In [149]:
```
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 [150]:
```
df = pd.read_csv('kidney_disease.csv')
df.head()
```

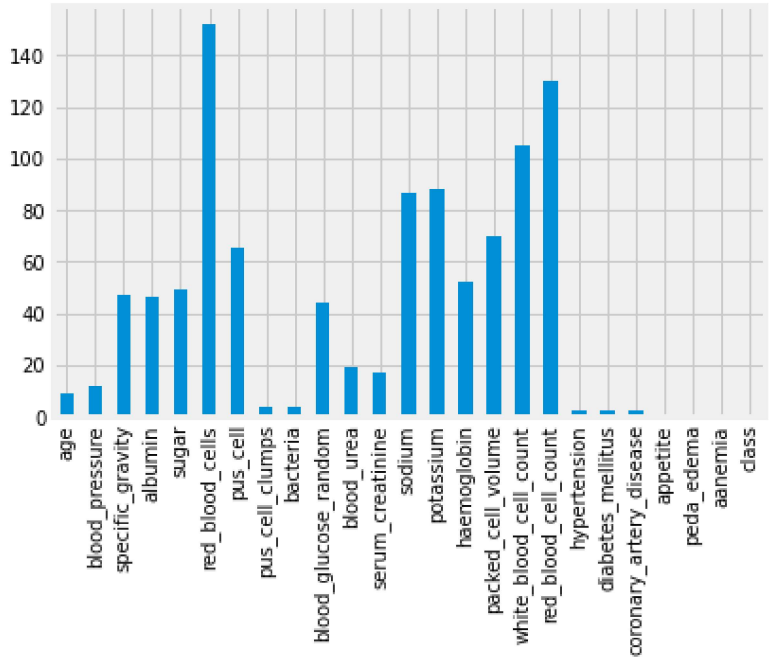
Out[150]:

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	bu	sc	sod	pot	hei
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	121.0	36.0	1.2	NaN	NaN	1
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	NaN	18.0	0.8	NaN	NaN	1
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	423.0	53.0	1.8	NaN	NaN	
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	1
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	106.0	26.0	1.4	NaN	NaN	1

In [151]:
```
# dropping id column
df.drop('id', axis = 1, inplace = True)
```

In [152]:
```
#rename
df.columns = ['age', 'blood_pressure', 'specific_gravity', 'albumin', 'sugar', 'red_bl
              'pus_cell_clumps', 'bacteria', 'blood_glucose_random', 'blood_urea', 'se
              'potassium', 'haemoglobin', 'packed_cell_volume', 'white_blood_cell_coun
              'hypertension', 'diabetes_mellitus', 'coronary_artery_disease', 'appetit
              'aanemia', 'class']
```

In [153]:
```
# Here we are plotting the graph to see the null values in the dataset.
p = df.isnull().sum().plot.bar()
```



In [154]:
```
df.head()
```

Out[154]:

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

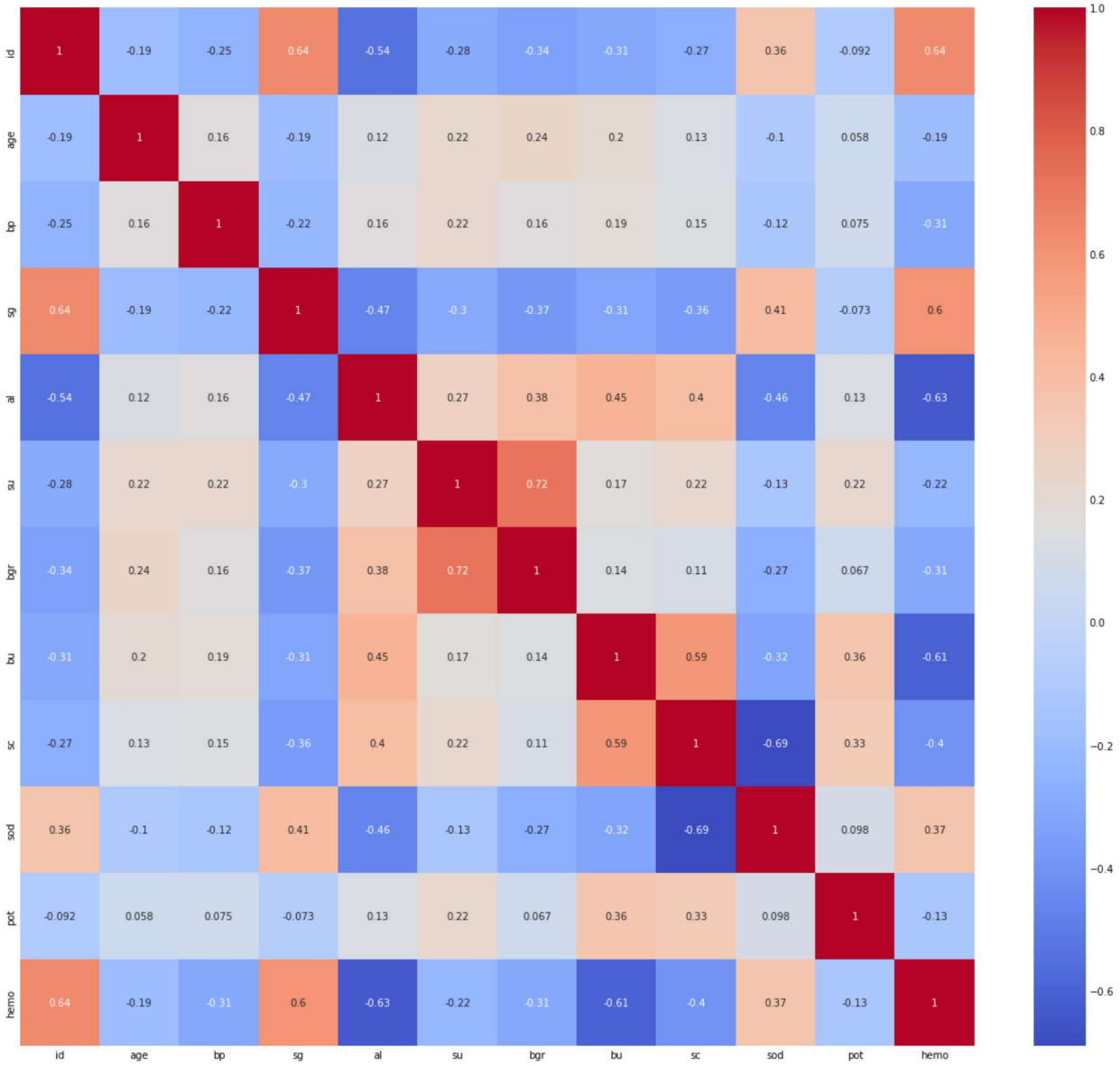
In [155]:
```
df.describe()
```

Out[155]:

	age	blood_pressure	specific_gravity	albumin	sugar	blood_glucose_random	blood_
count	391.000000	388.000000	353.000000	354.000000	351.000000	356.000000	381.0(
mean	51.483376	76.469072	1.017408	1.016949	0.450142	148.036517	57.42
std	17.169714	13.683637	0.005717	1.352679	1.099191	79.281714	50.5(
min	2.000000	50.000000	1.005000	0.000000	0.000000	22.000000	1.5(
25%	42.000000	70.000000	1.010000	0.000000	0.000000	99.000000	27.0(
50%	55.000000	80.000000	1.020000	0.000000	0.000000	121.000000	42.0(
75%	64.500000	80.000000	1.020000	2.000000	0.000000	163.000000	66.0(
max	90.000000	180.000000	1.025000	5.000000	5.000000	490.000000	391.0(

In [156]:
```
#Finding the Correlation between the plots
plt.figure(figsize = (19,19))
sns.heatmap(data.corr(), annot = True, cmap = 'coolwarm') # looking for strong correla
```

Out[156]: <AxesSubplot:>



In [157]:
```
# checking for null values

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

Out[157]:
```
red_blood_cells            152
red_blood_cell_count       130
white_blood_cell_count     105
potassium                   88
sodium                      87
packed_cell_volume          70
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 [158]:
```
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='coe
df['red_blood_cell_count'] = pd.to_numeric(df['red_blood_cell_count'], errors='coerce'
```

In [159]:
```
# 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 [160]:
```
# 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 [161]:
```
df['diabetes_mellitus'].replace(to_replace = {'\tno':'no','\tyes':'yes',' yes':'yes'},

df['coronary_artery_disease'] = df['coronary_artery_disease'].replace(to_replace = '\t

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

In [162]:
```
df['class'] = df['class'].map({'ckd': 0, 'not ckd': 1})
df['class'] = pd.to_numeric(df['class'], errors='coerce')
```

In [163]:
```
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 [164]:
```
df.isna().sum().sort_values(ascending = False)
```

Out[164]:
```
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 [165]:
```
df[num_cols].isnull().sum()
```

Out[165]:
```
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 [166]:
```
df[cat_cols].isnull().sum()
```

Out[166]:
```
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 [167]:
```
# filling null values, we will use two methods, random sampling for higher null values
# 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 [168]:
```
# filling num_cols null values using random sampling method

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

In [169]:
```
df[num_cols].isnull().sum()
```

Out[169]:
```
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 [170]:
```
# filling "red_blood_cells" and "pus_cell" using random sampling method and rest of ca

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

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

In [171]:
```
df[cat_cols].isnull().sum()
```

Out[171]:
```
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 [172]:
```
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 [173]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

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

In [174]:
```
df.head()
```

Out[174]:

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

In [175]:
```
ind_col = [col for col in df.columns if col != 'class']
dep_col = 'class'
X = df[ind_col]
y = df[dep_col]
```

In [176]:
```
# 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_sta
```

K-NEAREST NEIGHBOUR

In [177]:
```
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 0.7928571428571428
Test Accuracy of KNN is 0.6

Confusion Matrix :-
[[44 28]
 [20 28]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.69      0.61      0.65        72
           1       0.50      0.58      0.54        48

    accuracy                           0.60       120
   macro avg       0.59      0.60      0.59       120
weighted avg       0.61      0.60      0.60       120
```

DECISION TREE CLASSIGIER

In [178]:
```
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.
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 0.95

Confusion Matrix :-
[[71  1]
 [ 5 43]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.93      0.99      0.96        72
           1       0.98      0.90      0.93        48

    accuracy                           0.95       120
   macro avg       0.96      0.94      0.95       120
weighted avg       0.95      0.95      0.95       120
```

RANDOM FOREST CLASSIFIER

In [179]:
```
from sklearn.ensemble import RandomForestClassifier

rd_clf = RandomForestClassifier(criterion = 'entropy', max_depth = 11, max_features =
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_c
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_te
```

```
Training Accuracy of Random Forest Classifier is 1.0
Test Accuracy of Random Forest Classifier is 0.9583333333333334

Confusion Matrix :-
[[72  0]
 [ 5 43]]

Classification Report :-
               precision    recall  f1-score   support

           0       0.94      1.00      0.97        72
           1       1.00      0.90      0.95        48

    accuracy                           0.96       120
   macro avg       0.97      0.95      0.96       120
weighted avg       0.96      0.96      0.96       120
```

SUPPORT VECTOR REGFRESSION

In [180]:
```
from sklearn.svm import SVC
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [181]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15)
```

In [183]:
```
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)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_58256/2678115755.py in <module>
----> 1 SVC = SVC()
      2 print(SVC)
      3
      4 SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
      5     decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',

TypeError: 'SVC' object is not callable
```

In [184]:
```
SVC.fit(X_train, y_train)
score = SVR.score(X_train, y_train)
print("Score: ", score)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_58256/220490471.py in <module>
      1 SVC.fit(X_train, y_train)
----> 2 score = SVR.score(X_train, y_train)
      3 print("Score: ", score)

NameError: name 'SVR' is not defined
```

In [185]:
```


y_pred = SVC.predict(X_test)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[46  0]
 [14  0]]
```

LINEAR REGRESSION

In [186]:
```
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, random_state=

# 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): 0.7048748428489737
```

LOGISTIC REGRESSION

In [187]:
```
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=

# 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: 0.875
```

RANDOM FOREST

In [188]:
```
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 [189]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [190]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_state=0
```

In [191]:
```
# Instantiate and fit the RandomForestClassifier
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

Out[191]:
```
RandomForestClassifier()
```

In [192]:
```
# Make predictions for the test set
y_pred_test = forest.predict(X_test)
```

In [193]:
```
# View accuracy score
accuracy_score(y_test, y_pred_test)
```

Out[193]:
```
0.95
```

In [194]:
```
# View confusion matrix for test data and predictions
confusion_matrix(y_test, y_pred_test)
```

Out[194]:
```
array([[52,  0],
       [ 4, 24]], dtype=int64)
```

In [195]:
```
# View the classification report for test data and predictions
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.93      1.00      0.96        52
           1       1.00      0.86      0.92        28

    accuracy                           0.95        80
   macro avg       0.96      0.93      0.94        80
weighted avg       0.95      0.95      0.95        80
```

POLYNOMIAL REGRESSION

In [196]:
```
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)
```

Out[196]:
```
LinearRegression()
```

In [197]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print(r2_score(y_test, y_pred))
```

```
0.7048748428489737
```

In [198]:
```
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 [199]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

In [200]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_state=0
```

In [201]:
```
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

Out[201]:
```
RandomForestClassifier()
```

In [202]:
```
y_pred_test = forest.predict(X_test)
```

In [203]:
```
accuracy_score(y_test, y_pred_test)
```

Out[203]:
```
0.9625
```

In [204]:
```
confusion_matrix(y_test, y_pred_test)
```

Out[204]:
```
array([[52,  0],
       [ 3, 25]], dtype=int64)
```

In [205]:
```
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
```

Naive bayes

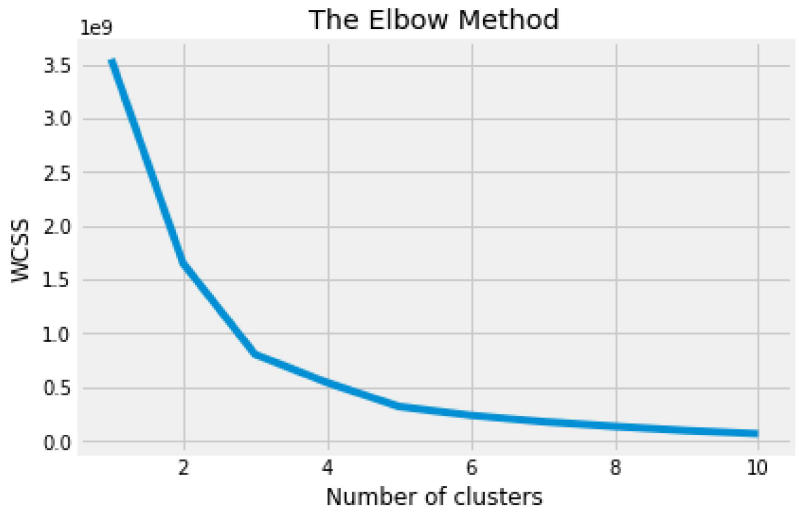
In [206]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,random_state=0
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

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

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

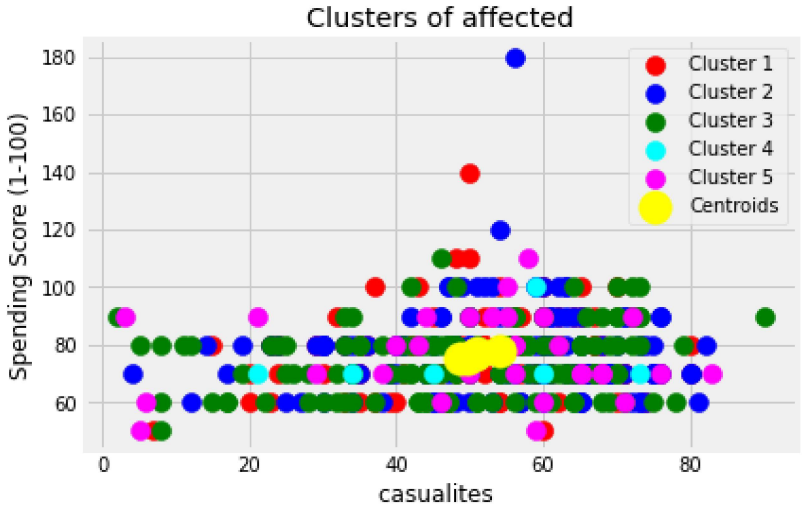
```
Gaussian Naive Bayes model accuracy(in %): 97.5
```

K mean clustering

In [83]:

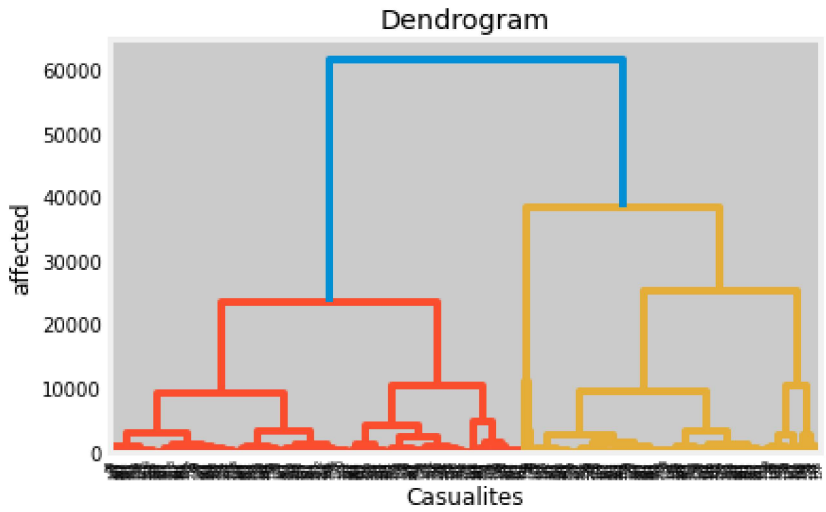
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, 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 = 100, c = 'red', label = 'Clu
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cl
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'C
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'cyan', label = 'Cl
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'magenta', label =
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c =
plt.title('Clusters of affected')
plt.xlabel('casualites ')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```





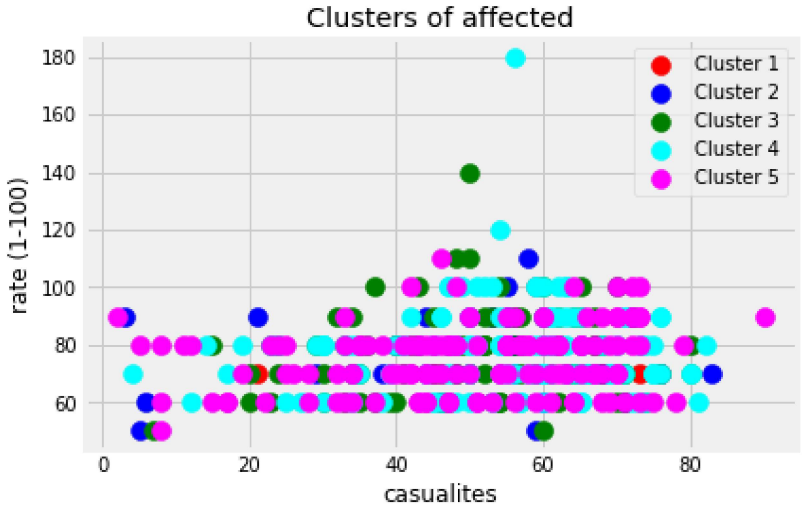
HIERARCHICAL CLUSTERING

In [84]:
```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Casualites')
plt.ylabel('affected')
plt.show()
```



In [85]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

In [86]:
```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2'
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4'
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster
plt.title('Clusters of affected')
plt.xlabel('casualites')
plt.ylabel('rate (1-100)')
plt.legend()
plt.show()
```



#REINFORCEMENT LEARNING

kernal svm

In [101]:

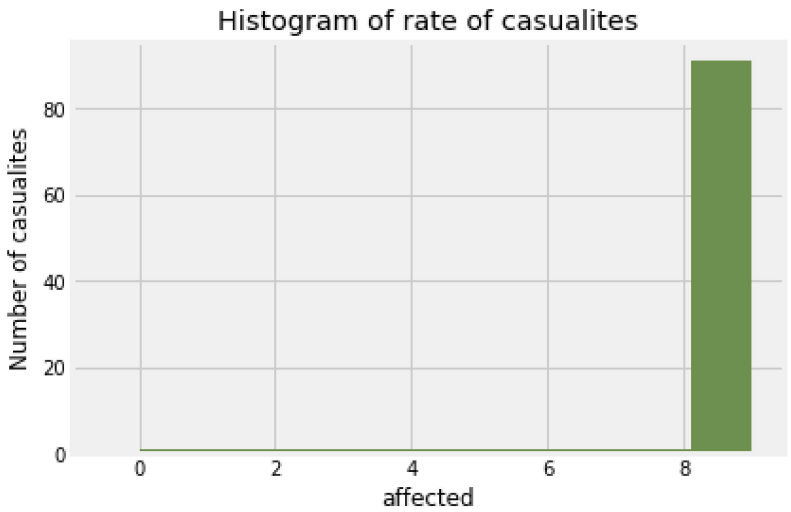
```
# Import the Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

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,random_state=0
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
[[28 24]
 [25  3]]
              precision    recall  f1-score   support

           0       0.53      0.54      0.53        52
           1       0.11      0.11      0.11        28

    accuracy                           0.39        80
   macro avg       0.32      0.32      0.32        80
weighted avg       0.38      0.39      0.38        80
```

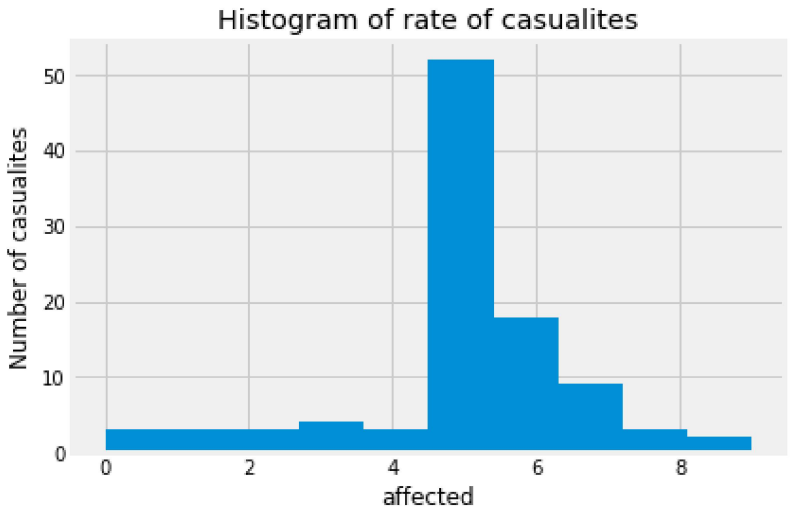
Upper confidence bound

In [207]:

```
# Implementing UCB
import math
N = 100
d = 10
ads_selected = []
numbers_of_selections = [0] * d
sums_of_rewards = [0] * d
total_reward = 0
for n in range(0, N):
    ad = 0
    max_upper_bound = 0
    for i in range(0, d):
        if (numbers_of_selections[i] > 0):
            average_reward = sums_of_rewards[i] / numbers_of_selections[i]
            delta_i = math.sqrt(3/2 * math.log(n + 1) / numbers_of_selections[i])
            upper_bound = average_reward + delta_i
        else:
            upper_bound = 1e200
        if upper_bound > max_upper_bound:
            max_upper_bound = upper_bound
            ad = i
    ads_selected.append(ad)
    numbers_of_selections[ad] = numbers_of_selections[ad] + 1
    reward = df.values[n, ad]
    sums_of_rewards[ad] = sums_of_rewards[ad] + reward
    total_reward = total_reward + reward
    plt.hist(ads_selected)
plt.title('Histogram of rate of casualites')
plt.xlabel('affected')
plt.ylabel('Number of casualites')
plt.show()
```



In [129]:
```
#thomson bound
import random
N = 100
d = 10
ads_selected = []
numbers_of_rewards_1 = [0] * d
numbers_of_rewards_0 = [0] * d
total_reward = 0
for n in range(0, N):
    ad = 0
    max_random = 0
    for i in range(0, d):
        random_beta = random.betavariate(numbers_of_rewards_1[i] + 1, numbers_of_rewar
        if random_beta > max_random:
            max_random = random_beta
            ad = i
    ads_selected.append(ad)
    reward = df.values[n, ad]
    if reward == 1:
        numbers_of_rewards_1[ad] = numbers_of_rewards_1[ad] + 1
    else:
        numbers_of_rewards_0[ad] = numbers_of_rewards_0[ad] + 1
    total_reward = total_reward + reward

# Visualising the results - Histogram
plt.hist(ads_selected)
plt.title('Histogram of rate of casualites')
plt.xlabel('affected')
plt.ylabel('Number of casualites')
plt.show()
```



In [ ]: