Final Project

Full name: Kahla Archibald

R#: 11706661

HEX: 0xb2a125

Final Project

Full name: Audrey Brown

R#: 11669186

HEX: 0xb20ec2

Final Project

Full name: Mishael Martinez

R#: 11714529

HEX: 0xb2bfe1

Final Project

Full name: Caelan Neece

R#: 11666614

HEX: 0xb204b6

Final Project

Literature research:

Describe the challenge of concrete mixture design and the importance of compressive strength.

It is challenging to mix concrete because it needs to have a good enough strength that will allow the concrete to hold up in quality over time (3). Concrete contains a mixture of water, aggregates, and cement (8). There can be regular concrete and also high-performance concrete (3). To have high-performance concrete, you must have "high strength, high durability, and high workability (3)." The workability of concrete is essential to designing a mix. It will be difficult to compact and place the concrete if it has inadequate workability, implying it is unlikely for the concrete to have good strength (8). To determine if the utility of a concrete mix, tests need to be run. Using tests will tell the concrete designer that the workability will be good (8). Some of the tests include the slump test, the flow-table test, and the compacting-factor test (8). The most used test, however, is the slump test (8). High-performance concrete will have a slump of at least 100mm (3). High-performance concrete usually has a strength greater than 60 MPa (2). Concrete also has yield stress, which means different flow values need testing to see how the concrete forms when poured at different speeds (8). One of the main challenges of concrete mixture design is meeting the demand while still maintaining a low cost (2).

Compressive strength is one of the most valuable criteria when mixing concrete (5). Through multiple tests, how well the concrete maintains its original shape can tell us the compressive strength is (5). Knowing the compressive strength of concrete is fundamental for companies to ensure quality (1). Without strong structures, if there were to be an accident, the company responsible would bear the consequences for any damages (7). The strength is what determines if the concrete will be accepted or rejected. Concrete must be at least 2500 psi (pounds per square inch) to be safe enough to build structures (7). Depending on the type of concrete, different psi strengths are necessary (5). For example, temperature affects how strong concrete should be. The colder the climate is, the stronger concrete needs to be to maintain the structure (5).

stronger concrete needs to be to maintain the structure (5).

Summarize the value of a data model in the context of the conventional approach to strength prediction

The conventional approach to test the compressive strength of concrete requires a 28-day test (4). It can become very time consuming if multiple tests need to be run (4). To make this process simpler, a company could use a machine that tests the compressive strength based on chosen predictors (4). Information will need to be feed to the computer regarding the different combinations of the predictors, and it will produce information about the compressive strength of these combinations (4). It will save companies time and reduce potential errors (4). Data models allow users to put different prediction lines onto underlying data. The prediction line shows the performance, and the best prediction line is chosen (4). The data model with the best prediction line tells us the assessment of concrete strength (6). Data modeling replaces the conventional 28-day test, which reduces a lot of cost and time. It is valuable since companies want a quicker and cheaper alternative to 28-day testing.

Cited:

1. Jamal, Haseeb. “Haseeb Jamal.” Compressive Strength of Concrete | Definition, Importance, Applications, 29 Jan. 2017, www.aboutcivil.org/compressive-strength-of-concrete.html.
2. Larrard, Francois de, and Thierry Sedran. Mixture - Proportioning of High-Performance Concrete, Nov. 2002, 54.243.252.9/engr-1330-psuedo-course/CECE-1330-PsuedoCourse/6-Projects/P-ConcreteStrength/Mixture-ProportioningCCR-deLarrardSedran-full.pdf.
3. Laskar, Aminul. Mix Design of High-Performance Concrete, 28 July 2008, 54.243.252.9/engr-1330-psuedo-course/CECE-1330-PsuedoCourse/6-Projects/P-ConcreteStrength/Mix_Design_of_High-performance_Concrete.pdf.
4. Modukuru, Pranay. “Concrete Compressive Strength Prediction Using Machine Learning.” Medium, Towards Data Science, 9 Mar. 2020, towardsdatascience.com/concrete-compressive-strength-prediction-using-machine-learning-4a531b3c43f3.
5. Uknown, Uknown. “Everything You Need to Know About Concrete Strength: Cor-Tuf.” Cor, 24 June 2020, cor-tuf.com/everything-you-need-to-know-about-concrete-strength/.
6. Uknown, Uknown. “Read ‘Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification’ at NAP.edu.” National Academies Press: OpenBook, 0AD, www.nap.edu/read/13395/chapter/7.
7. Uknown, Uknown. The Strength of Concrete, 2015, shop.iccsafe.org/media/wysiwyg/material/9090S15-Sample.pdf.
8. Valenzuela, Federico, and Victor C. Pandolfelli. The Application of Rheological Concepts on the Evaluation of High-Performance Concrete Workability, Jan. 2008, 54.243.252.9/engr-1330-psuedo-course/CECE-1330-PsuedoCourse/6-Projects/P-ConcreteStrength/ArtigoHPC026_CASTROetalfinal.pdf.

In [1]:

```
import pandas as pd
import numpy as np
import statistics
import scipy.stats
from matplotlib import pyplot as plt
import statsmodels.formula.api as smf
import sklearn.metrics as metrics
from scipy.stats import pearsonr
import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
from statsmodels.stats.outliers_influence import summary_table
from __future__ import print_function
import numpy as np
from scipy import stats
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from statsmodels.iolib.table import (SimpleTable, default_txt_fmt)
```

Analysis and implementation for the data model

In [2]:

```
file = 'concreteData.xls'
df = pd.read_excel(file)
df
```

Out[2]:

	Cement (component	Blast Furnace Slag	Fly Ash (component	Water (component	Superplasticizer	Coarse Aggregate	Fine Aggregate	Concrete

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Fine Aggregate (component 7)(kg in a m^3 mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075
...	...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.284354
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.178794
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.696601
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.768036
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.401235

1030 rows × 9 columns

- The data frame above is showing us our file that we are going to use and its contents. As we can see at the very bottom, there are 1030 rows x 9 columns, and in each of the rows and columns, there are numbers. These numbers relate to each column, with the numbers under cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate telling us the amount of each. The age column tells us how old the mixtures are. The very last column, concrete compressive strength, tells us the durability of each combination.

In [3]:

```
df.describe()
```

Out[3]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5) (kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Fine Aggregate (component 7)(kg in a m^3 mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.165631	73.895485	54.187136	181.566359	6.203112	972.918592	773.578883	45.662136	35.817836
std	104.507142	86.279104	63.996469	21.355567	5.973492	77.753818	80.175427	63.169912	16.705679
min	102.000000	0.000000	0.000000	121.750000	0.000000	801.000000	594.000000	1.000000	2.331808
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.707115
50%	272.900000	22.000000	0.000000	185.000000	6.350000	968.000000	779.510000	28.000000	34.442774
75%	350.000000	142.950000	118.270000	192.000000	10.160000	1029.400000	824.000000	56.000000	46.136287
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.599225

- There are 1030 total data points in each category. The blast furnace slag has a mean of 73.9, a max of 359.4, and a min of 0. The average falls closer to the minimum, so this implies that the data is more likely to be small. At least half of the Fly Ash data is 0. It's mean value is quite a bit smaller than the max value because of this. The max is 200, but the mean is only 54.2. For fly ash it can also be expected for the data to fall closer to 0. The course aggregate has the highest data values. The max and the mean are larger than all the other predictors. The water has a mean of 181.6, a max of 247, and a min of 121.8. The median is also 185. Based on this range it implies a normal distribution. The superplasticizer has a mean of 6.2, a max of 32.2, and a min of 0. The coarse aggregate has a mean of 972.9, a max of 1145, and a min of 801. The values on coarse aggregate are pretty high. The mean is fairly close to the minimum value which implies the values lie closer to the lower values. The mean of the fine aggregate is 773.6. That is fairly close to the median value implying that the values are fairly evenly distributed. The age predictor has a mean of 45.7, the max is 365 and the min is 1. There are more data values that are closer to the min value because the median value is 28. There are a lot more smaller values than larger values. The numbers on most of the predictors lay closer to the minimum values.

In [4]:

```
df.rename(columns = {'Cement (component 1)(kg in a m^3 mixture)':'c', 'Blast Furnace Slag
(component 2)(kg in a m^3 mixture)':'bfs','Fly Ash (component 3)(kg in a m^3 mixture)':'fa','Water
(component 4)(kg in a m^3 mixture)':'w',
                       'Superplasticizer (component 5)(kg in a m^3 mixture)':'s','Coarse Aggregate
(component 6)(kg in a m^3 mixture)':'ca','Fine Aggregate (component 7)(kg in a m^3 mixture)':'fia'
,'Age (day)':'a',
                              'Concrete compressive strength(MPa, megapascals) ':'ccs'}, inplace =
rue)
df.head()
```

Out[4]:

	c	bfs	fa	w	s	ca	fia	a	ccs
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075

- In order to make the coding easier later on in the project, we changed the column names to be just a few letters.

In [5]:

```
df.corr(method ='pearson')
```

Out[5]:

	c	bfs	fa	w	s	ca	fia	a	ccs
c	1.000000	-0.275193	-0.397475	-0.081544	0.092771	-0.109356	-0.222720	0.081947	0.497833
bfs	-0.275193	1.000000	-0.323569	0.107286	0.043376	-0.283998	-0.281593	-0.044246	0.134824
fa	-0.397475	-0.323569	1.000000	-0.257044	0.377340	-0.009977	0.079076	-0.154370	-0.105753
w	-0.081544	0.107286	-0.257044	1.000000	-0.657464	-0.182312	-0.450635	0.277604	-0.289613
s	0.092771	0.043376	0.377340	-0.657464	1.000000	-0.266303	0.222501	-0.192717	0.366102
ca	-0.109356	-0.283998	-0.009977	-0.182312	-0.266303	1.000000	-0.178506	-0.003016	-0.164928
fia	-0.222720	-0.281593	0.079076	-0.450635	0.222501	-0.178506	1.000000	-0.156094	-0.167249
a	0.081947	-0.044246	-0.154370	0.277604	-0.192717	-0.003016	-0.156094	1.000000	0.328877
ccs	0.497833	0.134824	-0.105753	-0.289613	0.366102	-0.164928	-0.167249	0.328877	1.000000

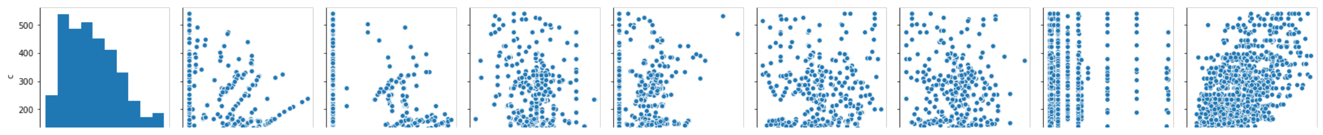
- Pearson's correlation method can give us a basic overview on how good the predictors are. For this method an indication of good correlation is being close to 1 or -1. For this data we are trying to determine how good of a fit different predictors are in relation to concrete compressive strength. The last column shows the correlation coefficient for all the predictors in relation to concrete compressive strength. The best predictors are, cement (0.49), Superplasticizer (0.37), and age (0.33). These are the values that are the closest to 1. This indicates a strong positive correlation. There are a few negative correlations as well, the strongest of those being water (-0.29). This means that as the amount of water increases the concrete compressive strength decreases. This gives an idea of how the predictors will perform.

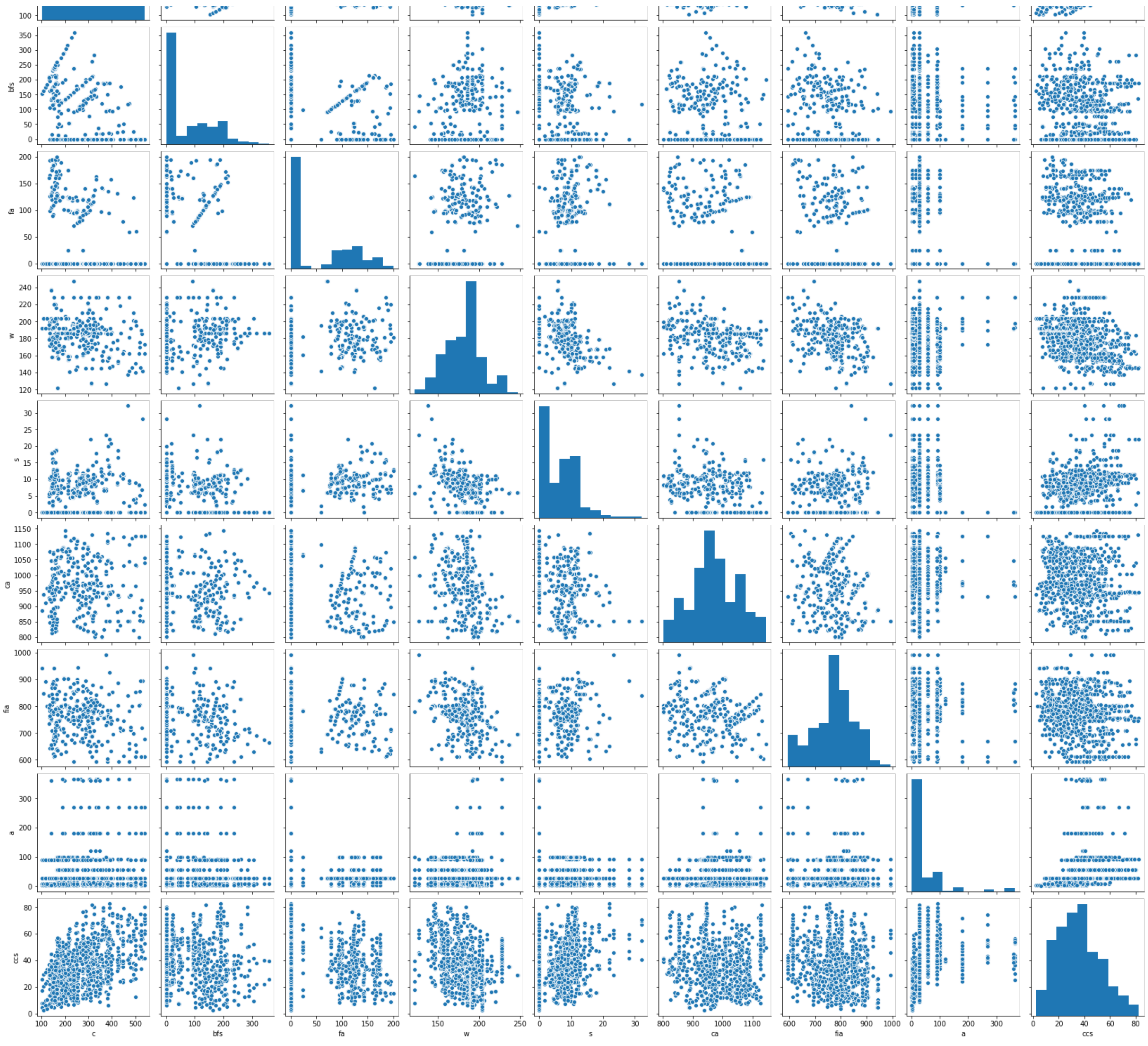
In [6]:

```
import seaborn as sns
sns.pairplot(df)
```

Out[6]:

```
<seaborn.axisgrid.PairGrid at 0x2a6f7070fa0>
```





- This seaborn pair plot gives us the same information as the Pearson's correlation coefficient, but as a visual representation. The last column of the graphs shows us the correlation of the predictors to the compressive strength. In order to get a better idea of the correlation, we will run more exploratory tests. Next we will run ordinary least squares on each predictor to see what it tells us about the data. Later, we will create linear regression lines through the scatter plots to gain even more information.

In [7]:

```
c = np.array(df['c'])
bfs = np.array(df['bfs'])
fa = np.array(df['fa'])
w = np.array(df['w'])
s = np.array(df['s'])
ca = np.array(df['ca'])
fia = np.array(df['fia'])
a = np.array(df['a'])
ccs = np.array(df['ccs'])
```

In [8]:

```
x = np.array(c)
Y = np.array(ccs)

import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std    #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.248
Model:                            OLS   Adj. R-squared:                  0.247
Method:                 Least Squares   F-statistic:                     338.7
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           1.32e-65
Time:                        17:01:19   Log-Likelihood:                -4214.6
No. Observations:                1030   AIC:                             8433.
Df Residuals:                    1028   BIC:                             8443.
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         13.4428      1.297     10.365      0.000      10.898      15.988
x1             0.0796      0.004     18.405      0.000       0.071       0.088
==============================================================================
Omnibus:                       19.695   Durbin-Watson:                   1.012
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               17.890
Skew:                           0.271   Prob(JB):                     0.000130
Kurtosis:                       2.649   Cond. No.                         861.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[13.44279487  0.07957957]
```

In [9]:

```
x = np.array(bfs)
Y = np.array(ccs)

import statsmodels.api as sm     #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction 
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.018
Model:                            OLS   Adj. R-squared:                  0.017
Method:                 Least Squares   F-statistic:                     19.03
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           1.41e-05
Time:                        17:01:19   Log-Likelihood:                -4351.8
No. Observations:                1030   AIC:                             8708.
Df Residuals:                    1028   BIC:                             8717.
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         33.8888      0.680     49.869      0.000      32.555      35.222
x1             0.0261      0.006      4.363      0.000       0.014       0.038
==============================================================================
Omnibus:                       30.578   Durbin-Watson:                   0.860
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               28.557
Skew:                           0.359   Prob(JB):                     6.29e-07
Kurtosis:                       2.613   Cond. No.                         150.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3.38887788e+01 2.61052078e-02]
```

In [10]:

```
x = np.array(fa)
Y = np.array(ccs)
```

```
import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.011
Model:                            OLS   Adj. R-squared:                  0.010
Method:                 Least Squares   F-statistic:                     11.63
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           0.000675
Time:                        17:01:19   Log-Likelihood:                -4355.4
No. Observations:                1030   AIC:                             8715.
Df Residuals:                    1028   BIC:                             8725.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         37.3137      0.679     54.978      0.000      35.982      38.646
x1            -0.0276      0.008     -3.410      0.001      -0.043      -0.012
==============================================================================
Omnibus:                       29.014   Durbin-Watson:                   0.848
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               27.216
Skew:                           0.351   Prob(JB):                     1.23e-06
Kurtosis:                       2.624   Cond. No.                         110.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[ 3.73137216e+01 -2.76059213e-02]
```

In [11]:

```
x = np.array(w)
Y = np.array(ccs)

import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.084
Model:                            OLS   Adj. R-squared:                  0.083
Method:                 Least Squares   F-statistic:                     94.12
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           2.37e-21
Time:                        17:01:19   Log-Likelihood:                -4316.1
No. Observations:                1030   AIC:                             8636.
Df Residuals:                    1028   BIC:                             8646.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         76.9524      4.269     18.025      0.000      68.575      85.330
x1            -0.2266      0.023     -9.701      0.000      -0.272      -0.181
==============================================================================
Omnibus:                       33.418   Durbin-Watson:                   0.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               24.860
Skew:                           0.281   Prob(JB):                     4.00e-06
Kurtosis:                       2.488   Cond. No.                     1.57e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.57e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
strong multicollinearity or other numerical problems.
[76.95242646 -0.22655403]
```

In [12]:

```
x = np.array(s)
Y = np.array(ccs)

import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.134
Model:                            OLS   Adj. R-squared:                  0.133
Method:                 Least Squares   F-statistic:                     159.1
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           5.08e-34
Time:                        17:01:19   Log-Likelihood:                -4287.1
No. Observations:                1030   AIC:                             8578.
Df Residuals:                    1028   BIC:                             8588.
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         29.4668      0.699     42.165      0.000      28.095      30.838
x1             1.0239      0.081     12.614      0.000       0.865       1.183
==============================================================================
Omnibus:                       24.083   Durbin-Watson:                   1.052
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               24.387
Skew:                           0.353   Prob(JB):                     5.06e-06
Kurtosis:                       2.738   Cond. No.                         12.5
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[29.46675078  1.0238547 ]
```

In [13]:

```
x = np.array(ca)
Y = np.array(ccs)

import statsmodels.api as sm      #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                      y   R-squared:                       0.027
Model:                            OLS   Adj. R-squared:                  0.026
Method:                 Least Squares   F-statistic:                     28.74
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           1.02e-07
Time:                        17:01:19   Log-Likelihood:                -4347.0
No. Observations:                1030   AIC:                             8698.
Df Residuals:                    1028   BIC:                             8708.
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         70.2935      6.451     10.897      0.000      57.635      82.952
x1            -0.0354      0.007     -5.361      0.000      -0.048      -0.022
==============================================================================
Omnibus:                       32.679   Durbin-Watson:                   0.883
```

```
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               34.561
Skew:                           0.434   Prob(JB):                     3.13e-08
Kurtosis:                       2.772   Cond. No.                     1.23e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.23e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
[ 7.02935155e+01 -3.54353179e-02]
```

In [14]:

```
x = np.array(fia)
Y = np.array(ccs)

import statsmodels.api as sm     #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.028
Model:                            OLS   Adj. R-squared:                  0.027
Method:                 Least Squares   F-statistic:                     29.58
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           6.69e-08
Time:                        17:01:19   Log-Likelihood:                -4346.6
No. Observations:                1030   AIC:                             8697.
Df Residuals:                    1028   BIC:                             8707.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         62.7760      4.983     12.598      0.000      52.998      72.554
x1            -0.0348      0.006     -5.439      0.000      -0.047      -0.022
==============================================================================
Omnibus:                       39.244   Durbin-Watson:                   0.858
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               41.937
Skew:                           0.478   Prob(JB):                     7.83e-10
Kurtosis:                       2.745   Cond. No.                     7.55e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.55e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
[ 6.27760358e+01 -3.48486761e-02]
```

In [15]:

```
x = np.array(a)
Y = np.array(ccs)

import statsmodels.api as sm     #needed for linear regression
from statsmodels.sandbox.regression.predstd import wls_prediction_std   #needed to get prediction
interval
X = sm.add_constant(x)
re = sm.OLS(Y, X).fit()
print(re.summary())
print(re.params)
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.108
Model:                            OLS   Adj. R-squared:                  0.107
Method:                 Least Squares   F-statistic:                     124.7
Date:                Thu, 03 Dec 2020   Prob (F-statistic):           2.10e-27
Time:                        17:01:19   Log-Likelihood:                -4302.3
No. Observations:                1030   AIC:                             8609.
```

```
Df Residuals:                    1028   BIC:                             8618.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         31.8464      0.607     52.470      0.000      30.655      33.037
x1             0.0870      0.008     11.166      0.000       0.072       0.102
==============================================================================
Omnibus:                       46.816   Durbin-Watson:                   0.763
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               52.405
Skew:                           0.548   Prob(JB):                     4.17e-12
Kurtosis:                       2.859   Cond. No.                         96.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[31.84643648  0.08697358]
```

- All of the OLS charts above, go into depth about each predictor. They give information, such as how the graph would look, some different tests, and other things. The OLS chart is just a more in-depth look at how each predictor correlates with the concrete compressive strength.

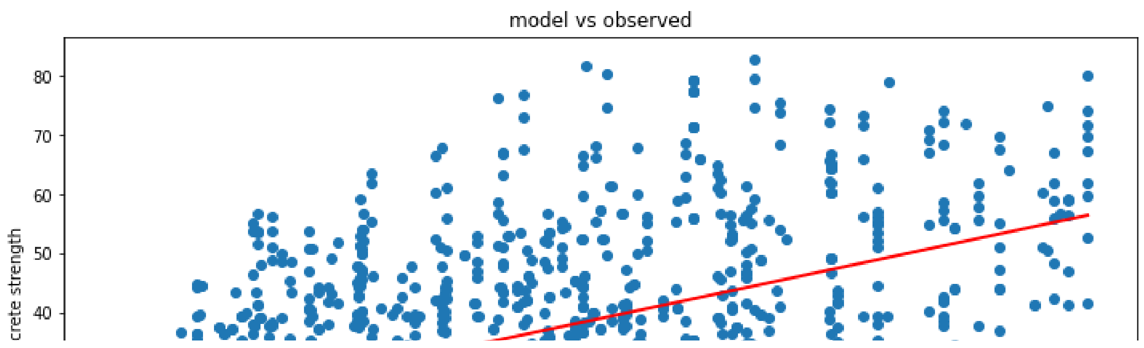
Modeling with the predictors

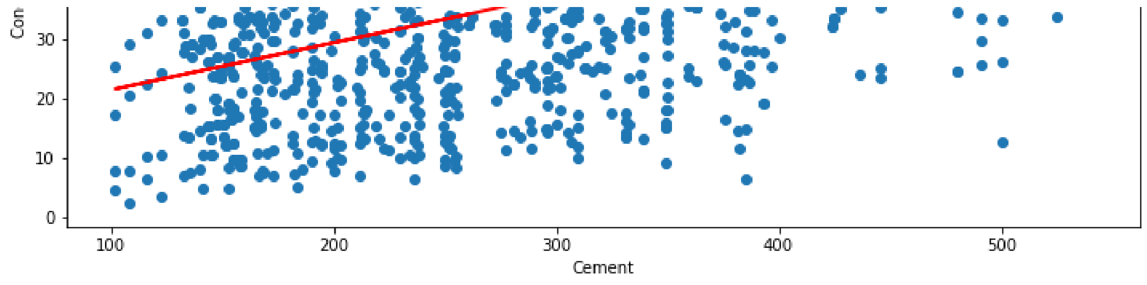
In [16]:

```python
c = np.array(df['c'])
bfs = np.array(df['bfs'])
fa = np.array(df['fa'])
w = np.array(df['w'])
s = np.array(df['s'])
ca = np.array(df['ca'])
fia = np.array(df['fia'])
a = np.array(df['a'])
ccs = np.array(df['ccs'])
```

In [17]:

```python
#Cement as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ c', data=df)
model = model.fit()

c_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['c'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['c'], c_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Cement')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

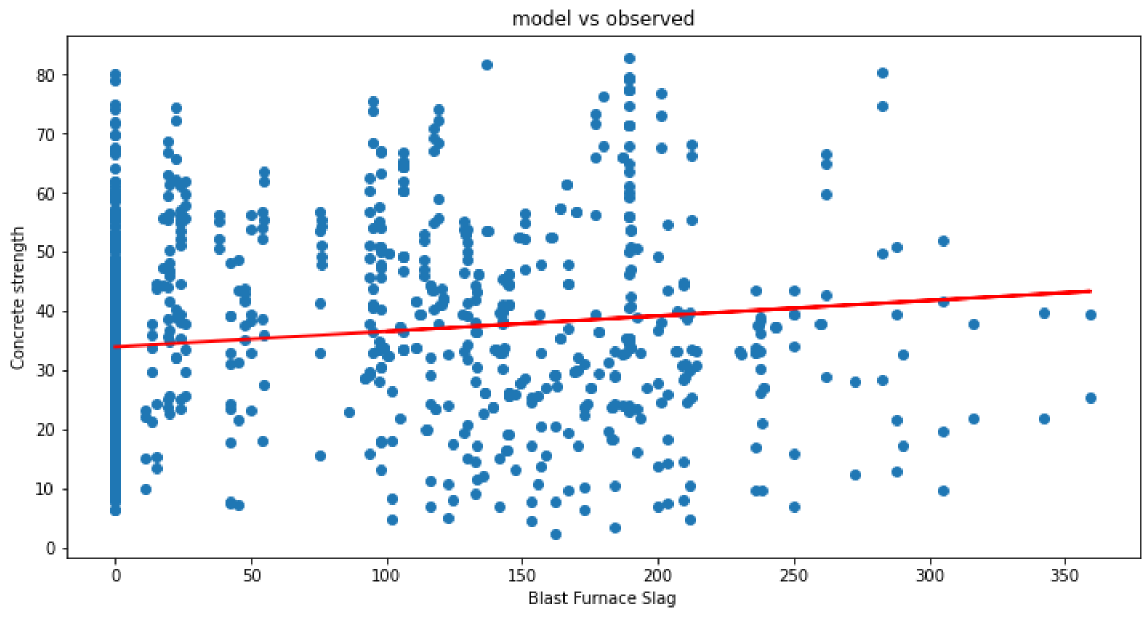




- From this graph, the cement appears to be a pretty good predictor. The reason is that the line looks correlated to the points. The points are a little spread out from the line, but the line follows the dots pretty well. And they both are going upwards and to the right.

In [18]:

```
# Blast Furnace Slag as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ bfs', data=df)
model = model.fit()

bfs_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['bfs'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['bfs'], bfs_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Blast Furnace Slag')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

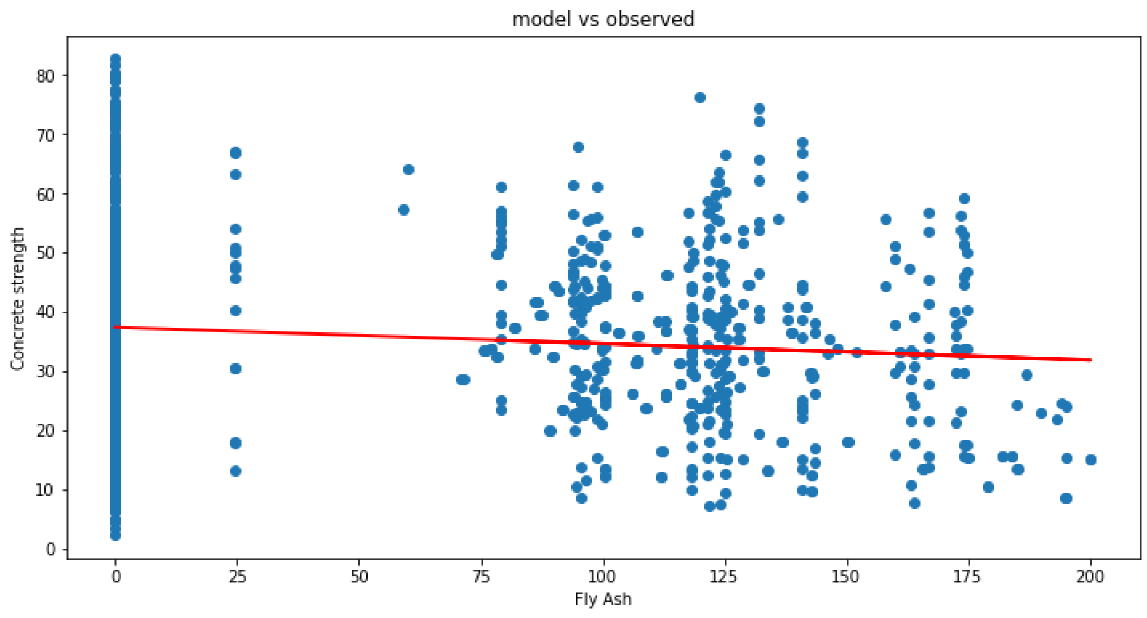


- Blast furnace slag does not look as good of a predictor as the cement did. The line is almost flat or horizontal, which means that there is little to no correlation. And the dots also do not seem to go upwards or downwards but instead, appear to fill up the entire graph.

In [19]:

```
#Fly Ash as predictor

import statsmodels.formula.api as smf
```

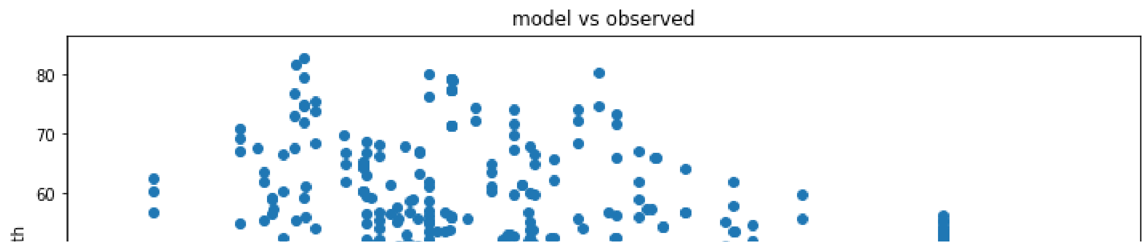
```
# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ fa', data=df)
model = model.fit()

fa_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['fa'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['fa'], fa_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Fly Ash')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

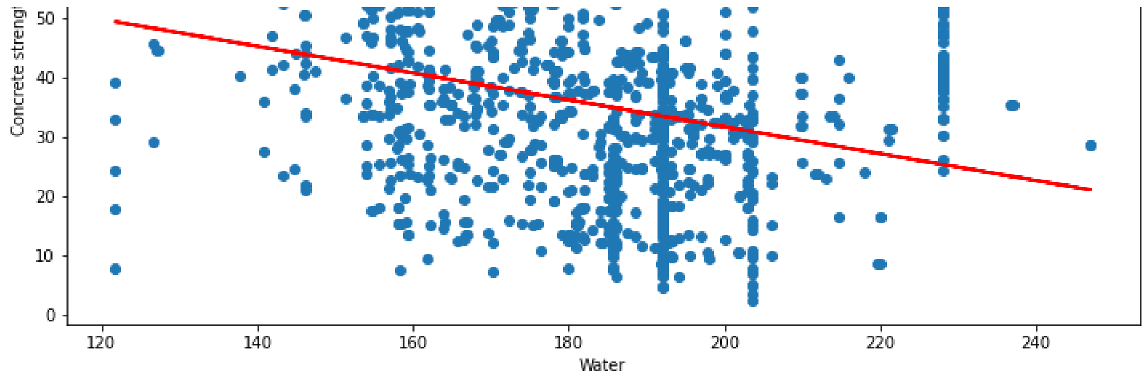


- Fly ash also does not seem to be that good of a predictor. The line is sloping a little downwards but is still almost flat. And the points do not seem to follow any path at all. They also seem to fill up most of the graph.

In [20]:

```
#Water as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ w', data=df)
model = model.fit()

w_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['w'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['w'], w_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Water')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

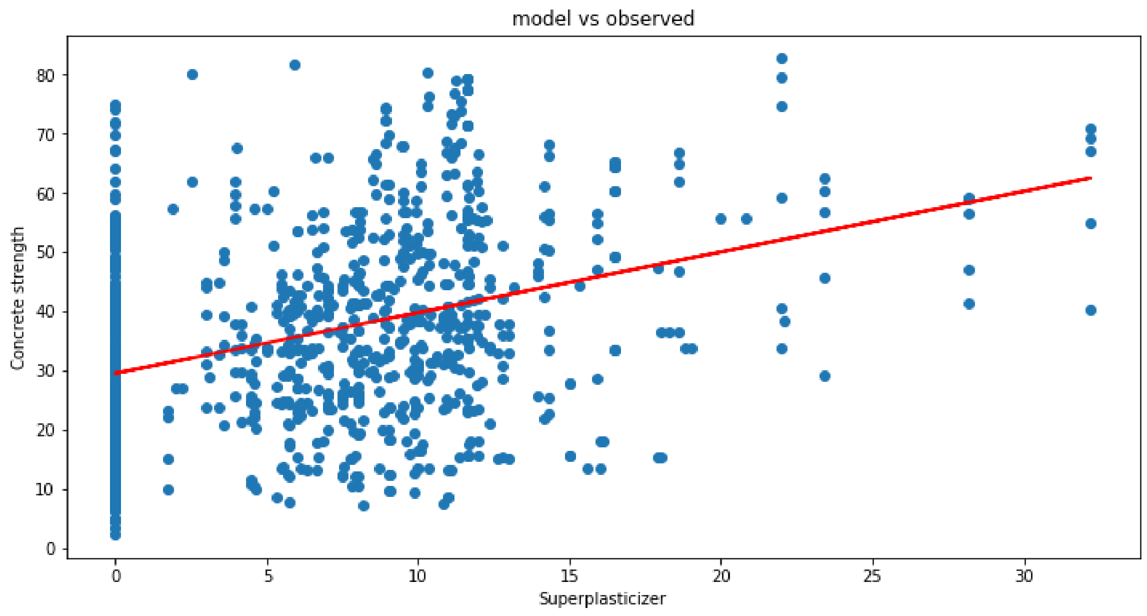




- Water seems like a decent predictor because both the line and the dots seem to be in a downwards trend. The red line is not flat and is downwards, which indicates a negative correlation. The points seem to be moving downwards, but they are spread out quite a bit.

In [21]:

```
#Superplasticizer as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ s', data=df)
model = model.fit()

s_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['s'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['s'], s_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Superplasticizer')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

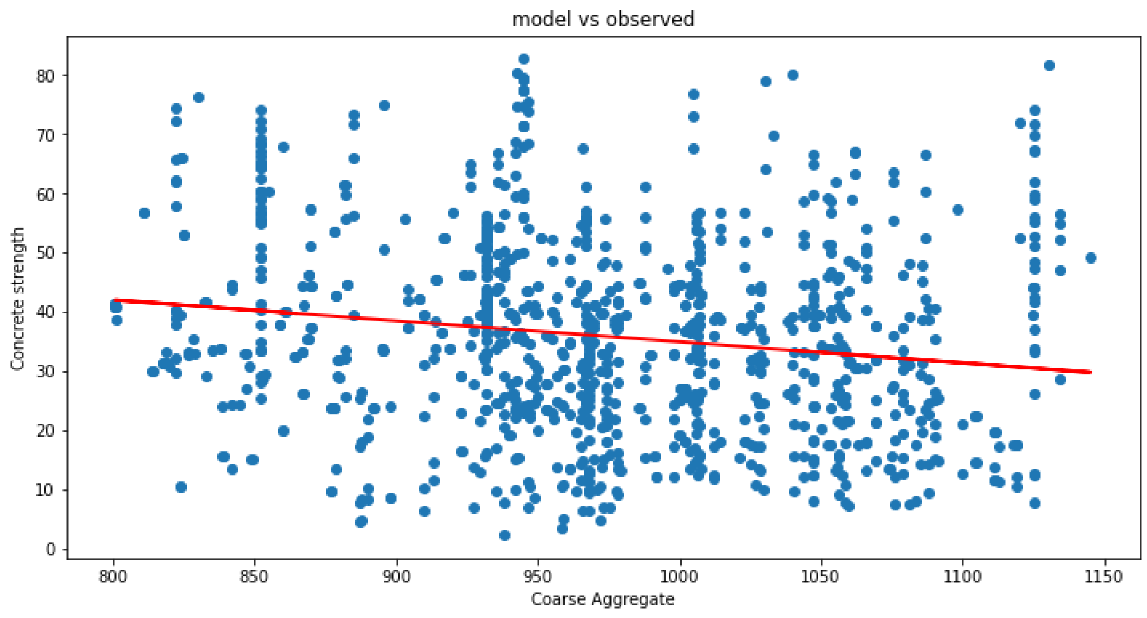


- The superplasticizer looks like a pretty good predictor based on the steepness of the slope. There are several points bunched up around 5-15 on the x-axis. There are not many points after 15. The line matches the dots, but they are not as dense in the second half of the graph compared to the first half.

In [22]:

```
#Coarse Aggregate as predictor
```

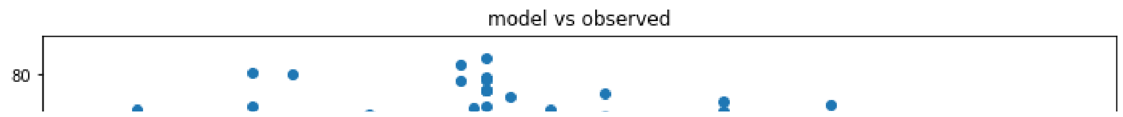
```
import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ ca', data=df)
model = model.fit()

ca_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['ca'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['ca'], ca_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Coarse Aggregate')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

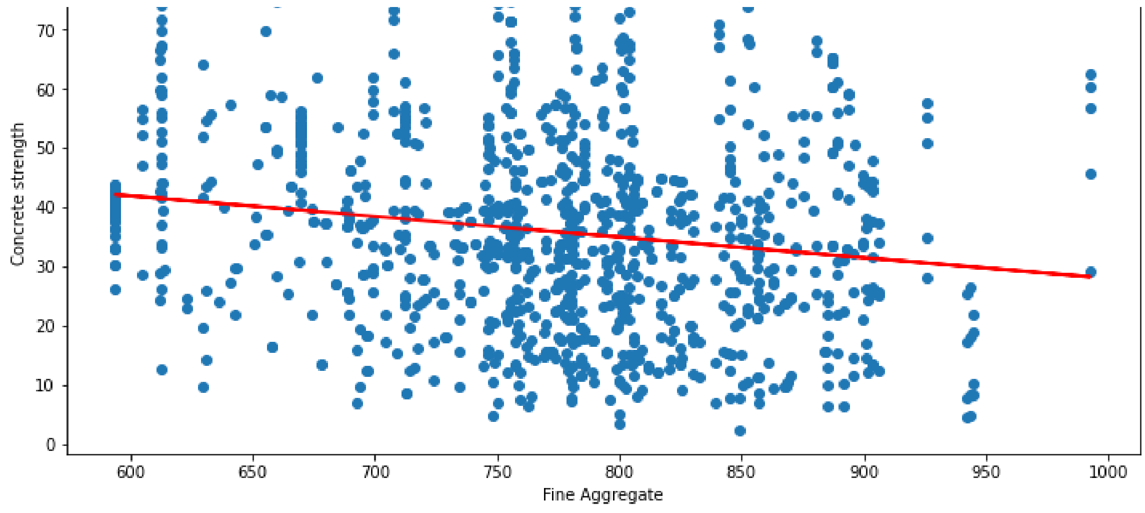


- The coarse aggregate has a negative correlation with concrete strength. Without the regression line, the points would not make any sense. The dots are pretty spread out from the regression line, and the slope is not very steep. The coarse aggregate line is not the best predictor when compared to a lot of the other predictors.

In [23]:

```
#Fine Aggregate as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ fia', data=df)
model = model.fit()

fia_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['fia'], df['ccs'], 'o')           # scatter plot showing actual data
plt.plot(df['fia'], fia_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Fine Aggregate')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```

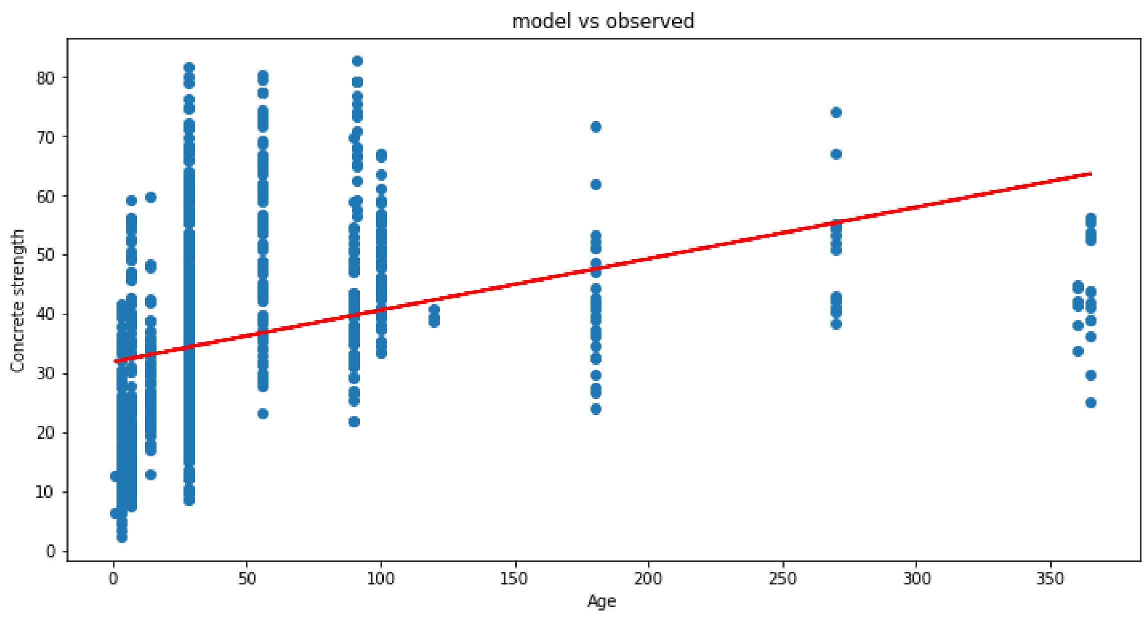




- Fine aggregate looks very similar to the coarse aggregate regression line, they both have a negative correlation, and neither are great predictors. The x-axis, however, does not extend out to 1150 but only goes to 1000.

In [24]:

```
# Age as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ a', data=df)
model = model.fit()

a_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(df['a'], df['ccs'], 'o')            # scatter plot showing actual data
plt.plot(df['a'], a_pred, 'r', linewidth=2)    # regression line
plt.xlabel('Age')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



- This graph looks different than all the other predictors. From this graph, age does not appear to be a good predictor. It is because all of the points are in vertical lines. The vertical lines, however, appear to be moving up and to the right on this graph. Just because age does not appear to be a good predictor does not mean that it is an inherently poor predictor. The GOF tests, or (goodness of fit tests) will be run on all the linear regression lines to tell us more information about the predictors (goodness of fit tests), will be run on all the linear regression lines to tell us more information about the predictors.

Goodness of fit metrics

In [25]:

```
strength = df['ccs']
# cement as predictor
model_c= smf.ols('ccs ~ c', data=df)
model_c = model_c.fit()
c_pred = model_c.predict()
# blast furnace slag as predictor
model_bfs = smf.ols('ccs ~ bfs', data=df)
model_bfs = model_bfs.fit()
bfs_pred = model_bfs.predict()
# fly ash as predictor
model_fa= smf.ols('ccs ~ fa', data=df)
model_fa = model_fa.fit()
fa_pred = model_fa.predict()
# water as predictor
model_w = smf.ols('ccs ~ w', data=df)
model_w = model_w.fit()
w_pred = model_w.predict()
# Superplasticizer as predictor
model_s= smf.ols('ccs ~ s', data=df)
model_s = model_s.fit()
s_pred = model_s.predict()
# Coarse Aggregate as predictor
model_ca = smf.ols('ccs ~ ca', data=df)
model_ca = model_ca.fit()
ca_pred = model_ca.predict()
# fine Aggregate as predictor
model_fia = smf.ols('ccs ~ fia', data=df)
model_fia = model_fia.fit()
fia_pred = model_fia.predict()
#age as pred
model_a = smf.ols('ccs ~ a', data=df)
model_a = model_a.fit()
a_pred = model_a.predict()
```

In [26]:

```
print("RMSE for cement as predictor is ",np.sqrt(metrics.mean_squared_error(strength, c_pred)))
print("RMSE for blast furnace slag as predictor is ",np.sqrt(metrics.mean_squared_error(strength, 
bfs_pred)))
print("RMSE for fly ash as predictor is ",np.sqrt(metrics.mean_squared_error(strength, fa_pred)))
print("RMSE for water as predictor is ",np.sqrt(metrics.mean_squared_error(strength, w_pred)))
print("RMSE for superplasticizer as predictor is ",np.sqrt(metrics.mean_squared_error(strength, s_
pred)))
print("RMSE for coarse aggregate as predictor is ",np.sqrt(metrics.mean_squared_error(strength, ca
_pred)))
print("RMSE for fine aggregate as predictor is ",np.sqrt(metrics.mean_squared_error(strength, fia_
pred)))
print("RMSE for age as predictor is ",np.sqrt(metrics.mean_squared_error(strength, a_pred)))
```

```
RMSE for cement as predictor is  14.4813507854314445
RMSE for blast furnace slag as predictor is  16.545110536077487
RMSE for fly ash as predictor is  16.603934240275063
RMSE for water as predictor is  15.981971441924697
RMSE for superplasticizer as predictor is  15.53833253185352
RMSE for coarse aggregate as predictor is  16.46890513601241
RMSE for fine aggregate as predictor is  16.46237731010913
RMSE for age as predictor is  15.768728271683065
```

- RMSE was utilized to test all of the predictors, and the output given was numbers. The numbers indicate how far apart the data points are or how concentrated it is around the prediction line. The way to tell which predictors are better is by looking at their numbers. The closer the number of the predictor is, the better. This is because the closer the number is to zero, the better it is.

In [27]:

```
print("R2 for cement as predictor is ",metrics.r2_score(strength, c_pred))
print("R2 for blast furnace slag as predictor is ",metrics.r2_score(strength, bfs_pred))
print("R2 for fly ash as predictor is ",metrics.r2_score(strength, fa_pred))
print("R2 for water as predictor is ",metrics.r2_score(strength, w_pred))
print("R2 for superplasticizer as predictor is ",metrics.r2_score(strength, s_pred))
print("R2 for coarse aggregate as predictor is ",metrics.r2_score(strength, ca_pred))
print("R2 for fine aggregate as predictor is ",metrics.r2_score(strength, fia_pred))
print("R2 for age as predictor is ",metrics.r2_score(strength, a_pred))
```

```
R2 for cement as predictor is  0.24783741936758186
R2 for blast furnace slag as predictor is  0.01817763100821035
R2 for fly ash as predictor is  0.01118377053623798
R2 for water as predictor is  0.08387596530148289
R2 for superplasticizer as predictor is  0.13403089236893628
R2 for coarse aggregate as predictor is  0.027201186158720025
R2 for fine aggregate as predictor is  0.027972215283952218
R2 for age as predictor is  0.10816006502079256
```

- The R2 measures the strength of the relationship, on a 0% to 100% scale, between the model and a dependent variable. The closer to 0% an R2 value is, the less explanation of variation in the response variable around its mean. The closer to 100% an R2 value is, the more explanation of variation in the response variable around its mean is provided.

In [28]:

```
from scipy.stats import pearsonr
c_r = pearsonr(c_pred, strength)
bfs_r = pearsonr(bfs_pred, strength)
fa_r = pearsonr(fa_pred, strength)
w_r = pearsonr(w_pred, strength)
s_r = pearsonr(s_pred, strength)
ca_r = pearsonr(ca_pred, strength)
fia_r = pearsonr(fia_pred, strength)
a_r = pearsonr(a_pred, strength)

print("Pearson's r for cement as predictor is ",c_r[0])
print("Pearson's r for blast furnace slag as predictor is ",bfs_r[0])
print("Pearson's r for fly ash as predictor is ",fa_r[0])
print("Pearson's r for water as predictor is ",w_r[0])
print("Pearson's r for superplasticizer as predictor is ",s_r[0])
print("Pearson's r for coarse aggregate as predictor is ",ca_r[0])
print("Pearson's r for fine aggregate as predictor is ",fia_r[0])
print("Pearson's r for age as predictor is ",a_r[0])
```

```
Pearson's r for cement as predictor is  0.4978327222748439
Pearson's r for blast furnace slag as predictor is  0.13482444514334269
Pearson's r for fly ash as predictor is  0.10575334763608216
Pearson's r for water as predictor is  0.28961347569041557
Pearson's r for superplasticizer as predictor is  0.36610229768322455
Pearson's r for coarse aggregate as predictor is  0.1649278210573347
Pearson's r for fine aggregate as predictor is  0.1672489619816888
Pearson's r for age as predictor is  0.3288769755102851
```

- Pearson's R tells us about the correlation of the line to the predictors. The closer to 1, the better the fit, and the closer to 0, the worse the fit is. The three best Pearson's r values are cement (0.498), superplasticizer (0.366), and age (0.329). None of these values are very close to 1. However, out of all of the predictors, these predictors are the closest to 1.

Goodness of fit metrics tell us about how good different predictors are. While there are some predictors that are better than others such as, cement, superplasticizer, and age, because concrete uses all of the materials (predictors) they are all important for the user. For this reason, all 8 predictors will be utilized in the interface.

Implementation for the data model user interface

In [29]:

```
# Multiple Linear Regression with scikit-learn:
from sklearn.linear_model import LinearRegression

# Build linear regression model using ... as predictors
# Split data into predictors X and output Y
```

```
predictors = ['c', 'bfs','fa','w','s','ca','fia', 'a']
X = df[predictors]
y = df.ccs

# Initialise and fit model
lm = LinearRegression()
model = lm.fit(X, y)
```

In [30]:

```
print(f'alpha = {model.intercept_}')
print(f'betas = {model.coef_}')
```

```
alpha = -23.16375581107919
betas = [ 0.11978526  0.10384725  0.08794308 -0.1502979   0.29068694  0.01803018
  0.02015446  0.11422562]
```

Therefore, our model can be written as:

- Concrete compressive strength = -23.163755811078538 + 0.11978526 *Cement + 0.10384725* Blast Furnace Slag + 0.08794308 *Fly Ash - 0.1502979* Water + 0.29068694 *Superplasticizer + 0.01803018* Coarse Aggregate + 0.02015446 *Fine Aggregate + 0.11422562* Age
- This follows the equation of $Y_e = \alpha + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p$, where p is the number of predictors.

In [31]:

```
def line(A, B, C, D, E, F, G, H) :
    y = -23.163755811078538 + 0.11978526*A + 0.10384725*B + 0.08794308*C - 0.1502979*D + 0.29068694
*E + 0.01803018*F + 0.02015446*G + 0.11422562*H
    return y
```

In [32]:

```
n = 1030                          #Sample size
Xbar = 35.817836                  #Sample mean (from df.describe() ccs column)
C = 0.95                          #Confidence level
std = 16.705679                   #Standard deviation (σ), (from df.describe(), ccs column)
z = 1.96                          #The z value associated with 95% Confidence Interval

# Calculate the margin of error
import math
margin = z*(std/math.sqrt(n))
```

In [52]:

```
try:
    A = float(input('enter amount of cement (kg in a m^3 mixture)'))
    B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)'))
    C = float(input('enter amount of fly ash (kg in a m^3 mixture)'))
    D = float(input('enter amount of water (kg in a m^3 mixture)'))
    E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)'))
    F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)'))
    G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)'))
    H = float(input('enter the age (days)'))
    yvalue = line(A,B,C,D,E,F,G,H)

    if A >= 0 and B >= 0 and C >= 0 and D >= 0 and E >= 0 and F >= 0 and G >= 0 and H >= 0 :
        print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a
margin of error of +/-',margin,'.')
    else:
        print("You cannot make concrete out of nothing! Please enter values greater than 0.")
except:
    print("Please enter a numerical value.")
```

```
enter amount of cement (kg in a m^3 mixture)175.0
enter amount of blast furnace slag (kg in a m^3 mixture)13.0
enter amount of fly ash (kg in a m^3 mixture)172.0
enter amount of water (kg in a m^3 mixture)156.0
enter amount of superplasticizer (kg in a m^3 mixture)4.0
enter amount of coarse aggregate (kg in a m^3 mixture)1000.0
enter amount of fine aggregate (kg in a m^3 mixture)856.0
enter the age (days)3.0
With a 95% confidence the concrete compressive strength is = 27.616238678921466 MPa, with a margin
of error of +/- 1.0202382251816031 .
```

Chart from project to show the concrete compressive strength for 5 different mixtures

Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	ConcreteCompressiveStrength
175.0	13.0	172.0	156.0	4.0	1000.0	856.0	3.0	27.616238678921466
320.0	0.0	0.0	192.0	0.0	970.0	850.0	7.0	21.73047552892146
320.0	0.0	126.0	209.0	5.70	860.0	856.0	28.0	32.44949984692146
320.0	73.0	54.0	181.0	6.0	972.0	773.0	45.0	40.28239013892146
530.0	359.0	200.0	247.0	32.0	1145.0	992.0	365.0	149.70074323892146

Implementation for the database to update interface

In [34]:

```python
DF = df[['c', 'bfs', 'fa', 'w', 's', 'ca', 'fia', 'a', 'ccs']]
```

In [35]:

```python
dfnew = DF
```

In [36]:

```python
def add_row(df, row):
    df.loc[-1] = row
    df.index = df.index + 1
    return df.sort_index()
```

In [48]:

```python
try:
    A = float(input('enter amount of cement (kg in a m^3 mixture)'))
    B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)'))
    C = float(input('enter amount of fly ash (kg in a m^3 mixture)'))
    D = float(input('enter amount of water (kg in a m^3 mixture)'))
    E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)'))
    F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)'))
    G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)'))
    H = float(input('enter the age (days)'))
    yvalue = line(A,B,C,D,E,F,G,H)

    if A >= 0 and B >= 0 and C >= 0 and D >= 0 and E >= 0 and F >= 0 and G >= 0 and H >= 0 :
        print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a
margin of error of +/-',margin,'. Run the next cell to ')
        row = [A,B,C,D,E,F,G,H,yvalue]
        add_row(dfnew, row)
    else:
        print("You cannot make concrete out of nothing! Please enter values greater than 0.")
except:
    print("Please enter a numerical value.")
```

```
enter amount of cement (kg in a m^3 mixture)320.0
enter amount of blast furnace slag (kg in a m^3 mixture)0.0
enter amount of fly ash (kg in a m^3 mixture)126.0
enter amount of water (kg in a m^3 mixture)209.0
enter amount of superplasticizer (kg in a m^3 mixture)5.70
enter amount of coarse aggregate (kg in a m^3 mixture)860.0
enter amount of fine aggregate (kg in a m^3 mixture)856.0
enter the age (days)28.0
With a 95% confidence the concrete compressive strength is = 32.44949984692146 MPa, with a margin
of error of +/- 1.0202382251816031 . Run the next cell to
of error of +/- [illegible] . Run the next cell to
```

In [49]:

```
dfnew
```

Out[49]:

	c	bfs	fa	w	s	ca	fia	a	ccs
2	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28.0	79.986111
3	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28.0	61.887366
4	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270.0	40.269535
5	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365.0	41.052780
6	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360.0	44.296075
...	...	...	...	...	...	...	...	...	...
1029	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28.0	23.696601
1030	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28.0	32.768036
1031	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28.0	32.401235
1	320.0	0.0	0.0	192.0	0.0	970.0	850.0	7.0	21.730476
0	320.0	0.0	126.0	209.0	5.7	860.0	856.0	28.0	32.449500

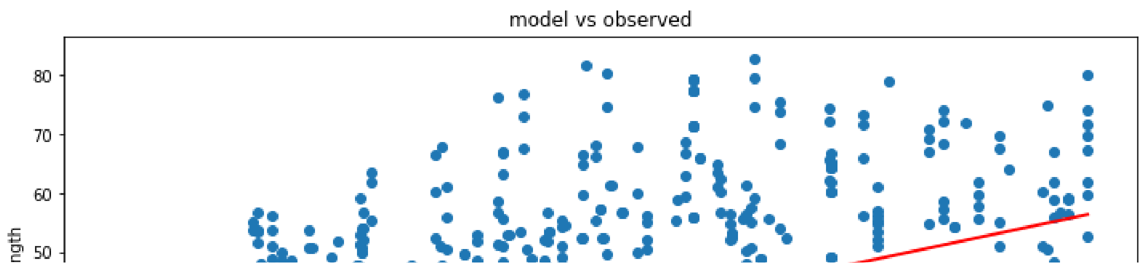
1032 rows × 9 columns

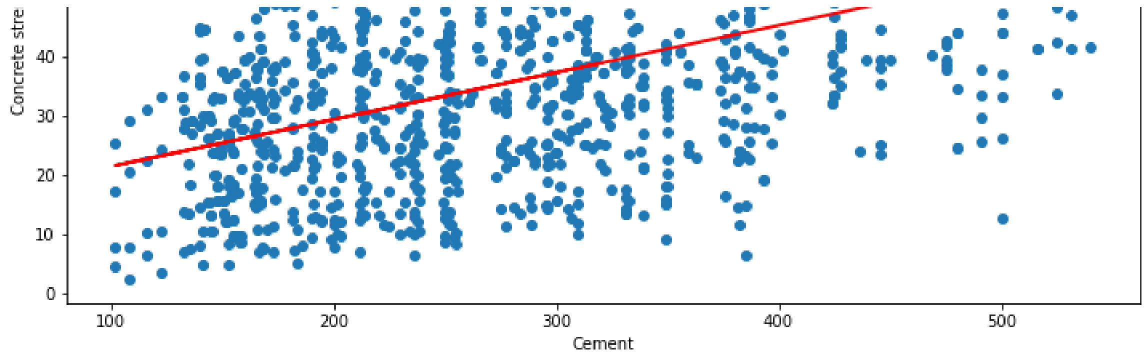
In [50]:

```
c = np.array(dfnew['c'])
bfs = np.array(dfnew['bfs'])
fa = np.array(dfnew['fa'])
w = np.array(dfnew['w'])
s = np.array(dfnew['s'])
ca = np.array(dfnew['ca'])
fia = np.array(dfnew['fia'])
a = np.array(dfnew['a'])
ccs = np.array(dfnew['ccs'])
```

In [51]:

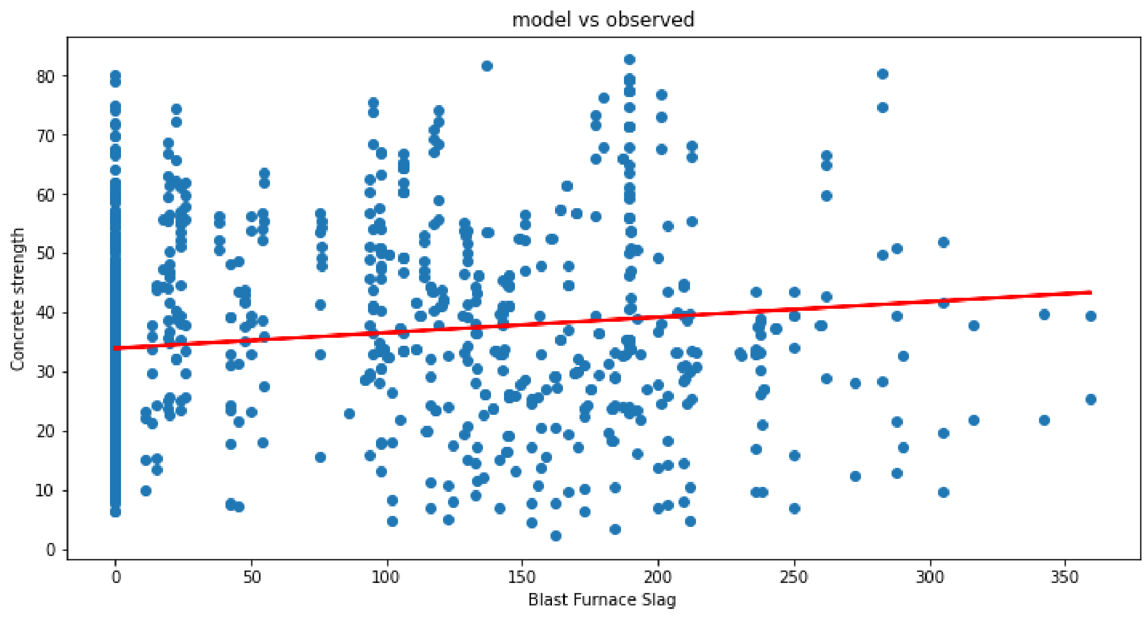
```
#Cement as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ c', data=dfnew)
model = model.fit()

c_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['c'], dfnew['ccs'], 'o')           # scatter plot showing actual data
plt.plot(dfnew['c'], c_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Cement')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```





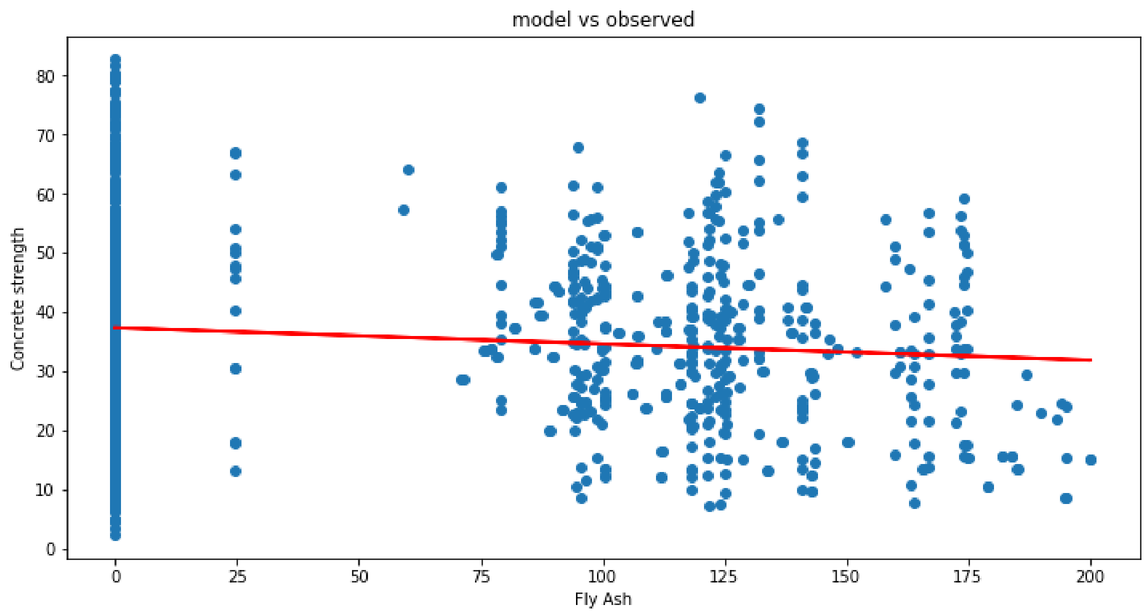
In [41]:

```
# Blast Furnace Slag as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ bfs', data=dfnew)
model = model.fit()

bfs_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['bfs'], dfnew['ccs'], 'o')            # scatter plot showing actual data
plt.plot(dfnew['bfs'], bfs_pred, 'r', linewidth=2)    # regression line
plt.xlabel('Blast Furnace Slag')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



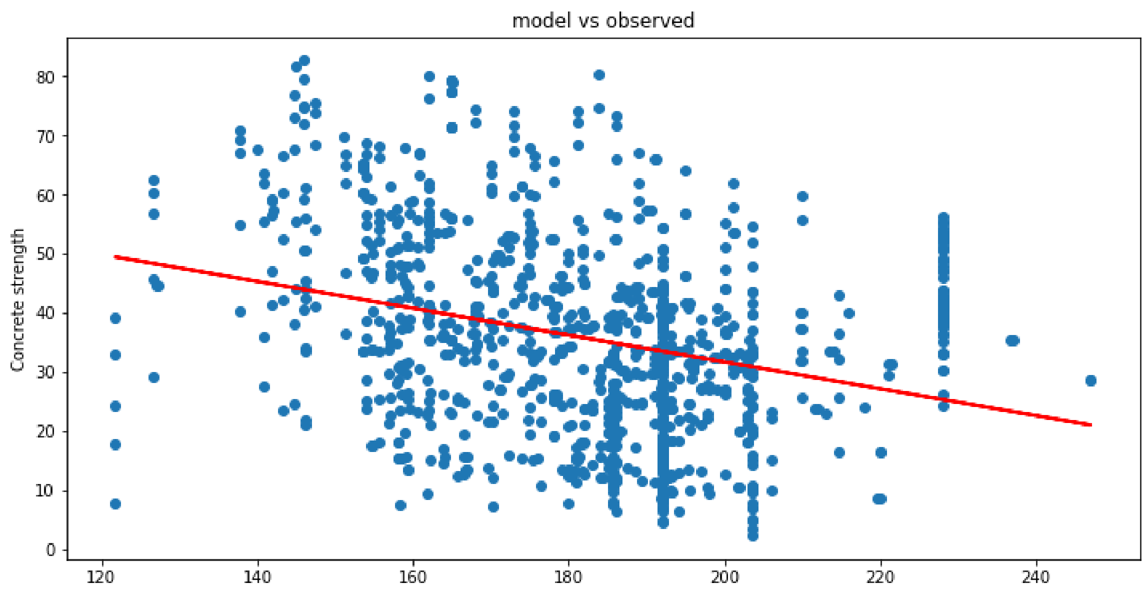
In [42]:

```
#Fly Ash as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ fa', data=dfnew)
model = model.fit()

fa_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['fa'], dfnew['ccs'], 'o')            # scatter plot showing actual data
plt.plot(dfnew['fa'], dfnew['ccs'], 'o')             # scatter plot showing actual data
plt.plot(dfnew['fa'], fa_pred, 'r', linewidth=2)    # regression line
plt.xlabel('Fly Ash')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



In [43]:

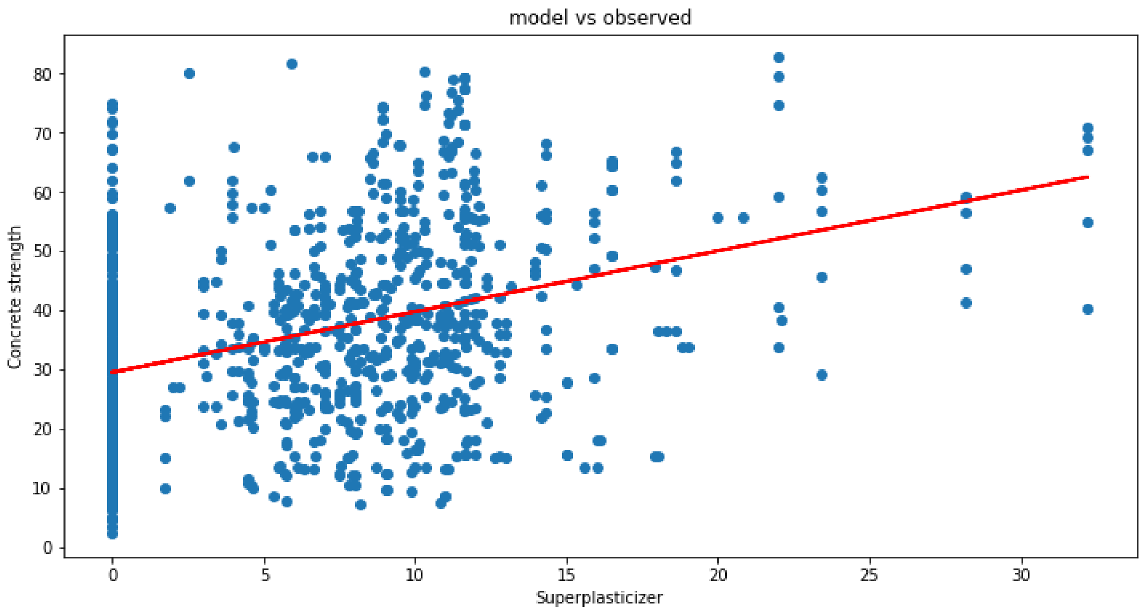
```
#Water as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ w', data=dfnew)
model = model.fit()

w_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['w'], dfnew['ccs'], 'o')             # scatter plot showing actual data
plt.plot(dfnew['w'], w_pred, 'r', linewidth=2)    # regression line
plt.xlabel('Water')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



Water

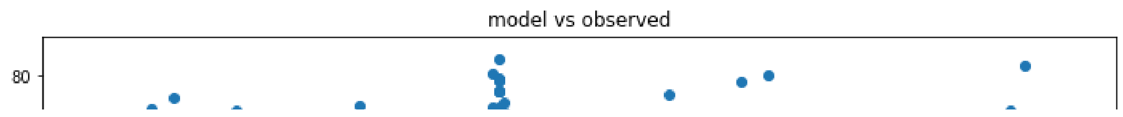
In [44]:

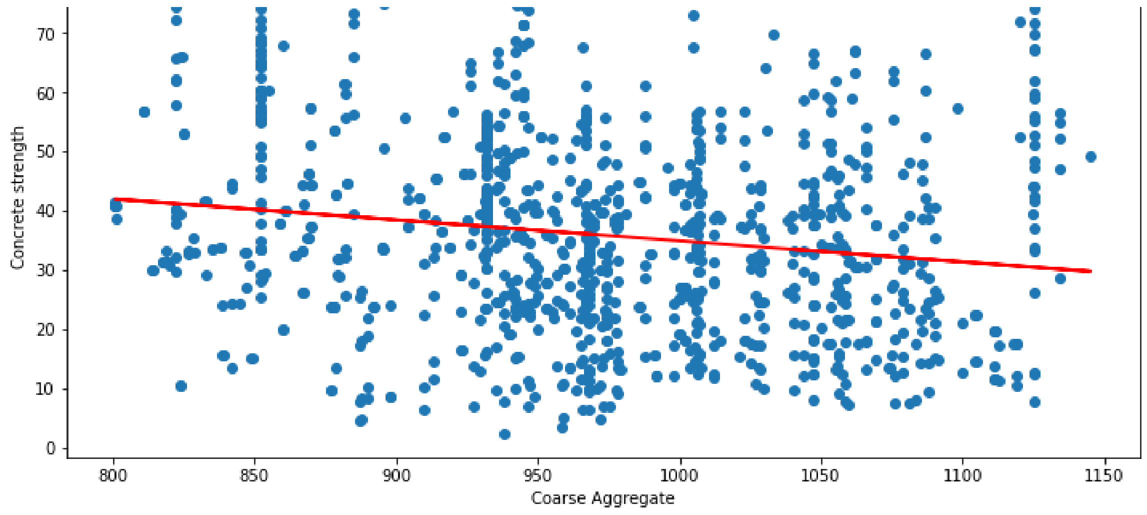
```
#Superplasticizer as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ s', data=dfnew)
model = model.fit()

s_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['s'], dfnew['ccs'], 'o')           # scatter plot showing actual data
plt.plot(dfnew['s'], s_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Superplasticizer')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



In [45]:

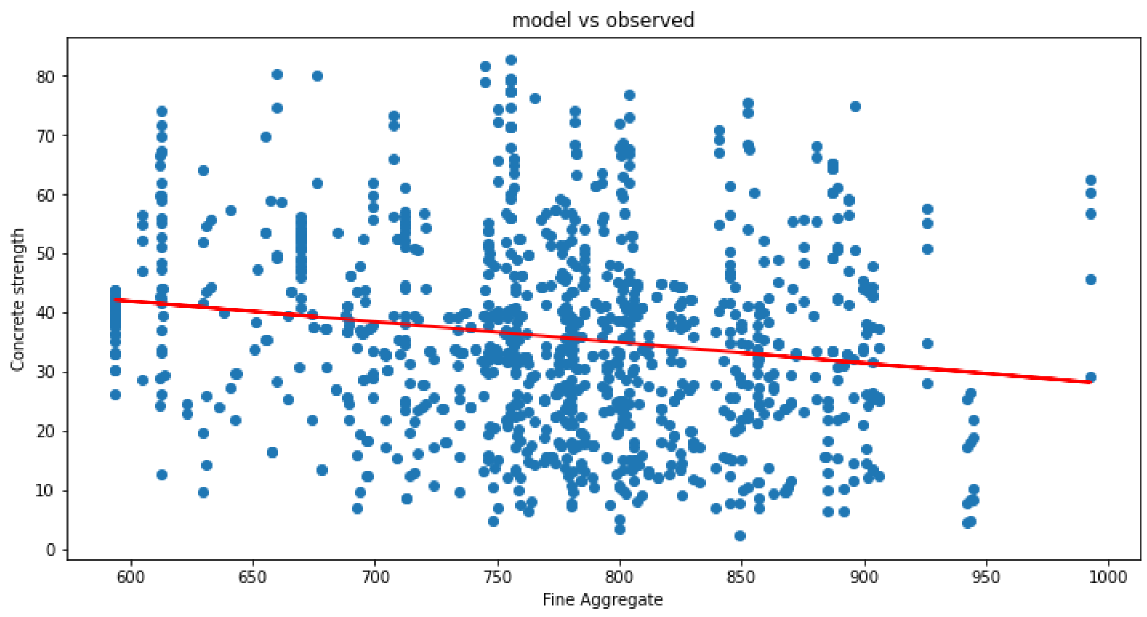
```
#Coarse Aggregate as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ ca', data=dfnew)
model = model.fit()

ca_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['ca'], dfnew['ccs'], 'o')           # scatter plot showing actual data
plt.plot(dfnew['ca'], ca_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Coarse Aggregate')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```





In [46]:

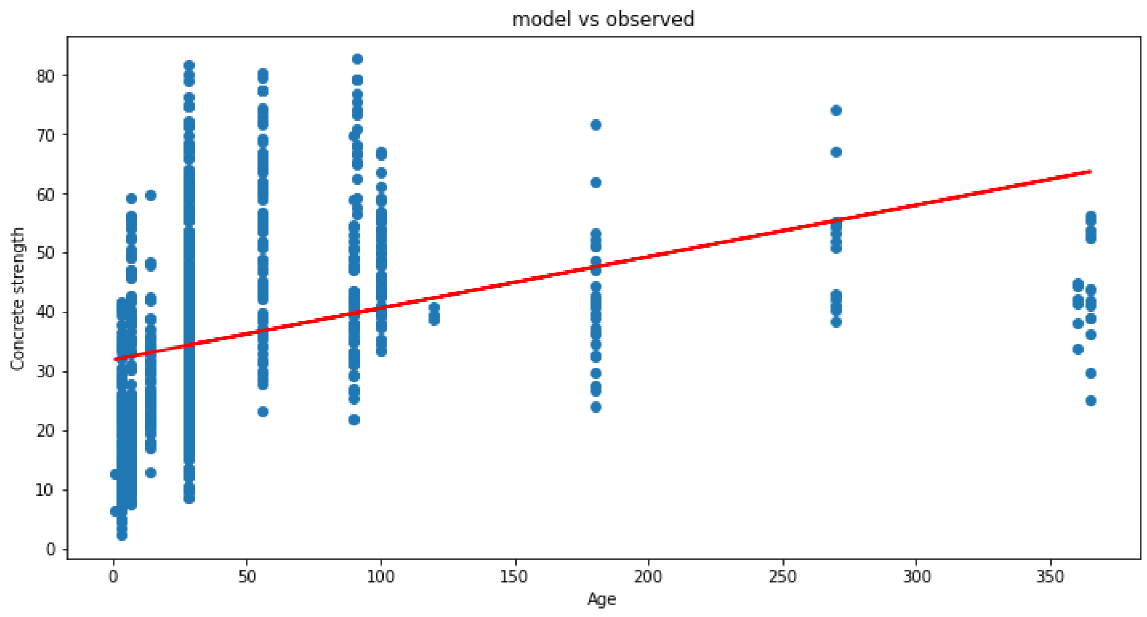
```
#Fine Aggregate as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ fia', data=dfnew)
model = model.fit()

fia_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['fia'], dfnew['ccs'], 'o')             # scatter plot showing actual data
plt.plot(dfnew['fia'], fia_pred, 'r', linewidth=2)    # regression line
plt.xlabel('Fine Aggregate')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



In [47]:

```
# Age as predictor

import statsmodels.formula.api as smf

# Initialise and fit linear regression model using `statsmodels`
model = smf.ols('ccs ~ a', data=dfnew)
model = model.fit()
```

```
a_pred = model.predict()

# Plot regression against actual data
plt.figure(figsize=(12, 6))
plt.plot(dfnew['a'], dfnew['ccs'], 'o')           # scatter plot showing actual data
plt.plot(dfnew['a'], a_pred, 'r', linewidth=2)   # regression line
plt.xlabel('Age')
plt.ylabel('Concrete strength')
plt.title('model vs observed')

plt.show()
```



Problems we ran into

- One of the biggest problems we ran into was creating a function to utilize for the interfaces. We were able to write the function and tested it multiple times to make sure it worked. We then attempted to implement it into the interface, but it would not let us for some reason. We knew that the function was functioning and ran properly. But whenever we tried to trap errors, it would give us different errors that we were not sure how to fix. We left some of our codes below as examples of the different ways we tried to code the function with the different ways we tried to trap errors but were unsuccessful.

- For the first interface, we were initially unable to get the function to work. This was because we were trying to error trap for negative numbers and strings. We were putting the error trapping outside of the function, so it was not working. Instead, we decided to not use a function for user inputs. We still called the line function in the interface to calculate the concrete strength value based on the predictors.
- For the second interface, we finally figured out the reason why our first interface was not working. It was because we had used the error trapping outside of the function. We tried on this interface to put the error trapping inside the function. It still did not work because it was giving us an error about the return yvalue. We tried so many different things, and we could not figure out how to fix it. We ended up building the interface without the function, but we still needed the add_row function to get the data to add to the database. We called the add_row function in the interface, but we could not get the ccs() function to work.

def ccs(): A = float(input('enter amount of cement (kg in a m^3 mixture)')) B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)')) C = float(input('enter amount of fly ash (kg in a m^3 mixture)')) D = float(input('enter amount of water (kg in a m^3 mixture)')) E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)')) F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)')) G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)')) H = float(input('enter the age (days)')) yvalue = line(A,B,C,D,E,F,G,H) ll = [A,B,C,D,E,F,G,H,yvalue] add_row(dfnew, ll) return yvaluedef ccs(): try: A = float(input('enter amount of cement (kg in a m^3 mixture)')) B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)')) C = float(input('enter amount of fly ash (kg in a m^3 mixture)')) D = float(input('enter amount of water (kg in a m^3 mixture)')) E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)')) F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)')) G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)')) H = float(input('enter the age (days)')) yvalue = line(A,B,C,D,E,F,G,H) if A > 0 or B > 0 or C > 0 or D > 0 or E > 0 or F > 0 or G > 0 or H > 0 : print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a margin of error of +/-',margin,'.') ll = [A,B,C,D,E,F,G,H,yvalue] add_row(dfnew, ll) return yvalue else: print("You cannot make concrete out of nothing! Please enter at least one value greater than 0.") except: print("Please enter a numerical value.") return yvaluedef ccs(): try: A = float(input('enter amount of cement (kg in a m^3 mixture)')) B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)')) C =

float(input('enter amount of fly ash (kg in a m^3 mixture)')) D = float(input('enter amount of water (kg in a m^3 mixture)')) E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)')) F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)')) G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)')) H = float(input('enter the age (days)')) yvalue = line(A,B,C,D,E,F,G,H) ll = [A,B,C,D,E,F,G,H,yvalue] add_row(dfnew, ll) if A > 0 or B > 0 or C > 0 or D > 0 or E > 0 or F > 0 or G > 0 or H > 0 : print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a margin of error of +/-',margin,'.') else: print("You cannot make concrete out of nothing! Please enter at least one value greater than 0.") except: print('Please enter a numerical value.') return yvaluedef ccs(): A = float(input('enter amount of cement (kg in a m^3 mixture)')) B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)')) C = float(input('enter amount of fly ash (kg in a m^3 mixture)')) D = float(input('enter amount of water (kg in a m^3 mixture)')) E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)')) F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)')) G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)')) H = float(input('enter the age (days)')) if A <=0 or B <= 0 or C <= 0 or D <= 0 or E <= 0 or F <= 0 or G <= 0 or H <= 0 : print("You cannot make concrete out of nothing! Please enter at least one value greater than 0.") elif A > 0 or B > 0 or C > 0 or D > 0 or E > 0 or F > 0 or G > 0 or H > 0 : print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a margin of error of +/-',margin,'.') yvalue = line(A,B,C,D,E,F,G,H) ll = [A,B,C,D,E,F,G,H,yvalue] add_row(dfnew, ll) else: print('Please enter a numerical value.') return yvaluedef ccs(): A = float(input('enter amount of cement (kg in a m^3 mixture)')) B = float(input('enter amount of blast furnace slag (kg in a m^3 mixture)')) C = float(input('enter amount of fly ash (kg in a m^3 mixture)')) D = float(input('enter amount of water (kg in a m^3 mixture)')) E = float(input('enter amount of superplasticizer (kg in a m^3 mixture)')) F = float(input('enter amount of coarse aggregate (kg in a m^3 mixture)')) G = float(input('enter amount of fine aggregate (kg in a m^3 mixture)')) H = float(input('enter the age (days)')) if A <=0 or B <= 0 or C <= 0 or D <= 0 or E <= 0 or F <= 0 or G <= 0 or H <= 0 : print("You cannot make concrete out of nothing! Please enter at least one value greater than 0.") elif A != 0 or B != 0 or C != 0 or D != 0 or E != 0 or F != 0 or G != 0 or H != 0 : print('Please enter a numerical value.') else: print('With a 95% confidence the concrete compressive strength is =',yvalue, 'MPa, with a margin of error of +/-',margin,'.') yvalue = line(A,B,C,D,E,F,G,H) ll = [A,B,C,D,E,F,G,H,yvalue] add_row(dfnew, ll) return yvalue

Citation(s):

- Examples, Python. “How to Add or Insert Row to Pandas DataFrame?” How to Add or Insert Row to Pandas DataFrame? - Python Examples, 2020, pythonexamples.org/pandas-dataframe-add-append-row/.
- Frost, Jim. “How To Interpret R-Squared in Regression Analysis.” Statistics By Jim, 3 Nov. 2020, statisticsbyjim.com/regression/interpret-r-squared-regression/.