CONCRETE STRENGTH PREDICTOR PROJECT

Literature research : The Difficulties and Proccess of High Performance Concrete

Concrete strength is very important when it comes to designing and laying out blueprints for civil and industrial work. Concrete has been a very important material throughout the years dating way back to the roman empire. For centuries there have been different types of concrete from using different types of materials to strengthen and take an approach to making better well-made concrete. Although it sounds simple on paper, there are many difficulties when it comes to making a mix for high performance concrete. There are many factors that can conclude if a batch of mix is well suited for high performance use. For example, its long-term use, the yield stress, plastic viscosity, mix design, the strength of the concrete, and overall quality. The ratio between high and low performance is quite different when it comes to using certain materials and measurements but why is this?

High performance concrete is not as simple as low performance concrete. Is it because of the quality of the material? Not quite, it primarily points towards the ratios and measurements of what materials are used. When it comes to making high performance concrete what is very important is the “Mix Design” for HPC (High Performance Concrete). There are many behaviors that affect how concrete will perform based on the mix design used. This is hat complicates and puts bumps towards making a strong high performance concrete. Where this comes into play is the strength of cement- aggregate where the mix between many variables like water ash, cement, and age can be less clear towards making HPC.

Another point that is very critical in producing HPC is the material content. There are many different types of concretes that have evolved and changed throughout history, for example back during the roman empire are famous for having made on of the strongest HPC. With their supply of volcanic rock due to their geographic location they were able to make concrete very strong due to the minerals and properties that they had put into it. This led to high quality durable concrete that was used for architectural means as proven with their amazing structural work like the Roman Colosseum and their art in forms of statues. So how does this translate to modern day concrete mix? Well for experimental and project reasons we use Cement, blast furnace slag, water, coarse aggregate, fine aggregate, and many other materials which are the building blocks to making and achieving High Performance Concrete. Although we have the material available it still leads back to the amount and measurements of how much is needed. Plus, the process for making HPC with machine work needs to be programmed so there is no error, but that is a whole other facet of the process towards achieving high performance concrete.

To conclude, high performance concrete is a very tedious and difficult process. Which has many variables that play major roles in the high-performance concrete. With one of the most important points being the measurements and mix design for the HPC being complicated for having to know how much water to cement ratio there needs to be. As well as using quality material to ensure durable concrete that can be used for projects of any type of liking. There can be many bumps and difficulties along the way but making high performance concrete is a mist in our modern age.

ANALYZING OUR DATA

In [68]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

In [33]:
```
data=pd.read_csv('concrete data.csv')
data.head()
```

Out[33]:

	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)	Age (day)	Concrete compressive strength(MPa, megapascals)
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 [34]:
```
# Rename columns if necessary (example names)
datav2 = data.columns = [
    'Cement', 'BlastFurnaceSlag', 'FlyAsh', 'Water', 'Superplasticizer',
    'CoarseAggregate', 'FineAggregate', 'Age', 'CompressiveStrength'
]
data.head()
```

Out[34]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	CompressiveStrength
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 [35]:
```
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   CompressiveStrength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

In [36]:
```
data.describe()
```

Out[36]:

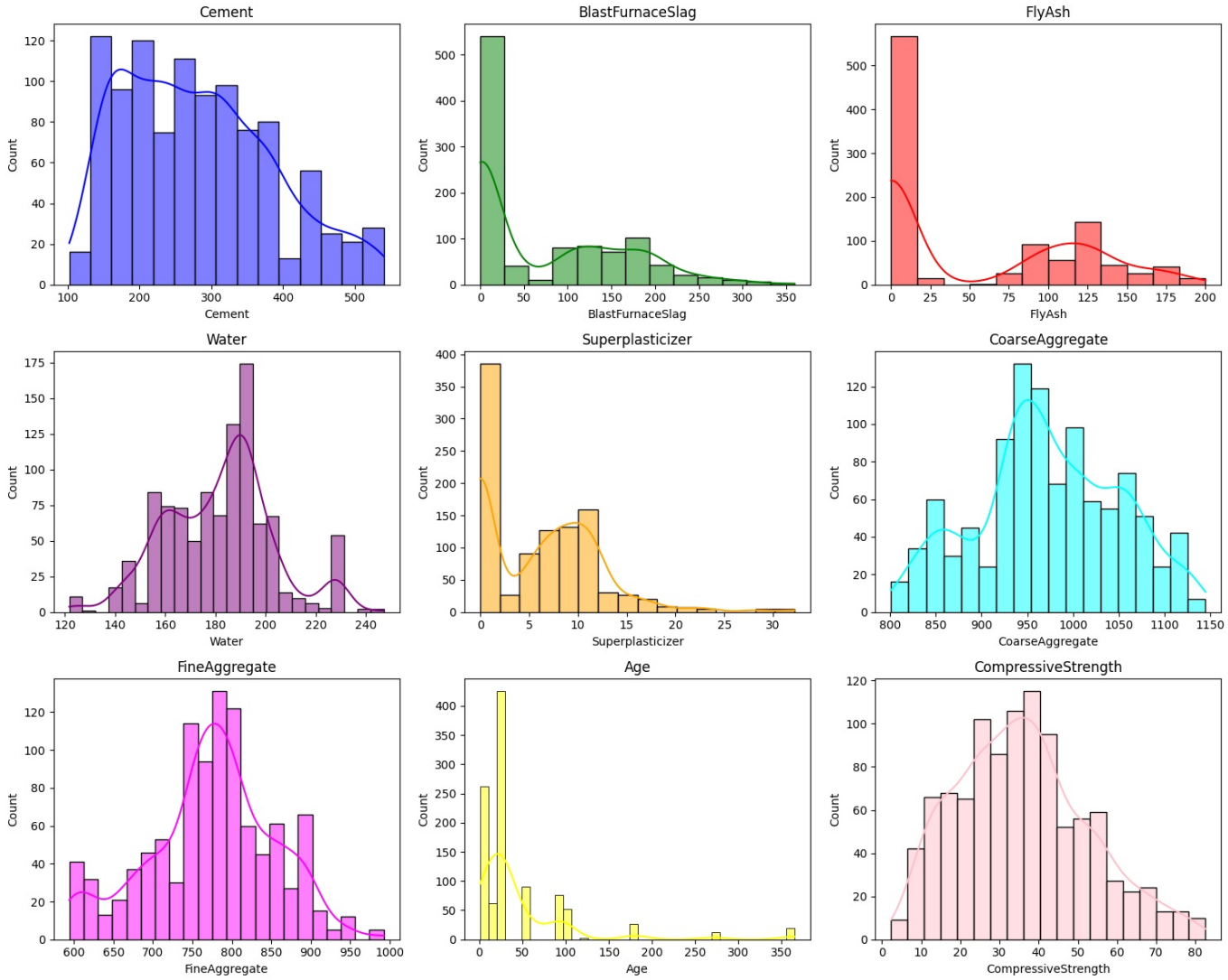
	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age	C
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.918932	773.580485	45.662136	
std	104.506364	86.279342	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	

In [37]:
```
print(data.isnull().sum())
```

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

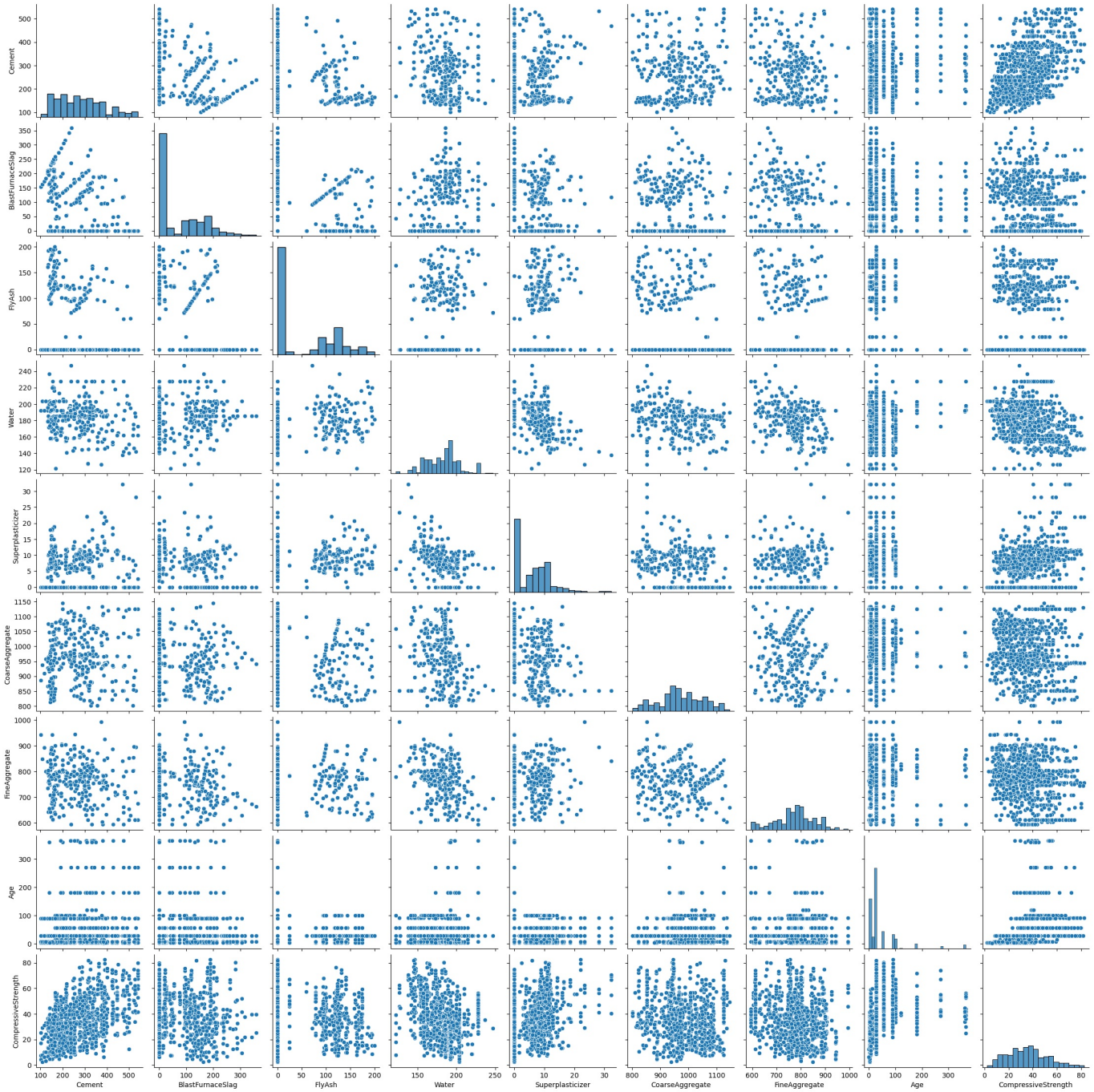
EXPLORATORY DATA ANALYSIS

In [38]:
```
# Define a list of colors
colors = ['blue', 'green', 'red', 'purple', 'orange', 'cyan', 'magenta', 'yellow', 'pink']

# Determine the number of rows and columns for the grid
num_cols = 3  # Number of columns in the grid
num_rows = (len(data.columns) + num_cols - 1) // num_cols  # Calculate rows dynamically

# Create subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 4))

# Flatten the axes array to make it easy to iterate
axes = axes.flatten()

# Plot each histogram
for i, col in enumerate(data.columns):
    sns.histplot(data[col], kde=True, color=colors[i % len(colors)], ax=axes[i])
    axes[i].set_title(col)

# Turn off unused subplots (if any)
for j in range(i + 1, len(axes)):
    axes[j].axis('off')

# Adjust layout
plt.tight_layout()
plt.show()
```



In [39]:
```
sns.pairplot(data)
```

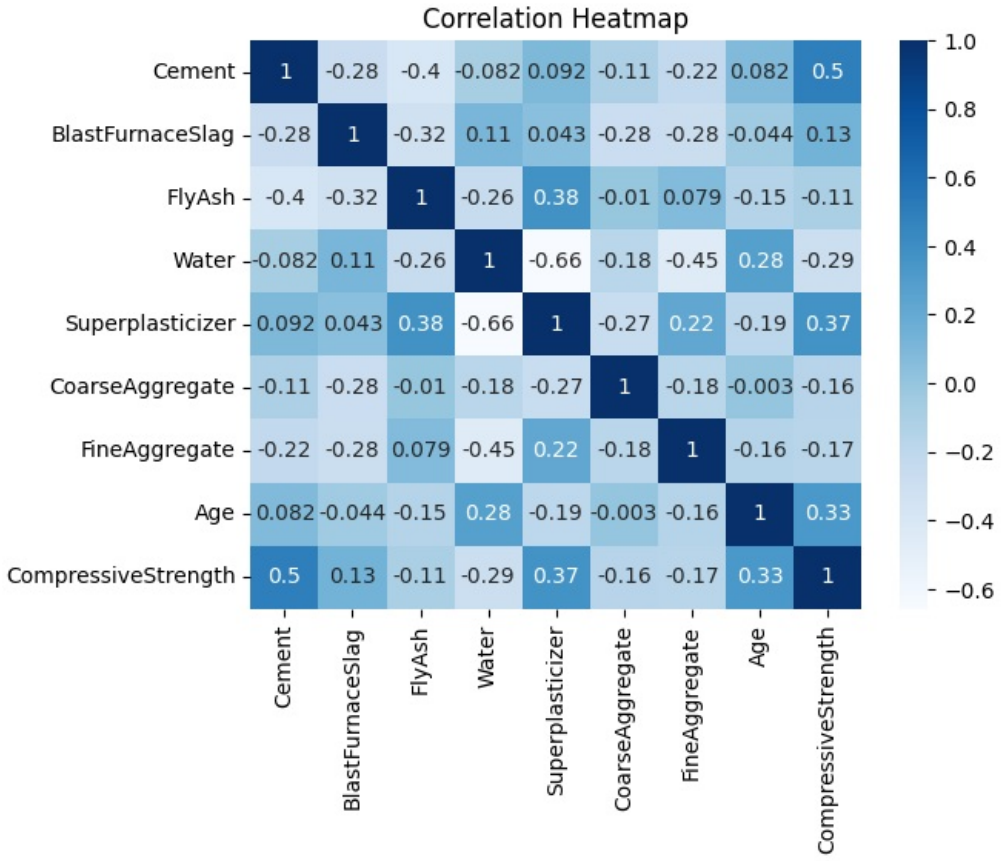
Out[39]:
```
<seaborn.axisgrid.PairGrid at 0x316279430>
```



HEAT MAPS

In [40]:
```
corr = data.corr()
sns.heatmap(corr, annot=True, cmap='Blues')
plt.title('Correlation Heatmap')
```

Out[40]:
```
Text(0.5, 1.0, 'Correlation Heatmap')
```



MODEL DATA

In [41]:
```
X = data.iloc[:, :-1]
Y = data.iloc[:, -1]
```

In [42]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,train_size=0.7)
x_train
```

Out[42]:

	Cement	BlastFurnaceSlag	FlyAsh	Water	Superplasticizer	CoarseAggregate	FineAggregate	Age
777	339.0	0.0	0.0	185.0	0.0	1069.0	754.0	28
896	313.0	161.0	0.0	178.0	10.0	917.0	759.0	28
955	272.6	0.0	89.6	198.7	10.6	931.3	762.2	28
448	167.4	129.9	128.6	175.5	7.8	1006.3	746.6	56
379	500.0	0.0	0.0	140.0	4.0	966.0	853.0	28
...	...	...	...	...	...	...	...	...
533	289.0	0.0	0.0	192.0	0.0	913.2	895.3	3
226	168.0	42.1	163.8	121.8	5.7	1058.7	780.1	28
114	362.6	189.0	0.0	164.9	11.6	944.7	755.8	7
878	133.0	210.0	0.0	196.0	3.0	949.0	795.0	28
638	375.0	0.0	0.0	186.0	0.0	1038.0	758.0	28

721 rows × 8 columns

In [43]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

```
poly=PolynomialFeatures(degree=2)
x_train_poly = poly.fit_transform(x_train)
x_train_poly.shape
```

Out[43]:
```
(721, 45)
```

In [44]:
```
x_train_poly
```

Out[44]:
```
array([[1.0000000e+00, 3.3900000e+02, 0.0000000e+00, ..., 5.6851600e+05,
        2.1112000e+04, 7.8400000e+02],
       [1.0000000e+00, 3.1300000e+02, 1.6100000e+02, ..., 5.7608100e+05,
        2.1252000e+04, 7.8400000e+02],
       [1.0000000e+00, 2.7260000e+02, 0.0000000e+00, ..., 5.8094884e+05,
        2.1341600e+04, 7.8400000e+02],
       ...,
       [1.0000000e+00, 3.6260000e+02, 1.8900000e+02, ..., 5.7123364e+05,
        5.2906000e+03, 4.9000000e+01],
       [1.0000000e+00, 1.3300000e+02, 2.1000000e+02, ..., 6.3202500e+05,
        2.2260000e+04, 7.8400000e+02],
       [1.0000000e+00, 3.7500000e+02, 0.0000000e+00, ..., 5.7456400e+05,
        2.1224000e+04, 7.8400000e+02]])
```

In [45]:
```
model=LinearRegression()
model.fit(x_train_poly,y_train)
a=model.coef_
a
```

Out[45]:
```
array([ 2.52248670e-11,  2.49252924e+00,  2.36331015e+00,  2.05992355e+00,
        1.59965509e+01,  3.04423748e+01,  2.75201001e+00,  3.90537097e+00,
       -2.89603855e-01, -3.00057796e-04, -5.23898792e-04, -2.91217791e-04,
       -4.84244240e-03, -1.12247456e-02, -5.50792813e-04, -8.78936705e-04,
        3.49131997e-04, -3.37766914e-04, -1.58927585e-04, -4.36720808e-03,
       -1.02326293e-02, -5.81564090e-04, -8.47366848e-04,  4.94447889e-04,
        4.72496207e-05, -5.34257444e-03, -1.84054724e-02, -3.23655687e-04,
       -7.16070037e-04,  8.54221847e-04, -1.04844628e-02, -3.24302078e-02,
       -5.59524914e-03, -6.13654852e-03,  6.19099265e-04, -4.13041932e-02,
       -1.03268685e-02, -1.13791434e-02,  4.94642351e-03, -3.37952802e-04,
       -1.02712601e-03, -1.79271027e-05, -8.90779366e-04,  4.23342404e-04,
       -5.97690663e-04])
```

In [46]:
```
y_train_pred=model.predict(x_train_poly)
```

In [47]:
```
x_test_poly = poly.fit_transform(x_test)
x_test_poly.shape
```

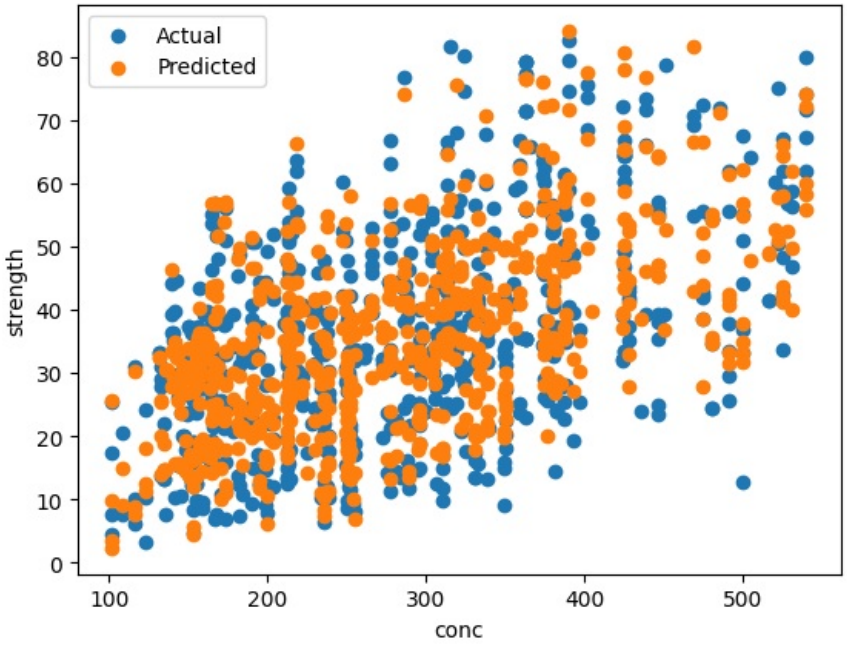
Out[47]:
```
(309, 45)
```

In [48]:
```
y_test_pred=model.predict(x_test_poly)
```

TRAIN AND TEST MODELS

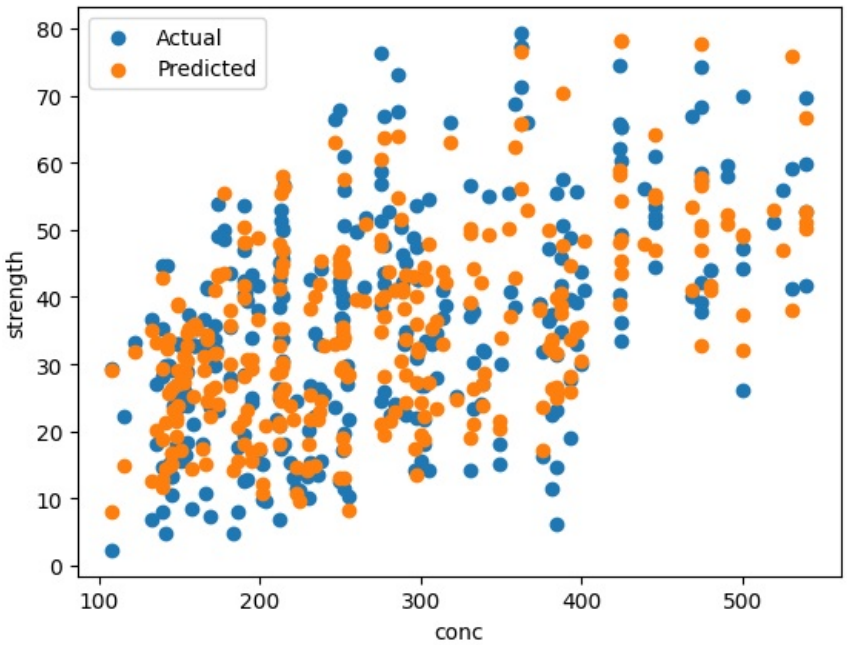
In [49]:
```
plt.scatter(x_train["Cement"],y_train)
plt.scatter(x_train["Cement"],y_train_pred)
plt.xlabel("conc")
plt.ylabel("strength")
plt.legend(["Actual","Predicted"])
```

Out[49]:
```
<matplotlib.legend.Legend at 0x31abfffe0>
```



In [50]:
```
plt.scatter(x_test["Cement"],y_test)
plt.scatter(x_test["Cement"],y_test_pred)
plt.xlabel("conc")
plt.ylabel("strength")
plt.legend(["Actual","Predicted"])
```

Out[50]: <matplotlib.legend.Legend at 0x31aa0c740>



Polynomial (Quantitive measure)

In [51]:
```
from sklearn import metrics
trainMSE=metrics.mean_squared_error(y_train,y_train_pred)
trainRMSE=np.sqrt(trainMSE)
trainRMSE
```

Out[51]:
```
np.float64(7.385508030038419)
```

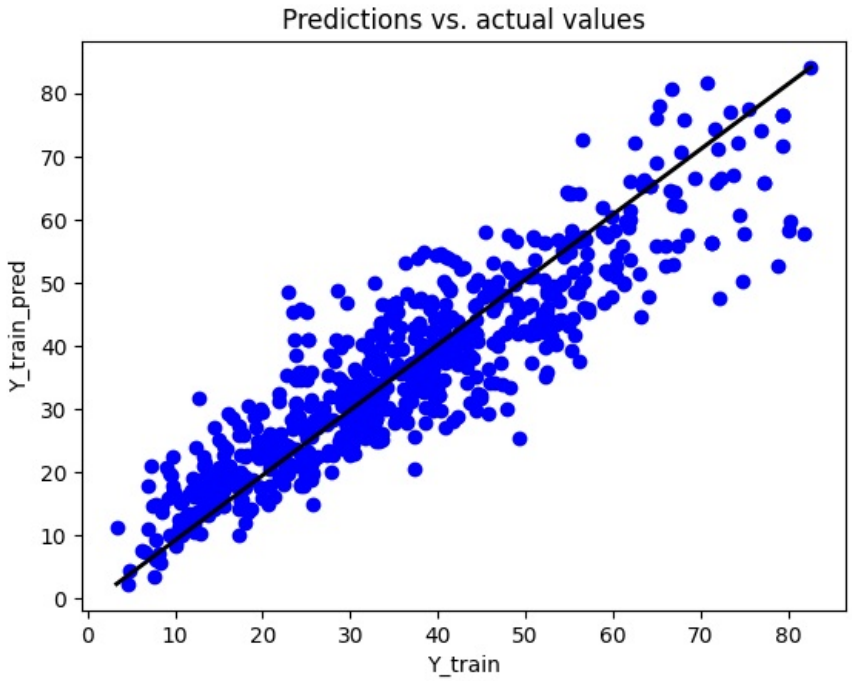
In [52]:
```
testMSE=metrics.mean_squared_error(y_test,y_test_pred)
testRMSE=np.sqrt(testMSE)
testRMSE
```

Out[52]:
```
np.float64(7.33684984709028)
```

Linear (Quantitive measure)

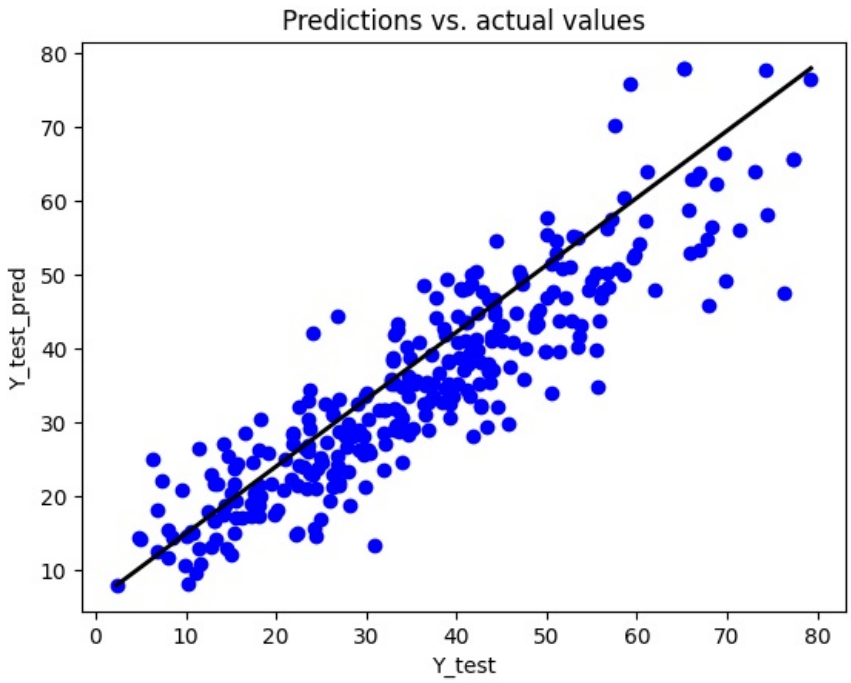
In [53]:
```
plt.scatter(y_train,y_train_pred,color='b')
plt.plot([y_train.min(), y_train.max()], [y_train_pred.min(), y_train_pred.max()], color = 'black', lw=2)
plt.xlabel("Y_train")
plt.ylabel("Y_train_pred")
plt.title("Predictions vs. actual values")
```

Out[53]:
```
Text(0.5, 1.0, 'Predictions vs. actual values')
```



In [54]:
```
plt.scatter(y_test,y_test_pred,color='b')
plt.plot([y_test.min(), y_test.max()], [y_test_pred.min(), y_test_pred.max()], color = 'black', lw=2)
plt.xlabel("Y_test")
plt.ylabel("Y_test_pred")
plt.title("Predictions vs. actual values")
```

Out[54]:
```
Text(0.5, 1.0, 'Predictions vs. actual values')
```



In [55]:
```
linear_model=LinearRegression()
linear_model.fit(x_train,y_train)
y_train_pred_lin=linear_model.predict(x_train)
y_test_pred_lin=linear_model.predict(x_test)
```

In [56]:
```
train_linRMSE=np.sqrt(metrics.mean_squared_error(y_train,y_train_pred_lin))
train_linRMSE
```

Out[56]:
```
np.float64(10.5385194634064)
```

In [57]:
```
test_linRMSE=np.sqrt(metrics.mean_squared_error(y_test,y_test_pred_lin))
test_linRMSE
```

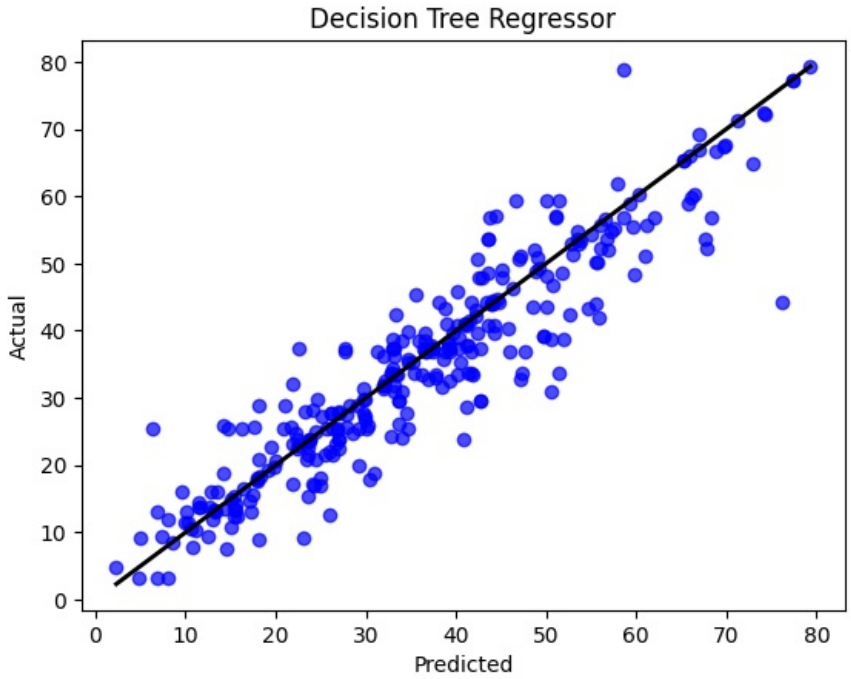
Out[57]:
```
np.float64(10.12087017008438)
```

DECISION TREES

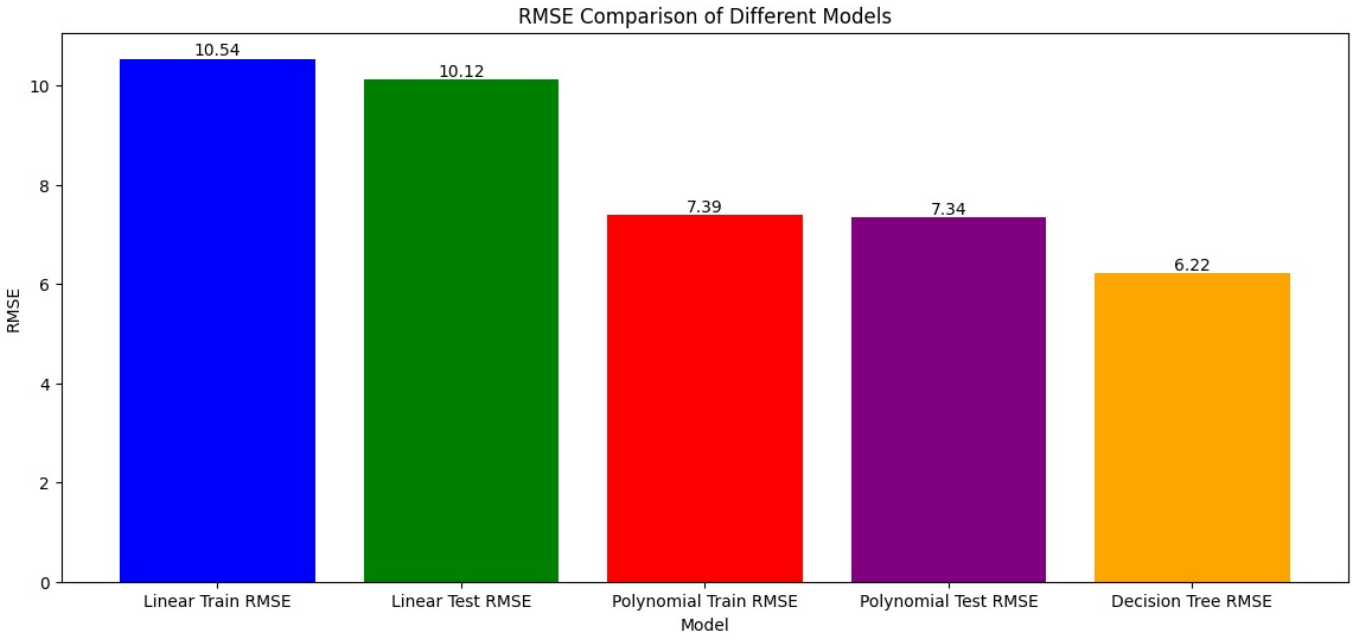
In [58]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
dtr = DecisionTreeRegressor()

dtr.fit(x_train, y_train)

y_pred_dtr = dtr.predict(x_test)

print("Model\t\t\t\t\t RMSE \t\t MSE \t\tMAE \t\tR2")
print("""DecisionTreeRegressor \t\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_pred_dtr),
            mean_absolute_error(y_test, y_pred_dtr), r2_score(y_test, y_pred_dtr)))

plt.scatter(y_test, y_pred_dtr, color='blue', alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'black', lw=2)
```

```
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Regressor')
plt.show()
```

Model	RMSE	MSE	MAE	R2
DecisionTreeRegressor	6.22	38.64	4.27	0.86



In [ ]:
```
model_names = ['Linear Train RMSE', 'Linear Test RMSE', 'Polynomial Train RMSE', 'Polynomial Test RMSE', 'Decis
rmse_values = [train_linRMSE, test_linRMSE, trainRMSE, testRMSE, np.sqrt(mean_squared_error(y_test, y_pred_dtr)

plt.figure(figsize=(14, 6))
bars = plt.bar(model_names, rmse_values, color=['blue', 'green', 'red', 'purple', 'orange'])

plt.xlabel('Model')
plt.ylabel('RMSE')
plt.title('RMSE Comparison of Different Models')

for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 2), ha='center', va='bottom')

plt.show()
```



INTERFACE

In [69]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
```

```
model = LinearRegression()
model.fit(X, y)

csv_file = 'concrete data.csv'

def predict_strength(cement, age, water):
    input_data = np.array([[cement, age, water]])
    predicted_strength = model.predict(input_data)
    return predicted_strength[0]

def save_to_csv(cement, age, water, strength):

    new_data = pd.DataFrame([[cement, age, water, strength]],
                            columns=['Cement', 'Age', 'Water', 'Strength'])
    try:
        new_data.to_csv(csv_file, mode='a', header=not pd.io.common.file_exists(csv_file), index=False)
        print("Data saved to CSV successfully!")
    except Exception as e:
        print(f"Error saving to CSV: {e}")

def main():
    print("Concrete Strength Prediction")
    print("Please enter the following concrete mixture properties:")

    cement = float(input("Enter cement (kg): "))
    age = float(input("Enter age (days): "))
    water = float(input("Enter water (kg): "))

    strength = predict_strength(cement, age, water)
    print(f"Estimated Concrete Strength: {strength:.2f} MPa")
    save_to_csv(cement, age, water, strength)

if __name__ == "__main__":
    main()
```

```
Concrete Strength Prediction
Please enter the following concrete mixture properties:
Estimated Concrete Strength: 66.90 MPa
Data saved to CSV successfully!
```