Linear Regression with Two Input Variables using the Iris Flower Dataset

- We would like to predict "petal width" (column 4 in the original dataset) using sepal width (column 2) and petal length (column 3)
- Dataset: rawdata and metadata

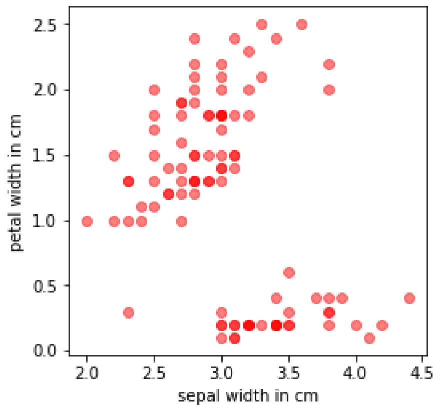
```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt

# Column 2. sepal width in cm (load as col 0)
# Column 3. petal length in cm (load as col 1)
# Column 4. petal width in cm (load as col 2)
datapath = 'https://raw.githubusercontent.com/badriadhikari/2019-Fall-AI/master/MODULE-I/iris.data'
dataset = np.genfromtxt(datapath, delimiter=",", usecols=(1, 2, 3))

print('')
print(dataset.shape)
print('')
print(dataset[0:5])
```

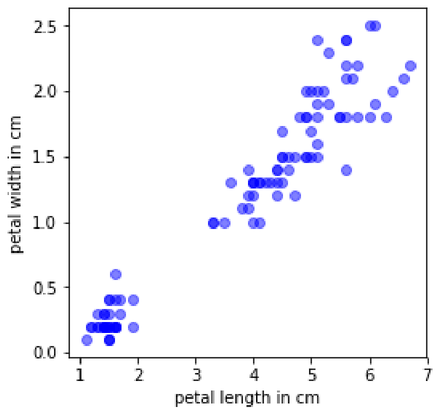
```
Using TensorFlow backend.

(150, 3)

[[3.5 1.4 0.2]
 [3.  1.4 0.2]
 [3.2 1.3 0.2]
 [3.1 1.5 0.2]
 [3.6 1.4 0.2]]
```

```
# Q1. Why is shuffling important before splitting?
np.random.shuffle(dataset)
print('')
print(dataset[0:5])
train = dataset[:100]
valid = dataset[100:]
print('')
print(train.shape)
print('')
print(valid.shape)
```

```

[[2.6 5.6 1.4]
 [3.6 6.1 2.5]
 [3.2 5.1 2. ]
 [3.2 1.6 0.2]
 [2.9 4.5 1.5]]

(100, 3)

(50, 3)
```

```
#Q2. Which of the two input features are more useful
#    for predicting petal width?
plt.figure(figsize=(4,4))
plt.scatter(train[:, 0], train[:, 2], color = 'r', alpha = 0.5)
plt.xlabel('sepal width in cm')
plt.ylabel('petal width in cm')
plt.show()
plt.figure(figsize=(4,4))
plt.scatter(train[:, 1], train[:, 2], color = 'b', alpha = 0.5)
plt.xlabel('petal length in cm')
plt.ylabel('petal width in cm')
plt.show()
```






```
train_input = train[:, 0:2] # col 2 & 3
train_output = train[:, 2] # col 4
valid_input = valid[:, 0:2]
valid_output = valid[:, 2]

print('')
print(train_input[0:5])
print('')
print(train_output[0:5])
```

```
[[2.6 5.6]
 [3.6 6.1]
 [3.2 5.1]
 [3.2 1.6]
 [2.9 4.5]]

[1.4 2.5 2.  0.2 1.5]
```

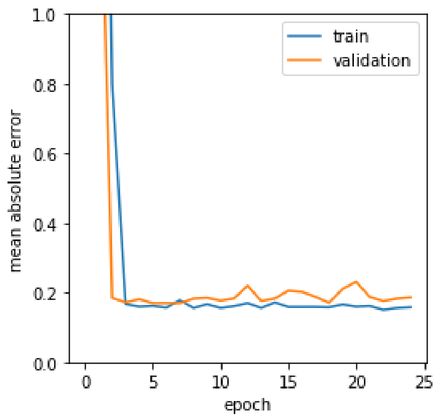
```
#Q3. Why is the number of parameters (Param #) = 3?
model = Sequential()
model.add(Dense(1, input_dim = len(train_input[0]), activation='linear'))
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
None
```

```
# Changing 'mae' to 'mse' should improve the smoothness of
#  the learning curve and possibly the overall errors
model.compile(loss='mae', optimizer='sgd', metrics=['mae'])

```

```
# Verbose = 0 shows no updates, can be changed to 1 or 2
history = model.fit(train_input, train_output, epochs=25,
                    verbose = 0, batch_size=10,
                    validation_data = (valid_input, valid_output))
```

```
#Q4. Why eventually validation MAE is not
#    always less than train MAE?
plt.figure(figsize=(4,4))
plt.plot(history.history['mean_absolute_error'])
plt.plot(history.history['val_mean_absolute_error'])
plt.ylabel('mean absolute error')
plt.ylim(0, 1)
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```




```
#Q5. Are these predictions reasonable?
np.set_printoptions(precision = 2)
print ('True Validation Data:')
print(valid_output[0:5])
prediction = model.predict(valid_input)
print ('Prediction:')
print(prediction[0:5].T)
```

```
True Validation Data:
[2.5 1.6 0.2 0.3 1.6]
Prediction:
[[1.87 1.92 0.04 0.15 1.4 ]]
```

```
#Q6. What weight corresponds to which input feature?
#    Which input feature is important? Why?
print('Model weights (w0, w1, and bias):')
w0 = model.layers[0].get_weights()[0][0]
w1 = model.layers[0].get_weights()[0][1]
b0 = model.layers[0].get_weights()[1]
print(w0)
print(w1)
print(b0)
```

```
Model weights (w0, w1, and bias):
[-0.01]
[0.39]
[-0.31]
```

```
#Q7. Why do we use model.predict(), if we can compute
#    the predictions from weights that the model learns?
print('Validation Data 0:')
print(valid_input[0], valid_output[0])
print('Prediction:')
print(valid_input[0, 0] * (w0) + valid_input[0, 1] * (w1) + (b0))
print('Validation Data 1:')
print(valid_input[1], valid_output[1])
print('Prediction:')
print(valid_input[1, 0] * (w0) + valid_input[1, 1] * (w1) + (b0))
```

```
Validation Data 0:
[3.3 5.7] 2.5
Prediction:
[1.87]
Validation Data 1:
[3.  5.8] 1.6
Prediction:
[1.92]
```

```
model = Sequential()
model.add(Dense(1, input_dim = len(train_input[0]), activation='linear'))
```

```
model = Sequential()
model.add(Dense(8, input_dim = len(train_input[0]), activation='sigmoid'))
model.add(Dense(4, input_dim = len(train_input[0]), activation='sigmoid'))
model.add(Dense(2, input_dim = len(train_input[0]), activation='sigmoid'))
model.add(Dense(1, input_dim = len(train_input[0]), activation='linear'))
```

```
def mymodel(X):
    l1_n1 = X[0] * 0.6 + X[1] * 0.78 + ... + 0.98
    l1_n2 = X[0] * 0.4 + X[1] * 0.54 + ... + 0.80
    ...
    l1_n8 = X[0] * 0.2 + X[1] * 0.81 + ... + 0.38
    l2_n1 = ...
    ...
    l3_n1 = ...
    ...
    l4 =
    return l4
```