Introduction

As we discussed in the videos, despite the popularity of more powerful libraries such as PyToch and TensorFlow, they are not easy to use and have a steep learning curve. So, for people who are just starting to learn deep learning, there is no better library to use other than the Keras library.

Keras is a high-level API for building deep learning models. It has gained favor for its ease of use and syntactic simplicity facilitating fast development. As you will see in this lab and the other labs in this course, building a very complex deep learning network can be achieved with Keras with only few lines of code. You will appreciate Keras even more, once you learn how to build deep models using PyTorch and TensorFlow in the other courses.

So, in this lab, you will learn how to use the Keras library to build a regression model.

Objective for this Notebook
1. How to use the Keras library to build a regression model. 2. Download and Clean dataset
3. Build a Neural Network 4. Train and Test the Network.

Table of Contents

Download and Clean Dataset

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

```
# All Libraries required for this lab are listed below. The libraries
pre-installed on Skills Network Labs are commented.
# If you run this notebook on a different environment, e.g. your
desktop, you may need to uncomment and install certain libraries.

#!pip install numpy==1.21.4
#!pip install pandas==1.3.4
#!pip install keras==2.1.6

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.

```
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()
```

```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age  Strength
0            1040.0           676.0   28     79.99
1            1055.0           676.0   28     61.89
2             932.0           594.0  270     40.27
3             932.0           594.0  365     41.05
4             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.

Let's check how many data points we have.

```
concrete_data.shape
```

```
(1030, 9)
```

So, there are approximately 1000 samples to train our model on. Because of the few samples, we have to be careful not to overfit the training data.

Let's check the dataset for any missing values.

```
concrete_data.describe()
```

```
            Cement  Blast Furnace Slag      Fly Ash        Water  \
count  1030.000000         1030.000000  1030.000000  1030.000000
mean    281.167864           73.895825    54.188350   181.567282
std     104.506364           86.279342    63.997004    21.354219
min     102.000000            0.000000     0.000000   121.800000
25%     192.375000            0.000000     0.000000   164.900000
```

```
50%     272.900000           22.000000     0.000000   185.000000
75%     350.000000          142.950000   118.300000   192.000000
max     540.000000          359.400000   200.100000   247.000000

       Superplasticizer  Coarse Aggregate  Fine Aggregate          Age
\
count       1030.000000       1030.000000     1030.000000  1030.000000

mean           6.204660        972.918932      773.580485    45.662136

std            5.973841         77.753954       80.175980    63.169912

min            0.000000        801.000000      594.000000     1.000000

25%            0.000000        932.000000      730.950000     7.000000

50%            6.400000        968.000000      779.500000    28.000000

75%           10.200000       1029.400000      824.000000    56.000000

max           32.200000       1145.000000      992.600000   365.000000


          Strength
count  1030.000000
mean     35.817961
std      16.705742
min       2.330000
25%      23.710000
50%      34.445000
75%      46.135000
max      82.600000

concrete_data.isnull().sum()

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
```

The data looks very clean and is ready to be used to build our model.

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.

```
concrete_data_columns = concrete_data.columns

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

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

```
predictors.head()
```

```
   Cement  Blast Furnace Slag  Fly Ash  Water  Superplasticizer  \
0   540.0                 0.0      0.0  162.0               2.5
1   540.0                 0.0      0.0  162.0               2.5
2   332.5               142.5      0.0  228.0               0.0
3   332.5               142.5      0.0  228.0               0.0
4   198.6               132.4      0.0  192.0               0.0

   Coarse Aggregate  Fine Aggregate  Age
0            1040.0           676.0   28
1            1055.0           676.0   28
2             932.0           594.0  270
3             932.0           594.0  365
4             978.4           825.5  360
```

```
target.head()
```

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

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

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

```
     Cement  Blast Furnace Slag   Fly Ash     Water  Superplasticizer
\
0  2.476712           -0.856472 -0.846733 -0.916319         -0.620147

1  2.476712           -0.856472 -0.846733 -0.916319         -0.620147
```

```
2  0.491187            0.795140 -0.846733  2.174405         -1.038638

3  0.491187            0.795140 -0.846733  2.174405         -1.038638

4 -0.790075            0.678079 -0.846733  0.488555         -1.038638
```

```
   Coarse Aggregate  Fine Aggregate       Age
0          0.862735       -1.217079 -0.279597
1          1.055651       -1.217079 -0.279597
2         -0.526262       -2.239829  3.551340
3         -0.526262       -2.239829  5.055221
4          0.070492        0.647569  4.976069
```

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

```
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

```
import keras
```

```
Using TensorFlow backend.
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:516: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:517: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:518: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:519: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:520: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorf
low/python/framework/dtypes.py:525: FutureWarning: Passing (type, 1)
or '1type' as a synonym of type is deprecated; in a future version of
numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:541: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:543: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:544: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:545: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/tensorb
oard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: Passing
(type, 1) or '1type' as a synonym of type is deprecated; in a future
version of numpy, 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.

```
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.

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

Train and Test the Network

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

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

```
WARNING:tensorflow:From
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
backend/tensorflow_backend.py:68: The name tf.get_default_graph is
deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
backend/tensorflow_backend.py:508: The name tf.placeholder is
deprecated. Please use tf.compat.v1.placeholder instead.

WARNING:tensorflow:From
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
backend/tensorflow_backend.py:3837: The name tf.random_uniform is
deprecated. Please use tf.random.uniform instead.

WARNING:tensorflow:From
```

```
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
optimizers.py:757: The name tf.train.Optimizer is deprecated. Please
use tf.compat.v1.train.Optimizer instead.
```

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 100 epochs.

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

```
WARNING:tensorflow:From
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
backend/tensorflow_backend.py:977: The name tf.assign_add is
deprecated. Please use tf.compat.v1.assign_add instead.

WARNING:tensorflow:From
/home/jupyterlab/conda/envs/python/lib/python3.7/site-packages/keras/
backend/tensorflow_backend.py:964: The name tf.assign is deprecated.
Please use tf.compat.v1.assign instead.

Train on 721 samples, validate on 309 samples
Epoch 1/100
```

```
2022-10-18 16:41:44.768065: I
tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports
instructions that this TensorFlow binary was not compiled to use:
SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
2022-10-18 16:41:44.774206: I
tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency:
2394320000 Hz
2022-10-18 16:41:44.775068: I
tensorflow/compiler/xla/service/service.cc:168] XLA service
0x55e28d9ae820 executing computations on platform Host. Devices:
2022-10-18 16:41:44.775131: I
tensorflow/compiler/xla/service/service.cc:175]   StreamExecutor
device (0): <undefined>, <undefined>
2022-10-18 16:41:44.878365: W
tensorflow/compiler/jit/mark_for_compilation_pass.cc:1412] (One-time
warning): Not using XLA:CPU for cluster because envvar TF_XLA_FLAGS=--
tf_xla_cpu_global_jit was not set.  If you want XLA:CPU, either set
that envvar, or use experimental_jit_scope to enable XLA:CPU.  To
confirm that XLA is active, pass --vmodule=xla_compilation_cache=1 (as
a proper command-line flag, not via TF_XLA_FLAGS) or set the envvar
XLA_FLAGS=--xla_hlo_profile.
```

```
 - 0s - loss: 1644.5117 - val_loss: 1133.1573
Epoch 2/100
 - 0s - loss: 1510.1827 - val_loss: 1004.4581
Epoch 3/100
```

```
 - 0s - loss: 1289.9682 - val_loss: 807.1376
Epoch 4/100
 - 0s - loss: 969.9264 - val_loss: 558.2431
Epoch 5/100
 - 0s - loss: 617.7082 - val_loss: 331.2072
Epoch 6/100
 - 0s - loss: 370.2036 - val_loss: 211.7572
Epoch 7/100
 - 0s - loss: 276.6057 - val_loss: 181.3697
Epoch 8/100
 - 0s - loss: 249.4148 - val_loss: 175.7381
Epoch 9/100
 - 0s - loss: 232.1977 - val_loss: 170.1015
Epoch 10/100
 - 0s - loss: 218.9256 - val_loss: 164.5616
Epoch 11/100
 - 0s - loss: 207.9249 - val_loss: 162.7383
Epoch 12/100
 - 0s - loss: 199.1976 - val_loss: 161.1574
Epoch 13/100
 - 0s - loss: 191.0211 - val_loss: 157.6680
Epoch 14/100
 - 0s - loss: 184.7556 - val_loss: 154.2926
Epoch 15/100
 - 0s - loss: 179.0151 - val_loss: 152.8809
Epoch 16/100
 - 0s - loss: 173.8244 - val_loss: 149.1393
Epoch 17/100
 - 0s - loss: 168.6981 - val_loss: 150.2035
Epoch 18/100
 - 0s - loss: 164.6236 - val_loss: 147.5037
Epoch 19/100
 - 0s - loss: 160.6826 - val_loss: 145.9333
Epoch 20/100
 - 0s - loss: 157.0267 - val_loss: 146.4106
Epoch 21/100
 - 0s - loss: 153.7032 - val_loss: 144.9643
Epoch 22/100
 - 0s - loss: 150.6735 - val_loss: 144.2074
Epoch 23/100
 - 0s - loss: 147.9630 - val_loss: 144.8231
Epoch 24/100
 - 0s - loss: 145.8764 - val_loss: 144.4690
Epoch 25/100
 - 0s - loss: 144.1202 - val_loss: 143.6746
Epoch 26/100
 - 0s - loss: 141.7900 - val_loss: 144.1100
Epoch 27/100
 - 0s - loss: 140.0928 - val_loss: 146.0033
Epoch 28/100
```

```
 - 0s - loss: 137.9570 - val_loss: 142.2927
Epoch 29/100
 - 0s - loss: 136.5367 - val_loss: 142.6384
Epoch 30/100
 - 0s - loss: 135.1474 - val_loss: 142.1617
Epoch 31/100
 - 0s - loss: 133.9130 - val_loss: 143.5150
Epoch 32/100
 - 0s - loss: 131.9723 - val_loss: 142.8274
Epoch 33/100
 - 0s - loss: 130.1442 - val_loss: 143.7100
Epoch 34/100
 - 0s - loss: 129.0413 - val_loss: 142.4068
Epoch 35/100
 - 0s - loss: 128.6205 - val_loss: 143.9766
Epoch 36/100
 - 0s - loss: 126.5897 - val_loss: 143.1071
Epoch 37/100
 - 0s - loss: 124.6350 - val_loss: 142.2858
Epoch 38/100
 - 0s - loss: 123.7001 - val_loss: 141.1261
Epoch 39/100
 - 0s - loss: 122.5572 - val_loss: 141.9369
Epoch 40/100
 - 0s - loss: 121.1075 - val_loss: 142.6121
Epoch 41/100
 - 0s - loss: 119.0717 - val_loss: 143.1569
Epoch 42/100
 - 0s - loss: 118.0639 - val_loss: 140.2584
Epoch 43/100
 - 0s - loss: 116.6109 - val_loss: 140.8614
Epoch 44/100
 - 0s - loss: 115.3587 - val_loss: 140.4326
Epoch 45/100
 - 0s - loss: 113.6630 - val_loss: 141.5242
Epoch 46/100
 - 0s - loss: 112.3467 - val_loss: 141.7154
Epoch 47/100
 - 0s - loss: 110.3507 - val_loss: 139.0487
Epoch 48/100
 - 0s - loss: 109.3578 - val_loss: 142.1259
Epoch 49/100
 - 0s - loss: 107.4637 - val_loss: 142.7773
Epoch 50/100
 - 0s - loss: 106.1409 - val_loss: 138.4710
Epoch 51/100
 - 0s - loss: 104.7192 - val_loss: 140.1595
Epoch 52/100
 - 0s - loss: 102.6359 - val_loss: 139.6243
Epoch 53/100
```

```
 - 0s - loss: 100.8389 - val_loss: 139.7685
Epoch 54/100
 - 0s - loss: 99.3945 - val_loss: 140.9836
Epoch 55/100
 - 0s - loss: 97.4844 - val_loss: 138.6600
Epoch 56/100
 - 0s - loss: 95.4376 - val_loss: 139.0452
Epoch 57/100
 - 0s - loss: 93.6261 - val_loss: 138.7159
Epoch 58/100
 - 0s - loss: 91.5072 - val_loss: 137.2660
Epoch 59/100
 - 0s - loss: 89.7870 - val_loss: 136.5427
Epoch 60/100
 - 0s - loss: 88.0969 - val_loss: 136.7688
Epoch 61/100
 - 0s - loss: 86.1229 - val_loss: 135.1855
Epoch 62/100
 - 0s - loss: 84.2942 - val_loss: 138.3378
Epoch 63/100
 - 0s - loss: 82.7624 - val_loss: 129.8187
Epoch 64/100
 - 0s - loss: 80.3903 - val_loss: 136.6328
Epoch 65/100
 - 0s - loss: 78.8802 - val_loss: 130.4530
Epoch 66/100
 - 0s - loss: 77.1153 - val_loss: 132.0732
Epoch 67/100
 - 0s - loss: 74.6922 - val_loss: 127.8946
Epoch 68/100
 - 0s - loss: 72.9344 - val_loss: 126.2705
Epoch 69/100
 - 0s - loss: 70.3852 - val_loss: 125.8484
Epoch 70/100
 - 0s - loss: 68.7041 - val_loss: 126.8786
Epoch 71/100
 - 0s - loss: 66.8664 - val_loss: 124.3058
Epoch 72/100
 - 0s - loss: 63.9074 - val_loss: 126.9071
Epoch 73/100
 - 0s - loss: 62.1078 - val_loss: 124.1286
Epoch 74/100
 - 0s - loss: 60.6102 - val_loss: 120.2987
Epoch 75/100
 - 0s - loss: 58.0878 - val_loss: 127.6730
Epoch 76/100
 - 0s - loss: 57.0545 - val_loss: 121.2124
Epoch 77/100
 - 0s - loss: 54.9581 - val_loss: 118.8324
Epoch 78/100
```

```
 - 0s - loss: 53.3270 - val_loss: 124.5582
Epoch 79/100
 - 0s - loss: 51.8133 - val_loss: 127.5327
Epoch 80/100
 - 0s - loss: 50.3724 - val_loss: 127.2597
Epoch 81/100
 - 0s - loss: 49.4950 - val_loss: 119.8470
Epoch 82/100
 - 0s - loss: 48.5576 - val_loss: 127.0639
Epoch 83/100
 - 0s - loss: 47.1856 - val_loss: 123.2462
Epoch 84/100
 - 0s - loss: 46.1058 - val_loss: 131.1689
Epoch 85/100
 - 0s - loss: 45.0241 - val_loss: 123.5716
Epoch 86/100
 - 0s - loss: 43.8843 - val_loss: 129.1709
Epoch 87/100
 - 0s - loss: 43.2061 - val_loss: 127.7864
Epoch 88/100
 - 0s - loss: 42.5982 - val_loss: 130.4927
Epoch 89/100
 - 0s - loss: 41.9426 - val_loss: 123.5607
Epoch 90/100
 - 0s - loss: 41.2141 - val_loss: 126.6237
Epoch 91/100
 - 0s - loss: 40.8171 - val_loss: 125.4455
Epoch 92/100
 - 0s - loss: 40.1707 - val_loss: 128.9629
Epoch 93/100
 - 0s - loss: 38.2329 - val_loss: 145.4464
Epoch 94/100
 - 0s - loss: 38.2433 - val_loss: 123.3293
Epoch 95/100
 - 0s - loss: 37.2148 - val_loss: 141.5565
Epoch 96/100
 - 0s - loss: 37.0780 - val_loss: 131.5692
Epoch 97/100
 - 0s - loss: 36.2031 - val_loss: 133.5341
Epoch 98/100
 - 0s - loss: 35.6939 - val_loss: 131.8094
Epoch 99/100
 - 0s - loss: 34.8598 - val_loss: 124.5743
Epoch 100/100
 - 0s - loss: 34.9298 - val_loss: 139.0503

<keras.callbacks.History at 0x7f7620852b90>
```

You can refer to this link to learn about other functions that you can use for prediction or evaluation.

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

Thank you for completing this lab!

This notebook was created by Alex Aklson. I hope you found this lab interesting and educational. Feel free to contact me if you have any questions!

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-09-21	2.0	Srishti	Migrated Lab to Markdown and added to course repo in GitLab