

```
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
```

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

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

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classific
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	
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	
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	
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	
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	
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	
6	6	68.0	70.0	1.010	0.0	0.0	NaN	normal	notpresent	notpresent	...	36	NaN	NaN	no	no	no	good	no	no	
7	7	24.0	NaN	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44	6900	5	no	yes	no	good	yes	no	
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33	9600	4.0	yes	yes	no	good	no	yes	
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29	12100	3.7	yes	yes	no	poor	no	yes	

10 rows × 26 columns

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

```
ckd      248
notckd    150
ckd\t      2
Name: classification, dtype: int64
```

```
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	cad	appet	pe	ane	classificat
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	...	44	7800	5.2	yes	yes	no	good	no	no	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	...	38	6000	NaN	no	no	no	good	no	no	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	...	31	7500	NaN	no	yes	no	poor	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	no	poor	yes	yes	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	...	35	7300	4.6	no	no	no	good	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', 'sodium',
               'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count', 'red_blood_cell_count',
               'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite', 'peda_edema',
               'aanemia', 'class']
```

```
df.head()
```

```

age  blood_pressure  specific_gravity  albumin  sugar  red_blood_cells  pus_cell  pus_cell_clumps  bacteria  blood_glucose_random  ..
0  48.0             80.0             1.020      1.0    0.0             NaN      normal      notpresent  notpresent             121.0
1   7.0             50.0             1.020      4.0    0.0             NaN      normal      notpresent  notpresent             NaN
2  62.0             80.0             1.010      2.0    3.0             normal     normal      notpresent  notpresent             423.0
3  48.0             70.0             1.005      4.0    0.0             normal     abnormal     present  notpresent             117.0
4  51.0             80.0             1.010      2.0    0.0             normal     normal      notpresent  notpresent             106.0

5 rows × 25 columns

```

```
df.describe()
```

```

age  blood_pressure  specific_gravity  albumin  sugar  blood_glucose_random  blood_urea  serum_creatinine  sodiu
count  391.000000      388.000000      353.000000  354.000000  351.000000      356.000000  381.000000      383.000000  313.00000
mean   51.483376      76.469072      1.017408   1.016949   0.450142      148.036517   57.425722      3.072454  137.52875
std    17.169714     13.683637      0.005717   1.352679   1.099191      79.281714   50.503006      5.741126  10.40875
min     2.000000     50.000000      1.005000   0.000000   0.000000      22.000000    1.500000      0.400000   4.50000
25%    42.000000     70.000000      1.010000   0.000000   0.000000      99.000000   27.000000      0.900000  135.00000
50%    55.000000     80.000000      1.020000   0.000000   0.000000     121.000000  42.000000      1.300000  138.00000
75%    64.500000     80.000000      1.020000   2.000000   0.000000     163.000000  66.000000      2.800000  142.00000
max    90.000000    180.000000      1.025000   5.000000   5.000000     490.000000 391.000000     76.000000 163.00000

```

```
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  pda_edema             399 non-null    object
23  anemia                399 non-null    object
24  class                 400 non-null    object
dtypes: float64(11), object(14)
memory usage: 78.2+ 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: 78.2+ 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]

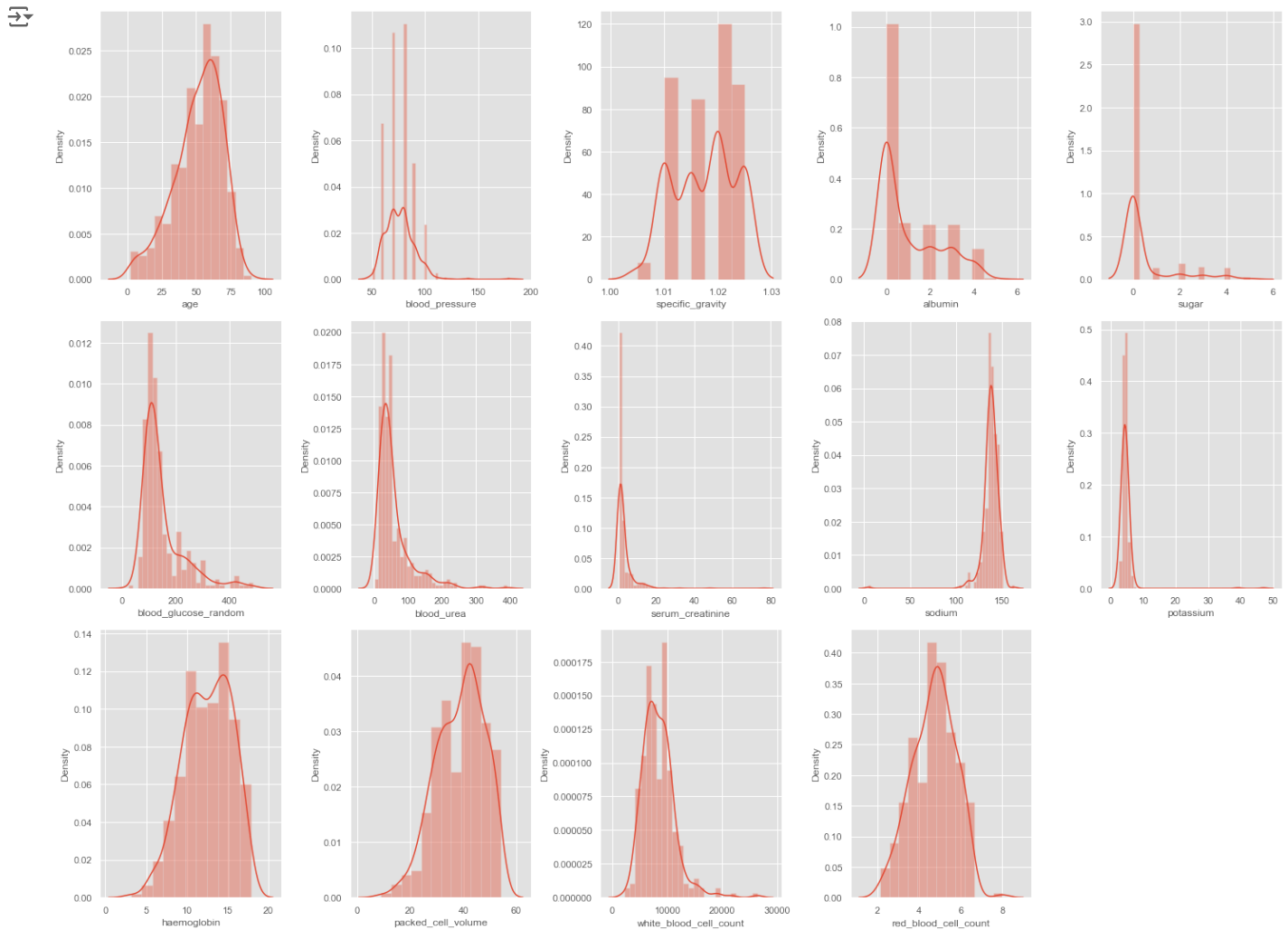
plt.figure(figsize = (20, 15))
plotnumber = 1

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()
```

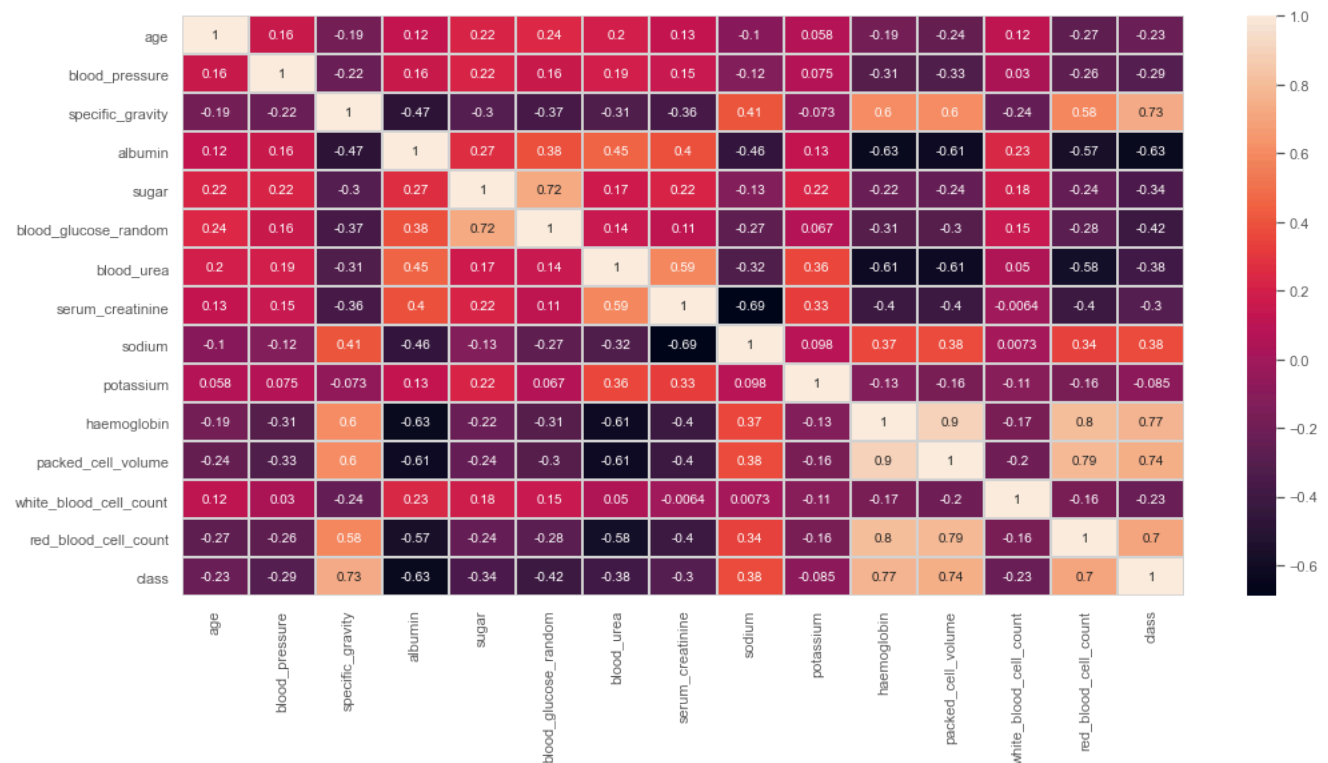


df.corr()



	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_urea	serum_creatinin
age	1.000000	0.159480	-0.191096	0.122091	0.220866	0.244992	0.196985	0.13253
blood_pressure	0.159480	1.000000	-0.218836	0.160689	0.222576	0.160193	0.188517	0.14622
specific_gravity	-0.191096	-0.218836	1.000000	-0.469760	-0.296234	-0.374710	-0.314295	-0.36147
albumin	0.122091	0.160689	-0.469760	1.000000	0.269305	0.379464	0.453528	0.39919
sugar	0.220866	0.222576	-0.296234	0.269305	1.000000	0.717827	0.168583	0.22324
blood_glucose_random	0.244992	0.160193	-0.374710	0.379464	0.717827	1.000000	0.143322	0.11487
blood_urea	0.196985	0.188517	-0.314295	0.453528	0.168583	0.143322	1.000000	0.58636
serum_creatinine	0.132531	0.146222	-0.361473	0.399198	0.223244	0.114875	0.586368	1.00000
sodium	-0.100046	-0.116422	0.412190	-0.459896	-0.131776	-0.267848	-0.323054	-0.69015
potassium	0.058377	0.075151	-0.072787	0.129038	0.219450	0.066966	0.357049	0.32610
haemoglobin	-0.192928	-0.306540	0.602582	-0.634632	-0.224775	-0.306189	-0.610360	-0.40167
packed_cell_volume	-0.242119	-0.326319	0.603560	-0.611891	-0.239189	-0.301385	-0.607621	-0.40419
white_blood_cell_count	0.118339	0.029753	-0.236215	0.231989	0.184893	0.150015	0.050462	-0.00639
red_blood_cell_count	-0.268896	-0.261936	0.579476	-0.566437	-0.237448	-0.281541	-0.579087	-0.40085
class	-0.227268	-0.294077	0.732163	-0.627090	-0.344070	-0.419672	-0.380605	-0.29996

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



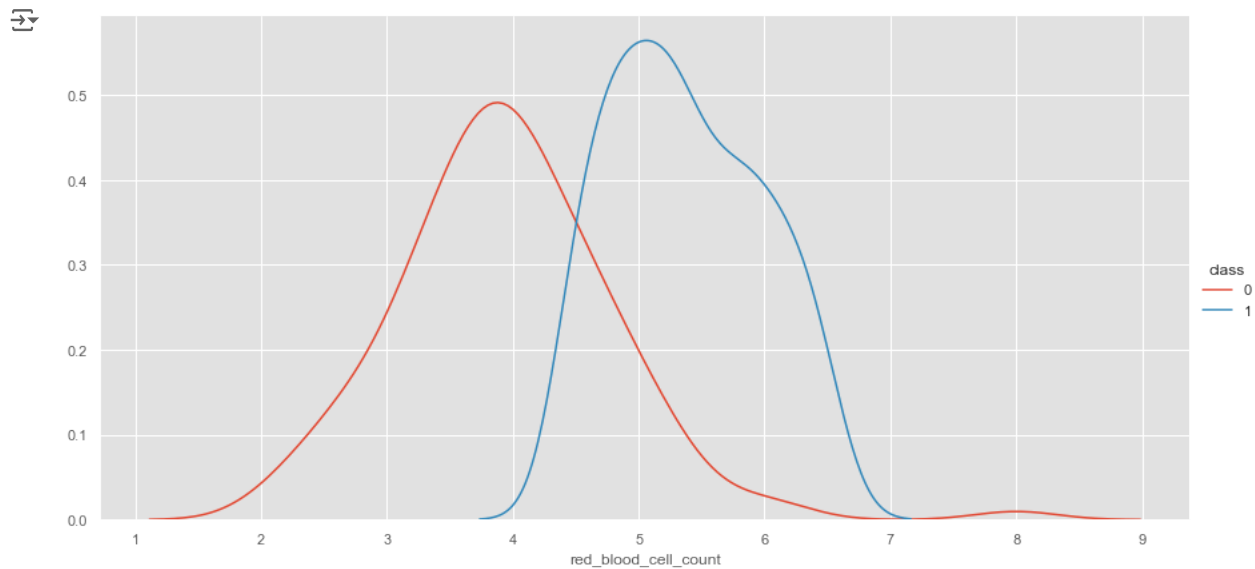
```
# EDA
```

```
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)
```

```
red_blood_cells      152
red_blood_cell_count 131
white_blood_cell_count 106
potassium            88
sodium              87
packed_cell_volume   71
pus_cell             65
haemoglobin          52
sugar               49
specific_gravity     47
albumin             46
blood_glucose_random 44
blood_urea          19
serum_creatinine     17
blood_pressure       12
age                  9
bacteria             4
pus_cell_clumps      4
hypertension         2
diabetes_mellitus    2
coronary_artery_disease 2
appetite             1
peda_edema           1
aanemia              1
class                0
dtype: int64
```

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

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



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

```

red_blood_cells    152
pus_cell           65
pus_cell_clumps     4
bacteria            4
hypertension        2
diabetes_mellitus    2
coronary_artery_disease 2
appetite            1
peda_edema          1
aanemia             1
class              0
dtype: int64

```

```
df.head()
```

```

age  blood_pressure  specific_gravity  albumin  sugar  red_blood_cells  pus_cell  pus_cell_clumps  bacteria  blood_glucose_random  ..
0  48.0             80.0             1.020     1.0    0.0           NaN      normal      notpresent  notpresent                121.0
1   7.0             50.0             1.020     4.0    0.0           NaN      normal      notpresent  notpresent                NaN
2  62.0             80.0             1.010     2.0    3.0          normal     normal      notpresent  notpresent                423.0
3  48.0             70.0             1.005     4.0    0.0          normal  abnormal      present    notpresent                117.0
4  51.0             80.0             1.010     2.0    0.0          normal     normal      notpresent  notpresent                106.0
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()
```

```

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

```

```

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

```

```

for col in cat_cols:
    impute_mode(col)

```

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

```

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

```

Feature Encoding

```

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

```

```

red_blood_cells has 2
pus_cell has 2
pus_cell_clumps has 2
bacteria has 2
hypertension has 2
diabetes_mellitus has 2
coronary_artery_disease has 2
appetite has 2
peda_edema has 2
aanemia has 2
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  bacteria  blood_glucose_random  ..
0  48.0             80.0             1.020     1.0    0.0              1         1              0         0             121.0  .
1   7.0             50.0             1.020     4.0    0.0              1         1              0         0             130.0  .
2  62.0             80.0             1.010     2.0    3.0              1         1              0         0             423.0  .
3  48.0             70.0             1.005     4.0    0.0              1         0              1         0             117.0  .
4  51.0             80.0             1.010     2.0    0.0              1         1              0         0             106.0  .

```

5 rows × 25 columns

Model Building

```

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

```

```
df.to_csv("data.csv")
```

```
X.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'],
      dtype='object')

```

```
y
```

```

0      0
1      0
2      0
3      0
4      0
..
395    1
396    1
397    1
398    1
399    1
Name: class, Length: 400, dtype: int64

```

```
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 0.78125
Testing Accuracy of KNN is 0.7875
Confusion Matrix of KNN is
[[40 12]
 [ 5 23]]

```

```

Classification Report of KNN is
              precision    recall  f1-score   support

     0           0.89       0.77       0.82         52
     1           0.66       0.82       0.73         28

   accuracy                   0.79         80
  macro avg           0.77       0.80       0.78         80
 weighted avg           0.81       0.79       0.79         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 0.975
Confusion Matrix of DTC is
[[51  1]
 [ 1 27]]

```

```

Classification Report of DTC is
              precision    recall  f1-score   support


```

0	0.98	0.98	0.98	52
1	0.96	0.96	0.96	28
accuracy			0.97	80
macro avg	0.97	0.97	0.97	80
weighted avg	0.97	0.97	0.97	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

```
GridSearchCV
├── estimator: DecisionTreeClassifier
│   └── DecisionTreeClassifier
```


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

 {'criterion': 'gini', 'max_depth': 7, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter': 'best'}
0.98125

```
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 0.984375
Testing Accuracy of DTC is 0.9625
Confusion Matrix of DTC is
[[49 3]
[0 28]]

```
Classification Report of DTC is
```

	precision	recall	f1-score	support
0	1.00	0.94	0.97	52
1	0.90	1.00	0.95	28
accuracy			0.96	80
macro avg	0.95	0.97	0.96	80
weighted avg	0.97	0.96	0.96	80

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
```

```
rand_clf = RandomForestClassifier(criterion = "gini", max_depth = 10, max_features="sqrt", min_samples_leaf= 1, min_samples_split= 7, n_estimators=400)
rand_clf.fit(X_train, y_train)
```

 RandomForestClassifier
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 0.9875
 Confusion Matrix of Random Forest is

```
[[52  0]
 [ 1 27]]
```

Classification Report of Random Forest is

	precision	recall	f1-score	support
0	0.98	1.00	0.99	52
1	1.00	0.96	0.98	28
accuracy			0.99	80
macro avg	0.99	0.98	0.99	80
weighted avg	0.99	0.99	0.99	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

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
               colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
               early_stopping_rounds=None, enable_categorical=False,
               eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
               grow_policy='depthwise', importance_type=None,
               interaction_constraints='', learning_rate=0.001, max_bin=256,
               max_cat_threshold=64, max_cat_to_onehot=4, max_delta_step=0,
               max_depth=10, max_leaves=0, min_child_weight=1, missing=nan,
               monotone_constraints=()), n_estimators=100, n_jobs=0,
               num_parallel_tree=1, predictor='auto', random_state=0, ...)
```

```
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 0.98125
 Testing Accuracy of XGB is 0.9625
 Confusion Matrix of XGB is

```
[[51  1]
 [ 2 26]]
```

Classification Report of XGB 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

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

LogisticRegression

```
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 0.934375
Testing Accuracy of LR is 0.9375
Confusion Matrix of LR is
[[48  4]
 [ 1 27]]

```

```

Classification Report of LR is
              precision    recall  f1-score   support

     0       0.98         0.92         0.95         52
     1       0.87         0.96         0.92         28

 accuracy          0.94
 macro avg         0.93         0.94         0.93         80
 weighted avg      0.94         0.94         0.94         80

```

```

# 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)

```

```

> GridSearchCV
> estimator: SVC
  > SVC

```

```

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

```

```

{'C': 20, 'gamma': 0.0001}
0.7375

```

```

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

```

```

SVC
SVC(C=15, gamma=0.0001, probability=True)

```

```

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 0.7875
Confusion Matrix of SVC is
[[41 11]
 [ 6 22]]

```

```

Classification Report of SVC is
              precision    recall  f1-score   support

     0       0.87         0.79         0.83         52
     1       0.67         0.79         0.72         28

 accuracy          0.79
 macro avg         0.77         0.79         0.77         80
 weighted avg      0.80         0.79         0.79         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

```

GridSearchCV
└─ estimator: GradientBoostingClassifier
   └─ GradientBoostingClassifier

```

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

→ {'learning_rate': 1, 'loss': 'exponential', 'n_estimators': 100}

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

→ 0.984375

```

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
 [[51 1]
 [1 27]]

```

Classification Report of GBC is
      precision    recall  f1-score   support

     0       0.98      0.98      0.98        52
     1       0.96      0.96      0.96        28

   accuracy          0.97
  macro avg          0.97
 weighted avg          0.97

```

Model Comparison

```

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

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

```

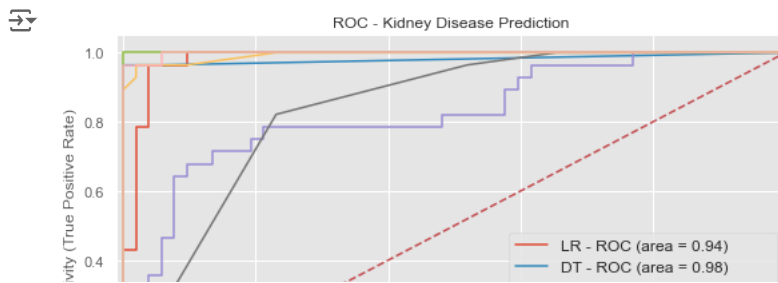


	Model	Score
4	Random Forest Classifier	0.9875
6	Gradient Boosting	0.9750
3	DT	0.9625
5	XgBoost	0.9625
0	Logistic Regression	0.9375
1	KNN	0.7875
2	SVM	0.7875

```
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',
        'model': xgb,
    },
    {
        'label': 'RF',
        'model': rand_clf,
    },
    {
        'label': 'GBDT',
        'model': gbc,
    }
]

means_roc = []
means_accuracy = [100*round(lr_acc,4), 100*round(dtc_acc,4), 100*round(svm_acc,4), 100*round(knn_acc,4), 100*round(xgb_acc,4),
                  100*round(rand_clf_acc,4), 100*round(gbc_acc,4)]

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))
    auc = 100*round(auc,4)
    means_roc.append(auc)

print(means_accuracy)
print(means_roc)

n_groups = 7
means_accuracy = tuple(means_accuracy)
means_roc = tuple(means_roc)

fig, ax = plt.subplots(figsize=(8,5))

```