0- Preliminary observations and road-map based on the so-far acquired know-how :

1- The data type of the Output variable [Concrete compressive strength] is quantitative: hence linear/polynomial regression models are expected to be used for prediction.

2- Since the problem is not of the classification type : Confiusion Matrix , TP , TN , FP and FN and their dependent measures of performance such as Recall, Precision , Accuracy are not valid indicators of model performance rather the R-squarred or SSE.

3- Also due to the nature of the problem: Oversampling and Downsampling by using Imblean techniques seem irrelevant.

4- Hyperpapramters tuning in Linear/polynomial models by using GridsearchCV or RandomSearchCV also seem to be out of the picture.

5- The only relevant performance enhancers of the predictive models that could be explored : K-fold cross-validation, exploration of proxies in the attributes (features), regularisation of the attributes, application of possible features elimination by using Lasso and Ridge Shrinkage methods.

All these are to be confimred as we move onto exploring the data set.

1- Univariate Analysis

In [1]:

```
#1.1 Libraries and tools for this stage

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
#1.2 Loading the Data Frame : conc_df

conc_df = pd.read_csv('concrete (1).csv')
```

In [3]:

```
#1.3 First Impression

conc_df.head()
```

Out[3]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29

In [4]:

```
#1.4 Data size and types

conc_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
cement          1030 non-null float64
slag            1030 non-null float64
ash             1030 non-null float64
water           1030 non-null float64
superplastic    1030 non-null float64
coarseagg       1030 non-null float64
fineagg         1030 non-null float64
age             1030 non-null int64
strength        1030 non-null float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

In [5]:

```
#1.5 The 5-numbers statistical summary of each column
conc_df.describe().transpose()

# First Obdservations on distributions : Strong Right-skweness (mean >> median)
and indicator of possible outliers,
# in the columns:
# - slag
# - ash
# - age
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
cement	1030.0	281.167864	104.506364	102.00	192.375	272.900	350.000	540.0
slag	1030.0	73.895825	86.279342	0.00	0.000	22.000	142.950	359.4
ash	1030.0	54.188350	63.997004	0.00	0.000	0.000	118.300	200.1
water	1030.0	181.567282	21.354219	121.80	164.900	185.000	192.000	247.0
superplastic	1030.0	6.204660	5.973841	0.00	0.000	6.400	10.200	32.2
coarseagg	1030.0	972.918932	77.753954	801.00	932.000	968.000	1029.400	1145.0
fineagg	1030.0	773.580485	80.175980	594.00	730.950	779.500	824.000	992.6
age	1030.0	45.662136	63.169912	1.00	7.000	28.000	56.000	365.0
strength	1030.0	35.817961	16.705742	2.33	23.710	34.445	46.135	82.6

In [6]:

```
#1.6 Looking for possible null values

for x in conc_df.columns:
    print( conc_df[x].value_counts())

# TOO MANY '0' were counted in the columns :
# superplastic(379) / ash(566) / slag(471) !

# With no domain expertise I cannot tell if this is a normal result in concrete
mix types.
```

```
425.0    20
362.6    20
251.4    15
446.0    14
310.0    14
         ..
312.9     1
261.9     1
325.6     1
143.8     1
```

```
145.4      1
Name: cement, Length: 278, dtype: int64
0.0       471
189.0      30
106.3      20
24.0       14
20.0       12
         ...
161.0       1
160.5       1
129.0       1
100.6       1
209.0       1
Name: slag, Length: 185, dtype: int64
0.0       566
118.3      20
141.0      16
24.5       15
79.0       14
         ...
119.0       1
134.0       1
95.0        1
130.0       1
129.7       1
Name: ash, Length: 156, dtype: int64
192.0     118
228.0      54
185.7      46
203.5      36
186.0      28
         ...
165.0       1
237.0       1
166.7       1
191.3       1
184.4       1
Name: water, Length: 195, dtype: int64
0.0      379
11.6      37
8.0       27
7.0       19
6.0       17
        ...
2.2        1
11.5       1
6.3        1
10.5       1
9.8        1
Name: superplastic, Length: 111, dtype: int64
932.0      57
852.1      45
944.7      30
```

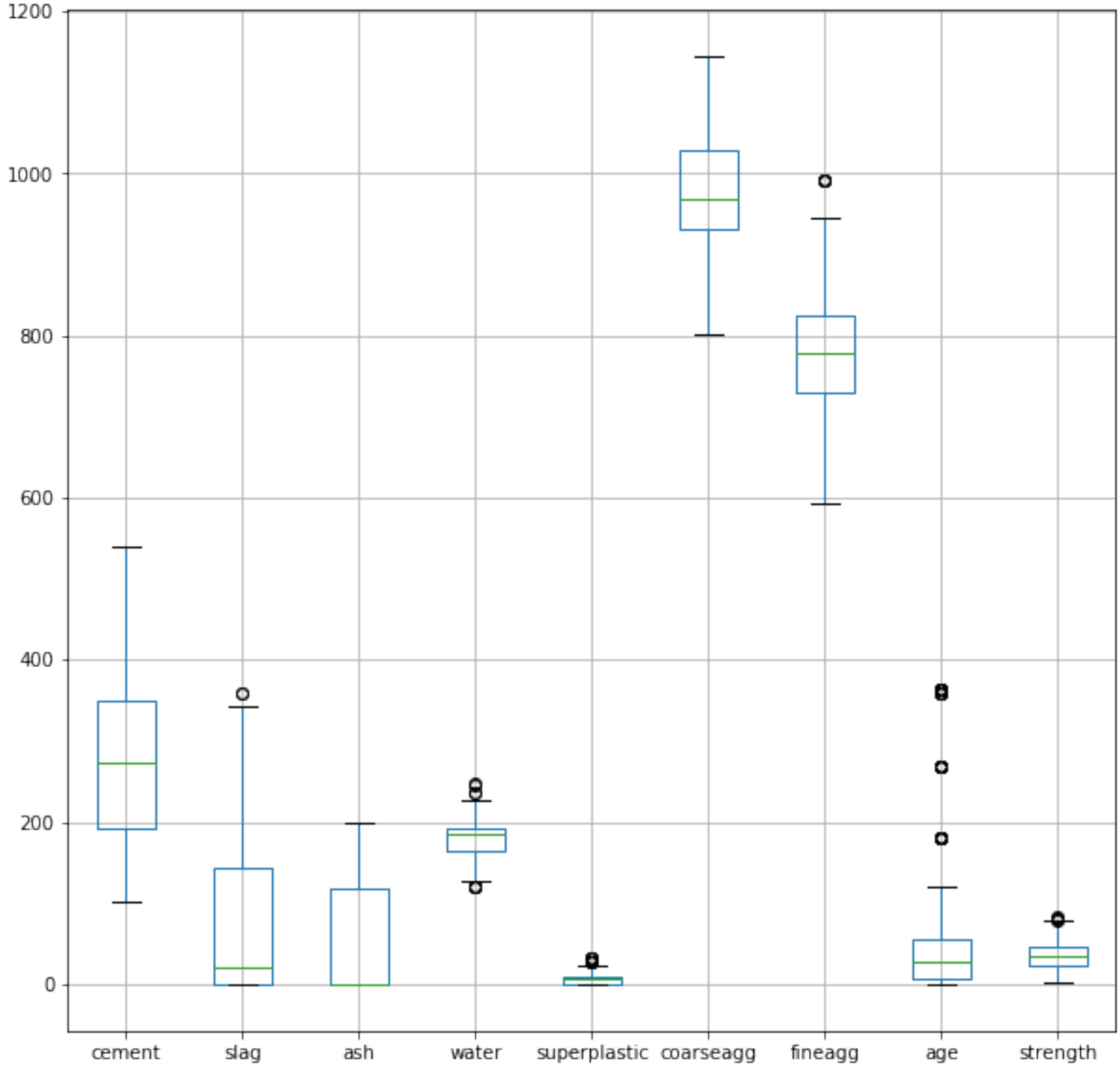
```
968.0      29
1125.0     24
           ..
909.7       1
925.3       1
845.0       1
868.6       1
923.2       1
Name: coarseagg, Length: 284, dtype: int64
594.0     30
755.8     30
670.0     23
613.0     22
801.0     16
          ..
792.5      1
762.9      1
674.8      1
658.0      1
762.2      1
Name: fineagg, Length: 302, dtype: int64
28     425
3      134
7      126
56      91
14      62
90      54
100     52
180     26
91      22
365     14
270     13
360      6
120      3
1        2
Name: age, dtype: int64
33.40    6
79.30    4
41.05    4
71.30    4
35.30    4
         ..
61.23    1
26.31    1
38.63    1
47.74    1
15.75    1
Name: strength, Length: 845, dtype: int64
```

In [8]:

```
#1.7 Comparative Boxplots confirming the precedent remarks on outliers, skewness
and Null values.

plt.figure(figsize=(10,10))
conc_df.boxplot()
```

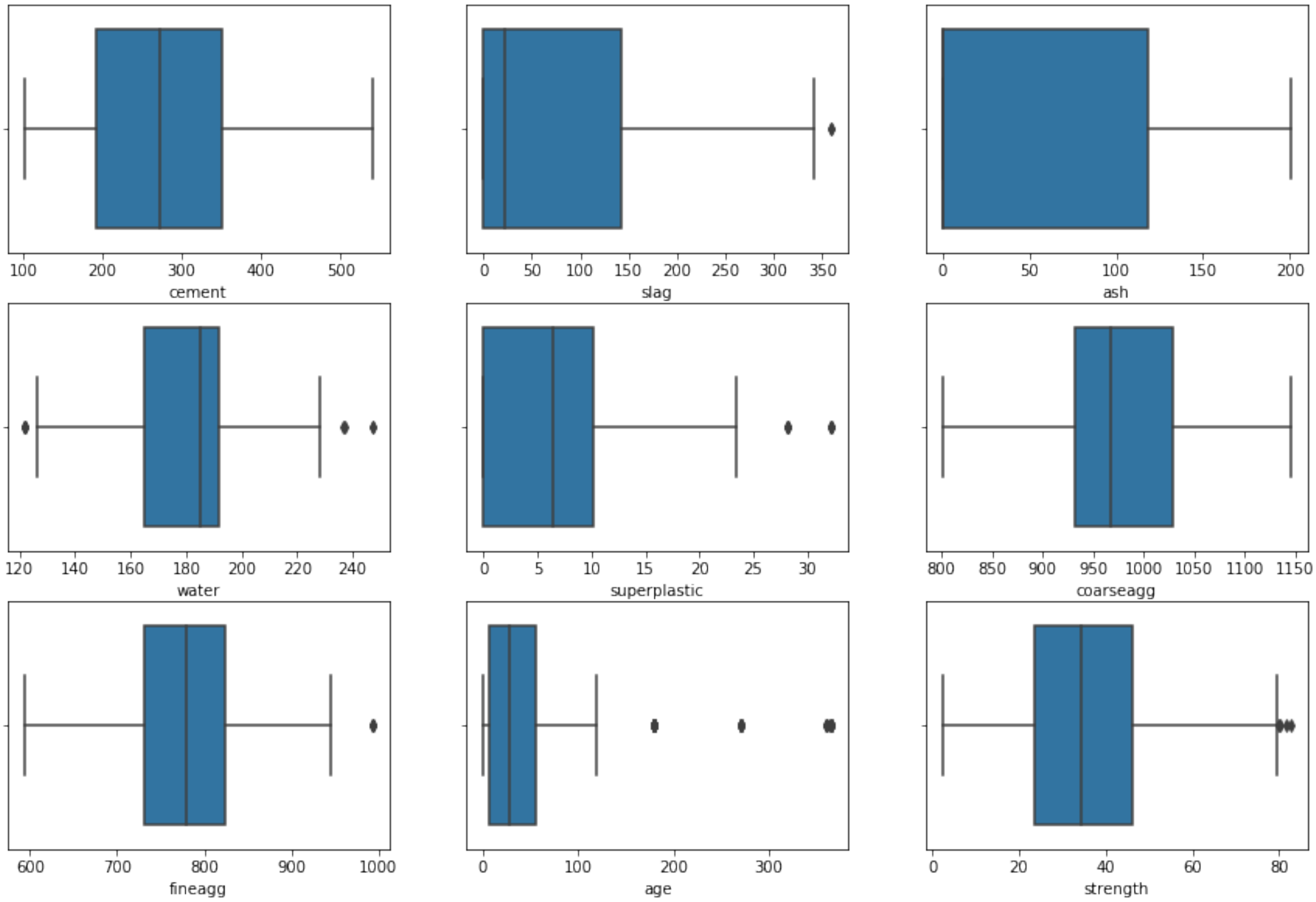
Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1015a26cd0>
```



In [9]:

```
#1.8 Fancier Boxplots to confirm the distribution shapes and peculiar outliers s
een in:
# - slag
# - superplastic
# - age


plt.figure(figsize=(15,10))
pos = 1
for i in conc_df.columns:
    plt.subplot(3, 3, pos)
    sns.boxplot(conc_df[i])
    pos += 1
```



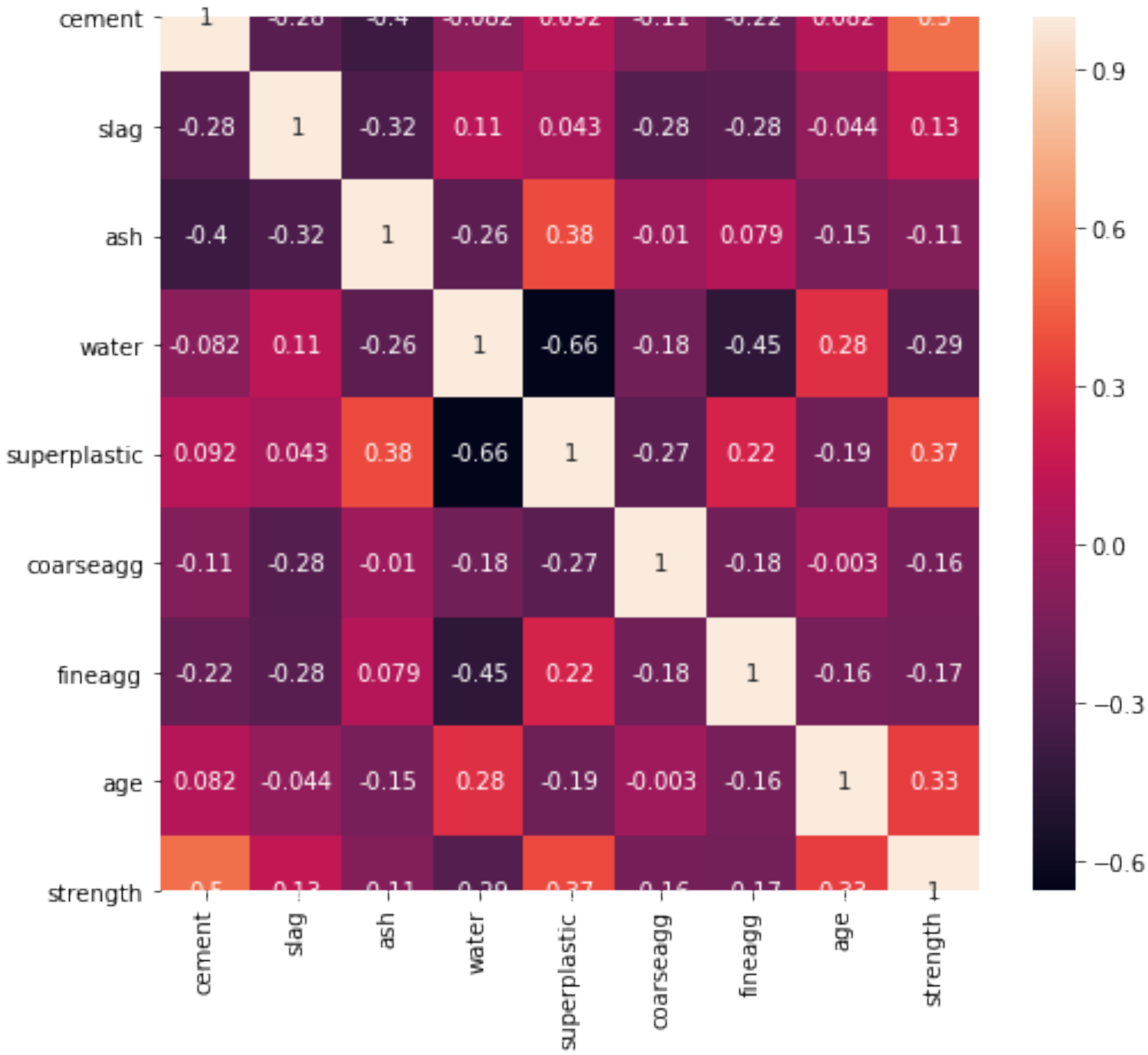
2- Bi-variate analysis

In [10]:

```
#2.1 Correlations betweeen variables and output

plt.figure(figsize=(8,7))
sns.heatmap(conc_df.corr(), annot = True)

# the are no strong correlations |r|>0.9 that would allow us to easily drop some proxy
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a16b3f050>
```

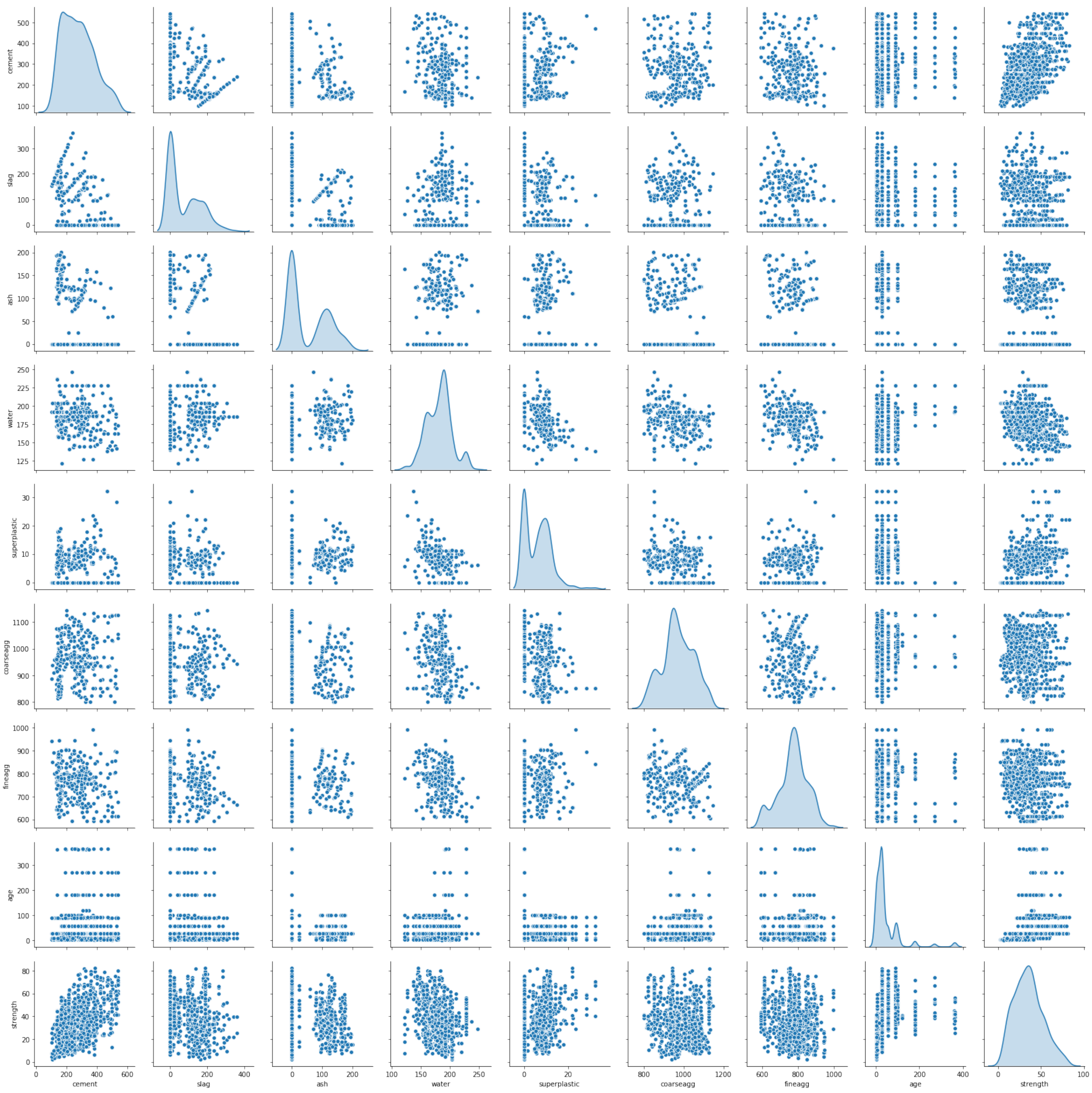


In [12]:

```
#2.2 pairplot that shows density curves , possible linear realtions

sns.pairplot(conc_df, diag_kind= 'kde')
```

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x1a193422d0>
```



In [ ]:

```
# COMMENTS : Zeros that bother

 # The Zeros in Slag and Ash and Superplasticizer: can concrete mixes be made wi
thout those ingredients?


 # Wihtout domain expertise,I would be tending to replace these nulls with their
repspective medians.
 # If my assumptions are wrong it might negatively impact the whole meaning of t
he models;
 # This will be investigated further in part 3.


 #   Output:Strength   seems to be mostly correlated with cement;
 #   and lesser with superplasticiser and age.
```

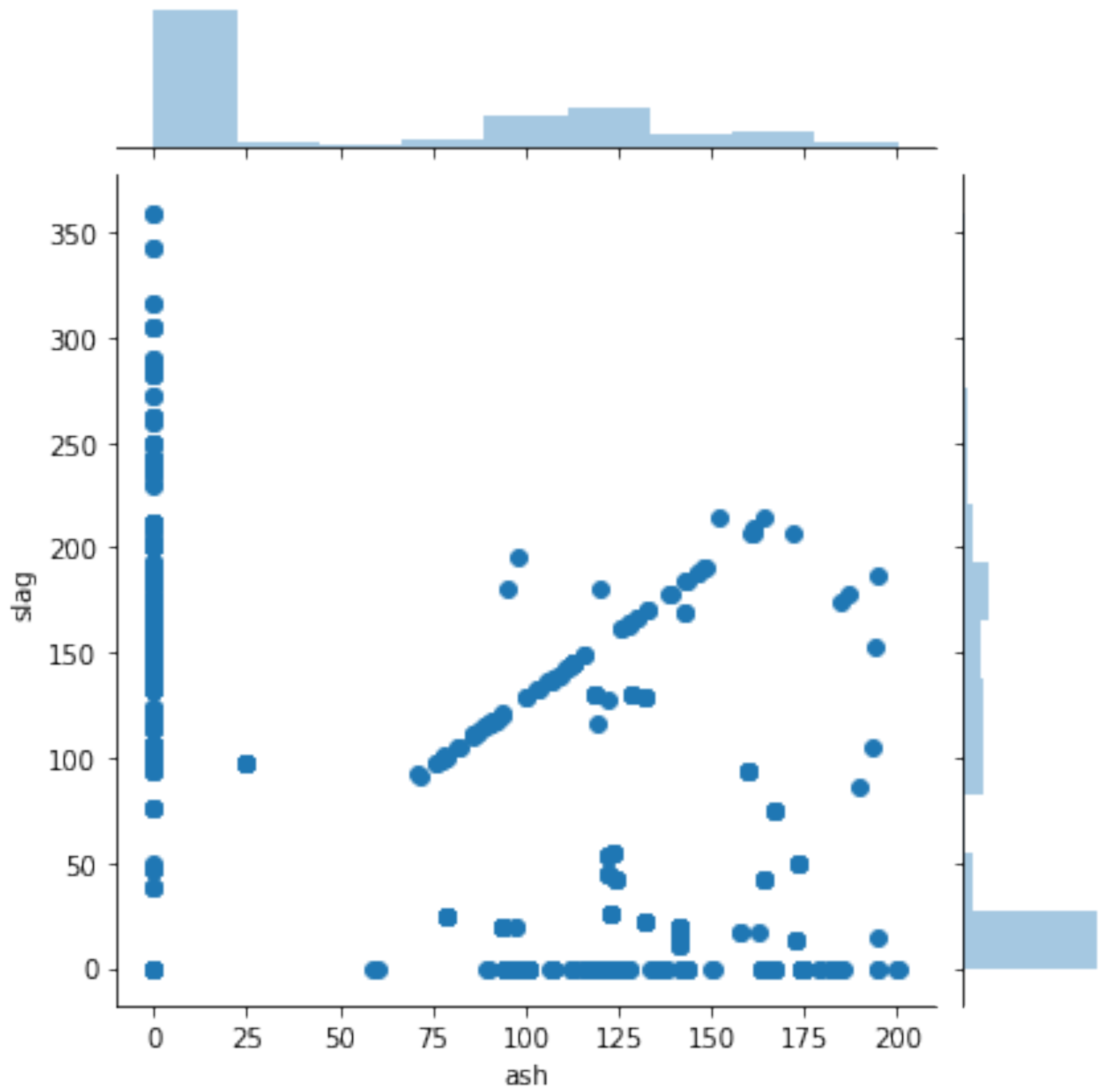
3- Data Challenges

In [13]:

```
#3.1 Investigating the Paranormal I: ASH and SLAG

sns.jointplot( conc_df.ash , conc_df.slag )

# Observation : I CANNOT cancel the zeros of those two respective columns !
# Reason : When one of them is 0 , the other seems to have some value.

# Insight:
# In concrete mixes , it seems common to have either one or the other quite ofte
n.

# The central linear line seems to show also a common proportionality practise:
#  while the mix is inclusive of both components as well.
```

Out[13]:

<seaborn.axisgrid.JointGrid at 0x1a1ccd6590>

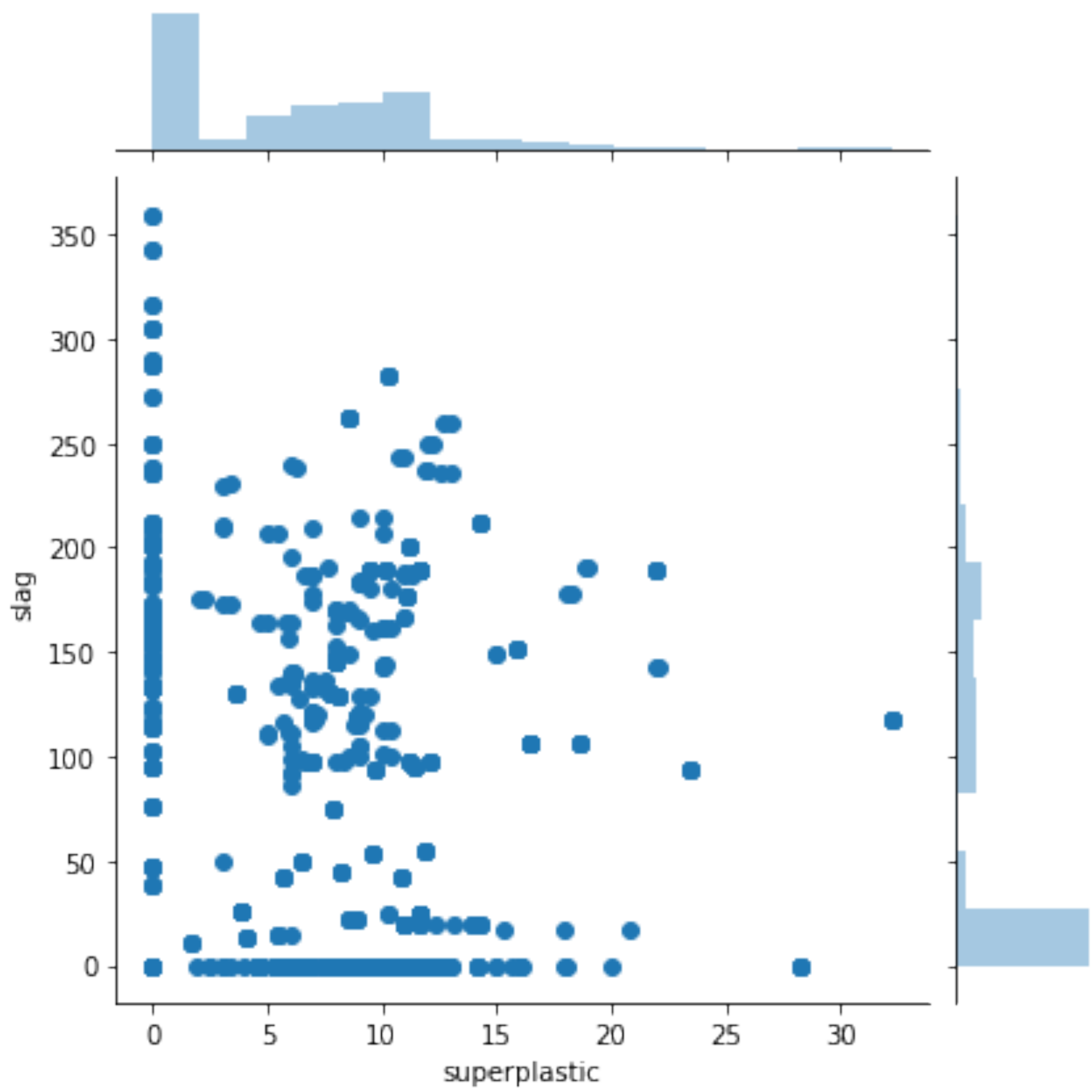


In [14]:

```
#3.2 Investigating the Paranormal II : Superplasticizer and SLAG :)

sns.jointplot( conc_df.superplastic , conc_df.slag )

# Insight: similar to above ,
# When one of them is 0 , the other seems to have some value.
# In concrete mixes , it seems common to have either one or the other quite ofte
n as well.
```

Out[14]:

```
<seaborn.axisgrid.JointGrid at 0x1a169d6150>
```

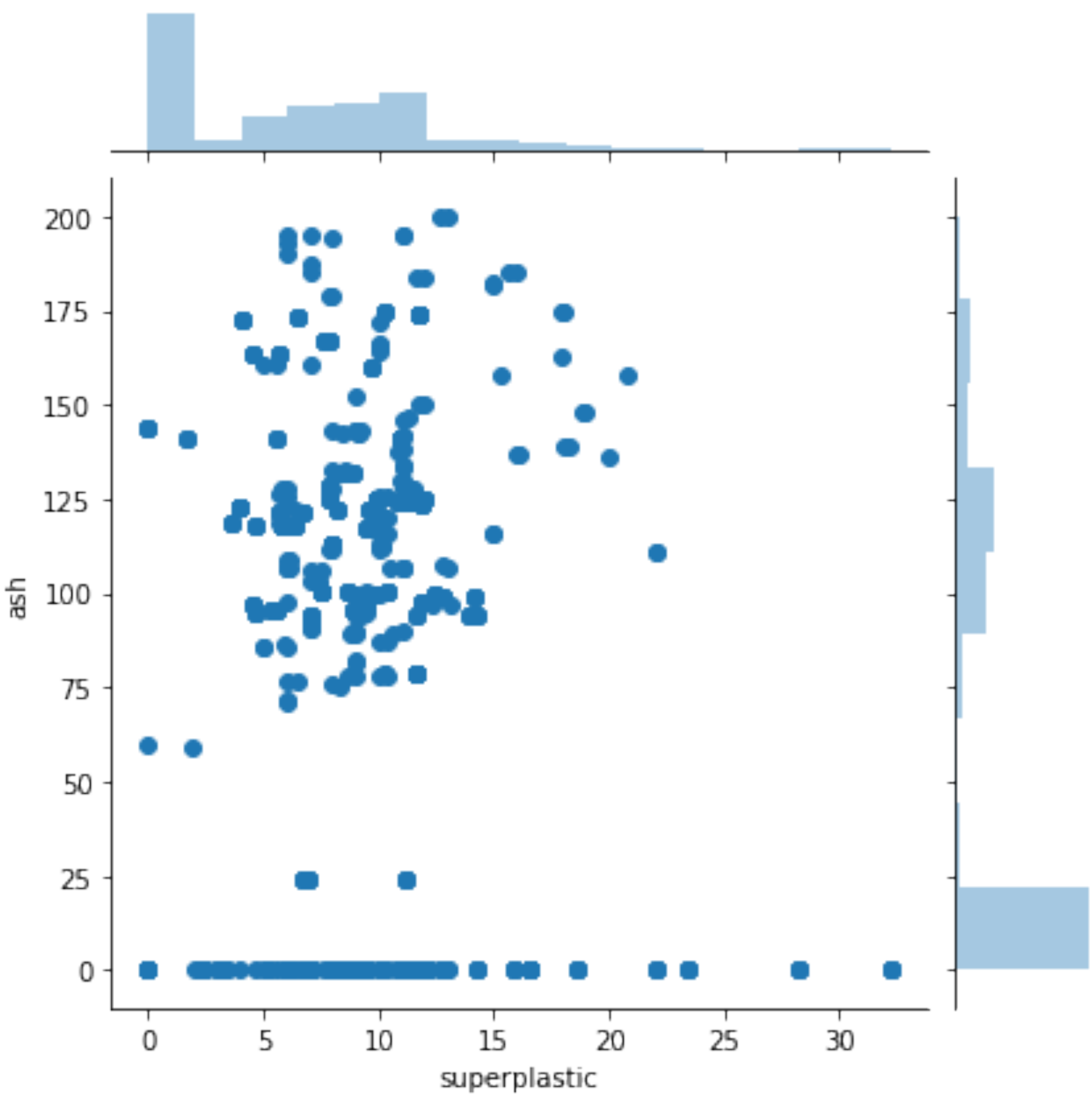


In [15]:

```
#3.2 Investigating the Paranormal III : Superplasticizer and ash
sns.jointplot( conc_df.superplastic , conc_df.ash )

# Insight: similar to above too ,
# When Ash is 0 , the Superplasticizer seems to have some value.
# In concrete mixes , it could be a mixing protocol followed based on endgoal pu
porses of the concrete.
```

Out[15]:

```
<seaborn.axisgrid.JointGrid at 0x1a1dff0c50>
```

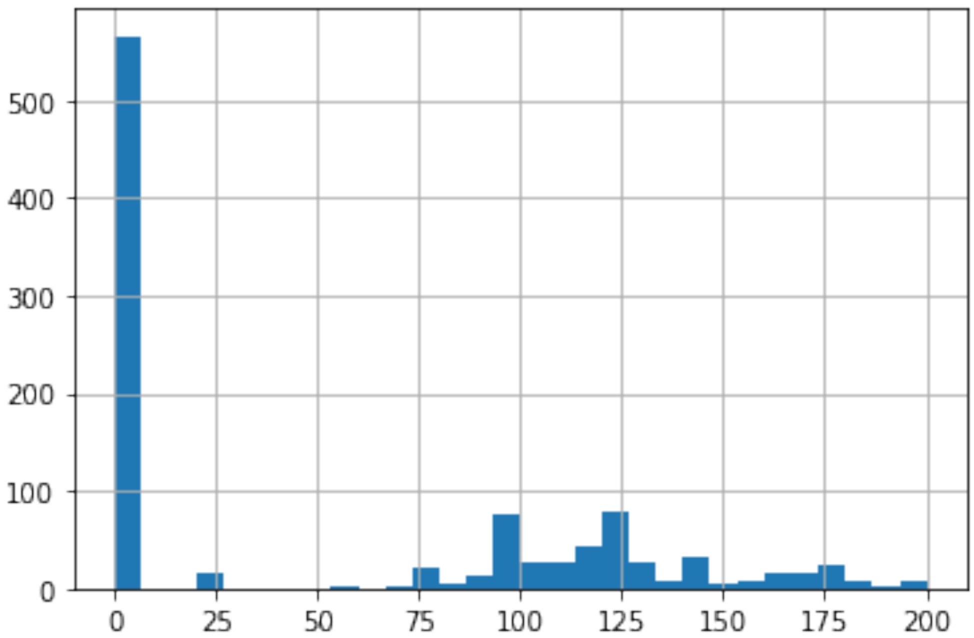


In [16]:

```
conc_df.ash.hist(bins =30)
```

Out[16]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1e213950>

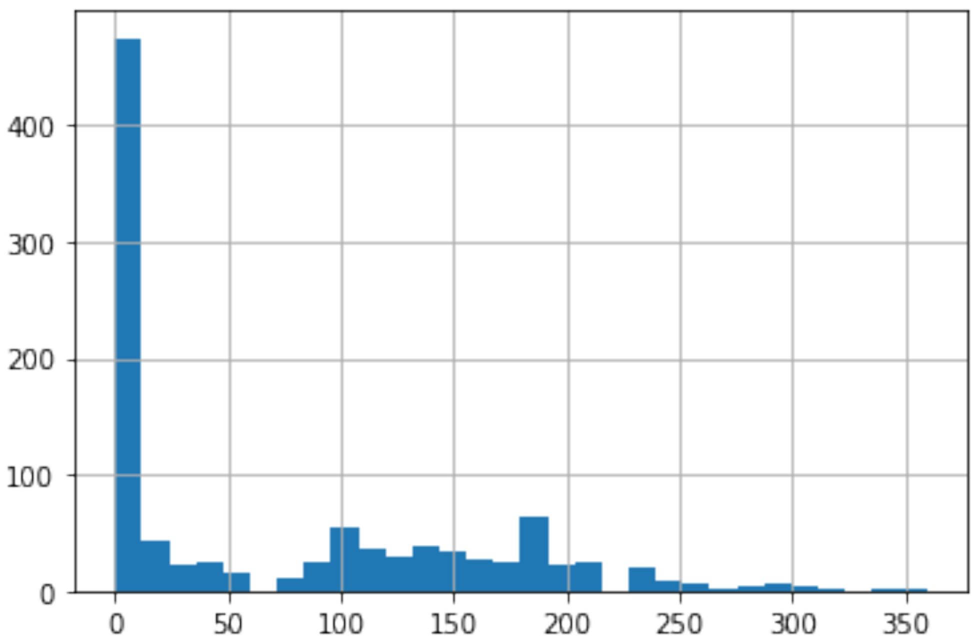


In [17]:

```
conc_df.slag.hist(bins =30)
```

Out[17]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1e441310>

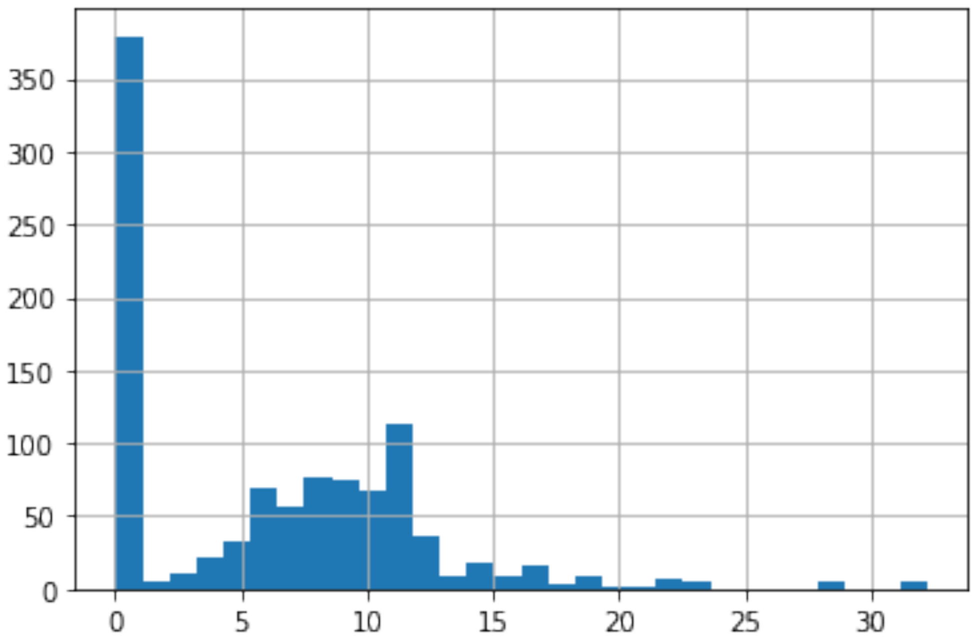


In [18]:

```
conc_df.superplastic.hist(bins =30)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e3d5410>
```



Decision:

While there are no missing values in the data set , The sheer amount of Zeros in the three attributes: Ash , Slag and Superplasticizer along with their relationship which seemed to be replacing one component by the other in the concrete mixes, is preventing me from considering all those 0s as data pollution or errors.

Hence I did not go for the option of replacing 0 by columns median or mean values : [conc_df.replace(0,conc_df.mean(axis=0),inplace=True)]

It could be a genuine consideration while prepeparing the concrete mixes: the existance of one component could replace the need for another in many cases.

4- Creating the prediction models , using simple train_test split , and comparing their scores :

- Linear regression
- Linear regression Ridge
- Linear regression Lasso
- Quadratic
- Quadratic Ridge
- Quadratic Lasso

In [70]:

```
#4.1 Libraries and modeling tools for this stage

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
```

In [60]:

```
#4.2 The Attributes
X= conc_df.drop('strength', axis= 1)
X.head()
```

Out[60]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
0	141.3	212.000000	54.18835	203.5	6.20466	971.8	748.5	28
1	168.9	42.200000	124.30000	158.3	10.80000	1080.8	796.2	14
2	250.0	73.895825	95.70000	187.4	5.50000	956.9	861.2	28
3	266.0	114.000000	54.18835	228.0	6.20466	932.0	670.0	28
4	154.8	183.400000	54.18835	193.3	9.10000	1047.4	696.7	28

In [61]:

```
#4.3 The Outputs
y= conc_df.strength
y.head()
```

Out[61]:

```
0     29.89
1     23.51
2     29.22
3     45.85
4     18.29
Name: strength, dtype: float64
```

In [38]:

```
#4.4 Data split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random
_state=1)
```

In [63]:

```
#4.5 Evaluating Simple Linear regression model

linear = LinearRegression()
linear.fit(X_train, y_train)

print(linear.intercept_)
print(linear.coef_)

print('\nThe R2 scores for the Linear model on training and testing sets respect
ively: ')
print(linear.score(X_train, y_train))
print(linear.score(X_test, y_test))
```

```
178.62224848293928
[ 0.06905796  0.05310178  0.01420686 -0.45896376 -0.34394531 -0.0540
8485
 -0.04583568  0.1112763 ]

The R2 scores for the Linear model on training and testing sets resp
ectively:
0.5773732703468136
0.6222031540277765
```

In [68]:

```
#4.6 Evaluating Ridge regression model

ridge = Ridge(alpha=10)
ridge.fit(X_train,y_train)

print(ridge.intercept_)
print(ridge.coef_)

print('\nThe R2 scores for the Linear_ridge model on training and testing sets r
espectively: ')
print(ridge.score(X_train, y_train))
print(ridge.score(X_test, y_test))
```

```
178.56477747776185
[ 0.06906238  0.05310878  0.01422324 -0.45884548 -0.34336436 -0.0540
6679
 -0.04582238  0.11127248]

The R2 scores for the Linear_ridge model on training and testing set
s respectively:
0.5773732590904472
0.6221938444007582
```

In [69]:

```
#4.6 Evaluating Lasso regression model

lasso = Lasso(alpha=10)
lasso.fit(X_train,y_train)

print(lasso.intercept_)
print(lasso.coef_)

print('\nThe R2 scores for the Linear_lasso model on training and testing sets r
espectively: ')
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))
```

```
142.96896890363465
[ 0.0679982   0.04927016  0.01036267 -0.37122052 -0.         -0.0423
4865
 -0.03708336  0.10406494]

The R2 scores for the Linear_lasso model on training and testing set
s respectively:
0.5713653399082463
0.6058746006214266
```

In [83]:

```
#4.7 Evaluating Quadratic regression models : simple , Ridge and Lasso


poly = PolynomialFeatures(degree = 2, interaction_only=True)
X_poly = poly.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20, r
andom_state=1)


# Quadratic_linear
linear.fit(X_train, y_train)
print(linear.intercept_)
print(linear.coef_)

print('\nThe R2 scores for the Quadratic model on training and testing sets resp
ectively: \n')
print(linear.score(X_train, y_train))
print(linear.score(X_test, y_test))



# Quadratic_Ridge
ridge = Ridge(alpha=10)
ridge.fit(X_train,y_train)

print(ridge.intercept_)
print(ridge.coef_)

print('\nThe R2 scores for the Qyadratic_ridge model on training and testing set
s respectively: \n')
print(ridge.score(X_train, y_train))
print(ridge.score(X_test, y_test))



# Quadratic_Lasso
lasso = Lasso(alpha=10)
lasso.fit(X_train,y_train)

print(lasso.intercept_)
print(lasso.coef_)

print('\nThe R2 scores for the Quadratic_lasso model on training and testing set
s respectively: \n')
print(lasso.score(X_train, y_train))
print(lasso.score(X_test, y_test))
```

```
-182.67153751701986
[-8.81746363e-11  3.11880816e-01  4.33763885e-01  3.07571341e-02
  1.94367172e+00  9.65675210e+00 -1.43433011e-02 -5.10102676e-02
  1.45037434e+00  3.39767140e-04  7.32163900e-04 -1.90165021e-03
 -7.61701553e-03  3.00945967e-05  6.71574575e-05 -1.55930070e-04
  4.99517426e-04 -2.67213644e-03 -1.15000377e-02 -2.12382894e-04
  3.59099839e-04  2.02186178e-04 -2.39645790e-03 -2.32271695e-02
 -6.58506970e-05  5.45172286e-04  1.34332928e-03 -1.34866993e-03
 -4.57443655e-04 -9.59712062e-04 -3.84688729e-03 -9.74910828e-04
 -4.16264650e-03  4.46432686e-03  1.14428904e-04 -4.00021971e-04
 -3.49142725e-04]

The R2 scores for the Quadratic model on training and testing sets r
espectively:

0.7147839447619093
0.7444650073333847
-47.48116252441036
[ 0.00000000e+00  2.57396543e-01  3.63365272e-01 -1.64306032e-01
  1.60469697e+00  1.40895522e+00 -7.72499933e-02 -1.00462455e-01
  1.40103464e+00  3.11482904e-04  7.35743753e-04 -1.77837405e-03
 -4.73246507e-03  4.30335341e-05  6.47009528e-05 -1.53198775e-04
  5.06061468e-04 -2.46245437e-03 -6.67739509e-03 -2.05518234e-04
  3.46316710e-04  2.08551851e-04 -1.96469292e-03 -1.64257095e-02
 -1.49555384e-05  5.56073654e-04  1.36798966e-03  6.19375098e-03
 -3.29920222e-04 -8.86672873e-04 -3.75519058e-03  1.75832689e-03
 -1.13883044e-03  4.79593150e-03  1.25685137e-04 -3.83599623e-04
 -3.35859013e-04]

The R2 scores for the Qyadratic_ridge model on training and testing
sets respectively:

0.7142436901963847
0.7450396826541592
108.31015258965846
[ 0.00000000e+00  0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00 -0.00000000e+00 -0.00000000e+00 -0.00000000e+00
  0.00000000e+00  4.12181578e-04  7.03858986e-04 -5.76803387e-04
 -3.28752029e-03  9.60460259e-06  1.17213487e-04  5.31182907e-05
  4.16716126e-04 -6.83695180e-04 -4.05188339e-03 -2.48469022e-04
  3.81688895e-04  4.38804514e-04 -5.58169739e-04 -1.38622923e-02
 -3.01454464e-04  3.09084451e-04  2.32837253e-03  2.00216079e-03
  1.95084075e-04 -3.96755586e-04 -9.60850502e-04  2.83721178e-03
 -1.30693192e-03  1.24278860e-02 -8.44976370e-05 -1.64699491e-05
  5.36033446e-05]

The R2 scores for the Quadratic_lasso model on training and testing
sets respectively:

0.6960353872245841
0.7297955292409917
```

```
/Users/eddie/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/coordinate_descent.py:475: ConvergenceWarning: Objective did
not converge. You might want to increase the number of iterations. D
uality gap: 25022.989656111677, tolerance: 22.640607526492726
  positive)
```

The best Model seems so far the Quadratic Lasso considering the balance of performance and number of attributes needed.

The R2 results do not seem to exceed the 73-74% , and this is normal given that null values were not dropped or replaced in : Ash , Slag and Super plasticize attributes as explained earlier.

5- Squeezing performance and K-fold CV on Two models

In [118]:

```
#5.1 First: Standardize all values

X = conc_df_scaled.iloc[:,0:8]
Y = conc_df_scaled.iloc[:,8]
```

In [119]:

```
# 5.2 Libraries needed at this stage

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [120]:

```
# 5.3 Trying Kfold cross-validation on the first model : the Linear_regression

kfold = KFold(n_splits=10, random_state=1)
results = cross_val_score(LinearRegression(), X, Y, cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
[0.4172524  0.65718137 0.66802796 0.56228764 0.56841806 0.62404722
 0.57880224 0.5774628  0.43528243 0.52315551]
Accuracy: 56.119% (7.951%)
```

In [117]:

```
# 5.4 Trying Kfold cross-validation on the second model : the Quadratic_regressi
on

poly = PolynomialFeatures(degree = 2, interaction_only=True)
X_poly = poly.fit_transform(X)

kfold = KFold(n_splits=10, random_state=1)
results = cross_val_score(Ridge(), X_poly, Y, cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
[0.56503249 0.66585456 0.77990066 0.65742288 0.66769065 0.7392189
 0.71379454 0.73220122 0.62172293 0.63836202]
Accuracy: 67.812% (6.044%)
```

Conclusion :

The cross validation seems to have provided a better picture of the True performance of the regression models. An Average Predictive power. The Quadratic model proved to perform better that the linear, with an average R2 score of 67%.

Last thought : all those previous performances would have spiked much higher were we to drop the null values mentionned in Step 3, however that may not be relevant as concrete mixtures may very well contain exclusively one or another of the components mentionned and eliminating so many zeros (hundreds) or substituting them by cental values will dilute a lot of the true meaning of the data.

Edouard Toutounji - March 13 , 2020.

In [ ]: