In [1]:

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

import scipy.stats as stats
```

In [66]:

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score

from sklearn.preprocessing import PowerTransformer
```

Power Transform

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modeling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

Currently, PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.

Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.

By default, zero-mean, unit-variance normalization is applied to the transformed data.

In [67]:

```
df = pd.read_csv('concrete_data.csv')
```

In [68]:

```
df.head()
```

Out[68]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.05
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.30

In [69]:

```
df.shape
```

Out[69]:

```
(1030, 9)
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

In [70]:
```python
df.describe()
```

Out[70]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Strength
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	35.817961
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	16.705742
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	2.330000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	23.710000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	34.445000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	46.135000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	82.600000

In [71]:
```python
X = df.drop(columns=['Strength'])
y = df.iloc[:,-1]
```

In [72]:
```python
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

Applying Regression without any transformation

In [73]:
```python
lr = LinearRegression()

lr.fit(X_train,y_train)

y_pred = lr.predict(X_test)

r2_score(y_test,y_pred)
```

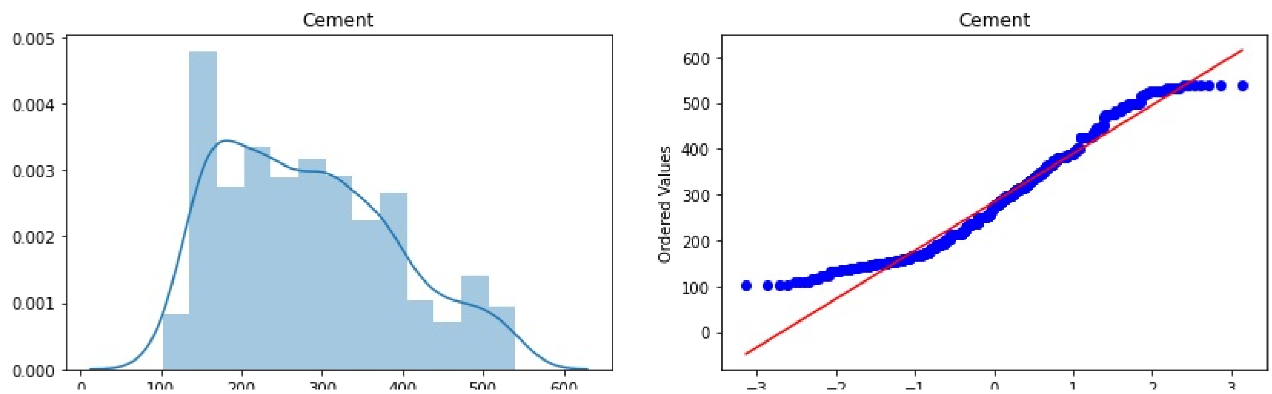
Out[73]: 0.6275531792314851

In [74]:
```python
# Cross checking with cross val score
lr = LinearRegression()
np.mean(cross_val_score(lr,X,y,scoring='r2'))
```

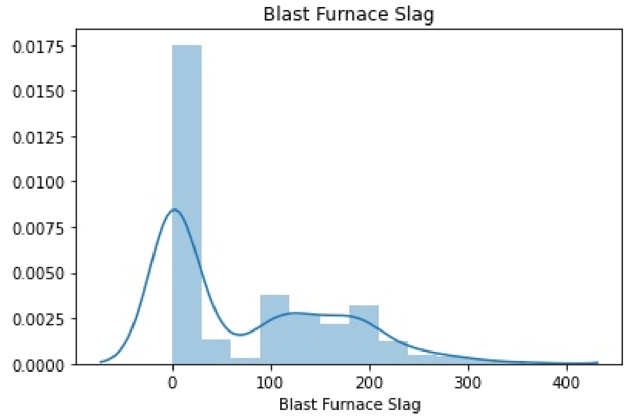
Out[74]: 0.46099404916628633

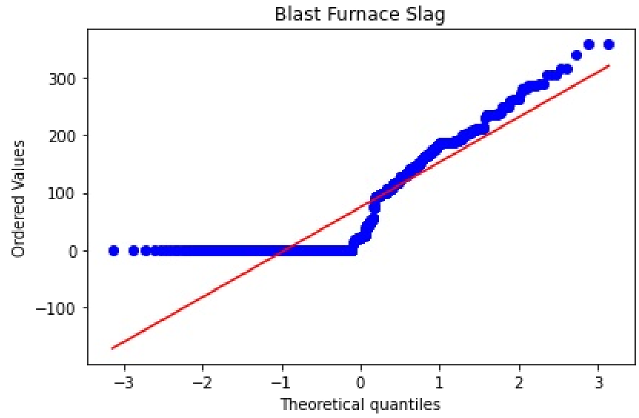
Plotting the distplots without any transformation

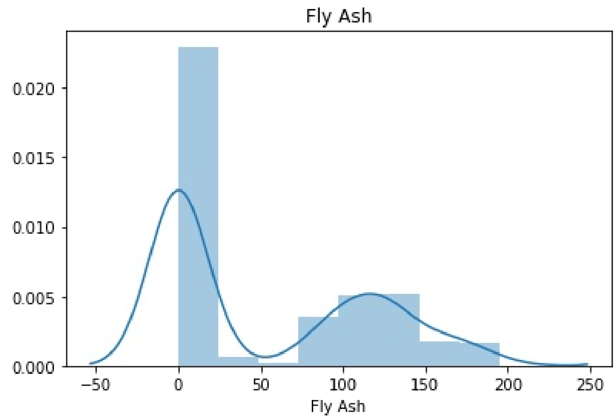
In [75]:
```python
for col in X_train.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    stats.probplot(X_train[col], dist="norm", plot=plt)
    plt.title(col)

    plt.show()
```

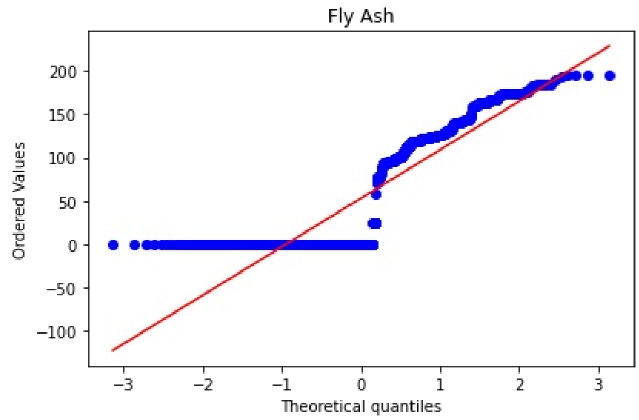


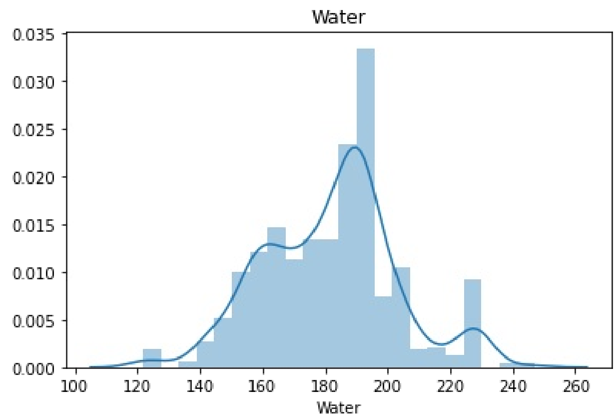

Cement
Theoretical quantiles

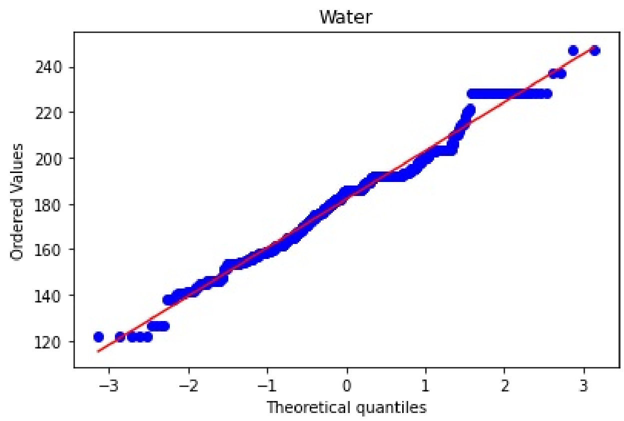

Blast Furnace Slag
Blast Furnace Slag

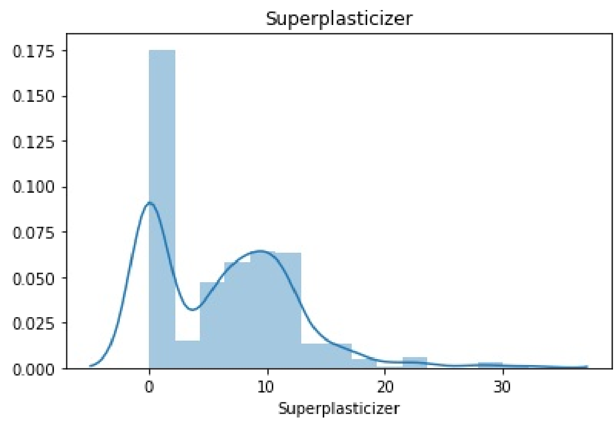

Blast Furnace Slag
Ordered Values
Theoretical quantiles

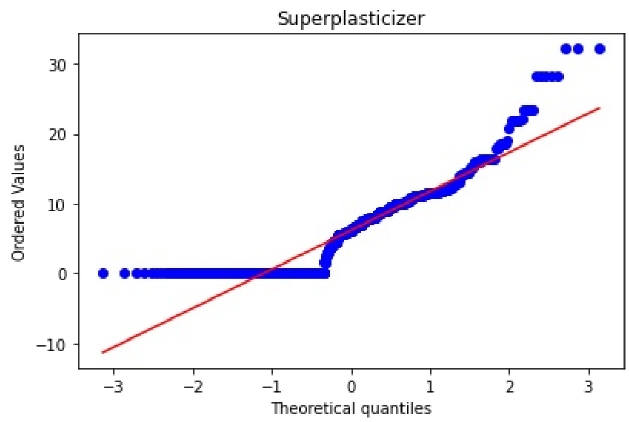

Fly Ash
Fly Ash

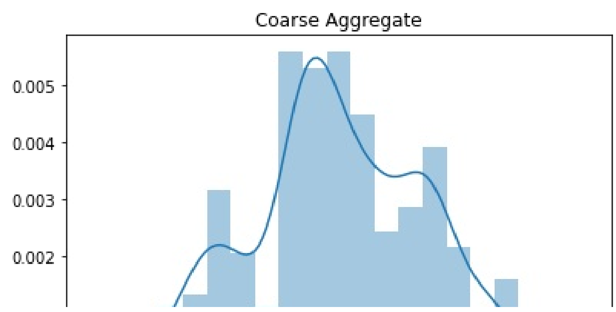

Fly Ash
Ordered Values
Theoretical quantiles

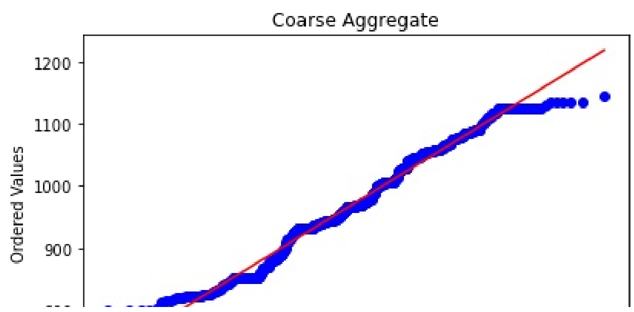

Water
Water

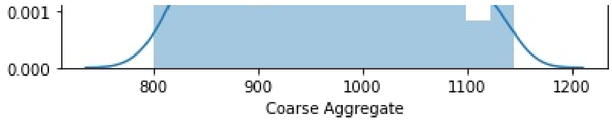

Water
Ordered Values
Theoretical quantiles

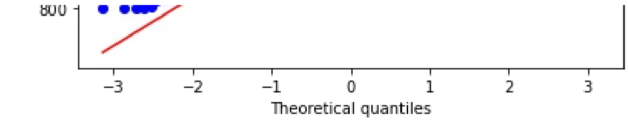

Superplasticizer
Superplasticizer

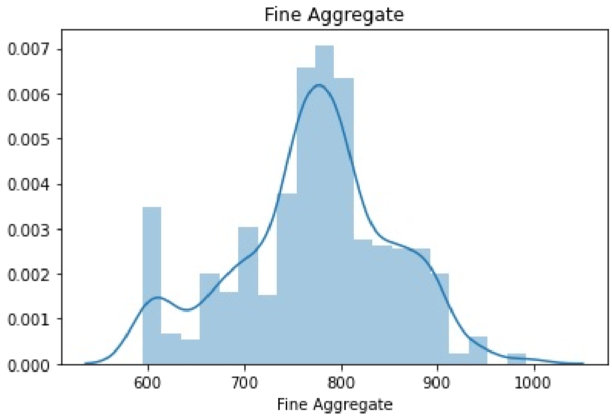

Superplasticizer
Ordered Values
Theoretical quantiles

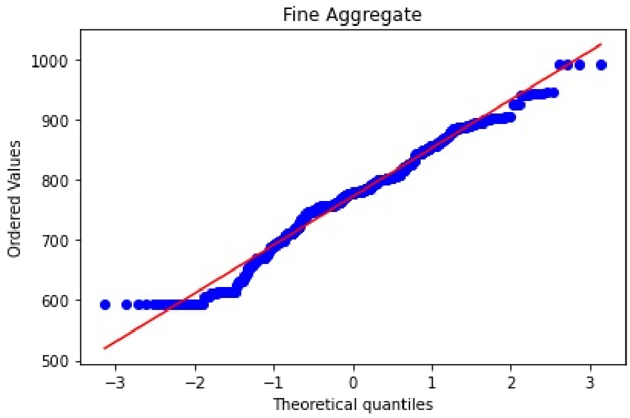

Coarse Aggregate

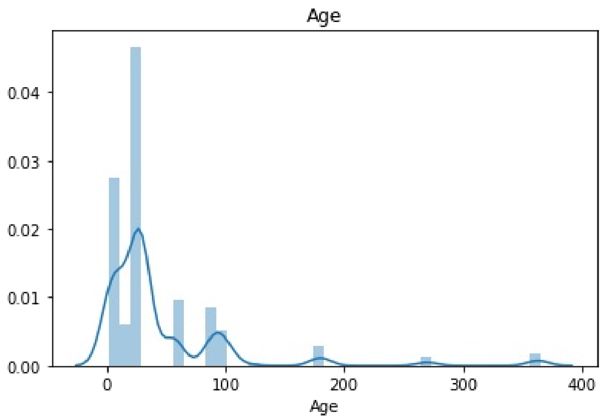

Coarse Aggregate
Ordered Values

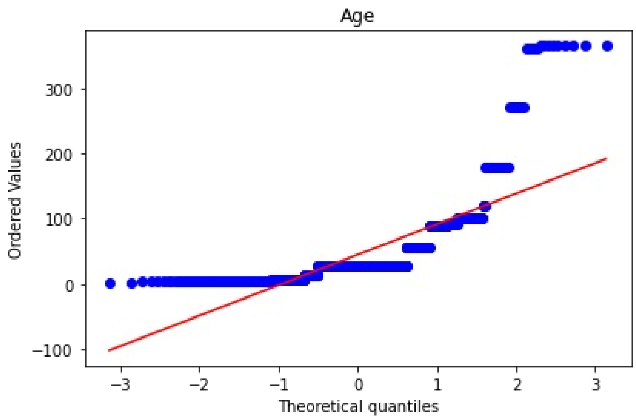












Applying Box-Cox Transform

A Box Cox transformation is a transformation of non-normal dependent variables into a normal shape. Normality is an important assumption for many statistical techniques; if your data isn't normal, applying a Box-Cox means that you are able to run a broader number of tests. At the core of the Box Cox transformation is an exponent, lambda (λ), which varies from -5 to 5. All values of λ are considered and the optimal value for your data is selected; The “optimal value” is the one which results in the best approximation of a normal distribution curve. The transformation of Y has the form:

$$y_i^{(\lambda)} = \begin{cases} \dfrac{y_i^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0, \\ \ln(y_i) & \text{if } \lambda = 0, \end{cases}$$

In [80]:

```
pt = PowerTransformer(method='box-cox')

X_train_transformed = pt.fit_transform(X_train+0.000001)
X_test_transformed = pt.transform(X_test+0.000001)

pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_})
```

Out[80]:

	cols	box_cox_lambdas
0	Cement	0.177025
1	Blast Furnace Slag	0.025093
2	Fly Ash	-0.038970
3	Water	0.772682
4	Superplasticizer	0.098811

5	Coarse Aggregate	1.129813
6	Fine Aggregate	1.782019
7	Age	0.066631

Applying linear regression on transformed data

In [77]:
```
lr = LinearRegression()
lr.fit(X_train_transformed,y_train)

y_pred2 = lr.predict(X_test_transformed)

r2_score(y_test,y_pred2)
```

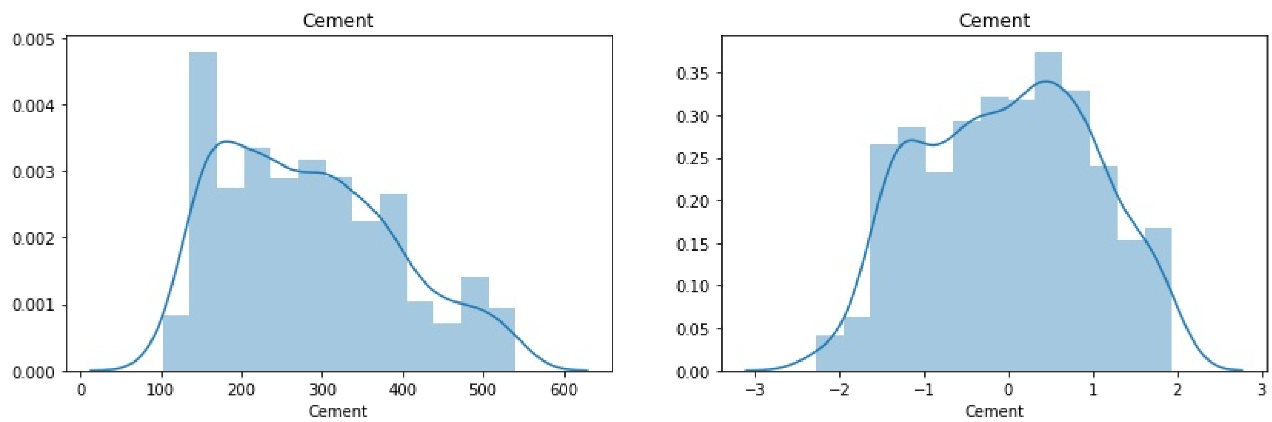
Out[77]:
```
0.8047825006181187
```

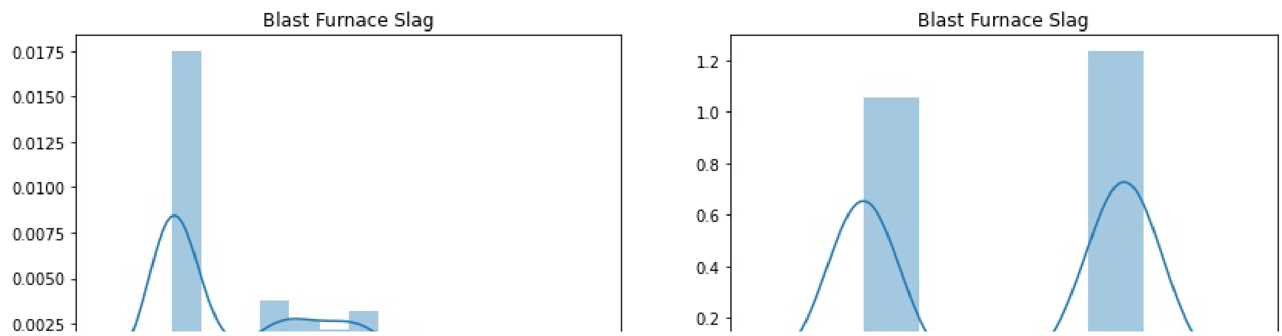
In [78]:
```
# Using cross val score

pt = PowerTransformer(method='box-cox')
X_transformed = pt.fit_transform(X+0.0000001)

lr = LinearRegression()
np.mean(cross_val_score(lr,X_transformed,y,scoring='r2'))
```

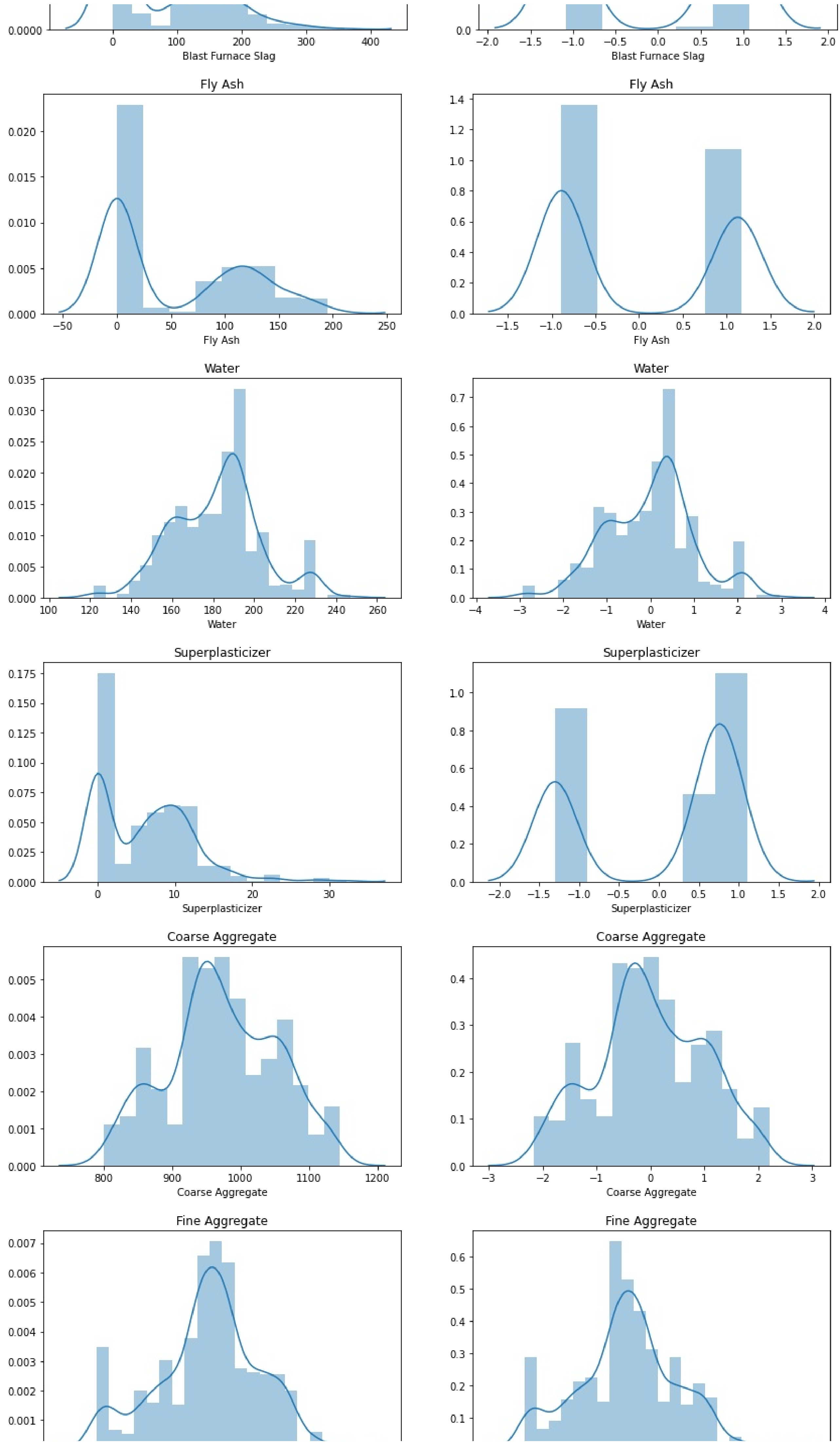
Out[78]:
```
0.6658537942219861
```

Before and after comparision for Box-Cox Plot

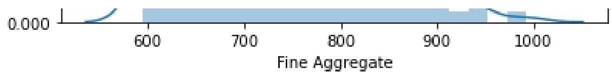
In [81]:
```
X_train_transformed = pd.DataFrame(X_train_transformed,columns=X_train.columns)

for col in X_train_transformed.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed[col])
    plt.title(col)

    plt.show()
```

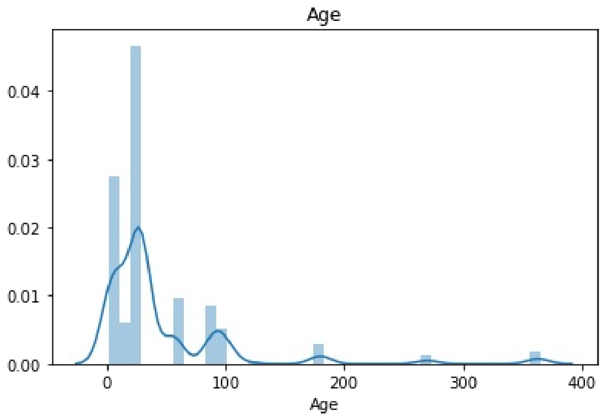


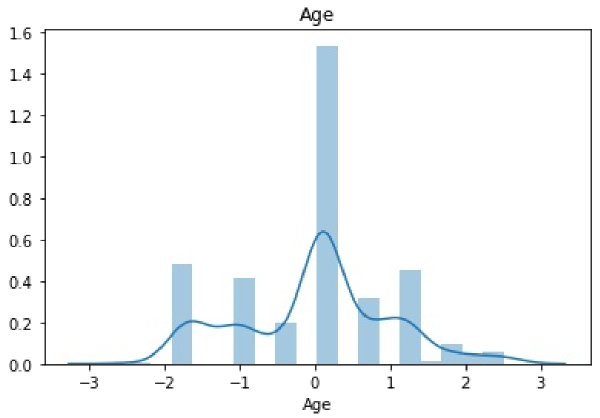




Blast Furnace Slag
Blast Furnace Slag
Fly Ash
Fly Ash
Fly Ash
Fly Ash
Water
Water
Water
Water
Superplasticizer
Superplasticizer
Superplasticizer
Superplasticizer
Coarse Aggregate
Coarse Aggregate
Coarse Aggregate
Coarse Aggregate
Fine Aggregate
Fine Aggregate







Apply Yeo-Johnson transform

This Transformation is somewhat of an adjustment to the Box-Cox transformation by which we can apply it to negative numbers

$$\psi(\lambda, y) = \begin{cases} ((y+1)^\lambda - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y+1)^{2-\lambda} - 1)]/(2-\lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

In [82]:

```
pt1 = PowerTransformer()

X_train_transformed2 = pt1.fit_transform(X_train)
X_test_transformed2 = pt1.transform(X_test)

lr = LinearRegression()
lr.fit(X_train_transformed2,y_train)

y_pred3 = lr.predict(X_test_transformed2)

print(r2_score(y_test,y_pred3))

pd.DataFrame({'cols':X_train.columns,'Yeo_Johnson_lambdas':pt1.lambdas_})
```

```
0.8161906513339305
```

Out[82]:

	cols	Yeo_Johnson_lambdas
0	Cement	0.174348
1	Blast Furnace Slag	0.015715
2	Fly Ash	-0.161447
3	Water	0.771307
4	Superplasticizer	0.253935
5	Coarse Aggregate	1.130050
6	Fine Aggregate	1.783100
7	Age	0.019885

In [83]:

```
# applying cross val score

pt = PowerTransformer()
X_transformed2 = pt.fit_transform(X)

lr = LinearRegression()
```

```
np.mean(cross_val_score(lr,X_transformed2,y,scoring='r2'))
```

Out[83]: 0.6834625134285746

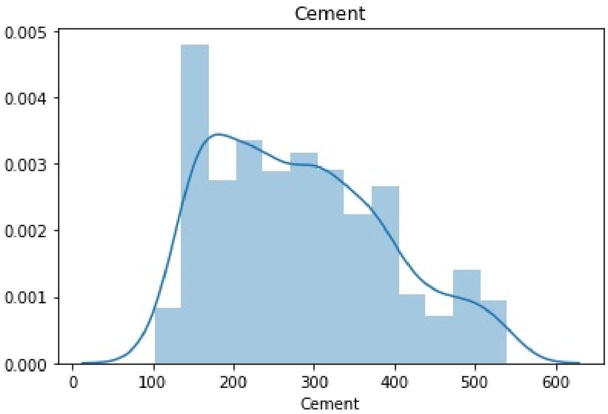
In [84]:

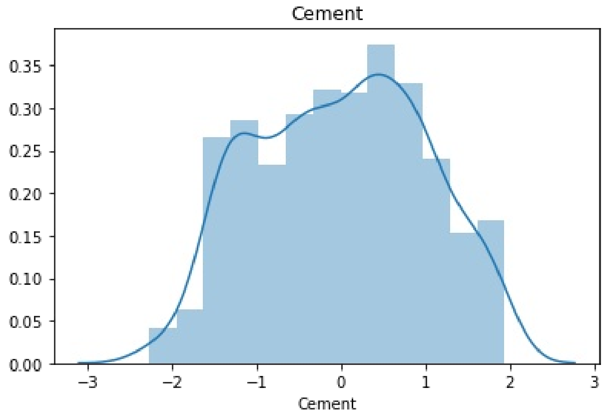
```
X_train_transformed2 = pd.DataFrame(X_train_transformed2,columns=X_train.columns)
```

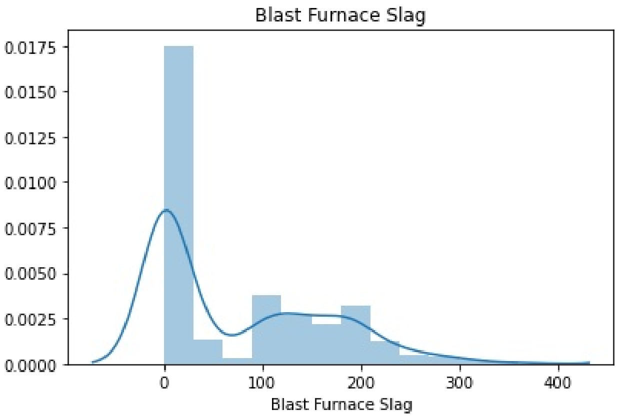
Before and after comparision for Yeo-Johnson

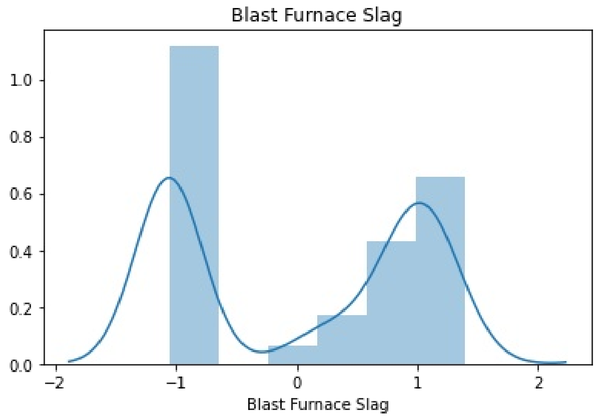
In [85]:

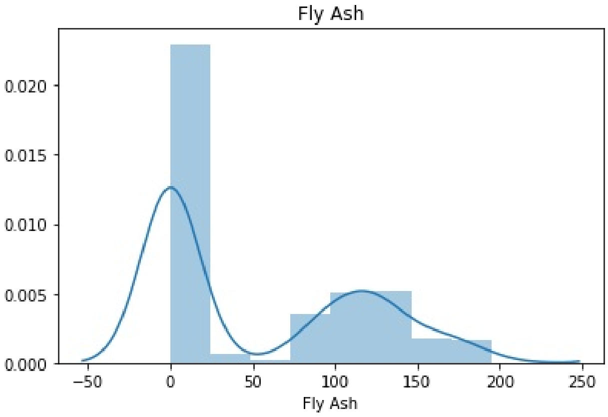
```
for col in X_train_transformed2.columns:
    plt.figure(figsize=(14,4))
    plt.subplot(121)
    sns.distplot(X_train[col])
    plt.title(col)

    plt.subplot(122)
    sns.distplot(X_train_transformed2[col])
    plt.title(col)

    plt.show()
```

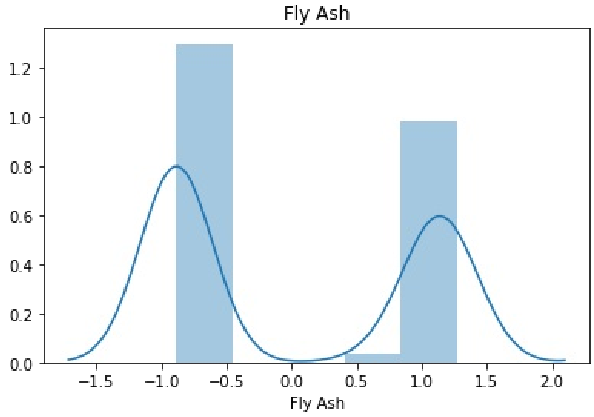


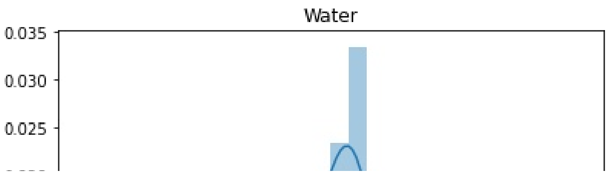


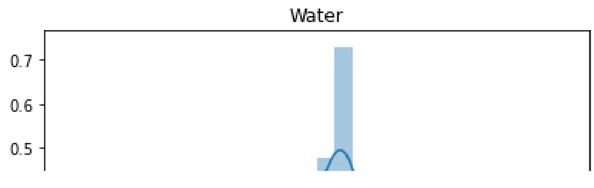


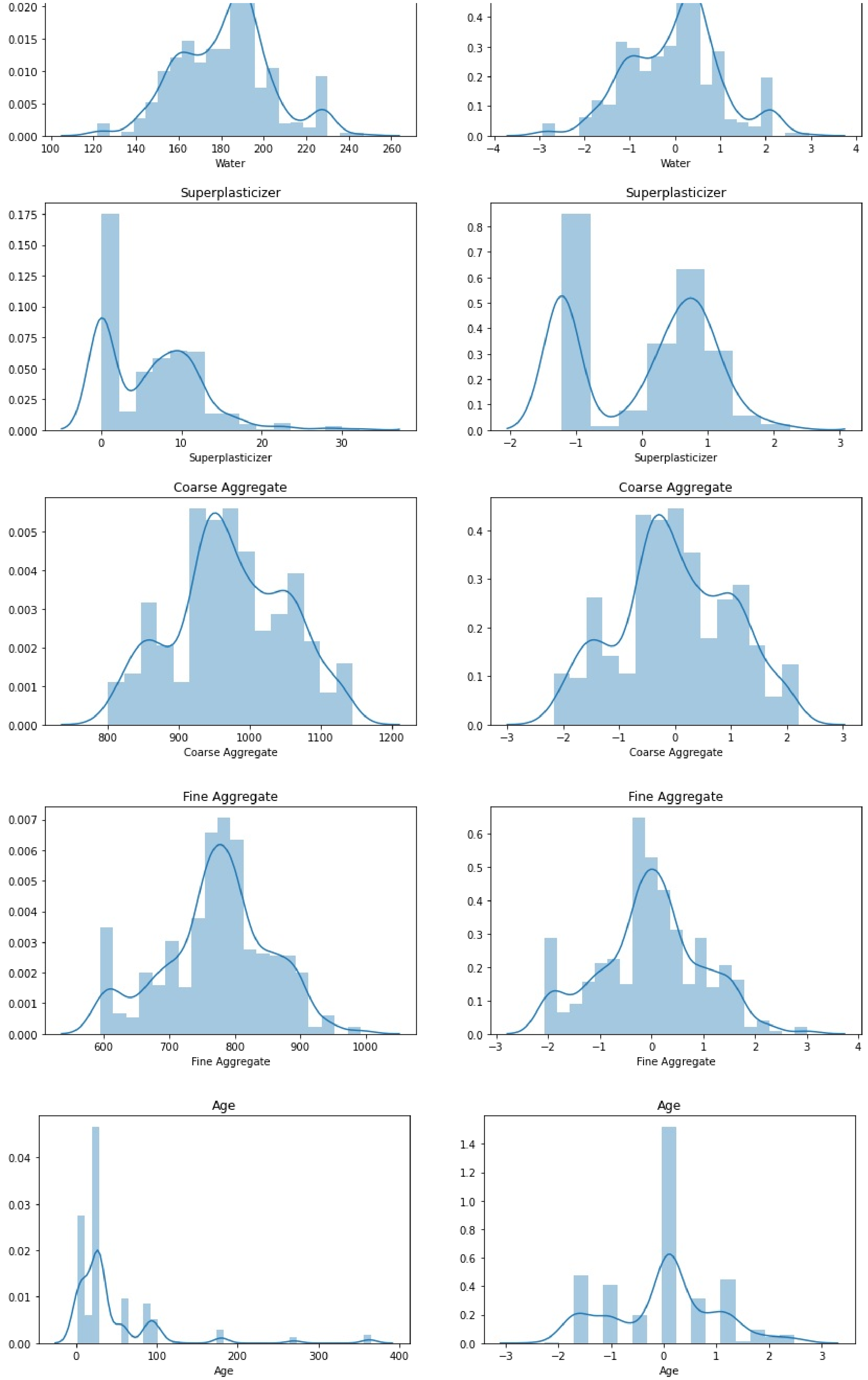












In [87]:

```
# Side by side Lambdas
```

```python
# Side by side Lambdas
pd.DataFrame({'cols':X_train.columns,'box_cox_lambdas':pt.lambdas_,'Yeo_Johnson_lambdas':pt1.lambdas_})
```

Out[87]:

	cols	box_cox_lambdas	Yeo_Johnson_lambdas
0	Cement	0.169544	0.174348
1	Blast Furnace Slag	0.016633	0.015715
2	Fly Ash	-0.136480	-0.161447
3	Water	0.808438	0.771307
4	Superplasticizer	0.264160	0.253935
5	Coarse Aggregate	1.129395	1.130050
6	Fine Aggregate	1.830763	1.783100
7	Age	0.001771	0.019885

Thank you

Author

Muhammad Zaman Ali
