In [122]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
%matplotlib inline
pd.set_option('display.max_columns', 26)
```

In [9]:
```
df = pd.read_csv('kidney_disease.csv')
df.head()
```

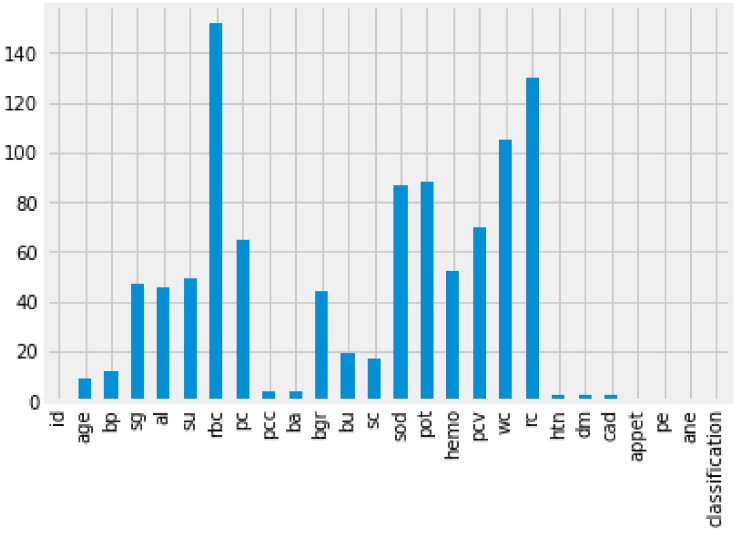
Out[9]:

bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hemo	pcv	wc	rc
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
50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	11.3	38	6000	NaN
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
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
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

In [10]:
```
# dropping id column
df.drop('id', axis = 1, inplace = True)
```

In [13]:
```
#rename
df.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_count',
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetite',
              'aanemia', 'class']
```

In [120]:
```
# Here we are plotting the graph to see the null values in the dataset.
p = df.isnull().sum().plot.bar()
```



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

Out[14]:

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

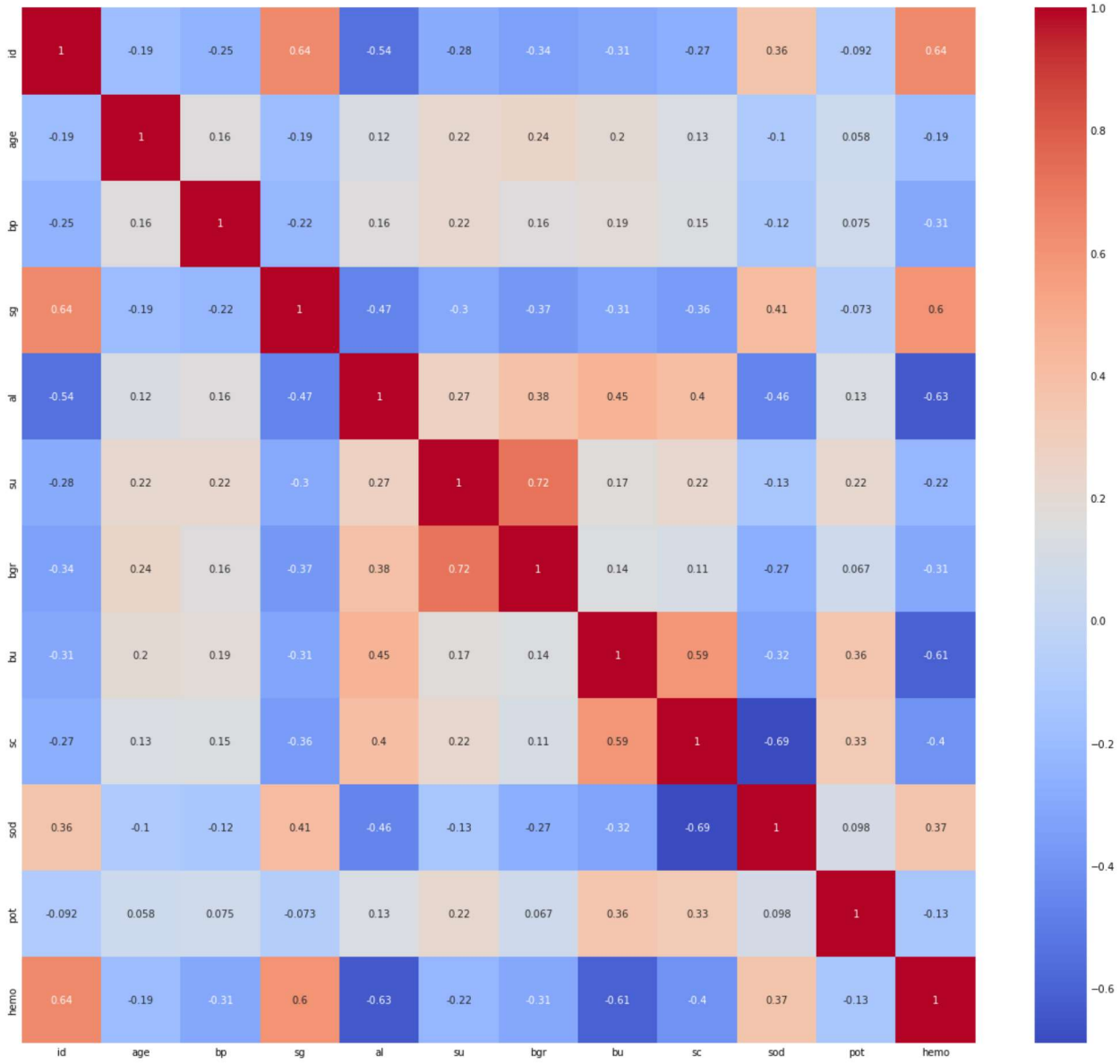
In [15]:
```
df.describe()
```

Out[15]:

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_urea	serum_c
391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.000000	38
51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.425722	
17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.503006	
2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.500000	
42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.000000	
55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.000000	
64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.000000	
90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.000000	7

In [123]:
```
#Finding the Correlation between the plots
plt.figure(figsize = (19,19))
sns.heatmap(data.corr(), annot = True, cmap = 'coolwarm') # looking for strong correlation
```

Out[123]: <AxesSubplot:>



In [4]:
```
# checking for null values

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

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

In [17]:
```
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 [18]:
```
# 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 [19]:
```
# 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 [21]:
```
df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},inpl

df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_replace = '\tno',

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

In [22]:
```
df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

In [23]:
```
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 [24]:
```
df.isna().sum().sort_values(ascending = False)
```

Out[24]:
```
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
```

In [25]:
```
df[num_cols].isnull().sum()
```

Out[25]:
```
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 [26]:
```
df[cat_cols].isnull().sum()
```

Out[26]:
```
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 [28]:
```
# 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 [29]:
```
# filling num_cols null values using random sampling method

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

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

Out[30]:
```
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 [31]:
```
# filling "red_blood_cells" and "pus_cell" using random sampling method and rest of cat_co

random_value_imputation('red_blood_cells')
random_value_imputation('pus_cell')

for col in cat_cols:
    impute_mode(col)
```

In [32]:
```
df[cat_cols].isnull().sum()
```

Out[32]:
```
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 [33]:
```
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 [34]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

In [35]:
```
df.head()
```

Out[35]:

acked_cell_volume	white_blood_cell_count	red_blood_cell_count	hypertension	diabetes_mellitus	coronary_artery_
44.0	7800.0	5.2	1	1	
38.0	6000.0	5.1	0	0	
31.0	7500.0	3.9	0	1	
32.0	6700.0	3.9	1	0	
35.0	7300.0	4.6	0	0	

In [ ]:

In [ ]:

In [ ]:

In [38]:
```
ind_col = [col for col in df.columns if col != 'class']
dep_col = 'class'
X = df[ind_col]
y = df[dep_col]
```

In [39]:
```
# 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 =
```

K-NEAREST NEIGHBOUR

In [40]:
```
#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 0.7964285714285714
Test Accuracy of KNN is 0.6666666666666666

Confusion Matrix :-
[[52 20]
 [20 28]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.72      0.72      0.72        72
           1       0.58      0.58      0.58        48

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

DECISION TREE CLASSIGIER

In [41]:
```
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.pred
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 :-
[[71  1]
 [ 6 42]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.92      0.99      0.95        72
           1       0.98      0.88      0.92        48

    accuracy                           0.94       120
   macro avg       0.95      0.93      0.94       120
weighted avg       0.94      0.94      0.94       120
```

RANDOM FOREST CLASSIFIER

In [42]:
```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features = 'aut
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.p
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 0.9666666666666667

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
```

In [ ]:

In [118]:

In [119]:
```
x = df[ind_col]
y = df[dep_col]
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29180/3116842793.py in <module>
----> 1 x = df[ind_col]
      2 y = df[dep_col]
      3 xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)

~\OneDrive\Documents\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3462             if is_iterator(key):
   3463                 key = list(key)
-> 3464             indexer = self.loc._get_listlike_indexer(key, axis=1)[1]
   3465
   3466         # take() does not accept boolean indexers

~\OneDrive\Documents\lib\site-packages\pandas\core\indexing.py in _get_listlike_indexer(s
elf, key, axis)
   1312             keyarr, indexer, new_indexer = ax._reindex_non_unique(keyarr)
   1313
-> 1314         self._validate_read_indexer(keyarr, indexer, axis)
   1315
   1316         if needs_i8_conversion(ax.dtype) or isinstance(

~\OneDrive\Documents\lib\site-packages\pandas\core\indexing.py in _validate_read_indexer
(self, key, indexer, axis)
   1375
   1376             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique
())
-> 1377             raise KeyError(f"{not_found} not in index")
   1378
   1379

KeyError: "['blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_blood_cells',
'pus_cell', 'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'serum_c
reatinine', 'sodium', 'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell
_count', 'red_blood_cell_count', 'hypertension', 'diabetes_mellitus', 'coronary_artery_di
sease', 'appetite', 'peda_edema', 'aanemia'] not in index"
```

In [85]:
```
svc = SVC()
print(svc)

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
SVC()
```

Out[85]:
```
SVC()
```

In [87]:
```
svc.fit(xtrain, ytrain)
score = svc.score(xtrain, ytrain)
print("Score: ", score)
```

```
Score:  0.6176470588235294
```

In [77]:
```
ypred = svc.predict(xtest)

cm = confusion_matrix(ytest, ypred)
print(cm)
```

```
[[40  0]
 [20  0]]
```

LINEAR REGRESSION

In [111]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
import pandas as pd

# Load data into a Pandas DataFrame
data = pd.read_csv('data.csv')

# Separate the input (X) and output (y) variables
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create a Linear Regression object and fit it to the training data
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict the output values for the testing set
y_pred = regressor.predict(X_test)

# Calculate the R-squared score
accuracy = r2_score(y_test, y_pred)

# Print the accuracy score
print("Accuracy (R-squared):", accuracy)
```

```
Accuracy (R-squared): 0.7045578655846056
```

from sklearn.linear_model import LinearRegression from sklearn.metrics import r2_score from sklearn.model_selection import train_test_split import pandas as pd data = pd.read_csv('kidney_disease.csv') X = df.iloc[:, :-1].values y = df.iloc[:, -1].values X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0) regressor = LinearRegression() regressor.fit(X_train, y_train) y_pred = regressor.predict(X_test) accuracy = r2_score(y_test, y_pred) print("Accuracy (R-squared):", accuracy)

POLYNOMIAL REGRESSION

In [113]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import numpy as np



# Transform the input data to include polynomial features up to degree 2
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)


# Create a Linear Regression object and fit it to the training data
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict the output values for the testing set
y_pred = regressor.pre

dict(X_test)

# Calculate the R-squared score
accuracy = r2_score(y_test, y_pred)

# Print the accuracy score
print("Accuracy (R-squared):", accuracy)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29180/3116478538.py in <module>
     16
     17 # Predict the output values for the testing set
---> 18 y_pred = regressor.pre
     19
     20 dict(X_test)

AttributeError: 'LinearRegression' object has no attribute 'pre'
```

LOGISTIC REGRESSION

In [112]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import pandas as pd

# Load data into a Pandas DataFrame
data = pd.read_csv('kidney_disease.csv')

# Separate the input (X) and output (y) variables
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create a Logistic Regression object and fit it to the training data
c
lassifier = LogisticRegression()
classifier.fit(X_train, y_train)

# Predict the output values for the testing set
y_pred = classifier.predict(X_test)

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy score
print("Accuracy:", accuracy)
```

```
Accuracy: 0.925
```

RANDOM FOREST

In [125]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

In [127]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [128]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=0)
```

In [129]:
```
# Instantiate and fit the RandomForestClassifier
forest = RandomForestClassifier()
forest.fit(x_train, y_train)
```

Out[129]:
```
RandomForestClassifier()
```

In [130]:
```
# Make predictions for the test set
y_pred_test = forest.predict(x_test)
```

In [132]:
```
# View accuracy score
accuracy_score(y_test, y_pred_test)
```

Out[132]:
```
0.9625
```

In [133]:
```
# View confusion matrix for test data and predictions
confusion_matrix(y_test, y_pred_test)
```

Out[133]:
```
array([[51,  0,  0],
       [ 1,  0,  0],
       [ 2,  0, 26]], dtype=int64)
```

In [135]:
```
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

         ckd       0.94      1.00      0.97        51
        ckd            0.00      0.00      0.00         1
      notckd       1.00      0.93      0.96        28

    accuracy                           0.96        80
   macro avg       0.65      0.64      0.64        80
weighted avg       0.95      0.96      0.96        80
```

DECISION TREE

In [150]:
```
import sys
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

import numpy as np
import pylab as pl
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

clf = DecisionTreeClassifier()
clf = clf.fit(X,y)
labels_test = clf.predict(X_test)

acc = accuracy_score(y_test, y_train)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29180/3347318169.py in <module>
      9
     10 clf = DecisionTreeClassifier()
---> 11 clf = clf.fit(X,y)
     12 labels_test = clf.predict(X_test)
     13

~\OneDrive\Documents\lib\site-packages\sklearn\tree\_classes.py in fit(self, X, y, sample
_weight, check_input, X_idx_sorted)
    901         """
    902
--> 903         super().fit(
    904             X, y,
    905             sample_weight=sample_weight,

~\OneDrive\Documents\lib\site-packages\sklearn\tree\_classes.py in fit(self, X, y, sample
_weight, check_input, X_idx_sorted)
    155             check_X_params = dict(dtype=DTYPE, accept_sparse="csc")
    156             check_y_params = dict(ensure_2d=False, dtype=None)
--> 157             X, y = self._validate_data(X, y,
    158                                        validate_separately=(check_X_params,
    159                                                             check_y_params))

~\OneDrive\Documents\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, rese
t, validate_separately, **check_params)
    428                 # :(
    429                 check_X_params, check_y_params = validate_separately
--> 430                 X = check_array(X, **check_X_params)
    431                 y = check_array(y, **check_y_params)
    432             else:

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kw
args)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in check_array(array,
accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allo
w_nd, ensure_min_samples, ensure_min_features, estimator)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~\OneDrive\Documents\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order,
like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104

ValueError: could not convert string to float: 'normal'
```

POLYNOMIAL REGRESSION

In [154]:
```
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_state=0)
# Transform the input data to include polynomial features up to degree 2
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29180/62937563.py in <module>
      6 # Transform the input data to include polynomial features up to degree 2
      7 poly = PolynomialFeatures(degree=2)
----> 8 X_poly = poly.fit_transform(X)
      9 lin_reg_2 = LinearRegression()
     10 lin_reg_2.fit(X_poly, y)

~\OneDrive\Documents\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit
_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transformation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transformation)

~\OneDrive\Documents\lib\site-packages\sklearn\preprocessing\_data.py in fit(self, X, y)
   1702             Fitted transformer.
   1703         """
-> 1704         n_samples, n_features = self._validate_data(
   1705             X, accept_sparse=True).shape
   1706         combinations = self._combinations(n_features, self.degree,

~\OneDrive\Documents\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, rese
t, validate_separately, **check_params)
    419             out = X
    420         elif isinstance(y, str) and y == 'no_validation':
--> 421             X = check_array(X, **check_params)
    422             out = X
    423         else:

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kw
args)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in check_array(array,
accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allo
w_nd, ensure_min_samples, ensure_min_features, estimator)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~\OneDrive\Documents\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order,
like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104

ValueError: could not convert string to float: 'normal'
```

In [155]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split

# Load the Boston Housing dataset
df=pd.read_csv('kidney_disease.csv')

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=42)

# Define a polynomial regression model
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Use the trained model to make predictions on the testing data
y_pred = model.predict(X_test_poly)

# Calculate the mean squared error and R-squared as accuracy metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print('Mean Squared Error:', mse)
print('R-squared:', r2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_29180/517723795.py in <module>
     13 # Define a polynomial regression model
     14 poly_features = PolynomialFeatures(degree=2)
---> 15 X_train_poly = poly_features.fit_transform(X_train)
     16 X_test_poly = poly_features.transform(X_test)
     17 model = LinearRegression()

~\OneDrive\Documents\lib\site-packages\sklearn\base.py in fit_transform(self, X, y, **fit
_params)
    697         if y is None:
    698             # fit method of arity 1 (unsupervised transformation)
--> 699             return self.fit(X, **fit_params).transform(X)
    700         else:
    701             # fit method of arity 2 (supervised transformation)

~\OneDrive\Documents\lib\site-packages\sklearn\preprocessing\_data.py in fit(self, X, y)
   1702             Fitted transformer.
   1703         """
-> 1704         n_samples, n_features = self._validate_data(
   1705             X, accept_sparse=True).shape
   1706         combinations = self._combinations(n_features, self.degree,

~\OneDrive\Documents\lib\site-packages\sklearn\base.py in _validate_data(self, X, y, rese
t, validate_separately, **check_params)
    419             out = X
    420         elif isinstance(y, str) and y == 'no_validation':
--> 421             X = check_array(X, **check_params)
    422             out = X
    423         else:

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in inner_f(*args, **kw
args)
     61             extra_args = len(args) - len(all_args)
     62             if extra_args <= 0:
---> 63                 return f(*args, **kwargs)
     64
     65             # extra_args > 0

~\OneDrive\Documents\lib\site-packages\sklearn\utils\validation.py in check_array(array,
accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allo
w_nd, ensure_min_samples, ensure_min_features, estimator)
    671                     array = array.astype(dtype, casting="unsafe", copy=False)
    672                 else:
--> 673                     array = np.asarray(array, order=order, dtype=dtype)
    674             except ComplexWarning as complex_warning:
    675                 raise ValueError("Complex data not supported\n"

~\OneDrive\Documents\lib\site-packages\numpy\core\_asarray.py in asarray(a, dtype, order,
like)
    100         return _asarray_with_like(a, dtype=dtype, order=order, like=like)
    101
--> 102     return array(a, dtype, copy=False, order=order)
    103
    104

ValueError: could not convert string to float: 'normal'
```

In [ ]:

In [ ]: