In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as  plt
```

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

Data Preprocessing

In [3]:
```
df.head()
```

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)
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Cement (component 1)(kg in a m^3 mixture)                0
Blast Furnace Slag (component 2)(kg in a m^3 mixture)    0
Fly Ash (component 3)(kg in a m^3 mixture)               0
Water  (component 4)(kg in a m^3 mixture)                0
Superplasticizer (component 5)(kg in a m^3 mixture)      0
Coarse Aggregate  (component 6)(kg in a m^3 mixture)     0
Fine Aggregate (component 7)(kg in a m^3 mixture)        0
Age (day)                                                0
strength                                                 0
dtype: int64
```

In [5]:
```
df.shape
```

Out[5]:
```
(1030, 9)
```

In [6]:
```
df.describe()
```

Out[6]:

	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)	Agg (comp 7)(l m
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.(
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.5
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.1
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.(
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.9
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.5
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.(
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.6

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                                                 Non-Null Count  D
type
---  ------                                                 --------------  -
----
 0   Cement (component 1)(kg in a m^3 mixture)              1030 non-null   f
loat64
 1   Blast Furnace Slag (component 2)(kg in a m^3 mixture)  1030 non-null   f
loat64
 2   Fly Ash (component 3)(kg in a m^3 mixture)             1030 non-null   f
loat64
 3   Water  (component 4)(kg in a m^3 mixture)              1030 non-null   f
loat64
 4   Superplasticizer (component 5)(kg in a m^3 mixture)    1030 non-null   f
loat64
 5   Coarse Aggregate  (component 6)(kg in a m^3 mixture)   1030 non-null   f
loat64
 6   Fine Aggregate (component 7)(kg in a m^3 mixture)      1030 non-null   f
loat64
 7   Age (day)                                              1030 non-null   i
nt64
 8   strength                                               1030 non-null   f
loat64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

columns rename

In [8]:
```
df.rename(columns={'Cement (component 1)(kg in a m^3 mixture)' : 'Cement' , 'B
```

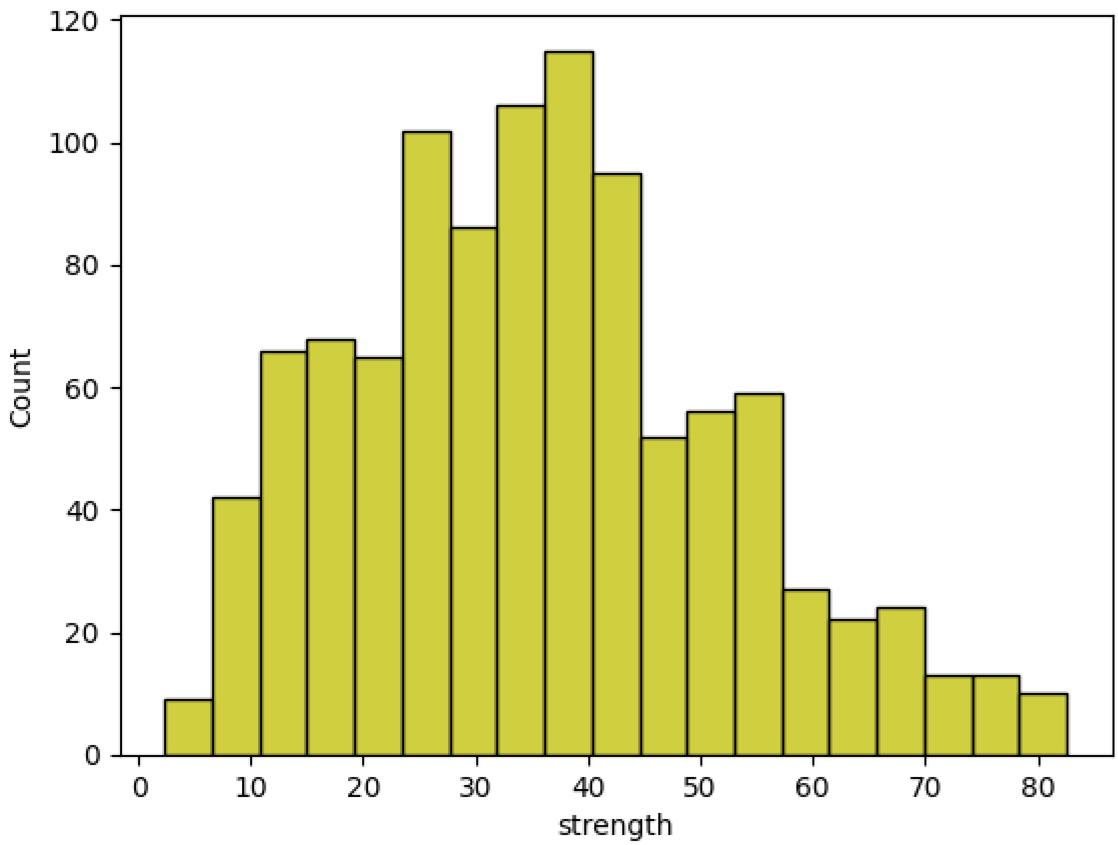
In [9]:
```
df.head()
```

Out[9]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	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

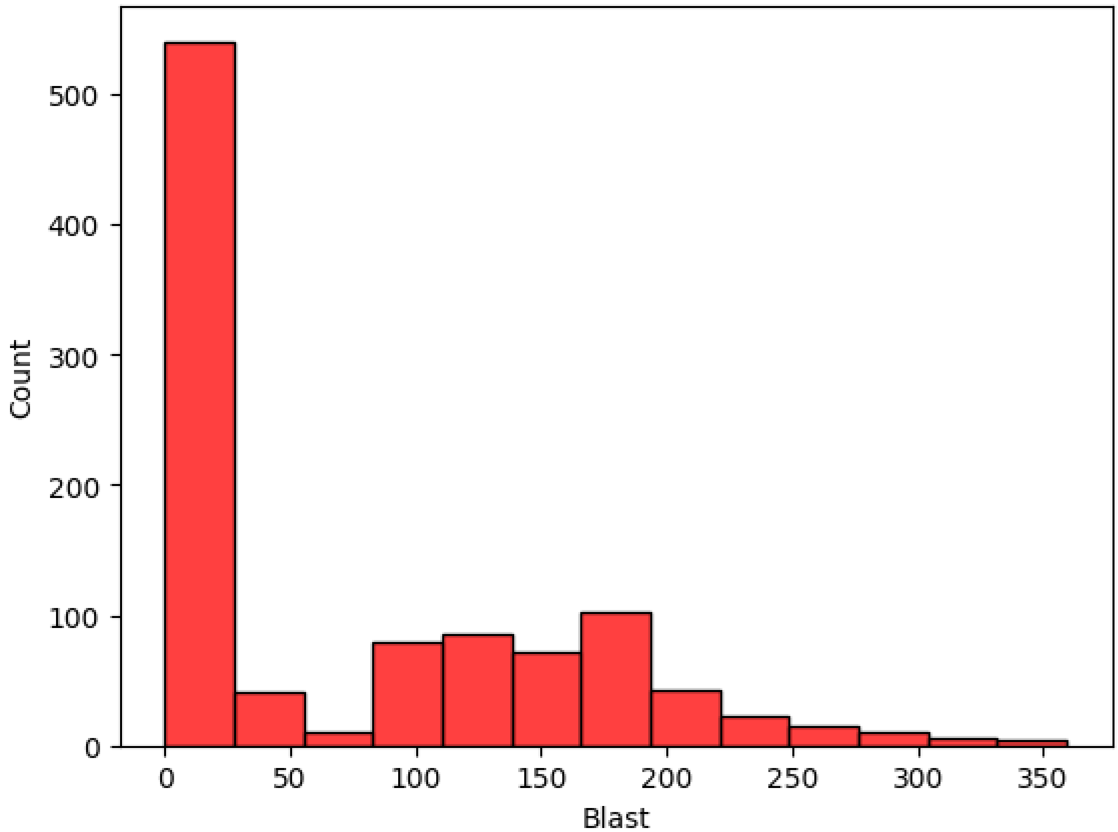
In [10]:
```
sns.histplot(x = df.strength , color = 'y' )
```

Out[10]: <AxesSubplot: xlabel='strength', ylabel='Count'>



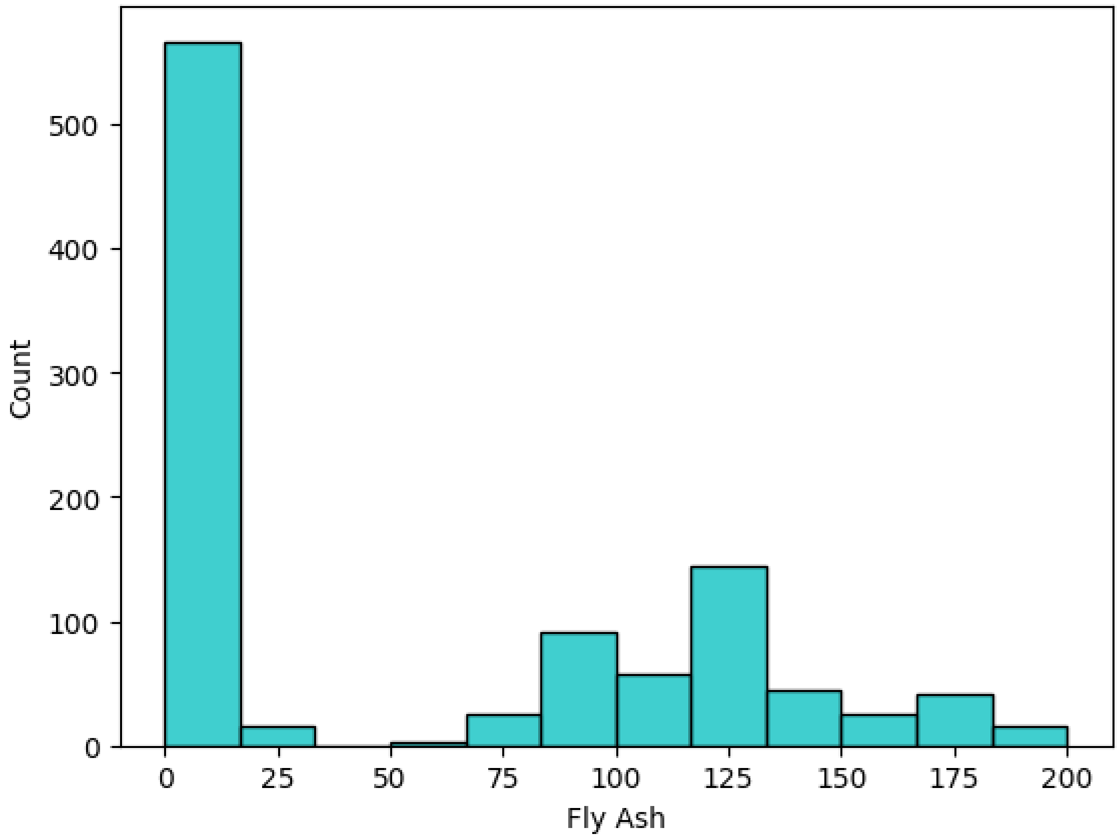
In [11]:
```
sns.histplot(x = df.Blast , color='r')
```

Out[11]: <AxesSubplot: xlabel='Blast', ylabel='Count'>



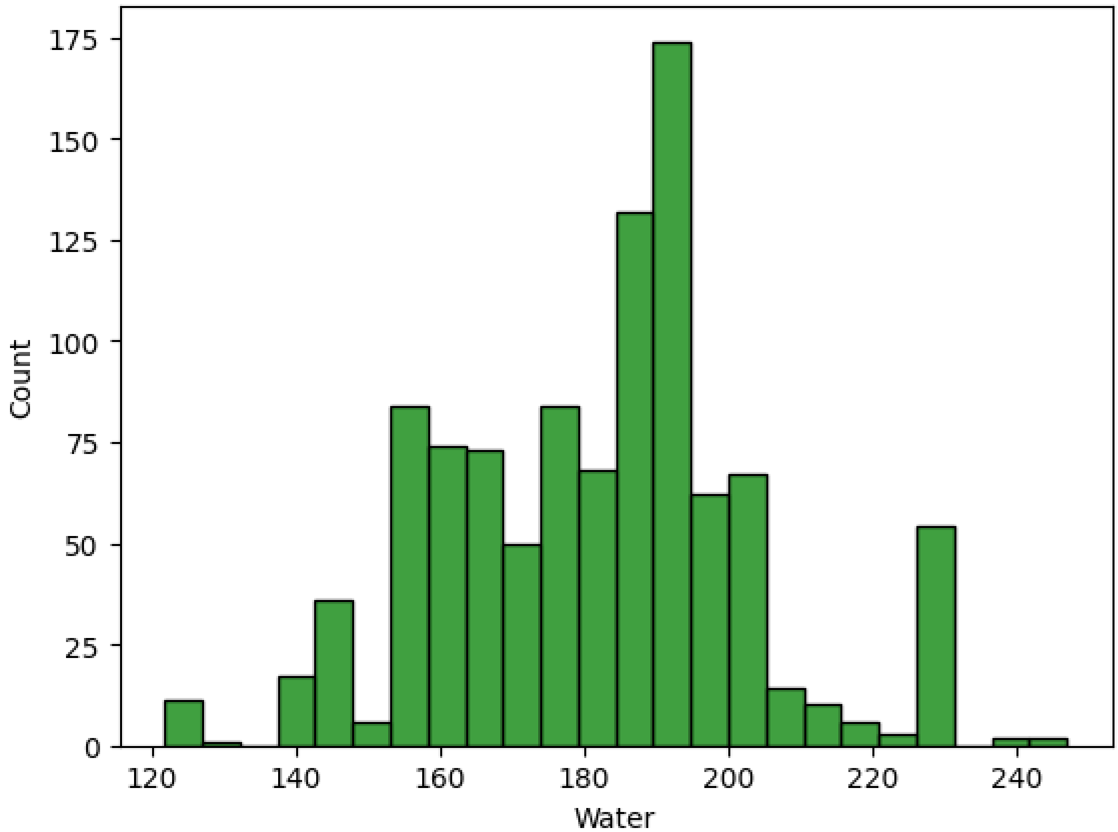
In [12]:
```
sns.histplot(x = df['Fly Ash'] , color = 'c' )
```

Out[12]:
```
<AxesSubplot: xlabel='Fly Ash', ylabel='Count'>
```



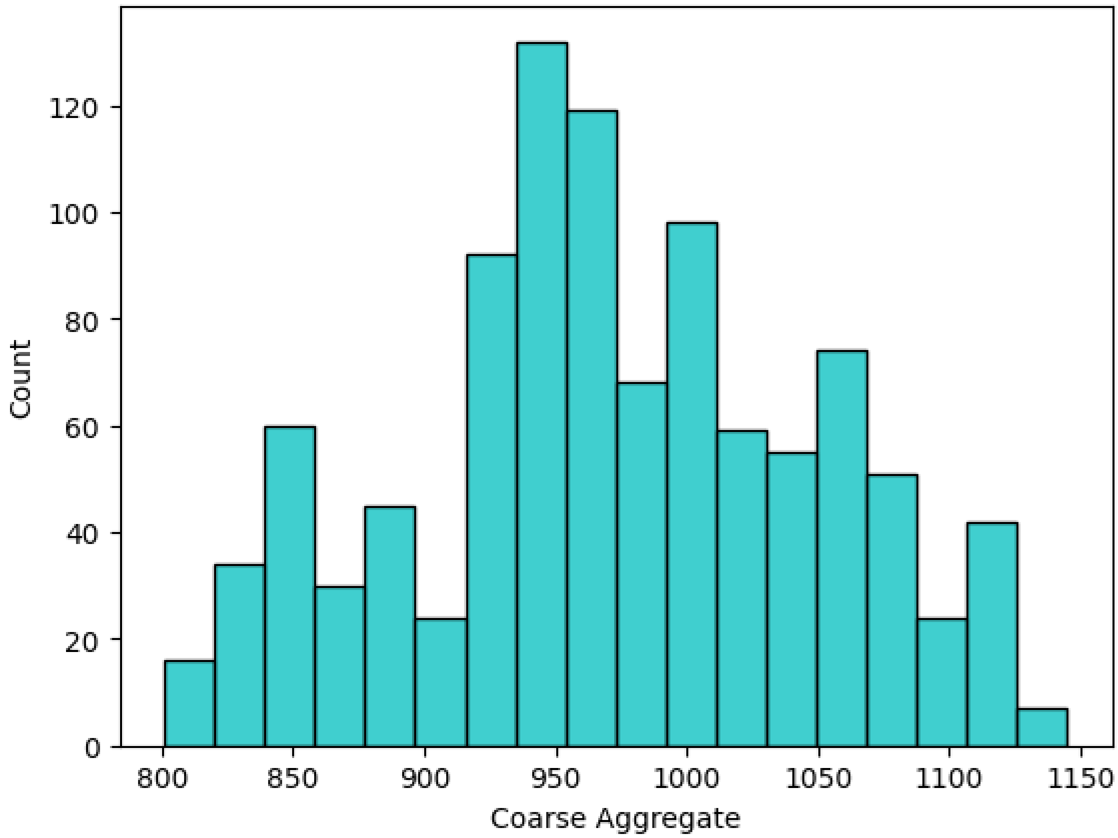
In [13]:
```
sns.histplot(x = df.Water , color='g')
```

Out[13]: <AxesSubplot: xlabel='Water', ylabel='Count'>



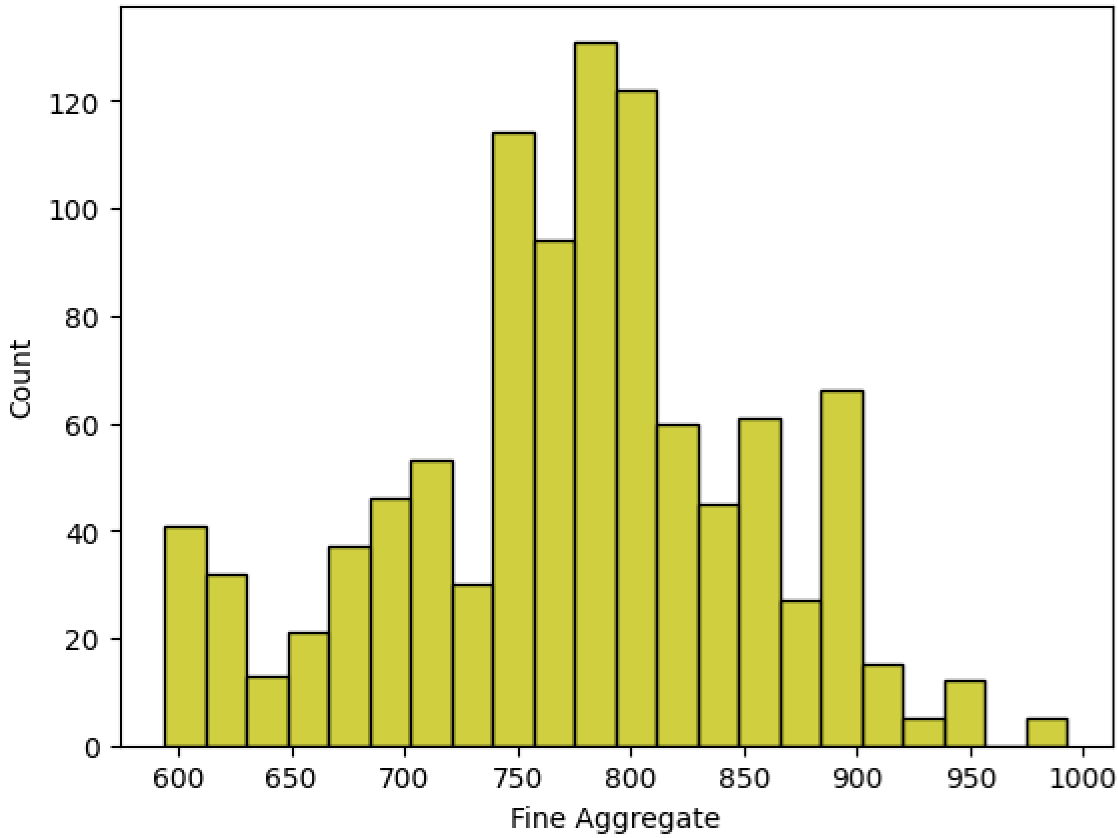
In [14]:
```
sns.histplot(x = df['Coarse Aggregate'] , color = 'c' )
```

Out[14]: <AxesSubplot: xlabel='Coarse Aggregate', ylabel='Count'>

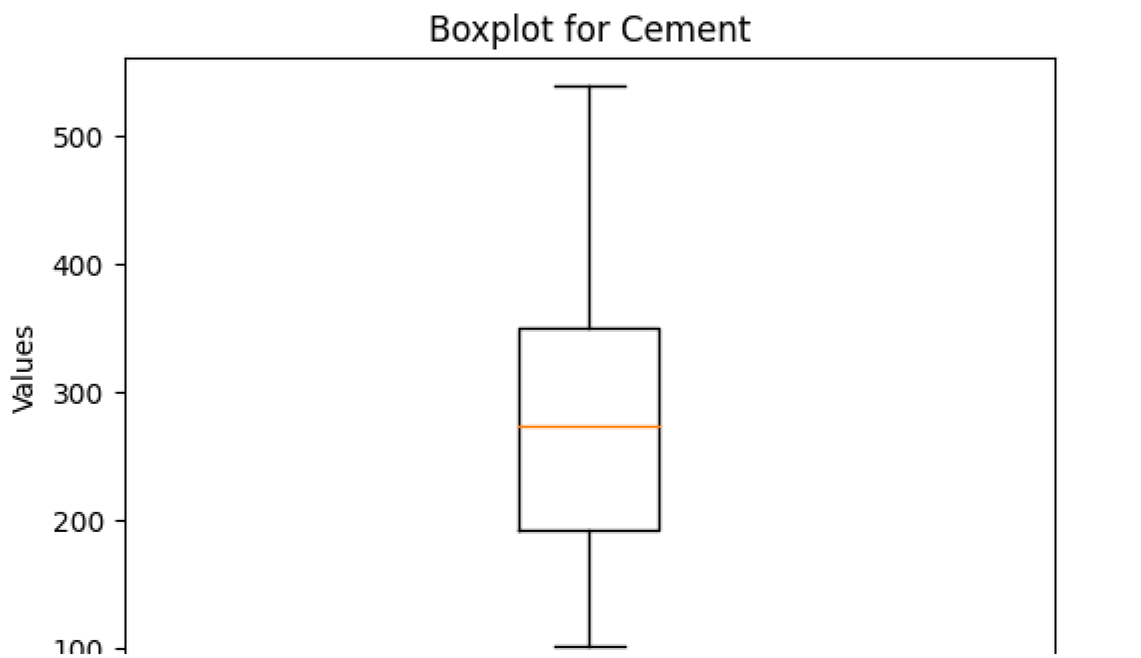


In [15]:
```
sns.histplot(x = df['Fine Aggregate'] , color = 'y' )
```

Out[15]: <AxesSubplot: xlabel='Fine Aggregate', ylabel='Count'>



In [16]:
```
for column in df.columns:
    plt.figure(figsize=(6, 4))  # Adjust the figure size if needed
    plt.boxplot(df[column])
    plt.title(f'Boxplot for {column}')
    plt.ylabel('Values')
    plt.xlabel('Column')
    plt.grid(False)  # Remove grid lines if needed
    plt.show()
```



In [17]:
```
df = df[(df['Water'] < 230) & (df.Water > 130)]
```

In [18]:
```
df.head()
```

Out[18]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	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 [19]:
```
df.describe()
```

Out[19]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Agg
count	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.000000	1014.(
mean	281.566667	73.601479	53.842998	182.002367	6.119822	973.480572	772.7
std	104.603931	86.764523	63.791103	20.252171	5.897110	77.338418	79.(
min	102.000000	0.000000	0.000000	137.800000	0.000000	801.000000	594.(
25%	194.700000	0.000000	0.000000	164.925000	0.000000	932.000000	730.9
50%	273.000000	20.000000	0.000000	185.000000	6.400000	968.000000	779.3
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.(
max	540.000000	359.400000	200.100000	228.000000	32.200000	1145.000000	945.(

In [20]:
```
df =df[(df.Blast < 350)]
```

In [21]:
```
df.head()
```

Out[21]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	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 [22]:
```
df = df[(df.Superplasticizer < 17) & (df.Superplasticizer > 4)]
```

In [23]:
```
df.head()
```

Out[23]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
70	374.0	189.2	0.0	170.1	10.1	926.1	756.7	3	34.4
71	313.3	262.2	0.0	175.5	8.6	1046.9	611.8	3	28.8
72	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
75	475.0	118.8	0.0	181.1	8.9	852.1	781.5	3	37.8
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4

In [24]:
```
df = df[((df['Fine Aggregate'] < 990) & (df['Fine Aggregate'] > 640))]
```

In [25]:
```
df.head()
```

Out[25]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
70	374.0	189.2	0.0	170.1	10.1	926.1	756.7	3	34.4
72	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
75	475.0	118.8	0.0	181.1	8.9	852.1	781.5	3	37.8
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
78	388.6	97.1	0.0	157.9	12.1	852.1	925.7	3	28.1

In [26]:
```
df = df[(df['Age (day)'] < 170)]
```

In [27]:
```
df.head()
```

Out[27]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
70	374.0	189.2	0.0	170.1	10.1	926.1	756.7	3	34.4
72	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
75	475.0	118.8	0.0	181.1	8.9	852.1	781.5	3	37.8
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
78	388.6	97.1	0.0	157.9	12.1	852.1	925.7	3	28.1

In [28]:
```
df = df[(df.strength < 75)]
```

In [29]:
```
df.head()
```

Out[29]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
70	374.0	189.2	0.0	170.1	10.1	926.1	756.7	3	34.4
72	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
75	475.0	118.8	0.0	181.1	8.9	852.1	781.5	3	37.8
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
78	388.6	97.1	0.0	157.9	12.1	852.1	925.7	3	28.1

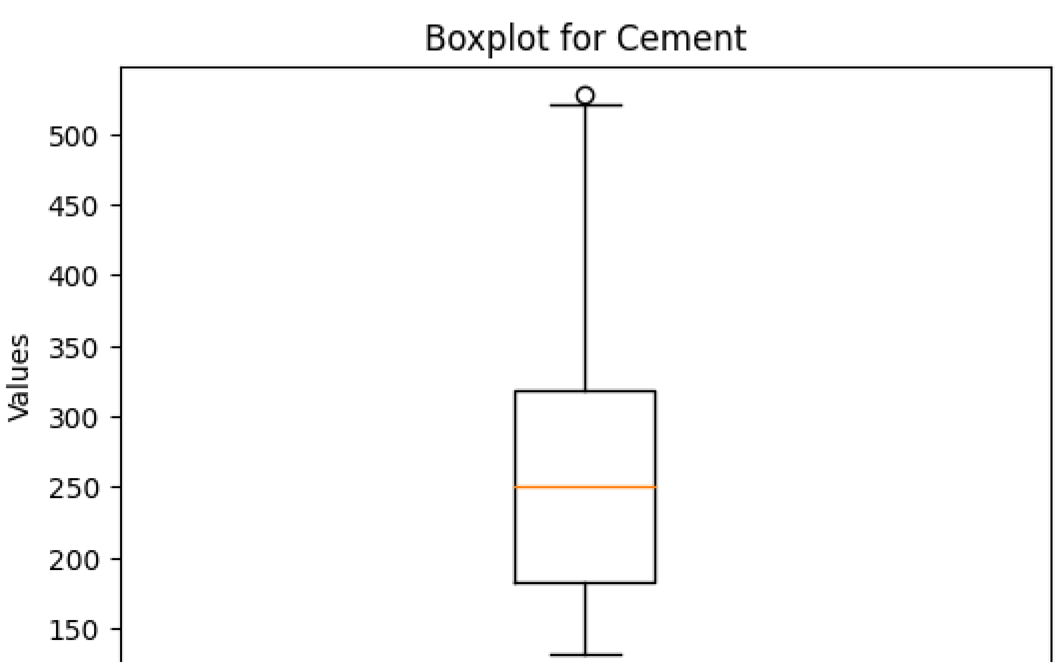
In [30]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 540 entries, 70 to 1029
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Cement             540 non-null    float64
 1   Blast              540 non-null    float64
 2   Fly Ash            540 non-null    float64
 3   Water              540 non-null    float64
 4   Superplasticizer   540 non-null    float64
 5   Coarse Aggregate   540 non-null    float64
 6   Fine Aggregate     540 non-null    float64
 7   Age (day)          540 non-null    int64  
 8   strength           540 non-null    float64
dtypes: float64(8), int64(1)
memory usage: 42.2 KB
```

In [31]:
```
df.shape
```

Out[31]:
```
(540, 9)
```

In [32]:
```
for column in df.columns:
    plt.figure(figsize=(6, 4))  # Adjust the figure size if needed
    plt.boxplot(df[column])
    plt.title(f'Boxplot for {column}')
    plt.ylabel('Values')
    plt.xlabel('Column')
    plt.grid(False)  # Remove grid lines if needed
    plt.show()
```



In [33]:
```
df.head()
```

Out[33]:

	Cement	Blast	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	strength
70	374.0	189.2	0.0	170.1	10.1	926.1	756.7	3	34.4
72	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
75	475.0	118.8	0.0	181.1	8.9	852.1	781.5	3	37.8
77	425.0	106.3	0.0	153.5	16.5	852.1	887.1	3	33.4
78	388.6	97.1	0.0	157.9	12.1	852.1	925.7	3	28.1

In [60]:
```
Y = df.strength
X = df.drop('strength' , axis = 1)
```

In [102]:
```
from sklearn.model_selection import train_test_split , RandomizedSearchCV
from sklearn.tree import ExtraTreeRegressor , DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
```

In [97]:
```
xtrain , xtest , ytrain ,ytest = train_test_split(X,Y, test_size=.2)
```

In [63]:
```
Extra = ExtraTreeRegressor()
Extra.fit(xtrain , ytrain)
```

Out[63]:
```
ExtraTreeRegressor()
```

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 [64]:
```
Extra.score(xtrain , ytrain)
```

Out[64]:
```
0.9904328616239464
```

In [65]:
```
Extra.score(xtest , ytest)
```

Out[65]:
```
0.7685519030084791
```

In [81]:
```
random = RandomForestRegressor(n_estimators=1550 ,random_state=200,min_weight_
random.fit(xtrain , ytrain)
```

Out[81]:
```
RandomForestRegressor(n_estimators=1550, random_state=200)
```

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 [82]:
```
random.score(xtrain , ytrain)
```
Out[82]: 0.9767334604340495

In [83]:
```
random.score(xtest , ytest)
```
Out[83]: 0.8820580544618124

In [69]:
```
reg = LinearRegression()
reg.fit(xtrain , ytrain)
```
Out[69]: 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 [70]:
```
reg.score(xtest , ytest)
```
Out[70]: 0.691534100637976

In [71]:
```
reg.score(xtrain , ytrain)
```
Out[71]: 0.7663284873502556

In [72]:
```
sv = SVR(kernel='linear')
sv.fit(xtrain , ytrain)
```
Out[72]: SVR(kernel='linear')

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 [73]:
```
sv.score(xtest , ytest)
```
Out[73]: 0.6831866450407

In [74]:
```
dtr = DecisionTreeRegressor(random_state=200 ,)
dtr.fit(xtrain , ytrain)
```
Out[74]: DecisionTreeRegressor(random_state=200)

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 [75]:
```
dtr.score(xtrain , ytrain)
```
Out[75]: 0.9904328616239464

In [76]:
```
dtr.score(xtest,ytest)
```

Out[76]:
```
0.8322353974278694
```

In [92]:
```
param_grid = {
    'n_estimators': [50, 100, 200,300,400,500],
    'max_depth': [None, 5,10,15, 20,25, 30],
    'min_samples_split': [2, 5, 7, 10],
    'min_samples_leaf': [1, 2, 4 ,6],
    'max_features': ['auto', 'sqrt', 'log2' , 4,5,.9]
}

# Perform Randomized Search Cross Validation
random_search = RandomizedSearchCV(estimator=RandomForestRegressor(), param_di
                                   n_iter=100, cv=5, random_state=42)

# Fit the RandomizedSearchCV to the data
random_search.fit(xtrain , ytrain)

# Print the best parameters and the best score
print("Best Parameters:", random_search.best_params_)
print("Best Score (negative mean squared error):", -random_search.best_score_)
```

```
Best Parameters: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_l
eaf': 1, 'max_features': 5, 'max_depth': 15}
Best Score (negative mean squared error): -0.8742010296168663
```

In [93]:
```
random_search.best_params_
```

Out[93]:
```
{'n_estimators': 500,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 5,
 'max_depth': 15}
```

In [94]:
```
tuning_result_dt_gs = pd.DataFrame(random_search.cv_results_)
tuning_result_dt_gs
```

Out[94]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_m
0	0.818571	0.064722	0.033106	0.003426	400	
1	0.665848	0.032885	0.022631	0.006571	300	
2	0.904633	0.039846	0.033131	0.009938	400	
3	1.118776	0.030074	0.031724	0.009712	400	
4	0.358349	0.028044	0.014495	0.002244	200	
...	...	...	...	...	...	
95	0.265562	0.000002	0.003124	0.006249	100	
96	0.471760	0.006245	0.024999	0.007650	300	
97	0.752944	0.006250	0.037495	0.007659	500	
98	0.284308	0.011691	0.015626	0.000009	200	
99	0.174954	0.006252	0.009372	0.007652	100	

100 rows × 18 columns

In [95]:
```
tuning_result_dt_gs.nsmallest(n=10,columns='rank_test_score')
```

Out[95]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	param_m
32	1.318490	0.164951	0.043423	0.006881	500	
89	1.090364	0.011695	0.037496	0.007660	500	
17	0.633141	0.095339	0.021861	0.007661	200	
24	1.120796	0.203486	0.041006	0.008008	400	
14	0.109350	0.000002	0.003124	0.006249	50	
40	0.988209	0.052628	0.032120	0.001756	300	
21	1.099532	0.197365	0.054502	0.046520	400	
95	0.265562	0.000002	0.003124	0.006249	100	
91	0.518628	0.011690	0.018746	0.006248	200	
42	1.268462	0.015304	0.031232	0.000015	500	

In [105]:
```
y_pred_dt_gs = random_search.predict(xtest)
print("\nrf random_search Performance:")
print("mean_absolute_error:", mean_absolute_error(ytest, y_pred_dt_gs))
random_search.score(xtest,ytest)
```

```
rf random_search Performance:
mean_absolute_error: 1.9898705452519634
```

Out[105]: 0.9641703661643218

In [ ]: