In [19]:
```
# Load the necessary packages
import numpy as np
import pandas as pd
import statistics
import math
import requests
from matplotlib import pyplot as plt
import statsmodels.formula.api as smf
import seaborn as sns
%matplotlib inline
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
```

Exploratory Data Analysis

> make connection to online dataframe -> establish a local file and write data into the local file

In [20]:
```
remote_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/Concrete_Data.xls'
response = requests.get(remote_url)
output = open('Concrete_Data.xls', 'wb')
output.write(response.content)
output.close()
```

> open into panda dataframe -> test and summarize the data

In [21]:
```
df = pd.read_excel("Concrete_Data.xls")
```

In [22]:
```
df.describe()
```

Out[22]:

	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

In [23]:
```
df
```

Out[23]:

	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

Observations about the data model.

- 9 Columns
- 1030 Rows

> We rename the columns to make working with them easier.

In [24]:
```
df.columns = ['Cement','BlastFurnaceSlag','FlyAsh','Water','Superplasticizer','CoarseAggregate',
              'FineAggregate','Age','ConcreteCompressiveStrength']
```

In [25]:
```
df
```

Out[25]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	ConcreteCompressiveStrength
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

In [26]:
```
df.dtypes
```

Out[26]:
```
Cement                        float64
BlastFurnaceSlag              float64
FlyAsh                        float64
Water                         float64
Superplasticizer              float64
CoarseAggregate               float64
FineAggregate                 float64
Age                             int64
ConcreteCompressiveStrength   float64
dtype: object
```

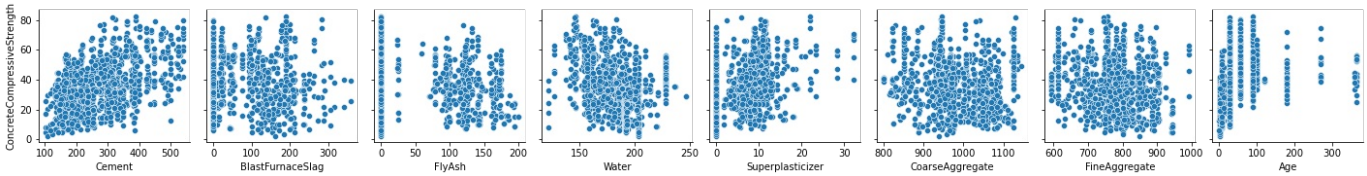
> Check for Null values

In [27]:
```
df.isna().sum()
```

Out[27]:
```
Cement                        0
BlastFurnaceSlag              0
FlyAsh                        0
Water                         0
Superplasticizer              0
CoarseAggregate               0
FineAggregate                 0
Age                           0
ConcreteCompressiveStrength   0
dtype: int64
```

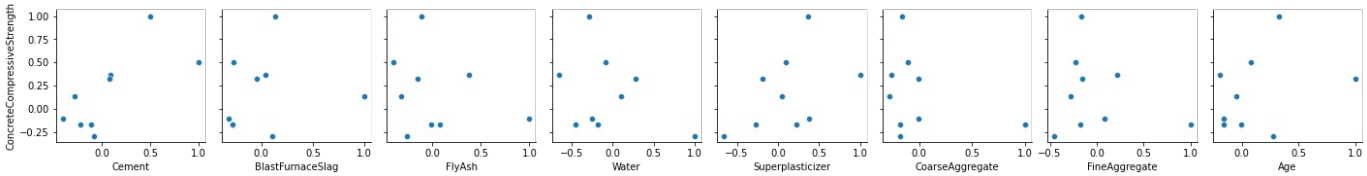
In [28]:
```
sns.pairplot(df,y_vars=['ConcreteCompressiveStrength'],x_vars=['Cement', 'BlastFurnaceSlag', 'FlyAsh', 'Water','S
```

Out[28]:
```
<seaborn.axisgrid.PairGrid at 0x2aed8c1c3a0>
```

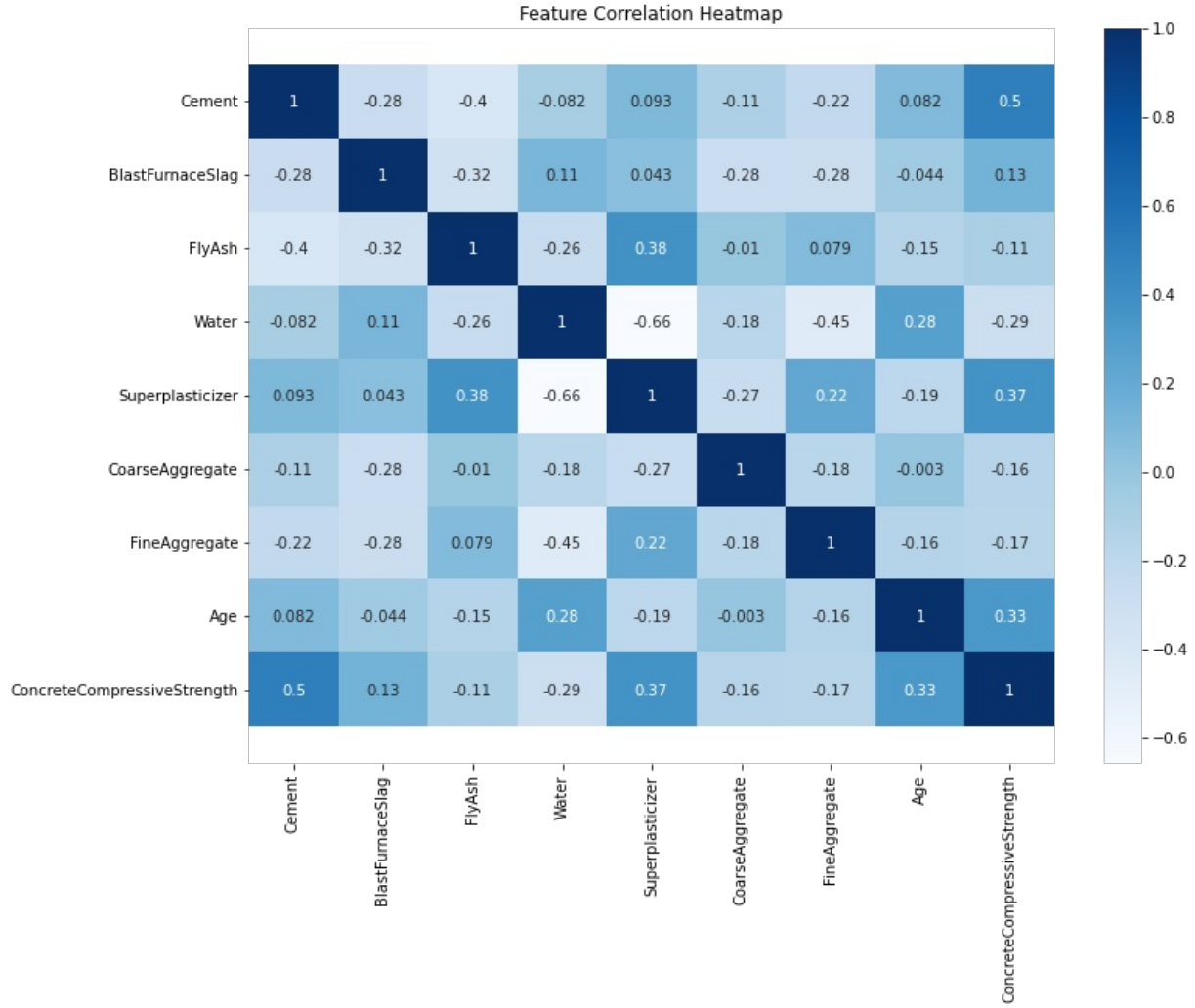


In [29]:
```
corr = df.corr()
sns.pairplot(corr,y_vars=['ConcreteCompressiveStrength'],x_vars=['Cement', 'BlastFurnaceSlag', 'FlyAsh', 'Water',
```

Out[29]:
```
<seaborn.axisgrid.PairGrid at 0x2aed8dfd550>
```



In [30]:
```
plt.figure(figsize=(12,9))
sns.heatmap(corr, annot=True, cmap='Blues')
b, t = plt.ylim()
plt.ylim(b+0.5, t-0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```

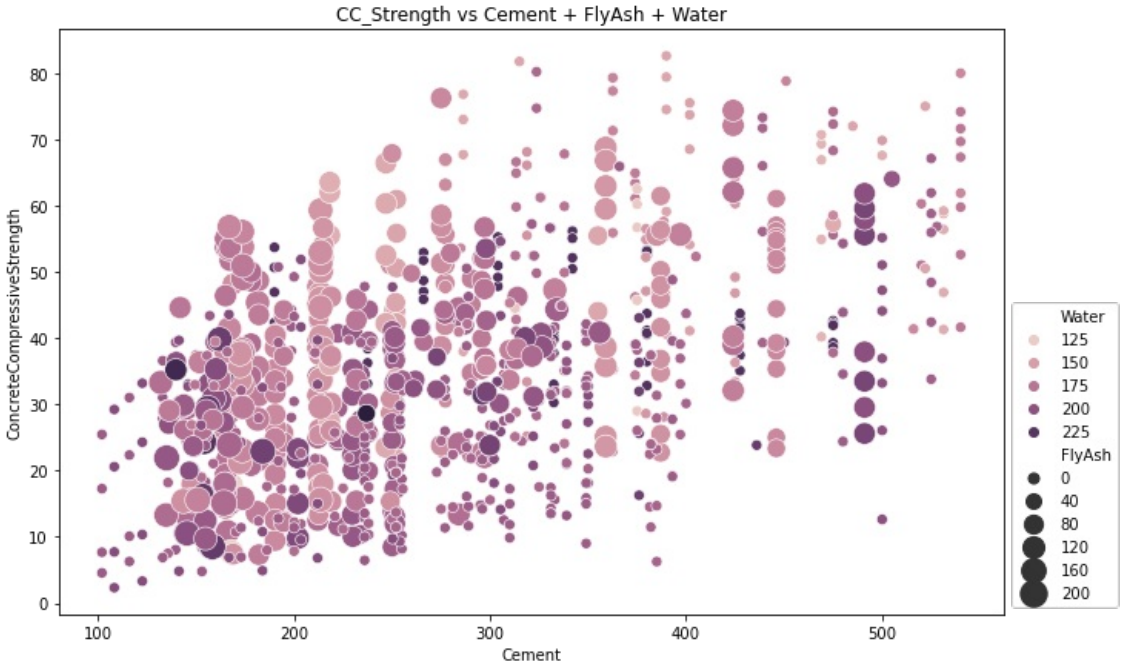


Initial Observations

The 4 greatest correlations are **Water, Age, Superplasticizer,** and the greatest of them is **Cement** at **.5**. We do also see some correlations between certain features such as **Superplasticizer** and **Water**, as well as **Cement** and **Blast Furnace Slag**.

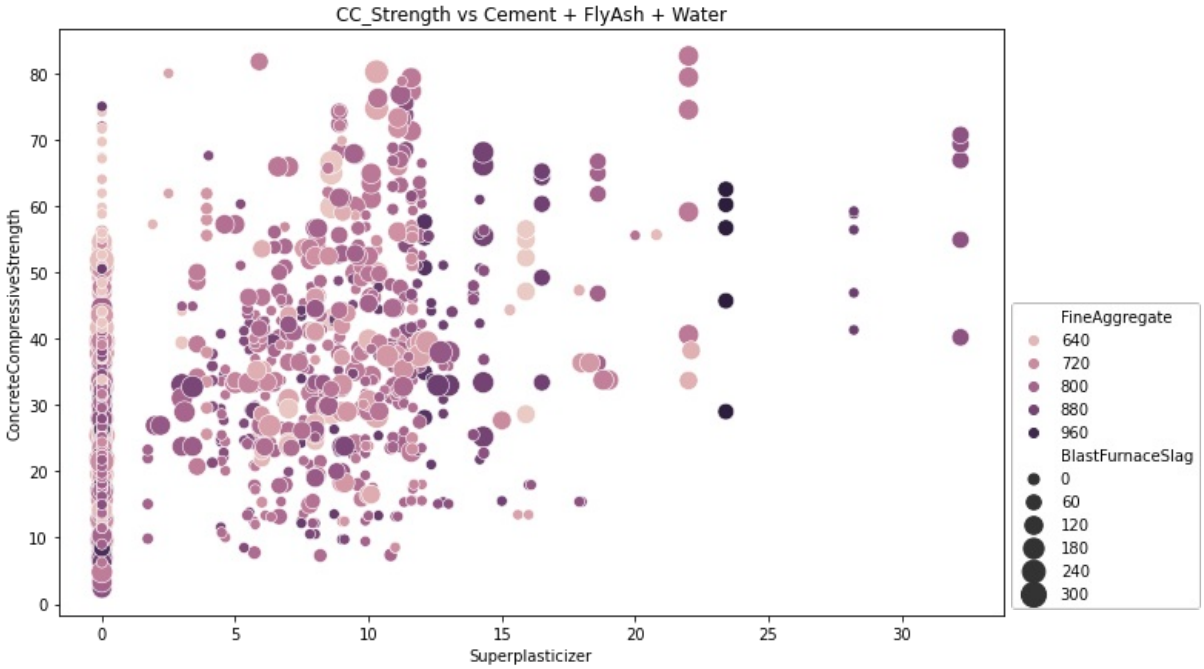
Further Investigating the Cement relationship to CC_Strength

In [32]:
```
fig, ax = plt.subplots(figsize=(11,7))
sns.scatterplot(y="ConcreteCompressiveStrength", x="Cement", hue="Water", size="FlyAsh", data=df, ax=ax, sizes=(5
ax.set_title("CC_Strength vs Cement + FlyAsh + Water")
ax.legend(loc="lower left", bbox_to_anchor=(1,0))
plt.show()
```



This scatter plot shows the negative relationship betweeen FlyAsh and Water, as well as the positive relationship betweeen cement and concrete compressive strength.

In [33]:
```
fig, ax = plt.subplots(figsize=(11,7))
sns.scatterplot(y="ConcreteCompressiveStrength", x="Superplasticizer", hue="FineAggregate", size="BlastFurnaceSla
ax.set_title("CC_Strength vs Cement + FlyAsh + Water")
ax.legend(loc="lower left", bbox_to_anchor=(1,0))
plt.show()
```



This scatter plot shows the positive relationship between Superplasticizer and Concrete Compressive Strength, as well as the negative relationship between Fine Aggregate and Blast Furnace Slag.

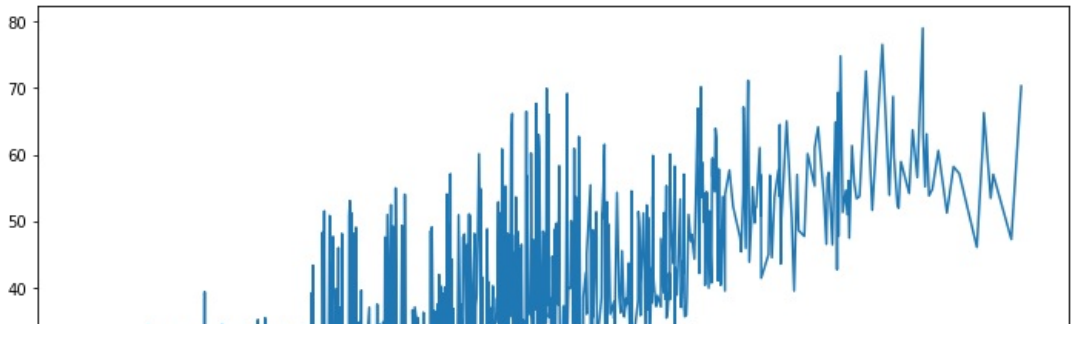
In [34]:
```
model = smf.ols('ConcreteCompressiveStrength ~ Cement + BlastFurnaceSlag + FlyAsh + Water + Superplasticizer + Co
model = model.fit()
pred = model.predict()
print(model.summary())
```

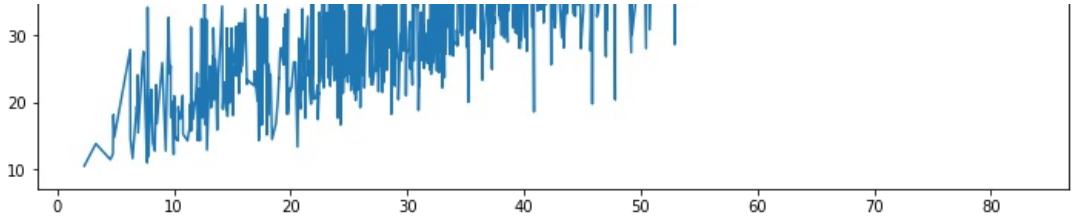
```
                                OLS Regression Results
==============================================================================
Dep. Variable:     ConcreteCompressiveStrength   R-squared:                       0.615
Model:                                     OLS   Adj. R-squared:                  0.612
Method:                          Least Squares   F-statistic:                     204.3
Date:                         Tue, 03 May 2022   Prob (F-statistic):          6.76e-206
Time:                                 15:28:31   Log-Likelihood:                -3869.0
No. Observations:                         1030   AIC:                             7756.
Df Residuals:                             1021   BIC:                             7800.
Df Model:                                    8
Covariance Type:                     nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept        -23.1638     26.588     -0.871      0.384     -75.338      29.010
Cement             0.1198      0.008     14.110      0.000       0.103       0.136
BlastFurnaceSlag   0.1038      0.010     10.245      0.000       0.084       0.124
FlyAsh             0.0879      0.013      6.988      0.000       0.063       0.113
Water             -0.1503      0.040     -3.741      0.000      -0.229      -0.071
Superplasticizer   0.2907      0.093      3.110      0.002       0.107       0.474
CoarseAggregate    0.0180      0.009      1.919      0.055      -0.000       0.036
FineAggregate      0.0202      0.011      1.883      0.060      -0.001       0.041
Age                0.1142      0.005     21.046      0.000       0.104       0.125
==============================================================================
Omnibus:                        5.379   Durbin-Watson:                   1.281
Prob(Omnibus):                  0.068   Jarque-Bera (JB):                5.305
Skew:                          -0.174   Prob(JB):                       0.0705
Kurtosis:                       3.045   Cond. No.                     1.06e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.06e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Linear Regression

In [35]:
```
b0 = model.params[0]
b1 = model.params[1]
b2 = model.params[2]
b3 = model.params[3]
b4 = model.params[4]
b5 = model.params[5]
b6 = model.params[6]
b7 = model.params[7]
b8 = model.params[8]

sdf=df.sort_values("ConcreteCompressiveStrength")

s = b0+b1*sdf['Cement']+b2*sdf['BlastFurnaceSlag']+b3*sdf['FlyAsh']+b4*sdf['Water']+b5*sdf['Superplasticizer']+b6

plt.figure(figsize=[12,6])
plt.plot(sdf['ConcreteCompressiveStrength'],s)
```

Out[35]:
```
[<matplotlib.lines.Line2D at 0x2aed96e6e50>]
```

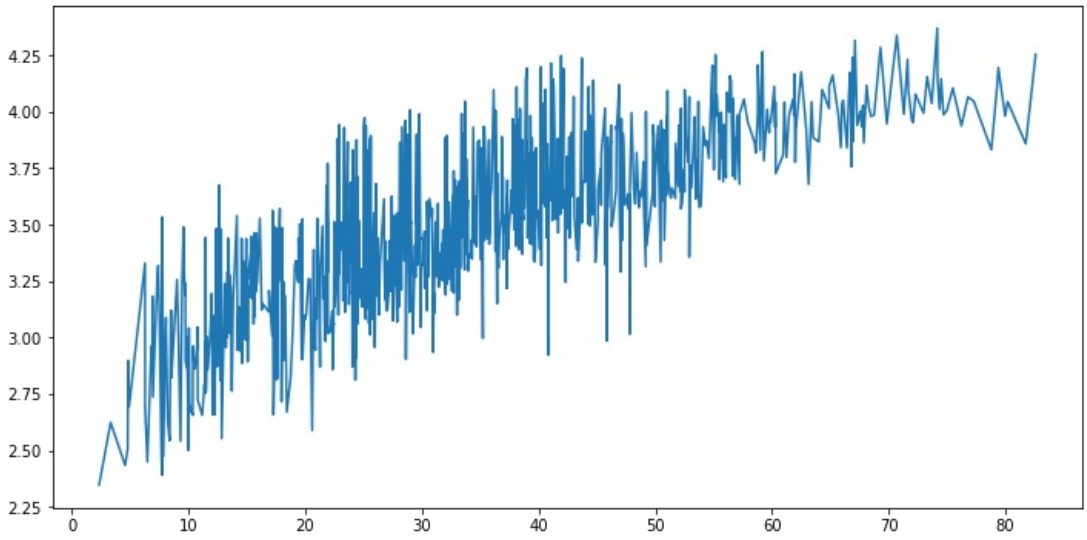




Logistical Regression

In [36]:
```
ls = np.log(b0+b1*sdf['Cement']+b2*sdf['BlastFurnaceSlag']+b3*sdf['FlyAsh']+b4*sdf['Water']+b5*sdf['Superplastici

plt.figure(figsize=[12,6])
plt.plot(sdf['ConcreteCompressiveStrength'],ls)
```

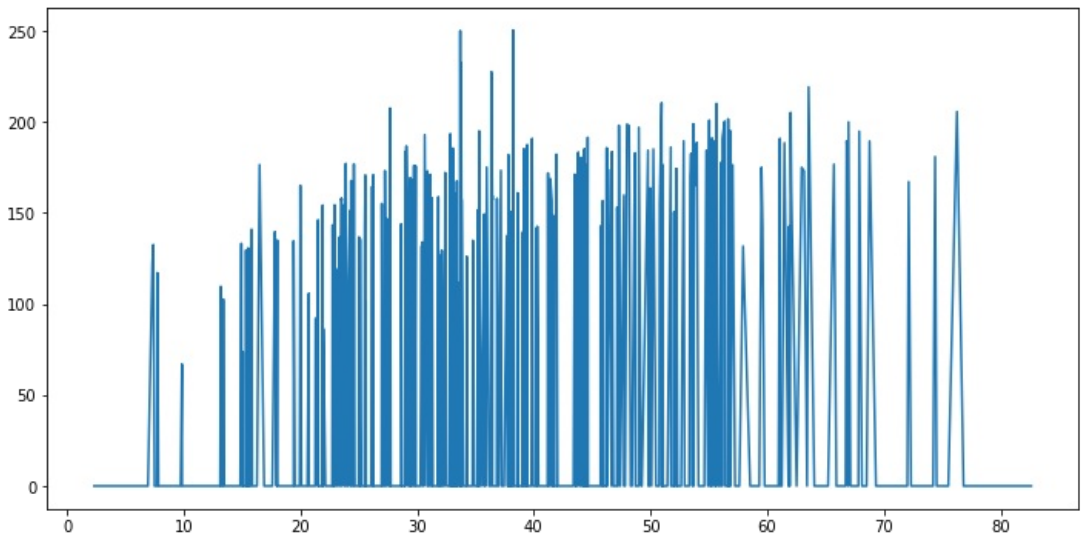
Out[36]:
```
[<matplotlib.lines.Line2D at 0x2aed975b940>]
```



Expontential Regression

In [37]:
```
es = b0*(sdf['Cement']**b1*sdf['BlastFurnaceSlag']**b2*sdf['FlyAsh']**b3*sdf['Water']**b4*sdf['Superplasticizer']

plt.figure(figsize=[12,6])
plt.plot(sdf['ConcreteCompressiveStrength'],np.abs(es))
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x2aed97d1220>]
```



R^2 and RMSE Values

In [38]:
```
lrr2 = r2_score(sdf['ConcreteCompressiveStrength'],s)
logr2 = r2_score(sdf['ConcreteCompressiveStrength'],ls)
expr2 = r2_score(sdf['ConcreteCompressiveStrength'],es)
```

In [40]:
```
lrrmse=np.sqrt(mean_squared_error(sdf['ConcreteCompressiveStrength'],s))
logrmse=np.sqrt(mean_squared_error(sdf['ConcreteCompressiveStrength'],ls))
exprmse=np.sqrt(mean_squared_error(sdf['ConcreteCompressiveStrength'],es))
```

In [41]:
```
print("R2 value for linear regression:",lrr2)

print("R2 value for log regression:",logr2)

print("R2 value for exponential regression:",expr2)

print("RMSE value for linear regression:",lrrmse)

print("RMSE value for log regression:",logrmse)

print("RMSE value for exponential regression:",exprmse)
```

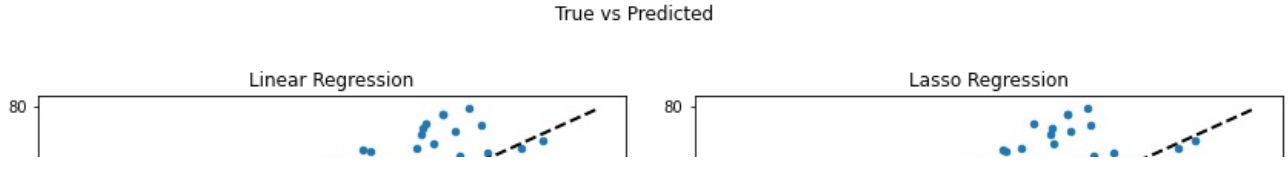
```
R2 value for linear regression: 0.6154647342687215
R2 value for log regression: -3.7087614373616358
R2 value for exponential regression: -36.37175935714657
RMSE value for linear regression: 10.354313243016426
RMSE value for log regression: 36.233188701010114
RMSE value for exponential regression: 102.07631503630532
```

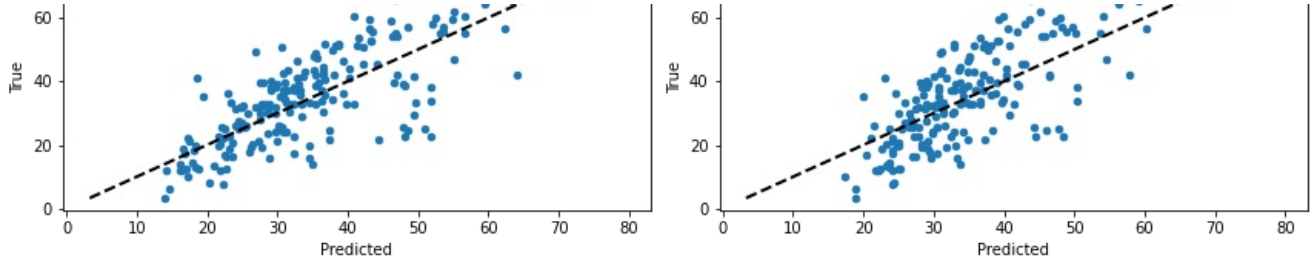
In [42]:
```
X = sdf.iloc[:,0:8] # Features
y = sdf.iloc[:,-1:] # Target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [43]:
```
# Linear Regression
lr = LinearRegression()
# Lasso Regression
lasso = Lasso()
# Fitting models on Training data
lr.fit(X_train, y_train)
lasso.fit(X_train, y_train)
# Making predictions on Test data
y_pred_lr = lr.predict(X_test)
y_pred_lasso = lasso.predict(X_test)

print("Model\t\t\t RMSE \t\t R2")
print("""LinearRegression \t {:.2f} \t\t{:.2f}""".format(  np.sqrt(mean_squared_error(y_test, y_pred_lr)), r2_sc
print("""LassoRegression \t {:.2f} \t\t{:.2f}""".format( np.sqrt(mean_squared_error(y_test, y_pred_lasso)), r2_s
```

```
Model                   RMSE            R2
LinearRegression        10.16           0.61
LassoRegression         10.89           0.55
```

In [44]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,4))
ax1.scatter(y_pred_lr, y_test, s=20)
ax1.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
ax1.set_ylabel("True")
ax1.set_xlabel("Predicted")
ax1.set_title("Linear Regression")
ax2.scatter(y_pred_lasso, y_test, s=20)
ax2.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
ax2.set_ylabel("True")
ax2.set_xlabel("Predicted")
ax2.set_title("Lasso Regression")
fig.suptitle("True vs Predicted")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```

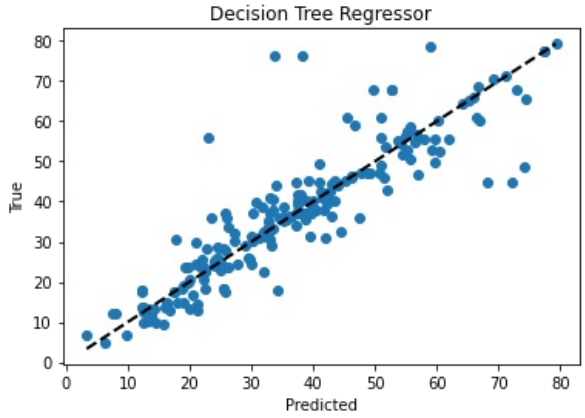




In [45]:

```
dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
y_pred_dtr = dtr.predict(X_test)
print("Model\t\t\t\t RMSE \t\t R2")
print("""Decision Tree Regressor \t {:.2f} \t\t{:.2f}""".format( np.sqrt(mean_squared_error(y_test, y_pred_dtr)),
plt.scatter(y_test, y_pred_dtr)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Decision Tree Regressor")
plt.show()
```

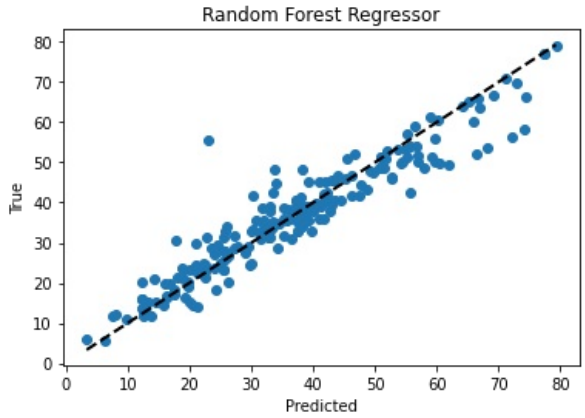
```
Model                           RMSE            R2
Decision Tree Regressor         7.63            0.78
```



In [46]:

```
rfr = RandomForestRegressor(n_estimators=100)
rfr.fit(X_train, y_train)
y_pred_rfr = rfr.predict(X_test)
print("Model\t\t\t\t RMSE \t\t R2")
print("""Random Forest Regressor \t {:.2f} \t\t{:.2f}""".format(np.sqrt(mean_squared_error(y_test, y_pred_rfr)),r
plt.scatter(y_test, y_pred_rfr)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Random Forest Regressor")
plt.show()
```

```
Model                           RMSE            R2
Random Forest Regressor         5.27            0.90
```



As we can see not only from visual observation, but also by the R sqared of ~.9, The Random Forest Regressor is the best model for

predicting concrete compressive strength.

5 User input

Here we made a user input interface so we can predict as many new/random mixtures we want.

In [88]:
```
one = input("Would you like to add values?(Y/N)")
if(( one != 'Y') and (one != 'N')):
    one=input("That answer is not valid, try again.")
while (one == "Y"):
    Cement_value = float(input("Enter Cement value here:"))
    Blast_Furnace_Slag = float(input('Enter Blast Furncace Slag Value:'))
    Fly_Ash = float(input('Enter Fly Ash here:'))
    Water_value = float(input('Enter Water value here:'))
    Superplasterizer_value = float(input('Enter Superplasterizer value here:'))
    Course_aggragate_value = float(input('Enter Course Aggreagate value here:'))
    Fine_Aggragate = float(input('Enter Fine Aggregate value here:'))
    Age = float(input('Enter Age value here:'))

    AVFQ = [[Cement_value, Blast_Furnace_Slag, Fly_Ash,Water_value, Superplasterizer_value, Course_aggragate_valu
    pred=rfr.predict(AVFQ)
    print(pred)
    df.loc[len(df.index)] = [Cement_value, Blast_Furnace_Slag, Fly_Ash,Water_value, Superplasterizer_value, Cours
    one= input("Keep going?'Y/N'")
    if(( one != 'Y') and (one != 'N')):
        one=input("That answer is not valid, try again.")
    if (one =='N'):
        break
```

```
[61.79610967]
```

In [91]:
```
df
```

Out[91]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	ConcreteCompressiveStrength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28.0	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28.0	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270.0	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365.0	41.05278
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360.0	44.296075
...	...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28.0	44.284354
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28.0	31.178794
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28.0	23.696601
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28.0	32.768036
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28.0	32.401235

1030 rows × 9 columns

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js