In [1]:
```
import pandas as pd
import numpy as np
```

Read The Data

In [2]:
```
df = pd.read_csv('concrete_data.csv')
```

In [3]:
```
df
```

Out[3]:

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strengtl
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.9
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.2
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.0
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.3
...	...	...	...	...	...	...	...	...	.
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.2
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.1
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.7
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.7
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.4

1030 rows × 9 columns

Information About Data

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   cement                         1030 non-null   float64
 1   blast_furnace_slag             1030 non-null   float64
 2   fly_ash                        1030 non-null   float64
 3   water                          1030 non-null   float64
 4   superplasticizer               1030 non-null   float64
 5   coarse_aggregate               1030 non-null   float64
 6   fine_aggregate                 1030 non-null   float64
 7   age                            1030 non-null   int64  
 8   concrete_compressive_strength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

In [5]:
```
df.describe()
```

Out[5]:

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	conc
count	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	
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980	63.169912	
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000	1.000000	
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000	7.000000	
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000	28.000000	
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000	56.000000	
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000	365.000000	

Check Null Values in the dataset

In [6]:
```
df.isnull().sum()
# Null Values are not present in my dataset.
```

Out[6]:
```
cement                           0
blast_furnace_slag               0
fly_ash                          0
water                            0
superplasticizer                 0
coarse_aggregate                 0
fine_aggregate                   0
age                              0
concrete_compressive_strength    0
dtype: int64
```

Check Duplicacy in the data

In [7]:
```
df.duplicated().sum()
# 25 values duplicate in my dataset
```

Out[7]: 25

In [8]:
```
# These are duplicated values
df[df.duplicated()==True]
```

Out[8]:

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strength
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.40
80	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.40
86	362.6	189.0	0.0	164.9	11.6	944.7	755.8	3	35.30
88	362.6	189.0	0.0	164.9	11.6	944.7	755.8	3	35.30
91	362.6	189.0	0.0	164.9	11.6	944.7	755.8	3	35.30
100	425.0	106.3	0.0	153.5	16.5	852.1	887.1	7	49.20
103	425.0	106.3	0.0	153.5	16.5	852.1	887.1	7	49.20
109	362.6	189.0	0.0	164.9	11.6	944.7	755.8	7	55.90
111	362.6	189.0	0.0	164.9	11.6	944.7	755.8	7	55.90
123	425.0	106.3	0.0	153.5	16.5	852.1	887.1	28	60.29
126	425.0	106.3	0.0	153.5	16.5	852.1	887.1	28	60.29
132	362.6	189.0	0.0	164.9	11.6	944.7	755.8	28	71.30
134	362.6	189.0	0.0	164.9	11.6	944.7	755.8	28	71.30
137	362.6	189.0	0.0	164.9	11.6	944.7	755.8	28	71.30
146	425.0	106.3	0.0	153.5	16.5	852.1	887.1	56	64.30
149	425.0	106.3	0.0	153.5	16.5	852.1	887.1	56	64.30
155	362.6	189.0	0.0	164.9	11.6	944.7	755.8	56	77.30
157	362.6	189.0	0.0	164.9	11.6	944.7	755.8	56	77.30
160	362.6	189.0	0.0	164.9	11.6	944.7	755.8	56	77.30
169	425.0	106.3	0.0	153.5	16.5	852.1	887.1	91	65.20
172	425.0	106.3	0.0	153.5	16.5	852.1	887.1	91	65.20
177	362.6	189.0	0.0	164.9	11.6	944.7	755.8	91	79.30
179	362.6	189.0	0.0	164.9	11.6	944.7	755.8	91	79.30
182	362.6	189.0	0.0	164.9	11.6	944.7	755.8	91	79.30
809	252.0	0.0	0.0	185.0	0.0	1111.0	784.0	28	19.69

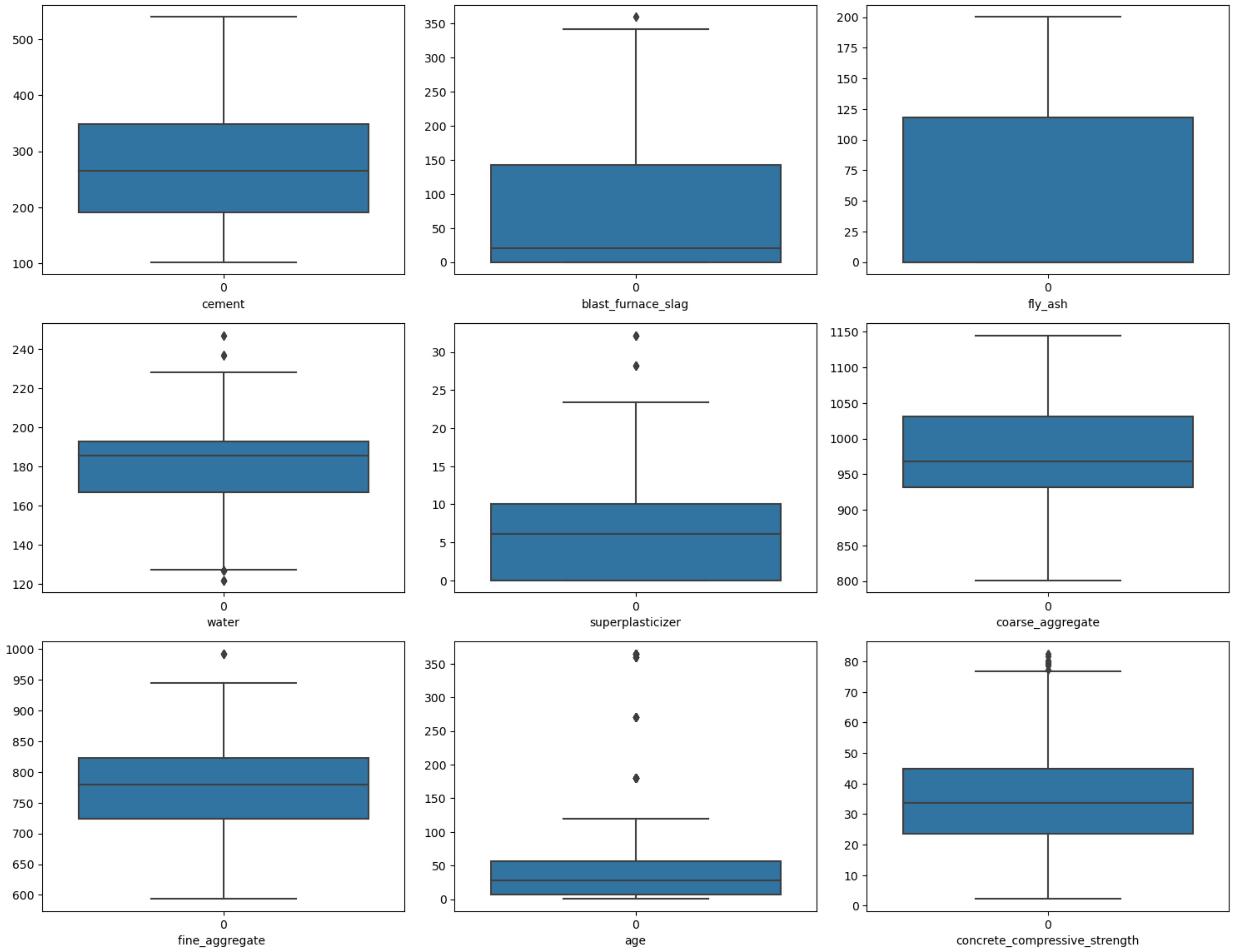
In [9]:
```
# Delete Duplicated Values
df.drop_duplicates(keep = 'first', inplace = True)
df.duplicated().sum()
# Now There are no duplicated value in my dataset
```

Out[9]: 0

In [10]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

Check Outliers

In [11]:
```
# This code is showing Outliers by Boxplot.
plt.figure(figsize = (15,15), facecolor = 'white')
plotnumber = 1
for i in df.columns:
    ax = plt.subplot(4,3, plotnumber)
    sns.boxplot(df[i])
    plt.xlabel(i, fontsize = 10)
    plotnumber +=1
plt.tight_layout()
plt.show()
```



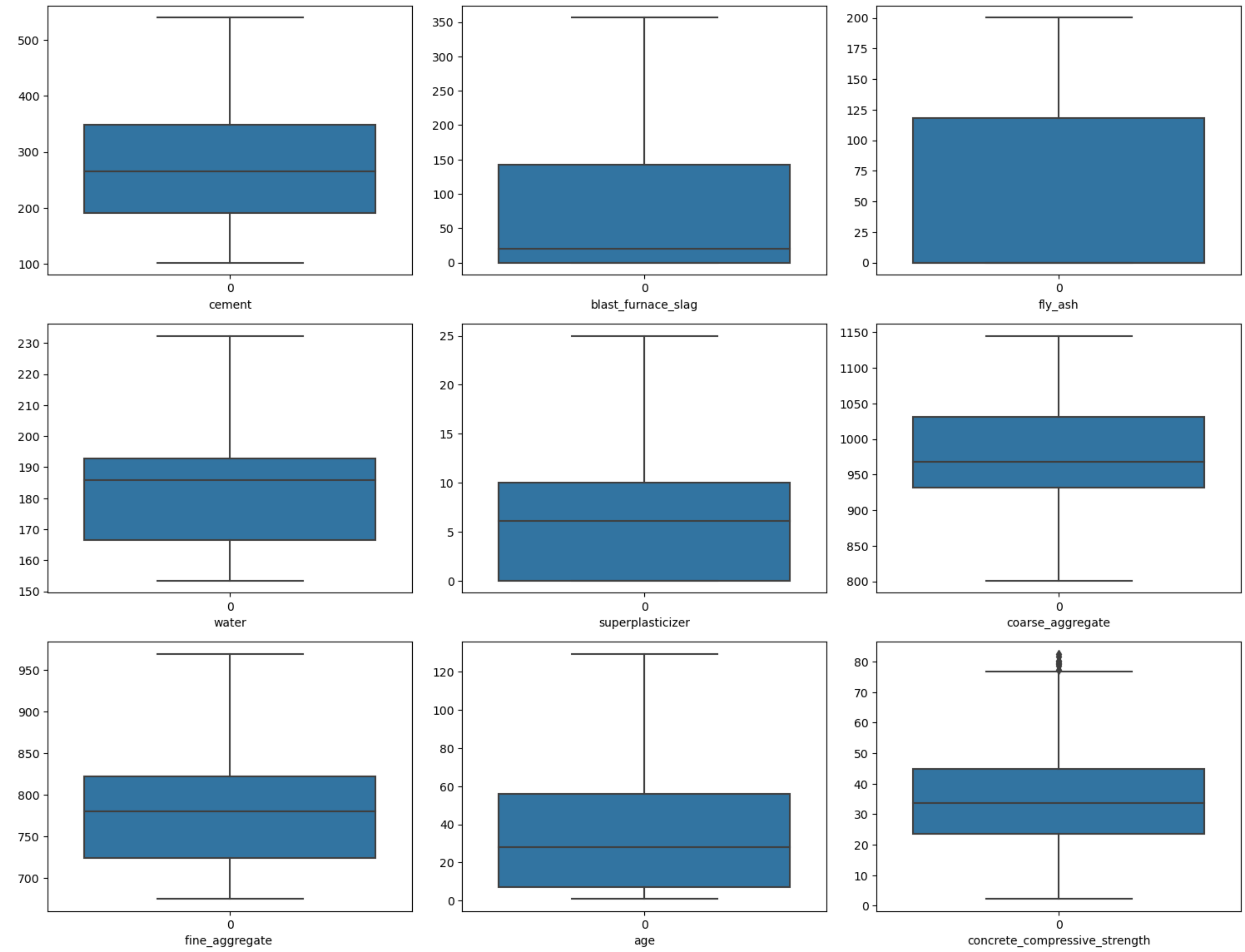
In [12]:
```
# Columns in my dataset
df.columns
```

Out[12]:
```
Index(['cement', 'blast_furnace_slag', 'fly_ash', 'water', 'superplasticizer',
       'coarse_aggregate', 'fine_aggregate ', 'age',
       'concrete_compressive_strength'],
      dtype='object')
```

In [13]:
```
# Columns That have outliers
outliers = ['blast_furnace_slag','water', 'superplasticizer', 'fine_aggregate ', 'age']
```

In [14]:
```
# This Code is doing IQR based filtering
def outlier_capping(dataframe: pd.DataFrame, outliers:list):
    df = dataframe.copy()
    for i in outliers:
        q1 = df[i].quantile(0.25)
        q3 = df[i].quantile(0.75)
        iqr = q3 - q1
        upper_limit = q3 + 1.5 *iqr
        lower_limit = q3 - 1.5 *iqr
        df.loc[df[i] >upper_limit, i] = upper_limit
        df.loc[df[i] <lower_limit, i] = lower_limit
    return df
df = outlier_capping(dataframe = df, outliers = outliers)
```

In [15]:
```
# Now there is no outliers in my dataset
plt.figure(figsize = (15,15), facecolor = 'white')
plotnumber = 1
for i in df.columns:
    ax = plt.subplot(4,3, plotnumber)
    sns.boxplot(df[i])
    plt.xlabel(i, fontsize = 10)
    plotnumber +=1
plt.tight_layout()
plt.show()
```



In [16]:
```
df.sample(10)
```

Out[16]:

	cement	blast_furnace_slag	fly_ash	water	superplasticizer	coarse_aggregate	fine_aggregate	age	concrete_compressive_strength
417	194.7	0.0	100.5	170.2	7.5	998.0	901.80	3.0	12.18
369	218.9	0.0	124.1	158.5	11.3	1078.7	794.90	3.0	15.34
84	323.7	282.8	0.0	183.8	10.3	942.7	675.35	3.0	28.30
743	397.0	0.0	0.0	186.0	0.0	1040.0	734.00	28.0	36.94
538	480.0	0.0	0.0	192.0	0.0	936.2	712.20	7.0	34.57
18	380.0	95.0	0.0	228.0	0.0	932.0	675.35	90.0	40.56
961	336.5	0.0	0.0	181.9	3.4	985.8	816.80	28.0	44.87
819	525.0	0.0	0.0	189.0	0.0	1125.0	675.35	90.0	58.78
855	326.0	166.0	0.0	174.0	9.0	882.0	790.00	28.0	61.23
741	480.0	0.0	0.0	192.0	0.0	936.0	721.00	28.0	43.89

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [18]:
```
X = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [19]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state=42)
```

In [20]:
```
lr = LinearRegression()
```

In [21]:
```
lr.fit(X_train,y_train)
```

Out[21]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [22]:
```
y_pred = lr.predict(X_test)
```

In [23]:
```
R_Square = r2_score(y_test,y_pred)
print("Accuracy of the model without checking assumption of linear regression is",R_Square * 100 ,"percent")
```

```
Accuracy of the model without checking assumption of linear regression is 70.03708219864819 percent
```

Assumptions Of Linear Regression

1.Mean of Residuals

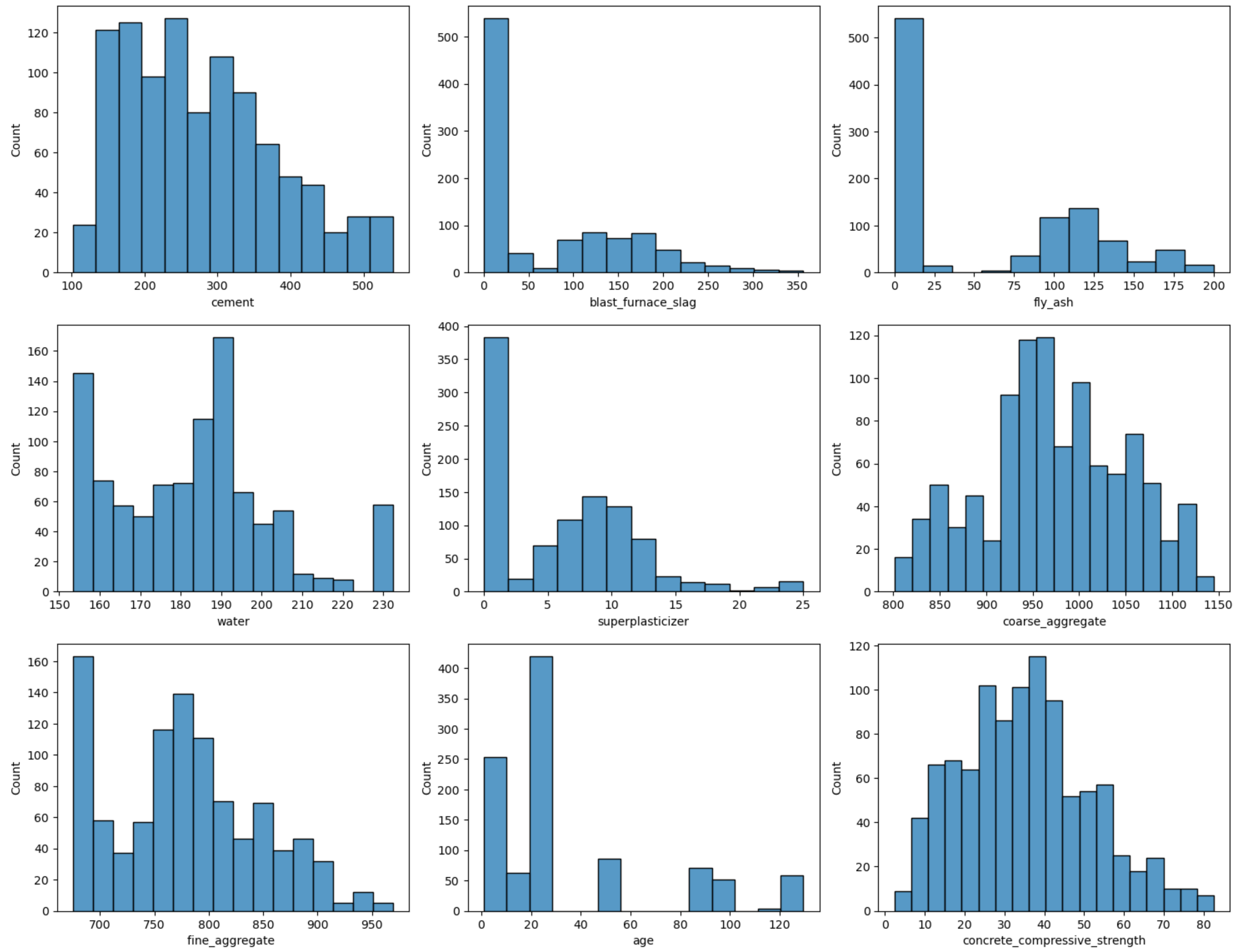
Residuals as we know are the differences between the true value and the predicted value. One of the assumptions of linear regression is that the mean of the residuals should be zero. So let's find out.

In [24]:
```
residuals = y_test.values-y_pred
mean_residuals = np.mean(residuals)
print("Mean of Residuals {}".format(mean_residuals))
```

```
Mean of Residuals 0.7303497906942218
```

2. Independent features are Distributed Normally

In [25]:
```
plt.figure(figsize=(15,15),facecolor='white')
plotnumber = 1
for i in df.columns:
    ax = plt.subplot(4,3,plotnumber)
    sns.histplot(df[i])
    plt.xlabel(i,fontsize=10)
    plotnumber +=1
plt.tight_layout()
plt.show()
print(df.skew())
```

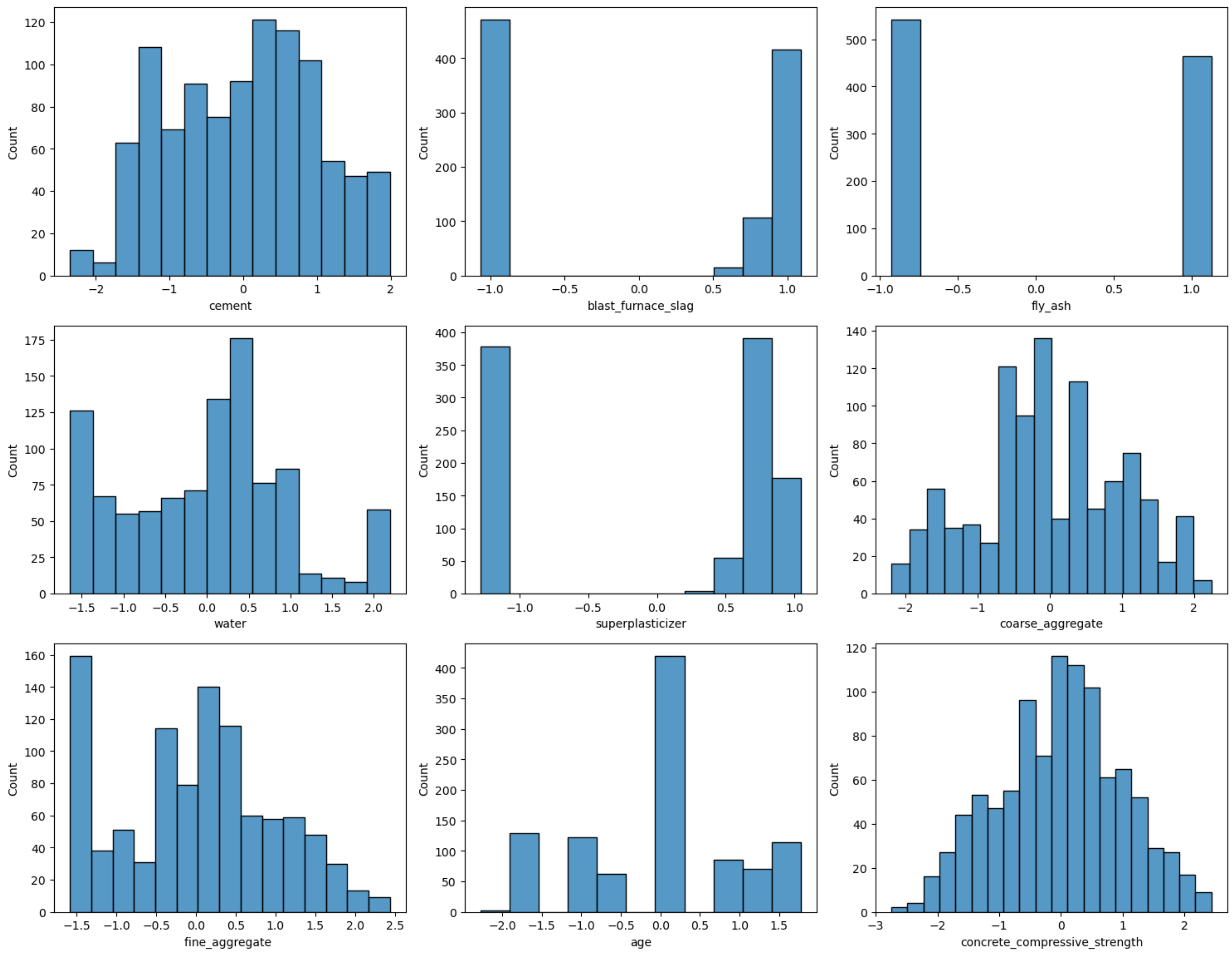



```
cement                           0.564959
blast_furnace_slag               0.853654
fly_ash                          0.497231
water                            0.369428
superplasticizer                 0.708386
coarse_aggregate                -0.065256
fine_aggregate                   0.239544
age                              1.277316
concrete_compressive_strength    0.395696
dtype: float64
```

Apply BOX-COX Method

In [30]:
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer(method='box-cox')
df_trans1 = pd.DataFrame(pt.fit_transform(df+0.000001),columns=df.columns)
```

In [31]:
```
plt.figure(figsize=(15,15),facecolor='white')
plotnumber = 1
for i in df.columns:
    ax = plt.subplot(4,3,plotnumber)
    sns.histplot(df_trans1[i])
    plt.xlabel(i,fontsize=10)
    plotnumber+=1
plt.tight_layout()
plt.show()
print(df_trans1.skew())
```

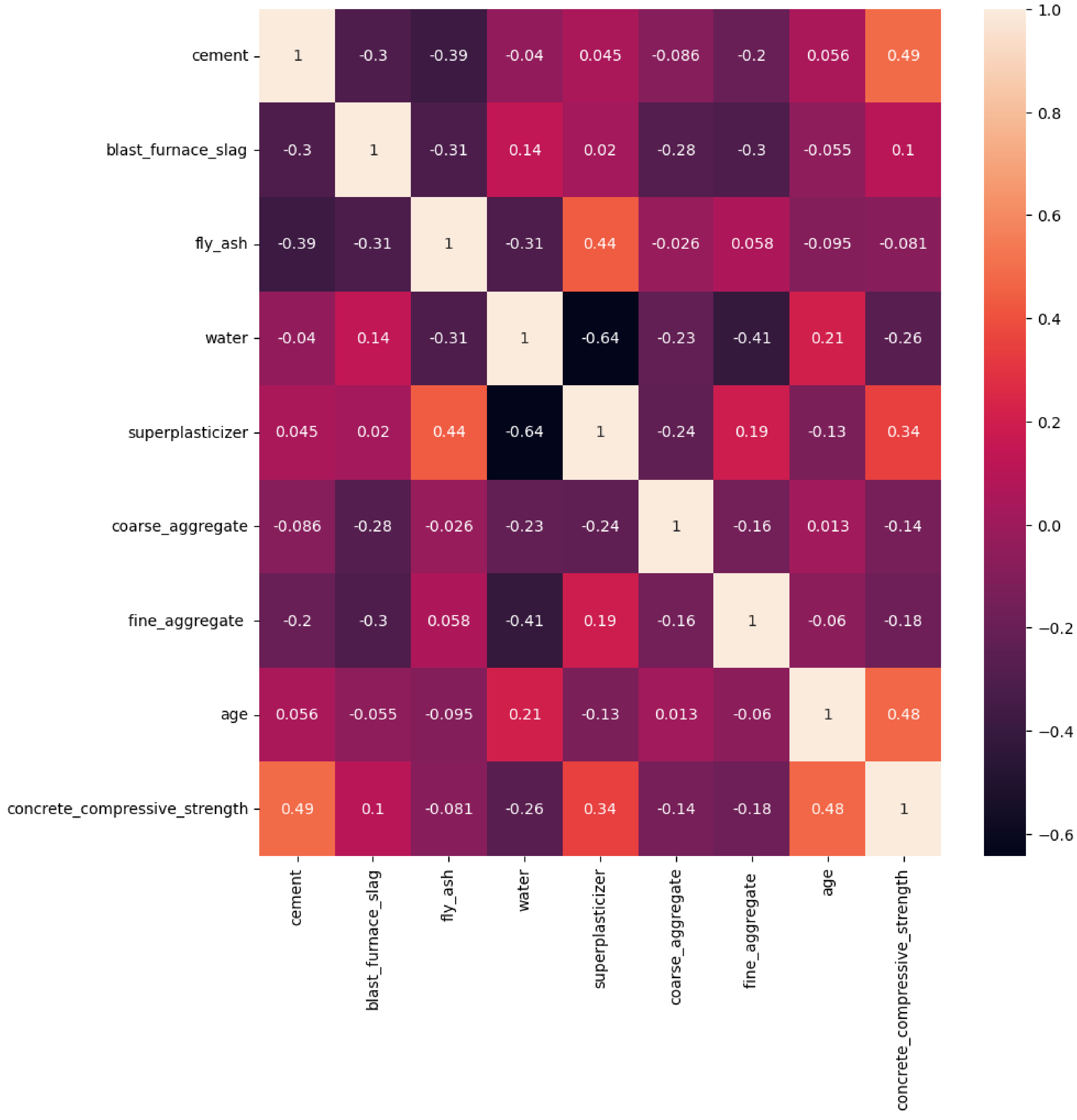



```
cement                          -0.012649
blast_furnace_slag              -0.114416
fly_ash                          0.155235
water                            0.011402
superplasticizer                -0.489456
coarse_aggregate                -0.020292
fine_aggregate                   0.007200
age                             -0.055427
concrete_compressive_strength   -0.059105
dtype: float64
```

3. No Multicolinearity in Data

If the predictors(independent features) are correlated among themselves. then the data is said to have a multicolinearity problem. But why is this a problem? The answer to this question is that high collinearity means that the two variables vary very similarty and conatins the same kind of information. This will leads to redundancy in the dataset. Due to redundancy only the complexity of the model is increases and no new information or pattern is learned by the model.So we generally try to avoid highly correlated features even while using complex model.

In [32]:
```
plt.figure(figsize = (10,10))
sns.heatmap(df.corr(), annot = True)
plt.show()
# here no multicolinearity in data
```



Now build Linear Regression Model using Scikit- Learn

In [33]:
```
# Splitting data in independent features and dependent feature
X1 = df_trans1.iloc[:,:-1]
y1 = df_trans1.iloc[:,-1]
```

In [34]:
```
# Splitting the data in trainning data and test data
X_train1,X_test1,y_train1,y_test1 = train_test_split(X1,y1,test_size=0.2,random_state=42)
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train1_scaled = scaler.fit_transform(X_train1)
X_test1_scaled = scaler.transform(X_test1)
```

In [35]:
```
lrm = LinearRegression()
lrm.fit(X_train1_scaled,y_train1)
y_pred1 = lrm.predict(X_test1_scaled)
R_Square1 = r2_score(y_test1,y_pred1)
print("Accuracy of the model without checking assumption of linear regression is",R_Square * 100 ,"percent")
print("Accuracy of the model after checking assumption of the linear regression model",R_Square1*100,"percent'
```

```
Accuracy of the model without checking assumption of linear regression is 70.03708219864819 percent
Accuracy of the model after checking assumption of the linear regression model 82.83689015078185 percent
```

Making my own linear regression class

In [36]:
```
class myclass:

    def __init__(self):
        self.intercept_ = None
        self.coef_ = None

    def fit(self,X_train1_scaled,y_train1):
        X_train1_scaled = np.insert(X_train1_scaled,0,1,axis=1)
        # Ordinary Least Square Method
        betas = np.linalg.inv(np.dot(X_train1_scaled.T,X_train1_scaled)).dot(X_train1_scaled.T).dot(y_train)
        self.intercept_ = betas[0]
        self.coef_ = betas[1:]

    def predict(self,X_test1_scaled):
        return np.dot(X_test1_scaled,self.coef_) + self.intercept_
```

In [37]:
```
mylr = myclass()
mylr.fit(X_train1,y_train1)
y_pred3 = mylr.predict(X_test1_scaled)
R_Square = r2_score(y_test,y_pred3)
print("Accuracy myclass",R_Square*100,"percent")
```

```
Accuracy myclass 80.09220685860211 percent
```

Making own Batch Gradient Descent class

In [38]:
```
class BGD:

    def __init__(self,learning_rate=0.01,epochs=100):
        self.coef_ = None
        self.intercept_ = None
        self.learning_rate = learning_rate
        self.epochs = epochs

    def fit(self,X_train1_scaled,y_train1):
        self.intercept_ = 0
        self.coef_ = np.ones(X_train1_scaled.shape[1])
        for i in range(self.epochs):
            y_hat = np.dot(X_train1_scaled,self.coef_) + self.intercept_
            intercept_der = -2*np.mean(y_train1 - y_hat)
            self.intercept_ = self.intercept_ - (self.learning_rate * intercept_der)

            coef_der = -2*np.dot((y_train1-y_hat),X_train1_scaled)/X_train1_scaled.shape[0]
            self.coef_ = self.coef_ - (self.learning_rate*coef_der)
        print("Value of intercept:",self.intercept_)
        print("Values of coefficients:",self.coef_)

    def predict(self,X_test1_scaled):
        return np.dot(X_test1_scaled,self.coef_) + self.intercept_

bgd = BGD(learning_rate=0.1,epochs=50)
bgd.fit(X_train1_scaled,y_train1)
y_pred4 = bgd.predict(X_test1_scaled)
print("Accuracy of the model: ",r2_score(y_test1,y_pred4)*100,"percent")
```

```
Value of intercept: -0.008138250266156081
Values of coefficients: [0.7205952  0.50383808 0.08873394 0.09374271 0.34363413 0.26351422
 0.18704693 0.60508769]
Accuracy of the model:  76.64891240400526 percent
```

Stochastic Gradient Class using Scikit-Learn

In [39]:
```
from sklearn.linear_model import SGDRegressor
sgd = SGDRegressor()
sgd.fit(X_train1_scaled,y_train1)
y_pred5 = sgd.predict(X_test1_scaled)
r2_score(y_test,y_pred5)
```

Out[39]:
```
-4.255334224923953
```

Making My Own Stochastic Gradient Class

In [40]:
```
import random
class SGDRegressor:

    def __init__(self,learning_rate=0.01,epochs=100):

        self.coef_ = None
        self.intercept_ = None
        self.lr = learning_rate
        self.epochs = epochs

    def fit(self,X_train1_scaled,y_train1):
        # init your coefs
        self.intercept_ = 0
        self.coef_ = np.ones(X_train1_scaled.shape[1])

        for i in range(self.epochs):
            for j in range(X_train1_scaled.shape[0]):
                idx = np.random.randint(0,X_train1_scaled.shape[0])

                y_hat = np.dot(X_train1_scaled[idx],self.coef_) + self.intercept_

                intercept_der = -2 * (y_train1[idx] - y_hat)
                self.intercept_ = self.intercept_ - (self.lr * intercept_der)

                coef_der = -2 * np.dot((y_train1[idx] - y_hat),X_train1_scaled[idx])
                self.coef_ = self.coef_ - (self.lr * coef_der)

        print(self.intercept_,self.coef_)

    def predict(self,X_test1_scaled):
        return np.dot(X_test1_scaled,self.coef_) + self.intercept_
```

Conclusion

1. Check Duplicated values and handle it

25 duplicated values in this dataset

2. Detect Outliers

Detect outlier by Box plot.

Outliers present in this dataset .So I fix outlies by Interquartile Range method

Interquartile Range (IQR): IQR identifies outliers as data points falling outside the range defined by Q1-k*(Q3- Q1) and Q3+k*(Q3-Q1), where Q1 and Q3 are the first and third quartiles, and k is a factor (typically 1.5).

3. Independent features are not normally distributed in this dataset. Therefore, Box-Cox method was applied to find independent features that are normally distributed.

4. Feature Scaling

Feature Scaling is a technique to standardize the independent features present in the data in a fixed range. It is performed during the data pre-processing to handle highly varying magnitudes or values or units. If feature scaling is not done, then a machine learning algorithm tends to weigh greater values, higher and consider smaller values as the lower values, regardless of the unit of the values.

When I applied linear regression algorithm without any correction in dataset, then the accuracy of my model was 70 percent. But when I did the points given above, the accuracy of my model went up to 80 percent.

In [47]:
```
print("Accuracy of the model without checking assumption of linear regression is",R_Square * 100 ,"percent")
print("Accuracy of the model after checking assumption of the linear regression model",R_Square1*100,"percent'
```

```
Accuracy of the model without checking assumption of linear regression is 80.09220685860211 percent
Accuracy of the model after checking assumption of the linear regression model 82.83689015078185 percent
```

In [ ]: