Machine Learning Final Project Report

Topic: Chronic Kidney Disease Prediction Using Machine Learning

Course	: COMP6577001 - Machine Learning
Name of Lecturer	: I Gede Putra Kusuma Negara, B.Eng., Ph.D.
Class	: LC01
Major	: Computer Science

Group Members :

1. Ivan Setiawan - 2602109473
2. Avriella Sofianti - 2602138372
3. Kent Alber Fredson - 2602112543

Table of Contents

Unit I - Business Understanding and Data Acquisition

Chronic kidney disease (CKD) is a longstanding kidney disease that can lead to renal failure. It is a significant public health issue worldwide. This disease is estimated to affect about 13.4% of the world population. CKD directly impacts on the global burden of morbidity and mortality through its effect on cardiovascular risk and end-stage kidney disease (ESKD). The increase in the prevalence of diabetes mellitus, hypertension, obesity, and aging are the major leading factors contributing to the rise of CKD.

CKD has received a lot of attention in the medical world due to its high death rate. However, this chronic renal disease can be prevented from progressing to kidney failure if diagnosed and treated early (Iftikhar, H., et al., 2023). It can be difficult to detect because its symptoms may develop slowly or not at all and aren't specific to the disease. Early detection using machine learning can be very helpful in this case. Using machine learning techniques for predictive analysis can help diagnose CKD. Our team employed and test various machine learning algorithms which have been proven to achieve high accuracy such as logistic regression (LR), k-nearest neighbor (KNN), support vector classifier (SVC), gaussian naive bayes (GNB), decision tree (DT), random forest (RF), and extreme gradient boosting (XGBoost) algorithms (Haque, A. et al., 2021), as well as Decision Tree (DT) (Khalid, H. et al., 2023).

In this project, we decided to create a web app by using the best of the tested machine learning algorithms in the analysis, which is XGBoost, to effectively predict whether a patient suffers from CKD or not (Kale, Y., 2024). Our team sees the importance of CKD detection to minimize health complications, including hypertension, anemia, mineral bone disorder, and other complications. We aspire for this program to serve as a catalyst for raising awareness about CKD, facilitating research endeavors, and potentially integrating into medical practice for CKD detection methods.

The dataset we used is referenced from the UCI Machine Learning Repository, obtained from the link as follows, https://archive.ics.uci.edu/dataset/336/chronic+kidney+disease. We chose this dataset because it provides a comprehensive collection of medical data that is crucial for the analysis and detection of Chronic Kidney Disease (CKD). The dataset includes various attributes related, which are essential for building predictive models and understanding the progression of CKD.

By leveraging this dataset, we aim to develop accurate predictive models that can help in the diagnosis of CKD, potentially improving patient outcomes and informing clinical decisions. The detailed medical attributes and structured format of the dataset make it an ideal choice for this analysis. Below are the specifics of the dataset:

Dataset Condition				
Columns	**Data Type Before Cleaning**	**Description**	**Data Type After Cleaning**	**Total Number of Missing Values**
age	Object	The age of the individual in years	Numerical	9
bp	Object	Blood Pressure in mm/Hg	Categorical	14
sg	Object	Specific Gravity of urine	Categorical	49
al	Object	Presence of albumin in urine	Categorical	48
su	Object	Presence of sugar in urine	Categorical	51
rbc	Object	Presence of red blood cells in urine	Categorical	154
pc	Object	Presence of pus cells in urine	Categorical	67
pcc	Object	Presence of pus cells clumps in urine	Categorical	6

ba	Object	Presence of bacteria in urine	Categorical	6
bgr	Object	Random blood glucose level in mg/dl	Numerical	46
bu	Object	Blood urea level in mg/dl	Numerical	21
sc	Object	Serum creatinine level in mg/dl	Numerical	19
sod	Object	Sodium level in mEq/L	Numerical	89
pot	Object	Potassium level in mEq/L	Numerical	90
hemo	Object	Hemoglobin level in gms	Numerical	54
pcv	Object	Volume occupied by packed red blood cells in the blood	Numerical	72
wbcc	Object	White blood cell count in cells/cmm	Numerical	107
rbcc	Object	Red blood cell count in millions/cmm	Numerical	132
htn	Object	Presence of hypertension	Categorical	4
dm	Object	Presence of diabetes mellitus	Categorical	5
cad	Object	Presence of coronary artery disease	Categorical	4
appet	Object	Patient’s appetite	Categorical	3
pe	Object	Presence of pedal edema	Categorical	3

ane	Object	Presence of anemia	Categorical	3
class	Object	Class label indicating presence of Chronic Kidney Disease (CKD) or not	Categorical	2

Table 1 Dataset Condition Table

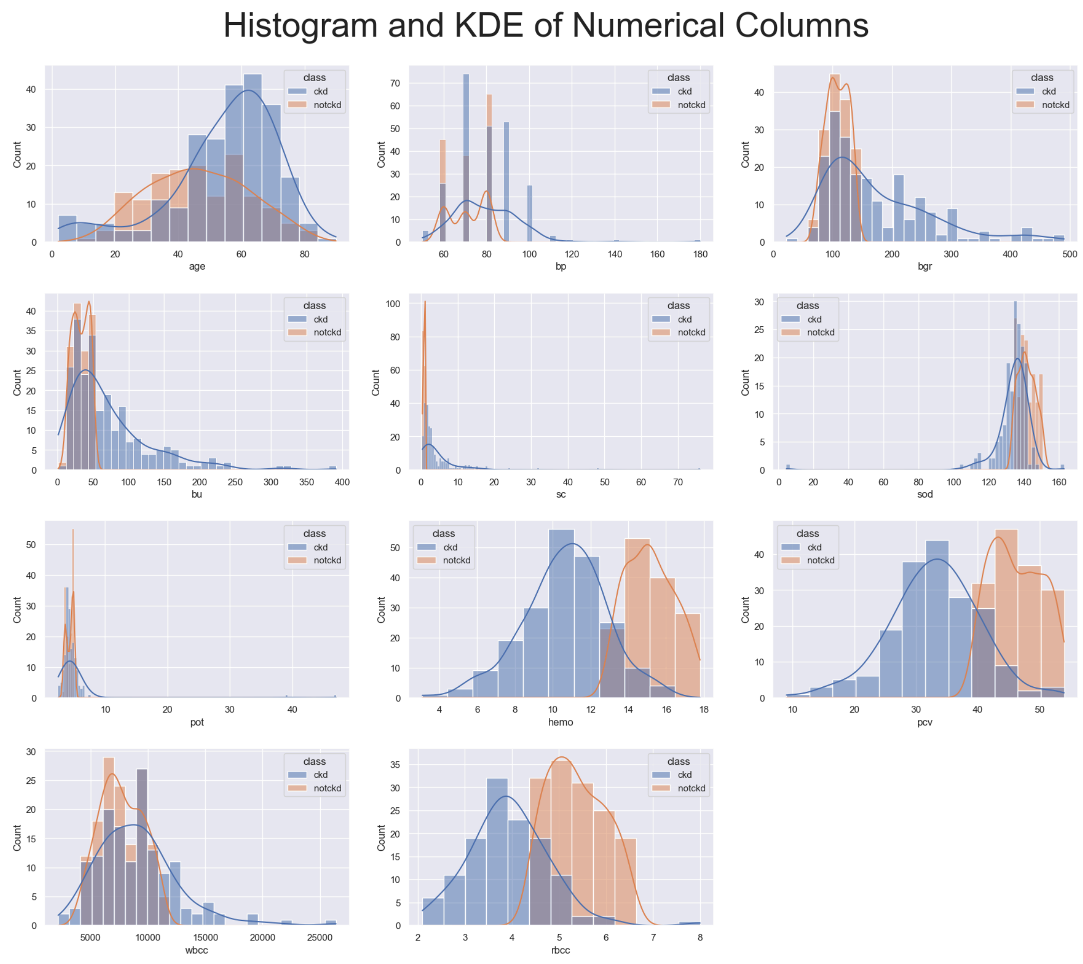
From Table 1 we can see that this dataset consists of 26 columns and 402 rows of data. The target variable is the class column. The data is very dirty with many missing values, one duplicated row, and different format values. The data cleaning is done by replacing some words in the dataset so they all have the same format, then forcing the numerical values to be the numerical data type. More detail will be explained in the following unit.

Unit II - Data Preparation: EDA and Pre-processing

1. **EDA (Exploratory Data Analysis)**

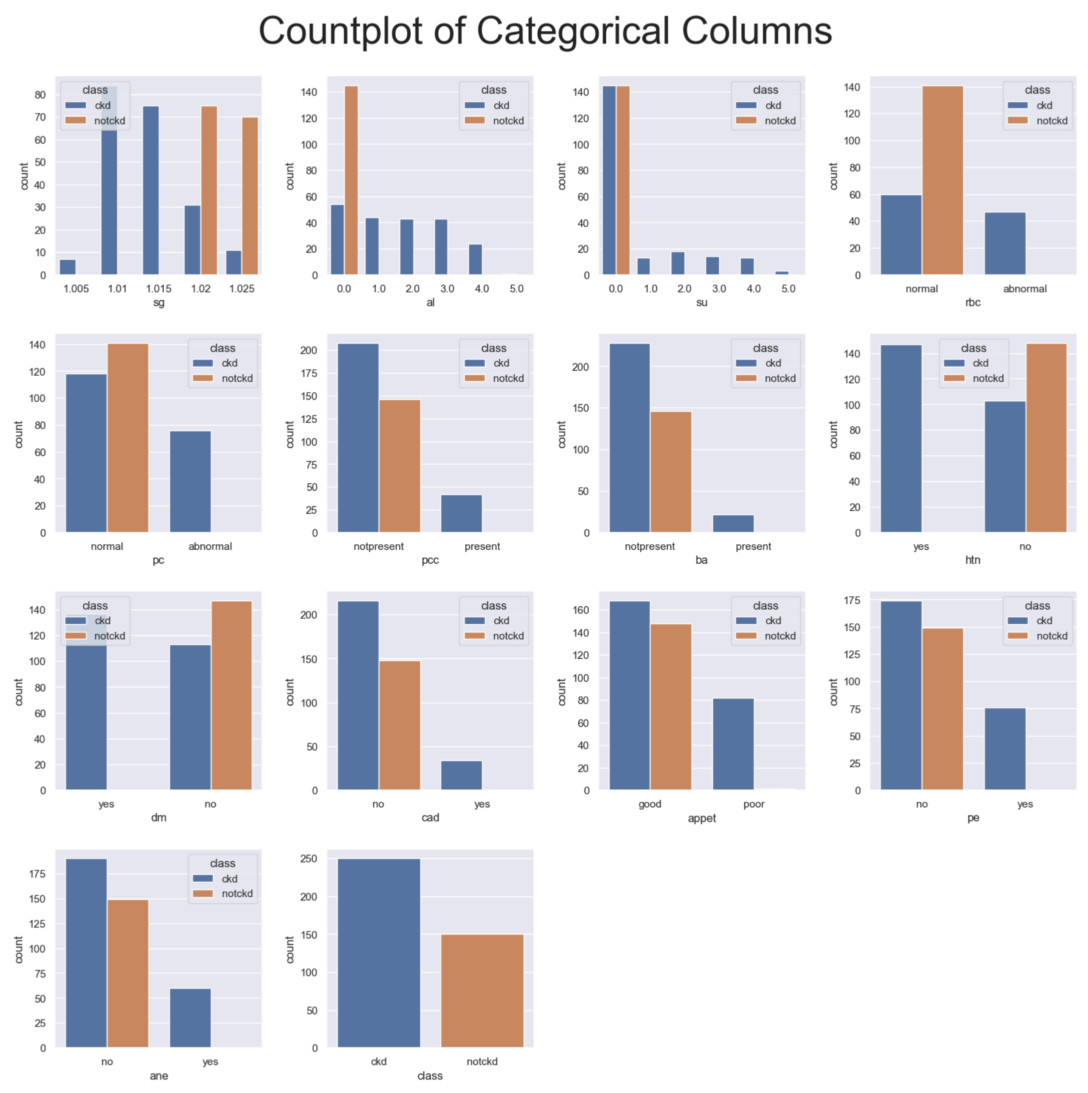
This section focuses on Exploratory Data Analysis (EDA), a crucial step in our project. EDA employs statistics and visualization techniques to uncover patterns, trends, and central tendencies within our dataset. In this unit, bar graphs, histograms, kernel density estimation, and heatmap will be used to effectively summarize and analyze the various variables. This visual exploration allows us to gain a deeper understanding of the data's distribution and potential relationships between variables before proceeding with further analysis.

The dataset visualizations offer a comprehensive insight into the distribution of various variables among individuals with and without chronic kidney disease, ensuring a balanced representation across both cohorts. Divided into two distinct categories of numerical and categorical variables, Graph 2.1.1 sheds light specifically on the numerical attributes, encompassing a range of factors such as age, blood pressure, blood glucose random, blood urea, serum creatinine, sodium, potassium, hemoglobin, packed cell volume, white blood cell count, and red blood cell count. In this dataset, individuals diagnosed with chronic kidney disease typically demonstrate distinctive patterns, characterized by age brackets spanning from 40 to 80, with a notable zenith around the age of 60. Their physiological parameters also exhibit recognizable trends, with blood pressure levels averaging between 60 to 90, blood glucose random concentrations ranging from 100 to 300, and other biochemical markers such as blood urea, serum creatinine, sodium, potassium, hemoglobin, packed cell volume, white blood cell count, and red blood cell count manifesting within specific prescribed ranges delineated by the study parameters.



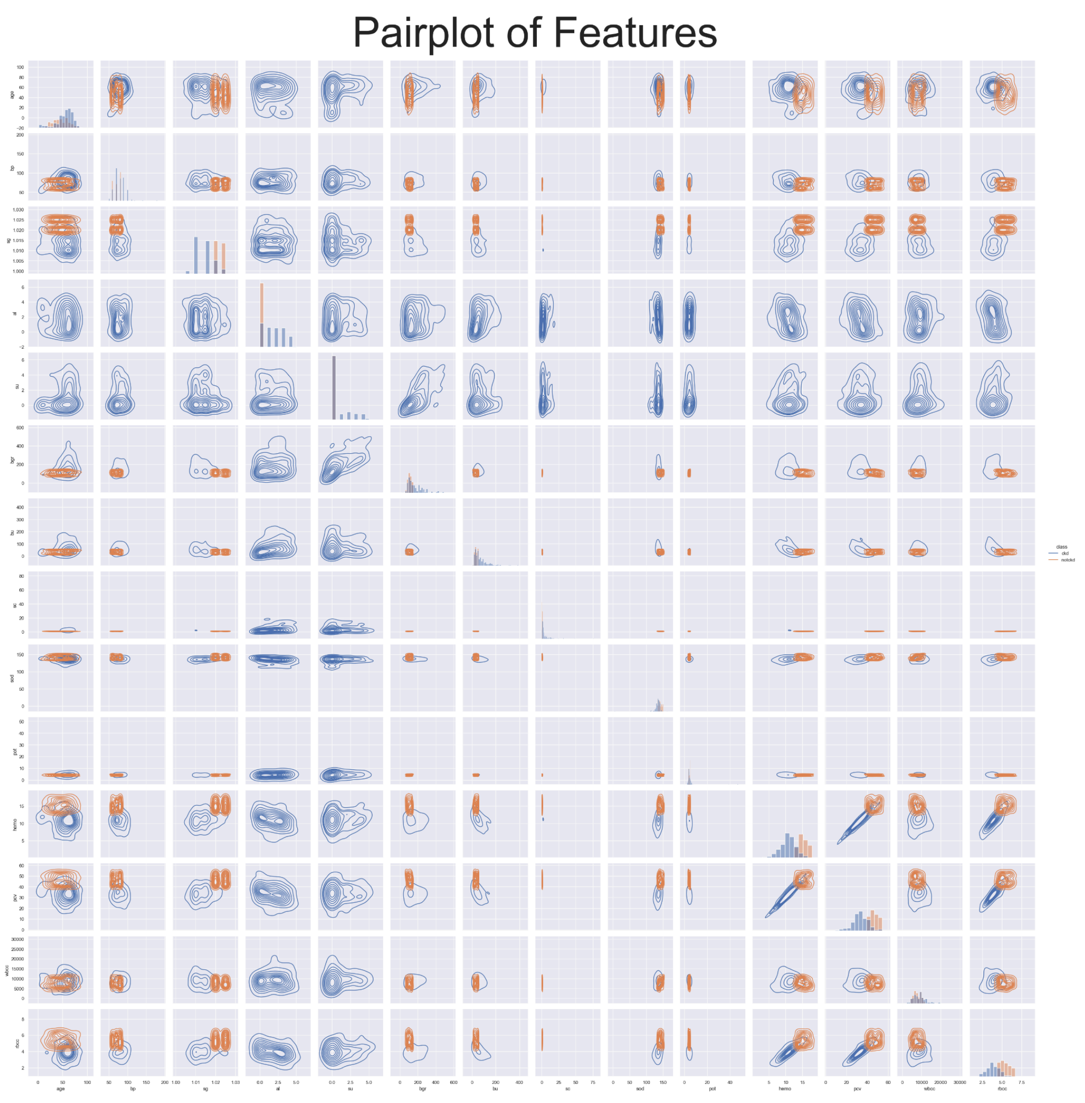
Graph 2.1.1. Histogram and KDE (Kernel Density Estimate) Plot

The categorical variables, including Specific Gravity, Albumin, Sugar, Red Blood Cells, Pus Cell, Pus Cell Clumps, Bacteria, Hypertension, Diabetes Mellitus, Coronary Artery Disease, Appetite, Pedal Edema, and Anemia, are visually represented in Graph 2.1.2. This dataset comprises 250 instances of chronic kidney disease and 150 instances without chronic kidney disease. Analysis of this dataset reveals notable patterns in which chronic kidney disease often presents with a specific gravity exceeding 1.02, whereas individuals without this condition typically show a lower specific gravity. In the absence of chronic kidney disease, albumin and sugar levels are typically undetectable. Furthermore, those unaffected by chronic kidney disease typically exhibit normal levels of red blood cells and pus cells without any clumping. Bacterial presence, hypertension, diabetes mellitus, and coronary artery disease are also commonly absent in individuals without chronic kidney disease. Additionally, individuals not afflicted by chronic kidney disease typically demonstrate a healthy appetite, lack pedal edema, and do not suffer from anemia.



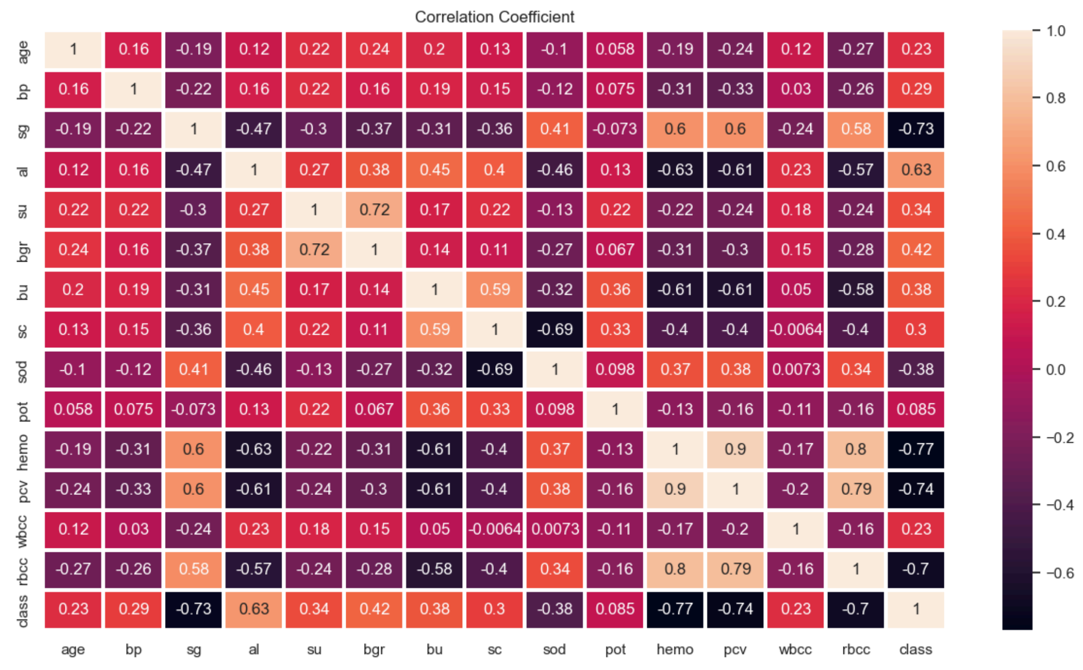
Graph 2.1.2. Count Plot

In addition, we created pair plots among the features in respect of the class to explore potential multicollinearity, where the addition of independent variables can form relationships among themselves. This means that they not only have correlations with the dependent variable but can also exhibit correlations with each other. These can be seen in Graph 2.1.3.



Graph 2.1.3. Pair Plot

Following the completion of the pair plot analysis, we proceed to examine our dataset through a correlation matrix. This matrix offers a depiction of the correlation coefficients among variables, with values spanning from -1 to 1. A correlation close to 1 signifies a strong positive correlation, while proximity to -1 indicates a strong negative correlation. Conversely, values nearing 0 suggest minimal linear association. This tool helps find patterns and connections in the data.



	age	bp	sg	al	su	bgr	bu	sc	sod	pot	hemo	pcv	wbcc	rbcc	class
age	1	0.16	-0.19	0.12	0.22	0.24	0.2	0.13	-0.1	0.058	-0.19	-0.24	0.12	-0.27	0.23
bp	0.16	1	-0.22	0.16	0.22	0.16	0.19	0.15	-0.12	0.075	-0.31	-0.33	0.03	-0.26	0.29
sg	-0.19	-0.22	1	-0.47	-0.3	-0.37	-0.31	-0.36	0.41	-0.073	0.6	0.6	-0.24	0.58	-0.73
al	0.12	0.16	-0.47	1	0.27	0.38	0.45	0.4	-0.46	0.13	-0.63	-0.61	0.23	-0.57	0.63
su	0.22	0.22	-0.3	0.27	1	0.72	0.17	0.22	-0.13	0.22	-0.22	-0.24	0.18	-0.24	0.34
bgr	0.24	0.16	-0.37	0.38	0.72	1	0.14	0.11	-0.27	0.067	-0.31	-0.3	0.15	-0.28	0.42
bu	0.2	0.19	-0.31	0.45	0.17	0.14	1	0.59	-0.32	0.36	-0.61	-0.61	0.05	-0.58	0.38
sc	0.13	0.15	-0.36	0.4	0.22	0.11	0.59	1	-0.69	0.33	-0.4	-0.4	-0.0064	-0.4	0.3
sod	-0.1	-0.12	0.41	-0.46	-0.13	-0.27	-0.32	-0.69	1	0.098	0.37	0.38	0.0073	0.34	-0.38
pot	0.058	0.075	-0.073	0.13	0.22	0.067	0.36	0.33	0.098	1	-0.13	-0.16	-0.11	-0.16	0.085
hemo	-0.19	-0.31	0.6	-0.63	-0.22	-0.31	-0.61	-0.4	0.37	-0.13	1	0.9	-0.17	0.8	-0.77
pcv	-0.24	-0.33	0.6	-0.61	-0.24	-0.3	-0.61	-0.4	0.38	-0.16	0.9	1	-0.2	0.79	-0.74
wbcc	0.12	0.03	-0.24	0.23	0.18	0.15	0.05	-0.0064	0.0073	-0.11	-0.17	-0.2	1	-0.16	0.23
rbcc	-0.27	-0.26	0.58	-0.57	-0.24	-0.28	-0.58	-0.4	0.34	-0.16	0.8	0.79	-0.16	1	-0.7
class	0.23	0.29	-0.73	0.63	0.34	0.42	0.38	0.3	-0.38	0.085	-0.77	-0.74	0.23	-0.7	1

Graph 2.1.4. Correlation Matrix

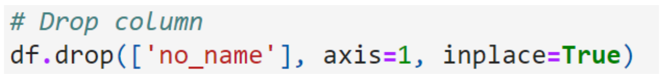
In Graph 2.1.4, we can see the correlation between various features in the Chronic Kidney Disease dataset. The first thing to note is that we want the correlation between features to be low. This is important because if two features have a very high correlation, they may present redundant information and not provide significant additional value for our predictive model. For example, the features **hemo** and **pcv** have a high correlation (0.9), indicating that they might convey similar information.

However, more importantly, we want to look at the correlation between features and the class label. We want a high correlation between features and the class label because this indicates that the feature has a strong influence on the final outcome or classification of Chronic Kidney Disease. From the heatmap, we can see that the features **sg** (specific gravity), **hemo** (hemoglobin), **pcv** (packed cell volume), **rbcc** (red blood cell count), and **al** (albumin) have a relatively high correlation with the class label, which are -0.73, -0.77, -0.74, -0.7, and 0.63 respectively.

This means that these features have a significant impact on the final classification outcome of CKD. Therefore, when building the predictive model, these features will be very important and should be carefully considered in the feature selection process to ensure our model performs optimally.

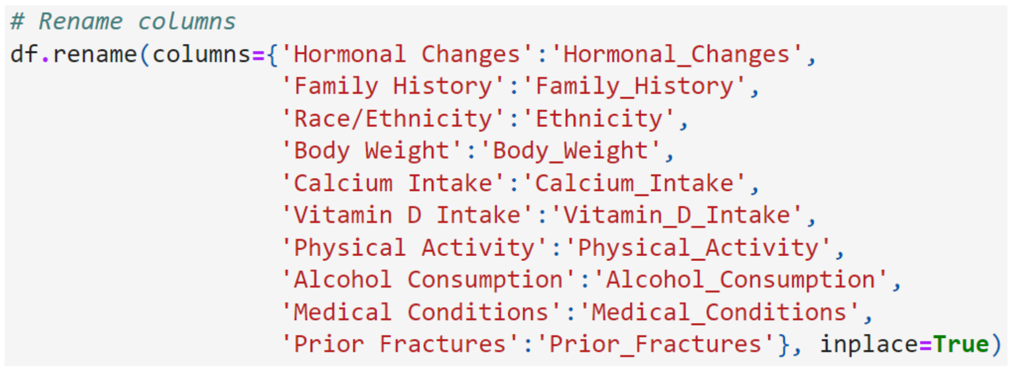
2. Pre-Processing

Pre-processing data refers to a process in data analysis that has already been cleaned from the dataset prior to the pre-processing phase to have a better visualization throughout the EDA process. Fig. 3.2.1 to 3.2.5 shows how we clean our data. Our team initiates the data cleaning phase by eliminating unused columns and renaming them (Fig. 3.2.1 and Fig. 3.2.2). Subsequently, we handle the format of data so it's consistent throughout (Fig. 3.2.3). Additionally, duplicated rows are removed to refine the dataset and enhance its quality (Fig. 3.2.4). Further optimization is achieved by discarding rows where the class and/or all values are null (Fig. 3.2.5).



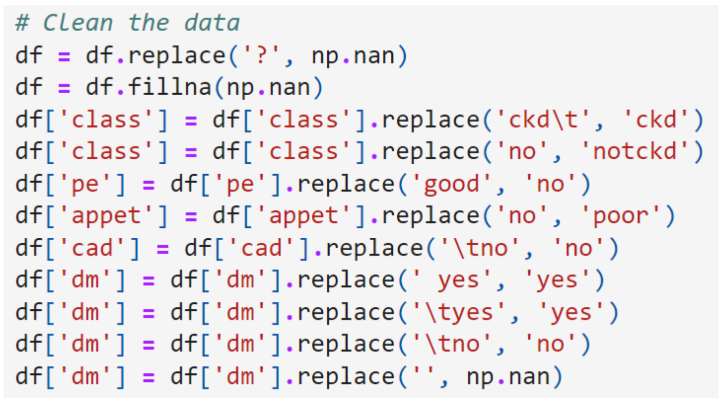
```python
# Drop column
df.drop(['no_name'], axis=1, inplace=True)
```

Fig. 3.2.1. Drop Columns



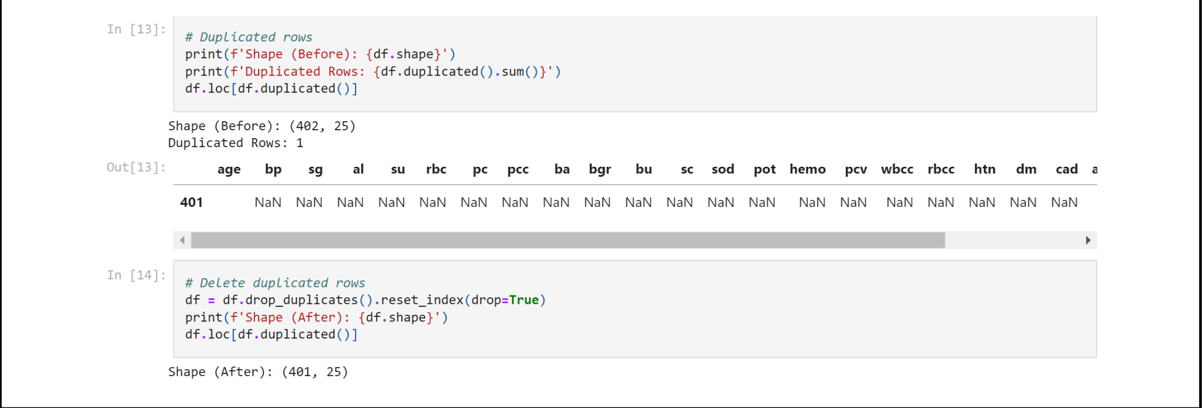
```python
# Rename columns
df.rename(columns={'Hormonal Changes':'Hormonal_Changes',
                   'Family History':'Family_History',
                   'Race/Ethnicity':'Ethnicity',
                   'Body Weight':'Body_Weight',
                   'Calcium Intake':'Calcium_Intake',
                   'Vitamin D Intake':'Vitamin_D_Intake',
                   'Physical Activity':'Physical_Activity',
                   'Alcohol Consumption':'Alcohol_Consumption',
                   'Medical Conditions':'Medical_Conditions',
                   'Prior Fractures':'Prior_Fractures'}, inplace=True)
```

Fig. 3.2.2. Rename Columns



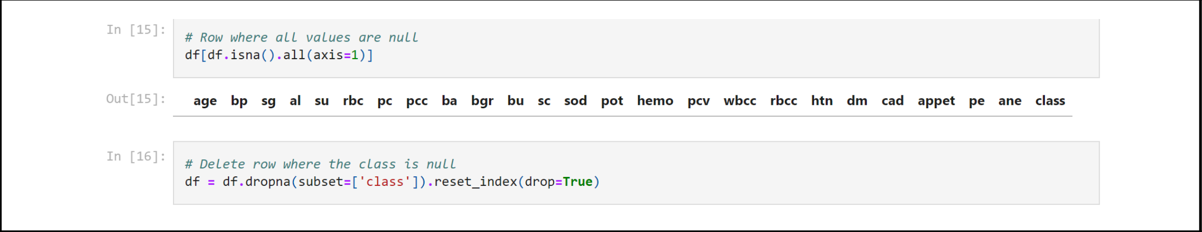
```python
# Clean the data
df = df.replace('?', np.nan)
df = df.fillna(np.nan)
df['class'] = df['class'].replace('ckd\t', 'ckd')
df['class'] = df['class'].replace('no', 'notckd')
df['pe'] = df['pe'].replace('good', 'no')
df['appet'] = df['appet'].replace('no', 'poor')
df['cad'] = df['cad'].replace('\tno', 'no')
df['dm'] = df['dm'].replace(' yes', 'yes')
df['dm'] = df['dm'].replace('\tyes', 'yes')
df['dm'] = df['dm'].replace('\tno', 'no')
df['dm'] = df['dm'].replace('', np.nan)
```

Fig. 3.2.3. Data Cleaning



```
In [13]: # Duplicated rows
         print(f'Shape (Before): {df.shape}')
         print(f'Duplicated Rows: {df.duplicated().sum()}')
         df.loc[df.duplicated()]

         Shape (Before): (402, 25)
         Duplicated Rows: 1

Out[13]:      age  bp   sg   al   su   rbc  pc   pcc  ba   bgr  bu   sc   sod  pot  hemo pcv  wbcc rbcc htn  dm   cad  a
         401       NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN

In [14]: # Delete duplicated rows
         df = df.drop_duplicates().reset_index(drop=True)
         print(f'Shape (After): {df.shape}')
         df.loc[df.duplicated()]

         Shape (After): (401, 25)
```

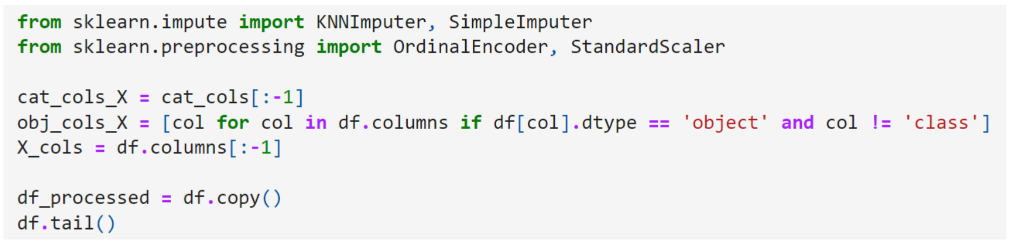
Fig. 3.2.4. Delete Duplicate Rows



```
In [15]: # Row where all values are null
         df[df.isna().all(axis=1)]

Out[15]:   age  bp  sg  al  su  rbc  pc  pcc  ba  bgr  bu  sc  sod  pot  hemo  pcv  wbcc  rbcc  htn  dm  cad  appet  pe  ane  class

In [16]: # Delete row where the class is null
         df = df.dropna(subset=['class']).reset_index(drop=True)
```

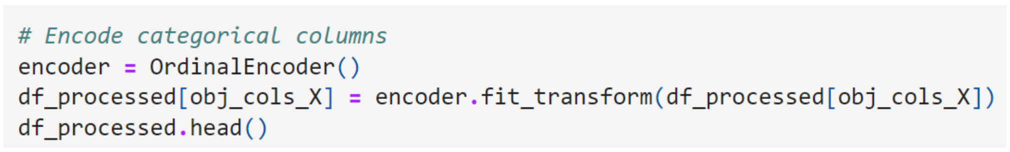
Fig. 3.2.5. Delete Rows Where Class is Null

We also found that this dataset has many missing values. However, it is not appropriate to ignore all the missing values (Senan, E.M. et al., 2021). Hence, we handle missing values in categorical columns by employing the 'most frequent' strategy with a simple imputer (Fig. 3.2.6. & Fig. 3.2.7), which replaces missing values with the most frequent value found in each column. We choose to use an imputer instead of deleting them because the dataset is already small. It allows us to make somewhat accurate predictions even when some data points are incomplete. By filling in the missing values with estimated ones, the imputer ensures that no valuable data is discarded, which helps in maintaining the integrity and performance of our predictive model. This way, we can still utilize the dataset effectively and generate reliable predictions. After all the categorical columns are imputed, they are then encoded using ordinal encoders to transform the categorical data (Fig. 3.2.8). This transformation assigns numerical values to categorical variables based on the order of appearance in the dataset. After all the categorical data is handled, we use KNN imputer to handle the missing value on the numerical data (Fig. 3.2.9). The KNN imputer considers the similarity between instances, making it a robust choice for the imputation of missing numerical values in CKD datasets (Ekanayake, I.U. & Herath, D., 2020). It uses the k-nearest neighbors algorithm, wherein missing values are replaced with estimated values computed using the k-nearest neighbors approach.



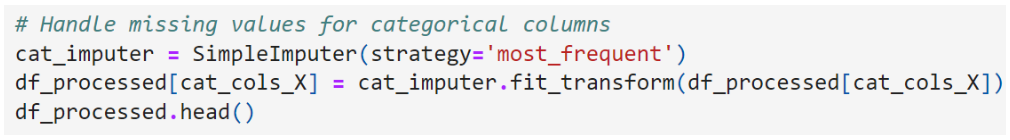
```python
from sklearn.impute import KNNImputer, SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler

cat_cols_X = cat_cols[:-1]
obj_cols_X = [col for col in df.columns if df[col].dtype == 'object' and col != 'class']
X_cols = df.columns[:-1]

df_processed = df.copy()
df.tail()
```

Fig. 3.2.6. Features Initialization



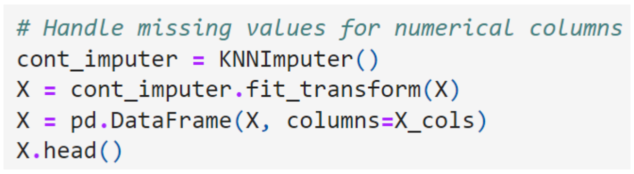
```python
# Encode categorical columns
encoder = OrdinalEncoder()
df_processed[obj_cols_X] = encoder.fit_transform(df_processed[obj_cols_X])
df_processed.head()
```

Fig. 3.2.7. Categorical Column Encoding



```python
# Handle missing values for categorical columns
cat_imputer = SimpleImputer(strategy='most_frequent')
df_processed[cat_cols_X] = cat_imputer.fit_transform(df_processed[cat_cols_X])
df_processed.head()
```

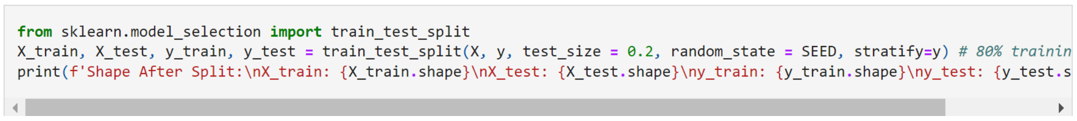
Fig. 3.2.8. Categorical Column Missing Value Handling



```python
# Handle missing values for numerical columns
cont_imputer = KNNImputer()
X = cont_imputer.fit_transform(X)
X = pd.DataFrame(X, columns=X_cols)
X.head()
```

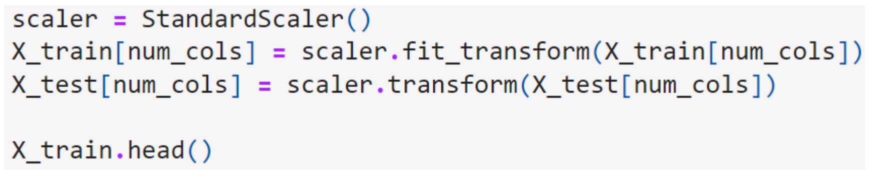
Fig. 3.2.9. Numerical Column Missing Value Handling

We split the dataset into an 80% training set and a 20% test set for accurate model evaluation (Fig. 3.2.10). The training set will be splitted again using 4-fold cross-validation in the unit below (Unit III), making it a 60:20:20 ratio between training, validation, and testing. Then, we use standard scaling fitted to the training set to normalize the numerical features, making sure they're centered around 0 with a standard deviation of 1 (Fig. 3.2.11). Doing this after splitting prevents data leakage, keeping the scaling process fair for both training and test sets. After that, we use Kernel Principal Component Analysis (KernelPCA) with the radial basis function so it can capture nonlinear relationships to reduce the number of features/dimensions down to 8 (Fig. 3.2.12) (Wibowo, M.R. & Palupi. I, 2023). This helps simplify our data while still retaining important information, prevents overfitting, and reduces redundancy that we found on the correlation coefficient before. Lastly, we save trained models and other relevant information using pickle and json (Fig. 3.2.13).



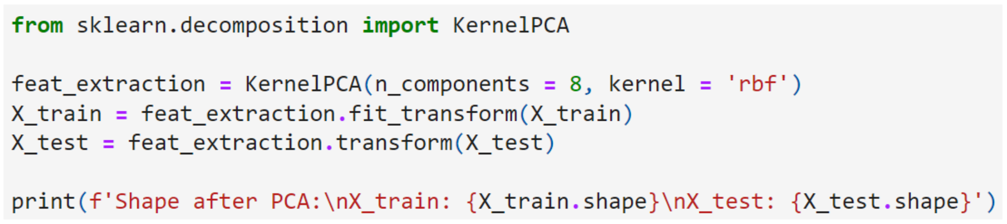
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = SEED, stratify=y) # 80% trainin
print(f'Shape After Split:\nX_train: {X_train.shape}\nX_test: {X_test.shape}\ny_train: {y_train.shape}\ny_test: {y_test.s
```

Fig. 3.2.10. Train and Test Dataset Splitting



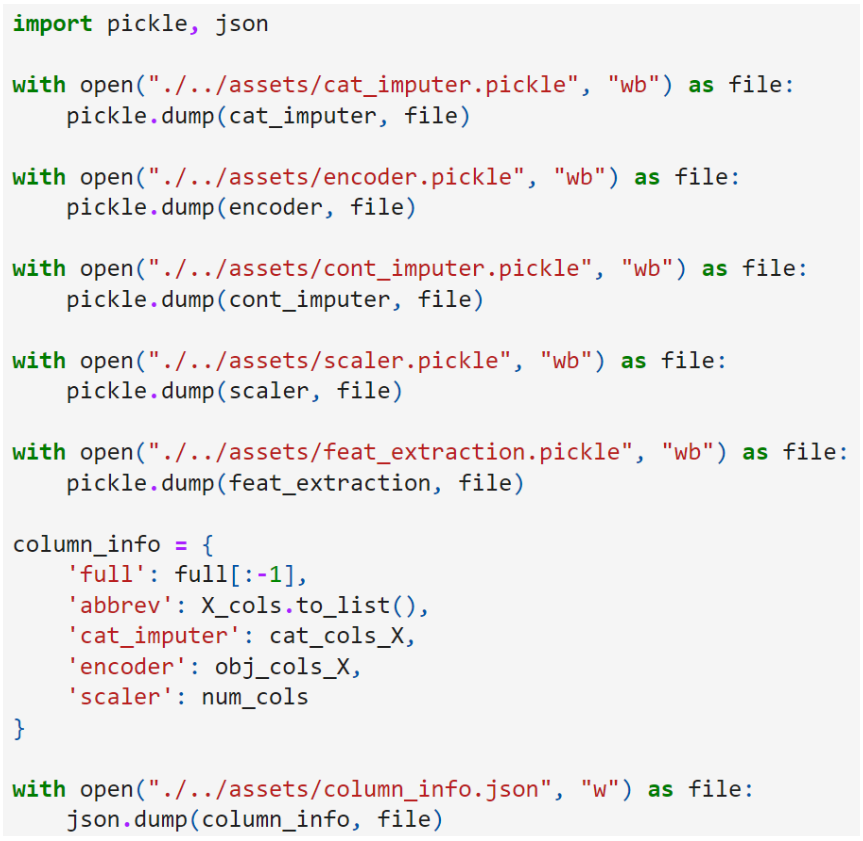
```
scaler = StandardScaler()
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])

X_train.head()
```

Fig. 3.2.11. Normalization



```
from sklearn.decomposition import KernelPCA

feat_extraction = KernelPCA(n_components = 8, kernel = 'rbf')
X_train = feat_extraction.fit_transform(X_train)
X_test = feat_extraction.transform(X_test)

print(f'Shape after PCA:\nX_train: {X_train.shape}\nX_test: {X_test.shape}')
```

Fig. 3.2.12. Dimension Reduction



```python
import pickle, json

with open("./../assets/cat_imputer.pickle", "wb") as file:
    pickle.dump(cat_imputer, file)

with open("./../assets/encoder.pickle", "wb") as file:
    pickle.dump(encoder, file)

with open("./../assets/cont_imputer.pickle", "wb") as file:
    pickle.dump(cont_imputer, file)

with open("./../assets/scaler.pickle", "wb") as file:
    pickle.dump(scaler, file)

with open("./../assets/feat_extraction.pickle", "wb") as file:
    pickle.dump(feat_extraction, file)

column_info = {
    'full': full[:-1],
    'abbrev': X_cols.to_list(),
    'cat_imputer': cat_cols_X,
    'encoder': obj_cols_X,
    'scaler': num_cols
}

with open("./../assets/column_info.json", "w") as file:
    json.dump(column_info, file)
```

Fig. 3.2.13. Model Saving

Unit III - Modeling and Evaluation

1. Modeling

Modeling in machine learning involves creating and training a predictive model using algorithms and data. This process includes selecting the right algorithm, preparing the data, and training the model to recognize patterns between input features and target variables. The trained model can then be used to make predictions or classify new data. The goal is to develop a model that accurately captures patterns in the data and can make reliable predictions on new data, aiding in decision-making and insights.

In this project, we encapsulate a comprehensive approach to model selection and hyperparameter tuning in machine learning. We start by defining a dictionary named 'model_params' where each key corresponds to a specific classification algorithm, including Logistic Regression, K-Nearest Neighbors, Support Vector Classifier, Gaussian Naive Bayes, Decision Tree Classifier, Random Forest Classifier, and XGBoost Classifier. For each algorithm, there is a nested dictionary containing the model object and a set of hyperparameters to be optimized through grid search cross-validation. By systematically exploring various combinations of hyperparameters, this approach aims to identify the optimal configuration for each model, thereby enhancing predictive accuracy and robustness across diverse datasets.

We use GridSearchCV to iteratively optimize the model's performance to a specified evaluation metric. It systematically explores predefined hyperparameter grids for various classification algorithms. Specifying the model, hyperparameter grid, scoring metric (F1 score), and 4-fold cross-validation is done for each algorithm. 4-fold is used so the validation set is the same size as the test set, to provide a more representative final evaluation. We selected the F1 score as the main evaluation metrics, because the class distribution is a little bit imbalanced. We aim to achieve a balance between precision and recall, which can be important for diagnosing disease in a medical environment. The process records the best parameters and their corresponding F1 scores for each model, along with the standard deviation of the best score. These results are then organized into a DataFrame for thorough analysis and comparison of model performances. Next, the models will be sorted in descending order by the 'best_score' column obtained from grid search results. Eventually, XGBoost was identified as the best model.

model	best_param	best_score	std_of_best
lgr	{'C': 0.75, 'max_iter': 100}	0.974488	0.005108
knn	{'metric': 'manhattan', 'n_neighbors': 13, 'we…}	0.968798	0.016811
svc	{'C': 1000, 'kernel': 'linear'}	0.982321	0.004494
gnb	{'var_smoothing': 1e-09}	0.977421	0.008304
dt	{'criterion': 'entropy', 'splitter': 'best'}	0.972505	0.014209
rf	{'criterion': 'entropy', 'max_depth': 6, 'max_…}	0.984947	0.011118
xgb	{'learning_rate': 0.1, 'max_depth': 3, 'n_esti…}	0.984998	0.008488

Table 3 Metrics Results

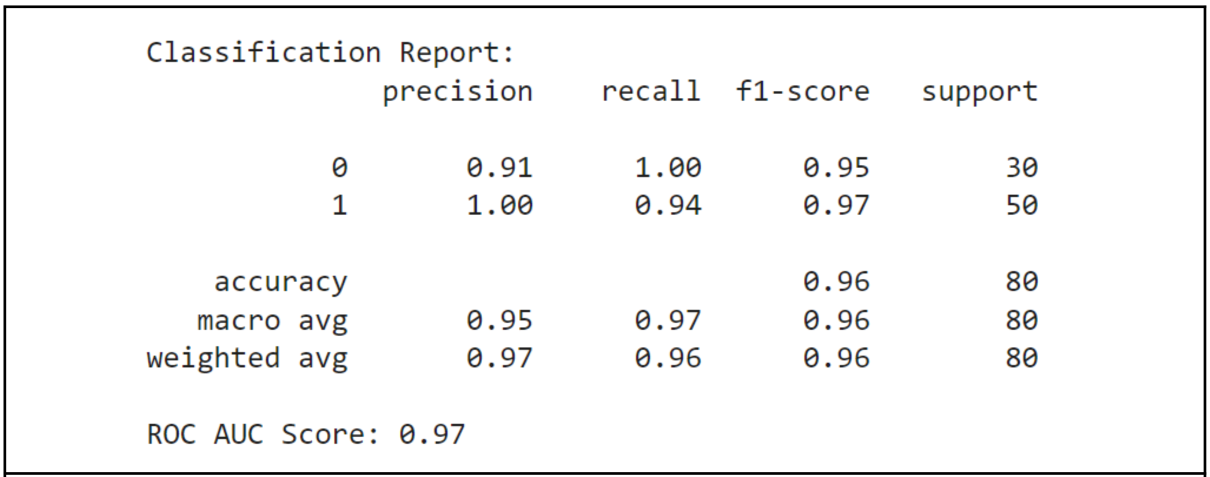
The table presents a summary of the performance of various machine learning models after hyperparameter tuning. The models included are Logistic Regression (lgr), K-Nearest Neighbors (knn), Support Vector Classifier (svc), Gaussian Naive Bayes (gnb), Decision Tree (dt), Random Forest (rf), and XGBoost (xgb). For each model, the best hyperparameters found during the tuning process are listed in the "best_param" column. For instance, the Logistic Regression model was optimized with a regularization parameter set to 0.75 and a maximum number of iterations set to 100. The K-Nearest Neighbors model used the Manhattan distance metric, 13 neighbors, and weighted voting, and so on.

The "best_score" column displays the highest performance metric achieved by each model using the grid search tuned hyperparameters. The “std_of_best” column shows the standard deviation of the best F1 score, providing insight into how our model performs across different validation. A high score and low standard deviation indicate that the model has low bias and low variance, which are the qualities that we desire. We can conclude that all models that we tested have low bias and variance. This approach ensures a finely-tuned model selection process, which is essential for tasks such as disease diagnosis where predictive accuracy and reliability are extremely important.

2. Evaluation

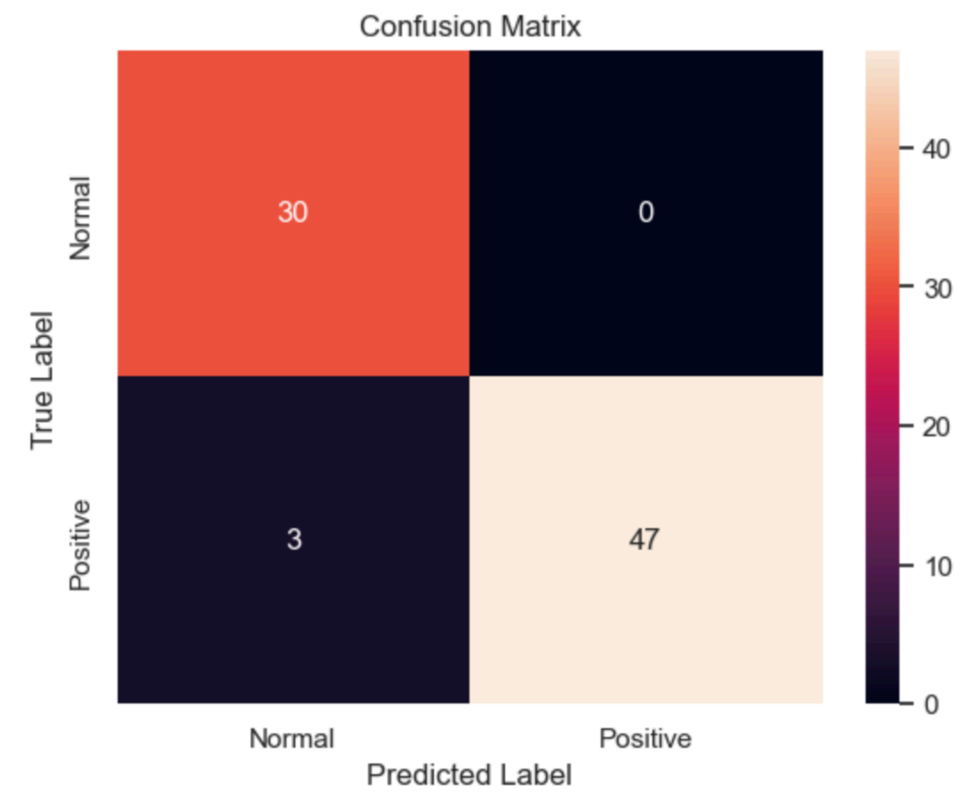
With an impressive best score of 0.984998, the XGBoost model was selected as the best performance among other models. With a standard deviation of 0.008488, this model was the fourth lowest with a close margin, after Support Vector Classifier, Logistic Regression, and Gaussian Naive Bayes which has standard deviation of 0.004494, 0.005108, and 0.008304 respectively.

After using the recommended models and parameters from the GridSearch algorithm, we obtained the following results. In Graph 3.2.1, the classification report for the CKD prediction model using unseen data (test set), is illustrated. The model achieved an accuracy of 0.96. The overall (macro and weighted average) f1-score is 0.96, weighted average precision of 0.97, and the weighted average recall is 0.96. Mostly, the weighted average is used due to the imbalance class distribution. Despite that, all model performances are excellent, with a ROC-AUC score of 0.97, indicating excellent performance in distinguishing between classes.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      1.00      0.95        30
           1       1.00      0.94      0.97        50

    accuracy                           0.96        80
   macro avg       0.95      0.97      0.96        80
weighted avg       0.97      0.96      0.96        80

ROC AUC Score: 0.97
```

Graph 3.2.1. Classification Report XGBoost After Hyperparameter Tuning



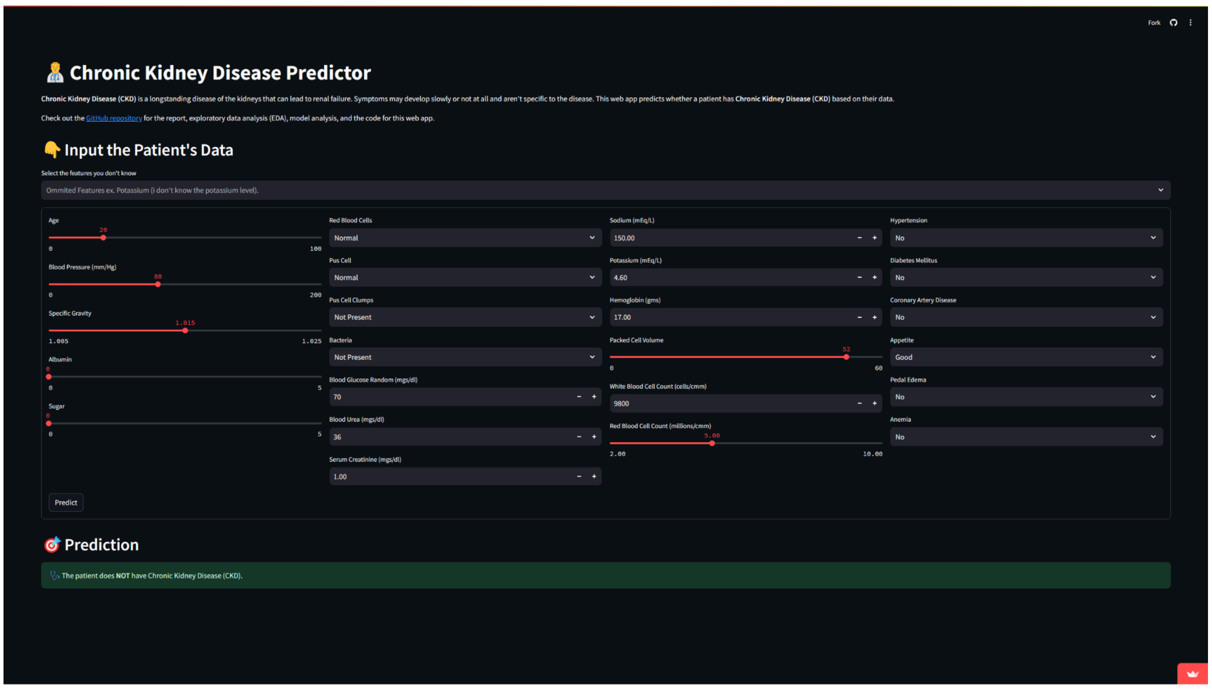
Graph 3.2.2. Confusion Matrix XGBoost After Hyperparameter Tuning

Graph 3.2.2 displays the visualization of the confusion matrix obtained from the CKD classification model. The matrix shows that the model correctly classified 30 normal cases and 47 positive cases. There were no normal cases misclassified as positive, and only 3 positive cases were misclassified as normal. This result indicates the model's high accuracy and precision in its prediction.

Unit IV - Deployment and Reporting

After completing all the coding process, we built the web app using Streamlit, which is a python framework designed for creating interactive web apps that are easy to access and usable. Streamlit made building web apps more straightforward by using the functionality of widgets, allowing for user input with minimal lines of codes.

One of the notable features our web app has is the ability to omit unknown data. It utilizes all the assets from the analysis pre-processing step to impute, process, and predict all the user-inputted data. Session state is used to store the omitted features. It uses a session state to save the omitted features because the way Streamlit renders contents is different compared to the usual website frameworks. It always uses top to bottom order to render and run the code. Session state is one of the ways we can use it to save some value. After getting the user input, we load all the save assets in the analysis, process the data, and display the prediction. Then, we deploy the report, analysis, and web app to GitHub and use the Streamlit Community Cloud to ensure thorough documentation and accessibility of the web app on the internet.



Chronic Kidney Disease Predictor

Chronic Kidney Disease (CKD) is a longstanding disease of the kidneys that can lead to renal failure. Symptoms may develop slowly or not at all and aren't specific to the disease. This web app predicts whether a patient has **Chronic Kidney Disease (CKD)** based on their data.

Check out the GitHub repository for the report, exploratory data analysis (EDA), model analysis, and the code for this web app.

Input the Patient's Data

Select the features you don't know

Ommited Features ex. Potassium (i don't know the potassium level).

Age	20 (0–100)
Blood Pressure (mm/Hg)	80 (0–200)
Specific Gravity	1.015 (1.005–1.025)
Albumin	0 (0–5)
Sugar	0 (0–5)
Red Blood Cells	Normal
Pus Cell	Normal
Pus Cell Clumps	Not Present
Bacteria	Not Present
Blood Glucose Random (mgs/dl)	70
Blood Urea (mgs/dl)	36
Serum Creatinine (mgs/dl)	1.00
Sodium (mEq/L)	150.00
Potassium (mEq/L)	4.60
Hemoglobin (gms)	17.00
Packed Cell Volume	52 (0–60)
White Blood Cell Count (cells/cmm)	9800
Red Blood Cell Count (millions/cmm)	5.00 (2.00–10.00)
Hypertension	No
Diabetes Mellitus	No
Coronary Artery Disease	No
Appetite	Good
Pedal Edema	No
Anemia	No

Prediction

The patient does **NOT** have Chronic Kidney Disease (CKD).

Then, we made the demo video and published it on YouTube. You can explore our project, interactive web app, and all the assets used through this report in the links provided below:

Web App	https://ckdpred.streamlit.app/
Report	https://github.com/vanssnn/chronic-kidney-disease-predictor/blob/main/report/Machine_Learning_Final_Project_Report.pdf
Demo Video	https://www.youtube.com/watch?v=Kd6EHU4CBpo&ab_channel=ivanssnn
Analysis	https://github.com/vanssnn/chronic-kidney-disease-predictor/blob/main/analysis/analysis.ipynb
Web App Source Code	https://github.com/vanssnn/chronic-kidney-disease-predictor/blob/main/app.py
Github Repository	https://github.com/vanssnn/chronic-kidney-disease-predictor *Note: the analysis (EDA, Model Selection, Model Builder, Evaluation), report, source code, and all the assets are in this repository.

Coding Process Overview

Our code illustrates the process of building classification models for chronic kidney disease from a specific dataset that we obtained from the UCI Machine Learning Repository. Our coding process can be broken down into several steps:

1. We begin by creating the environment and installing libraries using pip install which includes numpy, matplotlib, pandas, scikit-learn, xgboost, and seaborn. These libraries are commonly used for data analysis, visualization, and machine learning modeling.
2. After the library installation step, we then import the libraries along with other necessary modules.
3. Consequently, we load the csv dataset using pandas.
4. Next, we perform **Exploratory Data Analysis (EDA)** by analyzing data and distribution of target variables by visualizing it using histogram, kernel density estimation, count

plots and pair plots. We then analyze the possibility of multicollinear by using correlation matrix.

5. We handle missing categorical attributes by using **Simple Imputer** with strategy 'most_frequent' to fill the missing entries with the most common value from each categorical attribute.
6. Next, we convert the categorical variables using **Ordinal Encoder**. This involves assigning numerical values to categories based on their order or importance, making them suitable for machine learning algorithms that require numerical input.
7. We also use **KNN imputers** to handle missing numerical attributes. This technique estimates missing values based on the values of the nearest neighbors.
8. Before building our machine learning model, we prepare our data by splitting it into two sets, 80% for features X and 20% for target y.
9. We then normalize the numerical features fitted to the training set to bring them onto a similar scale using **Standard Scaler**. This step typically involves transforming features so that they have a mean of 0 and a standard deviation of 1, which helps in improving the performance and convergence of many machine learning algorithms.
10. After that, we utilize **KernelPCA** to reduce the dimensions to 8 for better performance and save all the trained models using pickle and json.
11. Then, we did model selection using hyperparameter tuning with grid search and 4-fold cross-validation on different classification models such as **Logistic Regression, K-Nearest Neighbours, Support Vector Machine, Gaussian Naive Bayes, Decision Tree, Random Forest, and XGBoost**. We evaluate each model using key metrics such as best F1 score, standard deviation, and parameters that provide best results during the hyperparameter tuning.
12. From the best model selection, which is **XGBoost**, we evaluate the model performance using the test set and make the classification report with the confusion matrix so we can visualize the model performance. Finally, we create the web app using **Streamlit** and deploy it to the web using **Streamlit Community Cloud**.

This concludes our project. We aim for our work to provide meaningful insights and useful tools for the early detection and better understanding of Chronic Kidney Disease. Thank you!

Bibliography

Abdel-Fattah, M. A., Othman, N. A., & Goher, N. (2022). Predicting chronic kidney disease using hybrid machine learning based on apache spark. *Computational Intelligence and Neuroscience*, *2022*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8890824/

Lv, J. C., & Zhang, L. X. (2019). Prevalence and disease burden of chronic kidney disease. *Renal fibrosis: mechanisms and therapies*, 3-15. https://pubmed.ncbi.nlm.nih.gov/31399958/

Debal, D. A., & Sitote, T. M. (2022). Chronic kidney disease prediction using machine learning techniques. *Journal of Big Data*, *9*(1), 109. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00657-5\

Ekanayake, I. U., & Herath, D. (2020, July). Chronic kidney disease prediction using machine learning methods. In 2020 Moratuwa Engineering Research Conference (MERCon) (pp. 260-265). IEEE. https://ieeexplore.ieee.org/abstract/document/9185249

Ghosh, P., Shamrat, F. J. M., Shultana, S., Afrin, S., Anjum, A. A., & Khan, A. A. (2020, November). Optimization of prediction method of chronic kidney disease using machine learning algorithm. In 2020 15th international joint symposium on artificial intelligence and natural language processing (iSAI-NLP) (pp. 1-6). IEEE. https://ieeexplore.ieee.org/abstract/document/9376787

Kale, Y., Rathkanthiwar, S., Fulzele, P., & Bankar, N. J. (2024). XGBoost Learning for Detection and Forecasting of Chronic Kidney Disease (CKD). International Journal of Intelligent Systems and Applications in Engineering, 12(17s), 137-150. https://ijisae.org/index.php/IJISAE/article/view/4843

Khalid, H., Khan, A., Zahid Khan, M., Mehmood, G., & Shuaib Qureshi, M. (2023). Machine learning hybrid model for the prediction of chronic kidney disease. Computational Intelligence and Neuroscience, 2023(1), 9266889. https://onlinelibrary.wiley.com/doi/full/10.1155/2023/9266889

Iftikhar, H., Khan, M., Khan, Z., Khan, F., Alshanbari, H. M., & Ahmad, Z. (2023). A comparative analysis of machine learning models: a case study in predicting chronic kidney disease. Sustainability, 15(3), 2754. https://www.mdpi.com/2071-1050/15/3/2754

Rashed-Al-Mahfuz, M., Haque, A., Azad, A., Alyami, S. A., Quinn, J. M., & Moni, M. A. (2021). Clinically applicable machine learning approaches to identify attributes of chronic kidney disease (CKD) for use in low-cost diagnostic screening. IEEE Journal of

Translational Engineering in Health and Medicine, 9, 1-11. https://ieeexplore.ieee.org/abstract/document/9405681

Senan, E. M., Al-Adhaileh, M. H., Alsaade, F. W., Aldhyani, T. H., Alqarni, A. A., Alsharif, N., ... & Alzahrani, M. Y. (2021). Diagnosis of chronic kidney disease using effective classification algorithms and recursive feature elimination techniques. Journal of healthcare engineering, 2021(1), 1004767. https://onlinelibrary.wiley.com/doi/full/10.1155/2021/1004767

Wibowo, M. R., & Palupi, I. (2023). Enhancing Accuracy on Chronic-Kidney Disease Detection Using Machine Learning with Technique of Resampling and Missing Value Treatment. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control. https://kinetik.umm.ac.id/index.php/kinetik/article/view/1761