Introduction:

The prediction of concrete compressive strength is a critical task in the field of civil engineering, as it directly impacts the safety and durability of structures. To improve the accuracy of these predictions, this study utilizes regression models and data science techniques. By exploring various algorithms and analyzing their performance, we aim to uncover key insights into the factors influencing concrete strength. Through this research, we seek to enhance traditional civil engineering practices by incorporating data-driven approaches, enabling more reliable predictions and optimizing the design and construction of concrete structures. This study represents a fusion of domain expertise and advanced analytics, pushing the boundaries of concrete strength prediction in the industry.

In [1]:

```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\admin\anaconda3\lib\sit
e-packages (1.4.4)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\admin\a
naconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: numpy>=1.18.5 in c:\users\admin\anaconda3
\lib\site-packages (from pandas) (1.21.5)
Requirement already satisfied: pytz>=2020.1 in c:\users\admin\anaconda3\l
ib\site-packages (from pandas) (2022.1)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\lib\s
ite-packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

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

Loading the Data

In [3]:

```
data = pd.read_excel("Concrete_Data.xls")
```

In [4]:

```
len(data)
```

Out[4]:

```
1030
```

In [5]:

```
data.sample(10)
```

Out[5]:

	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 (com 7)(m
379	500.00	0.0	0.00	140.00	4.000	966.00	
7	380.00	95.0	0.00	228.00	0.000	932.00	
398	160.00	128.0	122.00	182.00	6.400	824.00	
619	254.00	0.0	0.00	198.00	0.000	968.00	
190	233.81	0.0	94.58	197.89	4.567	947.04	
177	362.60	189.0	0.00	164.90	11.600	944.70	
593	252.50	0.0	0.00	185.70	0.000	1111.60	
925	164.00	163.0	128.00	197.00	8.000	961.00	
703	200.00	133.0	0.00	192.00	0.000	965.40	
678	288.00	192.0	0.00	192.00	0.000	932.00	

Simplifying Column names, since they appear to be too lengthy.

In [6]:

```
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 [7]:

```
data.sample(10)
```

Out[7]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggr
830	162.00	190.00	148.00	179.00	19.00	838.00	7
918	145.00	0.00	179.00	202.00	8.00	824.00	8
234	213.76	98.06	24.52	181.74	6.65	1066.00	7
250	250.00	0.00	95.69	187.42	5.53	956.86	8
701	288.00	192.00	0.00	192.00	0.00	932.00	7
849	165.00	0.00	150.00	182.00	12.00	1023.00	7
745	281.00	0.00	0.00	185.00	0.00	1104.00	7
789	349.00	0.00	0.00	192.00	0.00	1047.00	8
454	250.00	0.00	95.69	191.84	5.33	948.90	8
253	250.00	0.00	95.69	187.42	5.53	956.86	8

Checking for 'null' values

In [8]:

```
data.isna().sum()
```

Out[8]:

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

There are no null values in the data.

EDA

In [9]:

```
data.describe()
```

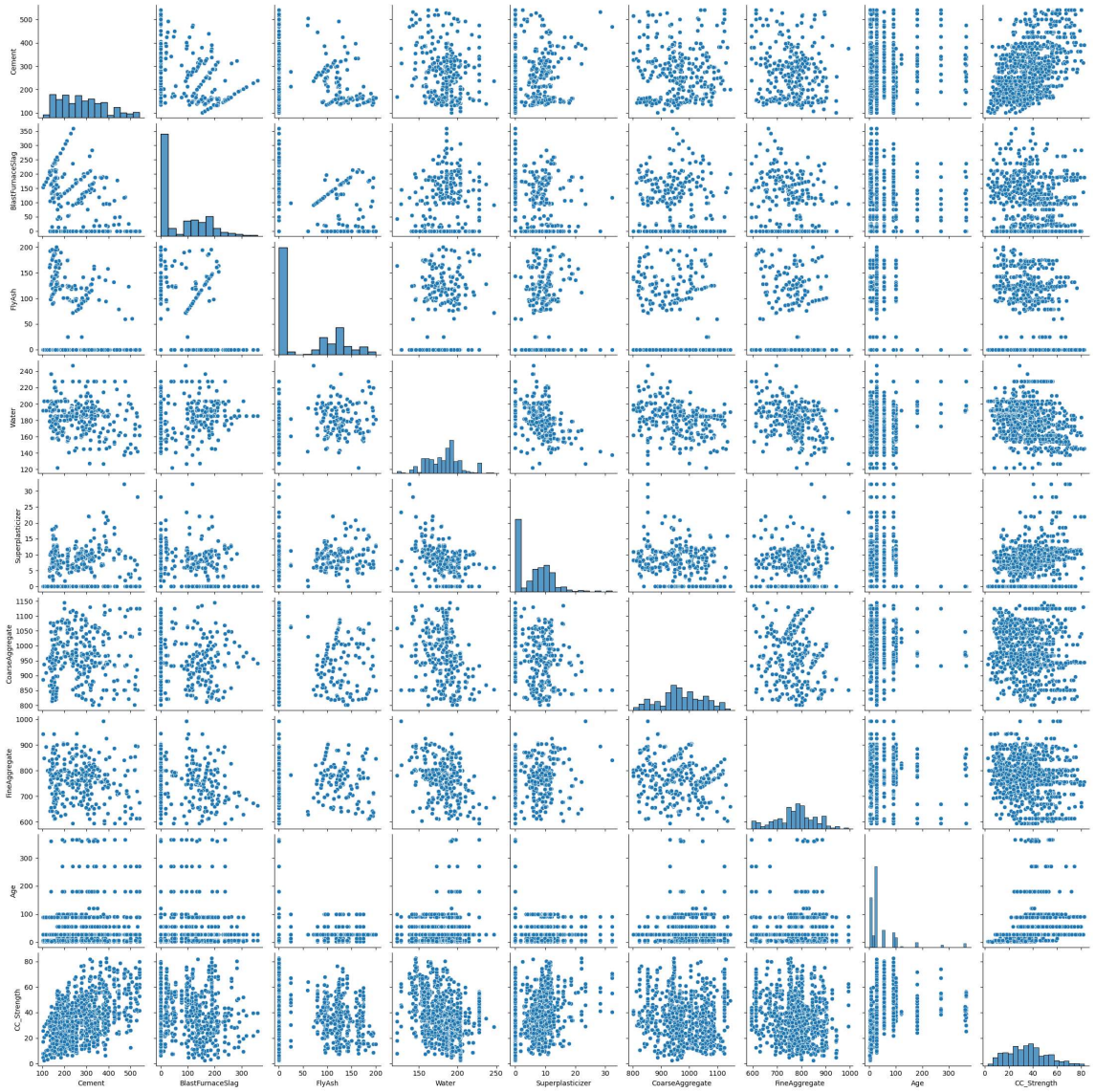
Out[9]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAgg
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.
mean	281.165631	73.895485	54.187136	181.566359	6.203112	972.
std	104.507142	86.279104	63.996469	21.355567	5.973492	77.
min	102.000000	0.000000	0.000000	121.750000	0.000000	801.
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.
50%	272.900000	22.000000	0.000000	185.000000	6.350000	968.
75%	350.000000	142.950000	118.270000	192.000000	10.160000	1029.
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.

Checking the pairwise relations of Features.

In [10]:

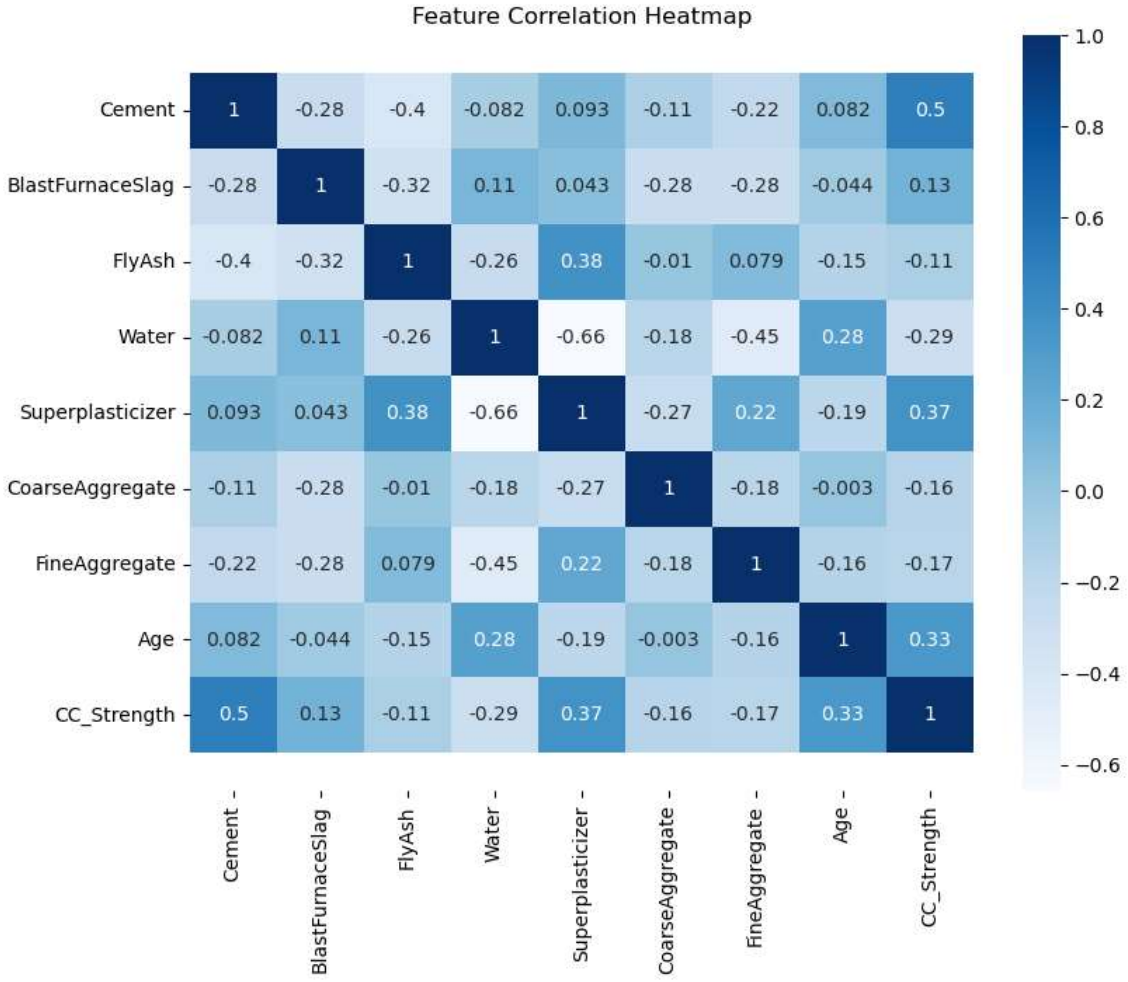
```
sns.pairplot(data)
plt.show()
```



There seems to be no high correlation between independant variables (features). This can be further confirmed by plotting the Pearson Correlation coefficients between the features.

In [11]:

```
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()
```



Observations:

There are'nt any high correlations between Compressive strength and other features except for Cement, which should be the case for more strength. Age and Super plasticizer are the other two features which are strongly correlated with Compressive Strength. Super Plasticizer seems to have a negative high correlation with Water, positive correlations with Fly ash and Fine aggregate.

We can further analyze these correlations visually by plotting these relations.

In [12]:

```
data.columns
```

Out[12]:

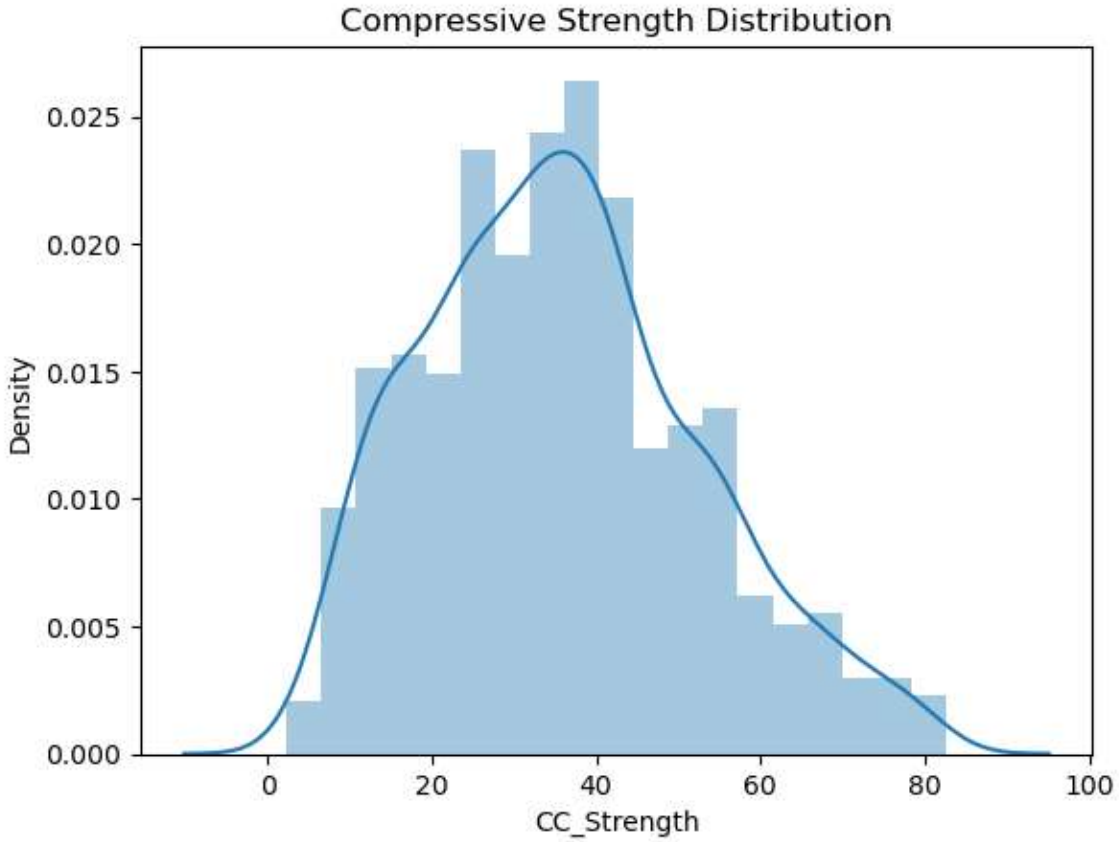
```
Index(['Cement', 'BlastFurnaceSlag', 'FlyAsh', 'Water', 'Superplasticize
r',
       'CoarseAggregate', 'FineAggregate', 'Age', 'CC_Strength'],
      dtype='object')
```

In [13]:

```
ax = sns.distplot(data.CC_Strength)
ax.set_title("Compressive Strength Distribution")
```

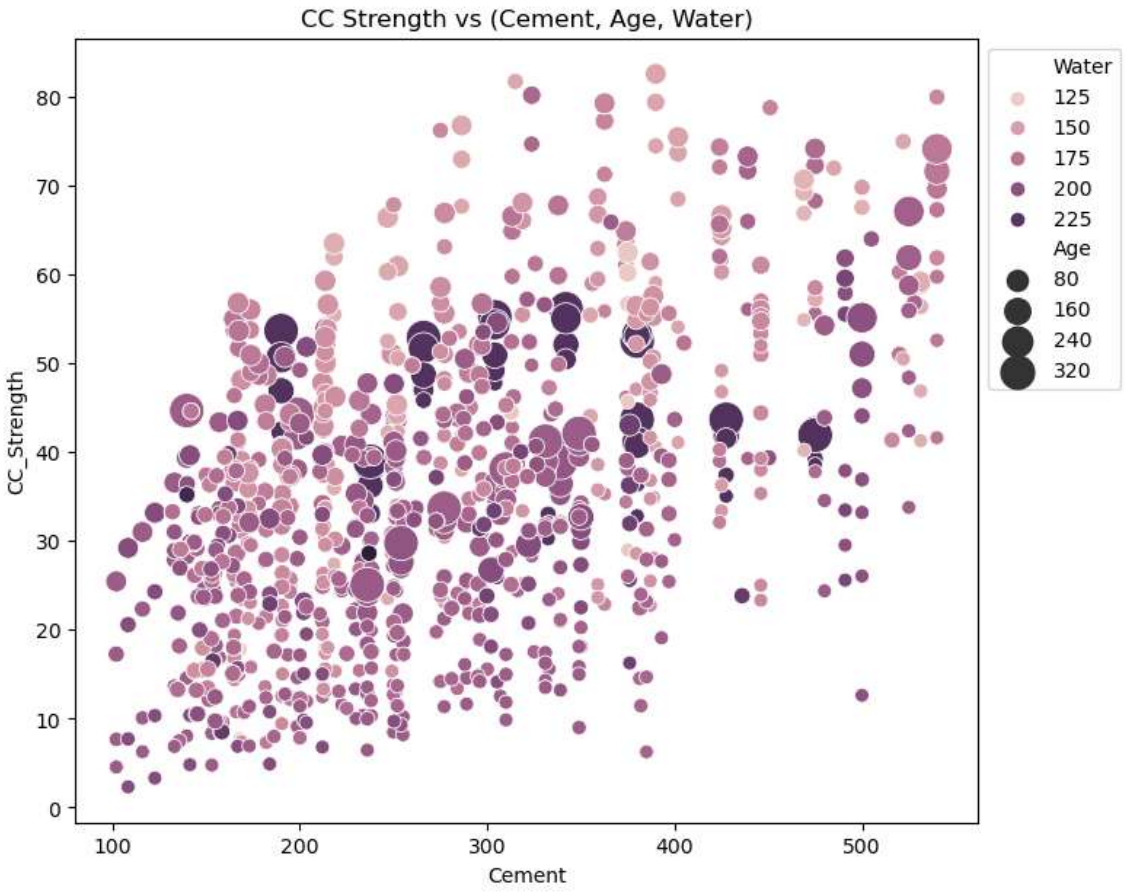
Out[13]:

```
Text(0.5, 1.0, 'Compressive Strength Distribution')
```



In [14]:

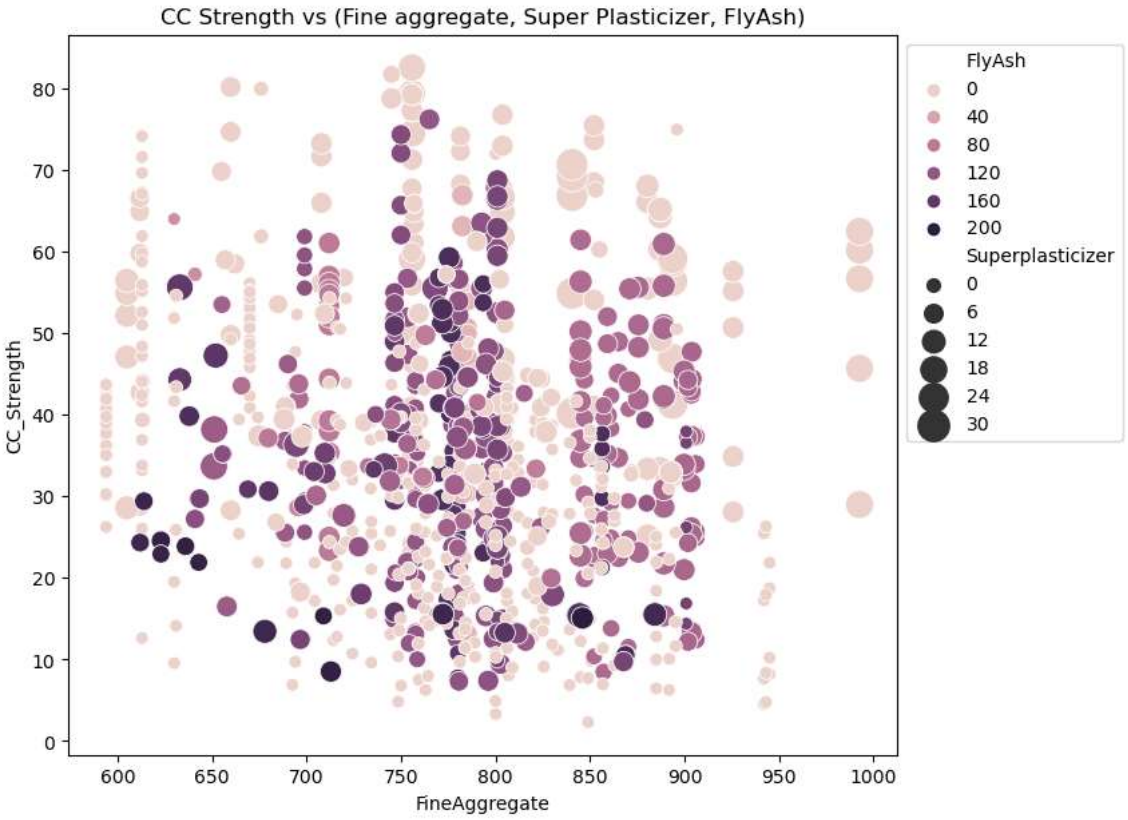
```
fig, ax = plt.subplots(figsize=(8,7))
sns.scatterplot(y="CC_Strength", x="Cement", hue="Water", size="Age", data=data, ax=ax,
ax.set_title("CC Strength vs (Cement, Age, Water)")
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
plt.show()
```



Observations from Strength vs (Cement, Age, Water) Compressive strength increases with amount of cement Compressive strength increases with age Cement with low age requires more cement for higher strength The older the cement is the more water it requires Concrete strength increases when less water is used in preparing it

In [15]:

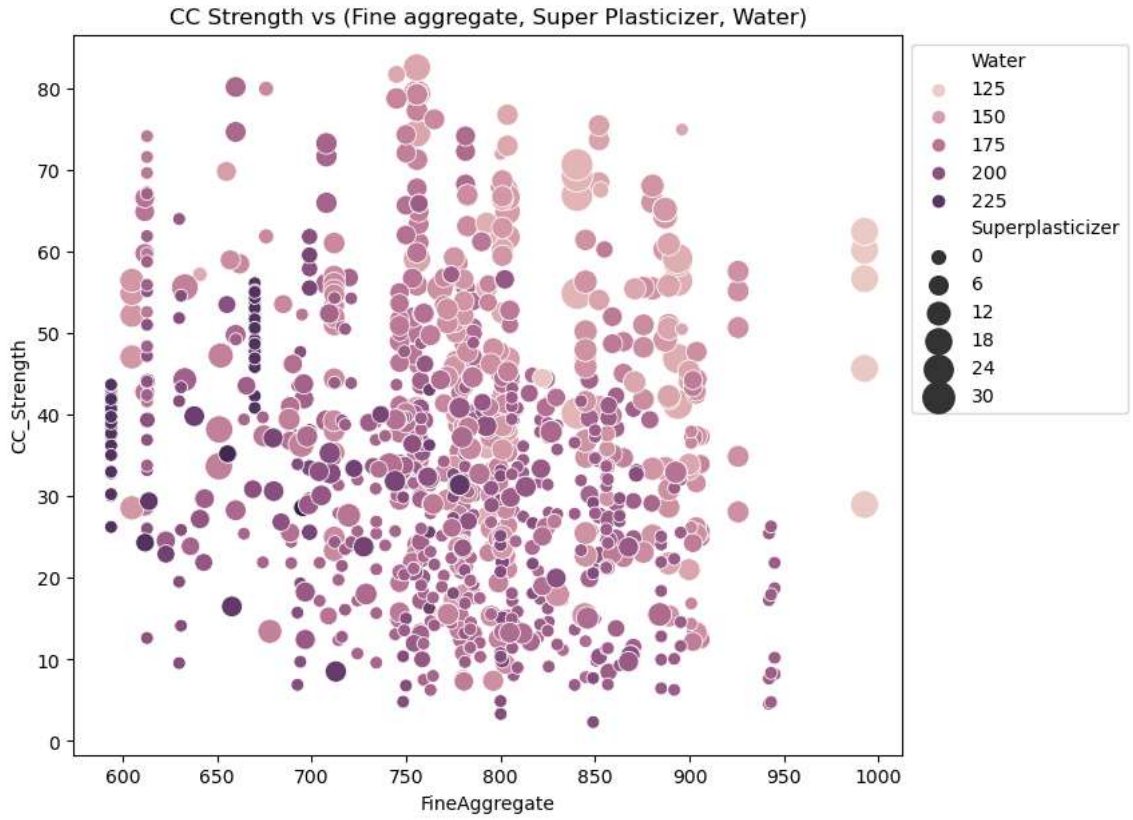
```
fig, ax = plt.subplots(figsize=(8,7))
sns.scatterplot(y="CC_Strength", x="FineAggregate", hue="FlyAsh", size="Superplasticizer
                data=data, ax=ax, sizes=(50, 300))
ax.set_title("CC Strength vs (Fine aggregate, Super Plasticizer, FlyAsh)")
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
plt.show()
```



Observations from CC Strength vs (Fine aggregate, Super Plasticizer, FlyAsh) As Flyash increases the strength decreases Strength increases with Super plasticizer

In [16]:

```
fig, ax = plt.subplots(figsize=(8,7))
sns.scatterplot(y="CC_Strength", x="FineAggregate", hue="Water", size="Superplasticizer"
                data=data, ax=ax, sizes=(50, 300))
ax.set_title("CC Strength vs (Fine aggregate, Super Plasticizer, Water)")
ax.legend(loc="upper left", bbox_to_anchor=(1,1))
plt.show()
```



Observations from CC Strength vs (Fine aggregate, Super Plasticizer, Water) Strength decreases with increase in water, strength increases with increase in Super plasticizer (already from above plots) More Fine aggregate is used when less water, more Super plasticizer is used.

Although we are making conclusions by observing the scatter plots, there is an underlying non linear interaction between features which we cannot visualize.

We can visually understand 2D, 3D and max upto 4D plots (by 4D I mean color and size represented by features) as shown above, we can further use row wise and column wise plotting features by seaborn to do further analysis, but still we lack the ability to track all these correlations by ourselves. For this reason, we can turn to Machine Learning to capture these relations and give better insights into the problem.

Fom here we will start processing the data and feed it to machine learning models to correctly predict the Compressive Strength of Concrete given the input features.

Data Preprocessing

In [17]:

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

Splitting data into Training and Test splits.

In [18]:

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

Scaling

Standardizing the data i.e. to rescale the features to have a mean of zero and standard deviation of 1.

In [19]:

```
sc = StandardScaler()

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

The scaler is fit on the training data and not on testing data. Since, we are training our model on rescaled Training data and the model performs well when the testing data follows same distribution. And if the scaler is fit on testing data again, this would result in testing data having a different mean and standard deviation. Resulting in loss of performance.

Model Building

Training Machine Learning Algorithms on the training data and making predictions on Test data.

Linear Regression

- The Go-to method for Regression problems.
- The Algorithm assigns coefficients to each input feature to form a linear relation between input features and target variable, so as to minimize an objective function.
- The objective function used in this case is Mean Squared Error.
- There are three versions of Linear Regression
  - Linear Regression - No regularisation
  - Lasso Regression - L1 regularisation (Tries to push coefficients to zero)
  - Ridge Regression - L2 regularisation (Tries to keep coefficients as low as possible)

We will compare these three algorithms

In [20]:

```
# 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)
```

Evaluation

Comparing the Root Mean Squared Error (RMSE), Mean Squared Error (MSE), Mean Absolute Error(MAE) and R2 Score.

In [21]:

```
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)))
```

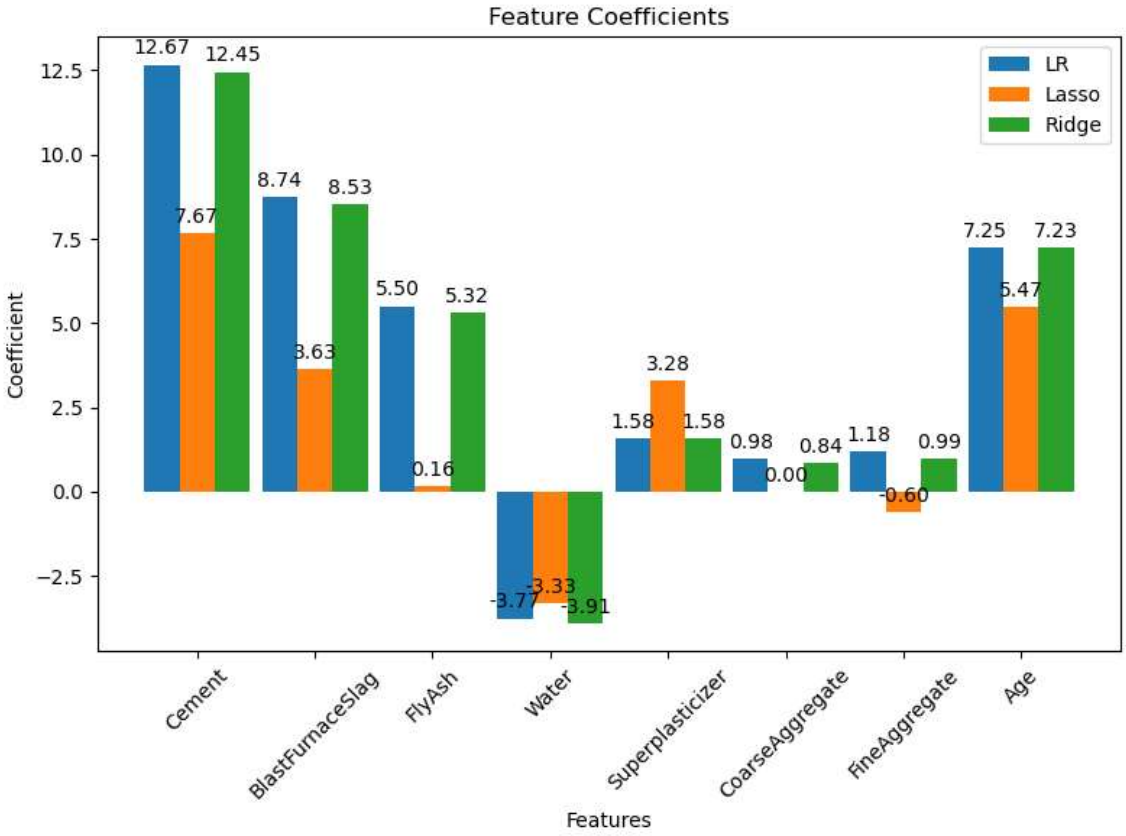
```
Model                    RMSE            MSE              MAE
R2
LinearRegression         10.29           105.78           8.23
0.57
LassoRegression          10.68           114.13           8.66
0.54
RidgeRegression          10.29           105.86           8.24
0.57
```

The performance seem to be similar with all the three methods.

Plotting the coefficients

In [22]:

```
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=(8,6))
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()
```

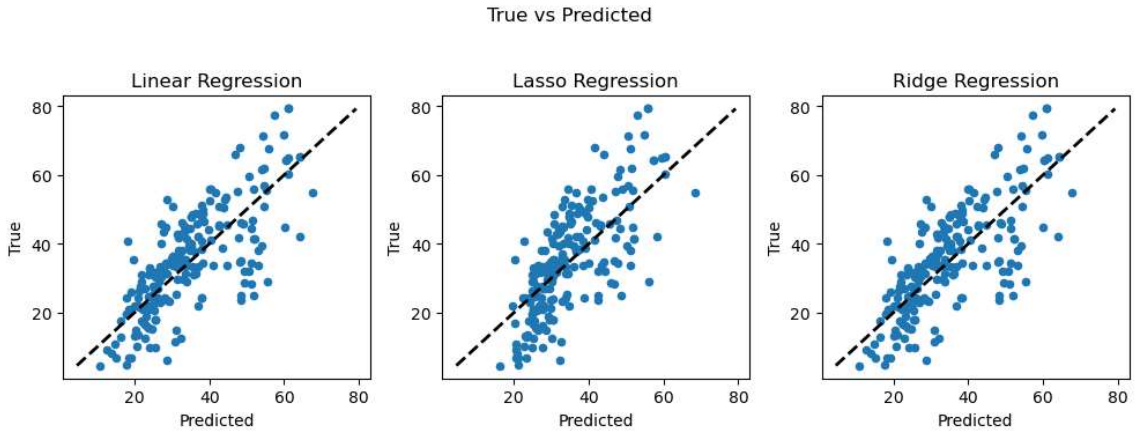


Lasso Regression, reduces the complexity of the model by keeping the coefficients as low as possible. Also, Coefficients with Linear and Ridge are almost same.

Plotting Predictions

In [23]:

```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(10,4))

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])
```



Looking at the graphs between predicted and true values of the target variable, we can conclude that Linear and Ridge Regression perform well as the predictions are closer to the actual values. While Lasso Regression reduces the complexity at the cost of loosing performance in this case. (The closer the points are to the black line, the less the error is.)

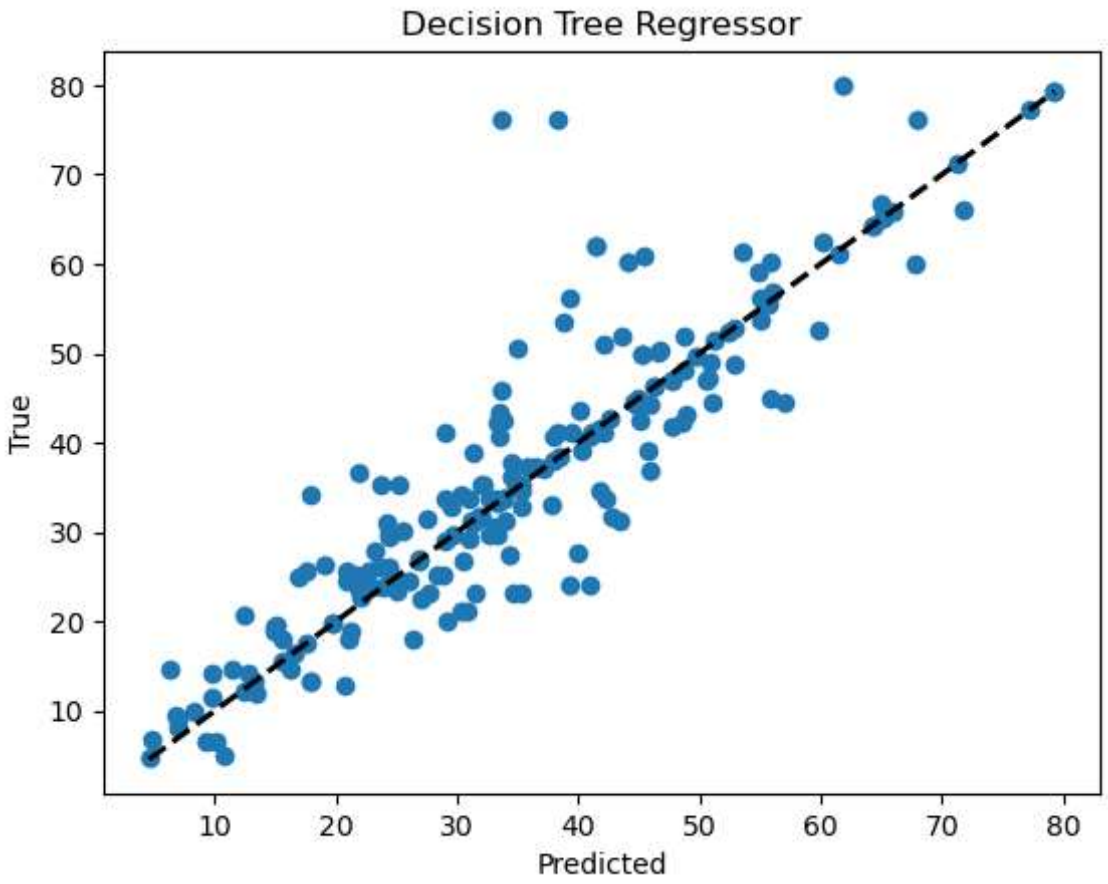
Decision Trees

Another algorithm that would give better performance in this case would be Decision Trees, since we have a lot of zeros in some of the input features as seen from their distributions in the pair plot above. This would help the decision trees build trees based on some conditions on features which can further improve performance.

In [24]:

```
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         7.37            54.27           4.54
0.78
```



The Root Mean Squared Error (RMSE) has come down from 10.29 to 7.31, so the Decision Tree Regressor has improved the performance by a significant amount. This can be observed in the plot as well as more points are on the line.

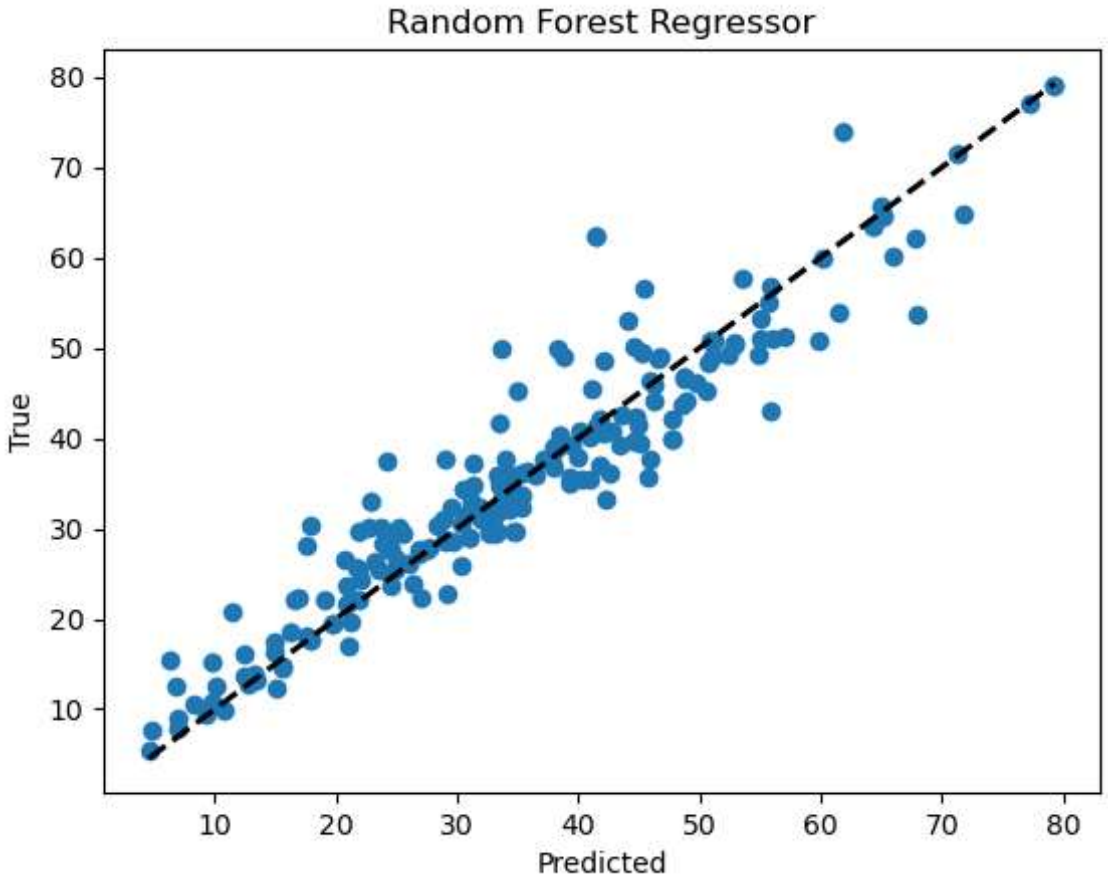
Random Forest Regressor

Since Using a Decision Tree Regressor has improved our performance, we can further improve the performance by ensembling more trees. Random Forest Regressor trains randomly initialized trees with random subsets of data sampled from the training data, this will make our model more robust.

In [25]:

```
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         5.08            25.77           3.50
0.90
```

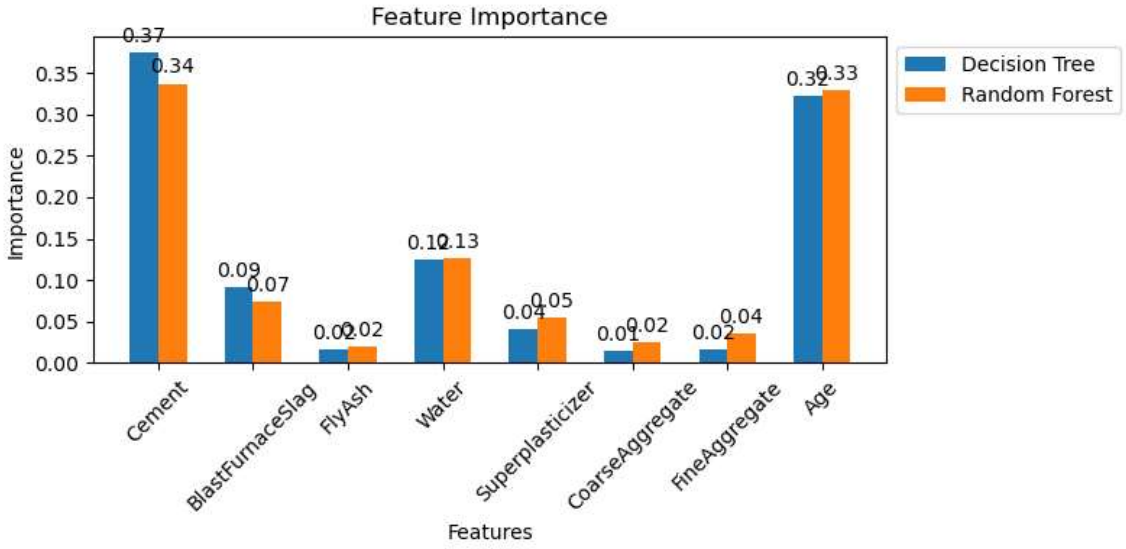


The RMSE with Random Forest Regressor is now 5.11, we have reduced the error by ensembling multiple trees.

Feature importances for Decision Tree and Random Forest

In [26]:

```
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=(8,4))
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()
```



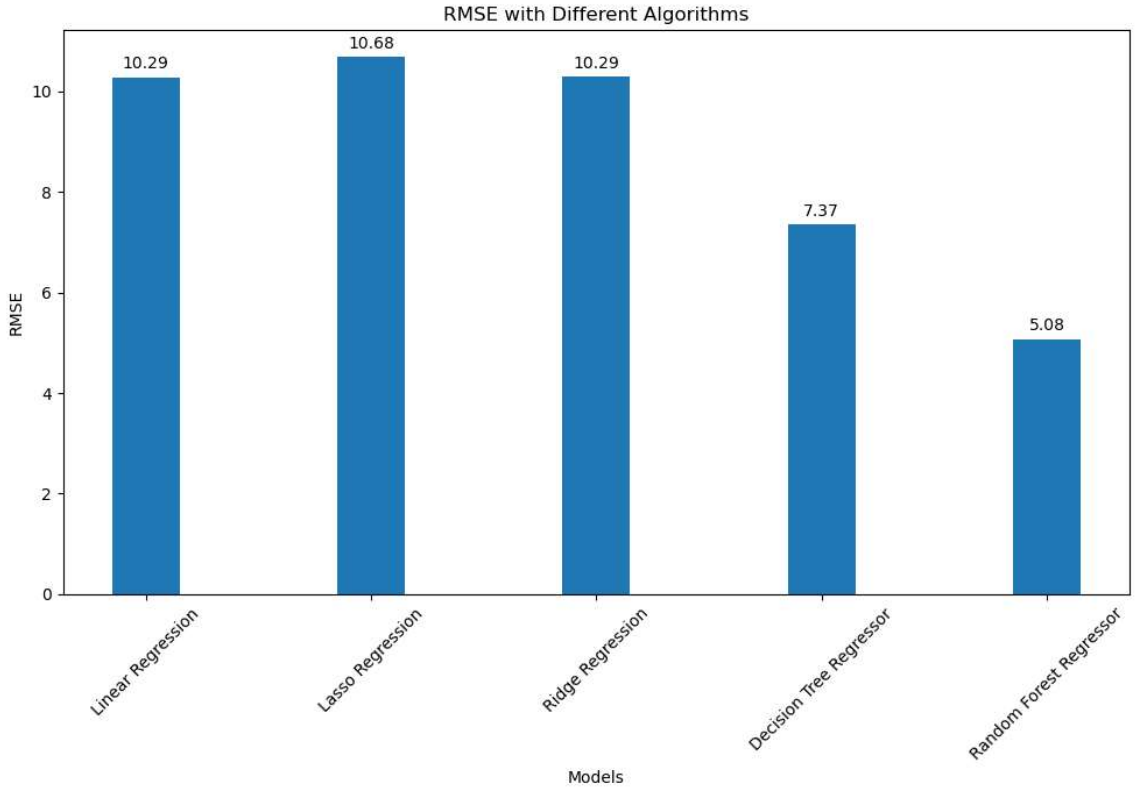
Cement and Age are treated as the most important features by tree based models. Flyash, Coarse and Fine

Comparision

Finally, lets compare the results of all the algorithms.

In [27]:

```
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()
```



Conclusion

In this study, we explored various regression models to predict concrete compressive strength. After careful evaluation, the Random Forest Regressor emerged as the optimal choice, exhibiting high accuracy with an RMSE of 2.57. By capturing feature interactions effectively, this model provides valuable insights for optimizing concrete mix designs. The importance of key features such as Cement, Water, and Superplasticizer was identified, offering valuable knowledge for civil engineering applications. This research demonstrates the transformative potential of data science in civil engineering, enabling accurate predictions and informed decision-making. The findings contribute to advancing practices and enhancing understanding in the field of concrete strength prediction.