```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'ckdisease:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F1111%2F2005%2Fbundle%2Farchive.zip

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Failed to load (likely expired) https://storage.googleapis.com/kaggle-data-sets/1111/2005/bundle/archive.zip?X-Goog-A
Data source import complete.
```

Chronic Kidney Disease Prediction

Table of Contents


```
# necessary imports

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')
%matplotlib inline
pd.set_option('display.max_columns', 26)
```

```
# loading data

df= pd.read_csv('/content/kidney_disease (11).csv')
df.head()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	15.4	44	7800	5.2	yes	yes
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	38	6000	NaN	no	no
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	9.6	31	7500	NaN	no	yes
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.0	2.5	11.2	32	6700	3.9	yes	no
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	11.6	35	7300	4.6	no	no

```
df.shape
```

```
(400, 26)
```

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


# rename column names to make it more user-friendly

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_gluc
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	

```
df.describe()
```

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_urea	serum_creatinin
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.00000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.07245
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.74112
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.40000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.90000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.30000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.80000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.00000

```
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  peda_edema               399 non-null    object
 23  aanemia                  399 non-null    object
 24  class                    400 non-null    object
dtypes: float64(11), object(14)
memory usage: 78.2+ KB
```

As we can see that 'packed_cell_volume', 'white_blood_cell_count' and 'red_blood_cell_count' are object type. We need to change them to numerical dtype.

```
# converting necessary columns to numerical type

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
```

```
# Extracting categorical and numerical columns

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']
```

```
# looking at unique values in categorical columns

for col in cat_cols:
    print(f"{col} has {df[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

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

diabetes_mellitus has ['yes' 'no' ' yes' '\tno' '\tyes' 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
```

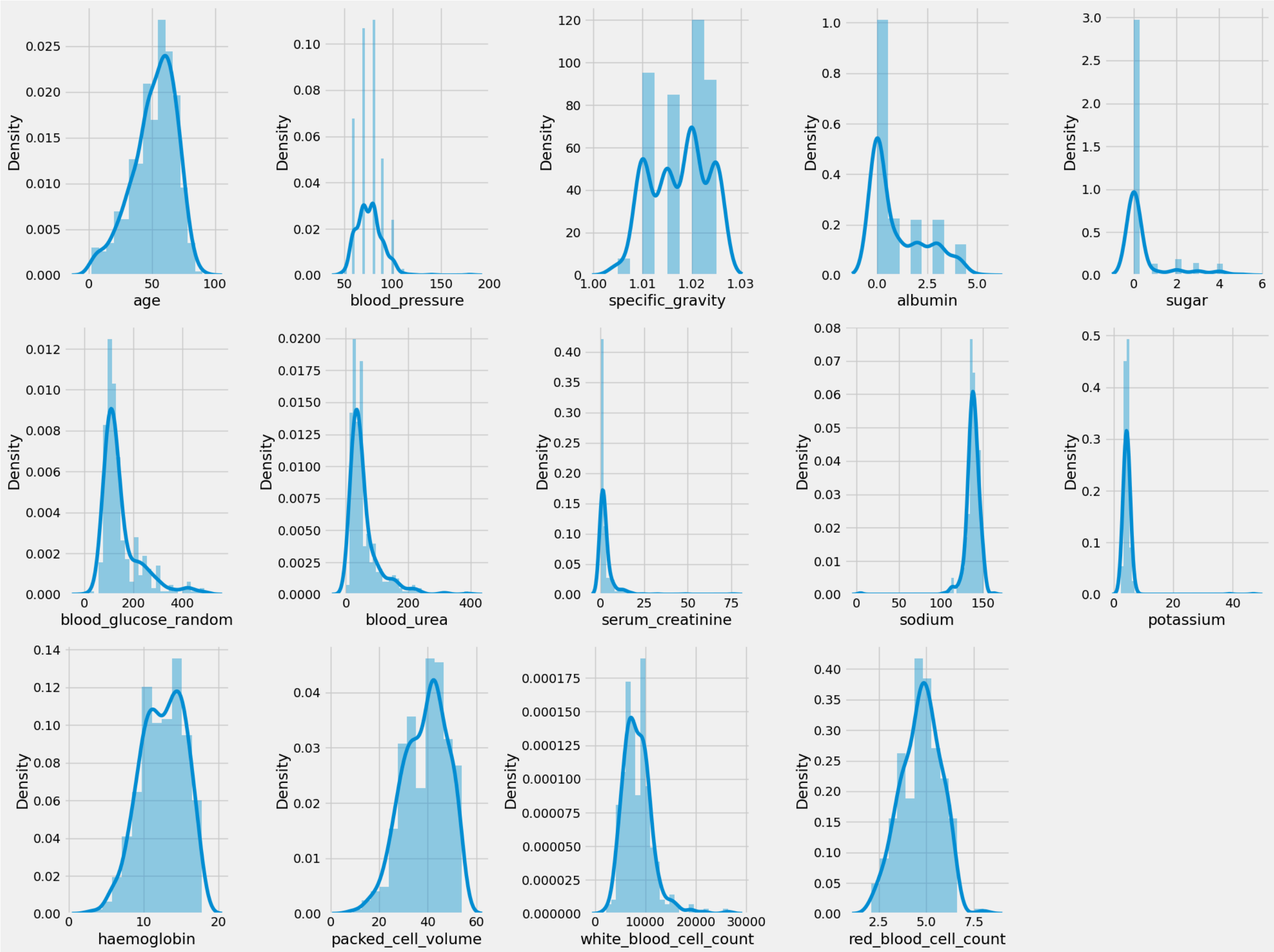
There is some ambugity present in the columns we have to remove that.

```
# replace incorrect values

df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inplace=True)

df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_replace = '\tno', value='no')

df['class'] = df['class'].replace(to_replace = {'ckd\t': 'ckd', 'notckd': 'not ckd'})
```

```
df['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()} values\n")
```

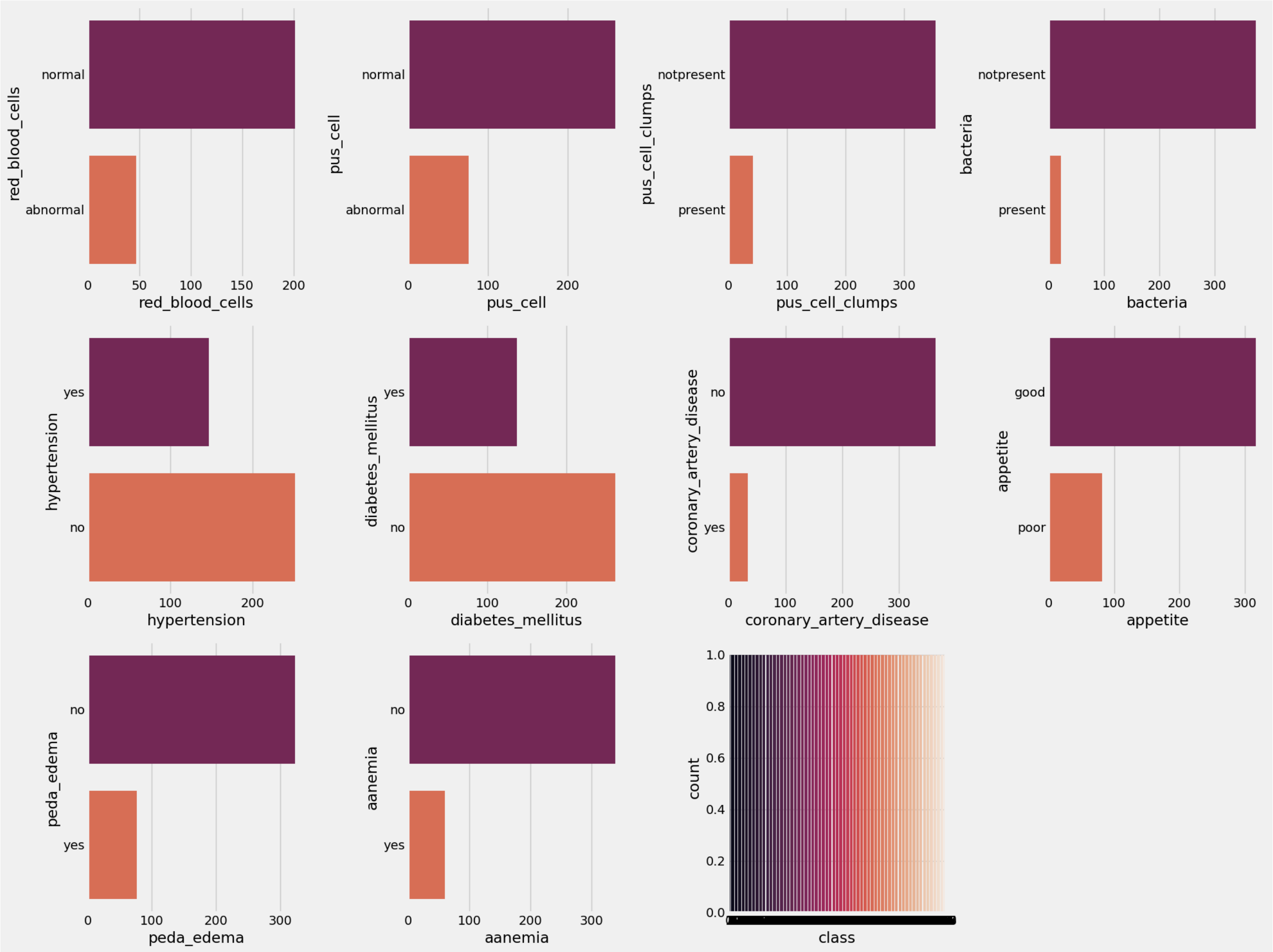
```
diabetes_mellitus has ['yes' 'no' nan] values

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

class has [0 1] values
```

```
# checking numerical features distribution

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()
```



Skewness is present in some of the columns.

```
# looking at categorical columns

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

for column in cat_cols:
    if plotnumber <= 11:
        ax = plt.subplot(3, 4, plotnumber)
        sns.countplot(df[column], palette = 'rocket')
        plt.xlabel(column)

    plotnumber += 1

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




```
df.columns
```

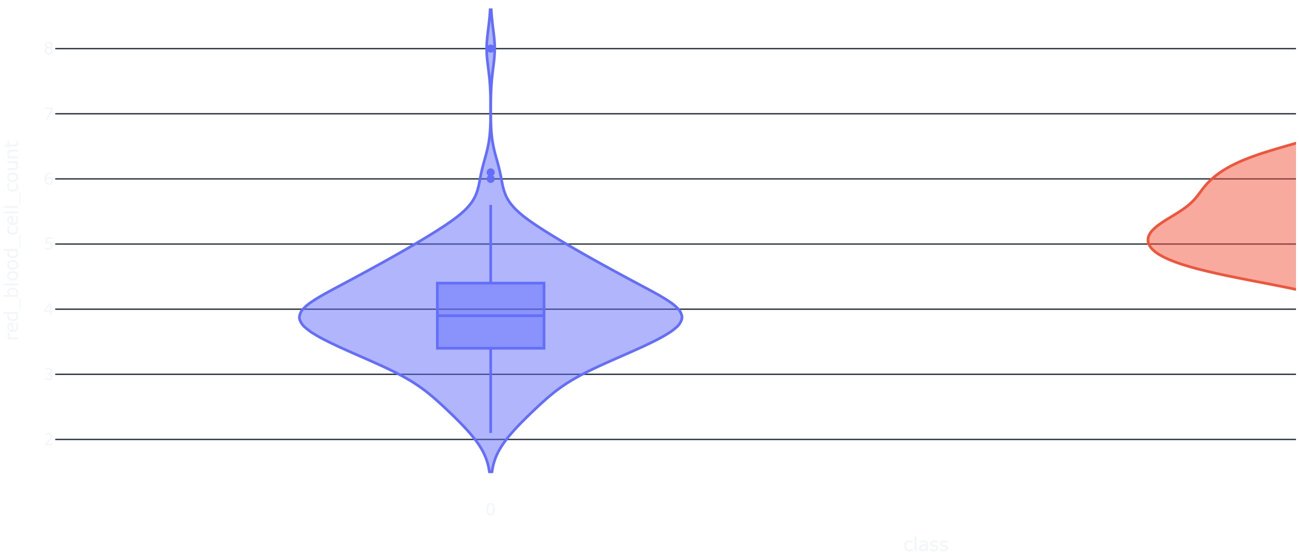
```
Index(['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
       'red_blood_cells', 'pus_cell', 'pus_cell_clumps', 'bacteria',
       'blood_glucose_random', 'blood_urea', 'serum_creatinine', 'sodium',
       'potassium', 'haemoglobin', 'packed_cell_volume',
       'white_blood_cell_count', 'red_blood_cell_count', 'hypertension',
       'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
       'peda_edema', 'aanemia', 'class'],
      dtype='object')
```

Exploratory Data Analysis (EDA)

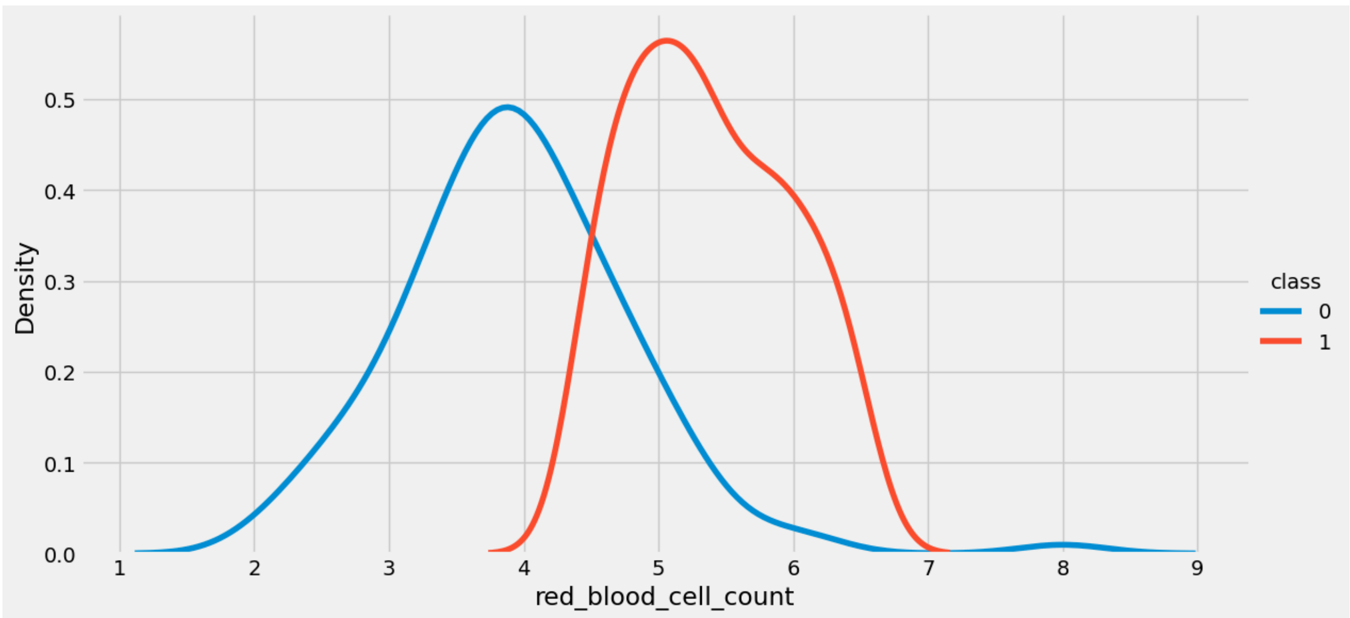
```
# defining functions to create plot

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

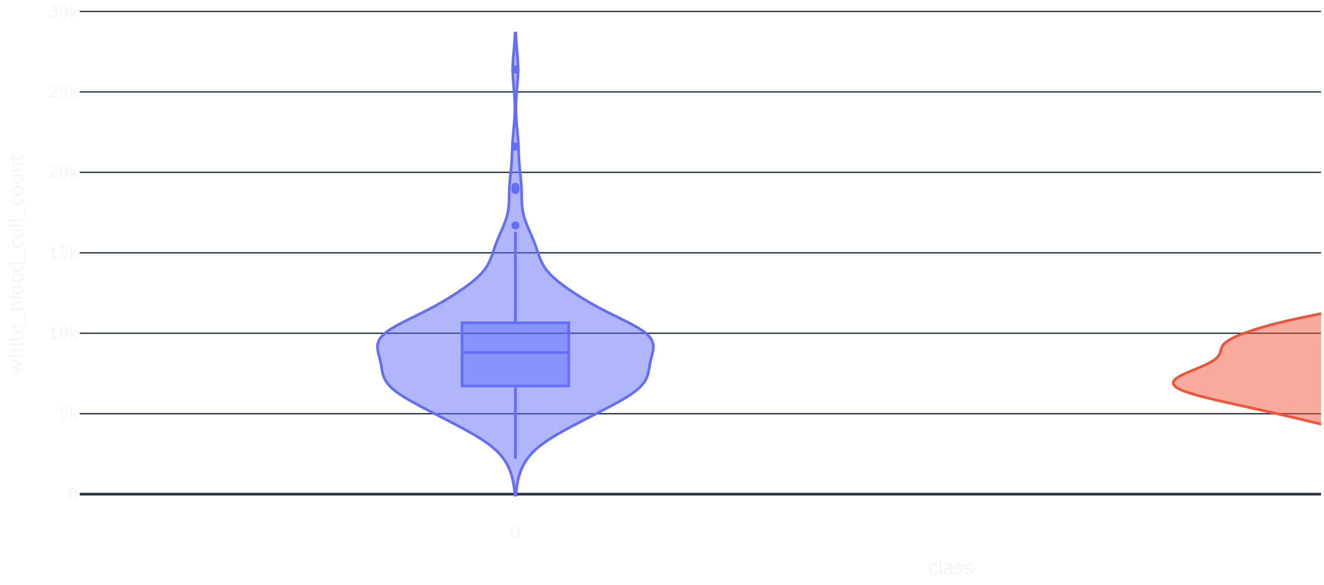
```
violin('red_blood_cell_count')
```



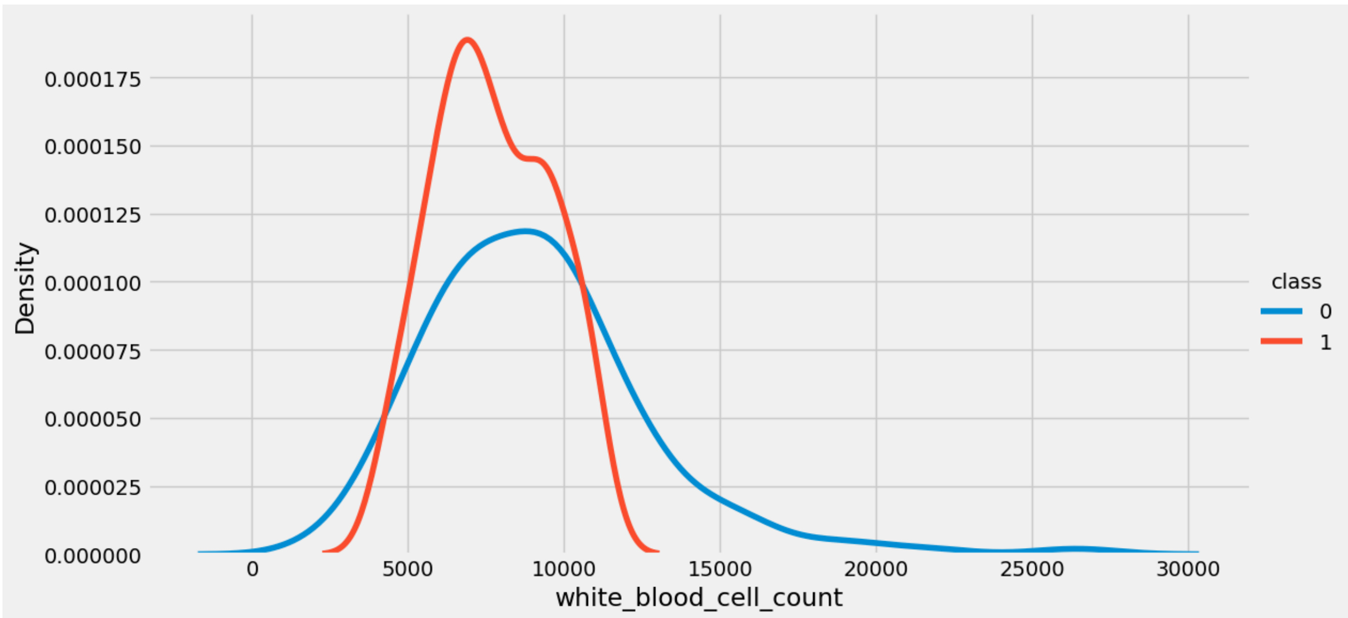
```
kde('red_blood_cell_count')
```



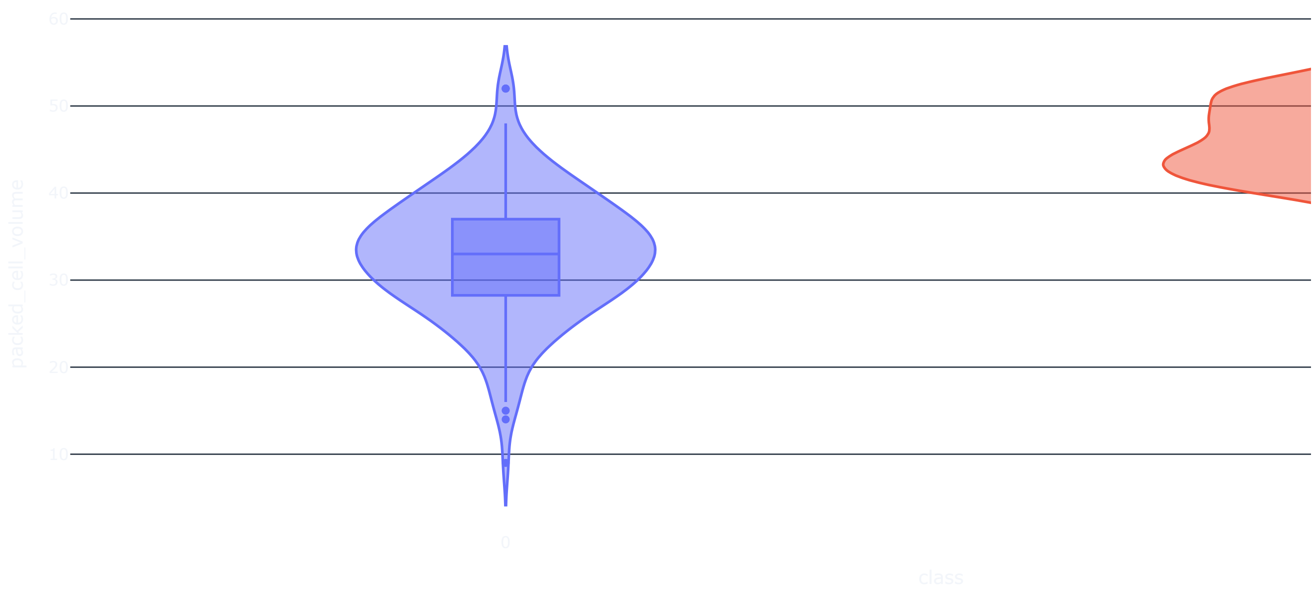

```
violin('white_blood_cell_count')
```



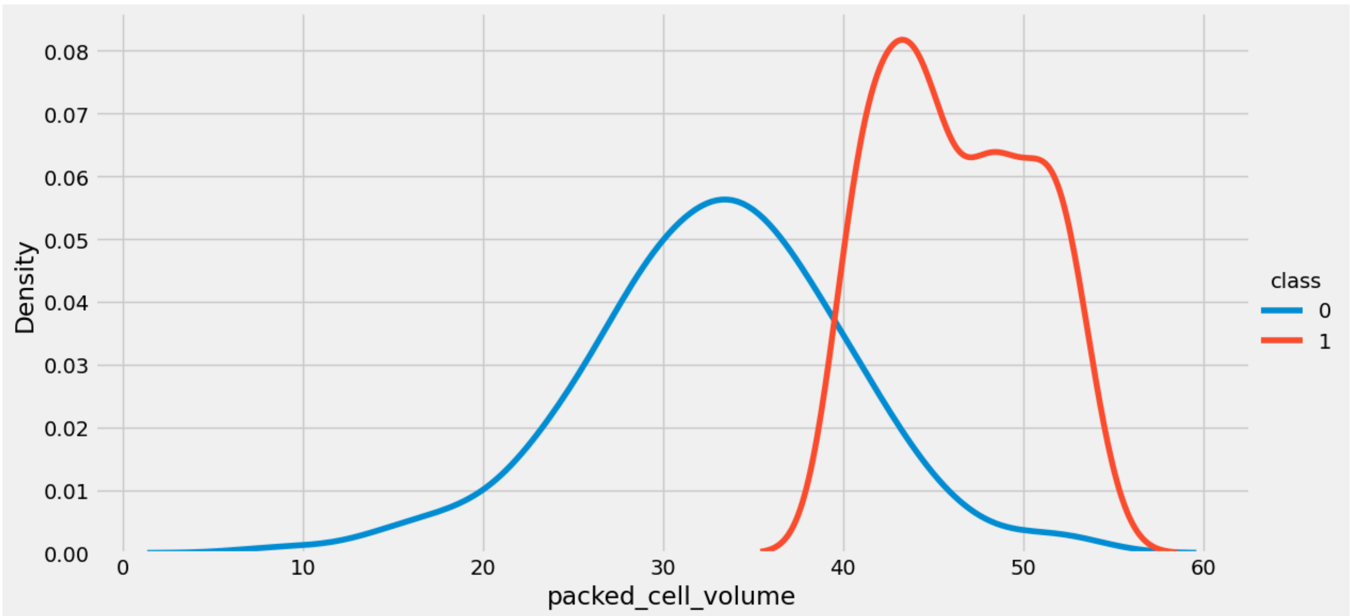

```
kde('white_blood_cell_count')
```



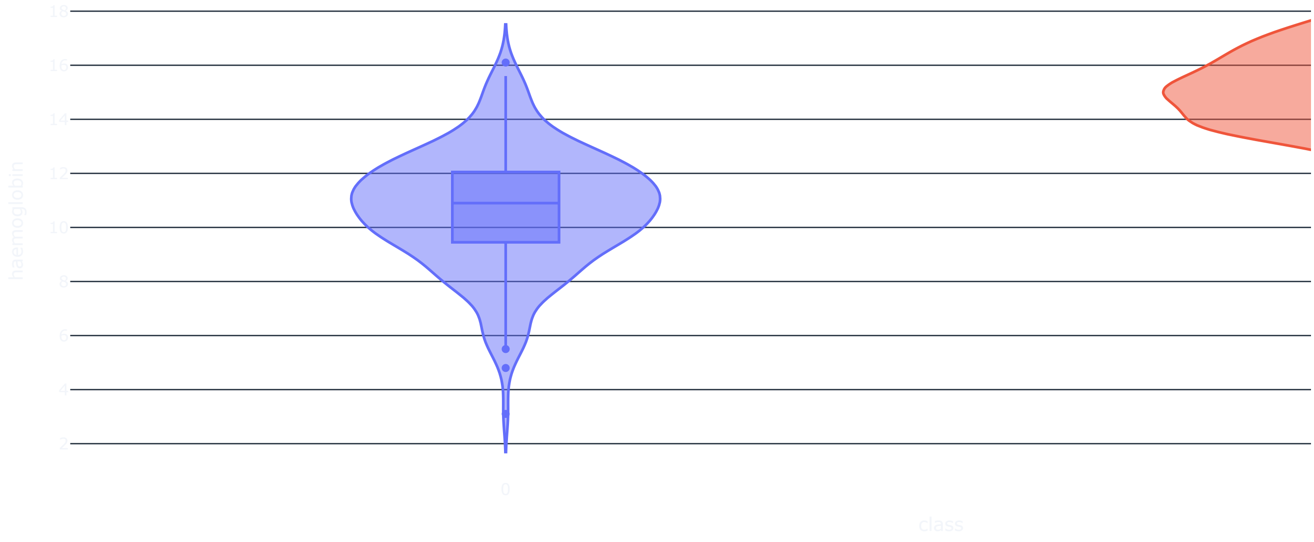

```
violin('packed_cell_volume')
```



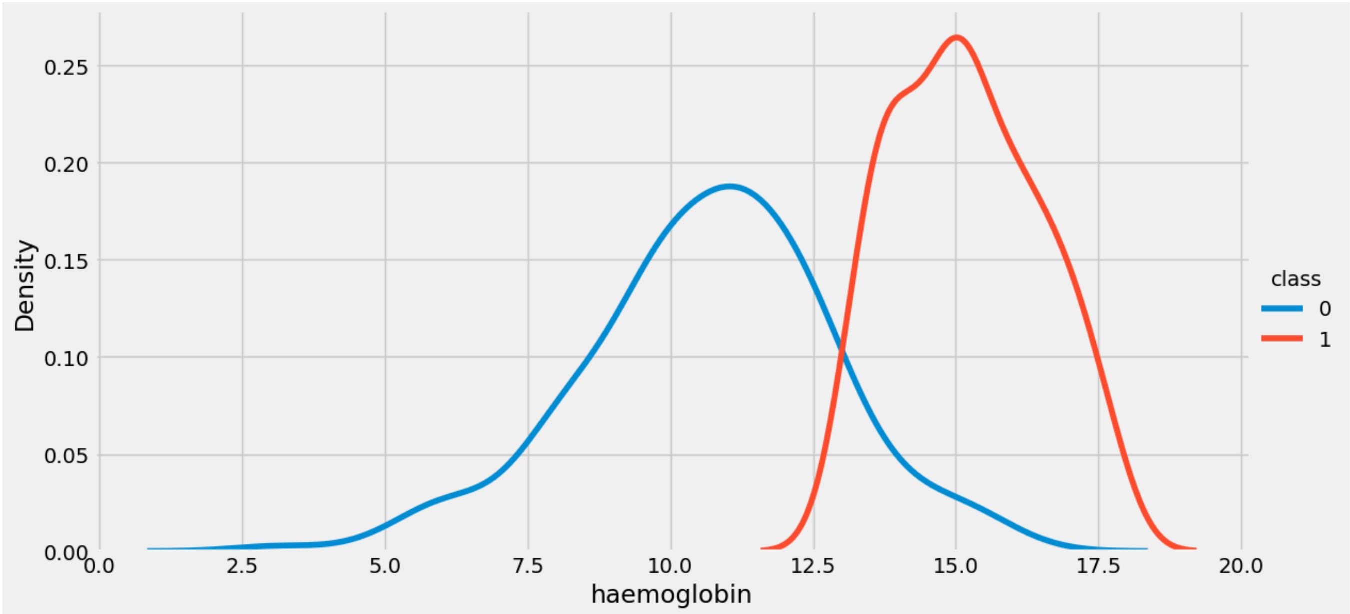

```
kde('packed_cell_volume')
```



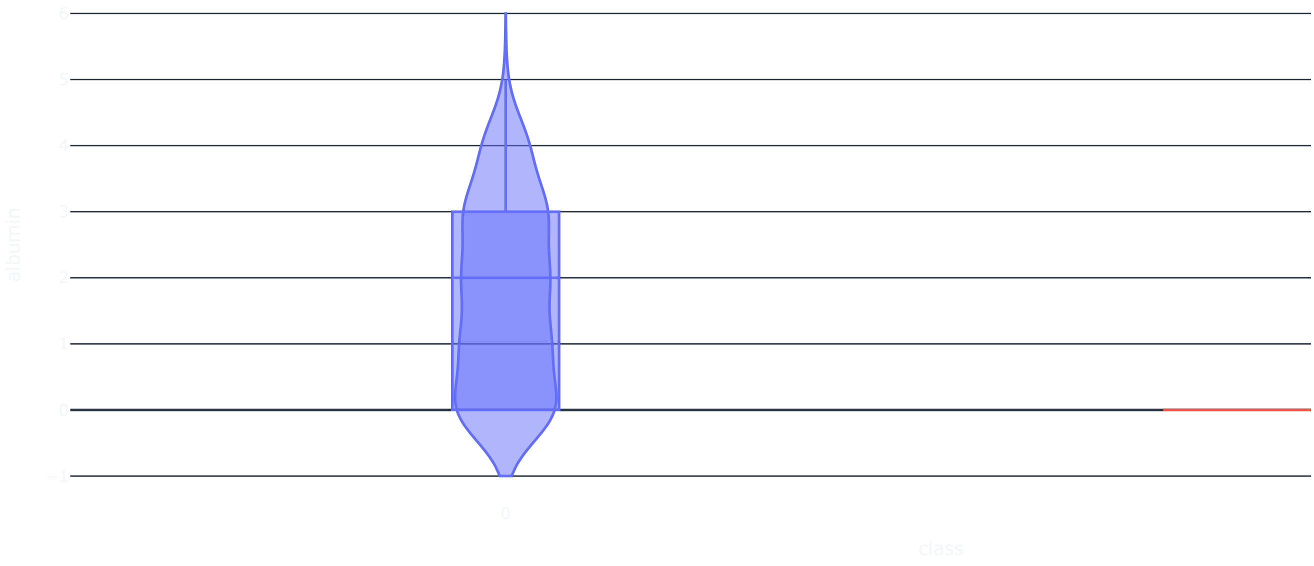

```
violin('haemoglobin')
```



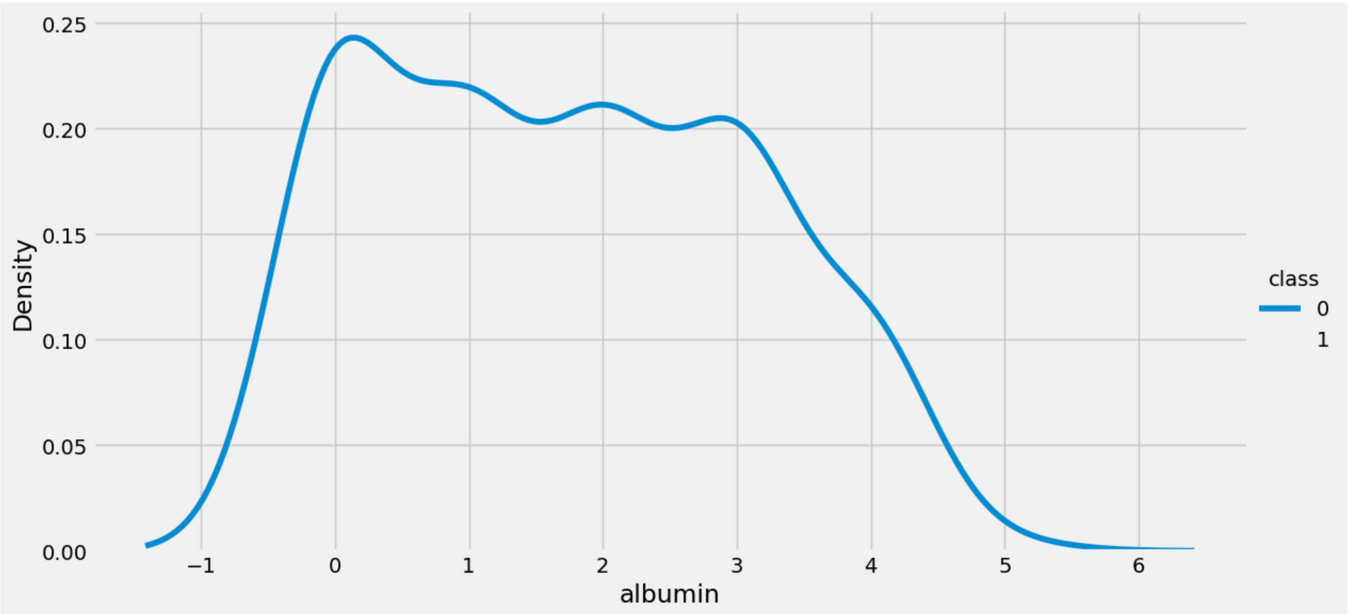
```
kde('haemoglobin')
```



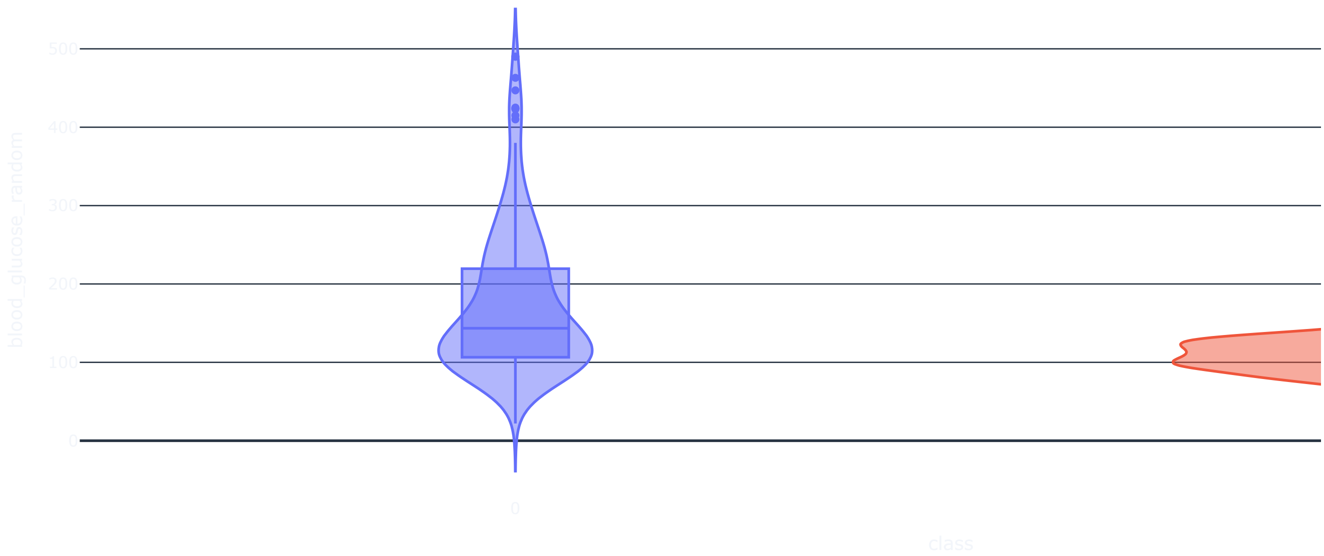

```
violin('albumin')
```



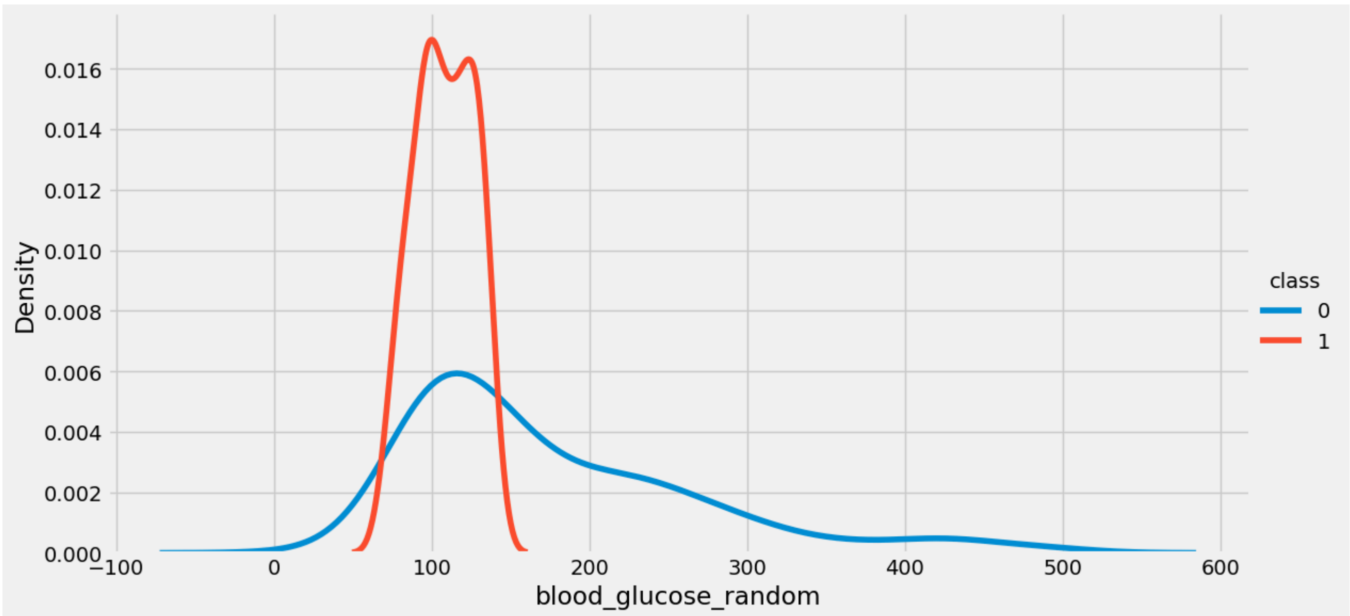
```
kde('albumin')
```



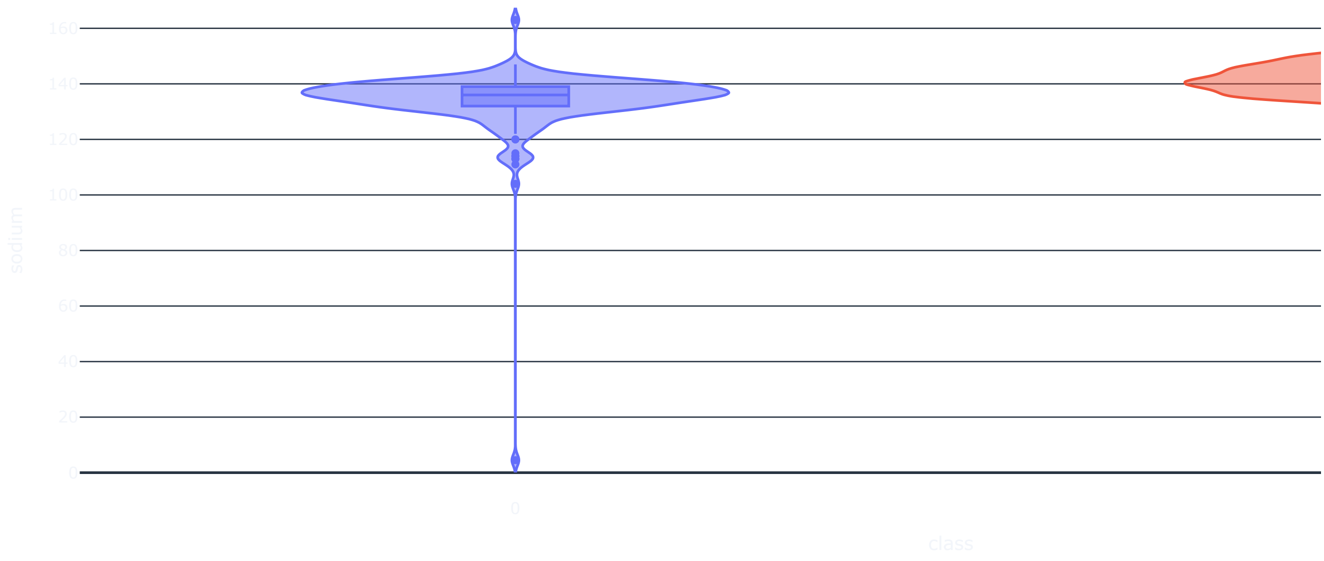

```
violin('blood_glucose_random')
```



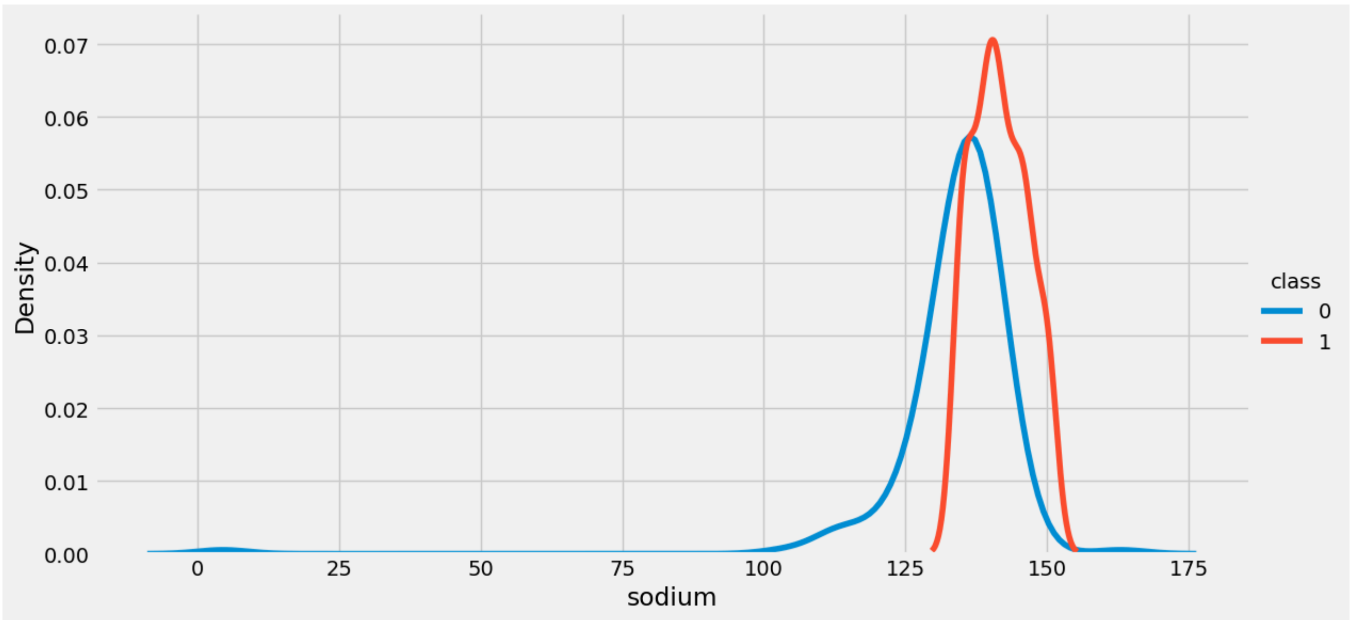

```
kde('blood_glucose_random')
```



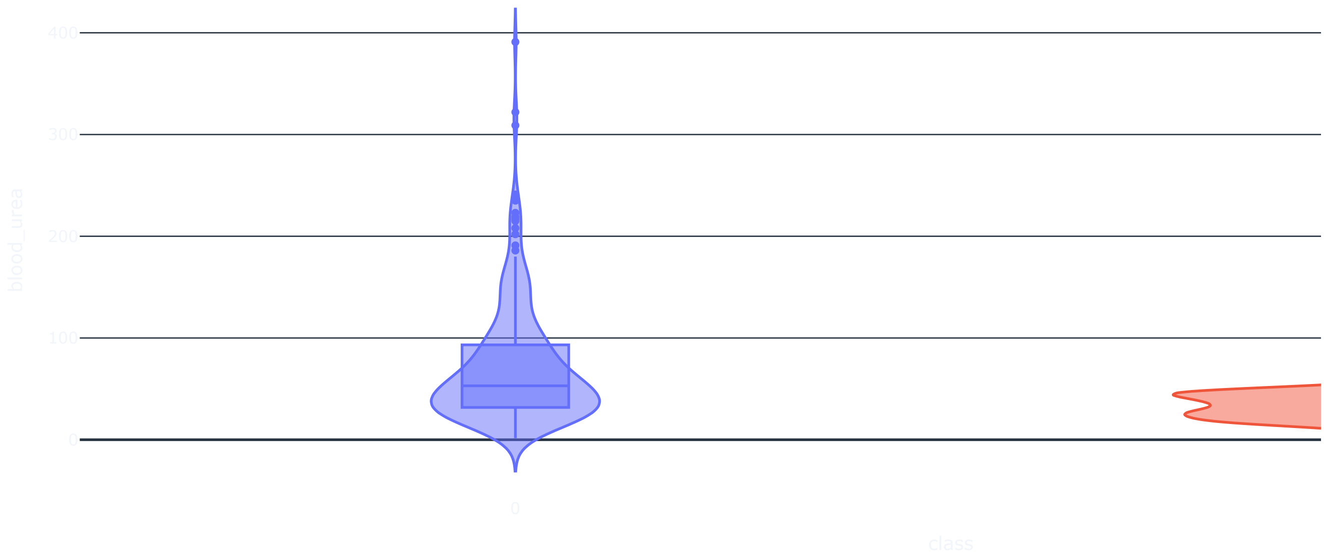

```
violin('sodium')
```



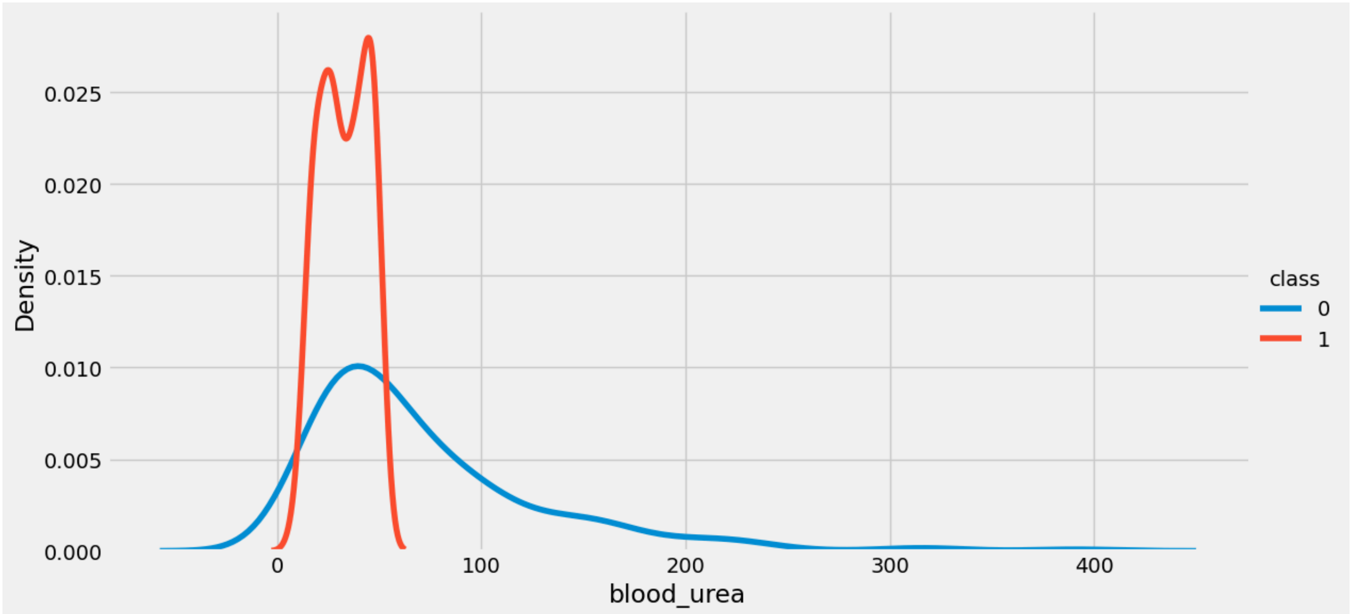

```
kde('sodium')
```



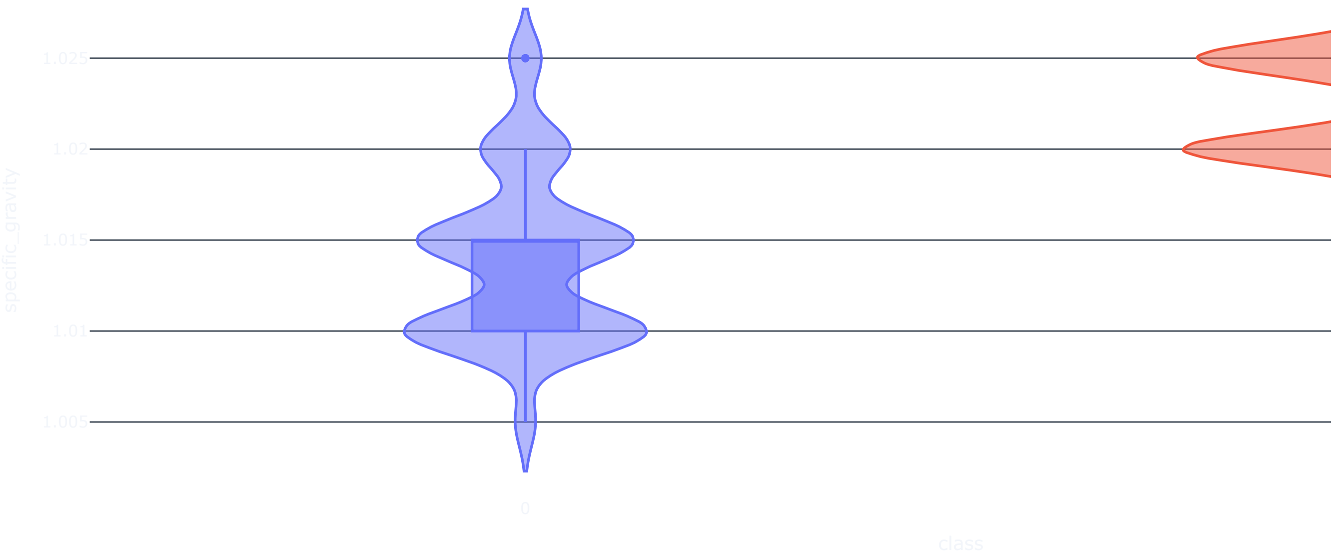

```
violin('blood_urea')
```



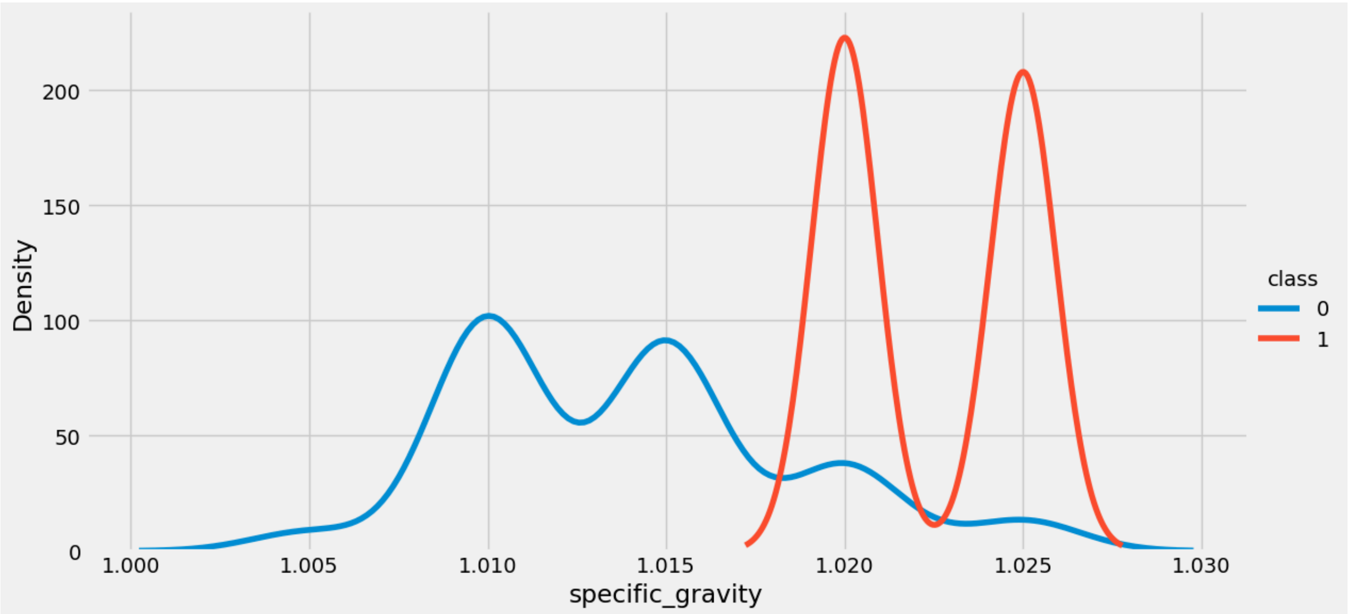

```
kde('blood_urea')
```



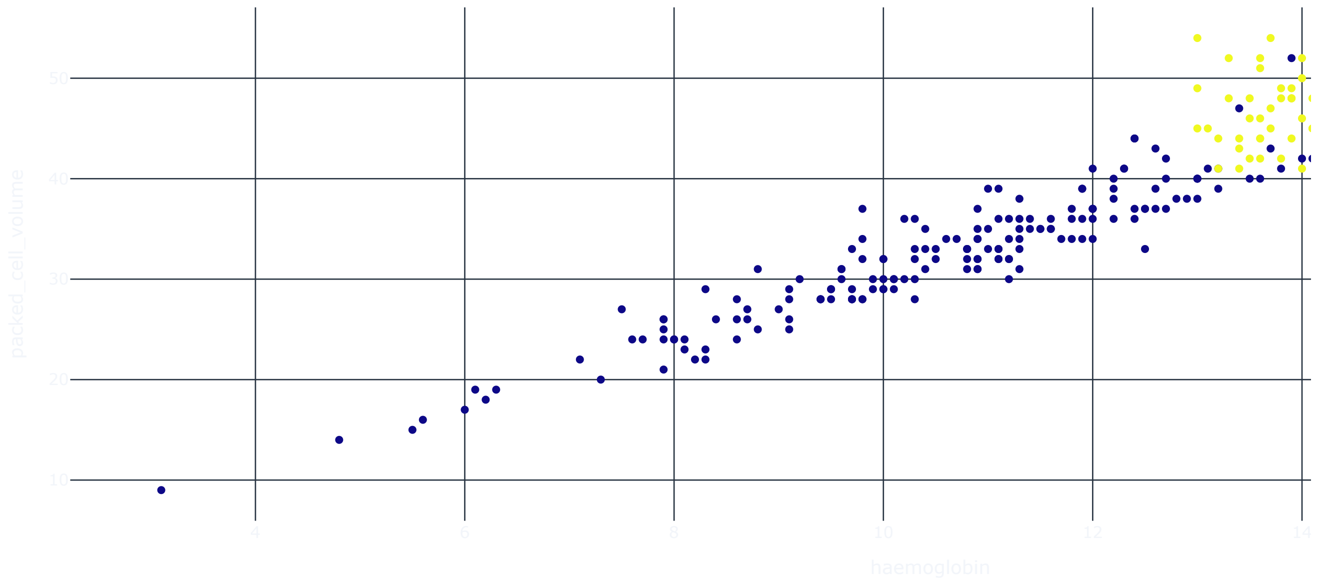

```
violin('specific_gravity')
```



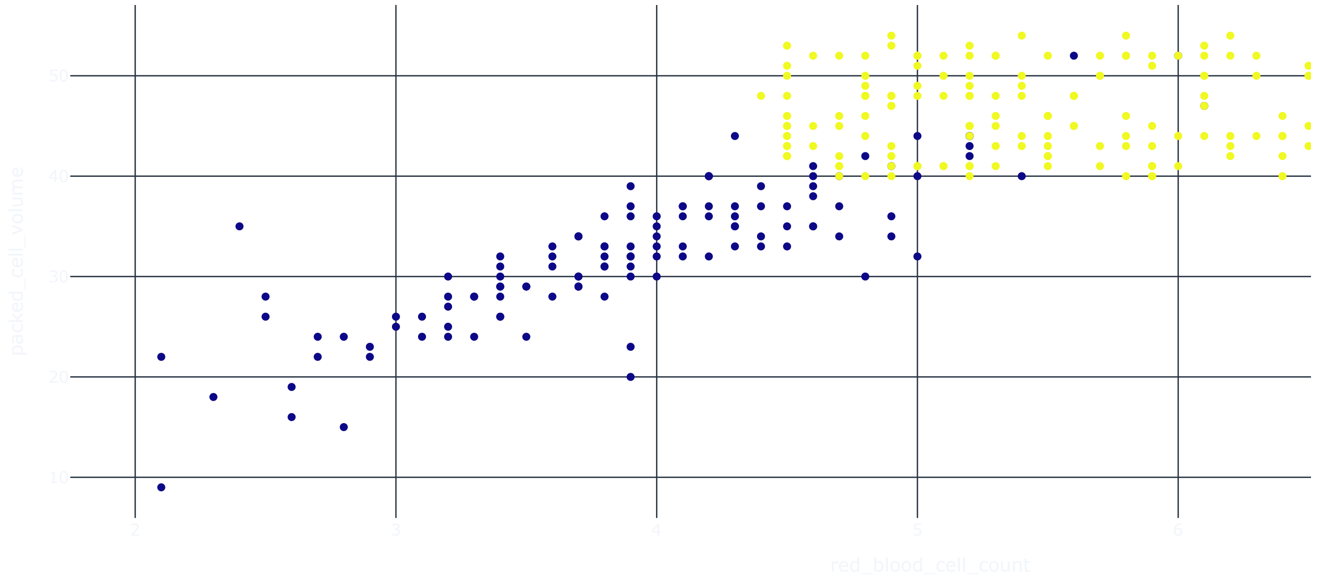

```
kde('specific_gravity')
```



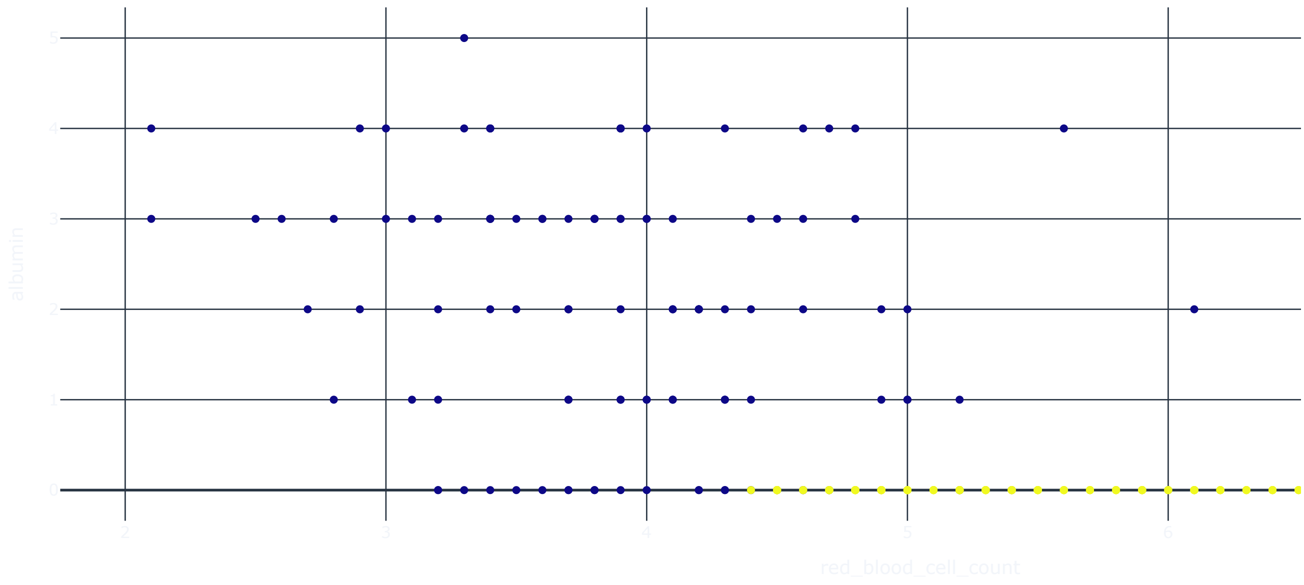

```
scatter('haemoglobin', 'packed_cell_volume')
```



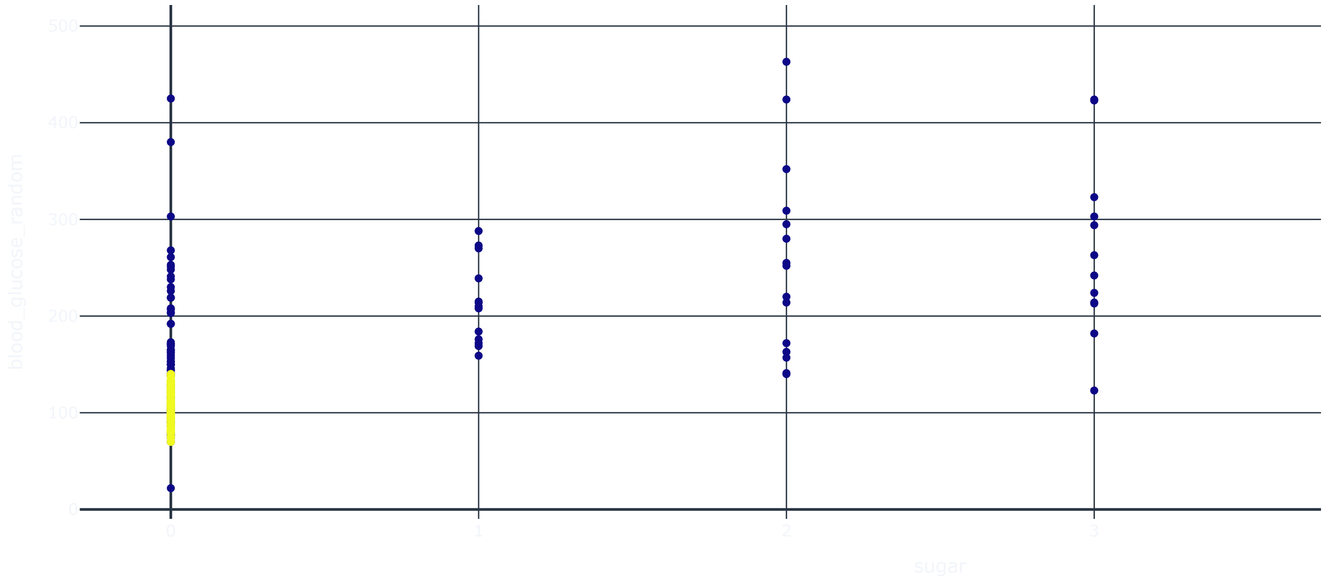

```
scatter('red_blood_cell_count', 'packed_cell_volume')
```



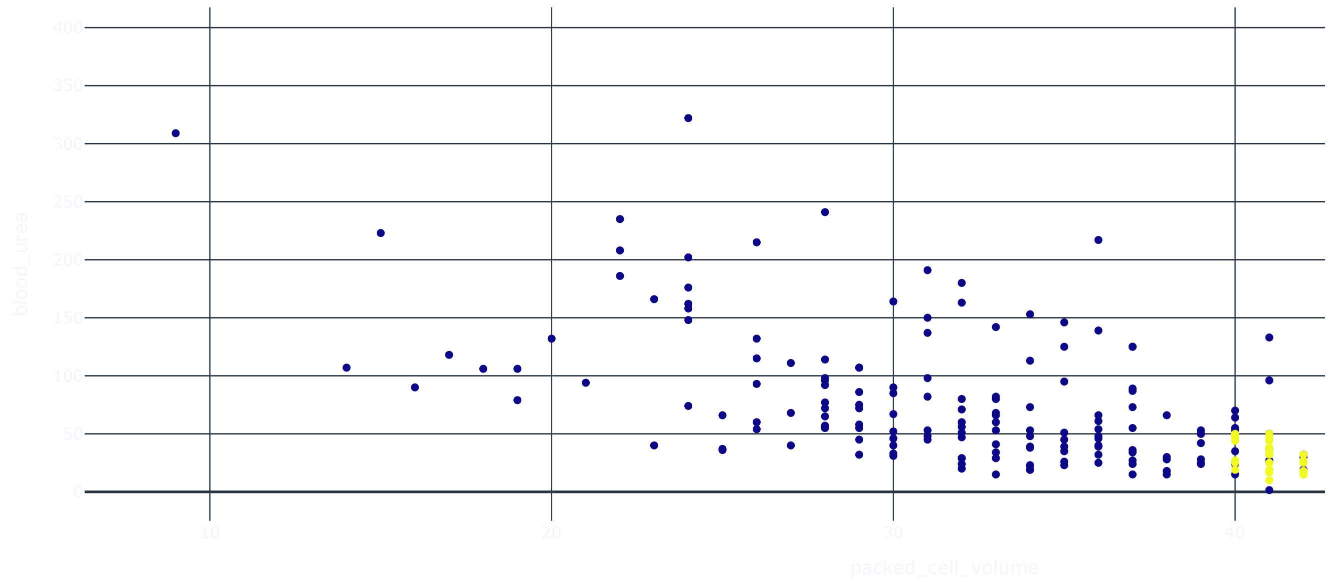

```
scatter('red_blood_cell_count', 'albumin')
```



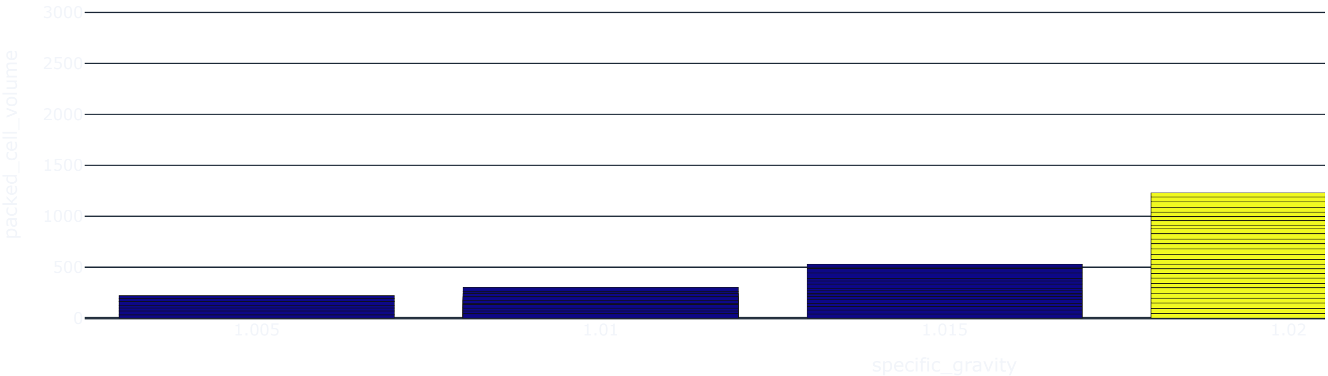

```
scatter('sugar', 'blood_glucose_random')
```



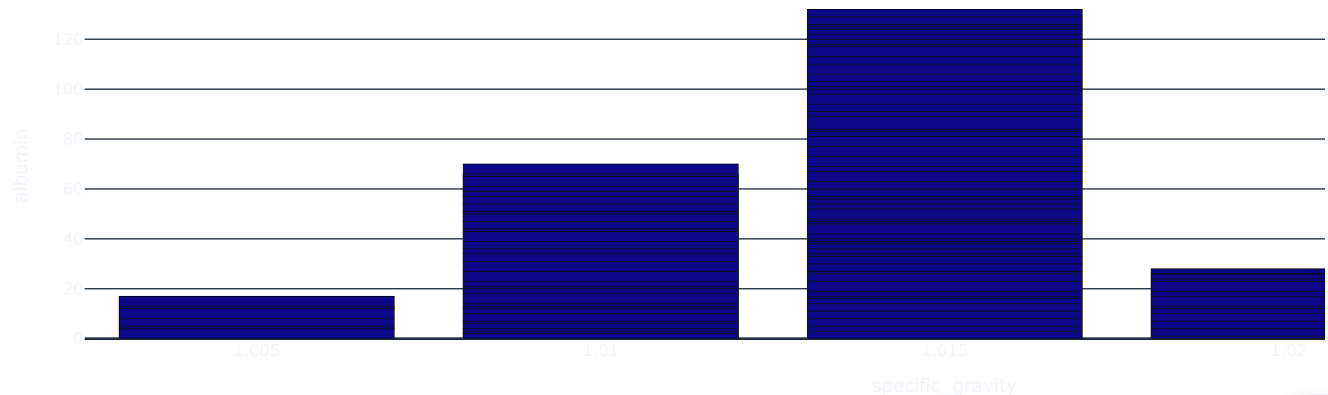

```
scatter('packed_cell_volume','blood_urea')
```



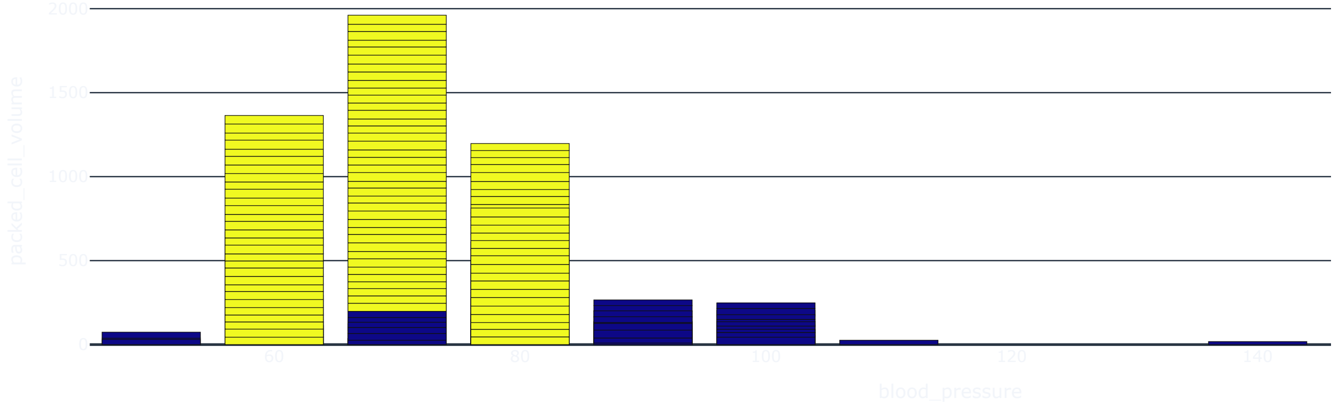

```
px.bar(df, x="specific_gravity", y="packed_cell_volume", color='class', barmode='group', template = 'plotly_dark', height =
```



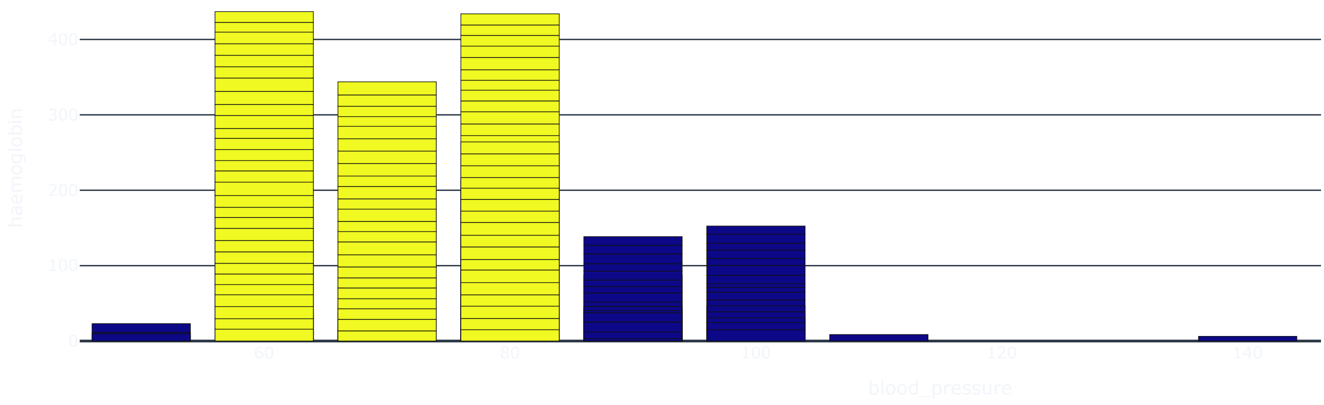

```
px.bar(df, x="specific_gravity", y="albumin", color='class', barmode='group', template = 'plotly_dark', height = 400)
```




```
px.bar(df, x="blood_pressure", y="packed_cell_volume", color='class', barmode='group', template = 'plotly_dark', height = 4
```




```
px.bar(df, x="blood_pressure", y="haemoglobin", color='class', barmode='group', template = 'plotly_dark', height = 400)
```



Data Pre Processing

```
# checking for null values

df.isna().sum().sort_values(ascending = False)
```

	0
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()
```

	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

dtype: int64

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

	0
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

```
# filling null values, we will use two methods, random sampling for higher null values and
# mean/mode sampling for lower null values

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

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


# filling num_cols null values using random sampling method

for col in num_cols:
    random_value_imputation(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

dtype: int64

```
# filling "red_blood_cells" and "pus_cell" using random sampling method and rest of cat_cols using mode imputation

random_value_imputation('red_blood_cells')
random_value_imputation('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

dtype: int64

Double-click (or enter) to edit

All the missing values are handeled now, lets do ctaegorical features encding now

Feature Encoding

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

```
red_blood_cells has 2 categories

pus_cell has 2 categories

pus_cell_clumps has 2 categories

bacteria has 2 categories

hypertension has 2 categories

diabetes_mellitus 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
```

As all of the categorical columns have 2 categories we can use 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_gluc
0	48.0	80.0	1.020	1.0	0.0	0	1	0	0	
1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	

```
ind_col = [col for col in df.columns if col != 'class']
dep_col = 'class'

X = df[ind_col]
y = df[dep_col]


# splitting data intp training and test set

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state = 0)
```

MODEL BUILDING

KNN

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

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

# accuracy score, confusion matrix and classification report of knn

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"Test Accuracy of KNN is {knn_acc} \n")

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

```
Training Accuracy of KNN is 0.775
Test Accuracy of KNN is 0.6833333333333333

Confusion Matrix :-
[[49 23]
 [15 33]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.77      0.68      0.72        72
           1       0.59      0.69      0.63        48

    accuracy                           0.68       120
   macro avg       0.68      0.68      0.68       120
weighted avg       0.70      0.68      0.69       120
```

Decision Tree Classifier Initially, the model achieved a perfect training score (100%) and 95% test accuracy, which suggests potential overfitting. After hyperparameter tuning (using GridSearchCV), a more optimal model was found, reducing overfitting slightly.

```
from sklearn.tree import DecisionTreeClassifier

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

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

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

```
Training Accuracy of Decision Tree Classifier is 1.0
Test Accuracy of Decision Tree Classifier is 0.9666666666666667

Confusion Matrix :-
[[72  0]
 [ 4 44]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        72
           1       1.00      0.92      0.96        48
```

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

```
# hyper parameter tuning of decision tree

from sklearn.model_selection import GridSearchCV
grid_param = {
    '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_param, 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
▸ best_estimator_: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

```
# best parameters and best score

print(grid_search_dtc.best_params_)
print(grid_search_dtc.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 10, 'max_features': 'log2', 'min_samples_leaf': 2, 'min_samples_split': 7, 'splitter'
0.9821428571428571
```

```
# best estimator

dtc = grid_search_dtc.best_estimator_

# accuracy score, confusion matrix and classification report of decision tree

dtc_acc = accuracy_score(y_test, dtc.predict(X_test))

print(f"Training Accuracy of Decision Tree Classifier is {accuracy_score(y_train, dtc.predict(X_train))}")
print(f"Test Accuracy of Decision Tree Classifier is {dtc_acc} \n")

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

```
Training Accuracy of Decision Tree Classifier is 0.9607142857142857
Test Accuracy of Decision Tree Classifier is 0.9333333333333333

Confusion Matrix :-
[[68  4]
 [ 4 44]]

Classification Report :-
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        72
           1       0.92      0.92      0.92        48

    accuracy                           0.93       120
   macro avg       0.93      0.93      0.93       120
weighted avg       0.93      0.93      0.93       120
```

Random Forest Classifier An ensemble method using bagging to combine multiple decision trees. Achieved 97% test accuracy, with excellent performance across precision, recall, and f1-score metrics.

Double-click (or enter) to edit

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
rd_clf = RandomForestClassifier(criterion='entropy', max_depth=11, max_features='sqrt',
                                 min_samples_leaf=2, min_samples_split=3, n_estimators=130)
rd_clf.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of random forest
# Assuming X_train, y_train, X_test, and y_test are already defined



# accuracy score, confusion matrix and classification report of random forest
rd_clf_acc = accuracy_score(y_test, rd_clf.predict(X_test))

print(f"Training Accuracy of Random Forest Classifier is {accuracy_score(y_train, rd_clf.predict(X_train))}")
print(f"Test Accuracy of Random Forest Classifier is {rd_clf_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, rd_clf.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, rd_clf.predict(X_test))}")
```

```
Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 0.975

Confusion Matrix :-
[[72  0]
 [ 3 45]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

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

Ada Boost Classifier This boosting method built a sequence of models to correct the errors from previous ones. Achieved a high test accuracy of 98.3%, indicating that it handled the data and class separation very well

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize the weak learner
dtc = DecisionTreeClassifier(max_depth=1)

# Initialize AdaBoostClassifier without base_estimator or using `estimator` parameter
ada = AdaBoostClassifier(estimator=dtc)  # Use 'estimator' if 'base_estimator' is not accepted
ada.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of ada boost

ada_acc = accuracy_score(y_test, ada.predict(X_test))

print(f"Training Accuracy of Ada Boost Classifier is {accuracy_score(y_train, ada.predict(X_train))}")
print(f"Test Accuracy of Ada Boost Classifier is {ada_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, ada.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, ada.predict(X_test))}")
```

```
Training Accuracy of Ada Boost Classifier is 1.0
Test Accuracy of Ada Boost Classifier is 0.975

Confusion Matrix :-
[[72  0]
 [ 3 45]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

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

Gradient Boosting Classifier Similar to AdaBoost but typically more robust and efficient for large datasets. Gradient boosting iteratively reduces prediction errors and performs well on this dataset, achieving strong results.

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of gradient boosting classifier

gb_acc = accuracy_score(y_test, gb.predict(X_test))

print(f"Training Accuracy of Gradient Boosting Classifier is {accuracy_score(y_train, gb.predict(X_train))}")
print(f"Test Accuracy of Gradient Boosting Classifier is {gb_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, gb.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, gb.predict(X_test))}")
```

```
Training Accuracy of Gradient Boosting Classifier is 1.0
Test Accuracy of Gradient Boosting Classifier is 0.9666666666666667

Confusion Matrix :-
[[72  0]
 [ 4 44]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        72
           1       1.00      0.92      0.96        48

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

XgBoost

```
from xgboost import XGBClassifier

xgb = XGBClassifier(objective = 'binary:logistic', learning_rate = 0.5, max_depth = 5, n_estimators = 150)
xgb.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of xgboost

xgb_acc = accuracy_score(y_test, xgb.predict(X_test))

print(f"Training Accuracy of XgBoost is {accuracy_score(y_train, xgb.predict(X_train))}")
print(f"Test Accuracy of XgBoost is {xgb_acc} \n")

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

```
Training Accuracy of XgBoost is 1.0
Test Accuracy of XgBoost is 0.975

Confusion Matrix :-
[[72  0]
 [ 3 45]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

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

```
pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->cat
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (20
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (
```

```
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (2
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (10.
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 7.8 MB/s eta 0:00:00
Installing collected packages: catboost
Successfully installed catboost-1.2.7
```

```
from catboost import CatBoostClassifier

cat = CatBoostClassifier(iterations=10)
cat.fit(X_train, y_train)
```

```
Learning rate set to 0.408198
0:	learn: 0.2765163	total: 48.7ms	remaining: 439ms
1:	learn: 0.1454991	total: 50.8ms	remaining: 203ms
2:	learn: 0.0840350	total: 52.7ms	remaining: 123ms
3:	learn: 0.0633415	total: 54.6ms	remaining: 81.9ms
4:	learn: 0.0456211	total: 56.4ms	remaining: 56.4ms
5:	learn: 0.0374016	total: 58.4ms	remaining: 39ms
6:	learn: 0.0280000	total: 60.3ms	remaining: 25.8ms
7:	learn: 0.0220919	total: 62ms	remaining: 15.5ms
8:	learn: 0.0197883	total: 63.8ms	remaining: 7.09ms
9:	learn: 0.0169266	total: 65.7ms	remaining: 0us
<catboost.core.CatBoostClassifier at 0x78fc214c36a0>
```

```
# accuracy score, confusion matrix and classification report of cat boost

cat_acc = accuracy_score(y_test, cat.predict(X_test))

print(f"Training Accuracy of Cat Boost Classifier is {accuracy_score(y_train, cat.predict(X_train))}")
print(f"Test Accuracy of Cat Boost Classifier is {cat_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, cat.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, cat.predict(X_test))}")
```

```
Training Accuracy of Cat Boost Classifier is 1.0
Test Accuracy of Cat Boost Classifier is 0.975

Confusion Matrix :-
[[72  0]
 [ 3 45]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

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

Extra Trees Classifier

```
from sklearn.ensemble import ExtraTreesClassifier

etc = ExtraTreesClassifier()
etc.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of extra trees classifier

etc_acc = accuracy_score(y_test, etc.predict(X_test))

print(f"Training Accuracy of Extra Trees Classifier is {accuracy_score(y_train, etc.predict(X_train))}")
print(f"Test Accuracy of Extra Trees Classifier is {etc_acc} \n")

print(f"Confusion Matrix :- \n{confusion_matrix(y_test, etc.predict(X_test))}\n")
print(f"Classification Report :- \n {classification_report(y_test, etc.predict(X_test))}")
```

```
Training Accuracy of Extra Trees Classifier is 1.0
Test Accuracy of Extra Trees Classifier is 0.975

Confusion Matrix :-
[[72  0]
 [ 3 45]]

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

```
           0       0.96      1.00      0.98        72
           1       1.00      0.94      0.97        48

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

LGBM Classifier

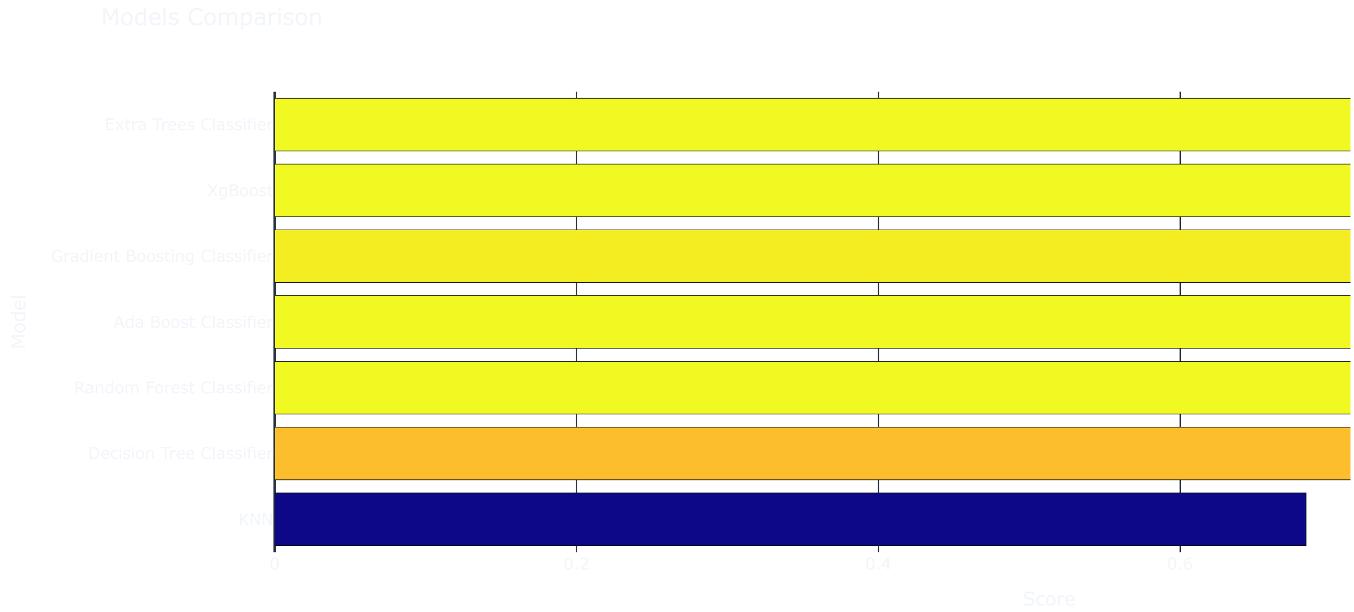
```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(learning_rate = 1)
lgbm.fit(X_train, y_train)

# accuracy score, confusion matrix and classification report of lgbm classifier

lgbm_acc = accuracy_score(y_test, lgbm.predict(X_test))

print(f"Training Accuracy of LGBM Classifier is {accuracy_score(y_train, lgbm.predict(X_train))}")
print(f"Test Accuracy of LGBM Classifier is {lgbm_acc} \n")

print(f"{confusion_matrix(y_test, lgbm.predict(X_test))}\n")
print(classification_report(y_test, lgbm.predict(X_test)))
```

```
[LightGBM] [Info] Number of positive: 102, number of negative: 178
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000614 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 519
[LightGBM] [Info] Number of data points in the train set: 280, number of used features: 23
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.364286 -> initscore=-0.556811
[LightGBM] [Info] Start training from score -0.556811
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
models = pd.DataFrame({
    'Model' : [ 'KNN', 'Decision Tree Classifier', 'Random Forest Classifier','Ada Boost Classifier',
             'Gradient Boosting Classifier', 'XgBoost', 'Extra Trees Classifier'],
    'Score' : [knn_acc, dtc_acc, rd_clf_acc, ada_acc, gb_acc, xgb_acc, etc_acc]
})


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

	Model	Score
2	Random Forest Classifier	0.975000
3	Ada Boost Classifier	0.975000
5	XgBoost	0.975000
6	Extra Trees Classifier	0.975000
4	Gradient Boosting Classifier	0.966667
1	Decision Tree Classifier	0.933333
0	KNN	0.683333

```
px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score', template = 'plotly_dark',
       title = 'Models Comparison')
```

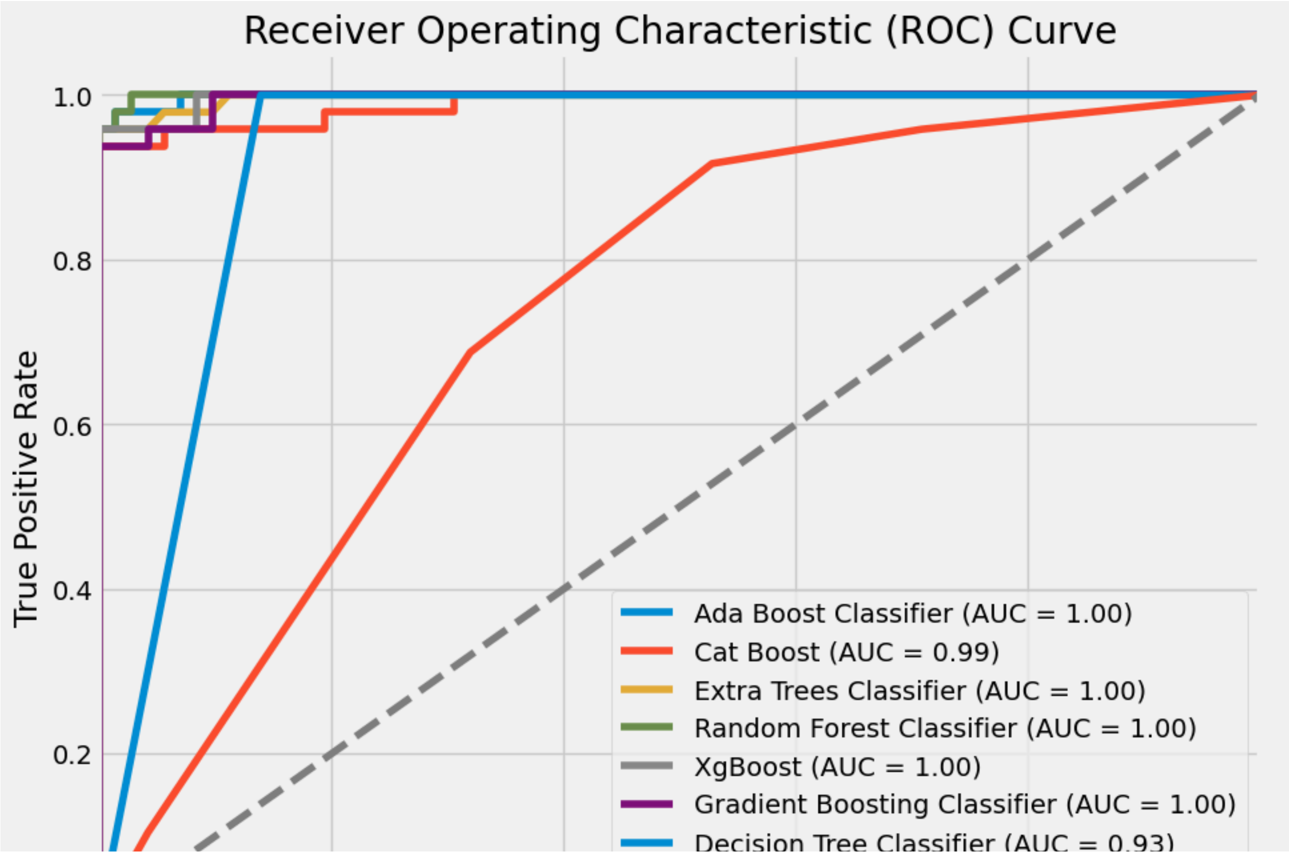



```
# Fit each model before using it
knn.fit(X_train, y_train)
dtc.fit(X_train, y_train)
rd_clf.fit(X_train, y_train)
ada.fit(X_train, y_train)
gb.fit(X_train, y_train)
xgb.fit(X_train, y_train)
cat.fit(X_train, y_train)
etc.fit(X_train, y_train)

# Dictionary of models' predicted probabilities
y_pred_probas = {
    knn_acc: knn.predict_proba(X_test)[:, 1],
    dtc_acc: dtc.predict_proba(X_test)[:, 1],
    rd_clf_acc: rd_clf.predict_proba(X_test)[:, 1],
    ada_acc: ada.predict_proba(X_test)[:, 1],
    gb_acc: gb.predict_proba(X_test)[:, 1],
    xgb_acc: xgb.predict_proba(X_test)[:, 1],
    cat_acc: cat.predict_proba(X_test)[:, 1],
    etc_acc: etc.predict_proba(X_test)[:, 1]
}

# Now continue with the ROC curve plotting code...
```

```
Learning rate set to 0.408198
0:	learn: 0.2765163	total: 3.74ms	remaining: 33.6ms
1:	learn: 0.1454991	total: 6.3ms	remaining: 25.2ms
2:	learn: 0.0840350	total: 8.48ms	remaining: 19.8ms
3:	learn: 0.0633415	total: 10.2ms	remaining: 15.4ms
4:	learn: 0.0456211	total: 12.3ms	remaining: 12.3ms
5:	learn: 0.0374016	total: 16.1ms	remaining: 10.7ms
6:	learn: 0.0280000	total: 19.3ms	remaining: 8.29ms
7:	learn: 0.0220919	total: 22.1ms	remaining: 5.52ms
8:	learn: 0.0197883	total: 24.5ms	remaining: 2.73ms
9:	learn: 0.0169266	total: 26.8ms	remaining: 0us
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split

# Ensure y_true has the correct length by using y_test directly
y_true = y_test  # Replace with your actual y_test labels for X_test

# Re-run the model fitting and prediction code as before
knn.fit(X_train, y_train)
dtc.fit(X_train, y_train)
rd_clf.fit(X_train, y_train)
ada.fit(X_train, y_train)
gb.fit(X_train, y_train)
xgb.fit(X_train, y_train)
cat.fit(X_train, y_train)
etc.fit(X_train, y_train)

# Dictionary of models' predicted probabilities
y_pred_probas = {
    "Ada Boost Classifier": ada.predict_proba(X_test)[:, 1],
    "Gradient Boosting Classifier": gb.predict_proba(X_test)[:, 1],
    "Cat Boost": cat.predict_proba(X_test)[:, 1],
    "Extra Trees Classifier": etc.predict_proba(X_test)[:, 1],
    "Random Forest Classifier": rd_clf.predict_proba(X_test)[:, 1],
    "XgBoost": xgb.predict_proba(X_test)[:, 1],
    "Decision Tree Classifier": dtc.predict_proba(X_test)[:, 1],
    "KNN": knn.predict_proba(X_test)[:, 1]
}

# Sort models by score
models = pd.DataFrame({
    'Model': ["Ada Boost Classifier", "Gradient Boosting Classifier", "Cat Boost", "Extra Trees Classifier",
              "Random Forest Classifier", "XgBoost", "Decision Tree Classifier", "KNN"],
    'Score': [ada_acc, gb_acc, cat_acc, etc_acc, rd_clf_acc, xgb_acc, dtc_acc, knn_acc]
})
sorted_models = models.sort_values(by='Score', ascending=False)

# Plot ROC Curve
plt.figure(figsize=(10, 8))

for model_name in sorted_models['Model']:
    y_pred_proba = y_pred_probas[model_name]

    # Check that y_true and y_pred_proba have the same length
    assert len(y_true) == len(y_pred_proba), f"Length mismatch: {len(y_true)} vs {len(y_pred_proba)} for {model_name}"

    fpr, tpr, _ = roc_curve(y_true, y_pred_proba)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot settings
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
Learning rate set to 0.408198
0:	learn: 0.2765163	total: 7.41ms	remaining: 66.7ms
1:	learn: 0.1454991	total: 17.6ms	remaining: 70.6ms
2:	learn: 0.0840350	total: 23.3ms	remaining: 54.4ms
3:	learn: 0.0633415	total: 33ms	remaining: 49.5ms
4:	learn: 0.0456211	total: 51.6ms	remaining: 51.6ms
5:	learn: 0.0374016	total: 62ms	remaining: 41.3ms
6:	learn: 0.0280000	total: 77.7ms	remaining: 33.3ms
7:	learn: 0.0220919	total: 93.7ms	remaining: 23.4ms
8:	learn: 0.0197883	total: 96.4ms	remaining: 10.7ms
9:	learn: 0.0169266	total: 101ms	remaining: 0us
```




```python
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
import pandas as pd

# Assuming y_true is already defined as the true labels for X_test, as in the previous setup
y_true = y_test  # Use actual binary labels of X_test

# Assuming each model has been fitted and y_pred_probas contains probability predictions for X_test
y_pred_probas = {
    "Ada Boost Classifier": ada.predict_proba(X_test)[:, 1],
    "Gradient Boosting Classifier": gb.predict_proba(X_test)[:, 1],
    "Cat Boost": cat.predict_proba(X_test)[:, 1],
    "Extra Trees Classifier": etc.predict_proba(X_test)[:, 1],
    "Random Forest Classifier": rd_clf.predict_proba(X_test)[:, 1],
    "XgBoost": xgb.predict_proba(X_test)[:, 1],
    "Decision Tree Classifier": dtc.predict_proba(X_test)[:, 1],
    "KNN": knn.predict_proba(X_test)[:, 1]
}

# Sort models by score for ordered plotting
models = pd.DataFrame({
    'Model': ["Ada Boost Classifier", "Gradient Boosting Classifier", "Cat Boost", "Extra Trees Classifier",
              "Random Forest Classifier", "XgBoost", "Decision Tree Classifier", "KNN"],
    'Score': [ada_acc, gb_acc, cat_acc, etc_acc, rd_clf_acc, xgb_acc, dtc_acc, knn_acc]
})
sorted_models = models.sort_values(by='Score', ascending=False)

# Plot ROC and AUC for each model
plt.figure(figsize=(10, 8))

for model_name in sorted_models['Model']:
    y_pred_proba = y_pred_probas[model_name]
```