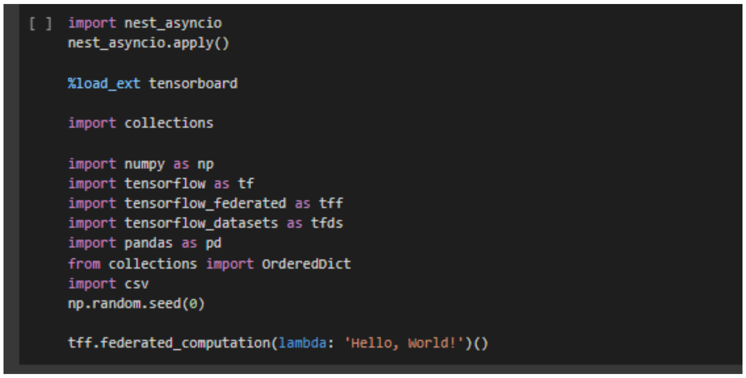
Documentation

Due to my inability to solve errors on my attempt to create federated learning based Multilayer Perceptron I have created documentations for both federated and non federated python script.

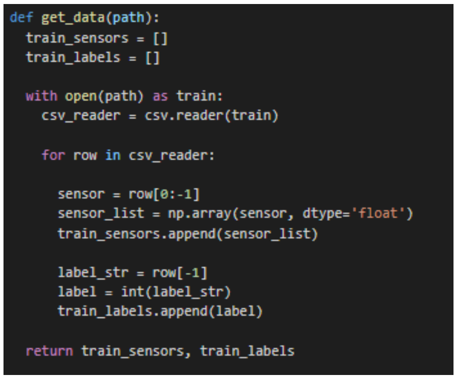
Federated (federated.py)

1. Import the libraries.



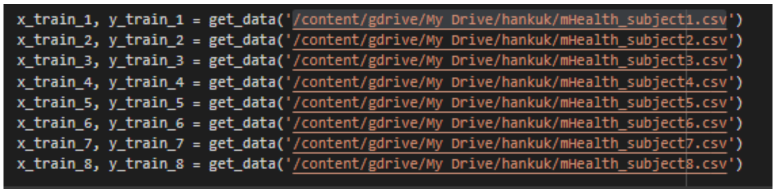
```
[ ]  import nest_asyncio
     nest_asyncio.apply()

     %load_ext tensorboard

     import collections

     import numpy as np
     import tensorflow as tf
     import tensorflow_federated as tff
     import tensorflow_datasets as tfds
     import pandas as pd
     from collections import OrderedDict
     import csv
     np.random.seed(0)

     tff.federated_computation(lambda: 'Hello, World!')()
```

2. Create function get_data() to load data, preprocess and return x (train_sensors) and y (train_labels).



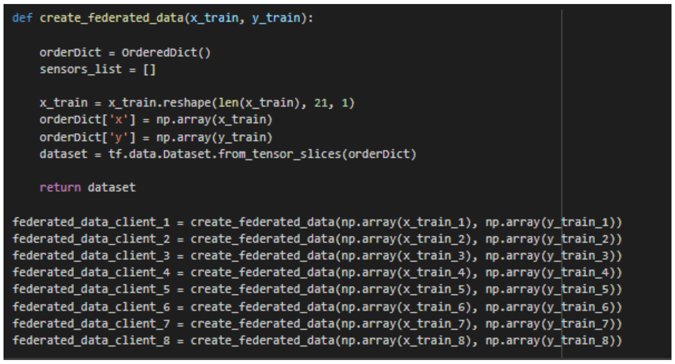
```
def get_data(path):
  train_sensors = []
  train_labels = []

  with open(path) as train:
    csv_reader = csv.reader(train)

    for row in csv_reader:

      sensor = row[0:-1]
      sensor_list = np.array(sensor, dtype='float')
      train_sensors.append(sensor_list)

      label_str = row[-1]
      label = int(label_str)
      train_labels.append(label)

  return train_sensors, train_labels
```

3. Get each client data.



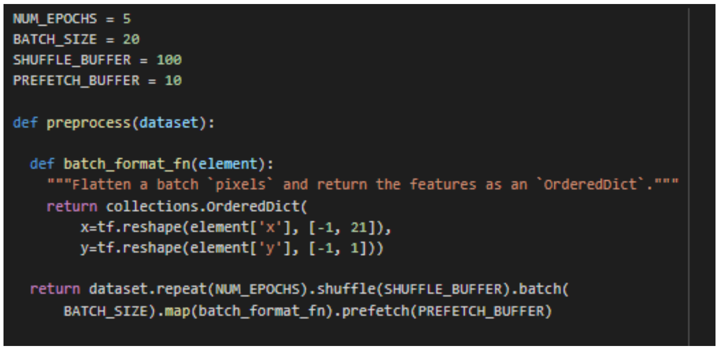
```
x_train_1, y_train_1 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject1.csv')
x_train_2, y_train_2 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject2.csv')
x_train_3, y_train_3 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject3.csv')
x_train_4, y_train_4 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject4.csv')
x_train_5, y_train_5 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject5.csv')
x_train_6, y_train_6 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject6.csv')
x_train_7, y_train_7 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject7.csv')
x_train_8, y_train_8 = get_data('/content/gdrive/My Drive/hankuk/mHealth_subject8.csv')
```

4. Create function to create federated data in the form of tf.data.Dataset



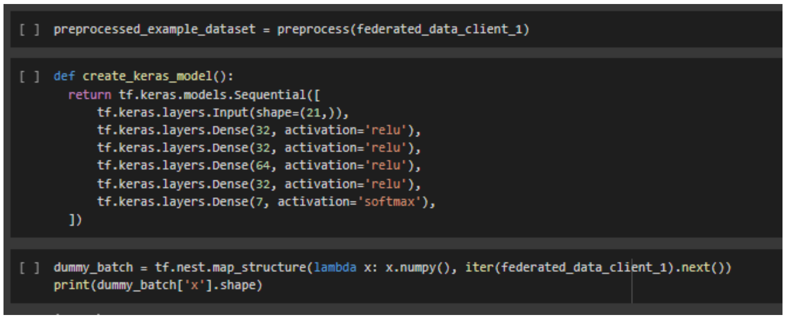
```
def create_federated_data(x_train, y_train):

    orderDict = OrderedDict()
    sensors_list = []

    x_train = x_train.reshape(len(x_train), 21, 1)
    orderDict['x'] = np.array(x_train)
    orderDict['y'] = np.array(y_train)
    dataset = tf.data.Dataset.from_tensor_slices(orderDict)

    return dataset

federated_data_client_1 = create_federated_data(np.array(x_train_1), np.array(y_train_1))
federated_data_client_2 = create_federated_data(np.array(x_train_2), np.array(y_train_2))
federated_data_client_3 = create_federated_data(np.array(x_train_3), np.array(y_train_3))
federated_data_client_4 = create_federated_data(np.array(x_train_4), np.array(y_train_4))
federated_data_client_5 = create_federated_data(np.array(x_train_5), np.array(y_train_5))
federated_data_client_6 = create_federated_data(np.array(x_train_6), np.array(y_train_6))
federated_data_client_7 = create_federated_data(np.array(x_train_7), np.array(y_train_7))
federated_data_client_8 = create_federated_data(np.array(x_train_8), np.array(y_train_8))
```

5. Create preprocess function to preprocess tf.data.Dataset and shuffle it.



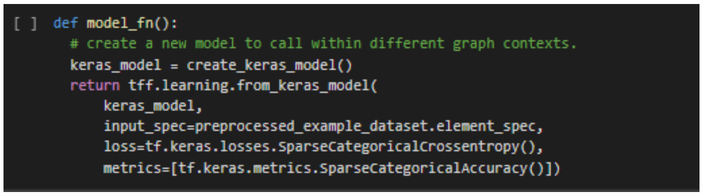
```
NUM_EPOCHS = 5
BATCH_SIZE = 20
SHUFFLE_BUFFER = 100
PREFETCH_BUFFER = 10

def preprocess(dataset):

  def batch_format_fn(element):
    """Flatten a batch `pixels` and return the features as an `OrderedDict`."""
    return collections.OrderedDict(
        x=tf.reshape(element['x'], [-1, 21]),
        y=tf.reshape(element['y'], [-1, 1]))

  return dataset.repeat(NUM_EPOCHS).shuffle(SHUFFLE_BUFFER).batch(
      BATCH_SIZE).map(batch_format_fn).prefetch(PREFETCH_BUFFER)
```

6. Testing preprocess function and create a function that create keras model.



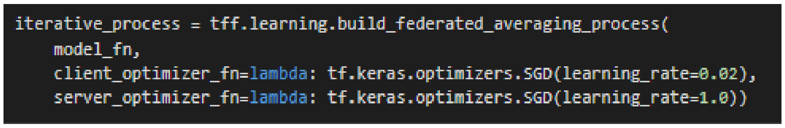
```
[ ] preprocessed_example_dataset = preprocess(federated_data_client_1)

[ ] def create_keras_model():
      return tf.keras.models.Sequential([
          tf.keras.layers.Input(shape=(21,)),
          tf.keras.layers.Dense(32, activation='relu'),
          tf.keras.layers.Dense(32, activation='relu'),
          tf.keras.layers.Dense(64, activation='relu'),
          tf.keras.layers.Dense(32, activation='relu'),
          tf.keras.layers.Dense(7, activation='softmax'),
      ])

[ ] dummy_batch = tf.nest.map_structure(lambda x: x.numpy(), iter(federated_data_client_1).next())
    print(dummy_batch['x'].shape)
```

7. Create model_fn function to instantiate keras model which will be called in tff function.



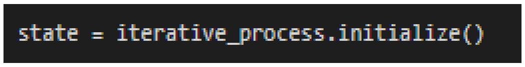
```
[ ] def model_fn():
      # create a new model to call within different graph contexts.
      keras_model = create_keras_model()
      return tff.learning.from_keras_model(
          keras_model,
          input_spec=preprocessed_example_dataset.element_spec,
          loss=tf.keras.losses.SparseCategoricalCrossentropy(),
          metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
```

8. Define the iterative process by calling tff federated averaging process function.



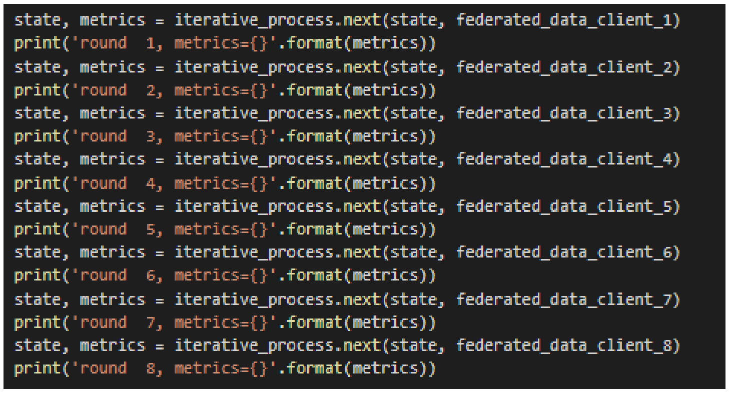
```
iterative_process = tff.learning.build_federated_averaging_process(
    model_fn,
    client_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=0.02),
    server_optimizer_fn=lambda: tf.keras.optimizers.SGD(learning_rate=1.0))
```

9. Initialize the iterative process.



```
state = iterative_process.initialize()
```

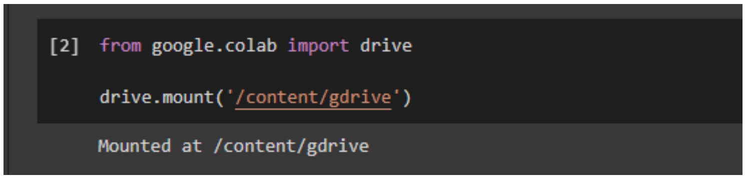
10. Do the next iterative process for the next 8 federated client data.



```
state, metrics = iterative_process.next(state, federated_data_client_1)
print('round  1, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_2)
print('round  2, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_3)
print('round  3, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_4)
print('round  4, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_5)
print('round  5, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_6)
print('round  6, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_7)
print('round  7, metrics={}'.format(metrics))
state, metrics = iterative_process.next(state, federated_data_client_8)
print('round  8, metrics={}'.format(metrics))
```

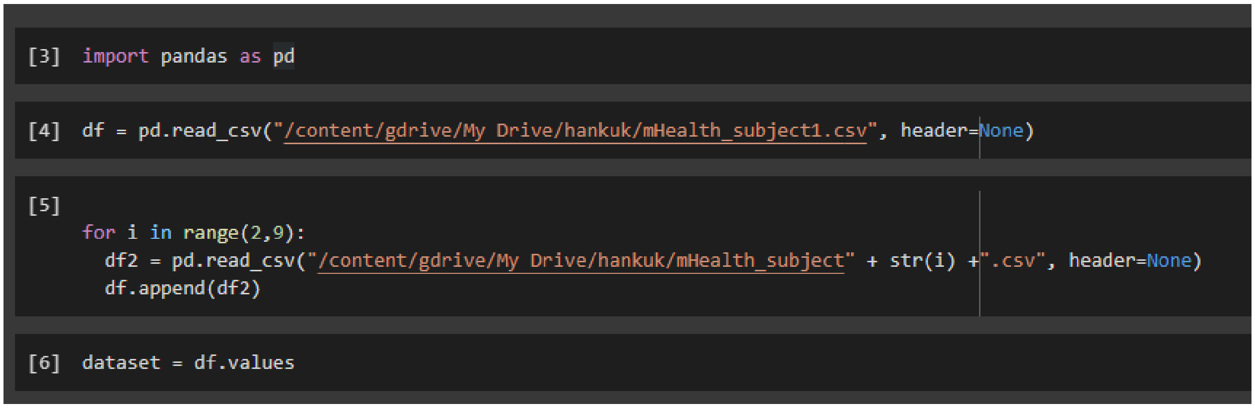
Non Federated (non_federated.py)

1. Import drive library to mount to google drive.



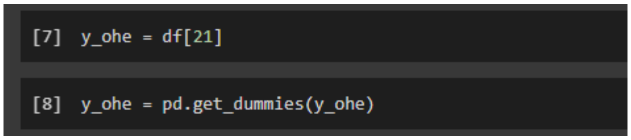
```
[2] from google.colab import drive

    drive.mount('/content/gdrive')

    Mounted at /content/gdrive
```

2. Import pandas, load the client data 1, and then iterate the process of appending dataframe after loading the next 7 client data. After that defining dataset variable by extracting df values.



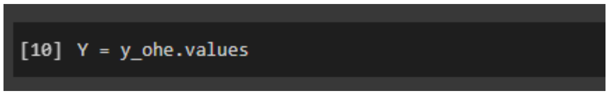
```
[3] import pandas as pd

[4] df = pd.read_csv("/content/gdrive/My Drive/hankuk/mHealth_subject1.csv", header=None)

[5]
    for i in range(2,9):
      df2 = pd.read_csv("/content/gdrive/My Drive/hankuk/mHealth_subject" + str(i) +".csv", header=None)
      df.append(df2)

[6] dataset = df.values
```

3. Defining y_ohe variable and load 22nd column of the df for label data. Then apply one hot encoding to y_ohe.



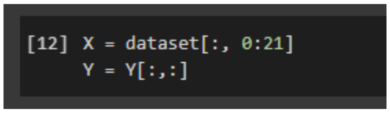
```
[7] y_ohe = df[21]

[8] y_ohe = pd.get_dummies(y_ohe)
```

4. Define variable Y and load extracted data from y_ohe.



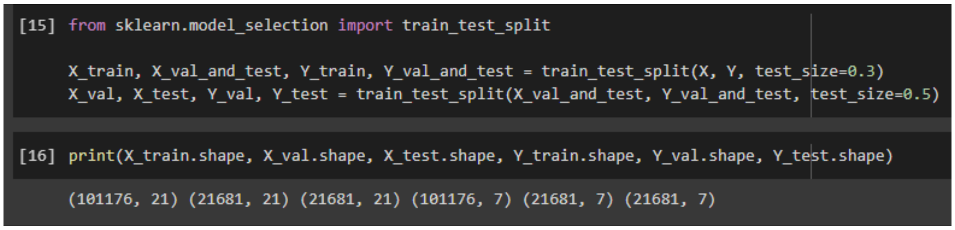
```
[10] Y = y_ohe.values
```

5. Load the 1st – 21st column from dataset into variable X, and redefine variable Y.



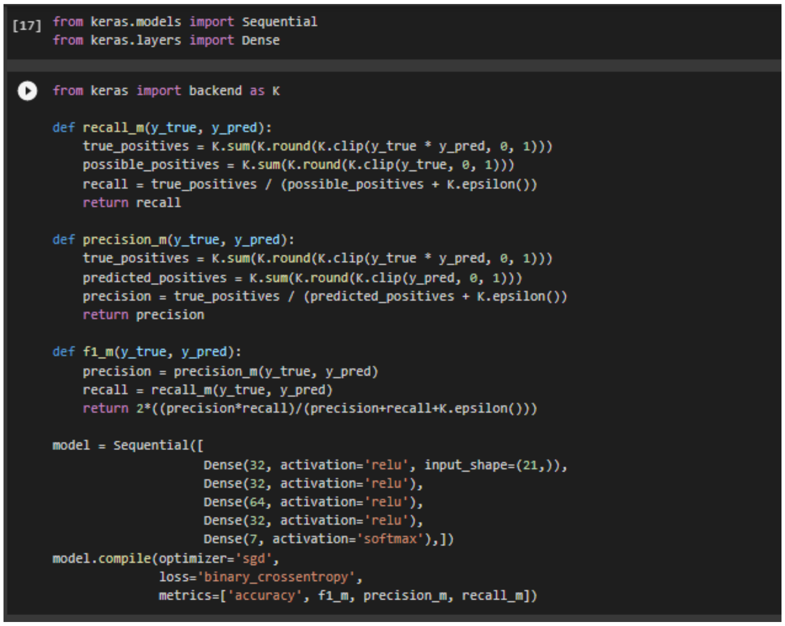
```
[12] X = dataset[:, 0:21]
     Y = Y[:,:]
```

6. Import train_test_split from sklearn and split data into train (70%), test (15%), and validation (15%).



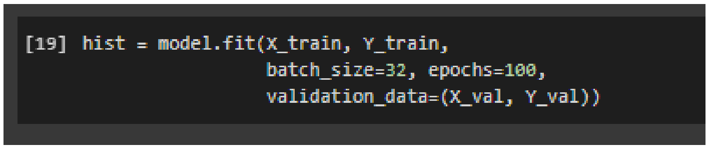
```
[15] from sklearn.model_selection import train_test_split

     X_train, X_val_and_test, Y_train, Y_val_and_test = train_test_split(X, Y, test_size=0.3)
     X_val, X_test, Y_val, Y_test = train_test_split(X_val_and_test, Y_val_and_test, test_size=0.5)

[16] print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)

     (101176, 21) (21681, 21) (21681, 21) (101176, 7) (21681, 7) (21681, 7)
```

7. Import Sequential, Dense, backend from keras. Create recall_m, precision_m and f1_m to be used for F1 score metric. Then instantiate and compile the MLP model, defining the metrics.



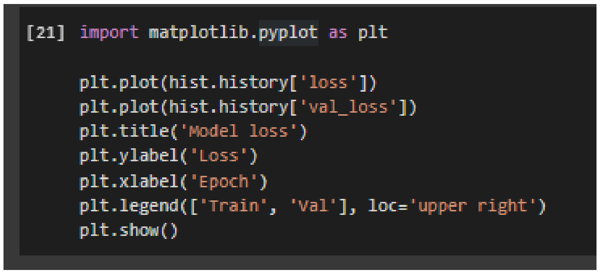
```
[17] from keras.models import Sequential
     from keras.layers import Dense

     from keras import backend as K

     def recall_m(y_true, y_pred):
         true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
         possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
         recall = true_positives / (possible_positives + K.epsilon())
         return recall

     def precision_m(y_true, y_pred):
         true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
         predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
         precision = true_positives / (predicted_positives + K.epsilon())
         return precision

     def f1_m(y_true, y_pred):
         precision = precision_m(y_true, y_pred)
         recall = recall_m(y_true, y_pred)
         return 2*((precision*recall)/(precision+recall+K.epsilon()))

     model = Sequential([
                         Dense(32, activation='relu', input_shape=(21,)),
                         Dense(32, activation='relu'),
                         Dense(64, activation='relu'),
                         Dense(32, activation='relu'),
                         Dense(7, activation='softmax'),])
     model.compile(optimizer='sgd',
                   loss='binary_crossentropy',
                   metrics=['accuracy', f1_m, precision_m, recall_m])
```

8. Begin training process with 100 epochs.



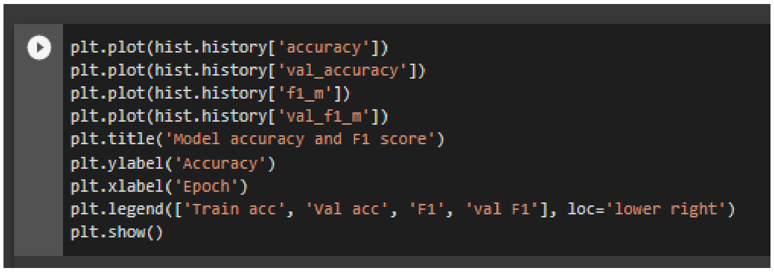
```
[19] hist = model.fit(X_train, Y_train,
                      batch_size=32, epochs=100,
                      validation_data=(X_val, Y_val))
```

9. Visualize the plot for loss and validation loss.



```
import matplotlib.pyplot as plt

plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```

10. Visualize the plot for accuracy and F1 score.



```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.plot(hist.history['f1_m'])
plt.plot(hist.history['val_f1_m'])
plt.title('Model accuracy and F1 score')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train acc', 'Val acc', 'F1', 'val F1'], loc='lower right')
plt.show()
```