In [1]:
```python
import numpy as np
from sklearn import preprocessing
input_data = np.array([[5.1, -2.9, 3.3],
[-1.2, 7.8, -6.1],
[3.9, 0.4, 2.1],
[7.3, -9.9, -4.5]])
# Binarize data
data_binarized = preprocessing.Binarizer(threshold=2.1).transform(input_data)
print("\nBinarized data:\n", data_binarized)
# Print mean and standard deviation
print("\nBEFORE:")
print("Mean =", input_data.mean(axis=0))
print("Std deviation =", input_data.std(axis=0))
# Remove mean
data_scaled = preprocessing.scale(input_data)
print("\nAFTER:")
print("Mean =", data_scaled.mean(axis=0))
print("Std deviation =", data_scaled.std(axis=0))
# Min max scaling
data_scaler_minmax = preprocessing.MinMaxScaler(feature_range=(0, 1))
data_scaled_minmax = data_scaler_minmax.fit_transform(input_data)
print("\nMin max scaled data:\n", data_scaled_minmax)
# Normalize data
data_normalized_l1 = preprocessing.normalize(input_data, norm='l1')
data_normalized_l2 = preprocessing.normalize(input_data, norm='l2')
print("\nL1 normalized data:\n", data_normalized_l1)
print("\nL2 normalized data:\n", data_normalized_l2)
```

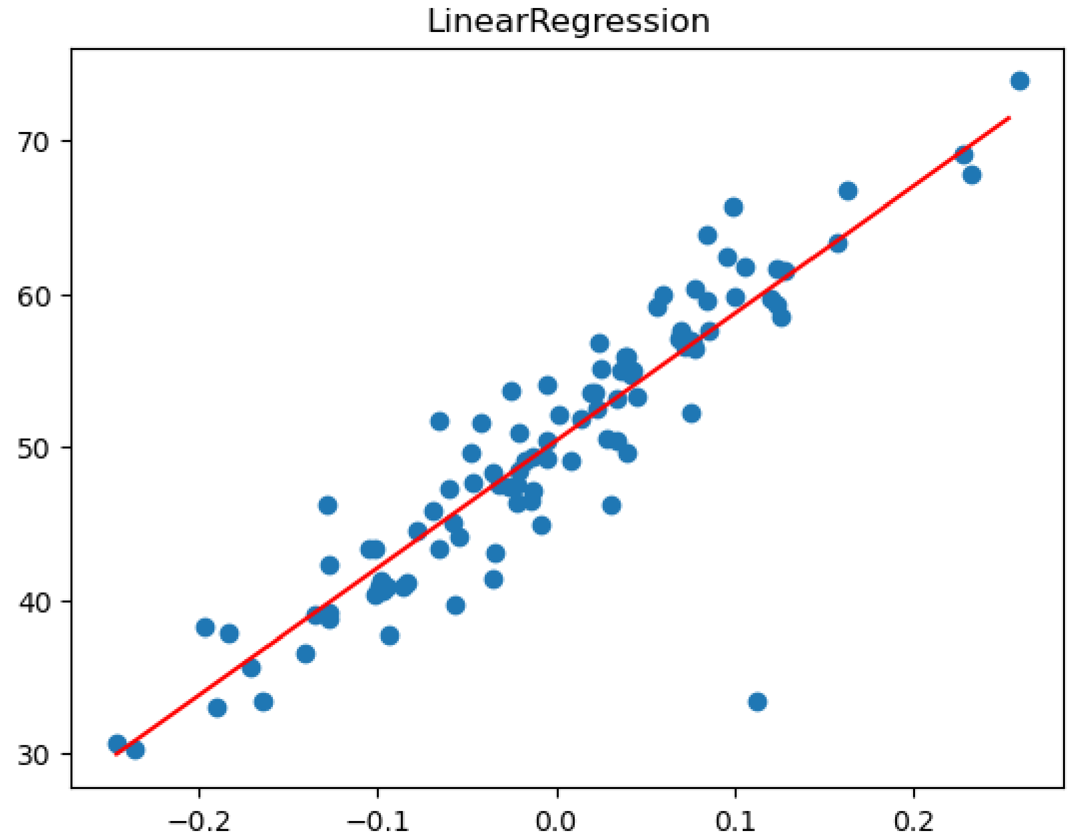
```
Binarized data:
 [[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]

BEFORE:
Mean = [ 3.775 -1.15  -1.3  ]
Std deviation = [3.12039661 6.36651396 4.0620192 ]

AFTER:
Mean = [1.11022302e-16 0.00000000e+00 2.77555756e-17]
Std deviation = [1. 1. 1.]

Min max scaled data:
 [[0.74117647 0.39548023 1.        ]
 [0.         1.         0.        ]
 [0.6        0.5819209  0.87234043]
 [1.         0.         0.17021277]]

L1 normalized data:
 [[ 0.45132743 -0.25663717  0.2920354 ]
 [-0.0794702   0.51655629 -0.40397351]
 [ 0.609375    0.0625      0.328125  ]
 [ 0.33640553 -0.4562212  -0.20737327]]

L2 normalized data:
 [[ 0.75765788 -0.43082507  0.49024922]
 [-0.12030718  0.78199664 -0.61156148]
 [ 0.87690281  0.08993875  0.47217844]
 [ 0.55734935 -0.75585734 -0.34357152]]
```

In [5]:
```python
# linear regression model
from numpy.random import rand
from sklearn.datasets import make_regression
from sklearn.metrics import mean_squared_error
# linear regression
def predict_row(row, coefficients):
    # add the bias, the last coefficient
    result = coefficients[-1]
    # add the weighted input
    for i in range(len(row)):
        result += coefficients[i] * row[i]
    return result
# use model coefficients to generate predictions for a dataset of rows
def predict_dataset(X, coefficients):
    yhats = list()
    for row in X:
    # make a prediction
        yhat = predict_row(row, coefficients)
    # store the prediction
        yhats.append(yhat)
    return yhats
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=2,
noise=0.2, random_state=1)
# determine the number of coefficients
n_coeff = X.shape[1] + 1
# generate random coefficients
coefficients = rand(n_coeff)
# generate predictions for dataset
yhat = predict_dataset(X, coefficients)
# calculate model prediction error
score = mean_squared_error(y, yhat)
print('MSE: %f' % score)
```

```
MSE: 7173.497511
```

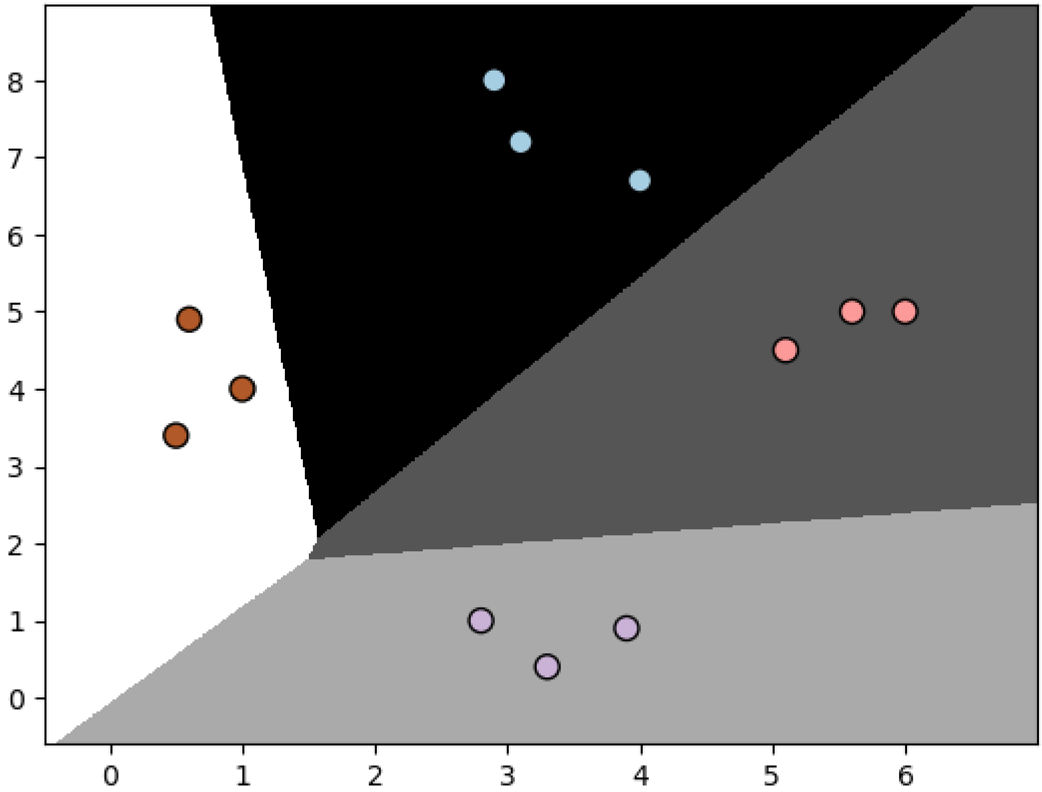
In [8]:
```python
# linear regression on a dataset with outliers
from random import random
from random import randint
from random import seed
from numpy import arange
from numpy import mean
from numpy import std
from numpy import absolute
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from matplotlib import pyplot
# prepare the dataset
def get_dataset():
    X, y = make_regression(n_samples=100, n_features=1, tail_strength=0.9,
effective_rank=1, n_informative=1, noise=3, bias=50, random_state=1)
    # add some artificial outliers
    seed(1)
    for i in range(10):
        factor = randint(2, 4)
    if random() > 0.5:
        X[i] += factor * X.std()
    else:
        X[i] -= factor * X.std()
    return X, y
# evaluate a model
def evaluate_model(X, y, model):
    # define model evaluation method
    cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
    # evaluate model
    scores = cross_val_score(model, X, y,
scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
    # force scores to be positive
    return absolute(scores)

# plot the dataset and the model's line of best fit
def plot_best_fit(X, y, model):
    # fut the model on all data
    model.fit(X, y)
    # plot the dataset
    pyplot.scatter(X, y)
    # plot the line of best fit
    xaxis = arange(X.min(), X.max(), 0.01)
    yaxis = model.predict(xaxis.reshape((len(xaxis), 1)))
    pyplot.plot(xaxis, yaxis, color='r')
    # show the plot
    pyplot.title(type(model).__name__)
    pyplot.show()

# load dataset
X, y = get_dataset()
# define the model
model = LinearRegression()
# evaluate model
results = evaluate_model(X, y, model)
print('Mean MAE: %.3f (%.3f)' % (mean(results), std(results)))
# plot the line of best fit
plot_best_fit(X, y, model)
```

```
Mean MAE: 2.413 (0.819)
```


LinearRegression
70
60
50
40
30
−0.2
−0.1
0.0
0.1
0.2

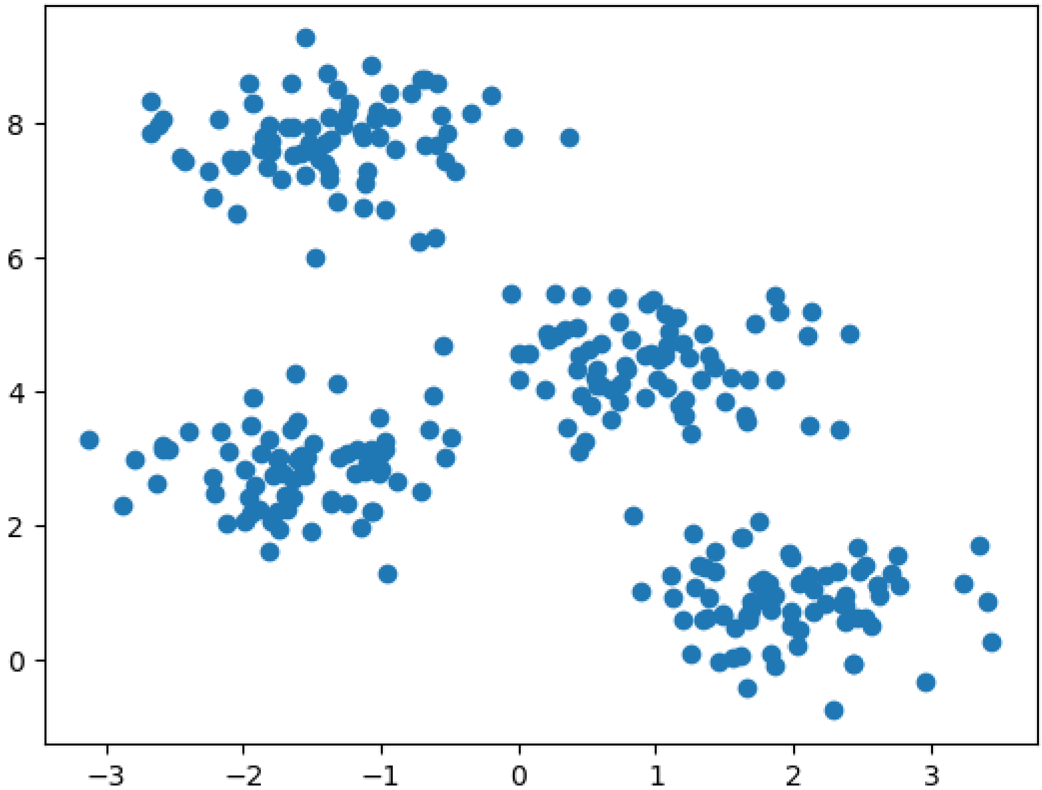
In [17]:

```
import numpy as np
import matplotlib.pyplot as plt
def visualize_classifier(classifier, X, y):
# Define the minimum and maximum values for X and Y
# that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
# Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01
# Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size),
np.arange(min_y, max_y, mesh_step_size))
 # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
# Reshape the output array
    output = output.reshape(x_vals.shape)
# Create a plot
    plt.figure()
# Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)
# Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black',
linewidth=1, cmap=plt.cm.Paired)
 # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())
# Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1),
1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1),
1.0)))
    plt.show()
# Define sample input data
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5], [6, 5], [5.6, 5],
[3.3, 0.4], [3.9, 0.9], [2.8, 1], [0.5, 3.4], [1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])
# Create the logistic regression classifier
classifier = linear_model.LogisticRegression(solver='liblinear', C=1)
# Train the classifier
classifier.fit(X, y)
# Visualize the performance of the classifier
visualize_classifier(classifier, X, y)
```



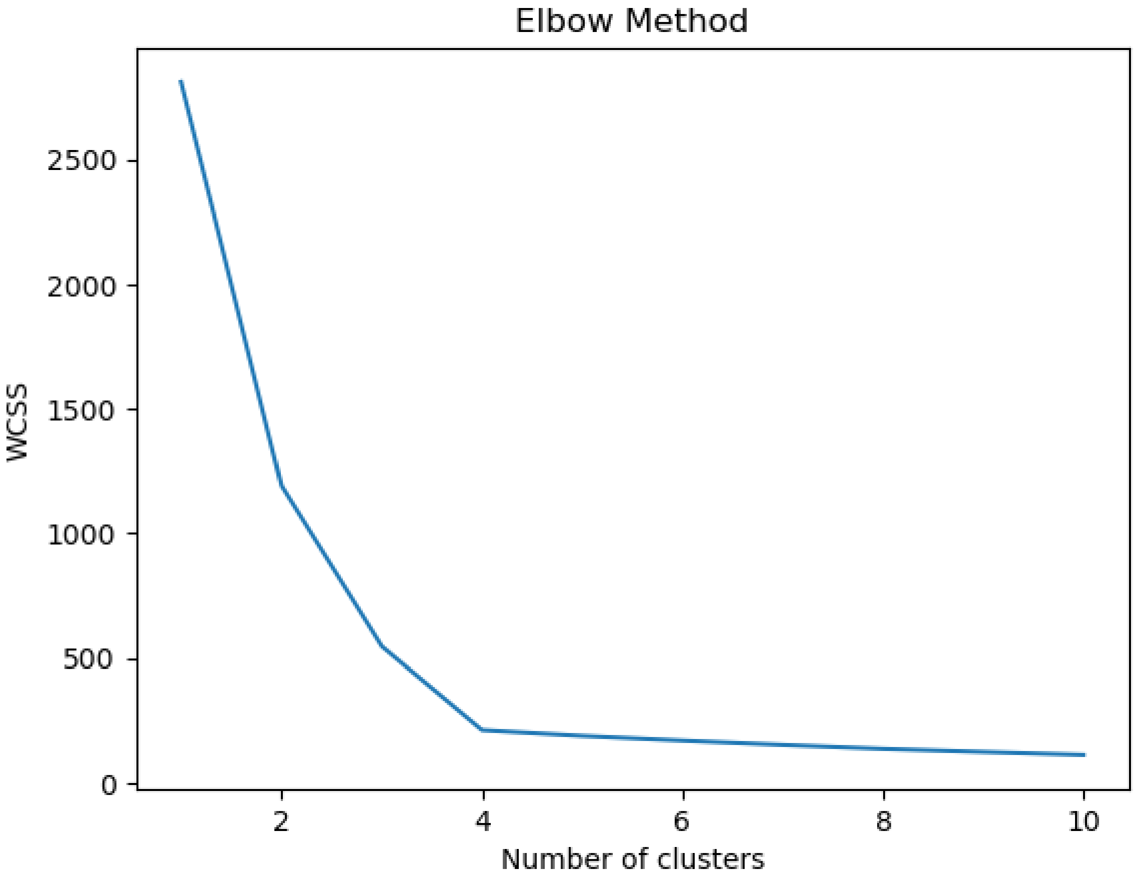
In [23]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
plt.scatter(X[:,0], X[:,1])
```

Out[23]:
```
<matplotlib.collections.PathCollection at 0x1de514234f0>
```

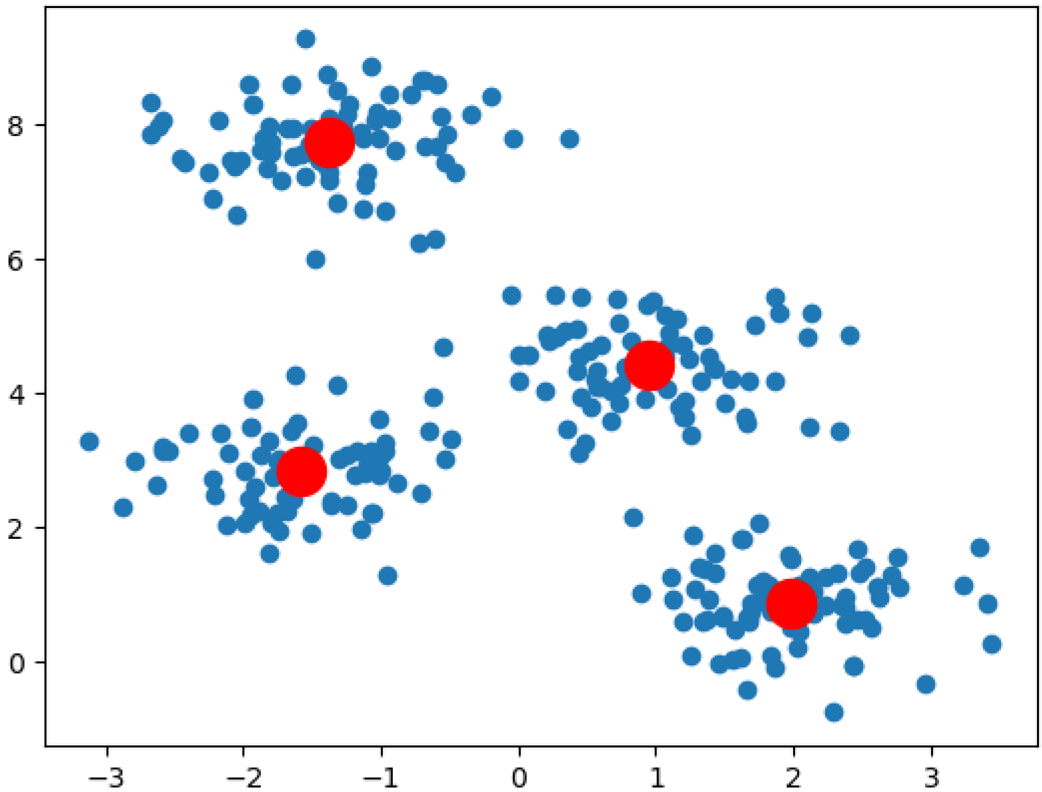


In [25]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
# plt.scatter(X[:,0], X[:,1])
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,
    random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, random_state=
pred_y = kmeans.fit_predict(X)
plt.scatter(X[:,0], X[:,1])
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s=300, c='red')
plt.show()
```

```
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=2.
  warnings.warn(
```




```
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=2.
  warnings.warn(
```



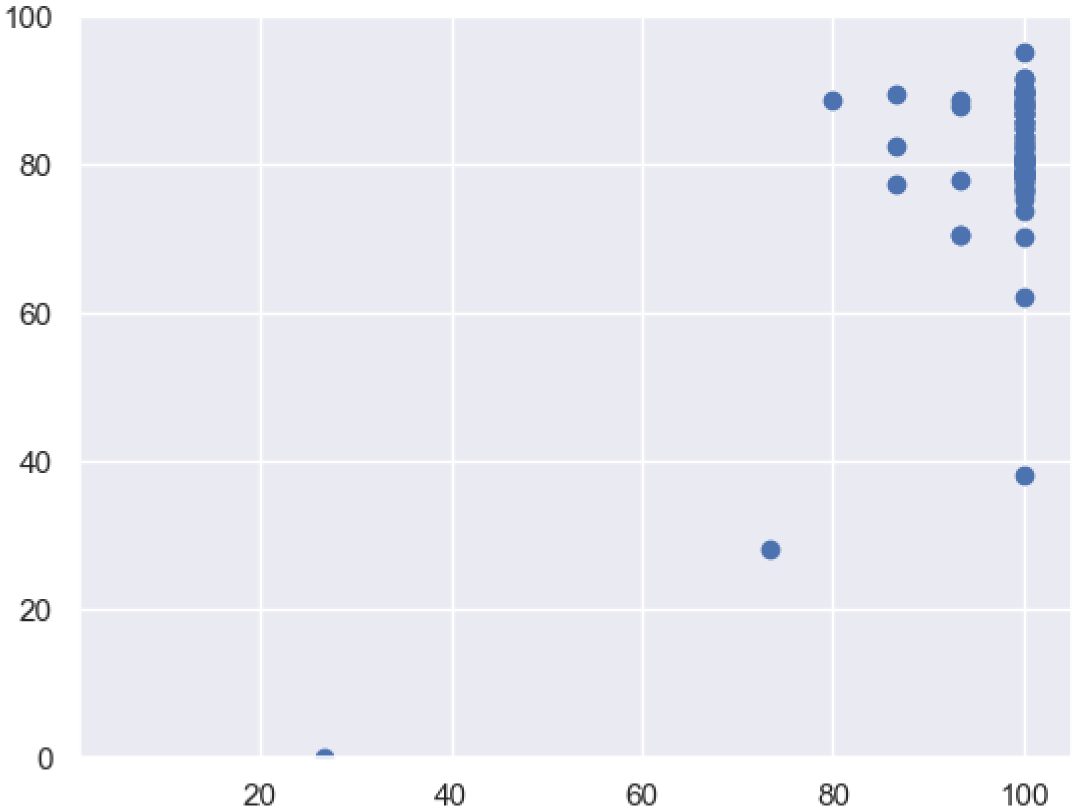
In [30]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
data = pd.read_csv('daspro.csv')
data
```

Out[30]:

	Kehadiran	Part	Tugas	UTS	UAS	NilaiAkhir
0	73.33	75	0.0	5	40	28.0
1	100.00	90	91.0	90	90	90.3
2	100.00	90	91.0	90	85	88.8
3	100.00	85	91.0	85	90	88.3
4	100.00	75	91.0	65	95	83.8
...	...	...	...	...	...	...
110	100.00	85	82.0	74	73	78.3
111	100.00	85	85.0	70	70	77.5
112	100.00	85	85.0	76	70	78.7
113	100.00	85	85.0	72	73	78.8
114	100.00	85	85.0	74	70	78.3

115 rows × 6 columns

In [29]:
```
plt.scatter(data['Kehadiran'],data['NilaiAkhir'])
plt.xlim(1,105)
plt.ylim(0,100)
plt.show()
```



In [31]:
```python
x = data.iloc[:,1:3] # 1t for rows and second for columns
x
```

Out[31]:

	Part	Tugas
0	75	0.0
1	90	91.0
2	90	91.0
3	85	91.0
4	75	91.0
...	...	...
110	85	82.0
111	85	85.0
112	85	85.0
113	85	85.0
114	85	85.0

115 rows × 2 columns

In [32]:
```python
kmeans = KMeans(3)
kmeans.fit(x)
```

```
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
```

Out[32]:
```
KMeans(n_clusters=3)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

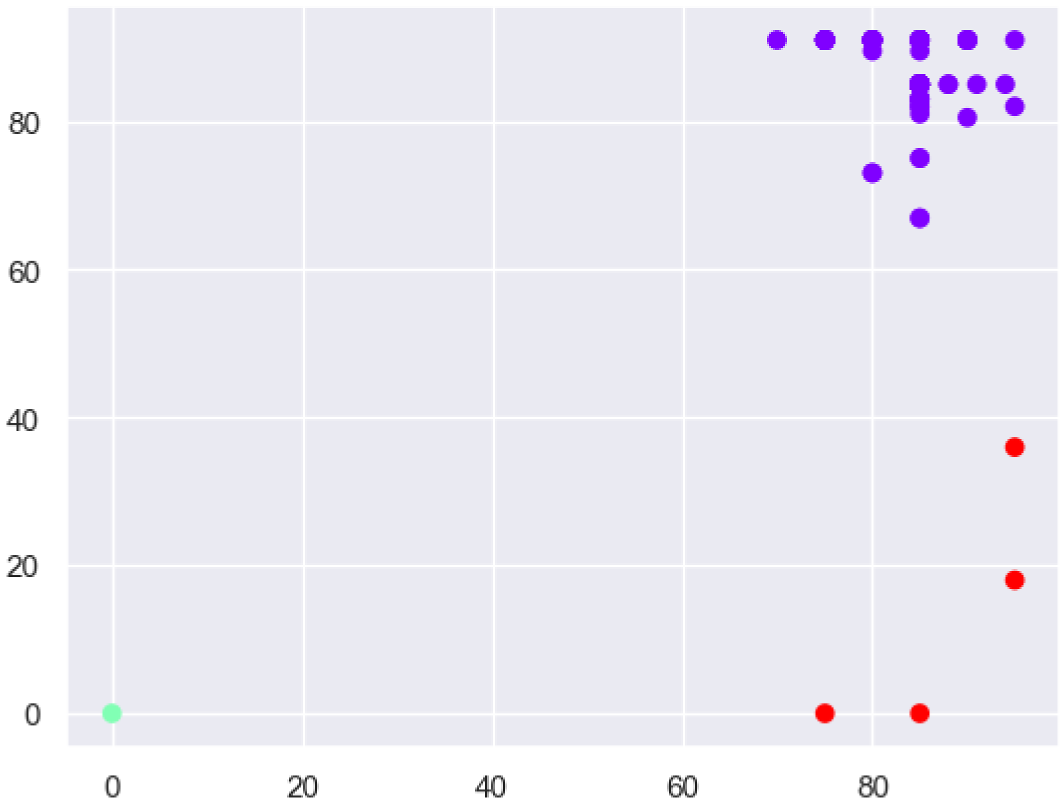
In [33]:
```python
identified_clusters = kmeans.fit_predict(x)
identified_clusters
```

```
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
```

Out[33]:
```
array([2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0])
```

In [35]:
```
data_with_clusters = data.copy()
data_with_clusters['Clusters'] = identified_clusters
plt.scatter(data_with_clusters['Part'],data_with_clusters['Tugas'],c=data_with_cluster
```

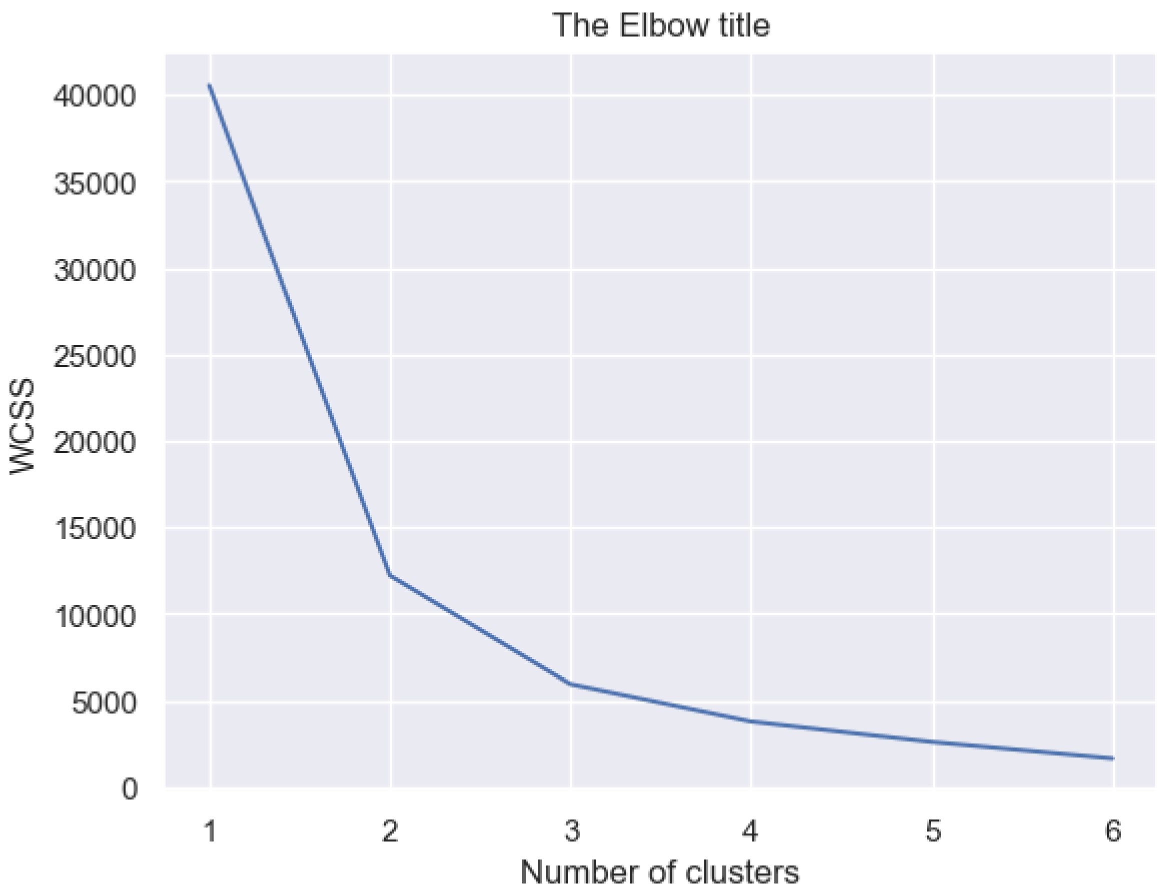
Out[35]: <matplotlib.collections.PathCollection at 0x1de53f0b7c0>



In [36]:
```
wcss=[]
for i in range(1,7):
 kmeans = KMeans(i)
 kmeans.fit(x)
 wcss_iter = kmeans.inertia_
 wcss.append(wcss_iter)
number_clusters = range(1,7)
plt.plot(number_clusters,wcss)
plt.title('The Elbow title')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

```
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
C:\Users\Iyes\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less c
hunks than available threads. You can avoid it by setting the environment variable O
MP_NUM_THREADS=1.
  warnings.warn(
```

Out[36]:
```
Text(0, 0.5, 'WCSS')
```


The Elbow title
WCSS
40000
35000
30000
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
Number of clusters

In [ ]:

```
S
```