Import Libraries

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

import warnings
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
sns.set()
plt.style.use('ggplot')
%matplotlib inline
```

Read Dataset And Information

```
df = pd.read_csv("kidney_disease.csv")
```

```
df.head(10)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	d
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	ye
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	n
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	ye
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	n
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	n
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	ye
6	6	68.0	70.0	1.010	0.0	0.0	NaN	normal	notpresent	notpresent	...	36	NaN	NaN	no	n
7	7	24.0	NaN	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44	6900	5	no	ye
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33	9600	4.0	yes	ye
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29	12100	3.7	yes	ye

10 rows × 26 columns

```
df['classification'].value_counts()
```

classification	count
ckd	248
notckd	150
ckd\t	2

```
df.shape
```

```
(400, 26)
```

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

```
df.head()
```

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wc	rc	htn	dm
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44	7800	5.2	yes	yes
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38	6000	NaN	no	no
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31	7500	NaN	no	yes
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	...	32	6700	3.9	yes	no
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35	7300	4.6	no	no

5 rows × 25 columns

```
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', 'sod
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_c
              '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	pus_cell_clumps	
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	n
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	n
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	n
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	n
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	n

5 rows × 25 columns

```
df.describe()
```

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.0(
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.4;
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.5(
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.5(
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.0(
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.0(
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.0(
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.0(

```
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: 78.3+ KB
```

```
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: 78.3+ KB
```

```
df.columns
```

```
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',
       'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
       'peda_edema', 'aanemia', 'class'],
      dtype='object')
```

```
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']
```

```
cat_cols
```

```
['red_blood_cells',
 'pus_cell',
 'pus_cell_clumps',
 'bacteria',
 'hypertension',
 'diabetes_mellitus',
 'coronary_artery_disease',
 'appetite',
 'peda_edema',
 'aanemia',
 'class']
```

```
num_cols
```

```
['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']
```

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

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

```
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'})
```

```
cols = ['diabetes_mellitus', 'coronary_artery_disease', 'class']
for col in cols:
    print(f"{col} has {df[col].unique()}")
```

```
diabetes_mellitus has ['yes' 'no' nan]
coronary_artery_disease has ['no' 'yes' nan]
class has ['ckd' 'not ckd']
```

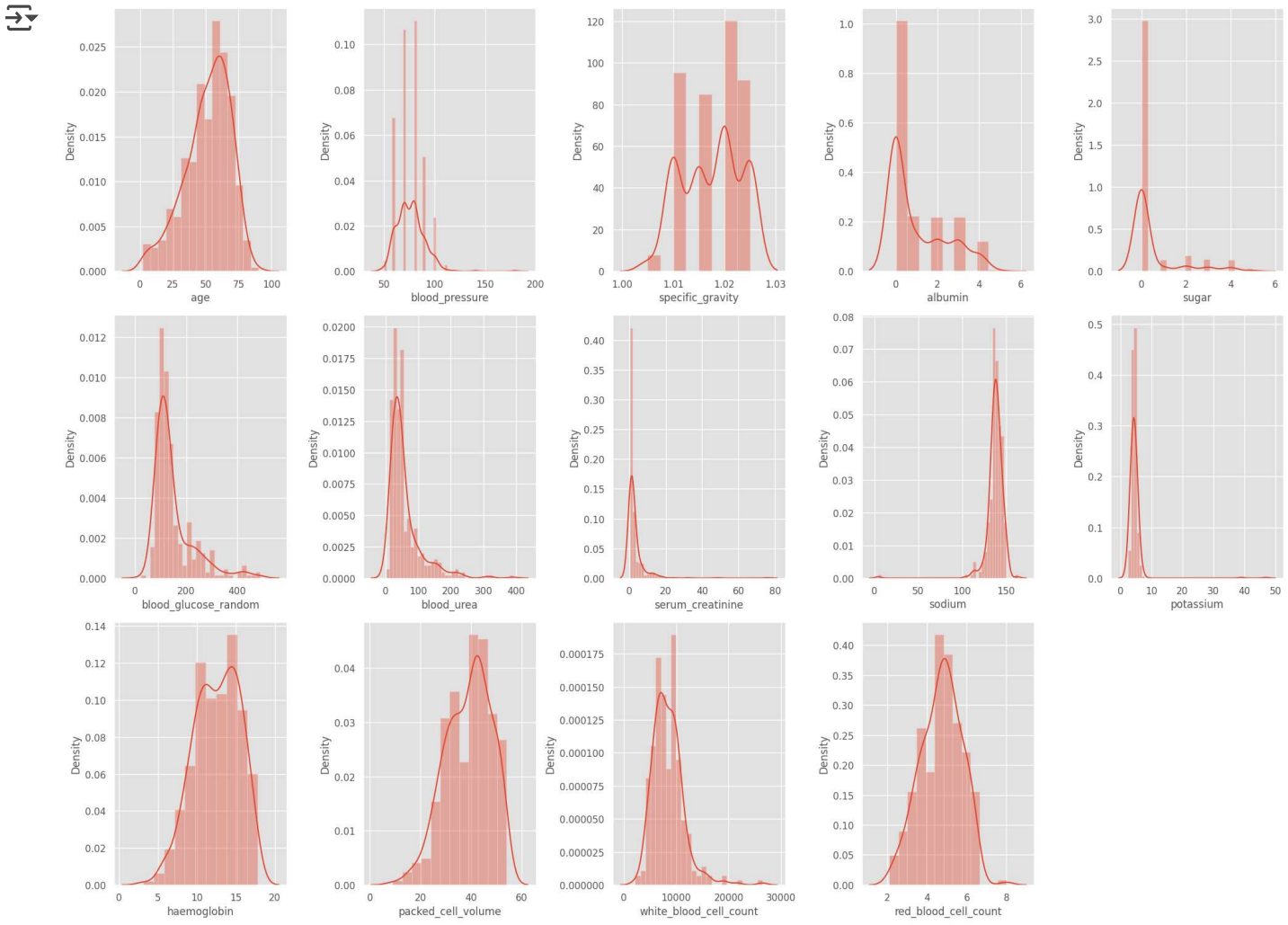
```
df['class'] = df['class'].map({'ckd':0, 'not ckd': 1})
df['class'] = pd.to_numeric(df['class'], errors = 'coerce')
```

```
cols = ['diabetes_mellitus', 'coronary_artery_disease', 'class']
for col in cols:
    print(f"{col} has {df[col].unique()}")
```

```
diabetes_mellitus has ['yes' 'no' nan]
coronary_artery_disease has ['no' 'yes' nan]
class has [0 1]
```

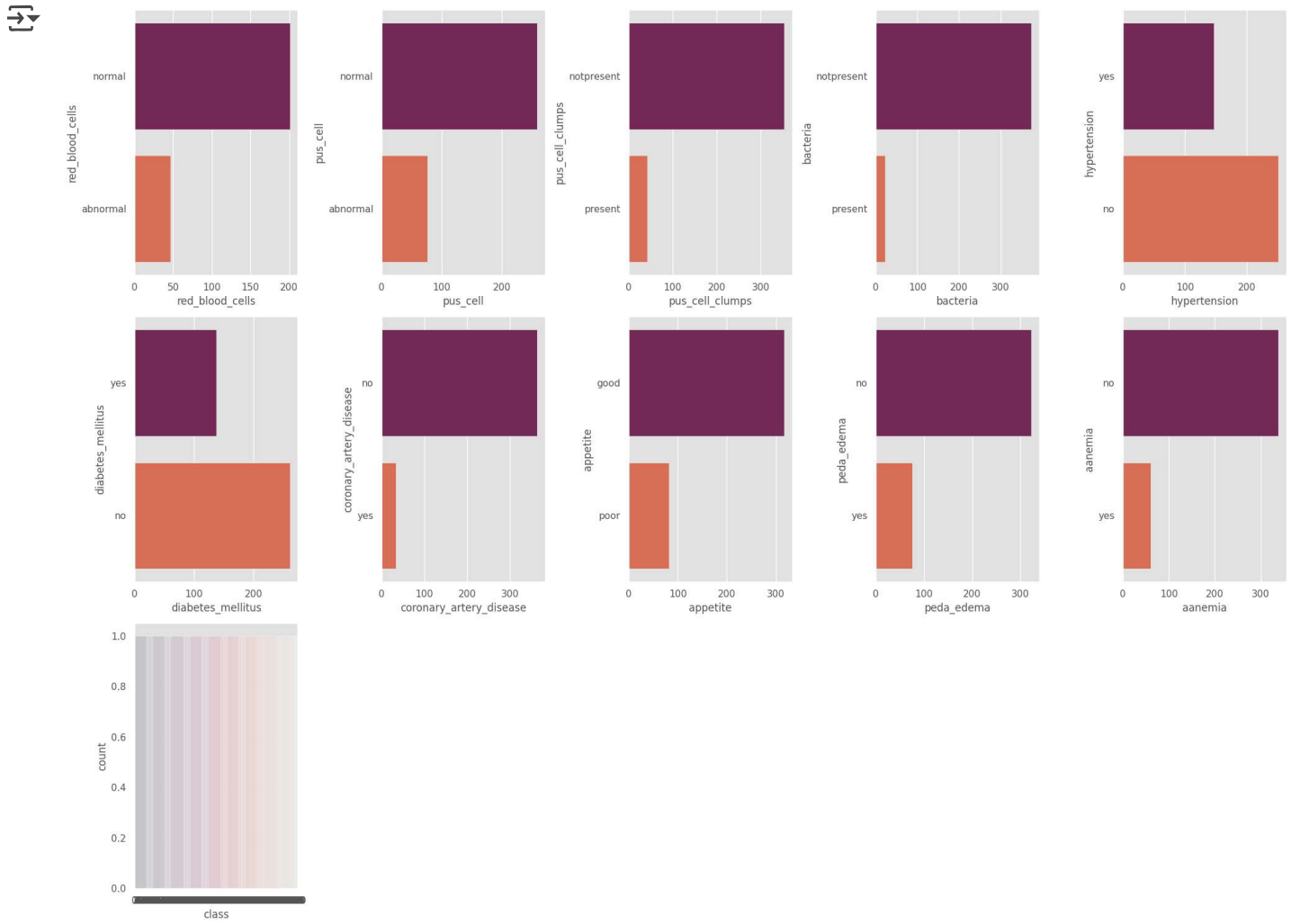
EDA

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

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

    plotnumber += 1

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




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

for column in cat_cols:
    if plotnumber <= 14:
        ax = plt.subplot(3, 5, plotnumber)
        sns.countplot(df[column] ,palette = 'rocket')
        plt.xlabel(column)
```

```
    plotnumber += 1

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



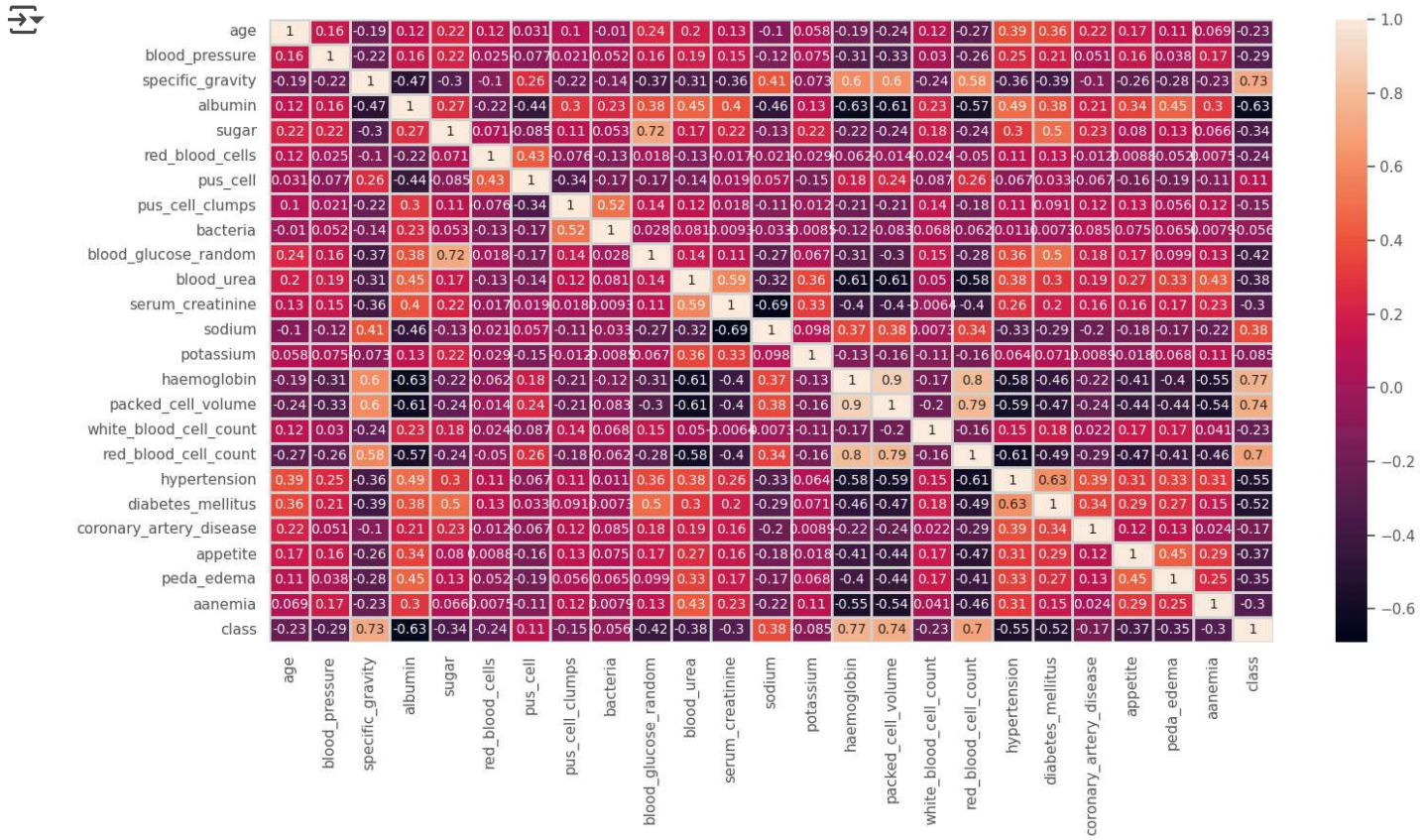
Correaltion

```
# Assuming 'red_blood_cells' is the problematic column containing "normal" and other categories
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
# Before calculating correlation, encode all object type columns
for col in df.select_dtypes(include=['object']).columns:
    df[col] = encoder.fit_transform(df[col])

# Calculate correlation after encoding
df.corr()
```

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells
age	1.000000	0.159480	-0.191096	0.122091	0.220866	0.121924
blood_pressure	0.159480	1.000000	-0.218836	0.160689	0.222576	0.024923
specific_gravity	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	-0.104557
albumin	0.122091	0.160689	-0.469760	1.000000	0.269305	-0.222257
sugar	0.220866	0.222576	-0.296234	0.269305	1.000000	0.070906
red_blood_cells	0.121924	0.024923	-0.104557	-0.222257	0.070906	1.000000
pus_cell	0.030580	-0.076983	0.258829	-0.444731	-0.084982	0.430674
pus_cell_clumps	0.099692	0.021158	-0.222646	0.298550	0.105624	-0.075644
bacteria	-0.010120	0.051808	-0.136611	0.227916	0.053064	-0.126138
blood_glucose_random	0.244992	0.160193	-0.374710	0.379464	0.717827	0.018438
blood_urea	0.196985	0.188517	-0.314295	0.453528	0.168583	-0.128008
serum_creatinine	0.132531	0.146222	-0.361473	0.399198	0.223244	-0.017462
sodium	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	-0.021008
potassium	0.058377	0.075151	-0.072787	0.129038	0.219450	-0.029140
haemoglobin	-0.192928	-0.306540	0.602582	-0.634632	-0.224775	-0.061777
packed_cell_volume	-0.242119	-0.326319	0.603560	-0.611891	-0.239189	-0.014277
white_blood_cell_count	0.118339	0.029753	-0.236215	0.231989	0.184893	-0.024369
red_blood_cell_count	-0.268896	-0.261936	0.579476	-0.566437	-0.237448	-0.049906
hypertension	0.391885	0.249046	-0.358271	0.492296	0.302518	0.110808
diabetes_mellitus	0.361414	0.207055	-0.386235	0.382786	0.495232	0.132903
coronary_artery_disease	0.215681	0.050542	-0.103474	0.206896	0.229803	-0.012018
appetite	0.172053	0.159945	-0.255065	0.337739	0.079725	0.008775
peda_edema	0.110624	0.038386	-0.283309	0.452803	0.133207	-0.051949
aanemia	0.069250	0.174901	-0.225576	0.296070	0.065805	0.007508
class	-0.227268	-0.294077	0.732163	-0.627090	-0.344070	-0.239595

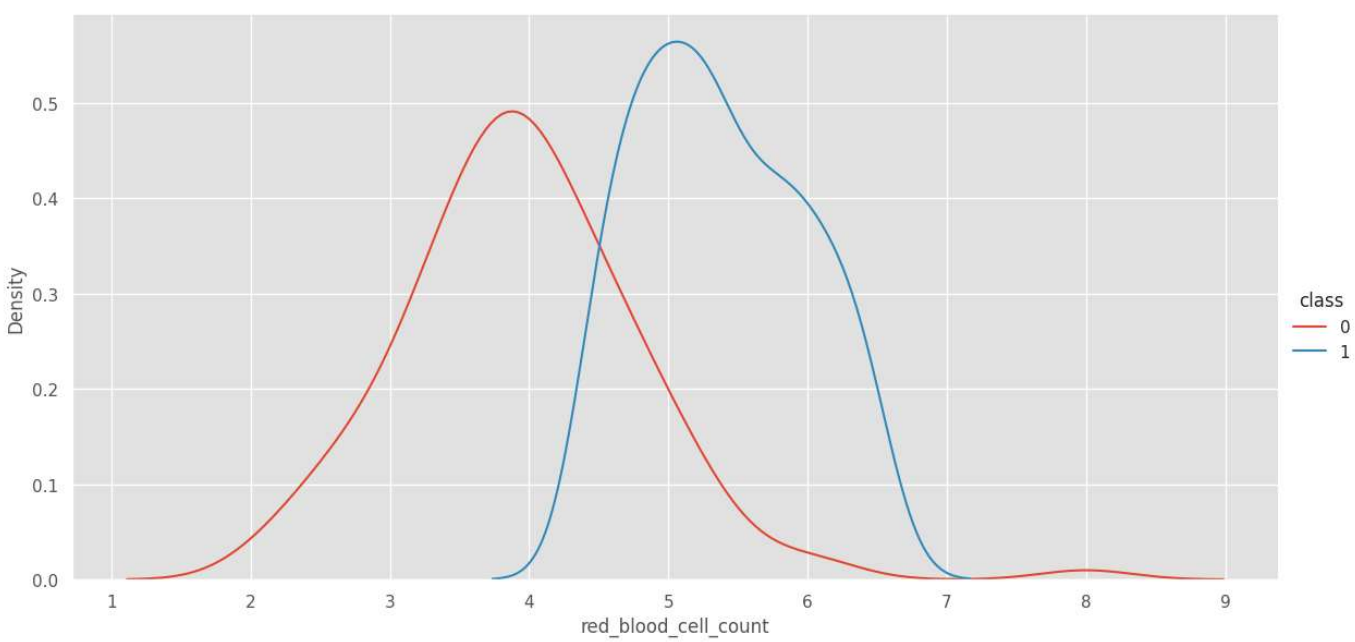
25 rows × 25 columns

```
plt.figure(figsize = (15,8))
sns.heatmap(df.corr(), annot=True, linewidth=2, linecolor = 'lightgray')
plt.show()
```



EDA

```
def voilin(col):
    fig  = px.violin(df, y=col, x='class', color='class', box=True, template='plotly_dark')
    return fig.show()

def kde(col):
    grid = sns.FacetGrid(df, hue='class', height = 6, aspect = 2)
    grid.map(sns.kdeplot, col)
    grid.add_legend()

def scatter_plot(col1, col2):
    fig  = px.scatter(df, x=col1, y=col2, color="class",  template='plotly_dark')
    return fig.show()


kde('red_blood_cell_count')
```



Data Preprocessing

```
# checking for missing value
df.isnull().sum().sort_values(ascending=False)
```

	0
red_blood_cell_count	131
white_blood_cell_count	106
potassium	88
sodium	87
packed_cell_volume	71
haemoglobin	52
sugar	49
specific_gravity	47
albumin	46
blood_glucose_random	44
blood_urea	19
serum_creatinine	17
blood_pressure	12
age	9
pus_cell_clumps	0
pus_cell	0
red_blood_cells	0
bacteria	0
hypertension	0
diabetes_mellitus	0
coronary_artery_disease	0
appetite	0
peda_edema	0
aanemia	0
class	0

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

	0
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

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

	0
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

```
df.head()
```

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

5 rows × 25 columns

```
# two method
# radom sampling->higer null value
# mean/mode-> lower null value


def random_sampling(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

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


# random sampling for numerical value
for col in num_cols:
    random_sampling(col)


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

	0
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

```
random_sampling('red_blood_cells')
random_sampling('pus_cell')

for col in cat_cols:
    impute_mode(col)


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

	0
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

Feature Encoding

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

```
red_blood_cells has 3
pus_cell has 3
pus_cell_clumps has 3
bacteria has 3
hypertension has 3
diabetes_mellitus has 3
coronary_artery_disease has 3
appetite has 3
peda_edema has 3
aanemia has 3
class has 2
```

```
# label_encoder
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

```
df.head()
```

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

5 rows × 25 columns

Model Building

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

```
X
```

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

400 rows × 24 columns

y

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

400 rows × 1 columns

```
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 = 0)
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

knn_acc = accuracy_score(y_test, knn.predict(X_test))
print(f"Training Accuracy of KNN is {accuracy_score(y_train, knn.predict(X_train))}")
print(f"Testing Accuracy of KNN is {accuracy_score(y_test, knn.predict(X_test))}")

print(f"Confusion Matrix of KNN is \n {confusion_matrix(y_test, knn.predict(X_test))}\n")
print(f"Classification Report of KNN is \n{classification_report(y_test, knn.predict(X_test))}")
```

```
Training Accuracy of KNN is 0.76875
Testing Accuracy of KNN is 0.6875
Confusion Matrix of KNN is
 [[36 16]
 [ 9 19]]

Classification Report of KNN is
              precision    recall  f1-score   support

           0       0.80      0.69      0.74        52
           1       0.54      0.68      0.60        28

    accuracy                           0.69        80
   macro avg       0.67      0.69      0.67        80
weighted avg       0.71      0.69      0.69        80
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(f"Training Accuracy of DTC is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Testing Accuracy of DTC is {accuracy_score(y_test, dtc.predict(X_test))}")

print(f"Confusion Matrix of DTC is \n {confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report of DTC is \n{classification_report(y_test, dtc.predict(X_test))}")
```

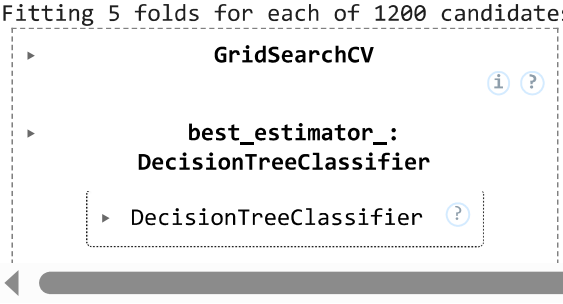
```
Training Accuracy of DTC is 1.0
Testing Accuracy of DTC is 0.9625
Confusion Matrix of DTC is
 [[51  1]
 [ 2 26]]

Classification Report of DTC is
              precision    recall  f1-score   support
```

```
                0       0.96      0.98      0.97        52
                1       0.96      0.93      0.95        28

         accuracy                           0.96        80
        macro avg       0.96      0.95      0.96        80
     weighted avg       0.96      0.96      0.96        80
```

Hyper Parameter Tuning

```
from sklearn.model_selection import GridSearchCV

GRID_PARAMETER = {
    'criterion':['gini','entropy'],
    'max_depth':[3,5,7,10],
    'splitter':['best','random'],
    'min_samples_leaf':[1,2,3,5,7],
    'min_samples_split':[1,2,3,5,7],
    'max_features':['auto', 'sqrt', 'log2']
}

grid_search_dtc = GridSearchCV(dtc, GRID_PARAMETER, cv=5, n_jobs=-1, verbose = 1)
grid_search_dtc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 1200 candidates, totalling 6000 fits
```




```
# best paramer and best score
print(grid_search_dtc.best_params_)
print(grid_search_dtc.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_spli
0.984375
```

```
dtc = grid_search_dtc.best_estimator_

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))
print(f"Training Accuracy of DTC is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Testing Accuracy of DTC is {accuracy_score(y_test, dtc.predict(X_test))}")

print(f"Confusion Matrix of DTC is \n {confusion_matrix(y_test, dtc.predict(X_test))}\n")
print(f"Classification Report of DTC is \n{classification_report(y_test, dtc.predict(X_test))}")
```

```
Training Accuracy of DTC is 0.96875
Testing Accuracy of DTC is 0.9375
Confusion Matrix of DTC is
 [[51  1]
```

```
 [ 4 24]]

Classification Report of DTC is
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        52
           1       0.96      0.86      0.91        28

    accuracy                           0.94        80
   macro avg       0.94      0.92      0.93        80
weighted avg       0.94      0.94      0.94        80
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rand_clf = RandomForestClassifier(criterion = "gini", max_depth = 10, max_features="sqrt", min_samples_leaf
rand_clf.fit(X_train, y_train)
```

```
RandomForestClassifier(max depth=10, min samples split=7, n estimators=400)
```

```
rand_clf_acc = accuracy_score(y_test, rand_clf.predict(X_test))
print(f"Training Accuracy of Random Forest is {accuracy_score(y_train, rand_clf.predict(X_train))}")
print(f"Testing Accuracy of Random Forest is {accuracy_score(y_test, rand_clf.predict(X_test))}")

print(f"Confusion Matrix of Random Forest is \n {confusion_matrix(y_test, rand_clf.predict(X_test))}\n")
print(f"Classification Report of Random Forest is \n{classification_report(y_test, rand_clf.predict(X_test)
```

```
Training Accuracy of Random Forest is 1.0
Testing Accuracy of Random Forest is 0.975
Confusion Matrix of Random Forest is
 [[52  0]
 [ 2 26]]

Classification Report of Random Forest is
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        52
           1       1.00      0.93      0.96        28

    accuracy                           0.97        80
   macro avg       0.98      0.96      0.97        80
weighted avg       0.98      0.97      0.97        80
```

XgBoost

```
from xgboost import XGBClassifier
xgb = XGBClassifier(objective="binary:logistic", learning_rate = 0.001, max_depth = 10, n_estimators = 100)
xgb.fit(X_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.001, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=10, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

```
xgb_acc = accuracy_score(y_test, xgb.predict(X_test))
print(f"Training Accuracy of XGB is {accuracy_score(y_train, xgb.predict(X_train))}")
print(f"Testing Accuracy of XGB is {accuracy_score(y_test, xgb.predict(X_test))}")

print(f"Confusion Matrix of XGB is \n {confusion_matrix(y_test, xgb.predict(X_test))}\n")
print(f"Classification Report of XGB is \n{classification_report(y_test, xgb.predict(X_test))}")
```

```
Training Accuracy of XGB is 0.61875
Testing Accuracy of XGB is 0.65
Confusion Matrix of XGB is
 [[52  0]
 [28  0]]

Classification Report of XGB is
              precision    recall  f1-score   support

           0       0.65      1.00      0.79        52
           1       0.00      0.00      0.00        28

    accuracy                           0.65        80
   macro avg       0.33      0.50      0.39        80
weighted avg       0.42      0.65      0.51        80
```

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, y_train)
```

```
LogisticRegression()
```

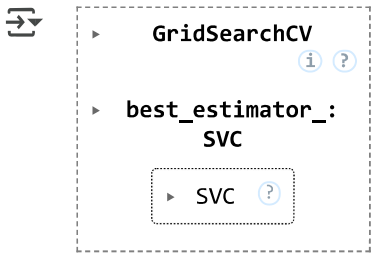
```
lr_acc = accuracy_score(y_test, lr.predict(X_test))
print(f"Training Accuracy of LR is {accuracy_score(y_train, lr.predict(X_train))}")
print(f"Testing Accuracy of LR is {accuracy_score(y_test, lr.predict(X_test))}")

print(f"Confusion Matrix of LR is \n {confusion_matrix(y_test, lr.predict(X_test))}\n")
print(f"Classification Report of LR is \n{classification_report(y_test, lr.predict(X_test))}")
```

```
Training Accuracy of LR is 0.903125
Testing Accuracy of LR is 0.9
```

```
Confusion Matrix of LR is
 [[46  6]
 [ 2 26]]

Classification Report of LR is
              precision    recall  f1-score   support

           0       0.96      0.88      0.92        52
           1       0.81      0.93      0.87        28

    accuracy                           0.90        80
   macro avg       0.89      0.91      0.89        80
weighted avg       0.91      0.90      0.90        80
```

SVM

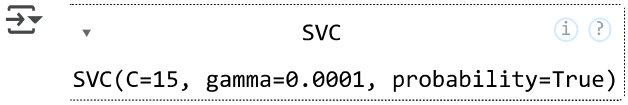
```
# SVM
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

svm  = SVC(probability=True)

parameter = {
    'gamma':[0.0001, 0.001, 0.01, 0.1],
    'C':[0.01, 0.05, 0.5, 0.1, 1, 10, 15, 20]
}

grid_search = GridSearchCV(svm, parameter)
grid_search.fit(X_train, y_train)
```




```
print(grid_search.best_params_)
print(grid_search.best_score_)
```

```
{'C': 10, 'gamma': 0.001}
0.753125
```

```
svm  = SVC(gamma = 0.0001, C  = 15, probability=True)
svm.fit(X_train, y_train)
```




```
svm_acc = accuracy_score(y_test, svm.predict(X_test))
print(f"Training Accuracy of SVC is {accuracy_score(y_train, svm.predict(X_train))}")
print(f"Testing Accuracy of SVC is {accuracy_score(y_test, svm.predict(X_test))}")
```

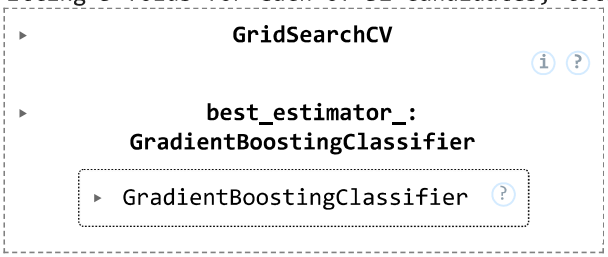
```
print(f"Confusion Matrix of SVC is \n {confusion_matrix(y_test, svm.predict(X_test))}\n")
print(f"Classification Report of SVC is \n{classification_report(y_test, svm.predict(X_test))}")
```

```
Training Accuracy of SVC is 0.990625
Testing Accuracy of SVC is 0.8125
Confusion Matrix of SVC is
 [[44  8]
 [ 7 21]]

Classification Report of SVC is
              precision    recall  f1-score   support

           0       0.86      0.85      0.85        52
           1       0.72      0.75      0.74        28

    accuracy                           0.81        80
   macro avg       0.79      0.80      0.80        80
weighted avg       0.81      0.81      0.81        80
```

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier()

PARAMETERS = {
    'loss': ['log_loss', 'exponential'],
    'learning_rate':[0.001, 0.1, 1, 10],
    'n_estimators':[100,150,180, 200]
}
grid_search_gbc = GridSearchCV(gbc, PARAMETERS, cv=5, n_jobs=-1, verbose= 1)
grid_search_gbc.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```




```
print(grid_search_gbc.best_params_)
```

```
{'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 180}
```

```
print(grid_search_gbc.best_score_)
```

```
0.996875
```

```
gbc = GradientBoostingClassifier(learning_rate= 0.1, loss = 'log_loss', n_estimators = 100)
gbc.fit(X_train, y_train)
```

```
GradientBoostingClassifier
GradientBoostingClassifier()
```

```
gbc_acc = accuracy_score(y_test, gbc.predict(X_test))
print(f"Training Accuracy of GBC is {accuracy_score(y_train, gbc.predict(X_train))}")
print(f"Testing Accuracy of GBC is {accuracy_score(y_test, gbc.predict(X_test))}")

print(f"Confusion Matrix of GBC is \n {confusion_matrix(y_test, gbc.predict(X_test))}\n")
print(f"Classification Report of GBC is \n{classification_report(y_test, gbc.predict(X_test))}")
```

```
Training Accuracy of GBC is 1.0
Testing Accuracy of GBC is 0.975
Confusion Matrix of GBC is
 [[52  0]
 [ 2 26]]

Classification Report of GBC is
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        52
           1       1.00      0.93      0.96        28

    accuracy                           0.97        80
   macro avg       0.98      0.96      0.97        80
weighted avg       0.98      0.97      0.97        80
```

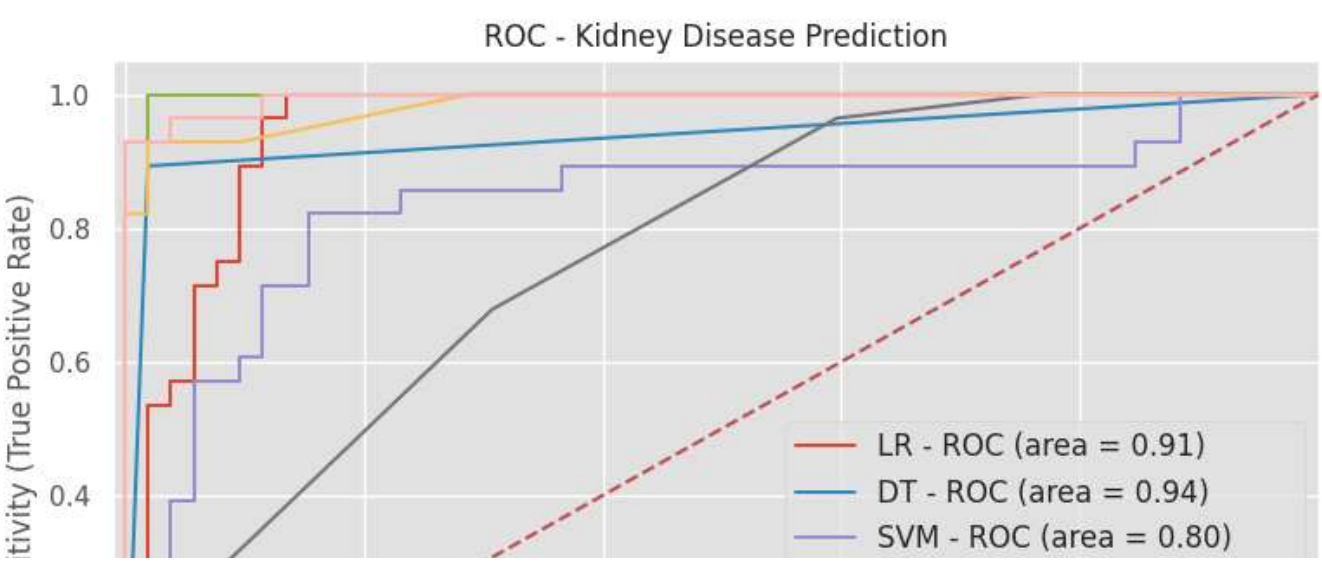
Model Comparison

```
models = pd.DataFrame({
    'Model':['Logistic Regression', 'KNN', 'SVM', 'DT', 'Random Forest Classifier', 'XgBoost','Gradient Boo
    'Score':[lr_acc, knn_acc, svm_acc, dtc_acc, rand_clf_acc, xgb_acc, gbc_acc]
})

models.sort_values(by='Score', ascending = False)
```

	Model	Score
4	Random Forest Classifier	0.9750
6	Gradient Boosting	0.9750
3	DT	0.9375
0	Logistic Regression	0.9000
2	SVM	0.8125
1	KNN	0.6875
5	XgBoost	0.6500

```
import pickle
model = dtc
pickle.dump(model, open("kindey.pkl", 'wb'))
```

```
from sklearn import metrics
plt.figure(figsize=(8,5))
models = [
{
    'label': 'LR',
    'model': lr,
},
{
    'label': 'DT',
    'model': dtc,
},
{
    'label': 'SVM',
    'model': svm,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
    'model': xgb,
},
{
    'label': 'RF',
    'model': rand_clf,
},
{
    'label': 'GBDT',
    'model': gbc,
}
]
for m in models:
    model = m['model']
    model.fit(X_train, y_train)
    y_pred=model.predict(X_test)
    fpr1, tpr1, thresholds = metrics.roc_curve(y_test, model.predict_proba(X_test)[:,1])
    auc = metrics.roc_auc_score(y_test,model.predict(X_test))
    plt.plot(fpr1, tpr1, label='%s - ROC (area = %0.2f)' % (m['label'], auc))

plt.plot([0, 1], [0, 1],'r--')
plt.xlim([-0.01, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1 - Specificity (False Positive Rate)', fontsize=12)
plt.ylabel('Sensitivity (True Positive Rate)', fontsize=12)
plt.title('ROC - Kidney Disease Prediction', fontsize=12)
plt.legend(loc="lower right", fontsize=12)
plt.savefig("roc_kidney.jpeg", format='jpeg', dpi=400, bbox_inches='tight')
plt.show()
```




```
from sklearn import metrics
import numpy as np
import matplotlib.pyplot as plt
models = [
{
    'label': 'LR',
    'model': lr,
},
{
    'label': 'DT',
    'model': dtc,
},
{
    'label': 'SVM',
    'model': svm,
},
{
    'label': 'KNN',
    'model': knn,
},
{
    'label': 'XGBoost',
```