# Build a Regression Model in Keras _ Part C

## Result

**predictors.mean()**

| Item | Mean |
|---|---|
| Cement | 2.432224e-15 |
| Blast Furnace Slag | -8.513686e-16 |
| Fly Ash | 3.837815e-16 |
| Water | 1.846743e-15 |
| Superplasticizer | -9.641155e-16 |
| Coarse Aggregate | 6.818710e-15 |
| Fine Aggregate | 1.232571e-14 |
| Age | 3.640022e-16 |

dtype: float64

## Table of Contents



## Download and Clean Dataset

Let's start by importing the *pandas* and the Numpy libraries.

In [1]:

```
import pandas as pd
import numpy as np
```

We will be playing around with the same dataset that we used in the videos.

**The dataset is about the compressive strength of different samples of concrete based on the volumes of the different ingredients that were used to make them. Ingredients include:**

**1. Cement**

**2. Blast Furnace Slag**

**3. Fly Ash**

**4. Water**

**5. Superplasticizer**

**6. Coarse Aggregate**

**7. Fine Aggregate**

Let's download the data and read it into a *pandas* dataframe.

In [2]:

```
concrete_data = pd.read_csv('https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/DL0101EN/labs/data/concrete_data.csv')
concrete_data.head()
```

Out[2]:

| | 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 |

So the first concrete sample has 540 cubic meter of cement, 0 cubic meter of blast furnace slag, 0 cubic meter of fly ash, 162 cubic meter of water, 2.5 cubic meter of superplaticizer, 1040 cubic meter of coarse aggregate, 676 cubic meter of fine aggregate. Such a concrete mix which is 28 days old, has a compressive strength of 79.99 MPa.

**Split data into predictors and target**

The target variable in this problem is the concrete sample strength. Therefore, our predictors will be all the other columns.

In [3]:

```
concrete_data_columns = concrete_data.columns

predictors = concrete_data[concrete_data_columns[concrete_data_columns != 'Strengt
h']] # all columns except Strength
target = concrete_data['Strength'] # Strength column
```

In [11]:

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

Out[11]:

| | Cement | Blast Furnace Slag | Fly Ash | Water | Superplasticizer | Coarse Aggregate | Fine Aggregate | Age |
|---|---|---|---|---|---|---|---|---|
| **0** | 2.476712 | -0.856472 | -0.846733 | -0.916319 | -0.620147 | 0.862735 | -1.217079 | -0.279597 |
| **1** | 2.476712 | -0.856472 | -0.846733 | -0.916319 | -0.620147 | 1.055651 | -1.217079 | -0.279597 |
| **2** | 0.491187 | 0.795140 | -0.846733 | 2.174405 | -1.038638 | -0.526262 | -2.239829 | 3.551340 |
| **3** | 0.491187 | 0.795140 | -0.846733 | 2.174405 | -1.038638 | -0.526262 | -2.239829 | 5.055221 |
| **4** | -0.790075 | 0.678079 | -0.846733 | 0.488555 | -1.038638 | 0.070492 | 0.647569 | 4.976069 |

Let's do a quick sanity check of the predictors and the target dataframes.

Finally, the last step is to normalize the data by substracting the mean and dividing by the standard deviation.

Let's save the number of predictors to *n_cols* since we will need this number when building our network.

In [12]:

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

# Import Keras

Recall from the videos that Keras normally runs on top of a low-level library such as TensorFlow. This means that to be able to use the Keras library, you will have to install TensorFlow first and when you import the Keras library, it will be explicitly displayed what backend was used to install the Keras library. In CC Labs, we used TensorFlow as the backend to install Keras, so it should clearly print that when we import Keras.

#### Let's go ahead and import the Keras library

In [5]:

```
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:521: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:522: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:523: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.6/site-packages/tensorfl
ow/python/framework/dtypes.py:528: FutureWarning: Passing (type, 1) or
'1type' as a synonym of type is deprecated; in a future version of nump
y, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

As you can see, the TensorFlow backend was used to install the Keras library.

Let's import the rest of the packages from the Keras library that we will need to build our regressoin model.

In [6]:

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

# Build a Neural Network

Let's define a function that defines our regression model for us so that we can conveniently call it to create our model.

In [7]:

```
# define regression model
def regression_model():
    # 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
```

The above function create a model that has two hidden layers, each of 10 hidden units.

## Train and Test the Network

Let's call the function now to create our model.

In [8]:

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

Next, we will train and test the model at the same time using the *fit* method. We will leave out 30% of the data for validation and we will train the model for 50 epochs.

In [13]:

```
# fit the model
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/100
 - 1s - loss: 1677.8265 - val_loss: 1191.8550
Epoch 2/100
 - 0s - loss: 1659.5390 - val_loss: 1178.4561
Epoch 3/100
 - 0s - loss: 1641.2614 - val_loss: 1164.7851
Epoch 4/100
 - 0s - loss: 1622.1717 - val_loss: 1150.6548
Epoch 5/100
 - 0s - loss: 1602.4241 - val_loss: 1136.2989
Epoch 6/100
 - 0s - loss: 1581.8101 - val_loss: 1121.3557
Epoch 7/100
 - 0s - loss: 1560.3573 - val_loss: 1106.2320
Epoch 8/100
 - 0s - loss: 1537.7711 - val_loss: 1090.1466
Epoch 9/100
 - 0s - loss: 1513.8376 - val_loss: 1073.8485
Epoch 10/100
 - 0s - loss: 1488.6177 - val_loss: 1057.1422
Epoch 11/100
 - 0s - loss: 1462.1014 - val_loss: 1039.3880
Epoch 12/100
 - 0s - loss: 1434.7787 - val_loss: 1021.2821
Epoch 13/100
 - 0s - loss: 1405.7419 - val_loss: 1002.8425
Epoch 14/100
 - 0s - loss: 1376.0892 - val_loss: 984.1317
Epoch 15/100
 - 0s - loss: 1345.0303 - val_loss: 964.8836
Epoch 16/100
 - 0s - loss: 1313.1168 - val_loss: 945.2004
Epoch 17/100
 - 0s - loss: 1280.1880 - val_loss: 925.3691
Epoch 18/100
 - 0s - loss: 1246.2422 - val_loss: 905.1251
Epoch 19/100
 - 0s - loss: 1211.5559 - val_loss: 884.6067
Epoch 20/100
 - 0s - loss: 1176.4601 - val_loss: 864.2089
Epoch 21/100
 - 0s - loss: 1140.0835 - val_loss: 842.5427
Epoch 22/100
 - 0s - loss: 1103.4853 - val_loss: 821.1206
Epoch 23/100
 - 0s - loss: 1066.3388 - val_loss: 799.4557
Epoch 24/100
 - 0s - loss: 1029.3381 - val_loss: 778.0722
Epoch 25/100
 - 0s - loss: 991.3763 - val_loss: 755.5705
Epoch 26/100
 - 0s - loss: 954.1311 - val_loss: 732.9846
Epoch 27/100
 - 0s - loss: 916.3112 - val_loss: 710.9786
Epoch 28/100
 - 0s - loss: 879.2736 - val_loss: 688.3192
```

```
Epoch 29/100
 - 0s - loss: 842.1403 - val_loss: 665.6607
Epoch 30/100
 - 0s - loss: 804.9254 - val_loss: 643.7195
Epoch 31/100
 - 0s - loss: 768.9109 - val_loss: 621.3816
Epoch 32/100
 - 0s - loss: 733.6156 - val_loss: 599.0430
Epoch 33/100
 - 0s - loss: 698.9403 - val_loss: 576.6552
Epoch 34/100
 - 0s - loss: 665.0074 - val_loss: 555.4195
Epoch 35/100
 - 0s - loss: 632.7835 - val_loss: 533.8966
Epoch 36/100
 - 0s - loss: 601.6302 - val_loss: 513.0836
Epoch 37/100
 - 0s - loss: 572.0801 - val_loss: 493.1035
Epoch 38/100
 - 0s - loss: 543.8197 - val_loss: 473.4950
Epoch 39/100
 - 0s - loss: 517.2305 - val_loss: 454.6439
Epoch 40/100
 - 0s - loss: 492.2514 - val_loss: 435.6605
Epoch 41/100
 - 0s - loss: 468.1735 - val_loss: 418.7509
Epoch 42/100
 - 0s - loss: 446.1025 - val_loss: 401.9970
Epoch 43/100
 - 0s - loss: 425.3785 - val_loss: 386.3763
Epoch 44/100
 - 0s - loss: 406.7299 - val_loss: 370.4711
Epoch 45/100
 - 1s - loss: 388.7329 - val_loss: 356.0838
Epoch 46/100
 - 0s - loss: 372.1676 - val_loss: 342.4839
Epoch 47/100
 - 0s - loss: 357.4099 - val_loss: 329.5617
Epoch 48/100
 - 0s - loss: 343.6867 - val_loss: 317.5464
Epoch 49/100
 - 0s - loss: 331.1482 - val_loss: 306.4303
Epoch 50/100
 - 0s - loss: 319.8935 - val_loss: 295.6751
Epoch 51/100
 - 0s - loss: 309.5716 - val_loss: 285.1688
Epoch 52/100
 - 0s - loss: 299.8502 - val_loss: 276.5733
Epoch 53/100
 - 0s - loss: 291.2833 - val_loss: 267.5215
Epoch 54/100
 - 0s - loss: 283.0051 - val_loss: 259.5265
Epoch 55/100
 - 0s - loss: 275.8261 - val_loss: 251.2975
Epoch 56/100
 - 0s - loss: 268.9292 - val_loss: 244.2703
Epoch 57/100
```

```
 - 0s - loss: 262.8301 - val_loss: 238.0293
Epoch 58/100
 - 0s - loss: 257.1806 - val_loss: 231.9414
Epoch 59/100
 - 0s - loss: 252.0734 - val_loss: 225.8488
Epoch 60/100
 - 0s - loss: 247.2302 - val_loss: 221.0155
Epoch 61/100
 - 0s - loss: 242.9895 - val_loss: 215.6814
Epoch 62/100
 - 0s - loss: 238.7874 - val_loss: 211.2504
Epoch 63/100
 - 0s - loss: 235.1996 - val_loss: 206.7252
Epoch 64/100
 - 0s - loss: 231.4788 - val_loss: 203.5827
Epoch 65/100
 - 0s - loss: 228.2762 - val_loss: 199.9474
Epoch 66/100
 - 0s - loss: 225.2439 - val_loss: 196.6950
Epoch 67/100
 - 0s - loss: 222.4033 - val_loss: 193.6582
Epoch 68/100
 - 0s - loss: 219.8163 - val_loss: 190.7742
Epoch 69/100
 - 0s - loss: 217.2152 - val_loss: 187.8773
Epoch 70/100
 - 0s - loss: 214.8543 - val_loss: 185.7493
Epoch 71/100
 - 0s - loss: 212.6010 - val_loss: 183.0075
Epoch 72/100
 - 0s - loss: 210.4122 - val_loss: 181.0957
Epoch 73/100
 - 0s - loss: 208.4526 - val_loss: 178.3978
Epoch 74/100
 - 0s - loss: 206.3812 - val_loss: 176.7312
Epoch 75/100
 - 0s - loss: 204.4966 - val_loss: 174.3474
Epoch 76/100
 - 0s - loss: 202.7658 - val_loss: 172.5921
Epoch 77/100
 - 0s - loss: 200.9462 - val_loss: 170.7816
Epoch 78/100
 - 0s - loss: 199.3701 - val_loss: 168.8103
Epoch 79/100
 - 0s - loss: 197.7140 - val_loss: 167.2761
Epoch 80/100
 - 0s - loss: 196.0644 - val_loss: 165.7606
Epoch 81/100
 - 0s - loss: 194.5059 - val_loss: 164.4676
Epoch 82/100
 - 0s - loss: 192.9682 - val_loss: 163.5476
Epoch 83/100
 - 0s - loss: 191.4791 - val_loss: 161.8815
Epoch 84/100
 - 0s - loss: 190.0479 - val_loss: 160.7672
Epoch 85/100
 - 0s - loss: 188.6127 - val_loss: 159.4149
```

```
Epoch 86/100
 - 0s - loss: 187.3013 - val_loss: 158.1137
Epoch 87/100
 - 0s - loss: 185.7669 - val_loss: 156.7247
Epoch 88/100
 - 0s - loss: 184.4714 - val_loss: 155.0459
Epoch 89/100
 - 0s - loss: 183.0652 - val_loss: 153.8989
Epoch 90/100
 - 0s - loss: 181.7297 - val_loss: 153.0871
Epoch 91/100
 - 0s - loss: 180.4391 - val_loss: 151.9932
Epoch 92/100
 - 0s - loss: 179.0818 - val_loss: 150.9210
Epoch 93/100
 - 0s - loss: 177.7965 - val_loss: 149.5498
Epoch 94/100
 - 0s - loss: 176.6005 - val_loss: 148.1491
Epoch 95/100
 - 0s - loss: 175.3019 - val_loss: 147.0728
Epoch 96/100
 - 0s - loss: 173.9420 - val_loss: 146.1712
Epoch 97/100
 - 0s - loss: 172.7150 - val_loss: 145.1739
Epoch 98/100
 - 0s - loss: 171.5095 - val_loss: 144.3322
Epoch 99/100
 - 0s - loss: 170.2768 - val_loss: 143.6150
Epoch 100/100
 - 0s - loss: 169.0754 - val_loss: 142.4218
```

Out[13]:

```
<keras.callbacks.History at 0x7fb3982e8cf8>
```

**You can refer to this [link](https://keras.io/models/sequential/) to learn about other functions that you can use for prediction or evaluation.**

In [14]:

```
model = regression_model()
```

In [15]:

```
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/100
 - 1s - loss: 1693.3338 - val_loss: 1246.3551
Epoch 2/100
 - 0s - loss: 1673.6435 - val_loss: 1234.3379
Epoch 3/100
 - 0s - loss: 1654.4907 - val_loss: 1222.6707
Epoch 4/100
 - 0s - loss: 1635.5999 - val_loss: 1211.7981
Epoch 5/100
 - 0s - loss: 1616.7576 - val_loss: 1201.0943
Epoch 6/100
 - 0s - loss: 1598.1833 - val_loss: 1190.3216
Epoch 7/100
 - 0s - loss: 1579.2152 - val_loss: 1179.7291
Epoch 8/100
 - 0s - loss: 1559.8664 - val_loss: 1169.3345
Epoch 9/100
 - 0s - loss: 1540.4792 - val_loss: 1158.4123
Epoch 10/100
 - 0s - loss: 1520.2574 - val_loss: 1147.9544
Epoch 11/100
 - 0s - loss: 1499.7094 - val_loss: 1137.2910
Epoch 12/100
 - 0s - loss: 1478.6839 - val_loss: 1126.2637
Epoch 13/100
 - 0s - loss: 1457.1567 - val_loss: 1115.1911
Epoch 14/100
 - 0s - loss: 1435.1683 - val_loss: 1103.8629
Epoch 15/100
 - 0s - loss: 1412.6598 - val_loss: 1091.9747
Epoch 16/100
 - 0s - loss: 1389.4889 - val_loss: 1079.8916
Epoch 17/100
 - 0s - loss: 1365.2619 - val_loss: 1067.3732
Epoch 18/100
 - 0s - loss: 1340.4334 - val_loss: 1053.8305
Epoch 19/100
 - 0s - loss: 1314.0945 - val_loss: 1039.8512
Epoch 20/100
 - 0s - loss: 1286.1821 - val_loss: 1024.8820
Epoch 21/100
 - 0s - loss: 1257.5668 - val_loss: 1009.2348
Epoch 22/100
 - 0s - loss: 1227.4550 - val_loss: 993.2383
Epoch 23/100
 - 0s - loss: 1196.7430 - val_loss: 976.8472
Epoch 24/100
 - 0s - loss: 1165.4294 - val_loss: 959.4753
Epoch 25/100
 - 0s - loss: 1133.6033 - val_loss: 942.0859
Epoch 26/100
 - 1s - loss: 1101.4978 - val_loss: 923.6491
Epoch 27/100
 - 0s - loss: 1068.9772 - val_loss: 905.2832
Epoch 28/100
 - 0s - loss: 1036.5242 - val_loss: 886.6015
```

```
Epoch 29/100
 - 0s - loss: 1004.3109 - val_loss: 868.4305
Epoch 30/100
 - 0s - loss: 972.5569 - val_loss: 849.4670
Epoch 31/100
 - 0s - loss: 940.7220 - val_loss: 830.4012
Epoch 32/100
 - 0s - loss: 909.4304 - val_loss: 812.2248
Epoch 33/100
 - 0s - loss: 878.4361 - val_loss: 793.1341
Epoch 34/100
 - 0s - loss: 848.1779 - val_loss: 773.8699
Epoch 35/100
 - 0s - loss: 818.1649 - val_loss: 755.6252
Epoch 36/100
 - 0s - loss: 789.2641 - val_loss: 736.8351
Epoch 37/100
 - 0s - loss: 760.3749 - val_loss: 718.3553
Epoch 38/100
 - 0s - loss: 732.4655 - val_loss: 700.2973
Epoch 39/100
 - 0s - loss: 705.8300 - val_loss: 682.3486
Epoch 40/100
 - 0s - loss: 679.4759 - val_loss: 664.3899
Epoch 41/100
 - 0s - loss: 654.2536 - val_loss: 647.2084
Epoch 42/100
 - 0s - loss: 629.4502 - val_loss: 630.1995
Epoch 43/100
 - 0s - loss: 606.1950 - val_loss: 613.3913
Epoch 44/100
 - 0s - loss: 583.5612 - val_loss: 596.9662
Epoch 45/100
 - 0s - loss: 561.8041 - val_loss: 581.0266
Epoch 46/100
 - 0s - loss: 541.0131 - val_loss: 565.2766
Epoch 47/100
 - 0s - loss: 521.0372 - val_loss: 550.4230
Epoch 48/100
 - 0s - loss: 502.1880 - val_loss: 535.4345
Epoch 49/100
 - 0s - loss: 483.8898 - val_loss: 521.1520
Epoch 50/100
 - 0s - loss: 466.4893 - val_loss: 507.6431
Epoch 51/100
 - 0s - loss: 449.9917 - val_loss: 494.2682
Epoch 52/100
 - 0s - loss: 434.3135 - val_loss: 481.0785
Epoch 53/100
 - 0s - loss: 419.4618 - val_loss: 468.0434
Epoch 54/100
 - 0s - loss: 404.9428 - val_loss: 456.3529
Epoch 55/100
 - 0s - loss: 391.5696 - val_loss: 444.2754
Epoch 56/100
 - 0s - loss: 378.8326 - val_loss: 433.2717
Epoch 57/100
```

```
 - 0s - loss: 366.6814 - val_loss: 422.4638
Epoch 58/100
 - 0s - loss: 355.2077 - val_loss: 412.5993
Epoch 59/100
 - 0s - loss: 344.4979 - val_loss: 402.4746
Epoch 60/100
 - 0s - loss: 334.2363 - val_loss: 392.8497
Epoch 61/100
 - 1s - loss: 324.3640 - val_loss: 384.3858
Epoch 62/100
 - 0s - loss: 315.2787 - val_loss: 375.7389
Epoch 63/100
 - 0s - loss: 306.8083 - val_loss: 367.4703
Epoch 64/100
 - 0s - loss: 298.5074 - val_loss: 359.8033
Epoch 65/100
 - 0s - loss: 290.7736 - val_loss: 352.5656
Epoch 66/100
 - 0s - loss: 283.5764 - val_loss: 345.6337
Epoch 67/100
 - 1s - loss: 277.0270 - val_loss: 338.5595
Epoch 68/100
 - 0s - loss: 270.4590 - val_loss: 332.4932
Epoch 69/100
 - 0s - loss: 264.3794 - val_loss: 326.7869
Epoch 70/100
 - 0s - loss: 258.8193 - val_loss: 321.3206
Epoch 71/100
 - 0s - loss: 253.4047 - val_loss: 316.1325
Epoch 72/100
 - 0s - loss: 248.1998 - val_loss: 311.3075
Epoch 73/100
 - 0s - loss: 243.5068 - val_loss: 306.5744
Epoch 74/100
 - 0s - loss: 238.9252 - val_loss: 302.0066
Epoch 75/100
 - 1s - loss: 234.6347 - val_loss: 297.8809
Epoch 76/100
 - 1s - loss: 230.5412 - val_loss: 293.8143
Epoch 77/100
 - 1s - loss: 226.6343 - val_loss: 290.5666
Epoch 78/100
 - 0s - loss: 222.8858 - val_loss: 287.4738
Epoch 79/100
 - 1s - loss: 219.3686 - val_loss: 284.0563
Epoch 80/100
 - 1s - loss: 216.0344 - val_loss: 280.9610
Epoch 81/100
 - 1s - loss: 212.9823 - val_loss: 277.9896
Epoch 82/100
 - 1s - loss: 210.0647 - val_loss: 275.3821
Epoch 83/100
 - 1s - loss: 207.3614 - val_loss: 272.9918
Epoch 84/100
 - 1s - loss: 204.7598 - val_loss: 270.0899
Epoch 85/100
 - 0s - loss: 202.3289 - val_loss: 268.2550
```

```
Epoch 86/100
 - 1s - loss: 199.9448 - val_loss: 266.0205
Epoch 87/100
 - 0s - loss: 197.6496 - val_loss: 263.8131
Epoch 88/100
 - 1s - loss: 195.4233 - val_loss: 261.5825
Epoch 89/100
 - 1s - loss: 193.2880 - val_loss: 259.5089
Epoch 90/100
 - 1s - loss: 191.1775 - val_loss: 257.5507
Epoch 91/100
 - 0s - loss: 189.1127 - val_loss: 256.2022
Epoch 92/100
 - 1s - loss: 187.0800 - val_loss: 253.9367
Epoch 93/100
 - 1s - loss: 185.1251 - val_loss: 251.9038
Epoch 94/100
 - 0s - loss: 183.1602 - val_loss: 250.3450
Epoch 95/100
 - 0s - loss: 181.2869 - val_loss: 248.4009
Epoch 96/100
 - 0s - loss: 179.4986 - val_loss: 246.1040
Epoch 97/100
 - 1s - loss: 177.6831 - val_loss: 244.2603
Epoch 98/100
 - 1s - loss: 176.0041 - val_loss: 242.3420
Epoch 99/100
 - 1s - loss: 174.2797 - val_loss: 240.8385
Epoch 100/100
 - 1s - loss: 172.6157 - val_loss: 239.0388
```

Out[15]:

```
<keras.callbacks.History at 0x7fb380708cf8>
```

In [16]:

```
model = regression_model()
```

In [17]:

```
model.fit(predictors_norm, target, validation_split=0.3, epochs=100, verbose=2)
```

```
Train on 721 samples, validate on 309 samples
Epoch 1/100
 - 2s - loss: 1731.6548 - val_loss: 1231.0840
Epoch 2/100
 - 0s - loss: 1715.8742 - val_loss: 1219.5417
Epoch 3/100
 - 1s - loss: 1699.9916 - val_loss: 1207.5979
Epoch 4/100
 - 1s - loss: 1683.7170 - val_loss: 1195.2666
Epoch 5/100
 - 1s - loss: 1666.5571 - val_loss: 1182.3532
Epoch 6/100
 - 1s - loss: 1648.5400 - val_loss: 1168.6432
Epoch 7/100
 - 1s - loss: 1629.1765 - val_loss: 1154.3221
Epoch 8/100
 - 1s - loss: 1608.8444 - val_loss: 1139.3361
Epoch 9/100
 - 1s - loss: 1586.8843 - val_loss: 1123.6221
Epoch 10/100
 - 1s - loss: 1563.8526 - val_loss: 1106.8359
Epoch 11/100
 - 1s - loss: 1539.1129 - val_loss: 1089.7654
Epoch 12/100
 - 1s - loss: 1512.7851 - val_loss: 1071.5498
Epoch 13/100
 - 0s - loss: 1485.0845 - val_loss: 1052.5979
Epoch 14/100
 - 0s - loss: 1455.7747 - val_loss: 1032.4217
Epoch 15/100
 - 0s - loss: 1424.7001 - val_loss: 1012.1413
Epoch 16/100
 - 0s - loss: 1392.4765 - val_loss: 991.0291
Epoch 17/100
 - 1s - loss: 1358.2297 - val_loss: 968.8729
Epoch 18/100
 - 1s - loss: 1322.3873 - val_loss: 946.7859
Epoch 19/100
 - 1s - loss: 1284.6444 - val_loss: 922.9460
Epoch 20/100
 - 1s - loss: 1245.1389 - val_loss: 899.5557
Epoch 21/100
 - 1s - loss: 1204.4183 - val_loss: 874.6899
Epoch 22/100
 - 1s - loss: 1162.3161 - val_loss: 850.4148
Epoch 23/100
 - 1s - loss: 1120.0551 - val_loss: 826.0299
Epoch 24/100
 - 0s - loss: 1077.2276 - val_loss: 801.2008
Epoch 25/100
 - 1s - loss: 1034.2424 - val_loss: 775.9141
Epoch 26/100
 - 0s - loss: 991.7390 - val_loss: 750.9091
Epoch 27/100
 - 1s - loss: 949.2468 - val_loss: 726.0150
Epoch 28/100
 - 1s - loss: 907.8861 - val_loss: 700.9141
```

```
Epoch 29/100
 - 1s - loss: 866.5997 - val_loss: 677.2451
Epoch 30/100
 - 0s - loss: 826.2287 - val_loss: 653.1665
Epoch 31/100
 - 1s - loss: 787.4131 - val_loss: 629.6121
Epoch 32/100
 - 1s - loss: 749.9280 - val_loss: 606.7060
Epoch 33/100
 - 1s - loss: 713.5659 - val_loss: 584.2740
Epoch 34/100
 - 1s - loss: 678.3976 - val_loss: 562.1911
Epoch 35/100
 - 1s - loss: 644.7892 - val_loss: 541.2971
Epoch 36/100
 - 1s - loss: 612.9572 - val_loss: 520.5976
Epoch 37/100
 - 1s - loss: 582.3783 - val_loss: 500.7111
Epoch 38/100
 - 0s - loss: 553.9209 - val_loss: 481.1963
Epoch 39/100
 - 1s - loss: 526.4510 - val_loss: 462.1251
Epoch 40/100
 - 1s - loss: 500.7198 - val_loss: 444.5202
Epoch 41/100
 - 1s - loss: 477.2097 - val_loss: 426.7895
Epoch 42/100
 - 0s - loss: 454.5468 - val_loss: 410.1125
Epoch 43/100
 - 1s - loss: 433.3844 - val_loss: 394.3735
Epoch 44/100
 - 1s - loss: 414.0939 - val_loss: 379.0321
Epoch 45/100
 - 1s - loss: 395.7977 - val_loss: 364.8104
Epoch 46/100
 - 1s - loss: 379.3720 - val_loss: 350.9601
Epoch 47/100
 - 1s - loss: 363.4616 - val_loss: 338.6864
Epoch 48/100
 - 0s - loss: 349.6527 - val_loss: 326.0492
Epoch 49/100
 - 0s - loss: 336.4794 - val_loss: 314.7576
Epoch 50/100
 - 1s - loss: 324.3411 - val_loss: 304.1916
Epoch 51/100
 - 0s - loss: 313.4028 - val_loss: 294.5215
Epoch 52/100
 - 1s - loss: 303.2064 - val_loss: 285.1481
Epoch 53/100
 - 1s - loss: 293.8540 - val_loss: 276.5787
Epoch 54/100
 - 1s - loss: 285.2501 - val_loss: 268.8580
Epoch 55/100
 - 1s - loss: 277.4645 - val_loss: 261.6720
Epoch 56/100
 - 1s - loss: 270.4537 - val_loss: 254.0829
Epoch 57/100
```

```
 - 1s - loss: 263.9344 - val_loss: 247.5445
Epoch 58/100
 - 1s - loss: 258.1096 - val_loss: 242.1364
Epoch 59/100
 - 1s - loss: 252.5825 - val_loss: 237.0868
Epoch 60/100
 - 1s - loss: 247.7549 - val_loss: 232.0772
Epoch 61/100
 - 0s - loss: 243.1664 - val_loss: 227.8841
Epoch 62/100
 - 1s - loss: 239.1347 - val_loss: 224.1132
Epoch 63/100
 - 1s - loss: 235.2893 - val_loss: 220.3336
Epoch 64/100
 - 1s - loss: 231.7925 - val_loss: 216.7161
Epoch 65/100
 - 0s - loss: 228.5495 - val_loss: 213.3048
Epoch 66/100
 - 1s - loss: 225.4410 - val_loss: 210.5156
Epoch 67/100
 - 0s - loss: 222.6591 - val_loss: 208.0875
Epoch 68/100
 - 1s - loss: 220.1011 - val_loss: 205.9304
Epoch 69/100
 - 1s - loss: 217.7255 - val_loss: 203.4798
Epoch 70/100
 - 0s - loss: 215.5704 - val_loss: 201.6801
Epoch 71/100
 - 1s - loss: 213.3505 - val_loss: 200.1555
Epoch 72/100
 - 1s - loss: 211.3338 - val_loss: 198.3304
Epoch 73/100
 - 1s - loss: 209.4341 - val_loss: 196.6981
Epoch 74/100
 - 0s - loss: 207.6287 - val_loss: 194.9607
Epoch 75/100
 - 1s - loss: 205.7785 - val_loss: 193.7674
Epoch 76/100
 - 1s - loss: 204.1290 - val_loss: 192.6600
Epoch 77/100
 - 1s - loss: 202.4555 - val_loss: 191.2751
Epoch 78/100
 - 1s - loss: 200.9269 - val_loss: 189.7910
Epoch 79/100
 - 1s - loss: 199.2162 - val_loss: 188.8470
Epoch 80/100
 - 1s - loss: 197.6767 - val_loss: 187.7088
Epoch 81/100
 - 1s - loss: 196.2078 - val_loss: 186.4141
Epoch 82/100
 - 1s - loss: 194.7826 - val_loss: 185.1528
Epoch 83/100
 - 0s - loss: 193.2309 - val_loss: 183.9921
Epoch 84/100
 - 0s - loss: 191.8168 - val_loss: 183.2898
Epoch 85/100
 - 3s - loss: 190.5217 - val_loss: 181.7499
```

```
Epoch 86/100
 - 3s - loss: 189.1506 - val_loss: 180.9743
Epoch 87/100
 - 1s - loss: 187.9025 - val_loss: 179.7246
Epoch 88/100
 - 1s - loss: 186.5426 - val_loss: 178.8947
Epoch 89/100
 - 1s - loss: 185.2709 - val_loss: 178.1047
Epoch 90/100
 - 0s - loss: 183.9809 - val_loss: 177.0675
Epoch 91/100
 - 1s - loss: 182.8091 - val_loss: 176.3635
Epoch 92/100
 - 1s - loss: 181.5510 - val_loss: 175.9579
Epoch 93/100
 - 1s - loss: 180.3931 - val_loss: 175.1775
Epoch 94/100
 - 0s - loss: 179.3067 - val_loss: 173.8839
Epoch 95/100
 - 0s - loss: 178.1842 - val_loss: 173.6016
Epoch 96/100
 - 1s - loss: 177.1347 - val_loss: 172.1513
Epoch 97/100
 - 1s - loss: 175.9657 - val_loss: 171.9045
Epoch 98/100
 - 0s - loss: 174.7129 - val_loss: 170.7073
Epoch 99/100
 - 1s - loss: 173.6410 - val_loss: 170.6140
Epoch 100/100
 - 1s - loss: 172.5450 - val_loss: 170.0045
```

Out[17]:

```
<keras.callbacks.History at 0x7fb36014bc18>
```

In [18]:

```
score = model.evaluate(predictors_norm, target)
```

```
1030/1030 [==============================] - 0s 89us/step
```

Feel free to vary the following and note what impact each change has on the model's performance:

1. Increase or decreate number of neurons in hidden layers
2. Add more hidden layers
3. Increase number of epochs

In [19]:

```
target.mean()
```

Out[19]:

```
35.817961165048544
```

In [20]:

```
target.std()
```

Out[20]:

```
16.705741961912512
```

In [22]:

```
predictors_norm.mean()
```

Out[22]:

```
Cement                2.432224e-15
Blast Furnace Slag   -8.513686e-16
Fly Ash               3.837815e-16
Water                 1.846743e-15
Superplasticizer     -9.641155e-16
Coarse Aggregate      6.818710e-15
Fine Aggregate        1.232571e-14
Age                   3.640022e-16
dtype: float64
```

In [23]:

```
predictors_norm.std()
```

Out[23]:

```
Cement                1.0
Blast Furnace Slag    1.0
Fly Ash               1.0
Water                 1.0
Superplasticizer      1.0
Coarse Aggregate      1.0
Fine Aggregate        1.0
Age                   1.0
dtype: float64
```

In [ ]:

```

```