MALLA REDDY UNIVERSITY

BACHELOR OF TECHNOLOGY

IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

**TITLE:**

# “Prediction of chronic kidney disease using datamining techniques”

**NAME:**D.LAKSHMI GOWRI

**ROLL NUMBER:**2111CS020237

## 1. Objectives

To predict the chronic kidney disease using data mining techniques.

## 2. Dataset Description

**Dataset Name: Chronic Kidney Disease**

The Dataset we are using is from UCI machine learning repository. The Dataset contains 24 health related attributes like age, blood pressure, sugar, glucose, taken in 2-month period of 400 patients in which 11 numeric and 14 nominal attributes in which it consists of class label named ‘Class’ which classifies patients having disease and not present

https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease

**Data Set Information:**

There are 24 features + class = 25 attributes

1. Age(numerical): age in years
2. Blood Pressure(numerical): bp in mm/Hg
3. Specific Gravity(nominal): sg - (1.005,1.010,1.015,1.020,1.025)
4. Albumin(nominal): al - (0,1,2,3,4,5)
5. Sugar(nominal): su - (0,1,2,3,4,5)
6. Red Blood Cells(nominal): rbc - (normal,abnormal)
7. Pus Cell (nominal): pc - (normal,abnormal)
8. Pus Cell clumps(nominal): pcc - (present,notpresent)
9. Bacteria(nominal): ba - (present,notpresent)
10. Blood Glucose Random(numerical): bgr in mgs/dl
11. Blood Urea(numerical): bu in mgs/dl
12. Serum Creatinine(numerical): sc in mgs/dl
13. Sodium(numerical): sod in mEq/L
14. Potassium(numerical): pot in mEq/L
15. Hemoglobin(numerical): hemo in gms
16. Packed Cell Volume(numerical)
17. White Blood Cell Count(numerical): wc in cells/cumm

18. Red Blood Cell Count(numerical): rc in millions/cmm

19.Hypertension(nominal): htn - (yes,no)

20.Diabetes Mellitus(nominal): dm - (yes,no)

21.Coronary Artery Disease(nominal): cad - (yes,no)

22.Appetite(nominal): appet - (good,poor)

23.Pedal Edema(nominal): pe - (yes,no)

24.Anemia(nominal): ane - (yes,no)

25.Class (nominal): class - (ckd, notckd)

## 3. Data Pre-Processing Steps used

- The techniques used for filling the null values is mode.
- Used Box plot to find the outliers.
- Performed label encoding for categorical attributes.
- We used PCA to reduce the number of independent variables.
- Performed normalization of values using min-max normalization.

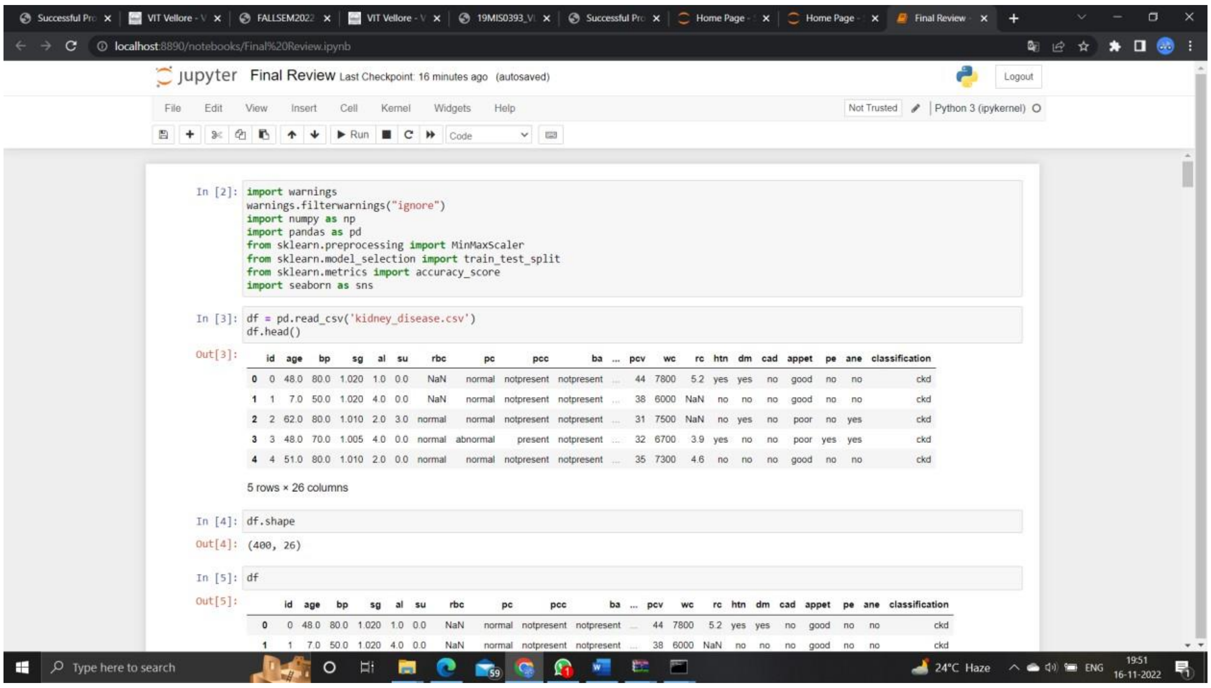
### 3.1 Code

```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import seaborn as sns
df = pd.read_csv('kidney_disease.csv')
df.head()
df.shapedf.isnull().sum()
# Impuing Null values
from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
df_imputed=pd.DataFrame(imp_mode.fit_transform(df))
```

```
df_imputed.columns=df.columns
df_imputed
df_imputed.isnull().sum()
# Finding Unique values in the columns
for i in df_imputed.columns:

print("**********************************************",i,"***********************
***********************************")
 print()
 print(set(df_imputed[i].tolist()))
print()
print(df_imputed["rc"].mode())
print(df_imputed["wc"].mode())
print(df_imputed["pcv"].mode())
df_imputed["classification"]
df_imputed["classification"]=df_imputed["classification"].apply(lambda x:'ckd' if x=="ckd\t" else x)
df_imputed["cad"]=df_imputed["cad"].apply(lambda x:'no' if x=="\tno" else x)
df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'no' if x=="\tno" else x)
df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'yes' if x=="\tyes" else x)
df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'yes' if x==' yes' else x)
df_imputed["rc"]=df_imputed["rc"].apply(lambda x:'5.2' if x=='\t?' else x)
df_imputed["wc"]=df_imputed["wc"].apply(lambda x:'9800' if x=='\t6200' else x)
df_imputed["wc"]=df_imputed["wc"].apply(lambda x:'9800' if x=='\t8400' else x)
df_imputed["wc"]=df_imputed["wc"].apply(lambda x:'9800' if x== '\t?' else x)
df_imputed["pcv"]=df_imputed["pcv"].apply(lambda x:'41' if x=='\t43' else x)
df_imputed["pcv"]=df_imputed["pcv"].apply(lambda x:'41' if x== '\t?'  else x)
# Finding Unique values in the columns
for i in df_imputed.columns:

print("**********************************************",i,"***********************
***********************************")
    print()
    print(set(df_imputed[i].tolist()))
    print()
#Check Label Imbalance
import matplotlib.pyplot as plt
import seaborn as sns
```

```
temp=df_imputed["classification"].value_counts()
temp_df= pd.DataFrame({'classification': temp.index,'values': temp.values})
print(sns.barplot(x = 'classification', y="values", data=temp_df))
df.dtypes
# fixing data types
df_imputed.dtypes
for i in df.select_dtypes(exclude=["object"]).columns:
    df_imputed[i]=df_imputed[i].apply(lambda x: float(x))
df_imputed.dtypes
sns.pairplot(df_imputed)
# Find and remove outliers of data
def boxplots(col):
    sns.boxplot(df[col])
    plt.show()
    for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
    boxplots(i)
# Find the distribution of data
def distplots(col):
    sns.distplot(df[col])
    plt.show()
    for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
    distplots(i)
# Label encoding to convert categorical values to numerical
from sklearn import preprocessing
df_enco=df_imputed.apply(preprocessing.LabelEncoder().fit_transform)
df_enco
# Finding Correlations
import matplotlib.pyplot as plt
plt.figure(figsize=(20,20))
corr=df_enco.corr()
sns.heatmap(corr,annot=True)
```

## 3.2 Screenshots


localhost:8890/notebooks/Final%20Review.ipynb
jupyter Final Review Last Checkpoint: 16 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Type here to search
24°C Haze
ENG
19:51
16-11-2022

```
In [2]: import warnings
        warnings.filterwarnings("ignore")
        import numpy as np
        import pandas as pd
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        import seaborn as sns
```

```
In [3]: df = pd.read_csv('kidney_disease.csv')
        df.head()
```

Out[3]:

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| **1** | 1 | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |
| **2** | 2 | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | ... | 31 | 7500 | NaN | no | yes | no | poor | no | yes | ckd |
| **3** | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | ... | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| **4** | 4 | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 35 | 7300 | 4.6 | no | no | no | good | no | no | ckd |

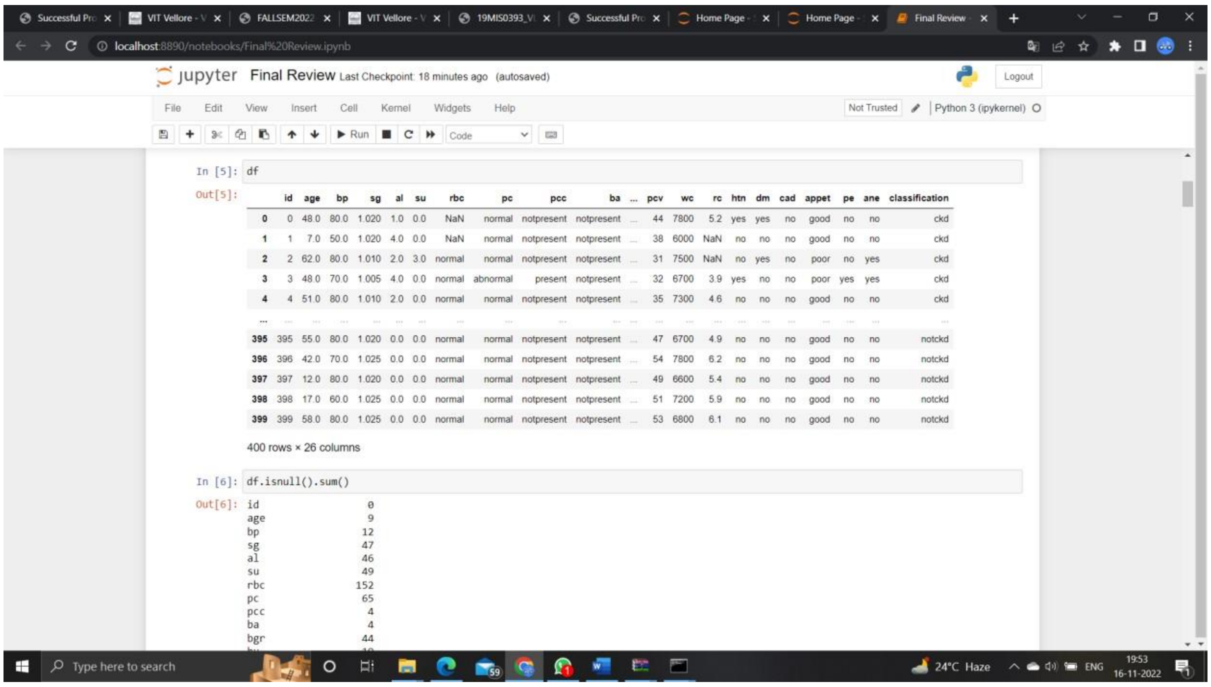
5 rows × 26 columns

```
In [4]: df.shape
Out[4]: (400, 26)
```

```
In [5]: df
```

Out[5]:

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| **1** | 1 | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |


localhost:8890/notebooks/Final%20Review.ipynb
jupyter Final Review Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Type here to search
24°C Haze
ENG
19:53
16-11-2022

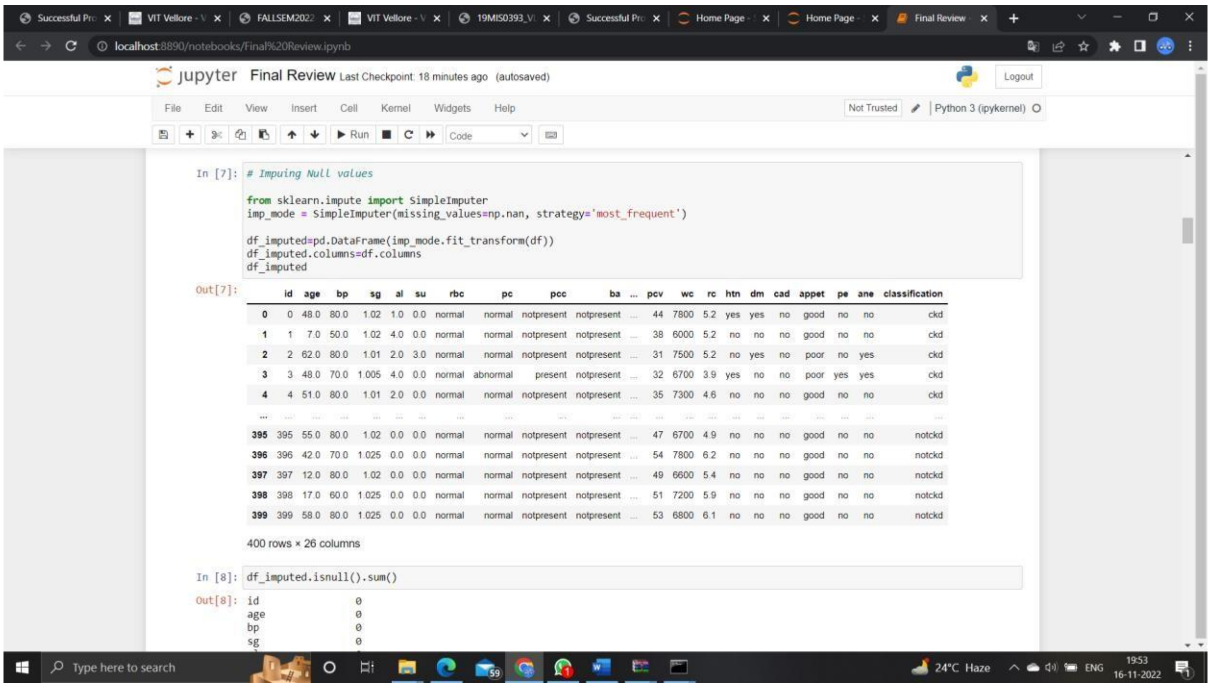
```
In [5]: df
```

Out[5]:

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 48.0 | 80.0 | 1.020 | 1.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| **1** | 1 | 7.0 | 50.0 | 1.020 | 4.0 | 0.0 | NaN | normal | notpresent | notpresent | ... | 38 | 6000 | NaN | no | no | no | good | no | no | ckd |
| **2** | 2 | 62.0 | 80.0 | 1.010 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | ... | 31 | 7500 | NaN | no | yes | no | poor | no | yes | ckd |
| **3** | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | ... | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| **4** | 4 | 51.0 | 80.0 | 1.010 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 35 | 7300 | 4.6 | no | no | no | good | no | no | ckd |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 395 | 55.0 | 80.0 | 1.020 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 47 | 6700 | 4.9 | no | no | no | good | no | no | notckd |
| **396** | 396 | 42.0 | 70.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 54 | 7800 | 6.2 | no | no | no | good | no | no | notckd |
| **397** | 397 | 12.0 | 80.0 | 1.020 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 49 | 6600 | 5.4 | no | no | no | good | no | no | notckd |
| **398** | 398 | 17.0 | 60.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 51 | 7200 | 5.9 | no | no | no | good | no | no | notckd |
| **399** | 399 | 58.0 | 80.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 53 | 6800 | 6.1 | no | no | no | good | no | no | notckd |

400 rows × 26 columns

```
In [6]: df.isnull().sum()
Out[6]: id           0
        age          9
        bp          12
        sg          47
        al          46
        su          49
        rbc        152
        pc          65
        pcc          4
        ba           4
        bgr         44
```


jupyter Final Review Last Checkpoint: 18 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

In [7]:
```
# Impuing Null values

from sklearn.impute import SimpleImputer
imp_mode = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

df_imputed=pd.DataFrame(imp_mode.fit_transform(df))
df_imputed.columns=df.columns
df_imputed
```

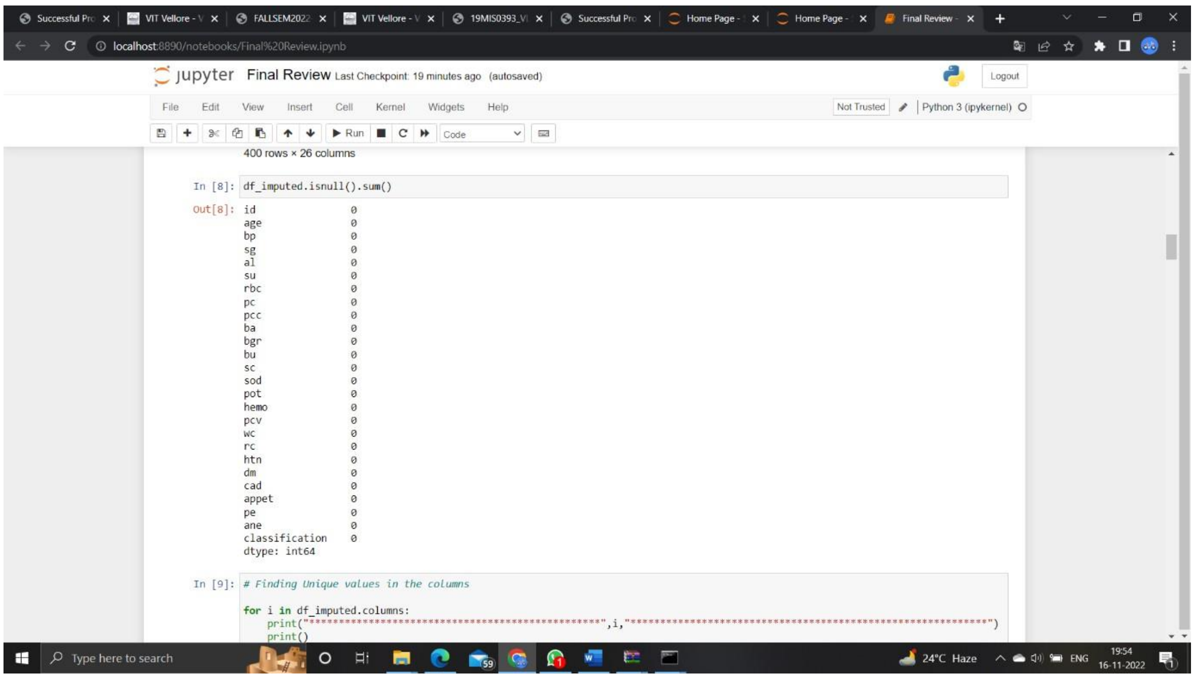
Out[7]:

| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 48.0 | 80.0 | 1.02 | 1.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 44 | 7800 | 5.2 | yes | yes | no | good | no | no | ckd |
| **1** | 1 | 7.0 | 50.0 | 1.02 | 4.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 38 | 6000 | 5.2 | no | no | no | good | no | no | ckd |
| **2** | 2 | 62.0 | 80.0 | 1.01 | 2.0 | 3.0 | normal | normal | notpresent | notpresent | ... | 31 | 7500 | 5.2 | no | yes | no | poor | no | yes | ckd |
| **3** | 3 | 48.0 | 70.0 | 1.005 | 4.0 | 0.0 | normal | abnormal | present | notpresent | ... | 32 | 6700 | 3.9 | yes | no | no | poor | yes | yes | ckd |
| **4** | 4 | 51.0 | 80.0 | 1.01 | 2.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 35 | 7300 | 4.6 | no | no | no | good | no | no | ckd |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **395** | 395 | 55.0 | 80.0 | 1.02 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 47 | 6700 | 4.9 | no | no | no | good | no | no | notckd |
| **396** | 396 | 42.0 | 70.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 54 | 7800 | 6.2 | no | no | no | good | no | no | notckd |
| **397** | 397 | 12.0 | 80.0 | 1.02 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 49 | 6600 | 5.4 | no | no | no | good | no | no | notckd |
| **398** | 398 | 17.0 | 60.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 51 | 7200 | 5.9 | no | no | no | good | no | no | notckd |
| **399** | 399 | 58.0 | 80.0 | 1.025 | 0.0 | 0.0 | normal | normal | notpresent | notpresent | ... | 53 | 6800 | 6.1 | no | no | no | good | no | no | notckd |

400 rows × 26 columns

In [8]:
```
df_imputed.isnull().sum()
```

Out[8]:
```
id                0
age               0
bp                0
sg                0
```

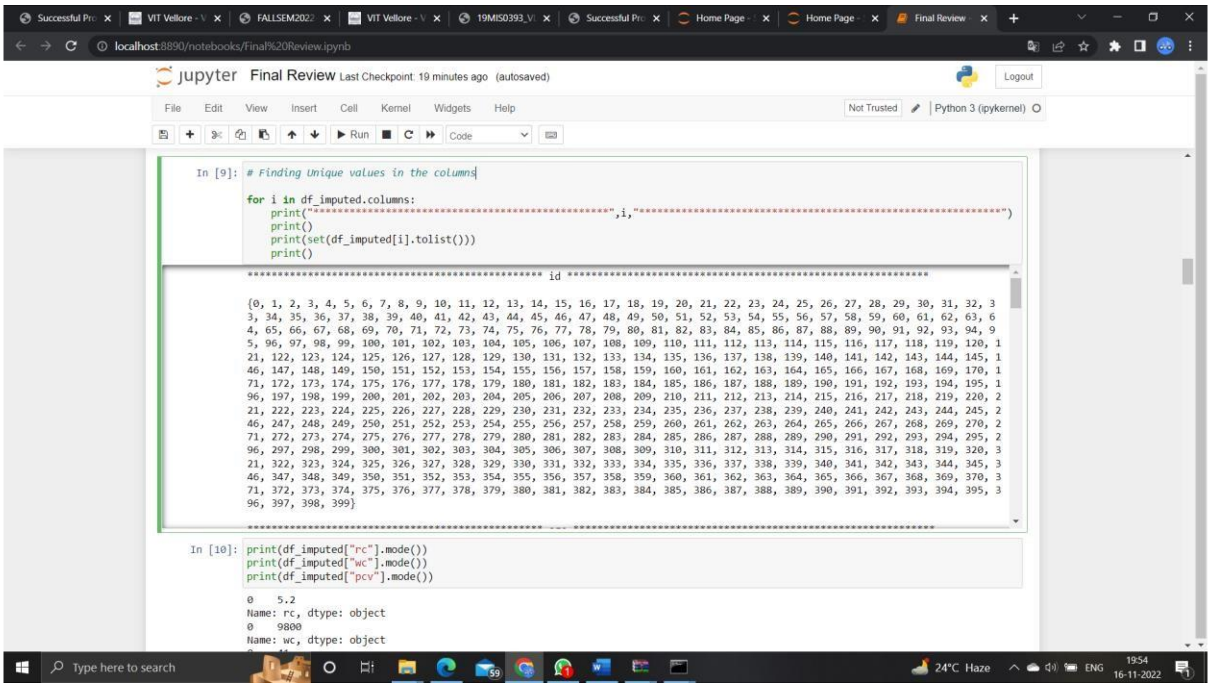

jupyter Final Review Last Checkpoint: 19 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

400 rows × 26 columns

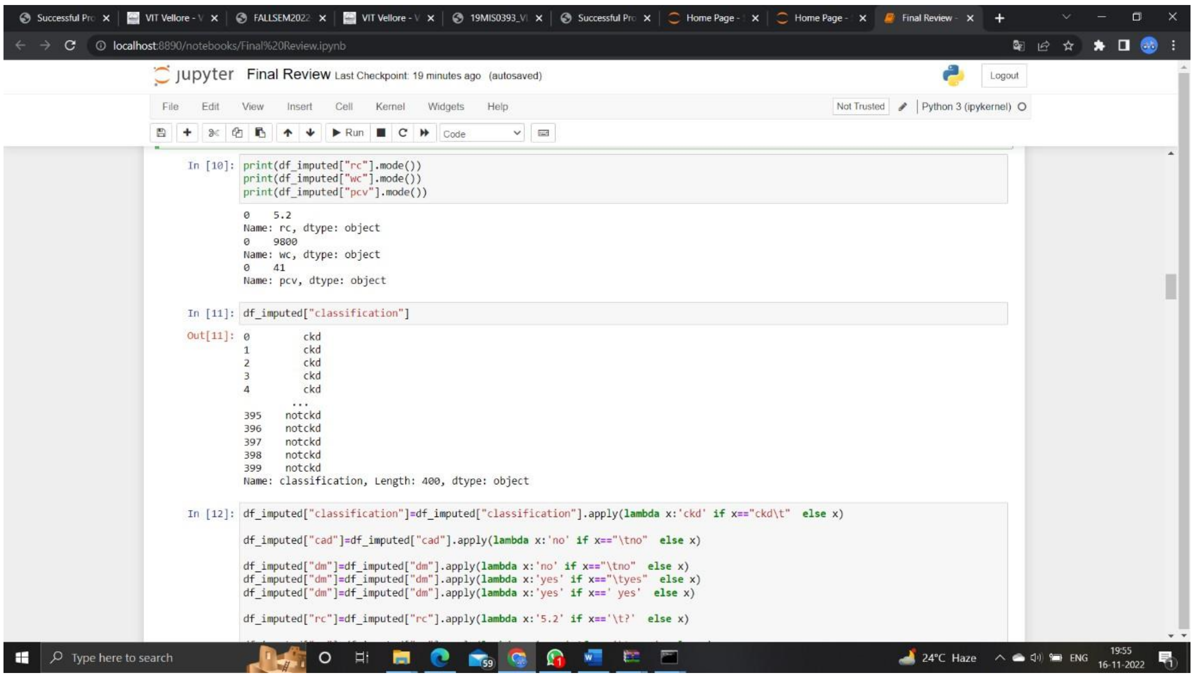
In [8]:
```
df_imputed.isnull().sum()
```

Out[8]:
```
id                0
age               0
bp                0
sg                0
al                0
su                0
rbc               0
pc                0
pcc               0
ba                0
bgr               0
bu                0
sc                0
sod               0
pot               0
hemo              0
pcv               0
wc                0
rc                0
htn               0
dm                0
cad               0
appet             0
pe                0
ane               0
classification    0
dtype: int64
```

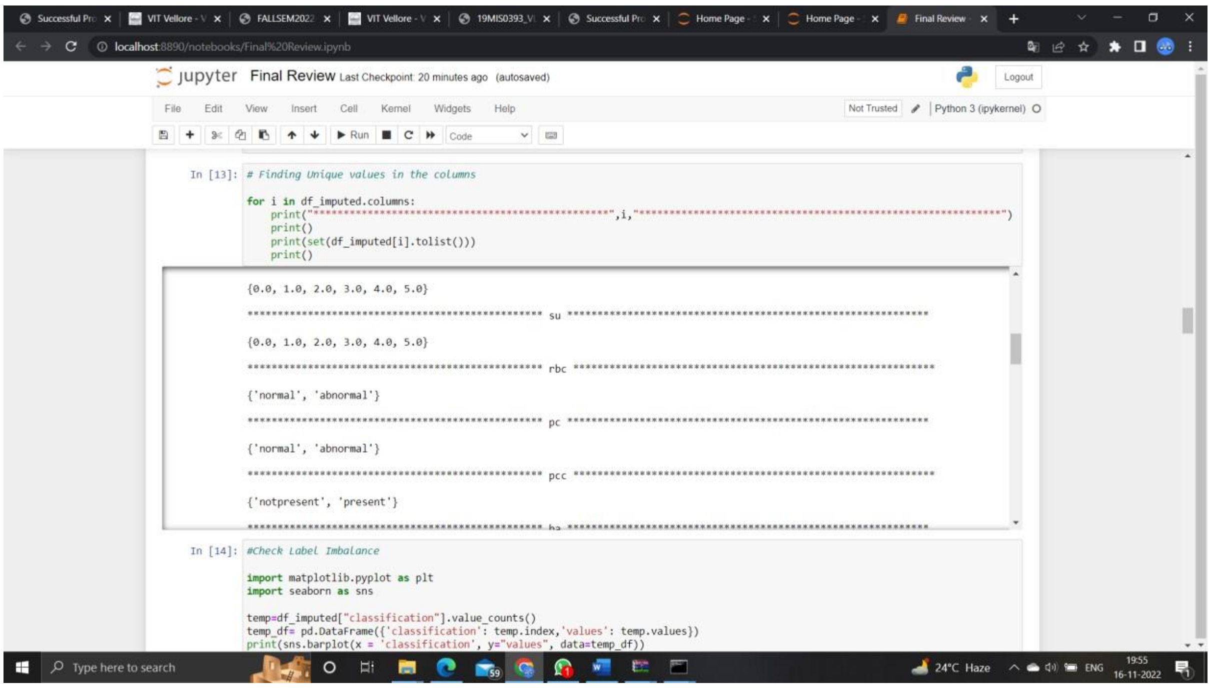
In [9]:
```
# Finding Unique values in the columns

for i in df_imputed.columns:
    print("**************************************************",i,"************************************************************")
    print()
```



```
In [9]: # Finding Unique values in the columns

        for i in df_imputed.columns:
            print("************************************************",i,"************************************************************")
            print()
            print(set(df_imputed[i].tolist()))
            print()

************************************************ id ************************************************************

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 6
4, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 9
5, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 1
21, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 1
46, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 1
71, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 1
96, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 2
21, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 2
46, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 2
71, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 2
96, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 3
21, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 3
46, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 3
71, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 3
96, 397, 398, 399}

************************************************ ... ************************************************************

In [10]: print(df_imputed["rc"].mode())
         print(df_imputed["wc"].mode())
         print(df_imputed["pcv"].mode())

         0    5.2
         Name: rc, dtype: object
         0    9800
         Name: wc, dtype: object
```



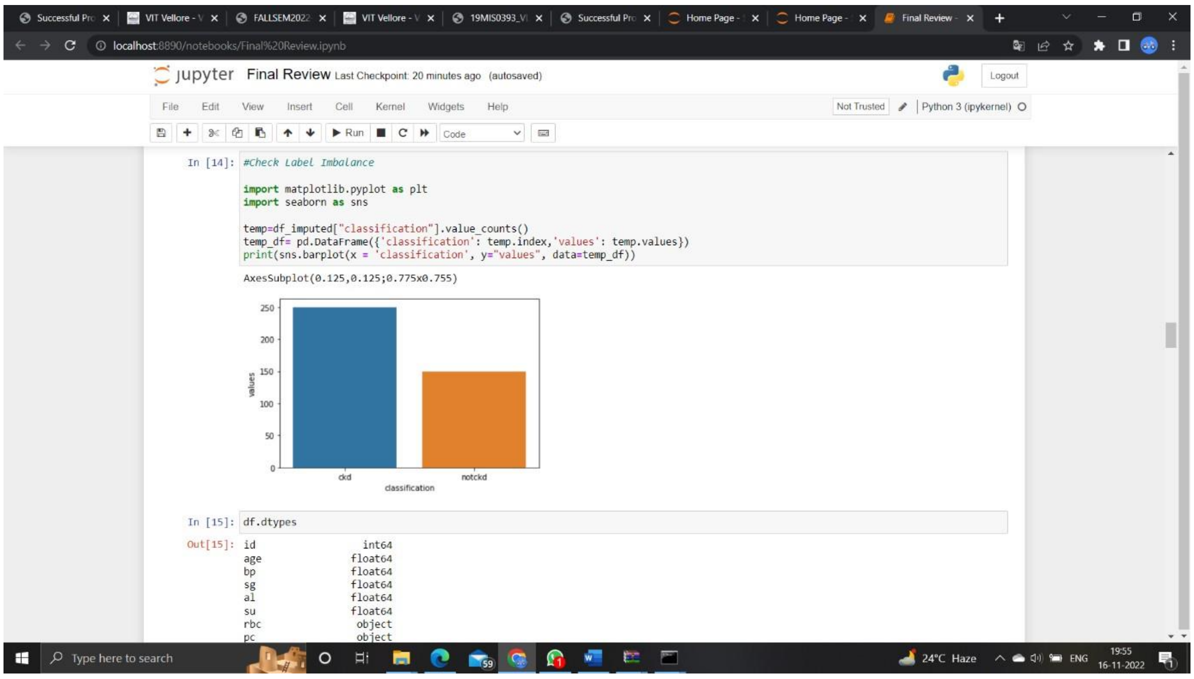
```
In [10]: print(df_imputed["rc"].mode())
         print(df_imputed["wc"].mode())
         print(df_imputed["pcv"].mode())

         0    5.2
         Name: rc, dtype: object
         0    9800
         Name: wc, dtype: object
         0    41
         Name: pcv, dtype: object

In [11]: df_imputed["classification"]

Out[11]: 0         ckd
         1         ckd
         2         ckd
         3         ckd
         4         ckd
                 ...
         395    notckd
         396    notckd
         397    notckd
         398    notckd
         399    notckd
         Name: classification, Length: 400, dtype: object

In [12]: df_imputed["classification"]=df_imputed["classification"].apply(lambda x:'ckd' if x=="ckd\t"  else x)

         df_imputed["cad"]=df_imputed["cad"].apply(lambda x:'no' if x=="\tno"  else x)

         df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'no' if x=="\tno"  else x)
         df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'yes' if x=="\tyes"  else x)
         df_imputed["dm"]=df_imputed["dm"].apply(lambda x:'yes' if x==' yes'  else x)

         df_imputed["rc"]=df_imputed["rc"].apply(lambda x:'5.2' if x=='\t?'  else x)
```


jupyter Final Review Last Checkpoint: 20 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
Logout
Type here to search
24°C Haze
ENG
19:55
16-11-2022

```
In [13]: # Finding Unique values in the columns

         for i in df_imputed.columns:
             print("*************************************************",i,"*************************************************************")
             print()
             print(set(df_imputed[i].tolist()))
             print()
```

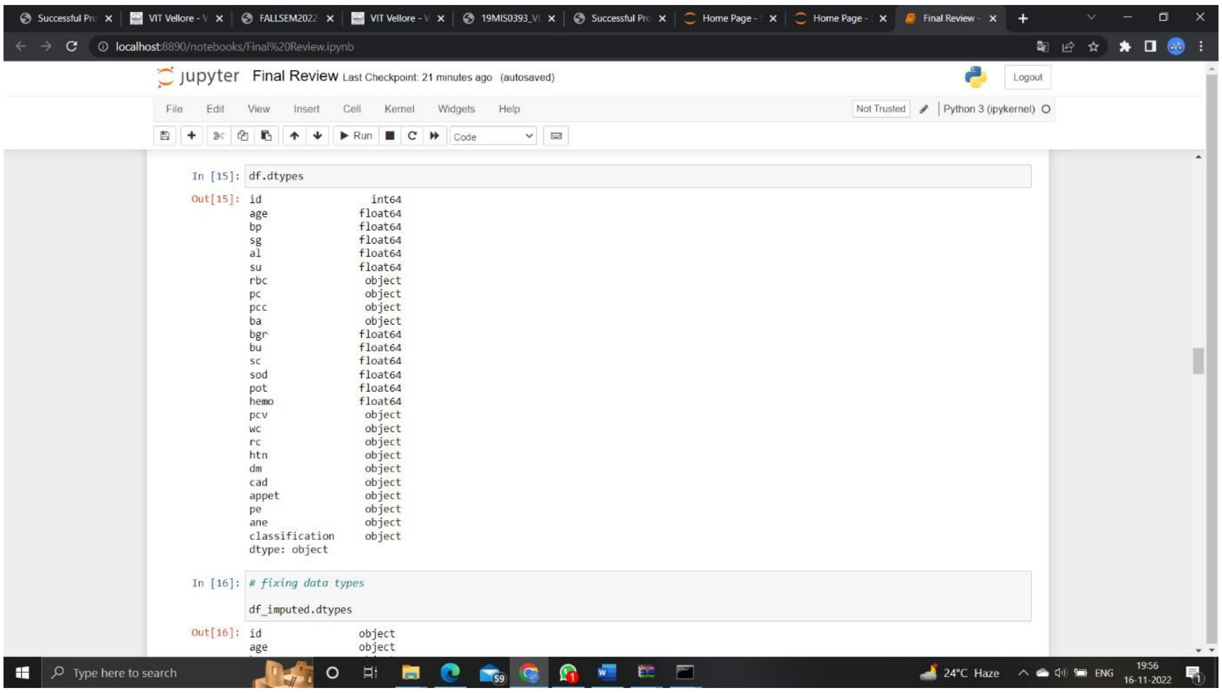
```
{0.0, 1.0, 2.0, 3.0, 4.0, 5.0}

************************************************* su ************************************************************

{0.0, 1.0, 2.0, 3.0, 4.0, 5.0}

************************************************* rbc ************************************************************

{'normal', 'abnormal'}

************************************************* pc ************************************************************

{'normal', 'abnormal'}

************************************************* pcc ************************************************************

{'notpresent', 'present'}
```

```
In [14]: #Check Label Imbalance

         import matplotlib.pyplot as plt
         import seaborn as sns

         temp=df_imputed["classification"].value_counts()
         temp_df= pd.DataFrame({'classification': temp.index,'values': temp.values})
         print(sns.barplot(x = 'classification', y="values", data=temp_df))
```

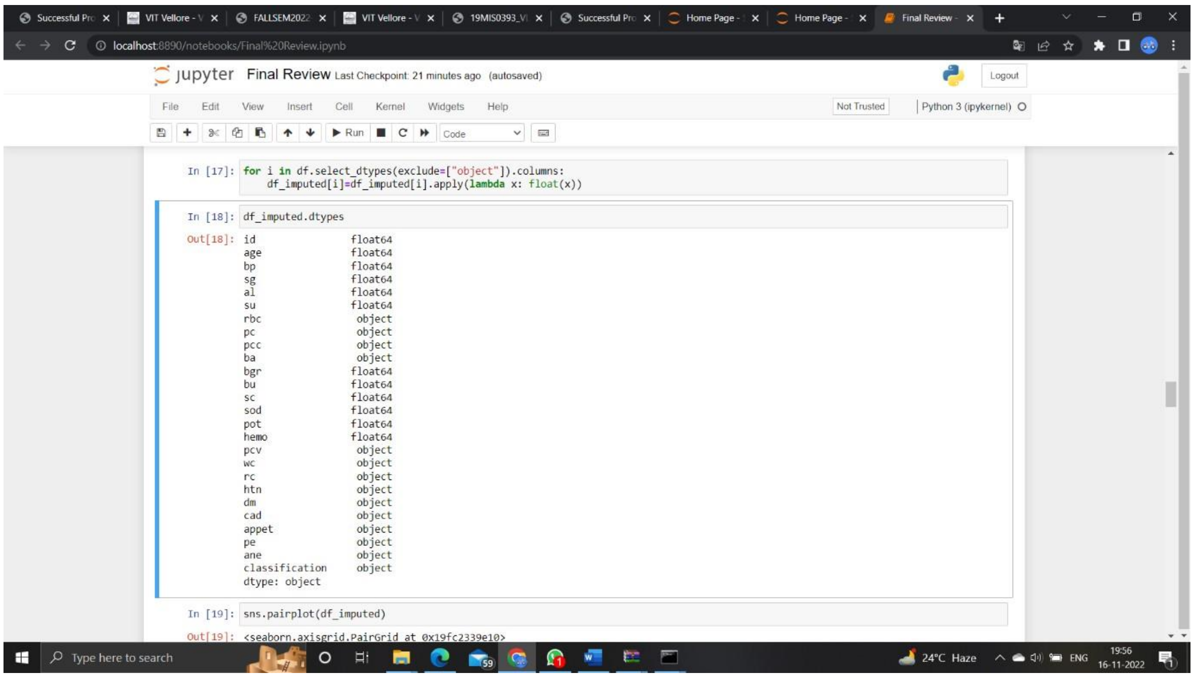


```
In [14]: #Check Label Imbalance

         import matplotlib.pyplot as plt
         import seaborn as sns

         temp=df_imputed["classification"].value_counts()
         temp_df= pd.DataFrame({'classification': temp.index,'values': temp.values})
         print(sns.barplot(x = 'classification', y="values", data=temp_df))

         AxesSubplot(0.125,0.125;0.775x0.755)
```

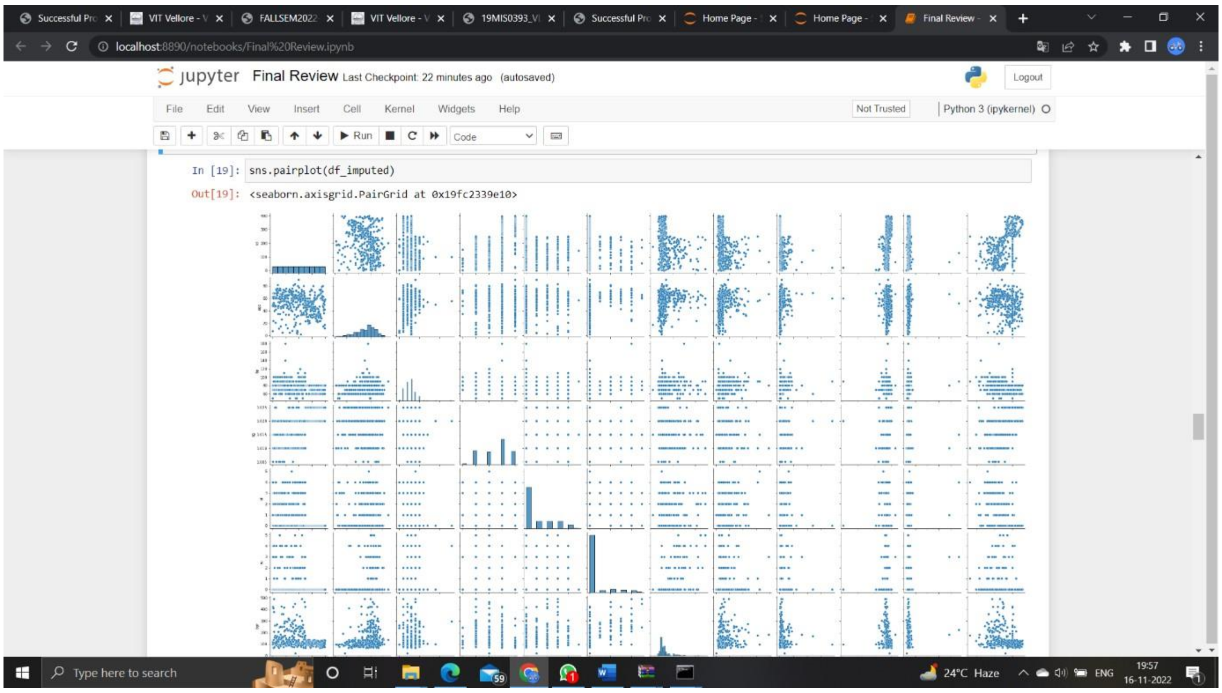
```
In [15]: df.dtypes

Out[15]: id                 int64
         age              float64
         bp               float64
         sg               float64
         al               float64
         su               float64
         rbc               object
         pc                object
```


jupyter Final Review Last Checkpoint: 21 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

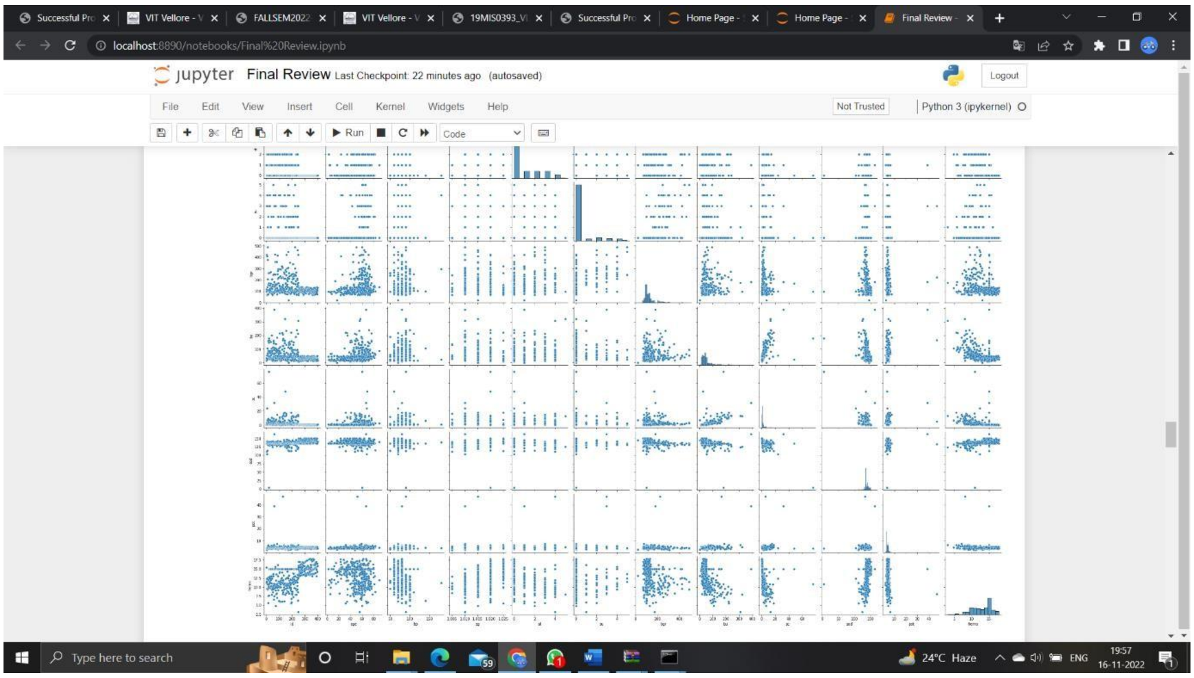
```
In [15]: df.dtypes

Out[15]: id                  int64
         age               float64
         bp                float64
         sg                float64
         al                float64
         su                float64
         rbc                object
         pc                 object
         pcc                object
         ba                 object
         bgr               float64
         bu                float64
         sc                float64
         sod               float64
         pot               float64
         hemo              float64
         pcv                object
         wc                 object
         rc                 object
         htn                object
         dm                 object
         cad                object
         appet              object
         pe                 object
         ane                object
         classification     object
         dtype: object

In [16]: # fixing data types

         df_imputed.dtypes

Out[16]: id                 object
         age                object
```


jupyter Final Review Last Checkpoint: 21 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code

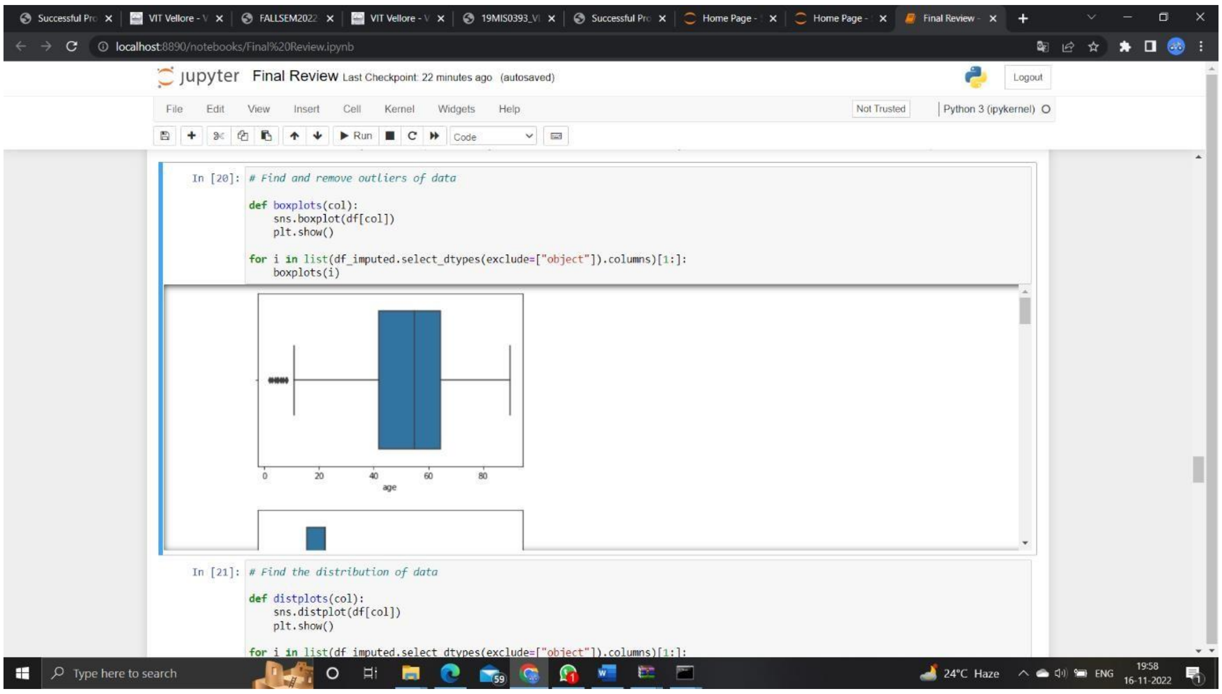
```
In [17]: for i in df.select_dtypes(exclude=["object"]).columns:
             df_imputed[i]=df_imputed[i].apply(lambda x: float(x))

In [18]: df_imputed.dtypes

Out[18]: id                float64
         age               float64
         bp                float64
         sg                float64
         al                float64
         su                float64
         rbc                object
         pc                 object
         pcc                object
         ba                 object
         bgr               float64
         bu                float64
         sc                float64
         sod               float64
         pot               float64
         hemo              float64
         pcv                object
         wc                 object
         rc                 object
         htn                object
         dm                 object
         cad                object
         appet              object
         pe                 object
         ane                object
         classification     object
         dtype: object

In [19]: sns.pairplot(df_imputed)

Out[19]: <seaborn.axisgrid.PairGrid at 0x19fc2339e10>
```



```
In [19]: sns.pairplot(df_imputed)

Out[19]: <seaborn.axisgrid.PairGrid at 0x19fc2339e10>
```



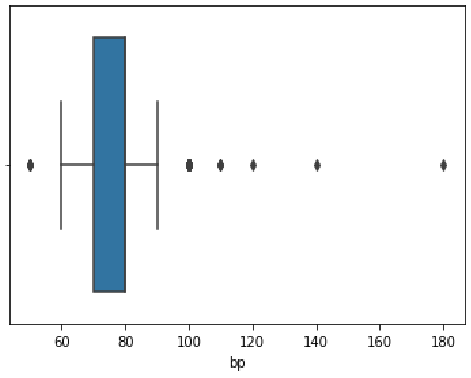
## BOXPLOT:

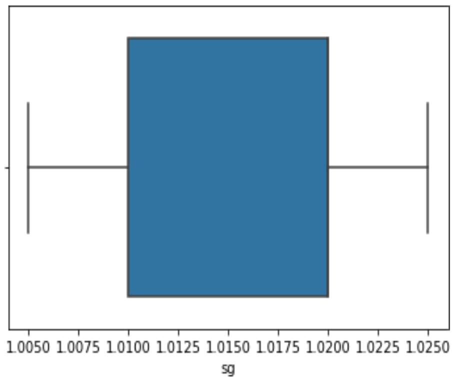


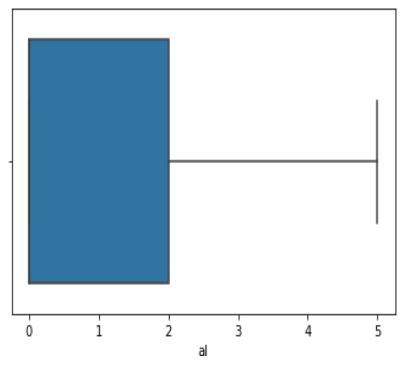
```
In [20]: # Find and remove outliers of data

         def boxplots(col):
             sns.boxplot(df[col])
             plt.show()

         for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
             boxplots(i)
```

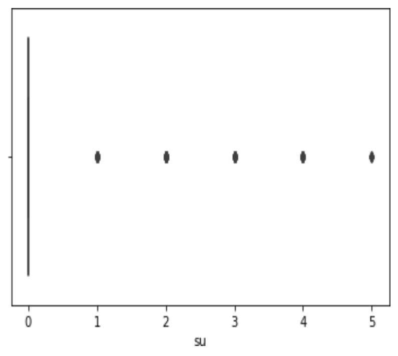
```
In [21]: # Find the distribution of data

         def distplots(col):
             sns.distplot(df[col])
             plt.show()

         for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
```

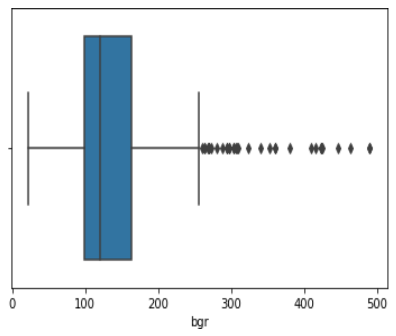
jupyter Final Review Last Checkpoint: 22 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
0
20
40
60
80
age

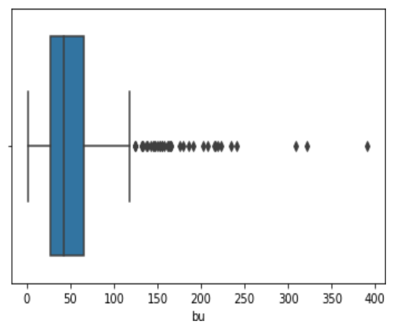


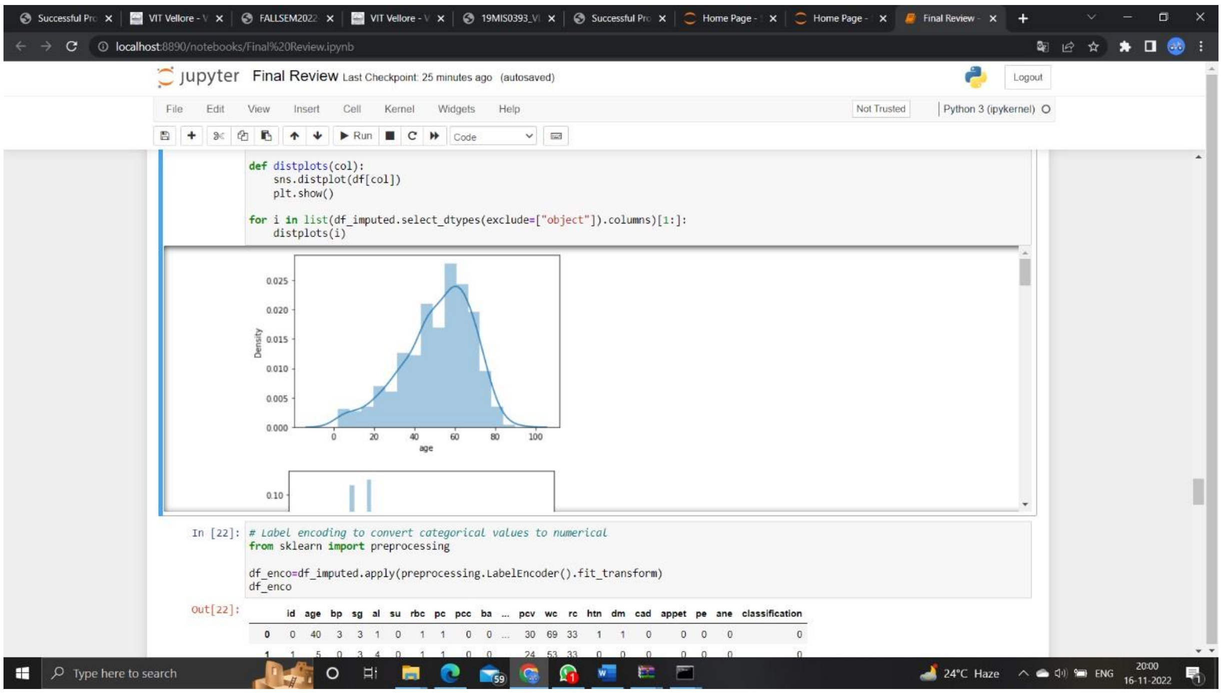



0
1
2
3
4
5
al


0
1
2
3
4
5
su


0
100
200
300
400
500
bgr


0
50
100
150
200
250
300
350
400
bu



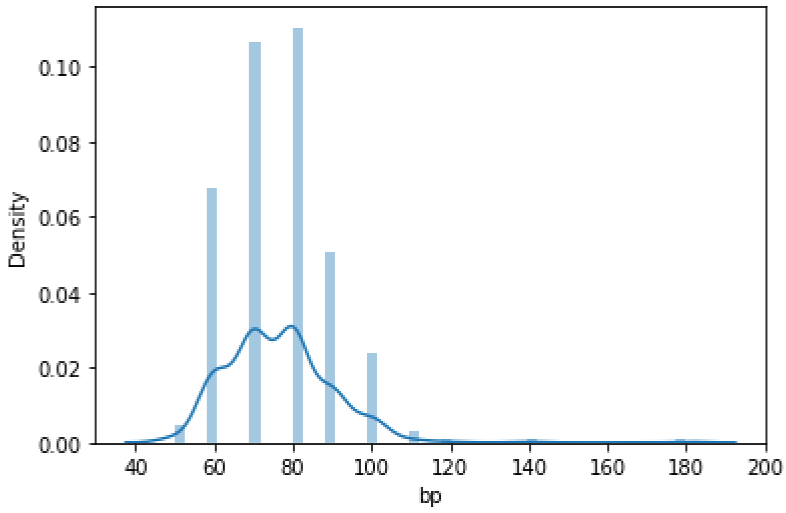
jupyter Final Review Last Checkpoint: 25 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Logout

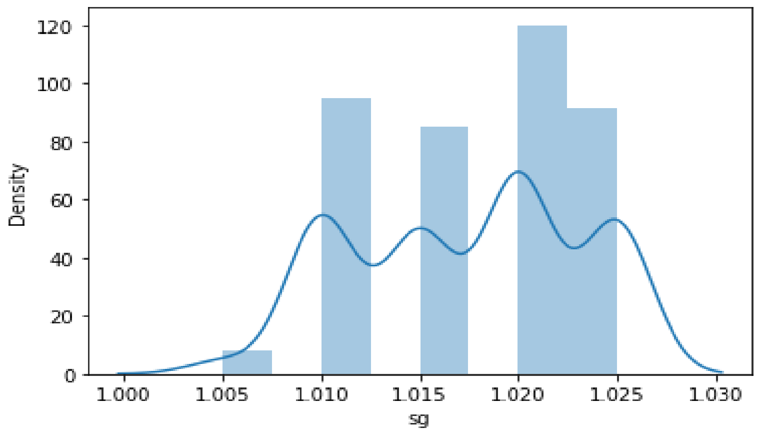
```
def distplots(col):
    sns.distplot(df[col])
    plt.show()

for i in list(df_imputed.select_dtypes(exclude=["object"]).columns)[1:]:
    distplots(i)
```

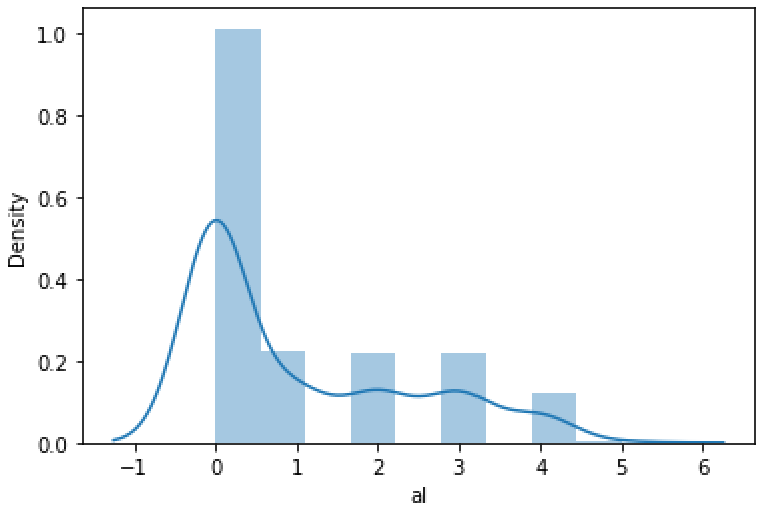
```
In [22]: # Label encoding to convert categorical values to numerical
         from sklearn import preprocessing

         df_enco=df_imputed.apply(preprocessing.LabelEncoder().fit_transform)
         df_enco
```

Out[22]:

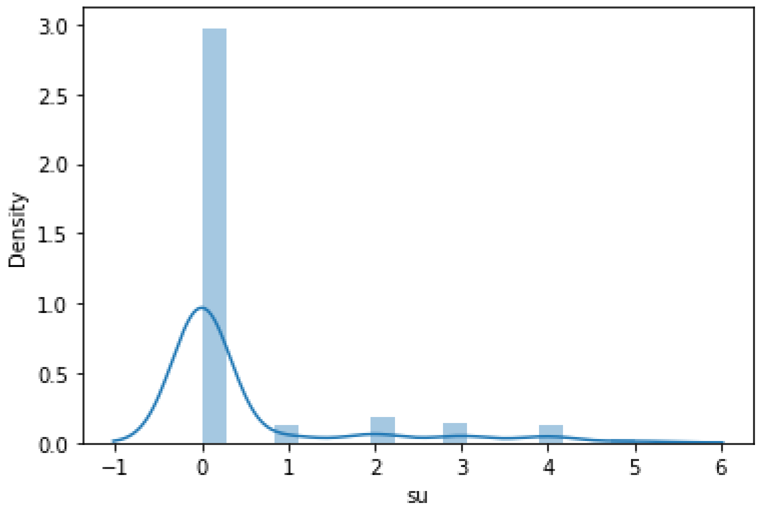
| | id | age | bp | sg | al | su | rbc | pc | pcc | ba | ... | pcv | wc | rc | htn | dm | cad | appet | pe | ane | classification |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 40 | 3 | 3 | 1 | 0 | 1 | 1 | 0 | 0 | ... | 30 | 69 | 33 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 5 | 0 | 3 | 4 | 0 | 1 | 1 | 0 | 0 | ... | 24 | 53 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



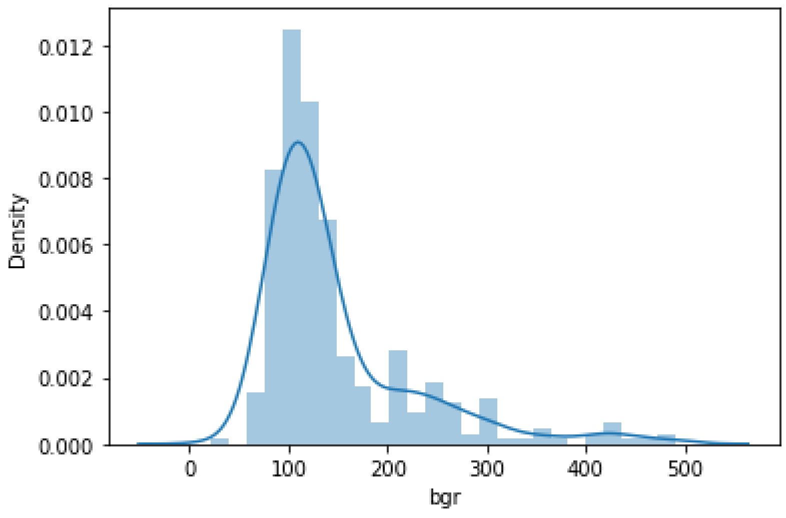




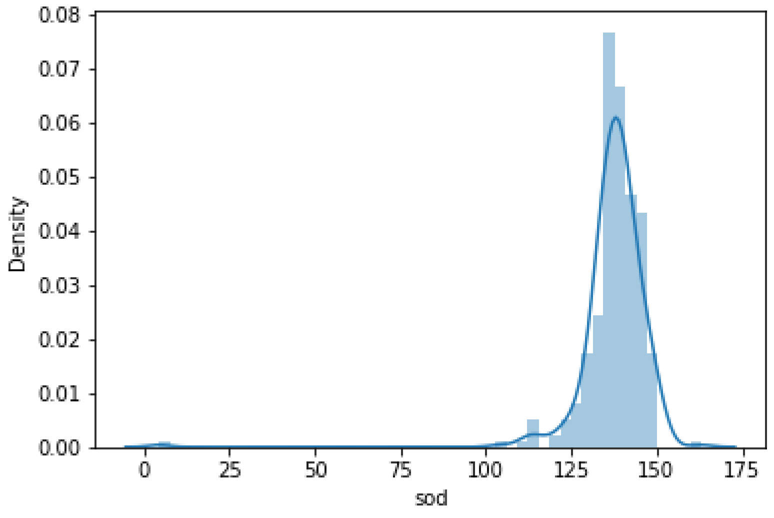
Density
al

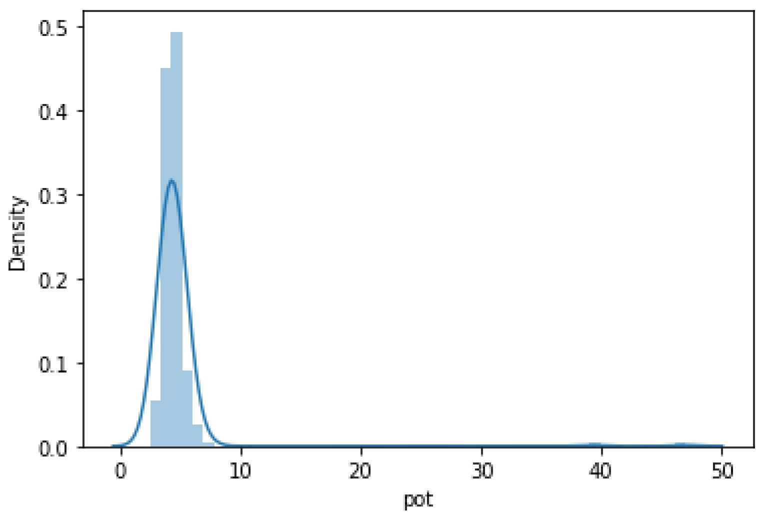


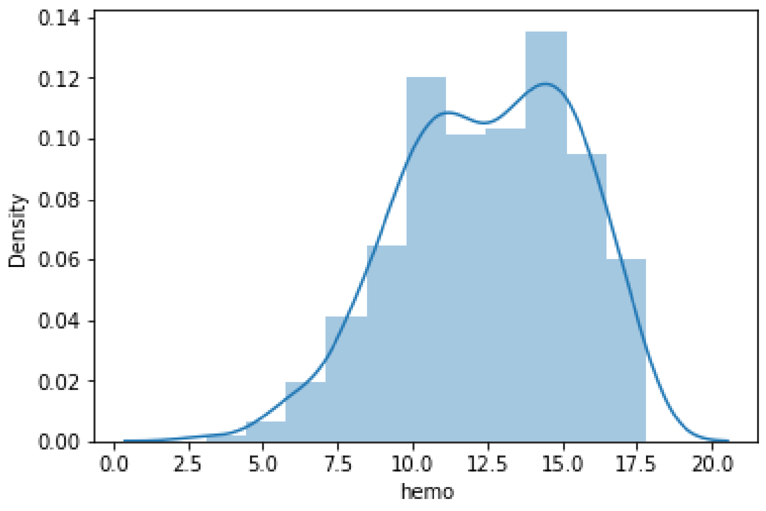
Density
su

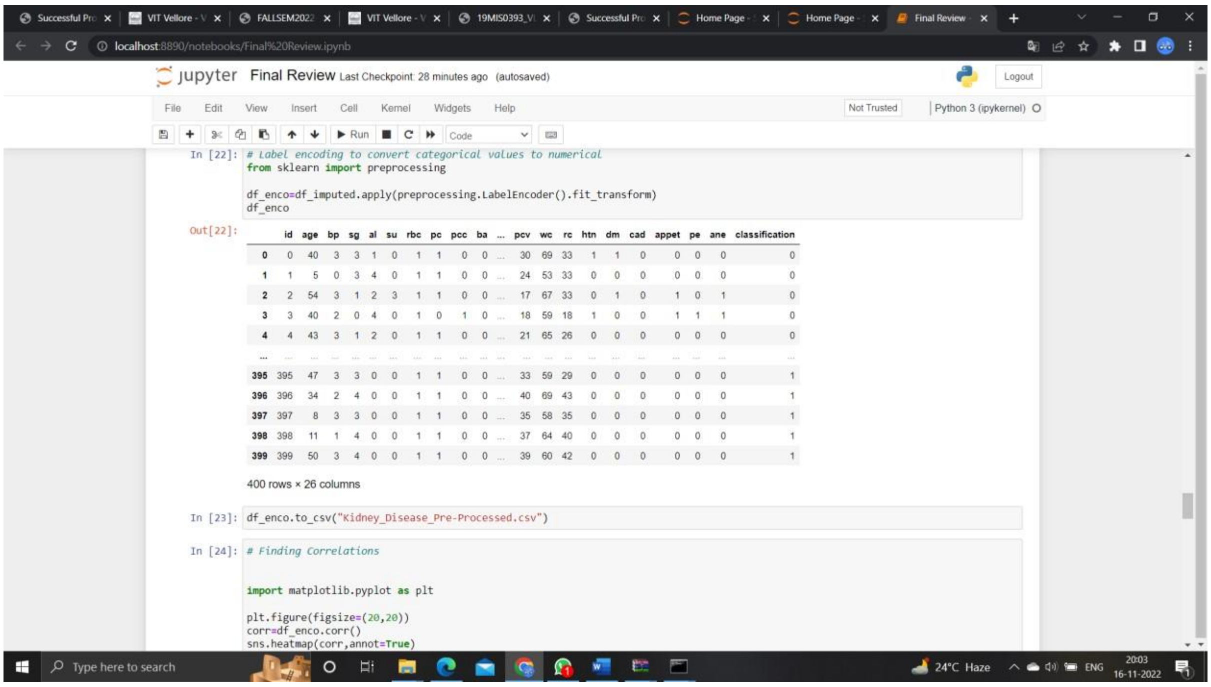


Density
bgr


Density
sod
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
25
50
75
100
125
150
175


Density
pot
0.0
0.1
0.2
0.3
0.4
0.5
0
10
20
30
40
50


Density
hemo
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

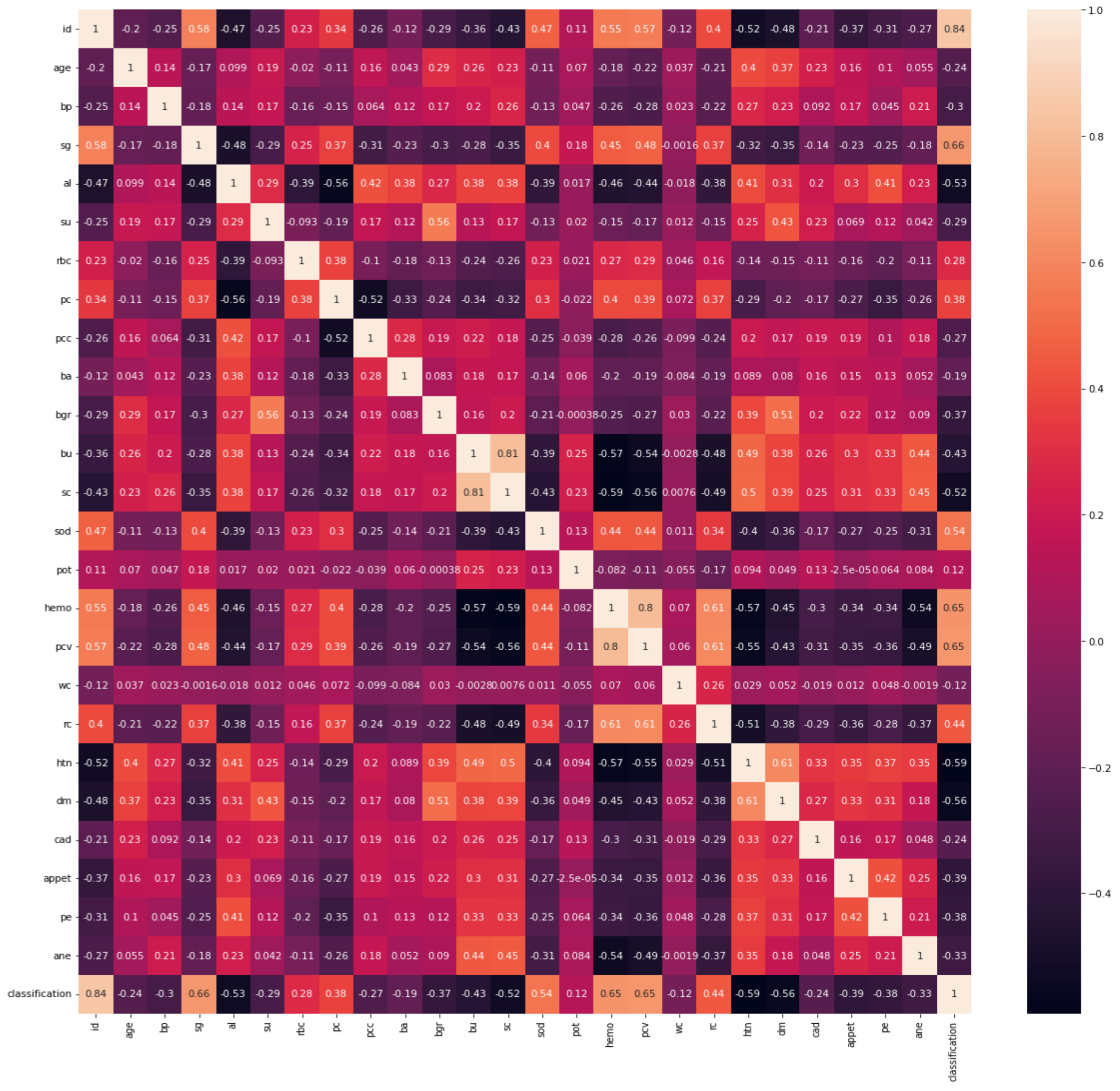


```
In [22]: # Label encoding to convert categorical values to numerical
         from sklearn import preprocessing

         df_enco=df_imputed.apply(preprocessing.LabelEncoder().fit_transform)
         df_enco
```

```
Out[22]:
      id  age  bp  sg  al  su  rbc  pc  pcc  ba  ...  pcv  wc  rc  htn  dm  cad  appet  pe  ane  classification
  0    0   40   3   3   1   0    1   1    0   0  ...   30  69  33    1   1    0      0   0    0               0
  1    1    5   0   3   4   0    1   1    0   0  ...   24  53  33    0   0    0      0   0    0               0
  2    2   54   3   1   2   3    1   1    0   0  ...   17  67  33    0   1    0      1   0    1               0
  3    3   40   2   0   4   0    1   0    1   0  ...   18  59  18    1   0    0      1   1    1               0
  4    4   43   3   1   2   0    1   1    0   0  ...   21  65  26    0   0    0      0   0    0               0
...  ...  ...  ..  ..  ..  ..  ...  ..  ...  ..  ...  ...  ..  ..  ...  ..  ...    ...  ..  ...             ...
395  395   47   3   3   0   0    1   1    0   0  ...   33  59  29    0   0    0      0   0    0               1
396  396   34   2   4   0   0    1   1    0   0  ...   40  69  43    0   0    0      0   0    0               1
397  397    8   3   3   0   0    1   1    0   0  ...   35  58  35    0   0    0      0   0    0               1
398  398   11   1   4   0   0    1   1    0   0  ...   37  64  40    0   0    0      0   0    0               1
399  399   50   3   4   0   0    1   1    0   0  ...   39  60  42    0   0    0      0   0    0               1

400 rows × 26 columns
```

```
In [23]: df_enco.to_csv("Kidney_Disease_Pre-Processed.csv")
```

```
In [24]: # Finding Correlations


         import matplotlib.pyplot as plt

         plt.figure(figsize=(20,20))
         corr=df_enco.corr()
         sns.heatmap(corr,annot=True)
```



## 4. Data Mining Techniques Applied:

**Type of Classification:** Binary Classification

**Algorithms Used:**

- Decision Tree
- Naïve Bayes
- K-NN Classifier
- Random Forest

### 4.1 Description about the Technique

**Classification**

Classification is the processing of finding a set of models (or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The determined model depends on the investigation of a set of training data information (i.e. data objects whose class label is known). The derived model may be represented in various forms, such as classification (if – then) rules, decision trees, and neural networks.

Data Mining has a different type of classifier:

- Decision Tree
- Naïve Bayes
- K-NN Classifier
- Random Forest

**Decision Tree:**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed based on features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

**Naïve Bayes:**

Bayesian classifier is a statistical classifier. They can predict class membership probabilities, for instance, the probability that a given sample belongs to a particular class. Bayesian classification is created on the Bayes theorem. Studies comparing the classification algorithms have found a simple Bayesian classifier known as the naive Bayesian classifier to be comparable in performance with decision tree and neural network classifiers. Bayesian classifiers have also displayed high accuracy and speed when applied to large databases. Naive Bayesian classifiers adopt that the exact attribute value on a given class is independent of the values of the other attributes. This assumption is termed class conditional independence. It is made to simplify the calculations involved, and is considered “naive”. Bayesian belief networks are graphical replicas, which unlike naive Bayesian classifiers allow the depiction of dependencies among subsets of attributes. Bayesian belief can also be utilized for classification.

**K-Nearest Neighbour (K-NN) classifier:**

The k-nearest neighbour (K-NN) classifier is considered as an example-based classifier, which means that the training documents are used for comparison instead of an exact class illustration, like the class profiles utilized by other classifiers. As such, there’s no real training section. once a new document must be classified, the k most similar documents (neighbours) are found and if a large enough proportion of them are allotted to a precise class, the new document is also appointed to the present class, otherwise not. Additionally, finding the closest neighbours is quickened using traditional classification strategies.

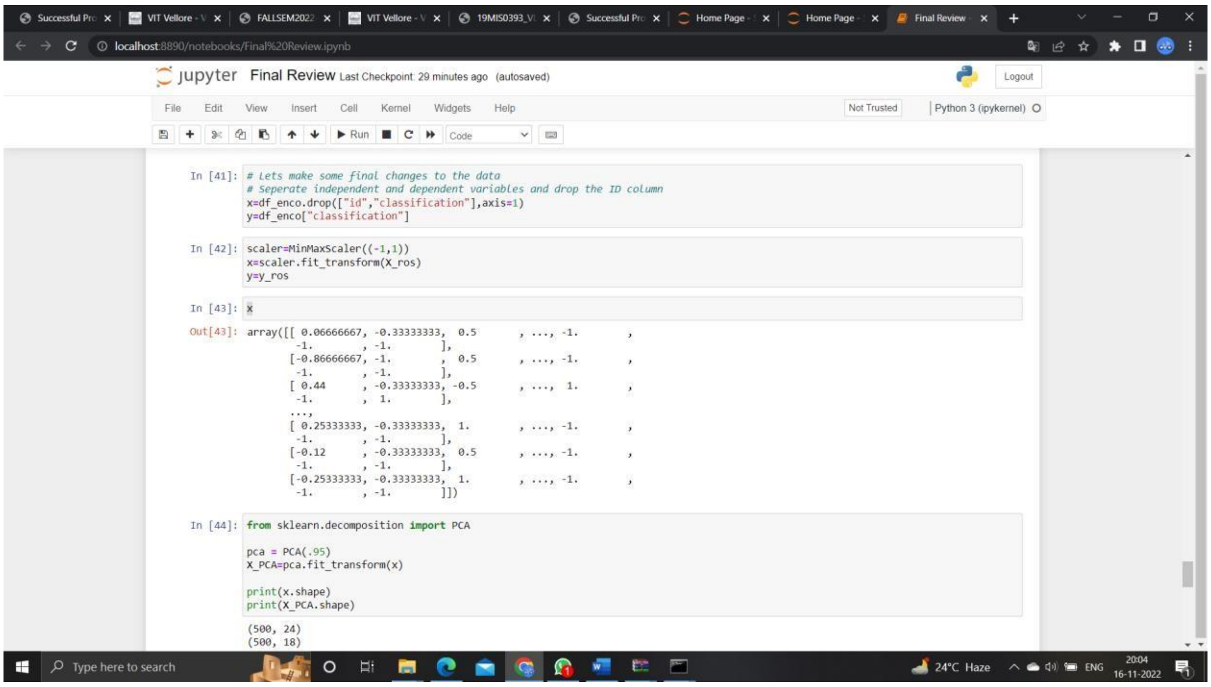
**Random Forest:**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

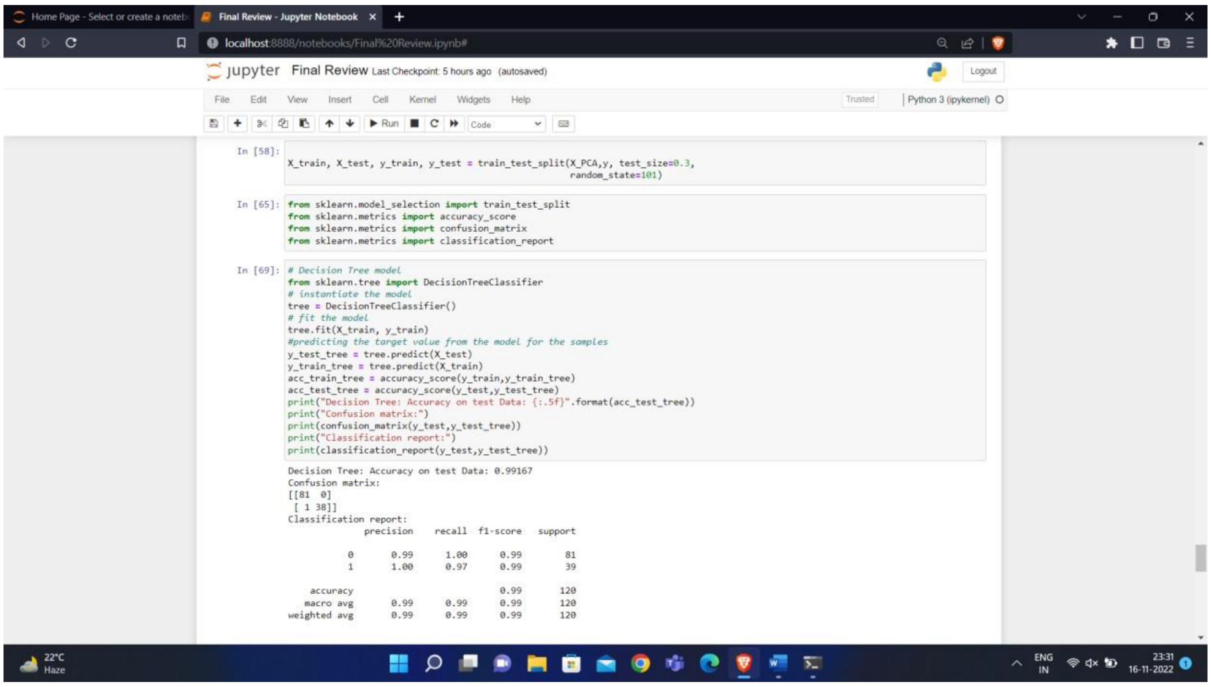
## 4.2 Code

```
# Seperate independent and dependent variables and drop the ID column
x=df_enco.drop(["id","classification"],axis=1)
y=df_enco["classification"]
scaler=MinMaxScaler((-1,1))
x=scaler.fit_transform(X_ros)
y=y_ros
from sklearn.decomposition import PCA
pca = PCA(.95)
X_PCA=pca.fit_transform(x)
print(x.shape)
print(X_PCA.shape)
X_train, X_test, y_train, y_test = train_test_split(X_ros,y_ros, test_size=0.30,
                                                    random_state=101)
from sklearn.metrics import accuracy_score
```

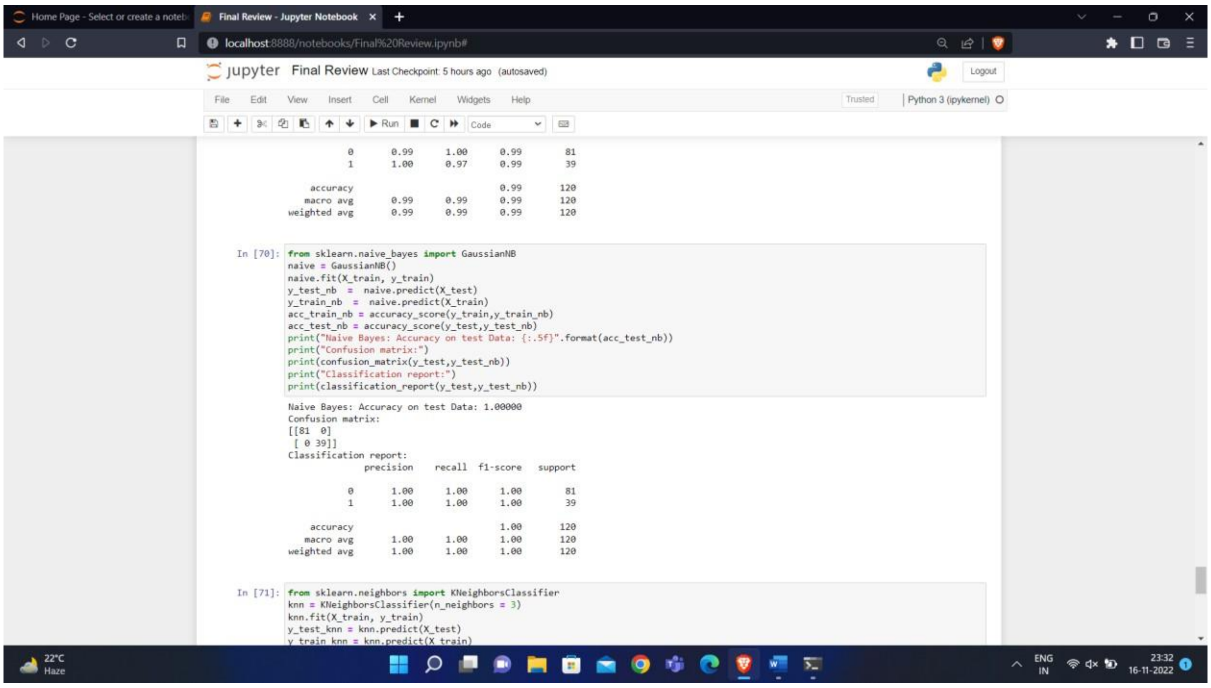
```
# Decision Tree model
from sklearn.tree import DecisionTreeClassifier
# instantiate the model
tree = DecisionTreeClassifier()
# fit the model
tree.fit(X_train, y_train)
#predicting the target value from the model for the samples
y_test_tree = tree.predict(X_test)
y_train_tree = tree.predict(X_train)
acc_train_tree = accuracy_score(y_train,y_train_tree)
acc_test_tree = accuracy_score(y_test,y_test_tree)
print("Decision Tree: Accuracy on training Data: {:.5f}".format(acc_train_tree))
print("Decision Tree: Accuracy on test Data: {:.5f}".format(acc_test_tree))
from sklearn.naive_bayes import GaussianNB
naive = GaussianNB()
naive.fit(X_train, y_train)
y_test_nb  =  naive.predict(X_test)
y_train_nb  =  naive.predict(X_train)
acc_train_nb = accuracy_score(y_train,y_train_nb)
acc_test_nb = accuracy_score(y_test,y_test_nb)
print("Naive Bayes: Accuracy on training Data: {:.5f}".format(acc_train_nb))
print("Naive Bayes: Accuracy on test Data: {:.5f}".format(acc_test_nb))
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
y_test_knn = knn.predict(X_test)
y_train_knn = knn.predict(X_train)
acc_train_knn = accuracy_score(y_train,y_train_knn)
acc_test_knn = accuracy_score(y_test,y_test_knn)
print("KNN: Accuracy on training Data: {:.5f}".format(acc_train_knn))
print("KNN: Accuracy on test Data: {:.5f}".format(acc_test_knn))
from sklearn.ensemble import RandomForestClassifier
rnd = RandomForestClassifier(n_estimators=5)
rnd.fit(X_train, y_train)
y_test_rnd = rnd.predict(X_test)
y_train_rnd = rnd.predict(X_train)
acc_train_rnd = accuracy_score(y_train,y_train_rnd)
acc_test_rnd = accuracy_score(y_test,y_test_rnd)
print("RandomForest: Accuracy on training Data: {:.5f}".format(acc_train_rnd))
print("RandomForest: Accuracy on test Data: {:.5f}".format(acc_test_rnd))
```



```
In [41]: # Lets make some final changes to the data
         # Seperate independent and dependent variables and drop the ID column
         x=df_enco.drop(["id","classification"],axis=1)
         y=df_enco["classification"]

In [42]: scaler=MinMaxScaler((-1,1))
         x=scaler.fit_transform(X_ros)
         y=y_ros

In [43]: x

Out[43]: array([[ 0.06666667, -0.33333333,  0.5       , ..., -1.        ,
                 -1.        , -1.        ],
                [-0.86666667, -1.        ,  0.5       , ..., -1.        ,
                 -1.        , -1.        ],
                [ 0.44      , -0.33333333, -0.5       , ...,  1.        ,
                 -1.        ,  1.        ],
                ...,
                [ 0.25333333, -0.33333333,  1.        , ..., -1.        ,
                 -1.        , -1.        ],
                [-0.12      , -0.33333333,  0.5       , ..., -1.        ,
                 -1.        , -1.        ],
                [-0.25333333, -0.33333333,  1.        , ..., -1.        ,
                 -1.        , -1.        ]])

In [44]: from sklearn.decomposition import PCA

         pca = PCA(.95)
         X_PCA=pca.fit_transform(x)

         print(x.shape)
         print(X_PCA.shape)

         (500, 24)
         (500, 18)
```



```
In [58]:
         X_train, X_test, y_train, y_test = train_test_split(X_PCA,y, test_size=0.3,
                                                             random_state=101)

In [65]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import classification_report

In [69]: # Decision Tree model
         from sklearn.tree import DecisionTreeClassifier
         # instantiate the model
         tree = DecisionTreeClassifier()
         # fit the model
         tree.fit(X_train, y_train)
         #predicting the target value from the model for the samples
         y_test_tree = tree.predict(X_test)
         y_train_tree = tree.predict(X_train)
         acc_train_tree = accuracy_score(y_train,y_train_tree)
         acc_test_tree = accuracy_score(y_test,y_test_tree)
         print("Decision Tree: Accuracy on test Data: {:.5f}".format(acc_test_tree))
         print("Confusion matrix:")
         print(confusion_matrix(y_test,y_test_tree))
         print("Classification report:")
         print(classification_report(y_test,y_test_tree))

         Decision Tree: Accuracy on test Data: 0.99167
         Confusion matrix:
         [[81  0]
          [ 1 38]]
         Classification report:
                       precision    recall  f1-score   support

                    0       0.99      1.00      0.99        81
                    1       1.00      0.97      0.99        39

             accuracy                           0.99       120
            macro avg       0.99      0.99      0.99       120
         weighted avg       0.99      0.99      0.99       120
```

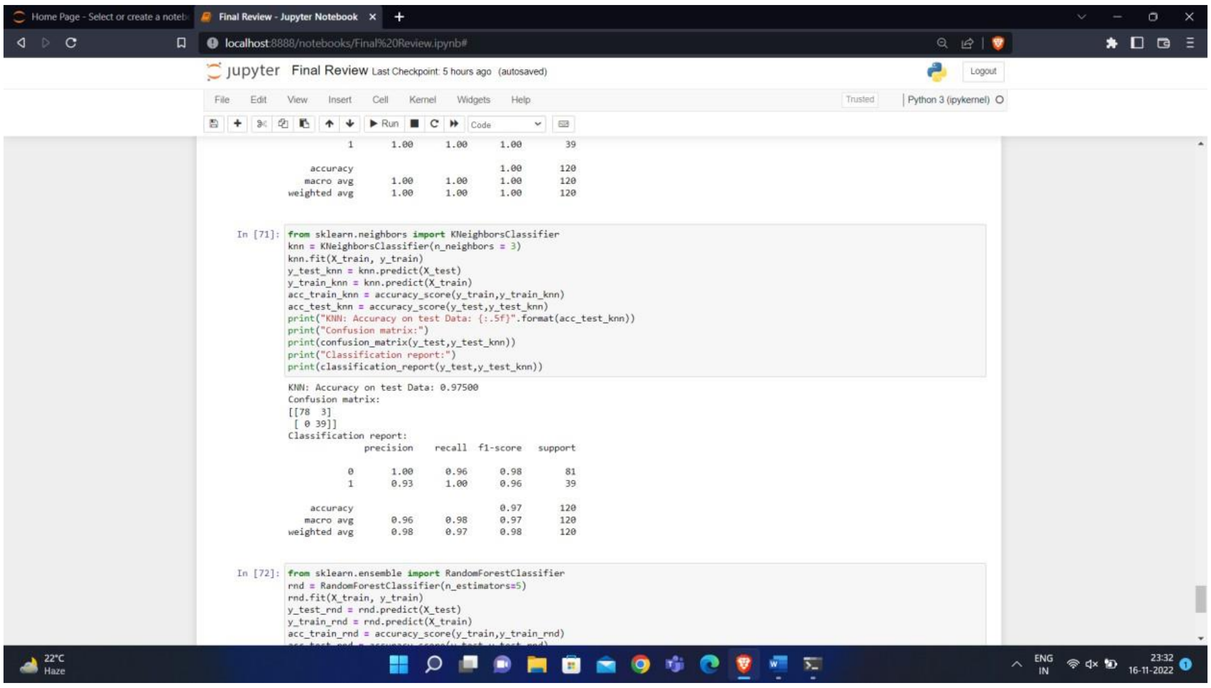


```
              0       0.99      1.00      0.99        81
              1       1.00      0.97      0.99        39

       accuracy                           0.99       120
      macro avg       0.99      0.99      0.99       120
   weighted avg       0.99      0.99      0.99       120
```

```
In [70]: from sklearn.naive_bayes import GaussianNB
         naive = GaussianNB()
         naive.fit(X_train, y_train)
         y_test_nb  =  naive.predict(X_test)
         y_train_nb  =  naive.predict(X_train)
         acc_train_nb = accuracy_score(y_train,y_train_nb)
         acc_test_nb = accuracy_score(y_test,y_test_nb)
         print("Naive Bayes: Accuracy on test Data: {:.5f}".format(acc_test_nb))
         print("Confusion matrix:")
         print(confusion_matrix(y_test,y_test_nb))
         print("Classification report:")
         print(classification_report(y_test,y_test_nb))
```

```
Naive Bayes: Accuracy on test Data: 1.00000
Confusion matrix:
[[81  0]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

              0       1.00      1.00      1.00        81
              1       1.00      1.00      1.00        39

       accuracy                           1.00       120
      macro avg       1.00      1.00      1.00       120
   weighted avg       1.00      1.00      1.00       120
```

```
In [71]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors = 3)
         knn.fit(X_train, y_train)
         y_test_knn = knn.predict(X_test)
         y_train_knn = knn.predict(X_train)
```

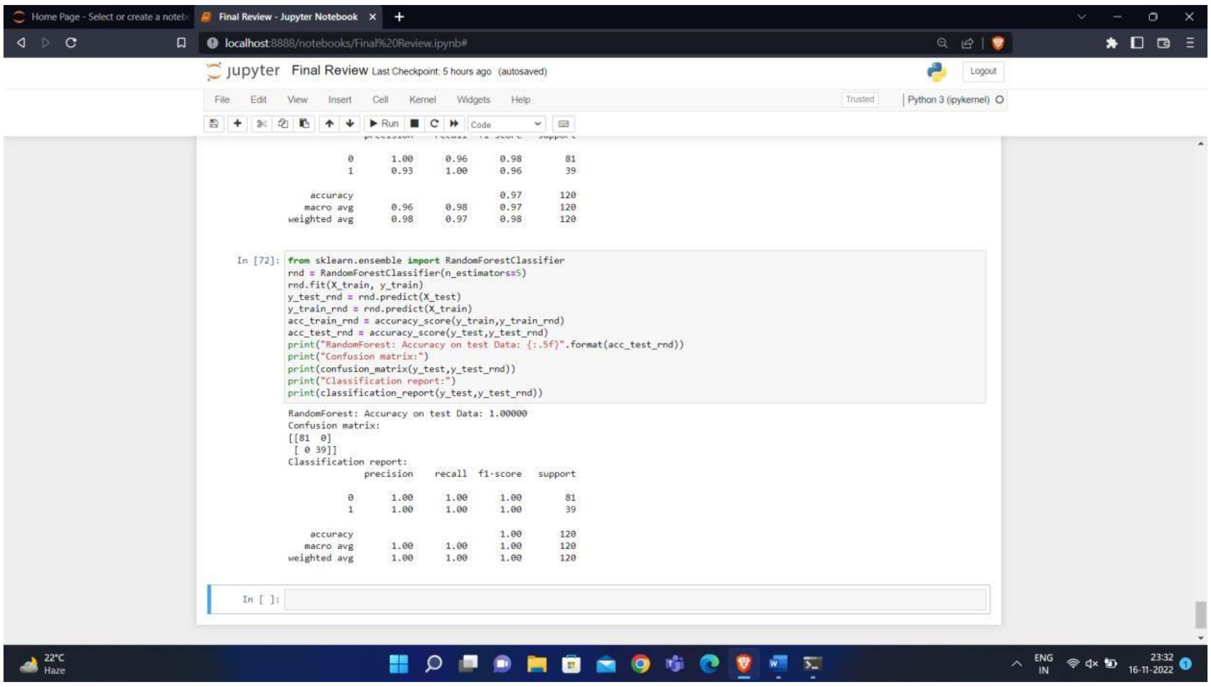


```
              1       1.00      1.00      1.00        39

       accuracy                           1.00       120
      macro avg       1.00      1.00      1.00       120
   weighted avg       1.00      1.00      1.00       120
```

```
In [71]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors = 3)
         knn.fit(X_train, y_train)
         y_test_knn = knn.predict(X_test)
         y_train_knn = knn.predict(X_train)
         acc_train_knn = accuracy_score(y_train,y_train_knn)
         acc_test_knn = accuracy_score(y_test,y_test_knn)
         print("KNN: Accuracy on test Data: {:.5f}".format(acc_test_knn))
         print("Confusion matrix:")
         print(confusion_matrix(y_test,y_test_knn))
         print("Classification report:")
         print(classification_report(y_test,y_test_knn))
```

```
KNN: Accuracy on test Data: 0.97500
Confusion matrix:
[[78  3]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

              0       1.00      0.96      0.98        81
              1       0.93      1.00      0.96        39

       accuracy                           0.97       120
      macro avg       0.96      0.98      0.97       120
   weighted avg       0.98      0.97      0.98       120
```

```
In [72]: from sklearn.ensemble import RandomForestClassifier
         rnd = RandomForestClassifier(n_estimators=5)
         rnd.fit(X_train, y_train)
         y_test_rnd = rnd.predict(X_test)
         y_train_rnd = rnd.predict(X_train)
         acc_train_rnd = accuracy_score(y_train,y_train_rnd)
```



```
              0       1.00      0.96      0.98        81
              1       0.93      1.00      0.96        39

       accuracy                           0.97       120
      macro avg       0.96      0.98      0.97       120
   weighted avg       0.98      0.97      0.98       120
```

```
In [72]: from sklearn.ensemble import RandomForestClassifier
         rnd = RandomForestClassifier(n_estimators=5)
         rnd.fit(X_train, y_train)
         y_test_rnd = rnd.predict(X_test)
         y_train_rnd = rnd.predict(X_train)
         acc_train_rnd = accuracy_score(y_train,y_train_rnd)
         acc_test_rnd = accuracy_score(y_test,y_test_rnd)
         print("RandomForest: Accuracy on test Data: {:.5f}".format(acc_test_rnd))
         print("Confusion matrix:")
         print(confusion_matrix(y_test,y_test_rnd))
         print("Classification report:")
         print(classification_report(y_test,y_test_rnd))
```

```
RandomForest: Accuracy on test Data: 1.00000
Confusion matrix:
[[81  0]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

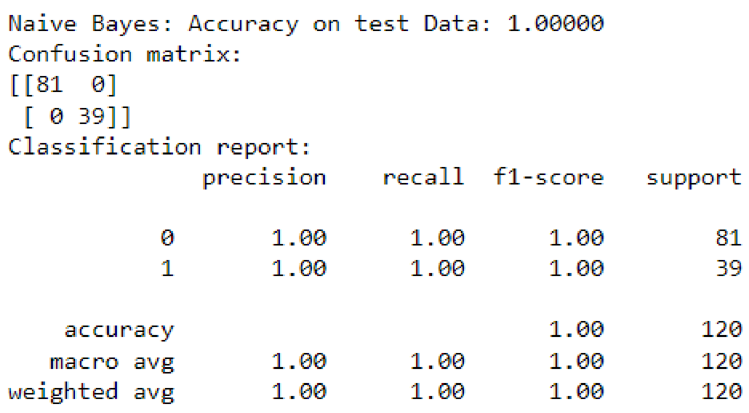
## 5. Results

### 5.1 Evaluation of Error Measures

### Decision Tree:

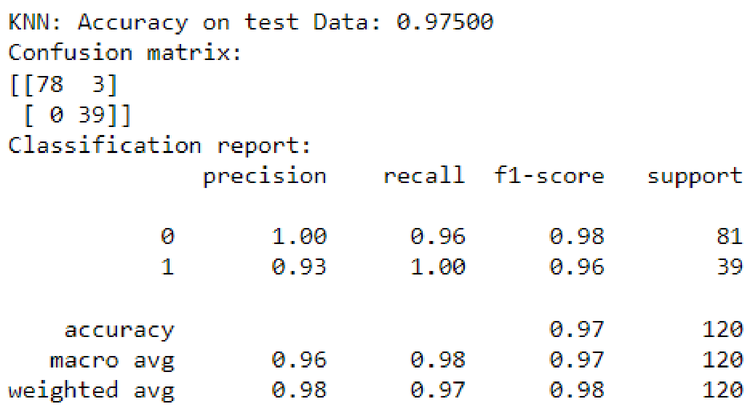
```
Decision Tree: Accuracy on test Data: 0.99167
Confusion matrix:
[[81  0]
 [ 1 38]]
Classification report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        81
           1       1.00      0.97      0.99        39

    accuracy                           0.99       120
   macro avg       0.99      0.99      0.99       120
weighted avg       0.99      0.99      0.99       120
```

## Naïve Bayes:



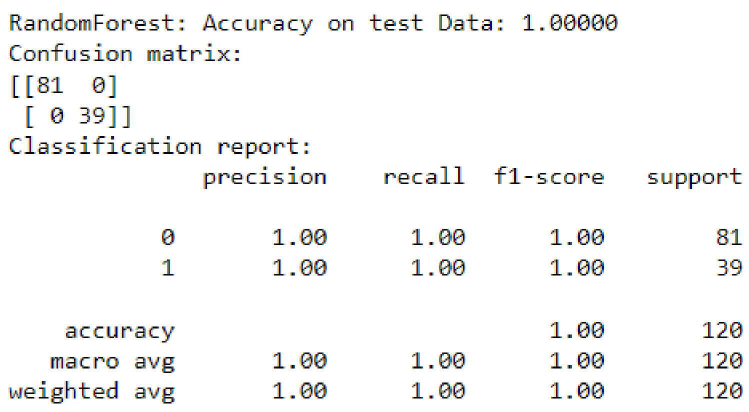
```
Naive Bayes: Accuracy on test Data: 1.00000
Confusion matrix:
[[81  0]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

## KNN:



```
KNN: Accuracy on test Data: 0.97500
Confusion matrix:
[[78  3]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98        81
           1       0.93      1.00      0.96        39

    accuracy                           0.97       120
   macro avg       0.96      0.98      0.97       120
weighted avg       0.98      0.97      0.98       120
```

## Random Forest:



```
RandomForest: Accuracy on test Data: 1.00000
Confusion matrix:
[[81  0]
 [ 0 39]]
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        81
           1       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

## 5.2 Discussion on Results

In this project, we have used different classification models for classification of chronic kidney disease in patients. The four machine learning techniques that were used they are Decision Tree, KNN, Naïve Bayes and Random Forest. The accuracy of different algorithms on the dataset was evaluated. The dataset contains 400 rows. It contains data of 250 not chronic kidney data and 150 chronic kidney disease data. The dataset spitted into training and testing data. Accuracy was evaluated based on TP, TN, FP, FN. The accuracy achieved by naïve bayes and random forest is 100%. This project could be useful in the current medical field with advancement in sciences and new emerging technologies it would be of good help.

## 6. Conclusion

From this we conclude that for the given dataset Random Forest and Naïve Bayes classifier that gives us the best accuracy results. But the same may not be true if a different testing set is used the accuracy will vary. In the real world, if the size of dataset increases by a large amount in the future. Then the overfitting problem of the models will be resolved.

## References

https://archive.ics.uci.edu/ml/datasets/chronic_kidney_disease