To find the strength of the concrete-Regression Problem

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [3]:

```
import warnings
warnings.filterwarnings('ignore')


import itertools
```

In [4]:

```
data=pd.read_csv("https://raw.githubusercontent.com/Premalatha-success/Yhills_July12_Ana
```

In [5]:

```
len(data)
```

Out[5]:

```
1030
```

In [150]:

```
req_col_names = ["Cement", "BlastFurnaceSlag", "FlyAsh", "Water", "Superplasticizer",
                 "CoarseAggregate", "FineAggregate", "Age", "CC_Strength"]
curr_col_names = list(data.columns)

mapper = {}
for i, name in enumerate(curr_col_names):
    mapper[name] = req_col_names[i]

data = data.rename(columns=mapper)
```

In [151]:

```
data.head()
```

Out[151]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggreg
0	141.3	212.0	0.0	203.5	0.0	971.8	74
1	168.9	42.2	124.3	158.3	10.8	1080.8	79
2	250.0	0.0	95.7	187.4	5.5	956.9	86
3	266.0	114.0	0.0	228.0	0.0	932.0	67
4	154.8	183.4	0.0	193.3	9.1	1047.4	69

In [152]:

```
data.tail()
```

Out[152]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAgg
1025	135.0	0.0	166.0	180.0	10.0	961.0	
1026	531.3	0.0	0.0	141.8	28.2	852.1	
1027	276.4	116.0	90.3	179.6	8.9	870.1	
1028	342.0	38.0	0.0	228.0	0.0	932.0	
1029	540.0	0.0	0.0	173.0	0.0	1125.0	

In [153]:

```
data.columns
```

Out[153]:

```
Index(['Cement', 'BlastFurnaceSlag', 'FlyAsh', 'Water', 'Superplasticize
r',
       'CoarseAggregate', 'FineAggregate', 'Age', 'CC_Strength'],
      dtype='object')
```

In [154]:

```
data.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   BlastFurnaceSlag  1030 non-null   float64
 2   FlyAsh            1030 non-null   float64
 3   Water             1030 non-null   float64
 4   Superplasticizer  1030 non-null   float64
 5   CoarseAggregate   1030 non-null   float64
 6   FineAggregate     1030 non-null   float64
 7   Age               1030 non-null   int64  
 8   CC_Strength       1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

In [155]:

```
data.isnull().sum()
```

Out[155]:

```
Cement              0
BlastFurnaceSlag    0
FlyAsh              0
Water               0
Superplasticizer    0
CoarseAggregate     0
FineAggregate       0
Age                 0
CC_Strength         0
dtype: int64
```

In [156]:

```
data.describe()
```

Out[156]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAg
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972
std	104.506364	86.279342	63.997004	21.354219	5.973841	77
min	102.000000	0.000000	0.000000	121.800000	0.000000	801
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145

In [157]:

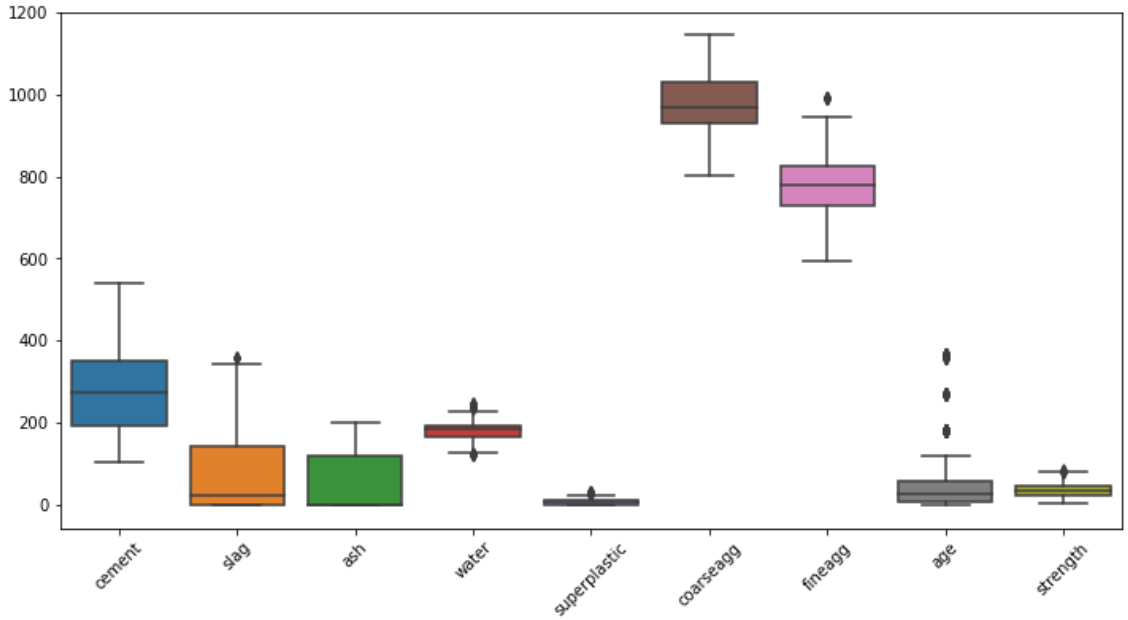
```
data.describe().transpose()
```

Out[157]:

	count	mean	std	min	25%	50%	75%	max
Cement	1030.0	281.167864	104.506364	102.00	192.375	272.900	350.000	540.0
BlastFurnaceSlag	1030.0	73.895825	86.279342	0.00	0.000	22.000	142.950	359.4
FlyAsh	1030.0	54.188350	63.997004	0.00	0.000	0.000	118.300	200.1
Water	1030.0	181.567282	21.354219	121.80	164.900	185.000	192.000	247.0
Superplasticizer	1030.0	6.204660	5.973841	0.00	0.000	6.400	10.200	32.2
CoarseAggregate	1030.0	972.918932	77.753954	801.00	932.000	968.000	1029.400	1145.0
FineAggregate	1030.0	773.580485	80.175980	594.00	730.950	779.500	824.000	992.6
Age	1030.0	45.662136	63.169912	1.00	7.000	28.000	56.000	365.0
CC_Strength	1030.0	35.817961	16.705742	2.33	23.710	34.445	46.135	82.6

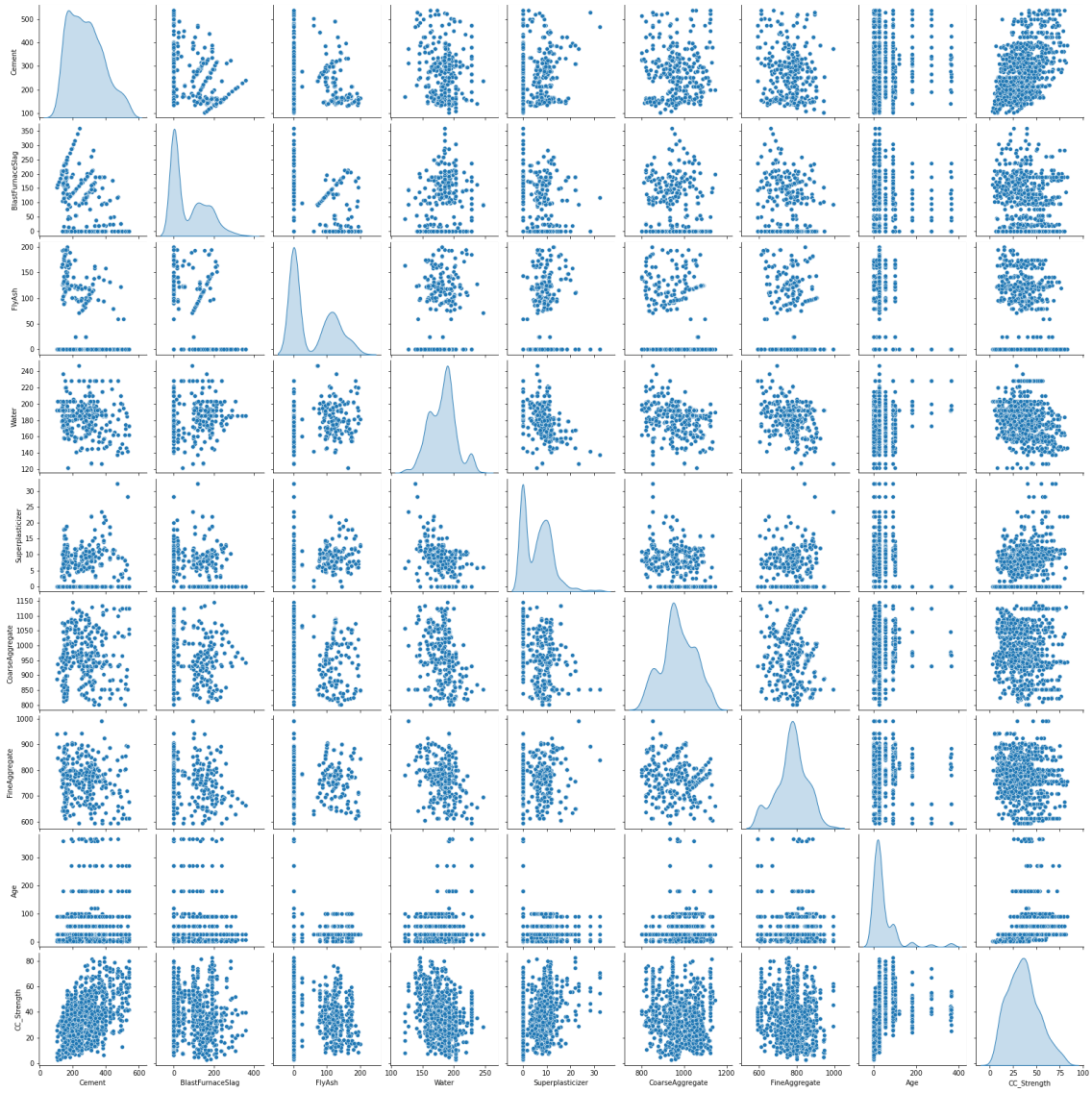
In [158]:

```
plt.subplots(figsize=(12, 6))
ax = sns.boxplot(data=mydata)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45);
```



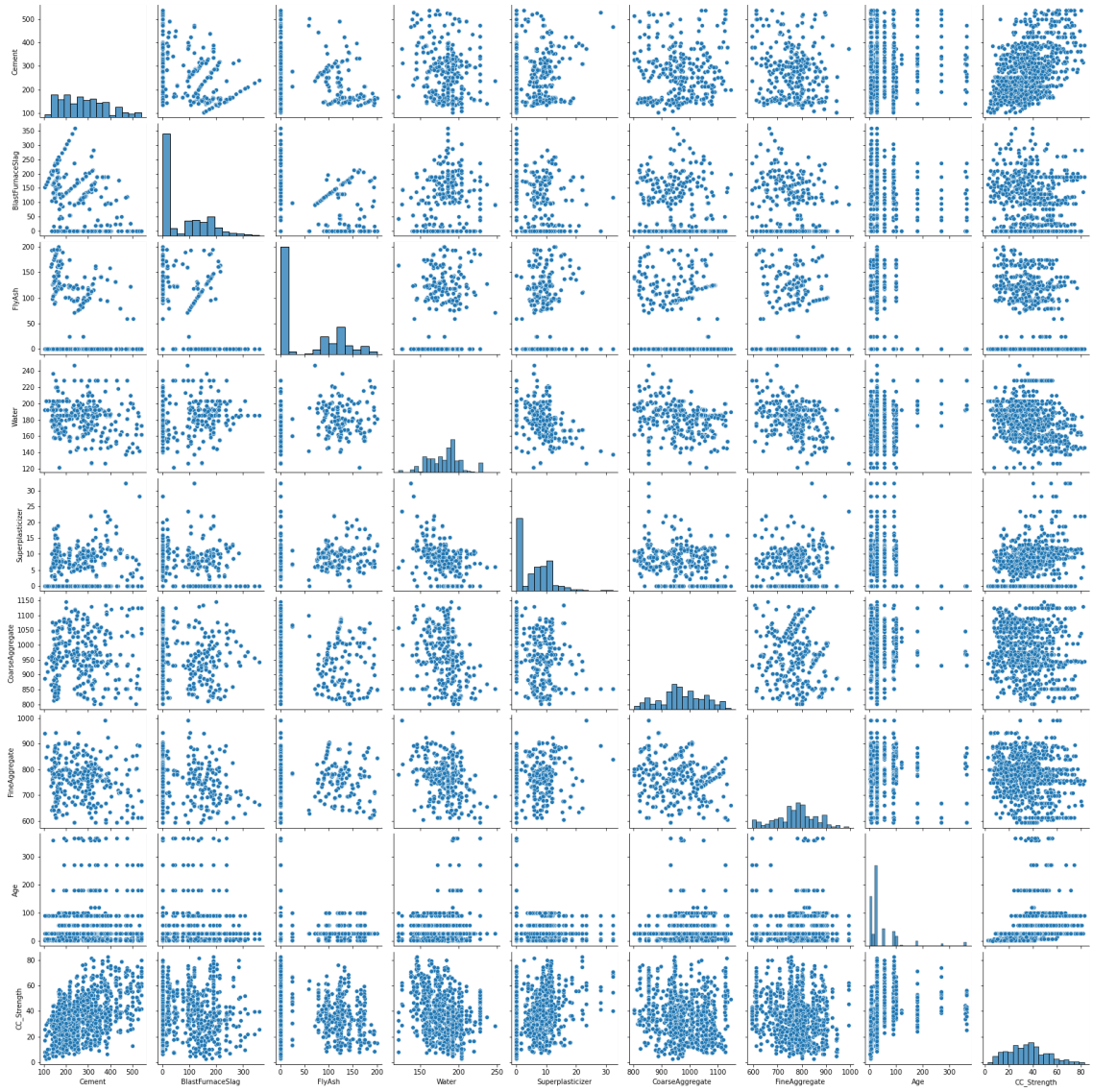
In [159]:

```
sns.pairplot(data, diag_kind = 'kde')
plt.show()
```



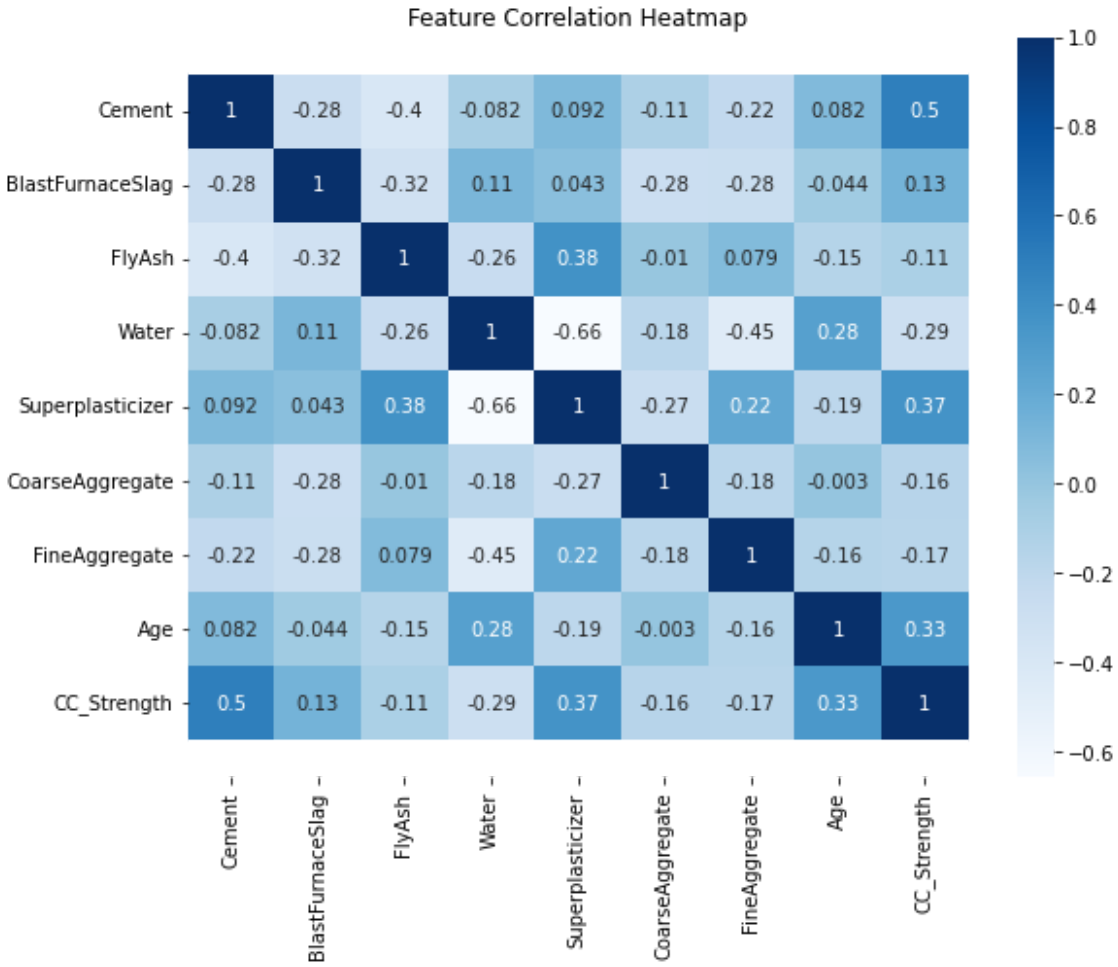
In [160]:

```
sns.pairplot(data)
plt.show()
```



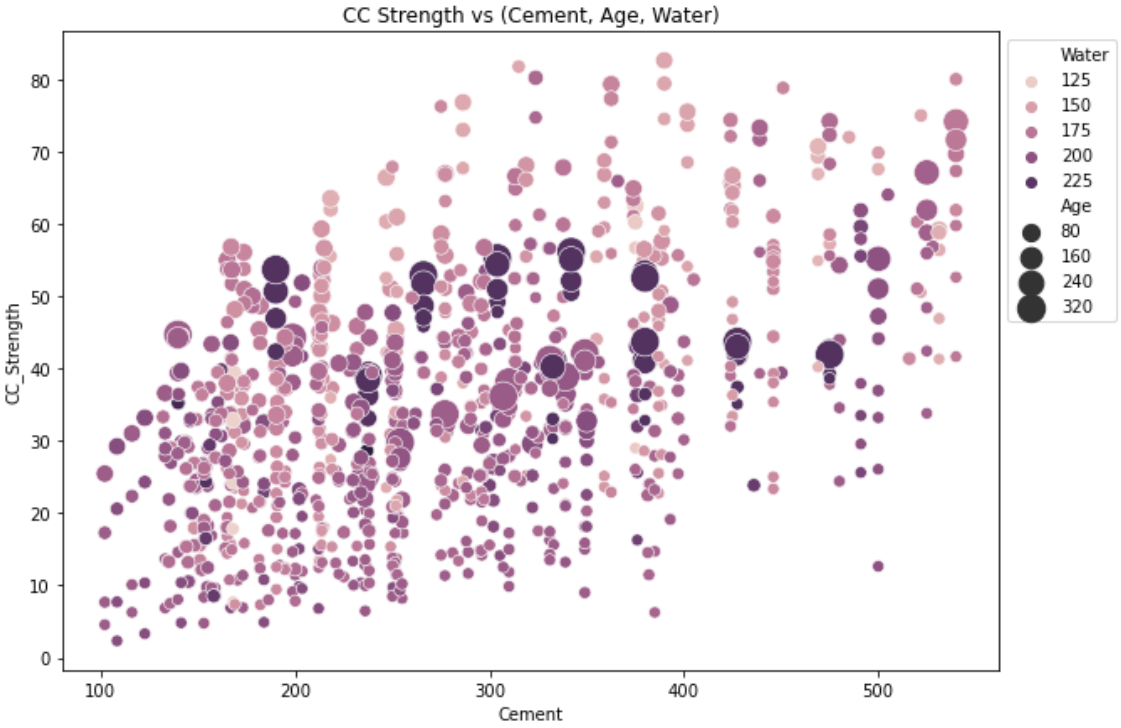
In [161]:

```
corr = data.corr()

plt.figure(figsize=(9,7))
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()
```



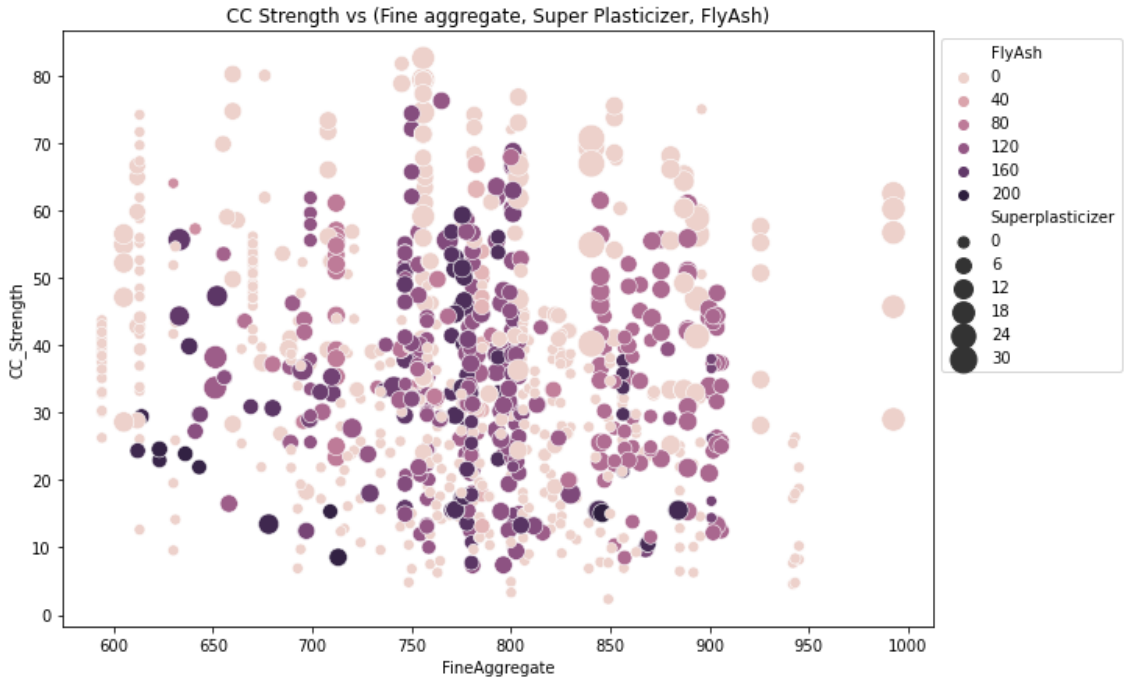
In [162]:

```
fig, ax = plt.subplots(figsize=(10,7))
sns.scatterplot(y="CC_Strength", x="Cement", hue="Water", size="Age", data=data, ax=ax,
ax.set_title("CC Strength vs (Cement, Age, Water)")
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
plt.show()
```



In [163]:

```
fig, ax = plt.subplots(figsize=(10,7))
sns.scatterplot(y="CC_Strength", x="FineAggregate", hue="FlyAsh", size="Superplasticizer
                data=data, ax=ax, sizes=(50, 300))
ax.set_title("CC Strength vs (Fine aggregate, Super Plasticizer, FlyAsh)")
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
plt.show()
```



In [176]:

```
X = data.iloc[:,:-1]          # Features - All columns but last
y = data.iloc[:,-1]
```

In [177]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)
```

In [178]:

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [179]:

```
# Split Train/Test data 70:30 ratio
from sklearn.model_selection import train_test_split

y = data['CC_Strength']
X = data.loc[:, data.columns != 'CC_Strength']

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.3, random_state=4
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[179]:

```
((721, 8), (309, 8), (721,), (309,))
```

In [180]:

```
#Scaling the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train),
                              columns = X_train.columns)
X_train_scaled.head()
```

Out[180]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	Fin
0	-0.448711	-0.879090	1.072130	-0.373869	0.084141	1.070423	
1	1.570641	-0.606933	0.406889	-0.938627	0.905081	-0.085519	
2	-1.540405	1.082708	-0.826775	0.473269	-1.038368	-0.825115	
3	2.079721	-0.879090	-0.826775	0.849775	-1.038368	1.957421	
4	1.372665	0.326338	-0.826775	-1.338665	1.726020	-1.571176	

In [181]:

```
# Importing models
from sklearn.linear_model import LinearRegression, Lasso, Ridge

# Linear Regression
lr = LinearRegression()
# Lasso Regression
lasso = Lasso()
# Ridge Regression
ridge = Ridge()

# Fitting models on Training data
lr.fit(X_train, y_train)
lasso.fit(X_train, y_train)
ridge.fit(X_train, y_train)

# Making predictions on Test data
y_pred_lr = lr.predict(X_test)
y_pred_lasso = lasso.predict(X_test)
y_pred_ridge = ridge.predict(X_test)
```

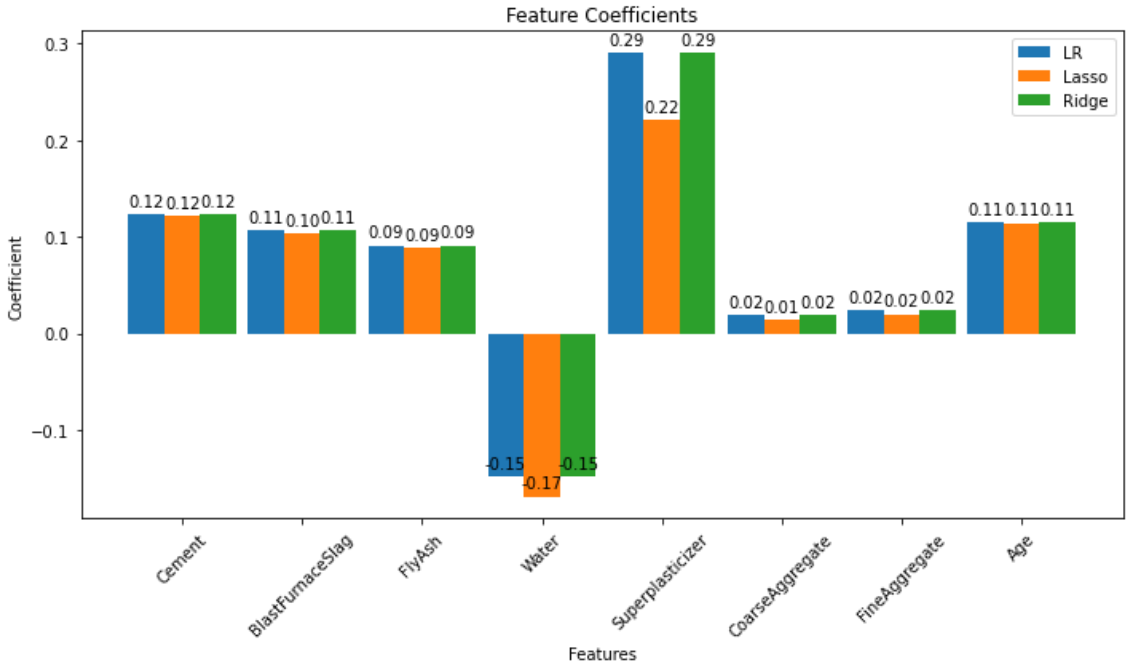
In [182]:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

print("Model\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")
print("""LinearRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(
            np.sqrt(mean_squared_error(y_test, y_pred_lr)),mean_squared_error(y_test, y_
            mean_absolute_error(y_test, y_pred_lr), r2_score(y_test, y_pred_lr)))
print("""LassoRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(
            np.sqrt(mean_squared_error(y_test, y_pred_lasso)),mean_squared_error(y_test,
            mean_absolute_error(y_test, y_pred_lasso), r2_score(y_test, y_pred_lasso)))
print("""RidgeRegression \t {:.2f} \t\t {:.2f} \t{:.2f} \t\t{:.2f}""".format(
            np.sqrt(mean_squared_error(y_test, y_pred_ridge)),mean_squared_error(y_test,
            mean_absolute_error(y_test, y_pred_ridge), r2_score(y_test, y_pred_ridge)))
```

```
Model                    RMSE            MSE             MAE
R2
LinearRegression         10.52           110.77          8.47
0.58
LassoRegression          10.53           110.78          8.48
0.58
RidgeRegression          10.52           110.77          8.47
0.58
```

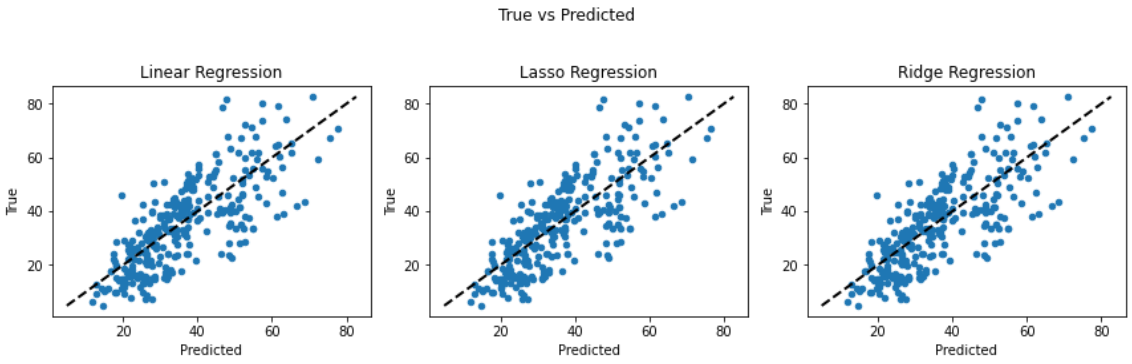
In [183]:

```
coeff_lr = lr.coef_
coeff_lasso = lasso.coef_
coeff_ridge = ridge.coef_

labels = req_col_names[:-1]

x = np.arange(len(labels))
width = 0.3

fig, ax = plt.subplots(figsize=(10,6))
rects1 = ax.bar(x - 2*(width/2), coeff_lr, width, label='LR')
rects2 = ax.bar(x, coeff_lasso, width, label='Lasso')
rects3 = ax.bar(x + 2*(width/2), coeff_ridge, width, label='Ridge')

ax.set_ylabel('Coefficient')
ax.set_xlabel('Features')
ax.set_title('Feature Coefficients')
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45)
ax.legend()

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{:.2f}'.format(height), xy=(rect.get_x() + rect.get_width() / 2, he
                    xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)
autolabel(rects3)

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



In [188]:

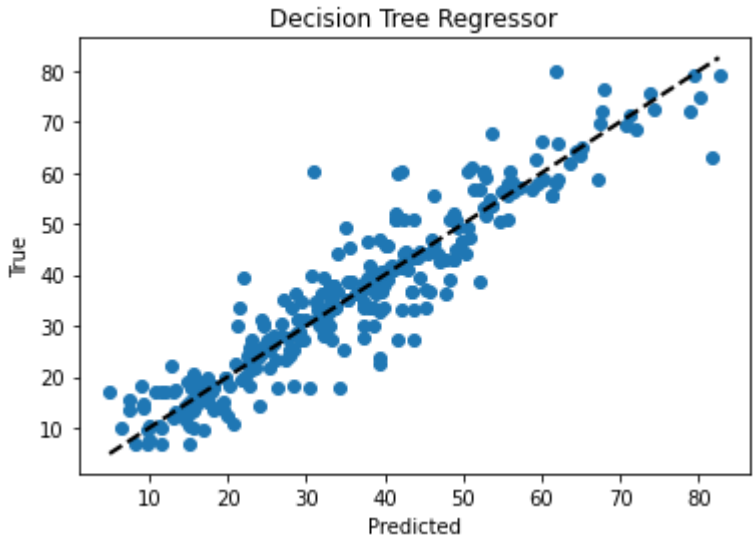
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(12,4))

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")

ax3.scatter(y_pred_ridge, y_test, s=20)
ax3.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
ax3.set_ylabel("True")
ax3.set_xlabel("Predicted")
ax3.set_title("Ridge Regression")

fig.suptitle("True vs Predicted")
fig.tight_layout(rect=[0, 0.03, 1, 0.95])
```



In [184]:

```
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()

dtr.fit(X_train, y_train)

y_pred_dtr = dtr.predict(X_test)

print("Model\t\t\t\t RMSE \t\t MSE \t\t MAE \t\t R2")
print("""Decision Tree Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}""".format(
            np.sqrt(mean_squared_error(y_test, y_pred_dtr)),mean_squared_error(y_test, y
            mean_absolute_error(y_test, y_pred_dtr), r2_score(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()
```

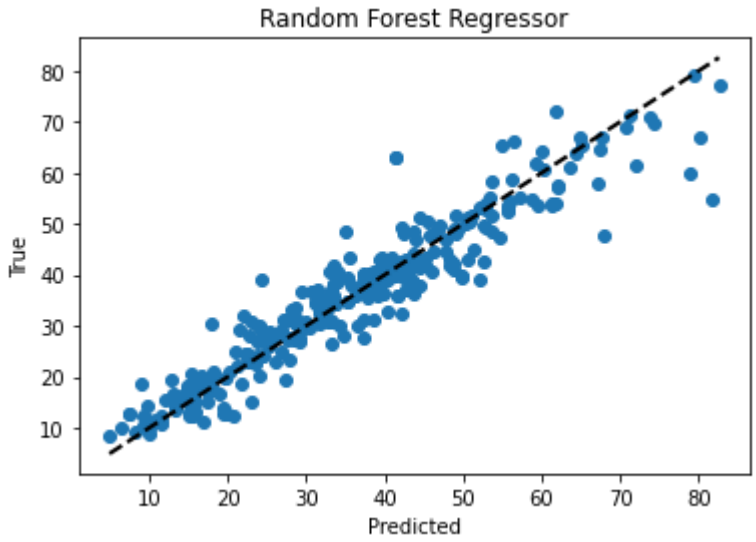
```
Model                           RMSE            MSE             MAE
R2
Decision Tree Regressor         5.82            33.87           3.93
0.87
```



In [185]:

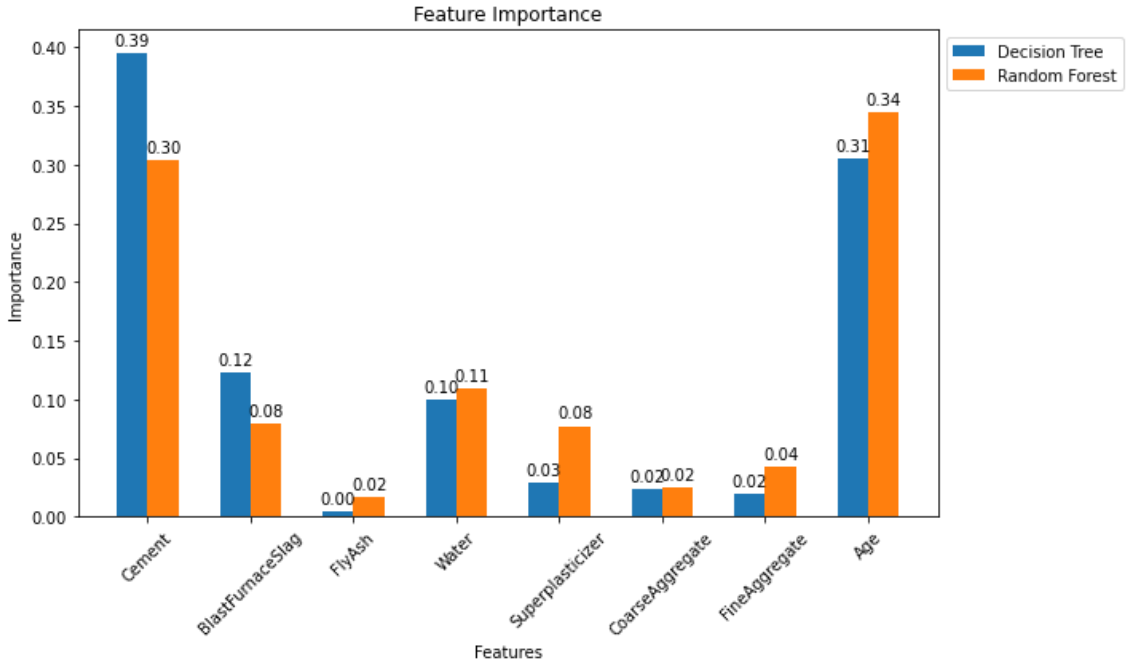
```
from sklearn.ensemble import RandomForestRegressor

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 MSE \t\t MAE \t\t R2")
print("""Random Forest Regressor \t {:.2f} \t\t {:.2f} \t\t{:.2f} \t\t{:.2f}""".format(
            np.sqrt(mean_squared_error(y_test, y_pred_rfr)),mean_squared_error(y_test, y
            mean_absolute_error(y_test, y_pred_rfr), r2_score(y_test, y_pred_rfr)))

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            MSE             MAE
R2
Random Forest Regressor         5.12            26.17           3.49
0.90
```



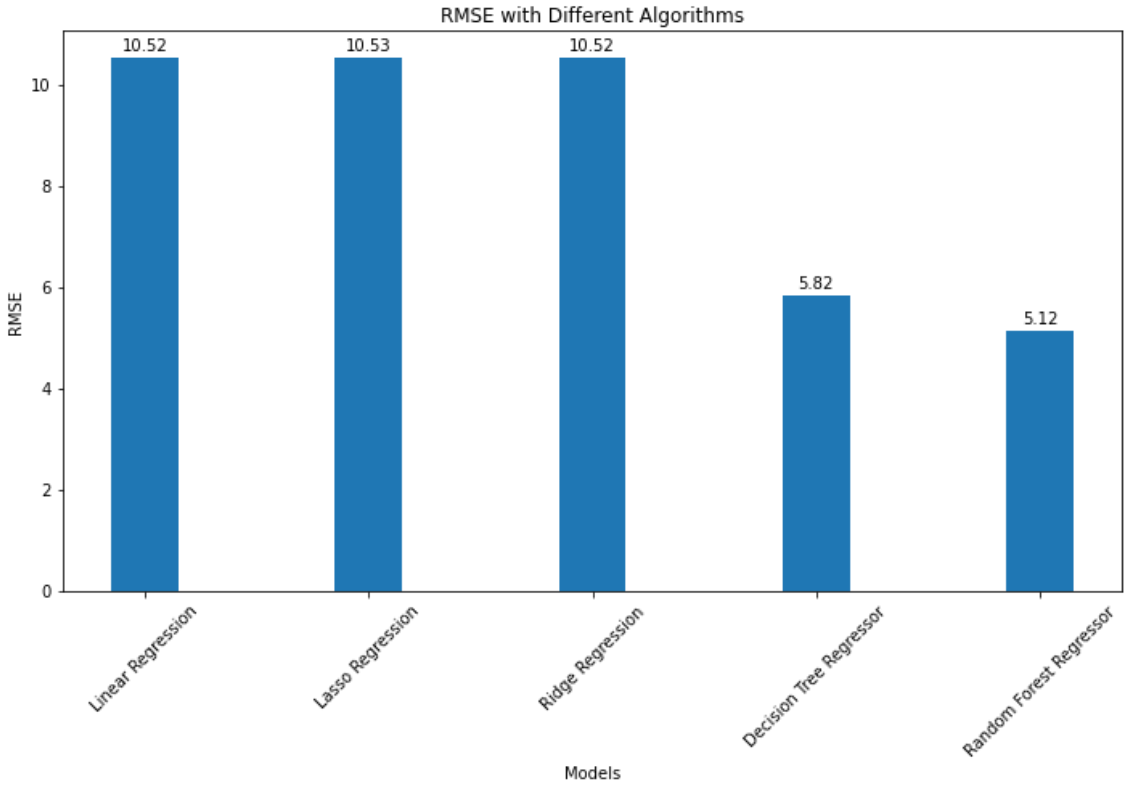
In [186]:

```
feature_dtr = dtr.feature_importances_
feature_rfr = rfr.feature_importances_

labels = req_col_names[:-1]

x = np.arange(len(labels))
width = 0.3

fig, ax = plt.subplots(figsize=(10,6))
rects1 = ax.bar(x-(width/2), feature_dtr, width, label='Decision Tree')
rects2 = ax.bar(x+(width/2), feature_rfr, width, label='Random Forest')

ax.set_ylabel('Importance')
ax.set_xlabel('Features')
ax.set_title('Feature Importance')
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45)
ax.legend(loc="upper left", bbox_to_anchor=(1,1))

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{:.2f}'.format(height), xy=(rect.get_x() + rect.get_width() / 2, he
                    xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')
autolabel(rects1)
autolabel(rects2)

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



In [187]:

```
models = [lr, lasso, ridge, dtr, rfr]
names = ["Linear Regression", "Lasso Regression", "Ridge Regression",
         "Decision Tree Regressor", "Random Forest Regressor"]
rmses = []

for model in models:
    rmses.append(np.sqrt(mean_squared_error(y_test, model.predict(X_test))))

x = np.arange(len(names))
width = 0.3

fig, ax = plt.subplots(figsize=(10,7))
rects = ax.bar(x, rmses, width)
ax.set_ylabel('RMSE')
ax.set_xlabel('Models')
ax.set_title('RMSE with Different Algorithms')
ax.set_xticks(x)
ax.set_xticklabels(names, rotation=45)
autolabel(rects)
fig.tight_layout()
plt.show()
```



Result

Random Forest Regressor is the best choice for this problem.

In [ ]: