Colab: https://colab.research.google.com/drive/1QlmcDSR8_NwZgmHPkevLUhWyz9f6Fjiq?usp=sharing

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

```
!wget "https://drive.google.com/uc?export=download&id=1GVhrh2rH6hUunV4Tf7lQoSFsqov9
```

```
--2022-12-13 15:35:36--  https://drive.google.com/uc?export=download&id=1GVhrh
Resolving drive.google.com (drive.google.com)... 142.250.141.139, 142.250.141.
Connecting to drive.google.com (drive.google.com)|142.250.141.139|:443... conn
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0s-50-docs.googleusercontent.com/docs/securesc/ha0ro937g
Warning: wildcards not supported in HTTP.
--2022-12-13 15:35:37--  https://doc-0s-50-docs.googleusercontent.com/docs/sec
Resolving doc-0s-50-docs.googleusercontent.com (doc-0s-50-docs.googleuserconte
Connecting to doc-0s-50-docs.googleusercontent.com (doc-0s-50-docs.googleuserc
HTTP request sent, awaiting response... 200 OK
Length: 761835 (744K) [text/csv]
Saving to: 'healthyfime.csv'

healthyfime.csv     100%[===================>] 743.98K  --.-KB/s    in 0.009s

2022-12-13 15:35:37 (78.8 MB/s) - 'healthyfime.csv' saved [761835/761835]
```

```
df = pd.read_csv("healthyfime.csv")
df.head()
```

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13393 entries, 0 to 13392
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
```

```
---  ------                    --------------  -----
 0   age                       13393 non-null  float64
 1   gender                    13393 non-null  object
 2   height_cm                 13393 non-null  float64
 3   weight_kg                 13393 non-null  float64
 4   body fat_%                13393 non-null  float64
 5   diastolic                 13393 non-null  float64
 6   systolic                  13393 non-null  float64
 7   gripForce                 13393 non-null  float64
 8   sit and bend forward_cm   13393 non-null  float64
 9   sit-ups counts            13393 non-null  float64
 10  broad jump_cm             13393 non-null  float64
 11  class                     13393 non-null  object
dtypes: float64(10), object(2)
memory usage: 1.2+ MB
```

```
df.replace({"M":0, "F":1} , inplace = True)
df.head()
```

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce
0	27.0	0	172.3	75.24	21.3	80.0	130.0	54.9
1	25.0	0	165.0	55.80	15.7	77.0	126.0	36.4
2	31.0	0	179.6	78.00	20.1	92.0	152.0	44.8
3	32.0	0	174.5	71.10	18.4	76.0	147.0	41.4
4	28.0	0	173.8	67.70	17.1	70.0	127.0	43.5

```
classes = list(df['class'].unique())
mapping_dict = { ch : i for i, ch in enumerate(sorted(classes, reverse=True)) }
print (mapping_dict)
df['class'].replace(mapping_dict , inplace = True)
df.head()
```

```
{'D': 0, 'C': 1, 'B': 2, 'A': 3}
```

	age	gender	height_cm	weight_kg	body fat_%	diastolic	systolic	gripForce
0	27.0	0	172.3	75.24	21.3	80.0	130.0	54.9
1	25.0	0	165.0	55.80	15.7	77.0	126.0	36.4
2	31.0	0	179.6	78.00	20.1	92.0	152.0	44.8
3	32.0	0	174.5	71.10	18.4	76.0	147.0	41.4
4	28.0	0	173.8	67.70	17.1	70.0	127.0	43.5

```
df["class"].unique()
```

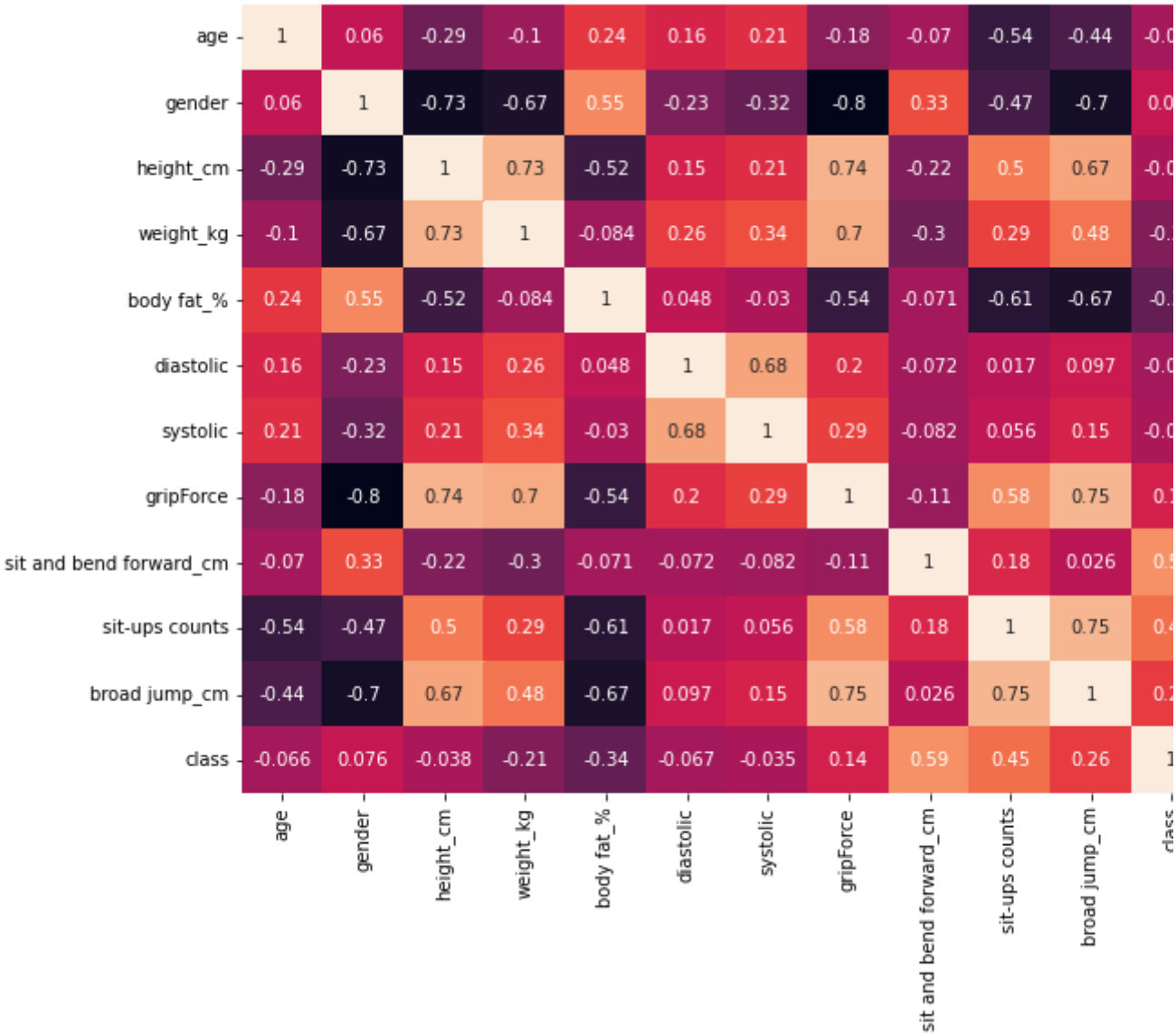
```
array([1, 3, 2, 0])
```

```
df["class"].value_counts()
```

```
1    3349
0    3349
3    3348
2    3347
Name: class, dtype: int64
```

```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcbdf52bd30>
```

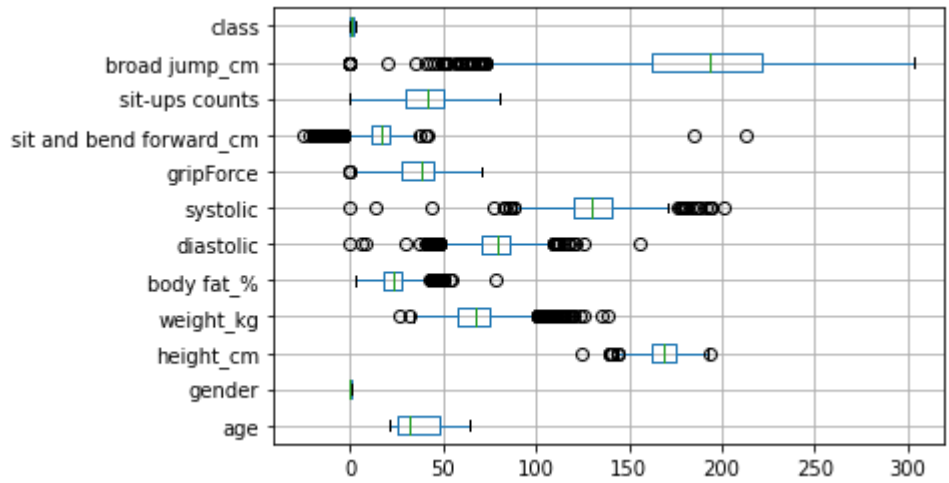



```
df.describe()
```

	age	gender	height_cm	weight_kg	body fat_%	diastol
count	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.0000
mean	36.775106	0.367804	168.559807	67.447316	23.240165	78.7968
std	13.625639	0.482226	8.426583	11.949666	7.256844	10.7420
min	21.000000	0.000000	125.000000	26.300000	3.000000	0.0000
25%	25.000000	0.000000	162.400000	58.200000	18.000000	71.0000
50%	32.000000	0.000000	169.200000	67.400000	22.800000	79.0000
75%	48.000000	1.000000	174.800000	75.300000	28.000000	86.0000
max	64.000000	1.000000	193.800000	138.100000	78.400000	156.2000

```
df.boxplot(rot=0, vert=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcbc8bf67c0>
```




```
X, y = df.iloc[:, :-1], df.iloc[:, -1]
print(X.shape, y.shape)
```

```
(13393, 11) (13393,)
```

```
from sklearn.model_selection import train_test_split
X_dev, X_test, y_dev, y_test = train_test_split(X, y, test_size=0.1, random_state=4
print('Train : ', X_dev.shape, y_dev.shape)
print('Test : ', X_test.shape, y_test.shape)
```

```
Train :  (12053, 11) (12053,)
Test :  (1340, 11) (1340,)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_dev)
```

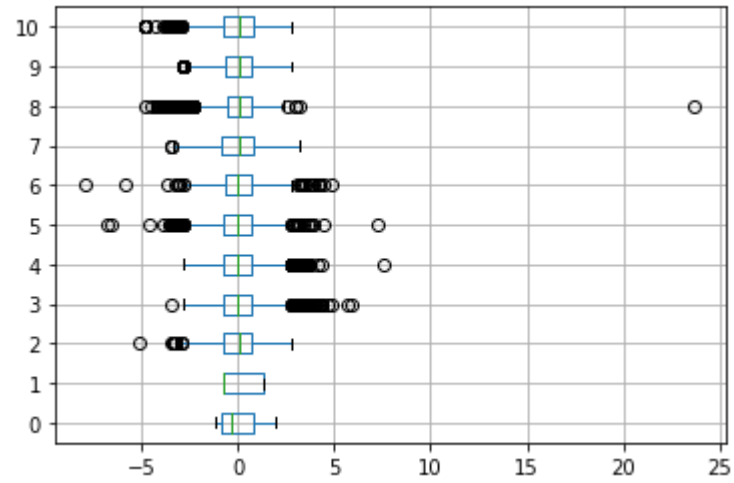
```
StandardScaler()
```

```
X_dev = scaler.transform(X_dev)
X_test = scaler.transform(X_test)
```

```
X_dev = pd.DataFrame(X_dev)
X_test = pd.DataFrame(X_test)
```

```
X_dev.boxplot(rot=0, vert=False)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcbc8bb8b50>
```




```
import tensorflow as tf
```

```
tf.__version__
```

```
'2.9.2'
```

```
dir(tf.keras)
```

```
['Input',
 'Model',
 'Sequential',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__internal__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '__version__',
 '_sys',
 'activations',
 'applications',
 'backend',
 'callbacks',
 'constraints',
 'datasets',
 'dtensor',
```

```
 'estimator',
 'experimental',
 'initializers',
 'layers',
 'losses',
 'metrics',
 'mixed_precision',
 'models',
 'optimizers',
 'preprocessing',
 'regularizers',
 'utils',
 'wrappers']
```

```
dir(tf.keras.activations)
```

```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_sys',
 'deserialize',
 'elu',
 'exponential',
 'gelu',
 'get',
 'hard_sigmoid',
 'linear',
 'relu',
 'selu',
 'serialize',
 'sigmoid',
 'softmax',
 'softplus',
 'softsign',
 'swish',
 'tanh']
```

```
dir(tf.keras.optimizers)
```

```
['Adadelta',
 'Adagrad',
 'Adam',
 'Adamax',
 'Ftrl',
 'Nadam',
 'Optimizer',
 'RMSprop',
 'SGD',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
```

```
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_sys',
 'deserialize',
 'experimental',
 'get',
 'legacy',
 'schedules',
 'serialize']
```

```
# Sequential API
# Functional API --> Complex non-sequential networks - CNNs, ResNet, post-read
```

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

```
model = Sequential([
          Dense(100, activation="relu", input_shape=(11,)),
          Dense(4, activation="softmax")
        ])
```

```
type(model.weights)
```

```
list
```

```
for param in model.weights:
  print(param.shape)
```

```
(11, 100)
(100,)
(100, 4)
(4,)
```

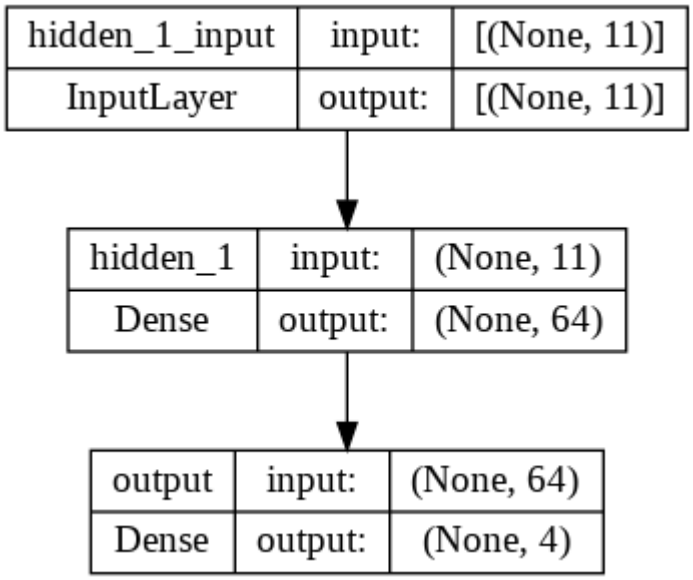
```
model = Sequential()
model.add(Dense(64, activation="relu", input_shape=(11,), name="hidden_1"))
model.add(Dense(4, activation="softmax", name="output"))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden_1 (Dense)            (None, 64)                768

 output (Dense)              (None, 4)                 260

=================================================================
Total params: 1,028
```

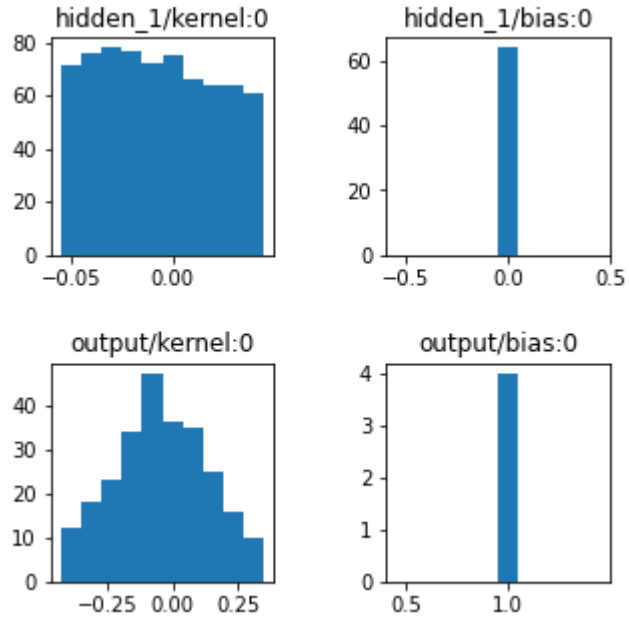
```
Trainable params: 1,028
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.utils import plot_model

plot_model(model,
    to_file='model.png',
    show_shapes=True, show_layer_names=True)
```




```
# Weigjts and bias
# Weghts - randomly,multiple ways --> Glorot Normal, Glorot Uniform, HE Normal, HE
# bias - zeros


model = Sequential()
model.add(Dense(64,
                activation="relu",
                input_shape=(11,),
                name="hidden_1",
                kernel_initializer = "random_uniform",
                bias_initializer = "zeros"))
model.add(Dense(4, activation="softmax", name="output",
                kernel_initializer = "he_normal",
                bias_initializer = "ones"))


# Plot histograms of weight and bias values
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 2, figsize=(5,5))
fig.subplots_adjust(hspace=0.5, wspace=0.5)

# get the weights from the layers
weight_layers = [layer for layer in model.layers]

for i, layer in enumerate(weight_layers):
    for j in [0, 1]:
```

```
        axes[i, j].hist(layer.weights[j].numpy().flatten(), align='left')
        axes[i, j].set_title(layer.weights[j].name)
```




```
# loss function - cce, sparse cce, mse, mae
# optimiser - adam, rmsprop, sgd, .....
# metrics - loss, accuracy
# losss function and optimiser with the model ---> model compilation

model_2C = Sequential([
          Dense(64, activation="relu", input_shape=(11,)),
          Dense(1, activation="sigmoid")])

model_2C.compile(optimizer = "sgd",
                 loss = "binary_crossentropy",
                 metrics = ["accuracy"])


model_2C.compile(optimizer = tf.keras.optimizers.SGD(learning_rate=0.001),
                 loss = tf.keras.losses.BinaryCrossentropy(),
                 metrics = ["accuracy"])


model.compile(optimizer = "adam",
                 loss = "sparse_categorical_crossentropy",
                 metrics = ["accuracy"])


history = model.fit(X_dev, y_dev, epochs=10, batch_size=256, validation_split=0.1,


# history (callbacks)
history = model.fit(X_dev, y_dev, epochs=500, batch_size=256, validation_split=0.1,
```

```
    Epoch 1/500
    43/43 [==============================] - 0s 7ms/step - loss: 0.8272 - accuracy
    Epoch 2/500
    43/43 [==============================] - 0s 5ms/step - loss: 0.8219 - accuracy
    Epoch 3/500
    43/43 [==============================] - 0s 5ms/step - loss: 0.8170 - accuracy
```

```
Epoch 4/500
43/43 [==============================] - 0s 5ms/step - loss: 0.8111 - accuracy
Epoch 5/500
43/43 [==============================] - 0s 7ms/step - loss: 0.8052 - accuracy
Epoch 6/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7994 - accuracy
Epoch 7/500
43/43 [==============================] - 0s 6ms/step - loss: 0.7929 - accuracy
Epoch 8/500
43/43 [==============================] - 0s 6ms/step - loss: 0.7865 - accuracy
Epoch 9/500
43/43 [==============================] - 0s 8ms/step - loss: 0.7795 - accuracy
Epoch 10/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7725 - accuracy
Epoch 11/500
43/43 [==============================] - 0s 4ms/step - loss: 0.7653 - accuracy
Epoch 12/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7584 - accuracy
Epoch 13/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7518 - accuracy
Epoch 14/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7447 - accuracy
Epoch 15/500
43/43 [==============================] - 0s 6ms/step - loss: 0.7394 - accuracy
Epoch 16/500
43/43 [==============================] - 0s 4ms/step - loss: 0.7333 - accuracy
Epoch 17/500
43/43 [==============================] - 0s 7ms/step - loss: 0.7272 - accuracy
Epoch 18/500
43/43 [==============================] - 0s 4ms/step - loss: 0.7225 - accuracy
Epoch 19/500
43/43 [==============================] - 0s 6ms/step - loss: 0.7178 - accuracy
Epoch 20/500
43/43 [==============================] - 0s 6ms/step - loss: 0.7141 - accuracy
Epoch 21/500
43/43 [==============================] - 0s 7ms/step - loss: 0.7089 - accuracy
Epoch 22/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7052 - accuracy
Epoch 23/500
43/43 [==============================] - 0s 5ms/step - loss: 0.7015 - accuracy
Epoch 24/500
43/43 [==============================] - 0s 5ms/step - loss: 0.6978 - accuracy
Epoch 25/500
43/43 [==============================] - 0s 8ms/step - loss: 0.6949 - accuracy
Epoch 26/500
43/43 [==============================] - 0s 6ms/step - loss: 0.6924 - accuracy
Epoch 27/500
43/43 [==============================] - 0s 5ms/step - loss: 0.6895 - accuracy
Epoch 28/500
43/43 [==============================] - 0s 5ms/step - loss: 0.6857 - accuracy
Epoch 29/500
```

```
history.__dict__.keys()
```

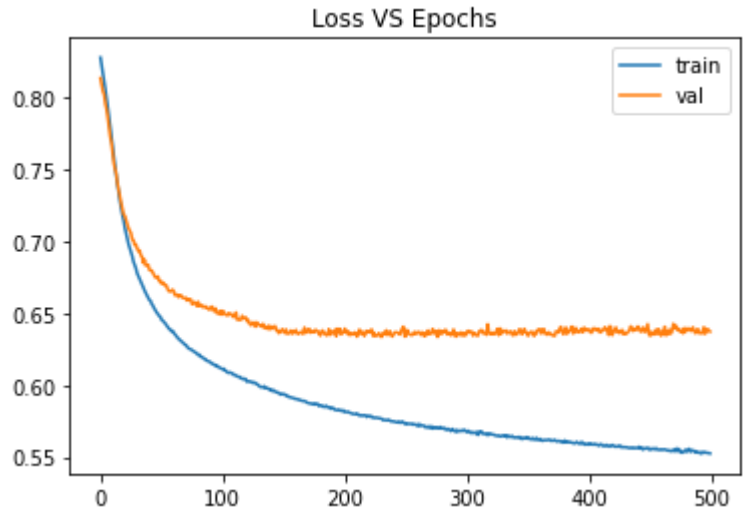
```
dict_keys(['validation_data', 'model', '_chief_worker_only',
'_supports_tf_logs', 'history', 'params', 'epoch'])
```

```
history.history.keys()
```

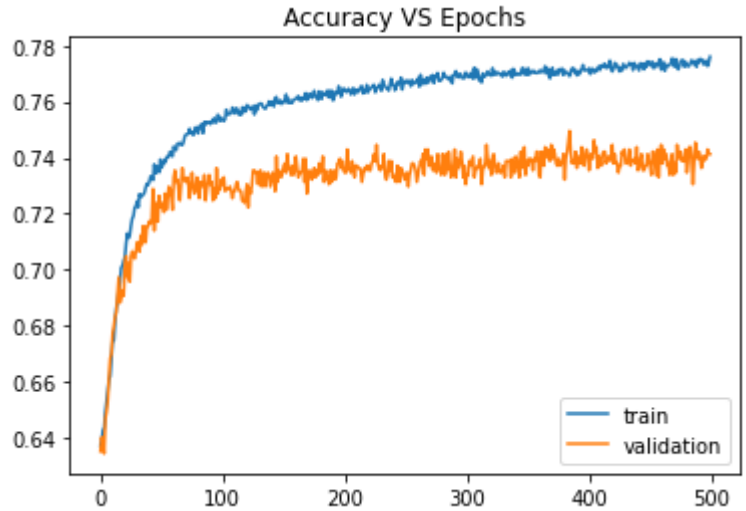
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
epochs = history.epoch
loss = history.history["loss"]
accuracy = history.history["accuracy"]
val_loss = history.history["val_loss"]
val_accuracy = history.history["val_accuracy"]
```

```
plt.figure()
plt.plot(epochs, loss, label="train")
plt.plot(epochs, val_loss, label="val")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()
```




```
plt.figure()
plt.plot(epochs, accuracy, label="train")
plt.plot(epochs, val_accuracy, label="validation")
plt.legend()
plt.title("Accuracy VS Epochs")
plt.show()
```



```
model.evaluate(X_test, y_test)
```

```
42/42 [==============================] - 0s 1ms/step - loss: 0.5844 - accuracy
[0.5843523740768433, 0.7597014904022217]
```

```
model.evaluate(X_dev, y_dev)
```

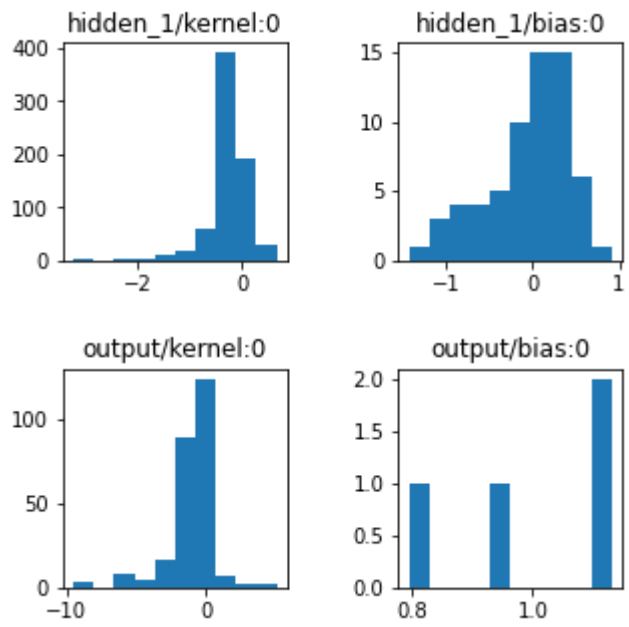
```
377/377 [==============================] - 1s 1ms/step - loss: 0.5579 - accura
[0.557871401309967, 0.7729195952415466]
```

```
np.argmax(model.predict(np.expand_dims(X_test.to_numpy()[0], axis=0))[0])
```

```
1/1 [==============================] - 0s 74ms/step
0
```

```
# HOMEWORK - Create the same 2-layer NN with 100N in L-1 for spiral dataset, Keras
# spiral dataset - https://drive.google.com/uc?id=1dLOPwh01o3k8p_hK633ixhD1ehz6nNWk
```

```
# Plot histograms of weight and bias values
import matplotlib.pyplot as plt
fig, axes = plt.subplots(2, 2, figsize=(5,5))
fig.subplots_adjust(hspace=0.5, wspace=0.5)

# get the weights from the layers
weight_layers = [layer for layer in model.layers]

for i, layer in enumerate(weight_layers):
    for j in [0, 1]:
        axes[i, j].hist(layer.weights[j].numpy().flatten(), align='left')
        axes[i, j].set_title(layer.weights[j].name)
```




```
# callback - a set of functions in a Callback class, which are executred at diff st
```

1. on_epoch_begin
2. on_epoch_end

```
class VerboseCallback(tf.keras.callbacks.Callback):

  def on_train_begin(self, logs=None):
    print("Starting the training...")

  def on_epoch_end(self, epoch, logs=None):
    if epoch % 50 == 0:
      print(f"Epoch: {str(epoch).zfill(3)}, Loss: {logs['loss']}")

  def on_train_end(self, logs=None):
    print("Finished the training..")


def create_model():
    model = Sequential([
                    Dense(32, activation="relu", input_shape=(11,), name="hidden_1"
                    Dense(16, activation="relu", name="hidden_2"),
                    Dense(4, activation="softmax", name="output")])
    model.compile(
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001),
    loss = tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"])
    return model


model = create_model()


history = model.fit(X_dev, y_dev, epochs=500,
          batch_size=256, validation_split=0.1,
          verbose = 0, callbacks = [VerboseCallback()])
```

```
Starting the training...
Epoch: 000, Loss: 1.3090120553970337
Epoch: 050, Loss: 0.6377137899398804
Epoch: 100, Loss: 0.5972737073898315
Epoch: 150, Loss: 0.580062210559845
Epoch: 200, Loss: 0.567562997341156
Epoch: 250, Loss: 0.5592490434646606
Epoch: 300, Loss: 0.5515960454940796
Epoch: 350, Loss: 0.5466218590736389
Epoch: 400, Loss: 0.5409554243087769
Epoch: 450, Loss: 0.5375148057937622
Finished the training..
```

```
# CSVLogger, EarlyStopping, LearningRateScheduler, ModelCheckpoint
```

pip install tensorboard

conda install -c conda-forge tensorboard

```
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

```
log_folder = "logs"
```

```
!ls
```

```
healthyfime.csv  model.png  sample_data
```

```
# tensorboard callback -
# 1.log_dir
# 2. update_freq - batch, int N, epoch
# 3. histogram (wrights and biases) - 1/0
```

```
from tensorflow.keras.callbacks import TensorBoard
```

```
tb_callback = TensorBoard(log_dir=log_folder, histogram_freq=1)
```

```
!ls
```

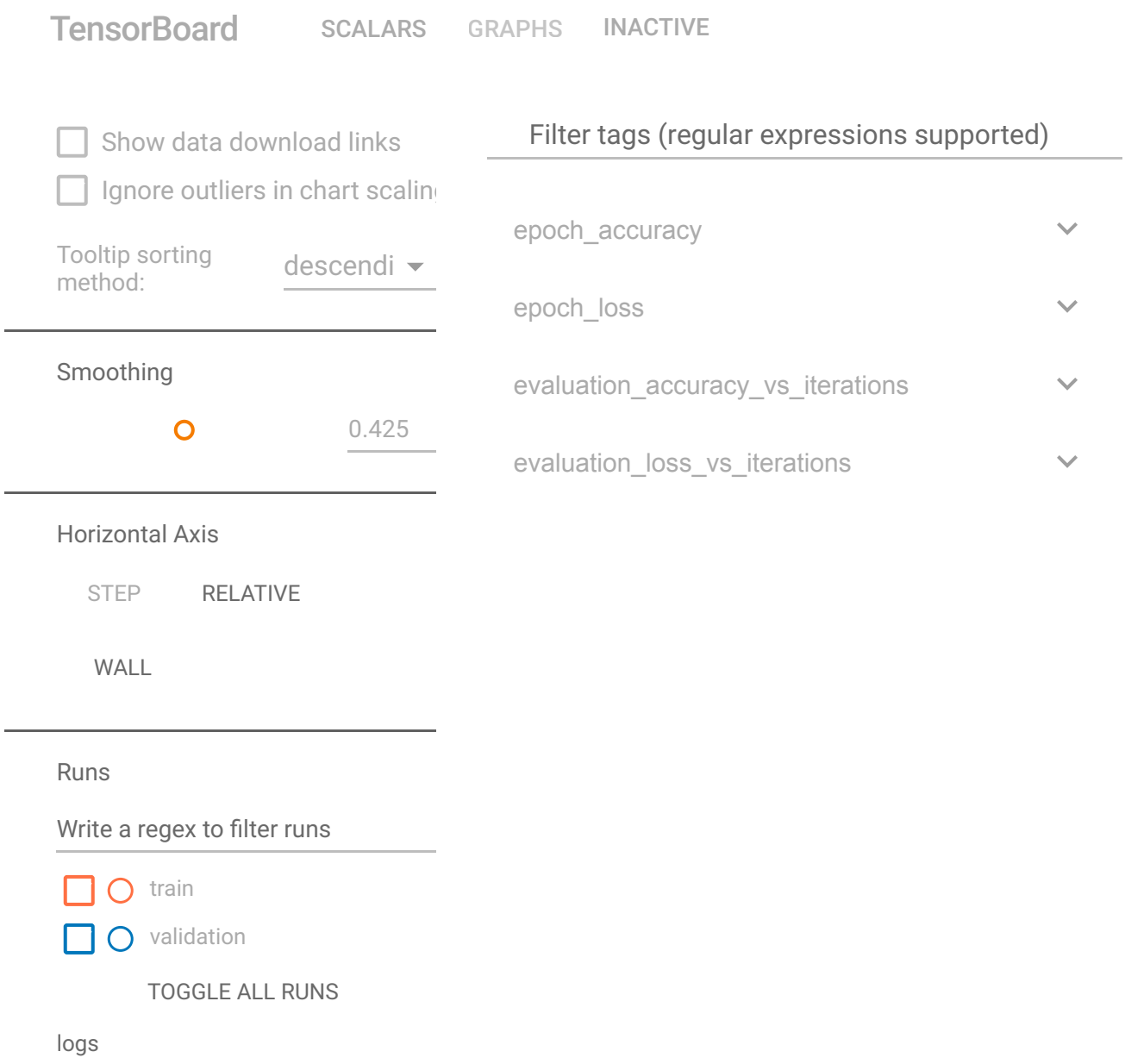
```
healthyfime.csv  model.png  sample_data
```

```
model = create_model()
history = model.fit(X_dev, y_dev, epochs=500, batch_size=256,
                    validation_split = 0.1, verbose=0,
                    callbacks=[tb_callback, VerboseCallback()])
```

```
Starting the training...
Epoch: 000, Loss: 1.3522670269012451
Epoch: 050, Loss: 0.6423649191856384
Epoch: 100, Loss: 0.5984207391738892
Epoch: 150, Loss: 0.5787996649742126
Epoch: 200, Loss: 0.5672569274902344
Epoch: 250, Loss: 0.5576054453849792
Epoch: 300, Loss: 0.5506313443183899
Epoch: 350, Loss: 0.5431100726127625
Epoch: 400, Loss: 0.5386828184127808
Epoch: 450, Loss: 0.5346130728721619
Finished the training..
```

```
%tensorboard --logdir={log_folder}
```

```
Reusing TensorBoard on port 6006 (pid 11135), started 0:00:18 ago. (Use
'!kill 11135' to kill it.)
```



Colab paid products - Cancel contracts here

✓ 0s completed at 22:27