P.S-> Relationship between strength of concrete and other attributes

In [37]:
```
## Import libraries

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 [38]:
```
## Load the dataset
df=pd.read_csv("https://raw.githubusercontent.com/Mukund94/Datasets/main/concrete.c
```

In [39]:
```
## Analyze the data
df.head()
```

Out[39]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	strength
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	29.89
1	168.9	42.2	124.3	158.3	10.8	1080.8	796.2	14	23.51
2	250.0	0.0	95.7	187.4	5.5	956.9	861.2	28	29.22
3	266.0	114.0	0.0	228.0	0.0	932.0	670.0	28	45.85
4	154.8	183.4	0.0	193.3	9.1	1047.4	696.7	28	18.29

In [40]:
```
df.shape
```

Out[40]:
```
(1030, 9)
```

In [41]:
```
df.isnull().sum()
```

Out[41]:
```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

In [42]:
```
df.describe()
```

Out[42]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg
count	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
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.753954	80.175980
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.000000	594.000000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.000000	730.950000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.000000	779.500000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.400000	824.000000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.000000	992.600000

In [43]:
```
#sns.pairplot(df)
```

In [44]:
```
X=df.drop(["strength"],axis=1)

y=df[["strength"]]
```

In [45]:
```
## Split into train and test data

X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2)
```

In [46]:
```
### Fit the model

model_1=LinearRegression()
```

In [47]:
```
model_1.fit(X_train,Y_train)
```

Out[47]:
```
LinearRegression()
```

In [48]:
```
model_1.score(X_train,Y_train)
```

Out[48]:
```
0.00997712335208023
```

In [49]:
```
model_1.score(X_test,Y_test)
```

Out[49]:
```
-0.032704117574969604
```

In [50]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


import keras

from keras.models import Sequential
from keras.layers import Dense
```

In [51]:
```
n_cols=X.shape[1]
n_cols
```

Out[51]:
```
8
```

A Baseline Model

Newtwork Properties:

> Hiden Layer:1

> Nodes:10

> Activation Function:ReLU

> Optimizer:Adam

> Loss Function:Mean Squared Error

> Epochs:50

In [52]:
```
mse_A = []
r2_A = []

for i in range(50):

    #Split Data to Train and Test Set
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)

    #Create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

    #Compile model
    model.compile(optimizer='adam', loss='mean_squared_error')

    #fit the model
    model.fit(X_train, y_train, epochs=50, verbose=0)

    #predict output on test set
    y_pred = model.predict(X_test)

    mse_A.append(mean_squared_error(y_test, y_pred))
    r2_A.append(r2_score(y_test, y_pred))
```

```
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
```

In [53]:
```
print('mse_Mean: {:.2f}'.format(np.mean(mse_A)))
print('mse_StdDev: {:.2f}'.format(np.std(mse_A)))
```

```
mse_Mean: 471.70
mse_StdDev: 610.63
```

In [54]:
```
print('R^2_Mean: {:.2f}'.format(np.mean(r2_A)))
print('R^2_StdDev: {:.2f}'.format(np.std(r2_A)))
```

```
R^2_Mean: -0.70
R^2_StdDev: 2.20
```

In [ ]:
```

```

B Mode

Newtwork Properties:

> Hiden Layer:1

> Nodes:10

> Activation Function:ReLU

> Optimizer:Adam

> Loss Function:Mean Squared Error

> Epochs:50

In [56]:
```
#With Standardization the daat

X_norm = (X - X.mean()) / X.std()
X_norm.head()
```

Out[56]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age
0	-1.338367	1.600663	-0.846733	1.027091	-1.038638	-0.014391	-0.312818	-0.279597
1	-1.074268	-0.367363	1.095546	-1.089587	0.769244	1.387467	0.282123	-0.501222
2	-0.298239	-0.856472	0.648650	0.273141	-0.117958	-0.206021	1.092840	-0.279597
3	-0.145138	0.464818	-0.846733	2.174405	-1.038638	-0.526262	-1.291914	-0.279597
4	-1.209188	1.269182	-0.846733	0.549433	0.484670	0.957907	-0.958897	-0.279597

In [57]:
```
mse_A = []
r2_A = []

for i in range(50):

    #Split Data to Train and Test Set
    X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size = 0.3)

    #Create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

    #Compile model
    model.compile(optimizer='adam', loss='mean_squared_error')

    #fit the model
    model.fit(X_train, y_train, epochs=50, verbose=0)

    #predict output on test set
    y_pred = model.predict(X_test)
```

```
    mse_A.append(mean_squared_error(y_test, y_pred))
    r2_A.append(r2_score(y_test, y_pred))
```

```
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
```

In [58]:

```
print('mse_Mean: {:.2f}'.format(np.mean(mse_A)))
print('mse_StdDev: {:.2f}'.format(np.std(mse_A)))
```

```
mse_Mean: 379.55
mse_StdDev: 106.20
```

In [59]:

```
print('R^2_Mean: {:.2f}'.format(np.mean(r2_A)))
print('R^2_StdDev: {:.2f}'.format(np.std(r2_A)))
```

```
R^2_Mean: -0.36
R^2_StdDev: 0.38
```

C Models

Newtwork Properties:

> Hiden Layer:1

> Nodes:10

> Activation Function:ReLU

> Optimizer:Adam

> Loss Function:Mean Squared Error

> Epochs:50

In [60]:
```
mse_A = []
r2_A = []

for i in range(50):

    #Split Data to Train and Test Set
    X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size = 0.3)

    #Create model
    model = Sequential()
    model.add(Dense(10, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

    #Compile model
    model.compile(optimizer='adam', loss='mean_squared_error')

    #fit the model
    model.fit(X_train, y_train, epochs=100, verbose=0)

    #predict output on test set
    y_pred = model.predict(X_test)

    mse_A.append(mean_squared_error(y_test, y_pred))
    r2_A.append(r2_score(y_test, y_pred))
```

```
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 1ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 4ms/step
```

In [61]:
```
print('mse_Mean: {:.2f}'.format(np.mean(mse_A)))
print('mse_StdDev: {:.2f}'.format(np.std(mse_A)))
```

```
mse_Mean: 167.76
mse_StdDev: 18.64
```

In [62]:
```
print('R^2_Mean: {:.2f}'.format(np.mean(r2_A)))
print('R^2_StdDev: {:.2f}'.format(np.std(r2_A)))
```

```
R^2_Mean: 0.40
R^2_StdDev: 0.07
```

D Model

Newtwork Properties:

> Hiden Layer:3

> Nodes:10

> Activation Function:ReLU

> Optimizer:Adam

> Loss Function:Mean Squared Error

> Epochs:100

In [65]:
```
mse_A = []
r2_A = []

for i in range(50):

    #Split Data to Train and Test Set
    X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size = 0.3)

    #Create model
    model = Sequential()
    model.add(Dense(6, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(6, activation='relu'))
    model.add(Dense(6, activation='relu'))
    model.add(Dense(1))

    #Compile model
    model.compile(optimizer='adam', loss='mean_squared_error')

    #fit the model
    model.fit(X_train, y_train, epochs=100, verbose=0)

    #predict output on test set
    y_pred = model.predict(X_test)

    mse_A.append(mean_squared_error(y_test, y_pred))
    r2_A.append(r2_score(y_test, y_pred))
```

```
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 5ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 1s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 5ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 2ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 3ms/step
10/10 [==============================] - 0s 4ms/step
10/10 [==============================] - 0s 2ms/step
```

In [66]:
```
print('mse_Mean: {:.2f}'.format(np.mean(mse_A)))
print('mse_StdDev: {:.2f}'.format(np.std(mse_A)))
```

```
mse_Mean: 133.21
mse_StdDev: 156.84
```

In [67]:
```
print('R^2_Mean: {:.2f}'.format(np.mean(r2_A)))
print('R^2_StdDev: {:.2f}'.format(np.std(r2_A)))
```

```
R^2_Mean: 0.51
R^2_StdDev: 0.66
```

In [ ]: