4 Fitting a linear regression

This chapter covers

- Model fitting
- Model evaluation
- Model assumption tests
- Data exploration, testing for normality, and detecting outliers

Linear regression is a supervised learning method, meaning it uses labeled data—where the input features and corresponding outputs are known for a subset of data—to predict a quantitative response from one or more independent variables. This model is then applied to make predictions on new, unknown data. Although linear regression might now lack the spark of random forests (see chapter 5) and other more contemporary methods, it's still an "implement" at or near the top of every data scientist's toolbox. Furthermore, linear regression is a foundational model that is easy both to understand and to implement, with virtually endless use cases. Here are just a few examples:

- Marketing organizations predicting sales revenues based on advertising expenditures across multiple channels

- Buyers and lenders predicting future home prices based on size and square footage, number of bedrooms and bathrooms, age and condition of the property, architectural design, and lot size
- University admissions officers predicting student performance based on high school GPAs and standardized test scores
- Retailers predicting product demand based on historical sales figures

Any relationship between independent and dependent variables—that is, between predictors that explain change and the outcomes they affect—that shows a linear pattern in the given feature space can be modeled with linear regression. Although more complex methods can transform data to model nonlinear relationships, linear regression is designed to capture relationships that align well with a straight line through the data. When fitting a linear regression, we are attempting, at minimum, to answer the following questions:

- *Is there a meaningful relationship between independent variables, also known as explanatory variables or predictors, and a dependent variable, known as the target or response variable?* Linear regression models the relationship between independent and dependent variables under the assumption that the relationship is linear, allowing us to test for statistical significance in the strength of this association.
- *What is the nature of the relationship?* Linear regression reveals if the relationship between variables is either positive or negative—positive if they change in the same direction, negative if they change in opposite directions.
- *How strong is the relationship?* Linear regression returns measures from which we can quantify the degree of association between variables, although these measures are most accurate when the independent variables are truly independent—a condition that is not always satisfied in real-world data.
- *Can we estimate changes in a dependent variable from variances in the independent variables?* Linear regression provides coefficients that represent the estimated effect of each predictor on the dependent variable, allowing us to model expected outcomes under the assumption that the observed relationships are consistent.
- *When fitting a multiple linear regression, where a response variable is modeled against two or more predictors, which predictors have a statistically significant influence on changes in the response variable?* Linear regression helps identify which predictors are statistically significant, although this significance can be affected by factors like collinearity and other model assumptions.
- *How well does the model fit the data?* Linear regression estimates the goodness of fit, indicating how well the model explains and predicts variances in the response variable under the assumption of linearity.
- *Have model assumptions been met?* Model integrity (the internal soundness of the model) and reliability (its consistency in returning valid results) are predicated on several assumptions being true. Tests and other checks are available to establish whether or not these assumptions have been satisfied.

We intend to answer all these questions and more. Our path forward will be broadly divided into two parts. Section 4.1 focuses on the theoretical: understanding the relationship between variables, simple versus multiple regressions, conditions that must prevail for linear regression to best fit the data, coefficients, constants, regression lines, and residuals. Section 4.2 is way more practical: we'll import a real data set and demonstrate how to fit a simple linear regression in Python, evaluate the results, and test model assumptions.

4.1 Primer on linear regression

Linear regression, one more time, is a popular and practical supervised learning method for predictions—when the response variable is numeric. You should never fit a linear regression when attempting to solve a classification problem, which involves predicting categorical outcomes such as labels or group membership. Linear regression is a method used to model the relationship between a numeric dependent variable and one or more independent variables, which may or may not also be numeric. It assumes a linear relationship between variables. Our purpose here is to get you thoroughly grounded in all things linear regression before fitting a model to a real data set.

4.1.1 Linear equation

The purpose of linear regression is to measure the correlation between independent and dependent variables by finding the line of best fit that models the relationship under the assumption of linearity. The so-called *regression line*, which represents the predicted values of the dependent variable, is determined by estimating the coefficients and the intercept in a way that minimizes the differences between it and the observed data.

The equation for a simple linear regression, where the dependent variable is regressed against just one independent variable, or predictor, is typically represented as follows:

$$y = \beta_0 + \beta_1 x + \epsilon$$

This formulation reflects the underlying population model, where

- y is the observed dependent variable (also called the *response* or *target variable*).
- x is the independent variable (also known as the *predictor* or *explanatory variable*).
- β_0 is the intercept, or the value of the response variable when x equals 0.
- β_1 is the slope or the change in the response variable for every one-unit change in x.
- ϵ is the error term, capturing the deviation of the observed values from the regression line.

When making predictions, the model is often expressed in terms of the predicted value, $\hat{y}$ (or *yhat*), omitting the error term:

$$\hat{y} = \beta_0 + \beta_1 x$$

Here, $\hat{y}$ distinguishes the predicted response from the actual observed response. The error term, ϵ, is still present conceptually but not explicitly included in the predictive formula.

The equation for a multiple linear regression, where the dependent variable is regressed against, let's say, three predictors, is expressed this way:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \epsilon$$

Or, for prediction purposes,

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

where

- x_1, x_2, and x_3 represent the three predictors.
- β_1, β_2, and β_3 are the regression coefficients (slopes) associated with each independent variable.

Let's explain further with the aid of an illustration: see figure 4.1.

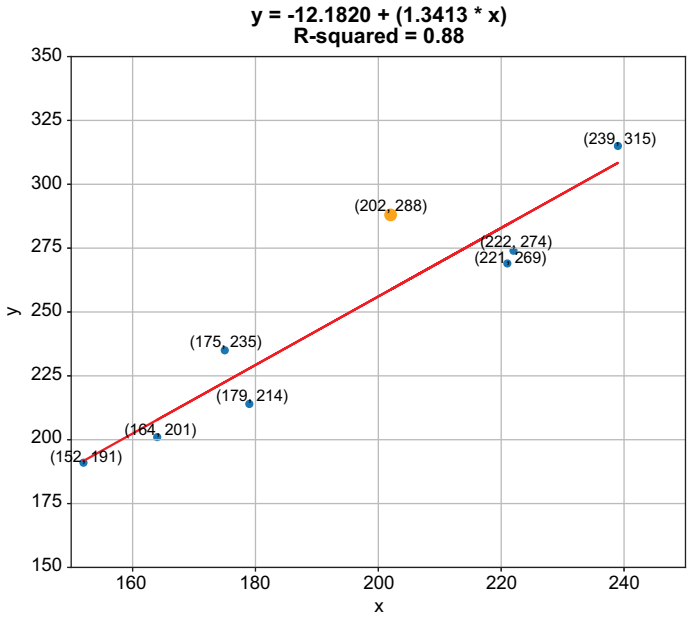

Figure 4.1 A scatter plot that displays eight data points, their respective *x* and *y* coordinates, and a regression line. The data points represent the observed data (for instance, when `x` equals `202`, `y` equals `288`, which has been highlighted and magnified). The regression line ties back to a simple linear regression that was fit on the data, where a response variable called `y`, which runs along the *y* axis, was regressed against a predictor called `x`, which runs along the *x* axis. It represents the linear equation at the top that can be derived from the model output, which is to say, it is also a representation of the predictions for `y` given `x`. R-squared is one of several metrics contained in the model output; it represents the percentage of variance in the response variable that can be explained by changes in the predictor.

This scatter plot was created with Matplotlib. A scatter plot visualizes the relationship between a pair of numeric variables on a two-dimensional plane, where each point represents a single observation in the data. Scatter plots have many uses, including

- Identifying patterns or trends in the relationship between two variables
- Assessing the strength and direction of the relationship between variables
- Detecting outliers or other potentially anomalous observations in the data
- Evaluating the fit of linear modeling

We'll demonstrate later how to create a similar plot using a series of methods from the Matplotlib library, but in the meantime, let's review what's shown in figure 4.1.

The horizontal axis (or *x* axis) represents the independent variable, labeled `x`, and the vertical axis (or *y* axis) represents the dependent variable, labeled `y`. Although it's conventional to place the predictor on the *x* axis and the response on the *y* axis, the linear relationship itself is symmetric and does not inherently depend on this orientation.

Each of the eight dots, or data points, corresponds to a pair of values from the same row in an 8×2 data frame. Consider, for example, the one data point we've intentionally magnified: the data contains one record where `x` equals `202` and `y` equals `288`. The points, therefore, represent the observed values from the data. A printout of the data source is provided here for cross-referencing purposes:

```
     x    y
0  152  191
1  175  235
2  179  214
3  222  274
4  202  288
5  221  269
6  239  315
7  164  201
```

The relationship between `x` and `y`—that is, between our independent and dependent variables—is positive, because as `x` increases, so does `y`. The relationship is negative when increases in the independent variable correspond to decreases in the dependent variable (see figure 4.2). Linear regression may be a good fit for the data either way, as long as the total difference between observed and predicted values (the residual error) is kept to a minimum. However, even if the residual error is low, a near-zero slope would suggest a weak or statistically insignificant relationship between the variables.

Alternatively, the relationship is considered neutral when changes in the independent variable appear to have little or no effect on the dependent variable (see figure 4.3). In such cases, linear regression may yield a best-fit line with a near-zero slope, indicating a weak or nonexistent relationship rather than a meaningful predictive model.

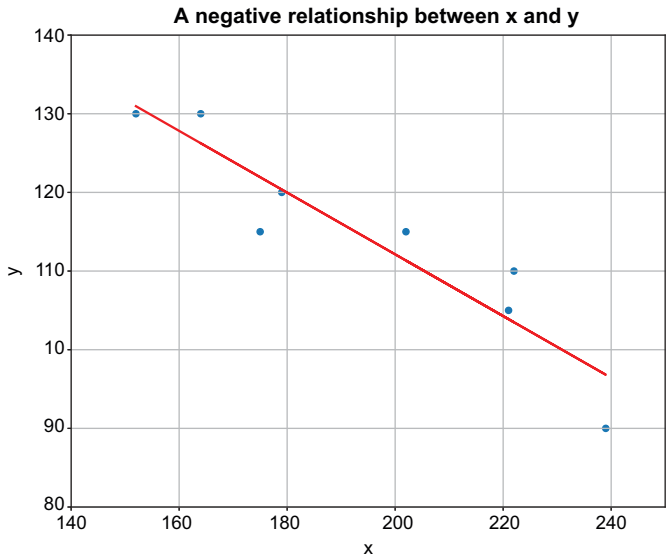

Figure 4.2 A scatter plot that shows a negative relationship between variables. The relationship is negative because the variables move in opposite directions: as the independent variable `x` increases, the dependent variable `y` decreases.

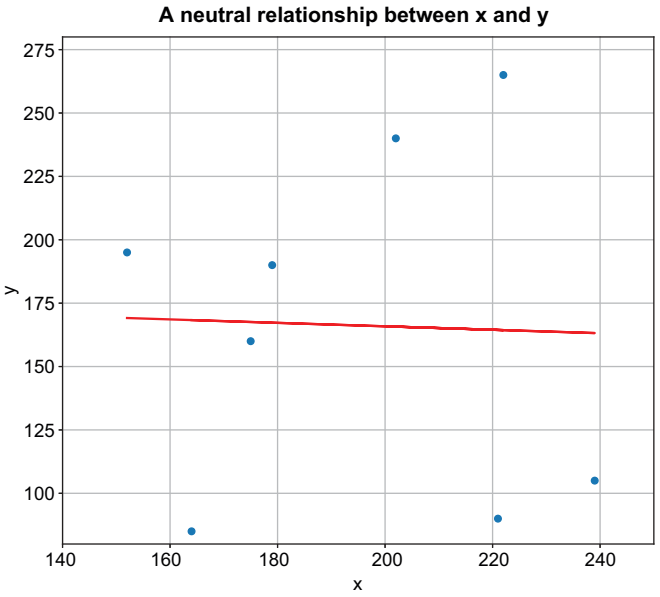

Figure 4.3 A scatter plot that shows a neutral relationship between variables. The relationship is neutral because changes in the independent variable `x` appear to have almost no effect on the dependent variable `y`.

4.1.2 Goodness of fit

The *regression line*, or *line of best fit*, is a straight line that represents the relationship between the independent variable `x` and the dependent variable `y` from a simple linear model. It is determined by estimating the values of the intercept and slope coefficients that best minimize the sum of squared residuals, which is a measure of the overall difference between the actual values of `y` and the predicted values of `y`. The lower the sum of squared residuals, the better the regression line fits the data, and vice versa. In multiple linear regression, where there are two or more predictors, the regression line becomes a hyperplane in a multidimensional space, but the concept of modeling the relationship between variables is unchanged.

Goodness of fit is most typically measured by a single statistic called R-squared. R-squared (or R^2), formally known as the coefficient of determination, is a statistical measure that indicates the proportion of variance in the dependent variable that is explained by the one predictor in a simple linear regression or by the two or more predictors in a multiple regression; it is used to assess the goodness of fit of the regression model to the observed data. Because it's a squared measure, R^2 will always equal some number between 0 and 1; the higher the coefficient of determination, the better the predictor(s) account for changes in the dependent variable, and vice versa.

When fitting a simple regression, R^2 represents the square of the correlation coefficient between the same dependent and independent variables (it's less straightforward when fitting a multiple regression). R^2 is a commonly used metric for assessing model fit, but it has important limitations that must be acknowledged. As a single-point measure, R^2 doesn't capture all aspects of model quality, and thorough model evaluation also requires checking the underlying data distribution and other diagnostic measures. Additionally, R^2 naturally increases when other predictors are added to a regression, even if those variables have little to no true relationship with the dependent variable; this can lead to overfitting. An overfitted model is overly tailored to the data it was trained on, capturing noise rather than general patterns. As a result, it may perform poorly when applied to new data, failing to generalize effectively.

Also, R^2 doesn't indicate the direction or causal relationship between variables; it simply tells you how well the model variables fit together. The direction must be inferred from other metrics or observed by plotting the data. We will talk much more about R^2 and other metrics when we fit a simple regression on real data.

4.1.3 Conditions for best fit

One more point: linear regression provides the best fit when the variables are normally distributed—forming a bell-shaped curve—and are free of outliers. Non-normal data should be transformed to normal distributions before fitting a linear regression. However, applying transformations to achieve normality changes the assumed relationship between dependent and independent variables, as the fitted model then describes associations on the transformed scale rather than the original scale. Transforming data can help meet the assumptions of a linear model, but it's essential to

exercise caution, as this approach may overlook other modeling options that could better capture complex, nonlinear relationships in the data.

A variable called `x` from a data frame named `df` can be normalized, depending on its original distribution, by applying one of the following transformation methods:

- Logarithmic transformation (requires `numpy`, a common Python library for scientific computing):

  ```
  df['x'] = np.log(df['x'])
  ```

  or

  ```
  df['x'] = np.log10(df['x'])
  ```

  Apply when the data spans several orders of magnitude or is right-skewed, as this transformation compresses the range and can normalize distributions.
- Reciprocal transformation:

  ```
  df['x'] = 1/df.x
  ```

  Use when dealing with data that is heavily right-skewed or when large values need to be compressed and small values expanded. Be cautious with zeros or negative values, as they require special handling.
- Square root transformation (also requires `numpy`):

  ```
  df['x'] = np.sqrt(df.x)
  ```

  Best applied to moderate right-skewed data or when the variance of the data increases with the mean. It's useful for stabilizing variance and making the data more normal.
- Exponential transformation:

  ```
  df['x'] = df.x**(1/1.2)
  ```

  This transformation can be applied when you need to reverse the effect of a power transformation or slightly reduce the range of variability in data. It's effective for moderately skewed data.

But the larger point to be discussed—and demonstrated—is the effect of outliers. *Outliers* are data points that differ significantly from other observations in the same set of data. (Incidentally, removing outliers is yet another method by which to normalize a data series.) One way of detecting outliers, although not necessarily the most accurate, is to plot the data in a scatter plot. Let's revisit our first scatter plot in figure 4.1, specifically the magnified data point where the `x` and `y` coordinates equal `202` and `288`, respectively.

For the sake of this demonstration, let's assume (or pretend) it qualifies as the one outlier in the data. After all, the distance between it and the fitted regression line is clearly greater than the distance between any other observed data point and the corresponding prediction for `y`. Because linear regression draws a straight line through

the data that minimizes the total distance between it and the observed data, this one outlier, which might actually be a measurement error or the representation of a rare or anomalous event, has undue influence on the model. Let's then remove that one record from the data, rerun our regression, and plot the results (see figure 4.4).

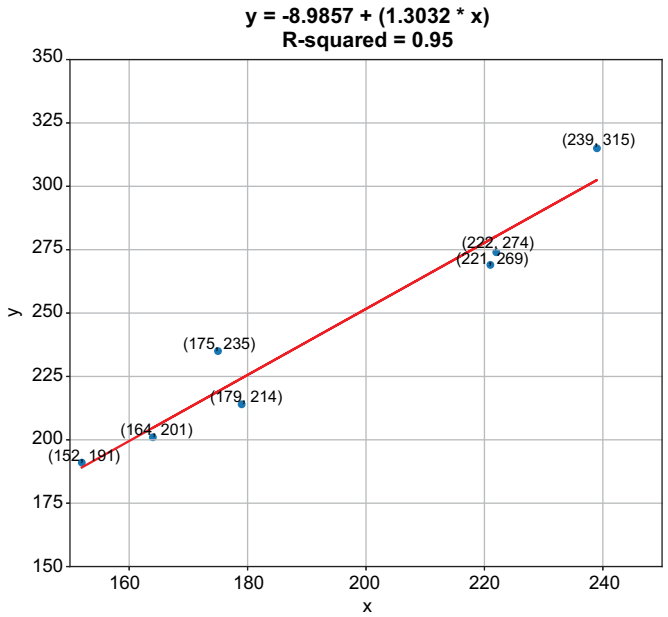

Figure 4.4 A scatter plot that displays seven data points rather than eight. The one presumed outlier was removed, and another regression was fit. The coefficient of determination, which was previously equal to 0.88, is now 0.95. By removing one outlier, we've made a strong model even stronger.

By removing one outlier from the data, we've minimized the residuals and—based just on the coefficient of determination (R^2), which increased from 0.88 to 0.95—improved the predictive power of our model. The independent variable `x` now explains 95% of the variance in the dependent variable `y`. However, be aware that removing outliers can sometimes lead to overfitting, as it may cause the model to fit too closely to the remaining data and reduce its generalizability to new data.

Now that we've covered many of the fundamentals of linear regression—goodness of fit measures, conditions for best fit, and other underlying assumptions—we'll import a real data set, demonstrate how to fit a simple linear regression on it, evaluate the key measures and show how they're derived, and test model assumptions.

4.2 Simple linear regression

Our plan is to do the following:

1. Import a real data set that contains 20 records and a pair of numeric variables.
2. Demonstrate how to perform a basic data exploration exercise.
3. Regress one of those variables against the other, print the results, and evaluate the key measures.

4 Provide instructions for how to run a series of tests and other checks to determine how well, or not so well, model assumptions have been met.

Data exploration is the process of analyzing data to reveal patterns, trends, and relationships that can influence statistical methods. When preparing to fit a linear regression, we should at least test for normality, determine whether the data contains any outliers that might weigh on our model, and take corrective action as necessary.

4.2.1 Importing and exploring the data

The *Marathon des Sables* (French for "Marathon of the Sands") is an ultramarathon that has been held every year since 1986 in the Sahara Desert. The race is divided into six stages over seven days; the total distance is 250 kilometers, or about 160 miles, roughly equivalent to six marathons. Daytime temperatures can sometimes reach 50° Celsius, or about 130° Fahrenheit. Runners are responsible for carrying their own supplies, like food and water, extra shoes and socks, etc.

Our data set contains the Stage 1 and Stage 2 split times for the top 20 male finishers from the 2021 race. The split times are in minutes, rounded to the nearest whole number. We import the data, a .csv file stored in our working directory, by passing the full filename, bounded by opening and closing quotation marks, to the `pd.read_csv()` method; `pd.read_csv()` directly accesses the working directory and, in this instance, imports a file called mds.csv. If mds.csv were stored elsewhere, we would need to pass the file's path—either the full path or a relative path—to `pd.read_csv()`.

The contents of mds.contents are read into a data frame called `mds`. But to prevent `pd.read_csv()` from throwing an error, we must first import the `pandas` library, a powerful Python library for data analysis and data manipulation:

```
>>> import pandas as pd
>>> mds = pd.read_csv('mds.csv')
```

Note that, by default, the `pd.read_csv()` method assumes that the first row of the file contains column headers. Now that we have imported our data, we can next call other methods to get a thorough understanding of it.

UNDERSTANDING THE DATA

The `info()` method in pandas returns a concise summary of the `mds` data frame. It's an easy and logical first data-exploration step:

```
>>> print(mds.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   stage1  20 non-null     int64
 1   stage2  20 non-null     int64
dtypes: int64(2)
memory usage: 452.0 bytes
None
```

More specifically, `info()` returns the following information:

- *The number of rows, or entries, in the data frame.* The `info()` method excludes the header and therefore returns just the number of entries that contain data. The `mds` data frame contains 20 rows of data. Python indexes the row count starting from 0 rather than 1.
- *The number of columns, listed from left to right.* Python also indexes columns starting at 0. The `mds` data frame contains two columns, `stage1` and `stage2`.
- *The number of non-null values in each column.* The sum of non-null values in both `stage1` and `stage2` equals 20; because the count of non-null values equals the `mds` record count, our data therefore contains no null values to worry about.
- *The data type of each column.* Both `stage1` and `stage2` are of type `int64` (short for integer).

Of course, it's always helpful to see the contents of a data frame instead of just being aware of its dimension and other properties. Next we'll show how to do that.

VIEWING THE DATA

There are several ways to get a glimpse of the data. If `mds` were a much longer data frame, we would likely call the `head()` and `tail()` methods in pandas to print the first few and last few records. But because `mds` is just 20 records long, we'll pass it as an argument to the `print()` method, which returns the entire data frame:

```
>>> print(mds)
    stage1  stage2
0      152     191
1      162     184
2      165     193
3      163     191
4      166     193
5      187     292
6      175     235
7      202     226
8      179     214
9      222     274
10     184     300
11     193     242
12     255     279
13     202     288
14     221     269
15     207     329
16     227     268
17     239     315
18     164     201
19     226     345
```

Seeing the data, even if it is just a small subset of records, piques our curiosity. We naturally want to know the size and shape of every variable. Even when the data comprises only 20 records, it's difficult or impossible to draw meaningful conclusions merely by observation.

COMPUTING BASIC STATISTICS

The `describe()` method in pandas interrogates the data and returns a series of *descriptive* statistics, provided the data is numeric, including the record count (`count`), mean (`mean`), standard deviation (`std`), minimum (`min`) and maximum (`max`) values, and three quartiles (`25%`, `50%`, and `75%`). When the data is categorical—that is, when the data represents categories or labels and therefore does not contain a natural numerical value—the `describe()` method instead returns *summary* statistics, such as the number of unique groups and the record count for each:

```
>>> print(mds.describe())
           stage1      stage2
count   20.000000   20.000000
mean   194.550000  251.450000
std     29.623026   51.112338
min    152.000000  184.000000
25%    165.750000  199.000000
50%    190.000000  255.000000
75%    221.250000  289.000000
max    255.000000  345.000000
```

Let's review these.

The *mean*, also known as the *average*, is a measure of central tendency that represents the sum of all values divided by the number of records; it is commonly used to describe the "typical" value in a set of numeric data. Another common measure of central tendency is the *median*, which represents the middle value in a sorted data set and can be more robust against outliers than the mean. When working with data stored in columns (a columnar format), you can apply these calculations across each column using methods like `apply()` and `median()`, available in pandas.

The *standard deviation* is a measure that quantifies the amount of variation, or dispersion, from the mean. A low standard deviation indicates that most of the data is close to the mean; a high standard deviation suggests longer tails and therefore more dispersion in the data. The standard deviation is obtained by calculating the square root of the variance, which is the average of the squared differences between each data point and the mean.

The *minimum* and *maximum* represent the smallest and largest values, respectively. From a more statistical perspective, they equal the lower and upper bounds of a numeric data series.

The second quartile (Q2) is `50%` and therefore equals the median, which is to say it divides the data into two equal halves. The first, or lower, quartile (Q1) is `25%`; it marks the boundary line below which lies the lowest 25% of the data. And `75%` is the third, or upper, quartile (Q3); it marks the boundary line above which lies the highest 75% of the data. By any of these measures, Stage 2 is clearly longer in distance, or at least more challenging, than Stage 1.

TESTING FOR NORMALITY

Linear regression assumes that the residuals—that is, the differences between the observed and predicted values—are normally distributed. We'll demonstrate how to

test this assumption after fitting our model, but for now, we want to know whether the variables themselves are normally distributed. That's because if the single predictor in a simple regression is normally distributed, the residuals are likely to be as well.

We can draw a histogram or density plot for each variable and determine from observation whether the data is or is not normally distributed. But that is a very imprecise method. Data that appears to be distributed normally (it resembles a bell shape) may in fact not be normal. And the reverse can also be true—data that looks non-normal may actually meet the standard for normality, which is why we'll refrain from drawing any plots and instead run a pair of Shapiro–Wilk tests.

Statistical testing and the 5% threshold for significance

Statistical tests are tools used to determine whether the results observed in data are likely due to a true effect or merely to random chance. These tests return a p-value, which quantifies the probability that the observed results could have occurred under the null hypothesis (assuming no real effect). Researchers often compare this p-value to a significance threshold to decide whether to reject or fail to reject the null hypothesis. The 5% significance level ($p < 0.05$) is a widely accepted threshold, where results are deemed statistically significant if there is less than a 5% probability they could occur under the null hypothesis; when that is the case, the null hypothesis is rejected. However, the choice of 5% is somewhat arbitrary; it's not derived from a universal principle but rather has become standard due to historical and practical reasons, making it a convenient convention in many fields.

Although 5% is not inherently special, it offers a balanced approach to controlling the risk of both Type I errors (false positives) and Type II errors (false negatives). This threshold provides a reasonably low probability of mistakenly rejecting the null hypothesis while maintaining sensitivity to detect real effects. Importantly, having a clear threshold—whether 5%, 1%, or any other predetermined value—is essential because it gives researchers a consistent, objective criterion for decision-making. Without such a threshold, interpreting statistical tests would become subjective, making it difficult to compare studies or replicate results.

The 5% significance level strikes a practical balance, helping maintain scientific rigor while allowing for the detection of meaningful results in data. The 5% significance threshold does not tell us the probability that the null hypothesis is true or false; it only indicates the likelihood of observing the data (or more extreme results) if the null hypothesis were true. Additionally, it does not measure the effect size or practical significance of the results.

A Shapiro–Wilk test is a statistical test that, based on the returned p-value, tells us whether a set of numeric data is normally distributed. The null hypothesis of a Shapiro–Wilk test is that the data is, in fact, normally distributed. We therefore require a really low p-value—less than 5%—to reject that null hypothesis and conclude the data is non-normal. But if a Shapiro–Wilk test instead returns a p-value greater than 5%, we will fail to reject that same null hypothesis and conclude that the data is normally distributed.

To run a pair of Shapiro–Wilk tests, one for each `mds` variable, we import the `stats` module from the `scipy` library, which contains various statistical functions for scientific computing, and then pass `stage1` and `stage2` to the `shapiro()` method:

```
>>> from scipy import stats

>>> print(stats.shapiro(mds.stage1))
ShapiroResult(statistic=0.9446673217389622, pvalue=0.29328758849732417)

>>> print(stats.shapiro(mds.stage2))
ShapiroResult(statistic=0.9290259439587454, pvalue=0.1478907551379664)
```

Because the p-values are above the 5% threshold, we should twice fail to reject the null hypothesis. So, according to our two Shapiro–Wilk tests, the `mds` variables `stage1` and `stage2` are both normally distributed.

DETECTING OUTLIERS

There are many ways to detect outliers in data. Statistical tests in this space have serious limitations, however. Boxplots display outliers, if there are any, as individual data points above and below the whiskers, but they don't actually tell us which records contain those outliers.

Let's first define an outlier as any data point that falls beyond three standard deviations from the mean, plus or minus, which is quite common. And of course we know the `stage1` and `stage2` means and standard deviations; we obtained these, along with other measures, when we called the `describe()` method.

So, a `stage1` outlier is any data point beyond the `stage1` mean plus or minus the product of 3 and the `stage1` standard deviation. We can define these, respectively, as `upper_threshold` and `lower_threshold`, where `194.55` is the mean and `29.62` is the standard deviation:

```
>>> upper_threshold = 194.55 + (3 * 29.62)
>>> lower_threshold = 194.55 - (3 * 29.62)
```

The following snippet of code subsets the `mds` data frame on those records that exceed either the `upper_threshold` or `lower_threshold`; the | operator means OR, so that only one condition must be satisfied. Python then returns a reduced data frame, `exceed_threshold`, or an empty data frame if `mds` contains zero records that meet either condition. Note the use of the backslash (\), which is used as a line continuation character in Python code:

```
>>> exceed_threshold = mds[(mds['stage1'] > upper_threshold) | \
>>>                        (mds['stage1'] < lower_threshold)]
>>> print(exceed_threshold)
Empty DataFrame
Columns: [stage1, stage2]
Index: []
```

The variable stage1 does not contain any outliers, at least according to our criteria. We get the same results for stage2, in fact, after changing our parameters and rerunning the same snippet of code:

```
>>> upper_threshold = 251.45 + (3 * 51.11)
>>> lower_threshold = 251.45 - (3 * 51.11)
>>> exceed_threshold = mds[(mds['stage2'] > upper_threshold) | \
>>>                        (mds['stage2'] < lower_threshold)]
>>> print(exceed_threshold)
Empty DataFrame
Columns: [stage1, stage2]
Index: []
```

If either stage1 or stage2 contained any outliers, we would at least consider taking corrective action by removing them. One method is to transform the data to a different distribution; this might be a good option if stage1 or stage2 were not already normally distributed. We could then, of course, solve two problems with one operation. Another method is to simply remove outliers from the data. Although this may solve one problem, it can also introduce new issues—especially for short data frames like mds. Yet another method is to winsorize the outlying data points. *Winsorization* is the process of replacing extreme values with less extreme values. A data point more than three standard deviations from the mean can be modified so that it then falls within a specified percentile range.

There is no right way or best way of treating outliers; it depends on the circumstances. However, there is absolutely a right or best *process*: carefully consider the implications of each alternative, and always document your thought process for the purposes of transparency and reproducibility. Now let's fit our linear regression.

4.2.2 Fitting the model

We'll fit a simple linear regression by calling the OLS() method from the statsmodels library; stage2 will be our dependent variable, and stage1 will be our independent variable. We are therefore regressing stage2 against stage1 to estimate Stage 2 split times from Stage 1 split times.

> **NOTE** OLS is short for *ordinary least squares*. That's because linear regression minimizes the distances between the observed and predicted values of the dependent variable by estimating the sum of the squared differences between them. The method of estimation is known as "ordinary" to distinguish it from other regression techniques and "least squares" because, once more, it minimizes the squared differences.

Our first order of business is to import the statsmodels library and define our response variable, y:

```
>>> import statsmodels.api as sm
>>> y = mds['stage2']
```

Then we define our predictor variable, `x`:

```
>>> x = mds['stage1']
```

If we were instead fitting a multiple linear regression with a total of two predictors, `stage1` and, let's say, `second_variable`, we would define `x` this way (note the extra set of brackets):

```
>>> x = mds[['stage1', 'second_variable']]
```

Next we add a constant to `x`, known as the *intercept*:

```
>>> x = sm.add_constant(x)
```

We can now fit our model, `lm`, and print the results:

- `sm.OLS(y, x)` initializes the regression using `x (stage1)` as the only independent variable and `y (stage2)` as the dependent variable. Because we previously *added* the constant to `x`, we only need to pass `y` and `x`; the constant is already accounted for.
- `fit()` fits the initialized model to the `mds` data frame and estimates the coefficients of the linear regression that minimizes the sum of squared differences between the observed and predicted values of `y`. These results and other meaningful statistics are read into an object called `lm`.
- `print()` returns the `lm` content in the form of what's called a *regression table*.

It takes just two lines of Python code to fit the regression and print the model output:

```
>>> lm = sm.OLS(y, x).fit()
>>> print(lm.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                 stage2   R-squared:                       0.575
Model:                            OLS   Adj. R-squared:                  0.551
Method:                 Least Squares   F-statistic:                     24.35
Date:                Mon, 11 Mar 2024   Prob (F-statistic):           0.000107
Time:                        14:06:15   Log-Likelihood:                -97.991
No. Observations:                  20   AIC:                             200.0
Df Residuals:                      18   BIC:                             202.0
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -3.0778     52.150     -0.059      0.954    -112.640     106.485
stage1         1.3083      0.265      4.934      0.000       0.751       1.865
==============================================================================
Omnibus:                        2.616   Durbin-Watson:                   2.326
Prob(Omnibus):                  0.270   Jarque-Bera (JB):                2.150
Skew:                           0.749   Prob(JB):                        0.341
Kurtosis:                       2.420   Cond. No.                     1.34e+03
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the
errors is correctly specified.
[2] The condition number is large, 1.34e+03. This might
indicate that there are strong multicollinearity or other
numerical problems.
```

It will take much more effort to review and evaluate these results and apply them.

4.2.3 Interpreting and evaluating the results

The statsmodels library returns regression results in tabular form, not as a scatter plot with a fitted regression line, because it prioritizes numerical analysis and statistical summaries over graphical representations. This makes sense for a couple of reasons. Although scatter plots can be useful for visualizing simple regressions with a single predictor, they aren't practical for models with multiple predictors, which require multidimensional plots that are difficult to display. For starters, we should be more interested in measures of predictive power, statistical significance, and the like, which we only get from a regression table. And in any event, scatter plots are derived from just a subset of these same results, so it only makes sense to first evaluate the raw data and then have the flexibility to create a plot over it. That being said, let's unpack these results by interpreting and evaluating the most critical measures one by one.

COEF

In the model output, under `coef`, the intercept (β_0) equals `-3.0778`, and the slope (β_1) equals `1.3083`. We therefore get the fitted regression by plugging these two values into the linear equation:

$$\hat{y} = -3.0778 + 1.3083(x)$$

or

$$\text{stage2} = -3.0778 + 1.3083(\text{stage1})$$

So, if a runner's Stage 1 split time was 193 minutes, we would then predict his Stage 2 split time to equal

$$\text{stage2} = -3.0778 + 1.3083(193)$$

or 249 minutes (rounded to the nearest whole number).

Let's illustrate these same results with a Matplotlib scatter plot that contains the observed values for `stage1` and `stage2` overlayed with the fitted regression line. Here's the snippet of code:

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(mds['stage1'], mds['stage2'])
>>> plt.xlim(150, 260)
```

Imports the matplotlib library

Draws a scatter plot with stage1 data on the x axis and stage2 data on the y axis

Establishes the minimum and maximum values of the x axis

```
>>> plt.ylim(150, 350)
>>> yhat = -3.078 + (1.3083 * x)
>>> plt.plot(x, yhat, linewidth = 2,
>>>          color = 'red')
>>> plt.title('2021 Marathon Des Sables - Top 20 Male Finishers\n'
>>>           'y = -3.078 + (1.3083 * x)\n'
>>>           'R-squared = 0.575',
>>>           fontweight = 'bold')
>>> plt.xlabel('Stage 1 Running Time (min)')
>>> plt.ylabel('Stage 2 Running Time (min)')
>>> plt.grid()
>>> plt.show()
```

Establishes the minimum and maximum values of the y axis

Calculates the predicted values from the simple linear regression equation

Plots the regression line and defines the aesthetics

Sets the title

Sets the x axis label

Sets the y axis label

Adds vertical and horizontal grid lines to the plot

Displays the plot

Figure 4.5 shows our scatter plot, where each point represents the observed `stage1` and `stage2` values from a row in the `mds` data frame, and the fitted regression line represents the predictions for `stage2`. It's especially important to understand how the slope and intercept are derived. Although other measures are critical for evaluating fit, the coefficient estimates represent the cornerstone of regression analysis by quantifying the relationship between variables and calculating predictions that drive decision-making.

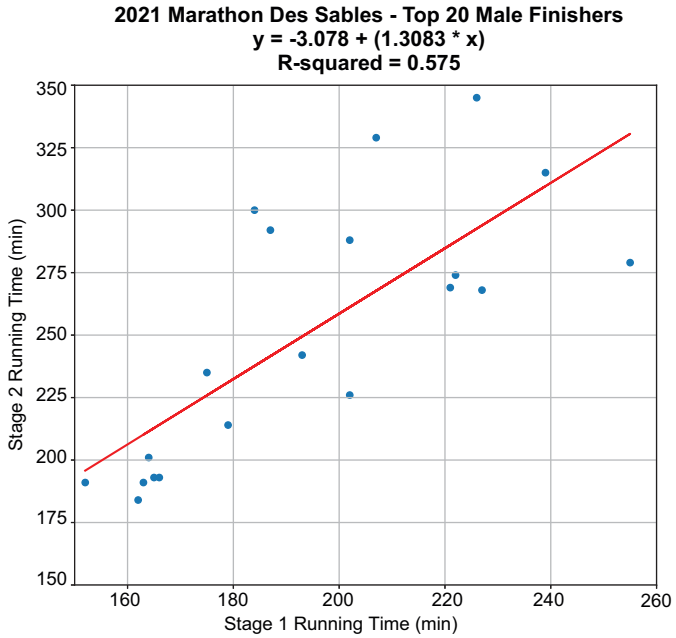

Figure 4.5 A scatter plot that displays the observed `stage1` and `stage2` values from the `mds` data frame, their respective *x* and *y* coordinates, and a regression line that represents the predictions for `stage2` from a simple linear regression where `stage2` was regressed against `stage1`. The regression line is drawn by applying the linear equation at the top of the plot. R-squared is one of several metrics contained in the model output; it represents the percentage of variance in `stage2` that can be explained by changes in `stage1`.

Let's begin by revisiting the linear equation for a simple regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$

We require the means and totals for `stage1` and `stage2` to calculate the intercept and slope. The `stage1` and `stage2` means equal `194.55` and `251.45`, respectively; these were obtained when we passed the `mds` data frame to the `describe()` method. The `stage1` and `stage2` sums can be derived by multiplying their means by the `mds` record count or, better yet, making a pair of calls to the `sum()` method:

```
>>> stage1_sum = mds['stage1'].sum()
>>> print(stage1_sum)
3891

>>> stage2_sum = mds['stage2'].sum()
>>> print(stage2_sum)
5029
```

The slope represents the rate of change in the dependent variable for a one-unit change in the independent variable; it therefore quantifies the strength and direction of the linear relationship between variables. It can be calculated by plugging the `stage1` and `stage2` totals (x and y) and the `stage1` and `stage2` means ($\bar{x}$ and $\bar{y}$) into the following formula (see table 4.1 for reference):

$$\beta_1 = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

or

$$\beta_1 = \frac{21813.05}{16672.95} = 1.308$$

The intercept, meanwhile, represents the value of the dependent variable when the independent variable (or variables) is equal to zero; it therefore indicates the point where the regression line intersects the y axis. It can be calculated by adding the slope and the `stage1` and `stage2` means to the following formula:

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

or

$$\beta_0 = 251.45 - 1.308\,(194.55) = -3.0778$$

The coefficient estimates represent the most significant output from regression analysis. They help us understand and quantify the association between independent and dependent variables. By regressing `stage2` against `stage1`, we've discovered that for every one-unit (or one-minute) increase in `stage1`, we should expect `stage2` to increase by 1.3083. The other measures worth evaluating have much less to do with quantifying the association between variables and more to do with assessing goodness of fit.

Table 4.1 Totals to help derive slope estimates. Fractional numbers are rounded to two decimal places. The first two columns, from left to right, represent the observed split times. The slope is derived by dividing the total in the last column by the total in the third column, but these totals can only be derived by first computing the same for each observation in the data.

x	y	$(x-\bar{x})^2$	$(x-\bar{x})(y-\bar{y})$
152	191	$(152 - 194.55)^2 = 1{,}810.50$	$(152 - 194.55)(191 - 251.45) = 2{,}572.15$
162	184	$(162 - 194.55)^2 = 1{,}059.50$	$(152 - 194.55)(184 - 251.45) = 2{,}195.50$
165	193	$(165 - 194.55)^2 = 873.20$	$(152 - 194.55)(193 - 251.45) = 1{,}727.20$
163	191	$(163 - 194.55)^2 = 995.40$	$(152 - 194.55)(191 - 251.45) = 1{,}907.20$
166	193	$(166 - 194.55)^2 = 815.10$	$(152 - 194.55)(193 - 251.45) = 1{,}668.75$
187	292	$(187 - 194.55)^2 = 57.00$	$(152 - 194.55)(292 - 251.45) = -306.15$
175	235	$(175 - 194.55)^2 = 382.20$	$(152 - 194.55)(235 - 251.45) = 321.60$
202	226	$(202 - 194.55)^2 = 55.50$	$(152 - 194.55)(226 - 251.45) = -189.60$
179	214	$(179 - 194.55)^2 = 241.80$	$(152 - 194.55)(214 - 251.45) = 582.35$
222	274	$(222 - 194.55)^2 = 753.50$	$(152 - 194.55)(274 - 251.45) = 618.00$
184	300	$(184 - 194.55)^2 = 111.30$	$(152 - 194.55)(300 - 251.45) = -512.20$
193	242	$(193 - 194.55)^2 = 2.40$	$(152 - 194.55)(242 - 251.45) = 14.65$
255	279	$(255 - 194.55)^2 = 3{,}654.20$	$(152 - 194.55)(279 - 251.45) = 1{,}665.40$
202	288	$(202 - 194.55)^2 = 55.50$	$(152 - 194.55)(288 - 251.45) = 272.30$
221	269	$(221 - 194.55)^2 = 699.60$	$(152 - 194.55)(269 - 251.45) = 464.20$
207	329	$(207 - 194.55)^2 = 155.00$	$(152 - 194.55)(329 - 251.45) = 965.50$
227	268	$(227 - 194.55)^2 = 1{,}053.00$	$(152 - 194.55)(268 - 251.45) = 537.05$
239	315	$(239 - 194.55)^2 = 1{,}975.80$	$(152 - 194.55)(315 - 251.45) = 2{,}824.80$
164	201	$(164 - 194.55)^2 = 933.30$	$(152 - 194.55)(201 - 251.45) = 1{,}541.25$
226	345	$(226 - 194.55)^2 = 989.10$	$(152 - 194.55)(345 - 251.45) = 2{,}942.15$
3891	*5,029*	*16,672.95*	*21,813.05*

R-SQUARED

In the model output, `R-squared` is equal to `0.575`. R-squared (R^2), the coefficient of determination, equals the percentage of the variance in the response variable that can be explained by the independent variable. So, 57.5% of the variance in Stage 2 split

times from the top 20 male finishers in the 2021 Marathon des Sables can be explained by their Stage 1 split times. R^2 is a fundamental and significant measure of fit for regression analysis—either fixed to just one model or even to compare different models fitted to the same data. Because it's a squared number and represents the percentage of the variance in the response variable caused by the one predictor in a simple regression or the two or more predictors in a multiple regression, R^2 will always equal a number between 0 and 1. In order for R^2 to equal 1, every data point must lie on the regression line, which is rare.

Adding independent variables typically produces increases in R^2, even when those additional variables are statistically insignificant and therefore don't help to further explain variances in the response variable. When that's the case, R^2 and Adjusted R^2, which are typically aligned in a simple regression, diverge. That's because Adjusted R^2 adds a penalty for including unnecessary predictors; it therefore attempts to balance model complexity with explanatory power. When R^2 and Adjusted R^2 differ significantly, Adjusted R^2 should be prioritized for quantifying model fit. In such cases, it's advisable to perform model reduction by systematically removing insignificant predictors to improve the model's accuracy and interpretability.

The most direct way of deriving R^2 is to divide the sum of squares regression (SSR) by the sum of squares total (SST):

$$R^2 = \frac{\text{SSR}}{\text{SST}}$$

The SST is a measure of the total variability in the dependent variable without consideration of any predictors (this and other measures are squared to prevent positive and negative results from potentially canceling each other out). Mathematically, it is equal to the sum of the squared differences between the observed values of the dependent variable and the overall mean of the dependent variable:

$$\text{SST} = \sum (y - \bar{y})^2$$

Thus, observed values far from the `stage2` mean will return a high sum of squares, and observed values close to the mean will return a low sum of squares. A pair of examples with the aid of a table should provide all the further transparency we need (see table 4.2). Consider just the second and third columns from the left: the stage2 column contains the observed values from the `mds` variable `stage2`, and the SST column equals their respective sum of squares, or the variability of each observed value relative to the `stage2` mean. Where the stage2 column value is 184, which of course is distant from the `stage2` mean of `251.45`, the sum of squares equals 4,549.50, or the product of (184 − 251.45) and (184 − 251.45), or simply $(184 - 251.45)^2$. Where the stage2 column value instead equals 242, a value much closer to the `stage2` mean, the sum of squares equals just 89.30, or $(242 - 251.45)^2$. Adding the individual sum of squares returns the SST, 49,636.95.

Table 4.2 The sum of squares summary for the fitted linear regression. The stage1 and stage2 columns represent, respectively, the observed values from the two `mds` variables. SST equals the sum of squares total, or the total variability in the dependent variable without the consideration of any predictors. The yhat column contains the predictions for the response variable from the fitted regression. SSE equals the sum of squares error, or the portion of the SST that can't be explained by the regression; and SSR equals the sum of squares regression, or the percentage of the variance in the dependent variable explained by the model.

x stage1	y stage2	$(y - \bar{y})^2$ SST	y yhat	$(y - \hat{y})^2$ SSE	$(\hat{y} - \bar{y})^2$ SSR
152	191	3,654.20	195.78	22.88	3098.73
162	184	4,549.50	208.87	618.36	1813.33
165	193	3,416.40	212.79	391.71	1494.46
163	191	3,654.20	210.18	367.68	1,703.62
166	193	3,416.40	214.10	445.21	1,395.02
187	292	1,644.30	241.57	2,542.75	97.53
175	235	270.60	225.87	83.27	654.10
202	226	647.70	261.20	1,238.96	95.04
179	214	1,402.50	231.11	292.68	413.80
222	274	508.50	287.36	178.62	1,289.87
184	300	2,357.10	237.65	3,887.60	190.46
193	242	89.30	249.42	55.12	4.10
255	279	759.00	330.54	2,656.24	6,255.02
202	288	1,335.90	261.20	718.30	95.04
221	269	308.00	286.06	290.92	1,197.61
207	329	6,014.00	267.74	3,752.75	265.37
227	268	273.90	293.91	671.14	1,802.54
239	315	4,038.60	309.61	29.10	3,382.11
164	201	2,545.20	211.48	109.90	1,597.33
226	345	8,751.60	292.60	2,745.97	1,693.16
		49,636.95		*21,099.16*	*28,538.24*

It's much easier to use Python as a calculator to get this result. In the following snippet of code, which requires `numpy`, `**` is the exponential operator. When coupled with the numeral `2`, it means we're raising the result contained within the parentheses by the power of 2:

```
>>> import numpy as np
>>> SST = np.sum((mds['stage2'] - np.mean(mds['stage2'])) ** 2)
```

```
>>> print(SST)
49636.950000000004
```

The SSR, meanwhile, quantifies the variability in the dependent variable explained by the regression model. It represents the sum of squared differences between the predicted values of the dependent variable and the mean of those values:

$$\text{SSR} = \sum (\hat{y} - \bar{y})^2$$

So, predictions far removed from the `stage2` mean will return a high sum of squares, whereas other predictions close to the mean will return a low sum of squares. Now consider the yhat and SSR columns in table 4.2: yhat represents the predicted values for the dependent variable from our regression model, and SSR equals the sum of squares, or the variability of our model's predictions for `stage2` relative to the `stage2` mean. With respect to the top record in our table, the sum of squares equals 3,098.73, which in turn is the product of (195.78 − 251.45) and (195.78 − 251.45), or $(195 - 251.45)^2$. By adding the individual sum of squares, we get the SSR, 28,538.24.

We get the same result by plugging the SSR equation into Python—that is, by squaring the differences between the predicted values for `stage2` and the mean of `stage2` and summing the results—as long as we ignore the rounding differences:

```
>>> SSR = np.sum((lm.predict(x) - np.mean(mds['stage2'])) ** 2)
>>> print(SSR)
28537.790271217757
```

The object we created when fitting our regression, `lm`, contains a variable called `fittedvalues`, which holds the predictions for the dependent variable `stage2`. Whereas yhat represents predictions that were more or less calculated offline by plugging the coefficients into the linear equation, `fittedvalues` contains the actual predictions for `stage2` from the fitted regression. They are slightly different paths to the same destination:

```
>>> print(lm.fittedvalues)
0     195.782270
1     208.865168
2     212.790037
3     210.173458
4     214.098327
5     241.572412
6     225.872935
7     261.196759
8     231.106094
9     287.362554
10    237.647543
11    249.422151
12    330.536117
13    261.196759
14    286.054265
```

```
15    267.738208
16    293.904003
17    309.603480
18    211.481747
19    292.595713
dtype: float64
```

Now that we have the SSR and the SST, we can divide one by the other and get the coefficient of determination for our linear regression:

$$R^2 = \frac{\text{SSR}}{\text{SST}} = \frac{28537.79}{49636.95} = 0.575$$

In Python, it's like so:

```
>>> R2 = SSR / SST
>>> print(R2)
0.5749303748763321
```

The SSE, or the sum of squares error, sometimes referred to as the sum of squared residuals, represents that portion of the SST that the SSR fails to explain. It therefore equals the difference between the SST and the SSR:

$$\text{SSE} = \text{SST} - \text{SSR} = 21099.16$$

This means the SST equals the sum of the SSR and the SSE:

$$\text{SST} = \text{SSR} + \text{SSE} = 49636.95$$

We can confirm both by again using Python as a calculator:

```
>>> SSE = SST - SSR
>>> print(SSE)
21099.159728782248

>>> SST_new = SSR + SSE
>>> print(SST_new)
49636.950000000004
```

The SSE represents the sum of the squared differences between the observed and predicted values for the dependent variable:

$$\text{SSE} = \sum (y - \hat{y})^2$$

In Python, we sum and square the residuals, like so:

```
>>> SSE = np.sum(lm.resid ** 2)
>>> print(SSE)
21099.159728782255
```

Let's return to table 4.2 one last time. The 20 individual values listed in the SSE column equal the squared differences between the observed and predicted values for `stage2`. Thus, the SSE for the top record equals 22.88, which is equal to $(191 - 195.78)^2$. The SSE, 21,099.16, is then obtained by adding the results of the 20 sum of squares.

Now that we have the SSE, we can next demonstrate a second way of deriving R^2. Because the SSE represents the portion of the SST that the SSR fails to account for, we can derive R^2 by dividing the SSE by the SST and subtracting the quotient from 1:

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}} = 1 - \frac{21099.16}{49636.95} = 0.575$$

The following are a few notes about R^2:

- R^2 represents the percentage of variation in the dependent variable explained by the one predictor in a simple regression and the two or more predictors in a multiple regression.
- R^2 will always equal a number between 0 and 1, where 0 means the predictors don't explain any of the variance in the response variable and 1 means the predictors explain all of the variability.
- R^2 is a critical goodness of fit measure for one model or for competing models fit to the same data.

Although R^2 is paramount among goodness of fit measures, model strength must be evaluated by considering other measures as well.

F-STATISTIC

The `F-statistic`, equal to `24.35`, is a measure of the overall significance of a regression. It evaluates whether the model explains a statistically significant amount of the variance in the dependent variable compared to a model with just an intercept and therefore no predictors. The formula for the F-statistic is

$$F = \frac{\frac{SSR}{p}}{\frac{SSE}{n-p-1}}$$

where

- SSR is the sum of squares regression.
- SSE is the sum of squares error.
- n is the number of observations.
- p is the number of predictors.

This means

$$F = \frac{\frac{25838.24}{1}}{\frac{21099.16}{20-1-1}} = \frac{25838.24}{1172.17} = 24.35$$

To interpret the F-statistic, we need to bump it against the critical value in an F-table; it doesn't have any immediate meaning by itself. If the F-statistic is greater than the critical value (see figure 4.6), we can then conclude that the model explains a statistically significant amount of the variance in the response variable.

The critical value is located by first selecting a significance level (5% is most common) and then cross-referencing the number of independent variables in the regression with the number of observations in the data minus 1. The rows represent the former (1), and the columns represent the latter (19). Because the F-statistic is greater than the critical value of 4.38, we conclude that a statistically significant amount of the variance in `stage2` is explained by our simple linear regression.

	F-table of Critical Values of α = 0.05 for F(df1, df2)																		
	DF1=1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**15**	**20**	**24**	**30**	**40**	**60**	**120**	**∞**
DF2=1	161.45	199.50	215.71	224.58	230.16	233.99	236.77	238.88	240.54	241.88	243.91	245.95	248.01	249.05	250.10	251.14	252.20	253.25	254.31
2	18.51	19.00	19.16	19.25	19.30	19.33	19.35	19.37	19.38	19.40	19.41	19.43	19.45	19.45	19.46	19.47	19.48	19.49	19.50
3	10.13	9.55	9.28	9.12	9.01	8.94	8.89	8.85	8.81	8.79	8.74	8.70	8.66	8.64	8.62	8.59	8.57	8.55	8.53
4	7.71	6.94	6.59	6.39	6.26	6.16	6.09	6.04	6.00	5.96	5.91	5.86	5.80	5.77	5.75	5.72	5.69	5.66	5.63
5	6.61	5.79	5.41	5.19	5.05	4.95	4.88	4.82	4.77	4.74	4.68	4.62	4.56	4.53	4.50	4.46	4.43	4.40	4.37
6	5.99	5.14	4.76	4.53	4.39	4.28	4.21	4.15	4.10	4.06	4.00	3.94	3.87	3.84	3.81	3.77	3.74	3.70	3.67
7	5.59	4.74	4.35	4.12	3.97	3.87	3.79	3.73	3.68	3.64	3.57	3.51	3.44	3.41	3.38	3.34	3.30	3.27	3.23
8	5.32	4.46	4.07	3.84	3.69	3.58	3.50	3.44	3.39	3.35	3.28	3.22	3.15	3.12	3.08	3.04	3.01	2.97	2.93
9	5.12	4.26	3.86	3.63	3.48	3.37	3.29	3.23	3.18	3.14	3.07	3.01	2.94	2.90	2.86	2.83	2.79	2.75	2.71
10	4.96	4.10	3.71	3.48	3.33	3.22	3.14	3.07	3.02	2.98	2.91	2.85	2.77	2.74	2.70	2.66	2.62	2.58	2.54
11	4.84	3.98	3.59	3.36	3.20	3.09	3.01	2.95	2.90	2.85	2.79	2.72	2.65	2.61	2.57	2.53	2.49	2.45	2.40
12	4.75	3.89	3.49	3.26	3.11	3.00	2.91	2.85	2.80	2.75	2.69	2.62	2.54	2.51	2.47	2.43	2.38	2.34	2.30
13	4.67	3.81	3.41	3.18	3.03	2.92	2.83	2.77	2.71	2.67	2.60	2.53	2.46	2.42	2.38	2.34	2.30	2.25	2.21
14	4.60	3.74	3.34	3.11	2.96	2.85	2.76	2.70	2.65	2.60	2.53	2.46	2.39	2.35	2.31	2.27	2.22	2.18	2.13
15	4.54	3.68	3.29	3.06	2.90	2.79	2.71	2.64	2.59	2.54	2.48	2.40	2.33	2.29	2.25	2.20	2.16	2.11	2.07
16	4.49	3.63	3.24	3.01	2.85	2.74	2.66	2.59	2.54	2.49	2.42	2.35	2.28	2.24	2.19	2.15	2.11	2.06	2.01
17	4.45	3.59	3.20	2.96	2.81	2.70	2.61	2.55	2.49	2.45	2.38	2.31	2.23	2.19	2.15	2.10	2.06	2.01	1.96
18	4.41	3.55	3.16	2.93	2.77	2.66	2.58	2.51	2.46	2.41	2.34	2.27	2.19	2.15	2.11	2.06	2.02	1.97	1.92
19	4.38	3.52	3.13	2.90	2.74	2.63	2.54	2.48	2.42	2.38	2.31	2.23	2.16	2.11	2.07	2.03	1.98	1.93	1.88
20	4.35	3.49	3.10	2.87	2.71	2.60	2.51	2.45	2.39	2.35	2.28	2.20	2.12	2.08	2.04	1.99	1.95	1.90	1.84
21	4.32	3.47	3.07	2.84	2.68	2.57	2.49	2.42	2.37	2.32	2.25	2.18	2.10	2.05	2.01	1.96	1.92	1.87	1.81
22	4.30	3.44	3.05	2.82	2.66	2.55	2.46	2.40	2.34	2.30	2.23	2.15	2.07	2.03	1.98	1.94	1.89	1.84	1.78
23	4.28	3.42	3.03	2.80	2.64	2.53	2.44	2.37	2.32	2.27	2.20	2.13	2.05	2.01	1.96	1.91	1.86	1.81	1.76
24	4.26	3.40	3.01	2.78	2.62	2.51	2.42	2.36	2.30	2.25	2.18	2.11	2.03	1.98	1.94	1.89	1.84	1.79	1.73

Figure 4.6 A snippet from a typical F-table where the selected significance level equals 5%. The critical value is located at the intersection of the predictor count (1) and the observation count minus 1 (19). Because the F-statistic is greater than the critical value, the model explains a statistically significant amount of the variance in the response variable.

However, we don't reject or fail to reject a null hypothesis based on the F-statistic. We do so by evaluating the p-value associated with the F-statistic.

PROB (F-STATISTIC)

`Prob (F-statistic)`, which is equal to `0.000107`, represents the p-value associated with the F-statistic. It indicates the probability of observing an F-statistic as extreme, or

even more extreme, than the one calculated from the data, under the assumption that the null hypothesis is true.

When modeling a linear relationship between variables, the null hypothesis states that all regression coefficients except the intercept are equal to zero, and therefore the independent variables have no effect whatsoever on the dependent variable. However, when the p-value from the F-statistic is less than the 5% threshold for significance, as it is here, we should reject that null hypothesis and conclude that at least one predictor in the regression has a statistically significant effect on the dependent variable. On the other hand, a p-value above that same 5% threshold would suggest the regression as a whole is not statistically significant, and therefore we should fail to reject the same null hypothesis.

In a simple linear regression, a p-value less than 5% indicates that the single predictor is statistically significant at the commonly used 5% significance level, meaning there's less than a 5% chance that the observed association is due to random variation. The 5% threshold is an arbitrary convention, often chosen for convenience, and can be applied in either a one-tailed or a two-tailed test depending on the hypothesis. However, in regression analysis, two-tailed tests are typically used because we are often testing whether a coefficient is significantly different from zero in either direction. In multiple regression, a model with a p-value below 5% suggests that at least one predictor is statistically significant, although others may not be. In this case, model reduction—removing statistically insignificant predictors—can help simplify the model and improve interpretability.

P>|t|

`P>|t|` represents the p-value for each predictor. The p-value for `stage1` equals `0.000`, so it is therefore statistically significant. Had we instead fit a multiple regression with a mix of significant and insignificant predictors, we would need to balance model complexity with explanatory power. The preferred approach in this case is to fit a reduced model that includes only the statistically significant predictors, simplifying the model while retaining its most meaningful explanatory variables.

We've fit our model and assessed its most critical outputs. Now it's important to determine whether our conclusions are valid by testing the underlying model assumptions.

4.2.4 Testing model assumptions

Once a regression has been fit and the key measures interpreted and evaluated, a series of tests should then be run to (hopefully) confirm that the regression results are reliable. Our trust in the model output should be temporarily paused until we've determined that all linear regression assumptions have been tested and validated.

Assumption #1: Linearity between variables

There must be a linear relationship between the independent and dependent variables. In other words, the relationship is assumed to be adequately represented by a straight line so that changes in the predictor cause proportional changes, positive or negative, in the

response variable. Although R^2 measures the proportion of variance in the dependent variable explained by the model, it doesn't confirm whether the relationship between variables is actually linear. A high R^2 value could still occur in cases where a nonlinear relationship exists, making it essential to use visual methods instead.

The best method of checking this assumption is to draw a residuals plot, or a scatter plot that displays the fitted values against the residual values, and evaluate the results. Once again, we'll use Matplotlib. Here's the snippet of code:

```
>>> plt.scatter(lm.fittedvalues, lm.resid)
>>> plt.axhline(y = 0, color = 'red',
>>>             linestyle = '--')
>>> plt.title('Residuals Plot')
>>> plt.xlabel('Fitted Values')
>>> plt.ylabel('Residuals')
>>> plt.show()
```

Draws a scatter plot with fittedvalues and lm.resid (the residuals) on the y axis

Draws a red and dashed horizontal line where y = 0

Sets the title

Sets the x-axis label

Sets the y-axis label

Displays the plot

Figure 4.7 shows the result: a scatter plot where the fitted values from the linear regression are plotted along the *x* axis and the residuals are plotted along the *y* axis. We should conclude that the assumption of linearity between variables is met when the residuals appear randomly scattered around the horizontal axis, as they do here. This randomness indicates that the model has successfully captured the systematic (linear) component of the relationship between the predictor and response variables. If a nonlinear pattern were present, such as a curve or funnel shape, it would suggest that a linear model was insufficient and that some structure in the data remained unexplained. But in this case, the absence of such a pattern supports the validity of the linearity assumption.

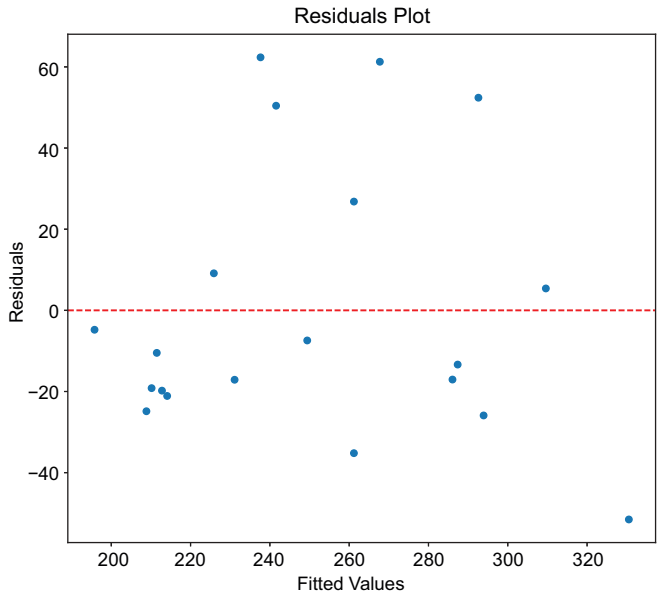

Figure 4.7 A residuals plot that displays the fitted values from a linear regression on the x axis and the residuals from the same model on the y axis. Linearity between variables is confirmed when the data points do not follow any obvious pattern or trend.

Assumption #2: Independence of residuals

The residuals are assumed to be independent of one another; in other words, it's assumed that there is no correlation between them. Correlation between the residuals specifically refers to the degree to which the errors from the linear regression are related to each other. A meaningful correlation between the residuals might be an indication that the regression model failed to detect some systematic pattern in the data.

The Durbin–Watson test is a statistical method used to detect correlation in the residuals of a regression model. It provides a test statistic ranging from 0 to 4, which helps determine whether we should reject the null hypothesis of no correlation. When the test statistic falls between 1.5 and 2.5, we generally fail to reject the null hypothesis, indicating that the assumption of independent residuals is likely satisfied. If the statistic falls outside this range, we may reject the null hypothesis, suggesting potential correlation in the residuals. The closer the test statistic is to the extremes (0 or 4), the stronger the evidence for positive or negative correlation, respectively.

The Durbin–Watson test statistic is calculated as the ratio of the squared differences between successive residuals (numerator) to the total sum of squared residuals (denominator). In the regression table, the Durbin–Watson statistic is given as 2.326, which allows us to conclude that we should fail to reject the null hypothesis and that the assumption of independence in the residuals is satisfied. Alternatively, you can calculate it by importing the `statstools` module from the `statsmodels.stats` package and passing the residuals of the regression (e.g., `lm.resid`) to the `durbin_watson()` method:

```
>>> from statsmodels.stats.stattools import durbin_watson
>>> print(durbin_watson(lm.resid))
2.3264601831112186
```

So far, so good, but we have two more assumptions to test.

Assumption #3: Homoscedasticity of residuals

The residuals are also assumed to be homoscedastic and thus not heteroscedastic. Homoscedasticity refers to the expectation that the variability, or spread, of the residuals is consistent across all levels of the predictors; that is, the spread of the residuals should not increase or decrease as the values of the predictors change. This assumption is important because heteroscedasticity can lead to inefficient estimates and biased standard errors, which in turn can distort hypothesis tests and confidence intervals. If the variance of residuals increases or decreases systematically, it undermines the reliability of the model's inferences—making it more likely to incorrectly accept or reject statistical significance. Ensuring homoscedasticity, therefore, strengthens the validity of any conclusions drawn from the model.

The Breusch–Pagan test is a statistical method for detecting heteroscedasticity in the residuals of a regression model or, alternatively, whether the residuals have constant variance (homoscedasticity) across observations. The test works by assessing whether the variance of the residuals depends systematically on the values of the

independent variables. The null hypothesis is that the residuals are homoscedastic. If the test returns a p-value less than 5%, we reject the null hypothesis and conclude that the residuals exhibit heteroscedasticity. Conversely, if the p-value is greater than 5%, we conclude that the residuals are homoscedastic, meaning the assumption of constant variance holds.

The Breusch–Pagan test statistic is calculated by regressing the squared residuals from the original regression model on the independent variables and examining the explained variance in this auxiliary regression. The test statistic is proportional to the explained variance in this second regression, where a high value indicates that variance in residuals may depend on predictor values, suggesting heteroscedasticity.

To perform a Breusch–Pagan test, first import the `het_breuschpagan` function from the `statsmodels.stats.diagnostic` module:

```
>>> from statsmodels.stats.diagnostic import het_breuschpagan
```

Then pass the residuals (`lm.resid`) and the independent variables (`lm.model.exog`) from the fitted regression model to the `het_breuschpagan()` method:

```
>>> lm_BreuschPagan = het_breuschpagan(lm.resid, lm.model.exog)
```

The `het_breuschpagan()` method calculates the test statistic, the p-value, and additional test details, returning them as a tuple. Because we are mainly interested in the p-value to assess the presence of heteroscedasticity, we can use `print()` to display only the second element of the tuple, which is the p-value:

```
>>> print(lm_BreuschPagan[1])
0.181071316321208
```

In this case, the p-value is greater than 5%, so we fail to reject the null hypothesis and conclude that the assumption of homoscedasticity (constant variance in residuals) has been satisfied.

ASSUMPTION #4: NORMALITY OF RESIDUALS

Finally, *the residuals are assumed to be normally distributed.* If this assumption is violated, it may affect the integrity of the coefficient estimates and the significance of the predictors. There are two main approaches to testing this: a visual, subjective method and a more precise statistical test.

One of the most common methods for assessing normality is to create a Quantile–Quantile (Q–Q) plot. In a Q–Q plot, the residuals are represented by points and are compared to a perfectly normal distribution, represented by a dashed line at a 45-degree angle. If the residuals follow a normal distribution, the points should roughly align along this line. Minor deviations may be acceptable depending on the context, but significant departures suggest a potential violation of the normality assumption. Here's a simple way to create a Q–Q plot using Matplotlib:

```
>>> stats.probplot(lm.resid,
>>>                dist = 'norm', plot = plt)
```

Creates the Q-Q plot and compares the distribution of the residuals to a normal distribution

```
>>> plt.gca().get_lines()[1].set_linestyle('--')
>>> plt.title('Q-Q Plot')
>>> plt.show()
```

Adds aesthetics to the plot

Sets the title

Displays the plot

After running this code, you'll see the Q–Q plot in figure 4.8, which visually indicates the degree of normality in the residuals. In this example, the Q–Q plot shows that the residuals deviate from the line of normality, especially at the tails, suggesting that they may not be normally distributed and may, in fact, have heavier tails or some degree of skewness. Although the Q–Q plot is a useful first check, interpreting it can be subjective. For a more definitive answer, we can apply a Jarque–Bera test, which provides a statistical basis for assessing normality.

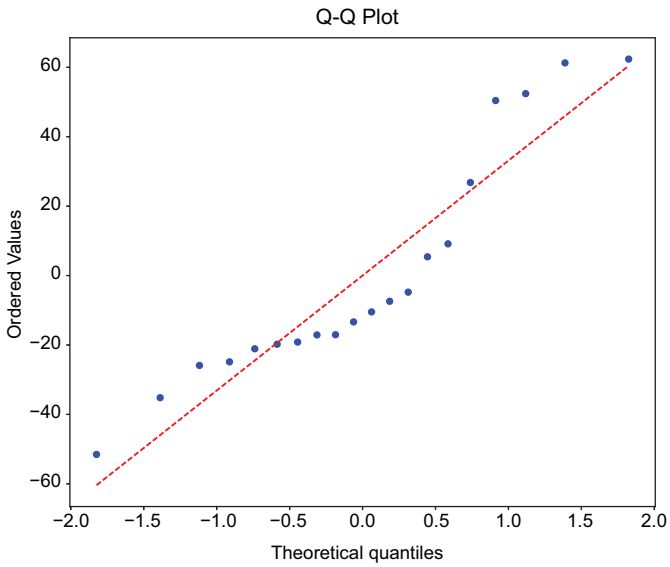

Figure 4.8 A Q–Q plot that displays the distribution of the residuals (points) versus a normal distribution (dashed line). Linear regression assumes that the residuals are normally distributed; however, our Q–Q plot might suggest otherwise.

The Jarque–Bera test calculates its statistic by measuring skewness and kurtosis (i.e., symmetry and tail thickness) in the distribution of the residuals. The null hypothesis is that the data is normally distributed. If the test returns a p-value less than 5%, we reject the null hypothesis and conclude that the residuals are not normally distributed. Here's how to calculate it directly from the residuals:

```
>>> from statsmodels.stats.stattools import jarque_bera
>>> print(jarque_bera(lm.resid))
SignificanceResult(statistic=2.150037983280995, pvalue=0.34129127355827205)
```

In this example, the p-value is 0.341, which of course is above the 5% threshold, allowing us to conclude that the residuals are likely normally distributed—even if the Q–Q

plot might have suggested otherwise. Although the Q–Q plot gives a visual overview, the Jarque–Bera test provides a more rigorous measure of normality.

Finally, it's worth noting that if we were fitting a multiple linear regression, we would need to test an additional assumption: that the predictors are not highly correlated with each other, known as *multicollinearity*. Excessive multicollinearity can lead to overfitting, so reducing the model by removing redundant predictors may be necessary. We'll explore this concept further in the next chapter.

We've demonstrated linear regression in theory and in practice. But just as significantly, we've (hopefully) established a solid foundation for learning other modeling techniques. We'll pivot toward logistic regression next to solve a classification problem.

Summary

- Linear regression is a statistical method used to model the relationship between a numeric dependent variable and one or more independent variables by fitting a linear equation to observed data. It assumes a linear relationship between the independent and dependent variables, with the goal of estimating the parameters of the linear equation to minimize the overall difference between observed and predicted values. Linear regression is commonly used to predict and understand the relationship between variables across multiple domains, including economics, finance, and the social sciences.
- Goodness of fit should be evaluated from multiple measures. R^2, or the coefficient of determination, which equals a number between 0 and 1 that specifies the proportion of variance in the dependent variable explained by the regression, might be the most meaningful measure, but hardly the only measure that matters. Overall fit is determined by the F-statistic. Significance (and insignificance) are fixed by the p-values for the individual coefficients as well as the p-value for the model.
- Testing for model assumptions—linearity between variables, independence, homoscedasticity, and normality of the residuals—warrants the reliability and validity of regression results and therefore further facilitates interpretation and decision-making.
- Applying best practices along the way, like testing for normality in the predictors and removing any and all outliers from the data, contributes considerably to getting the best possible fit and guaranteeing positive results in post-regression tests.