Part C

Increate the number of epochs and Repeat Part B but use 100 epochs this time for training

In [1]:
```
import keras
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# Read the data
concrete_data = pd.read_csv("concrete_data.csv")
concrete_data
```

Out[2]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	Stre
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	
...	...	...	...	...	...	...	...	...	
1025	276.4	116.0	90.3	179.6	8.9	870.1	768.3	28	
1026	322.2	0.0	115.6	196.0	10.4	817.9	813.4	28	
1027	148.5	139.4	108.6	192.7	6.1	892.4	780.0	28	
1028	159.1	186.7	0.0	175.6	11.3	989.6	788.9	28	
1029	260.9	100.5	78.3	200.6	8.6	864.5	761.5	28	

1030 rows × 9 columns

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

Out[3]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age	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 [4]:
```
# Size of the data
concrete_data.shape
```

Out[4]:
```
(1030, 9)
```

In [5]:
```
concrete_data.describe()
```

Out[5]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coars Aggregat
count	1030.000000	1030.000000	1030.000000	1030.000000	1030.000000	1030.00000
mean	281.167864	73.895825	54.188350	181.567282	6.204660	972.91893
std	104.506364	86.279342	63.997004	21.354219	5.973841	77.75395
min	102.000000	0.000000	0.000000	121.800000	0.000000	801.00000
25%	192.375000	0.000000	0.000000	164.900000	0.000000	932.00000
50%	272.900000	22.000000	0.000000	185.000000	6.400000	968.00000
75%	350.000000	142.950000	118.300000	192.000000	10.200000	1029.40000
max	540.000000	359.400000	200.100000	247.000000	32.200000	1145.00000

In [6]:
```
# Sum of the null values
concrete_data.isnull().sum()
```

Out[6]:
```
Cement                0
Blast Furnace Slag    0
Fly Ash               0
Water                 0
Superplasticizer      0
Coarse Aggregate      0
Fine Aggregate        0
Age                   0
Strength              0
dtype: int64
```

Split data into predictors and target

In [7]:
```
concrete_data_columns = concrete_data.columns

# all columns except Strength
predictors = concrete_data[concrete_data_columns[concrete_data_columns

# Strength column
target = concrete_data['Strength']
```

In [8]:
```
predictors.head()
```

Out[8]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360

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

Out[9]:
```
0    79.99
1    61.89
2    40.27
3    41.05
4    44.30
Name: Strength, dtype: float64
```

In [10]:
```
predictors_norm = (predictors - predictors.mean()) / predictors.std()
predictors_norm.head()
```

Out[10]:

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	
0	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	0.862735	-1.217079	-0
1	2.476712	-0.856472	-0.846733	-0.916319	-0.620147	1.055651	-1.217079	-0
2	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	3
3	0.491187	0.795140	-0.846733	2.174405	-1.038638	-0.526262	-2.239829	5
4	-0.790075	0.678079	-0.846733	0.488555	-1.038638	0.070492	0.647569	4

In [11]:
```
# number of predictors
n_cols = predictors_norm.shape[1]
n_cols
```

Out[11]: 8

Build a Neural Network

In [13]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras import backend as K
```

In [14]:

```
# define regression model
def regression_model():
    # create model
    model = Sequential()
    model.add(Dense(50, activation='relu', input_shape=(n_cols,)))
    model.add(Dense(1))

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

Train and Test the network

In [15]:

```
# build the model
model = regression_model()
```

```
C:\Users\nensi\anaconda3\lib\site-packages\keras\src\layers\core\dens
e.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
t to a layer. When using Sequential models, prefer using an `Input(sha
pe)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwarg
s)
```

In [16]:

```
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, ve
```

```
Epoch 1/100
23/23 - 1s - 39ms/step - loss: 1674.4822 - val_loss: 1201.7513
Epoch 2/100
23/23 - 0s - 4ms/step - loss: 1630.4917 - val_loss: 1173.3882
Epoch 3/100
23/23 - 0s - 7ms/step - loss: 1585.0181 - val_loss: 1142.4117
Epoch 4/100
23/23 - 0s - 5ms/step - loss: 1535.1676 - val_loss: 1108.8983
Epoch 5/100
23/23 - 0s - 8ms/step - loss: 1480.9929 - val_loss: 1071.9869
Epoch 6/100
23/23 - 0s - 6ms/step - loss: 1419.7267 - val_loss: 1033.2502
Epoch 7/100
23/23 - 0s - 4ms/step - loss: 1353.5219 - val_loss: 989.7574
Epoch 8/100
23/23 - 0s - 4ms/step - loss: 1280.2677 - val_loss: 942.6701
Epoch 9/100
23/23 - 0s - 4ms/step - loss: 1202.1476 - val_loss: 893.1171
Epoch 10/100
```

In [18]:

```
mean = []

for i in range(50):
    def regression_model():

        concrete_data_columns = concrete_data.columns
        # all columns except Strength
        predictors = concrete_data[concrete_data_columns[concrete_data_
        # Strength column
        target = concrete_data['Strength']

        predictors_norm = (predictors - predictors.mean()) / predictors
        predictors_norm.head()

        # number of predictors
        n_cols = predictors_norm.shape[1]

        # 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')
        return model

    model = regression_model()
    model.fit(predictors_norm, target, validation_split=0.3, epochs=100

    # Calculate mse
    y_pred = model.predict(predictors_norm)
    mse = mean_squared_error(y_pred, target)
    print("Mean Squared Error is: ",mse)
    mean.append(mse)

print(mean)
```

```
Epoch 1/100

C:\Users\nensi\anaconda3\lib\site-packages\keras\src\layers\core\de
nse.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` ar
gument to a layer. When using Sequential models, prefer using an `I
nput(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwa
rgs)

23/23 - 1s - 32ms/step - loss: 1703.2744 - val_loss: 1224.7988
Epoch 2/100
23/23 - 0s - 3ms/step - loss: 1684.4481 - val_loss: 1212.5121
Epoch 3/100
23/23 - 0s - 5ms/step - loss: 1665.8829 - val_loss: 1200.3340
Epoch 4/100
23/23 - 0s - 7ms/step - loss: 1647.6312 - val_loss: 1187.8262
Epoch 5/100
23/23 - 0s - 5ms/step - loss: 1628.9502 - val_loss: 1175.1282
Epoch 6/100
23/23 - 0s - 4ms/step - loss: 1609.8618 - val_loss: 1162.1360
```

In [19]:

```
# mean of mse
mean_mse = np.mean(mean)
print("Mean of mean squared error: ",mean_mse)
```

```
Mean of mean squared error:  175.03483208219413
```

In [20]:

```
# Standard deviation of mse
std_mse = np.std(mean)
print("Standard deviation of mean squared error: ",std_mse)
```

```
Standard deviation of mean squared error:  16.789506131442614
```

In [ ]:

```

```