Linear Regression in StatsModels

Introduction

So far, you learned how linear regression and R-Squared (coefficient of determination) work "under the hood" and created your own versions using NumPy.

Going forward, you're going to use a Python library called StatsModels to do the modeling and evaluation work for you!

Objectives

You will be able to:

- Perform a linear regression using StatsModels
- Evaluate a linear regression model using StatsModels
- Interpret linear regression model coefficients using StatsModels

What is StatsModels?

StatsModels is a powerful Python package for many types of statistical analyses. In particular, as you may have guessed from the name, StatsModels emphasizes statistical *modeling*, particular linear models and time series analysis. You can check out the User Guide (https://www.statsmodels.org/stable/user-guide.html) for an overview of all of the available models.

When using StatsModels, we'll need to introduce one more set of terminology: ***endogenous*** and ***exogenous*** variables. You'll see these as argument names `endog` and `exog` in the documentation (https://www.statsmodels.org/stable/generated/statsmodels.regression.linear_model.OLS.html) for the models, including `OLS` (ordinary least squares linear regression). These are simply the names used by StatsModels for the independent and dependent variables.

This table is drawn from the StatsModels documentation (https://www.statsmodels.org/stable/endog_exog.html):

endog	exog
y	x
dependent variable	independent variable
response variable	explanatory variable

Importing Necessary Libraries

Most of the import statements below should already be familiar. The one new import statement is

```
import statsmodels.api as sm
```

Going forward, `sm` refers to StatsModels.

In [1]:
```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
import numpy as np
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import statsmodels.api as sm

import warnings
warnings.filterwarnings('ignore')
```

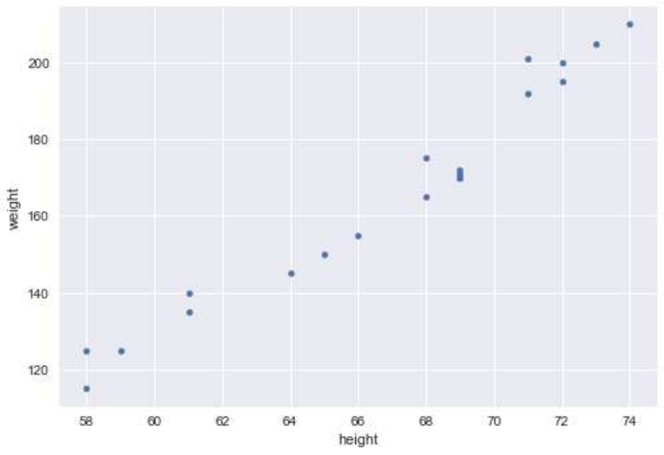
Loading the Data

Let's load a simple dataset for the purpose of understanding the process.

The dataset contains weight and height, and we'll set height as the independent variable (x) and weight as the dependent variable.

We'll start out by plotting them against each other in a scatter plot:

In [2]:
```
df = pd.read_csv('heightWeight.csv')
df.plot.scatter(x="height", y="weight");
```



Looks like a linear relationship!

Building Our Linear Regression

One way to build a linear regression would be by hand, calculating $\hat{m}$ and $\hat{c}$ with the least-squares formula:

In [3]:
```
rho = np.corrcoef(df["height"], df["weight"])[0][1]
s_y = df["weight"].std()
s_x = df["height"].std()
m = rho * s_y / s_x

mean_y = df["weight"].mean()
mean_x = df["height"].mean()
c = mean_y - m * mean_x

print(f"Our regression line is: y = {round(m, 5)}x + {round(c, 5)}")
```

```
Our regression line is: y = 5.53902x + -204.48344
```

That works, in a way. But now if we wanted to calculate the F-statistic, or R-Squared, or find the confidence intervals for the parameters, that would be a lot of additional calculations.

If we use StatsModels instead, all of the calculations are done for us! We just have to set up the appropriate x and y variables and plug them into a model.

Determining `X` and `y` Variables

Technically you can subset a pandas dataframe in the same line where you create the model, but it tends to be easier if you specify your variables first.

In [4]:
```
X = df[["height"]]
y = df["weight"]
```

You might notice that `X` is capitalized and `y` is not. This is not a mistake! The idea is to indicate that `X` is a 2D matrix, and `y` is a 1D array. Currently `X` only has one value in it (height) but this will change when we get to multiple regression.

Creating the Model

With StatsModels, the "model" is the result of calling the `OLS` constructor function.

We will also use the `add_constant` method because StatsModels expects a column of constant values if there should be a constant in the resulting regression.

In [5]:
```
model = sm.OLS(endog=y, exog=sm.add_constant(X))
model
```

Out[5]:
```
<statsmodels.regression.linear_model.OLS at 0x10dc19b80>
```

Fitting the Model

Once we have a model, we call the `fit` method, which returns a results object.

```
In [6]: results = model.fit()
        results
```

```
Out[6]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x13150af40>
```

Evaluating and Interpreting the Model

Now we can get all kinds of different information out of the model results!

F-statistic and F-statistic p-value

```
In [7]: results.fvalue, results.f_pvalue
```

```
Out[7]: (384.774028857076, 1.3461445985228957e-13)
```

Interpretation: Our model is statistically significant, with a p-value well below the standard alpha of 0.05

R-Squared (coefficient of determination):

```
In [8]: results.rsquared
```

```
Out[8]: 0.9553099288673668
```

Interpretation: Our model explains about 95.5% of the variance in weight, the dependent variable

Model parameters:

```
In [9]: results.params
```

```
Out[9]: const    -204.483436
        height      5.539019
        dtype: float64
```

Interpretation: For a height of 0, our model would predict a weight of about -204.5. An increase of 1 in height is associated with an increase of about 5.5 in weight.

Notes:

1. These are the same values we got from our "by hand" regression earlier! `m` corresponds to `height` in this output, and `c` corresponds to `const`
2. Intercept values are often nonsensical if a value of 0 in the independent variable is nonsensical. So don't worry too much about making sense of what a height of -204.5 for a weight of 0 means

Model parameter p-values:

```
In [10]: results.pvalues
```

```
Out[10]: const     2.688182e-09
         height    1.346145e-13
         dtype: float64
```

Interpretation: Both of our model parameters (coefficient for `height` and intercept) are statistically significant, with p-values well below the standard alpha of 0.05.

Model parameter confidence intervals:

```
In [11]: print(results.conf_int())
```

```
                 0           1
const  -244.252410 -164.714462
height    4.945766    6.132272
```

Interpretation: Our 95% confidence interval for the intercept is about -244.3 to about -164.7. Our 95% confidence interval for the coefficient of `height` is about 4.9 to about 6.1.

The Results Summary

Instead of extracting these individual values from the results object, there is also a method `summary` that will print out all of this information at once. It can get overwhelming, but take a look and see if you can find all of the information extracted above:

- F-statistic and F-statistic p-value
- R-Squared (coefficient of determination)
- Parameters
- Parameter p-values
- Parameter confidence intervals

Hints (click to reveal)

In [12]:
```
print(results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                 weight   R-squared:                       0.955
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     384.8
Date:                Fri, 06 May 2022   Prob (F-statistic):           1.35e-13
Time:                        15:24:18   Log-Likelihood:                -64.112
No. Observations:                  20   AIC:                             132.2
Df Residuals:                      18   BIC:                             134.2
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -204.4834     18.929    -10.802      0.000    -244.252    -164.714
height         5.5390      0.282     19.616      0.000       4.946       6.132
==============================================================================
Omnibus:                        2.588   Durbin-Watson:                   2.053
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                1.245
Skew:                           0.202   Prob(JB):                        0.537
Kurtosis:                       1.846   Cond. No.                         902.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

You can also omit the `print` part to see a summary styled by the notebook. Different people may find the version above or below more readable; either are fine.

In [13]:
```
results.summary()
```

Out[13]:

OLS Regression Results

Dep. Variable:	weight	**R-squared:**	0.955
Model:	OLS	**Adj. R-squared:**	0.953
Method:	Least Squares	**F-statistic:**	384.8
Date:	Fri, 06 May 2022	**Prob (F-statistic):**	1.35e-13
Time:	15:24:18	**Log-Likelihood:**	-64.112
No. Observations:	20	**AIC:**	132.2
Df Residuals:	18	**BIC:**	134.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-204.4834	18.929	-10.802	0.000	-244.252	-164.714
height	5.5390	0.282	19.616	0.000	4.946	6.132

Omnibus:	2.588	**Durbin-Watson:**	2.053
Prob(Omnibus):	0.274	**Jarque-Bera (JB):**	1.245
Skew:	0.202	**Prob(JB):**	0.537
Kurtosis:	1.846	**Cond. No.**	902.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

This summary contains a **lot** of information. If you're curious about all of the fields, feel free to expand the below explanation:

Regression results explanation (click to expand)

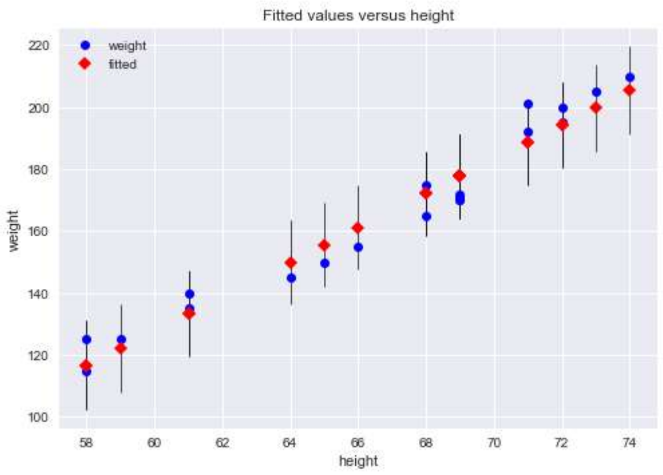
Visualizing Our Model

Plotting Fit

StatsModels also comes with some plotting utilities that are particularly useful for statistical modeling. You can find the full list [here (https://www.statsmodels.org/stable/graphics.html#regression-plots)](https://www.statsmodels.org/stable/graphics.html#regression-plots).

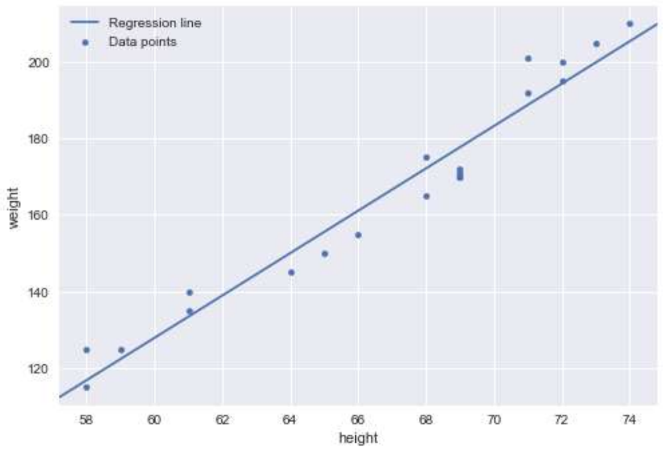
This method shows the scatter plot of actual values as well as points indicating the values predicted by the model. The black vertical lines represent the confidence intervals for each prediction. (These confidence intervals use the t-distribution.)

In [14]:
```
sm.graphics.plot_fit(results, "height")
plt.show()
```



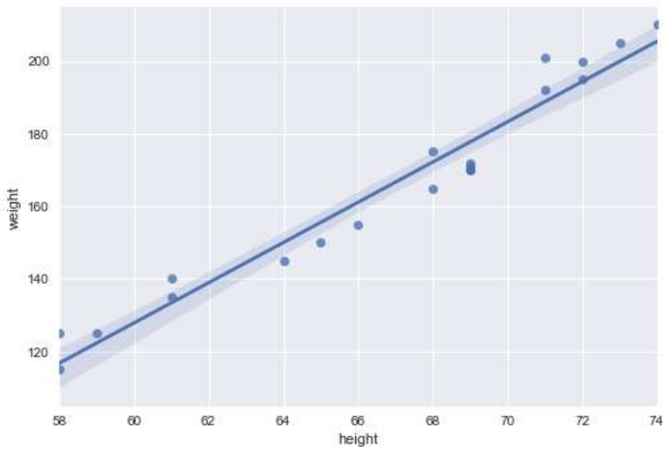
For a simpler visualization, you can use `abline_plot` from StatsModels to add a best-fit line to a regular scatter plot.

In [15]:
```
fig, ax = plt.subplots()
df.plot.scatter(x="height", y="weight", label="Data points", ax=ax)
sm.graphics.abline_plot(model_results=results, label="Regression line", ax=ax)
ax.legend();
```



There is also a Seaborn function, `regplot` ([documentation here (https://seaborn.pydata.org/generated/seaborn.regplot.html)](https://seaborn.pydata.org/generated/seaborn.regplot.html)), that can be useful for visualizing the original data alongside the regression line. In this case the shaded region also represents the confidence interval, which is computed using a different technique called bootstrapping.

In [16]:
```
sns.regplot(x="height", y="weight", data=df);
```



Plotting Residuals

Model residuals are the differences between the true values and the values predicted by the model. We can get them easily from the model results using the `resid` attribute.

In [17]:
```
results.resid
```

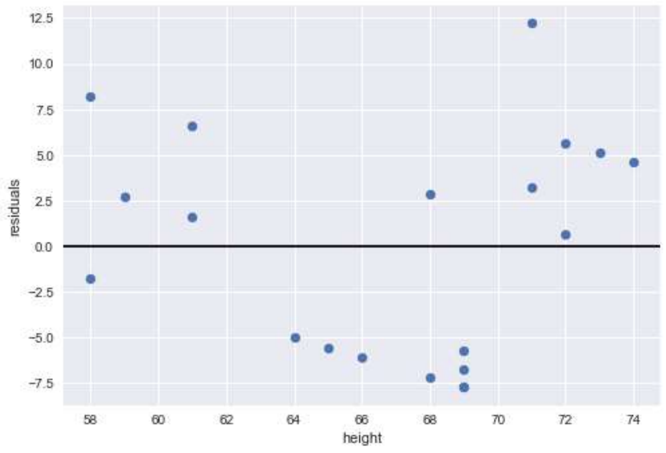
Out[17]:
```
0     -7.169872
1     12.213070
2      6.603263
3     -7.708891
4      3.213070
5      8.220320
6      0.674051
7      5.135032
8     -1.779680
9      4.596012
10     1.603263
11     2.681301
12    -5.708891
13     2.830128
14    -5.013795
15    -7.708891
16     5.674051
17    -6.091834
18    -5.552814
19    -6.708891
dtype: float64
```

As you can see, model residuals can be negative or positive depending on whether the model guessed too high or too low of a value.

The most typical way to plot the model residuals is like this:

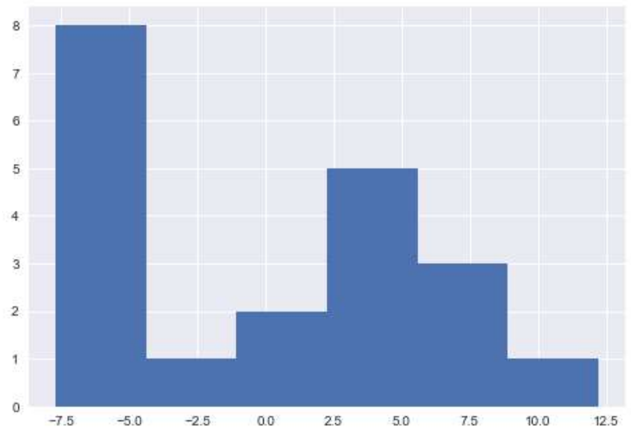
In [18]:
```
fig, ax = plt.subplots()

ax.scatter(df["height"], results.resid)
ax.axhline(y=0, color="black")
ax.set_xlabel("height")
ax.set_ylabel("residuals");
```



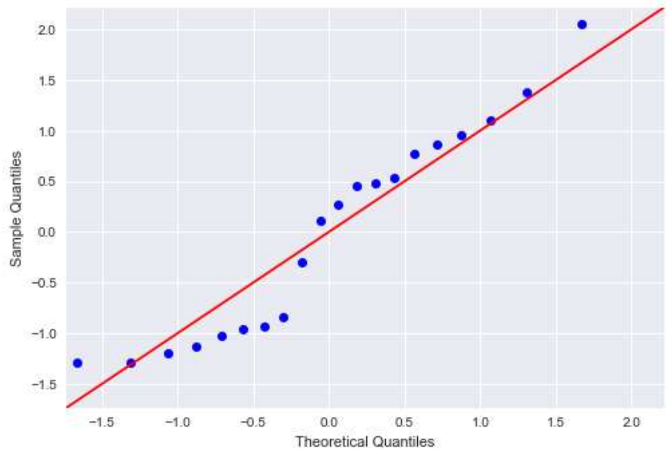
We also might be interested in the distribution of the residuals, which we could plot using a histogram:

In [19]:
```
plt.hist(results.resid, bins="auto");
```



Another tool we might use to visualize the distribution of the residuals is called a Q-Q plot. It compares the quantiles of the residuals to the quantiles of a theoretical normal distribution. The farther from the line that the data points appear, the farther from a normal distribution they are.

In [20]:
```
sm.graphics.qqplot(results.resid, dist=stats.norm, line='45', fit=True)
plt.show()
```



Why Visualize Residuals?

The main purpose of visualizing the residuals is to help you better understand where your model is performing well, and where it is performing poorly. For example, some models perform better on smaller values than larger values.

You might use this information to improve the model, or simply to communicate transparently about its strengths and weaknesses. These plots will also become relevant when investigating whether the model assumptions are being met.

R-Style Regression Formulas

We have been using the primary (`api`) interface to StatsModels. There is also an interface (`formula.api`) that uses R-style formulas. R-style meaning that the formulas for the models are written in the same style as they would be written in the programming language R (https://en.wikipedia.org/wiki/R_(programming_language)).

Unlike Python, R has built-in functionality for linear modeling, and so you will find that some foundational statistics and data science resources and papers used R rather than Python. StatsModels includes this R-style interface to be more convenient for people who learned R first. You can read more about it here (https://www.statsmodels.org/stable/example_formulas.html).

Below we provide an example of the same model above, except developed using an R-style formula.

In [21]:
```
import statsmodels.formula.api as smf

rstyle_model = smf.ols(formula="weight ~ height", data=df)
rstyle_results = rstyle_model.fit()
print(rstyle_results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                 weight   R-squared:                       0.955
Model:                            OLS   Adj. R-squared:                  0.953
Method:                 Least Squares   F-statistic:                     384.8
Date:                Fri, 06 May 2022   Prob (F-statistic):           1.35e-13
Time:                        15:24:20   Log-Likelihood:                -64.112
No. Observations:                  20   AIC:                             132.2
Df Residuals:                      18   BIC:                             134.2
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -204.4834     18.929    -10.802      0.000    -244.252    -164.714
height         5.5390      0.282     19.616      0.000       4.946       6.132
==============================================================================
Omnibus:                        2.588   Durbin-Watson:                   2.053
Prob(Omnibus):                  0.274   Jarque-Bera (JB):                1.245
Skew:                           0.202   Prob(JB):                        0.537
Kurtosis:                       1.846   Cond. No.                         902.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

The only difference in the results between using the R-style formula `ols` and the `OLS` we used previously is that one of the parameters is called `Intercept` rather than `const`. This is because `ols` automatically adds a constant to the `X` values, whereas with `OLS` you need to use `sm.add_constant`. Both forms result in an intercept term, they just have different names.

In general, we recommend using the `OLS` interface rather than the R-style formula interface because building the formula can get increasingly complicated as we move from simple linear regression to multiple regression. But you may see examples using either, so it's useful to be able to interpret both forms.

Summary

Now you know how to run a simple linear regression in StatsModels. After creating and fitting the model, you used attributes of the results object as well as the model summary to evaluate the model performance and interpret the values of its coefficients. You also saw some examples of visualizations to represent model performance, including some visualization tools directly from StatsModels.