0.0	11.800000	4.500000
50%	0.0	13.500000	4.912017
75%	0.0	15.175000	5.400000
max	0.0	17.800000	8.000000

```
# hemoglobin level and red blood cell count variation with anemia
data[['anemia','hemoglobin','red blood cell count']]
.loc[(data['anemia'] == 1)&(data['hemoglobin'].notnull())].describe()
```

	anemia	hemoglobin	red blood cell count
count	55.0	55.000000	36.000000
mean	1.0	8.610909	3.383333
std	0.0	1.870103	0.741234
min	1.0	3.100000	2.100000
25%	1.0	7.800000	2.875000
50%	1.0	8.600000	3.300000
75%	1.0	9.700000	3.900000
max	1.0	12.400000	4.900000

75% of the patients who have anemia have a haemoglobin level below 9.7 and a red blood cell count below 4

75% of the patients without anemia, have a haemoglobin value above 11.3 and a red blood cell count above 4.3

As we conclude, and according to the results, it is safe to say that a haemoglobin level below 11 is considered some kind of anemia. Nonetheless, a haemoglobin level equal to 11 and a red

blood cell count equal to 4.2 can be used as good thresholds to separate people who have anemia and people who do not. Moreover, these thresholds will also be used to fill the missing values with the mean of each category.

```
# we have established earlier that 4.2 red blood cell count can
# be a good threshold to separate the normal and abnormal red blood cell
rbc_null = data['red_blood_cells'].isnull()
rbcc_low = (data['red blood cell count'] < 4.2)
rbcc_normal = (data['red blood cell count'] > 4.2)
data['red_blood_cells'].loc[rbcc_low] = data['red_blood_cells']
.loc[rbcc_low].fillna('abnormal')
data['red_blood_cells'].loc[rbcc_normal] = data['red_blood_cells']
.loc[rbcc_normal].fillna('normal')
data['red_blood_cells'].value_counts()
```

```
normal      296
abnormal    104
Name: red_blood_cells, dtype: int64
```

pedal edema:

For the “pedal edema” feature we started by getting the number of missing values in each row and check if the records containing null values for pedal edema contain also null values in other columns

We found that the patient 294 disposed of nan value in pedal edema feature .

in order to be able to deal with it we started by getting the possible values for “ pedal edema” , “appetite” and “anaemia”

Through this we can conclude that this patient is clearly healthy,every biological value is in the normal interval and every pathology is denied. he has 3 categorical missing values that we are going to fill with :

'**good**' in 'appetite'

'**no**' in 'anaemia' and 'pedal edema'.

coronary artery disease

Therefore, for “the coronary artery disease” column we proceeded in the same way as the “Pedal edema ” column by checking records with null values.

```
lata.loc[(data['coronary artery disease'].isnull())].T
```

	288	297
age	56.0	53.0
Dias_blood_pressure	70.0	60.0
ur_specific_gravity	1.025	1.025
ur_albumin	0.0	0.0
ur_sugar	0.0	0.0
red_blood_cells	normal	normal
ur_pus_cell	normal	normal
ur_ pus_cell clumps	notpresent	notpresent
ur_bacteria	notpresent	notpresent
blood glucose random	70.0	116.0
blood urea	46.0	26.0
serum creatinine	1.2	1.0

We found two patients with null values in “the coronary artery disease” column (288 & 297)

To deal with these two records we checked first the possible values for this column

Through this we found that both patients seem healthy with pretty normal values, as a result we can assume that they don't have a coronary artery disease.

So, we replaced the missing values with ‘no'.

After dealing with missing data now we proceed to deal with categorical features where we choose to use encode the clean ones

To do the replacement we used the get_dummies() function that we apply to the columns

“anaemia”,”class”,”pedal edema”, ”appetite”,”hypertension”,”coronary artery disease”

```
dummy_anemia = pd.get_dummies(data['anemia'] , drop_first = True)
data.drop(['anemia'] , axis = 1 , inplace = True)
data = data.join(dummy_anemia)
data.rename(columns={'yes': 'anemia'}, inplace = True)
data
```

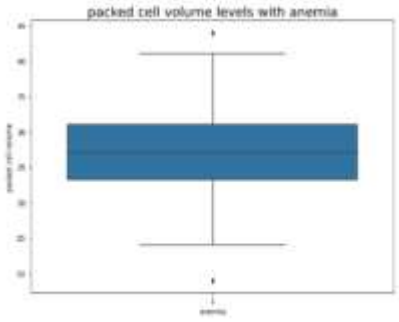
packed cell volume	white blood cell count	red blood cell count	hypertension	coronary artery disease	appetite	pedal edema	class	diabetes	anemia
44.0	7800.0	5.2	yes	no	good	no	ckd	1	0
38.0	6000.0	NaN	no	no	good	no	ckd	0	0
31.0	7500.0	NaN	no	no	poor	no	ckd	1	1
32.0	6700.0	3.9	yes	no	poor	yes	ckd	0	1
35.0	7300.0	4.6	no	no	good	no	ckd	0	0

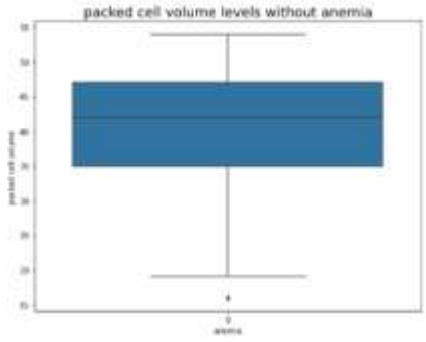
Packed cell volume:

Packed cell volume is the volume percentage (vol%) of red blood cells (RBC) in blood, measured as part of a blood test. It is normally 36.1–50.3%.

It is known to be correlated with haemoglobin and is a good indicator of anaemia.

let's calculate and use this correlation, if it exists, to predict the missing values of the packed cell volume variable.





correlation matrix:

```
hemo_cell_volume = data[['packed cell volume','hemoglobin','anemia']].corr()
hemo_cell_volume
```

	packed cell volume	hemoglobin	anemia
packed cell volume	1.000000	0.895094	-0.564253
hemoglobin	0.895094	1.000000	-0.601843
anemia	-0.564253	-0.601843	1.000000

through the correlation matrix we can extract that we have indeed a strong positive relationship between haemoglobin and packed cell volume with a correlation coef = 0.895

We also have a moderate negative relationship between anaemia and packed cell volume.

In order to be more sure about this relationship we dig more into the description of null values in "packed cell volume " and its effect on the values of anaemia and haemoglobin.

From the previous description we can see that 75% of the patients who has anaemia , have a packed cell volume below 31%

and 75% of the patients without anaemia

, have a packed cell volume above 35%.

As a result of all these observations we will use the correlation between the haemoglobin and the packed cell volume to predict the missing values

In order to do this let's build a linear regression model.

```
model = LinearRegression()

x = data[['hemoglobin']].loc[(data['packed cell volume'].notnull())]
y = data['packed cell volume'].loc[(data['packed cell volume'].notnull(

model.fit(x,y)
model.score(x,y)
```

```
0.8011937878895059
```

After fitting the linear regression model we got a score of 80% which is good score to start replacing the missing values now

```
print('y = {} x + {}'.format(model.coef_ , model.intercept_))
```

```
y = [2.80969261] x + 3.2046230662399537
```

Age:

For the “Age” feature we started by getting the number of missing values in each row and checking if the records contained null values.

```
data.loc[(data.age.isnull())]
```

	age	Dias_blood_pressure	ur_specific_gravity	ur_albumin
30	NaN	70.0	NaN	NaN
73	NaN	100.0	1.015	2.0
112	NaN	60.0	1.015	3.0
116	NaN	70.0	1.015	4.0
117	NaN	70.0	1.020	0.0
169	NaN	70.0	1.010	0.0
191	NaN	70.0	1.010	3.0
203	NaN	90.0	NaN	NaN
268	NaN	80.0	NaN	NaN

```
data.loc[(data.age.isnull())].isnull().sum().sum()
```

```
30
```

Through these results, we have 9 patients without age. However, it is a very important factor when it comes to pathologies,

Add to that, these 9 rows contain many missing values in other columns.

As a result, filling them all would be a mistake.

So we are going to drop them (they represent only 2.25% of the data).

```
data.dropna(subset=['age'], inplace = True)
data.age.isnull().sum()
```

```
0
```

'ur_bacteria','ur_pus_cell clumps', 'ur_pus_cell':

For those features, we started by getting the number of their missing values in each row

in order to be able to deal with null values we started by getting the possible values for " ur_bacteria", "ur_pus_cell clumps" and "ur_pus_cell"

The missing values in the urine bacteria happen to be in the same rows as the 4 missing values of the urine pus cell clumps.

Also, the one missing value in the ur_pus_cell is in one of these rows too.

The 4 patients seem to be young, healthy and have no pathologies, they have normal biological values and no signs of renal infection or any other infection, they are also predicted as non-susceptible to catching a kidney disease.

As a result, we think it's safe to fill the 5 missing values with '**not present**' in the urine bacteria and urine pus cell clumps columns and '**normal** in the urine pus cell column.

After dealing with null values, let's encode the 4 remaining categorical columns ("red_blood_cells","ur_pus_cell", "ur_ pus_cell clumps", "ur_bacteria")

```
dummy_rbc = pd.get_dummies(data['red_blood_cells'] )
data = data.join(dummy_rbc)
data.drop(['red_blood_cells','normal'] , axis = 1 , inplace = True)
data.rename(columns={'abnormal': 'abnormal_red_blood_cells'}, inplace = True)
data[['abnormal_red_blood_cells']]
```

	abnormal_red_blood_cells
0	0
1	0
2	0
3	0
4	0
...	...
386	0
387	0
388	0
389	0
390	0

Now, let's see if we have any rows with 4 or more missing values at the same time.

As we had mentioned earlier, filling in many missing values in one row or one column can mislead the study and the model later.

Through those results, we can see that 24 patients dispose of several missing values in different columns (at least 4).

These patients are from different ages and the standard deviation is relatively big in each variable's values.

After checking the variety in this list of patients and the number of missing values, we conclude it's not a good idea to try to fill these missing values with anything. Let's just drop them since it's 24 rows (6.13% of the remaining data)

```
data.dropna(axis=0, thresh=23,inplace=True)
data.reset_index(drop=True, inplace=True)
```

Blood urine and serum creatinine :

Now, we pass to the feature blood_urea.

As the figure below shows, the correlation matrix between the features of blood urea and serum creatinine has a high correlation equal to 0.813.

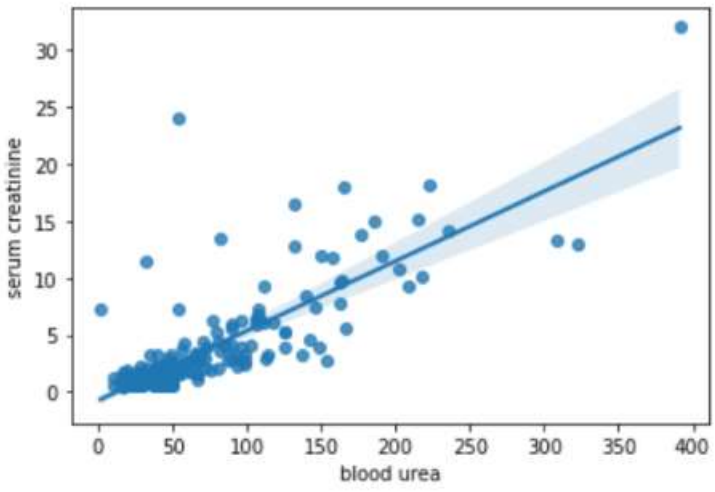
```
data[['blood urea','serum creatinine']].corr()
```

	blood urea	serum creatinine
blood urea	1.00000	0.81393
serum creatinine	0.81393	1.00000

Also, these two features present a linear relationship, when the blood urea increases, the serum creatinine also increases.

```
sns.regplot(data['blood urea'],data['serum creatinine'])
```

```
<AxesSubplot:xlabel='blood urea', ylabel='serum creatinine'>
```



After extracting data that have null values for the feature blood urine, we have noticed

that blood urine and serum creatinine are generally associated with renal malfunction

or even renal insufficiency and renal failure. we can group all the columns with missing

values by pathologies related directly to kidney problems.

n this part, we will proceed with working with 3 patients that have null

serum creatinine having indexes 43, 99, and 308, as shown in the figure below

```
data.loc[(data['blood urea'].isnull())].T
```

	43	99	208	243	264	308
age	63.00	47.00	20.000	80.00	24.000	71.000
Dias_blood_pressure	80.00	100.00	60.000	70.00	80.000	60.000
ur_specific_gravity	1.01	1.01	1.025	1.02	1.025	1.025
ur_albumin	2.00	NaN	0.000	0.00	0.000	0.000
ur_sugar	2.00	0.00	0.000	0.00	0.000	0.000
blood glucose random	117.00	122.00	117.000	117.00	125.000	117.000
blood urea	NaN	NaN	NaN	NaN	NaN	NaN
serum creatinine	3.40	16.90	NaN	NaN	NaN	0.900

Now, we will use the linear model to predict the blood urea on them.

```
w = data[['serum creatinine']].loc[(data['blood urea']
.notnull())&(data['serum creatinine'].notnull())]
z = data['blood urea'].loc[(data['blood urea']
.notnull())&(data['serum creatinine'].notnull())]
model.fit(w,z)
print(' blood urea = {} x serum creatinie + {}'
.format(model.coef_ , model.intercept_ ))
```

```
 blood urea = [10.86665584] x serum creatinie + 27.0608887007327
```

For this second part, we will work on the three patients that have null values for both blood urea and serum creatinine, having the following indexes 208, 243, and 264.

We noticed that they have a healthy situation. We will proceed by assigning to them the average of similar healthy patients.

```
print('average blood urea for healthy people = ',
      data['blood urea'].loc[data[my_list].eq(0).all(1)].mean())
print('average serum creatinine for healthy people =',
      data['serum creatinine'].loc[data[my_list].eq(0).all(1)].mean())
```

```
average blood urea for healthy people =  32.701245634301316
average serum creatinine for healthy people = 0.8700000000000001
```

We tried our best to keep the data clean and as realistic as possible, but the rest of the missing values are hard to get.

In real life, we would ask for a new blood or urine test if we really suspect a kidney disease. In this project, let's work with what we have.

Since the study is around the kidney disease variable, let's group all the columns with missing values by the CKD column and choose the right values to replace missing values in each column of them.

```
# missing values in cases of not kidney disease patients
data.loc[data.ckd == 1].isnull().sum()
```

```
age                             0
Dias_blood_pressure             0
ur_specific_gravity            12
ur_albumin                     12
ur_sugar                        0
blood glucose random           10
blood urea                      0
serum creatinine                0
sodium                         28
potassium                      28
hemoglobin                      0
packed cell volume              0
white blood cell count         41
red blood cell count            0
diabetes                        0
anemia                          0
ckd                             0
pedal edema                     0
poor                            0
hypertension                    0
coronary artery disease         0
abnormal_red_blood_cells        0
abnormal_ur_pus_cell            0
ur_pus_cell_clumps_present      0
ur_bacteria_present             0
dtype: int64
```

```
data[['white blood cell count', 'potassium', 'sodium',
'ur_specific_gravity', 'ur_albumin', 'ckd']]
.groupby('ckd').mean().T
```

ckd	0	1
white blood cell count	7710.144928	9183.958774
potassium	4.332394	4.956564
sodium	141.823944	134.661427
ur_specific_gravity	1.022411	1.014279
ur_albumin	0.000000	1.806767

Normal white blood cell count in a healthy adult is between 4,000 and 11,000 WBCs per microliter.

Normal potassium levels should be between 3.7 and 5.2 milliequivalents per litre.

The normal blood sodium levels range from 135 to 145 mEq/L

The normal range for urine specific gravity is 1.005 to 1.030.

The normal range of urine albumin is around 0-8 mg/dl.

We can see the difference between healthy and ill people in the extreme values(min/max). But, since in most of the cases either; more than 50% of the data is overlapping it's hard to separate them. the difference between the means is always clear even though it stills in the range of normal values

Let's replace the missing values in these columns with the mean value for healthy and kidney disease patients.

```
mean_wbcc_notckd = data['white blood cell count']
.loc[(data['white blood cell count']
.notnull())&(data.ckd == 0)].mean()
mean_wbcc_ckd = data['white blood cell count']
.loc[(data['white blood cell count']
.notnull())&(data.ckd == 1)].mean()
mean_pot_notckd = data['potassium']
.loc[(data['potassium'].notnull())
&(data.ckd == 0)].mean()
mean_pot_ckd = data['potassium']
.loc[(data['potassium'].notnull())
&(data.ckd == 1)].mean()
mean_sod_notckd = data['sodium']
.loc[(data['sodium'].notnull())
&(data.ckd == 0)].mean()
mean_sod_ckd = data['sodium']
.loc[(data['sodium'].notnull())
&(data.ckd == 1)].mean()
mean_usg_notckd = data['ur_specific_gravity']
.loc[(data['ur_specific_gravity'].notnull())
&(data.ckd == 0)].mean()
mean_usg_ckd = data['ur_specific_gravity']
.loc[(data['ur_specific_gravity'].notnull())
&(data.ckd == 1)].mean()
mean_ualb_notckd = data['ur_albumin']
.loc[(data['ur_albumin'].notnull())
&(data.ckd == 0)].mean()
mean_ualb_ckd = data['ur_albumin']
.loc[(data['ur_albumin'].notnull())
&(data.ckd == 1)].mean()
```

Data Scaling:

As we reached the last steps of our data pre-processing, it got to our attention that the continuous variables have different scales. As a result, we proceed now to scaling our data and choosing our features.

```
#scaling the continuous variables
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
cont_data_scaled = scaler.fit_transform(cont_data)
df_scaled = pd.DataFrame(cont_data_scaled)
df_scaled.rename(columns={0 : 'age', 1 :'Dias_blood_pressure',
                          2 :'ur_specific_gravity', 3 :'ur_albumin',
      4 :'ur_sugar', 5 :'blood glucose random', 6 :'blood urea',
                          7 :'serum creatinine',
      8 :'sodium', 9 :'potassium',10  :'hemoglobin',
                          11 :'packed cell volume',
       12 :'white blood cell count', 13 :'red blood cell count'},
                 inplace = True)
df_scaled
```

Thus, for a better observation of our data as complete, joining the scaled variables with the binary ones is the next step.

There was a lot of missing data in the dataset that needed a lot of exploratory analysis and required attentive oversight. We had to use the correlation between multiple columns

by fully exploiting the business understanding perspective to manage to handle these missing values in an appropriate manner.

9.3. Models

9.3.1. KNN model

As we hit excellent results using the **euclidean** metric for the k-nearest neighbors model, we processed to applying the **manhattan** metric to observe the result differences and accuracy predicting.

```
# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
kclassifier = KNeighborsClassifier(n_neighbors = 3, metric = 'manhattan')
kclassifier.fit(X_ktrain, y_ktrain)

# Predicting the Test set results# In[ ]:
y_kpred = kclassifier.predict(X_ktest)

# compute and print accuracy score
print('Model accuracy score with KNN and n_neighbours=3 : {0:0.4f}'. format(accuracy_score(y_ktest, y_kpred)))
```

```
y_kpred_train = kclassifier.predict(X_ktrain)

print('Training set score: {:.4f}'.format(kclassifier.score(X_ktrain, y_ktrain)))

print('Test set score: {:.4f}'.format(kclassifier.score(X_ktest, y_ktest)))
```

```
Training set score: 0.9600
Test set score: 0.9700
```

The results were exactly the same for both metrics with no accuracy prediction changing .

9.3.2. Naïve Bayes

Classification report:

```
print("Naive Bayes Classifier Model Accuracy:", accuracy_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model Precision:", precision_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model Recall:", recall_score(y_nbtest, y_nbpred))

print("Naive Bayes Classifier Model F-measure:", f1_score(y_nbtest, y_nbpred))
```

```
Naive Bayes Classifier Model Accuracy: 0.97
Naive Bayes Classifier Model Precision: 0.918918918918919
Naive Bayes Classifier Model Recall: 1.0
Naive Bayes Classifier Model F-measure: 0.9577464788732395
```

Observation:

Model accuracy went from 0.9500 up to 0.9700. Which leads to the conclusion that careful and comprehensive data preparation ensures more accurate and meaningful analyses.

Check for overfitting and underfitting:

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(classifier.score(X_nbtrain, y_nbtrain)))

print('Test set score: {:.4f}'.format(classifier.score(X_nbtest, y_nbtest)))
```

```
Training set score: 0.9500
Test set score: 0.9700
```

The training-set accuracy score is 0.9500 while the test-set accuracy is 0.9700. These two values are quite comparable. So,there is no sign of overfitting.

Compare model accuracy with null accuracy:

```
# check class distribution in test set

y_nbtest.value_counts()
```

```
0    66
1    34
Name: class, dtype: int64
```

We can see that the occurrences of the most frequent class is 66. So, we can calculate null accuracy by dividing 66 by total number of occurrences.

```
# check null accuracy score

null_accuracy = (66/(65+34))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

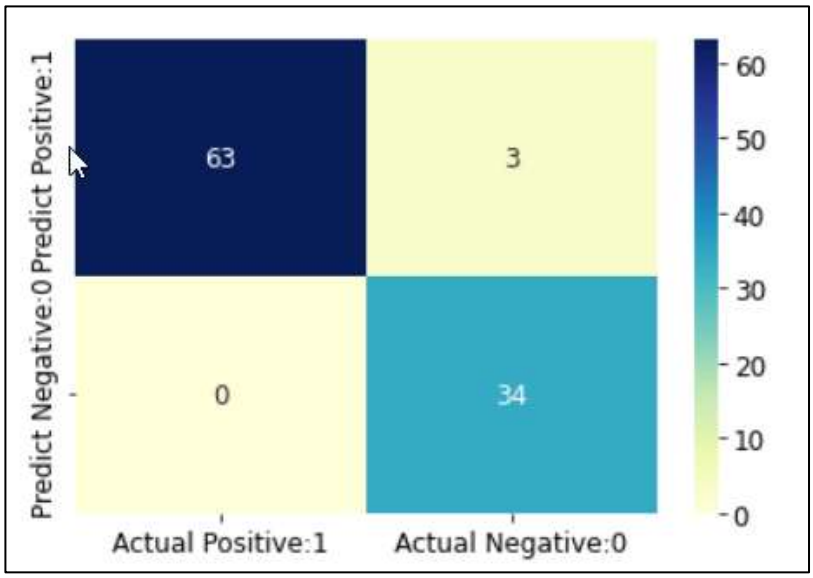
```
Null accuracy score: 0.6667
```

We can see that our model accuracy score is 0.9700 but null accuracy score is 0.6667. So, we can conclude that our Gaussian Naive Bayes Classification model is doing a very good job in predicting the class labels.

Confusion matrix:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

nbcm = confusion_matrix(y_nbtest, y_nbpred)

print('Confusion matrix\n\n', nbcm)

print('\nTrue Positives(TP) = ', nbcm[0,0])

print('\nTrue Negatives(TN) = ', nbcm[1,1])

print('\nFalse Positives(FP) = ', nbcm[0,1])

print('\nFalse Negatives(FN) = ', nbcm[1,0])
```

```
Confusion matrix

 [[63  3]
 [ 0 34]]

True Positives(TP) =  63

True Negatives(TN) =  34

False Positives(FP) =  3

False Negatives(FN) =  0
```



The confusion matrix shows 63+ 34= 97 correct predictions and 3 + 0 = 3 incorrect predictions.

In this case, we have:

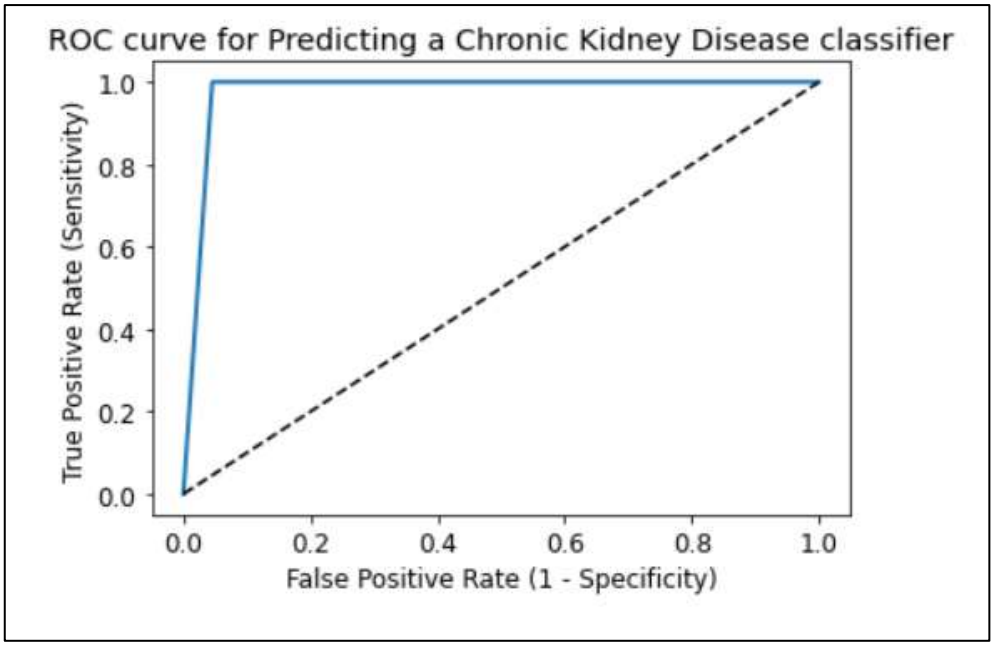
True Positives (Actual Positive:1 and Predict Positive:1) - 63

True Negatives (Actual Negative:0 and Predict Negative:0) - 34

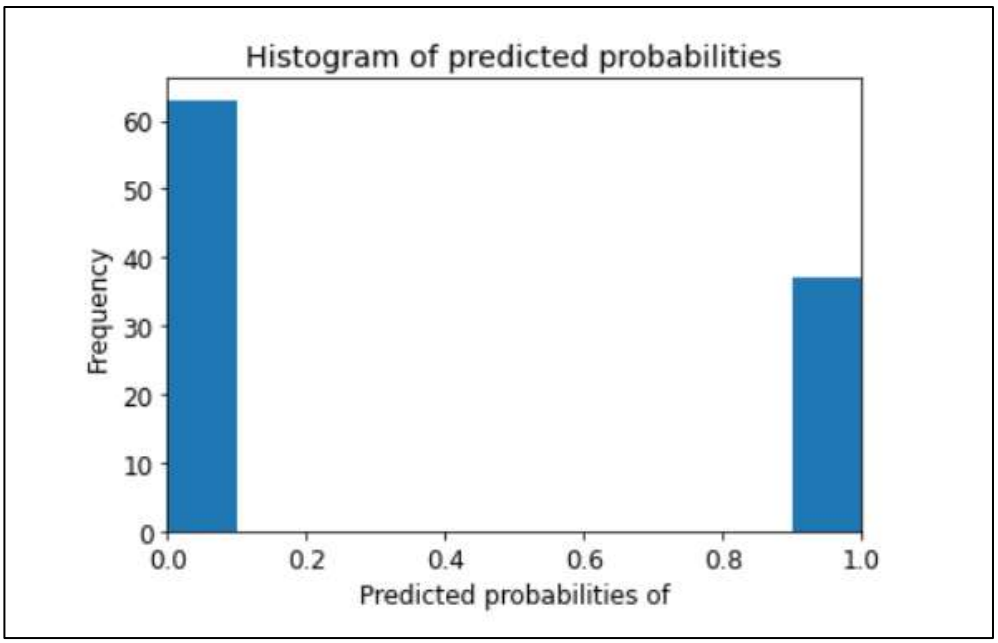
False Positives (Actual Negative:0 but Predict Positive:1) - 3 (Type I error)

False Negatives (Actual Positive:1 but Predict Negative:0) - 0 (Type II error)

ROC curve:



Histogram of predicted probabilities:



Observations:

We can see that the number of patients with ckd has increased compared to the results we got in Article 2. Now that we treated the null values

9.3.3. Decision Tree

In order to enhance the Decision Tree model used in the first article, we opted for the filter technique for feature selection.

Starting by checking the presence of constant features.

```
qconstant_filter = VarianceThreshold(threshold=0.01)
qconstant_filter.fit(X_train)
len(X_train.columns[qconstant_filter.get_support()])
```

```
20
```

```
qconstant_columns = [column for column in X_train.columns
                     if column not in X_train.columns[qconstant_filter.get_support()]]

print(len(qconstant_columns))
```

```
2
```

```
for column in qconstant_columns:
    print(column)
```

```
age
sodium
```

As we can see we don't have any constant features. Moving now to the second type which is quasi constant features and check their presence

As we can see we have two quasi constant features which are the Sodium and age feature so we will drop them.

```
X_train = qconstant_filter.transform(X_train)
X_test = qconstant_filter.transform(X_test)
```

```
X_train.shape, X_test.shape
```

```
((230, 20), (99, 20))
```

After checking constant and quasi constant features, let's check for the duplicate features and the correlated features

```
print(X_train_T.duplicated().sum())
```

```
0
```

We don’t have any duplicated features, moving now to the correlated ones

```
correlated_features = set()
X_train = pd.DataFrame(X_train)
correlation_matrix = X_train.corr()
for i in range(len(correlation_matrix .columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)
```

```
len(correlated_features)
```

```
2
```

```
print(correlated_features)
```

```
{9, 11}
```

We have 2 correlated features so we will drop feature 5

After applying the filter methods to our data set let’s check it’s efficiency by comparing it to the methods used in article. To do so we will use the the decision tree classifier on both data set and test the accuracy, recall and f1 Score for both cases (with and without filter methods).

Decision tree classifier model

With filter methods

```
dtree_2 = DecisionTreeClassifier()
dtree_2 = dtree_2.fit(X_train, y_train)

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
y_pred = dtree_2.predict(X_test)
print(" accuracy: {0:.5f}".format(accuracy_score(y_test, y_pred)))
print("precision: {0:.5f}".format(precision_score(y_test, y_pred)))
print("   recall: {0:.5f}".format(recall_score(y_test, y_pred)))
print(" f1 score: {0:.5f}".format(f1_score(y_test, y_pred)))

 accuracy: 0.93939
precision: 0.96296
   recall: 0.92857
 f1 score: 0.94545
```

Without filter methods

As we can see, the application of filter methods to our data set improved the model performance and accuracy from 0.9393 to 0.9494.

Comparison

Table 2 : Comparison table for Decision Tree model

	Decision Tree with RFE	Decision Tree with filter
Accuracy	0.939	0.949
Precision	0.962	0.963
Recall	0.928	0.946
F1 score	0.945	0.954

Imbalanced data:

Imbalanced data refers to those types of datasets where the target class has an uneven distribution of observations, i.e one class label has a very high number of observations

and the other has a very low number of observations. This is certainly the case with our dataset. Thus, we propose an idea that may help solve this issue: **Data Augmentation.**

Data augmentation in data analysis is a technique used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.

We've opted to use **SMOTE**, Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique for increasing the number of cases in your dataset in a balanced way. The component works by generating new instances from existing minority cases that you supply as input.

We can use smote to perform oversampling with the code shown below:

```
from imblearn.over_sampling import SMOTE

over = SMOTE(sampling_strategy=1)
X_over,y_over = over.fit_resample(X,y)
```

As we can see from the figure below, oversampling made our models perform worse for the most part except for **Random Forest** which kept its 100% score across the board and **Decision Tree** which also saw an improvement in all its metrics as well.

```
score_2 = acc_score(rfecv_df,y)
score_2
```

	Classifier	Accuracy	Precision	Recall	F-Measure
0	LinearSVM	1.0000	1.000000	1.000000	1.000000
1	Random Forest	1.0000	1.000000	1.000000	1.000000
2	Decision Tree	0.9875	0.965517	1.000000	0.982456
3	Naive_Bayes	0.9875	0.965517	1.000000	0.982456
4	SVM_AdaBoost	0.9875	1.000000	0.964286	0.981818
5	NB_AdaBoost	0.9875	0.965517	1.000000	0.982456
6	KNeighbors	0.9750	0.933333	1.000000	0.965517

```
score1 = acc_score(X_over,y_over)
score1
```

	Classifier	Accuracy	Precision	Recall	F-Measure
0	Random Forest	1.00	1.000000	1.000000	1.000000
1	Decision Tree	0.99	0.981818	1.000000	0.990826
2	LinearSVM	0.98	0.964286	1.000000	0.981818
3	SVM_AdaBoost	0.97	0.947368	1.000000	0.972973
4	Naive_Bayes	0.95	0.915254	1.000000	0.955752
5	NB_AdaBoost	0.95	0.915254	1.000000	0.955752
6	KNeighbors	0.84	0.796875	0.944444	0.864407

9.4. Conclusion

In Conclusion, although we've identified multiple areas of improvement in these two articles. We can say that both articles have done an excellent job in tackling the chronic kidney disease dataset study. We've clearly demonstrated that our own data preparation has proven to be far more superior than the statistical methods used by both articles.

This said, it is also a valid point of view to consider our approach more inclined to be subjective to the data. Although we have not noticed any indicators that could imply this, it remains a valid opinion.

Conclusion and Perspectives

The main goal of this research is to establish a model to ease dealing with the complexities of medical data and find an augmentative solution for detecting CKD in its initial stage using machine learning models.

This overall project emphasises the capacity of machine learning algorithms to detect CKD using the fewest possible tests or features. For detecting CKD at an early stage, this research employed Machine Learning.

The models applied in this study were referenced from two scientific articles given within an academic project.

The models that were used are K Nearest Neighbour (KNN), Support Vector Machine (SVM), Decision Tree, Random Forest, Naive Bayes. Also, multiple feature selection techniques were used such as Recursive Feature Elimination with Cross-Validation (RFECV) and Correlation based Feature Selection (CFS) that really enhanced the overall results of our models.

An enhanced study was presented in this research conducted by the owner of this report. First, we improved the data preparation phase, presenting feature selection techniques such as filter and wrapper methods.

In conclusion, it would prove difficult to end this study without a deployment phase, which is an important step in the CRISP-DM life cycle of a machine learning project. In fact, a model is not particularly useful unless the customer can utilise it . Machine learning model deployment is the process of placing a finished machine learning model into a live environment where it can be used for its intended purpose.