Chronic Kidney Disease Prediction

Introduction

Chronic kidney disease (CKD) is among the signifcant contributor to morbidity and mortality from non-communicable diseases that can afected 10–15% of the global population. Early and accurate detection of the stages of CKD is believed to be vital to minimize impacts of patient’s health complications such as hypertension, anemia (low blood count), mineral bone disorder, poor nutritional health, acid base abnormalities, and neurological complications with timely intervention through appropriate medications.

Goal

The Goal of this project is to use machine learning to identify patterns and trends within medical data to detect the patients likelyhood of getting chronic kidney disease (CKD) by providing some of the data related to the patients health. This can really help to prevent the patient from being a patient in the future.

Data Collection

Relevant data collection is really important for modelling and making effective prediction. The data is gathered from the internet that has around 26 different medical test results of 400 patients. Some of the results are absent and some of them are not clear. Data Preprocessing is really important here.

About Data

1. The data is available publically on https://www.kaggle.com/datasets/mansoordaku/ckdisease
2. The dataset contains 26 features which of different medical tests done of patients and the corresponding results.
3. Some of the features like Age, Blood pressure, Sugar levels, Blood Cell Counts, Sodium, etc and the dependent variable is the 'class' which has 'ckd' indicating the patient has the chance to get Kidney disease in the future, and 'no ckd' indicating the patient has less chance to get Kidney disease in the future.
4. The goal is to classify whether the patient has a chance to get the disease given the variables.

Implementation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('/content/kidney_disease.csv')
df
```

```
      id   age    bp     sg   al   su     rbc        pc         pcc          ba  ...  pcv    wc
  0    0  48.0  80.0  1.020  1.0  0.0     NaN    normal  notpresent  notpresent  ...   44  7800
  1    1   7.0  50.0  1.020  4.0  0.0     NaN    normal  notpresent  notpresent  ...   38  6000
  2    2  62.0  80.0  1.010  2.0  3.0  normal    normal  notpresent  notpresent  ...   31  7500
  3    3  48.0  70.0  1.005  4.0  0.0  normal  abnormal     present  notpresent  ...   32  6700
  4    4  51.0  80.0  1.010  2.0  0.0  normal    normal  notpresent  notpresent  ...   35  7300
...  ...   ...   ...    ...  ...  ...     ...       ...         ...         ...  ...  ...   ...
395  395  55.0  80.0  1.020  0.0  0.0  normal    normal  notpresent  notpresent  ...   47  6700
396  396  42.0  70.0  1.025  0.0  0.0  normal    normal  notpresent  notpresent  ...   54  7800
397  397  12.0  80.0  1.020  0.0  0.0  normal    normal  notpresent  notpresent  ...   49  6600
398  398  17.0  60.0  1.025  0.0  0.0  normal    normal  notpresent  notpresent  ...   51  7200
399  399  58.0  80.0  1.025  0.0  0.0  normal    normal  notpresent  notpresent  ...   53  6800

400 rows × 26 columns
```

```
df.drop('id', axis = 1, inplace = True)
```

```
# Copy for future use

df_copy = df.copy()
```

```
df.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_cells', 'pus_cell',
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count',
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite', 'peda_edema',
              'aanemia', 'class']
```

```
df.head()
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pu
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	

5 rows × 25 columns

```
df.describe()
```

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucos
count	391.000000	388.000000	353.000000	354.000000	351.000000	35
mean	51.483376	76.469072	1.017408	1.016949	0.450142	14
std	17.169714	13.683637	0.005717	1.352679	1.099191	7
min	2.000000	50.000000	1.005000	0.000000	0.000000	2
25%	42.000000	70.000000	1.010000	0.000000	0.000000	9
50%	55.000000	80.000000	1.020000	0.000000	0.000000	12
75%	64.500000	80.000000	1.020000	2.000000	0.000000	16
max	90.000000	180.000000	1.025000	5.000000	5.000000	49

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   age                      391 non-null    float64
 1   blood_pressure           388 non-null    float64
 2   specific_gravity         353 non-null    float64
 3   albumin                  354 non-null    float64
 4   sugar                    351 non-null    float64
 5   red_blood_cells          248 non-null    object 
 6   pus_cell                 335 non-null    object 
 7   pus_cell_clumps          396 non-null    object 
 8   bacteria                 396 non-null    object 
 9   blood_glucose_random     356 non-null    float64
 10  blood_urea               381 non-null    float64
 11  serum_creatinine         383 non-null    float64
 12  sodium                   313 non-null    float64
 13  potassium                312 non-null    float64
 14  haemoglobin              348 non-null    float64
 15  packed_cell_volume       330 non-null    object 
 16  white_blood_cell_count   295 non-null    object 
 17  red_blood_cell_count     270 non-null    object 
 18  hypertension             398 non-null    object 
 19  diabetes_mellitus        398 non-null    object 
 20  coronary_artery_disease  398 non-null    object 
 21  appetite                 399 non-null    object 
 22  peda_edema               399 non-null    object 
 23  aanemia                  399 non-null    object 
 24  class                    400 non-null    object 
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

```
object_columns = df.select_dtypes(include = ['object'])
print(object_columns)
```

```
    red_blood_cells  pus_cell pus_cell_clumps    bacteria packed_cell_volume  \
0               NaN    normal      notpresent  notpresent                 44
1               NaN    normal      notpresent  notpresent                 38
2            normal    normal      notpresent  notpresent                 31
3            normal  abnormal         present  notpresent                 32
4            normal    normal      notpresent  notpresent                 35
..              ...       ...             ...         ...                ...
395          normal    normal      notpresent  notpresent                 47
396          normal    normal      notpresent  notpresent                 54
397          normal    normal      notpresent  notpresent                 49
398          normal    normal      notpresent  notpresent                 51
399          normal    normal      notpresent  notpresent                 53

    white_blood_cell_count red_blood_cell_count hypertension  \
0                     7800                  5.2          yes
1                     6000                  NaN           no
2                     7500                  NaN           no
3                     6700                  3.9          yes
4                     7300                  4.6           no
..                     ...                  ...          ...
395                   6700                  4.9           no
396                   7800                  6.2           no
397                   6600                  5.4           no
398                   7200                  5.9           no
399                   6800                  6.1           no

    diabetes_mellitus coronary_artery_disease appetite peda_edema aanemia  \
0                 yes                      no     good         no      no
1                  no                      no     good         no      no
2                 yes                      no     poor         no     yes
3                  no                      no     poor        yes     yes
4                  no                      no     good         no      no
..                ...                     ...      ...        ...     ...
395                no                      no     good         no      no
396                no                      no     good         no      no
397                no                      no     good         no      no
398                no                      no     good         no      no
399                no                      no     good         no      no

      class
0       ckd
1       ckd
2       ckd
3       ckd
4       ckd
..      ...
395  notckd
396  notckd
397  notckd
398  notckd
399  notckd

[400 rows x 14 columns]
```

Changing the data type from object to numbers

```
df['packed_cell_volume'] = pd.to_numeric(df['packed_cell_volume'], errors = 'coerce')
df['white_blood_cell_count'] = pd.to_numeric(df['white_blood_cell_count'], errors = 'coerce')
df['red_blood_cell_count'] = pd.to_numeric(df['red_blood_cell_count'], errors = 'coerce')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      391 non-null    float64
 1   blood_pressure           388 non-null    float64
 2   specific_gravity         353 non-null    float64
 3   albumin                  354 non-null    float64
 4   sugar                    351 non-null    float64
 5   red_blood_cells          248 non-null    object
 6   pus_cell                 335 non-null    object
 7   pus_cell_clumps          396 non-null    object
```

```
  8   bacteria                 396 non-null    object
  9   blood_glucose_random     356 non-null    float64
  10  blood_urea               381 non-null    float64
  11  serum_creatinine         383 non-null    float64
  12  sodium                   313 non-null    float64
  13  potassium                312 non-null    float64
  14  haemoglobin              348 non-null    float64
  15  packed_cell_volume       329 non-null    float64
  16  white_blood_cell_count   294 non-null    float64
  17  red_blood_cell_count     269 non-null    float64
  18  hypertension             398 non-null    object
  19  diabetes_mellitus        398 non-null    object
  20  coronary_artery_disease  398 non-null    object
  21  appetite                 399 non-null    object
  22  peda_edema               399 non-null    object
  23  aanemia                  399 non-null    object
  24  class                    400 non-null    object
dtypes: float64(14), object(11)
memory usage: 78.2+ KB
```

```
cat_cols = [col for col in df.columns if df[col].dtype == 'object']
num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

```
for col in cat_cols:
    print(f"{col}: {df[col].unique()}:")
```

```
red_blood_cells: [nan 'normal' 'abnormal']:
pus_cell: ['normal' 'abnormal' nan]:
pus_cell_clumps: ['notpresent' 'present' nan]:
bacteria: ['notpresent' 'present' nan]:
hypertension: ['yes' 'no' nan]:
diabetes_mellitus: ['yes' 'no' ' yes' '\tno' '\tyes' nan]:
coronary_artery_disease: ['no' 'yes' '\tno' nan]:
appetite: ['good' 'poor' nan]:
peda_edema: ['no' 'yes' nan]:
aanemia: ['no' 'yes' nan]:
class: ['ckd' 'ckd\t' 'notckd']:
```

Remove and Replace the values that contains noise

```
df['diabetes_mellitus'].replace(to_replace = {'\tno':'no', '\tyes':'yes', ' yes':'yes'}, inplace = True)
df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_replace = '\tno', value = 'no')
df['class'] = df['class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd':'not ckd'})
```

```
df
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell
0	48.0	80.0	1.020	1.0	0.0	NaN	normal
1	7.0	50.0	1.020	4.0	0.0	NaN	normal
2	62.0	80.0	1.010	2.0	3.0	normal	normal
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal
4	51.0	80.0	1.010	2.0	0.0	normal	normal
...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	normal	normal
396	42.0	70.0	1.025	0.0	0.0	normal	normal
397	12.0	80.0	1.020	0.0	0.0	normal	normal
398	17.0	60.0	1.025	0.0	0.0	normal	normal
399	58.0	80.0	1.025	0.0	0.0	normal	normal

400 rows × 25 columns

```
# df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
# df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

Exploratory Data Analysis (EDA)

EDA is the technique of exploring the data and know its nature and characteristics including the distribution of each data using univariate analysis, relation between two variables (bivariate analysis) and relation between multiple variables (Multivariate analysis). Using statistical and visualization techniques, we can identify the patterns and the characteristics of the data which come under EDA.

Univariate Analysis

Univariate analysis includes analysing each variables or features in the dataset individually one at a time.

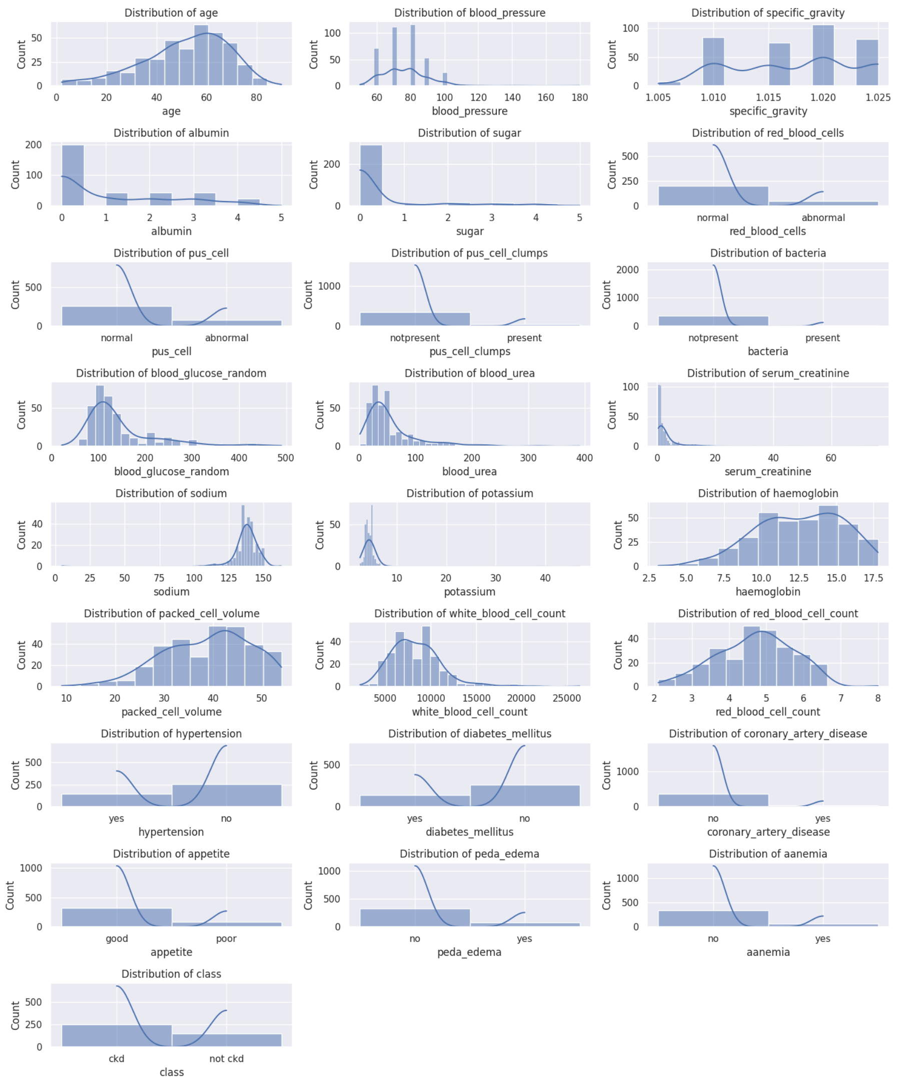
```
feature_columns = df.columns

sns.set_theme()

num_rows = 9
num_cols = 3
plt.figure(figsize=(15, 2 * num_rows))

for i, feature in enumerate(feature_columns, 1):
    plt.subplot(num_rows, num_cols, i)
    sns.histplot(df[feature], kde = True)
    plt.title(f"Distribution of {feature}")

plt.tight_layout()
plt.show()
```


Distribution of age
age
Distribution of blood_pressure
blood_pressure
Distribution of specific_gravity
specific_gravity
Distribution of albumin
albumin
Distribution of sugar
sugar
Distribution of red_blood_cells
normal
abnormal
red_blood_cells
Distribution of pus_cell
normal
abnormal
pus_cell
Distribution of pus_cell_clumps
notpresent
present
pus_cell_clumps
Distribution of bacteria
notpresent
present
bacteria
Distribution of blood_glucose_random
blood_glucose_random
Distribution of blood_urea
blood_urea
Distribution of serum_creatinine
serum_creatinine
Distribution of sodium
sodium
Distribution of potassium
potassium
Distribution of haemoglobin
haemoglobin
Distribution of packed_cell_volume
packed_cell_volume
Distribution of white_blood_cell_count
white_blood_cell_count
Distribution of red_blood_cell_count
red_blood_cell_count
Distribution of hypertension
yes
no
hypertension
Distribution of diabetes_mellitus
yes
no
diabetes_mellitus
Distribution of coronary_artery_disease
no
yes
coronary_artery_disease
Distribution of appetite
good
poor
appetite
Distribution of peda_edema
no
yes
peda_edema
Distribution of aanemia
no
yes
aanemia
Distribution of class
ckd
not ckd
class
Count

Univariate Analysis Summary

From the Graph, we can see that the some of the variables follows normal distribution like the Age, blood glucose random, etc. Some are categorical.

Univariate Analysis Summary

Age Distribution:

The plot reveals the distribution of ages in a population sample. A significant portion of individuals falls within the 50-60 years age range.

Blood Pressure:

This plot represents blood pressure levels, with the majority of people having readings around 70-80 mm Hg.

Specific Gravity of Urine:

The specific gravity plot indicates the concentration of urine, with most values clustering around 1.020.

Albumin Level in Urine:

Albumin, a protein found in urine, is mostly at low levels in the majority of samples, suggesting normal kidney function.

Sugar Levels in Urine:

Most individuals exhibit normal sugar levels, while a few show elevated levels.

Red Blood Cells (RBC):

The distribution of RBC is balanced between normal and abnormal cells.

Presence of Pus Cell Clumps:

Most samples do not show pus cell clumps, indicating no infection.

Bacterial Presence:

The majority of samples exhibit no bacterial presence.

Blood Glucose Random:

A peak around 100 mg/dl is observed, with some higher values suggesting possible diabetes.

Blood Urea:

Most values fall between 0-50 mg/dl, indicating normal kidney function.

Serum Creatinine:

Primarily low values are observed, indicating healthy kidney function.

Sodium and Potassium Levels:

Both sodium and potassium levels are normally distributed within healthy ranges for most individuals.

Bivariate Analysis

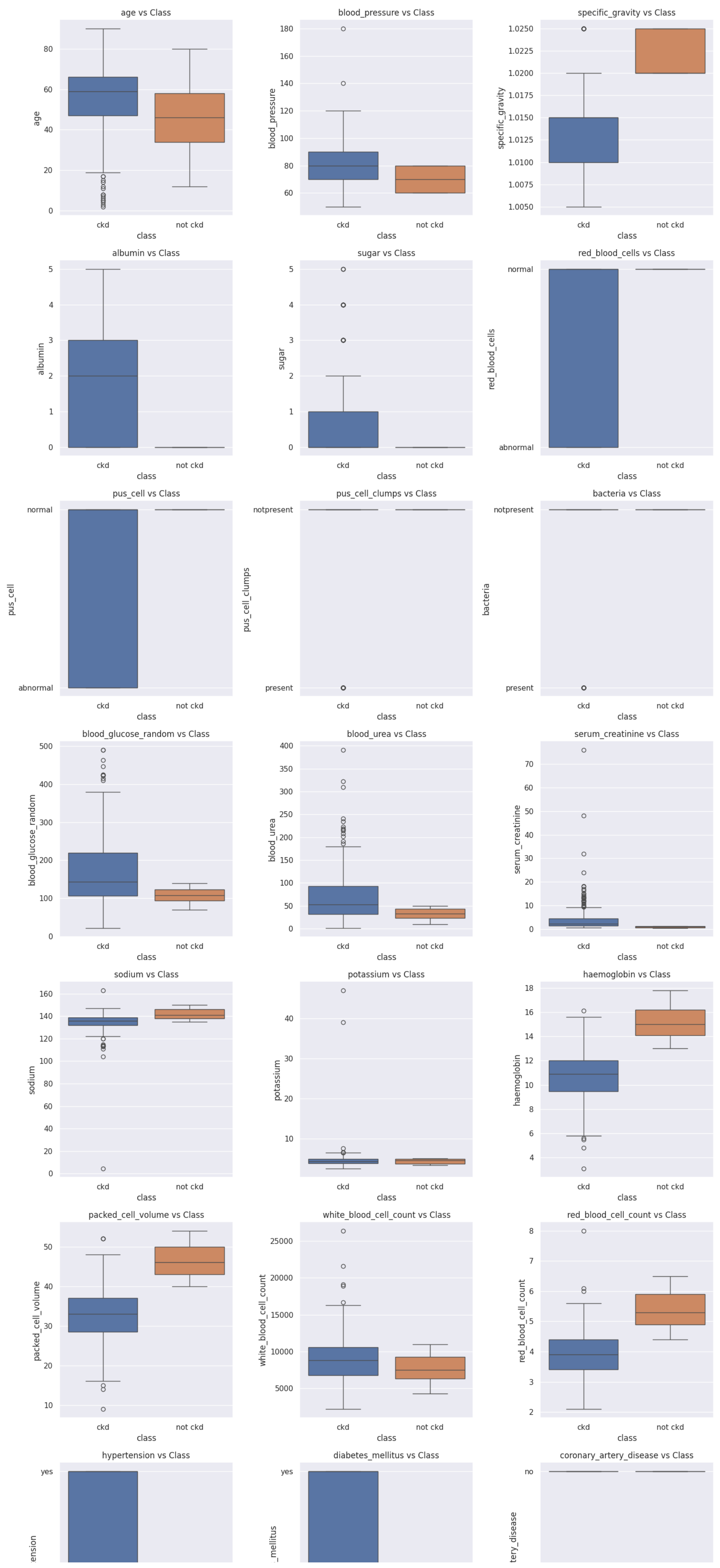
Bivariate analysis includes simultaniously analysing two variables at a time to find their correlation, distribution and how one affect the other. It uses method like correlation, covariance, scatter plots, etc for finding the relation between two variables.

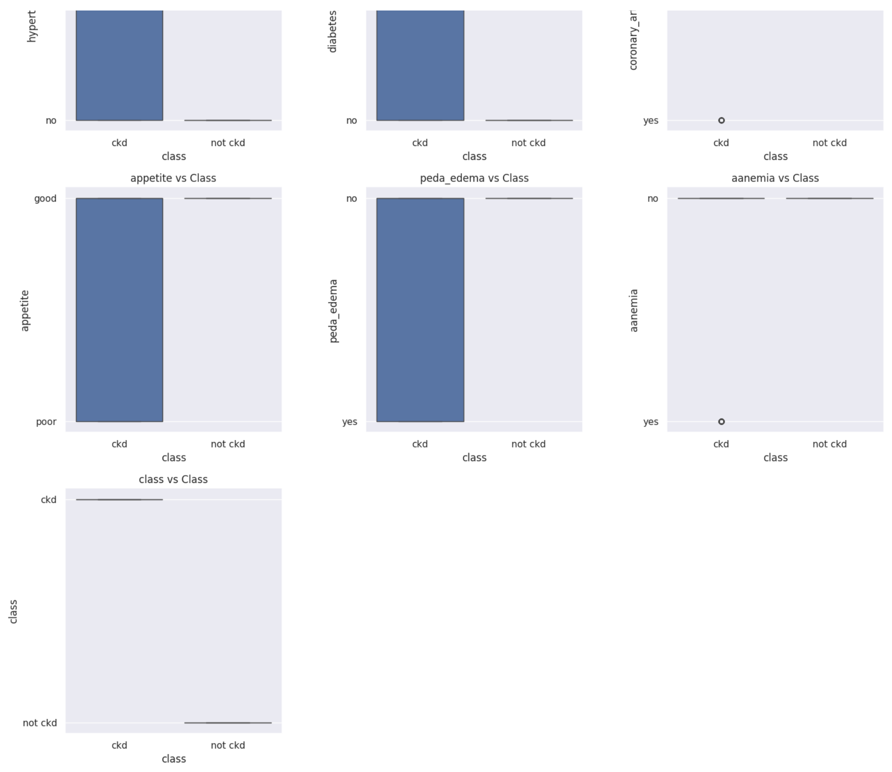
```
feature_columns = df.columns

num_rows = 9
num_cols = 3
plt.figure(figsize=(15, 5 * num_rows))

for i, feature in enumerate(feature_columns, 1):
    plt.subplot(num_rows, num_cols, i)
    sns.boxplot(data = df, x = 'class', y = feature, hue='class')
    plt.title(f"{feature} vs Class")

plt.tight_layout()
plt.show()
```


age vs Class
blood_pressure vs Class
specific_gravity vs Class
albumin vs Class
sugar vs Class
red_blood_cells vs Class
pus_cell vs Class
pus_cell_clumps vs Class
bacteria vs Class
blood_glucose_random vs Class
blood_urea vs Class
serum_creatinine vs Class
sodium vs Class
potassium vs Class
haemoglobin vs Class
packed_cell_volume vs Class
white_blood_cell_count vs Class
red_blood_cell_count vs Class
hypertension vs Class
diabetes_mellitus vs Class
coronary_artery_disease vs Class
class
ckd
not ckd
normal
abnormal
notpresent
present
yes
no


hypert
no
ckd
not ckd
class
diabetes
no
ckd
not ckd
class
coronary_ar
yes
ckd
not ckd
class
appetite vs Class
good
appetite
poor
ckd
not ckd
class
peda_edema vs Class
no
peda_edema
yes
ckd
not ckd
class
aanemia vs Class
no
aanemia
yes
ckd
not ckd
class
class vs Class
ckd
class
not ckd
ckd
not ckd
class

Multivariate Analysis

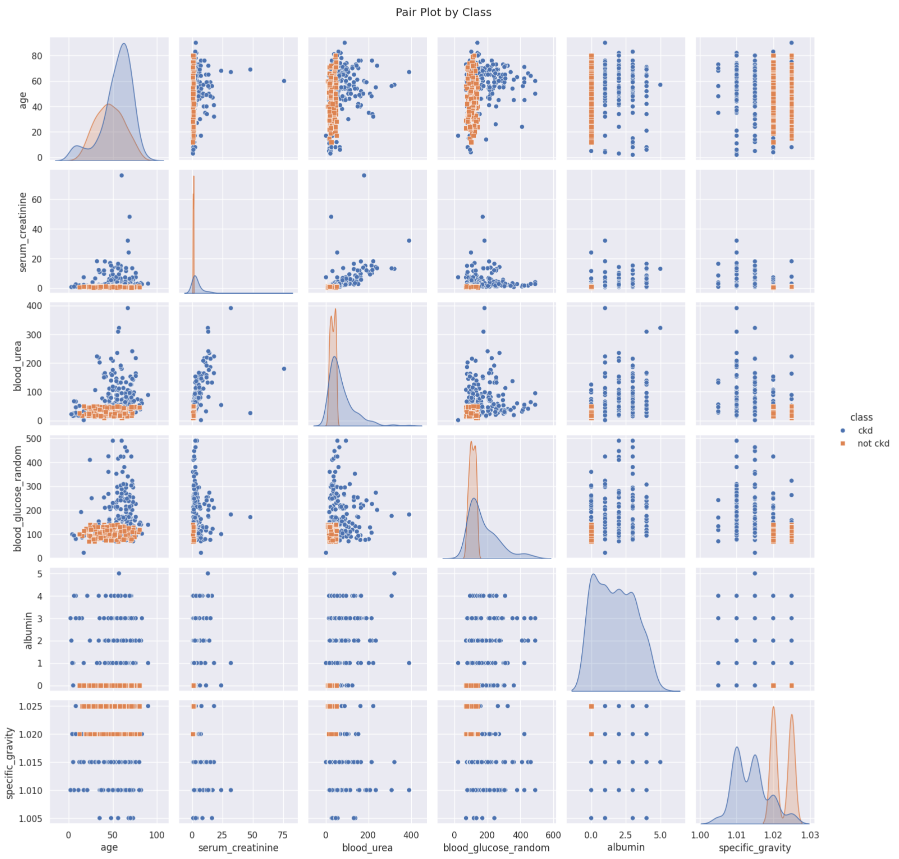
Multivariate analysis is the idea of taking mutiple variables simutaniously and analyising them. They take into accound multiple features and produce a combined result of all of the variables taken.

```
# sns.set_theme(style="darkgrid")
# plt.style.use('dark_background')

subsampled_df = df.sample(frac = 0.1, random_state = 42)

selected_features = ['age', 'serum_creatinine', 'blood_urea', 'blood_glucose_random', 'hypertension', 'albumin', 'red_blood_cells', 'specifi

sns.pairplot(df[selected_features], hue = 'class', markers = ['o', 's'], diag_kind = 'kde', height = 2.5)
plt.suptitle("Pair Plot by Class", y = 1.02)
plt.show()
```

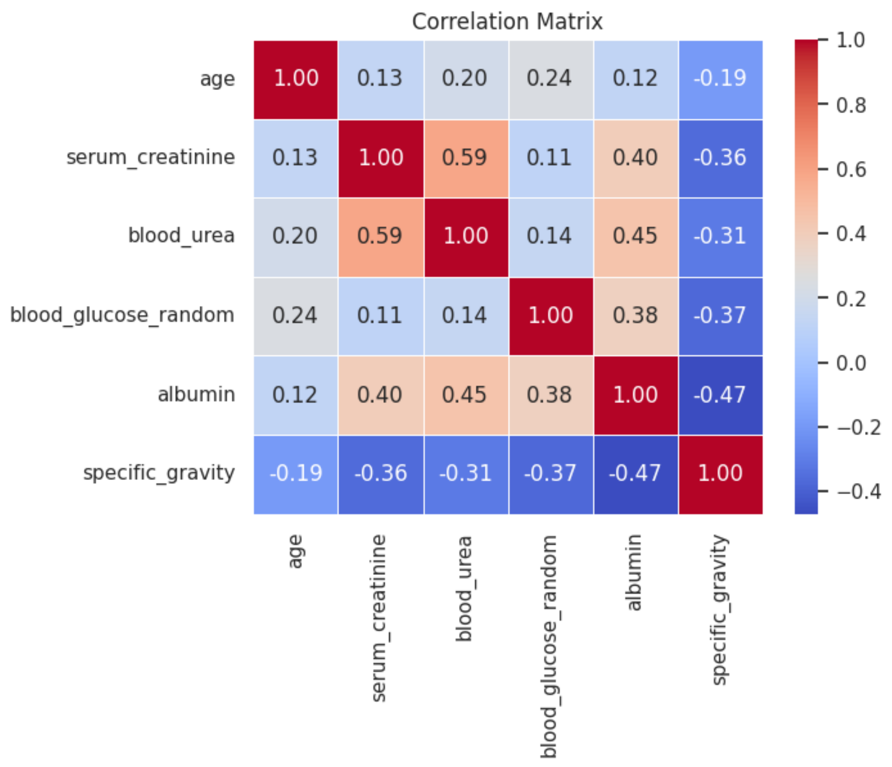


Correlation Matrix

```
corr_matrix = df[selected_features[:-1]].corr()

sns.heatmap(corr_matrix, annot = True, cmap = 'coolwarm', fmt = '.2f', linewidth = 0.5)
plt.title("Correlation Matrix")
plt.show()
```

```
<ipython-input-20-762e3c71b5e2>:1: FutureWarning: The default value of numeric_only in D
  corr_matrix = df[selected_features[:-1]].corr()
```



	age	serum_creatinine	blood_urea	blood_glucose_random	albumin	specific_gravity
age	1.00	0.13	0.20	0.24	0.12	-0.19
serum_creatinine	0.13	1.00	0.59	0.11	0.40	-0.36
blood_urea	0.20	0.59	1.00	0.14	0.45	-0.31
blood_glucose_random	0.24	0.11	0.14	1.00	0.38	-0.37
albumin	0.12	0.40	0.45	0.38	1.00	-0.47
specific_gravity	-0.19	-0.36	-0.31	-0.37	-0.47	1.00

Correlation Matrix Summary

Strong Positive Correlation:

The serum creatinine level and blood urea level exhibit a strong positive correlation with a coefficient of 0.59. This means that as serum creatinine increases, blood urea tends to increase as well.

Negative Correlation:

There's a negative correlation between specific gravity and blood urea with a coefficient of -0.31. When specific gravity increases, blood urea tends to decrease.

No Correlation:

Variables that are uncorrelated have coefficients close to 0. For instance, there is no significant correlation between age, blood glucose random, and albumin levels.

Data Preprocessing / Cleaning

Data Preprocessing / cleaning is the process of removing noise, handling outliers, handling imbalance, etc. This involves statistical techniques to imporve the quality of messy data which then can be used for training machine learning models.

Handling Missing Values

```
df.isna().any()
```

```
    age                       True
    blood_pressure            True
    specific_gravity          True
    albumin                   True
    sugar                     True
    red_blood_cells           True
    pus_cell                  True
```

```
pus_cell_clumps             True
bacteria                    True
blood_glucose_random        True
blood_urea                  True
serum_creatinine            True
sodium                      True
potassium                   True
haemoglobin                 True
packed_cell_volume          True
white_blood_cell_count      True
red_blood_cell_count        True
hypertension                True
diabetes_mellitus           True
coronary_artery_disease     True
appetite                    True
peda_edema                  True
aanemia                     True
class                      False
dtype: bool
```

```
# Missing value percentage

df.isna().mean() * 100
```

```
age                         2.25
blood_pressure              3.00
specific_gravity           11.75
albumin                    11.50
sugar                      12.25
red_blood_cells            38.00
pus_cell                   16.25
pus_cell_clumps             1.00
bacteria                    1.00
blood_glucose_random       11.00
blood_urea                  4.75
serum_creatinine            4.25
sodium                     21.75
potassium                  22.00
haemoglobin                13.00
packed_cell_volume         17.75
white_blood_cell_count     26.50
red_blood_cell_count       32.75
hypertension                0.50
diabetes_mellitus           0.50
coronary_artery_disease     0.50
appetite                    0.25
peda_edema                  0.25
aanemia                     0.25
class                       0.00
dtype: float64
```

```
cat_cols = [col for col in df.columns if df[col].dtype == 'object']
num_cols = [col for col in df.columns if df[col].dtype != 'object']
```

```
df[num_cols].isnull().sum()
```

```
age                         9
blood_pressure             12
specific_gravity           47
albumin                    46
sugar                      49
blood_glucose_random       44
blood_urea                 19
serum_creatinine           17
sodium                     87
potassium                  88
haemoglobin                52
packed_cell_volume         71
white_blood_cell_count    106
red_blood_cell_count      131
dtype: int64
```

```
df[cat_cols].isnull().sum()
```

```
red_blood_cells            152
pus_cell                    65
pus_cell_clumps              4
bacteria                     4
hypertension                 2
diabetes_mellitus            2
```

```
coronary_artery_disease      2
appetite                     1
peda_edema                   1
aanemia                      1
class                        0
dtype: int64
```

Imputing Numerical variables

Numerical varaibles can be imputed using mean which is a common approach. But, since the dataset has a huge portion of missing values, random sampling imputation works well.

```
def random_value_imputation(feature):
    random_sample = df[feature].dropna().sample(df[feature].isna().sum())
    random_sample.index = df[df[feature].isnull()].index
    df.loc[df[feature].isnull(), feature] = random_sample
```

```
for col in num_cols:
    random_value_imputation(col)
```

```
df[num_cols].isna().sum()
```

```
age                        0
blood_pressure             0
specific_gravity           0
albumin                    0
sugar                      0
blood_glucose_random       0
blood_urea                 0
serum_creatinine           0
sodium                     0
potassium                  0
haemoglobin                0
packed_cell_volume         0
white_blood_cell_count     0
red_blood_cell_count       0
dtype: int64
```

Imputing Categorical variables

Imputation in categorical variables are mostly done by mode (Most occuring value). If the variable contains a huge portion of missing values, random imputation works here as well.

```
def mode_imputation(feature):
    mode = df[feature].mode()[0]
    df[feature] = df[feature].fillna(mode)

# Filling the 'red_blood_cell' and 'puss_cell' with random value imputation since these two contains more number of missing values
random_value_imputation('red_blood_cells')
random_value_imputation('pus_cell')

# Rest of them are imputed using mean
for col in cat_cols:
    mode_imputation(col)
```

```
df[cat_cols].isna().sum()
```

```
red_blood_cells            0
pus_cell                   0
pus_cell_clumps            0
bacteria                   0
hypertension               0
diabetes_mellitus          0
coronary_artery_disease    0
appetite                   0
peda_edema                 0
aanemia                    0
class                      0
dtype: int64
```

```
df
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell
0	48.0	80.0	1.020	1.0	0.0	normal	normal
1	7.0	50.0	1.020	4.0	0.0	normal	normal
2	62.0	80.0	1.010	2.0	3.0	normal	normal
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal
4	51.0	80.0	1.010	2.0	0.0	normal	normal
...	...	...	...	...	...	...	...
395	55.0	80.0	1.020	0.0	0.0	normal	normal
396	42.0	70.0	1.025	0.0	0.0	normal	normal
397	12.0	80.0	1.020	0.0	0.0	normal	normal
398	17.0	60.0	1.025	0.0	0.0	normal	normal
399	58.0	80.0	1.025	0.0	0.0	normal	normal

400 rows × 25 columns

Categorical Featue Encoding

Since the values are mostly in string format, we need to encode it in numbers. For example: the 'puss_cell_clumps' has values 'present' and 'not present', we need to convert it to numbers like this,

present -> 1

not present -> 0

```
for col in cat_cols:
    print(df[col].nunique())
```

```
2
2
2
2
2
2
2
2
2
2
2
```

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()

for col in cat_cols:
    df[col] = encoder.fit_transform(df[col])
```

```
df.head()
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell
0	48.0	80.0	1.020	1.0	0.0	1	1
1	7.0	50.0	1.020	4.0	0.0	1	1
2	62.0	80.0	1.010	2.0	3.0	1	1
3	48.0	70.0	1.005	4.0	0.0	1	0
4	51.0	80.0	1.010	2.0	0.0	1	1

5 rows × 25 columns

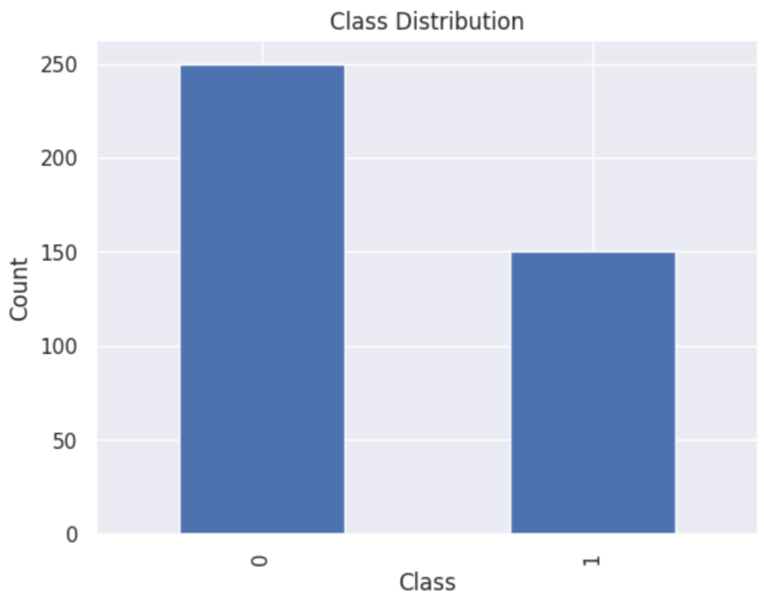
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 25 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   age                      400 non-null    float64
 1   blood_pressure           400 non-null    float64
 2   specific_gravity         400 non-null    float64
 3   albumin                  400 non-null    float64
 4   sugar                    400 non-null    float64
 5   red_blood_cells          400 non-null    int64
 6   pus_cell                 400 non-null    int64
 7   pus_cell_clumps          400 non-null    int64
 8   bacteria                 400 non-null    int64
 9   blood_glucose_random     400 non-null    float64
 10  blood_urea               400 non-null    float64
 11  serum_creatinine         400 non-null    float64
 12  sodium                   400 non-null    float64
 13  potassium                400 non-null    float64
 14  haemoglobin              400 non-null    float64
 15  packed_cell_volume       400 non-null    float64
 16  white_blood_cell_count   400 non-null    float64
 17  red_blood_cell_count     400 non-null    float64
 18  hypertension             400 non-null    int64
 19  diabetes_mellitus        400 non-null    int64
 20  coronary_artery_disease  400 non-null    int64
 21  appetite                 400 non-null    int64
 22  peda_edema               400 non-null    int64
 23  aanemia                  400 non-null    int64
 24  class                    400 non-null    int64
dtypes: float64(14), int64(11)
memory usage: 78.2 KB
```

Model Development and Evaluation

```
# Checking for imbalance in the dataset

df['class'].value_counts().plot(kind = 'bar')
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



```
X = df.drop(['class'], axis = 1)
y = df['class']
```

```
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 = 42)
```

Scaling the dataset

Scaling the dataset involves transforming the data points that are far apart into a specific range of values, this really helps machine learning models to perform well, since there is low computation required.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Logistic Regression
log_reg_params = {'C':[0.001, 0.01, 0.1, 1, 10, 100]}
log_reg = GridSearchCV(LogisticRegression(max_iter = 1000), log_reg_params, cv = 5)
log_reg.fit(X_train_scaled, y_train)


# Random Forest
rf_params = {'n_estimators':[50, 100, 200], 'max_depth': [None, 10, 20, 30], 'min_samples_split':[2, 5, 10], 'min_samples_leaf':[1, 2, 4]}
rf = GridSearchCV(RandomForestClassifier(), rf_params, cv = 5)
rf.fit(X_train_scaled, y_train)

# Support Vector Machine
svm_params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svm = GridSearchCV(SVC(probability=True), svm_params, cv=5)
svm.fit(X_train_scaled, y_train)

# K-Nearest Neighbors
knn_params = {'n_neighbors': [3, 5, 7, 10], 'weights': ['uniform', 'distance']}
knn = GridSearchCV(KNeighborsClassifier(), knn_params, cv=5)
knn.fit(X_train_scaled, y_train)
```

```
▸          GridSearchCV
▸ estimator: KNeighborsClassifier
```