Preview of Chapter 01 DATA

from the

No Bullshit Guide to Statistics

Ivan Savov

December 8, 2023

Contents

Chapter 1

Data

Data is the fuel for statistics. The successful application of statistical analysis procedures depends on the data collection and processing steps that precede them. Our ability to answer scientific and business questions from data is conditional on the way the data collection process was planned and executed, which determines whether we have the necessary type of data to answer the questions we're interested in.

Understanding data is a an essential prerequisite for applying statistics in real-world scenarios. This chapter aims to beef-up your knowledge about data collection (Section 1.1), data processing and visualization (Section 1.2), and data summarization (Section 1.3).

In this chapter, I'm going to show you how to ...

- **classify** the different types of data (numerical vs. categorical)
- **recognize** the importance of random sampling and random assignment
- **load** datasets stored in Comma-Separated Values (CSV) formatted files
- **compute** numerical data summaries (descriptive statistics)
- **generate** strip plots, histograms, box plots and other data visualizations

1.1 Introduction to data

In order to do statistical analysis, we need to have data to analyze. This is why we'll start our journey into statistics by introducing the core concepts of data collection. The more you know about data collection strategies, the better you'll be equipped to apply the statistical analysis techniques that we'll learn in later chapters.

We use statistics to answer questions about real-world phenomena we're interested in. We assume it is possible to collect relevant data through observations and measurements of the quantities of interest. Broadly speaking, the purpose of statistical analysis is to detect and measure the existence of some "pattern" in the data. Statistical analysis can be used to confirm that a predicted pattern exists, to estimate an unknown quantity, or to detect when an unexpected pattern occurs.

In this section, we'll talk about the central importance of data for all of statistical practice. We'll start by introducing the basic definitions and terminology used to describe datasets. We'll then discuss the *random sampling* and *random assignment* techniques used as part of the data collection process.

1.1.1 Definitions

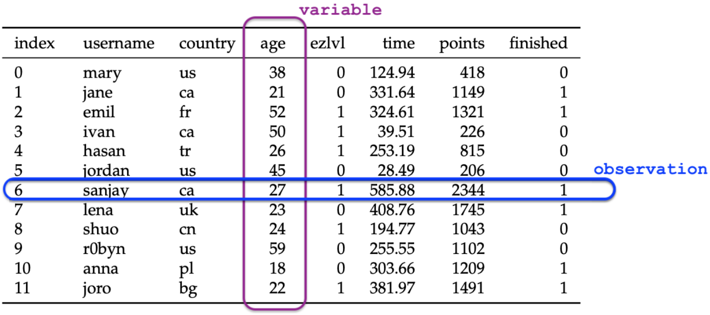
Let's look at the technical terms we use when talking about data.

Datasets

We refer to the data used in a statistical analysis as a *dataset*. In this book, we'll focus on *tabular data*, which is the most widely used kind of data in statistics. We use the following terminology to describe tabular datasets.

- *data table* or *data frame*: describes data stored in tabular format, like a spreadsheet with rows and columns. A *dataset* consists of one or more data tables.
- *variable*: a characteristic of the individual, item, or event that is measured in the data. Variables are also sometimes called *features* or *attributes*. For example, in a health study, the height and weight of individuals would be two variables that are measured.
- *observation*: the measurements for a single individual, item, or event. Observations are also sometimes called *cases* or *observational units*. For example, the measurements collected for one individual in a health study (name, age, height, weight, treatment, outcome, etc.) correspond to one observation.

Table 1.1 shows an example data table that contains 12 observations of seven variables.


variable
observation

index	username	country	age	ezlvl	time	points	finished
0	mary	us	38	0	124.94	418	0
1	jane	ca	21	0	331.64	1149	1
2	emil	fr	52	1	324.61	1321	1
3	ivan	ca	50	1	39.51	226	0
4	hasan	tr	26	1	253.19	815	0
5	jordan	us	45	0	28.49	206	0
6	sanjay	ca	27	1	585.88	2344	1
7	lena	uk	23	0	408.76	1745	1
8	shuo	cn	24	1	194.77	1043	0
9	r0byn	us	59	0	255.55	1102	0
10	anna	pl	18	0	303.66	1209	1
11	joro	bg	22	1	381.97	1491	1

Table 1.1: A data table that contains observations of seven variables for 12 players of a computer game. Each row in this table corresponds to one player. Each column corresponds to one characteristic that was measured for all the players.

In addition to the data, a dataset contains *metadata* (data about the data), which provides additional "context" information, including *when*, *how*, and *why* the data was collected. Metadata usually includes a *codebook* that describes each of the variables, and specifies the units of measurement. Detailed and complete metadata is essential for correct interpretation of the data.

Example: the players dataset Let's illustrate the new terminology by looking at an example dataset of player profiles from a computer game. The players dataset is shown in Table 1.1. This dataset was collected as part of a statistical experiment whose goal was to test if making the first level of the computer game easier will increase the time players spend in the game. Half the players were presented a special version of the game with an easy first level (`ezlvl=1`), while the other half played the normal version of the game (`ezlvl=0`). We want to know if the easy level made players spend more time in the game.

The players dataset contains observations of seven variables for 12 different players. The leftmost column is called the *index* and is equivalent to the row numbers in a spreadsheet. The first row of the table is called the *header* and contains the variable names.

Each row of the dataset shown in Table 1.1 corresponds to one observation (the data for one player). We have recorded the following characteristics for each player: `username`, `country`, `age`, `ezlvl`, `time`, `points`, and `finished`. For example, the player with username sanjay, is a 27-year-old Canadian (ca), who spent 585.88 minutes playing the game, earned 2344 points, and finished the game (finished=1). The value 1 for the ezlvl variable tells us that Sanjay played the game version with the easy first level.

Each column of the data table contains the values of one of the variables that we measured for all players. You can also think of each variable as a list of 12 values. For example, the variable age is a list of 12 values $[38, 21, 52, 50, 26, 45, 27, 23, 24, 59, 18, 22]$, where each value corresponds to the age of one of the players. We can analyze each of the variables on their own (e.g., compute the average age of the players), or look for relations between variables (e.g., how does the ezlvl variable influence the time variable).

Variable types

We make a distinction between *numerical* and *categorical* variables:

- *numerical variables* correspond to quantitative measurements recorded as numbers. Numerical variables can be integers (e.g. age) or decimal numbers (e.g. time).
- *categorical variables* are labels that take on one of a discrete set of possible values, like the answers to true or false questions, the presence or absence of some characteristic (1 or 0), blood group types (A, B, AB, O), or a person's country of residence. The variables username, country, ezlvl, and finished in the players dataset are all categorical variables.

The statistical operations we can perform on numerical and categorical variables are completely different. Numerical variables can be manipulated using arithmetic operations like sums, differences, and products, while categorical variables can only be used for grouping and counting observations.

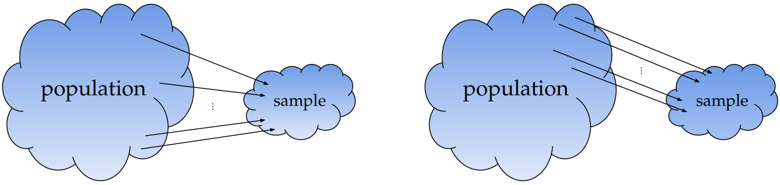
Note that categorical variables are sometimes represented using numbers, such as the values 1 and 0. For example, the variable finished in the players dataset (see Table 1.1) contains 1 for players who finished the game, and 0 for players who didn't finish the game.

Populations and samples

The *population* is the group of interest for the statistical analysis. This term can refer to people (players, students, patients, clients, website visitors, etc.) but also to groups of animals, insects, objects, or events.

- *population*: all the items or individuals in the group of interest. We'll denote the population size (the number of individuals in the population) using uppercase N, which can be in the tens, hundreds, thousands, millions, or billions. Often the population size is unknown.

- *census*: the process of collecting data for the entire population. This kind of exhaustive data collection is usually very costly to perform, so instead, most statistical analyses are performed on a subset of the population called a *sample*.
- *sample*: a subset of the population that has been measured for statistical analysis. We'll denote the sample size as lowercase n. Usually, n is much smaller than the population size N.
- *representative sample*: a sample is representative if it has the same characteristics as the population. See Figure 1.1 (a). For example, if the population contains a mix of people in different age groups, the people included in a representative sample must contain a similar mix of different age groups.
- *biased sample*: samples that are not representative of the population are called *biased*. See Figure 1.1 (b). For example, a sample that contains only young people is not representative of the general population.
- *random sample*: a sample selected from the population in such a way that each individual has an equal chance of being included in the sample. Using random sampling is one way to obtain representative samples.



(a) Representative sample selection

(b) Biased sample selection

Figure 1.1: A *sample* is a subset of the *population* selected for performing statistical analysis. If the sample is representative of the population as in (a), the results of the statistical analysis performed on the sample will also apply to the population as a whole. If the sample is biased as in (b), the results of the statistical analysis will not generalize to the population as a whole.

Variables names used in statistical analyses

The purpose of statistical analysis is to find patterns in the data, to extract scientific knowledge, and reach justified conclusions. Typically, we want to answer a question about how the values of one variable depend on the values of another variable.

In the context of the players dataset, an example of a statistical question we might want to answer is whether young players and old players spend different amounts of time in the game. What is the influence of the `age` variable on the `time` variable in the players dataset?

Statisticians use the following terms when referring to variables, depending on the role they play in the statistical analysis:

- *explanatory variable*: the variable that causes or predicts the different outcomes. Explanatory variables are also called *independent variables*, *predictor variables*, or *treatment variables*.
- *response variable*: the variable of interest that we suspect is influenced by the explanatory variable. Response variables are also called *dependent variables* or *outcome variables*.

In the study of the players dataset, we want to know whether younger or older players spend more time on the game. The explanatory variable is the player's `age`, and the response variable is `time` (the total time they spent playing the game).

In addition to the variables we include in the statistical analysis, there is another category of variables that you need to know about:

- *confounding variable*: a variable that influences both the explanatory and response variables, but is not considered in the study. Another term for this kind of variable is *lurking variable*, since it is hidden from our analysis.

Statisticians generally want to avoid confounding variables, so they carefully consider all factors that could potentially influence the response variable and try to measure them and include them in the statistical analysis.

An example of a confounding variable in the gaming scenario is a player's job status: whether they are currently unemployed or have a full-time job. We can expect that people who are unemployed will have more free time to play the game than people who have a full-time job. Young people tend to be unemployed more often, so they are more likely to have time to play the game. Since the job status variable is not measured, a statistical analysis of the influence of the `age` variable on the `time` variable might lead us to erroneously conclude that the game is more engaging for a younger audience. This conclusion is wrong because of the confounding effect of the job status variable on the relationship between `age` and `time` variables. Young people spend more time in the game not because they like it more, but because of their job status.

If we were to perform the same statistical analysis separately for groups of players with full-time jobs and unemployed players, we would no longer see a relationship between the age variable and the time variable. In statistics jargon, we say that the influence of age on time disappears when we *control for* job status.

Observational studies and statistical experiments

We can broadly subdivide statistical studies into two kinds, depending on the control researchers have over the explanatory variable.

- In a *statistical experiment*, researchers control the explanatory variable and observe its effects on the response variable.
- In an *observational study*, researchers observe the explanatory variable, but can't influence it or manipulate it.

An example of a statistical experiment is the question about the effect of the ezlvl variable on the time variable. The game developer controls whether the players are shown the regular game (ezlvl=0) or the alternative version with an easy first level (ezlvl=1).

An example of an observational study is the question about the effect of the age variable on the time variable. The game developer doesn't select the age variable, but only observes it.

Different types of data

Let's say a few more words about the characteristics of the data for different types of studies.

Data for experimental studies The key characteristic of an experimental study is that we control or choose the explanatory variable. We can subdivide the participants into two groups, depending on the value of the explanatory variable. We use the following terminology to refer to the different subsets of the participants in a statistical experiment:

- *intervention group*: a subset of the participants that received the new treatment or intervention.
- *control group*: a subset of the participants that didn't receive the new treatment or intervention. Ideally, the control group should be similar to the intervention group along all characteristics except for the value of the explanatory variable.

For example, introducing an easy first level of the game is a type of intervention. The subset of the players who were shown the game with an easy first level (ezlvl=1) are in the intervention group. The players who played the regular version of the game are in the control group (`ezlvl=0`). We can assign the participants randomly to the intervention group and the control group, in order to create two groups with roughly identical characteristics. In the players dataset, half the players were randomly assigned to the easy first level version (`ezlvl=1`), and the other half to the normal version of the game (`ezlvl=0`). By comparing the average `time` variable for these two groups, we can determine if the easy level feature increases user retention (the `time` players spend in the game).

Another example of an intervention is showing website visitors two different versions of a web design to determine which version leads to more conversions (sales or sign-ups). This is sometimes called an A/B test, since the intervention group consists of visitors who are shown an *alternative* version `A` of the website, and a control group consisting of visitors who are shown the *baseline* version `B`.

In both the game scenario and the website conversion scenario, we assume that the intervention group and the control group are approximately identical, except for the choice of the explanatory variable. The Latin phrase to describe this assumption is *ceteris paribus*, which means "all other things being equal." If we were to observe some difference in the response variable between the two groups, then we can attribute this difference to the effect of the intervention. In other words, we can make a claim that a *causal* relationship exists between the explanatory variable and the response variable.

Unfortunately, data suitable for a statistical experiment where we actively control the explanatory variable is a luxury—we only have access to this type of data when we collect it for that specific purpose as part of an organized effort. Collecting data for an experimental study where the explanatory variable is randomly assigned is often not possible because of logistics, ethics, insufficient funding, lack of time, or other constraints.

Data for observational studies We often have to content ourselves with *observational studies*, where we only observe the explanatory variable, but can't control it. Observational studies can be done using data that was originally collected for a different purpose or data that is already being collected as part of normal operations. The term "found data" is sometimes used to describe observational data.

For example, the players dataset was collected to study the effect of the `ezlvl` variable on the `time` variable, but the same data can also be used for an observational study of the relation between the `age` variable and the `time` variable.

Observational data can be used to discover *correlations* or *associations* between observed variables. Observational studies can't be used to make conclusions about causation, because of the possible presence of confounding variables. The maxim "correlation does not imply causation" is often cited to describe this fundamental reality.

Case reports A *case report* is a dataset with a single observation. We can't do statistics on case reports, since we only have the measurements for a single individual. Nevertheless, case reports can have scientific value, since they document unexpected outcomes, like the miraculous recovery of a patient suffering from a rare illness, after a particular treatment. We can't conclude that the treatment is responsible for the patient's recovery, but it's still worth recording this observation for posterity. This will enable the data from this case report to be included in later studies of this rare illness.

Data for meta-analyses A meta-analysis is a statistical analysis that combines the results of multiple previous studies of the same phenomenon of interest. Each of the prior studies is based on a different dataset of independent measurements. The purpose of a *meta-analysis* is to compare and combine the results of all these studies to look for a general trend, or provide a more accurate estimate of some quantity of interest. The results of each study have some degree of error that is independent of the other studies, so by "pooling" the results together, we can obtain a more accurate picture of the phenomenon.

Doing meta-analysis is only possible for certain phenomena that are widely studied, leading to an accumulation of data from multiple statistical experiments and observational studies, performed by different researchers, and in different conditions. This cumulative evidence from multiple studies is the strongest type of statistical result we can hope for.

1.1.2 Study design and randomization strategies

The data collection strategy of your study determines the conclusions that you can make as a result of your statistical analysis. In particular, these are two important aspects you need to think about:

- *sampling*: the process by which a sample of individuals is selected from the population. We want to select samples that are representative of the wider population.
- *assignment*: the process by which participants are assigned to intervention and control groups in a statistical experiment. We want the two groups to be as similar as possible so that we can

attribute any observed differences between the groups to the effect of the intervention.

The use of randomness is an essential tool that you have at your disposal for both of these aspects. Indeed, *random sampling* and *random assignment* are the two main "weapons" that statisticians use to obtain meaningful results, despite the pervasive presence of noise and variability in real-world data.

Random sampling

A *random sample* is a sample selected from the population in such a way that each individual has an equal chance of being selected. The hope is that by choosing individuals at random, we'll end up with a sample that is *representative* of the population.

Using representative samples from the population is essential if we want to draw conclusions about the whole population based on observations from a single sample. The technical term for this is *generalization*, which means that the statistical results we obtain from the sample apply more generally to the wider population from which the sample was selected.

The opposite of a representative sample is a *biased sample*. There are many sources of bias that you need to watch out for. We'll now briefly mention some of them. *Exclusion bias* exists when certain participants are excluded from the study for one reason or another. For example, a dishonest researcher could select only participants whose data tend to support a desired conclusion. The selective inclusion of certain observations is sometimes called *cherry-picking*. We use the general term *selection bias* to describe any preference for selecting certain participants over others. There is also the danger of *self-selection bias*, which happens when participants choose to enrol in the study due to a vested interest. Self-selection bias is also called *volunteer bias*. The opposite is called *non-response bias*. *Attrition bias* occurs when participants with negative or adverse effects drop out of the study over time, and their data is not included in the analysis. This is also called *survivorship bias* in the context of medical studies.

An example of a biased sample selection process is the common use of undergraduate psychology students for experiments in psychology. University students tend to fall in a very narrow age range and have similar socioeconomic backgrounds. The results obtained from studies involving student volunteers often do not generalize to the wider population.

A good way to avoid bias is to use a random sampling process: build a list of all the possible candidates for inclusion, and select the sample randomly from this list.

Random assignment

In experimental studies, we seek to establish a *causal relationship* between the explanatory variable and the response variable. In other words, we want to show that a given intervention *causes* certain outcomes to occur. For example, the players dataset is part of an experiment that tests if the easy first level intervention (`ezlvl=1`) causes players to spend more `time` in the game.

In an ideal world, to show a causal effect of an intervention on an outcome variable, we would be able to "clone" every participant in the study to obtain two identical individuals. We can then perform the intervention on one of the clones, while using the other clone as a baseline for comparison. If we observe a difference in the outcome variable between the two clones, then we can be sure this difference is due to the intervention, since, by definition, the two clones are identical on all other aspects and characteristics, except for the intervention variable.

In the real world, since cloning is not a thing yet, we're forced to replace the "identical individuals" requirement with the approximation "identical groups of individuals." Suppose we can partition the participants into two groups that are "roughly" identical along all their aspects and characteristics (age, location, socio-economical background, lifestyle, etc.). We then apply the intervention to one of these groups (the intervention group) and not on the other (the control group). The fancy Latin phrase *ceteris paribus*, which means "all other things being equal," is sometimes used to describe the concept of two groups that are similar in all aspects except for the intervention variable. If we observe a group-level difference between the intervention group and the control group, then we can attribute this difference to the intervention we performed.

The process of *random assignment* is one way to obtain two roughly identical groups: the intervention group and the control group. We can perform the random assignment by tossing a coin for each participant. If the outcome of the coin toss comes out heads, we assign the participant to the intervention group. If the coin toss comes out tails, we assign them to the control group. Assuming the coin is fair, we'll end up with a roughly 50-50 split of the participants into the two groups, as illustrated in Figure 1.2.

We hope that the random assignment process results in two groups that are balanced along all characteristics that might influence the response variable (*ceteris paribus*). We don't have any *guarantee* that the two groups will be identical, but, on average, the two groups are unlikely to have any systematic differences.

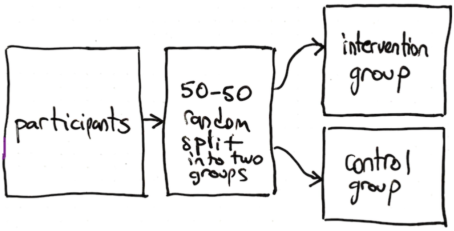


Figure 1.2: The process of random assignment of participants to the intervention and control groups allows us to study causal relationships.

Blinding An additional consideration when assigning participants to the intervention and control groups is the use of *blinding* to prevent knowledge of the treatment from affecting the outcome.

- *single-blinding* aims to ensure that participants don't know which group they are assigned to. In an experiment that tests the effectiveness of a new drug, patients in both intervention and control groups are given a pill, so that they can't tell which group they are part of. Patients in the intervention group receive the drug, while patients in the control group receive a *placebo*, which is a fake pill with no medical effects.
- *double-blinding* aims to ensure that even the researchers administering the study do not know which group the participants are part of. Double blinding aims to avoid researchers consciously or unconsciously biasing the results by treating participants in the two groups differently.

The gold standard for medical research is a *randomized control trial* (RCT), which is based on random assignment of patients to groups and uses double-blinding.

Summary of study design strategies

We can summarize the combined effect of random sampling and random assignment using a two-by-two table, as shown in Table 1.2. The most powerful kind of study is in the top-left corner: a study in which participants are selected using random sampling from the population, and randomly assigned to intervention and control groups. The fact we have selected participants at random allows us to make conclusions that generalize to the population as a whole, while the random assignment procedure allows us to make causal claims.

If a study uses samples that were not randomly selected from the population (bottom row of the table), it is not possible to make conclusions that apply to the whole population. Basically, if participants in the study are self-selected (volunteers), or chosen based on convenience sampling (friends and family), the sample simply won't contain the same diversity and variability as the whole population. In these cases, we can still look for interesting correlations in the data (bottom-right corner), or even uncover cause-and-effect relations (bottom-left corner), but we can't extend our conclusions from the sample to the whole population.

	group assignment	
sample selection	Random assignment	Observational data
Random sampling	Causal relationships that generalize to the whole population.	Correlations that generalize to the population, but the strength of conclusions may depend on confounding variables.
Other sampling	Causal relationships that may not generalize to the whole population.	Correlations that may not generalize to the population.

Table 1.2: The type of conclusions we can draw from a study depend on the sample selection process and the way we assign individuals to intervention and control groups. Random sampling allows us to make generalizations about the population. Random assignment allows us to make causal claims.

In observational studies (the right column in Table 1.2), we don't control the value of the explanatory variable, so we can't make cause-and-effect conclusions. We are limited to finding *correlations* and *associations* in the data. Let's see why this is so. Suppose we observe a strong correlation between the explanatory variable $\mathbf{x}$ and a response variable $\mathbf{y}$. Perhaps we would like to believe this $\mathbf{xy}$-correlation is the result of a causal relationship between $\mathbf{x}$ and $\mathbf{y}$, where the dependence $\mathbf{y}$ on $\mathbf{x}$ is described by some function: $\mathbf{y} = f(\mathbf{x})$. However, the same observed correlation could equally well be explained by a causal-relationship in the opposite direction: a dependence of $\mathbf{x}$ on $\mathbf{y}$ described by some other function: $\mathbf{x} = g(\mathbf{y})$. Since we only observed $\mathbf{x}$ and $\mathbf{y}$ and didn't control them, we can't distinguish these two scenarios. Furthermore, perhaps there exists a confounding variable $\mathbf{z}$ (either observed or lurking), that is the common cause of both $\mathbf{x}$ and $\mathbf{y}$, so the true underlying relations are $\mathbf{x} = g_z(\mathbf{z})$ and $\mathbf{y} = f_z(\mathbf{z})$. In observational studies we can't make any cause-and-effect conclusions like the existence of the functions f, g, g_z, and f_z, since our observations are consistent with all these possibilities. This is an inherent limitation of "found data" and something to be aware of.

Thinking about the assignment and selection procedures is also important when reviewing other people's findings. Whenever you're reading about some statistical result in a research paper, you should ask yourself "Is the sample representative of the population?" and "Was random assignment used?" and mentally place the study in the appropriate row and column of Table 1.2. This will tell you the kind of inferences that are "supported" by this kind of study. Don't expect the authors of the report to tell you! Their knowledge of the logic of statistical analysis may be more limited than yours!

1.1.3 Discussion

Data collection is a broad topic that we can't cover exhaustively. The above sections introduced the core ideas that you *must* know. There are some additional topics that are worth mentioning, so at least you'll have heard about them.

Levels of measurement

Statisticians sometimes further subdivide numerical and categorical variables into four subtypes to capture more precisely the measurement that each variable represents. These subtypes are called *levels of measurement* and can be ordered from least precise to most precise, as in the following list.

- Categorical variables subtypes:
  - ▻ *nominal*: discrete variables that can't be ordered. Each nominal value describes a name, a label, or a category. Examples: city of residence, sex, group membership.
  - ▻ *ordinal*: discrete variables that have a natural order. Examples: Likert scale responses (strongly disagree, disagree, neither agree nor disagree, agree, strongly agree), and star ratings in reviews. Ordinal values can be ranked and compared, but the differences are not quantifiable. For example, we know that a three-star rating is better than a two-star rating, but we don't know that the difference between a three-star rating and a two-star rating is the same as between a two-star rating and a one-star rating.
- Numerical variable subtypes:
  - ▻ *interval*: variables that can be compared numerically using differences, but do not have a natural zero value.

Examples: temperature in Celsius or Fahrenheit. The temperature difference between 10°C and 11°C is the same as the difference between 100°C and 101°C, but the meaning of 0°C is an arbitrary choice.

▷ *ratio*: values can be compared using differences and ratios. Ratio variables have all the characteristics of interval variables, but also have natural zero that corresponds to the absence of the quantity. Examples: points, time, height, weight, temperature in Kelvin. In each of these examples, the value 0 is a useful reference point. It is also meaningful to say than player one scored 30% more points that player two.

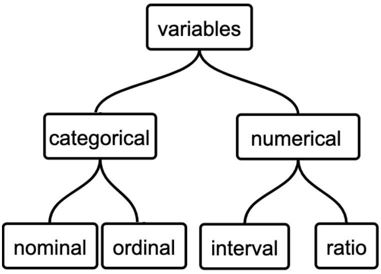


Figure 1.3: Levels of measurement for statistical variables.

The levels of measurement of variables determine the type of statistical analysis we can perform with them.

Statistical analysis in broader context

Let's take a moment to look at the broader context in which statistical analysis fits. The use of statistics is just a tool in the wider "scientific method" framework. Below you'll find a bird's eye overview of the steps required to run a scientific study.

1. **Identify** the population of interest and the scientific question you want to answer. Think about the data you'll need to study this question.

2. **Plan** the study. Will it be an observational study or an experimental study? How will participants be selected? How will data be collected? Is existing data already available? How large should your sample be? Consider the ethics of your study, and solicit input from the people who may be impacted by the data collection and statistical results.

3. **Run** the study. Make sure data is being collected while the study is running.

4. **Process** the data to prepare it for analysis. This is a crucial step that involves extracting data from various sources and transforming it into a form suitable for statistical analysis. We'll talk about data loading and data manipulation in Section 1.2, and discuss data transformation and data cleaning in Appendix D.

5. **Analyze** the data. This is when you finally get to apply the statistical analysis techniques that you'll learn in chapters 3, 4, and 5. It's a good idea to have a colleague or other peer look over the steps of your statistical analysis to make sure they are sound. If your study requires using some particularly tricky statistical analysis technique, it's best to consult with a statistician to confirm that you're applying the technique correctly, and all assumptions are met.

6. **Communicate** your results. This step usually involves writing a report or making a presentation of some sort. Communicating statistical results to non-experts requires using a simplified language, but it's important your simplifications not to be misleading. In an academic research context, this step involves publishing a paper in a scientific journal, which may involve a peer review process. Don't assume the peer review process will correct any mistakes you may have committed, all the "scientific due diligence" should have been done in the previous step. Whether the study is in a business or academic context, it's also your job to amplify its reach, including announcing it on social media, writing blog posts, recording 2 minute explainer videos, or giving 2 hour long lectures.

7. **Preserve** and share the data, metadata, code, publications, and other study outputs. Being a good citizen of the scientific community requires thinking about others who will come after you. You need to record detailed information about how you collected the data for this study, how you processed it, and how you analyzed it, so that other researchers will be able to find your data, reproduce your analysis, and potentially reuse the data you've collected.

8. **Repeat** the process starting back at step 1! A scientific discovery is rarely achieved through a single study. More often, it takes multiple studies and independent replications of the results in order to establish a new scientific theory.

Observe that the actual statistical analysis part (Step 5) is a small component of a whole process. In contrast, data plays a much more central role in all the steps. This is why this book begins with a whole chapter on data, including definitions (this section), hands-on data management practice (Section 1.2), and descriptive statistics (Section 1.3). Your ability to reach interesting statistical conclusions depends on both the "quality" and "quantity" of the data available for your analysis. The more you know about data, the better you'll be equipped to tackle the later chapters.

Data as a singular noun

I'd like to make a final note on terminology. The Latin word *data* is the plural of *datum*. Many people treat *data* as a plural noun in English, using it in sentences like "data are collected" with a plural verb accord. In this book, we'll use *data* as a mass noun and use singular verbs like "data is collected." This would be "incorrect" usage in Latin, but I believe singular verbs make the text easier to read in English, so I've adopted this modern usage.

I'll refer to individual data elements as *observations*, *data items*, *data points*, or *data values*. The word "datum" will not be used at all, because it sounds too fancy to me.

* * *

I hope this introduction to the terminology of data and datasets made sense, and you now understand the importance of random sampling and random assignment for statistical analysis. In the next section, we're staying on the topic of data, but we'll switch gears to talk about practical, hands-on data loading and data processing tasks.

1.1.4 Exercises

E1.1 TODO: Recognize and classify different types of variables / data

E1.2 TODO: Identify potential sources of bias in a given data collection scenario (word problems)

Links

[List of different types of statistical bias]
`https://en.wikipedia.org/wiki/Bias_(statistics)#Types`

[More info about randomized controlled trials]
`https://en.wikipedia.org/wiki/Randomized_controlled_trial`

[Info about the singular and plural usage of the word “data”]
`https://theguardian.com/news/datablog/2010/jul/16/data-plural-singular`

[More info about study design used in the medical domain]
`https://guides.himmelfarb.gwu.edu/studydesign101/`

1.2 Data in practice

We're blessed to be living in the XXIst century when computational tools for data analysis are easily accessible. We don't have to memorize complicated formulas or perform tedious calculations using pen-and-paper, since we can use computational tools like Python, Pandas, and Seaborn to do statistical calculations. By learning a thing or two about the Pandas and Seaborn libraries (which is the goal of this section), you'll know about the best-in-class toolset for data management currently used by data scientists, statisticians, business analysts, and machine learning researchers.

A common misconception about statistics is that it's someone else's job to collect data, and offer it to you in a well organized, clean format ready for statistical analysis. This is far from the truth! In reality, statisticians and other data professionals spend a large proportion of their time collecting, pre-processing, and informally exploring datasets in preparation for doing actual statistical analyses. The topic of practical data management is usually omitted from introductory statistics courses, because teachers think it would be too complicated for beginners to learn. I don't think so, and I plan to teach you the essential skills you need to work with realistic datasets. Specifically, I'm going to show you how to use the JupyterLab computational environment, the Pandas library for data manipulation, and the Seaborn library for generating statistical visualizations.

This is going to be a hands-on, try-things-for-yourself section and not a passive reading section. The main goal of this section is to ensure you have a working computational environment on your computer (JupyterLab Desktop), and know how to use the Pandas and Seaborn libraries for basic data analysis tasks. The secondary goal of this section is to introduce the datasets that we'll use in the remainder of the book. The two goals combine synergetically, since we need examples of datasets to showcase the power of the Pandas and Seaborn functionality.

1.2.1 Getting started with JupyterLab

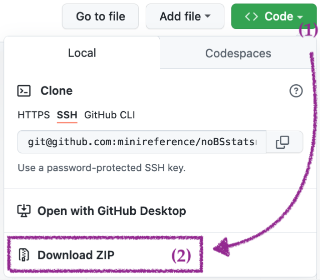
We'll start by setting up a statistical computing environment (JupyterLab Desktop) on your computer, which will allow you to run computational notebooks. You can think of a computational notebook as a fancy calculator—you input Python commands (similar to the buttons on a calculator), then run the commands to see the result (similar to what happens when you press the $\boxed{=}$ button on a calculator). Unlike calculators that have a limited number of operations (buttons), computational notebooks give you access to the entire Python programming language, and numerous powerful Python libraries for data management, data visualization, and statistical analysis.

Download and install JupyterLab

JupyterLab Desktop is a convenient all-in-one application that you can install on your computer to take advantage of everything Python has to offer for data analysis and statistics. Follow the instructions in Appendix C (see page TODO) to download and install JupyterLab Desktop. If you're new to Python, I strongly recommend that you go through the entire Python tutorial in Appendix C before continuing with the rest of this section. I'm not expecting you to be a Python expert, but I want you to be comfortable with the basic Python commands used for calculating expressions, manipulating lists, and calling functions.

Download the notebooks and datasets for the book

I have prepared a collection of notebooks and datasets to accompany this book. You can view and download these notebooks and datasets from the book's website `https://noBSstats.com` or from this GitHub page: `https://github.com/minireference/noBSstats`. Instead of downloading notebooks and datasets one by one, I recommend that you download the entire repository as a ZIP archive using the steps illustrated in Figure 1.4. After downloading the ZIP archive, double-click on the file to extract its contents, and move the resulting folder `noBSstats` to a location on your computer where you normally keep your documents.



```
git@github.com:minireference/noBSstats
```

Use a password-protected SSH key.

Figure 1.4: Illustration of the steps to download the contents of the entire `noBSstats` repository as a single ZIP archive. Use the **Code** dropdown (1) then select the **Download ZIP** option (2).

The ZIP archive includes all the datasets and computational notebooks for the book. Use the **File browser** pane in the JupyterLab to navigate to the location where you saved the `noBSstats` folder. Inside you should see subfolders called `datasets`, `notebooks`, `exercises`, `tutorials`, etc. Look around to get an idea of the files available in each subfolder.

Datasets for the book

You can find all the datasets inside the `datasets` subfolder of the `noBSstats` folder. Use the JupyterLab file browser pane to view the contents of the `datasets` subfolder. For example, the players dataset is stored in the file `datasets/players.csv`.

Alternatively, you can download individual datasets directly from the book's website, under the `datasets` directory. For example, the data file `players.csv` can be downloaded from the URL `https://noBSstats.com/datasets/players.csv`, and similarly for the other datasets.

Later in this section, we'll provide more information about all the datasets in this folder and discuss the statistical questions we want to answer from each of them. Look ahead to Table 1.3 on page 38 if you're feeling impatient.

Interactive notebooks for each section

Each section of this book has a notebook companion that includes the code examples from the text. I expect you to play with these notebooks in parallel with reading the text, so that you'll get some hands-on experience of doing data calculations and generating data visualizations. The notebooks are located in the `notebooks` subfolder. For example, the notebook companion for this section is `notebooks/12_data_in_practice.ipynb`. I recommend that you open this notebook now in JupyterLab, so that you'll be ready to run the code examples you'll encounter later in this section.

Exercises notebooks

I've also prepared starter notebooks for the exercises in each section. The exercises notebooks contain partially-filled code cells for each exercise question. You can find the exercises notebooks in the `exercises` subfolder. For example, to try the exercises for this section, open the notebook `exercises_12_practical_data.ipynb` in the `exercises` folder and start filling in the missing parts in the code cells.

* * *

From here on, I'll assume you have JupyterLab Desktop installed on your computer and have downloaded all the datasets and notebooks for the book, so that you can follow the code examples interactively. Here are a few quick exercises you can try, to make sure you've got the basic setup working correctly.

E1.3 Create a new notebook called `MyCalculations.ipynb` and use code cell to compute the sum of `3456` and `789`.

E1.4 Open the notebook `exercises_12_practical_data.ipynb` located in the `exercises` folder and repeat the calculation `3456+789` in the code cell labelled E1.4.

1.2.2 Data management with Pandas

Pandas is a versatile toolbox for data management in Python. You can think of Pandas as a Swiss Army knife for working with data, since it includes *a lot* of functions for working with various types of data, performing data manipulations, and doing statistical calculations. Learning a bit of Pandas will allow you to work with real-world datasets of all shapes and sizes, so it is a generally useful skill to have if you plan to do anything data-related in the future.

The good news is that you don't need to learn all this functionality at once. Knowing just a few basic Pandas concepts and commands is enough to get you started. This subsection is a Pandas crash course that will introduce you to the two main data structures that the Pandas library provides: *data frame* objects for storing tabular data, and *series* objects for storing lists of values. We'll focus on the specific data manipulation tasks that you need to know to understand the examples in the book. For a more in-depth coverage of Pandas functionality, I'll refer you to the Pandas tutorial in Appendix D.

Okay, enough talk, let's get started! Open the notebook `12_data_in_practice.ipynb` in JupyterLab Desktop so you can run the commands in parallel and follow the explanations interactively. The first step is to import the `pandas` module, which is usually done in the beginning of the notebook. There is a widespread convention to import the `pandas` module under the short alias `pd`.

```
>>> import pandas as pd
```

code 1.2.1

This import statement makes all the Pandas functionality available under the name `pd`.

Loading datasets

The first step to any data analysis is to load the data we want to work on into a Pandas *data frame*. We'll illustrate the process by loading the data file `players.csv` located in the `datasets` directory, which is a sibling the `notebooks` directory.

The file extension `.csv` tells us the file contains text data formatted as Comma-Separated Values (CSV). We can view the contents of the file `players.csv` using a text editor like Notepad.exe on Windows or TextEdit on macOS. The file contents are shown below.

code 1.2.2

```
username,country,age,ezlvl,time,points,finished
mary,us,38,0,124.94,418,0
jane,ca,21,0,331.64,1149,1
emil,fr,52,1,324.61,1321,1
ivan,ca,50,1,39.51,226,0
hasan,tr,26,1,253.19,815,0
jordan,us,45,0,28.49,206,0
sanjay,ca,27,1,350.0,1401,1
lena,uk,23,0,408.76,1745,1
shuo,cn,24,1,194.77,1043,0
r0byn,us,59,0,255.55,1102,0
anna,pl,18,0,303.66,1209,1
joro,bg,22,1,381.97,1491,1
```

We see the data file `players.csv` consists of 13 lines of text, and each line contains—as promised by the `.csv` file extension—values separated by commas. The first line in the data file is called the "header" and contains the names of the variable names.

The Pandas function for loading CSV files is `pd.read_csv(<path>)`, where `<path>` describes the location of the data file. The current notebook is located in the `notebooks` directory, which is a sibling to the `datasets` directory, so the *relative path* to the players dataset is `"../datasets/players.csv"`. In words, this path means "go up to the parent directory (the two dots), then go inside the `datasets` directory, and look for the file named `players.csv`."

The code below shows how to use the function `pd.read_csv` to load the `players.csv` data into a Pandas data frame object called `players`. We intentionally choose a name for the data frame that matches the dataset name to remind us where the data came from. We then print the contents of the variable `players` by entering its name on a second line.

code 1.2.3

```
>>> players = pd.read_csv("../datasets/players.csv")
>>> players
  username country  age  ezlvl    time  points  finished
0     mary      us   38      0  124.94     418         0
1     jane      ca   21      0  331.64    1149         1
2     emil      fr   52      1  324.61    1321         1
3     ivan      ca   50      1   39.51     226         0
4    hasan      tr   26      1  253.19     815         0
```

```
5     jordan    us  45     0   28.49    206      0
6     sanjay    ca  27     1  585.88   2344      1
7       lena    uk  23     0  408.76   1745      1
8       shuo    cn  24     1  194.77   1043      0
9      r0byn    us  59     0  255.55   1102      0
10      anna    pl  18     0  303.66   1209      1
11      joro    bg  22     1  381.97   1491      1
```

Note we didn't have to use the command `print(players)` to display the contents of the `players` data frame, but instead relied on the default behaviour of the notebook environment, which is to print the value of the last expression in a code cell.

Recall we've already seen the players dataset in the previous section (see Table 1.1 on page 3). The players dataset consists of $n = 12$ observations of players' activity in a computer game. The variable `username` is a unique identifier for each player. The variables `country` and `age` provide some basic player demographics. The variable `ezlvl` indicates whether the player was part of the "easy level" experiment. The `time`, `points`, and `finished` variables describe the player's total time in the game, the total points they scored, and whether they finished the game or not.

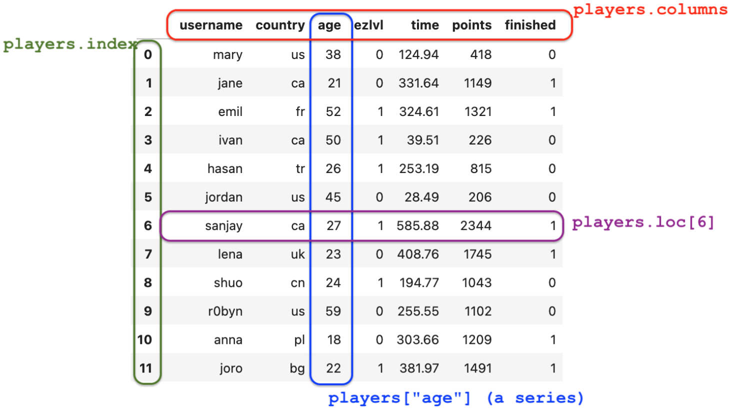
In the remainder of this section, we'll use the `players` data frame to illustrate the various Pandas functions for extracting specific rows and columns from the data frame and performing arbitrary calculations on them.

Data frame properties

A Pandas *data frame* is a container for tabular data organized into rows and columns. Figure 1.5 shows the `players` data frame, and includes extra annotations for the different parts of its "anatomy."

- The *rows* of a data frame contain the individual observations.
- The *index* (`players.index`) contains unique labels that we use to refer to the rows of the data frame.
- The *columns* of the data frame correspond to the different variables. Each column of the data frame is a Pandas series object, which is a list-like container for values.
- The *header* (`players.columns`) contains the variable names.

A Pandas data frame is similar to a spreadsheet—it's a way to store data organized into rows and columns. The attribute `players.columns` contains the names of the variables in the data frame, which is analogous to the letters we use when referring to the different columns in a spreadsheet. The data frame's index, `players.index`, tells us the labels we can use to refer to different



	username	country	age	ezlvl	time	points	finished
0	mary	us	38	0	124.94	418	0
1	jane	ca	21	0	331.64	1149	1
2	emil	fr	52	1	324.61	1321	1
3	ivan	ca	50	1	39.51	226	0
4	hasan	tr	26	1	253.19	815	0
5	jordan	us	45	0	28.49	206	0
6	sanjay	ca	27	1	585.88	2344	1
7	lena	uk	23	0	408.76	1745	1
8	shuo	cn	24	1	194.77	1043	0
9	r0byn	us	59	0	255.55	1102	0
10	anna	pl	18	0	303.66	1209	1
11	joro	bg	22	1	381.97	1491	1

Figure 1.5: The `players` data frame contains 12 observations (rows), and each observation consists of seven variables. Each column of the data frame is a Pandas *series* object that contains the measurements of one variable for all players.

rows within the data frame, which is similar to the numbers we use when referring to rows in a spreadsheet.

Let's use the Python function `type` to confirm that `players` is indeed a data frame object.

code 1.2.4

```
>>> type(players)
pandas.core.frame.DataFrame
```

The above message tells us that `players` is a Pandas `DataFrame` object. Specifically, the `players` object is an instance of the `DataFrame` class that is defined in the module `pandas.core.frame`.

The `players` data frame object has a bunch of useful properties (attributes) and functions (methods) "attached" to it, which we can access using the dot syntax. For example, the `.shape` attribute contains information about the shape of the data frame:

code 1.2.5

```
>>> players.shape
(12, 7)
```

This tells us the `players` data frame has 12 rows and 7 columns.

Let's explore the other attributes and methods of the `players` object. The `.index` attribute of the `players` data frame tells us the labels we use to refer to the rows in the data frame.

code 1.2.6

```
>>> len(players.index)
12
>>> players.index
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

The data frame `players` uses the "default index" consisting of a range of integers from `0` to `11`. The first row corresponds to index

0 and the last row corresponds to index 11, which is the standard 0-based indexing convention used to access the elements of a Python list. Note this is different from the convention used in spreadsheet software, where the first row has label 1. Data frame objects are more flexible than spreadsheets, since they allow us to use arbitrary labels to refer to the rows in the data frame. Instead of using generic row numbers, it's possible to use other labels that uniquely identify the rows in the data frame. In some scenarios, it might be convenient to use one of the columns in the data table as the index. For example, the player's names could be used as the index.

The columns-index attribute .columns tells us the names of the columns (variables) of the data frame.

code 1.2.7

```
>>> len(players.columns)
7
>>> players.columns
['username', 'country', 'age', 'ezlvl', 'time', 'points',
 'finished']
```

This result tells us that the players data frame has seven columns named username, country, age, ezlvl, time, points, and finished. The names of the columns were automatically determined based on the "header" line in the CSV file (see code block 1.2.2). Unlike spreadsheets that force us to use the labels A, B, C, etc. for the column names, data frame objects allow us to refer to the different columns using more descriptive labels.

Column names usually consist of short textual identifiers. Spaces and special characters are allowed in column names, which means you can use column names like "player points" and "finished (0 or 1)". However, using complicated column names makes data manipulation code more difficult to read, so I would generally discourage you from using them. Instead, stick to short, single-word, descriptive labels.

Refer back to the players.index and players.columns highlights in Figure 1.5 for an illustration of the row-index and the columns-index of the players data frame. Note the visual similarity to the way data is represented in a spreadsheet. Essentially, a data frame is just a fancy spreadsheet that allows us to use custom row-labels (players.index) and custom column-labels (players.columns) when referring to the data values.

Exploring data frame objects

When working with datasets with hundreds or thousands of rows, it's not practical to display the entire data frame as we did above. In those situations, we can still "look around" in the data frame by printing the first few rows to inspect what they look like. The data frame method `.head(k)` prints the first `k` rows of a data frame.

code 1.2.8

```
>>> players.head(3)
  username country  age  ezlvl    time  points  finished
0     mary      us   38      0  124.94     418         0
1     jane      ca   21      0  331.64    1149         1
2     emil      fr   52      1  324.61    1321         1
```

We can also use the method `.tail(k)` to print the last `k` rows of the data frame. The method `.sample(k)` selects a random sample of `k` rows from the data frame.

Data types

The `.dtypes` (data types) attribute contains information about the `types` of the values stored in each column of the data frame.

code 1.2.9

```
>>> players.dtypes
username     object
country      object
age           int64
ezlvl         int64
time        float64
points        int64
finished      int64
```

The function `pd.read_csv` automatically determined the data types of the columns when we loaded the CSV file in code block 1.2.3. Pandas looked at the values in each column and chose an appropriate variable type that can accurately represent all the values in that column. The three most common data types that Pandas uses are integers, floating point numbers, and strings. The information in `players.dtypes` tells us that the columns `age`, `ezlvl`, `points`, and `finished` are stored as integers. Pandas chose the default integer type `int64`, which uses 64 bits of memory. The column `time` contains decimals, so Panda uses the default floating point number type `float64` to store this variable. The columns `username` and `country` contain text, so Pandas stores them as generic Python objects (in this case string objects).

For the most part, you don't have to worry about data types, but it's sometimes useful to look "under the hood" and see how Pandas actually stores the values in the data frame. For example, if you see that a numerical variable is stored as `object`, this is a hint that there might be formatting issues in the data file that prevented Pandas from using a numeric data type. Another reason why you might care about data types is if you're working with large datasets with thousands or millions of rows. Calculations on columns that contain integer or floating point numbers will be very fast, since they will be performed by optimized number-crunching code (NumPy), while calculations containing `objects` will be much slower.

Note the Python type of the variable is not the same as the statistical type of the variable: *numerical* or *categorical*. Numerical variables can be stored as integers (`int64` like `age`) or floating point numbers (`float64` like `time`). Categorical variables can be stored as integers (e.g. `ezlvl` and `finished`) or strings (e.g. `country`).

Accessing and selecting data

We use the `.loc[]` selection attribute to access the values at different "`loc`ations" within a data frame. To obtain the value of the `points` variable for the third row (index `2`) in the `players` data frame, we use the expression:

```
>>> players.loc[2,"points"]
1321
```

code 1.2.10

The general syntax is `players.loc[<row>,<col>]`, where `<row>` is the index label and `<col>` is the column label of the value we want to obtain.

Selecting entire rows To select rows from a data frame, we use the syntax `players.loc[<row>,:]`, where `<row>` is a index label and the special symbol "`:`" refers to "all columns."

```
>>> players.loc[6,:]  # == players.loc[6]
username     sanjay
country          ca
age              27
ezlvl             1
time         585.88
points         2344
finished          1
```

code 1.2.11

The alternative syntax `players.loc[<row>]` (without the "`,:`" part) produces the same result as `players.loc[<row>,:]`. Note the `<row>` label we use to refer to a given row in the data frame is its index label (one of the labels in `players.index`).

Selecting entire columns We use the square-brackets syntax `players["<col>"]` to select the variable `<col>` from the `players` data frame. For example, we can extract the values of the `age` variable from the `players` data frame using the following expression:

```
>>> players["age"]
0      38
1      21
2      52
3      50
```

code 1.2.12

```
4     26
5     45
6     27
7     23
8     24
9     59
10    18
11    22
Name: age, dtype: int64
```

The syntax `players["<col>"]` is equivalent to the `.loc[]` selection expression `players.loc[:,"<col>"]`, which selects all rows (“`:`”) in the `"<col>"` column.

Statistical calculations using Pandas

Let’s now focus on the `age` variable in the `players` data frame. The code example below shows how to use the square brackets syntax to extract the `age` variable from the `players` dataset, and store it in the new Python variable called `ages`.

```
>>> ages = players["age"]
>>> ages
0     38
1     21
2     52
3     50
4     26
5     45
6     27
7     23
8     24
9     59
10    18
11    22
Name: age, dtype: int64
```

code 1.2.13

Using the name `ages` to describe the values of the `"age"` column is an example of the general naming convention we’ll use in this book: using the plural of the column name for the name of the Python variable that contains the values extracted from this column.

Let’s use the `type` function to check what kind of object is the variable `ages`.

```
>>> type(ages)
pandas.core.series.Series
```

code 1.2.14

The variable `ages` is a Pandas series object. Pandas series are list-like containers of values. The Pandas series `ages` has the same index as the `players` data frame, and it “remembers” the name of the column from which it was extracted.

```
>>> ages.index
```

code 1.2.15

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> ages.name
'age'
```

Pandas series objects have many methods for doing statistical calculations. For example, the method `.count()` tells us the length of the series:

```
>>> ages.count()
12
```

code 1.2.16

The method `.count()` is analogous to the function `COUNT(...)` in a spreadsheet, where the ... refers to a spreadsheet range expression.

The method `.sum()` computes the sum of the players' ages.

```
>>> ages.sum()
405
```

code 1.2.17

The Python expression `ages.sum()` is equivalent to calling the spreadsheet function `SUM(...)` on the range of cells that contain the age values.

We can combine the results of the above two expressions to calculate the *average value* (the arithmetic mean) of the players' ages. The average value of a list of n values $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is computed using the formula $\overline{\mathbf{x}} = \frac{1}{n}(x_1 + x_2 + \cdots + x_n)$. This formula says that the average is computed by summing together all the values in the list $\mathbf{x}$ and dividing by the length of the list n. The average is denoted with a bar on top of the variable name $\overline{\mathbf{x}}$. The expression for computing the average age using Pandas methods is as follows:

```
>>> ages.sum() / ages.count()
33.75
```

code 1.2.18

This Python expression is equivalent to the spreadsheet formula `SUM(...)/COUNT(...)`, where ... is the range that contains the ages.

An equivalent, more direct, way to compute the arithmetic mean of the values in the series `ages` is to call its `.mean()` method.

```
>>> ages.mean()
33.75
```

code 1.2.19

Calling the method `.mean()` is equivalent to using the function `AVERAGE(...)` in a spreadsheet.

The *standard deviation* (dispersion from the mean) is another common statistic that we might want to calculate for a variable in a dataset. To find the sample standard deviation of the values in the series `ages`, we call its `.std()` method:

```
>>> ages.std()
14.28365244861157
```

code 1.2.20

The calculation `ages.std()` is equivalent to using the function `STDEV(...)` in a spreadsheet.

Pandas series and data frames objects have numerous other methods for computing numerical data summaries including: `.min()`, `.max()`, `.median()`, `.var()`, `.quantile()`, etc. We refer to these collectively as the *descriptive statistics* of a variable. We defer the detailed discussion on descriptive statistics until the next section (Section 1.3). See Table 1.4 on page 58 for a complete list of the Pandas methods available for computing descriptive statistics.

Selecting only certain rows (filtering)

A common task when working with Pandas data frames is to select the rows that fit one or more criteria, which is equivalent to "filtering out" rows that don't satisfy these criteria. We usually select rows using a two-step procedure:

Step 1: Build a "selection mask" series that consists of boolean values (`True` or `False`). The mask series contains the value `True` for the rows we want to keep, and the value `False` for the rows we want to filter out.

Step 2: Select the subset of rows from the data frame using the mask. The result is a new data frame that contains only the rows that correspond to the `True` values in the selection mask.

For example, let's say that we want to select the rows from the `players` data frame where `ezlvl` is `1`. The first step is to create the selection mask (Step 1):

code 1.2.21

```
>>> mask = players["ezlvl"] == 1
>>> mask
0     False
1     False
2      True
3      True
4      True
5     False
6      True
7     False
8      True
9     False
10    False
11     True
```

The double equal sign is equivalent to asking the question "Which rows of the `players` data frame have the value `1` in the `"ezlvl"` column?" The rows that match the criterion "`ezlvl` equal to `1`" correspond to the `True` values in the mask, while the other values are `False`.

The actual selection (Step 2) is done by using the mask inside the square brackets.

code 1.2.22

```
>>> players[mask]
   username country  age  ezlvl    time  points  finished
2      emil      fr   52      1  324.61    1321         1
3      ivan      ca   50      1   39.51     226         0
4     hasan      tr   26      1  253.19     815         0
6    sanjay      ca   27      1  585.88    2344         1
8      shuo      cn   24      1  194.77    1043         0
11     joro      bg   22      1  381.97    1491         1
```

The result is a new data frame that contains only rows where the `ezlvl` variable has the value `1`.

We often combine the two steps of the selection procedure into a single Python expression `players[players["ezlvl"]==1]`, which produces exactly the same result, but avoids the need for creating an intermediate mask variable.

code 1.2.23

```
>>> players[players["ezlvl"]==1]
   username country  age  ezlvl    time  points  finished
2      emil      fr   52      1  324.61    1321         1
3      ivan      ca   50      1   39.51     226         0
4     hasan      tr   26      1  253.19     815         0
6    sanjay      ca   27      1  585.88    2344         1
8      shuo      cn   24      1  194.77    1043         0
11     joro      bg   22      1  381.97    1491         1
```

Note the data frame name appears twice in the combined selection expression: the inner `players` variable creates the mask, while the outer `players` variable is where we extract the data from. We'll use this type of expression often in the remainder of the text, whenever we want to select the rows that match some criterion.

Sorting

We can sort the rows of the data frame based on the values of the variable <var> by calling the method `.sort_values("<var>")`. For example, to sort the players data frame by the `time` variable in descending order, we use the following command.

code 1.2.24

```
>>> players.sort_values("time", ascending=False)
   username country  age  ezlvl    time  points  finished
7      lena      uk   23      0  408.76    1745         1
11     joro      bg   22      1  381.97    1491         1
6    sanjay      ca   27      1  350.00    1401         1
1      jane      ca   21      0  331.64    1149         1
2      emil      fr   52      1  324.61    1321         1
10     anna      pl   18      0  303.66    1209         1
9     r0byn      us   59      0  255.55    1102         0
4     hasan      tr   26      1  253.19     815         0
8      shuo      cn   24      1  194.77    1043         0
0      mary      us   38      0  124.94     418         0
3      ivan      ca   50      1   39.51     226         0
5    jordan      us   45      0   28.49     206         0
```

We specified the option ascending=False because the default behaviour of .sort_values is to sort in increasing order. Note the index in the sorted data frame is no longer in order, since the rows are now sorted by time. The sorted-by-time ordering allows us to see that lena is the player with the most points, and jordan has the least points.

* * *

In this section, we illustrated the most common Pandas data manipulation commands, but the Pandas library provides a lot more functionality. For a more in-depth reference of the Pandas functions, you can read the Pandas tutorial in Appendix D. You don't need to become a Pandas expert to understand the code examples in this book, but if you invest an hour or two going through the Pandas tutorial, you'll learn lots of cool data management techniques that will come in handy when working on real-world projects.

Pandas exercises

I highly recommend you try these exercises, because the hands-on approach is the best way to learn to use Pandas.

E1.5 Open the file players.csv using LibreOffice or another spreadsheet program. Compute the mean and the standard deviation of the age variable using spreadsheet functions.

Hint: Create new cells containing formulas based on the spreadsheet functions AVERAGE(...) and STDEV(...).

E1.6 Compute the mean of the points variable in the players dataset.

E1.7 Try loading a few of the other datasets into a data frame using the function pd.read_csv(), then use the .head() method to print the first few rows of each dataset, and .shape to display the number of rows and columns.

E1.8 Load dataset students.csv and compute the mean of the variable score.

E1.9 Create a new notebook cell and use the command ?ages.sum to display the Pandas help menu for the .sum method, which describes all the options you can use when calling the method, and often includes usage examples. Using the question mark prefix ?ages.sum is a is shortcut for calling help(ages.sum). Try using the ?-prefix to view the help menus of the other methods we used in this section: ages.count, ages.mean, ages.std, etc.

1.2.3 Data visualizations with Seaborn

Seaborn is a popular Python library for statistical data visualization. Seaborn provides functions for generating *strip plots*, *scatter plots*, *box plots*, *histograms*, and other statistical plots for data stored in Pandas series and data frames. In this section, we'll look at some examples of statistical visualizations of the players dataset to give you a taste of the type of plots we can generate using the Seaborn library.

To use Seaborn, the first step is to import the seaborn module in the current notebook.

```
>>> import seaborn as sns
```

code 1.2.25

Importing seaborn under the alias sns is a widespread convention, similar to the convention of importing pandas under the alias pd.

Strip plot of the time variable

A *strip plot* is a statistical visualization for numerical variables where each observation is represented as a point. The Seaborn function for drawing strip plots is stripplot. We'll now use this function to generate a strip plot of the time variable in the players data frame.

To generate a strip plot, we pass the data frame players as the data argument to the Seaborn function sns.stripplot, and specify the column name "time" (in quotes) as the x argument.

```
>>> sns.stripplot(data=players, x="time")
The result is shown in Figure E.2.
```

code 1.2.26

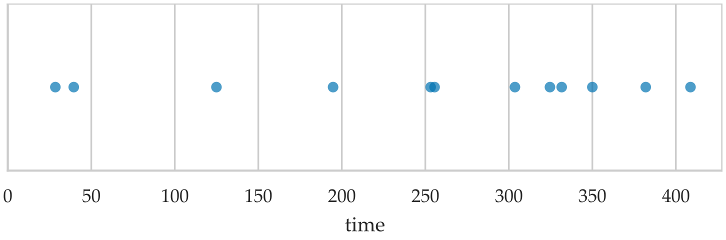


Figure 1.6: Strip plot of the time variable from the players dataset.

The first argument, data=players, tells Seaborn to take the data from the players data frame. The second argument, x="time", indicates we want to represent the time variable on the *x*-axis. This is the general pattern for calling all Seaborn plot functions: we describe where the data lives and the properties of the plot we want to see, and the Seaborn plot function takes care of all the rest. In this case, the function stripplot extracted the data from the "time" column of the players data frame, automatically chose the limits of the x-axis so the data will fit, and set the x-axis title based on the variable name.

Seaborn makes it easy to map multiple variables to different visual properties (aesthetics) of the plot. For example, we can enhance the strip plot by mapping the ezlvl variable to the colour (hue) of the points in the plot.

```
>>> sns.stripplot(data=players, x="time", hue="ezlvl")
Result is shown in Figure 1.7.
```

code 1.2.27

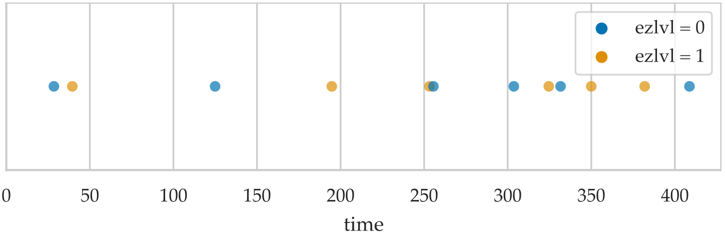


Figure 1.7: Strip plot of the time variable, where the colour of each point is determined by the ezlvl variable in the players dataset.

The addition of the argument hue="ezlvl" tells Seaborn to choose the colour of the points based on the ezlvl categorical variable.

Studying the effect of ezlvl on time

Recall the players dataset was collected as part of an experiment designed to answer the question "Does the easy first level lead to improved user retention?" We want to compare the time variable (total time players spent in the game) of players who were shown the "easy level" version of the game (ezlvl=1) to the control group of players who played the regular version of the game (ezlvl=0).

Figure 1.8 shows a strip plot that can help us visualize the time variable for the two groups of players. The code we used to generate this figure is as follows.

```
>>> sns.stripplot(data=players, x="time", y="ezlvl",
                  hue="ezlvl", orient="h", legend=None)
Result is shown in Figure 1.8.
```

code 1.2.28

Note we were able to customize the plot by passing different arguments and options to the function sns.stripplot. The arguments data=players and x="time" are the same as what we saw earlier. Next we tell Seaborn to use the ezlvl variable for the y-position and the hue of the points in the plot. The options orient="h" (horizontal layout) and legend=None (remove legend) perform additional visual customizations of the plot.

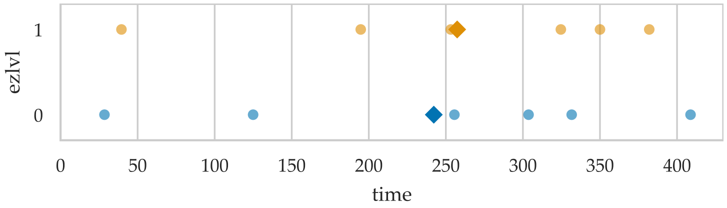


Figure 1.8: Comparison of the `time` variable in the `players` data grouped by `ezlvl`. The diamond shapes represent the means of the two groups.

The strip plot in Figure 1.8 includes additional diamond annotations that correspond to the means of the two groups, which we computed using the following Pandas expressions:

code 1.2.29

```
>>> players[players["ezlvl"]==0]["time"].mean()
242.17333333333332
>>> players[players["ezlvl"]==1]["time"].mean()
257.34166666666664
```

We see there is a difference in the average time spent in the game between the two groups, but this difference is very small compared to the variability in the `time` variable. The strip plot in Figure 1.8 makes this clear. In summary, this means that our experiment is inconclusive: we can't say if the easy level version leads to improved engagement, given how small the observed difference is.

Studying the relationship between `age` and `time`

The secondary research question for the `players` dataset is to look for an association between the `age` variable and the `time` variable. Do young players spend more time in the game?

We can use a *scatter plot* (`sns.scatterplot`) to visualize the relationship between two numerical variables. To use the function `sns.scatterplot`, we have to specify the variables we want to use as the `x` and `y` coordinates of the points:

code 1.2.30

```
>>> sns.scatterplot(data=players, x="age", y="time")
See Figure 1.9 (a).
```

The scatter plot in Figure 1.9 (a) seems to suggest there is an overall trend of the `time` variable to decrease as the `age` variable increases.

When studying the relationship between two numerical variables, we can use a *linear regression model* to describe how one variable depends on the other. In this context, the linear regression model corresponds to the line of best fit that passes through the points in the scatter plot, and we obtain this line using the Seaborn function `sns.regplot`.

code 1.2.31

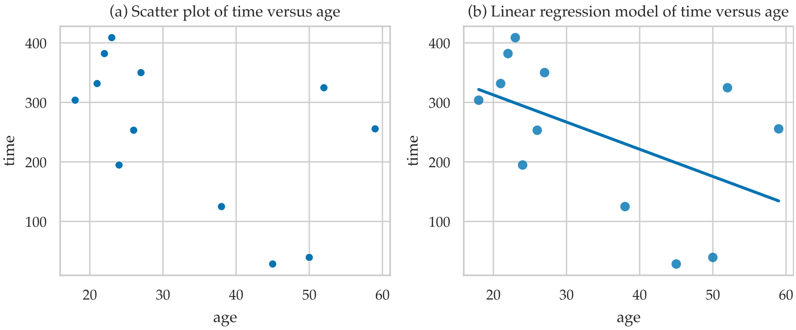


Figure 1.9: Visualizations of the relationship between the `age` variable and the `time` variable. The right panel shows the best fit *linear model* for the relationship between the two variables.

```
>>> sns.regplot(data=players, x="age", y="time", ci=None)
See Figure 1.9 (b).
```

The slope of the best fit line confirms our initial observation about an overall trend of `time` decreasing with `age`. Note however that the variability of the observations around the linear model is very large, so we shouldn't put too much trust in this model. We'll learn more about linear models in Chapter 4.

To be continued...

We'll be using Seaborn plot functions to visualize data, probability distributions, and statistical models throughout the rest of the book, so you'll have plenty of time to get to know the Seaborn functions. See Table E.1 on page 688 in Appendix E for a complete list of the Seaborn plot functions.

For now, the only thing you need to remember is the general syntax that Seaborn plot functions expect:

```
sns.<plotname>(data=players, x="var1", y="var2", hue="var3"),
```

where the first argument, `data=players`, tells Seaborn to look for the data stored in a data frame `players`, and the arguments `x`, `y`, `hue` determine which variables (columns of the data frame `players`) will be represented on the x-axis, the y-axis, and the colour of the plot.

Seaborn exercises

E1.10 Create a strip plot of the variable `age` from the `players` dataset.

1.2.4 Real-world datasets

Imagine you're a data scientist consulting with various clients. Clients come to you with datasets and real-world questions they want to answer using statistical analysis. Table 1.3 shows the complete list of the datasets that we'll use in examples and explanations in the rest of the book. The last column of the table tells us the sections of the book where each dataset will be discussed.

index	client name	filename	shape	sections
		`players.csv`	12x7	1.1, 1.2
1	Alice	`apples.csv`	30x1	3.1, 3.2
2	Bob	`eprices.csv`	18x2	3.1, 3.5
3	Charlotte	`students.csv`	15x5	1.3, 3.1, 3.5, 4.1
4	Khalid	`kombucha.csv`	347x2	3.1, 3.2, 3.3, 3.4
5	Dan	`doctors.csv`	224x4	3.1, 3.2, 3.5, 4.1
6	Vanessa	`visitors.csv`	2000x3	3.6
		`minimal.csv`	5x4	Appendix D

Table 1.3: List of the real-world datasets we'll use throughout the book.

Because we'll be spending a considerable amount of time with these datasets, it's worth knowing the context around each dataset, and trying to understand the statistical question that each client is interested in answering.

Dataset 1: Apple weights

Alice runs an apple orchard. She collected a sample from the apples harvested this year (the *population*) and sent you the data in a CSV file called `apples.csv`. You start by loading the data into Pandas and looking at its characteristics.

```
>>> apples = pd.read_csv("../datasets/apples.csv")
>>> apples.shape
(30, 1)
>>> apples.head(3)
   weight
0   205.0
1   182.0
2   192.0
```
code 1.2.32

The `apples` dataset contains $n = 30$ observations of the `weight` variable. The weights are measured in grams.

You decide to generate a strip plot in order to visualize the distribution of the apple weights.

```
>>> sns.stripplot(data=apples, x="weight")
Result is shown in Figure 1.10.
```
code 1.2.33

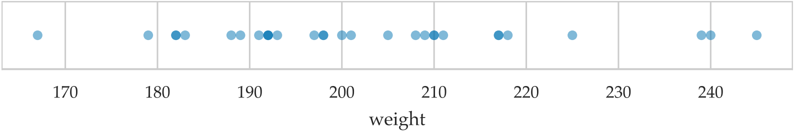


Figure 1.10: Strip plot of the `weight` variable from the apples dataset.

You also compute the average weight of the apples in this sample.

```
>>> apples['weight'].mean()
202.6
```

code 1.2.34

The mean of the apple weights from this sample is 202.6 grams.

Alice's estimation question Alice wants to know the average apple weight in the population. The sample mean 202.6 g is an approximation to the population mean, so that is a good place to start. But how good is this approximation? Alice is asking you to quantify the accuracy of this estimate by constructing a *confidence interval* for the population mean, which is a range of numbers that includes the plausible values.

To answer Alice's question, we'll learn how to model the *sampling distribution* of the mean (Section 3.1) and construct a *confidence interval* for the population mean (Section 3.2).

Dataset 2: Electricity prices

Bob recently bought an electric car. He doesn't have a charging station for his car at home, so he goes to public charging stations to recharge the car's batteries. Bob lives downtown, so he can go either to the East End or West End of the city for charging. He wants to know which side of the city has cheaper prices. Are electricity prices cheaper in the East End or the West End of the city?

To study this question, Bob collected electricity prices of East End and West End charging stations from a local price comparison website and provided you the prices in the dataset `eprices.csv`.

```
>>> eprices = pd.read_csv("../datasets/eprices.csv")
>>> eprices.shape
(18, 2)
>>> eprices
     loc   price
0    East    7.7
1    East    5.9
2    East    7.0
.    ....    ...      # six more rows
10   West   10.0
11   West   11.0
12   West    8.6
.    ....    ...      # six more rows
```

code 1.2.35

Bob's dataset contains 18 observations of the variables `loc` (location, East or West) and `price` (electricity price in ¢/kWh).

You start by generating a strip plot of the `price` variable, using the `loc` variable to control the *y*-position and colour of the points.

```
>>> sns.stripplot(data=eprices,x="price",y="loc",hue="loc")
Result is shown in Figure 1.11.
```

code 1.2.36

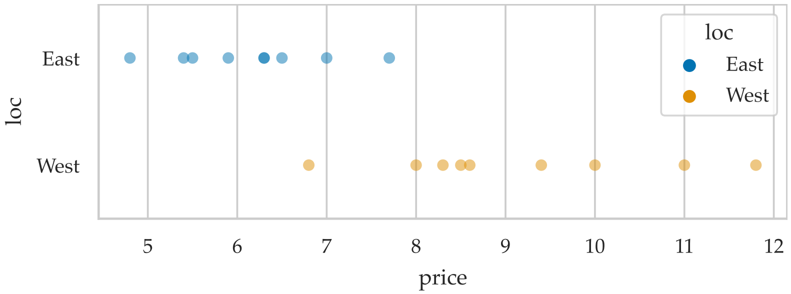


Figure 1.11: Strip plot of the prices in the East End and the West End.

Figure 1.11 seems to show that prices in the West End are higher than the East End. You next calculate the average price for each location.

```
>>> eprices[eprices["loc"]=="West"]["price"].mean()
9.155555555555557
>>> eprices[eprices["loc"]=="East"]["price"].mean()
6.155555555555556
```

code 1.2.37

The average price in the East is 6.156 ¢/kWh, while the average in the West is 9.156 ¢/kWh. Based on a comparison of these averages, it seems East End electricity prices are lower, but could the observed difference be due to chance?

Bob's question Bob is asking for your help with "running the stats" needed to determine if the observed difference in prices is *statistically significant*, which is one of the possible conclusions we can reach when using the *hypothesis testing* procedure.

We'll learn about hypothesis testing in Chapter 3 and discuss the specific hypothesis testing procedures for comparing two groups in Section 3.5.

Dataset 3: Students effort and scores

Charlotte is a science teacher who wants to test the effectiveness of a new teaching method in which material is presented in the form of a "scientific debate." Student actors initially express "wrong" opinions, which are then corrected by presenting the "correct" way to think about science concepts. This type of teaching is in contrast to the usual lecture method, in which the teacher presents only the correct facts.

To compare the effectiveness of the two teaching methods, she has prepared two variants of her course:

- In the `lecture` variant, the video lessons present the material in the usual lecture format that includes only correct facts and explanations.
- In the `debate` variant, the same material is covered through video lessons in which student actors express multiple points of view, including common misconceptions.

Except for the different video lessons, the two variants of the course are identical: they cover the same topics, use the same total lecture time, and test students' knowledge using the same assessment items.

The students dataset consists of activity obtained from the online learning platform that Charlotte used for the course. You load the data file `students.csv` into Pandas, and print the first few rows to see what the data looks like.

code 1.2.38

```
>>> students = pd.read_csv("../datasets/students.csv")
>>> students.shape
(15, 5)
>>> students.head()
   student_ID background curriculum  effort  score
0           1       arts     debate   10.96   75.0
1           2    science    lecture    8.69   75.0
2           3       arts     debate    8.60   67.0
3           4       arts    lecture    7.92   70.3
4           5    science     debate    9.90   76.1
```

The dataset contains information for 15 students enrolled in the course. Charlotte has provided you with the following *codebook* of information about the five variables recorded for each student:

- `student_ID`: a unique identifier for each student
- `background`: describes the student's academic background
- `curriculum`: which version of the course they took
- `effort`: the total time spent on the online learning platform
- `score`: the final grade for the course

We can generate a strip plot of the `score` variable for the two versions of the `curriculum` variable using the following Seaborn command.

code 1.2.39

```
>>> sns.stripplot(data=students, x="score", y="curriculum",
                  hue="curriculum")
Result is shown in Figure 1.12.
```

We can also compute the means for two versions of the curriculum.

code 1.2.40

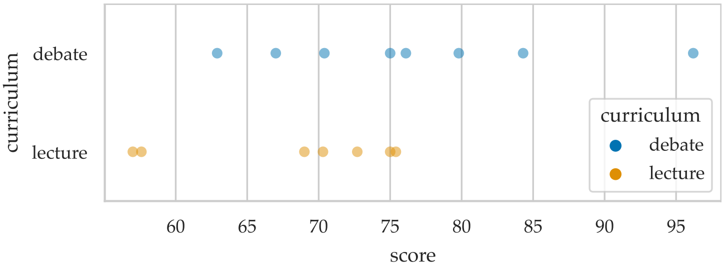


Figure 1.12: Strip plot of the `score` variable for the two versions of the `curriculum` variable.

```
>>> lscores = students[students["curriculum"]=="lecture"]
>>> lscores["score"].mean()
68.14285714285714
>>> dscores = students[students["curriculum"]=="debate"]
>>> dscores["score"].mean()
76.4625
```

We see the average score of students who took the course with the `debate`-style video lessons is higher than the average score of students in the usual `lecture`-style video lessons.

Charlotte's research questions Similar to Bob's question about the electricity prices, Charlotte wants to know if the observed difference in scores between the `lecture` and `debate` curriculum variants is statistically significant. In Section 3.5, we'll use the hypothesis testing procedure for comparing two groups to answer this question.

Charlotte also has a secondary research question about the relationship between the `effort` and `score` variables. Do students who spent more time on the learning platform get better final scores? In Chapter 4, we'll learn about linear regression models and try to find the best fit line for this relationship.

Dataset 4: Kombucha volumes

Khalid is responsible for the production line at a kombucha brewing company. He needs to make sure the volume of kombucha that goes into each bottle is exactly 1 litre (1000 ml), but because of day-to-day variations in the fermentation process, production batches may end up with under-filled or over-filled bottles. Sending such *irregular* batches to clients will cause problems for the company, so Khalid wants to find a way to detect when the brewing and bottling process is not working as expected.

Khalid compiled the dataset `kombucha.csv`, which contains the volume measurements from samples taken from 10 different production batches, and sent it to you for analysis. You load the dataset into Pandas and start poking around, to see the data it contains.

code 1.2.41

```
>>> kombucha = pd.read_csv("../datasets/kombucha.csv")
>>> kombucha.shape
(347, 2)
>>> kombucha.columns
['batch', 'volume']
>>> kombucha.head(3)
     batch   volume
0        1  1016.24
1        1   993.88
2        1   994.72
```

Each observation in the kombucha dataset tells you the `volume` of kombucha measured in one bottle and which `batch` it came from.

Let's generate a combined strip plot of the observations from the different batches so that we can visually inspect the data.

code 1.2.42

```
>>> sns.stripplot(data=kombucha, x="batch", y="volume")
Result is shown in Figure 1.13.
```

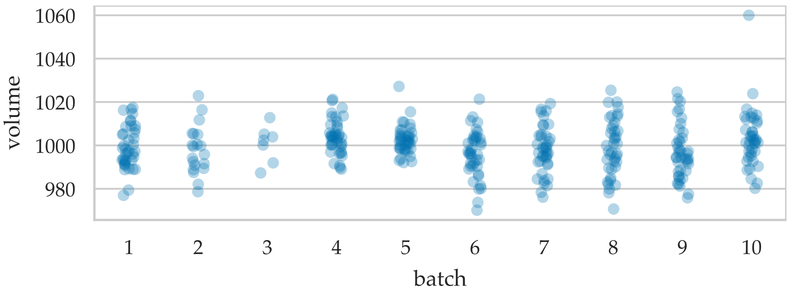


Figure 1.13: Strip plots of the `volume` variable for the ten batches in the kombucha dataset. The volume in each bottle is supposed to be 1000 ml, but we see the data contains a lot of variability around this value.

Looking at Figure 1.13, you can already see several interesting facts about the different batches. The sample from Batch 3 seems to have fewer observations than the other batches. Many of the observations from Batch 4 are above the 1000 ml line, so this batch could be one of the irregular batches (over-filled bottles). Batch 10 contains an outlier observation, that is *waaaay* above any of the other volume measurements. Could this be a measurement mistake? Can you even fit 1060 ml in the bottle? You make a mental note to ask Khalid about this outlier, so you'll know what to do with it when you start the statistical analysis.

Next you decide to extract the data from Batch 1 and compute the mean volume of that sample.

code 1.2.43

```
>>> batch01 = kombucha[kombucha["batch"]==1]
>>> ksample01 = batch01["volume"]
>>> ksample01.mean()
999.10375
```

The value 999.10 is pretty close to the expected value 1000 ml, but how can we tell if this is a regular batch or an irregular batch?

Khalid's quality control question Khalid is asking you to figure out a way to detect irregular batches based on samples of volume measurements. Recall, a batch is deemed "irregular" if the average volume in each bottle is too low or too high. The statistical machinery of *hypothesis testing* is exactly the tool we need for this quality control scenario. In Section 3.3, we'll learn how to analyze the data from the different batches and determine which batches are regular and which are irregular.

Dataset 5: Doctors' sleep study

Dan is a data analyst working at the Ministry of Health. His current assignment is to look for ways to improve the health of family doctors. He collected the doctors dataset (`doctors.csv`), which contains data about the demographics, life habits, and health metrics of 224 family doctors that Dan randomly selected from the populations of family doctors in the country.

code 1.2.44

```
>>> doctors = pd.read_csv("../datasets/doctors.csv")
>>> doctors.shape
(224, 4)
>>> doctors.head(3)
   permit            name location  score
0   93636   Yesenia Smith    urban   82.0
1   79288  Andrew Stanley    rural   85.0
2   94980  Jessica Castro    rural   97.0
```

The columns contain the following information for each doctor:

- `permit`: a unique identifier.
- `name`: the doctor's name.
- `location`: the location of doctor's practice (rural or urban).
- `score`: the sleep score (out of 100)

  TODO: add other columns (as needed for Chapter 4): age, experience (years), exercise, EtOH consumption (standard drink/week), cigs, pot, pills (meds), bmi, ?

Dan is interested in comparing the sleep scores of doctors in rural and urban locations, so he starts by generating a strip plot of the `score` variable for the two values of the `location` variable.

code 1.2.45

```
>>> sns.stripplot(data=doctors, x="score", y="location",
                  hue="location")
Result is shown in Figure 1.14.
```

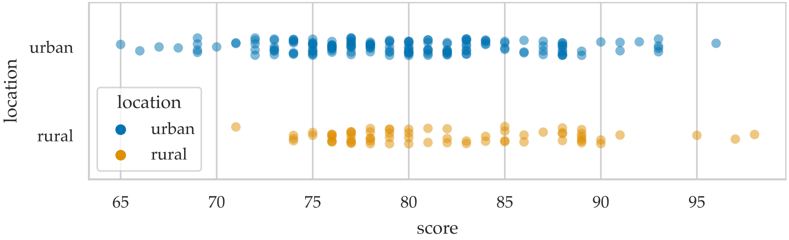


Figure 1.14: Strip plot of the sleep `score` variable for doctors in urban and rural locations.

You also compute the average sleep score for two groups of doctors.

code 1.2.46

```
>>> udoctors = doctors[doctors["location"]=="urban"]
>>> udoctors["score"].mean()
79.57051282051282
>>> rdoctors = doctors[doctors["location"]=="rural"]
>>> rdoctors["score"].mean()
81.79411764705883
```

Dan's research questions Dan's main question is whether doctors working in a rural setting have better sleep. Similar to Bob's question about the electricity prices and Charlotte's study of the student scores, the goal of his statistical analysis is to determine if the observed difference between the two groups is statistically significant, that is, to rule-out the possibility that it occurred by chance.

Dan also has a secondary research question, about the influence of alcohol consumption (`EtOH` variable) on the sleep `score` variable. We'll approach this secondary question in Chapter 4 by fitting a *multiple linear regression model* that captures the effect of several variables on the sleep scores.

Dataset 6: Website visitors conversion rates

Vanessa runs an e-commerce website and is about to launch a new design for the homepage. She wants to know if the new design is better or worse than the current design. Vanessa has access to the server logs from her website and is able to collect data about which visitors clicked the BUY NOW button and bought something. The term *conversion* is used when a visitor buys something, meaning they are "converted" from visitor to client. The *conversion rate* is the proportion of website visitors that become clients.

Vanessa performed an experiment to check if the new website design is better than the current design when it comes to getting visitors to click the BUY NOW button. For the 2000 new visitors that the site received during the previous month, Vanessa randomly sent half of them to the new design (A for alternative), and the other half to the old design (B for baseline). She also recorded if a visitor bought a product during their visit to the website.

The data consists of 2000 observations from visitors to the website from unique IP addresses. For each visitor, the column `version` contains which design they were presented with, and the column `bought` records whether the visitor purchased something or not. You use the usual Pandas commands to load the dataset `visitors.csv`, to inspect its properties, and print the first few rows.

code 1.2.47

```
>>> visitors = pd.read_csv("../datasets/visitors.csv")
>>> visitors.shape
(2000, 3)
>>> visitors.head(5)
        IP address version  bought
0      135.185.92.4       A       0
1       14.75.235.1       A       1
2    50.132.244.139       B       0
3   144.181.130.234       A       0
4       90.92.5.100       B       0
```

We can also compute the average (mean) conversion rate for the two versions of the website.

code 1.2.48

```
>>> visitors[visitors["version"]=="A"]["bought"].mean()
0.06482465462274177
>>> visitors[visitors["version"]=="B"]["bought"].mean()
0.03777148253068933
```

The old design has a conversion rate of 0.0377 or 3.8%. The new design's conversion rate is 0.0648 or 6.5%. The difference between the conversion rates is $6.5 - 3.8 = 2.7\%$. We can also represent the same information by generating a bar plot.

code 1.2.49

```
>>> sns.barplot(data=visitors, x="bought", y="version")
Result is shown in Figure 1.15.
```

Vanessa's question Vanessa wants to know if the new design for the landing page will generate more sales. It seems the new design has increased the conversion rate by 2.7%, but could this observed difference have occurred by chance? Vanessa wants you to perform the necessary statistical analysis to make sure that the observed difference is not due to chance. In Section 3.7, we'll learn about

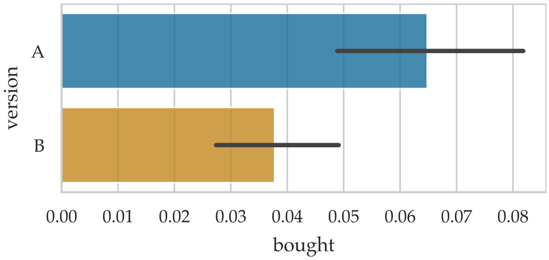


Figure 1.15: Bar plot of the conversion rates of the two versions of the website. The black lines represent the uncertainty of the two estimates.

hypothesis tests for the comparison of two proportions, which will help us answer Vanessa's question.

Statistical analysis types

We'll now classify the different questions that the clients are looking to answer according to the type of statistical analysis task they represent. This will also give a chance to review some of the key data concepts like random assignment and random sampling and that we introduced in the previous section (Section 1.1).

Alice's question about the apples dataset is an *estimation* task. She collected a sample of 30 apples from the population (all apples in this year's harvest) and she wants to estimate the average weight in the population, based on the weights of the apples in the sample. We already calculated the sample mean 202.6 g, which is an estimate for the population mean. We can also compute a *confidence interval* which quantifies the uncertainty in our estimate of the population mean. We'll learn more about estimates in Section 3.1, and discuss procedures for constructing confidence intervals in Section 3.2.

Now let's think about Khalid's question and the kombucha dataset. He obtained samples from different production batches, and he wants to implement a quality control process to detect "irregular" production batches. The statistical analysis technique we'll use to help Khalid is called the *one-sample hypothesis test*, which we'll discuss in Section 3.3.

Observational studies and statistical experiments The other client's questions involve the relationship between two variables. We want to study the effect of an *explanatory variable* on the *response variable*. Recall the distinction between *observational studies* and *statistical experiments* that we made in the previous section. Which of the datasets are observational in nature, and which are experiments?

Bob's electricity prices comparison is an observational study. He *observed* the values of the `loc` variable (East or West) and the `price` variable for the charging stations, but didn't control or choose the values of the `loc` variable.

In contrast, Charlotte's study of the influence of the `curriculum` variable on the `score` variable is a statistical experiment. She chose the value of the predictor variable (`curriculum`) when she randomly assigned students to the `debate` and `lecture` versions of the course. If we see the average score of students who took the `debate` curriculum is higher than the average score of students who took the `lecture` curriculum, then we can reasonably conclude that the `debate` curriculum is better. Note the strength of this conclusion depends on the *ceteris paribus* assumption (all other things being equal). The reason why Charlotte randomly assigned students to the two versions of the curriculum (`lecture` or `debate`) was in order to create two groups that are "roughly identical" except for the choice of the `curriculum` variable.

Charlotte's secondary question about the influence of the `effort` variable on the `score` variable is observational in nature, since she had no control over the `effort` variable. If we observe that higher effort is correlated to higher scores, we can't conclude that effort *caused* the higher scores, we can only say that there is a positive association between these two variables.

Dan's doctors dataset is also observational in nature. He didn't take identical groups of doctors and send them to work in rural and urban settings, but just observed their location and their sleep scores.

Vanessa's A/B test is an experiment. She used *random assignment* to decide which version of the homepage design each visitor saw, so if we observe an improvement in the conversion rate of the new design, we can attribute this improvement to the new design.

The classification into *observational* and *experimental* studies determines the type of conclusion we can make as a result of the statistical analysis. Assuming the random assignment procedure they used resulted in roughly identical groups (*ceteris paribus*), Charlotte and Vanessa can conclude there is a *causal* link between the predictor variable and the response variable. The statistical conclusions that Bob and Dan can make based on their datasets are only about *associations*.

Generalization To what extent do the results we obtain from the samples generalize to the population as a whole? We know that the observations in Alice, Khalid, and Dan's datasets were randomly selected from their respective populations, so there is a good chance they are representative of the population as a whole. This means that estimates and conclusions we make based on the samples are likely to generalize to the wider population.

The case for Bob's electricity prices data is less clear, since we don't know if the prices listed on the price comparison website are a random sample from all charging stations, or a biased sample. Therefore, we have no guarantee the results we obtain will generalize to the whole city.

Charlotte's students dataset consists of observations from a particular group of students, so we can't automatically assume that her findings will apply to all students. That being said, it is fair to assume that students who took the class this semester are similar to students who will take her class in future semesters, and in this sense, her findings will likely generalize in the future.

The situation is similar for Vanessa's dataset, which measures the behaviour of visitors to her website during the past month. The generalizability of her results is based on the assumption the visitors during future months will be similar, which may or may not be true. For example, seasonal events like holidays and the back-to-school rush might attract different demographics of website visitors, for which the results might not apply.

Note we haven't done any statistical analysis on the datasets yet, but we can already tell what kind of conclusions we'll be able to draw based on the data provided by each client! This is super important to understand, and one of the main takeaway messages from this chapter: **the data you start with determines the statistical conclusions you can make**.

1.2.5 Discussion

Before we move on from the topic of data management, I want to mention some important data pre-processing tasks that you need to know about.

Data extraction

The first step of any statistical analysis is to get your hands on the data. This step usually involves loading data stored in local files, downloading data from the web, or extracting data from a database (most common in a business context). You can also collect the data yourself (e.g. through scientific measurements or surveys).

In Appendix D, we'll discuss the different possible data sources (local files, internet files, databases, etc.) and the data formats (CSV, TSV, spreadsheets, HTML, JSON, SQL, etc.). Each data source scenario requires a different set of commands for loading the data, so it doesn't make sense to learn all these commands in advance. Instead, I recommend that you learn about the specific data extraction procedures as needed, on a case-by-case basis.

Data transformations

The "raw" data extracted from a data source often needs to undergo several *data transformation* steps before it is ready for statistical analysis. Data transformation steps include relabelling (changing index labels or column names), renaming of values, data merging (combining multiple data files into a single data frame), and data reshaping (changing the way data is organized into rows and columns). The Pandas library provides functions for doing such data transformation steps.

Data pre-processing steps are sometimes called *data wrangling* or *data munging*, and are often the most time-consuming part in the life of data professionals (data scientists, statisticians, analysts, machine learning practitioners). You can think of data pre-processing as the "manual labour" steps you need to do before you can use the data for statistics. You can learn more about these pre-processing and data transformations steps in Appendix D.

Tidy data

The concept of *tidy data* is a convention for organizing datasets that makes statistical calculations and visualizations easy to perform. A data frame is organized according to the *tidy data* format[Wic14] if it has the following characteristics:

- Each column contains the values for one variable.
- Each row contains the values for one observation.
- Each data cell contains a single value.

This specific organization of data into rows and columns makes it easy to perform statistical calculations on arbitrary subsets of the data, and allows us to create Seaborn plots by simply specifying column names, as we saw in the examples earlier in this section.

The structure of the players dataset displayed in Figure 1.5 follows the *tidy data* format, since each column contains measurements of a different variable, each row contains the data for a different player, and each value is a single measurement.

Let's now look at an example dataset that is not tidy. Bob initially provided you the electricity prices dataset as the data file `epriceswide.csv`, which is organized in a two-columns format:

code
1.2.50

```
>>> epriceswide= pd.read_csv("../datasets/epriceswide.csv")
>>> epriceswide.shape
(9, 2)
>>> epriceswide
   East  West
0   7.7  11.8
1   5.9  10.0
2   7.0  11.0
3   4.8   8.6
4   6.3   8.3
5   6.3   9.4
6   5.5   8.0
7   5.4   6.8
8   6.5   8.5
```

The data frame `epriceswide` doesn't follow the *tidy data* convention, since each row contains multiple observations—one value from the East End and one value from the West End. Datasets obtained through manual data entry are often in this "wide" format, since it's convenient for humans to record values for different groups in different columns.

When you received this data, your first step was to *reshape* the data to transform it into tidy format. You used the Pandas method `.melt` to convert the `epriceswide` data frame from "wide" format into "long" format, with one observation per row. The method `.melt` takes the argument `var_name` to specify the name of the variable that is encoded in the column positions, and the argument `value_name` to specify the name of the variable stored in the individual cells.

code 1.2.51

```
>>> epriceswide= pd.read_csv("../datasets/epriceswide.csv")
>>> epriceswide.melt(var_name="loc", value_name="price")
     loc  price
0   East    7.7
1   East    5.9
2   East    7.0
.. 12 more rows ..
15  West    8.0
16  West    6.8
17  West    8.5
```

The `.melt` operation transformed the implicit "which column is the data in" information into an explicit `loc` variable stored in a separate column. Each row in the transformed data frame contains only a single observation, so it is in tidy data format. Indeed, it is by saving the result of this `melt` command that we obtained the data file `eprices.csv`, which we used in the code examples earlier on.

See the section "Reshaping data frames" (Section D.0.5) in the Pandas tutorial (Appendix D) to learn more about the `.melt` operation.

Data cleaning

It would be a mistake to assume that each value in the dataset is ready for statistical analysis. More often, a *data cleaning* step is required, which aims to correct the following two common problems:

- *missing values* occur when no data has been observed for a given variable. Missing values are a fact of life, since data collection is not a perfect process. For example, a survey responder could have skipped a question, which means we have no answer for that question in the data. Missing values are often recorded as `NaN` (not a number), which is a special `float` object that represents the absence of a numerical value. Missing values can also be denoted as the Pandas symbol `<NA>` (not available), as empty strings `""`, or as special values like `"No answer"` in different contexts.
- *outliers* are particular values of a variable that are inconsistent with other observed values. Human errors during the data entry process are a frequent cause of outliers. For example, if a researcher records a patient's weight in pounds instead of kilograms, the value of the weight variable for that patient would need to be corrected. Outliers values can also occur as a result of equipment malfunction.

It's on you to decide how to handle missing values and outliers in the datasets you plan to analyze. Appendix D contains useful practical advice for dealing with missing values and outliers using Pandas functions. We'll also discuss missing values and outliers several more times in the remainder of the book. In the next section (Section 1.3), we'll describe some methods for *outlier detection* based on descriptive statistics, and later on in the book (Chapter 2) we'll also learn how to use probability models to detect outliers.

Learning on the job

Data management and data visualizations are essential skills that will come in handy for all kinds of data analysis tasks. In this section, we saw the basic operations we can perform using the Pandas and Seaborn libraries, but there is a lot more! We could spend hundreds of pages describing the numerous Pandas and Seaborn functions, and we would still only be scratching the surface of what we can do with these libraries. If you want to dig deeper, read the Pandas tutorial in Appendix D and the Seaborn tutorial in Appendix E.

I placed the in-depth discussion of Pandas and Seaborn functions in appendix, because this is a book about statistics, and we have lots of statistics topics waiting for us! I highly recommend that you read Appendix D and Appendix E and play with the notebooks `pandas_tutorial.ipynb` and `seaborn_tutorial.ipynb` at some point, but you don't need to learn all the details right now.

Instead, you can just keep reading the book and learn about Pandas and Seaborn functions "on the job" through the just-in-time explanations that we provide for all the code examples in the remainder of the book.

1.2.6 Exercises

It's now your time to play with the Pandas and Seaborn libraries by solving the following exercises. It's important for you to try running the commands for yourself, so you'll get some experience with the various `pd.` and `sns.` functions. Remember that you can use the notebook `exercises_12_practical_data.ipynb` as a starting point for your answers.

E1.11 Load each of the following datasets and compute the mean of the specified variable. **a)** effort in students.csv; **b)** score in doctors.csv; **c)** ... .

E1.12 Select subsets of rows for different groups and compute the mean in each group. **a)** effort in students.csv for students in the two curriculum variants; **b)** score in doctors.csv for doctors in rural and urban locations; **c)** ... .

Links

TODO: import from Appendix D and E: 1. best one or two tutorials on pandas and seaborn 2. one article about tidy data

[More info about data cleaning in the Pandas tutorial]
`https://nobsstats.com/tutorials/pandas_tutorial.html#data-cleaning`

[Detailed info about the datasets used in this book]
`https://nobsstats.com/datasets/`

1.3 Descriptive statistics

The goal of descriptive statistics is to characterize the essential properties of a dataset. We use numerical and graphical summaries to describe important aspects of datasets. Computing descriptive statistics is an essential first step in any data analysis, and a fundamental skill that you'll need throughout the book.

We can obtain a condensed summary of data by calculating certain representative values called *summary statistics*. A summary statistic is a numerical value computed from the data that succinctly describes a particular characteristic of the data, like the minimum, maximum, or the average. We'll use the methods of the Pandas library to compute summary statistics for data stored in Pandas series and data frames.

We can also get an overall impression of any dataset by making a *visual summary*: a plot that shows the characteristics of the data. We briefly introduced strip plots and scatter plots in the previous section. In this section we'll revisit these types of plots, and show other statistical visualizations that we can create using the Seaborn library, like bar plots and box plots. By mapping characteristics of data onto visual elements of a graph, we can get a quick overview of the dataset. The human visual cortex is surprisingly efficient at spotting patterns and trends in data presented graphically, so it's worth learning how to create visual summaries to take advantage of your innate pattern-spotting abilities.

In this section, we'll introduce the fundamentals of descriptive statistics, including definitions and general principles, showing relevant formulas, and providing illustrative examples based on the Pandas and Seaborn libraries. The descriptive statistics and data visualizations for numerical and categorical variables are very different, so we'll discuss them separately, starting with numerical variables first.

1.3.1 Numerical variables

Numerical variables describe quantities like weight, length, temperature, and time. The values of numerical variables can be compared, sorted, added together, subtracted, and included in other math operations.

Definitions and formulas

Let's start by defining the new terms and showing the formulas for calculating summary statistics. We'll state the definitions in terms of a generic sample of size n, denoted $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_n]$. You can think of **x** as the measurements of the variable x collected from n individuals. Note the convention to use the boldface symbol **x** to denote the sample as a whole.

Measures of central tendency One way to summarize the values $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_n]$ is to find a single number that represents the "centre" of the distribution of the values. There are several different ways to describe the "centre" of a list of numerical values:

- **Mean**: the *arithmetic mean* is the average value of the data. The mean is computed by taking the sum of all the values divided by the sample size:

$$\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x}) = \tfrac{1}{n}(x_1 + x_2 + x_3 + \cdots + x_n) = \tfrac{1}{n}\sum_{i=1}^{n} x_i.$$

  Note the shorthand notation for the mean uses a bar on top of the variable name. The symbol $\sum$ (capital Greek letter *sigma*) stands for summation, and the math expression $\sum_{i=1}^{n} x_i$ means "sum of all the values x_i from x_1 until x_n."

- **Med**: the *median* is the middle value in the dataset, when the values are sorted. Half the values x_i in the dataset are smaller than the median $\mathbf{Med}(\mathbf{x})$, and half the values are larger than $\mathbf{Med}(\mathbf{x})$, as shown in Figure 1.16.

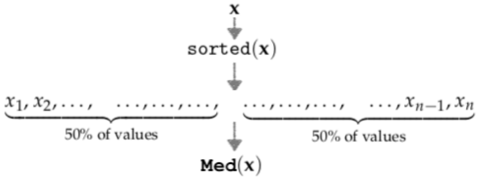

x
sorted(x)
$x_1, x_2, \ldots,\ \ldots, \ldots, \ldots,\ \ldots, \ldots, \ldots,\ \ldots, x_{n-1}, x_n$
50% of values
50% of values
Med(x)

Figure 1.16: Illustration of the median value, $\mathbf{Med}(\mathbf{x})$, which splits the dataset into two equal parts. Half the values in the dataset **x** are smaller than the median, and the other half are larger than the median.

- **Mode**: the *mode* is the most frequently observed value in the data. A variable can have no mode when no single value appears more often than any other, or it can have more than one mode when there is a "tie" for the most common value.

Consider, for example, the sample $\mathbf{x} = [1, 1, 2, 3, 93]$ of size $n = 5$. The mean of **x** is $\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x}) = \frac{1}{5}(1 + 1 + 2 + 3 + 93) = \frac{100}{5} = 20$, the median is $\mathbf{Med}(\mathbf{x}) = 2$, and the mode is $\mathbf{Mode}(\mathbf{x}) = 1$.

Measures of position We often want to describe the position of particular values within the dataset $\mathbf{x}$, when it appears in sorted order. The median $\mathbf{Med}(\mathbf{x})$ is the value in the middle of the dataset. In addition to the median, there are several other useful statistics for describing values at specific positions within the dataset.

- **Min**: the *minimum* is the smallest value in the data.
- **Max**: the *maximum* is the largest value in the data.
- $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$: the three *quartiles* divide the data into four equal parts, which is similar to how the median $\mathbf{Med}(\mathbf{x})$ divides the data into two equal parts. You can think of $\mathbf{Q}_1(\mathbf{x})$, $\mathbf{Q}_2(\mathbf{x})$, and $\mathbf{Q}_3(\mathbf{x})$ as "fence posts" that divide the data into four quarters, as illustrated in Figure 1.17. Note $\mathbf{Q}_2(\mathbf{x})$ is the same as $\mathbf{Med}(\mathbf{x})$.

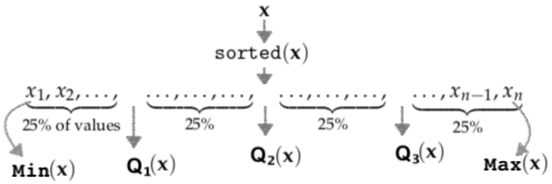


Figure 1.17: Illustration of the four quartiles $\mathbf{Q}_1(\mathbf{x})$, $\mathbf{Q}_2(\mathbf{x})$, and $\mathbf{Q}_3(\mathbf{x})$ that split the sorted data into four equal parts. Note $\mathbf{Q}_2(\mathbf{x}) = \mathbf{Med}(\mathbf{x})$.

- *Percentiles*: the percentiles are similar to the quartiles, but divide the data into 100 equal parts instead of four parts. For example, the 95th percentile is denoted $\mathbf{P}_{95}(\mathbf{x})$ and describes a value that is greater than 95% of the values in $\mathbf{x}$.
- *Quantiles*: the q^{th} quantile splits the data into two parts: a fraction q of the data is smaller, and the remaining fraction $1 - q$ of the data is larger. Quantiles are similar to percentiles, but are defined in terms of a fraction q between 0 and 1, while percentiles use a percentage value between 0 and 100.

The three measures of position, quartiles, percentiles, and quantiles, all provide the same information but use different units. Quartiles describe the data split into four chunks, percentiles use 100 chunks, while quantiles use a continuous quantity between 0 and 1. For example, the first quartile $\mathbf{Q}_1(\mathbf{x})$, is the same as the 25th percentile, which is the same as the $q = 0.25$ quantile. The second quartile $\mathbf{Q}_2(\mathbf{x}) = \mathbf{Med}(\mathbf{x})$ is equivalent to the 50th percentile and the $q = 0.5$ quantile. The third quartile $\mathbf{Q}_3(\mathbf{x})$ is equal to the 75th percentile and the 0.75th quantile.

Taken together, the five numbers $\mathbf{Min}(\mathbf{x})$, $\mathbf{Q}_1(\mathbf{x})$, $\mathbf{Q}_2(\mathbf{x})$, $\mathbf{Q}_3(\mathbf{x})$, and $\mathbf{Max}(\mathbf{x})$ are called the *five-number summary* of the data, which tells us the boundaries of four regions that each contain 25% of the data when it appears in sorted order.

Measures of dispersion Another important characteristic of any dataset is how "spread out" it is, which we call the *dispersion* of the data. There are several common measures for quantifying the dispersion of a dataset.

- **Range**: the *range* of a data is the difference between the maximum and minimum: $\mathbf{Range}(\mathbf{x}) = \mathbf{Max}(\mathbf{x}) - \mathbf{Min}(\mathbf{x})$.
- **IQR**: the *interquartile range* is defined as the distance between the first and third quartiles, $\mathbf{IQR}(\mathbf{x}) = \mathbf{Q}_3(\mathbf{x}) - \mathbf{Q}_1(\mathbf{x})$, and tells us the width of the middle fifty percent of the data.
- **Var**: the sample *variance* is computed from the sum of the squared differences from the mean divided by $n-1$:

  $$\mathbf{Var}(\mathbf{x}) = \tfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mathbf{x}})^2.$$

  We use the shorthand notation $s_{\mathbf{x}}^2$ to describe the variance. Note the formula contains a division by $(n-1)$ instead of n, which is called *Bessel's correction*. We'll explain the reason for using Bessel's correction in Section 3.1 where we'll learn how to use the sample variance to estimate the variance of the wider population from which the sample was taken.
- **Std**: the *standard deviation* is the square root of the variance:

  $$\mathbf{Std}(\mathbf{x}) = \sqrt{\mathbf{Var}(\mathbf{x})} = \sqrt{\tfrac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mathbf{x}})^2}.$$

  We use the shorthand notation $s_{\mathbf{x}}$ for the standard deviation.

The variance and the standard deviation are used often in statistics formulas and procedures, this is why statisticians use the shorthand notation $s_{\mathbf{x}}^2$ and $s_{\mathbf{x}}$ for these quantities. The variance is the square of the standard deviation, so they essentially measure the same thing. We usually show standard deviation when reporting results because standard deviation is measured in the same units as the data, unlike the variance, which is measured in squared units.

Knowing the mean $\overline{\mathbf{x}}$ and the standard deviation $s_{\mathbf{x}}$ of the data $\mathbf{x}$ is a very good way to summarize its distribution. The mean tells us where the centre of the distribution is, while the standard deviation tells us how tightly or loosely dispersed the data is around the mean. Many values will fall within the interval $[\overline{\mathbf{x}} - s_{\mathbf{x}}, \overline{\mathbf{x}} + s_{\mathbf{x}}]$, which describes one standard deviation around the mean. We sometimes write this interval as $\overline{\mathbf{x}} \pm s_{\mathbf{x}}$. See Figure 1.20 for an illustration.

Pandas methods When the data $\mathbf{x}$ is stored in a Pandas series object or a column in a Pandas data frame, we can compute all descriptive statistics by calling the appropriate method on the Pandas object. Table 1.4 shows the Pandas methods for computing all the descriptive statistics for numerical variables that we defined in this section. For example, if the data $\mathbf{x}$ is stored as a Pandas series `xs`, we can compute its mean by calling `xs.mean()`.

Statistic	Name	Pandas method
n	sample size	`.count()`
$\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$	mean	`.mean()`
$\mathbf{Med}(\mathbf{x})$	median	`.median()`
$s_{\mathbf{x}}^2 = \mathbf{Var}(\mathbf{x})$	variance	`.var()`
$s_{\mathbf{x}} = \mathbf{Std}(\mathbf{x})$	standard deviation	`.std()`
$\mathbf{Min}(\mathbf{x})$	minimum	`.min()`
$\mathbf{Q}_1(\mathbf{x})$	first quartile	`.quantile(0.25)`
$\mathbf{Q}_2(\mathbf{x}) = \mathbf{Med}(\mathbf{x})$	second quartile	`.quantile(0.50)`
$\mathbf{Q}_3(\mathbf{x})$	third quartile	`.quantile(0.75)`
$\mathbf{P}_{90}(\mathbf{x})$	90th percentile	`.quantile(0.90)`
$\mathbf{Max}(\mathbf{x})$	maximum	`.max()`

Table 1.4: Summary of descriptive statistics for numerical variables and the Pandas methods for computing them. The same Pandas methods are available on both series and data frame objects.

The Pandas method `.quantile(q)` computes the q$^{\text{th}}$ quantile, where `q` takes on values between 0 and 1. We use the `quantile` method to compute quartiles and percentiles, as shown in Table 1.4. In fact, the minimum and maximum values can also be computed using the `quantile` method: the minimum corresponds to `.quantile(q=0)`, while the maximum is `.quantile(q=1)`.

Descriptive statistics of the students dataset

Enough with the definitions and formulas! Let's look at a hands-on example that illustrates how to compute summary statistics and visualize numerical variables using Pandas and Seaborn. Recall Charlotte's students dataset, which we first introduced in Section 1.2 (see page 40 for the backstory). Table 1.5 shows a complete listing of the students dataset.

We start by importing the `pandas` module under the alias `pd`, then use the function `pd.read_csv` to load the students dataset from the file `datasets/students.csv` into a data frame called `students`.

```
>>> import pandas as pd
>>> students = pd.read_csv("../datasets/students.csv")
```

code 1.3.1

student_ID	background	curriculum	effort	score
1	arts	debate	10.96	75.0
2	science	lecture	8.69	75.0
3	arts	debate	8.60	67.0
4	arts	lecture	7.92	70.3
5	science	debate	9.90	76.1
6	business	debate	10.80	79.8
7	science	lecture	7.81	72.7
8	business	lecture	9.13	75.4
9	business	lecture	5.21	57.0
10	science	lecture	7.71	69.0
11	business	debate	9.82	70.4
12	arts	debate	11.53	96.2
13	science	debate	7.10	62.9
14	science	lecture	6.39	57.6
15	arts	debate	12.00	84.3

Table 1.5: The students dataset contains 15 observations of five variables.

The students dataset contains both numerical and categorical variables, which makes it suitable for use in all the examples in this section. The small number of observations ($n = 15$) makes it possible to show the details of the math calculations.

In this subsection, we'll focus on the numerical variable score in the players dataset, which corresponds to the students' final scores. We'll use a combination of Pandas methods and Seaborn plot functions to describe the distribution of students' scores. Let's start by extracting the score variable from the students data frame and storing the data as a new variable called scores (a Pandas series):

code 1.3.2

```
>>> scores = students["score"]
>>> scores
[75.0, 75.0, 67.0, 70.3, 76.1, 79.8, 72.7, 75.4,
 57.0, 69.0, 70.4, 96.2, 62.9, 57.6, 84.3]
```

In the above code, students is a data frame object, and the syntax students["score"] selects the "score" column from the students data frame. Note we're following the naming convention for series extracted from a data frame to use the plural of the variable name. We'll refer to the scores variable using the shorthand $\mathbf{s}$ in math equations, and denote individual student scores as s_i.

The number of observations, n, is the most basic summary statistic. In this case, $n = 15$. We can call the .count() method on the scores series to find the number of observations it contains.

code 1.3.3

```
>>> scores.count()
15
```

Let's sort the score values in increasing order.

code 1.3.4

```
>>> scores.sort_values()
[57.0, 57.6, 62.9, 67.0, 69.0, 70.3, 70.4, 72.7,
 75.0, 75.0, 75.4, 76.1, 79.8, 84.3, 96.2]
```

Looking at the sorted list of scores allows us to identify some important summary statistics.

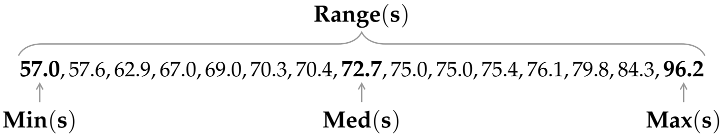


Figure 1.18: Illustration of the minimum, median, and maximum values in the `scores` series. The range is the distance between **Min**(**s**) and **Max**(**s**).

The smallest value is **Min**(**s**) = 57.0 (the minimum), the middle value is **Med**(**s**) = 72.7 (median), the largest value is **Max**(**s**) = 96.2 (the maximum), and the difference between the max and the min values is **Range**(**s**) = 96.2 − 57.0 = 39.2, as illustrated in Figure 1.18.

We can obtain the same information by calling the appropriate methods on `scores` series.

code 1.3.5

```
>>> scores.min()
57.0
>>> scores.median()
72.7
>>> scores.max()
96.2
>>> scores.max() - scores.min()  # range
39.2
```

The simplest statistical visualization is the *strip plot*. Strip plots have an x-axis in the units of the variable and no y-axis. Each observation is represented by a point located at its numerical value.

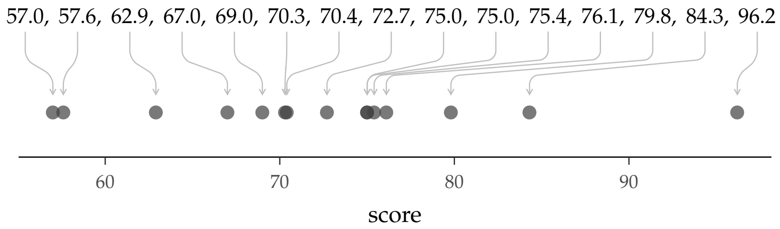


Figure 1.19: Strip plot showing the `score` variable in the `students` data frame. A strip plot is a one-dimensional plot where each observation is mapped to a point at the location that corresponds its value.

The Seaborn function for drawing a strip plot is called `stripplot`, and it is used as follows.

code 1.3.6

```
>>> import seaborn as sns
>>> sns.stripplot(data=students, x="score", jitter=0)
See Figure 1.19.
```

Recall the syntax for Seaborn plot functions: the argument `data=students` tells the `stripplot` function to take the data from the `students` data frame, and `x="score"` indicates we want to represent the `score` variable on the x-axis. The option `jitter=0` was used to disable the Seaborn default behaviour of adding a small amount of random vertical displacement to each data point, which we don't need for this plot.

The visual display of the scores data allows us to see some patterns that might not be visible when we're looking at the list of numbers. Strip plots are great for showing small datasets, since we can see the individual data points.

Mean, variance, and standard deviation

The *mean* is the sum of all values divided by the number of values. Using the formula for the mean, we see the mean of **s** is

$$\overline{\mathbf{s}} = \mathbf{Mean}(\mathbf{s}) = \tfrac{1}{n}\textstyle\sum_i^n s_i = \tfrac{1}{15}\big(57.0 + 57.6 + \cdots + 96.2\big) = 72.6.$$

The mean tells us what a "typical" observation in the dataset would be. It tells us that an average student in this class would score 72.6 on their assessment. We can obtain the mean by calling the `.mean()` method on the `scores` series.

```
>>> scores.mean()
72.6
```

code 1.3.7

To judge the variability of the values in the dataset, we can calculate the *variance* and the *standard deviation*. The variance computed using the complicated-looking formula that sums the squared deviations from the mean:

$$\begin{aligned}\mathbf{Var}(\mathbf{s}) &= \tfrac{1}{n-1}\textstyle\sum_{i=1}^n (s_i - \overline{\mathbf{s}})^2\\ &= \tfrac{1}{14}\big((57.0-72.6)^2 + (57.6-72.6)^2 + \cdots + (96.2-72.6)^2\big)\\ &= 99.6.\end{aligned}$$

Note the variance formula uses the denominator $n - 1 = 14$. To calculate the variance of the `scores` series, we simply call its `.var()` method.

```
>>> scores.var()
99.6
```

code 1.3.8

The *standard deviation* $\textbf{Std}(\mathbf{s})$ is the square root of the variance: $\textbf{Std}(\mathbf{s}) = \sqrt{\textbf{Var}(\mathbf{s})} = \sqrt{99.6} = 9.98$. We compute the standard deviation by calling the `.std()` method.

```
>>> scores.std()
9.98
```

code 1.3.9

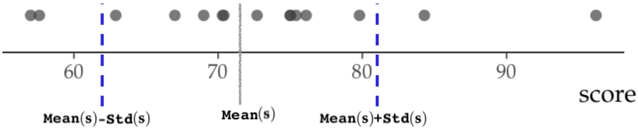


Figure 1.20: A strip plot of the `score` data with additional annotations for the mean and the standard deviation. The mean is shown as a solid line at $\textbf{Mean}(\mathbf{s}) = 72.6$. The two dashed lines are located at $\textbf{Mean}(\mathbf{s}) - \textbf{Std}(\mathbf{s}) = 72.6 - 9.98 = 62.6$ and $\textbf{Mean}(\mathbf{s}) + \textbf{Std}(\mathbf{s}) = 72.6 + 9.98 = 82.58$.

Note many of the scores are contained between the two dashed lines in Figure 1.20. Indeed, 11 out of the 15 values fall in the interval $[\textbf{Mean}(\mathbf{s}) - \textbf{Std}(\mathbf{s}), \textbf{Mean}(\mathbf{s}) + \textbf{Std}(\mathbf{s})] = [62.6, 82.58]$.

Histograms

Strip plots are excellent for displaying individual observations, but it can be difficult to see *how many* observations occur at each value, especially when there are many observations (large n) or when points overlap. We'll now learn about the *histogram*, which is a plot that shows the number of observations that fall within different ranges of possible values.

To make a histogram, we first divide the entire range of values into a series of consecutive, non-overlapping intervals called *bins*. For the student `score` data, we choose to use bins that are 10 units wide and count the number of observations that fall within each bin. Figure 1.21 shows the process of grouping the data points into bins, then counting the total number of observations in each bin. We refer to the count of how many observations fall in each bin as the *frequency*. We can display the frequencies for each bin in a table called a *one-way table* or a *frequency table*.

The final step of creating a histogram is to draw a rectangle whose height is proportional to the frequency (count) in each bin, as shown in Figure 1.22.

The Seaborn function for producing the histogram in Figure 1.22 is `histplot`, and its use is shown below.

```
>>> bins = [50, 60, 70, 80, 90, 100]
>>> sns.histplot(data=students, x="score", bins=bins)
See Figure 1.22.
```

code 1.3.10

bin	values	frequency
[50, 60)	57.0, 57.6	2
[60, 70)	67.0, 69.0, 62.9	3
[70, 80)	75.0, 75.0, 70.3, 76.1, 79.8, 72.7, 75.4, 70.4	8
[80, 90)	84.3	1
[90, 100]	96.2	1

Table 1.6: One-way table of the `scores` data grouped into bins of width 10.

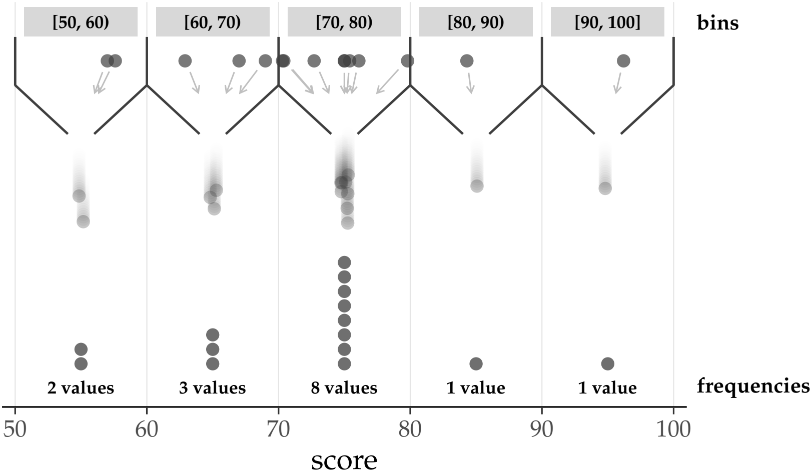


Figure 1.21: Visual representation of the histogram binning process.

In the above code, we call the function `histplot` specifying the data to use for the histogram is in the `students` data frame, and the argument `x="score"` indicates the name of the variable that we're interested in. We manually created a list of values that we want to use as the bin's boundaries in the histogram, then passed this list as the `bins` option when calling the `histplot` function.

The histogram in Figure 1.22 gives us a convenient summary of the students' `scores` data. We can quickly see how much data points fall within each bin. The bin with the highest frequency is called the *mode* of the histogram.

Quartiles

To draw the histogram, we divided the data into five bins. Each bin had the same width and could contain any number of observations. We'll now learn about another type of summary plot that divides the data into intervals of varying width, each containing the same

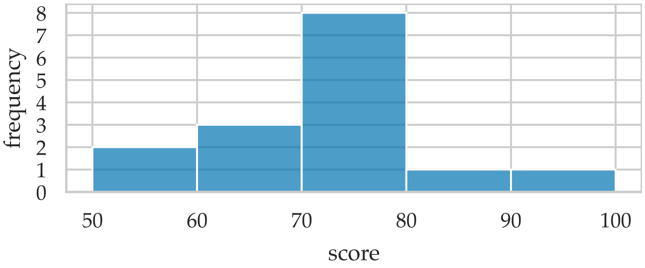


Figure 1.22: Histogram of the `score` variable from the `students` data frame. The width of each bar covers an interval of values called a *bin*. The heights of the bars are proportional to the number of observations within each bin. Bins are usually (but not always) of equal width, with no gaps between them.

number of observations.

The quartiles $\mathbf{Q}_1(\mathbf{s})$, $\mathbf{Q}_2(\mathbf{s})$, and $\mathbf{Q}_3(\mathbf{s})$ are three "fence posts" that separate the data into four intervals with an equal number of observations in each, as illustrated in Figure 1.23.

$$57.0, 57.6, 62.9, 67.0, \underset{\substack{\uparrow\\ \mathbf{Q}_1(\mathbf{s})}}{} \overbrace{69.0, 70.3, 70.4, \underset{\substack{\uparrow\\ \mathbf{Q}_2(\mathbf{s})}}{\mathbf{72.7}}, 75.0, 75.0, 75.4,}^{\mathbf{IQR}(\mathbf{s})} \underset{\substack{\uparrow\\ \mathbf{Q}_3(\mathbf{s})}}{} 76.1, 79.8, 84.3, 96.2$$

Figure 1.23: The three quartiles and the interquartile range of the `scores` data.

We compute the quartiles using the method `.quantile(q)`, for appropriate choice of the argument q.

code 1.3.11

```
>>> Q1 = scores.quantile(q=0.25)
>>> Q1
68.0
>>> Q2 = scores.quantile(q=0.5)
>>> Q2
72.7
>>> Q3 = scores.quantile(q=0.75)
>>> Q3
75.75
```

Note the values of the first and third quartiles correspond to numbers that don't appear in the scores data. Indeed, quartiles correspond to boundaries between data points, and will often fall in between observations.

Recall the *interquartile range* is the distance between the first and the third quartiles. The interquartile range of the `scores` data $\mathbf{s}$ is given by $\mathbf{IQR}(\mathbf{s}) = \mathbf{Q}_3(\mathbf{s}) - \mathbf{Q}_1(\mathbf{s})$.

code 1.3.12

```
>>> IQR = Q3 - Q1
>>> IQR
7.75
```

This result tells us that an interval of width 7.75 contains the middle 50% of the student scores.

When the variable we are describing is obvious from the context, we can simply write $\mathbf{Q}_1$ instead of $\mathbf{Q}_1(\mathbf{s})$ to lighten the notation. We'll use this approach in the equations and figures for the next few pages.

Box plots

We can represent the quartiles graphically using a *box plot*, as shown in Figure 1.24. The rectangular "box" goes from $\mathbf{Q}_1$ to $\mathbf{Q}_3$, so its width corresponds to the $\mathbf{IQR}$. A vertical line is placed at $\mathbf{Q}_2$ (the median). The whiskers in the box plot indicate the lowers and highest observations within the interval $[\mathbf{Q}_1 - 1.5 \cdot \mathbf{IQR}, \mathbf{Q}_3 + 1.5 \cdot \mathbf{IQR}]$. Points outside this interval are called *outliers* and are drawn as separate dots.

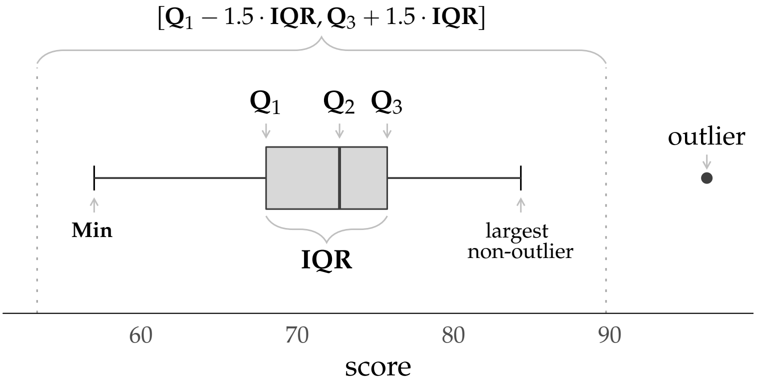


Figure 1.24: Box plot for the `scores` data with additional labels for quantities represented in the plot. The left and right boundaries of the box represent the first and third quartiles. The vertical line in the middle of the box indicates the median. The point on the far-right is called an *outlier*. The lines extending from the box are called *whiskers* and represent the range of the data excluding outliers. The whiskers reach from the smallest and largest values within the interval $[\mathbf{Q}_1 - 1.5 \cdot \mathbf{IQR}, \mathbf{Q}_3 + 1.5 \cdot \mathbf{IQR}]$. Any observations that fall outside the whiskers are considered outliers and are presented with a dot.

The box plot shown in Figure 1.24 is called a *Spear–Tukey box plot* in reference to Mary Eleanor Spear and John Tukey, who popularized this type of data visualization. Spear–Tukey box plots give special attention to the display of *outliers*, which are values that are extremely high or low compared to the other data points. A common criterion to determine if x is an outlier is to check if it satisfies one of the two inequalities $x < \mathbf{Q}_1 - 1.5 \cdot \mathbf{IQR}$ or $x > \mathbf{Q}_3 + 1.5 \cdot \mathbf{IQR}$. In words, x is an outlier if it is further than 1.5 times the interquartile range away from the outer quartiles. There are several other ways to define outliers, but this is the most common definition used for box plots.

Outliers are important because they can have disproportionate influence on some summary statistics and statistical analyses. In the `scores` data, one student's score is an outlier (the value 96.2). It is much higher than the other student scores. This student performed unexpectedly better than the rest of the students. The score 96.2 is greater than $\mathbf{Q}_3 + 1.5 \cdot \mathbf{IQR} = 87.4$, so we classify it as an outlier and display it as an independent point, as shown in Figure 1.24.

The whiskers span from the smallest observation that's larger than $\mathbf{Q}_1 - 1.5 \cdot \mathbf{IQR}$, and the largest observation that's still smaller than $\mathbf{Q}_3 + 1.5 \cdot \mathbf{IQR}$. The purpose of the whiskers is to provide an "honest" representation of the range of the data. By splitting off the outliers as independent points, the whiskers show us the non-outlier range of the data.

We can use the Seaborn function `boxplot` to produce a box plot.

```
>>> sns.boxplot(data=students, x="score")
The result is shown in Figure 1.24
```

code 1.3.13

Box plots are among the most common visual summaries for numerical data. In a box plot, we can't see the individual data points, but we see the position of the quartiles $\mathbf{Q}_1$, $\mathbf{Q}_2$, and $\mathbf{Q}_3$, which provides an excellent overview of the data. We can tell at once glance where the middle 50% of the data values fall, and the whiskers tell us the range of all the non-outlier values.

Summary of descriptive statistics for numerical variables

Let's review all the descriptive statistics we calculated from the `score` variable in the students dataset. Table 1.7 lists all the numerical statistics we computed and the relevant Pandas methods we used to compute each statistic from the `scores` series.

We can compute the most important summary statistics in a single step by calling the `.describe()` method on the `scores` series.

```
>>> scores.describe()
count     15
mean     72.58
std       9.98
min      57.00
25%      68.00    # = Q1
50%      72.70    # = Q2
75%      75.75    # = Q3
max      96.20
```

code 1.3.14

Statistic	Pandas method	Value	Measurement of
n	`scores.count()`	15	Sample size
Mean(**s**)	`scores.mean()`	72.6	Central tendency
Med(**s**)	`scores.median()`	72.7	Central tendency
Var(**s**)	`scores.var()`	99.6	Dispersion
Std(**s**)	`scores.std()`	9.98	Dispersion
Range(**s**)		39.2	Dispersion
IQR(**s**)		7.75	Dispersion
Min(**s**)	`scores.min()`	57.0	Position
$\mathbf{Q}_1$(**s**)	`scores.quantile(q=0.25)`	68.0	Position
$\mathbf{Q}_2$(**s**)	`scores.quantile(q=0.5)`	72.7	Position
$\mathbf{Q}_3$(**s**)	`scores.quantile(q=0.75)`	75.75	Position
Max(**s**)	`scores.max()`	96.2	Position

Table 1.7: Table of numerical summary statistics for the `scores` data.

The expression `students["score"].describe()` produces the same result. It is also possible to obtain summary statistics for multiple variables at once by calling the `.describe()` method on the data frame. See code block 1.3.15 for an example of this.

Tables of summary statistics like Table 1.7 are a succinct way to report the most important characteristics of numerical variables, so we often see them in research papers and reports. Basically, it's not practical to show all the data in a science report, but reporting the mean, the variance, the standard deviation, and the five-number summary (**Min**, $\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$, **Max**) gives an overall idea of the characteristics of the data.

It's important to keep in mind that numerical summaries offer only a limited view of the data, and you should always plot the data to get a better understanding. The strip plot (Figure 1.19), the histogram (Figure 1.22), and the box plot (Figure 1.24) all capture important aspects of the data and are worth looking at.

Exercises

E1.13 Compute the **Mean**, **Min**, **Max**, and **Range** of the `effort` variable in the `students` dataset.

Hint: Use `students["effort"]` to select the `"effort"` column.

E1.14 Find $\mathbf{Q}_1$, **Med**, and $\mathbf{Q}_3$ of the `effort` variable in the `students` dataset.

E1.15 Make a one-way frequency table for the `effort` variable. Use $(5,7]$, $(7,9]$, $(9,11]$, $(11,13]$ as the boundaries of the bins.

Hint: Use the `.value_counts` and pass in the `bins` argument.

TODO: add some data viz interpretation questions? especially for box plots

1.3.2 Relations between numerical variables

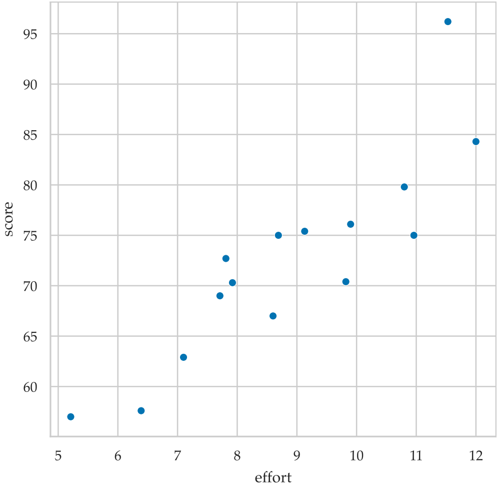
We're often interested in studying the relations between variables in a dataset. Consider the `effort` and `score` variables in the students dataset, which we'll denote **e** and **s** in math formulas. We can calculate the descriptive statistics for these two variables by selecting them from the data frame, then calling the `.describe()` method.

```
>>> students[ ["effort","score"] ].describe()
       effort  score
count   15.00  15.00
mean     8.90  72.58
std      1.95   9.98
min      5.21  57.00
25%      7.76  68.00
50%      8.69  72.70
75%     10.35  75.75
max     12.00  96.20
```

code 1.3.15

Looking at the descriptive statistics of the two variables separately doesn't tell us anything about the *relationship* between them.

The *scatter plot* is a common visualization for two numerical variables. Since we're interested in the relationship between the `effort` and `score` variables, we can generate a scatter plot of `score` versus `effort`, as shown in Figure 1.25.



effort (**e**)	score (**s**)
11.53	96.2
12	84.3
10.8	79.8
9.9	76.1
9.13	75.4
10.96	75.0
8.69	75.0
7.81	72.7
9.82	70.4
7.92	70.3
7.71	69.0
8.6	67.0
7.1	62.9
6.39	57.6
5.21	57.0

Figure 1.25: Scatter plot of the `score` variable versus the `effort` variable. Each point in the scatter plot has its x-position determined by the value of the `effort` variable and its y-position determined by the `score` variable.

In a scatter plot, two numerical variables are mapped to the x and y coordinates of points. If there is an association between the two variables, the points on the scatter plot will show a pattern. The pattern in Figure 1.25 seems to indicate that higher `effort` values are associated with higher `score` values. The code for producing this scatter plot is as follows.

```
>>> sns.scatterplot(data=students, x="effort", y="score")
The result is shown in Figure 1.25.
```

code 1.3.16

The arguments `x="effort"` and `y="score"` tell the `scatterplot` function to use the `effort` values as the x-coordinates and the `score` variable as the y-coordinates of the points.

Measures of association

Consider two numerical variables $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ and $\mathbf{y} = [y_1, y_2, \ldots, y_n]$. Rather than thinking of x_i and y_i as separate measurements, we want to think of them as joint measurements (x_i, y_i), and use the notation $[\mathbf{x}, \mathbf{y}] = [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$ to describe the pair of variables. You can also think about $[\mathbf{x}, \mathbf{y}]$ as two columns of a data frame. A *positive linear association* between the variables $\mathbf{x}$ and $\mathbf{y}$ means that large x-values tend to be associated with large y-values, and small x-values are associated with small y-values. A *negative linear association* describes the opposite phenomenon, where large x-values are associated with small y-values, and vice versa.

We can measure the strength of the association between two variables using the concepts of *covariance* and *correlation*.

Covariance The *covariance* of $\mathbf{x}$ and $\mathbf{y}$ is a measure of the joint variability of the two variables:

$$\mathbf{Cov}(\mathbf{x}, \mathbf{y}) = \tfrac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{\mathbf{x}})(y_i - \overline{\mathbf{y}}).$$

Note the formula for covariance is similar to the formula for the variance **Var** (see page 57), but is computed from the joint difference of the values x_i and y_i from their means $\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$ and $\overline{\mathbf{y}} = \mathbf{Mean}(\mathbf{y})$. The Pandas method `.cov()` computes the covariance between all the numerical variables in a data frame. We can use the following code to compute the covariance between the `effort` and `score` variables.

```
>>> students[["effort", "score"]].cov()
         effort  score
effort      3.8  17.10
score      17.1  99.59
```

code 1.3.17

The expression students[["effort", "score"]] selects the two columns we're interested in from the data frame students, then we call the method .cov() to compute the covariance between all pairs of variables. The results are presented as a 2 × 2 table called the *covariance matrix*. The entries on the diagonal correspond to the covariance of a variable with itself, which is equal to the variance. For example, the bottom-right entry of the covariance matrix is $\mathbf{Corr}(\mathbf{s}, \mathbf{s}) = \mathbf{Var}(\mathbf{s})$, which we calculated earlier in this section.

The covariance between two variables takes on values between $-\infty$ and $+\infty$. Covariance isn't a good measure of relationship *strength*, since its value depends on the magnitude of the two variables. If either $\mathbf{x}$ or $\mathbf{y}$ have high variance, then the value of $\mathbf{Cov}(\mathbf{x}, \mathbf{y})$ will also be high, regardless of whether the association between the two variables is strong or weak. This is why we prefer the *correlation coefficient* to measure the strength of the association between variables.

Correlation The *correlation coefficient* $\mathbf{Corr}(\mathbf{x}, \mathbf{y})$ is a measure of the linear relatedness between the variables $\mathbf{x}$ and $\mathbf{y}$. Correlation is the normalized version of covariance, which we compute by dividing the covariance by the standard deviations of individual variables:

$$\mathbf{Corr}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{Cov}(\mathbf{x}, \mathbf{y})}{\mathbf{Std}(\mathbf{x})\,\mathbf{Std}(\mathbf{y})}\,.$$

The value $\mathbf{Corr}(\mathbf{x}, \mathbf{y})$ is called the *Pearson correlation coefficient*, and sometimes denoted $\rho_{\mathbf{x},\mathbf{y}}$, where ρ is the Greek letter *rho*. We use the method .corr() on data frames to compute correlation coefficients.

The correlation coefficient is a dimensionless quantity that takes on values between -1 to $+1$. A correlation coefficient close to $+1$ indicates a strong positive association, while a value close to -1 indicates a strong negative association.

Correlation between effort and score

Let's use the concept of correlation to quantify the strength of the associations between the effort $= \mathbf{e}$ and score $= \mathbf{s}$ variables in the students dataset. We can compute the correlation coefficient $\mathbf{Corr}(\mathbf{e}, \mathbf{s})$ by selecting the effort and score columns from the students data frame, then calling the .corr() method.

```
>>> students[["effort", "score"]].corr()
        effort  score
effort    1.00   0.88
score     0.88   1.00
```

code 1.3.18

The result of the `.corr()` method is called the *correlation matrix*. Note the correlation of any variable with itself is 1, as we can see from the values on the diagonal. Note also that correlation is a symmetric quantity, meaning $\mathbf{Corr}(\mathbf{e}, \mathbf{s}) = \mathbf{Corr}(\mathbf{s}, \mathbf{e})$.

The correlation coefficient $\mathbf{Corr}(\mathbf{e}, \mathbf{s})$ is the top-right entry of the correlation matrix: $\mathbf{Corr}(\mathbf{e}, \mathbf{s}) = 0.88$. A correlation coefficient of 0.88 indicates a strong positive correlation, meaning that the `effort` variable is closely associated with the `score` variable. Because $\mathbf{Corr}(\mathbf{e}, \mathbf{s})$ is a positive number, we say that there is a *positive* association between the `effort` and `score` variables: students who put in more hours on the learning platform also got a better score. This confirms what we observed in the scatter plot in Figure 1.25, where we see the points seem scattered around an invisible line that points diagonally upward.

A negative correlation coefficient indicates an *inverse association*, meaning that students who put in more `effort` tended to have lower `scores`. A zero correlation value would suggest that there is no relationship between the two variables, or at least no simple linear relationship.

The correlation coefficient is a very limited tool for describing the relationship between two variables, because it only measures *simple linear association*, which might not be a good model for the data. Even if we find a high association between two variables, this doesn't necessarily mean that the variables follow a *linear* pattern. See example **F** in Figure 1.26.

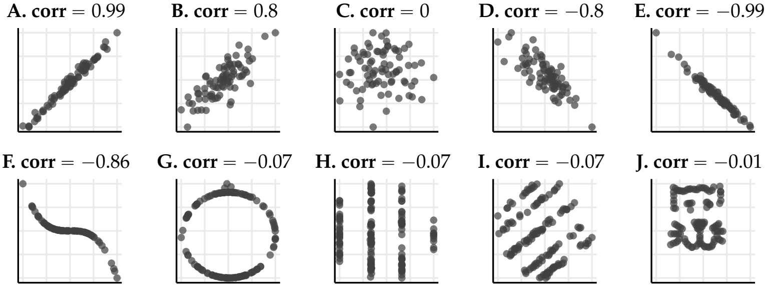


Figure 1.26: A correlation value close to 1 or -1 indicates that two variables have a linear association, as shown in plots **A** and **E**. Plots **B** and **D** also indicate a linear association between the variables, but it is a *noisy* relationship. Variables may have a correlation close to 1 or -1 and *not* follow a linear relationship, as shown in plot **F**. A zero correlation indicates that there *may* be no relationship, as shown in plot **C**. However, we can also calculate a correlation close to zero for cases when the data shows a strong, non-linear pattern, like plots **G**, **H**, **I**, and **J**.

In observational studies, we can't say that one variable *causes* the other, even in cases when we find a strong linear association. Two variables can occur at higher values together without one having a direct influence on the other. The maxim "correlation does not imply causation" is a fundamental idea in statistics. In this case, we cannot conclude that more effort lead to higher scores. It's equally plausible, for example, that an unobserved confounding leads some students to both put in more effort and perform better on the assessment. Maybe some students were more interested in the subject matter to begin with, which motivated them to get high scores, and to invest more hours of effort. To make conclusions about causation, we need a carefully designed experiment and more involved statistical analysis. We'll learn how to model linear relationships between numerical variables in Chapter 4.

Exercises

E1.16 Draw a scatter plot for the following dataset of (x, y) pairs: (2,2), (3,3), (4,3), (5,5), (6,4), (5,4), (7,6), (8,5).

1.3.3 Comparing two groups of numerical variables

We often want to compare the descriptive statistics of two groups. To do this, we can use all the data visualizations we saw for numerical variables (strip plots, histograms, box plots) to generate plots for each group. We can also generate combined plots for both groups, where the group variable is represented as a different dimension or colour.

Charlotte wants to see how student scores compare between the two curriculum variants. Recall that each student was randomly assigned to either the debate or the lecture curriculum variants, and this information is recorded in the curriculum variable of the students dataset (see Table 1.5 on page 59).

Let's calculate the summary statistics for each group. We can do this by selecting the rows of the students data frame that correspond to each group, then calling the .describe() method.

```
>>> dstudents = students[students["curriculum"]=="debate"]
>>> dstudents["score"].describe()
count     8.00
mean     76.46
std      10.52
...
>>> lstudents = students[students["curriculum"]=="lecture"]
>>> lstudents["score"].describe()
count     7.00
mean     68.14
```

code 1.3.19

```
std         7.76
...
```

In the above code, the data frame `dstudents` contains the subset of the rows from the `students` data frame where the `curriculum` variable has the value `debate`. Similarly, the expression `students["curriculum"]=="lecture"` is a selection mask that chooses only rows where the `curriculum` variable is `lecture`. Flip back to the explanations on page 31 if you need a refresher of the syntax we use to select subsets of the rows in a data frame.

Looking at the numerical summaries of the `score` variable for students in the `debate` and `lecture` curriculum groups, we see the scores were higher, on average, for students in the `debate` group.

We can visualize the data distribution of `score` variable within the two groups using two strip plots, two histograms, or two box plots, as shown in Figure 1.27. The plots are drawn side-by-side (on the same *x*-axis) in order to make the comparison between the two groups easier.

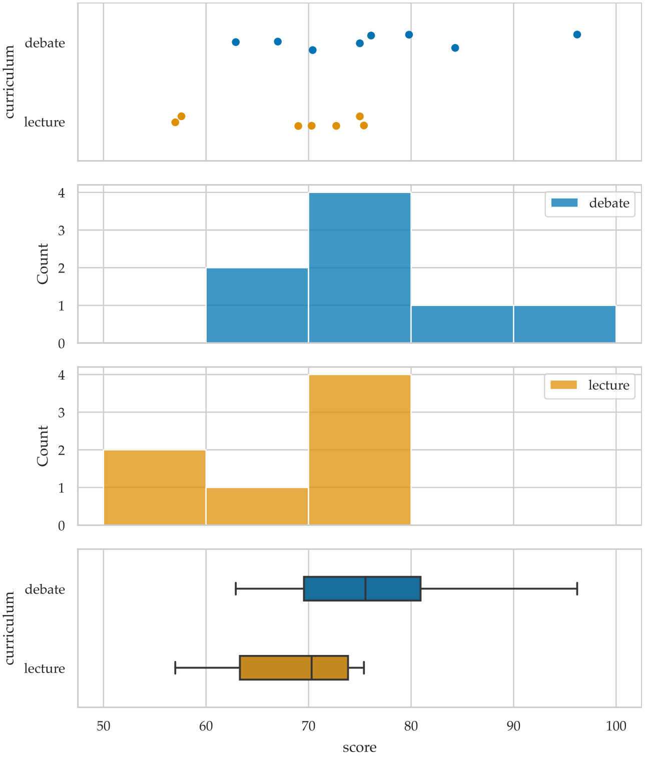


Figure 1.27: Strip plots, histograms, and box plots can be used to visually compare students' `scores` in the `debate` and `lecture` groups.

The code used to produce the strip plot and the box plot in Figure 1.27 is very similar to the single-variable plots we saw above, with the addition of y="curriculum" argument that adds another "dimension" to the plots. The Seaborn library recognizes that curriculum is a categorical variable, and automatically performs the appropriate data selection for the two groups.

```
>>> sns.stripplot(data=students, x="score", y="curriculum")  code
See first plot of Figure 1.27.                                1.3.20
>>> sns.boxplot(data=students, x="score", y="curriculum")
See last plot of Figure 1.27.
```

It's also possible to draw a combined histogram for the data by using colours to distinguish the two groups, but the results are not very legible for such a small dataset (I tried it!). Instead, we'll generate separate histograms for the two groups, as shown below.

```
>>> bins = [50, 60, 70, 80, 90, 100]                          code
>>> sns.histplot(data=dstudents, x="score", bins=bins)        1.3.21
>>> sns.histplot(data=lstudents, x="score", bins=bins)
The results are shown in the middle plots of Figure 1.27.
```

Visual inspection of the plots in Figure 1.27 seems to suggest that students who took the debate curriculum did better than students who took the lecture curriculum. The numerical summaries we calculated in code block 1.3.19 also support this observation: the difference between group means is $76.46 - 68.14 = 8.32$. However, we need to interpret the magnitude of this observed difference in relation to the variability in the data, which can be seen in the scatter plots in Figure 1.27 and measured by the standard deviations of the two groups (10.52 and 7.76, respectively). Given the variability in the observed scores, it is not immediately clear if we should count the observed difference between group means as evidence that the debate curriculum is better than the lecture curriculum, or if the observed differences could have occurred by chance.

We'll continue the exploration of Charlotte's research question about the relative effectiveness of the debate and lecture curriculum variants in Section 3.5, where we'll introduce the *hypothesis testing* procedure for comparing two groups.

Exercises

E1.17 TODO: add simple exercise

1.3.4 Categorical variables

Categorical variables take on one of a discrete set of possible values like the answers to true or false questions, the presence or absence of some characteristic (1 or 0), a person's country of residence, or group membership of an individual (intervention or control group).

It doesn't make sense to compute numerical statistics like the mean and the variance for categorical data, so we use descriptive statistics and visualizations based on frequencies (counts) and proportions. Recall the *frequency* of a given value is the number of occurrences of this value within the data.

Let's show some examples of descriptive statistics for categorical data by looking at the `background` variable in the `student` dataset, which is a categorical variable that takes on one of three possible values: `arts`, `science`, or `business`. See the column `"background"` in Table 1.5 on page 59. The `background` variable contains the following data:

$$\mathbf{b} = [\texttt{arts}, \texttt{science}, \texttt{arts}, \texttt{arts}, \texttt{science}, \texttt{business}, \texttt{science}, \texttt{business}, \\ \texttt{business}, \texttt{science}, \texttt{business}, \texttt{arts}, \texttt{science}, \texttt{science}, \texttt{arts}].$$

We'll use the notation **b** for the background variable in math equations and examples in the remainder of this section.

We can define the following summary statistics for categorical data:

- $\mathbf{Freq}_v(\mathbf{x})$: the *frequency* (count) of the value v is the number of times the value v occurs in the data $\mathbf{x}$:

$$\mathbf{Freq}_v(\mathbf{x}) \stackrel{\text{def}}{=} \text{number of } v \text{ in } \mathbf{x}.$$

  For example, $\mathbf{Freq}_{\texttt{arts}}(\mathbf{b}) = 5$ since the value `arts` appears five times in the `background` variable.

- $\mathbf{RelFreq}_v(\mathbf{x})$: the *relative frequency* or *proportion* of the value v in the data $\mathbf{x}$. The relative frequency is the number of times v occurs in $\mathbf{x}$ divided by the sample size:

$$\mathbf{RelFreq}_v(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\mathbf{Freq}_v(\mathbf{x})}{n} = \frac{\text{number of } v \text{ in } \mathbf{x}}{\text{total number of observations}}.$$

  For example, the relative frequency of the `arts` background is $\mathbf{RelFreq}_{\texttt{arts}}(\mathbf{b}) = \frac{5}{15} = 0.333$. This tells us that 33.3% of the students come from an arts background.

- **Mode**: the *mode* is the category with the most observations. The mode of the `background` variable is `science`, since science was the most common `background` among the students that participated.

We can display frequencies and relative frequencies in a one-way table, as shown in Table 1.8.

background	frequency	relative frequency
arts	5	0.33
business	4	0.27
science	6	0.40

Table 1.8: Summary statistics for the `background` variable in the students dataset. Relative frequencies are obtained by dividing the frequency of each value by the total number of observations ($n = 15$ in this case).

Note the sum of the frequencies is equal to the total number of observations: $\mathbf{Freq}_{\texttt{arts}}(\mathbf{b}) + \mathbf{Freq}_{\texttt{science}}(\mathbf{b}) + \mathbf{Freq}_{\texttt{business}}(\mathbf{b}) = 15$. The sum of the relative frequencies for the three categories is equal to one: $\mathbf{RelFreq}_{\texttt{arts}}(\mathbf{b}) + \mathbf{RelFreq}_{\texttt{science}}(\mathbf{b}) + \mathbf{RelFreq}_{\texttt{business}}(\mathbf{b}) = 1$.

We can use the Pandas methods `.value_counts()` to compute the frequencies of any series or data frame.

```
>>> backgrounds = students["background"]
>>> backgrounds.value_counts()
arts        5
business    4
science     6
```

code 1.3.22

Adding the option `normalize=True` to the `.value_counts()` method computes the relative frequencies, as shown below.

```
>>> backgrounds.value_counts(normalize=True)
science     0.40
arts        0.33
business    0.27
```

code 1.3.23

We can plot categorical data using a *bar plot*, as shown in Figure 1.28. In a bar plot, each rectangle (or "bar") describes one category. The height of the bar represents a numerical measurement within the given category, such as a frequency or a relative frequency. Unlike a histogram, the width of the bars has no meaning. The Seaborn function `countplot` can be used to generate a bar plot.

```
>>> sns.countplot(data=students, x="background")
The result is shown in Figure 1.28.
```

code 1.3.24

Exercises

E1.18 Make a bar plot displaying the frequencies of the `curriculum` variable in the `students` dataset.

Hint: Use the Seaborn function `countplot`.

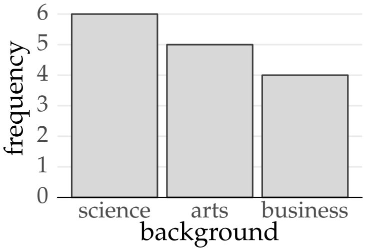


Figure 1.28: Bar plot of the frequencies (counts) of the students' `background`. The heights of the bars represent the number of students in each category.

E1.19 Compute the frequencies and the relative frequencies of the `curriculum` variable in the `students` dataset. Display the results in a one-way table.

E1.20 What is the mode for `curriculum`?

Comparing two categorical variables

Let's now look at the descriptive statistics for pairs of categorical variables. We'll use the `background` $= \mathbf{b}$ and `curriculum` $= \mathbf{c}$ variables from the `students` data frame, which we have reproduced in full below:

$$\begin{aligned}[\mathbf{b},\mathbf{c}] = \big[&(\texttt{arts},\texttt{debate}), (\texttt{science},\texttt{lecture}), (\texttt{arts},\texttt{debate}),\\ &(\texttt{arts},\texttt{lecture}), (\texttt{science},\texttt{debate}), (\texttt{business},\texttt{debate}),\\ &(\texttt{science},\texttt{lecture}), (\texttt{business},\texttt{lecture}), (\texttt{business},\texttt{lecture}),\\ &(\texttt{science},\texttt{lecture}), (\texttt{business},\texttt{debate}), (\texttt{arts},\texttt{debate}),\\ &(\texttt{science},\texttt{debate}), (\texttt{science},\texttt{lecture}), (\texttt{arts},\texttt{debate})\big].\end{aligned}$$

Each observation consists of a pair of values: the academic background of the student and which variation of the curriculum they are enrolled in.

The tools for describing multivariable categorical data are similar to what we saw above: we count the number of occurrence and draw bar plots that visually represent quantities. The analysis of two variables requires some new concepts like *joint frequencies*, *marginal frequencies*, and *conditional frequencies*, which we'll now introduce.

The *joint frequency* of the pair of values (v, w) in the data $[\mathbf{x}, \mathbf{y}]$ is defined as:

$$\mathbf{Freq}_{v,w}(\mathbf{x},\mathbf{y}) \overset{\text{def}}{=} \text{number of pairs } (v,w) \text{ in the data } [\mathbf{x},\mathbf{y}].$$

For example, $\mathbf{Freq}_{\texttt{arts},\texttt{debate}}(\mathbf{b},\mathbf{c}) = 4$, since the pair (`arts`,`debate`) occurs four times in the data $[\mathbf{b},\mathbf{c}]$. The term "joint" tells us we're counting the joint occurrence of two variables in the data, rather than studying the two variables separately. In case you were wondering, no, the term "joint frequency" is not related to how often students were smoking cannabis. This data was not collected.

The concept of a *marginal frequency* corresponds to counting the occurrences of one variable, while ignoring the value of the other. We already computed the marginal frequencies for the background variable in the previous section:

$$\mathbf{Freq}_{\texttt{arts}}(\mathbf{b}) = 5, \quad \mathbf{Freq}_{\texttt{business}}(\mathbf{b}) = 4, \quad \mathbf{Freq}_{\texttt{science}}(\mathbf{b}) = 6.$$

The marginal frequencies for the curriculum variable are

$$\mathbf{Freq}_{\texttt{debate}}(\mathbf{c}) = 8 \quad \text{and} \quad \mathbf{Freq}_{\texttt{lecture}}(\mathbf{c}) = 7.$$

These numbers were obtained by counting the number of occurrences of debate and lecture in the data. The reason for the name "marginal" will become apparent shortly.

	background			
curriculum	arts	business	science	TOTAL
lecture	1	2	4 ①	7 ②
debate	4	2	2	8
TOTAL	5	4	6 ③	15

Table 1.9: Two-way table of the joint frequencies for the variables background and curriculum from the students dataset. The totals for each curriculum type are indicated in the rightmost column. The totals for each background are indicated in the last row.

We can display the joint frequencies and marginal frequencies for a pair of categorical variables in a *two-way table*, as shown in Table 1.9. A two-way table shows the observed frequency of each combination of the two variables. The label ① refers to the joint frequency $\mathbf{Freq}_{\texttt{science,lecture}}(\mathbf{b}, \mathbf{c}) = 4$, which is the number of students with a science background who are enrolled in the lecture curriculum.

The row sum ② is the marginal frequency $\mathbf{Freq}_{\texttt{lecture}}(\mathbf{c}) = 7$, which is the total number of students in the lecture curriculum, $1 + 2 + 4 = 7$. The column sum labelled ③ refers to the marginal frequency $\mathbf{Freq}_{\texttt{science}}(\mathbf{b}) = 6$, which is the total number of students that have a science background, $4 + 2 = 6$. The reason for calling the row and column sums marginal frequencies should be clear now: we call them marginal because they appear in the margins of the table.

We can create a two-way table using the Pandas function `crosstab`, by passing the values of the row variable to the argument `index`, and the values of the column variable to the argument `columns`.

```
>>> pd.crosstab(index=students["curriculum"],
                columns=students["background"],
                margins=True, margins_name="TOTAL")
The result is shown Table 1.9.
```

code 1.3.25

The option `margins=True` tells the `crosstab` function to compute the marginal frequencies, and `margins_name` sets the name for the marginal columns.

* * *

We can use a grouped bar plot or a stacked bar plot to visualize the joint frequencies of the two variables, as shown in Figure 1.29.

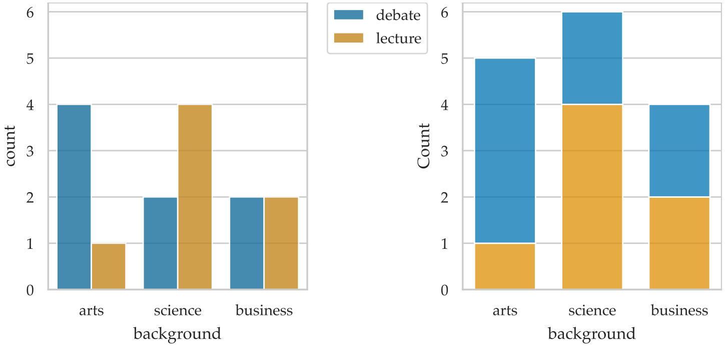


Figure 1.29: Bar plots showing the frequencies of the `curriculum` variable for each `background`. On the left we see a *grouped* bar plot in which values for each `curriculum` are shown side by side. On the right we see a *stacked* bar plot where the values are stacked on top of each other.

A *grouped bar plot* displays a numeric value for a set of groups and subgroups. The group variable is represented using a label, while the subgroups are represented using different colours. The numeric values are represented by the height of the bar. See the left side of Figure 1.29. The Seaborn code to produce this figure is based on the `countplot` function, which we've already seen, but we specify that the `hue` (colour) of the bars should be controlled by the grouping variable.

```
>>> sns.countplot(data=students, x="background",
                  hue="curriculum", alpha=0.8)
See Figure 1.29 (left).
```

code 1.3.26

In a *stacked bar plot*, a rectangle for each category in one variable is made up of smaller blocks that represent a second variable. The right side of Figure 1.29 is a stacked bar plot that shows the frequencies of each of the two `curriculum` types within each of the three academic `backgrounds`. The Seaborn code to produce this figure is based on the `histplot` function.

```
>>> sns.histplot(data=students, x="background",
             hue="curriculum", multiple="stack", shrink=.7)
See Figure 1.29 (right).
```

code 1.3.27

The bar plots in Figure 1.29 display the same frequency information as in Table 1.9, but help us visually identify the relative sizes of the groups. We see that more `arts` students were assigned to the `debate` curriculum, while `science` students ended up in the `lecture` curriculum. The `business` students are evenly split between the two types of curriculum.

Joint relative frequencies The *joint relative frequency* for the pair of values (v, w) is denoted $\mathbf{RelFreq}_{v,w}(\mathbf{x}, \mathbf{y})$ and is obtained by dividing the joint frequency by the total size of the dataset:

$$\mathbf{RelFreq}_{v,w}(\mathbf{x}, \mathbf{y}) \overset{\text{def}}{=} \frac{\mathbf{Freq}_{v,w}(\mathbf{x}, \mathbf{y})}{n}.$$

For example, the relative frequency $\mathbf{RelFreq}_{\texttt{arts,debate}}(\mathbf{b}, \mathbf{c}) = \frac{4}{15}$ is obtained by dividing the frequency $\mathbf{Freq}_{\texttt{arts,debate}}(\mathbf{b}, \mathbf{c}) = 4$ by the number of observations, $n = 15$.

We can display the joint relative frequencies using a two-way table, as shown in Table 1.10. The two-way table of relative frequencies is obtained by taking the values from the table of joint frequencies (Table 1.9) and dividing them by $n = 15$.

	background				
curriculum	arts	business	science	TOTAL	④
lecture	0.07	0.13	0.27	0.47	
debate	0.27	0.13	0.13	0.53	⑤
TOTAL	0.33	0.27	0.40	1.00	
					⑥

Table 1.10: Two-way table of the joint relative frequencies for the variables `background` and `curriculum` from the students dataset.

The label ④ refers to the *joint relative frequency* of students with a `science` background taking the `lecture` curriculum, which is given by $\mathbf{RelFreq}_{\texttt{science,lecture}}(\mathbf{b}, \mathbf{c}) = \frac{4}{15} = 0.27$.

The row and column sums in Table 1.10 are called *marginal relative frequencies*. Label ⑤ refers to the marginal relative frequency $\mathbf{RelFreq}_{\texttt{lecture}}(\mathbf{c})$, and it is obtained by dividing marginal frequency $\mathbf{Freq}_{\texttt{lecture}}(\mathbf{c})$ by n. This is the sum of all the values in the `lecture` curriculum: $0.07 + 0.13 + 0.27 = 0.47$. The label ⑥ refers to the column sum for the `science` background, $0.27 + 0.13 = 0.40$.

The code to produce a two-way table of joint relative frequencies is nearly identical to the `crosstab` invocation we saw in code 1.3.25, with the addition of the option `normalize=True`.

code 1.3.28

```
>>> pd.crosstab(index=students["curriculum"],
                columns=students["background"],
                margins=True, margins_name="TOTAL",
                normalize=True)
The result is shown in Table 1.10.
```

Conditional relative frequencies There is one last type of relative frequency calculation that you need to know about. Last one, I promise!

The *conditional relative frequency* of the value v given the value w is denoted $\mathbf{RelFreq}_{v|w}(\mathbf{x}, \mathbf{y})$, and it is computed by dividing the number of observations of the pair (v, w) by the number of observations that contain the value w.

$$\mathbf{RelFreq}_{v|w}(\mathbf{x}, \mathbf{y}) \overset{\text{def}}{=} \frac{\mathbf{Freq}_{v,w}(\mathbf{x}, \mathbf{y})}{\mathbf{Freq}_{w}(\mathbf{y})} = \frac{\text{number of pairs } (v, w) \text{ in } [\mathbf{x}, \mathbf{y}]}{\text{number of } w \text{ in } \mathbf{y}}.$$

The vertical bar symbol "|" is pronounced "given" or "conditioned on" in this context, and it indicates we're performing the counting within a subset of the data.

Suppose we're interested in knowing the proportion of students enrolled in the `lecture` curriculum within the subset of students that have an `arts` background. Looking back at Table 1.9 (page 78) that has the joint frequencies, we see there is a total of five students with an `arts` background and only one of these students is enrolled in the `lecture` curriculum, so the conditional relative frequency is $\mathbf{RelFreq}_{\texttt{lecture}|\texttt{arts}}(\mathbf{c}, \mathbf{b}) = \frac{1}{5} = 0.2$.

Table 1.11 contains all the relative frequencies of the `curriculum` variable conditioned within the `background` variable. Instead of dividing the frequencies by the total number of observations, we divide it by the number of observations within each column. Note the sum of the values in each column is equal to one.

	background			
curriculum	arts	business	science	TOTAL
lecture	0.20	0.50	0.67	0.47
debate	0.80	0.50	0.33	0.53
TOTAL	1.00	1.00	1.00	1.00

Table 1.11: Table of relative frequencies of the `curriculum` variable conditioned on the `background` variable.

The code for producing Table 1.11 is based on the `crosstab` function with the addition of the option `normalize="columns"`, which performs the normalization required to obtain the relative frequencies of the `curriculum` variable conditioned on the `background` variable.

```
>>> pd.crosstab(index=students["curriculum"],
                columns=students["background"],
                margins=True, margins_name="TOTAL",
                normalize="columns")
```

code 1.3.29

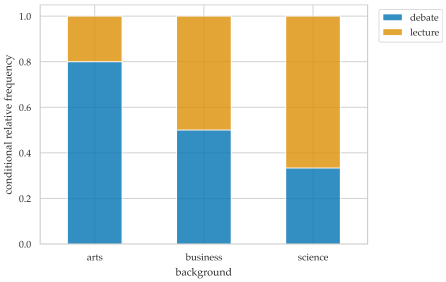


Figure 1.30: Relative frequencies of `curriculum` conditional on `background`.

Figure 1.30 shows the values from Table 1.11 represented as a stacked bar graph. Conditional relative frequencies allow us to compare the different proportions of the `curriculum` variable within groups of students with different `backgrounds`.

We can also calculate the relative frequencies conditional on `curriculum`, which are obtained by dividing each frequency by the total observation *per row*, as shown in Table 1.12.

The code for producing a two-way table with row normalization requires passing the option `normalize="index"` to the `crosstab` function, as shown below.

```
>>> pd.crosstab(index=students["curriculum"],
                columns=students["background"],
                margins=True, margins_name="TOTAL",
                normalize="index")
```

code 1.3.30

	background			
curriculum	arts	business	science	TOTAL
lecture	0.14	0.29	0.57	1.00
debate	0.50	0.25	0.25	1.00
TOTAL	0.33	0.27	0.40	1.00

Table 1.12: Relative frequencies of the `background` variable conditioned on the `curriculum` variable.

```
See Table 1.12.
```

Table 1.12 shows the relative composition of student backgrounds per curriculum. For example, the second row shows that students taking the `debate` curriculum comprised of 50% `arts` students, 25% `business` students, and 25% `science` students. The proportions are different for the `lecture` variant. Figure 1.31 shows a visual representation of these proportions as a stacked bar plot.

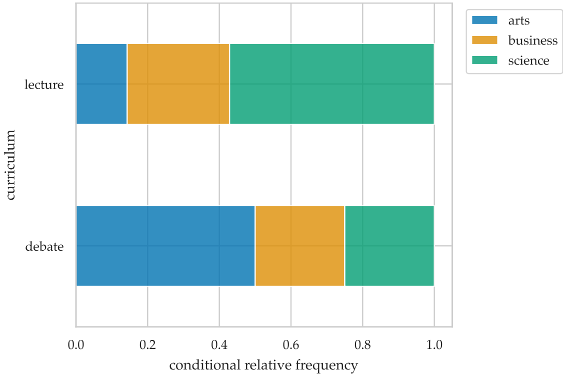


Figure 1.31: Relative frequencies of `background` conditional on `curriculum`.

Conditional relative frequencies are useful when we want to compare proportions (relative frequencies) between groups. In particular, conditional relative frequencies allow us to compare groups of different sizes, since the calculations are normalized based on the groups sizes.

Exercises

E1.21 given the following data, make a frequency, relative frequency, and two conditional relative frequency tables.

E1.22 TODO: question testing the concept of correlation != causation

1.3.5 Explanations

We'll now provide some additional details and explanations about descriptive statistics, which we skipped in the previous pages.

Measures of shape

The concepts of *skewness* and *modality* are used to describe the "shape" of a data distribution, when looking at its histogram.

Skewness Many data distributions have the bulk of their values concentrated in one central region, and the frequency of values gets smaller and smaller as we move away from the central region. The values extending to the left and the right of the main region are called the *tails* of the distribution. When the distribution has a long left tail, we say it is *left-skewed*, and conversely, when the distribution has a long right tail, we say it is *right-skewed*. Distributions that have similar left and right tails are called *symmetrical*. Figure 1.32 shows examples of histograms with different skews.

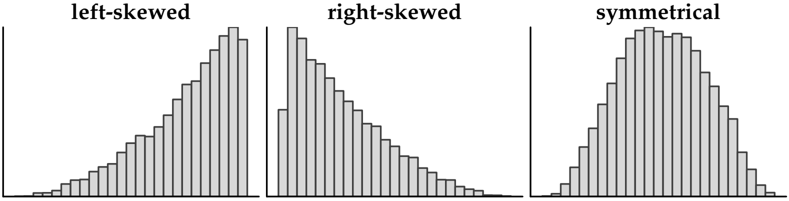


Figure 1.32: Histograms displaying distributions with three different *skews*. When the tail of the distribution extends further to the left, the data is *left-skewed*. When the values extend further to the right, the data is *right-skewed*.

The terms *left-skewed* and *right-skewed* are more qualitative than quantitative, but there are also numerical measures of skewness we can use (more on that in Section 2.6).

Modality The *modality* of a distribution describes how many "peaks" it has. Figure 1.33 shows examples of three distributions with different modalities.

The number of modes in a histogram of the data is an important characteristic describing any dataset. You must be aware if you're dealing with multimodal distribution in order to choose appropriate statistical analysis procedures in later chapters.

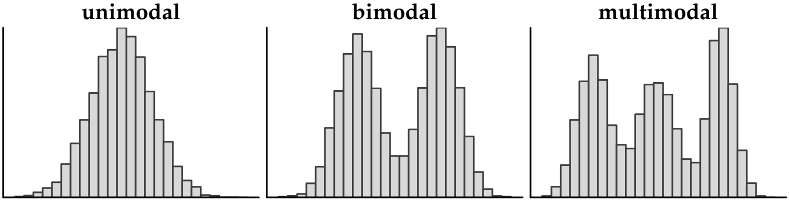


Figure 1.33: Histograms displaying datasets with three different *modalities*. Distributions with only one peak in are called *unimodal*. If we see two peaks in the histogram, we say the distribution is *bimodal*. Distributions with more than two peaks are called *multimodal*.

Intuitive interpretations of the mean

There is a useful physical analogy for understanding the mean $\overline{\mathbf{x}}$ that I want you to know about. Imagine that each data point has some weight to it, let's say one gram per data point. The location of the mean corresponds to the *centre of mass* of this distribution of weights. If all the weights were placed on a long ruler and lifted into the air, then you would be able to balance the ruler using a single finger by supporting the ruler at the location of the arithmetic mean (the centre of mass), as shown in Figure 1.34. Values smaller than the mean tend to tilt the ruler to the left of your finger, but values larger than the mean counterbalance them, so the ruler will stay balanced.

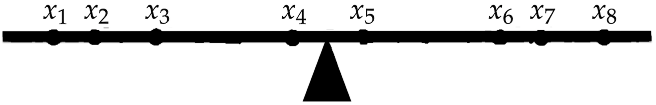


Figure 1.34: The mean $\overline{\mathbf{x}}$ is the centre of mass of a distribution of weights.

Another way to think of the mean is as a representative value for the data sample $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. Suppose we had to replace the values x_i with a single, common value repeated n times, while keeping the total sum of the values the same. We can accomplish this by repeating the mean n times $[\overline{\mathbf{x}}, \overline{\mathbf{x}}, \ldots, \overline{\mathbf{x}}]$. Figure 1.35 illustrates this process using the `score` variable $\mathbf{s} = [s_1, s_2, \ldots, s_{15}]$. Part (a) of the figure shows the observed 15 student scores s_i, represented as vertical bars. Part (b) shows how we can replace the data with 15 copies of a single representative value $[\overline{\mathbf{s}}, \overline{\mathbf{s}}, \ldots, \overline{\mathbf{s}}]$.

Linear interpolation for the median

When computing the median of a list that contains an even number of values, there is no value that splits the list into two equal parts.

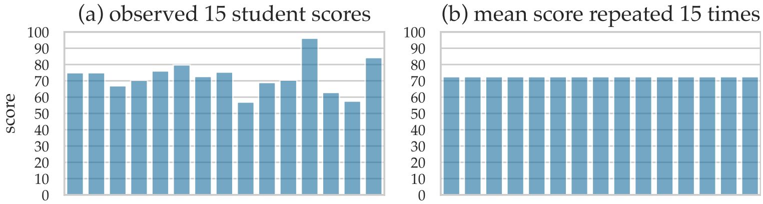


Figure 1.35: If we want to replace the 15 student scores by a single number repeated 15 times, we choose the mean score $\overline{\mathbf{s}}$. The total area of the bars in both plots is the same.

The convention in this situation is to create a new number that consists of a 50-50 mix of the two middle numbers, a process known as *linear interpolation*. For example, $\textbf{Med}([1,2,3,4]) = 2.5$ where the median 2.5 was computed by taking the average of the two middle numbers 2 and 3. Linear interpolation is also used when computing quantiles, quartiles, and percentiles. If you're interested to learn more about this, see the code examples and explanations in the notebook `13_descriptive_statistics.ipynb`.

Histogram binning

When we created the histogram of the `score` variable in Figure 1.22 (see page 64), we divided the data into bins of width 10. Choosing a different bin width results in differently shaped histograms, as we can see in Figure 1.36.

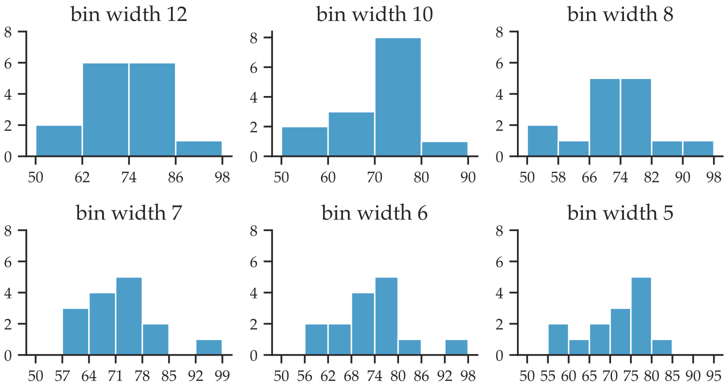


Figure 1.36: Histograms of the `score` with different bin widths.

The width of the bins changes the appearance of the histogram. If we choose wider bins, the histogram will show fewer details. In contrast, narrower bins show more details, but are less useful as a summarization tool. Bins that are too narrow show too much detail and distract from the overall shape of the data. Generally, we choose narrower bins when for bigger datasets (large n). In terms of the number of bins, there are formulas and heuristics for choosing the number of bins, such as $\sqrt{n}$, which work in most cases.

The starting point of each bin also impacts the overall shape of the histogram, even when the width of each bin is the same. Figure 1.37 shows histograms of the `score` data that all have bin width of 10, but have a different starting points.

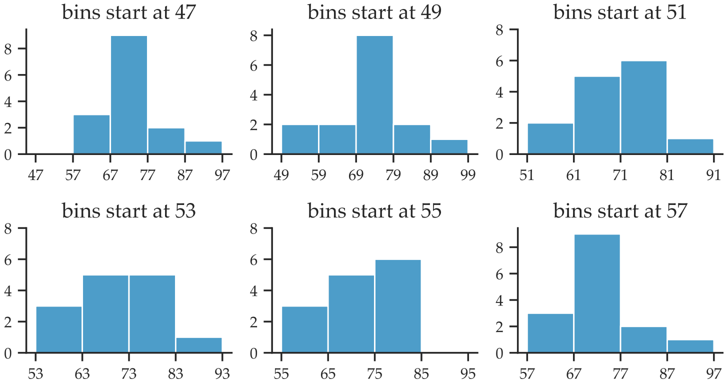


Figure 1.37: The `score` variable displayed as histograms with bin width 10 and bin boundaries starting at different locations. Note we obtain very different histogram shapes even though the data is the same in each plot.

The Seaborn function `histplot` will choose the bin width and starting point automatically if you don't specify the `bins` option. In certain situations, you might want to manually select the bin boundaries by specifying the `bins` option, as we did in code block 1.3.10. Using round numbers for the bin boundaries makes histograms easier to interpret.

Kernel density plots

One way to avoid the arbitrary bin width and bin start location choices of histograms is to draw a continuous *density* plot of the data. We can use the Seaborn function `kdeplot` to generate a *kernel density estimation* (KDE) plot of the data.

code 1.3.31

```
>>> sns.kdeplot(data=students, x="score", bw_adjust=0.2)
See Figure 1.38 (a).
>>> sns.kdeplot(data=students, x="score", bw_adjust=0.5)
See Figure 1.38 (b).
```

We won't discuss the details of the math behind kernel density plots, but I'll give you the general picture that you need to have in mind.

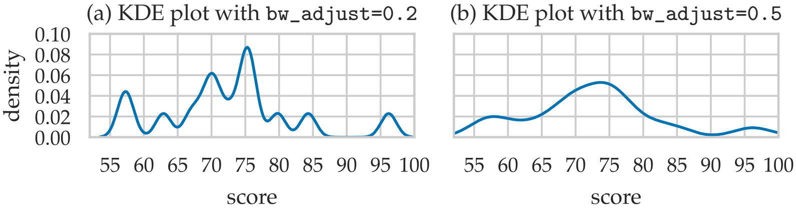


Figure 1.38: Kernel density plot of the `score` variable.

The word *kernel* is a translation of the French *noyeau*, which refers to the pit of a fruit. Look back at Figure 1.19 which shows a strip plot of the student scores, and think about each point as the pit of some fruit, say a peach. The flesh of each fruit is concentrated in a narrow region around the pit. You can think of each data point as a local region with a high density of peach flavour. The kernel density plot shows the combined distribution of "peach flavour" produced by all the data points. Regions where lots of values appear will have higher density.

The kernel density plots in Figure 1.38 are a continuous version of the histogram plots of the `score` variable. Instead of computing the frequencies of observations that fall within a discrete set of bins, we're treating each value as a smooth blob of data density centred at that point. The option `bw_adjust` controls the width of the density blobs around each point—how tightly concentrated the peach flavour is around each pit. The kernel density plot in Figure 1.38 (a) uses a small value for the `bw_adjust` option, so we can see the bumps around the individual data points, while the plot in part (b) of the figure uses more smoothing.

1.3.6 Discussion

Summary of different descriptive statistics

We'll now summarize the descriptive statistics we learned in this section, and mention the role these concepts will play in the rest of the book as we learn to model the properties of data distributions.

Measures of central tendency The mean and the median both describe the notion of "centre" of the data using a single number. The mean ($\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$) computes the average value, while the median ($\mathbf{Med}(\mathbf{x})$) computes the middle value (in a sorted list).

The mean is the most common measure of central tendency, and faithfully describes the "middle" of the distribution when the data is mostly symmetrical (see Figure 1.32), unimodal (see Figure 1.33), and contains no outliers. However, the mean is affected by the presence of skewness or outliers, which tend to "unbalance" the distribution to one side, and shifts the mean (centre of mass) in the direction of the imbalance. In extreme cases (heavily skewed data or many outliers), the mean can be far from the region where most of the data is situated.

In contrast, the median is not affected by the presence of outliers in the dataset. For example, if the largest value in the data is 100 or 10000, it will make no difference to the median. The median is the preferred measure of central tendency whenever data contains outliers, or is heavily skewed. The median is useful for describing distributions of any shape, and also works for ordinal data.

Measures of position The **Min**, the **Max**, the quartiles ($\mathbf{Q}_1$, $\mathbf{Q}_2$, $\mathbf{Q}_3$), and the percentiles give us the locations of specific values in the sorted data, which tells us a lot of useful information about the distribution of the data.

The percentiles allow us to know where a particular value fits in the distribution. For example, if your grade on a standardized test is at the 95^{th} percentile of all the grades on this test, this means 95% of other people got a grade lower than you, and only 5% of people got a higher grade.

Measures of dispersion The sample variance ($s_{\mathbf{x}}^2 = \mathbf{Var}(\mathbf{x})$) and the sample standard deviation ($s_{\mathbf{x}} = \mathbf{Std}(\mathbf{x})$) are both measures of dispersion calculated relative to the mean $\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$. These are the most common measures of dispersion, and are widely used for their practical and theoretical properties. The interquartile range ($\mathbf{IQR}(\mathbf{x}) = \mathbf{Q}_3(\mathbf{x}) - \mathbf{Q}_1(\mathbf{x})$) also tells us about the dispersion of the data, since it gives the width of the interval that contains the middle 50% of the data points. The range ($\mathbf{Range}(\mathbf{x}) = \mathbf{Max}(\mathbf{x}) - \mathbf{Min}(\mathbf{x})$) quantifies the dispersion by telling us the width of the overall interval where the data falls. The range is very sensitive to the presence of outliers. The span of the whiskers in a Spear-Tukey box plot (see Figure 1.24) is a better measure of overall dispersion, since they exclude the outliers.

Always plot your data!

Numerical summaries are useful for describing the properties of data samples, yet you shouldn't depend solely on them. Wildly different datasets can have identical summary statistics, as illustrated by the twelve data samples shown in Figure 1.39. Let these examples serve as a cautionary tale about the hidden structure in data, that you might miss if you rely only on numerical summaries.

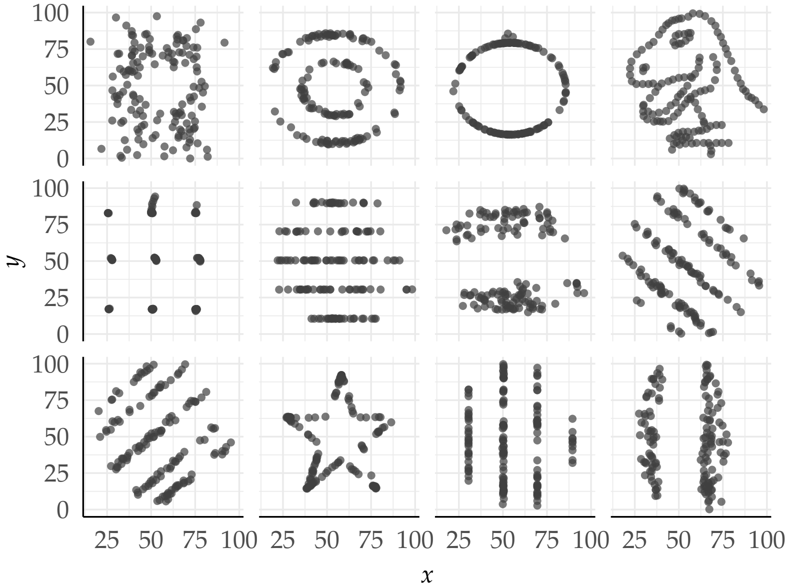


Figure 1.39: All the plots shown have the same descriptive statistics **Mean**(**x**) = 54.3, **Mean**(**y**) = 47.8, **Std**(**x**) = 16.8, **Std**(**y**) = 26.9, and **Corr**(**x**, **y**) = −0.1, but clearly show a very different relationship between the **x** and **y** variables. See the paper [MF17] for more details.

Which plot to use?

The plot we choose for visualizing a dataset depends on the specific characteristics we're interested in, and the purpose of the analysis. Here are some general comments about the strengths and weaknesses of the statistical plots we that discussed in this section:

- Strip plots (`sns.stripplot`) are the best choice for small datasets since they show the raw data, without any aggregation or use of abstract representations. Strip plots are not a good choice for large datasets since data points will tend to overlap. For medium-sized datasets, we can avoid overlapping points using the option `jitter`, which introduces a random vertical displacement for each data point. We can also use the option `alpha=0.5` to make the points half-transparent.
- Histograms (`sns.histplot`) are a good choice for medium and large datasets. The binning process creates a useful summary

of the data, and we can achieve different level of summarization by varying the bin widths. Histograms are particularly good at showing the shape of the data (skew and modality).

- Kernel density plots (`sns.kdeplot`) are similar to histograms, but without the discrete binning process.
- Box plots (`sns.boxplot`) show the exact position of the quartiles of the distribution, and provide a special treatment for the outliers, which can be very helpful to discover issues with the data. Box plots are better than histograms when used for comparing different distributions. One drawback of box plots is their high-level of abstraction—we cannot see the data distribution or the number of observations from a box plot, but only the information from the *five-number summary*.
- Scatter plots (`sns.scatterplot`) are the standard way to plot the relationship between numerical variables. Scatter plots are basically two-dimensional strip plots, so they have the same problem with overlapping data points. We can visualize large datasets using two-dimensional histograms (`sns.histplot`) or two-dimensional kernel density plots (`sns.kdeplot`).
- Bar plots (`sns.countplot`) are the usual way we visualize categorical data, although a one-way table numerical summary generated using the `.value_counts()` method often provides enough information. For bivariate categorical data, the grouped and stacked bar plots are two standard visuals for comparing two variables. Two-way tables generated using the Pandas function `pd.crosstab` are also very useful.

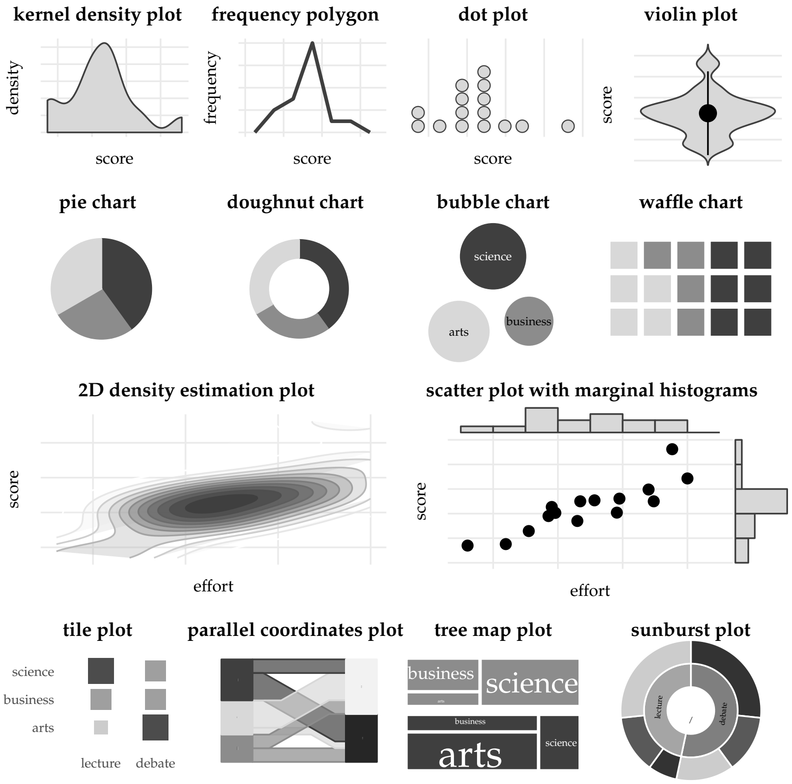
We usually need to generate multiple types of plots to get a comprehensive understanding of the data.

More plots

In this section, we introduced the most common statistical plots, but there are many more types of data visualizations out there! Graphical representations of data is a rapidly evolving field, especially with the use of interactive elements, animations, and 3D capabilities. It's not possible to cover the entire rich ecosystem of data visualizations in one section, so I encourage you to explore other ways to visualize data on your own. As inspiration, Figure 1.40 shows a few other ways we could have visualized the students dataset.

The importance of descriptive statistics

In this section, we learned how to summarize data using easily interpretable numerical descriptions and plots. These data summa-



kernel density plot

frequency polygon

dot plot

violin plot

pie chart

doughnut chart

bubble chart

waffle chart

2D density estimation plot

scatter plot with marginal histograms

tile plot

parallel coordinates plot

tree map plot

sunburst plot

Figure 1.40: Examples of the multitude of data visualization options.

rization techniques will be used throughout the rest of the book, whenever we'll be working with data, which is *all the time*!

One thing I want you to remember is to **always plot your data!** Never start doing any statistical analysis before getting to know the data. I promise this will make your life easier and potentially save you lots of future headaches. You're welcome.

In the next chapter, we'll learn about probability theory, which is a framework for describing data variability through the use of mathematical models. Concepts like the mean and the standard deviation that we learned in this section will also be useful for describing the properties of probability models. Probability theory is the main tool we use to formulate statistical questions.

Later on in the statistics chapters, we'll learn how numerical summaries like the mean $\overline{\mathbf{x}} = \mathbf{Mean}(\mathbf{x})$ and the standard deviation $s_{\mathbf{x}} = \mathbf{Std}(\mathbf{x})$ are used as part of the *statistical inference* process, and help us answer statistics questions. Indeed, the descriptive statistics calculations we learned in this section are the foundation for understanding the statistical inference topics we'll learn in Chapter 3.

Links

[Gallery of data visualizations produced using Seaborn]
`https://seaborn.pydata.org/examples/index.html`

[Seaborn tutorials featuring lots of useful plot examples]
`https://seaborn.pydata.org/tutorial.html`

[A collection of dataviz caveats to avoid]
`https://www.data-to-viz.com/caveats.html`

[Info about different methods for computing quantiles]
`https://en.wikipedia.org/wiki/Quantile`

[Lots of details about creating histograms]
`https://tinlizzie.org/histograms/`

[*40 years of boxplots* by H. Wickham and L. Stryjewski]
`https://vita.had.co.nz/papers/boxplots.pdf`

[A visual vocabulary for select the optimal data visualizations]
`https://ft.com/vocabulary`

[A gallery of interesting data visualizations]
`https://xeno.graphics`

1.4 Data problems

Given the central role that data plays in all of statistics, it's important that you get some hands-on experience with data manipulation and data visualization tasks. This is why I've prepared this set of problems for you to practice your knowledge of the concept definitions, Pandas and Seaborn functions, and calculating descriptive statistics. Working on the problems will also help you become familiar with the datasets that we'll use throughout the book.

P1.1 Create a bar plot to compare the mean `scores` for the `debate` and `lecture` groups.

TODO FINISH THIS

Hint: Use `sns.barplot`.

P1.2 To give an idea about dispersion within each group, we show *error bars* ⌶ that extend from **Mean** − **Std** to **Mean** + **Std**.

TODO FINISH THIS

Hint: Use the option `errorbar` on the `sns.barplot` function.

Problem ideas:

- numerical summary stats calculations (using `df.describe()` and from scratch)
- data plotting and visualizations
- guided pandas+seaborn problems asking to reproduce calculations on datasets similar to Circled6 visitors, ② eprices, and ③ students (practice computing descriptive statistics and visualizations)
- outliers pathologies (example calculations where non-robust statistics like the mean are heavily influenced by outliers)
- dangers of biased sampling simulation (compute descriptive statistics from biased samples, and show discrepancy with true population parameters)
- tidification: convert wide data to tall data (melt)
- data cleaning of Howell Excel file (rename columns, dropnan, etc) to produce howell.csv

Bibliography

[DLL17] Marie Delacre, Daniël Lakens, and Christophe Leys. Why psychologists should by default use welch's t-test instead of student's t-test. *International Review of Social Psychology*, 30(1):92–101, 2017. https://pure.tue.nl/ws/portalfiles/portal/80459772/82_534_3_PB.pdf.

[Fis35] R. A. Fisher. *The Design of Experiments*. Oliver and Boyd, 1935.

[LM08] Lawrence M Leemis and Jacquelyn T McQueston. Univariate distribution relationships. *The American Statistician*, 62(1):45–53, 2008. http://www.math.wm.edu/~leemis/2008amstat.pdf.

[MF17] Justin Matejka and George Fitzmaurice. Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 1290–1294, 2017. https://www.autodesk.com/research/publications/same-stats-different-graphs.

[Sav14] Ivan Savov. *No bullshit guide to math and physics*. Minireference Co., fifth edition, 2014. See https://gum.co/noBSmathphys.

[Sav17a] Ivan Savov. *No bullshit guide to linear algebra*. Minireference Co., second edition, 2017. See https://gum.co/noBSLA.

[Sav17b] Ivan Savov. Taming math and physics using SymPy. 2017. https://minireference.com/static/tutorials/sympy_tutorial.pdf.

[stu]

[Tho14] Silvanus P Thompson. *Calculus made easy*. Macmillan, second edition, 1914. https://www.gutenberg.org/ebooks/33283/.

[Wic14] Hadley Wickham. Tidy data. *Journal of statistical software*, 59(1):1–23, 2014. https://www.jstatsoft.org/article/view/v059i10.