In [1]:
```python
# 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)
```

In [11]:
```python
# loading data

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

Out[11]:

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

In [12]:
```python
df.shape
```

Out[12]:
```
(400, 26)
```

In [13]:
```python
# dropping id column
df.drop('id', axis = 1, inplace = True)
```

In [14]:
```python
df.head()
```

Out[14]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
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	no	good	no	no	ckd
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	no	good	no	no	ckd
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	no	poor	no	yes	ckd
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	no	poor	yes	yes	ckd
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	no	good	no	no	ckd

In [15]:
```python
# 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']
```

In [16]:
```python
df.head()
```

Out[16]:

	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
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
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
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
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
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

In [17]:
```python
df.describe()
```

Out[17]:

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_urea	serum_creatinine	sodium	potassium	haemoglobin
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	383.000000	313.000000	312.000000	348.000000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	3.072454	137.528754	4.627244	12.526437
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	5.741126	10.408752	3.193904	2.912587
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	0.400000	4.500000	2.500000	3.100000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	0.900000	135.000000	3.800000	10.300000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	1.300000	138.000000	4.400000	12.650000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	2.800000	142.000000	4.900000	15.000000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	76.000000	163.000000	47.000000	17.800000

In [18]:
```python
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
```

In [19]:
```python
# 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')
```

In [20]:
```python
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
```

In [21]:
```python
# 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']
```

In [22]:
```python
# 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
```

In [23]:
```python
# 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'})
```

In [24]:
```python
df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

In [25]:
```python
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
```

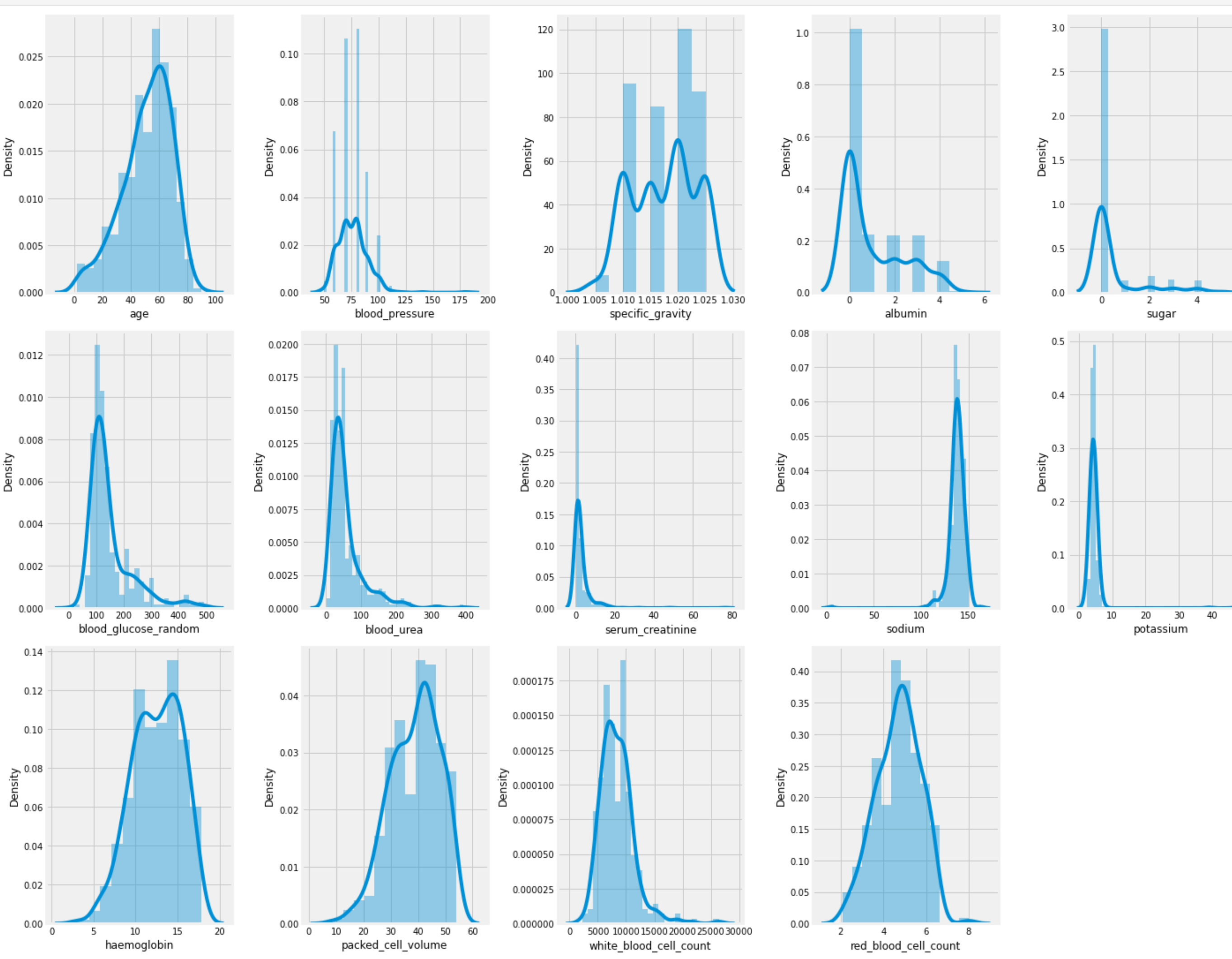
In [26]:
```python
# checking numerical features distribution

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()
```



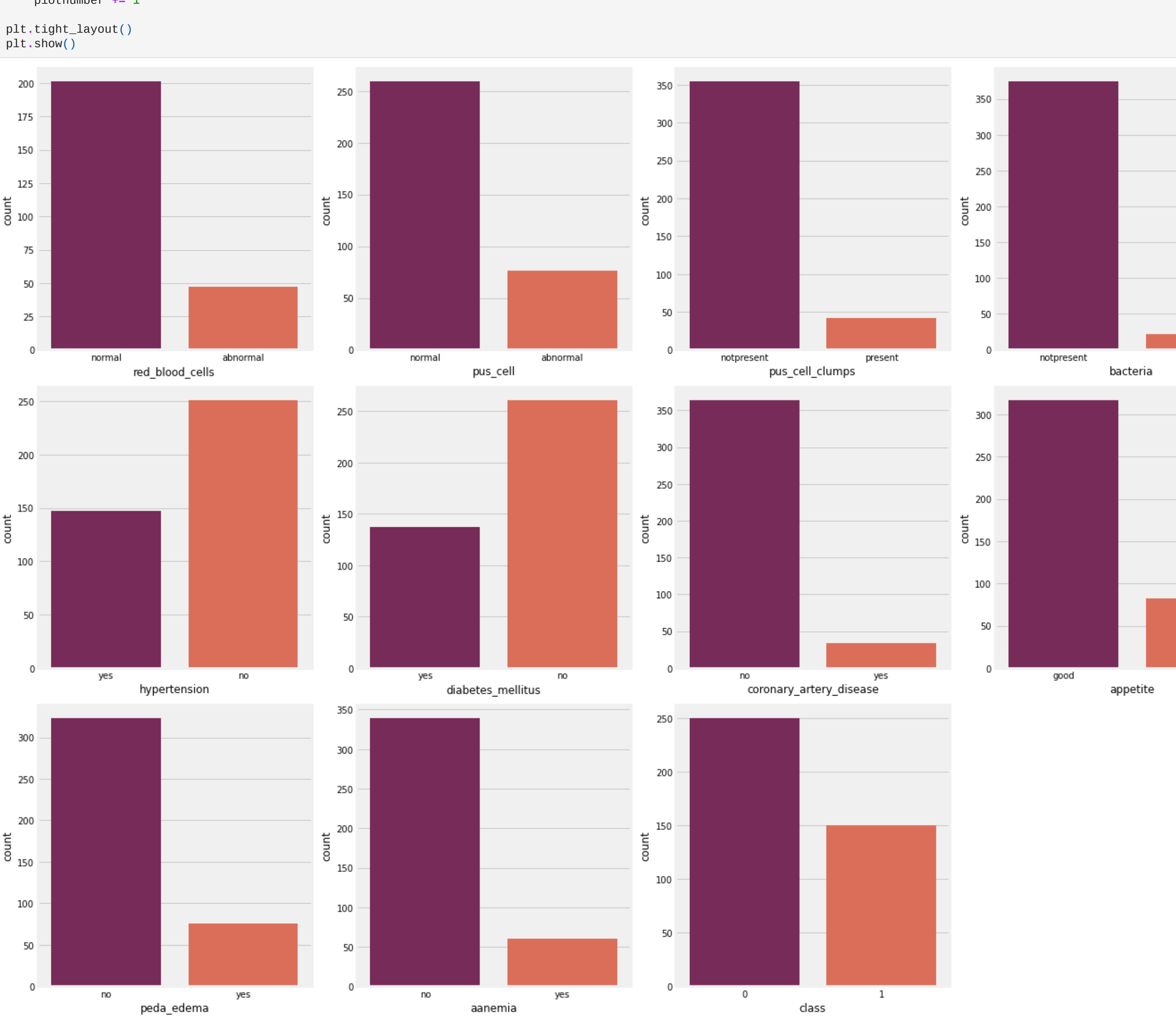
In [27]:
```python
# looking at categorical columns

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()
```



In [28]:
```python
df.columns
```

Out[28]:
```
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')
```

In [29]:
```python
# checking for null values

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

Out[29]:
```
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
aanemia                      1
appetite                     1
peda_edema                   1
class                        0
dtype: int64
```

In [30]:
```python
df[num_cols].isnull().sum()
```

Out[30]:
```
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
```

In [31]:
```python
df[cat_cols].isnull().sum()
```

Out[31]:
```
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
```

In [32]:
```python
# 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)
```

In [33]:
```python
# filling num_cols null values using random sampling method

for col in num_cols:
    random_value_imputation(col)
```

In [34]:
```python
df[num_cols].isnull().sum()
```

Out[34]:
```
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
```

In [35]:
```python
# 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)
```

In [36]:
```python
df[cat_cols].isnull().sum()
```

Out[36]:
```
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
```

In [37]:
```python
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
```

In [38]:
```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

In [39]:
```python
df.head()
```

Out[39]:

	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
0	48.0	80.0	1.020	1.0	0.0	0	1	0	0	121.0	36.0	1.2	142.0	4.8	15.4	44.0
1	7.0	50.0	1.020	4.0	0.0	1	1	0	0	424.0	18.0	0.8	135.0	4.4	11.3	38.0
2	62.0	80.0	1.010	2.0	3.0	1	1	0	0	423.0	53.0	1.8	131.0	3.6	9.6	31.0
3	48.0	70.0	1.005	4.0	0.0	1	0	1	0	117.0	56.0	3.8	111.0	2.5	11.2	32.0
4	51.0	80.0	1.010	2.0	0.0	1	1	0	0	106.0	26.0	1.4	135.0	4.2	11.6	35.0

In [40]:
```python
#model building

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

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

In [41]:
```python
# 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)
```

In [42]:
```python
#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 0.7678571428571429
Test Accuracy of KNN is 0.6583333333333333 

Confusion Matrix :- 
[[48 24]
 [17 31]]

Classification Report :- 
               precision    recall  f1-score   support

           0       0.74      0.67      0.70        72
           1       0.56      0.65      0.60        48

    accuracy                           0.66       120
   macro avg       0.65      0.66      0.65       120
weighted avg       0.67      0.66      0.66       120
```

In [43]:
```python
#Decision Tree Classifier

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 0.9416666666666667 

Confusion Matrix :- 
[[69  3]
 [ 4 44]]

Classification Report :- 
               precision    recall  f1-score   support

           0       0.95      0.96      0.95        72
           1       0.94      0.92      0.93        48

    accuracy                           0.94       120
   macro avg       0.94      0.94      0.94       120
weighted avg       0.94      0.94      0.94       120
```

In [44]:
```python
#Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'auto', 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

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 0.9833333333333333 

Confusion Matrix :- 
[[72  0]
 [ 2 46]]

Classification Report :- 
               precision    recall  f1-score   support

           0       0.97      1.00      0.99        72
           1       1.00      0.96      0.98        48

    accuracy                           0.98       120
   macro avg       0.99      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```

In [45]:
```python
#Ada Boost Classifier

from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(base_estimator = dtc)
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 0.9416666666666667 

Confusion Matrix :- 
[[68  4]
 [ 3 45]]

Classification Report :- 
               precision    recall  f1-score   support

           0       0.96      0.94      0.95        72
           1       0.92      0.94      0.93        48

    accuracy                           0.94       120
   macro avg       0.94      0.94      0.94       120
weighted avg       0.94      0.94      0.94       120
```

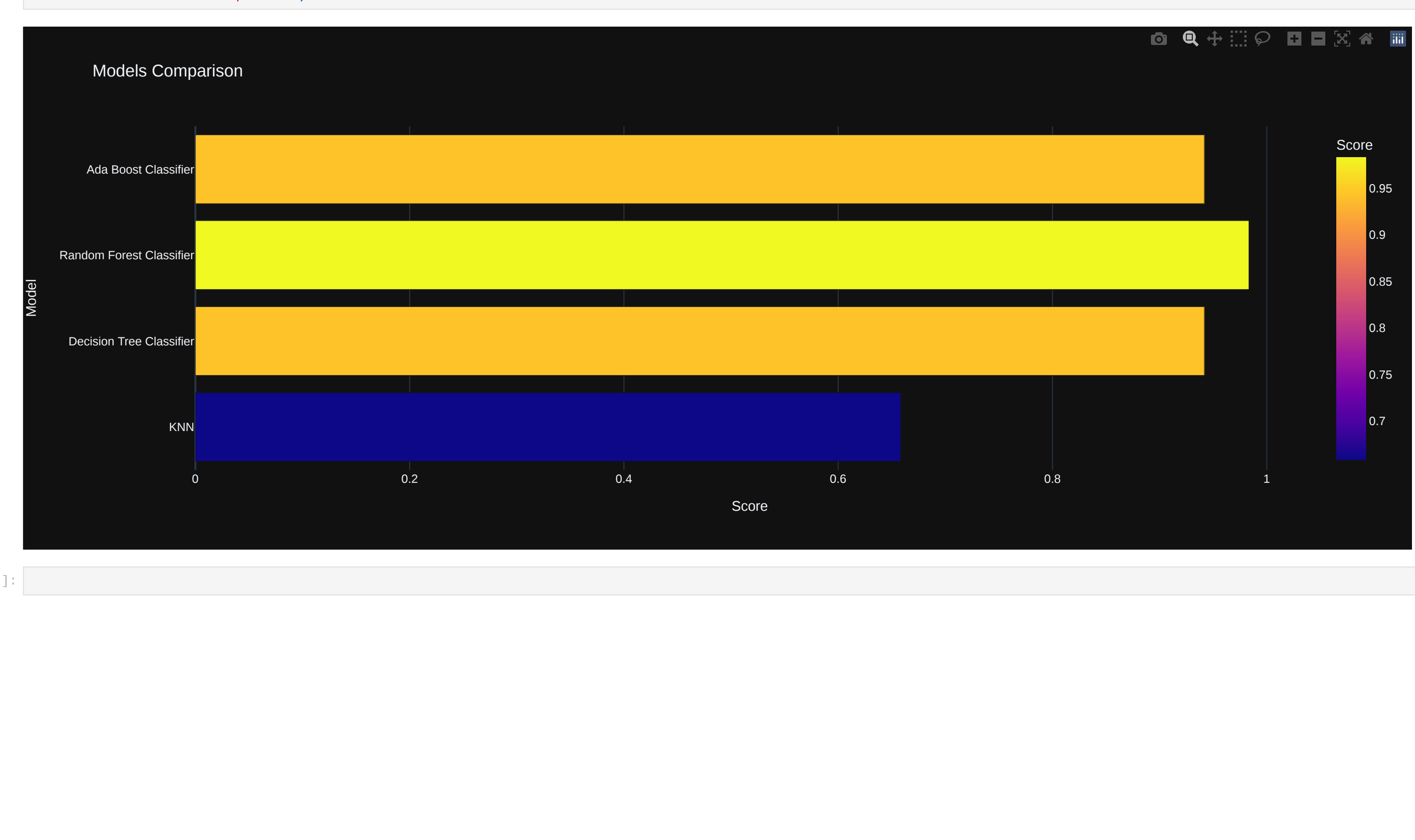
In [48]:
```python
#Models Comparision

models = pd.DataFrame({
    'Model' : [ 'KNN', 'Decision Tree Classifier', 'Random Forest Classifier','Ada Boost Classifier'],
    'Score' : [knn_acc, dtc_acc, rd_clf_acc, ada_acc]
})
models.sort_values(by = 'Score', ascending = False)
```

Out[48]:

	Model	Score
2	Random Forest Classifier	0.983333
1	Decision Tree Classifier	0.941667
3	Ada Boost Classifier	0.941667
0	KNN	0.658333

In [49]:
```python
px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score', template = 'plotly_dark',
       title = 'Models Comparison')
```



In [ ]: