Introduction to Business Statistics: A Modeling Approach

Viswa Viswanathan

2025-12-04

Table of contents

Introduction to Business Statistics

A Modeling and Regression-Based Approach

This text presents a modren, modeling-first approach to business statistics. Rather than a traditional probability-first sequence, we begin with data, relationships, patterns, models, and regression — the tools most relevant to business decision-making in marketing, finance, economics, HR, and analytics.

You will learn:

- how to visualize data
- how to describe relationships
- how to build regression models
- how to interpret uncertainty
- how to make data-driven decisions

Let's get started.

Preface

This book accompanies the course *Introduction to Business Statistics (BQUA2811)* at Stillman School of Business. It takes inspiration from *Lessons in Statistical Thinking* and builds on two core teaching principles:

1. **Students benefit from understanding structure before encountering uncertainty.**
2. **A modeling-centric approach helps to provide an overarchinthe courseg focus for**
3. **Courses that depend on BQUA2811 expect students to come in with competency in linear regression.**

The book is organized into 12 modules reflecting this storyline.

Part I

Data, Visualization, and Patterns

Module 1: Data, Visualization & Patterns

This module introduces the basic building blocks of statistical thinking: data frames, variables, point plots and relationships. We begin with visualization because patterns reveal what models will later formalize.

Learning Outcomes

- Describe data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

1 Data Frames and Variables

This chapter introduces **data frame** the type of R object in which we store data.

1.1 Learning outcomes

After working though this chapter, you will be able to:

- explain what a data frame is
- explain what a column and row of a data frame represent
- name the two types of variables that we will deal wigth in this course
- distinguish between numeric and categorical variables and provide examples of each
- explain what the term *level* of a categorical variable represents and provide examples
- explain why a column is also called a *variable*
- explain why a row is called an *instance* or *specimen*
- use R to do the following:
  - load a data frame from a loaded package
  - load a data frame from a Rdata file (later section)
  - load a data frame from a csv file (later section)
  - look at the first few rows in a data frame
  - look at the last few rows in a data frame
  - retrieve only the rows that satisfy certain conditions
  - retrieve a subset of the columns of a data frame
  - find the number of rows and columns in a data frame
  - compute the average of a numeric column in a data frame
  - find out how many rows are there in a data frame for each level of a categorical variable

1.2 The R Software Package for Statistics

In this course, we will be using the R software package for all computations and visualizations. While R offers a great deal of functionality, we will be using a limited subset that suffices for the course topics. In particular, we will be using a specific R package that contains all the functions and data we need for the course. You should install R and RStudio using the

instructions provided in Chapter 31. That section also shows you how to use RStudio and how to load a package – specificaly the LSTbook package – into R.

1.3 R Data Frames

In R, we generally represent data in a simple tabular structure named *data frame* – a table with rows and columns. Table 1.1 shows an example of some initial rows from the Boston_marathon data frame contained in the LSTbook package.

Table 1.1: Boston Marathon data (first 10 rows).

year	name	country	time	sex	minutes
2022	Evans Chebet	Kenya	02:06:51	male	127
2021	Benson Kipruto	Kenya	02:09:51	male	130
2019	Lawrence Cherono	Kenya	02:07:57	male	128
2018	Yuki Kawauchi	Japan	02:15:58	male	136
2017	Geoffrey Kirui	Kenya	02:09:37	male	130
2016	Lemi Berhanu	Ethiopia	02:12:45	male	133
2015	Lelisa Desisa	Ethiopia	02:09:17	male	129
2014	Mebrahtom “Meb” Keflezighi	United States	02:08:37	male	129
2013	Lelisa Desisa	Ethiopia	02:10:22	male	130
2012	Wesley Korir	Kenya	02:12:40	male	133

We first define the term *data frame* and then explain some terms that the definitoion uses.

Data frame: *A structured dataset organized as rows and columns, where rows correspond to individual observations and columns correspond to variables, with each column containing values of the same type.*

A data frame contains data about some topic. Each row represents an occurence of the topic. For example, Table 1.1 contains data about the results of the Boston Marathon over the years. Each row represents the result for one year, for one gender. We also refer to a row as an *observation*, or as an *instance*, or in a medical or biology context, as a *specimen*.

Each column represents some attribute of interest about instances. In Table 1.1, we have columns to represent the year of the race, the name, county and sex of the winner and the finishing time.

In statistics, we generally refer to columns of a data frame as *variables*. We explain why in Section 1.4.

In the 1880’s, Francis Galton was developing ways to quantify the heritability of traits. As part of this work, he collected data on the heights of adult children and their parents. Table 1.2 shows the initial few rows of that data.

Table 1.2: Galton data (first 10 rows).

family	father	mother	sex	height	nkids
1	78.5	67.0	M	73.2	4
1	78.5	67.0	F	69.2	4
1	78.5	67.0	F	69.0	4
1	78.5	67.0	F	69.0	4
2	75.5	66.5	M	73.5	4
2	75.5	66.5	M	72.5	4
2	75.5	66.5	F	65.5	4
2	75.5	66.5	F	65.5	4
3	75.0	64.0	M	71.0	2
3	75.0	64.0	F	68.0	2

In Table 1.2, each row represents an adult child. The variables in this data frame represent the father's height (*father*), mother's height (*mother*), sex of the adult child (*sex*), height of the adult child (*height*), and the number of children (*nkinds*) in the family. Each family also has a unique number for each family (*family*).

1.4 Variables

If we look at a column of a data frame, say the column *time* in Table 1.1, we see that each row can potentially have a different value in this column. More generally, the values in a column of a data frame can *vary* from row to row – no wonder we call columns of a data frame *variables*. Going forward, we will use the term *variable* to refer to a column of a data frame.

In this course, you will be working with many data frames. You should always be sure to understand what a row of a data frame contains information about. In the case of Table 1.1, each row rewpresents the results of one race. In the case of Table 1.2, each row represents one adult child. We use the term *unit of measurement* to refer to the object or concept about which each row stores information.

1.5 Numerical and Categorical Variables

In Table 1.1, the variable *minutes* has values that are numbers. We can such variables *numerical* variables. On the other hand, the variable *sex* can take on only the values *male* and *female* which are not numeric. Similarly, the variable *country* can also take on non-numetic values. We call such variables – those that take on non-numerical values *categorical*. The individual values of a particular categorical variable are called *levels*. Thus the values *mal* and *female* are the *levels* of the variable *sex* and the country names like *Kenya*, *Japan* and *Ethiopia* are the levels of the variable *country*.

1.6 Data Frames in R

In this section we look at the practical side of data frames – using them in R. We will only look at using data frames that are built in to the LSTbook package and mecome available as soon as we load the package LSTbook. If you have not already done so, you should install R and RStudio and learn how o use RStudio using the instructions from Chapter 31.

1.6.1 Viewing the first few rows of a data frame

In an RStudio command prompt you can enter the name of the data frame to get a preview of its contents.

```
Boston_marathon
```

```
# A tibble: 175 x 6
    year name        country time      sex    minutes
   <dbl> <chr>       <chr>   <time>    <chr>    <dbl>
 1  2022 "Evans Ch~ Kenya   02:06:51  male      127.
 2  2021 "Benson K~ Kenya   02:09:51  male      130.
 3  2019 "Lawrence~ Kenya   02:07:57  male      128.
 4  2018 "Yuki Kaw~ Japan   02:15:58  male      136.
 5  2017 "Geoffrey~ Kenya   02:09:37  male      130.
 6  2016 "Lemi Ber~ Ethiop~ 02:12:45  male      133.
 7  2015 "Lelisa D~ Ethiop~ 02:09:17  male      129.
 8  2014 "Mebrahto~ United~ 02:08:37  male      129.
 9  2013 "Lelisa D~ Ethiop~ 02:10:22  male      130.
10  2012 "Wesley K~ Kenya   02:12:40  male      133.
# i 165 more rows
```

Let us understand the output above. The first row says:

```
# A tibble: 175 Œ 6
```

This says that the data frame (also called *tibble*) has 175 rows and 6 columns.

The second line mentiones the column (variable) names. The third line has sone specific notation within angle brackets below each variable. This is telling us what type of value is stored in each column. - means that the value in a column is a *double*, that is, number that could potentially have a fractional part. - means that the column has character or non-numeric values - tells us that the column stores a time - and so on.

If the data frame has many columns and all of them will not fit in our output width, it will only display the columns that will fit and provide information below about the additional columns. In the present case all columns do fit and so there is not additional information provided.

1.6.2 Functions

You are likely familiar with the term *functions* as used in mathematics – you rpovide an input to the function and it performs some computation and provides an output. For example, we might have a function that takes a number as input and produces its square as output. Given the number 5 as input, it will produce the number 25 as its output. Given the number 1.5 as input, it will produce its square 2.25 as its output.

Pictorially we might represent a function as a box that has one or more inputs and produces a single output. (We can also have functions that take no inputs and produce an output, but we will not worry about them now.) See Figure 1.1.

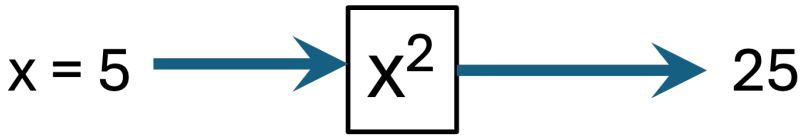

Figure 1.1: A function that squares its input

We might have functions that take more than one input. For example, a function might take two inputs x and y, and produce 23 + 3x + 4y as output. Figure 1.2 shows a function with two inputs.

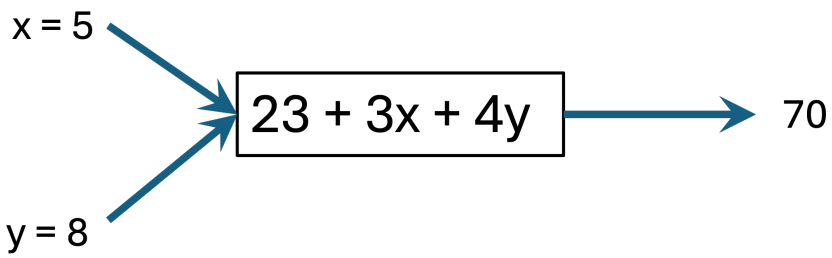

Figure 1.2: A function that computes 23+3x+4y where x and y are its inputs

In computing, we commonly refer to a function's inputs as its *arguments*. Figure 1.3 illustrates this.

The functions we considered in our examples thus far in Figure 1.1, Figure 1.2, and Figure 1.3 have been so simple that we were able to show the actual computation in the box representing the function. Most functions we use in real-life are more complex and we give them evocative names. In such cases, it makes more sense to place the name of the function in the box. Figure 1.4 shows a generirc repesentation of a function named *cost* with three arguments named *arg1*, *arg2*, and *arg3*.

When we use a function to perform some computation, we are said to *invoke* the function. We generally invoke a function by using its name and by supplying the argument values in

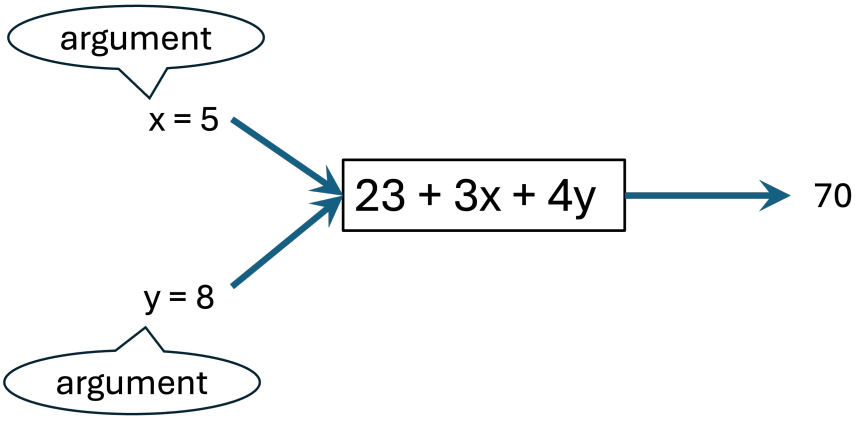

Figure 1.3: We generally refer to a function's inputs as its *arguments* – this function has two arguments, x and y

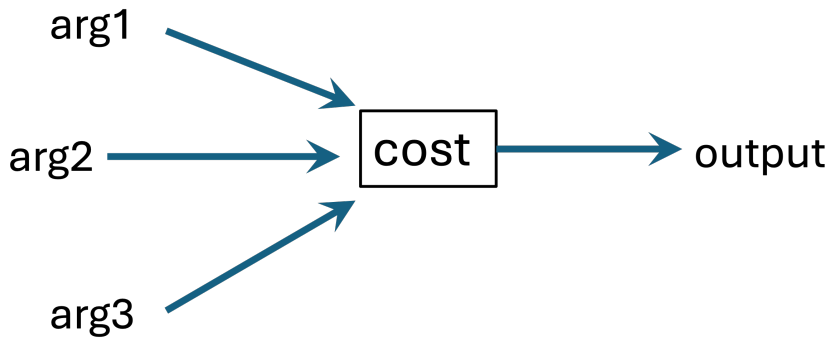

Figure 1.4: Generic representation of a function named *cost* that has three arguments

parentheses, as Figure 1.5 shows. We also use the term *passing arguments* to functions to refer to the act of supplying the inputs to a function. We also refer to the result of a function as its *return value*. We will also speak of a function as *returning* something. Figure 1.5 also shows that a function argumnent can be numeric or non-numeric.

```
total = cost(20, 32, "standard")
```

Figure 1.5: Invoking a named function and storing the function's return value in a variable named *total*

In the next section, we look at using functions to perform computing in R. At that time we will look at a different way of passing arguments to functions.

Almost all forms of computing, including statistical computing, make significant use of functions. In this course, many of the functions we use take data frame as input and produce a data frame as output. Let us now look at how we use functions in R to work with data and to perform statistical computations.

1.7 Using R functions

We will now look at some common R functions that we will use repeatedly in this course.

1.7.1 *nrow*

We will consider the *nrow()* function that allows us to find out how many rows a data frame contains. We pass a data frame as argument to the *nrow()* function and it returns a number specifying how many rows the argument data frame has. For example, to find out how many rows the Boston_marathon data frame has, we can use the following code.

```
nrow(Boston_marathon)
```

```
[1] 175
```

In the code above, we used the style of function invocation that Figure 1.5 introduced.

1.7.2 The *pipe* operator

In this course we also use another method of passing an argument to a function – via *pipes*. Let us first see the code and then learn how it works. We will invoke the *nrow()* function again, but using a different approach. The code below does exactly what the earlier code accomplished – finding the number of rows in a data frame.

```
Boston_marathon |> nrow()

[1] 175
```

In the code above, we are still passing the Boston_marathon data frame as an argument to the *nrow()* function, but we are passing it along a *pipe* which appears in the code as "|>".

Figure 1.6 clarifies code.

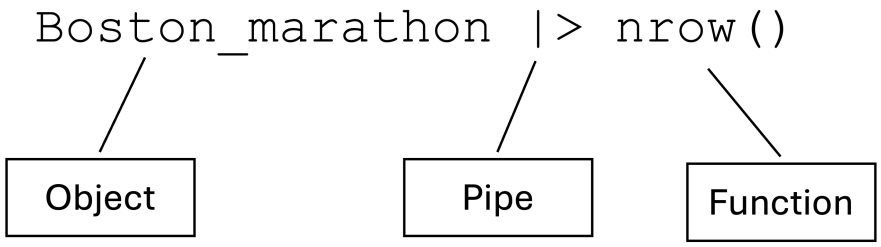

Figure 1.6: Using the pipe operator to supply the Boston_marathon data frame as an argument to the norw function

The pipe operator "|>" has the effect of *injecting* whatever appears on its left hand side as an argument to the function on the right hand side. Figure 1.7 makes this very explicit by literally depicting the pipe operator "|>" as a physical pipe.

Figure 1.7: Seeing the pipeoprtator literally as a pipe though which the argument flows into a function

Going forward, we will use this pipe approach extensively, but also mix in the earlier approach in places. You will catch on to our patters pretty easily.

1.7.3 *ncol*

We can use the *ncol()* function to find the number of columns in a data frame.

```
Boston_marathon |> ncol()
```

```
[1] 6
```

The above shows us that the Boston_marathon data frame has 6 columns or variables.

1.7.4 *head*

You can use the *head()* function to preview the first several rows of a data frame. This time we will use the *Births2022* data frame from the LSTbook package.

```
Births2022 |> head()
```

```
# A tibble: 6 x 38
  month dow   place  paternity meduc feduc married
  <chr> <ord> <chr>  <chr>     <ord> <ord> <chr>
1 05    Sat   hospi~ <NA>      Assoc HS    NA
2 09    Mon   hospi~ <NA>      <NA>  <NA>  NA
3 06    Thurs hospi~ N         HS    <NA>  not_ma~
4 05    Thurs hospi~ Y         Mast~ Bach~ married
5 08    Thurs hospi~ <NA>      HS    HS    NA
6 01    Tues  hospi~ Y         HS    <NA>  not_ma~
# i 31 more variables: fage <dbl>, mage <dbl>,
#   total_kids <int>, interval <ord>,
#   prenatal_start <ord>, prenatal_visits <dbl>,
#   mheight <dbl>, wt_pre <dbl>,
#   wt_delivery <dbl>, diabetes_pre <chr>,
#   diabetes_gest <chr>, hbp_pre <chr>,
#   hbp_gest <chr>, eclampsia <chr>, ...
```

The above previewed the first six rows of *Births2022* by default because we did not specify how many rows we wanted. We can specify that explicitly. The following previews the first 10 rows.

```
Births2022 |> head(10)
```

```
# A tibble: 10 x 38
   month dow   place  paternity meduc feduc married
   <chr> <ord> <chr>  <chr>     <ord> <ord> <chr>
 1 05    Sat   hosp~  <NA>      Assoc HS    NA
 2 09    Mon   hosp~  <NA>      <NA>  <NA>  NA
 3 06    Thurs hosp~  N         HS    <NA>  not_ma~
 4 05    Thurs hosp~  Y         Mast~ Bach~ married
 5 08    Thurs hosp~  <NA>      HS    HS    NA
 6 01    Tues  hosp~  Y         HS    <NA>  not_ma~
 7 07    Mon   hosp~  <NA>      <NA>  <NA>  NA
```

```
 8 11    Thurs hosp~ <NA>       Bach~ Bach~ NA
 9 03    Sat   hosp~ <NA>       HS+   <NA>  NA
10 08    Fri   hosp~ <NA>       Bach~ Bach~ NA
# i 31 more variables: fage <dbl>, mage <dbl>,
#   total_kids <int>, interval <ord>,
#   prenatal_start <ord>, prenatal_visits <dbl>,
#   mheight <dbl>, wt_pre <dbl>,
#   wt_delivery <dbl>, diabetes_pre <chr>,
#   diabetes_gest <chr>, hbp_pre <chr>,
#   hbp_gest <chr>, eclampsia <chr>, ...
```

1.7.5 *names*

We use this to find the names of the columns in a data frame.

```
Births2022 |> names()
```

```
 [1] "month"           "dow"
 [3] "place"           "paternity"
 [5] "meduc"           "feduc"
 [7] "married"         "fage"
 [9] "mage"            "total_kids"
[11] "interval"        "prenatal_start"
[13] "prenatal_visits" "mheight"
[15] "wt_pre"          "wt_delivery"
[17] "diabetes_pre"    "diabetes_gest"
[19] "hbp_pre"         "hbp_gest"
[21] "eclampsia"       "induction"
[23] "augmentation"    "anesthesia"
[25] "presentation"    "method"
[27] "trial_at_labor"  "attendant"
[29] "payment"         "apgar5"
[31] "apgar10"         "plurality"
[33] "sex"             "duration"
[35] "menses"          "weight"
[37] "living"          "breastfed"
```

1.7.6 *summarize*

We often calculate summaries (like the average of a variable) of the data contained in a data frame. We will discuss other examples of summaries – like variance, standard deviation, covariance, and correlation coefficient – at appropriate points in the course.

The code below shows how to compute the average of the *time* variable in the *Boston_marathon* data frame.

```
Boston_marathon |>
  summarize(avg_time = mean(time))

# A tibble: 1 x 1
  avg_time
  <drtn>
1 8635 secs
```

In the above code, we passed the Boston_marathon data frame via a pipe to the *summarize* function. Within the *summarize* function, we have used the R function *mean* to compute the average by passing the variable *time* as an argument to the *mean* function.

We chose to name the return value from the *mean* function (the average time) as *avg_time*. You do not necessarily need to name the summaries we compute in the *summarize* function, but we strongly recommend it.

For most of the summaries we compute we will use the general *summarize* function. WIthin the *summarize* function, we use specific functions depending on the kind of summary we are computing. In the previous example, we used the *mean* function for the average. Later we will see other functions to use within the *summarize* function.

1.7.7 *count*

The *count()* function comes in handy when we want to count how many occurrences of each distinct value in a column are present in a variable. For example, suppose we want to find how many rows are there for each sex we can do the following.

```
Boston_marathon |> count(sex)

# A tibble: 2 x 2
  sex        n
  <chr>  <int>
1 female    50
2 male     125
```

Similarly, if we wanted to find out how many times each country occurs, we can do this.

```
Boston_marathon |> count(country)

# A tibble: 25 x 2
   country              n
   <chr>            <int>
 1 Australia            1
 2 Belgium              2
```

```
 3 Canada                17
 4 Colombia               1
 5 Comm. Ind. States      1
 6 England                3
 7 Ethiopia              14
 8 Finland                7
 9 Germany                6
10 Greece                 2
# i 15 more rows
```

As you can see, the *count()* function comes in most handy when a variable is categorical. It also works for numerical variables, but will apply much more rarely in these cases.

1.8 Missing values (*NA*s)

Suppose the HR department of a company has a data frame of information about employees with variables (columns) *employee_id*, *first_name*, *last_name*, *base_salary*, *phone_no*, and *base_office*. It is not essential that we have data on every variabkle for every row. For example, it could be the case that the company does not have a *phone_no* for some employees and does not have a *base_office* for some. These are called *missing values*. A missing value denotes somnething whose value *we do not know*. A missing value is not the same as a blank character value or a zero numeric value. Blanks and zero are valid and known values for variables. A *missing value is different – we do not know the value*.

Table 1.3 shows a data frame with no missing values. In each row, we see values for every variable.

Table 1.3: A data frame with no missing values – in every row, every variable has a value

channel	advertising_spend_k	weekly_sales_k
Search Ads	28.1	152
Search Ads	30.6	152
Search Ads	93.4	369
Search Ads	51.3	289
Retail Promo	54.6	175
Search Ads	86.6	342
Retail Promo	46.5	169
Retail Promo	41.4	142
Search Ads	78.1	322
Retail Promo	41.1	182

On the other hand Table 1.4 shows a small data frame with missing values.

Table 1.4: A data frame with missing values – the *NA*s in columns *phone_no* and *base_office* represent missing values

employee_id	first_name	last_name	base_salary	phone_no	base_office
10210	Betty	Chu	67000	+1 (777) 555-1212	New Jersey
24532	Jason	Fingleton	85000	+1 (732) 555-1212	NA
56437	Shim	Sung	76000	NA	New York
20320	Ravi	Shankar	100000	+91 81487 03210	Shankar
67582	Emily	Weitz	87000	NA	Weitz

1.9 Codebook

Before using a data frame, we need to understand it well. That means, at the very least, that we know the unit of observation, the meaning of each variable and the units of measurement where applicable. This information is in the codebook for a data frame. You can use the ? operator for this purpose as the example code below shows.

```
?Boston_marathon
```

This will show the codebook in the bottom right pane of RStudio.

2 Exploring Relationships through Scatterplots

The term *statistics* often conjures up thoughts of numerical computations. However, before we perform various co mputations, we often get a feel for the data and the underlying patterns by first visualizing the data. In this course we will use only a few standard plots to communicate essential statistical ideas.

In fact, use a single function – *point_plot* – to generate all of our plots. Although R offers numerous, sophisticated plotting features to plot almost anything we can imagine, we will keep things simple and focus only on what we need to support the concepts that our course covers.

2.1 Learning outcomes

After working though this chapter, you will be able to:

- given a scatterplot of two numeric variables with the plot's axes labeled explain precisely what determines the location of a given point
- given a scatterplot of two numeric variables, the data frame on which it is based, and a specific point on the plot, identify the corresponding row in the data frame
- given a scatterplot of two numeric variables, the data frame on which it is based, and a specific row in the data frame, identify the corresponding point on the plot
- given a scatterplot one numeric variable on the y-axis against a categorical variable on the x-axis, explain how many different values exist for the x-axis
- given a scatterplot one numeric variable on the y-axis against a categorical variable on the x-axis, explain the speific appearance of the points along the x-axis for each of the category levels
- describe a tilde-expression (a model expression)
- given a tilde-expression with one variable on eithert side of the tilde, state how they relate to the plot axes
- given a tidle expression with two variables on the right of the tilde, expalin the mnappings of the elements of the tidle expression to the elements of the plot

- given a tidle expression with two variablkes on the right of the tilde, expalin the mnappings of the elements of the tidle expression to the elements of the plot
- explain why the points are jittered along the x-axis when the x-axis has a categorical variable
- Given a scatterplot of a num eric variable on the y-axis and a caregorical variable on the y-axis, identify a pair of points that would have overlapped if not for jittering
- use the point_plot function to
  - generate a scatterplot of two numeric variables
  - generate a scatterplot of one numeric variable on the y-axis against a categorical variable on the x-axis
  - generate ascatterplot of numeric variables on both axes with the points colored based on a third variable
  - generate ascatterplot of numeric variables on both axes with the points colored based on a third variable and the plots split into facets based on a fourth variable

2.2 Plotting the relationship between two numerical variables

The *point_plot()* function that we will rely on for plots in this course lives in the *LSTbook* package. We will be using the *LSTextra*s package. Once we load the *LSTextras* package (which in turn loads the *LSTbook* package and all its dependencies), the *point_plot()* function becomes available for us to use.

Let us first look at the *price_demand* data frame that is built into the *LSTbook* package.

A hypothetical company has gathered data for a single product. Over time, and across very similar sales areas, the company has charged different prices for the same product, It has recorded the monthly sales for each price. The dataset *price_demand* contains the price and the corresponding sales quantities. We would like to study if the two variables *price* and *sales_qty* are related in any way.

First let us see some rows from the dataset.

Table 2.1: Price Demand data (first 15 rows)

price	sales_qty
11.8	1260
12.1	1215
13.6	1214
12.3	1298

12.4	1160
13.7	1138
12.6	1174
11.2	1220
11.7	1313
11.9	1283
13.3	1170
12.6	1247
12.6	1283
12.3	1263
11.8	1259

The full dataset has 500 rows. With so many rows, we cannot easily get a handle on the overall pattern by looking at the data directly. Visualizing the data might help us. We will do several things in this section: - generate the plot using R code - understand how each element of the plot is related to the original dataset - understand the various elements of the code so that you can use similar code to generate plots that you many need - interpret the plot

The code and the plot appear below. For now, ignore the code. We will discuss it in detail after analyzing the plot itself.

```
price_demand |>
  point_plot(sales_qty ~ price)
```

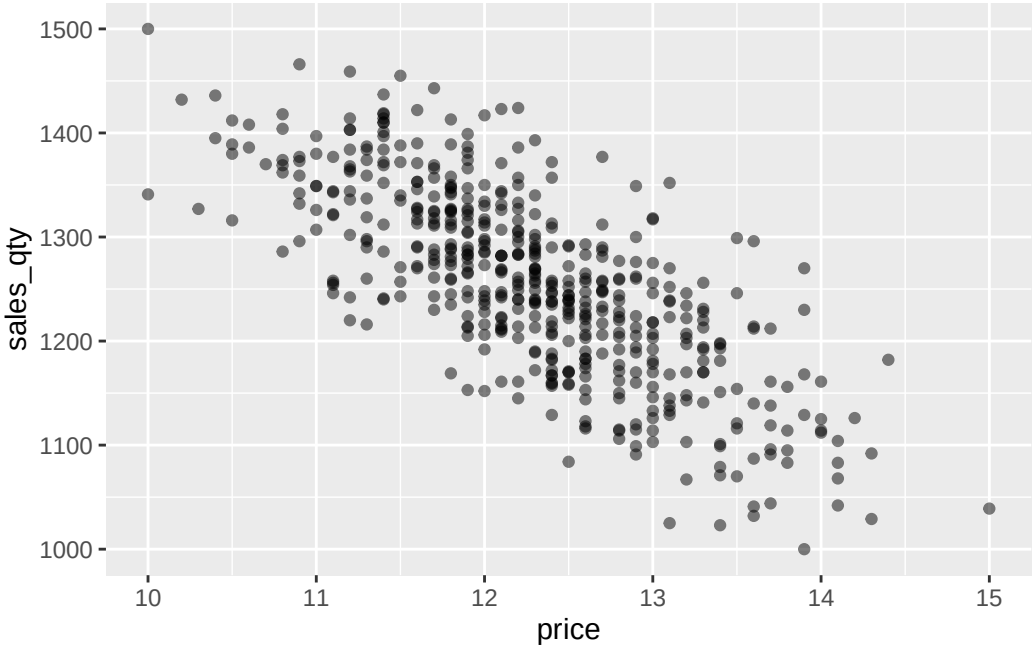

Figure 2.1: Price demand relationship

Figure 2.1 shows the relationship between *price* and *sales_qty* – the two variables in the data frame. Note the following about the plot:

- The plot has two axes – the x-axis along the width and the y-axis along tbe height.
- The plot shows thge values of price on the x-axis and the values of sales_qty on the y-axis.
- If you take any single point on the plot, you can drop a vertical line from the point to the x-axis and the point where it intersects the x-axis is the *price* corresponding to the point.
- Similarly, if you draw a horizontal line from the point to intersect the y-axis, you get the *sales_qty* corresponding to that point.
- Each point of the plot corresponds to one row of the dataset. So how many points are on the plot? 500, since the dataset has 500 rows.
- Thus given a point on the plot, we should be able to identify a corresponding row of the data frame
- Given a row of the data frame, we should be able to identify a corresponding point on the plot.

2.2.1 Finding the point corresponding to a row of the data frame

Let us now try to find the point on the plot corrresponding to the first row in Figure 2.1.

The first row of the data frame has *price = 11.8* and *sales_qty = 1260*. Therefore, we can find the point by drawing a vertical line from 11.8 on the x-axis to intersect a horizontal line from 1260 on the y-axis. Figure 2.2 illustrates this. We have highlighted the point by enlarging it.

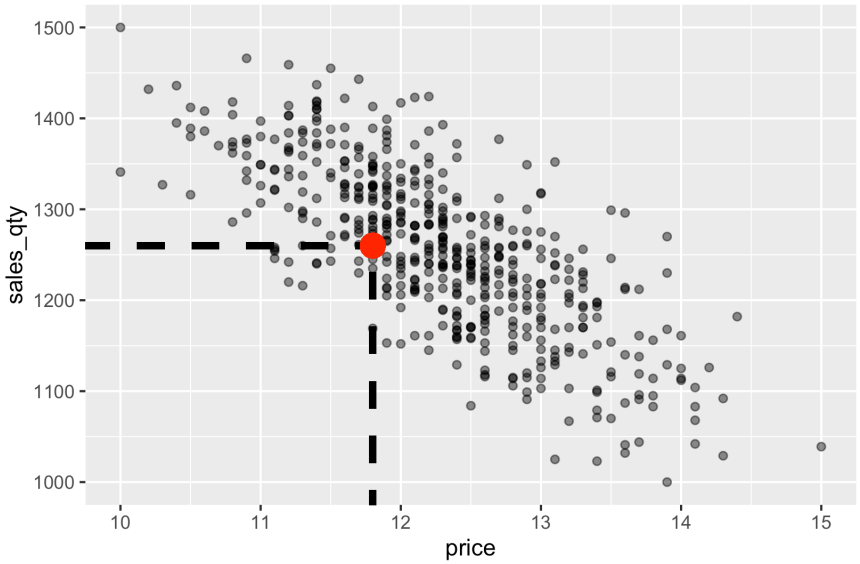

Figure 2.2: Finding the first row of the *price_demand* data frame on the plot: *price = 11.8*, *sales_qty = 1260*

Similarly, let us locate the point corresponding to row 11 (*price = 13.3*, *sales_qty = 1170*).

following the same approach we arrive at Figure 2.3.

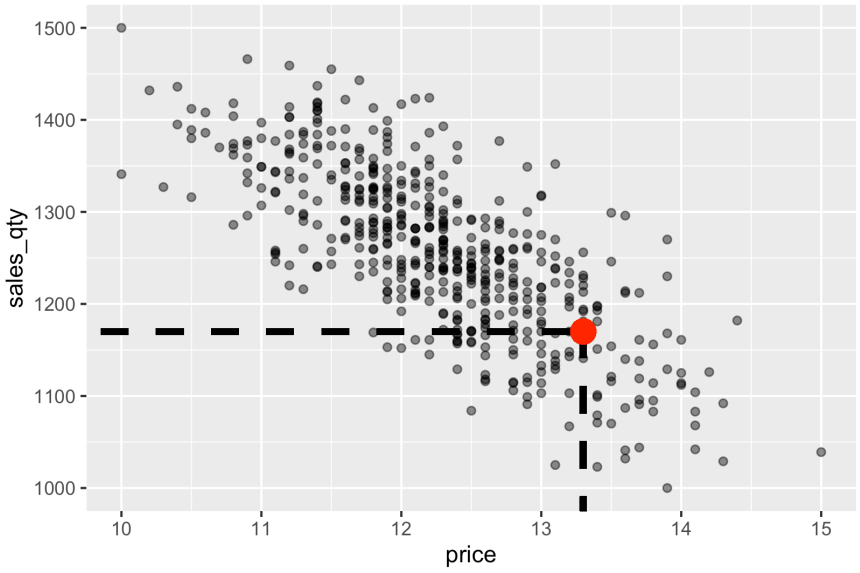

Figure 2.3: Finding the eleventh row of the *price_demand* data frame on the plot: *price = 13.3*, *sales_qty = 1170*

2.2.2 Finding the *price* and *sales_qty* corresponding to a point on the plot

We can reverse the above process to find the variable values (and hence the actual row) corresponding to a point on the plot. We will simply drop a vertical line from the point down to the x-axis to get the *price* and draw a horizontal line from the point to the y-axis to determine the *sales_qty*. The figure will look similar to the prior two figures.

2.2.3 Dissecting the code

Figure 2.4 decodes the various elements of the code that we used to generate out first point plot in Figure 2.1. We see that we pass the data frame *price_demand* to the *point_plot* function through the *pipe* operator. We add the tilde expression for the plot as an additional argument.

Figure 2.5 shows us that the *point_plot* function maps the variable on the left of the tilde operator to the y-axis of the plot and maps the variable to the right of the tilde expression to the x-axis of the plot.

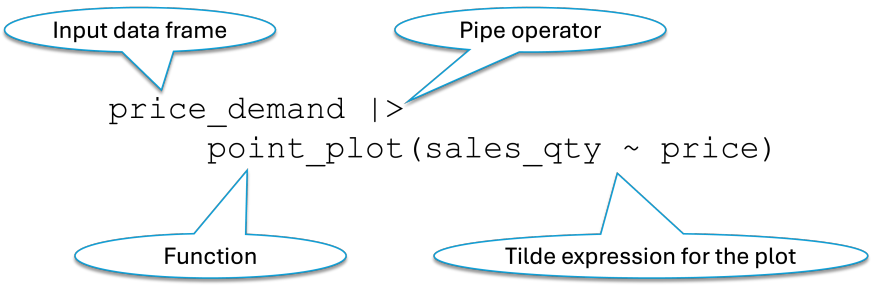

Figure 2.4: Passing the *price_demand* data frame as argument to the *point_plot* function through the pipe operator

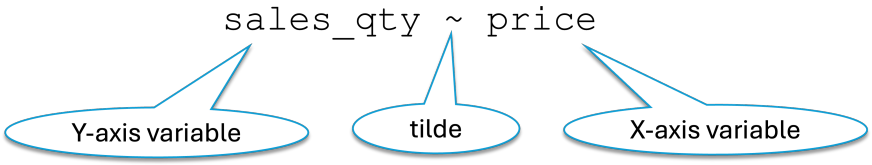

Figure 2.5: Using a tilde expression to map variables to the axes

2.2.4 Interpreting the plot

Figure 2.6 repeats the plot of *price* versus *sales_qty*.

```
price_demand |>
  point_plot(sales_qty ~ price)
```

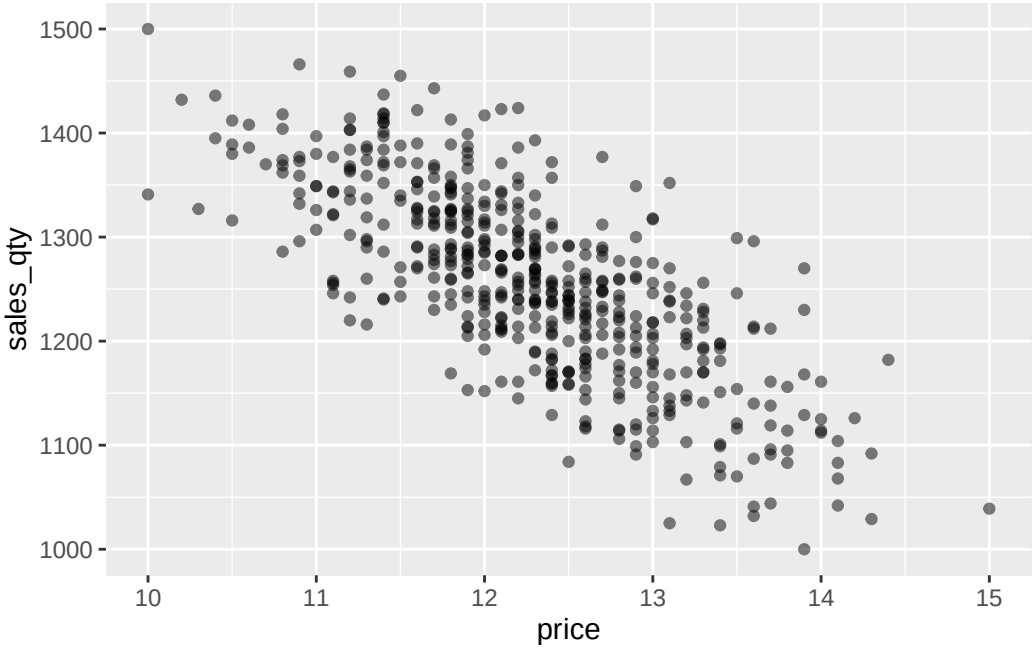

Figure 2.6: Price demand relationship (again)

We can see a clear trend showing that, in general, higher prices correspond to lower values of sales quantity.

To be sure, this is only a trend and not a strict rule. That is, given any two points, we do not have a guarantee that the point with the higher price will always have a lower sales quantity. However, this will be true for most of the pairs of points. Figure 2.7 illustrates this with some randomly selected pairs of points. We see that most of the lines point down and to the right – meaning that for most pairs of points, the one with higher *price* has a lower *sales_qty*. However, we also see a few examples where the opposite is true.

This is why we only calls this a *trend* and not a strict rule about the relationship between price and sales_qty.

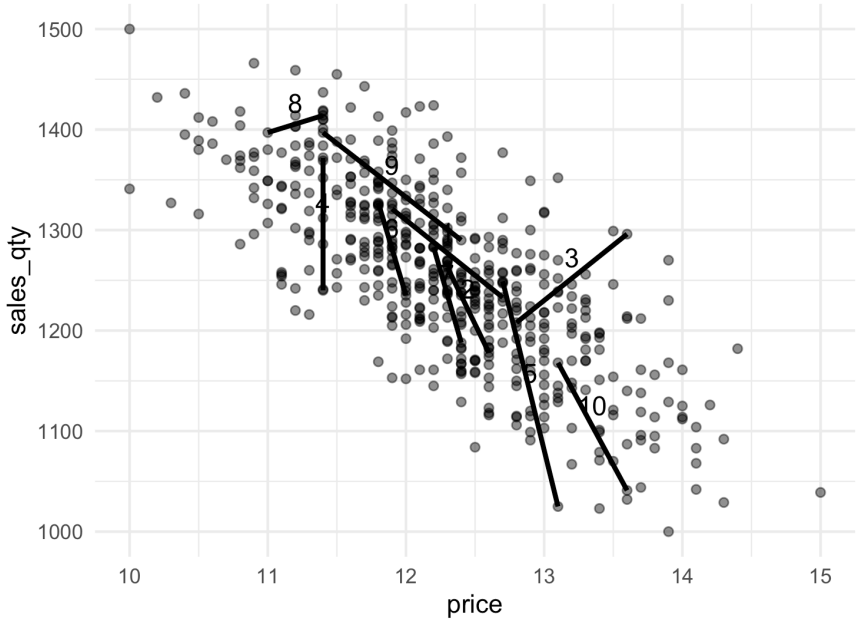

Figure 2.7: Line segments connecting some pairs of points showing that for most pairs of points the point with higher price has lower sales_qty

2.2.5 Positive and negative relationships

In general, when we discuss the relationship between two numerical variables, if the general trend is that higher values of one correspond to higher values of the other, then we have a positive relationship. On the other hand, is higher values of one variable correspond to lower values of the other then we have a negative relationship.

What kind of relationship does Figure 2.6 show – positive or negative?

You got it right if you said *negative* – higher values of *price* correspond to lower values of *sales_qty*.

Figure 2.8 shows two positively correlated variables. Here, higher values of one variable are generally related to higher values of the other. We used the *advertising_sales_channel* data frame for this plot.

```
advertising_sales_channel |>
  point_plot(weekly_sales_k ~ advertising_spend_k)
```

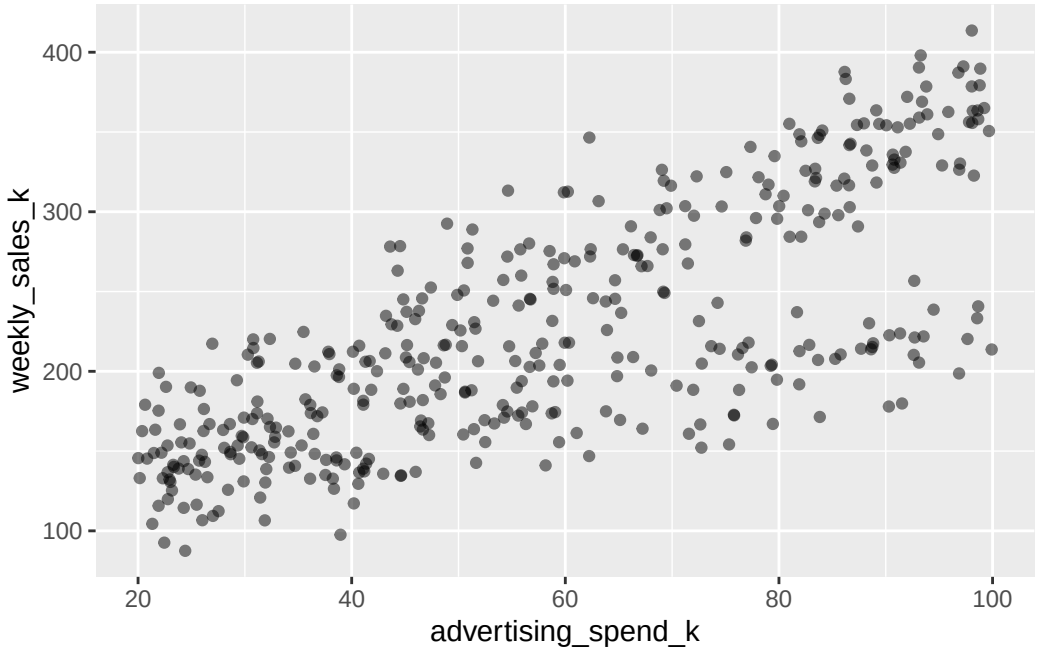

Figure 2.8: Weekly sales and advertising spend have a positive relationship – higher advertising generally has higher weekly sales

The data frame *returns_dpo* contains hypothetical data on various firms' daily stock returns and the number of days they take to pay their suppliers. Figure 2.9 shows the relationship between these variables. We cannot spot any trend. Higher or lower values of one variable do not indicate higher or lower values of the other. Hence there is neither a positive relationship nor a negative one.

```
returns_dpo |>
  point_plot(daily_return ~ dpo)
```

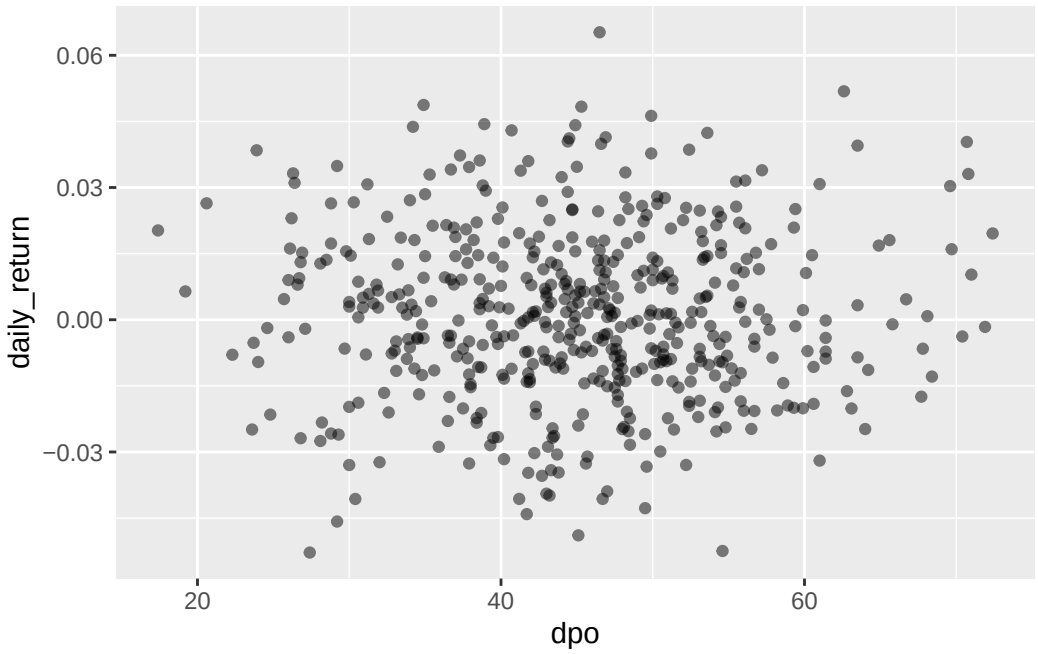

Figure 2.9: Days payment outstanding (dpo) does not have a relationship to the daily stock return

2.2.6 Putting a number on the relationship: The correlation coefficient

Thus far, we have been talking about relationships between variables. Exact sciences like mathematics and statistics like to be precise and assign numbers where possible. Statistics commonly uses the *correlation coefficient* to quantify the relationship between two numerical variables. This section just introduces the *correlation coefficient* and shows you how to compute it using R. We will get into the mechanics of the actual computation later in the book.

The correlation coefficient can take a value between -1 and +1. A value of 0 signifies a complete absence of relationship. -1 signifies a perfect negative correlation and +1 signifies a perfact positive correlation. We discuss these ideas below.

We look at some examples now. Revisiting the *price_demand* data frame, let us use R to compute the correlation coefficient between the variables *price* and *sales_qty*.

```
price_demand |>
  summarize(price_sales_qty_correlation = cor(price, sales_qty))
```

```
# A tibble: 1 x 1
  price_sales_qty_correlation
                        <dbl>
1                      -0.752
```

We are computing a summary and hence use the *summarize* function (see Section 1.7.6). Since the summary we are computing now is the correlation coefficient, we use the *cor*

function within the *summarize* function. We have chosen to call the comoputed result *price_sales_qty_correlation*. We could have named it anything we wanted, but chose the sensible approach of giving it a meaningul name.

Figure 2.1 had already shown us the negative relationship between *price* and *sales_qty*. The correlation confirms it, ... and makes it more precise.

Let us compute the correlation coefficient between advertising spend and the weekly sales from the *advertising_sales_channel* data frame. From Figure 2.8 we saw that these two variables are positively related. Let us see what the correlation coefficient says.

```
advertising_sales_channel |>
  summarize(ad_sales_corr = cor(advertising_spend_k, weekly_sales_k))
```

```
# A tibble: 1 x 1
  ad_sales_corr
          <dbl>
1         0.760
```

Sure enough, we see a positive correlation coefficient.

2.2.7 Perfect correlation

Consider Figure 2.10 showing strong, but imperfect, correlation.

```
price_demand |>
  point_plot(sales_qty ~ price)
```

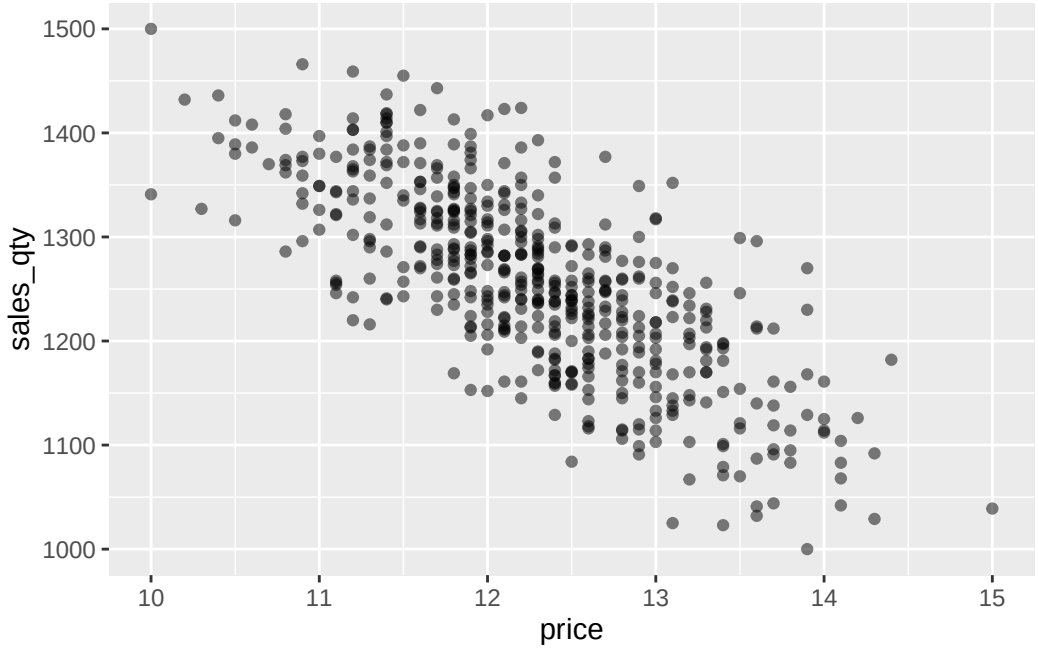

Figure 2.10: Price and sales_qty are not perfectly correlated – correlation coefficient is -0.75

We have already established that *price* and *sales_qty* are negatively related. That is, there is a general pattern that higher values of *price* have lower values of *sales_qty*.

In Figure 2.10, if we look only at the points corresponding to a particular value of *price*, say 13, the points along this vertical line range from approximately 1100 to 1320. That is, given a value of *price*, we cannot be sure about the exact value of the corresponding value of *sales_qty*. However, knowing *price* does give us some idea about *sale_qty*. This shows relationship, but not perfect correlation.

When the correlation coefficient is +1 or -1, we have perfect correlation. In this case, knowing the value of one variable enables us to precisely know the value of the other variable as well. This happens when all the points lie on a straight line instead of being dispersed as a cloud as in Figure 2.10.

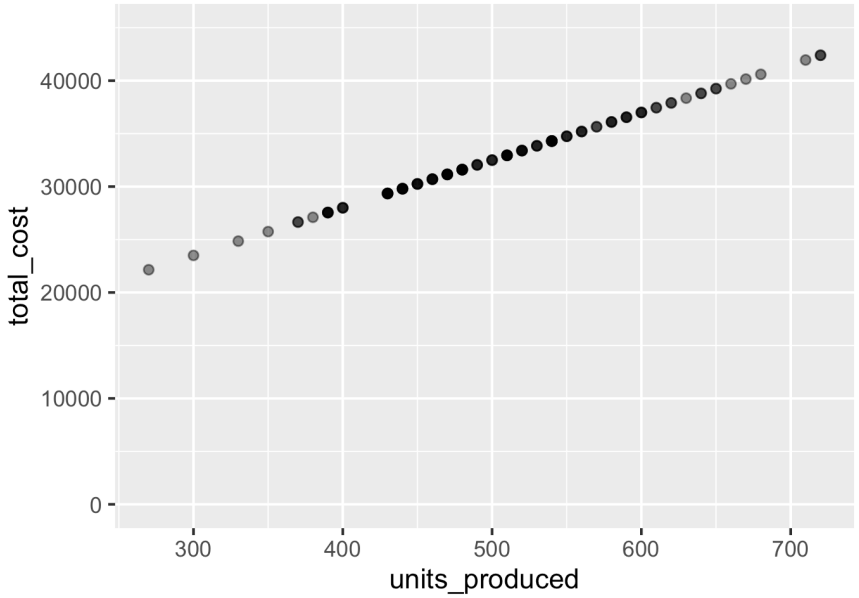

Figure 2.11: Perfect correlation arises when all the points lie perfectly on a straight line – the variables in this plot have a corelation coefficient of +1

Figure 2.11 shows a situation when all points lie on a perfect straight line. In this case, given a value for one variable, we can determine exactly the value of the other. We can determine it visually or by plugging the value of the known variable into the following equation and calculating the value of the other variable.

```
total_cost = 10000 + 45*units_produced
```

In Figure 2.11 the line slopes upward as it goes from left to right. WHen the line on which all points fall slopes down as it goes right, the correlation is still perfect, but the line slopes down as it goes to the right and so we have a correlation coefficien of -1.

2.2.8 How the point cloud looks for various correlation coefficients

Figure 2.12 shows the point plots for various values of correlation coefficients between -1 and +1.

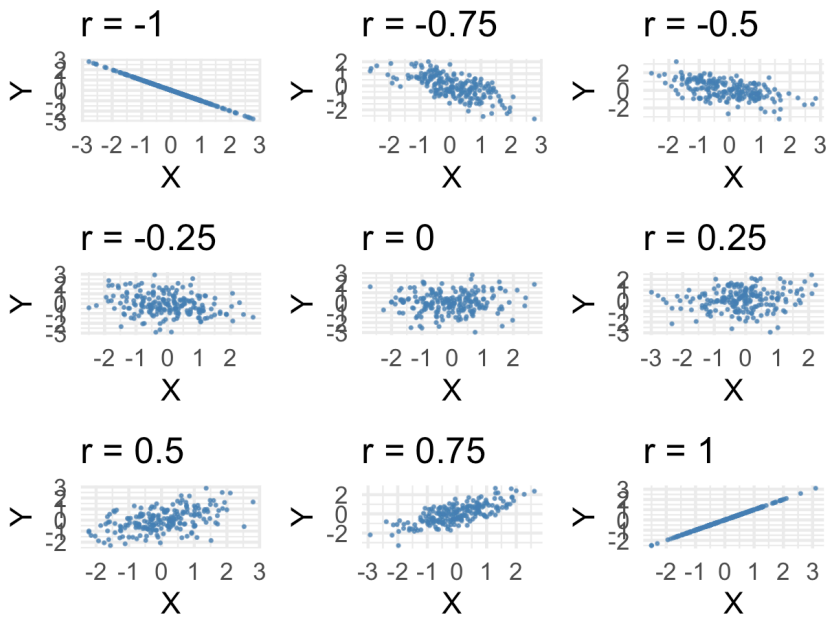

Figure 2.12: Point plots for various correlation-coefficients (r)

You can see the following: - in the plot wigth correlation coefficient = -1 (r = -1), the points align perfectly on a straight line *sloping downwards* as the x values increase - at +1 they align perfactly on a line *sloping upwards* as the x values increase - as the coefficient goes from -1 to 0, the points get more scattered away from perfect alignment on a straightline until at 0 there is no pattern whatsoever - as the correlation coefficient increases from 0 to +1, we see that the tendency to align on a straight line increases until perfect alignment at r = 1.

2.3 Plotting a numerical variable against a categorical one

In this course, we will only assign a numerical variable to the y-axis. Thus, if a plot has a categorical variable, it will only occupy the x-axis.

Numerical and categorical variables differ in how they behave in plots. In Figure 2.8 we have the variable *advertising_spend_k* on the x-axis. Being a numerical variable, it can potentially take on any value on the x-axis. On the other hand, look at Figure 2.13.

Here we have the variable *bank_account_type* on the x-axis and it can take on only one of two values *"Checking"* and *"Savings"*. Thios is why each of the rows in the data frame falls

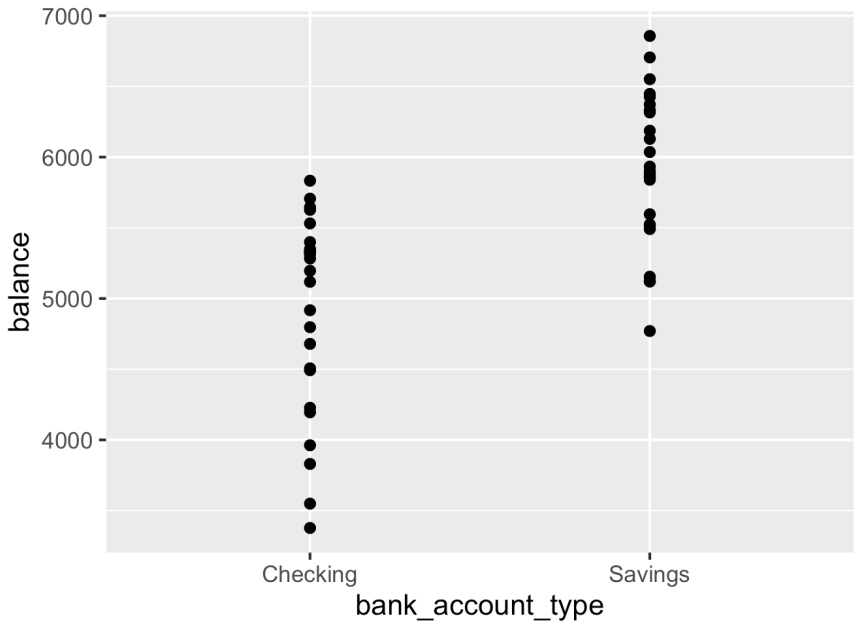

Figure 2.13: Plotting a categorical variable against a numerical one – using the data frame *acct_type_balance* (not plotted using the *point_plot* function)

along one of two perfectly vertical assortment of points If the row corresponds to a "Checking" account, it falls on the left set of points and on the right set otherwise.

2.3.1 *point_plot* and "jittering"

If we have two *Checking* accounts with the same or very similar balance, their points will overlap. Because of this overplotting, the plot in Figure 2.13 does not enable us to look at all the points.

To overcome this problem, the *point_plot* function plots this chart a bit differently. See Figure 2.14.

```
acct_type_balance |>
  point_plot(balance ~ bank_account_type)
```

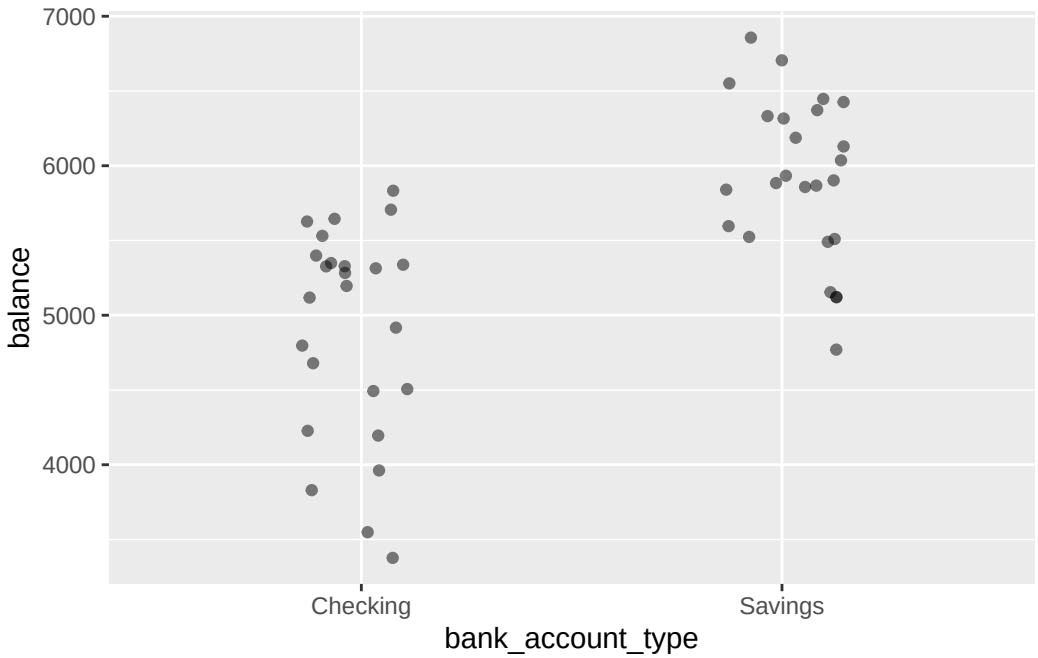

Figure 2.14: Plot of account type against account bakance using the *point_plot function*: it jitters the points along the x-axis to reduce overplotting

To enable us to see the points more clearly, the *point_plot* function randomly "jitters" the points along the x-axis to more clearly separate points that might be overplotted or close together.

2.4 Bringing a third variable into the plot

Two-dimensional plots can normally accommodate only two variables – one on each axis. However, we can often get more insights if we manage to get more variables into our plots. The *point_plot* functions allows us to bring one more dimension to our plots through color.

In Figure 2.8, we saw the positive relationship between *weekly_sales_k* and *advertising_spend_k* in the *advertising_sales_channel* data frame. That data frame has another variable *channel*. Each row corresponds to one of two sales channels – *Search Ads* and *Retail Promo*. We might want to study separately for each channel the relationship between advertising spend and the weekly sales.

```
advertising_sales_channel |>
  point_plot(weekly_sales_k ~ advertising_spend_k + channel)
```

```
Error in `fortify()`:
! `data` must be a <data.frame>, or an
  object coercible by `fortify()`, or a valid
  <data.frame>-like object coercible by
  `as.data.frame()`.
```

```
Caused by error in `check_data_frame_like()`:
! `colnames(data)` must return a
  <character> of length `ncol(data)`.
```

?@fig-advertising-sales-channel-color inserts, through color, the variable *channel* into the plot of *weekly_sales_k* against *advertising_spend_k* from the *advertising_sales_channel* data frame. From it we can clearly see that the points corresponding to each sales channel clearly shows the positive relationship that we saw earlier. But this plot also shows that for a given value of *advertising_spend_k* the *Retail Promo* channel generally has a comparatively higher *weekly_sales_k* value. Adding the variable *channel* through color enabled us to see more into the data than before.

2.4.1 Understanding the tilde expression to add a third variable

Figure 2.15 shows the tilde expression we used in a point plot of two variables to color the points based on a third variable.

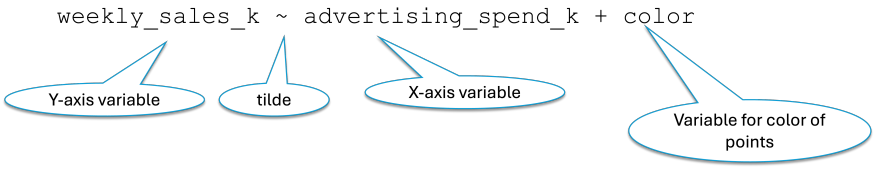

Figure 2.15: To color the points based on the value of a third variable, we just *add* it to the end of the tilde-expression after a plus sign

?@fig-color-tilde-expression-2 shows another example. In this, we use the *mpg* data frame which contains data on cars. We plot the city mileage of cars (variable *cty*) against their engine displacement (variable *displ*). We color each point based on the class of the vehicle (variable *class*).

```
mpg |>
  point_plot(cty ~ displ + class)
```

```
Error in `fortify()`:
! `data` must be a <data.frame>, or an
  object coercible by `fortify()`, or a valid
  <data.frame>-like object coercible by
  `as.data.frame()`.
Caused by error in `check_data_frame_like()`:
! `colnames(data)` must return a
  <character> of length `ncol(data)`.
```

Part II

Variation

Module 2: Variation

Variation plays a very important role in statistics. Some even go to the extent of viewing the discipline of statistics itself as the study of variation.

This module discusses variation generally by visualizing it through *violin* plots. It then introduces the key statistical concepts of *variance* and *standard deviation* and how to compute these using R.

Learning Goals

- a
- b
- c

Structure of This Module

- variation
- high and low variation
- visualizing variation using the violin plot
- measuring variation precisely using variance and standard deviation

3 Using Violin Plots to Visualize Distribution

We have already looks at the concept of *variable*. We have used the term to refer to a column of a data frame. We explained the rationale for the name from the fact that the values in a single column of a data frame *vary*. That is, not all the rows of a column have the same value. For example, we see in Table 3.1 that the values in the column *minutes* vary – that is, not all the values in the column are the same. The same is true of every column.

Table 3.1: Boston Marathon data (20 randonmly selected rows).

year	name	country	time	sex	minutes
1927	Clarence H. DeMar	United States	02:40:22	male	160
2008	Robert Kipkoech Cheruiyot	Kenya	02:07:46	male	128
2009	Salina Kosgei	Kenya	02:32:16	female	152
2014	Buzunesh Deba	Ethiopia	02:19:59	female	140
1915	Edouard Fabre	Canada	02:31:41	male	152
1936	Ellison M. Brown	United States	02:33:40	male	154
1918	Camp Devens Divisional Team	United States	02:24:53	male	145
1952	Doroteo Flores	Guatemala	02:31:53	male	152
1939	Ellison M. Brown	United States	02:28:51	male	149
1929	John C. Miles	Canada	02:33:08	male	153
1982	Alberto Salazar	United States	02:08:52	male	129
1997	Fatuma Roba	Ethiopia	02:26:23	female	146
1902	Sammy A. Mellor	United States	02:43:12	male	163
1926	John C. Miles	Canada	02:25:40	male	146
1974	Neil Cusack	Ireland	02:13:39	male	134
2017	Geoffrey Kirui	Kenya	02:09:37	male	130
1920	Peter Trivoulidas	Greece	02:29:31	male	150
1994	Uta Pippig	Germany	02:21:45	female	142
1898	Ronald J. MacDonald	Canada	02:42:00	male	162
1974	Michiko (Miki) Gorman	United States	02:47:11	female	167

Take care to note that we are not saying that every value in a column has to be distinct. Values can repeat. However we have variety in a column as long as all the values are not the same. Very rarely will you come across data frames in which all the values of a column are the same. In this case the column does not serve any useful purpose for data analysis.

In this course, we deal only with variability in numerical variables. Statistics deals with variability in categorical variables as well, but this book does not consider this in any depth.

3.1 Distribution

Before we look at the distribution of variables in a data frame, let us look at variability in sets of numbers to understand the idea of *distribution*. In this chapter we focus on visualization alone. The next chapter goes into measuring variation precisely.

3.1.1 Uniform distribution

Suppose we have 1000 numbers with each one being an integer between 1 and 100. Some of the numbers in this data might be 45, 23, 46, 89, 1, 23, 45, 56, and so on. If each number occurs exactly 10 times – that is we have 10 ones, 10 twos and so on up to 10 hundreds, we have what is called as the *uniform distrigution*. In a uniform distribution, each number in the range of the list occurs the same number of times.

Figure 3.1 shows a *histogram* of a set of numbers. A *histogram* breaks up the whole range of the data points into *bins*. In our example, the numbers range from 1 to 100. A histogram might break this up into some number of *bins*. If the number of bins is 10, then the numbers between 1 and 10 will fall unto 1 bin. The numbers between 11 and 20 into the next bin and so on. In this example, the bin-width is 10 because each bin spans a range of size 10. The x-axis of the histogram shows the number values and a bar for each bin. The y axis shows how many of the numbers fall into each bin and so the pattern formed by the heights of the bars helps us to see the *distribution* of the numbers.

Figure 3.1 shows numbers that are perfectly uniformly distributed. We have used a bin-width of 1 to have each individual integer fall into its own bin. In a *uniform distribution* every number in the overall range of the numbers occurs exactly the same number of times.

Our current example deals with integers, but in general when we have numerical variables, we will deal with numbers that also have a fractional part. Histograms work in the same way for those too, except that we cannot have one bin for each distinct value as there is potentially an inficite number of values in any range.

Figure 3.2 shows an example of a set of numbers distributed approximately uniformly. For example, Each number does not occur exactly the same number of times, but they are loosely similar. For example, the number 2 seems to appear around 13 times and the number 25 seems to occur around 18 times. In practice, uniformly distributed numbers will look more like Figure 3.2 than Figure 3.1.

A numerical variable does not necessarily have to be distributed uniformly. That is each number in the range does not have to occur an equal number of times exactly or approximately. We will now look at some other commmon possibilities.

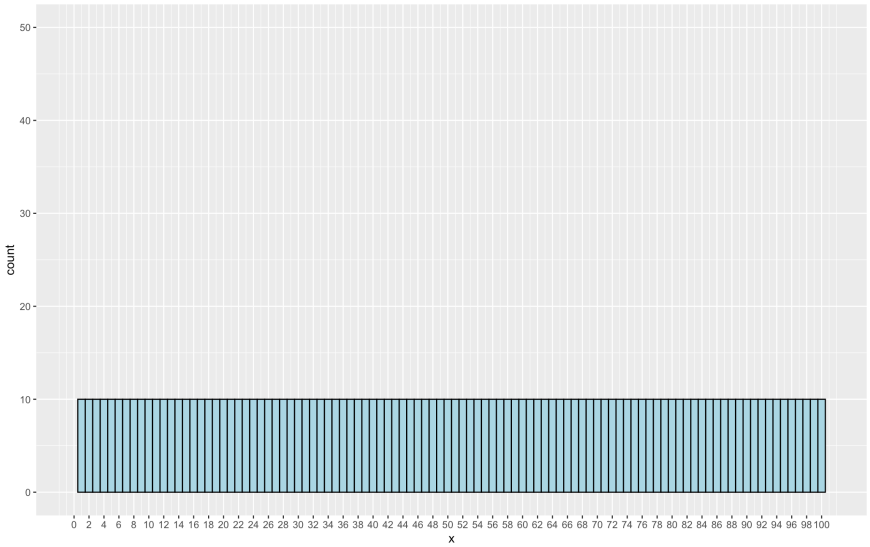

Figure 3.1: Perfect uniform distribution – each number between 1 and 100 occurs the same number of times (10 times), as you can see from the height of the bars

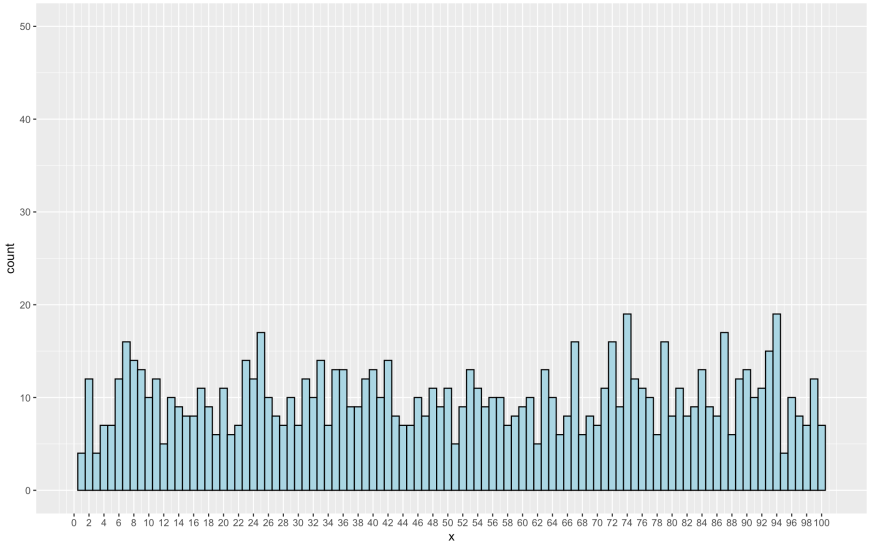

Figure 3.2: Approximate uniform distribution – each number between 1 and 100 occurs more or less the same number of times

3.1.2 Bell shaped distributions

Figure 3.3 and Figure 3.4 show two *bell shaped* distributions. Both have their peaks around the middle of the range between 1 and 100. Which means that numbers closer to the middle – or closer to 50 – occur more frequently than those far away from 50.

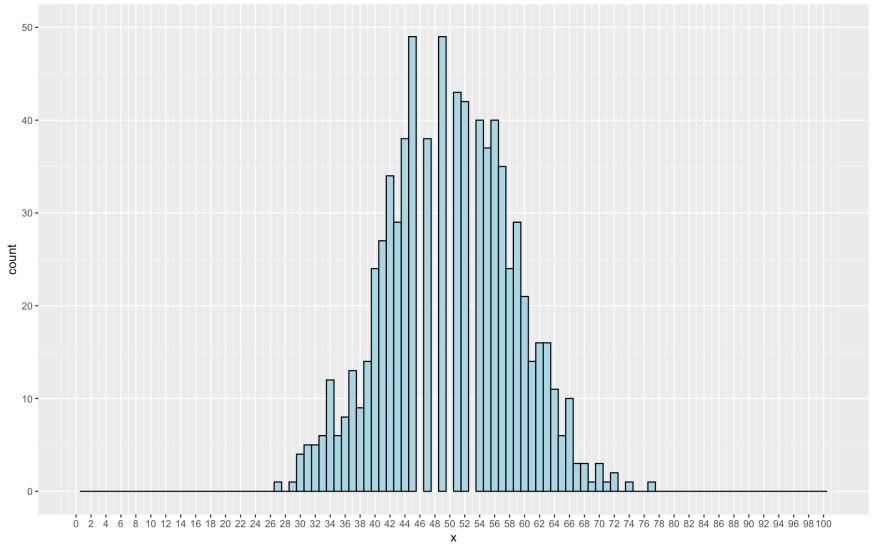

Figure 3.3: Symmetric bell shaped distribution with a peak at around 50

Both Figure 3.3 and Figure 3.4 show bell shaped distributions that are also symmetric. That is, the shape to the left of the peak is approximately similar to the shape on the right.

So, both of these are bell shaped and symmetric, and yet they are obviously different? Can you describe how they are different?

We see that the spread* of the numbers is quite different. In Figure 3.3 the numbers are more concentrated near the middle than is the case in Figure 3.4. Consequently, the peak in the first figure is higher. For example, the numbers 47 and 51 occur nearly 50 times. In Figure 3.4, the higest frequency is around 35. We call the ends of the bell shae as *tails*. We see that Figure 3.4 has *fatter* tails.

We have seen three different possibilities for how a set of 1000 numbers in the range 1 to 100 can be distributed. Even though the averages of a set of numbers and even their range might be the same they can still be *distributed* very differently.

3.1.3 Multi-modal distributions

In Figure 3.3 and Figure 3.4 we saw distributions that each had a single peak. This does not have to be the case. shows a case where we have two peaks.

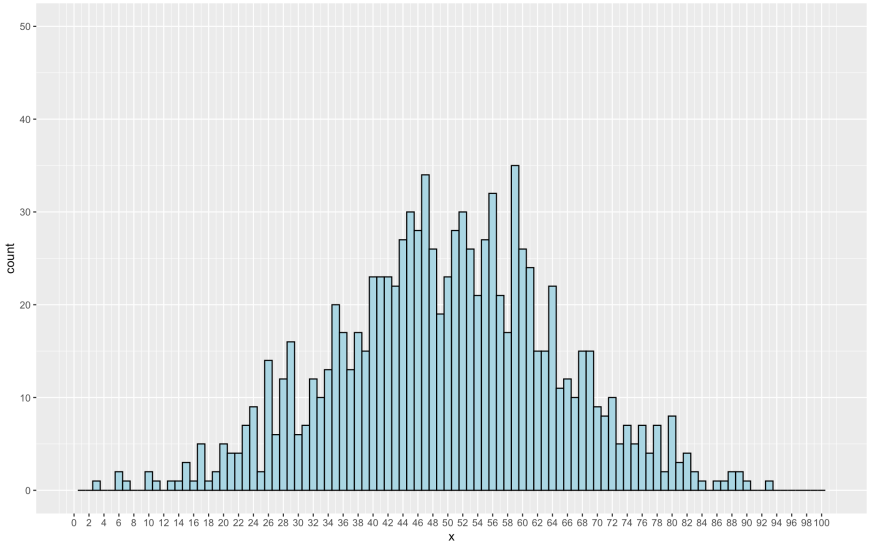

Figure 3.4: Symmetric bell shaped distribution with a peak at around 50 and higher spread than in Figure 3.3

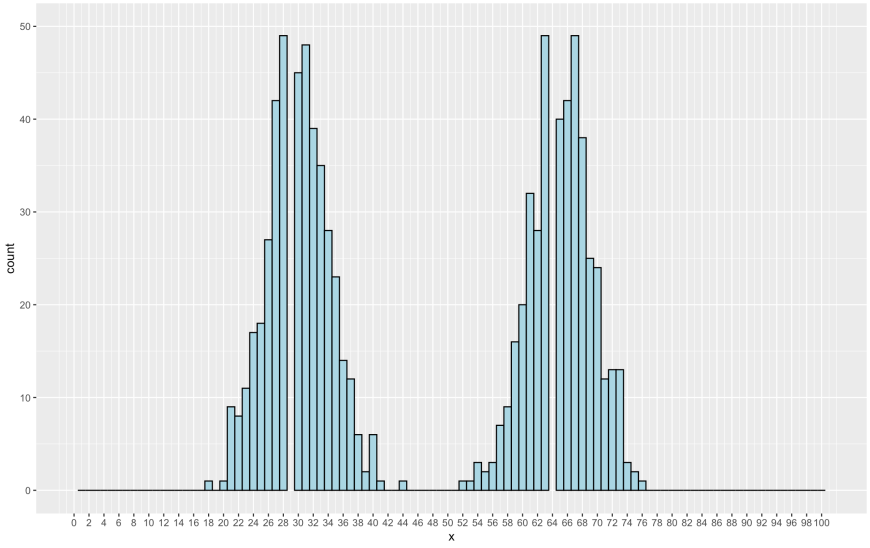

Figure 3.5: Distribution with more than one peak – multi-modal distribution

3.1.4 Skewed distributions

Thus far, we have seen distributions in which the shape has always been symmetric – explicitly so in the bell-shaped examples, but also tru for the *uniform* examples. Figure 3.6 and Figure 3.7 show examples of asymmetric distributions. The first has a long *tail* on the right and is said to be *right-skewed* and the second has its *tail* on the left and is said to be *left-skewed*.

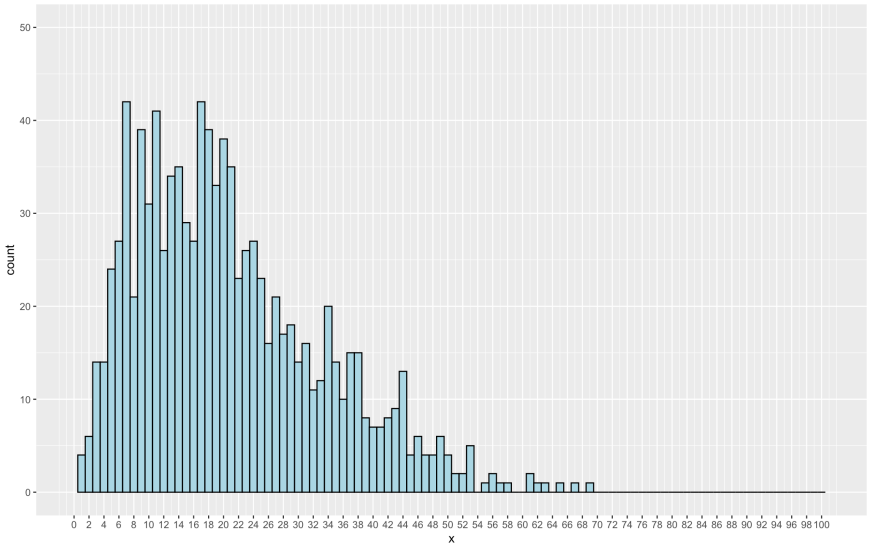

Figure 3.6: Asymmetric distribution with a right skew

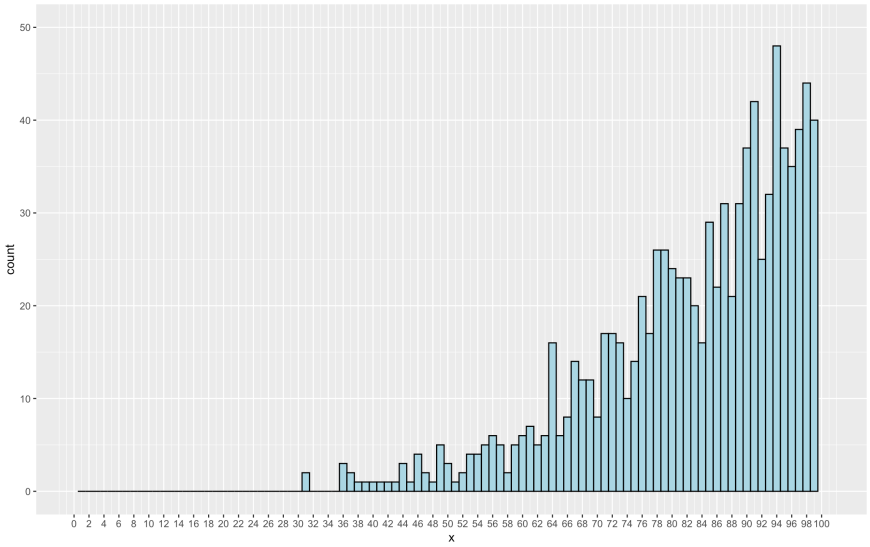

Figure 3.7: Asymmetric distribution with a left skew

3.2 Violin plots

Now that we have clarified the idea of a *distribution* let us look at distributions of numerical variables through *violin* plots. These plots serve the same function as the histograms we have looked at in the previous section, but we can do more. Let us start off by looking at a violin plot of the variable *minutes* from the *Boston_marathon* data frame

```
Boston_marathon |>
  point_plot(minutes ~ 1, annot = "violin")
```

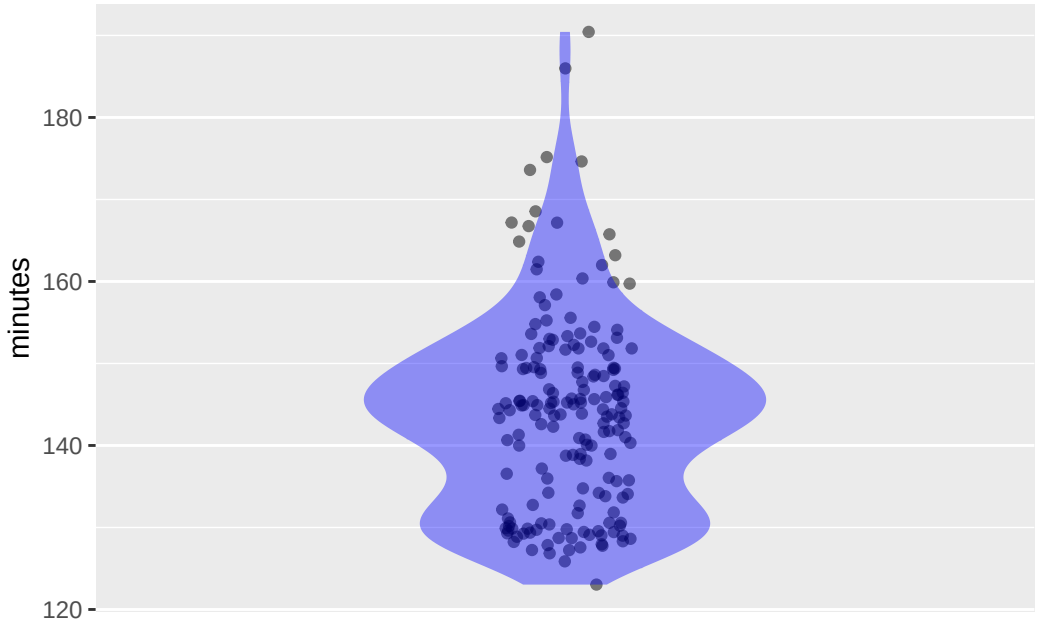

Figure 3.8: Violin plot of the *minutes* variable from the *Boston_marathon* data frame

The width of the violin at any point on the y-axis shows the relative frequency of data values close to that point. The widest part of the violin occurs at around 145 minutes. This means that the maximum concentration of winning times in the dataset is close to 145 minutes. The plot does not tell us exactly what the frequency is. The width only shows the relative frequency. Looking at where the violin is very narrow, we can say that very few runners took more than 175 minutes to complete the race. Also, relatively low number sof people took less than about 125 minutes. We can see a higher density of points around the broad regions of the violin and a low density in the narrow areas of the violin.

3.2.1 Elements of the plot

Until now, the plots we have generated using *point plot* have only shown the actual points. However, in Figure 3.8 we see two distinct elements:

- a jittered plot of the individual points
- a violin-shaped solid area – an *annotation* encompassing the points themselves

The points represent the raw data and the violin adds an *annotation* that interprets the data in some way. Here, the interpretation is a violin plot that hels us visualize the distribution of the values.

3.2.2 Examining the code

We examine the code now. In the code for Figure 3.8, we can see two arguments for the *point_plot* function. The first argument specifies the tilde expression mapping the axes. The second specifies the type of *annotation* we want. Figure 3.9 shows the code with the two arguments labeled.

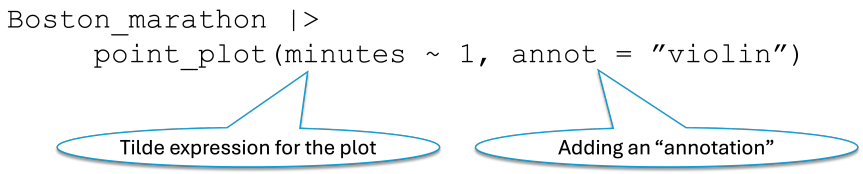

Figure 3.9: The code to generate Figure 3.8 passes two arguments to the *point_plot* function – a tilde expression and an *annotation* specification

Figure 3.10 explains the tilde expression used for the plot. Mapping of *minutes* to the y-axis conforms to what we did before. This plot deals with just a single variable – we have no variable on the x-axis. When there is no variable on the x-axis, we conventionally that by just using "1" for the x-axis variable.

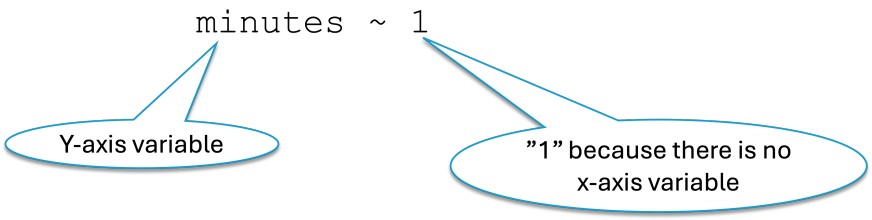

Figure 3.10: The tilde expression of Figure 3.9 we place *minutes* on the y-axis and since the x-axis has no variable, we just state "1"

Figure 3.11 explains the second argument in generating a *violin plot*.

3.3 Violin plots and distribution shapes

The violin plot displays vertical reflectional symmmetry. We can get all the information that the plot conveys just by slicing the plot vertically down the middle and looking at either of the two

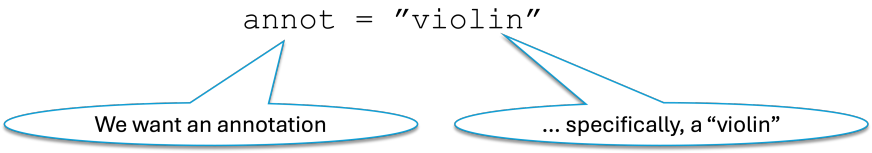

Figure 3.11: We specify that we want a *violin annotation*

chunks.

When we first talked about the various distribution shapes like *uniform*, *symmatric bell-shaped*, and *skewed*, we used histograms to convey the ideas. We can deduce the same information from violin plots as well. In the next section, we will see a few more examples. We will discuss distribution shapes in the context of those examples.

3.3.1 More examples

We see a few more examples to reinforce the idea. Figure 3.12 shows a violin plot of the variable *price* from the *price_demand* data frame.

```
price_demand |>
  point_plot(price ~ 1, annot = "violin")
```

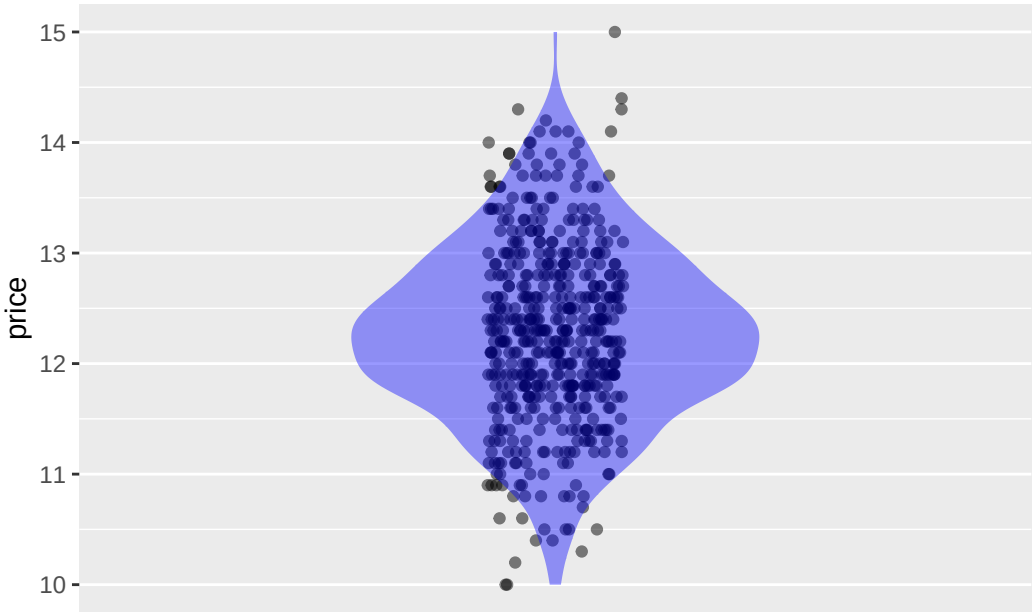

Figure 3.12: Violin plot of the *price* variable from the *price_demand* data frame

If we look at only the left or right half of Figure 3.12, turn it right by 90 degrees and get rid of the display of the individual points, we get Figure 3.13 – a symmetric bell-shaped distribution.

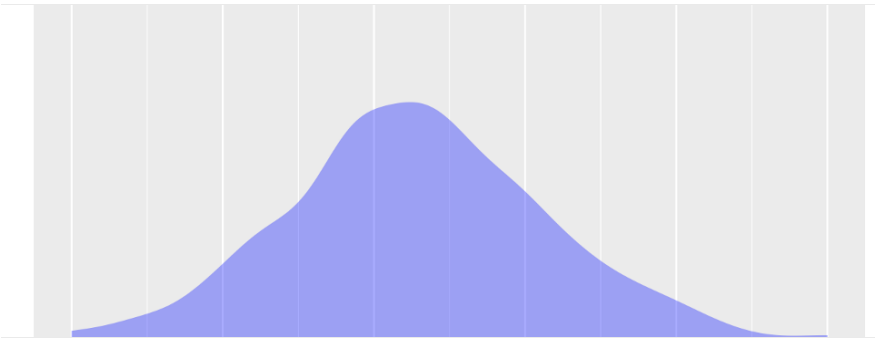

Figure 3.13: Violin from Figure 3.12 split and rotated 90 degrees right to reveal the distribution in the form we had seen before

?@fig-violin-balance-from-acct-type-balance-df shows a violin plot of the *balance* variable from the *acct_type_balance* data frame.

```
acct_type_balance |>
  point_plot(balance ~ 1, annot = "violin")
```

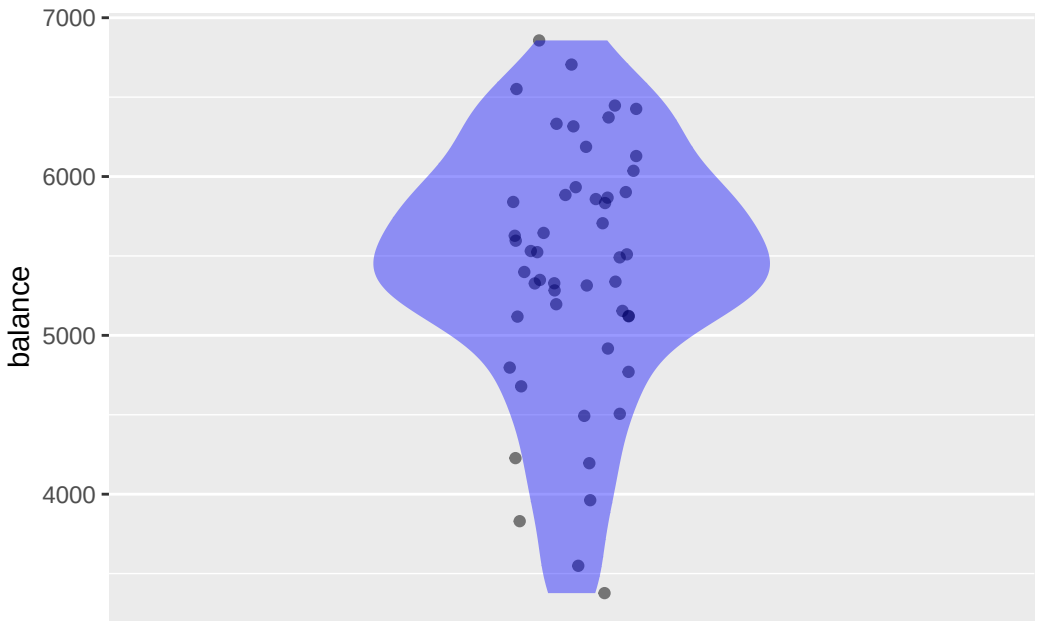

Figure 3.14: Violin plot of the *balance* variable from the *acct_type_balance* data frame

We can see from Figure 3.12 that *price* is bell-spaped and unimoda,but not symmetric. It is skewed towards the lower values because the tail is at the bottom. If we slice and rotate it as before then we would say that it is skewed left.

Next we look at the distribution of *advertising_spend_k* from the *advertising_sales_channel* data frame. Figure 3.15 shows the violin plot. From it we see that *advertising_spend_k* follows a nearly uniform distribution, but not poerfectly so. It is mildly bimodal.

```
advertising_sales_channel |>
  point_plot(advertising_spend_k ~ 1, annot = "violin")
```

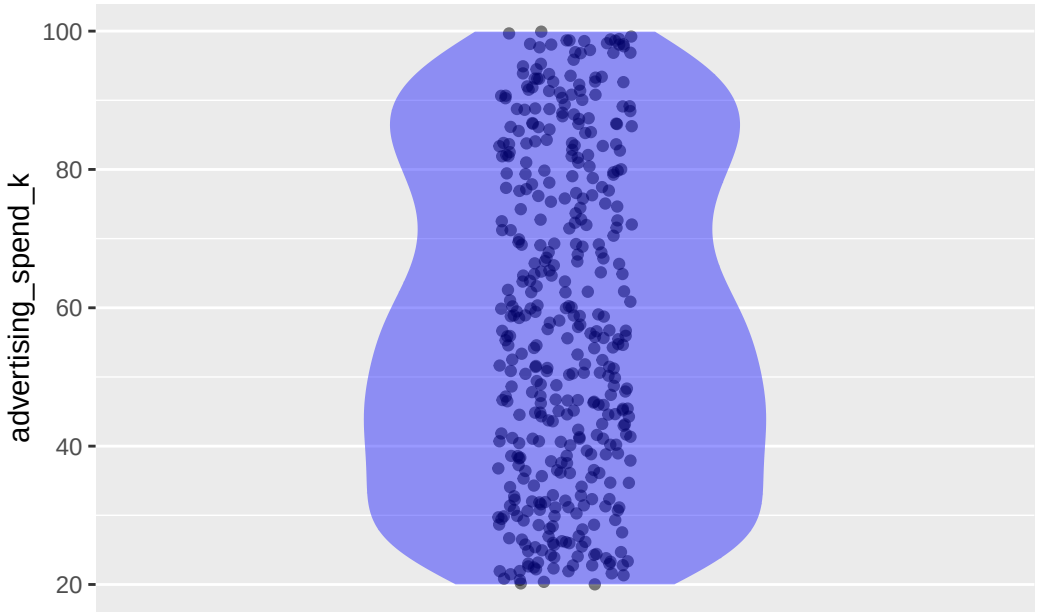

Figure 3.15: Violin plot of the *advertising_spend_k* variable from the *advertising_sales_channel* data frame

3.4 Comparing distributions with violin plots

Thus far, we have examined distributions of individual variables in isloation. In statistics we often want to compare distributions. For example, in the *acct_type_balance* data frame, how does the distribution of *balance* for *Checking* accounts compare with that for *Savings* accounts?

```
acct_type_balance |>
  point_plot(balance ~ bank_account_type, annot = "violin")
```

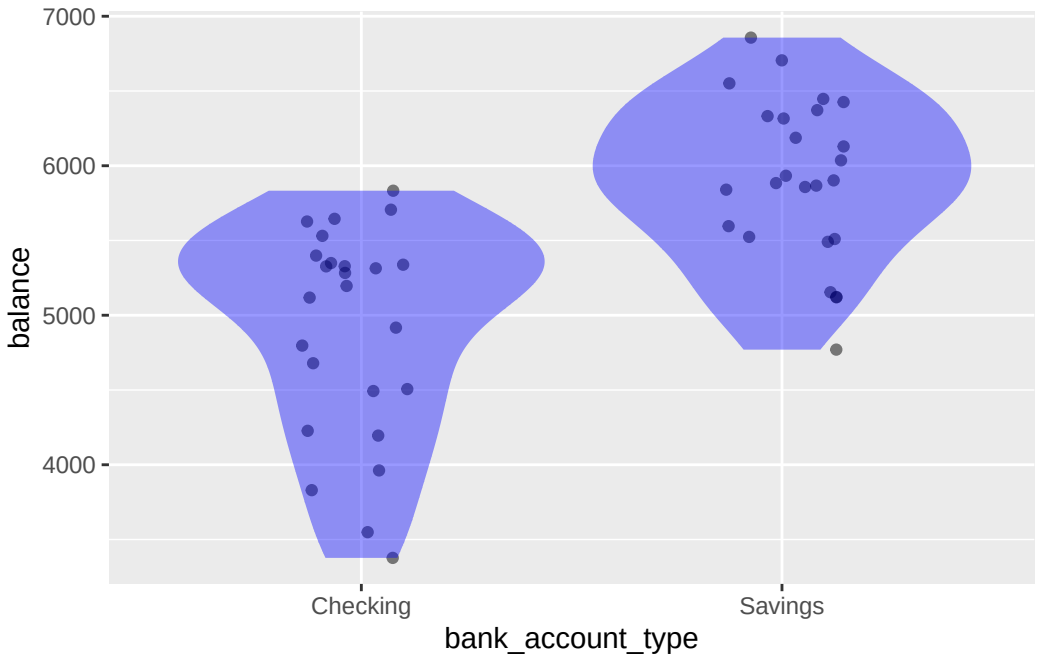

Figure 3.16: Violin plots of the *balance* variable for different accoiunt types – from the *price_demand* data frame

Since we want to compare the two violins corresponding to the two different account types, we now have *bank_account_type* on the x-axis and our tilde expression reflects this.

Figure 3.16 shows several things: - account balances in *Savings* accounts are generally larger - account balances of *Savings* accounts are distributed in an almost symmetrical bell shape with the maximum density around $6,000. - account balances of *Checking* accounts are generally smaller than those of *Savings* accounts - account balances of *Checking* accounts are bell shaped, but skewed towards lower values with the maximum density around $5,400 or so.

- As another example of comparing distributions, let us use the *advertising_sales_channel* data frame to compare the distribution of *weekly_sales_k* for different *channel*s .

```
advertising_sales_channel |>
  point_plot(weekly_sales_k ~ channel, annot = "violin")
```

Figure 3.17: Violin plots of the *weekly_sales_k* variable for different sales channels – from the *advertising_sales_channel* data frame

Figure 3.17 tells us the following: - *weekly_sales_k* values are generally lower for the *Retail Promo* channel - for the *Retail Promo* channel *weekly_sales_k* is distributed symmateically with three short peaks - *weekly_sales_k* values are almost uniformly distributed for the *Search Ads* channel and have a much larger range than the *Retail Sales* channel

4 Variance and standard deviation

This chapter discusses the key statistical concept of *variance*.

4.1 Variable

We use the term “variable” to describe a column of a data frame. Why? Let us make things more concrete. A data frame with data on employees of a company might have a column named “salary” to store the salaries of employees. Salaries of employees vary. That is, we know that not everyone in the company has the same salary (generally speaking). So, it makes eminent sense to call a column a “variable.”

Similarly, the same data frame might have another column named “Position” that stores the position of each employee. Some example values might be “Manager”, “Sales associate”, and “IT Specialist.” Here too we see “variability” because not all the values in the column are the same. So the term “variable” applies to columns whether they contain numeric or categorical values. However, for the rest of this document, we concern ourselves only with numerical variables.

4.2 Amount of Variation

Measuring the amount of variation plays an important role in statistics. If a column has no variation, then we cannot make much use of the information in the column. When a column does have variation, we often want to know the extent of variation in the values of the column. We want to ascribe a specific number – that is, *measure* the variation in a column. We call this measure the *variance* of the values in a column. When the numbers in a column are all the same, then we say that the column has no variance – that is, the variance is zero.

Consider the following three sets of 8 numbers each:

Set 1: (1, 2, 2, 1, 2, 1, 2, 1)
Set 2: (2, 6, 8, 2, 9, 2, 10, 9)
Set 3: (1, 3, 2, 3, 2, 2, 3, 4)

Which of the three sets has the highest amount of variation? That is, which set has numbers that differ by the most (among the three sets)?

Can you order the sets in increasing order of their overall variability?

Overall, we can see that the numbers in Set 1 are much closer to each other than those in the other two sets. The numbers in Set 2 are the furthest apart from each other and those in Set 3 are in-between. Therefore the order is:

Variance of Set 1 < Variance of Set 3 < Variance of Set 2

4.3 Measuring Variation through *variance*

From the foregoing, we see that the variance of a set of numbers is a measure of how much the numbers differ from each other. A set of numbers in which the elements are generally very close together has a low variance and a set that contains numbers that differ from each other by a lot has high variance. Over and above just describing what low and high variance look oike, we would like to assign a specific number to the variance of a set of numbers.

In reality we will compute the variance using the R function *var*. We describe two procedures below just to give you a good feel for what the variance represents.

4.3.1 Computing Variance: Method 1 – Using pairwise differences

We first look at an intuitive (but *impractical*) method for computing the variance of a set of numbers. Variance represents how different each member of a set of numbers is from the rest. So the difference between each number and the rest of them plays a central role. Our first approach to computing the variance involves looking at every possible pair of numbers in our set and seing how different the two numbers of the pair are from each other. We then combine all of these pairwise differences into a single number to represent the overall variability inthe set.

4.3.1.1 Steps to compute variance using pairwise differences

Let us assume that we want to compute the variance of the set of 4 num bers (2, 4, 5, 9)

1. Identify each possible pair of numbers from the set.

   (2, 4), (2, 5), (2, 9)
   (4, 5), (4, 9)
   (5, 9)
   We have 6 such pairs.

2. Compute the difference for each pair and square it.

Pair	Difference	Difference squared
(2, 4)	2	4
(2, 5)	3	9
(2, 9)	7	49
(4, 5)	1	1
(4, 9)	5	25
(5, 9)	4	16

3. Find the average of the squared differences.
   The sum of the squared differences is
   Sum = 4 + 9 + 49 + 1 + 25 + 16 = 104
   Average = 104/6 = 17.33333
4. Divide the average from the prior step by 2
   17.33333/2 = 8.66666

   So, the variance of (2, 4, 5, 9) is 8.66667

What are the characteristics of a set of numbers with zero variance? Think a little before reading on.

If the variance has to be zero then. the average of the differences also has to be zero, which means each of the differences has to be zero. That in turn means that all the numbers have to be the same. This makes a lot of sense, because if there is zero or no *variance*, then the numbers do not vary!

4.3.2 Computing Variance: Method 2 – Using deviations from the mean

We can also compute the variance using a different aproach – which also gives the same result as method 1. Most people will use this description of variance. Here, instead of finding the pairwise differences, we see how much each number in our set differs from the average (referred to in statistics as the *mean*). We then combine these individual differences into a number that represents the overall variance in the set.

Again, we use the same set of numbers (2, 4, 5, 9). Let us suppose that these numbers represent the prices of some items in a shop (in the US) and therefore they represent USD amounts.

4.3.2.1 Steps to compute variance using deviations from the mean

1. Find the average of the numbers

   Sum = 2 + 4 + 5 + 9 = 20 Average = 20/4 = 5

2. Find the squared deviations from the mean for each number.

Number	Difference from mean	Difference squared
2	3	9
4	1	1
5	0	0
9	4	16

3. Divide the total of the squared differenbces from the prior step by (n-1) whwre n is the number of elements.

   Sum of squared deviations = 9 + 1 + 0 + 16 = 26
   Result of dividing by (n-1) = 26/3 = 8.66667

4.3.3 Unit of measurement for variance

Whenever me measure anything, we measure it in terms of some *unit of measurement*. For example, we might use meter as the *unit of measurement* for people's heights, and USD as the *unit of measurement* for US company profits. In fact, in any data frame, whenever we see a numerical variable, we need to be aware of its *unit of measurement*.

4.3.4 Computing Variance: Preferred Method – Using R!

We have a data frame named *variance_example* that has a variable *num* with the values (2, 4, 5, 9). We can use the following R code to compute its variance.

```
variance_example |>
  summarize(num_var = var(num))
```

```
# A tibble: 1 x 1
  num_var
    <dbl>
1    8.67
```

Here is another example using the *mpg* data frame to compute the variance of one of its variables.

```
mpg |>
  summarize(hwy_var = var(hwy))
```

```
# A tibble: 1 x 1
  hwy_var
    <dbl>
1    35.5
```

In the previous section we computed the variance of a set of numbers that represented USD values. What are the units for variance?

In the two methods where we computed the variance, we found some differences in each step. These differences were differences between USD amounts and hence the differences have units of USD. We then squared the differences. The unit of measurement for the squares wiuld be USD-squared. We then averaged these in one approach and divided by a number in the other approach to get the variance. Therefore, variance also has USD-squared as its *unit of measurement*.

In general, when we compute the variance of a variable measured in some units (say, u, the variance has u-squared as its unit of measurement. That is, if we have a variable expressed in meters, then the variance of the variable will be in meters-squared.

We generally have a clear mental notion of units like USD and meters, but not units like USD-sauared or meters-squared.

4.4 Another measure of spread: *Standard Deviation*

We have een that *variance* has squared units and we cannot easily relate to these. Statisticians have therefore given us another measure of variability that has the same units as the original variable. This is the *standard deviation* and we compute it as the square-root of the variance.

If a variable is measured in inches, its variance has inches-squared as its unit. However, when we compute its square root, we get the *standard deviation* with inches as units.

We can now mentally relate the original variable values to the *standard deviation* and therefore statisticians use this measure vey widely. This is not to say that variance is not widely used as well. It is, and we will use it extensively in this course as well.

4.4.1 Computing *standard deviation* using R

We can use the *sd* function inside the *sumamrize* function to compute the *standard deviation*

Let us comnpute the *standard deviation* of the city mileage (variable *cty*) in the *mpg* data frame.

```
mpg |>
  summarize(cty_sd = sd(cty))

# A tibble: 1 x 1
  cty_sd
   <dbl>
1   4.26
```

Just to confirm that the computed *standard deviation* is in fact the square root of the *variance* let us compute both.

```
mpg |>
  summarize(cty_var = var(cty), cty_sd = sd(cty))

# A tibble: 1 x 2
  cty_var cty_sd
    <dbl>  <dbl>
1    18.1   4.26
```

Part III

Introduction to models

5 Module 3: Introduction to models

This module gently introduces the idea of a model starting with the idea that the average a variable is a simple and in some sense the *best model* of the variable in the absence of the values of any related variables.

The module introduces the concepts of outcome and explanatory variables.

It then goes on to consider more involved models when the values of other variables are given. Specifically it considers the idea of the *best model* for a variable if we know the value of a related categorical variable.

5.1 Learning Goals

- a
- b
- c

5.2 Structure of This Module

- Average or mean as the *best model*
- What does it mean for one variable to *explain* another?
- Response and explanatory variables (in this course we consider only numerical response variables)
- Mean as the model with no explanatory variables
- Best model with one categorical explanatory variable

6 The Simplest Model

We will play a simple game to understand the main poiint of this chapter.

6.1 Let's play *What's your best guess?*

Let us play a game. I have the following set of 10 numbers:

4, 5, 7, 1, 8, 4, 3, 9, 5, 4

Here are the steps in the game:

1. I will pick a number at random from this set
2. You have to guess the number that I chose
3. I reveal my choice.
4. We compute the difference between your guess and my chosen number (error) and square it

We will repeat this process 10 times and compute your score as the sum of all the 10 results (squared-differences) from the last step. Your goal is to keep this sum as small as possible.

Let us look at one play:

Choice	Guess	Err	Err-sq
9	1	-8	64
9	4	-5	25
4	3	-1	1
5	4	-1	1
5	4	-1	1
5	1	-4	16
4	9	5	25
8	1	-7	49
4	5	1	1
5	5	0	0
4	4	0	0
4	5	1	1
3	3	0	0
7	8	1	1

Choice	Guess	Err	Err-sq
1	4	3	9
1	4	3	9
4	7	3	9
8	4	-4	16
4	5	1	1
9	1	-8	64
		Total error	293

Let us see another play with the same choices, but different guesses.

Choice	Guess	Diff	Diff-sq
9	9	0	0
9	5	-4	16
4	3	-1	1
5	4	-1	1
5	8	3	9
5	8	3	9
4	1	-3	9
8	3	-5	25
4	5	1	1
5	4	-1	1
4	9	5	25
4	4	0	0
3	5	2	4
7	5	-2	4
1	5	4	16
1	3	2	4
4	4	0	0
8	7	-1	1
4	4	0	0
9	5	-4	16
		Total error	142

And one more:

Choice	Guess	Diff	Diff-sq
9	5	-4	16
9	7	-2	4
4	3	-1	1
5	5	0	0
5	3	-2	4

Choice	Guess	Diff	Diff-sq
5	4	-1	1
4	5	1	1
8	9	1	1
4	9	5	25
5	9	4	16
4	4	0	0
4	7	3	9
3	4	1	1
7	5	-2	4
1	1	0	0
1	7	6	36
4	5	1	1
8	7	-1	1
4	5	1	1
9	1	-8	64
		Total error	186

Of the three plays, the second play had the lowest total difference and emerged as the best of the three.

However, can we do even better? What approach will help us to reduce this total difference or error?

6.2 Average or *mean* as the best guess in the long run

It turns out that when we have a large number of tries, we will get the best results, that is the lowest sum of squared errors when we always guess the average of the numbers. In this case the average is 5 and so if we guess 5 everytime we get the best results in the long run. Here is an example for the same choices:

Choice	Mean guess	Diff	Diff-sq
9	5	-4	16
9	5	-4	16
4	5	1	1
5	5	0	0
5	5	0	0
5	5	0	0
4	5	1	1
8	5	-3	9
4	5	1	1

Choice	Mean guess	Diff	Diff-sq
5	5	0	0
4	5	1	1
4	5	1	1
3	5	2	4
7	5	-2	4
1	5	4	16
1	5	4	16
4	5	1	1
8	5	-3	9
4	5	1	1
9	5	-4	16
		Total error	113

We can see that the total error is smaller than any of the others. What we have said above **does not mean that we are guaranteed to get the smallest total error if we guess the average**. Clearly, if someone were lucky enough to guess all the numbers correctly, then their total error will be zero! Also, it is possible that someone could guess many numbers correctly just by chance and get a lower error than 113. However, those occurrences are unlikely in the long run. *Statistically* and realistically, the best course would be to guess the average.

6.3 What is a model?

People us ethe word *model* in many senses. In one of the more common usages, a model is an approximate representation of something for some purpose.

Figure 6.1 and Figure 6.2 show the image of a house and a miniature model of it. The miniature is a good representation of the house in some respoects. For example, it depicts the overall shape and some exterior details quite accurately. If someone were about to build a new house and the architect showed them this model, they would get a good idea about the overall external appearance of the house and the color scheme as well. However, this model differs from the actual house in many ways as well – one cannot actually step inside it! We cannot get a feel for how roomy the interios=r will feel. We do not know how the house will look in its actual surroundings. It is accurate in some respects and inaccurate in others.

Should we consider the model in Figure 6.2 good? In the words of statistician George E. P. Box, “All models are wrong. But some are useful for certain purposes.”

All models are wrong in the sense that they are only *representations*. However, as the miniature model of the house in Figure 6.2, they can still be useful for some purpose.

We will be building many models in this course and should remember the folloiwng:

Figure 6.1: Picture of a house

Figure 6.2: Model of the house in Figure 6.1

- Models are simplifications: They inherently omit details and idealize reality to make complex systems manageable.
- Utility over accuracy: The goal isn't perfect truth, but practical application and insight for a specific goal, whether it's scientific, business, or everyday decision-making.
- Context matters: A model's usefulness depends on the purpose; a simple model might be great for a quick estimate, while a complex one is needed for detailed forecasting.
- Pragmatic approach: We should focus on how wrong a model can be and still be helpful, rather than getting stuck on its imperfections.

How to apply this philosophy:

- Question your models: Continuously ask how and where a model breaks down.
- Understand its limitations: Know the assumptions and simplifications behind the model.
- Adapt and evolve: Be willing to refine or discard models as new information emerges or situations change.

6.4 Mathematical models

In this course, we will be building a specific kind of models – *statistical models*. However, before getting to *statistical* models, we will consider some simpler *mathematical models*. The term *mathgematical model* can connote many different things. Let us clarify the sense in which we use the term. Suppose I have a data frame with data about many homes. For each home, we have its *age*, *floor-area_sqft*, *number_of_bedrooms*, *number_of_bathrooms*, and *price*.

We might build a model to determine the value of *price* given the values of one or more of the other variables. Remember, this is a model and so we do not expect it be be exact. We do not aim for the model not determininbg the actual *price*, but only to compute a good *estimate* of the *price*. (We have obviously simplified the situation for pedagogical purposes. House prices depend on many other factors. But that need not hold us back. You will still learn the underlying concepts.) S, what might the model look like? Equation 6.1 shows the model. For now, do not worry about how we arrived the model.

$$\text{price} = 61{,}399 + 394\,\text{floor\_area\_sqft} \tag{6.1}$$

In the model, we have placed a *hat* over *price* because the model computes not the actual price of a home with the specified floor area and instead only computes an estimate. In statistics, we place a *hat* over quantities that estimate other quantities.

Equation 6.1 shows a *linear* model. We call it *linear* because all variables are only raised to the fist power. We have not squared or raised a variable to any other power. In this course we build only *linear models*.

In our context, we use the term *model* to denote a mathematical equation of the kind that Equation 6.1 shows. Our models help us to determine the value of a variable in a data frame

for any given instance. Specifically, in the house price example that we just considered, the model in Equation 6.1 enables us to estimate the *price* of a home, given its floor area.

6.4.1 *Response* and *Explanatory* variables

We call the variable whose value the model estimates as the *response* variable. The variables based on which the model estimates the *reponse* variable are called *explanatory* variables. We can also refer to the *response* variable as the *dependent* variable, and *explanatory* variables as *independent* variables.

6.4.2 Using the model

The house in the fifth row of the data frame has floor area of 1480. Its actual price is $553,000. What price does the model estimate? Well, we can plug the floor area into the model – Equation 6.1 – and find out. If we plug it in, we get $645,999. The model overestimates this price.

The house in the fourteenth row of the data frame has floor area of 1293 Its actual price is $561,000. What price does the model estimate? Well, we can plug the floor area into the model – Equation 6.1 – and find out. If we plug it in, we get $572,134. Quite close.

You should try out some more cases and wee how the model does. We will learn a lot more about such models, including precisely measuring their *quality* as we go forward.

I won't blame you if you are wondering "We already have the prices in the data frame. Why do we need a model to calculate these?"

Consider again the home in the fifth row. Why did the model overestimate the price? Well, you should consider that this particular home was not the only one whose floor area was in the vicinity of 1480. The dataset has other homes with similar floor areas and their prices vary. If we look at homes that have floor areas between 1430 and 1530, we see that their proces range from $553,000 to $647,000. In fact, many homes below 1480 square feet in floor area have higher prices. How could that be? Well, variables other than floor area play a role in price too. So our model makes an estimate based on the whole data frame.

The next section addresses two main uses of models.

6.5 Purpose of a model

We use models in two main ways:

1. *Prediction*: Suppose we have built a model using the data in a data frame to estimate the *price* of a home, given its *floor_area_sqft*. We can potentially use this model to estimate the price of any home whose floor area we know and shose price we do not know – even if this home is not in the dataset.
2. *Explanation*: Our model in Equation 6.1 tells us that the price of a home is related to its floor area. This makes sense and we could have said this even without a model. However, when people gather data about phenomena that they do not fully understand, and want to understand, they often build models to understand and explain which variables seem to be related and in what way to a variable of interest.

You might have noted that both of the above purposes actually require us to use a model in contexts that go outside the data that we based the model on. When we start using models outside of the data we used to build the model, we come face to face with the essence of statistics. We will get to that later in the course.

6.6 Average is a model too!

In Equation 6.1, we had a *response* variable and an *explanatory* variable. What if we do not have any explanatory variable? In the context of our home prices example, not having an *explanatory* variable is equivalent to saying:

- "I have a dataset of 100 homes"
- "Here are their prices"
- "I am not telling you anything else about these homes"
- "I have selected a random home from my dataset"
- "Guess the price of the home I nhave in mind"

Based on the game we played at the start of this chapter, you first compute the average of the prices as $733,440. Your best *model* to make your guess is:

$$\text{price} = 733{,}440 \tag{6.2}$$

That is, like in the game we played, no matter which home I choose, you always state the average as your guess.

The big difference between the models in Equation 6.1 and Equation 6.2 is that the second one *has no explanatory variable*. When we are not given any additional helpful information to estimate a value, our best estimate – *best model* – is the average!

Now you know why I titled the chapter **The Simplest Model**

7 Category Means

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

8 Residuals

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part IV

More on models – least squares

Module 4: More on Models – Least Squares

This module completes what the previous one began. It enhances the idea of the mean as the *best model* to the case when the response variable is numeric as is the explanatory variable. It introduces the idea of the line of best fit based on the least squares critereon as the *best linear model*.

Learning Goals

- a
- b
- c ## Structure of This Module
- Line that goes *closest* to all points
- Line as model
- Residual
- Least-squares line
- The sense in which the least-squares line the *best line*
-

9 Concept of model

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part V

Linear Regression Models

Module 4: Linear Regression Models

This module fills in the details on top of the intutional foundation for a linear model that the previous module laid. Specifically, the module covers R code for building linear models for the case of a single linear explanatory variable, a single categorical variable and a combination of numerical and categorical explanatory variables.It shows how to distill the model function based on the regression output.

Learning Goals

- a
- b
- c

Structure of This Module

- Notion of model function
- mention that the model gives an estimated value, but defer detailed discussion until later
- R code for building model with no explanatory variable and reconstructing the model function
- R code for building model with one numeric explanatory variable and reconstructing the model function
- R code for building model with one numeric explanatory variable and one categorical explanatory variable and reconstructing the model function

10 Line Models

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

11 Interpreting coefficients

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part VI

Explanatory Power of a Model

Module 6: Explanatory Power of a Model

We turn our attention now to measuring the quality of a model based on the notion of a model's ability to *explain* the variability in the response variable.

Learning Goals

- a
- b
- c

Structure of This Module

- What does it mean for a model to explain variability in the response variable?
- Computing the model values and residuals
- the variance equation
- R-squared

Part VII

Populations, Samples and Inference

Module 7: Population, Samples and Inference

At this point, we have learned about models in general, and linear regression models in particular. In addition, we can also now use R to build linear models and to construct the model function based on the output of the R code. We have also looked at the quality of a model based on its R-squared.

Until now we have looked at a mdoel as a mathematical function into which we can plug in the value of explanatory variable(s) and get an estimated value for the response variable.

We now transition from mathematical functions to *statistical models*. We first define the terms between *population* and *sample* and reveal that much of statistics deals with *inference*, that is, learning something about the population based on just a sample.

Learning Goals

- a
- b
- c

Structure of This Module

- Population and sample
- Statistc

12 Population vs Sample

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

13 Sampling Variability

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part VIII

Probability as Variation

Module 8: Probability as Variation

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

14 Randomness

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

15 Variability

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

16 Distributions

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part IX

Sampling Variation and Estimators

Module 9: Sampling Variation and Estimators

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

17 SE Estimators

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

18 Sampling Distributions Slopes

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part X

Confidence Intervals and Bands

Module 10: Confidence Intervals and Bands

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

19 CI Slope

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

20 CI Bands

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part XI

Signal, Noise, and R-Squared

Module 11: Signal and Noise

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

21 Variance Decomposition

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

22 R-Squared

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part XII

Hypothesis Testing via Regression

Module 12: Hypothesis Testing

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

23 t-tests for Regression

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

24 Interpretation

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part XIII

Business Applications of Regression

Module 13: Business Applications

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

25 Marketing

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

26 Finance

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

27 Human Resources

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

28 Operations

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part XIV

Projects and Datasets

Module 14: Projects

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

Structure of This Module

- Data frames
- Point plots
- Relationships between variables

29 Project Guidelines

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

30 Datasets

The mean of a numerical variable is the “best guess” of its value when no other information is available — in the specific sense that it minimizes the sum of squared errors.

library(ggplot2) mean(mtcars$mpg)

Part XV

Appendices

31 Appendices

This module introduces the basic building blocks of statistical thinking: data frames, variables, and point plots. We begin with visualization because patterns reveal what models will later formalize.

31.1 Learning Goals

- Understand data frames, variables, and instances
- Create and interpret point plots
- Recognize patterns: direction, form, strength

31.2 Structure of This Module

- Data frames
- Point plots
- Relationships between variables