```
data = pandas.read_csv("dog-training.csv", delimiter="\t")

print(data.shape)
print(data.head())
```

We're interested in the relationship between a dog's weight and the amount of rescues it performed in the previous year. Let's begin by plotting `rescues_last_year` as a function of `weight_last_year` :

```
import graphing
import statsmodels.formula.api as smf

# First, we define our formula using a special syntax
# This says that rescues_last_year is explained by weight_last_year
formula = "rescues_last_year ~ weight_last_year"

model = smf.ols(formula = formula, data = data).fit()

graphing.scatter_2D(data, "weight_last_year", "rescues_last_year", trendline = lambda x: model
```

There seems to be a pretty clear relationship between a dog's weight and the number of rescues it's performed. That seems pretty reasonable, as we'd expect heavier dogs to be bigger and stronger and thus better at saving lives!

Alt + A

Train/test split

This time, instead of fitting a model to the entirety of our dataset, we're going to separate our dataset into two smaller partitions: a *training set* and a *test set*.

The *training set* is the largest of the two, usually made up of between 70-80% of the overall dataset, with the rest of the dataset making up the *test set*.

By splitting our data, we're able to gauge the performance of our model when confronted with previously unseen data.

```
from sklearn.model_selection import train_test_split


# Obtain the label and feature from the original data
dataset = data[['rescues_last_year','weight_last_year']]

# Split the dataset in an 70/30 train/test ratio. We also obtain the respective corresponding 
train, test = train_test_split(dataset, train_size=0.7, random_state=21)

print("Train")
print(train.head())
print(train.shape)

print("Test")
print(test.head())
print(test.shape)
```

We notice that these sets are different, and that the *training set* and *test set* contain 70% and 30% of the overall data, respectively.

Let's have a look at how the *training set* and *test set* are separated out:

Alt + A

```
# You don't need to understand this code well
```

```
    # it's just used to create a scatter plot

    # concatenate training and test so they can be graphed
    plot_set = pandas.concat([train,test])
    plot_set["Dataset"] = ["train"] * len(train) + ["test"] * len(test)

    # Create graph
    graphing.scatter_2D(plot_set, "weight_last_year", "rescues_last_year", "Dataset", trendline = 
```

Training Set

We begin by training our model using the *training set*, testing its performance with the same *training set*:

```
    import statsmodels.formula.api as smf
    from sklearn.metrics import mean_squared_error as mse

    # First, we define our formula using a special syntax
    # This says that rescues_last_year is explained by weight_last_year
    formula = "rescues_last_year ~ weight_last_year"

    # Create and train the model
    model = smf.ols(formula = formula, data = train).fit()

    # Graph the result against the data
    graphing.scatter_2D(train, "weight_last_year", "rescues_last_year", trendline = lambda x: mode
```

We can gauge our model's performance by calculating the *mean squared error* (MSE)

Alt + A

```
# We use the in-buit sklearn function to calculate the MSE
correct_labels = train['rescues_last_year']
predicted = model.predict(train['weight_last_year'])

MSE = mse(correct_labels, predicted)
print('MSE = %f ' % MSE)
```

Test Set

Next, we test the same model's performance using the *test set*:

```
graphing.scatter_2D(test, "weight_last_year", "rescues_last_year", trendline = lambda x: model
```

Let's have a look at the MSE again.

```
correct_labels = test['rescues_last_year']
predicted = model.predict(test['weight_last_year'])

MSE = mse(correct_labels, predicted)
print('MSE = %f ' % MSE)
```

Alt + A

We learn that the model performs much better on the known *training data* than on the unseen *test data* (remember that higher MSE values are worse).

The reason can be due to a number of factors, but first and foremost is *overfitting*, which is when a model matches the data in the *training set* too closely. This means that it will perform very well on the *training set*, but will not *generalize* well. (that is, it won't work well with other datasets).

New Dataset

To illustrate our point further, let's have a look at how our model performs when confronted with a completely new, unseen, and larger dataset. For our scenario, we'll use data provided by the avalanche rescue charity's European branch.

```
# Load an alternative dataset from the charity's European branch
new_data = pandas.read_csv("dog-training-switzerland.csv", delimiter="\t")

print(new_data.shape)
new_data.head()
```

The features are the same, but we have much more data this time. Let's see how our model does!

```
# Plot the fitted model against this new dataset.

graphing.scatter_2D(new_data, "weight_last_year", "rescues_last_year", trendline = lambda x: m
```

And now, the MSE:

```
    correct_labels = new_data['rescues_last_year']
    predicted = model.predict(new_data['weight_last_year'])

    MSE = mse(correct_labels, predicted)
    print('MSE = %f ' % MSE)
```

As expected, the model performs better on the training dataset as it does on the unseen dataset. This is simply due to overfitting, as we noted previously.

Interestingly, the model performs better on this unseen dataset than it does on the *test set*. This is because our previous test set was quite small, and thus not a very good representation of real-world data. By contrast, this unseen dataset is large and a much better representation of data we'll find outside of the lab. In essence, this shows us that part of performance difference we see between training and test is due to model overfitting, and part of the error is due to the test set not being perfect. In the next exercises, we'll explore the trade-off we have to make between training and test dataset sizes.

Summary

In this exercise, we covered the following concepts:

- Splitting a dataset into a *training set* and a *test set*
- Training a model using the *training set* and testing its performance on the *training set*, *test set*, and on a new, unseen dataset
- Compared the respective MSEs to highlight the effects and dangers of *overfitting*

No compute Compute not connected Viewing Kernel not connected

Next unit: Nuances of test sets

Alt + A

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

Alt + A