Hand-in 3, Part 2: Data Modeling

In this part you will take the csv file "reduced_field_data.csv" from Part 1, and use it to estimate the line criticality indices.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Score function

Below is a scoring function we have written for you. The output of this function is a score of how well your procedure is doing. Higher scores are worse!

As you can see, it penalizes heavily when you don't predict accurately values when they are being overloaded, i.e. false negatives.

In [2]:
```
def score_func(y_est, y_real):
    """
        This function takes your estimates y_est and
        scores them against the real data y_real.
        You should use this function to show how good your estimation method is.
    """
    # Square deviation
    sqr_err = np.sum(((y_est - y_real)**2).sum())
    # Penalty for not estimating a critical value above 0.95 when it occurs.
    false_negative = np.sum(np.where(np.logical_and(y_est < 0.95, y_real > 0.95),
    return sqr_err + false_negative
```

Load and clean data

Load your data from the previous exercise as well as the criticality data.

In [27]:
```
# Data is loaded here
field_data = pd.read_csv("reduced_field_data.csv", index_col=0)
crit_data = pd.read_csv("flow_criticality_data.csv", index_col=0)
crit_data.describe()
```

Out[27]:

	flow_89_94	flow_20_23	flow_49_64	flow_101_532	flow_26_28	flow_172_176	flow_122_
count	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000000	8784.000
mean	0.217201	0.275454	0.333612	0.637028	0.154337	0.350644	0.572
std	0.225644	0.220928	0.285684	0.303556	0.215251	0.215765	0.431
min	0.000028	0.000057	0.000102	0.000372	0.000000	0.000020	0.000
25%	0.068380	0.111416	0.099566	0.394867	0.024677	0.186376	0.066
50%	0.148103	0.222921	0.230735	0.684006	0.078545	0.337664	0.940
75%	0.296423	0.381735	0.555277	0.913541	0.185999	0.470791	0.959
max	2.509490	1.578194	1.432856	1.010000	2.121284	1.394635	1.094

Linear regression

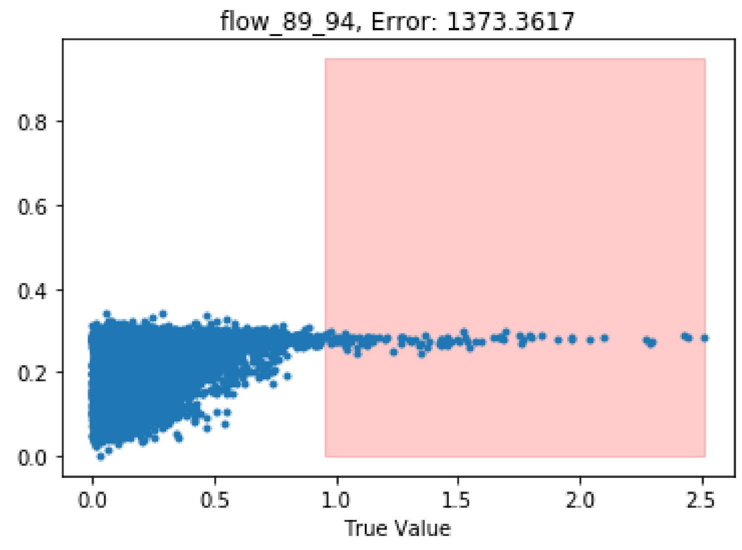
We have implemented a simple linear regression to apply to your data. You should use this as a benchmark for your neural network below.

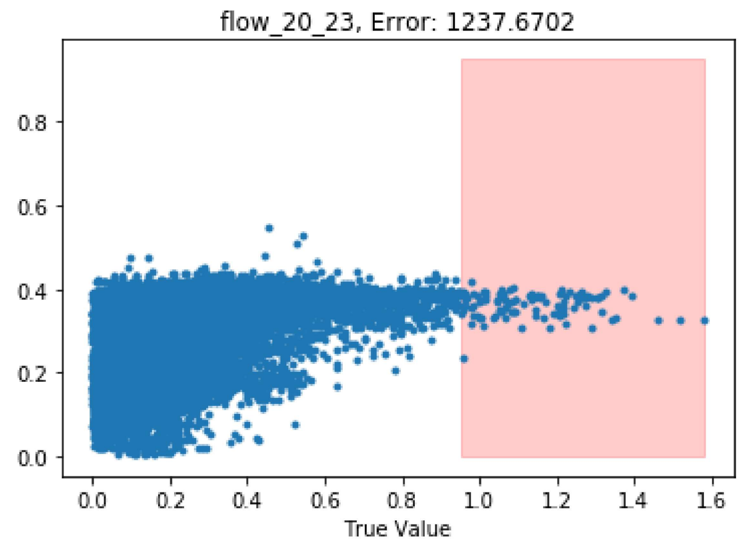
In [29]:
```
# Because the two dataframes is not the same length

crit_data2 = crit_data[crit_data.index.isin(field_data['time'])].dropna(axis=0)
print("Crit_data shape:", crit_data.shape)
print("Crit_data2 shape:", crit_data2.shape)
field_data = field_data.drop('time', 1)
print("Field_data shape:", field_data.shape)
print("The shapes now fit")
```

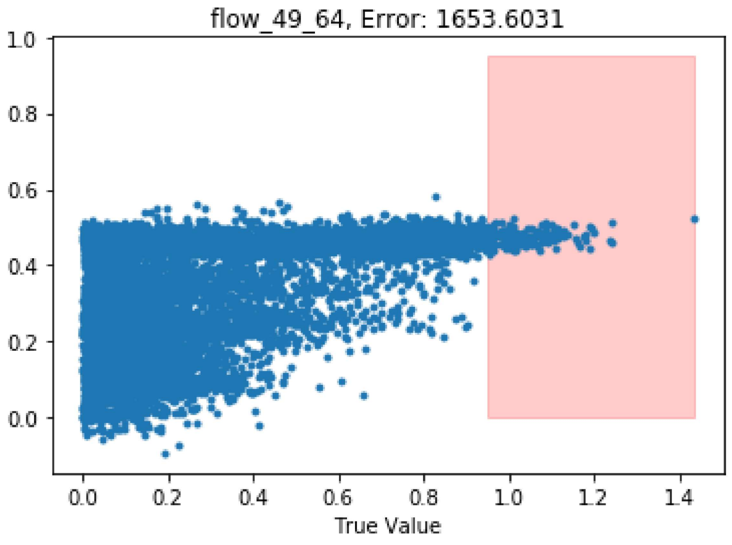
```
Crit_data shape: (8784, 10)
Crit_data2 shape: (8311, 10)
Field_data shape: (8311, 20)
The shapes now fit
```

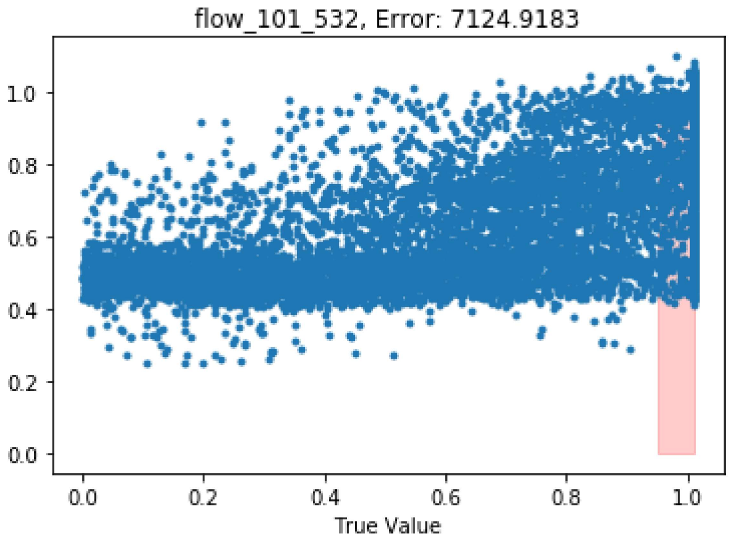
In [17]:
```
# The code below implements a linear regression on your data and compares predict
# On the plots there is a red square indicating the areas corresponding to false 
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(field_data,crit_data2)

y_pred = lm.predict(field_data)
for index in range(10):
    plt.figure()
    plt.fill_between([0.95, crit_data2.iloc[:,index].max()], [0.95, 0.95], color=
    plt.plot(crit_data2.iloc[:,index],y_pred[:,index],'.')
    error = score_func(y_pred[:,index],crit_data2.iloc[:,index])
    plt.title('{0}, Error: {1:.04f}'.format(crit_data2.columns[index], error))
    plt.xlabel('True Value')

plt.ylabel('Predicted Value')
print('Overall error: {0:.04f}'.format(score_func(y_pred, crit_data2)))
```

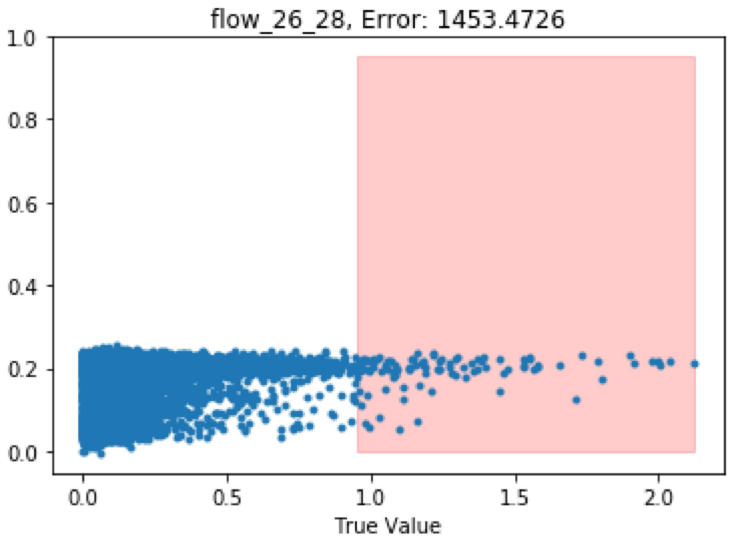
```
Overall error: 32308.8247
```

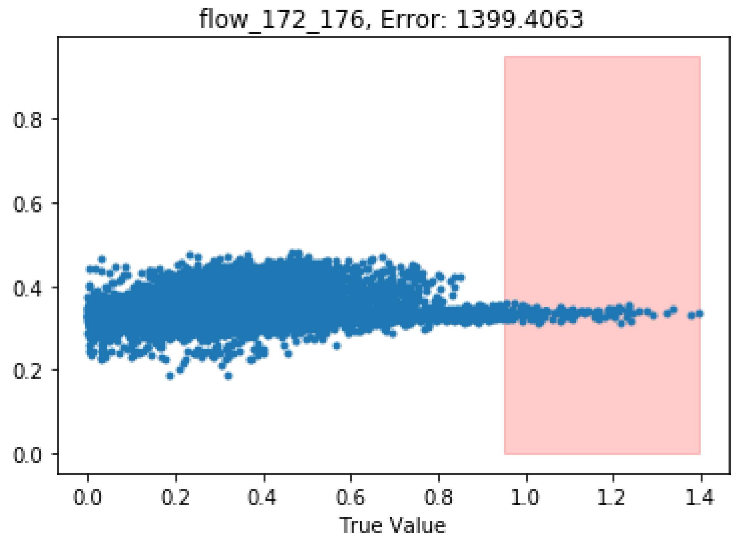


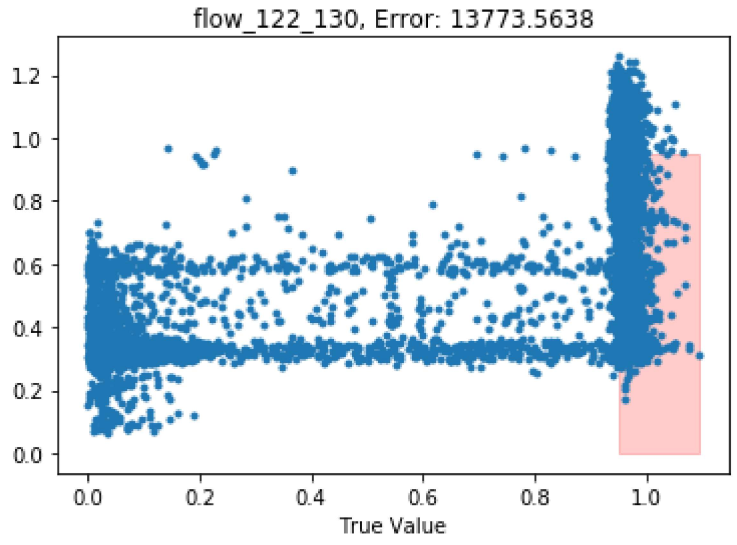


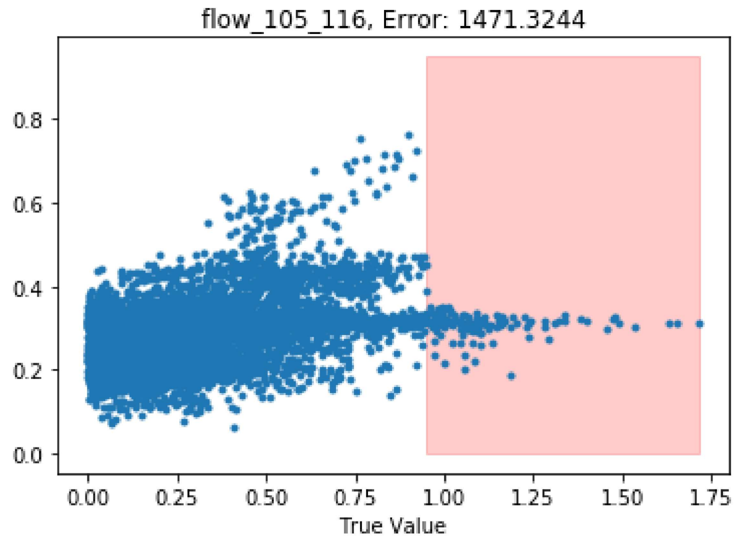

flow_49_64, Error: 1653.6031
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
True Value

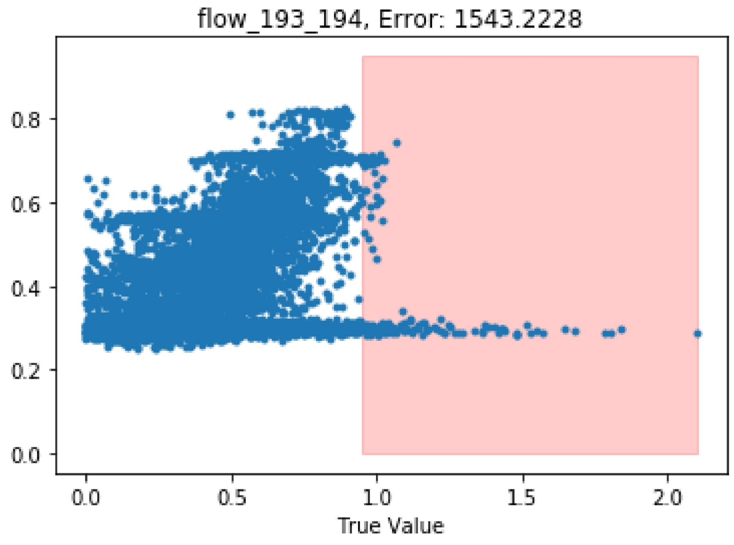

flow_101_532, Error: 7124.9183
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
True Value

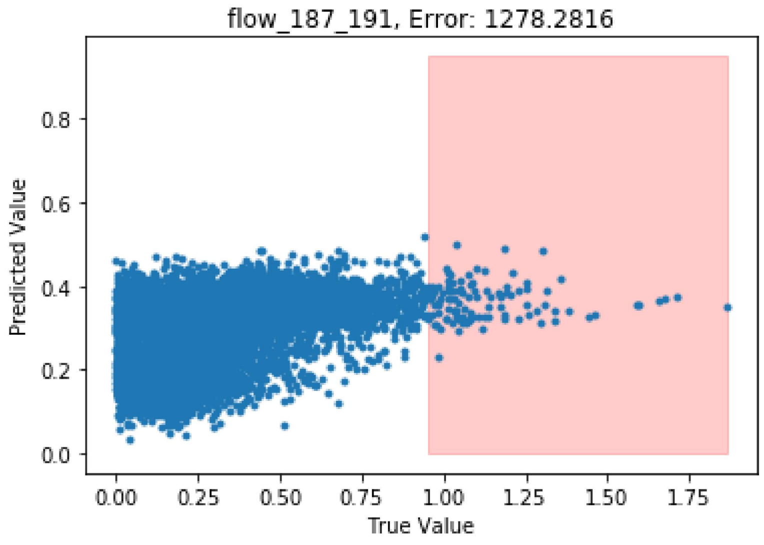

flow_26_28, Error: 1453.4726
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
True Value


flow_172_176, Error: 1399.4063
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
True Value


flow_122_130, Error: 13773.5638
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
0.2
0.4
0.6
0.8
1.0
True Value


flow_105_116, Error: 1471.3244
0.0
0.2
0.4
0.6
0.8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
True Value





Q#6 Would you recommend using Linear Regression for estimating line criticality indices? Why/why not?

No! Linear regression is kinda the default regression. It should be used as a baseline to compare the real model to. In some ideal cases linear regression might be the best, but if you are working with real-world data it is uncommon.

Neural Network

You should now produce a neural network that can estimate criticality indices. Split your data into a training and test set, build your neural network below, and use the score_func defined above to estimate your model quality.

Q#7 Explain why you chose the layers, nodes and activation functions you did. (No wrong answers, we want to know your thought process!) What is the best score you can get with a single layer?

Since we are trying to use a regression problem (continuous output) we selected the regular dense nodes and a multilayer perceptron architecture.

From our initial description of the dataset, we can see that there are no negative values, but values do exceed 1. Based on this we select the rectified linear unit (relu) as our activation function.

As a loss function we choose 'mean squared error' because we want outliers to count a lot.

```
In [35]: # Split data into training and test
         from sklearn.model_selection import train_test_split

         x_train, x_test, y_train, y_test = train_test_split(field_data,
                                                             crit_data2,
                                                             test_size=0.25,
                                                             random_state=42)
```

```
In [57]: #INSERT Neural Network Code here for a linear regression.

         # Remember to import the kind of layers you want from the keras library.
         from keras.models import Sequential
         from keras.layers import Dense, Activation
         from keras import optimizers
         from keras import losses

         model = Sequential()
         model.add(Dense(10, input_shape=(20,)))
         model.add(Activation('relu'))
         adam = optimizers.Adam(lr=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0
         model.compile(loss='mean_squared_error', optimizer=adam)
         history1 = model.fit(x_train.values, y_train.values, epochs=10, batch_size=500, s
         y_pred1 = model.predict(x_test.values, batch_size=1000)

         print("Score: ", score_func(y_pred1, y_test))
```

```
Epoch 1/10
6233/6233 [==============================] - 0s - loss: 15019.6183
Epoch 2/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 3/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 4/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 5/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 6/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 7/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 8/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 9/10
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 10/10
6233/6233 [==============================] - 0s - loss: 0.2182
Score:  19369.364132
```

K-fold cross validation

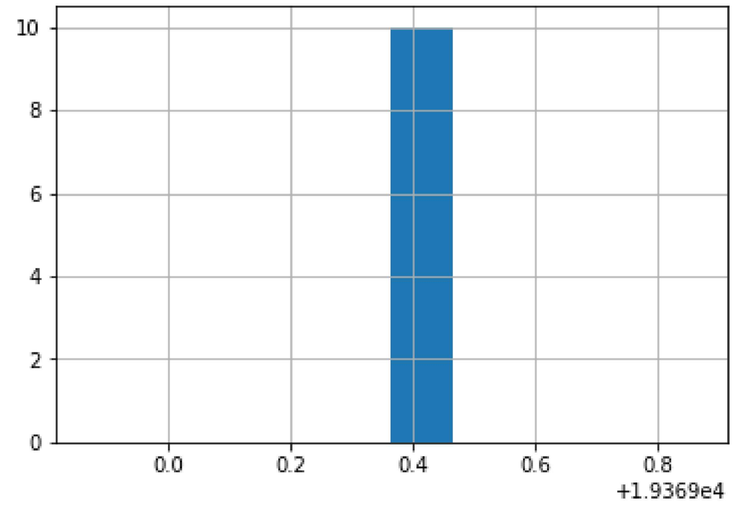
To ensure that your neural network actually works when presented with new data, take the neural network you defined above, and perform a k-fold cross validation on it.

Q#8 Using a test window size of one tenth of your data (ten-fold cross validation), plot a histogram of the output of score_func for the validation. Tweak your neural network to achieve the lowest mean score.H

In [58]:
```
# Your k-fold code goes here.
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras import optimizers

listOfErrors2 = list()
kf1 = KFold(n_splits=10,shuffle=True)
for train_index, test_index in kf1.split(field_data):
    X_train, X_test = field_data.iloc[train_index], field_data.iloc[test_index]
    Y_train, Y_test = crit_data2.iloc[train_index], crit_data2.iloc[test_index]
    #print(X_test)
    model = Sequential()
    model.add(Dense(10, input_shape=(20,)))
    model.add(Activation('relu'))

    adam = optimizers.Adam(lr=0.05, beta_1=0.9, beta_2=0.999, epsilon=1e-08, deca
    model.compile(loss='mean_squared_error', optimizer=adam)
    history1 = model.fit(x_train.values,y_train.values, epochs=20, batch_size=500
    y_pred2 = model.predict(x_test.values, batch_size=1000)
    listOfErrors2.append(score_func(y_pred2, y_test))
```

```
6233/6233 [==============================] - 0s - loss: 0.2208
Epoch 3/20
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 4/20
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 5/20
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 6/20
6233/6233 [==============================] - ETA: 0s - loss: 0.218 - 0s - l
oss: 0.2182
Epoch 7/20
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 8/20
6233/6233 [==============================] - 0s - loss: 0.2182
Epoch 9/20
6233/6233 [==============================] - ETA: 0s - loss: 0.220 - 0s - l
oss: 0.2182
Epoch 10/20
6233/6233 [==============================] - 0s - loss: 0.2182
```

In [49]:
```
plt.hist(listOfErrors2)
plt.grid(True)
plt.show()

print("Mean score: " , np.mean(listOfErrors2))
print("KFold scores: ", listOfErrors2)
```



```
Mean score:  19369.364132
KFold scores:  [19369.364131972798, 19369.364131972798, 19369.364131972798, 193
69.364131972798, 19369.364131972798, 19369.364131972798, 19369.364131972798, 19
369.364131972798, 19369.364131972798, 19369.364131972798]
```

Q#9 Are you satisfied with the quality of the final estimator? What would be your recommendation for your boss on this issue?

It seems that the predictor converges to the same value, with a much better score than our baseline prediction (linear regression).

Baseline score: 32308.8247 Neural Network score: 19369.3641

In future projects it would be recommended to experiment with our selection methods for the 20 sensors.

Extra task for 3-person groups

Q#10 Repeat the process, once reducing to 15 sensors, and once reducing to 25 sensors. The costs to keep these sensors running is directly proportional to the number of sensors used. Does changing the number of sensors used change your conclusion in Q#9?

This cell intentionally left blank