In this exercise, we'll have a look at a different type of regression called *polynomial regression*. In contrast to *linear regression* ,which models relationships as straight lines, *polynomial regression* models relationships as curves.

Recall in our previous exercise how the relationship between `core_temperature` and `protein_content_of_last_meal` couldn't be properly explained using a straight line. In this exercise, we'll use *polynomial regression* to fit a curve to the data instead.

```
import pandas
!pip install statsmodels
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/
!wget https://raw.githubusercontent.com/MicrosoftDocs/mslearn-introduction-to-machine-learning/

#Import the data from the .csv file
dataset = pandas.read_csv('doggy-illness.csv', delimiter="\t")

#Let's have a look at the data
dataset
```

Simple Linear Regression

Let's quickly jog our memory by performing the same *simple linear regression* as we did in the previous exercise, using the `temperature` and `protein_content_of_last_meal` columns of the dataset.

```
import statsmodels.formula.api as smf
import graphing # custom graphing code. See our GitHub repo for details

# Perform linear regression. This method takes care of
# the entire fitting procedure for us.
simple_formula = "core_temperature ~ protein_content_of_last_meal"
simple_model = smf.ols(formula = simple_formula, data = dataset).fit()

# Show a graph of the result
graphing.scatter_2D(dataset, label_x="protein_content_of_last_meal",
                            label_y="core_temperature",
```

```
trendline=lambda x: simple_model.params[1] * x + simple_model.par
```

Notice how the relationship between the two variables is not truly linear. Looking at the plot, it's fairly clear to see that the points tend more heavily towards one side of the line, especially for the higher `core-temperature` and `protein_content_of_last_meal` values.

A straight line might not be the best way to describe this relationship.

Let's have a quick look at the model's R-Squared score:

```
print("R-squared:", simple_model.rsquared)
```

That is quite a reasonable R-Squared score, but let's see if we can get an even better one!

Simple Polynomial Regression

Let's fit a *simple polynomial regression* this time. Similar to a *simple linear regression*, a *simple polynomial regression* models the relationship between a label and a single feature. Unlike a *simple linear regression*, a *simple polynomial regression* can explain relationships that aren't simply straight lines.

In our example, we're going to use a three-parameter polynomial.

```
# Perform polynomial regression. This method takes care of
# the entire fitting procedure for us.
polynomial_formula = "core_temperature ~ protein_content_of_last_meal + I(protein_content_of_l
polynomial_model = smf.ols(formula = polynomial_formula, data = dataset).fit()

# Show a graph of the result
```

```
graphing.scatter_2D(dataset, label_x="protein_content_of_last_meal",
                            label_y="core_temperature",
                            # Our trendline is the equation for the polynomial
                            trendline=lambda x: polynomial_model.params[2] * x**2 + polynomia
```

That looks a lot better already. Let's confirm by having a quick look at the R-Squared score:

```
print("R-squared:", polynomial_model.rsquared)
```

That's a better R-Squared score than the one obtained from the previous model! We can now confidently tell our vet to prioritize dogs who ate a high-protein diet the night before.

Let's chart our model as a 3D chart. We'll view X and X^2 as two separate parameters. Notice that if you rotate the visual just right, our regression model is still a flat plane. This is why polynomial models are still considered to be `linear models`.

```
import numpy as np
fig = graphing.surface(
    x_values=np.array([min(dataset.protein_content_of_last_meal), max(dataset.protein_content_
    y_values=np.array([min(dataset.protein_content_of_last_meal)**2, max(dataset.protein_conte
    calc_z=lambda x,y: polynomial_model.params[0] + (polynomial_model.params[1] * x) + (polyno
    axis_title_x="x",
    axis_title_y="x2",
    axis_title_z="Core temperature"
)
# Add our datapoints to it and display
```

```
# Add our datapoints to it and display
fig.add_scatter3d(x=dataset.protein_content_of_last_meal, y=dataset.protein_content_of_last_me
fig.show()
```

Extrapolating

Let's see what happens if we extrapolate our data. We'd like to see if dogs that ate meals even higher in protein are expected to get even sicker.

Let's start with the *linear regression*. We can set what range we'd like to extrapolate our data over by using the `x_range` argument in the plotting function. Let's extrapolate over the range `[0,100]`:

```
# Show an extrapolated graph of the linear model
graphing.scatter_2D(dataset, label_x="protein_content_of_last_meal",
                            label_y="core_temperature",
                            # We extrapolate over the following range
                            x_range = [0,100],
                            trendline=lambda x: simple_model.params[1] * x + simple_model.par
```

Next, we extrapolate the *polynomial regression* over the same range:

```
# Show an extrapolated graph of the polynomial model
graphing.scatter_2D(dataset, label_x="protein_content_of_last_meal",
                            label_y="core_temperature",
                            # We extrapolate over the following range
                            x_range = [0,100],
                            trendline=lambda x: polynomial_model.params[2] * x**2 + polynomia
```

These two graphs predict two very different things!

The extrapolated *polynolmial regression* expects `core_temperature` to go down, while the extrapolated *linear regression* expects linear expects `core_temperature` to go up.

A quick look at the graphs obtained in the previous exercise confirms that we should expect the `core_temeprature` to rise, not fall, as the `protein_content_of_last_meal` increases.

In general, it's not recommended to extrapolate from a *polynomial regression* unless you have an a-priori reason to do so (which is only very rarely the case, so it's best to err on the side of caution, and never extrapolate from *polynomial regressions*).

Summary

We covered the following concepts in this exercise:

- Built *simple linear regression* and *simple polynomial regression* models.
- Compared the performance of both models by plotting them and looking at R-Squared values.
- Extrapolated the models over a wider range of values.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆