```
Member of the project : Alexis Ribat
Advanced Machine Learning Project
```

We will analyse the Concrete Slump Test data. Data file and variables name are available on the slump.data and slump_test.names files.

Link of the Dataset : https://lipn.univ-paris13.fr/~grozavu/PredA/dataProject/6%20-%20slump/

The Dataset contains :

```
Input variables (7)(component kg in one M^3 concrete):
    Cement
    Slag
    Fly ash
    Water
    SP
    Coarse Aggr.
    Fine Aggr.

Output variables (3):
    SLUMP (cm)
    FLOW (cm)
    28-day Compressive Strength (Mpa)
```

What is Compressive Strength of Concrete ?

Compressive strength of concrete is the Strength of hardened concrete measured by the compression test. The compression strength of concrete is a measure of the concrete's ability to resist loads which tend to compress it. It is measured by crushing cylindrical concrete specimens in compression testing machine.

Compressive strength results are primarily used to determine that the concrete mixture as delivered on site meets the requirements of the specified strength.

A test result is the average of at least two standard-cured strength specimens made from the same concrete batch and tested at the same age. In most cases strength requirements for concrete are at 28 days.

As we face the problem of pollution and optimization of our ressources, we might want to find the Concrete with the best capacity required, and with the lest costs of construction. In order to do that, we will see the impact of other ingredients and their role in the Compressive Strength of Concrete.

In [62]:
```python
from platform import python_version

print(python_version())
```

```
3.8.3
```

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.tree import export_graphviz
from sklearn.metrics import r2_score
from sklearn.metrics import SCORERS
import statistics
```

In [2]:
```python
f = open('slump_test.data',"r")
data = []
for x in f:
    data.append(x)
df = pd.DataFrame(data)
df = df[0].str.split(',', expand=True)
df.iloc[:, -1] = df.iloc[:, -1].apply(lambda x: x.rstrip())
df.columns = df.iloc[0]
df = df.rename(columns = {'SLUMP(cm)': 'Slump',
                          'FLOW(cm)': 'Flow',
                          'Compressive Strength (28-day)(Mpa)': 'CS'})
df = df.iloc[1:, 1:]
df = df.apply(pd.to_numeric) #convert to numeric format
df = df.drop_duplicates(keep='first') #delete potential duplicates


with open('slump_test.names', 'r') as myfile:
    description = myfile.read().replace('\n', ' ').replace('\r', '')
```

In [3]:
```python
df.head(10)
```

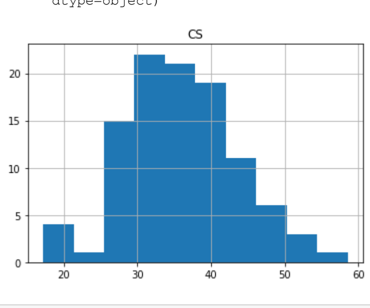
Out[3]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Slump	Flow	CS
1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
6	147.0	89.0	115.0	202.0	9.0	860.0	829.0	23.0	55.0	25.21
7	152.0	139.0	178.0	168.0	18.0	944.0	695.0	0.0	20.0	38.86
8	145.0	0.0	227.0	240.0	6.0	750.0	853.0	14.5	58.5	36.59
9	152.0	0.0	237.0	204.0	6.0	785.0	892.0	15.5	51.0	32.71
10	304.0	0.0	140.0	214.0	6.0	895.0	722.0	19.0	51.0	38.46

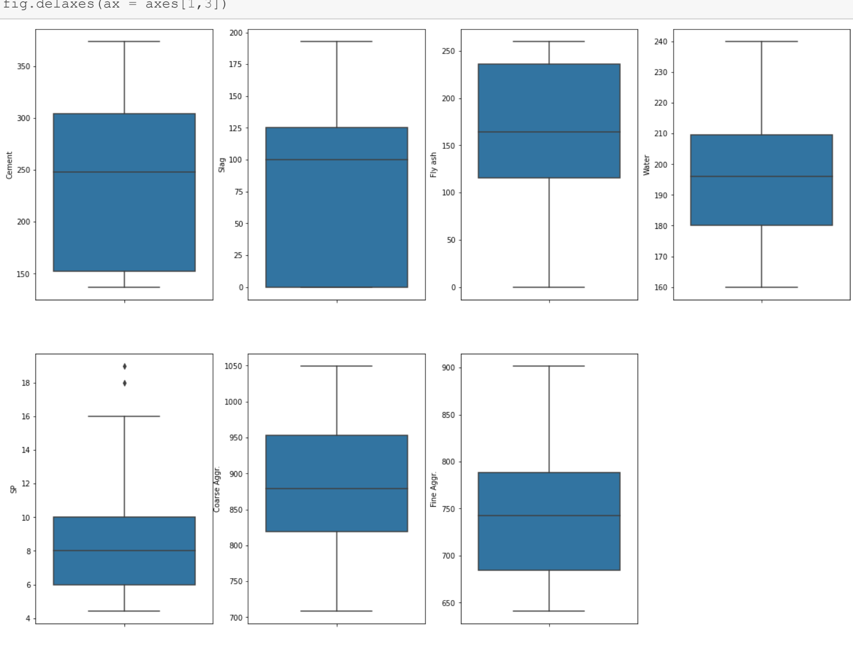
Comme on a des valeurs continue, on effectue un Histogramme sur la colonne 'Compressive_Strength'

In [4]:
```python
df.hist('CS')
```

Out[4]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x00000219A58C8E80>]],
      dtype=object)
```



In [5]:
```python
features_to_analyse = df.columns[:7]
current_palette = sns.color_palette("Blues")

fig, axes = plt.subplots(round(len(features_to_analyse)/4), 4, figsize = (20, 15))

for i, ax in enumerate(fig.axes):
    if i < len(features_to_analyse):
        sns.boxplot(x=features_to_analyse[i], data = df, ax = ax, orient = 'v')
fig.delaxes(ax = axes[1,3])
```



Conclusion. There is no a lot of ouliers in the data set, we can only observe 2 points of outliers in SP (superplasticizer).

Almost all features (except for Coarse aggr.) are dispersed.

Since we will us tree-base model, we will no need to normalize or scale data because tree-base models are not sensitive to this.

In [6]:
```python
df
```

Out[6]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Slump	Flow	CS
1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
2	163.0	149.0	191.0	180.0	12.0	843.0	746.0	0.0	20.0	41.14
3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
4	162.0	148.0	190.0	179.0	19.0	838.0	741.0	3.0	21.5	42.08
5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
...	...	...	...	...	...	...	...	...	...	...
99	248.3	101.0	239.1	168.9	7.7	954.2	640.6	0.0	20.0	49.97
100	248.0	101.0	239.9	169.1	7.7	949.9	644.1	2.0	20.0	50.23
101	258.8	88.0	239.6	175.3	7.6	938.9	646.0	0.0	20.0	50.50
102	297.1	40.9	239.9	194.0	7.5	908.9	651.8	27.5	67.0	49.17
103	348.7	0.1	223.1	208.5	9.6	786.2	758.1	29.0	78.0	48.77

103 rows × 10 columns

We want to get a total count of missing values.

In [7]:
```python
df.isnull().sum().sum()
```

Out[7]:
```
0
```

There is no missing value

In [8]:
```python
df.dtypes
```

Out[8]:
```
0
Cement          float64
Slag            float64
Fly ash         float64
Water           float64
SP              float64
Coarse Aggr.    float64
Fine Aggr.      float64
Slump           float64
Flow            float64
CS              float64
dtype: object
```

Checking if there are some outliers

In [9]:
```python
df.describe()
```

Out[9]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Slump	Flow	CS
count	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000	103.000000	103.00000	103.000000
mean	229.894175	77.973786	149.014563	197.167961	8.539806	883.978641	739.604854	18.048544	49.61068	36.039417
std	78.877230	60.461363	85.418080	20.208158	2.807530	88.391393	63.342117	8.750844	17.56861	7.838232
min	137.000000	0.000000	0.000000	160.000000	4.400000	708.000000	640.600000	0.000000	20.00000	17.190000
25%	152.000000	0.050000	115.500000	180.000000	6.000000	819.500000	684.500000	14.500000	38.50000	30.900000
50%	248.000000	100.000000	164.000000	196.000000	8.000000	879.000000	742.700000	21.500000	54.00000	35.520000
75%	303.900000	125.000000	235.950000	209.500000	10.000000	952.800000	788.000000	24.000000	63.75000	41.205000
max	374.000000	193.000000	260.000000	240.000000	19.000000	1049.900000	902.000000	29.000000	78.00000	58.530000

In [10]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 103 entries, 1 to 103
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Cement        103 non-null    float64
 1   Slag          103 non-null    float64
 2   Fly ash       103 non-null    float64
 3   Water         103 non-null    float64
 4   SP            103 non-null    float64
 5   Coarse Aggr.  103 non-null    float64
 6   Fine Aggr.    103 non-null    float64
 7   Slump         103 non-null    float64
 8   Flow          103 non-null    float64
 9   CS            103 non-null    float64
dtypes: float64(10)
memory usage: 8.9 KB
```

Interquartile range

In [11]:
```python
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
0
Cement          151.900
Slag            124.950
Fly ash         120.450
Water            29.500
SP                4.000
Coarse Aggr.    133.300
Fine Aggr.      103.500
Slump             9.500
Flow             25.250
CS               10.305
dtype: float64
```

The above output prints the IQR scores, which can be used to detect outliers. The code below generates an output with the 'True' and 'False' values. Points where the values are 'True' represent the presence of the outlier.

In [12]:
```python
print((df < (Q1 - 1.5 * IQR)) &(df > (Q3 + 1.5 * IQR)))
```

```
0    Cement   Slag  Fly ash  Water     SP  Coarse Aggr.  Fine Aggr.  Slump  \
1     False  False    False  False  False         False       False  False
2     False  False    False  False  False         False       False  False
3     False  False    False  False  False         False       False  False
4     False  False    False  False  False         False       False  False
5     False  False    False  False  False         False       False  False
..      ...    ...      ...    ...    ...           ...         ...    ...
99    False  False    False  False  False         False       False  False
100   False  False    False  False  False         False       False  False
101   False  False    False  False  False         False       False  False
102   False  False    False  False  False         False       False  False
103   False  False    False  False  False         False       False  False

0     Flow     CS
1    False  False
2    False  False
3    False  False
4    False  False
5    False  False
..     ...    ...
99   False  False
100  False  False
101  False  False
102  False  False
103  False  False

[103 rows x 10 columns]
```

This technique uses the IQR scores calculated earlier to remove outliers. The rule of thumb is that anything not in the range of (Q1 - 1.5 IQR) and (Q3 + 1.5 IQR) is an outlier, and can be removed. The first line of code below removes outliers based on the IQR range and stores the result in the data frame 'df_out'.

In [13]:
```python
df = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

Out[13]:
```
(91, 10)
```

After removing outliers, we now have a table of 91 rows and 10 columns. It removed 12 rows.

Our new clean Dataset df_out looks like that now

In [14]:
```python
df
```

Out[14]:

	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Slump	Flow	CS
1	273.0	82.0	105.0	210.0	9.0	904.0	680.0	23.0	62.0	34.99
3	162.0	148.0	191.0	179.0	16.0	840.0	743.0	1.0	20.0	41.81
5	154.0	112.0	144.0	220.0	10.0	923.0	658.0	20.0	64.0	26.82
6	147.0	89.0	115.0	202.0	9.0	860.0	829.0	23.0	55.0	25.21
8	145.0	0.0	227.0	240.0	6.0	750.0	853.0	14.5	58.5	36.59
...	...	...	...	...	...	...	...	...	...	...
97	215.6	112.9	239.0	198.7	7.4	884.0	649.1	27.5	64.0	39.13
98	295.3	0.0	239.9	236.2	8.3	780.3	722.9	25.0	77.0	44.08
100	248.0	101.0	239.9	169.1	7.7	949.9	644.1	2.0	20.0	50.23
102	297.1	40.9	239.9	194.0	7.5	908.9	651.8	27.5	67.0	49.17
103	348.7	0.1	223.1	208.5	9.6	786.2	758.1	29.0	78.0	48.77

91 rows × 10 columns

Now that we have a clean dataset to work with, we will evaluate the correlations between the data

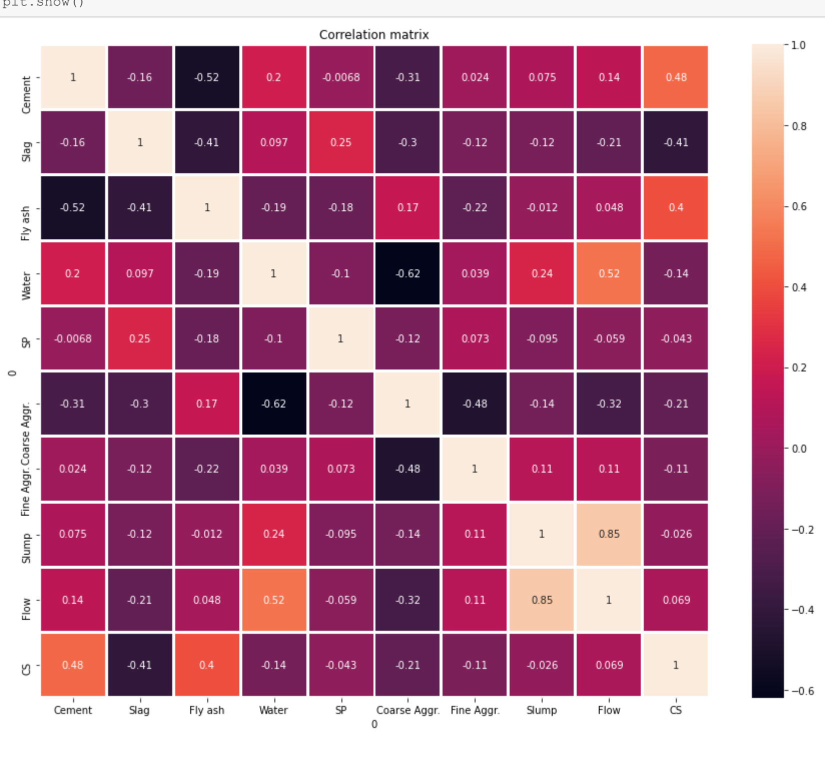
Correlation Matrix

In [15]:
```python
df.corr()
```

Out[15]:

0	Cement	Slag	Fly ash	Water	SP	Coarse Aggr.	Fine Aggr.	Slump	Flow	CS
Cement	1.000000	-0.162138	-0.521057	0.203663	-0.006840	-0.311702	0.024127	0.075126	0.138550	0.484557
Slag	-0.162138	1.000000	-0.408213	0.097353	0.249394	-0.296474	-0.119900	-0.121791	-0.211580	-0.406423
Fly ash	-0.521057	-0.408213	1.000000	-0.190889	-0.184992	0.165525	-0.222959	-0.012493	0.047922	0.399072
Water	0.203663	0.097353	-0.190889	1.000000	-0.100577	-0.619445	0.039414	0.243352	0.520136	-0.135579
SP	-0.006840	0.249394	-0.184992	-0.100577	1.000000	-0.121950	0.073446	-0.095434	-0.058584	-0.043374
Coarse Aggr.	-0.311702	-0.296474	0.165525	-0.619445	-0.121950	1.000000	-0.479102	-0.142116	-0.322205	-0.211781
Fine Aggr.	0.024127	-0.119900	-0.222959	0.039414	0.073446	-0.479102	1.000000	0.114184	0.113157	-0.111977
Slump	0.075126	-0.121791	-0.012493	0.243352	-0.095434	-0.142116	0.114184	1.000000	0.852936	-0.026058
Flow	0.138550	-0.211580	0.047922	0.520136	-0.058584	-0.322205	0.113157	0.852936	1.000000	0.069205
CS	0.484557	-0.406423	0.399072	-0.135579	-0.043374	-0.211781	-0.111977	-0.026058	0.069205	1.000000

In [16]:
```python
corr = df.corr()

plt.figure(figsize=(15,12))
sns.heatmap(corr,
           xticklabels=corr.columns.values,
           yticklabels=corr.columns.values, linewidth=2,annot = True)
plt.title("Correlation matrix")

plt.show()
```



We can see high correlation between:

. Cement & Fly_Ash

. Water & Coarse Aggregation

. Flow & Slump

For the Compressive Strength, we can see that the best correlations are Cement and Fly_ash.

Now, we will cut the data in order to segment and sort data values into bins, since we have a continuous variableas target and we want to convert it into categorical data to analyze it.

This technique is very usefull to convert a Regression Problems into a Classification one.

In [17]:
```python
correlation_mat = corr.corr()
corr_pairs = correlation_mat.unstack()
```

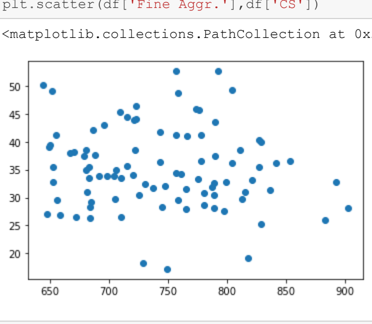
This will show the correlation that are > 0.5 and <-0.5

In [18]:
```python
sorted_pairs = corr_pairs.sort_values(kind="quicksort")
strong_pairs = sorted_pairs[abs(sorted_pairs) > 0.5]

print(strong_pairs)
```

```
Coarse Aggr.  Water          -0.790030
Water         Coarse Aggr.   -0.790030
Slag          CS             -0.625808
CS            Slag           -0.625808
Fine Aggr.    Coarse Aggr.   -0.625003
Coarse Aggr.  Fine Aggr.     -0.625003
Fly ash       Slag           -0.594630
Slag          Fly ash        -0.594630
Cement        Fly ash        -0.523353
Fly ash       Cement         -0.523353
SP            Slag            0.504440
Slag          SP              0.504440
Slump         Water           0.525532
Water         Slump           0.525532
Flow          Water           0.707778
Water         Flow            0.707778
Slump         Flow            0.949211
Flow          Slump           0.949211
Cement        Cement          1.000000
Slump         Slump           1.000000
Fine Aggr.    Fine Aggr.      1.000000
Coarse Aggr.  Coarse Aggr.    1.000000
SP            SP              1.000000
Water         Water           1.000000
Fly ash       Fly ash         1.000000
Slag          Slag            1.000000
Flow          Flow            1.000000
CS            CS              1.000000
dtype: float64
```

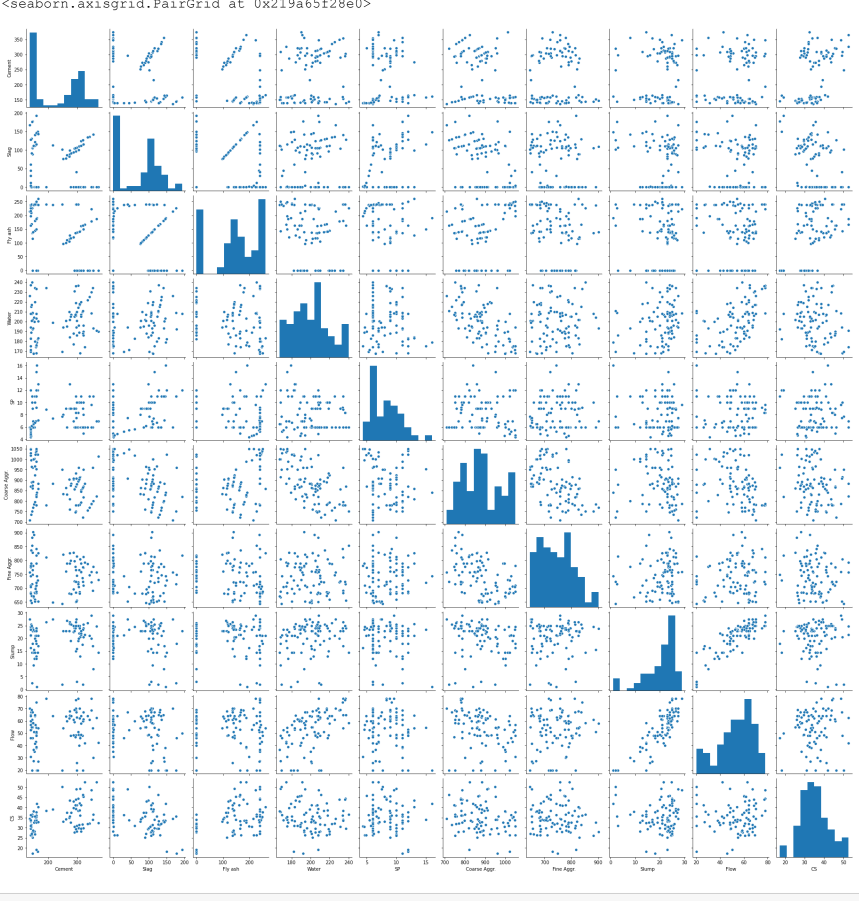
In [19]:
```python
plt.scatter(df['Fine Aggr.'],df['CS'])
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x219a63880d0>
```



In [20]:
```python
sns.pairplot(df)
```

Out[20]:
```
<seaborn.axisgrid.PairGrid at 0x219a65f28e0>
```



In [21]:
```python
X = df[['Cement','Fly ash']].values
y = df['CS'].values
```

KMeans before PCA

In [22]:
```python
import matplotlib.pyplot as plt
from kneed import KneeLocator
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
```

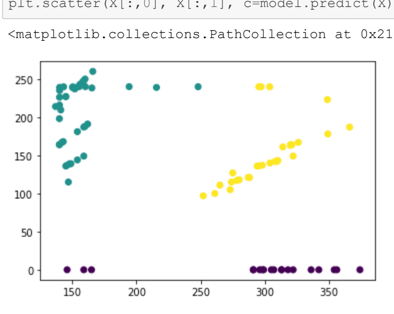
In [23]:
```python
model = KMeans(n_clusters = 3)
```

In [24]:
```python
model.fit(X)
model.predict(X)
```

Out[24]:
```
array([2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 0, 0, 1, 1, 2, 2, 0, 0, 2, 0, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 0, 1, 1, 1, 1, 1, 0, 0, 0, 2, 2, 2,
       2, 0, 0, 2, 2, 0, 0, 0, 1, 1, 2, 2, 0, 0, 2, 1, 1, 1, 2, 0, 0, 0,
       2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2,
       1, 2, 2])
```

In [25]:
```python
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x219ac747f10>
```

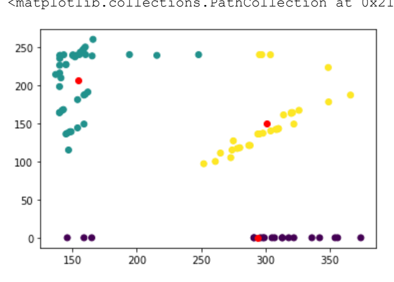


In [26]:
```python
model.cluster_centers_  #Affiche les centoides
```

Out[26]:
```
array([[ 2.93947368e+02, -2.84217094e-14],
       [ 1.54731707e+02,  2.06778049e+02],
       [ 3.01254839e+02,  1.50409677e+02]])
```

In [27]:
```python
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1], c='r')
```

Out[27]:
```
<matplotlib.collections.PathCollection at 0x219ac7a1d60>
```



In [28]:
```python
model.score(X)
```

Out[28]:
```
-231626.0702637791
```

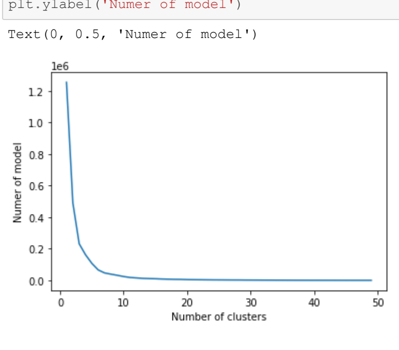
In [29]:
```python
inertia = []
```

In [30]:
```python
K_range = range(1,50)
for k in K_range:
    model=KMeans(n_clusters=k).fit(X)
    inertia.append(model.inertia_)
```

```
C:\Users\alrib\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: 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=1.
  warnings.warn(
```

In [31]:
```python
plt.plot(K_range, inertia)
plt.xlabel('Number of clusters')
plt.ylabel('Numer of model')
```

Out[31]:
```
Text(0, 0.5, 'Numer of model')
```



Optimal number of cluster is 5 or 6

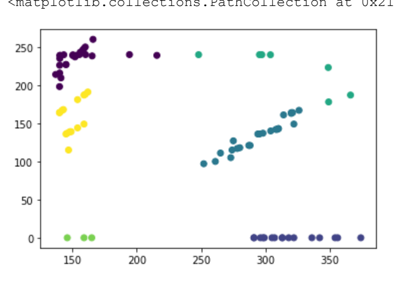
In [32]:
```python
model = KMeans(n_clusters = 6)
```

In [33]:
```python
model.fit(X)
model.predict(X)
```

Out[33]:
```
array([2, 5, 5, 5, 0, 0, 2, 5, 5, 5, 1, 1, 5, 0, 2, 2, 1, 1, 2, 4, 5, 2,
       2, 2, 2, 2, 3, 3, 2, 0, 2, 2, 4, 0, 0, 0, 5, 5, 1, 1, 1, 2, 2, 2,
       2, 1, 1, 2, 2, 4, 1, 1, 5, 5, 2, 2, 1, 1, 2, 0, 0, 0, 2, 1, 1, 1,
       2, 5, 5, 5, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3,
       3, 3, 3])
```

In [34]:
```python
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
```

Out[34]:
```
<matplotlib.collections.PathCollection at 0x219ad9250d0>
```

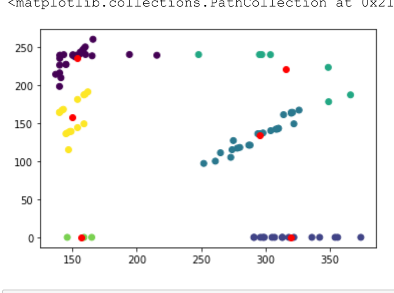


In [35]:
```python
model.cluster_centers_  #Affiche les centoides
```

Out[35]:
```
array([[ 1.53920000e+02,  2.34760000e+02],
       [ 3.19687500e+02, -2.84217094e-14],
       [ 2.95160000e+02,  1.34200000e+02],
       [ 3.15414286e+02,  2.21085714e+02],
       [ 1.56666667e+02,  0.00000000e+00],
       [ 1.49866667e+02,  1.57933333e+02]])
```

In [36]:
```python
plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1], c='r')
```

Out[36]:
```
<matplotlib.collections.PathCollection at 0x219ad971a90>
```



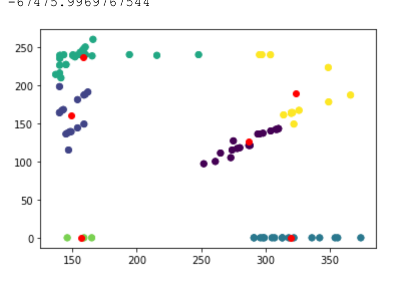
In [37]:
```python
model.score(X)
```

Out[37]:
```
-65935.78797619048
```

-65937 is still a bad score but it is better than the old score of -231626

In [38]:
```python
model.fit(X)
model.predict(X)

plt.scatter(X[:,0], X[:,1], c=model.predict(X))

model.cluster_centers_  #Affiche les centoides

plt.scatter(X[:,0], X[:,1], c=model.predict(X))
plt.scatter(model.cluster_centers_[:,0],model.cluster_centers_[:,1], c='r')

model.score(X)
```

Out[38]:
```
-67475.9969767544
```



PCA

In [39]:
```python
from sklearn.decomposition import PCA
```

In [40]:
```python
Xpca = df[['Cement','Slag', 'Fly ash','Water','SP','Coarse Aggr.','Fine Aggr.']].values
```

In [41]:
```python
Xpca.shape
```

Out[41]:
```
(91, 7)
```

In [42]:
```python
model = PCA(n_components=7)
```

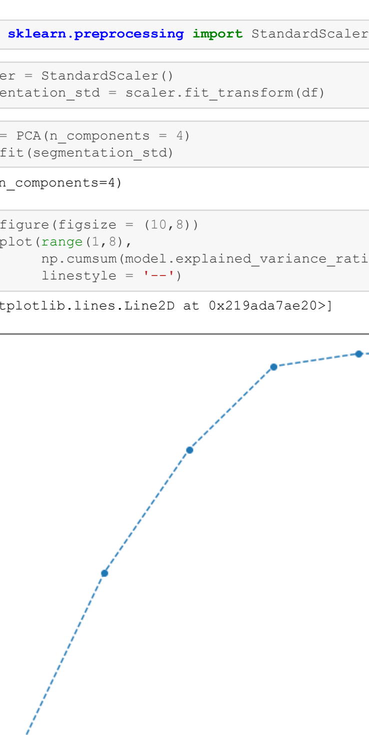
In [43]:
```python
Xpca_reduced = model.fit_transform(Xpca)
```

In [44]:
```python
np.cumsum(model.explained_variance_ratio_)
```

Out[44]:
```
array([0.43991165, 0.68943404, 0.86352295, 0.98144686, 0.99935413,
       0.99990526, 1.        ])
```

We want to reduce the dataset AND having the most amount of information. Let say that 98% of the information is enough. It means we will keep 4 variables.

In [45]:
```
plt.plot(np.cumsum(model.explained_variance_ratio_))
```
Out[45]:
```
[<matplotlib.lines.Line2D at 0x219ada23610>]
```



In [46]:
```
from sklearn.preprocessing import StandardScaler
```

In [47]:
```
scaler = StandardScaler()
segmentation_std = scaler.fit_transform(df)
```

In [48]:
```
pca = PCA(n_components = 4)
pca.fit(segmentation_std)
```
Out[48]:
```
PCA(n_components=4)
```

In [49]:
```
plt.figure(figsize = (10,8))
plt.plot(range(1,8),
        np.cumsum(model.explained_variance_ratio_), marker = 'o',
        linestyle = '--')
```
Out[49]:
```
[<matplotlib.lines.Line2D at 0x219ada7ae20>]
```

In [50]:
```
scores_pca = pca.transform(segmentation_std)
```

KMeans after PCA

In [51]:
```
inertia = []
```

In [52]:
```
K_range = range(1,21)
for k in K_range:
    kmeans_pca = KMeans(n_clusters=k , init = "k-means++", random_state = 42)
    kmeans_pca.fit(scores_pca)
    inertia.append(kmeans_pca.inertia_)
```
```
C:\Users\alrib\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is kno
wn 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=1.
  warnings.warn(
```

In [53]:
```
plt.plot(K_range, inertia)
plt.xlabel('Number of clusters')
plt.ylabel('Numer of model')
```
Out[53]:
```
Text(0, 0.5, 'Numer of model')
```


Optimal number of cluster is 5 or 6

Now, let implement it into our KMeans algo

In [54]:
```
kmeans_pca = KMeans(n_clusters = 6, init = "k-means++", random_state = 42)
```

In [55]:
```
kmeans_pca.fit(scores_pca)
```
Out[55]:
```
KMeans(n_clusters=6, random_state=42)
```

In [56]:
```
labels = kmeans_pca.predict(scores_pca)
```

In [57]:
```
PCA_components = pd.DataFrame(segmentation_std)
kmeans_pca.fit(PCA_components.iloc[:,:4])
```
Out[57]:
```
KMeans(n_clusters=6, random_state=42)
```

In [58]:
```
labels = kmeans_pca.predict(PCA_components.iloc[:,:4])
plt.scatter(PCA_components[0], PCA_components[1], c=labels)
plt.show()
```