In [308]:
```
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 [309]:
```
df = pd.read_csv('kidney_disease.csv')
df.head()
```

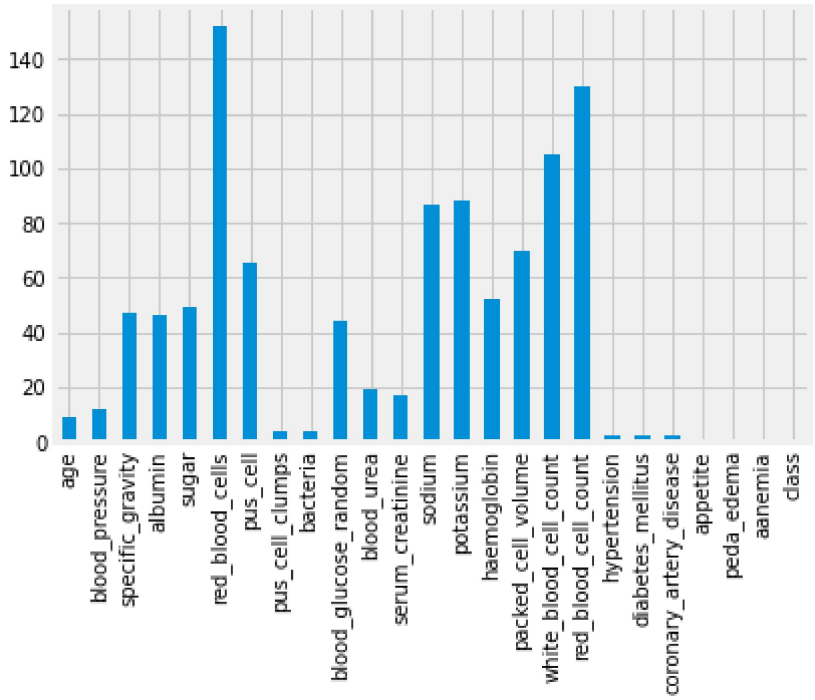
Out[309]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	soc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	117.0	56.0	3.8	111.(
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN

In [310]:
```
# dropping id column
df.drop('id', axis = 1, inplace = True)
```

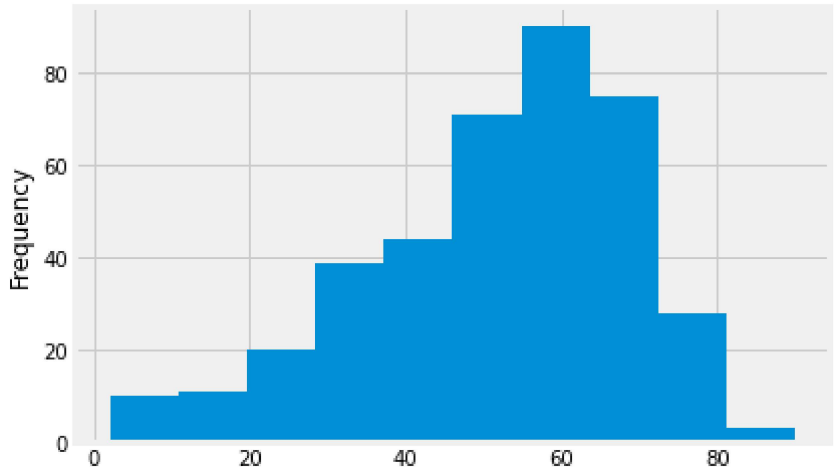
In [311]:
```
#rename
df.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar',
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_ur
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_c
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease',
              'aanemia', 'class']
```

In [312]:
```
# Here we are plotting the graph to see the null values in the dataset.
p = df.isnull().sum().plot.bar()
```



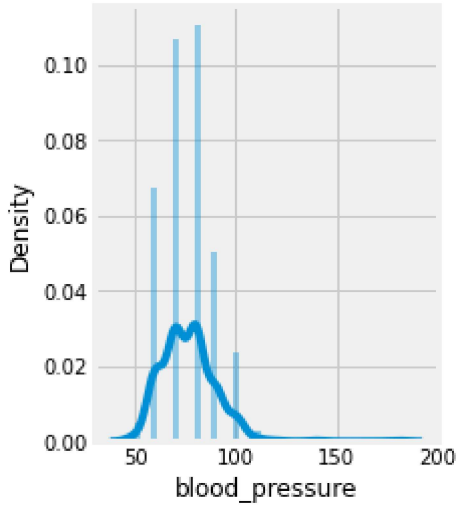
In [313]:
```
#checking the distribution of the age column
df['age'].plot(kind='hist')
```

Out[313]: <AxesSubplot:ylabel='Frequency'>



In [314]:
```
# distribution of bp column
plt.subplot(121), sns.distplot(df['blood_pressure'])
plt.show
```

Out[314]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [315]:
```
df.head()
```

Out[315]:

	age	blood_pressure	specific_gravity	albumin	sugar	red_blood_cells	pus_cell	pus_cell_cl
0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notp
1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notp
2	62.0	80.0	1.010	2.0	3.0	normal	normal	notp
3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	p
4	51.0	80.0	1.010	2.0	0.0	normal	normal	notp

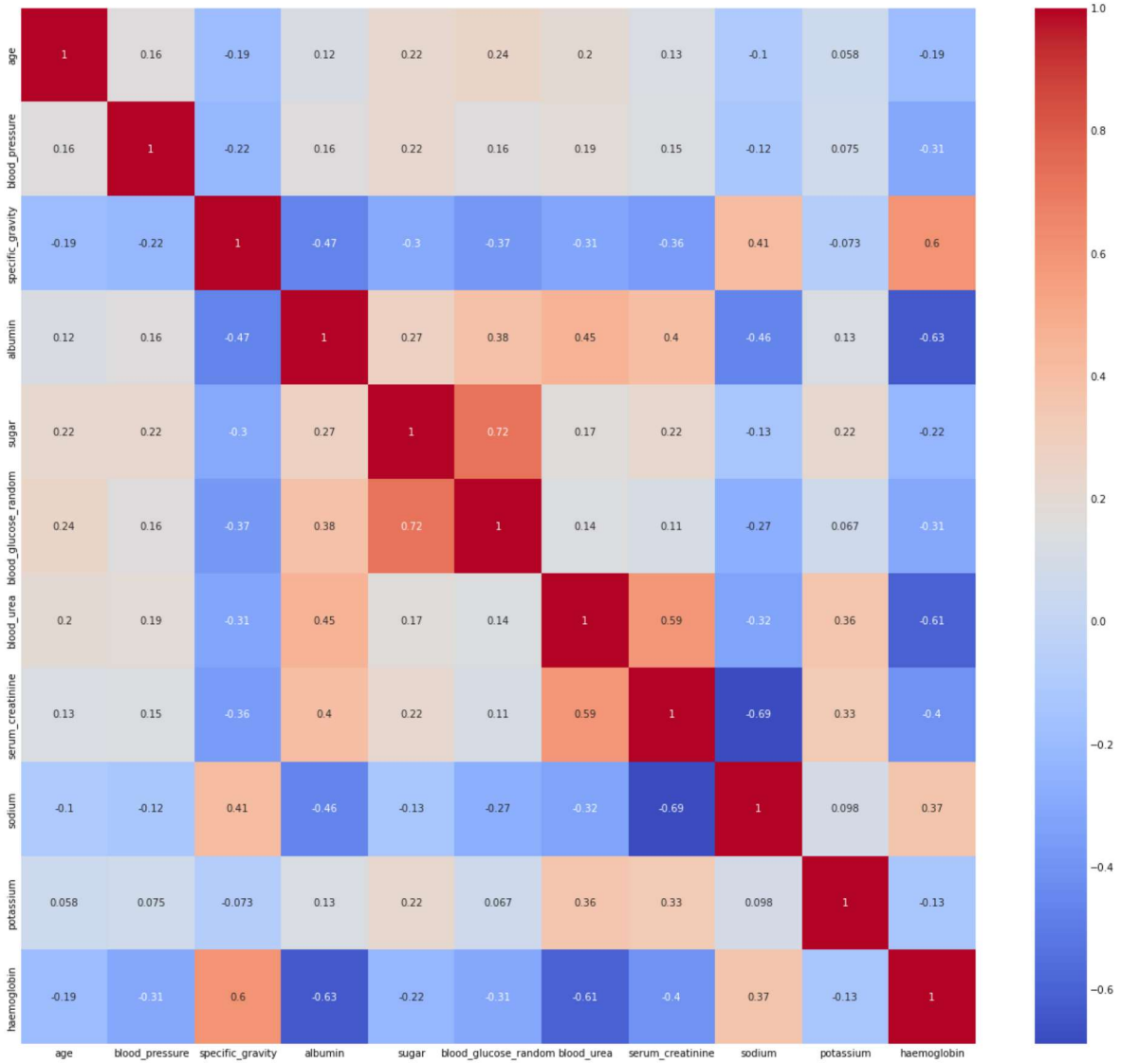
In [316]:
```
df.describe()
```

Out[316]:

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_rand
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000

In [317]:
```
#Finding the Correlation between the plots
plt.figure(figsize = (19,19))
sns.heatmap(df.corr(), annot = True, cmap = 'coolwarm') # looking for strong c
```

Out[317]: <AxesSubplot:>



In [318]:
```
df['packed_cell_volume'] = pd.to_numeric(df['packed_cell_volume'], errors='coe
df['white_blood_cell_count'] = pd.to_numeric(df['white_blood_cell_count'], err
df['red_blood_cell_count'] = pd.to_numeric(df['red_blood_cell_count'], errors=
```

In [319]:
```
# 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 [320]:
```
# 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 [321]:
```
df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes'

df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_repla

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

In [322]:
```
df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

In [323]:
```
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 [324]:
```
# checking for null values
df.isna().sum().sort_values(ascending = False)
```

Out[324]:
```
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 [325]:
```
df[num_cols].isnull().sum()
```

Out[325]:
```
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 [326]:
```
df[cat_cols].isnull().sum()
```

Out[326]:
```
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 [327]:
```
# filling null values, we will use two methods, random sampling for higher nul
# 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 [328]:
```
# filling num_cols null values using random sampling method

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

In [329]:
```
df[num_cols].isnull().sum()
```

Out[329]:
```
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 [330]:
```
# filling "red_blood_cells" and "pus_cell" using random sampling method and re

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

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

In [331]:
```
df[cat_cols].isnull().sum()
```

Out[331]:
```
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 [332]:
```
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 [333]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

In [334]:
```
df.head()
```

Out[334]:

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

In [335]:
```
ind_col = [col for col in df.columns if col != 'class']
dep_col = 'class'
X = df[ind_col]
y = df[dep_col]
```

In [336]:
```
# 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, ra
```

Logistic Regression

In [337]:
```
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, randc
# Create a Logistic Regression object and fit it to the training data
Classifier = 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
```

K-NEAREST NEIGHBOUR

In [338]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r

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_tra
print(f"Test Accuracy of KNN is {knn_acc} \n")

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

```
Training Accuracy of KNN is 0.8
Test Accuracy of KNN is 0.75

Confusion Matrix :-
[[38 14]
 [ 6 22]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.86      0.73      0.79        52
           1       0.61      0.79      0.69        28

    accuracy                           0.75        80
   macro avg       0.74      0.76      0.74        80
weighted avg       0.78      0.75      0.76        80
```

In [ ]:
```

```

DECISION TREE CLASSIFIER

In [339]:
```
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_tra
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))}\
print(f"Classification Report :- \n {classification_report(y_test, dtc.predict
```

```
Training Accuracy of Decision Tree Classifier is 1.0
Test Accuracy of Decision Tree Classifier is 0.9625

Confusion Matrix :-
[[52  0]
 [ 3 25]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        52
           1       1.00      0.89      0.94        28

    accuracy                           0.96        80
   macro avg       0.97      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

RANDOM FOREST CLASSIFIER

In [340]:
```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_fea
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_tra
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)
print(f"Classification Report :- \n {classification_report(y_test, rd_clf.pred
```

```
Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 0.9625

Confusion Matrix :-
[[52  0]
 [ 3 25]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.95      1.00      0.97        52
           1       1.00      0.89      0.94        28

    accuracy                           0.96        80
   macro avg       0.97      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

SVM

In [341]:
```
from sklearn.svm import SVC
classifier =SVC(random_state = 0)
classifier.fit(X_train, y_train)
print(accuracy)
```

```
0.925
```

LINEAR REGRESSION

In [342]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
import pandas as pd


# 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, randc

# 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): 0.6679328207916022
```

RANDOM FOREST REGRESSION

In [343]:
```
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_r
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
#Instantiate and fit the RandomForestClassifier
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
# Make predictions for the test set
y_pred_test = forest.predict(X_test)
# View accuracy score
accuracy_score(y_test, y_pred_test)
# View confusion matrix for test data and predictions
confusion_matrix(y_test, y_pred_test)
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        52
           1       1.00      0.89      0.94        28

    accuracy                           0.96        80
   macro avg       0.97      0.95      0.96        80
weighted avg       0.96      0.96      0.96        80
```

POLYNOMIAL REGRESSION

In [344]:
```
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
# 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)
```

Out[344]:
```
LinearRegression()
```

In [345]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.6679328207916022
```

Naive bayes

In [346]:
```
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,randor
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)

# comparing actual response values (y_test) with predicted response values (y_
from sklearn import metrics
nav_acc=metrics.accuracy_score(y_test, y_pred)*100
print("Gaussian Naive Bayes model accuracy(in %):", nav_acc)
```

```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

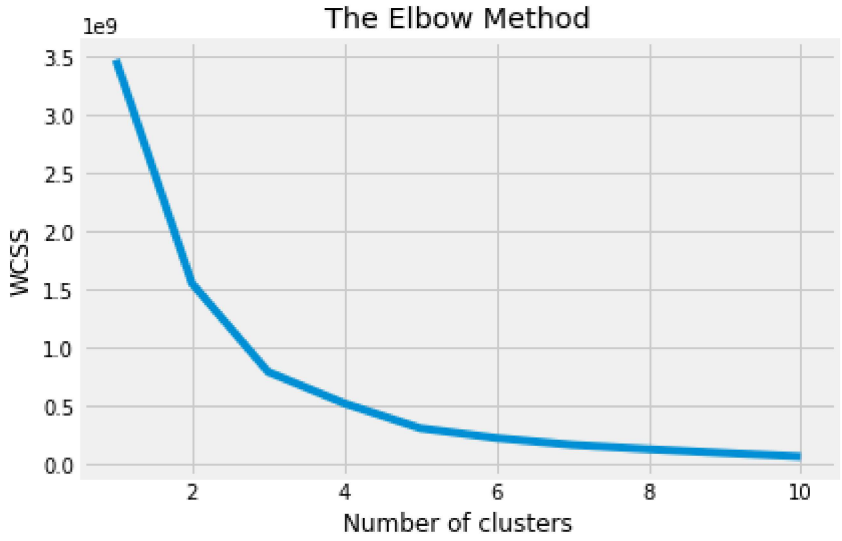
kernal svm

In [347]:
```
# Import the Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

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,rando
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print("KERNEL SVM WITH GAUSSIAN KERNEL")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
from sklearn.svm import SVC
svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix
print("KERNEL SVM WITH SIGMOID KERNEL")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

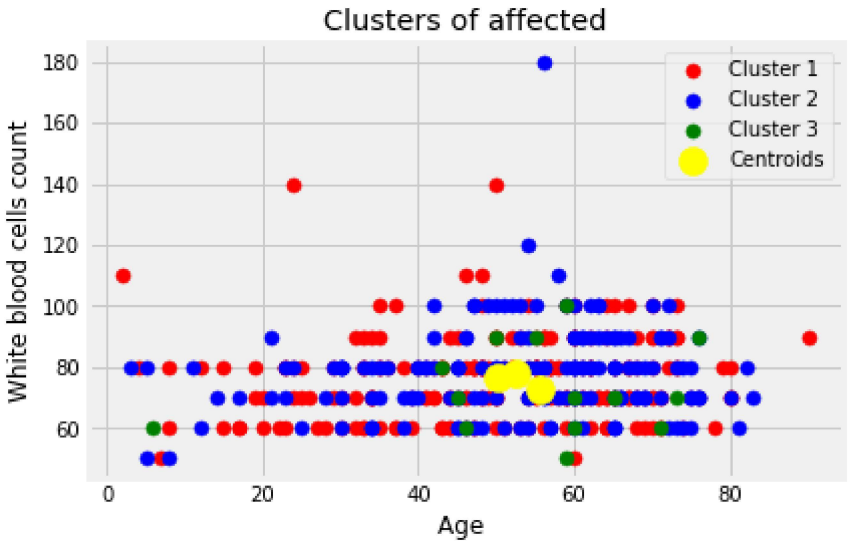
```
KERNEL SVM WITH GAUSSIAN KERNEL
[[52  0]
 [28  0]]
              precision    recall  f1-score   support

           0       0.65      1.00      0.79        52
           1       0.00      0.00      0.00        28

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

KERNEL SVM WITH SIGMOID KERNEL
[[31 21]
 [25  3]]
              precision    recall  f1-score   support

           0       0.55      0.60      0.57        52
           1       0.12      0.11      0.12        28

    accuracy                           0.42        80
   macro avg       0.34      0.35      0.34        80
weighted avg       0.40      0.42      0.41        80
```

K mean clustering

In [348]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
#dataset
df = pd.read_csv('kidney_disease.csv')
x = df.iloc[:,[1,17]].values
#importing k means
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



In [349]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 50, c = 'red', label
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 50, c = 'blue', label
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 50, c = 'green', lab
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
plt.title('Clusters of affected')
plt.xlabel('Age ')
plt.ylabel('White blood cells count')
plt.legend()
plt.show()
```



HIERARCHICAL CLUSTERING

In [ ]:
```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('hemoglobin')
plt.ylabel('Euclidean distance')
plt.show()
```

In [ ]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 3, affinity = 'euclidean', linkage =
y_hc = hc.fit_predict(X)
```

In [ ]:
```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 50, c = 'green', label = 'C
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 50, c = 'magenta', label =
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 50, c = 'orange', label = '

plt.title('Clusters of affected')
plt.xlabel('white blood cells')
plt.ylabel('Age')
plt.legend()
plt.show()
```

Decision tree Regression

In [ ]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

In [262]:
```
from math import sqrt
```

In [263]:
```
mse=mean_squared_error(y_test,y_pred)
```

In [264]:
```
print(mse)
```

```
0.6125
```

In [265]:
```
rmse=sqrt(mean_squared_error(y_test,y_pred))
```

In [266]:
```
print(rmse)
```

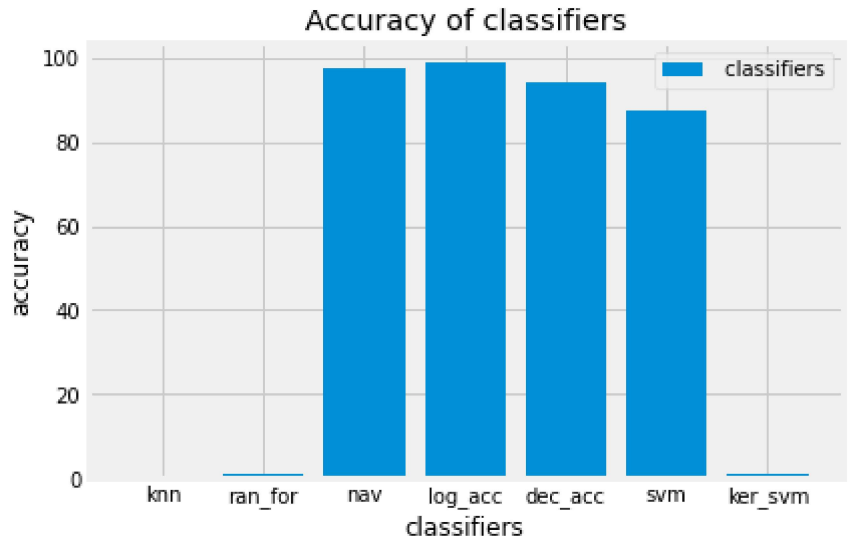
```
0.7826237921249264
```

SVR

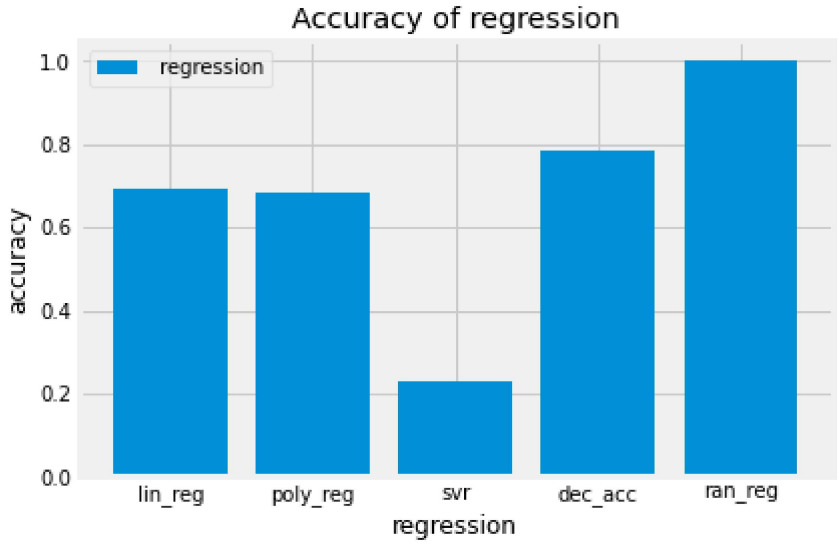
In [267]:
```
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Split the data into training and test sets
X_train, X_test, y_train, y_test =train_test_split(X,y,test_size = 0.2,random_

# Fit the SVR model to the training data
clf = SVR(kernel='rbf', C=1e3, gamma=0.1)
clf.fit(X_train, y_train)

# Predict the target values for the test data using the trained model
y_pred = clf.predict(X_test)

# Evaluate the performance of the model on the test data using mean squared er
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 0.230524999998807502
```

In [268]:
```
x=['knn','ran_for','nav','log_acc','dec_acc','svm','ker_svm']
y=knn_acc,rd_clf_acc,nav_acc,98.7,94.1,87.5,1

X_axis = np.arange(len(x))
plt.bar(X_axis , y, label = ' classifiers')
plt.xticks(X_axis, x)
plt.xlabel("classifiers")
plt.ylabel("accuracy")
plt.title("Accuracy of classifiers")
plt.legend()
plt.show()
```



In [269]:
```
x=['lin_reg','poly_reg','svr','dec_acc','ran_reg']
y=accuracy,0.682,mse,rmse,1

X_axis = np.arange(len(x))
plt.bar(X_axis , y, label = ' regression')
plt.xticks(X_axis, x)
plt.xlabel("regression")
plt.ylabel("accuracy")
plt.title("Accuracy of regression")
plt.legend()
plt.show()
```



In [ ]:
```
```