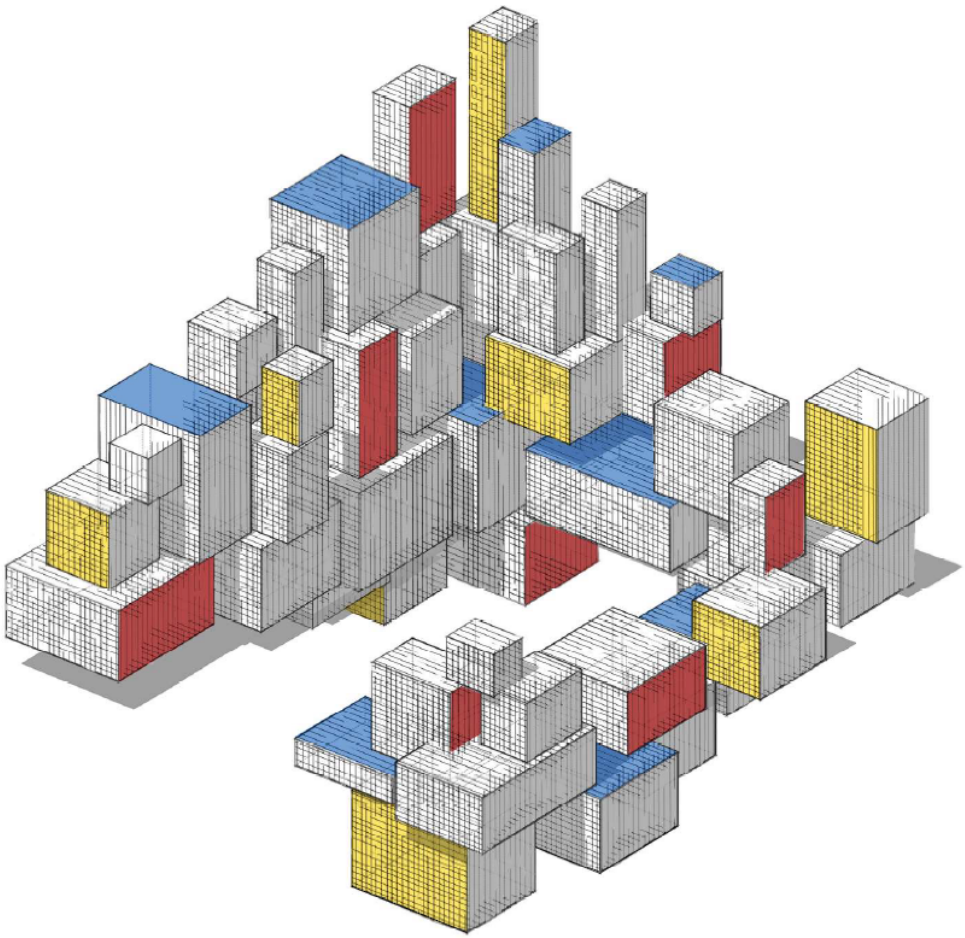
Practical Regression



Purpose

In this lecture we discuss:

- Strategies for nonlinear regression:
  - Extention of the feature space
  - Transformation of the data
  - Direct optimization of the training loss
- Linear models in Python

Nonlinear Regression Models

We discuss some strategies for handling general prediction functions $g(\boldsymbol{x} \mid \boldsymbol{\beta})$, where the functional form is known up to an unknown parameter vector $\boldsymbol{\beta}$. So the regression model becomes

$$Y_i = g(\boldsymbol{x}_i \mid \boldsymbol{\beta}) + \varepsilon_i, \quad i = 1, \ldots, n, \tag{1}$$

where $\varepsilon_1, \ldots, \varepsilon_n$ are independent with expectation 0 and unknown variance σ^2. The model can be further specified by assuming that the error terms have a normal distribution.

Table: Common nonlinear prediction functions for one-dimensional data.

Name	$g(x \mid \boldsymbol{\beta})$	$\boldsymbol{\beta}$
Exponential	$a\,\mathrm{e}^{bx}$	a, b
Power law	$a\,x^b$	a, b
Logistic	$(1 + \mathrm{e}^{a+bx})^{-1}$	a, b
Weibull	$1 - \exp(-x^b/a)$	a, b
Polynomial	$\sum_{k=0}^{p-1} \beta_k x^k$	$p, \{\beta_k\}_{k=0}^{p-1}$

Feature Space Extention

The first strategy for performing regression with nonlinear prediction functions is to extend the feature space to obtain a simpler (ideally linear) prediction function in the extended feature space.

In the running polynomial regression example, we extended the original feature u to the feature vector $\boldsymbol{x} = [1, u, u^2, \ldots, u^{p-1}]^\top$, yielding a linear prediction function.

In a similar way, the right-hand side of the polynomial 2-D prediction function

$$g(\boldsymbol{x} \mid \boldsymbol{\beta}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2$$

can be viewed as a linear function of the extended feature vector $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]^\top$.

In general, such a function $\boldsymbol{\phi}$ is called a feature map.

Transformation

The second strategy is to transform the response variable y and possibly also the explanatory variable $\boldsymbol{x}$ such that the transformed variables $\widetilde{y}, \widetilde{\boldsymbol{x}}$ are related in a simpler (ideally linear) way.

For example, for the exponential prediction function $y = a\,\mathrm{e}^{-bx}$, we have $\ln y = \ln a - bx$, which is a linear relation between $\ln y$ and $[1, x]^\top$.

Example: Chlorine

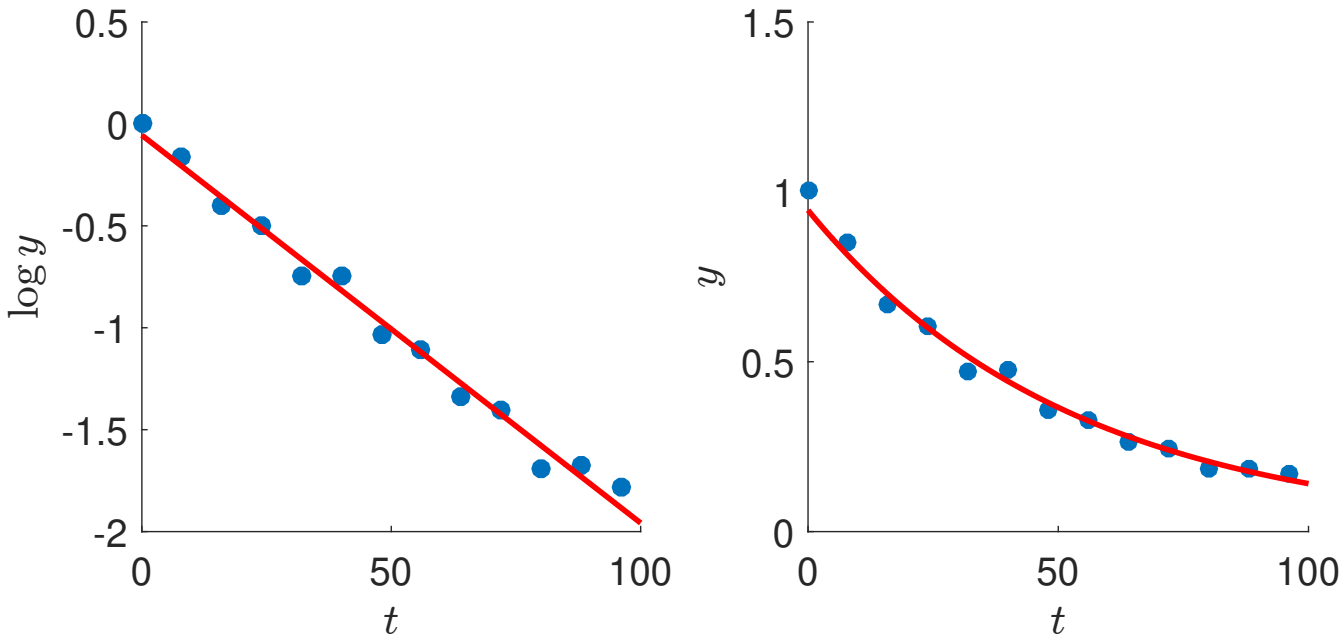
The free chlorine concentration (in mg per liter) in a swimming pool is recorded every 8 hours for 4 days. A simple chemistry-based model for the chlorine concentration y as a function of time t is $y = a\,\mathrm{e}^{-b\,t}$, where a is the initial concentration and $b > 0$ is the reaction rate.

Table: Chlorine concentration (in mg/L) as a function of time (hours).

Hours	Concentration	Hours	Concentration
0	1.0056	56	0.3293
8	0.8497	64	0.2617
16	0.6682	72	0.2460
24	0.6056	80	0.1839
32	0.4735	88	0.1867
40	0.4745	96	0.1688
48	0.3563		

Example: Chlorine

A log transformation of y will result in a *linear* relationship between $\ln y$ and the feature vector $[1, t]^\top$.



The intercept and slope of the regression line are $\beta_0 = -0.0555$ and $\beta_1 = -0.0190$. The right panel shows the original data with the fitted curve $y = \widehat{a}\,\mathrm{e}^{-\widehat{b}t}$, where $\widehat{a} = \exp(\widehat{\beta}_0) = 0.9461$ and $\widehat{b} = -\widehat{\beta}_1 = 0.0190$.

Direct Training Loss Minimization

The third strategy for regression with nonlinear prediction functions is to directly minimize, by any means possible, the (squared-error) training loss

$$\ell_\tau(g(\cdot \mid \boldsymbol{\beta})) = \frac{1}{n} \sum_{i=1}^{n} (y_i - g(\boldsymbol{x}_i \mid \boldsymbol{\beta}))^2 \tag{2}$$

for a learner $g_\tau(\boldsymbol{x})$ from a given training set τ.

Example: Hougen Function

The reaction rate y of a certain chemical reaction is posited to depend on three input variables: quantities of hydrogen x_1, n-pentane x_2, and isopentane x_3. The functional relationship is given by the *Hougen* function:

$$y = \frac{\beta_1 x_2 - x_3/\beta_5}{1 + \beta_2 x_1 + \beta_3 x_2 + \beta_4 x_3},$$

where $\beta_1, \ldots, \beta_5$ are the unknown parameters. The objective is to estimate the model parameters $\{\beta_i\}$ from the data below.

x_1	x_2	x_3	y
470	300	10	8.55
285	80	10	3.79
470	300	120	4.82
470	80	120	0.02
470	80	10	2.75
100	190	10	14.39
100	80	65	2.54

x_1	x_2	x_3	y
470	190	65	4.35
100	300	54	13.00
100	300	120	8.50
100	80	120	0.05
285	300	10	11.32
285	190	120	3.13

Example: Hougen Function

The estimation is carried out via the least-squares method. The objective function to minimize is thus

$$\ell_\tau(g(\cdot \mid \boldsymbol{\beta})) = \frac{1}{13} \sum_{i=1}^{13} \left(y_i - \frac{\beta_1 x_{i2} - x_{i3}/\beta_5}{1 + \beta_2 x_{i1} + \beta_3 x_{i2} + \beta_4 x_{i3}} \right)^2, \qquad (3)$$

where the $\{y_i\}$ and $\{x_{ij}\}$ are given in the previous table.

This is a highly nonlinear optimization problem, for which standard nonlinear least-squares methods do not work well.

Using the CE method, we found the minimal value 0.02299 for the objective function, which is attained at

$$\widehat{\boldsymbol{\beta}} = [1.2526,\ 0.0628,\ 0.0400,\ 0.1124,\ 1.1914]^\top.$$

Linear Models in Python

We can define and analyze linear models using Python and the data science module **statsmodels**, assuming the following imports:

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

In **statsmodels**, ordinary least-squares linear models are specified via the function **ols** (short for ordinary least-squares). The main argument of this function is a formula of the form

$$\texttt{y} \sim \texttt{x1} + \texttt{x2} + \cdots + \texttt{xd}, \tag{4}$$

where `y` is the name of the response variable and `x1`, . . . , `xd` are the names of the explanatory variables.

Quantitative Features

If all variables are *quantitative*, this describes the linear model

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_d x_{id} + \varepsilon_i, \quad i = 1, \ldots, n, \qquad (5)$$

where x_{ij} is the j-th explanatory variable for the i-th observation and the errors ε_i are independent normal random variables such that $\mathbb{E}\varepsilon_i = 0$ and $\mathbb{V}\mathrm{ar}\,\varepsilon_i = \sigma^2$.

Or, in matrix form: $\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, with

$$\boldsymbol{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nd} \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_d \end{bmatrix}, \text{ and } \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}.$$

Quantitative and Qualitative Features

Python treats quantitative and qualitative (factors) features in a formula *differently*, introducing indicator variables for qualitative features in the model matrix.

For any linear model, the model matrix can be retrieved via the construction:

```
model_matrix = pd.DataFrame(model.exog,columns=model.exog_names)
```

Let us look at some examples of linear models.

Example Models

In this model the variables `x1` and `x2` are both considered (by Python) to be quantitative.

```
myData = pd.DataFrame({'y' : [10,9,4,2,4,9],
    'x1' : [7.4,1.2,3.1,4.8,2.8,6.5],
    'x2' : [1,1,2,2,3,3]})
mod = ols("y~x1+x2", data=myData)
mod_matrix = pd.DataFrame(mod.exog,columns=mod.exog_names)
print(mod_matrix)
```

```
   Intercept   x1   x2
0        1.0  7.4  1.0
1        1.0  1.2  1.0
2        1.0  3.1  2.0
3        1.0  4.8  2.0
4        1.0  2.8  3.0
5        1.0  6.5  3.0
```

Example Models

Suppose that `x2` is actually qualitative; e.g., it represents a color, and the levels 1, 2, and 3 stand for red, blue, and green. We can account for such a categorical variable by using the **`astype`** method to redefine the data type.

```
myData['x2'] = myData['x2'].astype('category')
```

Alternatively, a categorical variable can be specified in the model formula by wrapping it with `C()`.

```
mod2 = ols("y~x1+C(x2)", data=myData)
mod2_matrix = pd.DataFrame(mod2.exog,columns=mod2.exog_names)
print(mod2_matrix)
```

```
   Intercept  C(x2)[T.2]  C(x2)[T.3]   x1
0        1.0         0.0         0.0  7.4
1        1.0         0.0         0.0  1.2
2        1.0         1.0         0.0  3.1
3        1.0         1.0         0.0  4.8
4        1.0         0.0         1.0  2.8
5        1.0         0.0         1.0  6.5
```

Quantitative and Qualitative Features

Thus, if a `statsmodels` formula of the form (4) contains factor (qualitative) variables, the model is no longer of the form (5), but contains indicator variables for each level of the factor variable, except the first level.
For the previous example, the corresponding linear model is

$$Y_i = \beta_0 + \beta_1 x_{i1} + \alpha_2 \mathbb{I}\{x_{i2} = 2\} + \alpha_3 \mathbb{I}\{x_{i2} = 3\} + \varepsilon_i, \quad i = 1, \ldots, 6,$$

where we have used parameters α_2 and α_3 to correspond to the indicator features of the qualitative variable.

The parameter α_2 describes how much the response is expected to change if the factor x_2 switches from level 1 to 2.

A similar interpretation holds for α_3. Such parameters can thus be viewed as incremental effects.

Interaction

It is also possible to model interaction between two variables. For two continuous variables, this simply adds the products of the original features to the model matrix. Adding interaction terms in Python is achieved by replacing "+" in the formula with "*", as the following example illustrates.

```
mod3 = ols("y~x1*C(x2)", data=myData)
mod3_matrix = pd.DataFrame(mod3.exog,columns=mod3.exog_names)
print(mod3_matrix)
```

```
   Intercept  C(x2)[T.2]  C(x2)[T.3]   x1  x1:C(x2)[T.2]  x1:C(x2)[T.3]
0        1.0         0.0         0.0  7.4            0.0           0.0
1        1.0         0.0         0.0  1.2            0.0           0.0
2        1.0         1.0         0.0  3.1            3.1           0.0
3        1.0         1.0         0.0  4.8            4.8           0.0
4        1.0         0.0         1.0  2.8            0.0           2.8
5        1.0         0.0         1.0  6.5            0.0           6.5
```

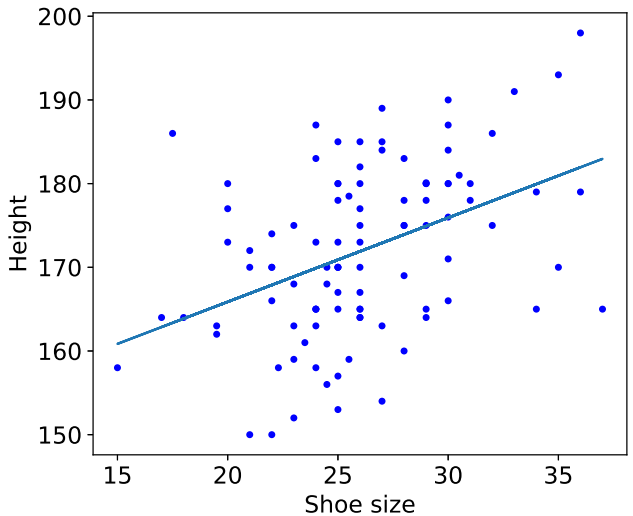
Example Analysis

The student survey data set `survey.csv` contains measurements such as height, weight, sex, etc., from a survey conducted among $n = 100$ university students.

We wish to investigate the relation between the shoe size (explanatory variable) and the height (response variable).

First, we load the data and draw a scatterplot.

```
survey = pd.read_csv('survey.csv')
plt.scatter(survey.shoe, survey.height)
plt.xlabel("Shoe size")
plt.ylabel("Height")
```



Example Analysis

We analyze the data through the simple linear regression model $Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, i = 1, \ldots, n.$

```
model = ols("height~shoe", data=survey) # define the model
fit = model.fit()  #fit the model defined above
b0, b1 = fit.params
print(fit.params)
```

```
Intercept    145.777570
shoe           1.004803
dtype: float64
```

We thus find the least-squares estimates $\widehat{\beta}_0 = 145.778$ and $\widehat{\beta}_1 = 1.005$.
We can add the estimated regression line to the scatterplot as follows:

```
plt.plot(survey.shoe, b0 + b1*survey.shoe)
plt.scatter(survey.shoe, survey.height)
plt.xlabel("Shoe size")
plt.ylabel("Height")
```

Example Analysis

A summary of the results can be obtained with the method **summary**.

```
print(fit.summary())
```

```
Dep. Variable:                  height   R-squared:                       0.178
Model:                             OLS   Adj. R-squared:                  0.170
Method:                  Least Squares   F-statistic:                     21.28
No. Observations:                  100   Prob (F-statistic):           1.20e-05
Df Residuals:                       98   Log-Likelihood:                -363.88
Df Model:                            1   AIC:                             731.8
Covariance Type:            nonrobust   BIC:                             737.0
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    145.7776      5.763     25.296      0.000     134.341     157.214
shoe           1.0048      0.218      4.613      0.000       0.573       1.437
==============================================================================
Omnibus:                        1.958   Durbin-Watson:                   1.772
Prob(Omnibus):                  0.376   Jarque-Bera (JB):                1.459
Skew:                          -0.072   Prob(JB):                        0.482
Kurtosis:                       2.426   Cond. No.                         164.
```

Output

The main output items are the following:

- `coef:` Estimates of the parameters of the regression line.
- `std error:` Standard deviations of the estimators of the regression line; i.e., the square roots of the variances of the $\{\widehat{\beta}_i\}$.
- `t:` Realization of Student's test statistics associated with the hypotheses $H_0 : \beta_i = 0$ and $H_1 : \beta_i \neq 0$, $i = 0, 1$.
- `P>|t|`: P-value of Student's test (two-sided test).
- `[0.025 0.975]`: 95% confidence intervals for the parameters.
- `R-Squared:` Coefficient of determination R^2 (percentage of variation explained by the regression).
- `F-statistic:` Realization of the F test statistic associated with testing the full model against the default model. The associated degrees of freedom (`Df Model` = 1 and `Df Residuals` = $n - 2$) are given, as is the P-value: `Prob (F-statistic)`.
- `AIC:` The Akaike information criterion, i.e., minus two times the log-likelihood plus two times the number of model parameters (which is 3 here).

Example Analysis

The `fit` object has many attributes, which can be obtained via:

```
dir(fit)
```

Access the values via the dot construction. For example, the following extracts the P-value for the slope.

```
fit.pvalues[1]
```

```
 1.1994e-05
```

The results show strong evidence that the slope of the regression line is not zero, as the P-value is very small ($1.2 \cdot 10^{-5}$).

The estimate of the slope indicates that the difference between the average height of students whose shoe size is different by one cm is 1.0048 cm.

Only 17.84% of the height variability is explained by shoe size.

Example Analysis

We continue the student survey example of the previous section, but now add an extra variable, and also consider an Analysis of Variance of the model. Instead of "explaining" the student height via their shoe size, we include `weight` as an explanatory variable. The corresponding **`ols`** formula for this model is

$$\texttt{height} \sim \texttt{shoe} + \texttt{weight},$$

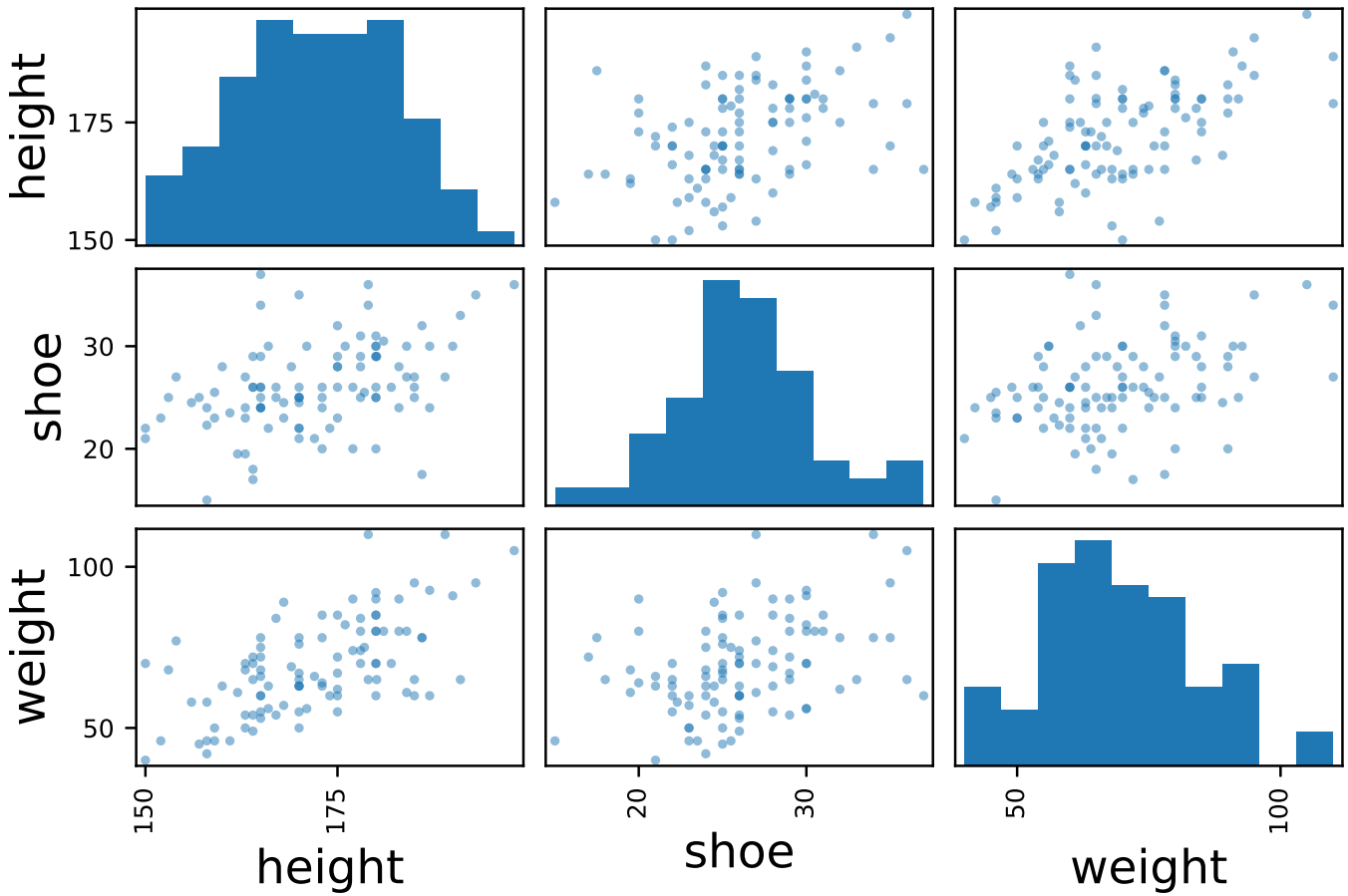
meaning that each random height, denoted by `Height`, satisfies

$$\texttt{Height} = \beta_0 + \beta_1 \texttt{shoe} + \beta_2 \texttt{weight} + \varepsilon,$$

where ε is a normally distributed error term with mean 0 and variance σ^2. Thus, the model has 4 parameters. Before analyzing the model we present a scatterplot of all pairs of variables, using **`scatter_matrix`**.

Example Analysis

```
model = ols("height~shoe+weight", data=survey)
fit = model.fit()
axes = pd.plotting.scatter_matrix(survey[['height','shoe','weight']])
```



Example Analysis

As for the simple linear regression model in the previous section, we can analyze the model using the **summary** method (below we have omitted some output):

```
fit.summary()
```

```
Dep. Variable:                  height   R-squared:                       0.430
Model:                             OLS   Adj. R-squared:                  0.418
Method:                  Least Squares   F-statistic:                     36.61
No. Observations:                  100   Prob (F-statistic):           1.43e-12
Df Residuals:                       97   Log-Likelihood:                -345.58
Df Model:                            2   AIC:                             697.2
                                         BIC:                             705.0
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    132.2677      5.247     25.207      0.000     121.853     142.682
shoe           0.5304      0.196      2.703      0.008       0.141       0.920
weight         0.3744      0.057      6.546      0.000       0.261       0.488
```

Example Analysis

The `F-statistic` is used to test whether the full model (here with two explanatory variables) is better at "explaining" the height than the default model. Given the result of this test (P-value = $1.429 \cdot 10^{-12}$), we can conclude that at least one of the explanatory variables is associated with height. The individual Student tests indicate that:

- shoe size is linearly associated with student height, after adjusting for weight, with P-value 0.0081. At the same weight, an increase of one cm in shoe size corresponds to an increase of 0.53 cm in average student height;
- weight is linearly associated with student height, after adjusting for shoe size (the P-value is actually $2.82 \cdot 10^{-09}$; the reported value of 0.000 should be read as "less than 0.001"). At the same shoe size, an increase of one kg in weight corresponds to an increase of 0.3744 cm in average student height.

Analysis of Variance

Further understanding is extracted from the model by conducting an analysis of variance. The standard `statsmodels` function is `anova_lm`. The input to this function is the fit object (obtained from `model.fit()`) and the output is a `DataFrame` object.

```
table = sm.stats.anova_lm(fit)
print(table)
```

```
              df  sum_sq       mean_sq       F          PR(>F)
shoe         1.0  1840.467359  1840.467359  30.371310  2.938651e-07
weight       1.0  2596.275747  2596.275747  42.843626  2.816065e-09
Residual    97.0  5878.091294    60.598879        NaN           NaN
```

The meaning of the columns is as follows.

- `df` : The degrees of freedom of the variables, according to the sum of squares decomposition. As both `shoe` and `weight` are quantitative variables, their degrees of freedom are both 1. The degrees of freedom for the residuals is $n - p = 100 - 3 = 97$.

Analysis of Variance

- `sum_sq`: The sum of squares according to the sum of squares decomposition. The total sum of squares is the sum of all the entries in this column. The residual error in the model that cannot be explained by the variables is RSS ≈ 5878.
- `mean_sq`: The sum of squares divided by their degrees of freedom. Note that the residual mean square error RSE $= \text{RSS}/(n-p) = 60.6$ is an unbiased estimate of the model variance σ^2.
- `F`: These are the outcomes of the F test statistic.
- `PR(>F)`: These are the P-values corresponding to the test statistic in the preceding column and are computed using an `F` distribution whose degrees of freedom are given in the `df` column.

Analysis of Variance

The ANOVA table indicates that the `shoe` variable explains a reasonable amount of the variation in the model, as evidenced by a sum of squares contribution of 1840 out of 1840 + 2596 + 5878 = 10314 and a very small P-value.

After `shoe` is included in the model, it turns out that the `weight` variable explains even more of the remaining variability, with an even smaller P-value.

The remaining sum of squares (5878) is 57% of the total sum of squares, yielding a 43% reduction, in accordance with the R^2 value reported in the summary for the `ols` method.

Analysis of Variance

The *order* in which the ANOVA is conducted is important. To illustrate this, consider the output of the following commands.

```
model = ols("height~weight+shoe", data=survey)
fit = model.fit()
table = sm.stats.anova_lm(fit)
print(table)
```

	df	sum_sq	mean_sq	F	PR(>F)
weight	1.0	3993.860167	3993.860167	65.906502	1.503553e-12
shoe	1.0	442.882938	442.882938	7.308434	8.104688e-03
Residual	97.0	5878.091294	60.598879	NaN	NaN

We see that `weight` as a single model variable explains much more of the variability than `shoe` did.

If we now also include `shoe`, we only obtain a small (but according to the P-value still significant) reduction in the model variability.

Confidence and Prediction Intervals

In **statsmodels** a method for computing confidence or prediction intervals from a dictionary of explanatory variables is **get_prediction**.

Suppose we wish to predict the height of a person with shoe size 30 cm and weight 75 kg.

```
x = {'shoe': [30.0], 'weight': [75.0]}  # new input (dictionary)
pred = fit.get_prediction(x)
pred.summary_frame(alpha=0.05).unstack()
```

```
mean            0    176.261722    # predicted value
mean_se         0      1.054015
mean_ci_lower   0    174.169795    # lower bound for CI
mean_ci_upper   0    178.353650    # upper bound for CI
obs_ci_lower    0    160.670610    # lower bound for PI
obs_ci_upper    0    191.852835    # upper bound for PI
dtype: float64
```

Note that the new explanatory variable is entered as a dictionary.

Model Validation

Various plots of the residuals can be used to inspect whether the assumptions on the errors $\{\varepsilon_i\}$ are satisfied.

```
plt.plot(fit.fittedvalues,fit.resid,'.')
plt.xlabel("fitted values")
plt.ylabel("residuals")
sm.qqplot(fit.resid)
```

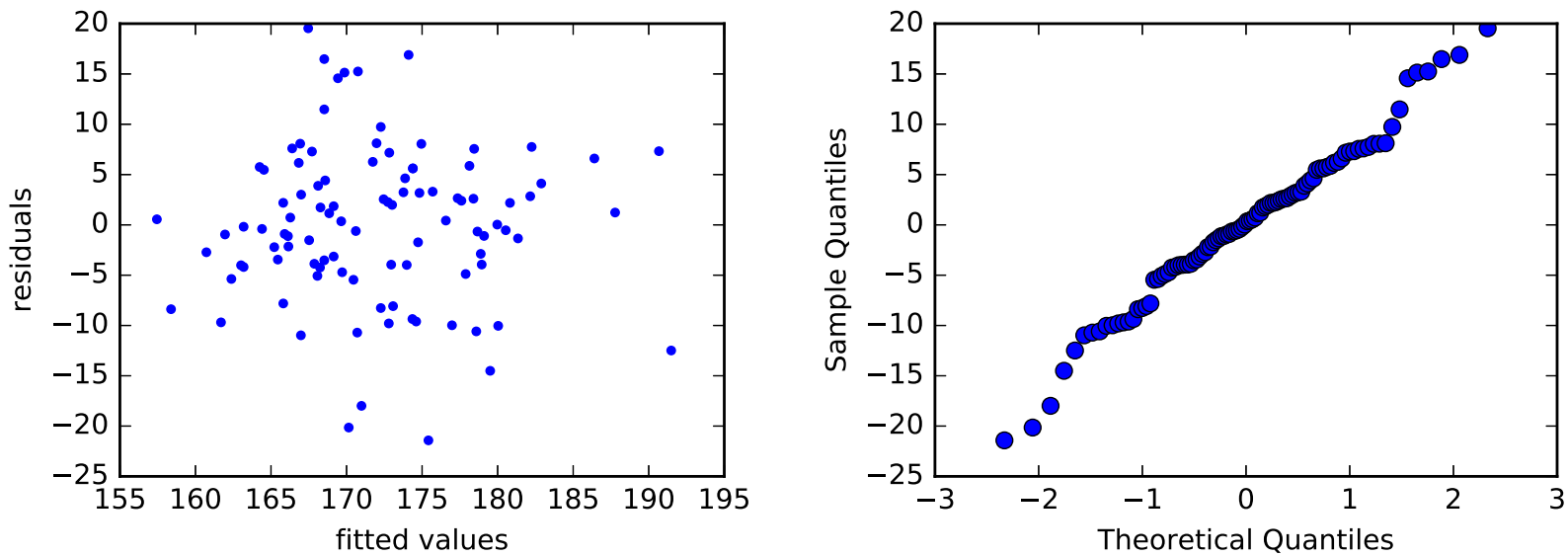


Figure: Left: residuals against fitted values. Right: a qq plot of the residuals. Both support Neither shows clear evidence against the model assumptions of constant variance and normality.

Model Validation

The scatterplot of the residuals $\{e_i\}$ against the fitted values $\widehat{y}_i$ behave approximately as iid normal random variables for each of the fitted values, with a constant variance.

The residuals are fairly evenly spread and symmetrical about the $y = 0$ line (not shown). We see no strong aberrant structure in this plot.

The quantile–quantile plot is a useful way to check for normality of the error terms, by plotting the sample quantiles of the residuals against the theoretical quantiles of the standard normal distribution. Under the model assumptions, the points should lie approximately on a straight line.

For the current case there does not seem to be an extreme departure from normality.