In [1]:
```
# Importing essential libraries
import numpy as np
import pandas as pd
import matplotlib .pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings ("ignore")
```

In [2]:
```
#importing dataset using pandas

concrete_df = pd.read_csv("Concrete Compressive Strength.csv")
```

In [3]:
```
concrete_df
```

Out[3]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	streng
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.9861
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8873
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.2695
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.0527
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.2960
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.2843
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.1787
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.6966
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.7680
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.4012

1030 rows × 9 columns

In [4]:
```
# Separating dependent and independent variable.
X_raw = concrete_df.drop(columns=['strength '], axis=1)
Y = concrete_df['strength ']
```

In [5]:
```
# Preparing the data to fit Linear regression
def Prepare_data(X):
    X.columns = ['X_1', 'X_2', 'X_3', 'X_4', 'X_5', 'X_6', 'X_7', 'X_8']
    X_0 = pd.DataFrame({'X_0' : [1]*1030})
    df = pd.concat([X_0,X],axis = 1, join = 'inner')
    arr=np.array(df)
    return df,arr
```

In [6]:
```
df, X = Prepare_data(X_raw)
print("____________________________________The prepared data is____________
df
```

```
____________________________________The prepared data is__________________
_____________________________________
```

Out[6]:

	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
0	1	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	1	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	1	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	1	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360
...	...	...	...	...	...	...	...	...	...
1025	1	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28
1026	1	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28
1027	1	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28
1028	1	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28
1029	1	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28

1030 rows × 9 columns

In [7]:
```
# function to estimate parameters
def Parameter_est( X ,y):
    Transpose = X.T
    mal = np.matmul(X.T,X)
    inv = np.linalg.inv(mal)
    b_hat = np.matmul(np.matmul(inv,X.T),y)
    return b_hat

# Assign the estimated parameters to b_hat
b_hat = Parameter_est( X ,Y)
print("_____________________________________The weight parameters are______
b_hat
```

```
_____________________________________The weight parameters are____________
_______________________________________________
```

Out[7]:
```
array([-2.31637558e+01,  1.19785255e-01,  1.03847249e-01,  8.79430817e-02,
       -1.50297904e-01,  2.90686943e-01,  1.80301836e-02,  2.01544557e-02,
        1.14225620e-01])
```

In [8]:
```
# Function the Linear Regression
def linearRegression(beta):
    print ( "___________________________________The fitted equation is_____
    print (f"y_hat={b_hat[0]:.3f}X_0+{b_hat[1]:.3f}X_1+{b_hat[2]:.3f}X_2+{b
    print ( "______________________________________________________________

# Fitting the equation:
linearRegression(b_hat)


# predicting the dependent variable using the fitted equation
y_hat = X@b_hat
print("________________________________________The Predicted Values are____
print(y_hat)
```

```
___________________________________The fitted equation is_________________
___________________________________________
y_hat=-23.164X_0+0.120X_1+0.104X_2+0.088X_3+-0.150X_4+0.291X_5+0.018X_6+0.
020X_7+0.114X_8}
__________________________________________________________________________
____________________________________________
________________________________________The Predicted Values are__________
________________________________________________
[53.4728591  53.74331185 56.81194746 ... 26.47099254 29.11564722
 31.89398622]
```

In [9]:
```
#Error calculation:
def Errors(y_true, y_pred):
    e = y_true - y_pred
    mse = ((e**2).sum())/X.shape[0]
    return mse
mse = Errors(Y,y_hat)

print("Mean squared error is: ", mse)
rmse = np.sqrt(mse)
print("Root mean squared error is: {}".format(rmse))
```

```
Mean squared error is:  107.21180273479736
Root mean squared error is: 10.354313243030527
```

In [10]:
```
# Function to create ANOVA Table
def Anova(y_true,y_pred):
    # Total variation
    SSt = ((y_true-y_true.mean())**2).sum()
    degree_t = X.shape[0] - 1
    # Residual variation
    SSres= ((y_true - y_pred)**2).sum()
    degree_res = X.shape[0] - X.shape[1]
    MSres = SSres/degree_res
    # variation due to regression
    SSreg = SSt-SSres
    degree_reg = X.shape[1]-1
    MSreg= SSreg/degree_reg
    F = MSreg/MSres
    return degree_res,degree_reg, SSres,SSreg,MSres,MSreg, F

degree_res, degree_reg, SSres,SSreg,MSres,MSreg, F = Anova(Y,y_hat)
```

In [11]:
```
A Table
DF':[degree_reg, degree_res, degree_reg+degree_res], 'SS':[SSreg, SSres, SSr
ataFrame(anova_dict,index=["Regression","Residual","Total"])
______________________________________________ANOVA Table_________________________
```

____ANOVA Table______________

____

Out[11]:

	DF	SS	MS	F
Regression	8	176744.871659	22093.108957	204.269137
Residual	1021	110428.156817	108.156863	204.269137
Total	1029	287173.028476	22201.265820	204.269137

In [12]:
```
# Testing Null hypothesis
print("______________________________________________________________________
print(f"H0: b0=b1=...........bk-1=0 against \nH1: bi!=0 for i=0 to k-1")
print("If F>F{alpha, k-1, n-k}, The H0 is rejected")
print("______________________________________________________________________
import scipy.stats
from scipy.stats import f
alpha = 0.05
q = 1 - alpha
f = f.ppf(q, degree_reg, degree_res)
print(f"The calcylated f value is: {f}" )
print(f"The observed f value is {F}")
if(abs(F)>f):
        print("The null hypothesis H0 is rejected")
else:
        print("The null hypothesis H0 is accepted")
```

```
__________________________________________________________________________
______________________________________________
H0: b0=b1=...........bk-1=0 against
H1: bi!=0 for i=0 to k-1
If F>F{alpha, k-1, n-k}, The H0 is rejected
__________________________________________________________________________
_______________________________________________
The calcylated f value is: 1.9474558084667661
The observed f value is 204.2691365659187
The null hypothesis H0 is rejected
```

In [13]:
```
# Predict R^2 value and adjusted R^2 value:
R_sq = (SSreg/ (SSres+SSreg) ) * 100
AdjR_sq = (1- MSres/(MSres+MSreg)) * 100
print(f"The R square value is:{R_sq}")
print(f"Adjusted R square value is: {AdjR_sq}")
```

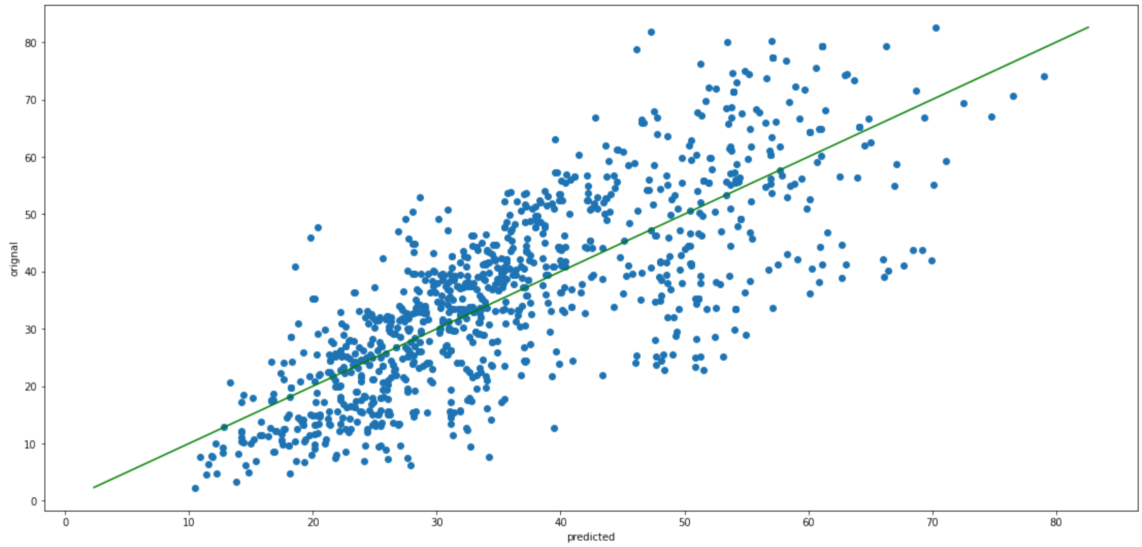
```
The R square value is:61.54647342657952
Adjusted R square value is: 99.51283470241574
```

In [14]:
```
# Test on individual regression coefficient (Partial test or Marginat Test)
Corr = np.linalg.inv(X.T@X)
from scipy.stats import t
def Marginal_test(beta, C, MSres,X):
    n = X.shape[1]
    for i in range(n):
        T = beta[i]/np.sqrt(MSres*C[i][i])
        if(abs(T)>t.ppf(1-0.05, degree_res)):
            print(f"H0:b{i}=0 is rejected")
        else:
            print(f"HØ:b{i}=0 is accepted")

Marginal_test(b_hat,Corr,MSres,X)
```

```
HØ:b0=0 is accepted
H0:b1=0 is rejected
H0:b2=0 is rejected
H0:b3=0 is rejected
H0:b4=0 is rejected
H0:b5=0 is rejected
H0:b6=0 is rejected
H0:b7=0 is rejected
H0:b8=0 is rejected
```

In [15]:
```
# Plot the regression Line fitted by the function made
plt.figure(figsize=[19 , 9])
plt.scatter( y_hat, Y)
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], color = 'green')
plt.xlabel('predicted')
plt.ylabel('orignal')
plt.show()
```

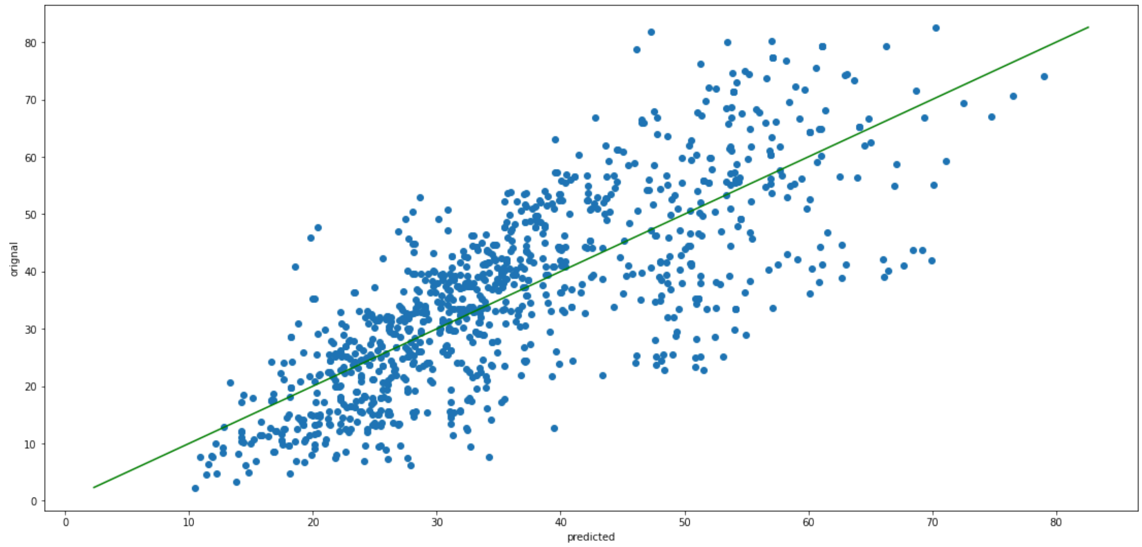


In [16]:
```
# trying the inbuilt regression functionn
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
lr = LinearRegression()
fit = lr.fit(X,Y)
print( '.......................................................................
y_predict = lr.predict(X)
print( 'mean_ squred_error is ==',mean_squared_error(Y, y_predict))
rms = np.sqrt(mean_squared_error(Y,y_predict))
print( 'root mean squared error is == {}'.format(rms))
print("________________________________________________The predicted values
y_predict
```

```
.......................................................................
mean_ squred_error is == 107.21180273479737
root mean squared error is == 10.354313243030528
________________________________________________The predicted values are__
_____________________________________________
```

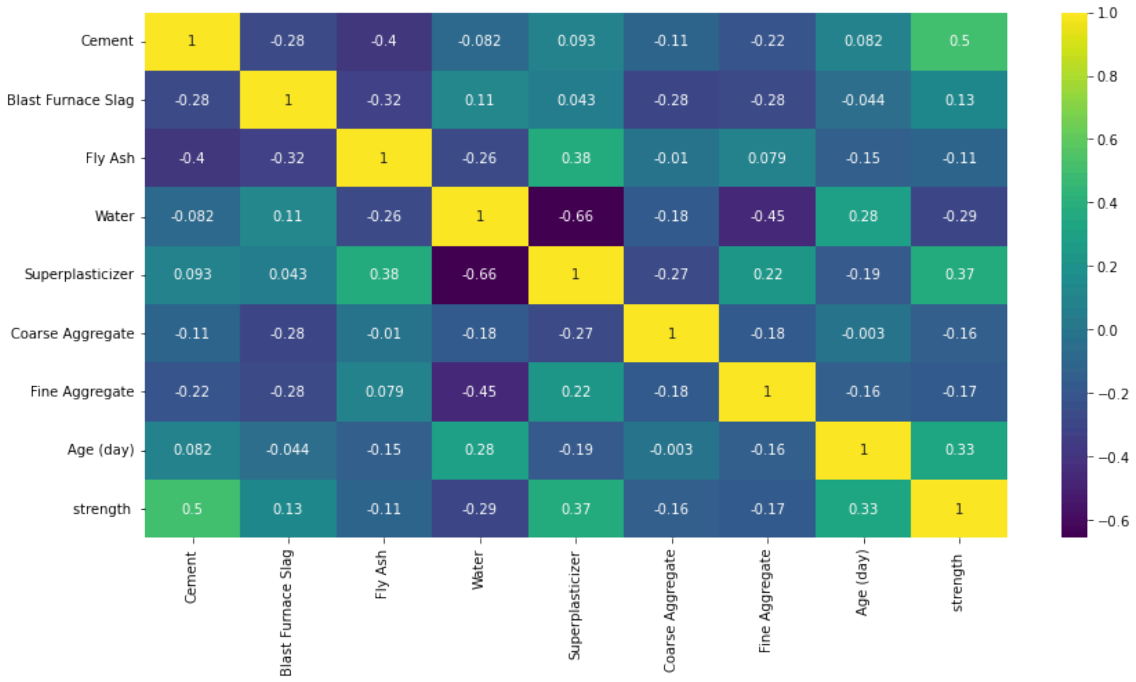
Out[16]:
```
array([53.4728591 , 53.74331185, 56.81194746, ..., 26.47099254,
       29.11564722, 31.89398622])
```

In [17]:
```
# Plot the regression Line fitted by the inbuilt libraries in python
plt.figure(figsize=[19 , 9])
plt.scatter( y_predict, Y)
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], color = 'green')
plt.xlabel('predicted')
plt.ylabel('orignal')
plt.show()
```



In [18]:
```
# Plot heatmap to check multicollinearity
plt.figure(figsize = (14,7))
sns.heatmap(concrete_df.corr( ), annot=True, cmap='viridis')
```

Out[18]: <AxesSubplot:>



In [19]:
```
# check for any duplicate values in the data
duplicates = concrete_df.duplicated()
concrete_df[duplicates]
duplicates.value_counts( )
```

Out[19]:
```
False    1005
True       25
dtype: int64
```

In [20]:
```
# Dropping duplicate values.
concrete_df=concrete_df.drop_duplicates()
concrete_df
```

Out[20]:

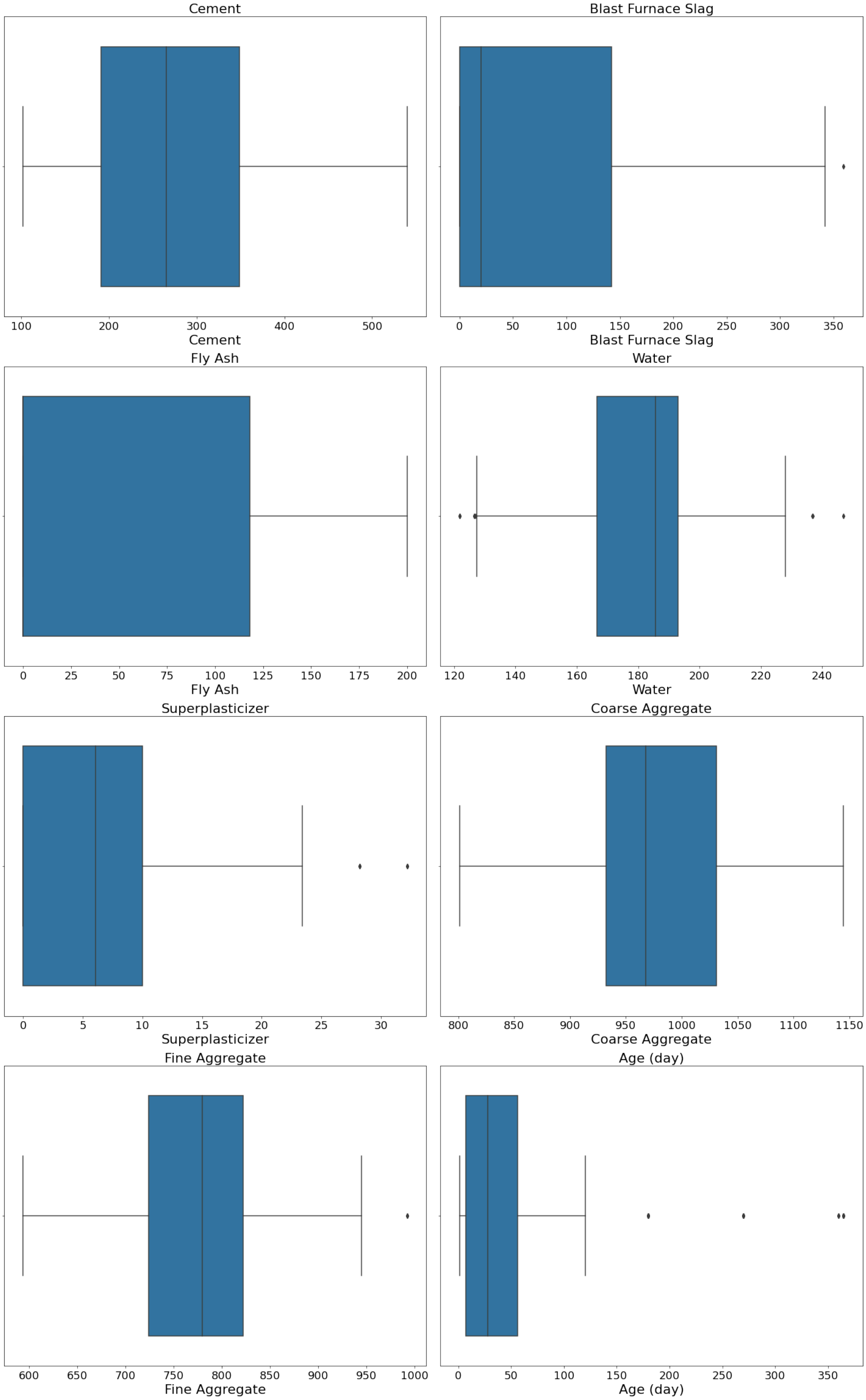
	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	streng
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.9861
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8873
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.2695
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.0527
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.2960
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.2843
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.1787
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.6966
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.7680
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.4012

1005 rows × 9 columns

In [21]:
```
fig, axes = plt.subplots(nrows=len(concrete_df.iloc[:,:-1].columns)//2, nco
axes = axes.flatten()

for i, column in enumerate(concrete_df.iloc[:,:-1].columns):
    sns.boxplot(concrete_df[column], ax=axes[i])
    axes[i].set_title(column, fontsize = 22)
    axes[i].tick_params(axis='x', labelsize=18)
    axes[i].set_xlabel(column, fontsize=22)

plt.tight_layout()
plt.show()
```


Cement
100 200 300 400 500
Cement
Blast Furnace Slag
0 50 100 150 200 250 300 350
Blast Furnace Slag
Fly Ash
0 25 50 75 100 125 150 175 200
Fly Ash
Water
120 140 160 180 200 220 240
Water
Superplasticizer
0 5 10 15 20 25 30
Superplasticizer
Coarse Aggregate
800 850 900 950 1000 1050 1100 1150
Coarse Aggregate
Fine Aggregate
600 650 700 750 800 850 900 950 1000
Fine Aggregate
Age (day)
0 50 100 150 200 250 300 350
Age (day)

In [ ]:

In [22]:
```
def remove_outlier(col):
    col_sorted = sorted(col, reverse=True)
    Q1, Q3 = pd.Series(col_sorted).quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_range = Q1 - (1.5 * IQR)
    upper_range = Q3 + (1.5 * IQR)
    return lower_range, upper_range

for i in concrete_df.columns:
    l_r, u_r = remove_outlier(concrete_df[i])
    concrete_df[i].loc[~concrete_df[i].between(l_r, u_r)] = pd.NA

concrete_df
```

Out[22]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	streng
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	<N
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8873
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	<NA>	40.2695
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	<NA>	41.052
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	<NA>	44.2960
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.2843
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.1787
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.6966
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.7680
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.4012

1005 rows × 9 columns

In [23]:
```
rows_with_nan = concrete_df[concrete_df.isna().any(axis=1)]

rows_with_nan
```

Out[23]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	streng
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	<N
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	<NA>	40.2695
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	<NA>	41.052
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	<NA>	44.2960
6	380.0	95.0	0.0	228.0	0.0	932.0	594.0	<NA>	43.6982
...	...	...	...	...	...	...	...	...	
862	140.0	164.0	128.0	<NA>	6.0	869.0	656.0	28	35.2253
873	237.0	92.0	71.0	<NA>	6.0	853.0	695.0	28	28.6270
908	313.0	145.0	0.0	<NA>	8.0	1000.0	822.0	28	44.5194
936	236.9	91.7	71.5	<NA>	6.0	852.9	695.4	28	28.6298
1019	139.7	163.9	127.7	<NA>	5.8	868.6	655.6	28	35.2253

94 rows × 9 columns

In [24]:
```
concrete_df.dropna(inplace=True)
concrete_df
```

Out[24]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	streng
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.8873
5	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90	47.0298
7	380.0	95.0	0.0	228.0	0.0	932.0	594.0	28	36.447
8	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.8542
9	475.0	0.0	0.0	228.0	0.0	932.0	594.0	28	39.289
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	44.2843
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	31.1787
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	23.6966
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	32.7680
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	32.4012

911 rows × 9 columns

In [25]:
```
# Separating dependent and independent variable.
X_raw = concrete_df.drop(columns=['strength '], axis=1)
Y = concrete_df['strength ']
```

In [26]:
```
X_raw
```

Out[26]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
5	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90
7	380.0	95.0	0.0	228.0	0.0	932.0	594.0	28
8	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28
9	475.0	0.0	0.0	228.0	0.0	932.0	594.0	28
...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28

911 rows × 8 columns

In [27]:
```
arr = np.array(X_raw)
```

In [28]:
```
print("________________________________The prepared data is____________
X_raw
```

```
________________________________The prepared data is________________
_________________________________
```

Out[28]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
5	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90
7	380.0	95.0	0.0	228.0	0.0	932.0	594.0	28
8	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28
9	475.0	0.0	0.0	228.0	0.0	932.0	594.0	28
...	...	...	...	...	...	...	...	...
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28

911 rows × 8 columns

In [29]:
```
X_raw = X_raw.astype(float)
X = X_raw
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 911 entries, 1 to 1029
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Cement              911 non-null    float64
 1   Blast Furnace Slag  911 non-null    float64
 2   Fly Ash             911 non-null    float64
 3   Water               911 non-null    float64
 4   Superplasticizer    911 non-null    float64
 5   Coarse Aggregate    911 non-null    float64
 6   Fine Aggregate      911 non-null    float64
 7   Age (day)           911 non-null    float64
dtypes: float64(8)
memory usage: 64.1 KB
```

In [30]:

```
# function to estimate parameters
def Parameter_est( X ,y):
    Transpose = X.T
    mal = np.matmul(X.T,X)
    inv = np.linalg.inv(mal)
    b_hat = np.matmul(np.matmul(inv,X.T),y)
    return b_hat

# Assign the estimated parameters to b_hat
b_hat = Parameter_est( X ,Y)
print("_____________________________________The weight parameters are______
b_hat
```

```
_____________________________________The weight parameters are____________
______________________________________________
```

Out[30]:

```
0    0.114994
1    0.090383
2    0.067183
3   -0.176913
4    0.266261
5    0.005134
6    0.010068
7    0.307632
dtype: object
```

In [31]:
```
# Function the Linear Regression
def linearRegression(beta):
    print ( "___________________________________The fitted equation is_____
    print (f"y_hat={b_hat[0]:.3f}X_1+{b_hat[1]:.3f}X_2+{b_hat[2]:.3f}X_3+{b
    print ( "______________________________________________________________

# Fitting the equation:
linearRegression(b_hat)


# predicting the dependent variable using the fitted equation
b_hat = b_hat.values
y_hat = X_raw@b_hat
print("________________________________________The Predicted Values are____
print(y_hat)
```

```
___________________________________The fitted equation is_________________
___________________________________________

y_hat=0.115X_1+0.090X_2+0.067X_3+-0.177X_4+0.266X_5+0.005X_6+0.010X_7+0.30
8X_8}
__________________________________________________________________________
____________________________________________
________________________________________The Predicted Values are__________
________________________________________________
1        54.938918
5          39.7736
7        31.327257
8        20.700388
9        33.665311
           ...
1025     39.747941
1026     33.914072
1027     25.553784
1028     28.750184
1029     31.866135
Length: 911, dtype: object
```

In [32]:
```
#Error calculation:
def Errors(y_true, y_pred):
    e = y_true - y_pred
    mse = ((e**2).sum())/X.shape[0]
    return mse
mse = Errors(Y,y_hat)

print("Mean squared error is: ", mse)
rmse = np.sqrt(mse)
print("Root mean squared error is: {}".format(rmse))
```

```
Mean squared error is:  60.91449792340837
Root mean squared error is: 7.804774046915668
```

In [33]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
poly = PolynomialFeatures ( degree=3, interaction_only=False , include_bias
x = poly.fit_transform(X)
poly_clf = linear_model.LinearRegression()
poly_clf.fit (x, Y)
print(poly_clf.score(x,Y))
```

```
0.9127489832026829
```

In [34]:
```
print( '.........................................................................
y_predict = poly_clf.predict(x)
print( 'mean_squared_error is ==' , mean_squared_error(Y,y_predict) )
rms = np.sqrt(mean_squared_error(Y,y_predict))
print( 'root mean squared error is == {} '.format(rms))
```

```
........................................................................
mean_squared_error is == 21.901554707092167
root mean squared error is == 4.679909690057295
```

In [35]:
```
# Function to create ANOVA Table
def Anova(y_true,y_pred):
    # Total variation
    SSt = ((y_true-y_true.mean())**2).sum()
    degree_t = X.shape[0] - 1
    # Residual variation
    SSres= ((y_true - y_pred)**2).sum()
    degree_res = X.shape[0] - X.shape[1]
    MSres = SSres/degree_res
    # variation due to regression
    SSreg = SSt-SSres
    degree_reg = X.shape[1]-1
    MSreg= SSreg/degree_reg
    F = MSreg/MSres
    return degree_res,degree_reg, SSres,SSreg,MSres,MSreg, F

degree_res, degree_reg, SSres,SSreg,MSres,MSreg, F = Anova(Y,y_hat)
```

In [36]:
```
# Creating ANOVA Table
anova_dict = {'DF':[degree_reg, degree_res, degree_reg+degree_res], 'SS':[S
Anova_df = pd.DataFrame(anova_dict,index=["Regression","Residual","Total"])
print("__________________________________________________ANOVA Table________
Anova_df
```

```
__________________________________________________ANOVA Table______________
_______________________________________________
```

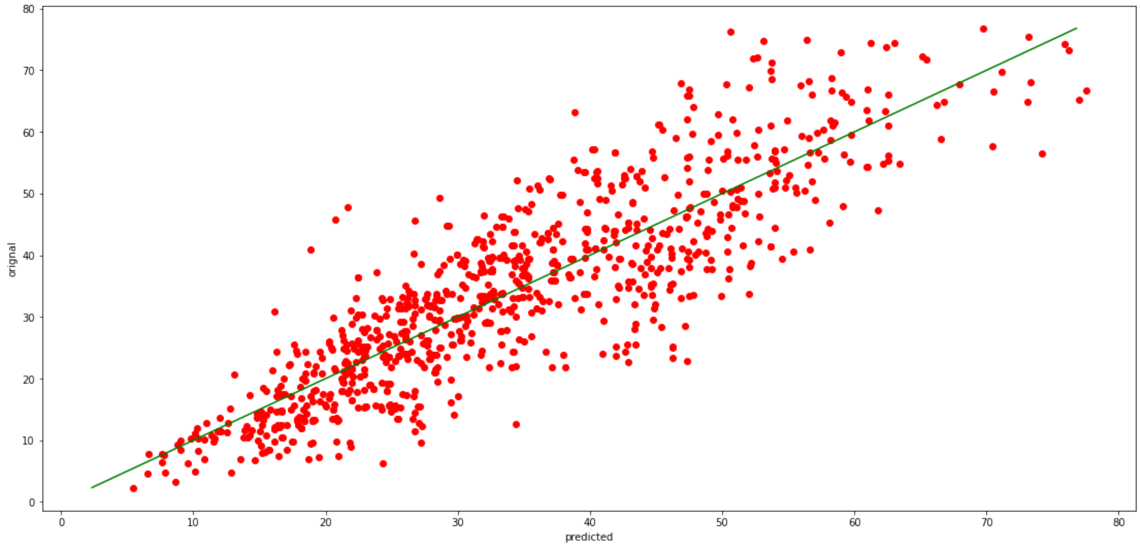
Out[36]:

	DF	SS	MS	F
Regression	7	173184.070842	24740.581549	402.585944
Residual	903	55493.107608	61.454161	402.585944
Total	910	228677.178451	24802.035710	402.585944

In [37]:
```
# Predict R^2 value and adjusted R^2 value:
R_sq = (SSreg /(SSreg + SSres)) * 100
AdjR_sq = (1-MSres/(MSres+MSreg)) * 100
print(f"The R square value is: {R_sq}")
print(f"Adjusted R square value is: {AdjR_sq}")
```

```
The R square value is: 75.73299269117916
Adjusted R square value is: 99.75222130164006
```

In [38]:
```
# Plot the regression Line fitted by the function made
plt.figure(figsize=[19 , 9])
plt.scatter( y_hat, Y, color='red')
plt.plot([Y.min(), Y.max()], [Y.min(), Y.max()], color = 'green')
plt.xlabel('predicted')
plt.ylabel('orignal')
plt.show()
```



In [39]:
```
poly_clf.coef_
```

Out[39]:
```
array([ 1.17601191e+06, -1.21797726e+02, -9.77272550e+01, -3.53238116e+02,
       -3.76676754e+02,  7.75298927e+02, -1.18724136e+02, -1.19179473e+02,
        6.76041295e+01,  4.58569642e-02,  9.58166294e-02,  2.45367886e-01,
        2.12243912e-01, -5.60072240e-01,  8.08168778e-02,  1.11840962e-01,
       -2.67501692e-02,  3.75291857e-02,  2.29013507e-01,  1.98304855e-01,
       -3.92699353e-01,  3.46025865e-02,  1.13145075e-01, -1.88148393e-02,
        2.29256936e-01,  6.41643557e-01, -4.57355352e-02,  2.31437236e-01,
        3.36810371e-01, -9.65166159e-02,  3.98257271e-01, -1.86652966e+00,
        3.21019876e-01,  2.71458231e-01, -1.92018555e-01, -4.76295366e-01,
       -2.95132699e-01, -9.73777597e-01,  5.63309522e-01,  4.51403155e-02,
        7.60088304e-02, -6.04224989e-02,  4.35843106e-02, -3.54123111e-02,
        8.47042616e-03, -6.06689386e-06, -2.20257117e-05, -4.76340947e-05,
       -2.63211185e-05,  1.38001638e-04, -1.53867352e-05, -2.34148406e-05,
        2.89179034e-06, -2.38577527e-05, -9.52081729e-05, -6.92818608e-05,
        2.02832790e-04, -2.33398617e-05, -5.41903497e-05,  7.61137495e-06,
       -7.55985889e-05, -1.91994847e-04,  1.47059216e-04, -7.83991712e-05,
       -1.23243121e-04,  3.86206262e-05, -7.56230220e-05,  6.23000777e-04,
       -7.63596192e-05, -1.11817041e-04,  3.69673001e-05,  1.42895591e-04,
        1.13123415e-04,  3.26173368e-04, -3.76855626e-04, -1.38787925e-05,
       -3.44717012e-05,  9.40944118e-06, -2.42730306e-05,  1.26708725e-05,
       -1.86758801e-05, -4.97891804e-06, -4.65633877e-05, -1.95809431e-05,
        1.57178620e-04, -4.83027976e-06, -2.69114663e-05, -3.52802014e-06,
       -7.93912150e-05, -1.84597292e-04,  6.19061213e-05, -5.91607219e-05,
       -1.27288785e-04,  2.64160414e-05, -1.14313346e-04,  2.62857960e-04,
       -3.53890651e-05, -1.23902927e-04,  3.98766328e-05, -2.99250314e-04,
        1.05528209e-04,  2.23774686e-04, -2.23500602e-04, -1.78748905e-07,
       -2.47184991e-05,  6.14728402e-06, -2.84803363e-05,  8.87615050e-06,
       -2.67067742e-05, -4.75929813e-05, -2.07327890e-04, -1.49501786e-04,
       -6.97120622e-05, -1.14466730e-04,  3.70891174e-05, -2.51396756e-04,
        4.58600950e-04, -2.05987207e-04, -3.45136754e-04,  9.02871623e-05,
        2.56148746e-04, -1.17655684e-04,  9.06445894e-05, -2.76395054e-04,
       -3.75746659e-05, -1.13619411e-04,  3.06424205e-05, -7.19747602e-05,
        4.61584203e-05, -2.17620055e-05, -9.22905995e-05,  1.12007898e-03,
       -2.14705143e-04, -1.38523167e-04,  1.02402436e-04, -2.17934930e-03,
        1.68745775e-04,  1.40963824e-03, -2.74291537e-04, -7.58363680e-05,
       -8.54061515e-05,  9.10081018e-05, -5.93612645e-05,  5.62805167e-05,
        5.38827762e-06,  2.62793533e-05, -5.74879886e-05,  8.53082669e-04,
       -1.89034298e-08,  3.27361420e-06,  3.03961629e-04, -2.07486199e-04,
        2.02739725e-04, -2.17014698e-04, -2.99956218e-05, -5.14819789e-06,
       -1.41634667e-05,  1.48009276e-05, -1.19784175e-05,  1.43270799e-05,
       -9.54203623e-06, -4.54050266e-06,  3.98634058e-06, -1.09147175e-05,
        7.88327226e-05])
```

In [ ]: