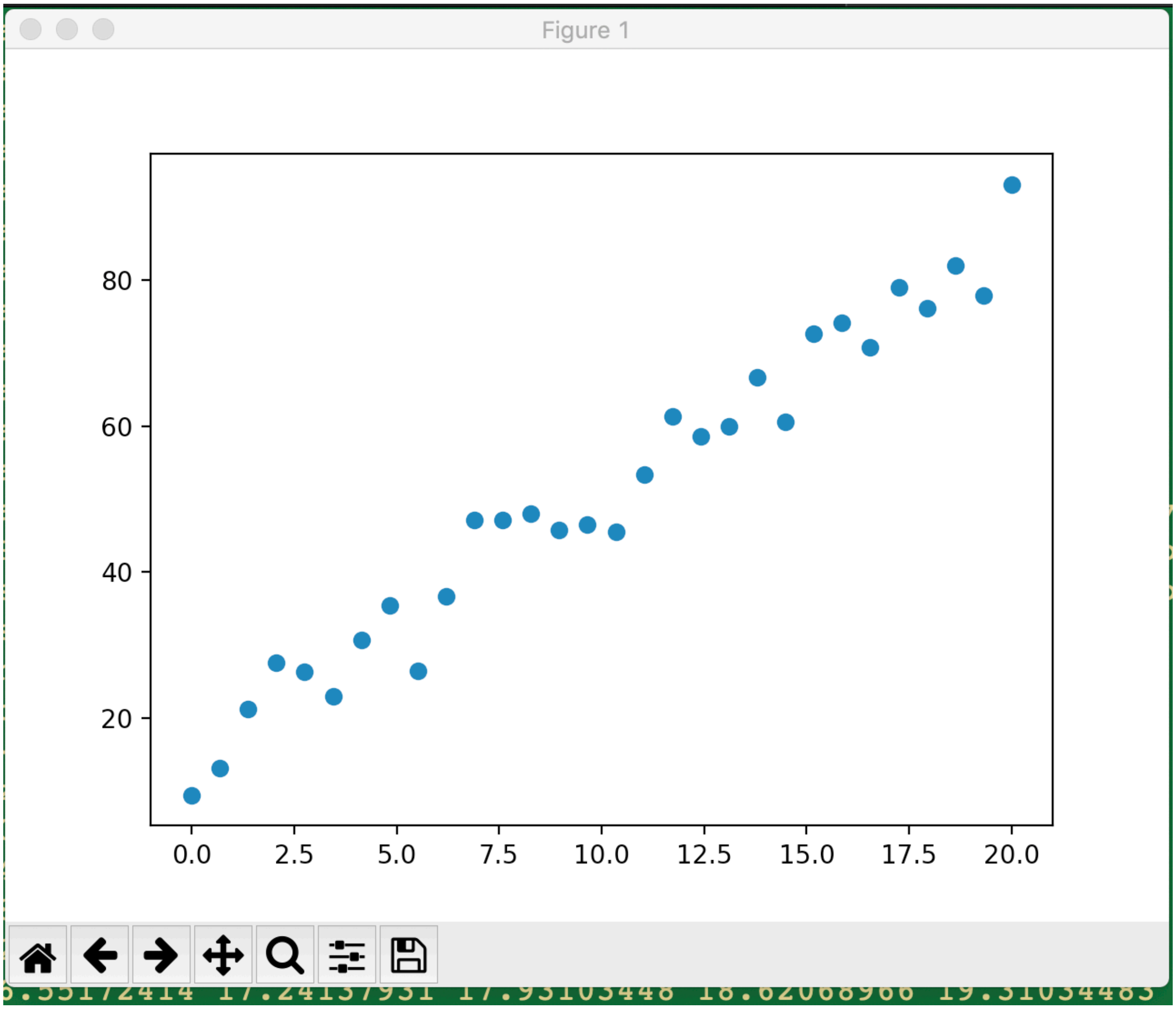
Week #3 Assignments – Satish Ramachandran

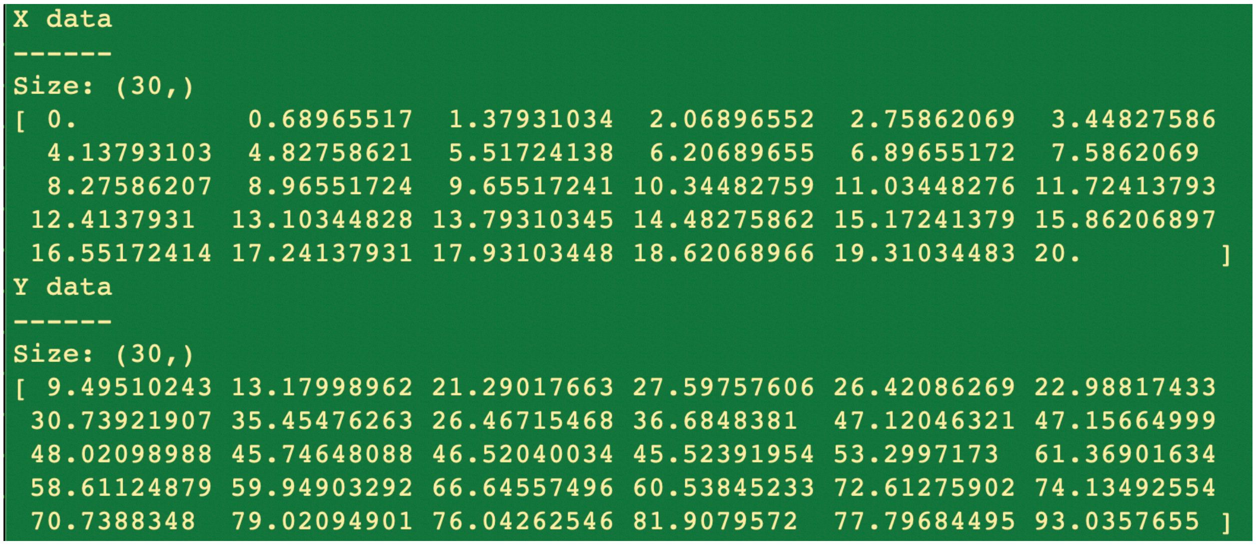
Problem #1

```
'''
Week 3 - Problem 1
Solution using Scikit-learn and Tensorflow

'''
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.compat.v1 as tf
from sklearn import linear_model

# Disable 2.0 behavior
tf.disable_v2_behavior()

### Generate the data ###
def generate_data(random_seed, n_samples):
    tf.set_random_seed(random_seed)
    train_x = np.linspace(0,20,n_samples)
    train_y = 3.7 * train_x + 14 + 4 * np.random.randn(n_samples)
    print("X data")
    print("------")
    print("Size: " + str(np.shape(train_x)))
    print(train_x)
    print("Y data")
    print("------")
    print("Size: " + str(np.shape(train_y)))
    print(train_y)
    plt.plot(train_x, train_y,'o')
    plt.waitforbuttonpress()
    plt.close()
    return(train_x, train_y)

### SciKit Learn method
def scikit_method(x_data, y_data):
    print("Using SciKit learn..")
    linear_reg = linear_model.LinearRegression()
    print("Dimensions: X: " + str(x_data.ndim) + ", Y: " + str(y_data.ndim))
    # IMPORTANT: LinearRegression expects a 2-D array. So, add a dimension using
    # reshape()
    linear_reg.fit(x_data.reshape(-1,1), y_data.reshape(-1,1))
    print("Slope : " + str(linear_reg.coef_))
    print("Intercept: " + str(linear_reg.intercept_))
    return (linear_reg.coef_, linear_reg.intercept_)

### TensorFlow method
def tensorflow_method(x_data, y_data, learn_rate, epochs):
    print("Tensor flow method..")
    graph = tf.Graph()
    with graph.as_default():
        slope = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
        intercept = tf.Variable(tf.zeros([1]))
        response = slope*x_data + intercept

        cost = tf.reduce_mean(tf.square(response - y_data))
        optimizer = tf.train.GradientDescentOptimizer(learn_rate).minimize(cost)

        with tf.Session(graph=graph) as session:
            init = tf.global_variables_initializer()
            session.run(init)

            for epoch in range(epochs):
                session.run(optimizer)
                if ( epoch % 1000 ) == 0:
                    print("Plot after " + str(epoch) + " iterations")
                    plt.plot(x_data, y_data, 'o', label = 'step = {}'.format(epoch))
                    plt.plot(x_data, session.run(slope)*x_data + session.run(intercept))
                    plt.legend()
                    plt.show()
            plt.waitforbuttonpress()
            print("Slope = ",session.run(slope))
            print("Intercept = ",session.run(intercept))

            #return(tf.cast(session.run(slope), tf.int32), tf.cast(session.run(intercept), tf.int32))
            return(session.run(slope), session.run(intercept))

train_x, train_y = generate_data(42, 30)
s_slope, s_intercept = scikit_method(train_x, train_y)
t_slope, t_intercept = tensorflow_method(train_x, train_y, 0.001, 10000)

print("RESULTS" + "\n" + "-------")
print("SciKit Learn : slope: " + str(s_slope) + " intecept: " + str(s_intercept))
print("TensorFlow : slope: " + str(t_slope) + " intecept: " + str(t_intercept))
```

This is the plot of the generated data set

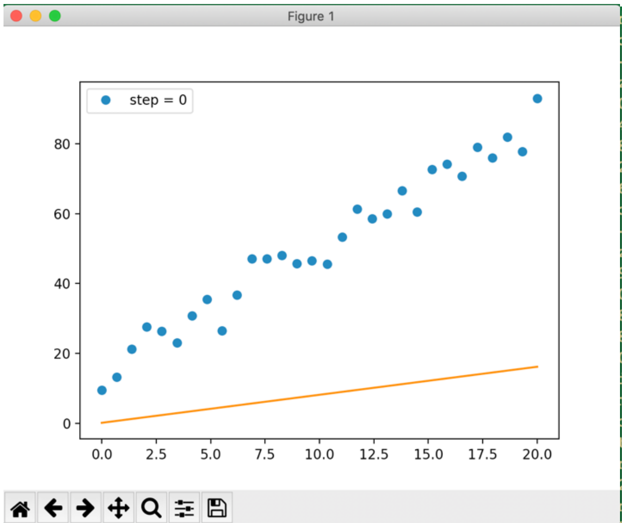


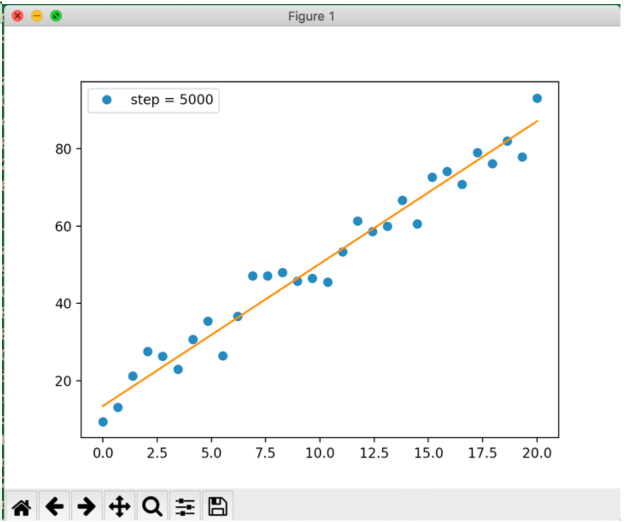
```
.55172414  17.24137931  17.93103448  18.62068966  19.31034483
```

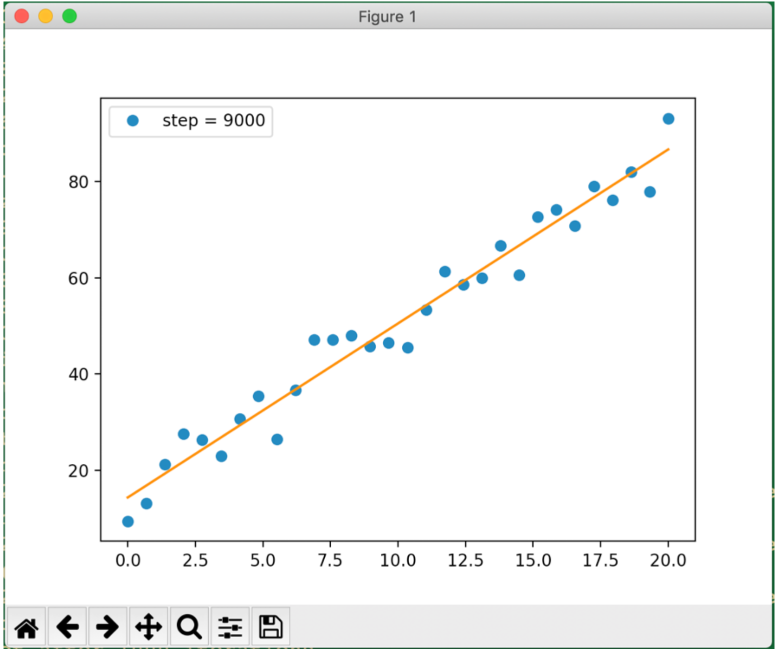


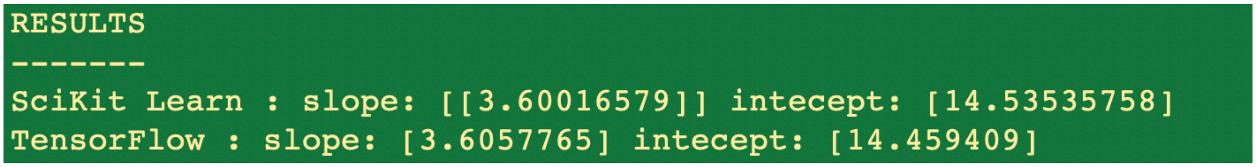
```
X data
------
Size: (30,)
[ 0.          0.68965517  1.37931034  2.06896552  2.75862069  3.44827586
  4.13793103  4.82758621  5.51724138  6.20689655  6.89655172  7.5862069
  8.27586207  8.96551724  9.65517241 10.34482759 11.03448276 11.72413793
 12.4137931  13.10344828 13.79310345 14.48275862 15.17241379 15.86206897
 16.55172414 17.24137931 17.93103448 18.62068966 19.31034483 20.        ]
Y data
------
Size: (30,)
[ 9.49510243 13.17998962 21.29017663 27.59757606 26.42086269 22.98817433
 30.73921907 35.45476263 26.46715468 36.6848381  47.12046321 47.15664999
 48.02098988 45.74648088 46.52040034 45.52391954 53.2997173  61.36901634
 58.61124879 59.94903292 66.64557496 60.53845233 72.61275902 74.13492554
 70.7388348  79.02094901 76.04262546 81.9079572  77.79684495 93.0357655 ]
```

Tensor flow plots to start with, and after a few thousand epochs:







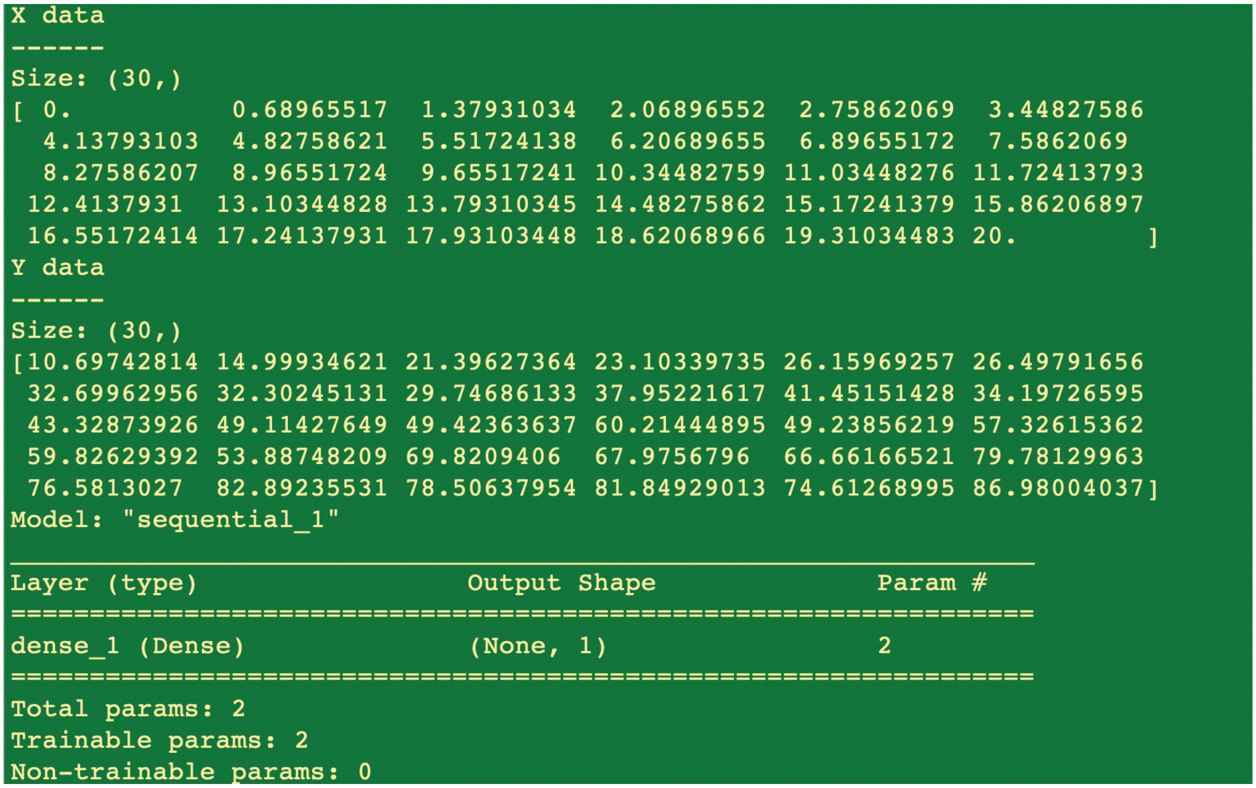


```
RESULTS
-------
SciKit Learn : slope: [[3.60016579]] intecept: [14.53535758]
TensorFlow : slope: [3.6057765] intecept: [14.459409]
```

Problem 2

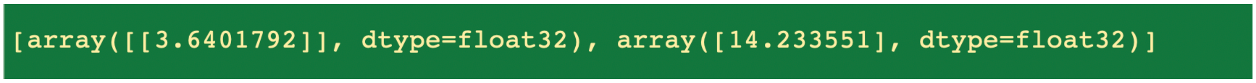
```
'''
Week 3 - Problem 2
Solution using Keras

'''
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.compat.v1 as tf
import keras
from keras.models import Sequential
from keras.layers import Dense

# Disable 2.0 behavior
tf.disable_v2_behavior()

### Generate the data ###
def generate_data(random_seed, n_samples):
    tf.set_random_seed(random_seed)
    train_x = np.linspace(0,20,n_samples)
    train_y = 3.7 * train_x + 14 + 4 * np.random.randn(n_samples)
    print("X data")
    print("------")
    print("Size: " + str(np.shape(train_x)))
    print(train_x)
    print("Y data")
    print("------")
    print("Size: " + str(np.shape(train_y)))
    print(train_y)
    plt.plot(train_x, train_y,'o')
    plt.waitforbuttonpress()
    plt.close()
    return(train_x, train_y)

def model_keras(x_data, y_data, epochs):
    model = Sequential()
    model.add(Dense(1, input_dim=1, kernel_initializer='normal', activation='linear'))

    #Compile the model
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mse'])

    #Dump the model
    model.summary()
    #Suppressing the per-epoch messages
    hist = model.fit(x_data, y_data, epochs=epochs, verbose=0)

    weightBias = model.layers[0].get_weights()
    #print('Weight and Bias with Keras: " +  weightBias)
    print(weightBias)
    plt.plot(train_x, train_y,'o')
    plt.plot(x_data, weightBias[0][0]*x_data + weightBias[1])
    plt.waitforbuttonpress()

train_x, train_y = generate_data(42, 30)
model_keras(train_x, train_y, 20000)
```

Generated test data:



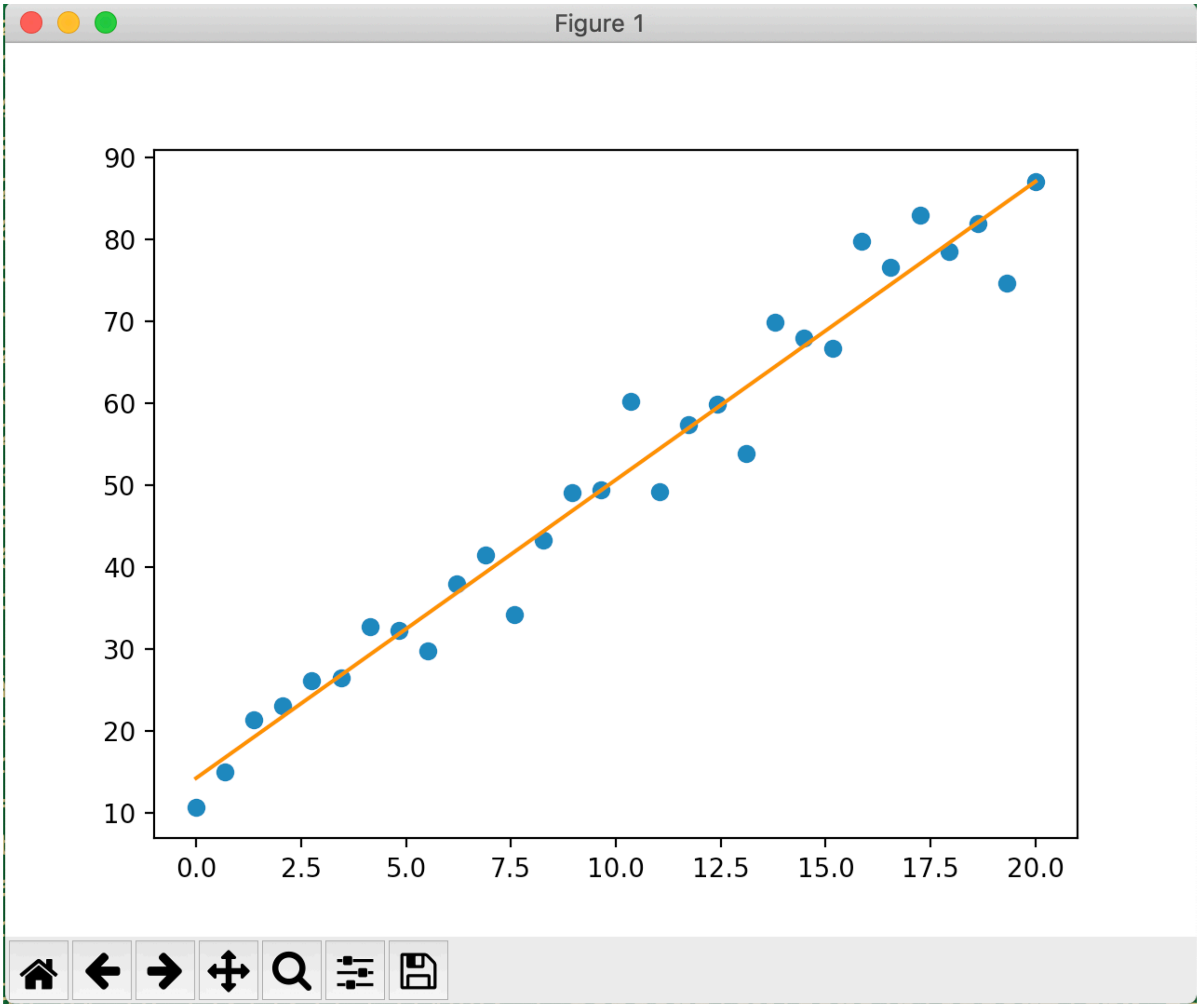
```
X data
------
Size: (30,)
[ 0.          0.68965517  1.37931034  2.06896552  2.75862069  3.44827586
  4.13793103  4.82758621  5.51724138  6.20689655  6.89655172  7.5862069
  8.27586207  8.96551724  9.65517241 10.34482759 11.03448276 11.72413793
 12.4137931  13.10344828 13.79310345 14.48275862 15.17241379 15.86206897
 16.55172414 17.24137931 17.93103448 18.62068966 19.31034483 20.        ]
Y data
------
Size: (30,)
[10.69742814 14.99934621 21.39627364 23.10339735 26.15969257 26.49791656
 32.69962956 32.30245131 29.74686133 37.95221617 41.45151428 34.19726595
 43.32873926 49.11427649 49.42363637 60.21444895 49.23856219 57.32615362
 59.82629392 53.88748209 69.8209406  67.9756796  66.66166521 79.78129963
 76.5813027  82.89235531 78.50637954 81.84929013 74.61268995 86.98004037]
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
```

Model output for slope and intercept with Keras:



```
[array([[3.6401792]], dtype=float32), array([14.233551], dtype=float32)]
```

This is the plot of the samples generated and the final line obtained after keras training

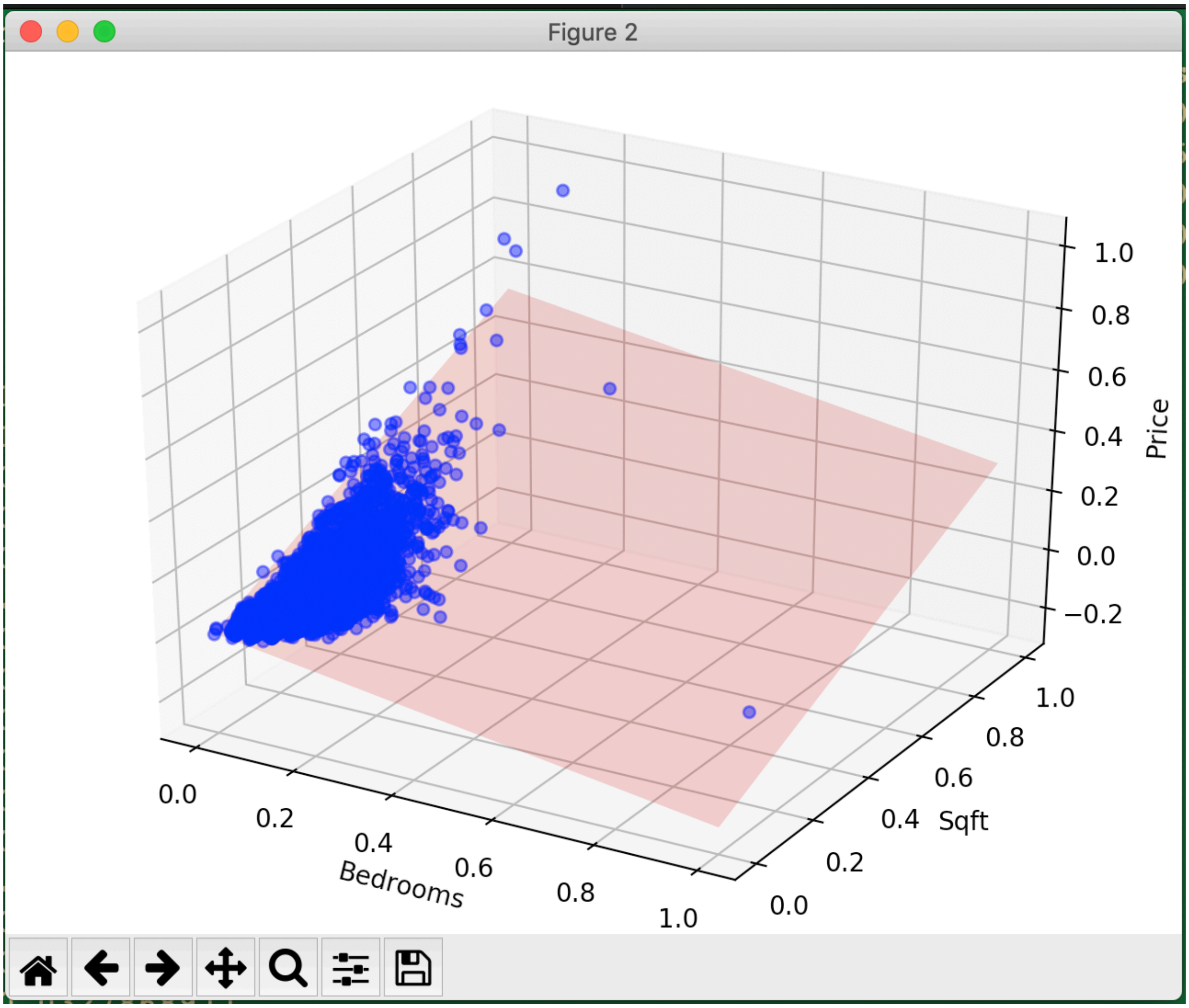


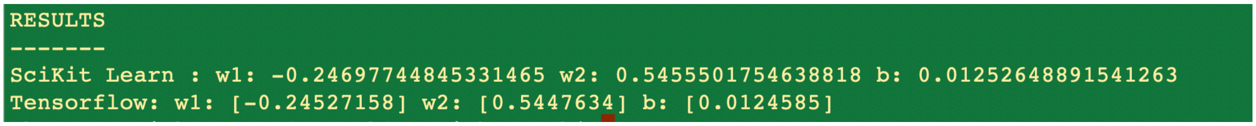
It can be seen that the values are consistent across SciKit learn, TensorFlow and Keras

Problem 3

```
'''
Week 3 - Problem 3
Solution using Scikit-learn and Tensorflow
Housing data in a CSV file
'''
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tensorflow as tf
from sklearn import linear_model
from sklearn import preprocessing
import pandas as pd

def plot_plane(predictors, target, b1, b2, intercept):
    plt.clf()
    figure = plt.figure()
    x1_surf, x2_surf = np.meshgrid(np.linspace(predictors[:,0].min(), predictors[:,0].max(), 500),
np.linspace(predictors[:,1].min(), predictors[:,1].max(), 500))
    y_surf = x1_surf*b1 + x2_surf*b2 + intercept
    ax = Axes3D(figure)
    ax.scatter(predictors[:,0], predictors[:,1], target, c='blue', alpha=0.5)
    plt_surface = ax.plot_surface(x1_surf, x2_surf, y_surf, color='red', alpha=0.2)
    ax.set_xlabel('Bedrooms')
    ax.set_ylabel('Sqft')
    ax.set_zlabel('Price')
    plt.show()
    plt.waitforbuttonpress()

### Import the data from the CSV file ###
def extract_predictor_target():
    data = pd.read_csv('kc_house_data.csv')
    print(data.head())
    predictors = preprocessing.minmax_scale(data[['bedrooms','sqft_living']])
    target = preprocessing.minmax_scale(data[['price']])
    #plot_data(predictors, target)
    print(predictors.shape)
    print(target.shape)
    return (predictors, target)

### SciKit Learn method
def scikit_method(predictors, target):
    print("Using SciKit learn..")
    linear_reg = linear_model.LinearRegression()
    #print("Dimensions: X: " + str(predictors.ndim) + ", Y: " + str(target.ndim))
    # IMPORTANT: LinearRegression expects a 2-D array. So, add a dimension using
    # reshape()
    #linear_reg.fit(x_data.reshape(-1,1), y_data.reshape(-1,1))
    linear_reg.fit(predictors, target)
    print("Slope : " + str(linear_reg.coef_))
    print("Intercept: " + str(linear_reg.intercept_))
    return (linear_reg.coef_, linear_reg.intercept_)

### TensorFlow method
def tensorflow_method(x_data, y_data, learn_rate, epochs):
    print("Tensor flow method..")
    graph = tf.Graph()
    with graph.as_default():
        slope = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
        x1 = tf.placeholder(dtype=np.float32)
        x2 = tf.placeholder(dtype=np.float32)
        y = tf.placeholder(dtype=np.float32)

        #w1 = tf.Variable([0], dtype=np.float32, name="weight1")
        #w2 = tf.Variable([0], dtype=np.float32, name="weight2")
        w1 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
        w2 = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
        b = tf.Variable([0], dtype=np.float32, name="bias")

        response = w1*x1 + w2*x2 + b
        cost = tf.reduce_mean(tf.square(response - y))
        optimizer = tf.train.GradientDescentOptimizer(learn_rate).minimize(cost)

        with tf.Session(graph=graph) as session:
            init = tf.global_variables_initializer()
            session.run(init)

            x1_list = x_data[:,0].tolist()
            x2_list = x_data[:,1].tolist()
            y_list = y_data[:,0].tolist()
            print(x1_list[0:6])
            print(x2_list[0:6])
            print(y_list[0:6])
            for epoch in range(epochs):
                session.run(optimizer, {x1:x1_list, x2:x2_list, y:y_list})
                if (epoch % 10000) == 0:
                    print("w1 = ",session.run(w1))
        print("w2 = ",session.run(w2))
        print("b = ",session.run(b))

    return(session.run(w1), session.run(w2), session.run(b))

predictors, target = extract_predictor_target()
print(predictors)
print(target)
print(predictors[:,0], predictors[:,1], target[:,0])

#Get the linear regression using SciKit Learn
coef, intercept = scikit_method(predictors, target)
print(coef[0][0])
print(coef[0][1])
print(intercept[0])
plot_plane(predictors, target, coef[0][0], coef[0][1], intercept[0])
w1, w2, b = tensorflow_method(predictors, target, 0.008, 1000000)
#plot_plane(predictors, target, w1, w2, b)

print("RESULTS" + "\n" + "-------")
print("SciKit Learn : w1: " + str(coef[0][0]) + " w2: " + str(coef[0][1]) + " b: " + str(intercept[0]))
print("Tensorflow: w1: " + str(w1) + " w2: " + str(w2) + " b: " + str(b))
```

Solution plot





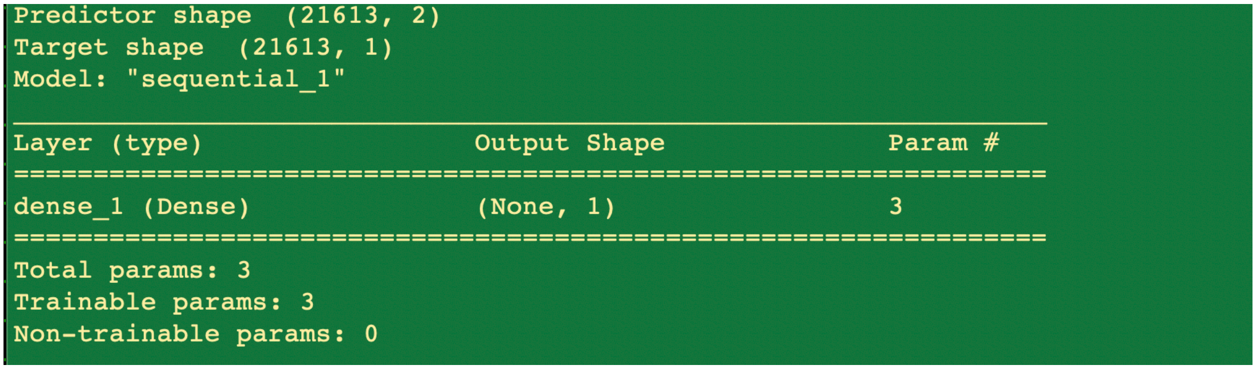
```
RESULTS
-------
SciKit Learn : w1: -0.24697744845331465 w2: 0.5455501754638818 b: 0.01252648891541263
Tensorflow: w1: [-0.24527158] w2: [0.5447634] b: [0.0124585]
```

The SciKit learn values and TensorFlow values are extremely close.

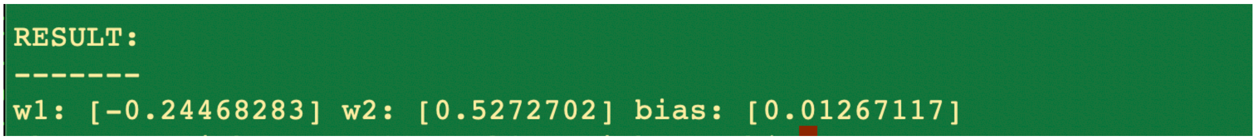
Problem 4

```
'''
Week 3 - Problem 4
Solution using Keras

'''
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import keras
from keras.models import Sequential
from keras.layers import Dense
from sklearn import preprocessing

# Extract data from the CSV file
def extract_predictor_target():
    data = pd.read_csv('kc_house_data.csv')
    print(data.head())
    predictors = preprocessing.minmax_scale(data[['bedrooms','sqft_living']])
    target = preprocessing.minmax_scale(data[['price']])
    #plot_data(predictors, target)
    print(predictors.ndim)
    print(target.ndim)
    return (predictors, target)

# Keras model definition and execution
def model_keras(x_data, y_data, epochs):
    model = Sequential()
    model.add(Dense(1, input_dim=2, kernel_initializer='normal', activation='linear'))

    #Compile the model
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mse'])

    #Dump the model
    model.summary()
```

```
   #Suppressing the per-epoch messages
   hist = model.fit(x_data, y_data, epochs=epochs, verbose=0)

   weightBias = model.layers[0].get_weights()
   return weightBias

predictors, target = extract_predictor_target()
print('Predictor shape ', predictors.shape)
print('Target shape ', target.shape)

#Convert to arrays
pred_array = np.array(predictors)
target_array = np.array(target)
weightBias = model_keras(pred_array, target_array, 100)
print("RESULT:" + "\n" + "-------")
print("w1: " + str(weightBias[0][0]) + " w2: " + str(weightBias[0][1]) + " bias: " +
str(weightBias[1]))
```



```
Predictor shape  (21613, 2)
Target shape  (21613, 1)
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
```



```
RESULT:
-------
w1: [-0.24468283] w2: [0.5272702] bias: [0.01267117]
```

It can be seen that the values are consistent across SciKit learn, TensorFlow and Keras