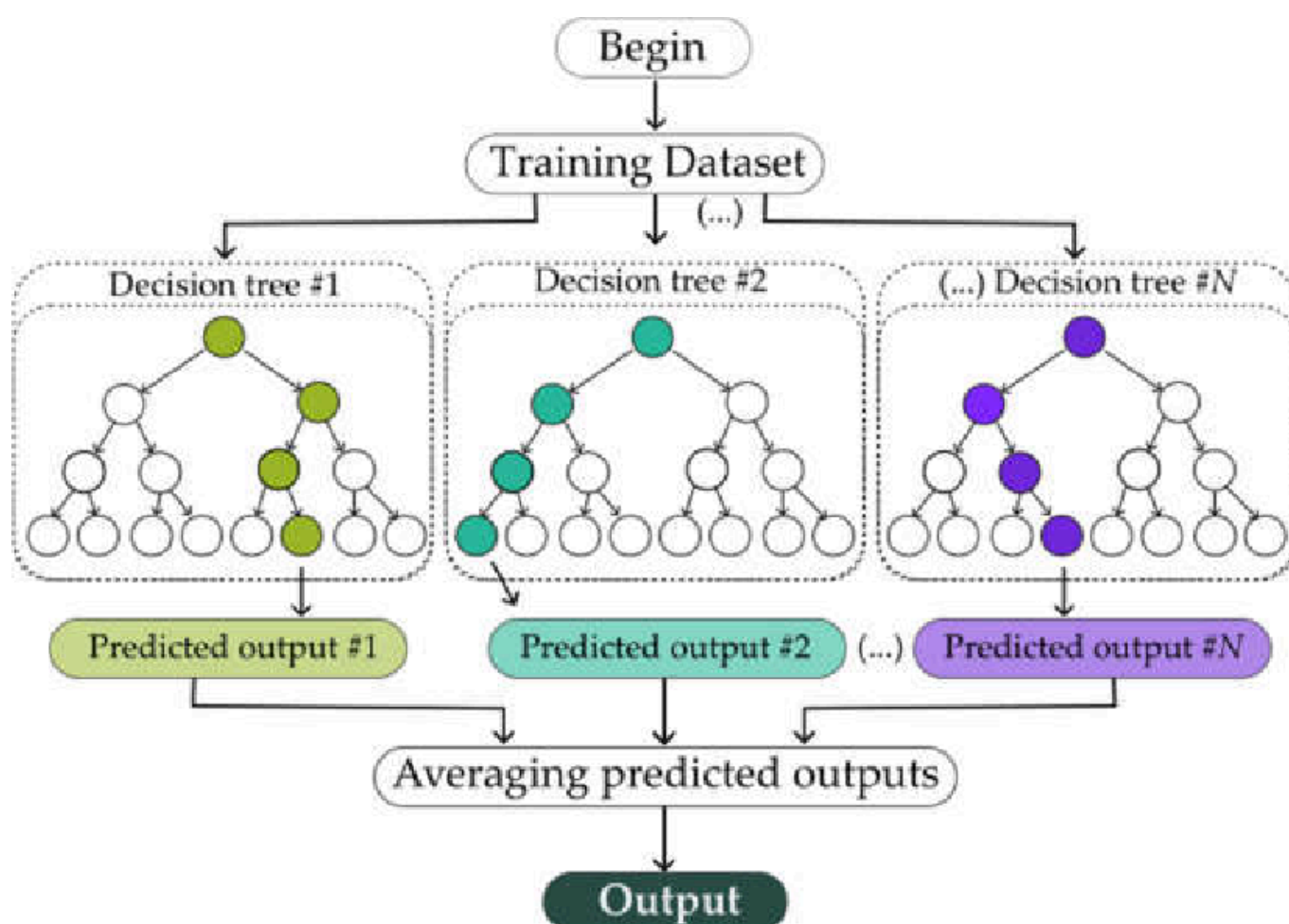


Extra Trees(Extremely Randomized Trees)



- Extra trees (short for extremely randomized trees) is an ensemble supervised machine learning method that uses decision trees.
- Extra Trees is like Random Forest, in that it builds multiple trees and splits nodes using random subsets of features
- The extra trees algorithm creates many decision trees, but the sampling for each tree is random, without replacement. This creates a dataset for each tree with unique samples. A specific number of features, from the total set of features, are also selected randomly for each tree. The most important and unique characteristic of extra trees is the random selection of a splitting value for a feature.
- Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.
- Extra Trees offer a powerful alternative to Random Forests, especially when dealing with high-dimensional data or seeking improved generalizability.
- By leveraging the additional layer of randomness, they create strong ensembles that can effectively tackle complex prediction tasks.
- **Advantages**
 - **Improved Generalizability:** By reducing correlation between trees, Extra Trees often exhibit better generalization capabilities compared to standard Random Forests. This means they perform well on unseen data, making them a good choice for real-world prediction tasks.
 - **Handles High-Dimensional Data:** The feature randomness in Extra Trees can be particularly beneficial for datasets with many features. By considering only a subset at each split, the algorithm can navigate high-dimensional spaces more effectively.
 - **Reduced Overfitting:** Like Random Forests, the additional randomness helps to prevent individual trees from overfitting the training data, leading to more stable and generalizable predictions.
- **Disadvantages**
 - **Computational Cost:** Extra Trees come with a slightly higher computational cost compared to Random Forests due to the added complexity of feature randomness. However, the potential gains in generalization can often outweigh this increase.
 - **Hyperparameter Tuning:** Like Random Forests, Extra Trees require tuning hyperparameters for optimal performance. Experimentation and validation techniques are essential to find the right settings for your specific data.

```
In [1]: ### Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.figure_factory as ff
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')
```

```
In [2]: ### Import the Dataset
df = pd.read_csv(r'C:\Users\hp\Desktop\100DaysOfDataScience\Day 51\kidney_disease.csv', header=0)
df.head()
```

```
Out[2]:
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6

5 rows × 26 columns

```
In [3]: df.shape ### Checking Shape
```

```
Out[3]: (400, 26)
```

```
In [4]: df.describe() ### Get information of the Dataset
```

```
Out[4]:
```

	id	age	bp	sg	al	su	bgr	
count	400.000000	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	3
mean	199.500000	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	
std	115.614301	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	
min	0.000000	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	
25%	99.750000	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	
50%	199.500000	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	
75%	299.250000	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	
max	399.000000	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	3

```
In [5]: df.columns ### Checking Columns
```

```
Out[5]: Index(['id', 'age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bg',
              'bu', 'sc', 'sod', 'pot', 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'c',
              'ad',
              'appet', 'pe', 'ane', 'classification'],
              dtype='object')
```

```
In [6]: df.info() ### Checking Information About a DataFrame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 26 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                     400 non-null    int64
1   age                   391 non-null    float64
2   bp                    388 non-null    float64
3   sg                    353 non-null    float64
4   al                    354 non-null    float64
5   su                    351 non-null    float64
6   rbc                   248 non-null    object
7   pc                    335 non-null    object
8   pcc                   396 non-null    object
9   ba                    396 non-null    object
10  bgr                   356 non-null    float64
11  bu                    381 non-null    float64
12  sc                    383 non-null    float64
13  sod                   313 non-null    float64
14  pot                   312 non-null    float64
15  hemo                  348 non-null    float64
16  pcv                   330 non-null    object
17  wc                    295 non-null    object
18  rc                    270 non-null    object
19  htn                   398 non-null    object
20  dm                    398 non-null    object
21  cad                   398 non-null    object
22  appet                 399 non-null    object
23  pe                    399 non-null    object
24  ane                   399 non-null    object
25  classification        400 non-null    object
dtypes: float64(11), int64(1), object(14)
memory usage: 81.4+ KB
```

```
In [7]: df.isnull().sum() ### Checking Null Values in the Data
```

```
Out[7]: id                0
age                9
bp                12
sg                47
al                46
su                49
rbc              152
pc                65
pcc               4
ba                4
bgr              44
bu               19
sc               17
sod              87
pot              88
hemo             52
pcv              70
wc              105
rc              130
htn               2
dm               2
cad               2
appet            1
pe               1
ane              1
classification    0
dtype: int64
```

```
In [8]: df1 = pd.DataFrame.copy(df)
df1.shape
```

```
Out[8]: (400, 26)
```

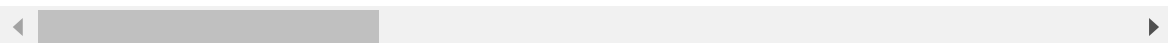
```
In [9]: ### Dropping id column
df1.drop('id', axis = 1, inplace = True)
```

```
In [10]: ### Renaming columns
df1.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']
df1.head()
```

```
Out[10]:
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	nc
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	nc
2	62.0	80.0	1.010	2.0	3.0	normal	normal	nc
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	nc

5 rows × 25 columns



```
In [11]: ### Dropping diabetes_mellitus column
df1.drop('diabetes_mellitus', axis = 1, inplace = True)
```

```
In [12]: for i in df1.columns:  
         print({i:df1[i].unique()}) ### Checking Unique values in each columns
```

```

{'age': array([48., 7., 62., 51., 60., 68., 24., 52., 53., 50., 63., 40.,
47.,
        61., 21., 42., 75., 69., nan, 73., 70., 65., 76., 72., 82., 46.,
        45., 35., 54., 11., 59., 67., 15., 55., 44., 26., 64., 56., 5.,
        74., 38., 58., 71., 34., 17., 12., 43., 41., 57., 8., 39., 66.,
        81., 14., 27., 83., 30., 4., 3., 6., 32., 80., 49., 90., 78.,
        19., 2., 33., 36., 37., 23., 25., 20., 29., 28., 22., 79.]})
{'blood_pressure': array([ 80., 50., 70., 90., nan, 100., 60., 110., 1
40., 180., 120.]})
{'specific_gravity': array([1.02 , 1.01 , 1.005, 1.015, nan, 1.025])})
{'albumin': array([ 1., 4., 2., 3., 0., nan, 5.]})
{'sugar': array([ 0., 3., 4., 1., nan, 2., 5.]})
{'red_blood_cells': array([nan, 'normal', 'abnormal'], dtype=object)}
{'pus_cell': array(['normal', 'abnormal', nan], dtype=object)}
{'pus_cell_clumps': array(['notpresent', 'present', nan], dtype=object)}
{'bacteria': array(['notpresent', 'present', nan], dtype=object)}
{'blood_glucose_random': array([121., nan, 423., 117., 106., 74., 100., 4
10., 138., 70., 490.,
        380., 208., 98., 157., 76., 99., 114., 263., 173., 95., 108.,
        156., 264., 123., 93., 107., 159., 140., 171., 270., 92., 137.,
        204., 79., 207., 124., 144., 91., 162., 246., 253., 141., 182.,
        86., 150., 146., 425., 112., 250., 360., 163., 129., 133., 102.,
        158., 165., 132., 104., 127., 415., 169., 251., 109., 280., 210.,
        219., 295., 94., 172., 101., 298., 153., 88., 226., 143., 115.,
        89., 297., 233., 294., 323., 125., 90., 308., 118., 224., 128.,
        122., 214., 213., 268., 256., 84., 105., 288., 139., 78., 273.,
        242., 424., 303., 148., 160., 192., 307., 220., 447., 309., 22.,
        111., 261., 215., 234., 131., 352., 80., 239., 110., 130., 184.,
        252., 113., 230., 341., 255., 103., 238., 248., 120., 241., 269.,
        201., 203., 463., 176., 82., 119., 97., 96., 81., 116., 134.,
        85., 83., 87., 75.]})
{'blood_urea': array([ 36. , 18. , 53. , 56. , 26. , 25. , 54. , 31.
, 60. ,
        107. , 55. , 72. , 86. , 90. , 162. , 46. , 87. , 27. ,
        148. , 180. , 163. , nan, 50. , 75. , 45. , 28. , 155. ,
        33. , 39. , 153. , 29. , 65. , 103. , 70. , 80. , 20. ,
        202. , 77. , 89. , 24. , 17. , 32. , 114. , 66. , 38. ,
        164. , 142. , 96. , 391. , 15. , 111. , 73. , 19. , 92. ,
        35. , 16. , 139. , 48. , 85. , 98. , 186. , 37. , 47. ,
        52. , 82. , 51. , 106. , 22. , 217. , 88. , 118. , 50.1,
        71. , 34. , 40. , 21. , 219. , 30. , 125. , 166. , 49. ,
        208. , 176. , 68. , 145. , 165. , 322. , 23. , 235. , 132. ,
        76. , 42. , 44. , 41. , 113. , 1.5, 146. , 58. , 133. ,
        137. , 67. , 115. , 223. , 98.6, 158. , 94. , 74. , 150. ,
        61. , 57. , 95. , 191. , 93. , 241. , 64. , 79. , 215. ,
        309. , 10. ]})
{'serum_creatinine': array([ 1.2 , 0.8 , 1.8 , 3.8 , 1.4 , 1.1 , 24.
, 1.9 , 7.2 ,
        4. , 2.7 , 2.1 , 4.6 , 4.1 , 9.6 , 2.2 , 5.2 , 1.3 ,
        1.6 , 3.9 , 76. , 7.7 , nan, 2.4 , 7.3 , 1.5 , 2.5 ,
        2. , 3.4 , 0.7 , 1. , 10.8 , 6.3 , 5.9 , 0.9 , 3. ,
        3.25, 9.7 , 6.4 , 3.2 , 32. , 0.6 , 6.1 , 3.3 , 6.7 ,
        8.5 , 2.8 , 15. , 2.9 , 1.7 , 3.6 , 5.6 , 6.5 , 4.4 ,
        10.2 , 11.5 , 0.5 , 12.2 , 5.3 , 9.2 , 13.8 , 16.9 , 6. ,
        7.1 , 18. , 2.3 , 13. , 48.1 , 14.2 , 16.4 , 2.6 , 7.5 ,
        4.3 , 18.1 , 11.8 , 9.3 , 6.8 , 13.5 , 12.8 , 11.9 , 12. ,
        13.4 , 15.2 , 13.3 , 0.4 ]})
{'sodium': array([ nan, 111. , 142. , 104. , 114. , 131. , 138. , 135. , 1
30. ,
        141. , 139. , 4.5, 136. , 129. , 140. , 132. , 133. , 134. ,
        125. , 163. , 137. , 128. , 143. , 127. , 146. , 126. , 122. ,

```



```

147. , 124. , 115. , 145. , 113. , 120. , 150. , 144. ])}
{'potassium': array([ nan,  2.5,  3.2,  4. ,  3.7,  4.2,  5.8,  3.4,  6.4,
 4.9,  4.1,
 4.3,  5.2,  3.8,  4.6,  3.9,  4.7,  5.9,  4.8,  4.4,  6.6, 39. ,
 5.5,  5. ,  3.5,  3.6,  7.6,  2.9,  4.5,  5.7,  5.4,  5.3, 47. ,
 6.3,  5.1,  5.6,  3. ,  2.8,  2.7,  6.5,  3.3])}
{'haemoglobin': array([15.4, 11.3,  9.6, 11.2, 11.6, 12.2, 12.4, 10.8,  9.
5,  9.4,  9.7,
 9.8,  5.6,  7.6, 12.6, 12.1, 12.7, 10.3,  7.7, 10.9,  nan, 11.1,
 9.9, 12.5, 12.9, 10.1, 12. , 13. ,  7.9,  9.3, 15. , 10. ,  8.6,
13.6, 10.2, 10.5,  6.6, 11. ,  7.5, 15.6, 15.2,  4.8,  9.1,  8.1,
11.9, 13.5,  8.3,  7.1, 16.1, 10.4,  9.2,  6.2, 13.9, 14.1,  6. ,
11.8, 11.7, 11.4, 14. ,  8.2, 13.2,  6.1,  8. , 12.3,  8.4, 14.3,
 9. ,  8.7, 10.6, 13.1, 10.7,  5.5,  5.8,  6.8,  8.8,  8.5, 13.8,
11.5,  7.3, 13.7, 12.8, 13.4,  6.3,  3.1, 17. , 15.9, 14.5, 15.5,
16.2, 14.4, 14.2, 16.3, 14.8, 16.5, 15.7, 13.3, 14.6, 16.4, 16.9,
16. , 14.7, 16.6, 14.9, 16.7, 16.8, 15.8, 15.1, 17.1, 17.2, 15.3,
17.3, 17.4, 17.7, 17.8, 17.5, 17.6])}
{'packed_cell_volume': array(['44', '38', '31', '32', '35', '39', '36', '3
3', '29', '28', nan,
'16', '24', '37', '30', '34', '40', '45', '27', '48', '\t?', '52',
'14', '22', '18', '42', '17', '46', '23', '19', '25', '41', '26',
'15', '21', '43', '20', '\t43', '47', '9', '49', '50', '53', '51',
'54'], dtype=object)}
{'white_blood_cell_count': array(['7800', '6000', '7500', '6700', '7300', n
an, '6900', '9600',
'12100', '4500', '12200', '11000', '3800', '11400', '5300', '9200',
'6200', '8300', '8400', '10300', '9800', '9100', '7900', '6400',
'8600', '18900', '21600', '4300', '8500', '11300', '7200', '7700',
'14600', '6300', '\t6200', '7100', '11800', '9400', '5500', '5800',
'13200', '12500', '5600', '7000', '11900', '10400', '10700',
'12700', '6800', '6500', '13600', '10200', '9000', '14900', '8200',
'15200', '5000', '16300', '12400', '\t8400', '10500', '4200',
'4700', '10900', '8100', '9500', '2200', '12800', '11200', '19100',
'\t?', '12300', '16700', '2600', '26400', '8800', '7400', '4900',
'8000', '12000', '15700', '4100', '5700', '11500', '5400', '10800',
'9900', '5200', '5900', '9300', '9700', '5100', '6600'],
dtype=object)}
{'red_blood_cell_count': array(['5.2', nan, '3.9', '4.6', '4.4', '5', '4.
0', '3.7', '3.8', '3.4',
'2.6', '2.8', '4.3', '3.2', '3.6', '4', '4.1', '4.9', '2.5', '4.2',
'4.5', '3.1', '4.7', '3.5', '6.0', '5.0', '2.1', '5.6', '2.3',
'2.9', '2.7', '8.0', '3.3', '3.0', '3', '2.4', '4.8', '\t?', '5.4',
'6.1', '6.2', '6.3', '5.1', '5.8', '5.5', '5.3', '6.4', '5.7',
'5.9', '6.5'], dtype=object)}
{'hypertension': array(['yes', 'no', nan], dtype=object)}
{'coronary_artery_disease': array(['no', 'yes', '\tno', nan], dtype=objec
t)}
{'appetite': array(['good', 'poor', nan], dtype=object)}
{'peda_edema': array(['no', 'yes', nan], dtype=object)}
{'aanemia': array(['no', 'yes', nan], dtype=object)}
{'class': array(['ckd', 'ckd\t', 'notckd'], dtype=object)}

```

```
In [13]: ### Finding numerical variables
colname_num = [var for var in df1.columns if df1[var].dtype!='O']
print('There are {} numerical variables\n'.format(len(colname_num)))
print('The numerical variables are :', colname_num)
```

There are 11 numerical variables

The numerical variables are : ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium', 'potassium', 'haemoglobin']

```
In [14]: # Looking at unique values in numerical columns
for col in colname_num:
    print(f"{col} has {df1[col].unique()} values\n")
```

age has [48. 7. 62. 51. 60. 68. 24. 52. 53. 50. 63. 40. 47. 61. 21. 42. 7
5. 69.
nan 73. 70. 65. 76. 72. 82. 46. 45. 35. 54. 11. 59. 67. 15. 55. 44. 26.
64. 56. 5. 74. 38. 58. 71. 34. 17. 12. 43. 41. 57. 8. 39. 66. 81. 14.
27. 83. 30. 4. 3. 6. 32. 80. 49. 90. 78. 19. 2. 33. 36. 37. 23. 25.
20. 29. 28. 22. 79.] values

blood_pressure has [80. 50. 70. 90. nan 100. 60. 110. 140. 180. 120.]
values

specific_gravity has [1.02 1.01 1.005 1.015 nan 1.025] values

albumin has [1. 4. 2. 3. 0. nan 5.] values

sugar has [0. 3. 4. 1. nan 2. 5.] values

blood_glucose_random has [121. nan 423. 117. 106. 74. 100. 410. 138. 70.
490. 380. 208. 98.
157. 76. 99. 114. 263. 173. 95. 108. 156. 264. 123. 93. 107. 159.
140. 171. 270. 92. 137. 204. 79. 207. 124. 144. 91. 162. 246. 253.
141. 182. 86. 150. 146. 425. 112. 250. 360. 163. 129. 133. 102. 158.
165. 132. 104. 127. 415. 169. 251. 109. 280. 210. 219. 295. 94. 172.
101. 298. 153. 88. 226. 143. 115. 89. 297. 233. 294. 323. 125. 90.
308. 118. 224. 128. 122. 214. 213. 268. 256. 84. 105. 288. 139. 78.
273. 242. 424. 303. 148. 160. 192. 307. 220. 447. 309. 22. 111. 261.
215. 234. 131. 352. 80. 239. 110. 130. 184. 252. 113. 230. 341. 255.
103. 238. 248. 120. 241. 269. 201. 203. 463. 176. 82. 119. 97. 96.
81. 116. 134. 85. 83. 87. 75.] values

blood_urea has [36. 18. 53. 56. 26. 25. 54. 31. 60. 107.
55. 72.
86. 90. 162. 46. 87. 27. 148. 180. 163. nan 50. 75.
45. 28. 155. 33. 39. 153. 29. 65. 103. 70. 80. 20.
202. 77. 89. 24. 17. 32. 114. 66. 38. 164. 142. 96.
391. 15. 111. 73. 19. 92. 35. 16. 139. 48. 85. 98.
186. 37. 47. 52. 82. 51. 106. 22. 217. 88. 118. 50.1
71. 34. 40. 21. 219. 30. 125. 166. 49. 208. 176. 68.
145. 165. 322. 23. 235. 132. 76. 42. 44. 41. 113. 1.5
146. 58. 133. 137. 67. 115. 223. 98.6 158. 94. 74. 150.
61. 57. 95. 191. 93. 241. 64. 79. 215. 309. 10.] values

serum_creatinine has [1.2 0.8 1.8 3.8 1.4 1.1 24. 1.9 7.2
4. 2.7 2.1
4.6 4.1 9.6 2.2 5.2 1.3 1.6 3.9 76. 7.7 nan 2.4
7.3 1.5 2.5 2. 3.4 0.7 1. 10.8 6.3 5.9 0.9 3.
3.25 9.7 6.4 3.2 32. 0.6 6.1 3.3 6.7 8.5 2.8 15.
2.9 1.7 3.6 5.6 6.5 4.4 10.2 11.5 0.5 12.2 5.3 9.2
13.8 16.9 6. 7.1 18. 2.3 13. 48.1 14.2 16.4 2.6 7.5
4.3 18.1 11.8 9.3 6.8 13.5 12.8 11.9 12. 13.4 15.2 13.3
0.4] values

sodium has [nan 111. 142. 104. 114. 131. 138. 135. 130. 141. 13
9. 4.5
136. 129. 140. 132. 133. 134. 125. 163. 137. 128. 143. 127.
146. 126. 122. 147. 124. 115. 145. 113. 120. 150. 144.] values

potassium has [nan 2.5 3.2 4. 3.7 4.2 5.8 3.4 6.4 4.9 4.1 4.3
5.2 3.8
4.6 3.9 4.7 5.9 4.8 4.4 6.6 39. 5.5 5. 3.5 3.6 7.6 2.9
4.5 5.7 5.4 5.3 47. 6.3 5.1 5.6 3. 2.8 2.7 6.5 3.3] values

```

haemoglobin has [15.4 11.3  9.6 11.2 11.6 12.2 12.4 10.8  9.5  9.4  9.7  9.
8  5.6  7.6
12.6 12.1 12.7 10.3  7.7 10.9  nan 11.1  9.9 12.5 12.9 10.1 12. 13.
 7.9  9.3 15.  10.  8.6 13.6 10.2 10.5  6.6 11.  7.5 15.6 15.2  4.8
 9.1  8.1 11.9 13.5  8.3  7.1 16.1 10.4  9.2  6.2 13.9 14.1  6. 11.8
11.7 11.4 14.  8.2 13.2  6.1  8. 12.3  8.4 14.3  9.  8.7 10.6 13.1
10.7  5.5  5.8  6.8  8.8  8.5 13.8 11.5  7.3 13.7 12.8 13.4  6.3  3.1
17. 15.9 14.5 15.5 16.2 14.4 14.2 16.3 14.8 16.5 15.7 13.3 14.6 16.4
16.9 16. 14.7 16.6 14.9 16.7 16.8 15.8 15.1 17.1 17.2 15.3 17.3 17.4
17.7 17.8 17.5 17.6] values

```

```

In [15]: ### Finding categorical variables
colname_cat = [var for var in df1.columns if df1[var].dtype=='O']
print('There are {} categorical variables\n'.format(len(colname_cat)))
print('The categorical variables are :', colname_cat)

```

There are 13 categorical variables

The categorical variables are : ['red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count', 'hypertension', 'coronary_artery_disease', 'appetite', 'peda_edema', 'aanemia', 'class']

```

In [16]: ### Converting necessary columns to numerical type
df1['packed_cell_volume'] = pd.to_numeric(df1['packed_cell_volume'], errors='coerce')
df1['white_blood_cell_count'] = pd.to_numeric(df1['white_blood_cell_count'], errors='coerce')
df1['red_blood_cell_count'] = pd.to_numeric(df1['red_blood_cell_count'], errors='coerce')
df1.dtypes

```

```

Out[16]: age                float64
blood_pressure            float64
specific_gravity          float64
albumin                  float64
sugar                    float64
red_blood_cells           object
pus_cell                  object
pus_cell_clumps           object
bacteria                  object
blood_glucose_random      float64
blood_urea                float64
serum_creatinine          float64
sodium                   float64
potassium                 float64
haemoglobin               float64
packed_cell_volume        float64
white_blood_cell_count    float64
red_blood_cell_count      float64
hypertension              object
coronary_artery_disease   object
appetite                  object
peda_edema                object
aanemia                   object
class                     object
dtype: object

```

```
In [17]: # Looking at unique values in categorical columns
for col in colname_cat:
    print(f"{col} has {df1[col].unique()} values\n")
```

red_blood_cells has [nan 'normal' 'abnormal'] values

pus_cell has ['normal' 'abnormal' nan] values

pus_cell_clumps has ['notpresent' 'present' nan] values

bacteria has ['notpresent' 'present' nan] values

packed_cell_volume has [44. 38. 31. 32. 35. 39. 36. 33. 29. 28. nan 16. 24. 37. 30. 34. 40. 45. 27. 48. 52. 14. 22. 18. 42. 17. 46. 23. 19. 25. 41. 26. 15. 21. 43. 20. 47. 9. 49. 50. 53. 51. 54.] values

white_blood_cell_count has [7800. 6000. 7500. 6700. 7300. nan 6900. 9600. 12100. 4500. 12200. 11000. 3800. 11400. 5300. 9200. 6200. 8300. 8400. 10300. 9800. 9100. 7900. 6400. 8600. 18900. 21600. 4300. 8500. 11300. 7200. 7700. 14600. 6300. 7100. 11800. 9400. 5500. 5800. 13200. 12500. 5600. 7000. 11900. 10400. 10700. 12700. 6800. 6500. 13600. 10200. 9000. 14900. 8200. 15200. 5000. 16300. 12400. 10500. 4200. 4700. 10900. 8100. 9500. 2200. 12800. 11200. 19100. 12300. 16700. 2600. 26400. 8800. 7400. 4900. 8000. 12000. 15700. 4100. 5700. 11500. 5400. 10800. 9900. 5200. 5900. 9300. 9700. 5100. 6600.] values

red_blood_cell_count has [5.2 nan 3.9 4.6 4.4 5. 4. 3.7 3.8 3.4 2.6 2.8 4.3 3.2 3.6 4.1 4.9 2.5 4.2 4.5 3.1 4.7 3.5 6. 2.1 5.6 2.3 2.9 2.7 8. 3.3 3. 2.4 4.8 5.4 6.1 6.2 6.3 5.1 5.8 5.5 5.3 6.4 5.7 5.9 6.5] values

hypertension has ['yes' 'no' nan] values

coronary_artery_disease has ['no' 'yes' '\tno' nan] values

appetite has ['good' 'poor' nan] values

peda_edema has ['no' 'yes' nan] values

aanemia has ['no' 'yes' nan] values

class has ['ckd' 'ckd\t' 'notckd'] values

```
In [18]: ### Cleaning categorical columns

df1['coronary_artery_disease'] = df1['coronary_artery_disease'].replace(to_replace = '\tno', value='no')
df1['class'] = df1['class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
```

```
In [19]: colname_num = [var for var in df1.columns if df1[var].dtype!='O']
colname_cat = [var for var in df1.columns if df1[var].dtype=='O']
print(colname_cat)
print(colname_num)
```

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

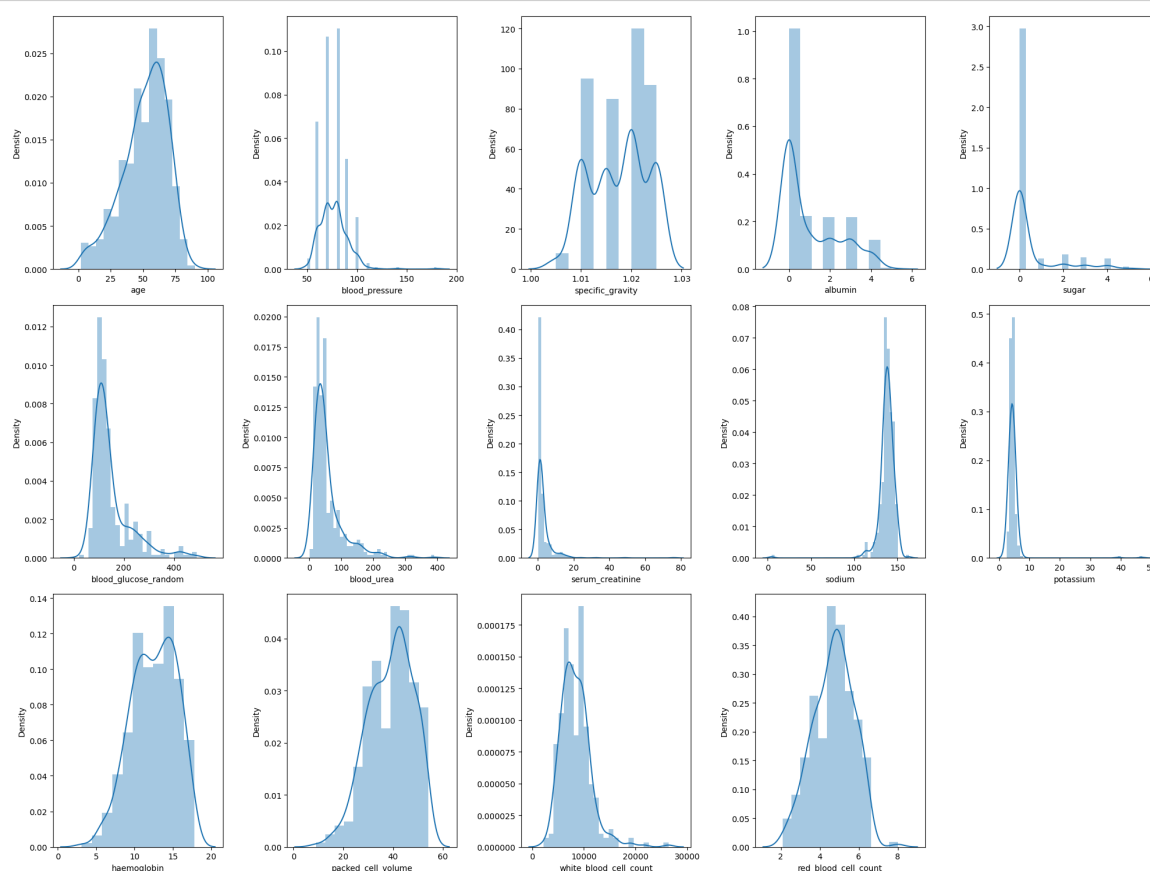
```
In [20]: ### Checking numerical features distribution
```

```
plt.figure(figsize = (20, 15))
plotnumber = 1

for column in colname_num:
    if plotnumber <= 14:
        ax = plt.subplot(3, 5, plotnumber)
        sns.distplot(df1[column])
        plt.xlabel(column)

    plotnumber += 1

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



```
In [21]: df2 = df1.copy()
df2.shape
```

```
Out[21]: (400, 24)
```

```
In [22]: df2.isna().sum()
```

```
Out[22]: age          9
blood_pressure      12
specific_gravity    47
albumin            46
sugar              49
red_blood_cells    152
pus_cell           65
pus_cell_clumps     4
bacteria            4
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
hypertension         2
coronary_artery_disease 2
appetite             1
peda_edema           1
aanemia              1
class               0
dtype: int64
```



```
In [23]: for i in colname_num:
          print("Column Names: ", i)
          print("Null Values: ", df2[i].isna().sum())
          print("Mean Values: ", df2[i].mean())
          print("Median Values: ", df2[i].median())
          print("Mode Values: ", df2[i].mode())
          print('-' * 50)
```

Column Names: age
Null Values: 9
Mean Values: 51.48337595907928
Median Values: 55.0
Mode Values: 0 60.0
Name: age, dtype: float64

Column Names: blood_pressure
Null Values: 12
Mean Values: 76.46907216494846
Median Values: 80.0
Mode Values: 0 80.0
Name: blood_pressure, dtype: float64

Column Names: specific_gravity
Null Values: 47
Mean Values: 1.0174079320113314
Median Values: 1.02
Mode Values: 0 1.02
Name: specific_gravity, dtype: float64

Column Names: albumin
Null Values: 46
Mean Values: 1.0169491525423728
Median Values: 0.0
Mode Values: 0 0.0
Name: albumin, dtype: float64

Column Names: sugar
Null Values: 49
Mean Values: 0.45014245014245013
Median Values: 0.0
Mode Values: 0 0.0
Name: sugar, dtype: float64

Column Names: blood_glucose_random
Null Values: 44
Mean Values: 148.0365168539326
Median Values: 121.0
Mode Values: 0 99.0
Name: blood_glucose_random, dtype: float64

Column Names: blood_urea
Null Values: 19
Mean Values: 57.425721784776904
Median Values: 42.0
Mode Values: 0 46.0
Name: blood_urea, dtype: float64

Column Names: serum_creatinine
Null Values: 17
Mean Values: 3.072454308093995
Median Values: 1.3
Mode Values: 0 1.2
Name: serum_creatinine, dtype: float64

Column Names: sodium
Null Values: 87
Mean Values: 137.52875399361022
Median Values: 138.0
Mode Values: 0 135.0

```
Name: sodium, dtype: float64
-----
Column Names: potassium
Null Values: 88
Mean Values: 4.62724358974359
Median Values: 4.4
Mode Values: 0 3.5
1 5.0
Name: potassium, dtype: float64
-----
Column Names: haemoglobin
Null Values: 52
Mean Values: 12.526436781609195
Median Values: 12.649999999999999
Mode Values: 0 15.0
Name: haemoglobin, dtype: float64
-----
Column Names: packed_cell_volume
Null Values: 71
Mean Values: 38.88449848024316
Median Values: 40.0
Mode Values: 0 41.0
1 52.0
Name: packed_cell_volume, dtype: float64
-----
Column Names: white_blood_cell_count
Null Values: 106
Mean Values: 8406.122448979591
Median Values: 8000.0
Mode Values: 0 9800.0
Name: white_blood_cell_count, dtype: float64
-----
Column Names: red_blood_cell_count
Null Values: 131
Mean Values: 4.707434944237917
Median Values: 4.8
Mode Values: 0 5.2
Name: red_blood_cell_count, dtype: float64
-----
```

```
In [24]: df2['age'] = df2['age'].fillna(df2['age'].median())
df2['blood_pressure'] = df2['blood_pressure'].fillna(df2['blood_pressure'].median())
df2['specific_gravity'] = df2['specific_gravity'].fillna(df2['specific_gravity'].median())
df2['albumin'] = df2['albumin'].fillna(df2['albumin'].mean())
df2['sugar'] = df2['sugar'].fillna(df2['sugar'].mean())
df2['blood_glucose_random'] = df2['blood_glucose_random'].fillna(df2['blood_glucose_random'].median())
df2['blood_urea'] = df2['blood_urea'].fillna(df2['blood_urea'].mean())
df2['serum_creatinine'] = df2['serum_creatinine'].fillna(df2['serum_creatinine'].mean())
df2['sodium'] = df2['sodium'].fillna(df2['sodium'].mean())
df2['potassium'] = df2['potassium'].fillna(df2['potassium'].mean())
df2['haemoglobin'] = df2['haemoglobin'].fillna(df2['haemoglobin'].mean())
df2['packed_cell_volume'] = df2['packed_cell_volume'].fillna(df2['packed_cell_volume'].mean())
df2['white_blood_cell_count'] = df2['white_blood_cell_count'].fillna(df2['white_blood_cell_count'].median())
df2['red_blood_cell_count'] = df2['red_blood_cell_count'].fillna(df2['red_blood_cell_count'].median())
df2.isna().sum()
```

```
Out[24]: age                                0
blood_pressure                             0
specific_gravity                           0
albumin                                    0
sugar                                       0
red_blood_cells                           152
pus_cell                                   65
pus_cell_clumps                            4
bacteria                                   4
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
hypertension                               2
coronary_artery_disease                    2
appetite                                    1
peda_edema                                 1
aanemia                                     1
class                                       0
dtype: int64
```

```
In [25]: for i in colname_cat:
          print("Column Names: ", i)
          print("Unique : ", df2[i].unique())
          print("Null Values: ", df2[i].isna().sum())
          print("Mode Values: ", df2[i].mode())
          print('-' * 50)
```

```
Column Names: red_blood_cells
Unique : [nan 'normal' 'abnormal']
Null Values: 152
Mode Values: 0 normal
Name: red_blood_cells, dtype: object
-----
Column Names: pus_cell
Unique : ['normal' 'abnormal' nan]
Null Values: 65
Mode Values: 0 normal
Name: pus_cell, dtype: object
-----
Column Names: pus_cell_clumps
Unique : ['notpresent' 'present' nan]
Null Values: 4
Mode Values: 0 notpresent
Name: pus_cell_clumps, dtype: object
-----
Column Names: bacteria
Unique : ['notpresent' 'present' nan]
Null Values: 4
Mode Values: 0 notpresent
Name: bacteria, dtype: object
-----
Column Names: hypertension
Unique : ['yes' 'no' nan]
Null Values: 2
Mode Values: 0 no
Name: hypertension, dtype: object
-----
Column Names: coronary_artery_disease
Unique : ['no' 'yes' nan]
Null Values: 2
Mode Values: 0 no
Name: coronary_artery_disease, dtype: object
-----
Column Names: appetite
Unique : ['good' 'poor' nan]
Null Values: 1
Mode Values: 0 good
Name: appetite, dtype: object
-----
Column Names: peda_edema
Unique : ['no' 'yes' nan]
Null Values: 1
Mode Values: 0 no
Name: peda_edema, dtype: object
-----
Column Names: aanemia
Unique : ['no' 'yes' nan]
Null Values: 1
Mode Values: 0 no
Name: aanemia, dtype: object
-----
Column Names: class
Unique : ['ckd' 'not ckd']
Null Values: 0
Mode Values: 0 ckd
Name: class, dtype: object
-----
```

```

In [26]: df2['red_blood_cells'] = df2['red_blood_cells'].fillna(df2['red_blood_cell
s'].mode()[0])
df2['pus_cell'] = df2['pus_cell'].fillna(df2['pus_cell'].mode()[0])
df2['pus_cell_clumps'] = df2['pus_cell_clumps'].fillna(df2['pus_cell_clump
s'].mode()[0])
df2['bacteria'] = df2['bacteria'].fillna(df2['bacteria'].mode()[0])
df2['hypertension'] = df2['hypertension'].fillna(df2['hypertension'].mode()
[0])
df2['coronary_artery_disease'] = df2['coronary_artery_disease'].fillna(df2
['coronary_artery_disease'].mode()[0])
df2['appetite'] = df2['appetite'].fillna(df2['appetite'].mode()[0])
df2['peda_edema'] = df2['peda_edema'].fillna(df2['peda_edema'].mode()[0])
df2['aanemia'] = df2['aanemia'].fillna(df2['aanemia'].mode()[0])
df2.isna().sum()

```

```

Out[26]: age                0
blood_pressure             0
specific_gravity           0
albumin                   0
sugar                     0
red_blood_cells            0
pus_cell                   0
pus_cell_clumps            0
bacteria                   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
hypertension               0
coronary_artery_disease    0
appetite                   0
peda_edema                 0
aanemia                    0
class                      0
dtype: int64

```

```
In [27]: ### Converting all categorical data into numerical data
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for x in colname_cat:
    df2[x]=le.fit_transform(df2[x])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print("Feature",x)
    print("Mapping", le_name_mapping)
```

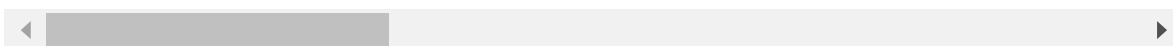
```
Feature red_blood_cells
Mapping {'abnormal': 0, 'normal': 1}
Feature pus_cell
Mapping {'abnormal': 0, 'normal': 1}
Feature pus_cell_clumps
Mapping {'notpresent': 0, 'present': 1}
Feature bacteria
Mapping {'notpresent': 0, 'present': 1}
Feature hypertension
Mapping {'no': 0, 'yes': 1}
Feature coronary_artery_disease
Mapping {'no': 0, 'yes': 1}
Feature appetite
Mapping {'good': 0, 'poor': 1}
Feature peda_edema
Mapping {'no': 0, 'yes': 1}
Feature aanemia
Mapping {'no': 0, 'yes': 1}
Feature class
Mapping {'ckd': 0, 'not ckd': 1}
```


In [28]: df2.head(15)

Out[28]:

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_
0	48.0	80.0	1.020	1.000000	0.000000	1	1	
1	7.0	50.0	1.020	4.000000	0.000000	1	1	
2	62.0	80.0	1.010	2.000000	3.000000	1	1	
3	48.0	70.0	1.005	4.000000	0.000000	1	0	
4	51.0	80.0	1.010	2.000000	0.000000	1	1	
5	60.0	90.0	1.015	3.000000	0.000000	1	1	
6	68.0	70.0	1.010	0.000000	0.000000	1	1	
7	24.0	80.0	1.015	2.000000	4.000000	1	0	
8	52.0	100.0	1.015	3.000000	0.000000	1	0	
9	53.0	90.0	1.020	2.000000	0.000000	0	0	
10	50.0	60.0	1.010	2.000000	4.000000	1	0	
11	63.0	70.0	1.010	3.000000	0.000000	0	0	
12	68.0	70.0	1.015	3.000000	1.000000	1	1	
13	68.0	70.0	1.020	1.016949	0.450142	1	1	
14	68.0	80.0	1.010	3.000000	2.000000	1	0	

15 rows × 24 columns



```
In [42]: for col in df2.columns:
          print(f"{col} has {df2[col].nunique()} categories\n")
```

```
age has 76 categories
blood_pressure has 10 categories
specific_gravity has 5 categories
albumin has 7 categories
sugar has 7 categories
red_blood_cells has 2 categories
pus_cell has 2 categories
pus_cell_clumps has 2 categories
bacteria has 2 categories
blood_glucose_random has 146 categories
blood_urea has 119 categories
serum_creatinine has 85 categories
sodium has 35 categories
potassium has 41 categories
haemoglobin has 116 categories
packed_cell_volume has 43 categories
white_blood_cell_count has 89 categories
red_blood_cell_count has 45 categories
hypertension has 2 categories
coronary_artery_disease has 2 categories
appetite has 2 categories
peda_edema has 2 categories
aanemia has 2 categories
class has 2 categories
```

```
In [43]: df3 = df2.copy()
df3.columns
```

```
Out[43]: Index(['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',
               'coronary_artery_disease', 'appetite', 'peda_edema', 'aanemia',
               'class'],
              dtype='object')
```

```
In [44]: ### Splitting Data into X and y
X = df3.values[:, :-1]
y = df3.values[:, 1]
print('X:', X.shape)
print('*' * 13)
print('y:', y.shape)
```

```
X: (400, 23)
*****
y: (400,)
```

```
In [45]: ### Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
#x = scaler.fit_transform(x)
print(X)
```

```
[[-0.21003135  0.25421378  0.4214856   ... -0.50780078 -0.4843221
 -0.42008403]
 [-2.62723421 -1.97247624  0.4214856   ... -0.50780078 -0.4843221
 -0.42008403]
 [ 0.615355    0.25421378 -1.4210744   ...  1.96927621 -0.4843221
  2.38047614]
 ...
 [-2.33245337  0.25421378  0.4214856   ... -0.50780078 -0.4843221
 -0.42008403]
 [-2.03767254 -1.23024623  1.3427656   ... -0.50780078 -0.4843221
 -0.42008403]
 [ 0.37953033  0.25421378  1.3427656   ... -0.50780078 -0.4843221
 -0.42008403]]
```

```
In [46]: y = y.astype(int) ### convert y in to integer always perform this operation
```

```
In [47]: ### Splitting into Training and Testing Data
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=10)

print("X_train: ",X_train.shape)
print("X_test: ",X_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)

X_train: (320, 23)
X_test: (80, 23)
y_train: (320,)
y_test: (80,)
```

```
In [48]: from sklearn.ensemble import ExtraTreesClassifier
#create a model object
model_etc = ExtraTreesClassifier()
#train the model object
model_etc.fit(X_train,y_train)
#predict using the model
y_pred = model_etc.predict(X_test)
print(y_pred)
```

```
[ 70  80  60  70  70  80  90  80  90  90  70  80  70  80 100  60  80  80
  80  70  80  80  80  80  80  90  70  80  80  60  50  70  70  80  70  70
  60  80  90  80  90  80  70  80  70  60  80  90  80  70  60 100  80 100
  70  80  80 100  60  90  80 100  80  60  90  60  90  80  90  70  60  60
 100  90  90  80  80  60  80  70]
```

```
In [49]: # Checking confusion matrix for the model
cfm = confusion_matrix(y_test,y_pred)
dff = pd.DataFrame(cfm)
dff.style.set_properties(**{"background-color": "#F3FFFF", "color": "black", "border": "2px solid black"})
```

Out[49]:

	0	1	2	3	4	5	6	7	8
0	1	1	0	0	0	0	0	0	0
1	0	11	0	0	0	0	0	0	0
2	0	0	17	0	0	0	0	0	0
3	0	0	0	30	0	0	0	0	0
4	0	0	0	1	13	0	0	0	0
5	0	0	0	0	0	3	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	1	0	0	0

```
In [50]: # Checking classification report score for the model
cr = classification_report(y_test,y_pred)
print("Classification report: ")
print(cr)

# Checking accuracy score for the model
acc = accuracy_score(y_test,y_pred)
print("Accuracy of the model: ",acc)
```

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

   50           1.00        0.50        0.67         2
   60           0.92        1.00        0.96        11
   70           1.00        1.00        1.00        17
   80           0.97        1.00        0.98        30
   90           1.00        0.93        0.96        14
  100           0.50        1.00        0.67         3
  110           0.00        0.00        0.00         1
  140           0.00        0.00        0.00         1
  180           0.00        0.00        0.00         1

 accuracy                   0.94         80
 macro avg           0.60        0.60        0.58         80
 weighted avg           0.92        0.94        0.92         80
```

```
Accuracy of the model: 0.9375
```

Made with ❤️ by Zahid Salim Shaikh

```
In [ ]:
```