Data Science

thomas mcandrew

March 3, 2022

Contents

List of Figures

List of Tables

Part I

Probability

1

Sets, the sample space, and probability

thomas mcandrew

Lehigh University

CONTENTS

1.1 Introduction

When we decide to study a topic, we often have in mind a population, a protocol to observe members of our population, a hypothesis that posits some aspects of our observed members. We collect data, we compile results. At the close of our study we want to know how the results ("what happened") contribute to our hypotheses ("what we thought may happen") and, importantly, the chances that if we reproduced the same study that the results would lead us towards the same conclusions.

For example, suppose we decide to study the impact of percutaneous intervention (PCI) versus optimal medical therapy (OMT) to treat patients how have had a myocardial infarction, often called a "heart attack". Patients are enrolled in a study and randomly assigned PCI or OMT. We hypothesize that PCI will result in a prolonged life from when the intervention took place onward. After our study we find patients who underwent PCI live on average 100 days longer than those who underwent OMT. However, it feels unsatisfactory to have a single number describing how effective PCI was at prolong life. We may want to know the chances PCI prolongs life 80 days, 30 days, or maybe zero days, compared to OMT.

We want to understand how likely we are to see specific events. The formal method of assigning chances to a set of events is called probability theory.

1.2 Probability lingo

1.2.1 Sets

We define a **set** as a collection of items, sometimes called elements. A set is typically given a capital letter (for example A) and the elements are included inside curly braces.

Here, we define three sets

$$A = \{a, b, c\} \tag{1.1}$$
$$C = \{-1, 1, 3.14\} \tag{1.2}$$
$$Z = \{\text{orange}, 1, \text{puppy dog}\} \tag{1.3}$$

The elements inside a set are unordered and do not necessarily need to be numbers. A set can contain any object. The set A could have been defined as $\{a, b, c\}$ or $\{c, b, a\}$. We use the symbol $\in$ to express that an element "is a member of" a set. For example, $a \in A$ can be read "the element a is a member of the set A" or "the element a is in the set A". We can also communicate when an element is not a member of a set with the symbol $\notin$. For example, $c \notin C$ is read "the element c is not a member of (not in) the set C".

Generating sets: We can define a set by enclosing in brackets each individual element. $S = \{a, b, c, 10\}$. However, some sets may be easier to build with properties. A property is a function from elements to either true or false. For example we could define the property $P(x) = 1$ when x is even and 0 when x is odd. To build a set that contains elements with a specific property, we write $S = \{x|P(x)\}$ or $S = \{x|x \text{ is an even integer}\}$. This is read "the elements x such that x is an even integer". The vertical bar replaces the words "such that".

Two sets are **equal** is they contain the same elements. In other words, if for an element x, $x \in A$ implies that $x \in B$ **and** if for an element y, $y \in B$ implies that $y \in A$ then the set A and set B are equal. If a set A and set B are equal we write $A = B$. If two sets do not contain the same elements then they are **unequal** and we write $A \neq B$.

Lets look at an example.

$$A = \{a, b, c\} \tag{1.4}$$
$$B = \{b, a, c\} \tag{1.5}$$
$$C = \{b, c\} \tag{1.6}$$

Above, $A = B$ but $A \neq C$ and $B \neq C$.

A set A is a **subset** of a second set B if all of the elements in A are also elements in B. We can say that A is a subset of B if every element in x implies that element is in B, or $x \in A$ implies $x \in B$. We write $A \subset B$ to denote that A is a subset of B. To denote that a set A is **not a subset** of B, we write $A \not\subset B$. In the example above, the C is a subset of A $C \subset A$, but A is not a subset of C, or $A \not\subset C$.

We can build new sets by operating on two or more existing sets.

Set **intersection** takes as input two or more sets, A,B, and returns a set that contains all the elements that are in both A <u>and</u> B. We use the symbol "cap" to denote set intersection $A \cap B$ and often say "A intersect B".

For the above sets A and C, their intersection is

$$A \cap C = \{c, b\} \tag{1.7}$$

because the element c belongs to both set A and set C and because the element b belongs to A and C.

We can intersect more than two sets.

Let $A = \{a, b, 10, 12\}$, $Z = \{b, 10, -1\}$, $U = \{a, b, c, d, 10\}$. Then

$$A \cap Z = \{b, 10\} \tag{1.8}$$

and

$$A \cap Z \cap U = \{b, 10\}. \tag{1.9}$$

If A is a subset of B, $A \subset B$, then the only elements both sets have in common are those in A. The intersection of A and B is then

$$\begin{aligned} &\text{If } A \subset B \text{ then} \\ &A \cap B = A. \end{aligned} \tag{1.10}$$

Set **union** takes as input two or more sets and output a set that contains all the elements that belong to first set <u>or</u> the second set <u>or</u> the third set, and so on. We use the "cup" symbol to denote set union. As an example, consider the sets A, Z, and U to be the same as they are above. Then A **union** Z is

$$A \cup Z = \{a, b, 10, 12, -1\} \tag{1.11}$$

This is because each of the above elements belongs to at least one of the sets A and Z. As another example,

$$A \cup Z \cup U = \{a, b, c, 10, 12, -1\} \tag{1.12}$$

Intersection and union are the most common set operations. Before we define the set operation **complement**, we need to discuss two special sets.

The **universal set** is the set of all elements. We denote the universal set with a $\mathcal{G}$. The set which contains no elements is called the **empty set**, and we denote the empty set as $\emptyset$.

If a set A and set B have no elements in common, then $A \cap B = \emptyset$. We often say that the set A and B are **disjoint**. For example,

$$\begin{aligned} A &= \{1, 3, 5\} \\ Q &= \{2, 4, 6\} \\ A \cap Q &= \emptyset \end{aligned} \tag{1.13}$$

Because the intersection between A and Q is empty, these sets are disjoint.

Now for our final set operation—complement. The set complement takes as input a single set, A, and outputs a set with elements that are members of $\mathcal{G}$ but are not members of A. We denote set complement in one of two ways: A^c or A'.

Set operations example

Let's look at an example of how these sets and set operations work. Define the universal set, $\mathcal{G}$, to be the set of all positive integers (i.e. $1, 2, 3, 4, \cdots$). Lets define the sets $A_1 = \{1\}, A_2 = \{1, 2\}, A_3 = \{1, 2, 3\}, \cdots, A_n = \{1, 2, 3, 4 \cdots, n\}$. We use a subscript and a number to "index" our sets. An index is an easy way to keep track of sets (and many mathematical objects).

$$\begin{aligned} A_1 \cap A_2 &= \{1\} \\ A_1 \cap A_5 &= \{1\} \\ A_3 \cap A_8 &= \{3\} \end{aligned} \tag{1.14}$$

Note that $A_3 \cap A_8$ is not the value 3, but the set {3}. The examples above suggest a pattern for any set A_i and A_j where $i \leq j$:

$$A_i \cap A_j = \{i\} \tag{1.15}$$

Lets look at set union in this example

$$\begin{aligned} A_1 \cup A_2 &= \{1, 2\} \\ A_1 \cup A_5 &= \{1, 2, 3, 4, 5\} \\ A_3 \cup A_8 &= \{1, 2, 3, 4, 5, 6, 7, 8\} \end{aligned} \tag{1.16}$$

and we see the following pattern for $i \leq j$

$$A_i \cup A_j = \{1, 2, 3, 4 \cdots, j\} = A_j \tag{1.17}$$

Because we defined a universal set, we can look at set complement

$$\begin{aligned} A_1^c &= \{2, 3, 4, 5, \cdots\} \\ A_5^c &= \{6, 7, 8, 9 \cdots\} \\ A_8^c &= \{9, 10, 11, 12, \cdots\} \end{aligned} \tag{1.18}$$

1.3 Applying set theory to probability

1.3.1 Foundation

The ideas about sets and set operations are the foundations for how we think about probability, about experiments, and hypotheses. We only need to recast the above set ideas to results from an experiment.

We use $\mathcal{G}$ to define the set of all possible outcomes from an experiment and call this set the **sample space**. The term "experiment" has a broad meaning. An experiment can mean everything from a randomized controlled trial to an observational study. Here an **experiment** is the process that generates outcomes.

An **outcome** is defined as an element of the sample space. An outcome is a single observation from an experiment, and we define an **event** as a set of outcomes. Most often we use o_i to denote an outcome and E_i to denote an event.

The **probability** of an event E is a mapping from the set E to a number between, or equal to, the values 0 and 1. We use $P(E)$ to denote the probability of event E. We require that the probabilities assigned to individual sets consisting of a single element in E add up to the probability of E. Lets suppose there are n outcomes in the event E. Then

$$\begin{aligned} E &= \{o_1, o_2, o_3, \cdots, o_n\} \\ P(E) &= P(\{o_1\}) + P(\{o_2\}) \cdots P(\{o_n\}) \end{aligned} \tag{1.19}$$

We further require $P(\mathcal{G}) = 1$. In other words, the probability that something happens is certain. Note that we do not need to describe how we assign probabilities to events, we only describe what values we expect those numbers to be.

Lets further detail relationships between probabilities of sets that we would expect.

1. If $A \subset B$ then $P(A) \leq P(B)$.
2. $P(A \cup B) \leq P(A) + P(B)$ (why?)
3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
4. If A and B are disjoint then $P(A \cup B) = P(A) + P(B)$

The following three axioms (knowledge we assume true without proof) are called **Kolmorogov's axioms** ()

1. For any event E, $P(E) \geq 0$
2. $P(\mathcal{G}) = 1$
3. If E_1 and E_2 are disjoint then $P(E_1 \cup E_2) = P(E_1) + P(E_2)$

Example: When we want to compute the probability that an outcome fall in any one of the events, say E_1, E_2, E_3, knowing that these events have no outcomes in common, that they are disjoint, makes the computation easy. An intuitive way to think about events that are disjoint is if they cannot all occur at the same time. Suppose we wish to study the prevalence of stroke among patients who are older than sixty-five. Strokes can be categorized into transient ischemia attacks (TIA), ischemic stroke (IS), and hemorrhagic stroke (HS). Further lets assume the following probabilities for each event: $P(\{\text{TIA}\}) = 0.15$, $P(\{\text{IS}\}) = 0.75$, $P(\{\text{HS}\}) = 0.10$, and that these events cannot occur simultaneously. Then the probability of the event $E = \{TIA, HS\}$ can be broken into the union of two disjoint events $E = \{TIA\} \cup \{HS\}$ and we can use what we know about these disjoint events to compute the probability of E

$$\begin{aligned} P(E) &= P(\{\text{TIA}, \text{HS}\}) && (1.20)\\ &= P(\{\text{TIA}\} \cup \{\text{HS}\}) && (1.21)\\ &= P(\{\text{TIA}\}) + P(\{\text{HS}\}) \quad \text{disjoint} && (1.22)\\ &= 0.15 + 0.75 = 0.90 && (1.23) \end{aligned}$$

1.3.2 The principle of equally likely outcomes—one way to assign probabilities

There are many different ways to assign probabilities to events. In this text, we will only consider using the principle of equally likely outcomes to assign probabilities. The **principle of equally likely outcomes** (PELO) states that the probability we assign to every event which contains a single outcome, $E_i = \{o_i\}$, in a sample space $\mathcal{G}$ is equal.

We can use this principle to assign probabilities first to every individual outcome and then to arbitrary events. Assume the PELO is true, then

$$\begin{aligned} o_1, o_1, o_1, \cdots, o_n \in \mathcal{G} && (1.24)\\ E_i = \{o_i\} && (1.25)\\ E_1 \cup E_2 \cup E_3 \cup \cdots \cup E_n = \mathcal{G} \ \text{(why?)} && (1.26)\\ P(E_1 \cup E_2 \cup \cdots \cup E_n) = P(\mathcal{G}) = 1 && (1.27)\\ P(E_1 \cup E_2 \cup \cdots \cup E_n) = P(E_1) + P(E_2 \cup \cdots \cup E_n) \ \text{(disjoint)} && (1.28)\\ P(E_1) + P(E_2) + \cdots P(E_n) = 1 && (1.29)\\ n \cdot p = 1 \ \ (p \text{ is the constant prob. we are after}) && (1.30)\\ p = 1/n && (1.31) \end{aligned}$$

The PELO tells us that the probability of an event with a single outcome is equal to one divided by the total number of outcomes in the sample space. The PELO also tells us that for a sample space with n elements the probability of an event $E = \{o_1, o_{20}, o_4\}$ that contains three outcomes is $P(E) = \frac{3}{20}$.

An example when, and when not, the PELO applies.

PELO only works when we expect each outcome in our sample space to be equally likely.

Example 1 Suppose our experiment is a coin toss and we will observe whether or not the coin lands heads up (H) or tails up (T). If we assume the coin has not been altered in any way, then PELO applies. Our sample space is $\mathcal{G} = H, T$. Because there are two outcomes in our sample space, $P(\{H\}) = 1/2$ and $P(\{T\}) = 1/2$.

Example 2 Suppose for our experiment we observe for one year a single patient who was admitted to the hospital because of an influenza infection and we plan to observe if that patient returns to the hospital because of a second infection. Our sample space contains one outcome R if the patient is re-admitted to the hospital within one year and a second outcome if they are no re-admitted (N). Our sample space is $\mathcal{G} = R, N$. If we applied the PELO to this problem we would have to assume the probability this patient returns, or does not return, are equal. Intuitively,it feels unreasonable to assume these two events have equal probabilities.

We need an additional way to assign probabilities.

1.3.3 A frequentist approach to assigning probabilities

An empirical approach to assign probabilities to an event A with sample space $\mathcal{G}$ is to do the following:

1. Generate $1, 2, 3, 4, \cdots, N$ outcomes from $\mathcal{G}$
2. Define a variable $N(A)$ and set that variable to zero.
3. If the 1st outcome is a member of A than add one $N(A)$, otherwise move on
4. If the 2nd outcome is a member of A than add one to $N(A)$

   $\vdots$
5. If the Nth outcome is a member of A than add one to $N(A)$

The above algorithm defined a variable $N(A)$ that counts the number of outcomes that belong to A out of the N generated outcomes. We can assign to A the probability

$$P(A) = \frac{N(A)}{N} \tag{1.32}$$

that is, the number of times we observed an outcome that belonged to A divided by the number of outcomes we observed.

As an example, suppose we want to understand the probability that patients with non-medically treated type II diabetes transition to needing treatment within one year of their diagnosis. Our outcomes are $\mathcal{G}$ = {"need treatment" and "no treatment"}. To assign a probability to the event $E = \{\text{need treatment}\}$ using the frequentist approach we decide to request anonymized patient records from a set of hospitals, asking that each hospital only send those patients who had an original diagnosis of type II diabetes without medication and who have had a physician visit one year more later. After our data request, we received 5,000 patient records and find that 1,585 of these patients were asked to start medication for their type II diabetes within one year. A frequentist approach would assign

$$P(E) = 1,585/5,000 = 0.317 \tag{1.33}$$

and also assign to the event $F = \{\text{no treament}\}$

$$P(F) = 0.683 \text{ (why?)} \tag{1.34}$$

.

1.3.4 Products, Conditionals, Baye's Theorem, and Repeated Experiments

1.3.4.1 Product Sets

Suppose a novel vaccine was developed and we are asked to compare two probabilities: the probability a vaccinated patient is infected before 30 days after they receive their vaccine and the probability a patient who has not received a vaccine is infected before 30 days since they have enrolled in the experiment. To estimate these probabilities, we enroll 200 volunteer patients. We will assign 100 patients to be given the treatment and the remaining 100 patients will be observed without treatment.

Because of our experimental design, $P(\{\text{vaccinated}\}) = 1/2$ and $P(\{\text{not vaccinated}\}) = 1/2$ with a sample space $\mathcal{G}_{\text{assignment}}$, containing two outcomes $\{\text{vaccinated}, \text{not vaccinated}\}$. But our main interest is in the probability of infection within 30 days of enrolling in the study, and so our main interest is in the a related sample space $\mathcal{G}_{\text{infection}} = \{\text{infection}, \text{no infection}\}$. We need a way to combine these sample spaces together so that we can estimate the probability of a patient that is vaccinated and that is infected.

Up until this point we have learned how to assign probabilities to events in one sample space. Lets look at how we may assign events to a combination of, and later a sequence of, sample space**s**.

Products of sets: First, we'll need to talk about the product of two sets. Let $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$. Then the Cartesian product $C = A \times B$ is a set where each element in A is paired with each element in B.

$$C = \{(a,1),(a,2),(a,3),(b,1),(b,2),(b,3),(c,1)(c,2)(c,3)\} \tag{1.35}$$

We use the notation $(,)$, called a "tuple", to denote pairs. A **tuple** is an outcome belonging to a sample space built from the Cartesian product of many individual sample spaces. Tuples are ordered. The tuple $(a, 1) \neq (1, a)$. The above is called a Cartesian product because you could imagine creating a grid where the horizontal axis has gridlines at "a", "b", "c" and the vertical axis has grid lines at "1", "2", and "3". Where the gridlines intersect are the tuples that belong to C.

We can apply the Cartesian product to combine samples spaces. Define $\mathcal{G}_{\text{experiment}}$ as the Cartesian product between $\mathcal{G}_{\text{assignment}}$ and $\mathcal{G}_{\text{infection}}$, or

$$\mathcal{G}_{\text{experiment}} = \mathcal{G}_{\text{assignment}} \times \mathcal{G}_{\text{infection}} \tag{1.36}$$

This new sample space has the following outcomes:

$$\begin{aligned}\mathcal{G}_{\text{experiment}} = \{&(\text{vaccinated}, \text{infection}), (\text{vaccinated}, \text{not infected})\\ &, (\text{not vaccinated}, \text{infection}), (\text{not vaccinated}, \text{not infected})\}\end{aligned} \tag{1.37}$$

Our new sample space has 4 outcomes, two outcomes from the assignment sample space are paired with two outcomes from the infection sample space (2 outcomes $\times$ 2 outcomes equals 4 outcomes in the new space).

With this new space in hand, we can assign probabilities to the outcomes "vaccinated and infected" and the outcome "not vaccinated and infected". You may hear events like the above called **compound events**, or **joint events**.

Let's use the frequentist approach to estimate the probabilities of these four events. We enroll all 200 patients over a 6 months period and then follow each patient for 30 days. At 7 days, 14 days, 21 days, and at the end of there 30 day period they meet with a physician to relay information about how they feel that could indicate they had an infection. We collect the following data:

From our experimental evidence we would assign a probability of $P(\{\text{vacc}, \text{infected}\}) = 0.10$ to those who were vaccinated and infected and

Vaccinated	Infected	Frequency	Estimated Prob.
Yes	Yes	20	20/200 = 0.1
Yes	No	80	80/200 = 0.4
No	Yes	40	40/200 = 0.2
No	No	60	60/200 = 0.3

TABLE 1.1
Frequencies collected about our vaccination experiment

a probability of $P(\{\text{no vacc}, \text{infected}\}) = 0.20$ to those who were not vaccinated and were infected. But this was not what were were asked to compute. We were asked to compute the probability that someone is infected after they received (given that they already had) a vaccine, and the probability of infection given a patient was not vaccinated. Probabilities of one event, given we have observed another event are called **conditional probabilities**.

1.3.4.2 Conditional probabilities

Assume a sample space $\mathcal{G}$. We define the conditional probability of event A given event B as

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{1.38}$$

Example: Lets use the definition above to compute the probability of an event A given $\mathcal{G}$, the sample space (remember that the sample space is a set and so technically an event). By the definition of conditional probbility

$$P(A|\mathcal{G}) = \frac{P(A \cap \mathcal{G})}{P(\mathcal{G})} \tag{1.39}$$

We know that $P(\mathcal{G}) = 1$ (Kolmogorov axioms in **??**) and so

$$P(A|\mathcal{G}) = \frac{P(A \cap \mathcal{G})}{P(\mathcal{G})} \tag{1.40}$$

$$P(A|\mathcal{G}) = \frac{P(A \cap \mathcal{G})}{1} \tag{1.41}$$

$$= P(A \cap \mathcal{G}) \tag{1.42}$$

The set A must be a subset of $\mathcal{G}$ and so by (1.10)

$$P(A|\mathcal{G}) = P(A \cap \mathcal{G}) = P(A) \tag{1.43}$$

The above example shows that we can think of the event that we condition on as a new sample space. Lets return to our example of vaccination and the incidence of infection.

To compute the probability of infection given a patient was vaccinated we need to compute

$$P(\text{inf}|\text{vacc}) = \frac{P(\text{inf} \cap \text{vacc})}{P(\text{vacc})} \tag{1.44}$$

where the event "inf" contains those outcomes in $\mathcal{G}_{\text{experiment}}$ which have "infected" in the fist position of their tuple $(\text{infected}, \cdot)$ and the event "vacc" contains those outcomes which have "vaccinated" in the second position of their tuple $(\cdot, \text{vaccinated})$. Events which hold one or more positions in a tuple constant are often called **marginal events** and the associated probability is called a **marignal probability**. Then $P(\text{inf} \cap \text{vacc})$ are those outcomes with "infected" in the first position and "vaccinated" in the second position. We have a single outcome where this happens: {(infected, vaccinated)} and this outcome has a probability of $P(\text{inf} \cap \text{vacc}) = 0.1$.

Computing $P(\text{vacc})$ is only slightly more difficult. We can use what we learned about unions and disjoint events for help.

$$\begin{aligned} P(\text{vacc}) &= P(\{(\text{infected}, \text{vacc}), (\text{not infected}, \text{vacc})\}) \\ &= P(\{(\text{infected}, \text{vacc})\} \cup \{(\text{not infected}, \text{vacc})\}) \\ &= P(\{(\text{infected}, \text{vacc})\}) + P(\{(\text{not infected}, \text{vacc})\}) \\ &= 0.1 + 0.4 = 0.5 \end{aligned} \tag{1.45}$$

We arrive at our final result.

$$P(\text{inf}|\text{vacc}) = \frac{P(\text{inf} \cap \text{vacc})}{P(\text{vacc})} = \frac{0.1}{0.5} = 0.2 \tag{1.46}$$

and we can do the same for

$$P(\text{inf}|\text{no vacc}) = \frac{P(\text{inf} \cap \text{no vacc})}{P(\text{ no vacc})} = \frac{0.2}{0.5} = 0.4 \tag{1.47}$$

We report back to our experimental team that our estimated probability of infection from someone who received a vaccine is 0.2 and the probability of infection from someone who did not received a vaccine is 0.4, double the probability of a vaccinated individual.

Conditional probabilities give us, for free, an alternative way to compute the probability of the intersection of two sets. All we need to do is rearrange the definition of conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{1.48}$$

$$P(A \cap B) = P(B)P(A|B) \tag{1.49}$$

Equation (1.49) is called the **multiplication rule**. Lets explore how the multiplication rule may make computing probabilities easier.

Example: We want to compute the probability that it rains and we remember to leave the house with an umbrella. We can imagine our sample space having the following four outcomes: $\mathcal{G}$ = {(rain,remember),(rain,forgot), (no rain, remember), (no rain, forgot)}. We try to remember all the times that the outcome (rain,remember) occurred in our past—but we cannot. Instead, we can certainly estimate the probability we forgot our umbrella given that it rained because it was an unpleasant experience (for some). We estimate $P(\text{forgot}|\text{rain}) = 0.95$. To compute $P(\text{forgot} \cap \text{rain})$ all we need now is the probability it rains. We take a look at the weather station reports over the past year and use the frequentest approach to estimate the probability it rains:$P(\text{rain}) = 0.25$. So then the probability it rains and we forget our umbrella is

$$\begin{aligned} P(\text{forgot} \cap \text{rain}) &= P(\text{forgot}|\text{rain})P(\text{rain}) \\ &= 0.95 \times 0.25 = 0.2375 \end{aligned} \tag{1.50}$$

The multiplication rule makes computations easier because we can think of events occurring in sequence. First it rains and second we forget our umbrella.

Conditional probabilities also allow us to express what it means for two events to be **independent**. An event A is independent from an event B if

$$P(A|B) = P(A) \tag{1.51}$$

We know the event B occurred, but this has not changed the probability that A has occurred. When two events are independent we can compute their intersection more easily.

$$\begin{aligned} &\text{Event } A \text{ and } B \text{ are independent} \\ &P(A \cap B) = P(A|B)P(B) \qquad \text{multiplication rule} \\ &P(A \cap B) = P(A)P(B) \qquad \text{independence} \end{aligned} \tag{1.52}$$

If two events are independent than the probability they occur together is the product of their probabilities. Statistical independence can be difficult to see.

Example We are recruited to track the evolution of an infectious agent for a team of public health officials (PHOs). To support future strategic planning the PHOs want to know the impact of intervention X on increases or decreases in the incidence of this infectious agent. The PHO team collected for each county in their state whether the intervention was enacted, and whether the incidence of case counts of this infectious agent increased or decreased 60 days after the intervention was in place data. When we look at this table, our first thought may be that when the intervention is enacted there is a high probability that the infection rate decreases (0.525), evidence that this is an important intervention for preventing the spread of this agent.

Intervention	Raise in infection	Probability
yes	yes	0.225
yes	no	0.525
no	yes	0.075
no	no	0.175

TABLE 1.2
Probabilities estimated from our data using a frequentest approach.

But be careful. Let's compute the probability of intervention $P(\text{intervention})$ where we define the event intervention as $\{(\text{intervention}, \text{rise}), (\text{intervention}, \text{no rise})\}$ and the probability of a raise in infection $P(\text{raise})$ where the event raise is defined as $\{(\text{intervention}, \text{rise}), (\text{no intervention}, \text{rise})\}$. To do this we should define our sample space to be clear about what outcomes we can observe from our experiment.

Our sample space is the set of four pairs

$$\begin{aligned} \mathcal{G} = \{&(\text{intervention}, \text{raise}), (\text{intervention}, \text{lower}) \\ &, (\text{no intervention}, \text{raise}), (\text{no intervention}, \text{lower})\} \end{aligned} \tag{1.53}$$

and we want to compute $P(\text{intervention}) = 0.225 + 0.525 = 0.75$ (why?) and $P(\text{raise}) = 0.225 + 0.075 = 0.30$ (why?).

One way that can test whether intervention is independent of a rise or fall in incidence, is compare the probabilities we estimated from our data (in table 1.2) to the product of individual events we computed above. Lets look at an example. The probability of an intervention and rise in incidence was estimated from our data to be $P(\text{intervention}, \text{raise}) = 0.225$. Lets compare this probability to the product $P(\text{intervention}) \cdot P(\text{raise})$.

$$P(\text{intervention}) \cdot P(\text{raise}) = 0.75 \cdot 0.30 = 0.225 \tag{1.54}$$

We have a match. Our estimated probability of an intervention and a rise occurring together is equal to the probability that an intervention occurs and a rise in incidence occurs. We can do the same procedure above for the remaining three scenario.

We see that the probabilities we collected form the data match the product of the probabilities of each individual event (Table 1.3). Intervention is independent, does not change the probability, of a rise or fall in incidence of the infectious agent we are tracking. We will need to report these results to the PHO team and recommend they try alternative interventions to curb the spread of this agent.

Intervention	Raise	Probability from data	Probability assuming indep.
yes	yes	0.225	0.75 (0.30) = 0.225
yes	no	0.525	0.75 (0.70) = 0.525
no	yes	0.075	0.25 (0.30) = 0.075
no	no	0.175	0.25 (0.70) = 0.175

TABLE 1.3
Probabilities of the cooccurance of an intervention and the rise or fall of an infectious agent, and the probability of each of these four events assuming they are independent.

1.3.4.3 The multiplication rule as a tool to compute sequential events

The multiplication rule equips us with a way to break down an event that involves many simultaneous phenomena into a sequence of, potentially easier to compute, probabilities.

Example: Suppose we wish to study the impact diabetes and smoking may have on the probability of a stroke before the age of 50. We decide to observe a population of humans in the US who are all 18 years old and host annual check-ins to record whether each person has acquired diabates, is considered obese, and has experienced a stroke.

Our sample space is $\mathcal{G} = \{(x,y,z) | \; x \in \{\text{diab}, \; \text{no diab}\}$ and $y \in \{\text{obese}, \; \text{no obese}\}$ and $z \in \{\text{stroke}, \; \text{no stroke}\}\}$. For example, one outcome in our sample is $o = (\text{diab}, \text{no obese}, \text{no stroke})$ or the outcome that a patient has acquired diabates, is not considered obese, and has not experienced a stroke.

We may find it difficult to collect data that purposely recorded these three variables together, however it may be easier to find data that has paired diabetes with obesity and diabetes with stroke. The multiplication rule can give us some intuition about the outcomes we wish to study.

Define three events:

1. $D = \{(x,y,z) | x = diab\}$
2. $O = \{(x,y,z) | y = obese\}$
3. $S = \{(x,y,z) | z = stroke\}$

We may find information to assign a probability to O. Lets suppose we found $P(O) = 0.60$. Investigating further we find a dataset that compares those who have obesity and diabetes. This data may alow us to assign the following probabilities $P(D|O) = 0.5$ and $P(D|O^c) = 0.31$. Finally, we find a dataset that measured the prevalence of stroke among patients with-/without diabetes and obesity, giving us the following probability assignments: $P(S|D \cap O) = 0.2$, $P(S|D \cap O^c) = 0.1$, $P(S|D^c \cap O) = 0.15$, and $P(S|D^c \cap O^c) = 0.01$.

If we feel that the above information applies to our study, we can compute the probability of simultaneous events by breaking them into a sequence of conditional probabilities. Suppose we wish to compute $P(D \cap O \cap S)$. This can be separated using the multiplication rule:

$$\begin{aligned} P(D \cap O \cap S) &= P(S|O \cap D)P(O \cap D) && (1.55)\\ &= P(S|O \cap D)P(D|O)P(O). && (1.56) \end{aligned}$$

We can now compute the probability of our simultaneous event with the information we collected across several studies.

$$\begin{aligned} P(D \cap O \cap S) &= P(S|O \cap D)P(D|O)P(O) && (1.57)\\ &= 0.2 \times 0.5 \times 0.60 = 0.06 && (1.58) \end{aligned}$$

If we want, we can visualize all the conditional relationships related to D, O, and S with a **tree diagram**. A tree diagram starts with a dot that represents the "root". We pick an event, say O, and create two branches: one for when O occurs and one for when O^c occurs.

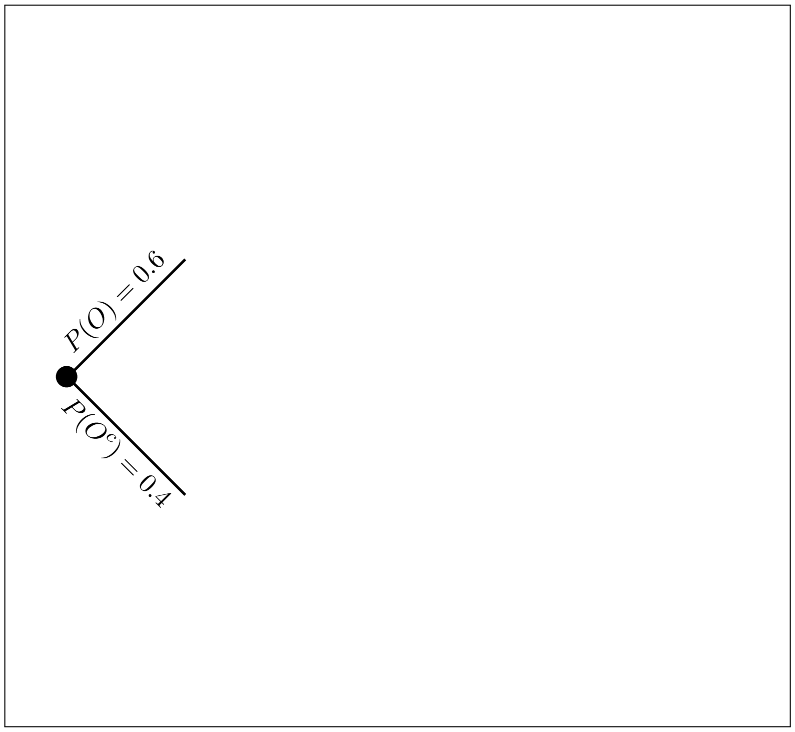


FIGURE 1.1

We pick the next event, say D, and create two branches from O and two branches from O^c for a total of four branches. Conditional on O, two branches represent the occurrence of D and occurrence of D^c. Conditional on O^c, two branches represent the occurrence of D and occurrence of D^c.

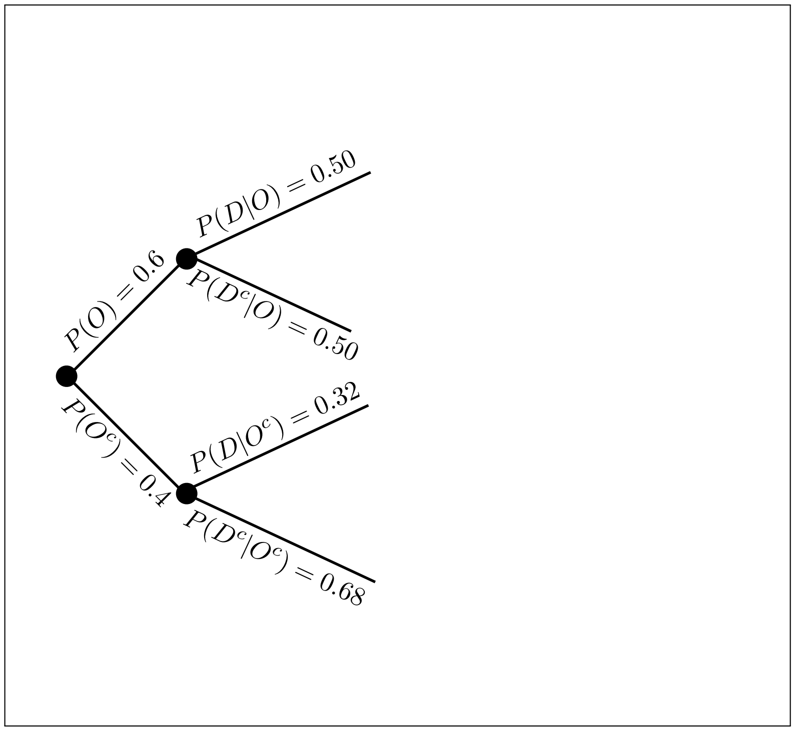
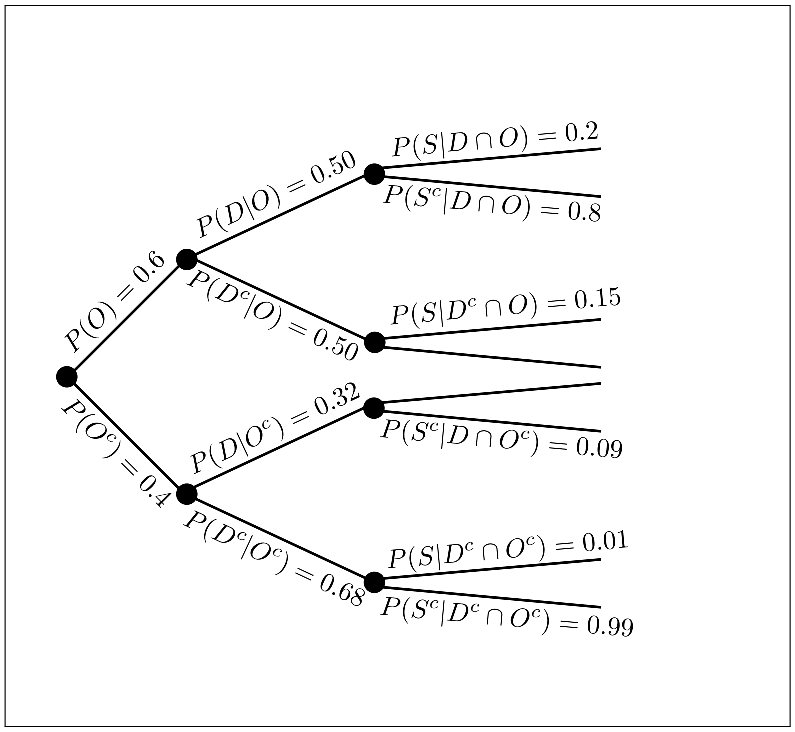


FIGURE 1.2

We can, in this example, do the same procedure for S that we did for D, but this time we need to look at the occurrence of S and S^c for the four newest branches we created.



$P(O) = 0.6$
$P(O^c) = 0.4$
$P(D|O) = 0.50$
$P(D^c|O) = 0.50$
$P(D|O^c) = 0.32$
$P(D^c|O^c) = 0.68$
$P(S|D \cap O) = 0.2$
$P(S^c|D \cap O) = 0.8$
$P(S|D^c \cap O) = 0.15$
$P(S^c|D \cap O^c) = 0.09$
$P(S|D^c \cap O^c) = 0.01$
$P(S^c|D^c \cap O^c) = 0.99$

FIGURE 1.3

1.3.4.4 Partitions and the law of total probability

The multiplication rule is one way to use conditional probabilities to make computing probabilities of the intersection of many events more manageable. We can also use conditional probabilities to simplify computing single events easier.

A **partition** of the event A is a collection of sets $\mathcal{P} = \{B_1, B_2, \cdots, B_n\}$ such that $A = (A \cap B_1) \cup (A \cap B_2) \cup (A \cap B_3) \cdots \cup (A \cap B_n)$ and for each i, k pair of sets $B_i \cap B_k = \emptyset$.

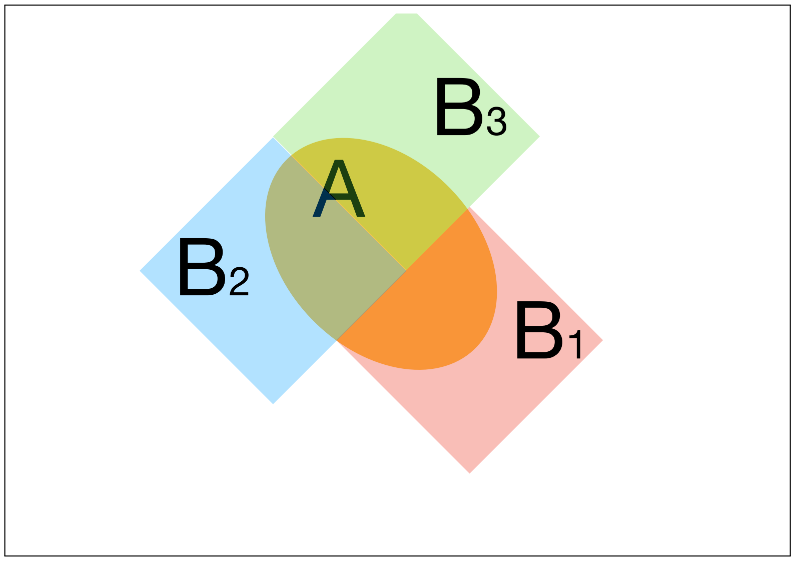


FIGURE 1.4
A partition $\mathcal{P} = \{B_1, B_2, B_2\}$ of the set A

Because B_i and B_k are disjoint for any choice of i and k, then $A \cap B_i$ and $A \cap B_k$ are also disjoint sets. We can use the disjoint property of our partition to break down the probability of A into the sum of probabilities of A intersect B_k for k from 1 to n.

$$\begin{aligned} P(A) &= P((A \cap B_1) \cup (A \cap B_2) \cup (A \cap B_3) \cdots \cup (A \cap B_n) && (1.59) \\ &= P(A \cap B_1) + P(A \cap B_2) + + \cdots + P(A \cap B_n) \quad \text{(Disjoint)} && (1.60) \end{aligned}$$

We can further manipulate the above using the multiplication rule to find

$$\begin{aligned} P(A) &= P(A \cap B_1) + P(A \cap B_2) + + \cdots + P(A \cap B_n) && (1.61) \\ &= P(B_1)P(A|B_1) + P(B_2)P(A|B_2) + \cdots + P(B_n)P(A|B_n) && (1.62) \end{aligned}$$

This is called **the law of total probability**. Intuitively the law of total probability says that the probability of an event A can be computed by first considering the probabilities of a collection of related events, and then considering the probability that A occurs given each related event.

Example: Let $\mathcal{G} = \{(a,1),(a,2),(a,3),(b,4)\}$ and further suppose we assign the following probabilities to each outcome.

Outcome	Probability
$(a,1)$	0.10
$(a,2)$	0.25
$(a,3)$	0.14
$(b,4)$	0.51

Define the events $A = \{(a,1),(b,4),(a,2)\}$. The finest (as opposed to course) partition is a a collection of sets where each outcome in A is in one, and only one, of the sets $B_1, B_2, \ldots$. For example a partition of A could be $\mathcal{P} = \{B_1, B_2, B_3\}$ where $B_1 = \{(a,1)\}$, $B_2 = \{(b,4)\}$, and $B_3 = \{(a,2)\}$. A courser partition is $\mathcal{P} = \{B_1, B_2\}$ where $B_1 = \{(a,1)\}$ and $B_2 = \{(a,2),(b,4)\}$.

1.3.4.5 Baye's Theorem

We have investigated how the conditional probability is related to simultaneous events, gives us a natural definition of Independence, and can allow us to compute the probability of an event if we have a partition of that event.

The final relationship we will explore is between two conditional probabilities.

Baye's Theorem relates two conditional probabilities to one another

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.63}$$

We can show that Baye's theorem is true from the definition of conditional probability.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \qquad \text{definition} \tag{1.64}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad \text{multiplication rule} \tag{1.65}$$

1.3.4.6 Repeated experiments

Many natural experiments involve repeated observations. It may be of interest to observe the whether and assign a probability to "sunny" or "cloudy" weather conditions. However, many vacationers, who plan to spend 2, 3, or more days at a remote destination may want to know the probability that the next X days are sunny.

The Cartesian product gives us a natural way to express repeated experiments. If we chose to repeat an experiment N times where each experiment produced an outcome in $\mathcal{G}$, we could imagine the set of outcomes as tuples of length N where each entry in the tuple is from our sample space. In other words, a single outcome in this repeated experiment would be $(o_1, o_2, o_3, \cdots, o_N)$ and this outcome is a member of the set $\mathcal{G} \times \mathcal{G} \times \mathcal{G} \times \cdots \times \mathcal{G}$.

Back to our vacation example. If a vacationer wanted to understand the chances they had of enjoying 5 sunny days in a row, then we know that we could defined a sample space $\mathcal{G} = \{\text{"sunny"}, \text{"not sunny"}\}$ and we also know that the vacationer is interested in outcomes in the sample space

$$(d_1, d_2, d_3, d_4, d_5) \in \mathcal{G} \times \mathcal{G} \times \mathcal{G} \times \mathcal{G} \times \mathcal{G} \tag{1.66}$$

where d_i is the outcome on day i. We can frame the problem, and clearly layout the potential outcomes, for example on outcome is ("sunny", "sunny", "not sunny", "sunny", "not sunny").

1.3.5 The datapoint, dataset, and dataframe

The sample space, event, and outcome are all potential results from an experiment. When we conduct an experiment we will generate an outcome from our sample space and call this realized outcome a **data point**.

Example: Consider the experiment of flipping a coin and recording whether the coin lands heads or tails side up. We can define a sample space $\mathcal{G} = \{H, T\}$ where H is an outcome that represents the coin landing heads up and T represents tails up. Up until this point we have structured our experiment, but we have generated no data. We flip the coin and the coin lands tails side up. Now we have performed an experiment and generated the data point T.

Now suppose that we conduct an experiment with the same sample space ($\mathcal{G}$) a number N times and with each experiment we record a data point. A tuple of data points d is called a **data set** $\mathcal{D} = (d_1, d_2, d_3, \cdots, d_N)$ where d_i is the data point generated from the i^{th} experiment. We say that we have <u>drawn</u> or that we have <u>sampled</u> a data set $\mathcal{D}$. Further, data points (d) are often called <u>realized</u> outcomes because they are no longer in a set of potential possibilities but are now determined items.

A data set $\mathcal{D}$ can be unwieldy depending on the number of data points, the complexity of the sample space, or both. A **data frame** is one way to organize a data set. A data frame $\mathcal{F}$ is a table where each data point d in a dataset $\mathcal{D}$

is represented as a row in the table and if the data point is a tuple then a separate column is created for each position in the tuple.

Example: Suppose we design an experiment to collect from humans predictions, two weeks ahead from the time of our experiment, of the number of incident cases and incident deaths at the US national level of COVID-19 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7523166/). We decide to collect from each human whether they are an expert in the modeling of infectious disease, a prediction of incident cases, and a prediction of incident deaths. We draw a data set $\mathcal{D}$ of 50 human judgement predictions. We can organize this data set into a data frame

Expert	Prediction of cases	Prediction of deaths
Yes	145	52
No	215	34
Yes	524	48
Yes	265	95
No	354	35

TABLE 1.4
Example data frame $\mathcal{F}$ built from a data set $\mathcal{D}$ that contains 5 data points where each data point is a tuple of length three.

Above, the first data point is $(Yes, 145, 52)$, the second data point is $(No, 215, 34)$, and so on until the last data point $(No, 354, 35)$. A data frame can also include informative information for others such as labels for each column.

1.4 Exercises

1.

$$A = \{1,2,3,4,5,6\}; \quad B = \{1,3,6\}$$
$$C = \{7\} \quad D = \emptyset$$

(a) Please compute $A \cap B$

(b) Please compute $A \cup C$

(c) Please compute $A \cup D$

(d) Please compute $A \cap D$

(e) Please compute $(A \cap B) \cup (C \cup D)$

2. Let the sample space $\mathcal{G} = \{1,2,3,4,5,6,7\}$
   (a) Please compute A^c
   (b) Please compute B^c
   (c) Please compute D^c
   (d) Please compute $\mathcal{G} \cap A$
   (e) Is $A \subset \mathcal{G}$?
   (f) Is $\emptyset \subset \mathcal{G}$?
3. Let the sample space $\mathcal{G} = \{0,1,2,a,b,c\}$ and let $A = \{0,1\}$, $B = \{x|x \text{ is a letter of the English alphabet}\}$
   (a) Please compute $A \cap B$
   (b) Please compute $A \cup B$
   (c) Please compute A^c
   (d) Is $A \cup B = \mathcal{G}$?
   (e) If we assigned probabilities to all outcomes, could $P(A \cup B) = 1$? why or why not?
4. Let $A = 0,1,2$ for some sample space $\mathcal{G} = \{0,1,2,3,4,5,6\}$. Further assume $P(A) = 0.2$.
   (a) Are the sets A and A^c disjoint? Why or why not.
   (b) Simplify $P(A \cup A^c)$ into an expression that involves $P(A)$ and $P(A^c)$.
   (c) Use Kolmogorov's axioms to show that $P(A) = 1 - P(A^c)$
5. Let $\mathcal{G} = \{x|x \text{ is a positive integer}\}$
   (a) Are the sets $\emptyset$ and $\mathcal{G}$ disjoint?
   (b) Simplify $P(\mathcal{G} \cup \emptyset)$ into an expression that involves $P(\mathcal{G})$ and $P(\emptyset)$
   (c) Use Kolmogorov's axioms to show that $P(\emptyset) = 0$
6. If $A = \{1,2,3\}$ and $B = \{2,3,4\}$ and $C = \{1,3\}$
   (a) Can $P(A) < P(B)$? Why or why not
   (b) Can $P(A) < P(C)$? Why or why not
7. Use what you know about the intersection, about subsets, and about probability to show that $P(A \cap B) \leq P(A)$. Hint: How are $A \cap B$ and A related?

8. Suppose we wish to study the reemergence of cancer among patients in remition. We collect data on 1,000 patients who are in cancer remition and follow them for 5 years. At five years we are interested in the probability of a second cancer.
   (a) Define a sample space $\mathcal{G}$ we can use to assign probabilities to a second cancer and no second cancer.
   (b) After five years of followup we find that 238 patients experienced a second cancer. Use the frequentist approach to assign probabilities to a second cancer and no second cancer.
   (c) If you collected data on 2,000 patients do you expect the probability of a second cancer to change? How do you expect the probability to be different for 2,000 patients than with 1,000 patients?
9. A study (link = https://www.science.org/doi/10.1126/science.abj8222 found that young adults were 32 times more at risk to develop multiple sclerosis (MS) after infection with the Epstein-Barr virus compared to young adults who were not infected by the virus. The experiment enrolled 10 million young adults and observed them for a period of 20 years.
   (a) Design a sample space if we wish to study outcomes that describe the number of young adults who develop MS.
   (b) Build the event (E_1) "less than 10% of young adults develop MS" using set notation.
   (c) Build the event (E_2) "less than 5% of young adults develop MS" using set notation.
   (d) Are E_1 and E_2 disjoint? Why or why not?
   (e) Can $P(E_1) < P(E_2)$?
10. Please compute the following
    (a) $A = \{1,2,3\}$ and $B = \{4,5,6\}$. Please compute $A \times B$ (Answer should be a set of tuples)
    (b) $A = \{1,2,3\}$. Please compute $A \times A$ (Answer should be a set of tuples)
    (c) How many elements are in $A \times A$ (looking for a number)
    (d) How many elements are in $A \times A \times A$? (looking for a number)
    (e) How many elements are in $A \times A \times A \times \cdots \times A$ where we take the Cartesian product N times? (looking for a number)
11. Define a sample space $\mathcal{G} = \{a,b,c,d,1,2,3,4,5\}$ and let $E_1 = \{1,3,5\}$, $E_2 = \{a,b,c\}$, $E_3 = \{a,d,5\}$. We further assign the following probabilities

Outcome	$P(\{\text{Outcome}\})$
a	0.10
b	0.05
c	0.15
d	0.02
1	0.14
2	0.25
3	0.08
4	0.04
5	0.17

(a) Compute $P(E_1)$ (looking for a number)

(b) Compute $P(E_2)$ (looking for a number)

(c) Compute $P(E_3)$ (looking for a number)

(d) Compute $P(E_1 \cap E_2)$ (looking for a number)

(e) Compute $P(E_1 \cap E_3)$ (looking for a number)

(f) Compute $P(E_2 \cap E_3)$ (looking for a number)

(g) Compute $P(E_1|E_2)$ (looking for a number)

(h) Compute $P(E_1|E_3)$ (looking for a number)

(i) Compute $P(E_2|E_3)$ (looking for a number)

(j) Compute $P(E_3|E_2)$ (looking for a number)

12. Define a sample space $\mathcal{G} = \{(a,1),(b,1),(c,1),(a,2),(b,2),(c,2)\}$ and let $E_1 = \{(a,1),(a,2),(c,2)\}$, $E_2 = \{(c,2),(a,1)\}$, $E_3 = \{(b,2)\}$. We further assign the following probabilities

Outcome	$P(\{\text{Outcome}\})$
(a,1)	0.05
(b,1)	0.22
(c,1)	0.15
(a,2)	0.02
(b,2)	0.13
(c,2)	0.43

(a) Compute $P(E_1)$ (looking for a number)

(b) Compute $P(E_2)$ (looking for a number)

(c) Compute $P(E_3)$ (looking for a number)

(d) Compute $P(E_1 \cap E_2)$ (looking for a number)

(e) Compute $P(E_1 \cap E_3)$ (looking for a number)

(f) Compute $P(E_2 \cap E_3)$ (looking for a number)

(g) Compute $P(E_1|E_2)$ (looking for a number)

(h) Compute $P(E_1|E_3)$ (looking for a number)

(i) Compute $P(E_2|E_3)$ (looking for a number)

(j) Compute $P(E_3|E_2)$ (looking for a number)

13. Given two events A and B, show that $P(A|B) \geq P(A \cap B)$ (looking for a short explanation and mathematical argument)

14. Suppose we wish to study adverse outcomes among patients who have unprotected left main disease (https://www.nejm.org/doi/full/10.1056/nejmoa1909406). In this experiment (called a clinical trial) we randomize patients to receive percutaneous intervention (PCI) or a coronary artery bypass graft (CABG). We wish to study the number of patients who received a PCI and experienced a myocardial infarction (MI) between the time they had their procedure and 5 years, however we only know three pieces of information: (i) the probability a patient was randomized to PCI was 0.5, (ii) the probability a patient was randomized to CABG was 0.5, and (iii) we know that the probability of a MI was 0.2 among patients who received a PCI.

    (a) Define a sample space that will allow us to compute the probability a patient was randomized to PCI and experienced an MI (looking for a description of the sample space and a set)

    (b) Please compute the probability a patient experiences an MI and was randomized to PCI. (looking for a number)

    (c) Please compute the probability a patient does not experience an MI and was randomized to PCI. (looking for a number)

15. If two events A and B are disjoint, are they also independent? (looking for a description and mathematical argument to backup your description.)

16. Researchers studied epidemiological characteristics of a variant of SARS-CoV-2 called B.1.526 (https://www.science.org/doi/10.1126/sciadv.abm0300). Suppose we too wanted to study the impact of B.1.526 on a population of patients who have been admitted to the hospital for COVID-19 by collecting each patient's age, whether they have were vaccinated against COVID-19, and whether their infection was due to the B.1.526 variant.

    (a) If we define our experiment as generating the above three pieces of information from a single patient, define a sample space ($\mathcal{G}$) for these potential outcomes. (looking for a short description and the sample space)

(b) Define a sample space if we repeated our above experiment 2 times. In other words, what would be our sample space if we collected information from 2 patients?

17. Can a data set ever have more elements than a sample space? Explain. (looking for a paragraph with some examples)

18. Suppose we collect the following data about the co-occurence of patients admitted to the hospital for influenza-like illness (ILI) and whether the patient does or does not work in a clinical setting. After data collection we estimate the following probabilities: The probability that a patient works in a clinical setting is 0.12. The probability that a patient who works in a clinical setting is admitted to the hospital for ILI is 0.2, and the probability a patient who does not work in a clinical setting was admitted to the hospital for ILI is 0.3.

    (a) Define a sample space to study outcomes related to the co-occurance of ILI/No ILI and patients who do/do not work in a clinical setting. (Looking for a set and a very brief description of an outcome or two.)

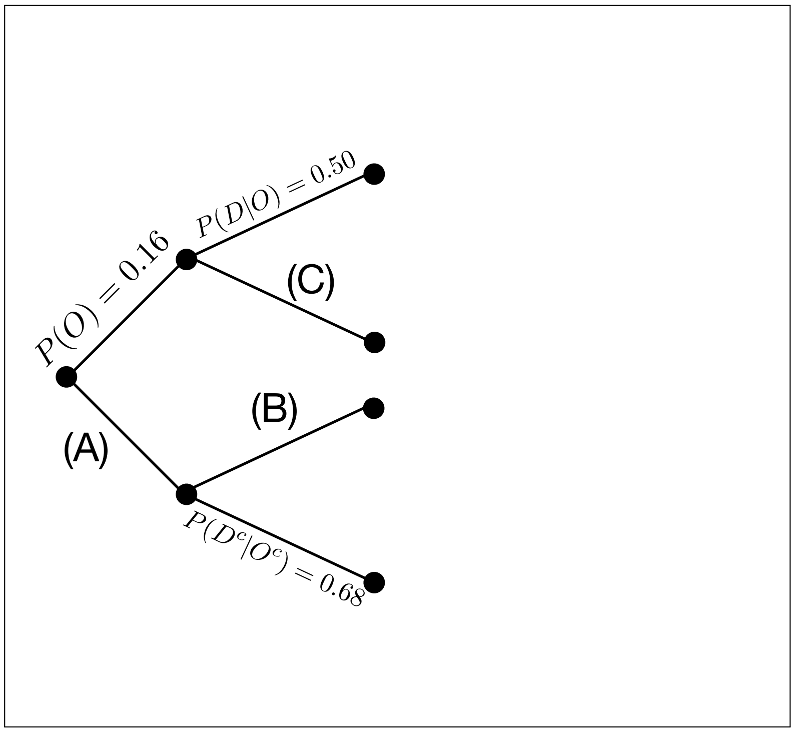
    (b) Compute the probability a patient is admitted to the hospitals for ILI and they work in a clinical setting. (Looking for an equation and number)

    (c) Compute the probability a patient is not admitted to the hospital for ILI and they work in a clinical setting. (Looking for an equation and number)

    (d) Compute the probability a patient is admitted to the hospital for ILI and they do not work in a clinical setting. (Looking for an equation and number)

    (e) Compute the probability a patient is not admitted to the hospital for ILI and they do not work in a clinical setting. (Looking for an equation and number)

19. Show that for two events A and B the $P(A|B)P(B) \leq P(B)$. Why is this the case intuitively? (Looking for two brief descriptions)



20. Please use the above tree diagram to answer the following questions.
    (a) Fill in the $P(O^c)$ which corresponds to (A) in the figure
    (b) Fill in the $P(D|O^c)$ which corresponds to (B) in the figure
    (c) Fill in the $P(D^c|O)$ which corresponds to (C) in the figure
    (d) Please compute $P(D)$
    (e) Please compute $P(D^c)$

21. Influenza is a contagious virus that enters the respiratory system from a human's nose, throat, or lungs. Common symptoms are cough, chills, and a fever. Suppose we collect data from a local hospital about those who were infected and not infected by the influenza virus and the above three symptoms. Let $\mathcal{G} = \{(x,y)|x$ is "Flu" or "not Flu" and $y \in \{$cough , chills , fever$\}\}$. Suppose further that we define events F = { (x,y) | x is "Flu" }, C = { (x,y) | y is "Cough" }, H = { (x,y) | y is "Chills" } , E = { (x,y) | y is "Fever" }, and that $P(C|F) = 0.5$ and $P(C|F^c) = 0.15$, $P(H|F) = 0.25$ and $P(H|F^c) = 0.01$, $P(E|F) = 0.98$ and $P(E|F^c) = 0.35$, $P(F) = 0.02$
    (a) A patient presents with the chills. What is the probability they have influenza?

(b) What symptom from the above, if present, would indicate with a high probability a patient has influenza?

1.5 Student contributions

Jessica Berman - Class of 2025-Advice: When I see a problem pertaining to conditional probability it helps me to talk the question out and actually say to myself "what is the probability of A given B has already occured" because P(A | B) is not clear to me without saying it aloud.

Jessica Berman - Class of 2025-Example: A fun fact about conditional probability is that it is used in politics. For example, if there are 4 candidates running in a presidential election, the chances of winning for each of them is 25

1.6 Glossary

Set:

Subset:

Set equality:

Set intersection:

Set union:

Universal set:

Empty set:

Sample space:

Experiment:

Outcome:

Event:

Probability:

Kolmorogov's Axioms:

Principle of Equally Likely Outcomes:

Product Set:

Compound event:

Conditional probability:

Multiplication Rule:

Independence:

2

Random variables

thomas mcandrew

Lehigh University

CONTENTS

2.1 Introduction

The foundations of probability are built on sets, yet data is more naturally stored and more easily computed on if it is represented numerically.

Random variables match each outcome in our sample space to a value on the number line.

In addition to computational advantages, random variables help us extract from our data the most important characteristics, and they serve as building blocks which we can use to create powerful models. Random variables are also a language we can use to communicate our modeling efforts to other mathematicians, statisticians, and data scientists.

Suppose we hypothesize that the frequency of social media posts on some popular outlet are related to influenza-like illness (ILI)—a syndromic diagnosis suggesting a patient may have influenza. A patient is diagnosed with ILI if their temperature is measured to be at or above 38C and symptoms resembling the flu. Because influenza is most active in winter and spring, we collect a random sample, each day, of social media posts from September to May and in addition we collect the proportion of patients who are admitted to the hospital and are diagnosed with influenza-like illness at the US national level.

The above hypothesis, data collection, and future inference has numerous details. However, we will see shortly that we can simplify our hypothesis by using random variables.

2.2 Maps from the sample space to the number line

Given a sample space $\mathcal{G}$, a **random variable**, (e.g. X), is a function from each element in $\mathcal{G}$—from each outcome—to a value on the real number line. The real number line contains all numbers: integer and decimal, from negative to positive infinity.

Example: Suppose our sample space contains two elements $\mathcal{G} = \{a, b\}$. We may decide to define a random variable X that maps the outcome a to the value -1 and the outcome b to the value 1. In otherwords, $X(a) = -1$ and $X(b) = 1$. We could as well define a random variable Y on the same sample space such that $Y(a) = 0$ and $Y(b) = 1$.

Example: Suppose our sample space contains all integers from 0 to 1000 $\mathcal{G} = \{0, 1, 2, 3 \cdots, 1000\}$. We may be most interested in when an integer is even or odd, and so we can define a random variable $Y(y) = 0$ when y, our outcome, is an odd integer and $Y(y) = 1$ when y is even. This is an example of how a random variable can distill down a sample space with many outcomes into a random variable with two.

Example: Suppose we decide to study the relationship between the cumulative total number of cigarettes smoked by a person form the date that they started smoking and the presence of lung cancer. We define our sample space to be $\mathcal{G} = \{(x,y)|x \in \mathbb{Z}, y \in \{0,1\}\}$. We define two random variables, a random variable X that maps the outcome (x,y) to the value in the first position x, and a random variable Y that maps the $outcome(x,y)$ to the value in the second position y. Though our outcomes are linked, we can use random variables to think about two separate outcomes—cigarettes smoked and lung cancer—and how they interact.

2.3 A new sample space

When we build a random variable (X) that maps outcomes to values on the number line we create a new sample space which we will call the support of X or $supp(X)$. Define a sample space $\mathcal{G}$ without outcomes o_i. Then the **support of X** is

$$supp(X) = \{x|X(o) = x \text{ for some outcome } o \text{ in } \mathcal{G}\} \tag{2.1}$$

Our new sample space is the set of all the potential values that our random variable X can produce. This is a sample space linked to $\mathcal{G}$, but in practice after we develop a random variable we often no longer reference $\mathcal{G}$.

Example: In our above example where $\mathcal{G} = \{a,b\}$, the random variable X has support $supp(X) = \{-1,1\}$ and $supp(Y) = \{0,1\}$. Lets look at another example, when above $\mathcal{G}$ is the set of all integers from 0 to 1000. Even though the sample space is quite large, the random variable that maps the integers to 0 when they are odd and 1 when even has a small support $(supp(Y) = \{0,1\})$.

2.4 How to assign probabilities to a random variable

Random variable themselves do not require that we include the probability of each of their values. Random variables are a function from outcomes to the real numbers—nothing more. That said, in practice we build random variables expecting that the probabilities we assign to outcomes in our sample space will correspond to probabilities assigned to values of our random variable.

We assign a probability to the value x, which belongs in the support of random variable X, the sum of the probabilities of all the outcomes that X maps to x.

$$P(X = x) = P(o_1) + P(o_2) + \cdots + P(o_n) \tag{2.2}$$

where each outcome $o_1, o_2, \cdots, o_n$ is mapped by X to the value x. In other words, $X(o) = x$ for each of $o_1, o_2, \cdots, o_n$.

Example: Define a $\mathcal{G} = \{a, b, c, d, e\}$ and a random variable X that maps the outcomes to the following values

Outcome	P(outcome)	X(outcome)
a	0.1	0
b	0.25	1
c	0.15	1
d	0.3	2
e	0.2	0

We assign the probability that $X = 0$ as the sum of the probabilities assigned to outcome a and outcome e, or

$$P(X = 0) = P(\{a\}) + P(\{e\}) \tag{2.3}$$
$$= 0.1 + 0.2 = 0.3 \tag{2.4}$$

We can run the same procedure for all the elements in the support of X,

$$P(X = 1) = P(\{b\}) + P(\{c\}) \tag{2.5}$$
$$= 0.25 + 0.15 = 0.40 \tag{2.6}$$
$$P(X = 2) = P(\{d\}) = 0.3 = 0.30, \tag{2.7}$$

and organize our work in a table

X	P(X=x)
0	0.30
1	0.40
2	0.30

A **probability distribution** for a random variable X is a set of tuples where the first position in each tuple is a value in the support of X and the second position in the tuple is the corresponding probability assigned to that value.

Example: A probability distribution for the random variable X above is $\{(0, 0.30), (1, 0.40), (2, 0.30)\}$.

Example: Imagine we run an experiment that collects data on marathon runners. We decide to collect the number of elapsed minutes until they finish the race. Our sample space is defined as all positive integers $\mathcal{G} = \{1, 2, 3, \cdots, \}$. We may decide to build a random variable X that maps outcomes less than 60 to the value 1, outcomes from 61 to 120 to the value 2, and outcomes greater than 120 to the value 3. One potential probability distribution for X is $\{(1, 0.10), (2, 0.50), (3, 0.40)\}$. For this probability distribution, $P(X = 1) = 0.10$, $P(X = 2) = 0.50$, and $P(X = 3) = 0.40$.

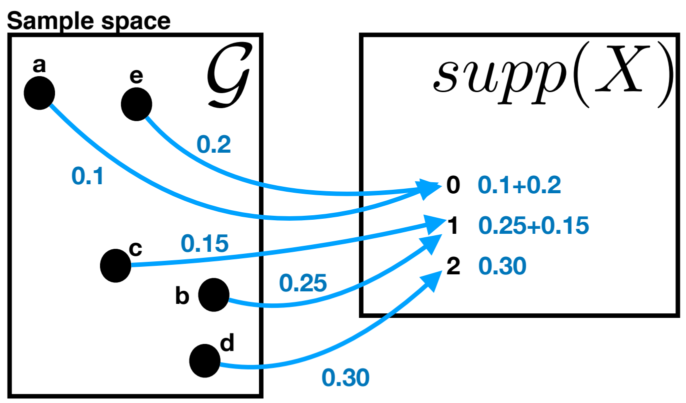


FIGURE 2.1
A sample space $\mathcal{G}$ with elements $\{a, b, c, d, e\}$ and a random variable X that maps each element in $\mathcal{G}$ to one the values: 0, 1, or 2. Probabilities corresponding to outcomes in the sample space and how they map to probabilities for each value of X are shown in blue.

2.5 Probability mass function

There are several supportive tools that we can use to help us better understand random variable we create. The first is the probability mass function, or p.m.f. The **probability mass function** is a function that maps values in the support of a random variable X to their corresponding probabilities. Inputs are values of X, outputs are probabilities.

The probability mass function is a convenient way to organize a probability distribution and it allows us to transfer all the information we know about functions to random variables.

Example: Define a random variable Y with support $\{-1, 0, 1\}$, probability distribution {(-1,0.2),(0,0.5),(1,0.3)}, and probability mass function

$$f(y) = \begin{cases} 0.2 & \text{when } y = -1 \\ 0.5 & \text{when } y = 0 \\ 0.3 & \text{when } y = 1 \end{cases} \tag{2.8}$$

The function—our probability mass function—is a type of function called a **piecewise** function.

We can ask our pm.f. to return the probability for a given value

$$f(1) = 0.3 \tag{2.9}$$

and we can visualize our probability mass function using, for example, a barplot.

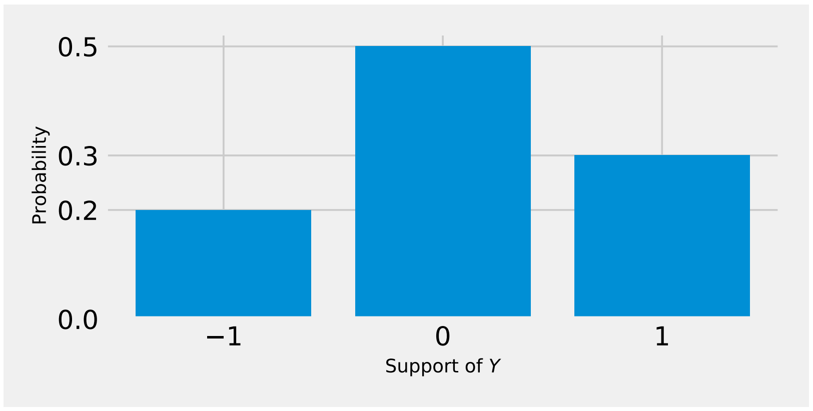


FIGURE 2.2
A barplot for visualizing the probability mass function of our random variable Y. The support of Y is plotted on the horizontal axis and height of each bar corresponds to the probability assigned to that value in the support.

Distributed as f: Because we can use the probability mass function to describe the probability distribution of a random variable, we will often write

$$Y \sim f \tag{2.10}$$

The above formula is read "the random variable Y is distributed as f", and what we mean when a random variable is distributed as f is that the support of Y is the same as the domain of the function f and that the probability of a value y is equal to $f(y)$, or

$$supp(Y) = dom(f) \tag{2.11}$$
$$P(Y = y) = f(y) \tag{2.12}$$

The probability mass function is a convenient method for assigning probabilities to random variables and visualizing the distribution of a random variable.

2.6 Cumulative mass function

The **cumulative mass function** is a function that maps values in the support of a random variable X to the probability that the random variable is less than or equal to this value, or $P(X \leq x)$.

We use a capital F to denote a cumulative mass function (c.m.f). The c.m.f. corresponding to random variable X has a domain equal to the support of X and produces values between 0 and 1 (the values a function can produce is called the function's **image**).

$$supp(X) = dom(F) \tag{2.13}$$
$$image(F) = [0, 1] \tag{2.14}$$
$$P(X \leq x) = F(x) \tag{2.15}$$

The c.m.f. too can be visualized and we could also use the c.m.f. to describe the probability distribution of a random variable. This is because we can use the c.m.f. to derive the p.m.f.

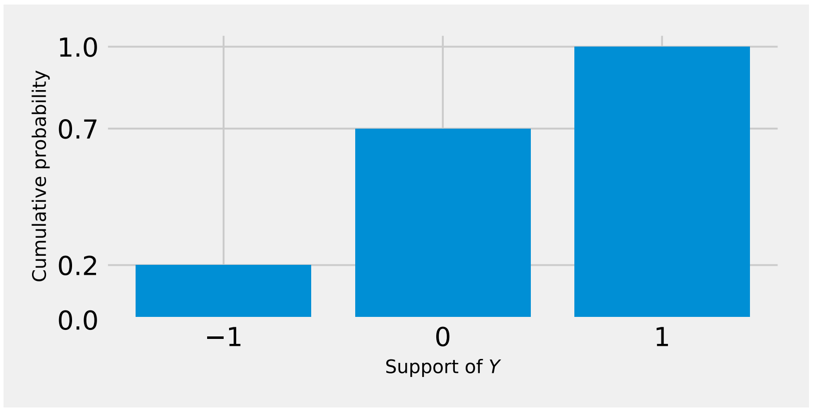


FIGURE 2.3
A barplot for visualizing the cumulative mass function of our random variable Y. The support of Y is plotted on the horizontal axis and height of each bar corresponds to the probability assigned to values (v) less than or equal to v in the support.

Example: For a random variable

$$X \sim f \tag{2.16}$$
$$supp(X) = \{0, 1, 2, 3\} \tag{2.17}$$
$$f(x) = \begin{cases} 0 & 0.1 \\ 1 & 0.3 \\ 2 & 0.2 \\ 3 & 0.4 \end{cases} \tag{2.18}$$

The c.m.f is then

$$F(x) = \begin{cases} 0 & 0.1 \\ 1 & 0.3 + 0.1 = 0.4 \\ 2 & 0.2 + 0.3 + 0.1 = 0.6 \\ 3 & 0.4 + 0.2 + 0.3 + 0.1 = 1 \end{cases} \tag{2.19}$$

We can use the c.m.f $(F(x))$ to compute any p.m.f $(f(x))$ too by noticing that for a support $\{x_0, x_1, x_2, x_3, \cdots, x_{n-1} \cdots, x_n\}$ where the values in this set are ordered form smallest to largest

$$f(x_i) = [f(x_i) + f(x_{i-1}) + \cdots f(x_0)] - [f(x_{i-1}) + \cdots f(x_0)] \tag{2.20}$$
$$= F(x_i) - F(x_{i-1}). \tag{2.21}$$

Because the p.d.f. and c.m.f equivalently describe the probability distribution of a random variable we can write $X \sim F$ or $X \sim f$.

2.7 Two or more random variables

A single random variable is a useful way to describe a sample space and the probabilities assigned to values on the number line corresponding to outcomes.

But more complicated scientific questions often require several random variables and rules for how they interact. We will explore how to generate multiple random variables on the same sample space. Then we will develop an approach to define probabilities for combinations of random variables, a joint probability mass function and joint cumulative mass function.

Suppose we design a sample space $\mathcal{G}$ and build two random variables on this space X and Y. Because X and Y are random variables we can think of the support of X and support of Y as two new sample spaces. Lets assume that the outcomes in $supp(X) = \{x_1, x_2, x_3, \cdots, x_N\}$ and that the outcomes in $supp(Y) = \{y_1, y_2, \cdots, y_M\}$. Then we can define a new set of outcomes in the space $supp(X) \times supp(Y) = \{x_i, y_j\}$ for all combinations of i from 1 to N and j from 1 to M.

This new space above maps outcomes in $\mathcal{G}$ to a tuple (x_i, y_j) where the outcomes were mapped by X to the value x_i and by Y to the value y_j. If we assign the probability of (x_i, y_j) to be the the sum of the probabilities of all outcomes in $\mathcal{G}$ that X maps to x_i and Y maps to y_i then we call this a **joint probability distribution**.

Example: A pharmaceutical company launches a clinical trial to study adverse events from a novel medicine. The trial enrolls patients and randomizes them to receive the novel drug or optimal medical therapy. The trial expects three potential adverse events which we call ae_1, ae_2, and ae_3. We may choose to define a sample space $\mathcal{G} = \{(\text{nov}, \text{ae}_1), (\text{nov}, \text{ae}_2), (\text{nov}, \text{ae}_3), (\text{omt}, \text{ae}_1), (\text{omt}, \text{ae}_2), (\text{omt}, \text{ae}_3)\}$ and further build a random variable X that maps outcomes to values 0 when an adverse event was experienced by a control patient and 1 when an adverse event was experienced by a novel drug patient, and a random variable Y that maps adverse event 1 to the value 1, adverse event 2 to the value 2, and adverse event 3 to the value 3.

$$\mathcal{G} = \{(\text{nov}, \text{ae}_1), (\text{nov}, \text{ae}_2), (\text{nov}, \text{ae}_3), (\text{omt}, \text{ae}_1), (\text{omt}, \text{ae}_2), (\text{omt}, \text{ae}_3)\}$$

Event	X
$(\text{nov}, \text{ae}_1)$	1
$(\text{nov}, \text{ae}_2)$	1
$(\text{nov}, \text{ae}_3)$	1
$(\text{omt}, \text{ae}_1)$	0
$(\text{omt}, \text{ae}_2)$	0
$(\text{omt}, \text{ae}_3)$	0

Event	Y
$(\text{nov}, \text{ae}_1)$	1
$(\text{nov}, \text{ae}_2)$	2
$(\text{nov}, \text{ae}_3)$	3
$(\text{omt}, \text{ae}_1)$	1
$(\text{omt}, \text{ae}_2)$	2
$(\text{omt}, \text{ae}_3)$	3

A joint probability distribution would assign probabilities to all possible pairs of values for X and Y.

We write $P(X = x, Y = y)$ to denote the probability assigned to the joint probability that the random variable X equals the value x and the random variable Y equals the value y. With the above example in mind, $P(X = 0, Y =$

X	Y	prob
1	1	0.1
1	2	0.05
1	3	0.23
0	1	0.05
0	2	0.30
0	3	0.27

$2) = 0.30$ and this probability could be described as the probability an OMT patient experiences adverse event 2.

Joint probability distributions are often visualized as a table with one random variable's values representing the rows and the second random variable's values representing the columns

	Y = 1	Y = 2	Y = 3
X=0	0.05	0.30	0.27
X=1	0.10	0.05	0.23

A joint distribution can be used to compute probabilities that the random variable X equals the value x for any value of Y—$P(X = x)$ and also the probability that the random variable Y equals the value y for any value of x—$P(Y = y)$. These probabilities are called **marginal probabilities**.

The marginal probabilities that X equals 0, or $P(X = 0)$, is computed by summing the joint probabilities where $X = 0$. We can use the same procedure to compute the marignal probability that $X = 1$.

$$\begin{aligned}
P(X = 0) &= P(X = 0, Y = 1) + P(X = 0, Y = 2) + P(X = 0, Y = 3) \\
&= 0.05 + 0.30 + 0.27 = 0.62 \\
P(X = 1) &= P(X = 1, Y = 1) + P(X = 1, Y = 2) + P(X = 1, Y = 3) \\
&= 0.10 + 0.05 + 0.23 = 0.38
\end{aligned}$$

The same procedure can be done for the random variable Y to find marginal probabilities:

$$\begin{aligned}
P(Y=1) &= P(X=0,Y=1)+P(X=1,Y=1)\\
&= 0.1+0.05=0.15\\
P(Y=2) &= P(X=0,Y=2)+P(X=1,Y=2)\\
&= 0.05+0.30=0.35\\
P(Y=3) &= P(X=0,Y=3)+P(X=1,Y=3)\\
&= 0.23+0.27=0.50
\end{aligned}$$

Joint distributions need not be restricted to only two random variables. We can build joint distributions of any number of random variables and can compute marginal probabilities in the same way that we do for a joint distribution of two random variables.

The rules, laws, and theorems of probability that we learned in chapter one carry over to random variables. This is because we can consider a new sample space of values that our random variable can take. Suppose we build two random variables X that maps outcomes to the the values -1,0,1 and Y that maps outcomes to the values 1,2,3. We can define a new sample space $\mathcal{G} = \{(x,y) | x \in supp(X) \text{ and } y \in supp(Y)\}$ With our new sample space, we can now discuss statements like $P(X = x|Y = y)$.

Example: If we continue with our pharmaceutical example, we can build a new sample space $\mathcal{G} = \{(0,1),(0,2),(0,3),(1,1),(1,2),(1,3)\}$ and compute, for example, $P(X = 1|Y = 2) = \frac{P(X=1,Y=2)}{P(Y=2)} = \frac{0.05}{0.35} = 0.14$.

We can define the conditional probability of the value of one random variable given another as

$$P(A=a|B=b) = \frac{P(A=a,B=b)}{P(B=b)} \tag{2.22}$$

and so define statistical independence between two random variables, A and B, as

$$P(A=a|B=b) = P(A=a) \text{ for all } a \in supp(A), b \in supp(B) \tag{2.23}$$

We can also translate the law of total probability from events to random variables. For a partition $[Y = y_1] \cup [Y = y_2] \cup [Y = y_3] \cup \cdots \cup [Y = y_n]$ of the event that the random variable $X = x$,

$$\begin{aligned}
P(X=x) &= P(X=x|Y=y_1)p(y_1) + P(X=x|Y=y_2)p(y_2)\\
&\quad + P(X=x|Y=y_3)p(y_3) \cdots + P(X=x|Y=y_n)p(y_n)
\end{aligned} \tag{2.24}$$

By thinking of the values of a random variable as outcomes in a new sample space, we can apply our past intuition and the past mechanics of events, sets to a collection of random variables.

2.8 Functions of a random variable

There are times that we may wish to summarize the behavior of a random variable. One common way to describe how probability is distributed among values in the support of a random variable is by computing some function of that random variable.

2.8.1 Multiplying and adding a constant to a random variable

Suppose we have defined a random variable X and wish to study a new random variable $Y = g(X)$ that is a function of X. The values and the probabilities assigned to X, along with the function g will determine the values in the support of Y and assigned probabilities to these values.

2.8.1.1 Translation

Define a random variable X with p.m.f. f_X and a new random variable $Y = X + c$ where c is a constant. Then the support for Y, $supp(Y) = \{x + c | x \in supp(X)\}$ and the p.m.f. of Y, $f_Y(y)$ is

$$\begin{aligned} f_Y(y) &= P(Y = y) && (2.25)\\ &= P(X + c = y) && (2.26)\\ &= P(X = y - c) && (2.27)\\ &= f_X(y - c) && (2.28) \end{aligned}$$

In otherwords, the probability distribution on Y is $\mathbb{P} = \{(y, p) \mid y = x + c \text{ and } P(X = x) = p\}$. This type of transformation of the random variable X to Y is called a **translation**.

Example: Suppose we define a random variable Z such that $supp(Z) = \{0, 1, 2, 3, 4, 5\}$ and

$$f_Z(z) = \begin{cases} 0.10 & \text{if } z = 0\\ 0.05 & \text{if } z = 1\\ 0.20 & \text{if } z = 2\\ 0.12 & \text{if } z = 3\\ 0.07 & \text{if } z = 4\\ 0.58 & \text{if } z = 5 \end{cases} \tag{2.29}$$

We can define a new random variable $Y = Z - 4$. This random variable will have $supp(Y) = \{-4, -3, -2, -1, 0, 1\}$ and

$$f_Y(y) = \begin{cases} 0.10 & \text{if } y = -4 \\ 0.05 & \text{if } y = -3 \\ 0.20 & \text{if } y = -2 \\ 0.12 & \text{if } y = -1 \\ 0.07 & \text{if } y = 0 \\ 0.58 & \text{if } y = 1 \end{cases} \tag{2.30}$$

2.8.1.2 Scaling

Define a random variable X with p.m.f. f_X and a new random variable $Y = c \cdot X$ where c is a constant. Then the support for Y, $supp(Y) = \{c \cdot x | x \in supp(X)\}$ and the p.m.f. of Y, $f_Y(y)$ is

$$\begin{aligned} f_Y(y) &= P(Y = y) && (2.31) \\ &= P(cX = y) && (2.32) \\ &= P(X = y/c) && (2.33) \\ &= f_X(y/c) && (2.34) \end{aligned}$$

When we multiply or divide a random variable by a constant to produce a new random variable this is called **scaling**.

Example: Suppose we define a random variable Z such that $supp(Z) = \{0, 1, 2, 3, 4, 5\}$ and

$$f_Z(z) = \begin{cases} 0.10 & \text{if } z = 0 \\ 0.05 & \text{if } z = 1 \\ 0.20 & \text{if } z = 2 \\ 0.12 & \text{if } z = 3 \\ 0.07 & \text{if } z = 4 \\ 0.58 & \text{if } z = 5 \end{cases} \tag{2.35}$$

We can define a new random variable $Y = 4 \cdot Z$. This random variable will have $supp(Y) = \{0, 4, 8, 12, 16, 20\}$ and

$$f_Y(y) = \begin{cases} 0.10 & \text{if } y = 0 \\ 0.05 & \text{if } y = 4 \\ 0.20 & \text{if } y = 8 \\ 0.12 & \text{if } y = 12 \\ 0.07 & \text{if } y = 16 \\ 0.58 & \text{if } y = 20 \end{cases} \tag{2.36}$$

2.8.2 General transformation of a discrete random variable

We explored translation and scaling as specific ways to use one random variable to generate another. However, there are many functions we can apply to a random variable X to create a new random variable Y, and we need a more general method to compute the probability mass function for Y.

Build a random variable X and define Y to be a new random variable such that $Y = g(X)$. Then the support for Y is $supp(Y) = \{y|y = g(x) \text{ and } x \in supp(X)\}$, and

$$f_Y(y) = f_X(x_1) + f_X(x_2) + \cdots f_X(x_n) \tag{2.37}$$

where $x_1, x_2, \cdots x_n \in g^{-1}(y)$. If Y is a function of X then the probability that the random variable Y will assign to the value y is the sum of the probabilities the random variable X assigns to the set of values $x_1, x_2, \cdots, x_n$ that are mapped to y by the function g.

Example: Let X be the random variable with p.m.f

$$f_X(x) = \begin{cases} 0.01 & \text{if } x = -2 \\ 0.05 & \text{if } x = -1 \\ 0.50 & \text{if } x = 0 \\ 0.34 & \text{if } x = 1 \\ 0.10 & \text{if } x = 2 \end{cases} \tag{2.38}$$

and also build a new random variable $Y = X^2$. Here the function g is $g(x) = x^2$. The support of Y is

$$\begin{aligned} supp(Y) &= \{-2^2, -1^2, 0^2, 1^2, 2^2\} && (2.39) \\ &= \{4, 1, 0, 1, 4\} && (2.40) \\ &= \{0, 1, 4\} && (2.41) \end{aligned}$$

To compute $P(Y = 0)$ we need to sum up the probabilities of all values in $supp(X)$ that g maps to 0. In our case the only value mapped to 0 is the value 0, and so $P(Y=0)=P(X=0)=0.50$. To compute $P(Y=1)$ we need to sum the probabilities of all values that g maps to the value 1. In this case two values in X map to 1, the values -1 and 1, and so $P(Y=1)=P(X=-1)+P(X=1)=0.05+0.34=0.39$. Finally, $P(Y=4)=P(X=-2)+P(X=2)=0.11$ (why?). The pm.f for Y is

$$f_Y(y)=\begin{cases}0.50 & \text{if } y=0\\ 0.39 & \text{if } y=1\\ 0.11 & \text{if } y=4\end{cases} \tag{2.42}$$

2.8.3 Expectation

Suppose we build a random variable X with a corresponding probability mass function f_X.

The **expected value** of a random variable X is computed as

$$\begin{aligned}\mathbb{E}\,(X) &= P(X=x_1)x_1+P(X=x_2)x_2+\cdots+P(X=x_n)x_n && (2.43)\\ &= f(x_1)x_1+f(x_2)x_2+\cdots+f(x_n)x_n && (2.44)\end{aligned}$$

where $x_1, x_2, \cdots, x_n$ are all values in the $supp(X)$.

An intuitive definition of the expected value is that $\mathbb{E}(X)$ is a weighted average of all values in the support of X where the weight for x_i is the probability of x_i. The expected value of X will be close to values in $supp(X)$ with high probability.

Example: Build a random variable Y with support $supp(Y)=\{-1,0,1\}$ and $f_Y=\{(-1,0.2),(0,0.5),(1,0.3)\}$. The expected value of Y is $\mathbb{E}(Y)=0.2(-1)+0.5(0)+0.3(1)=0.1$.

2.8.3.1 Properties of the expectation

The expectation is a linear function, that is $\mathbb{E}(aY+b)=a\mathbb{E}(Y)+b$. We can show this by defining a random variable $Z=aY+b$ and asking

$$\begin{aligned}\mathbb{E}(Z) &= z_1f_Z(z_1)+z_2f_Z(z_2)+\cdots z_nf_Z(z_n) && (2.45)\\ &= (ay_1+b)f_Z(z_1)+(ay_2+b)f_Z(z_2)+\cdots(ay_n+b)f_Z(z_n) && (2.46)\\ &= a\left[y_1f_Z(z_1)+y_2f_Z(z_2)+\cdots y_nf_Z(z_n)\right] \\ &\quad + b\left[f_Z(z_1)+f_Z(z_2)+\cdots+f_Z(z_n)\right] && (2.47)\\ &= a\left[y_1f_Z(z_1)+y_2f_Z(z_2)+\cdots y_nf_Z(z_n)\right]+b\ \text{ (why?)} && (2.48)\\ &= a\left(y_1f_Y(y_1)+y_2f_Y(y_2)+\cdots y_nf_Y(y_n)\right)+b && (2.49)\\ &= a\mathbb{E}(Y)+b && (2.50)\end{aligned}$$

The step (2.49) deserves some attention. Values z_i are equal to ay_i+b, they are mapped from the values y_i and so the probability that Z equals z_i is equivalent to the probability that Y equals y_i.

2.8.4 Second moment and variance

Define a random variable Y with $supp(Y) - \{y_1, y_2, \cdots, y_n\}$, then **variance** is the following function of Y

$$V(Y) = [y_1 - \mathbb{E}(Y)]^2 P(Y = y_1) + [y_2 - \mathbb{E}(Y)]^2 P(Y = y_2) + \cdots + [y_n - \mathbb{E}(Y)]^2 P(Y = y_n) \tag{2.51}$$

The variance can be thought of as the squared distance of each value in the support of the random variable Y from the expected value weighted by the probability of each value. In some sense, the variance attempts to measure the squared distance from the expected value.

Example: Define a random variable Z with probability mass function

$$f_Z(z) = \begin{cases} 0.14 & \text{if } z = 0 \\ 0.39 & \text{if } z = 1 \\ 0.21 & \text{if } z = 2 \\ 0.26 & \text{if } z = 4 \end{cases} \tag{2.52}$$

To compute $V(Z)$ we need to first compute the expected value of Z or $\mathbb{E}(Z)$:

$$\begin{aligned} \mathbb{E}(Z) &= f_Z(0) \cdot 0 + f_Z(1) \cdot 1 + f_Z(2) \cdot 2 + f_Z(4) \cdot 4 && (2.53) \\ &= 0.14 \cdot 0 + 0.39 \cdot 1 + 0.21 \cdot 2 + 0.26 \cdot 4 && (2.54) \\ &= 1.85 && (2.55) \end{aligned}$$

Now we can compute the variance

$$\begin{aligned} V(Z) = (0 - 1.85)^2 \cdot f_Z(0) + (1 - 1.85)^2 \cdot f_Z(1) + \\ (2 - 1.85)^2 \cdot f_Z(2) + (4 - 1.85)^2 \cdot f_Z(4) && (2.56) \\ = (0 - 1.85)^2 \cdot 0.14 + (1 - 1.85)^2 \cdot 0.39 + \\ (2 - 1.85)^2 \cdot 0.21 + (4 - 1.85)^2 \cdot 0.26 && (2.57) \\ = 1.97 && (2.58) \end{aligned}$$

2.8.5 How the expected value earned its name

\nobreak

2.8.5.1 Markov inequality

$$P(X > a) < \frac{\mathbb{E}(X)}{a} \tag{2.59}$$

Suppose we define a random variable X with a support that takes only non-negative numbers.

$$\mathbb{E}(X) = \sum_{x_i \in supp(X)} x_i f_X(x_i) \tag{2.60}$$

$$\mathbb{E}(X) = \sum_{x_i \leq a} x_i f_X(x_i) + \sum_{x_i > a} x_i f_X(x_i) \tag{2.61}$$

$$\mathbb{E}(X) > \sum_{x_i > a} x_i f_X(x_i) \tag{2.62}$$

$$\mathbb{E}(X) > \sum_{x_i > a} a f_X(x_i) \tag{2.63}$$

$$\mathbb{E}(X) > a \sum_{x_i > a} f_X(x_i) \tag{2.64}$$

$$\mathbb{E}(X) > aP(X > a) \tag{2.65}$$

$$\frac{\mathbb{E}(X)}{a} > P(X > a) \tag{2.66}$$

$$\tag{2.67}$$

2.8.5.2 Chebychev's inequality

\nobreak

2.8.6 Covariance

\nobreak

2.8.7 Correlation

\nobreak

2.9 A Model

\nobreak

2.10 Exercises

1. Suppose $\mathcal{G} = \{a, b, c\}$ and $P(\{a\}) = 0.2$, $P(\{b\}) = 0.3$, $P(\{c\}) = 0.5$. Define a random variable X such that $X(a) = 1$, $X(b) = 1$, and $X(c) = 0$. Define a second random variable Y such that $Y(a) = 0$, $Y(b) = 1$, $Y(c) = 2$.
   (a) Compute $P(X = 1)$
   (b) Compute $P(X = 0)$
   (c) What is $supp(X)$?
   (d) What new sample space does X generate?
   (e) Compute $P(Y = 1)$
   (f) Compute $P(Y = 0)$
   (g) What is $supp(Y)$?
   (h) What new sample space does Y generate?

2. Let $\mathcal{G} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$, the set of all positive integers. Further define a random variable K with the following probability mass function

$$f(k) = \left(\frac{1}{2}\right)^k \text{ when } k \leq 10$$
$$f(11) = 0.009$$

   (a) Is the pmf f a valid probability distribution? Why or why not?
   (b) What value of K is assigned the highest probability?
   (c) Please define the cumulative mass function for the random variable K

3. Define a random variable with $supp(Y) = \{-3, -2, -1, 0, 1, 2, 3\}$ and cumulative mass function

$$F(y) = \begin{cases} 0.10 & \text{when } y = -3 \\ 0.24 & \text{when } y = -2 \\ 0.36 & \text{when } y = -1 \\ 0.50 & \text{when } y = 0 \\ 0.67 & \text{when } y = 1 \\ 0.78 & \text{when } y = 2 \\ 1.00 & \text{when } y = 3 \end{cases}$$

(a) What is $P(Y \leq -1)$

(b) What is $P(Y = -1)$

(c) What is $P(Y = 1)$

(d) Please define the p.m.f for the random variable Y.

(e) Graph the c.m.f

(f) Graph the p.m.f

4. Define the following joint distribution of random variable Q and R that are mapped from the sample space $\mathcal{G}$

Q	R	prob
2	1	0.01
2	2	0.075
2	3	0.13
1	1	0.1
1	2	0.05
1	3	0.17
0	1	0.05
0	2	0.30
0	3	0.115

(a) What is the implied support of Q?

(b) What is the implied support of R?

(c) Compute P(Q=1,R=2)

(d) Compute the marginal probabilities for Q

(e) Compute the marginal probabilities for R

(f) Compute $P(R = 1|Q = 0)$

(g) The random variable Q is called **statistically independent** from R if for every value $q \in supp(Q)$ and for every value $r \in R$ the following is true $P(Q = q|R = r) = P(Q)$. Is Q statistically independent from R? Why or why not?

5. For two events A and B that are statistically independent, we found that $P(A \cap B) = P(A)P(B)$. Please derive an equivalent expression for two random variables by applying the definition of conditional probability and statistical independence to random variables.

6. Define the following joint distribution of random variable Q and R that are mapped from the sample space $\mathcal{G}$

(a) Please compute $\mathbb{E}(Q)$

Q	R	prob
2	1	0.01
2	2	0.075
2	3	0.13
1	1	0.1
1	2	0.05
1	3	0.17
0	1	0.05
0	2	0.30
0	3	0.115

(b) Please compute $\mathbb{E}(R)$

(c) Please compute $V(Q)$

(d) Please compute $V(R)$

7. Suppose X is a random variable with support $supp(X) = \{-2,-1,0,1,2,3\}$ and $Y = |X|$. Further assume the c.m.f of X is

$$F_X(x) \begin{cases} 0.05 & \text{if } x = -2 \\ 0.15 & \text{if } x = -1 \\ 0.35 & \text{if } x = 0 \\ 0.65 & \text{if } x = 1 \\ 0.95 & \text{if } x = 2 \\ 1.00 & \text{if } x = 3 \end{cases} \tag{2.68}$$

(a) Define the support of Y

(b) Build the p.m.f of Y

(c) Build the c.m.f of Y

8. Compute $\sum_{i=-5}^{i=5} i^2/2$

9. Simplify the $V(X)$ into the equation $E(X^2) - [E(X)]^2$. Hint: Write down the definition of variance using the expectation, then expand the squared terms and simplify.

10. Define the random variable A with pmf

$$f_B(b) = \begin{cases} 0.52 & \text{if } b = 0 \\ 0.12 & \text{if } b = 1 \\ 0.34 & \text{if } b = 2 \end{cases} \tag{2.69}$$

(a) Compute $\mathbb{E}(B)$

(b) Compute $V(B)$

(c) Use Chebychev's inequality to make a statement about $P(B \geq b)$

3

Probability distributions: templates

thomas mcandrew

Lehigh University

CONTENTS

3.1 Introduction

There exist several probability distributions that have been studied in depth. Templates like this allow us to model data generated from an experiment quickly. Random variable templates are so useful that several computer languages have optimized how to compute probabilities, expected values, and higher moments.

We will study parametric distributions for random variables. A set of **parameters**—constants— determine how our random variable assigns probabilities to outcomes.

3.2 The Bernoulli distribution

The Bernoulli distribution assigns probabilities to a random variable who support contains the values 0 and 1. The Bernoulli distribution has a single parameter, often denoted θ, that controls how probabilities are assigned to these two values 0 and 1.

To communicate that a random variable Z has a Bernoulli distribution with parameter θ, we write

$$Z \sim \text{Bernoulli}(\theta) \tag{3.1}$$

The parameter θ can take any value between 0 and 1 inclusive, or $\theta \in [0,1]$. The allowable values a set of parameters can take is called the **parameter space**.

The support of Z is $supp(Z) = \{0,1\}$, and the probability mass function for the random variable Z is

$$f_Z(z) = \begin{cases} \theta & \text{if } z = 1 \\ 1-\theta & \text{if } z = 0 \end{cases} \tag{3.2}$$

We can use the probability mass function to compute the expectation

$$\begin{aligned} \mathbb{E}(Z) &= f_Z(1)\cdot 1 + f_Z(0)\cdot 0 && (3.3)\\ &= f_Z(1)\cdot 1 = f_Z(1) && (3.4)\\ &= \theta, && (3.5) \end{aligned}$$

and we can use the probability mass function to compute the variance

$$\begin{aligned} V(Z) &= (1-\theta)^2 f_Z(1) + (0-\theta)^2 f_Z(0) && (3.6)\\ &= (1-\theta)^2\theta + \theta^2(1-\theta) && (3.7)\\ &= \theta(1-\theta)\left[(1-\theta)+\theta\right] && (3.8)\\ &= \theta(1-\theta) && (3.9) \end{aligned}$$

Example: Define $Z \sim$ Bernoulli(0.45). Then $supp(Z) = \{0,1\}$, $P(Z=0) =$ 0.55, the $P(Z=1) = 0.45$, $\mathbb{E}(Z) = 0.45$ and $V(Z) = 0.45(0.55) = 0.25$.

Example: A clinical trial enrolls patients and follows them for one year. The clinical team wants to understand the proportion of patients that experience or do not experience an adverse event. We could model whether each patient either experiences or does not experience an adverse event using a Bernoulli distribution. Define Z_i as a Bernoulli distributed random variable for the i^{th} patient in the study. When Z_i = 1 the i^{th} patient experienced an adverse event and 0 otherwise.

3.3 The Geometric distribution

If a random variable X has a **Geometric** distribution then the support is the values $supp(X) = \{1,2,3,4,5,\cdots\}$ and the probability mass function is

$$f_X(x) = p(1-p)^{x-1} \tag{3.10}$$

A random variable that has a geometric distribution has a single parameter p that can take any value between 0 and 1 inclusive, or $p \in [0,1]$.

The expected value and variance are

$$\mathbb{E}(X) = \frac{1}{p} \tag{3.11}$$

$$V(X) = \frac{1}{p}\left(\frac{1-p}{p}\right) \tag{3.12}$$

A random variable that follows the Geometric distribution often corresponds to an experiment where there is a p probability that an event occurs (often called a success), a $(1-p)$ probability an event does not occur (often called a failure). We assume that each experiment is independent from the previous experiments. The Geometric distribution assigns a probability to the the number of times the experiment is repeated until the event occurs (i.e. the number of repeated experiments until success).

Example:

3.4 Exercises

1. Let $X \sim$ Bernoulli(0.2)
   (a) P(X=0) = ?
   (b) P(X=1) = ?
   (c) Please compute $\mathbb{E}(X)$
   (d) Please compute $V(X)$
   (e) Define the $supp(X)$
2. Let $Y \sim$ Bernoulli(θ), and show that $P(Y=1) = \mathbb{E}(Y)$
3. Let $Y \sim$ Bernoulli(θ), and show that $V(Y) \leq \mathbb{E}(Y)$
4. Let $Y \sim$ Bernoulli(θ), and let Z be the following function of Y:

$$Z(y) = \begin{cases} 1 & \text{if } y = 0 \\ 0 & \text{if } y = 1 \end{cases} \tag{3.13}$$

   What probability distribution does Z follow and why?
5. Design an experiment (short description) and define a random variable

Y that may follow a geometric distribution. In the context of your experiment, how would you communicate $\mathbb{E}(Y)$ to another without statistical expertise?

Part II

Statistics

Part III

Algorithms lab

4

Laboratory 01

thomas mcandrew, david braun

Lehigh University

CONTENTS

4.1 Jupyter Notebooks

All of our lab work will take place in Jupyter Notebooks. Jupyter Notebooks are a tool for organizing textual descriptions of work and computer programs. The goal is to produce one document to communicate a set of scientific ideas and allow another to understand exactly how you arrived at yoru conclusions.

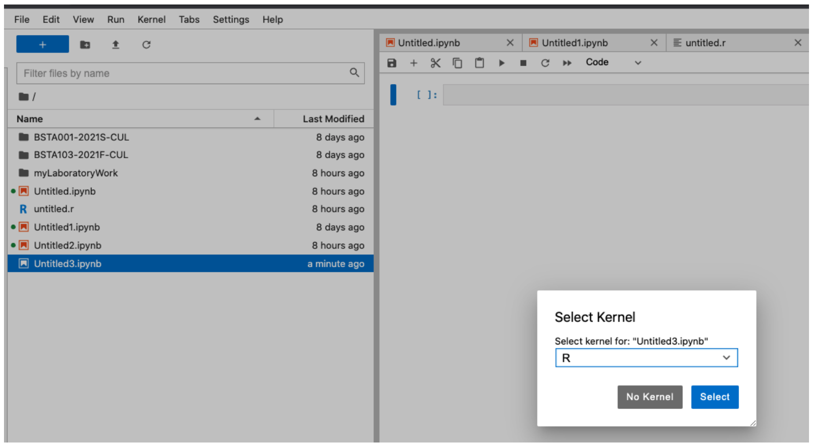
Jupyter has some important buttons.

4.1.1 File

\nobreak

4.1.1.1 A new notebook

Under file->New->Notebook you can create a new notebook. When asked to "Select Kernel" click on the drop down menu and select "R"



Name	Last Modified
BSTA001-2021S-CUL	8 days ago
BSTA103-2021F-CUL	8 days ago
myLaboratoryWork	8 hours ago
Untitled.ipynb	8 hours ago
untitled.r	8 hours ago
Untitled1.ipynb	8 days ago
Untitled2.ipynb	8 hours ago
Untitled3.ipynb	a minute ago

Select Kernel

Select kernel for: "Untitled3.ipynb"

R

FIGURE 4.1
kernelselect.png

4.1.1.2 The notebook

A notebook is a collection of cells. A **cell** is a container that can hold text or computer code. Cells in Jupyter looks like gray rectangles. There are three cell types in Jupyter: (i) Code, (ii) Markdown, and (iii) Raw. The two that we will focus on are Code and Markdown.

The “Code” cell holds computer code that the R kernel (see below about a kernel) can use to compute. We may want to import data, run a statistical analysis, and output results. This is for the “Code” cell.

“Markdown” is itself a special language that a Jupyter Notebook interprets as text. The “Markdown” cell is most useful for write ups, descriptions of a Code cell above or below, or scientific conclusions, comments, and thoughts. When you need to write, think Markdown.

4.1.1.3 Save your work

You can always save your work, and should do so often, by clicking File -> Save Notebook.

4.1.1.4 Export for submission

In class, we will ask that you submit your work on Coursesite as a **PDF**. Work in another format will not be accepted. To export your notebook as a PDF, choose File->Save and Export Notebook As->PDF

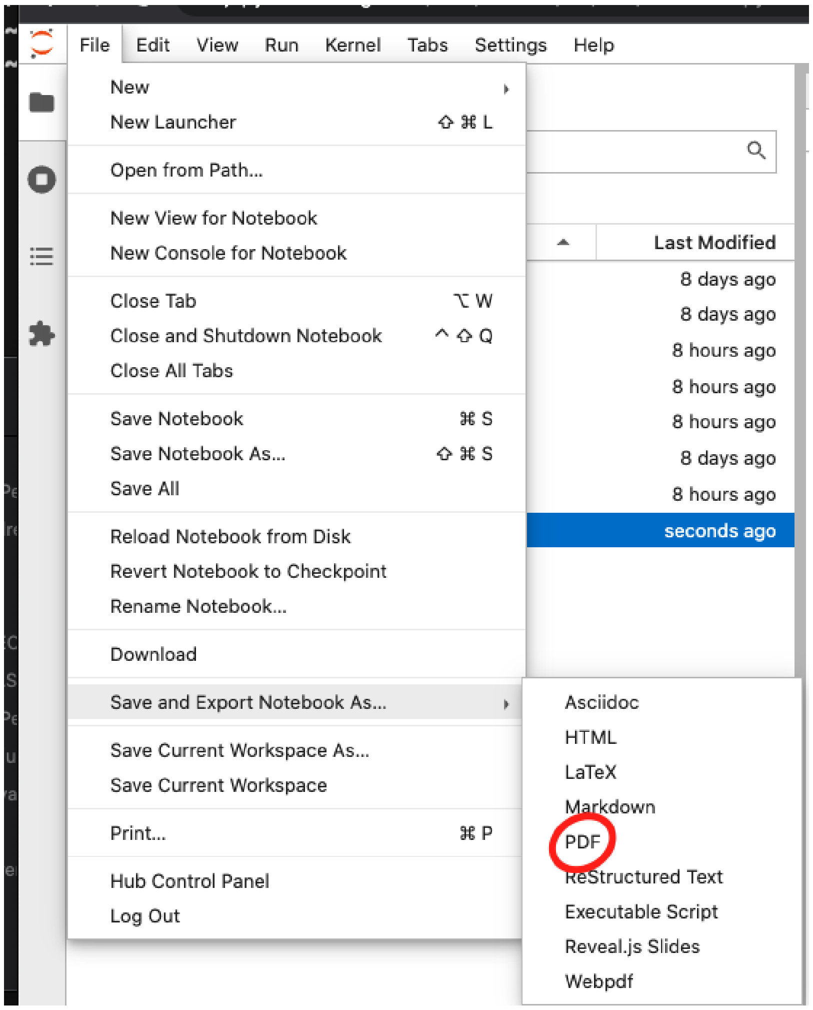


FIGURE 4.2
Caption

After you click PDF, a PDF file will be created and saved in a "Downloads" folder on your local machine. Make sure the PDF file contains (1) Your first and surname, the date, and a descriptive title.

4.1.2 Kernel

The kernel is the component that executes code inside your notebook. No kernel, no running code.

Over the course, you may find that your notebook has disconnected or otherwise will no longer execute the code you wrote. Most often, the kernel has stopped. To restart you kernel select Kernel->Restart Kernel.

4.2 Programming and R

The R programming language, while not explictly written for statistics, has a long history as a tool for data analysis, statistics, machine learning, and data science. R supports all of the main paradigms in computing and you will be able to transfer what you learn in R to other programming languages without much difficulty.

Programming is difficult. Like any skill, programming take time to master. Error messages will be commonplace, you will find it difficult to ask the computer to calculate what you want. You will be frustrate and that is ok. Over time you will learn to read the error messages, code will flow more easily. The most important part of programming is daily practice.

When we **execute** code, we ask the computer to translate what we wrote into binary and return a set of results that may or may not be stored in memory. In the Jupyter environment we execute code by pressing "Run" or by using the shortcut "Shift+Enter".

4.3 Arithmetic

R supports all standard artithmetic calculations. Lets "Run" our first computation.

R can interpret addition

[100]:
```
2+2
```

4

Subtraction

[101]:
```
9-3
```

6

Division

```
[102]: 3/4
```

0.75

multiplication

```
[103]: 4*4
```

16

and exponentiation

```
[104]: 3^9
```

19683

As expected, we can compute more difficult arithemetic expressions.

```
[105]: (2^4)+3/2 - 1
```

16.5

4.4 Vectors

The **vector** is the fundemental object in R.

A mathematical vector is an ordered list of numbers. They are denoted by a sequence of numbers surrounded by square brackets.

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \tag{4.1}$$

Above, the vector **v** is a vector of length 3 and contains, in order, the values 1, 2, and 3.

In R, vectors are goven a name and stored in the computer in one of two ways: (i) using the **c()** operator or (ii) using the assign function.

4.4.1 Assignment

\nobreak

4.4.1.1 c()

We can store a vector named v with the values 1,2,3 in R as follows

[106]:
```
v = c(1,2,3)
```

4.4.1.2 assign

We can also use the assign function to store a vector, named q, with the values 3,2,1 as follows

[107]:
```
assign("q",c(3,2,1))
```

4.4.1.3 equals

The equals sign **does not** represent two objects are equal to one another. The equals sign in compiuter programming stands for "assign".

When we write `v = c(1,2,3)`, this is understood as "we assign the variable v to the vector (1,2,3). As an example, lets create a vector `(4,5,6)` names `x` and then assign the variable `y` to be the same as `x`

[108]:
```
x = c(4,5,6)
y = x
```

The last line above does not ask whether or not `x` is the same as `y`. Instead, this line assigns the variable `y` to be the same vecor as `x`.

4.5 Print

When we created the vectors **v** and **q** "nothing happened". Though the vector v and q were created and stored in the computer, R does not display these on your screen by default. One way to view any object in R is to print it.

You can print an object, `x`, R by writing `print(x)`

[109]:
```
print(v)
```

```
[1] 1 2 3
```

[110]:
```
print(q)
```

```
[1] 3 2 1
```

[111]:
```
print(x)
print(y)
```

```
[1] 4 5 6
[1] 4 5 6
```

You do not need to print any object, ever. Printing is not necessary. You should use print to explore whether you programmed something write or to communicate scientific results.

4.6 Combining vectors

We can append one vector to another in R by using the c() operator. Suppose we wish to combine the two vectors

$$x = \begin{bmatrix}1\\2\\3\end{bmatrix} ; z = \begin{bmatrix}-1\\0.2\\90\end{bmatrix} \tag{4.2}$$

into one vector

$$r = \begin{bmatrix}1\\2\\3\\-1\\0.2\\90\end{bmatrix} \tag{4.3}$$

Lets first create the vectors x and z

[112]:
```
x = c(1,2,3)
z = c(-1,0.2,90)
```

Now we can create the vector r

[113]:
```
r = c(x,z)
```

If we want to check our work, we can print out r.

[114]:
```
print(r)
```

```
[1]  1.0  2.0  3.0 -1.0  0.2 90.0
```

4.7 Indexing and access

Vectors are useful for storing several different numbers. We can access single elements, or several elements inside a vector by (i) naming the vector we want to access, (ii) typing square brackets "[]".

4.7.1 Numeric indexing

If we want to access the 4th element in `r`, we can type

```
[115]: r[4]
```

-1

If we want to access, the 2nd, 4th, and then the first element of `r` we can include in square brackets the vector `c(2,4,1)`

```
[116]: r[c(2,4,1)]
```

1. 2 2. -1 3. 1

We can access the 1st,2nd, and 3rd elements in `r` using the vector `c(1,2,3)`, however a shortcut is to use the **colon** operator. The colon operator takes as input two integers (a,b) separated by a colon (a:b) and expands to the vector `c(a,a+1,a+2,a+3,...,b)`.

Watch

```
[117]: z = 3:5
```

```
[118]: print(z)
```

```
[1] 3 4 5
```

The colon operator is useful for accessing items in a vector

```
[119]: r[2:5]
```

1. 2 2. 3 3. -1 4. 0.2

The above is called **numeric indexing**. Numeric indexing is the access of elements in an object (here a vector) by inputting a single number, or vector of numbers. Indices are always integers. Fractional or decimal numbers cannot be used as indices. Up until now we only used *positive* integers to access elements of a vector.

R also accepts negative integers as indices. Warning: R handles negative indices different than the majority of other progamming languages. A negative index in R standard for **exclude**.

For example, if we want to return all the elements of a vector `q = c(1,4,6,10,0.5)` except for the 2nd element, we can write `q[-2]`

```
[120]: q = c(1,4,6,10,0.5)
       q[-2]
```

1. 1 2. 6 3. 10 4. 0.5

4.7.2 Logical vectors and logical indexing

\nobreak

4.7.2.1 True and False

R, like all programming languages, understands how to operate with binary logic (aside: binary logic is not the only type. If interested, google the tetralemma). True in R is represeted as the word `TRUE` in all capitals. False in R is represented as the word `FALSE` in all capitals. The symbols `TRUE` and `FALSE` are reserved, special symbols in R. You cannot assign a variable to `TRUE` or `FALSE`.

```
[121]: TRUE
```

TRUE

```
[122]: FALSE
```

FALSE

4.7.2.2 Logical comparisons

R understand the following logical operators: - > "Greater than" - >= "Greater than or equal to" - < "Less than" - <= "Less than or equal to" - == "Is equal to" - != "Not equal" - `|` "OR" - `&` "AND"

4.7.2.3 Logic

Logic is a method to evaluate statements, sometimes called propositions as either True or False. The above symbols are used to evaluate statements.

When you pose a proposition to R, such as `v > -1` R will evaluate that propositon for each individual element in the vector `v`. Lets create the vector `v = c(-10,10,4)` and ask R to evaluate the proposition `v>-1`.

[123]:
```
v = c(-10,10,4)
v > -1
```

1. FALSE 2. TRUE 3. TRUE

We see that R returns a vector with the same numebr of elements as in v containing the values TRUE or FALSE. A vector that contains values TRUE/-FALSE is called a **logical vector**. Like any other vector we can store a logical vector.

[124]:
```
log = v>-1
```

[125]:
```
print(log)
```

```
[1] FALSE  TRUE  TRUE
```

4.7.2.4 AND, OR, and NOT

AND, OR, and NOT are **logical operators**, they allow us to combine one or more propositions. Given two propositions p_1 and p_2, the AND, OR, and NOT operator will evaluate to the following

p_1	p_2	p_1 AND p_2	p_1 OR p_2	$!p_1$
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

TABLE 4.1
Caption

Logical operators come in handy in **logical indexing**. When we write `r[l]` where `l` is a logical vector, R will return the the values in `r` where `l` is TRUE.

[126]:
```
r = c(-10,0,10,55,0.34,-0.97)
l = c(TRUE,FALSE,TRUE,FALSE,TRUE,TRUE)

r[l]
```

1. -10 2. 10 3. 0.34 4. -0.97

More often we will include the logical statement directly inside the square brackets

[127]:
```
r[ r>0 ]
```

1. 10 2. 55 3. 0.34

4.7.2.5 Equivalence of TRUE and FALSE to 1 and 0

The symbol `TRUE` in R is understood to be the same as the value `1`, and the symbol `FALSE` in R is understood to be the same value as `0`.

[128]:
```
TRUE==1
```

TRUE

[129]:
```
FALSE==0
```

TRUE

[130]:
```
TRUE==0
```

FALSE

[131]:
```
FALSE==1
```

FALSE

4.7.3 Two functions that are useul for operating on vectors

Functions in mathematics take as input a list of objects and return a unique object. The same is true of functions in programming and so in R.

We can create our own functions in R (this will come later), but R also has a large library of built-in functions that are automatically included once you start R. Two very sueful ones are the `sum` function and the `length` function.

The `sum` function takes as input a vector and returns the sum of each element in the function.

[133]:
```
v = c(3,2,1)
sum(v)
```

6

The `length` function takes as input a vector and returns the number of elements in the vector

[134]:
```
length(v)
```

3

4.8 Assignment 01

We are recruited to track the evolution of an infectious agent for a team of public health officials (PHOs). To support future strategic planning the PHOs want to know the impact of intervention X on increases or decreases in the incidence of this infectious agent. The PHO team collected for each county in their state whether the intervention was enacted, and whether the incidence of case counts of this infectious agent increased or decreased 60 days after the intervention was in place data.

Using R, we will assign probabiltiies to the four events in our sample space { (intervention,raise),(intervention, no raise),(no intervention, raise),(no intervention, no raise) }

4.8.1 The data

In the below cell there is a few lines of code pre-programmed. Please runs this cell below.

This cell will create two vectors.

The first vector is called `intervention_rise` and contains one element for each county that has intervention X collected by the PHO team. An element is the value `1` if there was a rise in incidence for the infectious agent and `0` if there was a fall in incidence.

The second vector is called `nointervention_rise` and contains one element for each county that did not have intervention X collected by the PHO team. An element is the value `1` if there was a rise in incidence for the infectious agent and `0` if there was a fall in incidence.

4.8.2 Please complete the following

1. Use the `length` function to count the number of counties where intervention X took place
2. Use the `length` function to count the number of counties where no intervention X took place
3. Use the `sum` function to count the number of counties that observed a rise in incidence.
4. Use the `sum` function and the `not (!)` operator to count the number of counties that observed a fall in incidence.
5. Use the frequentist approach to compute the probability
    - that an intervention would take place in a county
    - that a rise in incidence was observed in a county
    - of a rise in incidence given a county implemented intervention X

    - of a rise in incidence given a county has not implemented intervention X
6. Use the multiplication rule to compute the probability
    - that an intervention and rise is observed (Hint: P(Intervention) * P(Rise | Intervention))
    - that an intervention and fall is observed
    - that no intervention and rise is observed
    - that no intervention and fall is observed
7. Compute the probability of the below events, assuming intervention and rise/fall. are independent
    - Intervention and Rise
    - Intervention and Fall
    - No Intervention and Rise

    - No Intervention and Fall
8. Do you think intervention X is effective at preventing the spread of our infectious agent?

[178]:
```
#RUN THIS CODE. DO NOT WORRY WHAT IT SAYS.
nums = runif(10^3,0,1)

intervention_rise   = c()
nointervention_rise = c()
for (i in nums){
    if (runif(1)> 0.4){
        if ( i>0.800 ){
          risefall = 1
        } else{risefall=0}
        intervention_rise = c(intervention_rise, risefall)
    }
    else {
        if ( i>0.325 ){
          risefall = 1
        } else{risefall=0}
        nointervention_rise = c(nointervention_rise, risefall)
    }
}
```

5

Laboratory 02

thomas mcandrew, david braun
Lehigh University

CONTENTS

5.0.1 Control flow

Now that we defined some fundemental objects and operations in R, we need to explore how R, and many other programmign language execute code. R executes code sequentially from top to bottom. The line of R code at the top is run first, then the second highest line, then the third highest and so on.

When code is executed line by line from top to bottom it is called **sequential control**. Sequentuntial control if the default way R executes statements.

However, we can change the order in which R executes lines of code in three ways: (i) choice (ii) repetition (iii) functions (we'll talk much more about functions next week).

5.0.1.1 Choice or Selection

An R **program** is a set of statements meant to produce one or more results. We can choose to execute all the lines in our program, or we can choose to execute some lines and not others depedent on conditions. When we execute some lines of R code to run and not others we are using a specific type of control flow called **Selection** ("we select some lines and not others").

While we often define statements as sentences that are either TRUE or

FALSE, in R we define a logical statement that evaluates to TRUE or to FALSE an **expression**.

The main ways to control which lines to execute is with the if/else, if/elsif/else, and the switch statement.

If/else

The following study at this link = https://doi.org/10.1200/jco.2005.03.0221 chose to study the effects of a novel treatment on newly diagnosed myleoma. The study investigators random;y assigned 103 patientrs to receive a control therapy and 104 patients to receieve the novel treatment. The primary outcome of interest was the rate of response among patients in control and treament where a response was defined as a 50% or greater decrease in detection of the cancer.

We could decide to define a vector v that will contain three pieces of information: a patient id that is an integer used to link a single patient to their clincial records, whether the patient was assigned to recieve a treatment or control therapy, and the percent response where 100 means complete response/reduction and 0 means no response/reduction.

[70]:
```
pt = c(1256, "TREATMENT", 87)
```

We stored a vector called `pt` with our three pieces of info. Now suppose we want to add an additional piece of info to our patient vector that determines if the patient had a succussful or unsuccussful response.

We need the if/else sytax. The if/else is an expression placed inside of parentheses and two blocks of code. if the expression evaluates to TRUE the first block of code will be executed and the second block is skipped. If the expression evaluates to FALSE then the second block will be executed and first block will be skippedd.

[71]:
```
if( pt[3] > 50 ){  # Our expression is pt[3] > 50
    pt = c(pt,1)
}else{
    pt = c(pt,0)
}
```

We expect that the code above will create a new vector `c(1256,"TREATMENT",87,1)`. To be sure, lets check the variable `pt` by printing it.

[72]:
```
print(pt)
```

```
[1] "1256"      "TREATMENT" "87"        "1"
```

if/elseif/else and switch

The **if/else** handles one condition that either evaluates to TRUE or FALSE. However, we may need to execute a specific piece of code that depends on more than one condition. The **if/else if/else** syntax can execute a specific block of code dependent on more than one condition.

For example, suppose that we want to classify a patient's response as either above 75, between 50 and 75, not including 50 and 75, or less than or equal to 50. Let us also assume our patient is a following vector:

```
[73]: pt = c(6587,"CONTROL",24)
```

We can use the if /else if/ else syntax.

```
[74]: if( pt[3] > 75 ){
          pt = c(pt,2)
      }else if ( pt[3] >50 ) {
          pt = c(pt,1)
      }else {pt = c(pt,0) } # Our if/else if/else will run this␣
       ↪code bc the above two conditions are not met.
```

The if/else if/ else syntax begins at the top condition (`pt[3] > 75`). If the top condition is FALSE then the condition second to the top is evaluated next (`pt[3] > 50`). If the second condition is FALSE, and there are no additional `else if` conditions, then code in the `else` block is executed.

```
[75]: print(pt)
```

```
[1] "6587"    "CONTROL" "24"      "0"
```

Compund expressions

The expressions we used above evaluated a single expression as either TRUE or FALSE. At times we may need more complicated expressions that must evaluate several expressions at once. We will call these compund expressions.

For example, we may want to treatment response values different if the patient was assigned to treatment versus control. Suppose if a patient is assigned control and has a response above 50 they receieve a 1, if a patient is assigned control with a response equal to or less than 50 they recieve a 0. If a patient is assigned treatment and has a response above 50 they receieve a 2 and if their response is equal to or less than 50 they recieve a 3.

We need to evaluate two expressions based on the patient assignment and response. We are allowed to include logical comparisons (LAB01) inside expressions.

[76]:
```
pt = c(1359, "TREATMENT", 72)

if( (pt[3] > 50) && (pt[2]=="CONTROL") ){
    pt = c(pt,1)
}else if ( (pt[3] <= 50) && (pt[2]=="CONTROL") ){
    pt = c(pt,0)
}else if ( (pt[3] > 50) && (pt[2]=="TREATMENT") ){
    pt = c(pt,2)
}else if ( (pt[3] <= 50) && (pt[2]=="TREATMENT") ){
    pt = c(pt,3)
}

# check our work
print(pt)
```

```
[1] "1359"      "TREATMENT" "72"        "2"
```

The above code used the logical comparison `AND (&&)` to test whether the patient was treatment/control and if there repsonse was above or below 50.

5.0.1.2 Repitition

Often we will need to repeat a certain number of lines of code. We may need to apply the same operation to many different patients or observations or apply some set of code repeatedly until a condition it met. Loops allows us to **repeat** lines of code before continuing to execute lines of code below our loop.

While loop

The while loop repeats lines of code—called a **block**—until a specific condition is met.

[77]:
```
x = 2
while(x<1000){
    x = x^(2)
}
print(x)
```

```
[1] 65536
```

In the code above, we used the syntax `while (CONDITION) {CODE}` to execute a while loop. The code above assigned the variable `x` the value 2. The next line of code was the while loop. Because the condition evaluated to FALSE (x was not less than 1000) the code inside the parentheses—the code block—was executed once. After executing the code block the condition `x<1000` was tested again. Becasue the condition evaluated to FALSE, the code block was executed again, and again, and again, until the condition evaluated to TRUE. After the code block evaluated to TRUE, we continued to execute lines below the while loop in sequence.

The above code is equivalent to the following expanded code

[78]:
```
x=2
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
if (x <1000){
    x=x^2
}
print(x)
```

```
[1] 65536
```

and so the while loop is a natural way to repeat a series of R code until meeting some condition.

For loop

Suppose we want to execute a code a block a *fixed* number of times. We could use a while loop to square the variable `x` four times using the below code:

[79]:
```
number_of_times=1
x = 2
```

```
while (number_of_times <5){
    x=x^2
    number_of_times = number_of_times+1
}
print(x)
```

```
[1] 65536
```

Because repeating code a fixed number of times is needed so often to solve a problem, a second type of loop was created—the **for loop**. The **for loop** repeats a code block a fixed number of times by iterating through a sequence.

Sequences

A sequence is a mapping, or function, from the numbers 1,2,3,4,… to some set of items $a, b, c, d, \cdots$. We often denote the items $a, b, c, d, ...$ with a single variable name and a subscript that is called an index: $a_1, a_2, a_3, a_4, \cdots$. In the above sequence the number 1 maps to a_1, the number 2 maps to a_2 and so on.

We can create sequences of integers in R with the function `seq`. We can provide `seq` two integers, a "from" and "to" and this function will produce all integers between "from" and "to", including both the "from" value and the "to" value. We will produce all integers from -2 to 8. Watch

```
[80]: seq(-2,8)
```

1. -2 2. -1 3. 0 4. 1 5. 2 6. 3 7. 4 8. 5 9. 6 10. 7 11. 8

This sort of request is so common in R that there is an easier way to write "from,to" sequences. We place the "from" value to the left and the "to" value to the right of a colon.

```
[81]: -2:8
```

1. -2 2. -1 3. 0 4. 1 5. 2 6. 3 7. 4 8. 5 9. 6 10. 7 11. 8

R understand that the colon asks to produce all integers starting with the value on the left and ending with the value on the right.

Back to the for loop

Lets look how we can use sequences and the idea of the for loop to rewrite our code.

```
[82]: x=2
      for(number_of_times in 1:4){
          x=x^2
      }
      print(x)
```

```
[1] 65536
```

We can think of the for loop above in expanded form.

[83]:
```
x=2

x=x^2
x=x^2
x=x^2
x=x^2

print(x)
```

```
[1] 65536
```

5.0.1.3 Assignment

Lets use our new control flow skills to compute probabilities and conditional probabilities.

Run the below code to create one vector that describes whether patients were assigned to treatment or control and a second vector describing whether the patient had a succussful or unsuccesful status.

[84]:
```
draw_random_number = function(){
    return(runif(1))
}
```

Practice loop and conditional structure

After running the above code block, we can create a random number by running the following code `draw_random_number()`

1. Run the code `draw_random_number()`
2. Create and if/else statement that prints "Less than 1/2" when `draw_random_number()` is smaller than the value 0.5 and else prints "greater than or equal to 1/2" when `draw_random_number()` is 0.5 or larger.
3. Create a variable `category` and assign this variable the value -1.
4. Create an if/else if/else statement that assigns the variable category the value 1 when `draw_random_number()` is smaller than 0.25, the value 2 when `draw_random_number()` is greater than or equal to 0.25 and smaller

than 0.50, the value 3 when `draw_random_number()` is greater than or equal to 0.50 and smaller than 0.75 and the value 4 otherwise.

5. Create a vector random_values that is empty (i.e. `random_values = c()`).
6. Assign the variable `value` equal to `draw_random_number()`.
7. Create a while loop with the condition that `value` returns a value less than 0.5 - Inside the while loop, in the code block, assign the variable `value` to a new random number `draw_random_number()` - Inside the while loop, in the code block, append `value` to your vector `random_values`
8. Create a for loop that iterates the code block you create for the while loop 10 times.
9. What can you say about the vector `random_values` that would be produced from 7. versus 8.?

[85]:
```
patient_status    =␣
  ↪c(0,1,1,0,0,1,1,1,0,0,0,1,1,1,0,0,0,0,1,0,0,0,1,1,1,0)
patient_treatment =␣
  ↪c("t","c","t","c","t","c","t","t","c","t","c","t","c","t","c","t","c","t","t","c","t",␣
  ↪"c","c","t","c","c")
```

Lets create a for loop that will iterate through the positions of our two vectors `patient_status` and `patient_treatment`, performing different operations as we move to each item in the vector

1. Use the `length` function to compute the number of patients in the study and store that length in the variable `N`
2. Create a variable called `num_of_treat_assigns` and assign that variable the value 0
3. Create a variable called `num_of_contr_assigns` and assign that variable the value 0
4. Create a variable called `num_of_status_success` and assign that variable the value 0
5. Create a for loop that will run a code block starting at 1 and ending at `N`
6. Inside the code block of the for loop, use the variable `num_of_status_success` to count the number of patients with a succussful status
7. Use `N` and `num_of_status_success` to compute the probability of a success
8. Lets make our for loop more complicated. Inside our code block, create an if/else such that when patient treatment is a "t" we increment the variable `num_of_treat_assigns` by 1, else we increment the variable `num_of_contr_assigns` by 1.
9. Lets add to our if/else statement. Create two variables `num_of_trt_status_success` and `num_of_ctr_status_success` and

assign them both the value 0. Inside our loop, when a patient treatment is “t” we will implement step 8 and in addition we will increment the variable `num_of_trt_status_success` by 1. If the patient treatment is “c” then we will increment `num_of_ctr_status_success` by 1.

10. Using `num_of_trt_status_success` and `num_of_treat_assigns` compute the conditional probability of success given the patient was assigned treatment.
11. Using `num_of_ctr_status_success` and `num_of_contr_assigns` compute the conditional probability of success given the patient was assigned control.

6

Laboratory 03

thomas mcandrew, david braun

Lehigh University

CONTENTS

\nobreak

6.0.1 Functions

Functions in R are used to organize our work, clarify code for others, and allows us to apply the same finite steps to many different objects.

Example: Suppose we want to use the frequentist approach to assign probabilities to a sample space $\mathcal{G} = \{-1, 0, 1\}$ based on a dataset $\mathcal{D}$. Our dataset may look like $\mathcal{D} = (-1, 0, 0, 1, -1, 1, 1, 1, -1)$. To assign a probability to the event $\{-1\}$ we can write R code to count the number of times the outcome -1 appears in our data and divide by the number of data points in our data set. For the event $\{0\}$ we can write R code to perform the *same steps* for 0 as we performed for the event $\{-1\}$. Because we are repeating the same steps for the event $\{-1\}$ and for event $\{0\}$, a function may simplify our code.

Example: We are asked to support a clinical team that collected data on patients who are current smokers and outcomes thought to be linked to smoking. The code needs (i) to be processed, (ii) analyzed, and (iii) reported. Though we can write our code in sequence to perform all three steps, we may be able to better organize our work into three functions: one that processes the data, a function that analyses the data, and a third to report.

6.0.1.1 Anatomy of a function

A function in R has the following X parts:

- *Assignment:* We create a function by assigning a variable name to the function
- *Function* : Next we need to use the reserved word "function" so that R knows we are creating a function.
- *Arguments*: After the workd function we will include open and closed parentheses. Inside these parentheses we can include one or more arguments for the function. An argument is an input to our function.
- *Code block*: We write a sequence of steps to execute R code inside two curly brackets {}. **Important: Any variables that were created inside this code block are deleted after the function is finished.**
- *Return*: Inside the code block we can also include a return statement. This statement includes variables that were generated inside the code block that we wish to keep when the function is finished executing.

6.0.1.2 Declaring a function

When store a function in memory, we call this **declaring a function**.

Lets declare a function called `sum_two_numbers`. This function will have two arguments: one called `x` and one called `y`. Arguments are names that we use to identify specific inputs to our function—they are placeholders for variables outside of our function that we may want to use as inputs. In the code block we will write a line of code that stores the sum of `x` and `y` as the variable `z`. The second line in the code block will `return(z)`. Because we did not return `x` or `y` they will be deleted from memory after the function is finished executing.

[10]:
```
sum_two_numbers = function(x,y){
    z = x+y       # line of code in the code block. This uses␣
 ↪our x,y arguments (place holders)
    return(z)     # Return the variable z
}
```

6.0.1.3 Calling a function

When we declare a function it is stored in memory. If we want to apply our function to a set of arguments (inputs) then we **call** our function.

Lets call our function `sum_to_numbers` with the arguments 2 and 4.

[11]:
```
result = sum_two_numbers(2,4)
```

We **called** our function by typing the name of the function and supplying the function with two arguments: 2 and 4. When we called the function, the following took place. 1. The variable `x` was assigned the value 2. 2. the variable `y` was assigned the value 4. 3. The first line of the code block was executed. 4. The second line of the code block was executed, returning the variable `z`. 5. The returned variable (`z`) was stored in the variable `result`.

Watch

[12]:
```
print(result)
```

```
[1] 6
```

6.0.1.4 Named vs unnamed arguments

When we input our argument 2 and 4 into the function `sum_to_numbers` we did not specify which value should be assigned to the argument `x` and which should be assigned to `y`. When we do not specify the argument names we are proving a function **unnamed arguments**.

We could have called the function `sum_to_numbers` by specifying which arguments correspond to which values. When we supply a name and the value to the argument we are providing **named arguments**.

[13]:
```
sum_to_numbers(y=2,x=4)  # named arguments
```

6

6.0.1.5 Default arguments

When we declare a function we specify which arguments are needed for the function to execute all the lines in the coded block. We expect all these arguments to have values, but there may be a time when we do not necessarily need someone who uses our function to specify all the arguments. Instead, we can provide **default** argument values.

Lets create a function called `sumMult` that takes as input a vector that we will assign the name `v` and a logical value called `sum_or_mult`. If the value of `sum_or_mult` is TRUE then we will add all the items in the vector. If the value of `sum_or_mult` is FALSE then we will multiply all the values.

By default, we will add all the items. This means that is the user does not supply a value for the argument `sum_or_mult` then we automatically assign to `sum_or_mult` the value TRUE. To give the function `sumMult` a default value for the argument `sumMult` we will include after `sum_or_mult` an equals sign and our desired default value.

```
[2]: sumMult = function(v,sum_or_mult=TRUE){ # This assigns the␣
      ↪sum_or_mult a default value of TRUE

         ## if sum_or_mult is TRUE, we sum. if sum_or_mult is␣
      ↪FALSE, we multiply
         if (sum_or_mult==TRUE){
             summation=0
             for (item in 1:length(v)){
                 summation = summation + item
             }
             return(summation)
         }
         else{
             product = 1
             for (item in 1:length(v)){
                 product = product * item
             }
             return(product)
         }
     }
```

Lets create a vector called fun_vector and assign to it c(4,2,-2,9,10,11,-0.3) and lets call the function sumMult.

```
[20]: fun_vector = c(4,2,-2,9,10,11,-0.3)

      result1 = sumMult(fun_vector)                     # Default for␣
       ↪sum_or_mult is TRUE

      result2 = sumMult(fun_vector,sum_or_mult=TRUE)  # We are␣
       ↪always allowed to assign this argument a value

      result3 = sumMult(fun_vector,sum_or_mult=FALSE)  # and we can␣
       ↪assign this argument a different value then the default
```

```
[21]: print(result1)
      print(result2)
      print(result3)
```

```
[1] 28
[1] 28
[1] 5040
```

6.0.1.6 Binding and Scope

To show that the variables summation and product are deleted after we call the function, lets try to print the variable `summation`.

[23]:
```
print(summation)
```

```
Error in print(summation): object 'summation' not found
Traceback:

1. print(summation)
```

R replies that it looked but cannot find the object summation. The function `sumMult` created a variable called `summation` operated with it inside the code block of our function and then deleted this variable.

When a variable name is assigned an object in R, the name and the object are associated with one another in the computer. The process of associating an object to a name in the computer is called **binding**.

When we call a function, the values we provide are bound to each argument name, the function is executed, and those variables are deleted. Which variables we can access during the executing of an R program is called **lexical scope**. Varables created inside a function can only be accessed and used by lines of code inside the code block. These variables are "in scope" of the function.

6.0.1.7 Assignment

Lets explore how functions can help us organize our work, clarify code for others, and repeat similar operations on objects of the same type.

Unnamed and named arguments

1. Declare a function `subtract` that takes two arguments: `x` and `y`. Inside the code block assign the variable `z` to be `x` minus `y` and return the variable `z`.
2. Call the function `subtract` on the values 2 and 6.
3. Call the function `subtract` on the values 2 and 6 but bind 2 to the argument `y` and 6 to the argument `x`.
4. Why did the results for 2. and for 3. change?

Successive subtraction

1. Declare a function `vec_subtract` that takes a vector argument that we will bind to `v`. Inside the code block compute the first item minus the second item minus the third item and so on. Store this subtarction in the variable `s` and return the variable `s`.

2. Call the function on the vector [1,2,3,4,5]

Frequentist approach to probability assignment

1. Run the code below called `random_vector`. This is a function that creates a vector of length 1000 filled with numbers between 0 and 1. The second line calls the function with no arguments and creates a variable called `rand_vec`.
2. Declare a function `freq_assign` that takes two arguments: `v` and `outcome`.
3. Inside the code block perform the following operations
4. Use the `length` function (pre-built by R) to compute the length of `v` and store this value in the variable `N`.
5. Create a variable called `outcome_of_interest` and assign to it an empty vector `c()`.
6. Create a for loop that iterates the variable `i` from the value 1 to the value `N`.
7. Inside the for loop use an if/else to identify if each item in `v` is less than or equal to `outcome`. If the item is less than or equal to `outcome` then append the value 1 to `outcome_of_interest` else append the value 0. We should expect a vector `outcome_of_interest` that is the same length as `v` which contains 1s for every value less than outcome and 0s otherwise.
8. Use the sum function in R (pre-built) to compute the sum of `outcome_of_interest` and assign this value the name `count`
9. return `count/N`
10. Call the function on `rand_vec` and record the result. Use 0.5 as the input value for `outcome`.

[8]:
```
random_vector = function(N=1000){
    return(runif(N))
}
rand_vec = random_vector()
```

7

Laboratory 04

thomas mcandrew, david braun
Lehigh University

CONTENTS

\nobreak

7.1 Matrix Algebra

Statisticians and data scientists use the language of vectors and matrices to organize data and work with models. We will take a close look at matrix algebra, taking time to relate matrix algebra to the more familar algebra that you have worked with in the past.

7.1.1 Recap on vectors and operations on vectors

\nobreak

7.1.1.1 Definition

A vector v is an ordered list of real numbers. We denote a vector with a lower case letter and enclose the values in the vector with square brackets. We can write a vector v as a *row vector*

$$v = [1, 2, 3, 4, 5] \tag{7.1}$$

or as a *column vector*

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix} \tag{7.2}$$

.

A vector has a **length**, defined as the number of values included in that vector. For example, the above vector is length 5.

We can create a vector in R by using the c operator.

```
[1]: v =c(1,2,3,4,5)
```

7.1.1.2 Vector times a scalar

A vector of length one is called a **scalar**. We can define the results of multiplying a vector $v = [1, 2, 3, 4]$ by a scalar α as each entry of the vector times the scalar α.

$$\alpha v = \begin{bmatrix} \alpha \times 1 \\ \alpha \times 2 \\ \alpha \times 3 \\ \alpha \times 4 \end{bmatrix} \tag{7.3}$$

For example, the vector $v = [1, 2, 3, 4, 5]$ times the scalar $\alpha = 7$ will result in a vector

$$\begin{aligned} \alpha v &= [7 \times 1, 7 \times 2, 7 \times 3, 7 \times 4] && (7.4) \\ &= [7, 14, 21, 28] && (7.5) \end{aligned}$$

R understands how to multiply vectors and scalars with no additional syntax.

[3]:
```
v = c(1,2,3,4)
alpha = 7

alpha*v
```

1. 7 2. 14 3. 21 4. 28

7.1.1.3 Vector plus/minus a vector

A vector v plus a vector q creates a new vector that adds the individual entries of v and q

$$v = [4, -1, 7] \tag{7.6}$$
$$q = [0, 32, 9] \tag{7.7}$$
$$v + q = [4+0, -1+32, 7+9] = [4, 31, 16] \tag{7.8}$$

A vector v minus a vector q creates a new vector that subtracts the individual entries of q from v

$$v = [4, -1, 7] \tag{7.9}$$
$$q = [0, 32, 9] \tag{7.10}$$
$$v - q = [4-0, -1-32, 7-9] = [4, -33, -2] \tag{7.11}$$
$$q - v = [0-4, 32-(-1), 9-7] = [-4, 33, 2] \tag{7.12}$$

R also understands vector addition and subtraction with no special syntax.

[6]:
```
v = c(4, -1, 7)
q = c(0, 32, 9)

add = v+q
subtract = v-q

print(add)
print(subtract)
```

```
[1]  4 31 16
[1]   4 -33  -2
```

7.1.2 The Matrix

A **matrix** is an ordered list of vectors.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 3 & 2 & 13 \\ 90 & 23 & 0 \end{bmatrix} \tag{7.13}$$

A matrix has a dimension which is an ordered pair of numbers: the first number indicates the number of rows of the matrix and the second number indicates the number of columns. For example, he matrix M above has dimension $(4,3)$.

To build a matrix in R we can issue the matrix command. Matrix is a function that takes a vector as an argument and can take one of many optional arguments.

If we give the matrix function only a vector, the default behavior of this function is to create a vector.

```
[8]: A = matrix(c(1,2,3,4,5,6))
     A
```

```
                                  1
                                  2
                                  3
                                  4
                                  5
A matrix: 6 × 1 of type dbl       6
```

This is because a vector can be thought of as a matrix with one column. In other words, a matrix with dimension $(N,1)$ is a vector of length N.

We can provide another arguement to the `matrix` function—ncol—to specify the number of columns we want to create.

```
[9]: A = matrix(c(1,2,3,4,5,6), ncol=2)
     A
```

```
                                 1  4
                                 2  5
A matrix: 3 × 2 of type dbl      3  6
```

Above we used the ncol argument to specify a matrix with two columns. R will automatically determine the number of rows for the matrix and fill-in the values of the matrix column by column.

If we want, we can ask R to fill in values of the matrix row by row by including an additional argument in the matrix function`byrow`

[10]:
```
v = c(1,2,3,4,5,6)
A = matrix(v, ncol=2)

B = matrix(v, ncol=2, byrow=TRUE)

print(A)
print(B)
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

We see that matrix A was "column filled" and matrix B was "row filled".

There are similar operations for matrices as there are for vectors.

7.1.2.1 Matrix plus/minus a matrix

Given a matrix A and matrix B, the sum of A and B is a new matrix C where the i,j entry of C (C_{ij}) is the sum of the corresponding entries from A and B ($C_{ij} = A_{ij} + B_{ij}$). To add two matrices they must have the same dimension.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \tag{7.14}$$

$$B = \begin{bmatrix} 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} \tag{7.15}$$

$$C = A + B = \begin{bmatrix} 1+6 & 2+5 & 3+4 \\ 4+3 & 5+2 & 6+1 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix} \tag{7.16}$$

$$D = A - B = \begin{bmatrix} 1-6 & 2-5 & 3-4 \\ 4-3 & 5-2 & 6-1 \end{bmatrix} = \begin{bmatrix} -5 & -3 & -1 \\ 1 & 3 & 5 \end{bmatrix} \tag{7.17}$$

$$\tag{7.18}$$

R understands matrix addition and subtraction without any additional syntax.

[17]:
```
A = matrix(c(-1,2,4,1,3,8), ncol=3)
D = matrix(c(90,0.34,1.54,-9.43,10,0), ncol=3)

print(A)
print(D)

A+D
```

```
     [,1] [,2] [,3]
[1,]   -1    4    3
[2,]    2    1    8
      [,1]  [,2] [,3]
[1,] 90.00  1.54   10
[2,]  0.34 -9.43    0
```

A matrix: 2 × 3 of type dbl

89.00	5.54	13
2.34	-8.43	8

[14]:
```
A-D
```

A matrix: 2 × 3 of type dbl

-91.00	2.46	-7
1.66	10.43	8

7.1.2.2 Matrix times a Matrix

Two matrices A and B can be multiplied together $C = AB$ if the number of columns of A is equal to the number of rows of B. In other words, if the dimension of A is (a,b) and the dimension of B is (c,d) then b and c must be equal.

The i,j entry of this product, C_{ij} is the following sum of products:

$$\begin{aligned} C_{i,j} &= a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + a_{i,3}b_{3,j} + \cdots + a_{i,N}b_{N,j} && (7.19) \\ &= \sum_{e=1}^{N} a_{i,e}b_{e,j} && (7.20) \end{aligned}$$

For example, suppose that

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{7.21}$$

and that

$$B = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 4 & -2 \end{bmatrix}. \tag{7.22}$$

Then the product $C = AB$ equals

$$C = AB = \begin{bmatrix} 1*1+2*3 & 1*2+2*4 & 1*-1+2*-2 \\ 3*1+4*3 & 3*2+4*4 & 3*-1+4*-2 \end{bmatrix} = \begin{bmatrix} 7 & 10 & -5 \\ 15 & 22 & -11 \end{bmatrix} \tag{7.23}$$

The above definition and computation can feel cumbersome. We can simplify the above calculations by introducing the inner product.

The inner product between two **vectors** v and q, or $v'q$, is

$$\begin{aligned} v &= [1,2,3] && (7.24) \\ q &= [4,5,6] && (7.25) \\ & && (7.26) \\ v'q &= 1\cdot 4 + 2\cdot 5 + 3\cdot 6 && (7.27) \\ &= 4+10+18 = 32 && (7.28) \end{aligned}$$

Let us use the inner product to simplify the above matrix multiplication. First, we rewrite the matrix A as a stack of two *row* vectors

$$A = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{7.29}$$

where $a_1 = [1,2]$ and $a_2 = [3,4]$

Second, we rewrite the matrix B as a stack of three *column* vectors

$$B = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \tag{7.30}$$

where $b_{1} =

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$, $b_{2} =

$$\begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$, and $b_{3} =

$$\begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

$

Then the product AB is a matrix of inner products

$$C = \begin{bmatrix} a_1' b_1 & a_1' b_2 & a_1' b_3 \\ a_2' b_1 & a_2' b_2 & a_2' b_3 \end{bmatrix} \tag{7.31}$$

Matrix multiplication is different than mulitplication between two variables that represent real numbers because matrix multiplication is **not** commutative—the product AB is not rarely equal to BA.

To compute the product of two matrices in R we need the %*% operator

[22]:
```
A = matrix(c(-1,0,1,-2,-1,3,4,5,6),ncol=3)
B = matrix(c(1,2,3,4,5,6,7,8,9),ncol=3)

print(A)
print(B)

print(A%*%B)
print(B%*%A)
```

```
     [,1] [,2] [,3]
[1,]   -1   -2    4
[2,]    0   -1    5
[3,]    1    3    6
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
     [,1] [,2] [,3]
[1,]    7   10   13
[2,]   13   25   37
[3,]   25   55   85
     [,1] [,2] [,3]
[1,]    6   15   66
[2,]    6   15   81
[3,]    6   15   96
```

Note above that we computed the product AB and BA and these products resulted in different matrices.

7.1.2.3 Matrix times a vector

A matrix A with dimension (r, c) can be multiplied by a vector v if the length of v is c. Then the product Av is a vector with the the i$^{\text{th}}$ entry of Av defined as

$$(Av)_i = \sum_{k=1}^{c} A_{i,k} v_k \tag{7.32}$$

For example, Let the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \tag{7.33}$$

and the vector

$$v = \begin{bmatrix} 4 \\ -3 \end{bmatrix} \tag{7.34}$$

then

$$Av = \begin{bmatrix} 1 * 4 + 2 * -3 \\ 3 * 4 + 4 * -3 \\ 5 * 4 + 6 * -3 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 2 \end{bmatrix} \tag{7.35}$$

The definition above can also be simplifed by using inner products. Let

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \tag{7.36}$$

where $a_1 = [1, 2]$, $a_2 = [3, 4]$, and $a_3 = [5, 6]$. Then the product Av simplifies to

$$Av = \begin{bmatrix} a_1' v \\ a_2' v \\ a_3' v \end{bmatrix} \tag{7.37}$$

We can use the %*% operator in R to multiply together a matrix and a vector

[32]:
```
A = matrix(c(1,2,3,4,5,6), ncol=2, byrow=TRUE)
v = c(4,-3)

print("Matrix A")
print(A)
print("Vector v")
print(v)

print("Av")
product = A%*%v
print(product)
```

```
[1] "Matrix A"
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[1] "Vector v"
[1]  4 -3
[1] "Av"
     [,1]
[1,]   -2
[2,]    0
[3,]    2
```

7.1.2.4 Matrix Transpose

Given a matrix M, the **tranpose** of M, $M^{'}$ or M^{T}, is a matrix where the first row of $M^{'}$ corresponds to the first column of M, the second row of $M^{'}$ corresponds to the second columns of M and so on. If M has dimension (r, c) than $M^{'}$ has dimensions (c, r).

For example, if

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 3 & 2 & 13 \\ 90 & 23 & 0 \end{bmatrix} \tag{7.38}$$

then the tranpose of M,

$$M^{'} = \begin{bmatrix} 1 & 4 & 3 & 90 \\ 2 & 5 & 2 & 23 \\ 3 & 6 & 13 & 0 \end{bmatrix} \tag{7.39}$$

We can use the `t()` operator to transpose a matrix in R.

```
[36]: M = matrix(c( 1,2,3,4,5,6,3,2,13,90,23,0),ncol=3,byrow=TRUE)

print("The matrix M")
print(M)

print("The transpose")
Mt = t(M)

print(Mt)
```

```
[1] "The matrix M"
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    3    2   13
[4,]   90   23    0
[1] "The transpose"
     [,1] [,2] [,3] [,4]
[1,]    1    4    3   90
[2,]    2    5    2   23
[3,]    3    6   13    0
```

7.1.2.5 Matrix inverse

For a super fun overview of matrix multiplication and the concept of identities and inverses, click here!

The identity matrix

The **identity matrix** of dimension r, usually labled I_r, is a matrix with r rows and r columns such that the diagonal elements of the matrix, the (1,1) entry, (2,2) entry, up to (r,r) entry are the number 1 and all other entries are 0. For example,

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7.40}$$

This matrix is called the identity matrix because any matrix A times I returns A. To be more precise, this matrix is the *multiplicative* identity matrix. But the adjective *multiplicative* is usually dropped.

The inverse matrix

The **inverse** of a matrix A—A^{-1}—is a matrix such that, if A has the same number of rows and columns (called square) then

$$AA^{-1} = A^{-1}A = I \tag{7.41}$$

The idea of a matrix inverse is the same as in algebra. If we have a variable a

then the inverse of a, called a^{-1} is the unique number such that

$$a \cdot a^{-1} = a^{-1} \cdot a = 1.$$

In matrix algebra, the identity matrix takes the place of the "1" in algebra.

We can compute the inverse of a matrix A in R using the `solve` function.

[41]:
```
A = matrix(c(3,4,5,6),ncol=2)

print("A")
print(A)

print("A inverse")
Ainverse = solve(A)
print(Ainverse)
```

```
[1] "A"
     [,1] [,2]
[1,]    3    5
[2,]    4    6
[1] "A inverse"
     [,1] [,2]
[1,]   -3  2.5
[2,]    2 -1.5
```

7.1.3 Assignment

1. Define the vector v of length 3 with the following numbers: -1,0,1
2. Define the matrix A with dimension (2,3) by filling, column wise, the values: 1,2,3,4,5,6
3. Multiply A by v (Av)
4. Multiply v by A (vA). Why does R return an error?
5. Write a function that takes two argument which are both matrices and returns the product of these matrices.

8

Laboratory 05

thomas mcandrew, david braun

Lehigh University

CONTENTS

\nobreak

8.0.1 Data Frame

We learned in lecture that a dataframe is a specific way to organize data points sampled from a sample space $\mathcal{G}$. A dataframe supposes a dataset $\mathcal{D} = (d_1, d_2, \cdots, d_N)$ should be organized so that each row is a single outcome d_i and each column is a position in the tuple $d_i = (x, y, z, ...)$

The dataframe is a common way computational scientists think about data.

8.0.1.1 How to read a CSV file into a data frame

The function `read.csv` takes as an argument a string that indicates a file on your local computer OR a URL online. Below the `read.csv` function is used to import into memory a **dataframe** related to bad drivers from the statsitcal news outlet called *FiveThirtyEight*.

The news article is here https://fivethirtyeight.com/features/which-state-has-the-worst-drivers/

[23]:
```
d = read.csv("https://raw.githubusercontent.com/
 ↪fivethirtyeight/data/master/bad-drivers/bad-drivers.csv")
d # Jupyter will by default print any variable
```

State	Number.of.drivers.involved.in.fatal.collisions.per.billion.miles
<chr>	<dbl>
Alabama	18.8
Alaska	18.1
Arizona	18.6
Arkansas	22.4
California	12.0
Colorado	13.6
Connecticut	10.8
Delaware	16.2
District of Columbia	5.9
Florida	17.9
Georgia	15.6
Hawaii	17.5
Idaho	15.3
Illinois	12.8
Indiana	14.5
Iowa	15.7
Kansas	17.8
Kentucky	21.4
Louisiana	20.5
Maine	15.1
Maryland	12.5
Massachusetts	8.2
Michigan	14.1
Minnesota	9.6
Mississippi	17.6
Missouri	16.1
Montana	21.4
Nebraska	14.9
Nevada	14.7
New Hampshire	11.6
New Jersey	11.2
New Mexico	18.4
New York	12.3
North Carolina	16.8
North Dakota	23.9
Ohio	14.1
Oklahoma	19.9
Oregon	12.8
Pennsylvania	18.2
Rhode Island	11.1
South Carolina	23.9
South Dakota	19.4
Tennessee	19.5
Texas	19.4
Utah	11.3
Vermont	13.6
Virginia	12.7
Washington	10.6
West Virginia	23.8
Wisconsin	13.8
Wyoming	17.4

A data.frame: 51 × 8

8.0.1.2 Selecting a column and the $ operator

With a dataframe you can select rows by asking R to return the number column where the first column in the dataframe is 1, the second column 2, and so on. You can also ask R to return a column of your dataframe by column name.

The 3rd column of the dataframe that we read records, by state, the percentage of fatal collision that involved a driver who was impaired by alcohol.

We can ask R for the 3rd column by using square brackets in the same way that we used square brackets to select rows and columns of matrices.

[24]:
```
d[,3]
```

```
1. 39 2. 41 3. 35 4. 18 5. 35 6. 37 7. 46 8. 38 9. 34 10. 21 11. 19 12. 54 13. 36 14. 36
15. 25 16. 17 17. 27 18. 19 19. 35 20. 38 21. 34 22. 23 23. 24 24. 23 25. 15 26. 43
27. 39 28. 13 29. 37 30. 35 31. 16 32. 19 33. 32 34. 39 35. 23 36. 28 37. 32 38. 33
39. 50 40. 34 41. 38 42. 31 43. 21 44. 40 45. 43 46. 30 47. 19 48. 42 49. 34 50. 36
51. 42
```

We can request a column from a dataframe by asking for 1. The dataframe 2. square brackets Inside the square brackets you divide the rows you want selected and the columns you want selected by a comma.

For example, if we wanted to look at the 10th row and 3rd column we can write

[25]:
```
d[10,3]
```

21

If instead we wanted to view rows 10,11,12 and columns 3 and 4 we could write

[26]:
```
d[ 10:12, c(3,4)  ]
```

A data.frame: 3 × 2

	Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.Speeding <int>	Percenta <int>
10	21	29
11	19	25
12	54	41

The above examples select columns and rows by number. One advantage of a dataframe is that we can select columns **by name**.

For example, we may be interested in rows 10-12 of the column "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.Speeding"

[27]:
```
d[10:12,"Percentage.Of.Drivers.Involved.In.Fatal.Collisions.
↪Who.Were.Speeding"]
```

1. 21 2. 19 3. 54

We can ask for **all** rows that correspond to a column by leaving the entry to the left of the comma blank.

[28]:
```
d[,"Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
↪Were.Speeding"]
```

1. 39 2. 41 3. 35 4. 18 5. 35 6. 37 7. 46 8. 38 9. 34 10. 21 11. 19 12. 54 13. 36 14. 36 15. 25 16. 17 17. 27 18. 19 19. 35 20. 38 21. 34 22. 23 23. 24 24. 23 25. 15 26. 43 27. 39 28. 13 29. 37 30. 35 31. 16 32. 19 33. 32 34. 39 35. 23 36. 28 37. 32 38. 33 39. 50 40. 34 41. 38 42. 31 43. 21 44. 40 45. 43 46. 30 47. 19 48. 42 49. 34 50. 36 51. 42

Finally, there is a shorthand for the code `d[,"Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.Speeding"]` using the dollarsign operator. The dollar sign operator can be thought of as a function that takes a column name as input and returns the column of data inside your data frame corresponding to that column.

[41]:
```
d$Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.
↪Speeding
```

```
NULL
```

8.0.1.3 Logical indexing

We are allowed to use logical indexing like we learned when selecting items in vectors and matrices to select rows and columns of a data frame. For example, suppose we are interested in **all** columns of our data frame where the Percentage Of Drivers Involved In Fatal Collisions Who Were Speeding is above 40%.

[30]:
```
d[ d$ "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
↪Were.Speeding" > 40, ]
```

A data.frame: 8 × 8

	State	Number.of.drivers.involved.in.fatal.collisions.per.billion.miles	P
	<chr>	<dbl>	<
2	Alaska	18.1	41
7	Connecticut	10.8	46
12	Hawaii	17.5	54
26	Missouri	16.1	43
39	Pennsylvania	18.2	50
45	Utah	11.3	43
48	Washington	10.6	42
51	Wyoming	17.4	42

or maybe we are interested in not all columns, but just the states where this percentage is above 40%. We can subset to a specific column by including in square brackets the name of the column

```
[34]: d[ d$ "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
        ↪Were.Speeding" > 40, "State" ]
```

1. 'Alaska' 2. 'Connecticut' 3. 'Hawaii' 4. 'Missouri' 5. 'Pennsylvania' 6. 'Utah' 7. 'Washington' 8. 'Wyoming'

If we wanted to include more than one column then we can include a vector that contains each column we want to select.

```
[36]: d[ d$"Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
        ↪Were.Speeding" > 40, c("State","Car.Insurance.Premiums....
        ↪") ]
```

A data.frame: 8 × 2

	State	Car.Insurance.Premiums....
	<chr>	<dbl>
2	Alaska	1053.48
7	Connecticut	1068.73
12	Hawaii	861.18
26	Missouri	790.32
39	Pennsylvania	905.99
45	Utah	809.38
48	Washington	890.03
51	Wyoming	791.14

8.0.1.4 Functions for data frames

There are several useful functions that take as an argument a data frame. The `nrow` function takes a data frame as an argument and returns the number of rows (i.e. the number of data points) in the data frame. The `ncol` function takes a data frame as an argument and returns the number of columns in the data frame.

[31]:
```
number_of_rows = nrow(d)
number_of_columns = ncol(d)

print(number_of_rows)
print(number_of_columns)
```

```
[1] 51
[1] 8
```

The colnames function takes as an argument a data frame and returns a vector that contains the name of each column in the data frame.

[32]:
```
column_names = colnames(d)
print(column_names)
```

```
[1] "State"
[2] "Number.of.drivers.involved.in.fatal.collisions.per.billion.
 ↪miles"
[3] "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
 ↪Were.Speeding"
[4]
"Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.
 ↪Alcohol.Impaired"
[5] "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.
 ↪Were.Not.Distracted"
[6] "Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Had.
 ↪Not.Been.Involve
d.In.Any.Previous.Accidents"
[7] "Car.Insurance.Premiums..."
[8]
"Losses.incurred.by.insurance.companies.for.collisions.per.
 ↪insured.driver..."
```

The summary function is a function that takes as an argument a data frame and returns, for each column in the data frame, the minimum value, maximum value, mean (average), median, 25th percentile (called the 1st quartile), and the 75th percentile (called the 3rd quartile).

[33]:
```
summary(d)
```

```
    State
 Length:51
 Class :character
 Mode  :character
```

```
Number.of.drivers.involved.in.fatal.collisions.per.billion.
↪miles
Min.   : 5.90
1st Qu.:12.75
Median :15.60
Mean   :15.79
3rd Qu.:18.50
Max.   :23.90
Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.
↪Speeding
Min.   :13.00
1st Qu.:23.00
Median :34.00
Mean   :31.73
3rd Qu.:38.00
Max.   :54.00
Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.
↪Alcohol.Impaired
Min.   :16.00
1st Qu.:28.00
Median :30.00
Mean   :30.69
3rd Qu.:33.00
Max.   :44.00
Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Were.
↪Not.Distracted
Min.   : 10.00
1st Qu.: 83.00
Median : 88.00
Mean   : 85.92
3rd Qu.: 95.00
Max.   :100.00
Percentage.Of.Drivers.Involved.In.Fatal.Collisions.Who.Had.Not.
↪Been.Involved.In.Any.Previous.Accidents
Min.   : 76.00
1st Qu.: 83.50
Median : 88.00
Mean   : 88.73
3rd Qu.: 95.00
Max.   :100.00
Car.Insurance.Premiums...
Min.   : 642.0
1st Qu.: 768.4
Median : 859.0
```

```
Mean   : 887.0
3rd Qu.:1007.9
Max.   :1301.5
Losses.incurred.by.insurance.companies.for.collisions.per.
↪insured.driver...
Min.   : 82.75
1st Qu.:114.64
Median :136.05
Mean   :134.49
3rd Qu.:151.87
Max.   :194.78
```

8.0.2 Assignment

We are going to look at a dataset that was collected on University "Fight Songs" that are played during sporting events. The article about the data set is here= https://projects.fivethirtyeight.com/college-fight-song-lyrics/

1. The data is at the URL (https://raw.githubusercontent.com/fivethirtyeight/data/master/fight-songs/fight-songs.csv). Please read in this raw CSV as a data.frame
2. Use the nrow and ncol function to describe the number of fight songs sampled and the number of different characteristics collected about each song.
3. How many songs were sampled with a BPM (beats per minute) above 150?
4. Select the rows where BPM is greater than 150 and select the column "nonsense" (Whether or not the song uses nonsense syllables (e.g. "Whoo-Rah" or "Hooperay"))
5. Summarize the data frame. Why do you think the summary function does not produce useful information for many of the columns?

Bibliography