4

Insights from ONE Variable

Any phenomenon, event, organization or the like that we study always has several if not lots of characteristics. In this chapter, I will pay attention to visuals that are used to represent one variable. I assume you have a basic knowledge of statistics, as several statistical measures are related to univariate exploration; however, I will include basic comments on what is appropriate to use in each situation.

4.1 Information from ONE Variable

If your data has one column it does not mean it is poor information, maybe it is a summary variable representing several dimensions, and which was computed via some process. So, *one* column can represent:

- **Raw values**: These are direct measurements that may need further processing to represent some insightful information. A raw value is a count of newborns.
- **Indicators**: These are processed raw values computed via some mathematical operations. Familiar indicators are presented as some sort of ratio or rate. The ratio of dead newborns by total newborns in a specific year for a particular location is an indicator.
- **Indexes**: These are aggregated or composite indicators (Babbie, 2013). The democracy index (The Economist Intelligence Unit, 2019) we used in the previous chapter is a clear example of a value that tries to aggregate several indicators; notice that these indicators were indexes themselves, then an index can have sub-indexes. In economics, indexes also represent a measure of change related to a base value (Dorin et al., 2020).

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

As any of the options above[1] can be in one column, please reflect on what you have. In this book, we often use indicators or indexes; and you must be aware that the raw values you may have in your data table might often need further processing. For a further discussion on this, please refer to OECD et al. (2008) and Magallanes Reyes (2017).

4.2 Categorical I: Visualizing Gaps

You think of gaps when you have a threshold, a value that represents a particular boundary from which some values represent something qualitatively different than the others. Then, once categorical data are organized into a frequency table, you can see which category count or percent is beyond or above a certain threshold value.

For this example, let me use the data from 2017 on public schools from the US Department of Education presented on page 16. Let me open that file again:

```
> #link to data
> linkRepo='https://github.com/resourcesbookvisual/data/'
> linkEDU='raw/master/eduwa.csv'
> fullLink=paste0(linkRepo,linkEDU)
> #getting the data:
> eduwa=read.csv(fullLink,stringsAsFactors = FALSE)
```

The data frame `eduwa` has a column *LocaleType*, a nominal variable (see page 14) that tells you the location of a school. I should start by preparing its frequency table as a data frame, this time I will include the percent values:

```
> ValuesAndCounts=table(eduwa$LocaleType,useNA = "ifany")
> values=names(ValuesAndCounts)
> counts=as.vector(ValuesAndCounts)
> FTloc=data.frame(Location=values,Count=counts)
> FTloc$Percent=100*round(counts/sum(counts),4)
> # ordering FreqTable by ascending counts
> FTloc=FTloc[order(FTloc$Count),]
> row.names(FTloc)=NULL # resetting row names
> # here it is:
> print(FTloc, row.names = F)
```

```
 Location Count Percent
     <NA>    72    2.97
     Town   338   13.93
    Rural   505   20.81
     City   714   29.42
   Suburb   798   32.88
```

[1] The options presented will be consistent throughout the book, but I am aware this distinction may not always be clear or discussed elsewhere.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

The previous frequency table included missing values as locations, because some schools had not been categorized (I forced that using `ifany`). However, you may want to name that category level:

```
> # adding a level:
> levels(FTloc$Location) = c(levels(FTloc$Location),'Uncategorized')
> # using that level for the 'NA' value:
> FTloc$Location[is.na(FTloc$Location)]='Uncategorized'
```

Notice that the column `Location` in **R** was created as a categorical one, then I needed to add a new level before replacing the missing value. In *Pandas*, later, that column will be considered as text, so its replacing will require one step less.

At this point, I need to decide the threshold value. For this example, let's use 25 percent, the value that represents the uniform share if you have four alternatives (remember we have four location types). Then, a gap value is simply the difference. Let me add that column to my `FT1loc` object:

```
> # new column with gap value
> FTloc$Gap=round(FTloc$Percent-25,0)
```

Let me also create a *flag* (`Above_Equal_Share`), which will tell me if a value is negative or positive:

```
> # new column with True if gap is positive (False otherwise)
> FTloc$Above_Equal_Share=FTloc$Gap>0
```

The **Python** code that will create a frequency table as a *Pandas* data frame in a similar fashion can be:

```
import pandas as pd

#link to data
linkRepo='https://github.com/resourcesbookvisual/data/'
linkFile='raw/master/eduwa.csv'
fullLink=linkRepo+linkFile
eduwa=pd.read_csv(fullLink)
#
# Frequency table
FTloc = pd.value_counts(eduwa.LocaleType,
                        ascending=True,
                        dropna=False).reset_index()
FTloc.columns = ['Location','Count']
# adding column
FTloc['Percent']=100*(FTloc.Count/FTloc.Count.sum()).round(4)
#
FTloc['Location'].fillna('Uncategorized', inplace=True)
# new column with gap value
FTloc['Gap']=(FTloc.Percent-25).round(0)
# new column with True if gap is positive (False otherwise)
FTloc['Above_Equal_Share']=FTloc.Gap>0
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

Let's make a barplot with `FTloc`, generally a safe choice for categorical data. Let me show you step by step:

1. Prepare the text for the titles (I will not use axes titles):

```
> texts=list(TITLE="Distance from even distribution",
+            sTITLE="Location of Schools in WA State (2018)",
+            SOURCE="Source: US Department of Education")
```

2. By default, *ggplot* will plot the categorical levels alphabetically, so you need the bars ordered by count:

```
> rePOSITION=FTloc$Location
```

3. Prepare the base layer. I use for `y` the values in `Gap`, which are positive and negative. I use `label` argument to later annotate bars with `geom_label`.

```
> library(ggplot2)
> info1=ggplot(data=FTloc, aes(x=Location,y=Gap,label=Gap))
```

4. Choose the geometry element needed:

```
> barFT1= info1 + geom_bar(stat = 'identity',width = 0.5)
```

5. Set the basic style:

```
> barFT1= barFT1 + theme_classic()
```

6. Change alignment defaults (notice that while *ggplot* requires coding for centering titles, **Python** will use center aligment by default):

```
> barFT1= barFT1 + theme(plot.title= element_text(hjust= 0.5),
+                        plot.subtitle= element_text(hjust= 0.5),
+                        plot.caption= element_text(hjust= 0))
```

7. Get rid of elements to maximize "data–ink ratio":

```
> barFT1= barFT1 + theme(axis.ticks= element_blank(),
+                        axis.text.y= element_blank(),
+                        axis.title.y= element_blank(),
+                        axis.title.x= element_blank(),
+                        axis.line.y= element_blank(),
+                        axis.line.x= element_blank())
```

8. Add titles using the list `texts` from step 1. above:

```
> barFT1= barFT1+ labs(title=texts$TITLE,
+                      subtitle = texts$sTITLE,
+                      caption = texts$SOURCE)
```

9. Reposition the bars:

```
> barFT1= barFT1 + scale_x_discrete(limits=rePOSITION)
```

10. Add the threshold line:

```
> barFT1=barFT1 + geom_hline(aes(yintercept=0))
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

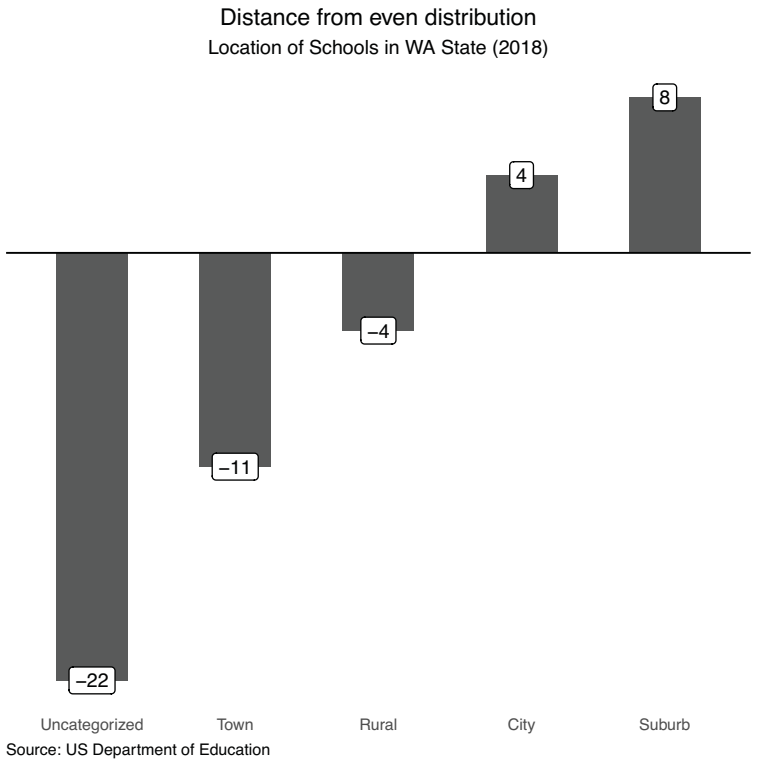


Figure 4.1 The barplot for gaps
Each bar represents the distance to the threshold.
Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

11. Annotate the bars. Remember that the `aes` for `geom_label` was defined in the object `info` in step 3. above. The final version is displayed in Figure 4.1.

```
> #text in bars
> barFT1=barFT1 + geom_label()
```

As expected, since the `y` represent `Gap`, the bars show positive and negative direction.

Using **Python**'s *plotnine*, I can obtain Figure 4.1 with this code:

```
texts={'TITLE':"Distance from even distribution",
       'sTITLE':"Location of Schools in WA State (2018)",
       'SOURCE':"Source: US Department of Education"}

rePOSITION=FTloc.Location

from plotnine import *
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
info1=ggplot(FTloc,aes(x='Location',y='Gap',label='Gap'))
barFT1= info1 + geom_bar(stat = 'identity',width=0.4)
# theme
barFT1= barFT1 + theme_classic()
#erasing
barFT1= barFT1 + theme(axis_ticks= element_blank(),
                       axis_text_y = element_blank(),
                       axis_title_y = element_blank(),
                       axis_title_x = element_blank(),
                       axis_line_y = element_blank())
#repositioning
barFT1= barFT1 + scale_x_discrete(limits=rePOSITION)
#threshold
barFT1=barFT1 + geom_hline(aes(yintercept=0))
#text in bars
barFT1=barFT1 + geom_label(aes(label=rePOSITION),
```

The code above needs the help of *matplotlib* (Hunter, 2007) for adding the text titles and caption:

```
# text with matplotlib
import matplotlib.pyplot as plt

fig = barFT1.draw()
fig.text(x=0.1,y=-0.02,s=texts['SOURCE'])
plt.suptitle(texts['TITLE'], y=0.97, fontsize=14)
plt.title(texts['sTITLE'], fontsize=10)
```

Do we have an alternative to barplots? Most people tend to use **pie charts** with categorical data, but this should not be the default option, as proposed by Hickey (2013). However, if you are looking for an alternative for categorical data consider the **Lollipop plot**. The lollipop, as proposed by Cleveland and McGill (1984), can replace the barplot. The position of head of the lollipop, a dot, will represent the value, and the stick (or stem) will highlight that value as a distance (as the bar does). Let me show you what you need to adapt from the **R** code from page 72.

- Step 1 and 2 are the same.
- Adapt step 3:

```
> # changes in aes:
> info2 = ggplot(FTloc, aes(x=Location,
+                           y=Gap,
+                           color=Above_Equal_Share,#new
+                           label=Gap))
```

- Change `geom` in step 4: We actually need two geoms (point and segment). Notice the structure of `geom_segment`. I recommend to add the dot layer after the stick, if not, when you use text for the dot, the stick will cover the value.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> # one for the lollipop stick
> lol1= info2 + geom_segment(aes(y = 0, #from
+                                yend = Gap, #to
+                                x = Location,#from
+                                xend = Location),#to
+                            color = "gray")
> # one for the lollipop head (just a dot)
> lol1= lol1 + geom_point(size=10)
```

- Step five and six are the same
- I will improve here the fact that in Figure 4.1 the category labels are far from the bars. So, in step 7, you will get rid of the categorical labels in the x-axis. The visual after this step will have no x-axis labels.

```
> # NO X-AXIS
> lol1 = lol1 + theme(axis.ticks= element_blank(),
+                     axis.text.y = element_blank(),
+                     axis.title.y = element_blank(),
+                     axis.line.y = element_blank(),
+                     # no more x-axis elements
+                     axis.text.x = element_blank(),
+                     axis.line.x = element_blank(),
+                     axis.title.x = element_blank())
```

- Steps eight and nine remain unchanged.
- I will change the default type of line in step ten:

```
> # annotating threshold
> lol1 = lol1 + geom_hline(yintercept=0,
+                          linetype = "dashed")
```

- Step eleven will have major changes. I will need two labels, one for the values of `Gap`, and one for the label of the location the bar represents (remember that I erased the x-axis elements). The former does not need to set `aes`, so the aesthetic `label` will be *inherited*. In the latter, I do need to tell what is the aesthetic to plot. Notice that `geom_label` will generate a legend element by default, so I am preventing that (you can try not preventing it, and see what happens).

```
> # for 'Gap' values.
> lol1 = lol1 + geom_text(show.legend = FALSE,color='white',size=4)
> # for 'Location' values.
> lol1 = lol1 + geom_label(aes(label=rePOSITION),
+                          color='black',size=3,
+                          y=0,show.legend = FALSE)
```

- A new step will be to use a *geom* that uses the aesthetics `color` I requested at the `ggplot`'s `aes`. In this case, I request gray colors. Since the color depends on the values of `Above_Equal_Share`, it will have two gray levels: I reverted the gray scale so that FALSE receives the lightest gray.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> #coloring
> lol1 = lol1 + scale_color_grey(start=0.6,end=0.2)
```

- The final and new step will be the legend positions. The element `scale_color_grey` will generate a legend, I am allowing it to reinforce the concept of gaps. However, the default position will shrink the plotting area (see that discussion on Section 3.2.6); then, I will move the legend to a different location. In this case, I will use the plotting area coordinates, where (0,0) represents the lower left corner, and (1,1) is the top right corner. Finally, I frame the legend.

```
> # legend position and frame
> lol1 = lol1 + theme(legend.position = c(0.8,0.4),
+         legend.background = element_rect(linetype="solid",
+                                          colour ="grey"))
```

Of course, if you believe the color legend is not needed, you simply write:

```
> #  IF YOU PREFER NO LEGEND:
> lol1 = lol1 + guides(color=FALSE)
```

I will use the color legend this time.

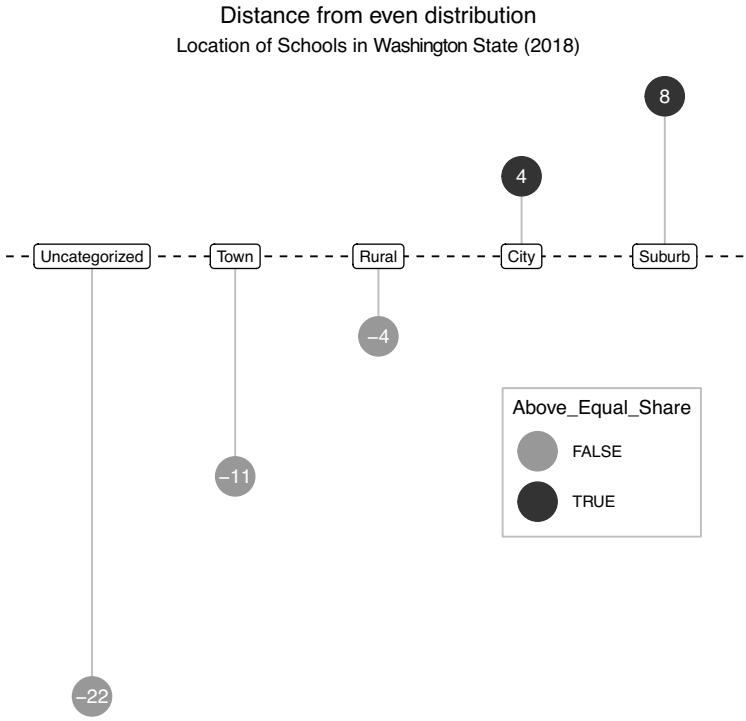
As you are following a *grammar of graphics* approach, you realize several points by now after the examples:

- Order of layers matter.
- *ggplot* in **R** will align title and subtitles to the left, and caption to the right. Plotnine in **Python** centers the titles by default.
- The plotting area has the same coordinate system, from (0,0) to (1,1).
- The size of elements is managed in the same way in both **R** and Python.
- The text size is not understood in the same way in **R** and Python. The value you use to increase or decrease text is different in each case, so you should try manually different sizes (**R** needs smaller values than Python).

Keep in mind all those differences, as I will not be giving that much detail later on this same issues in the book. With that in mind, see the result from the previous code in Figure 4.2.

Notice that the transformation from bars in Figure 4.1 to lollipops in Figure 4.2 was worth the effort. However, bars are a strong point as they are very familiar to most people with or without a statistical background. So having this as a default for categorical data is generally a good choice. Also, consider the possibility of removing the legend, as most details are already clear.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press



Source: US Department of Education

Figure 4.2 Lollipop to represent gaps

The lollipop can be used to easily show distance to threshold. Each dot represents the gap to a uniform distribution of schools per location type in Washington State. See code to realize how to put legend inside plotting area.

Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

The adaptation to the previous code using Plotnine to re create a **Python** version of Figure 4.2, and with the now usual help of Matplotlib, can be created using the next code (notice all the differences):

```
# changes in aes:
info2 = ggplot(FTloc, aes(x='Location',
                          y='Gap',
                          color='Above_Equal_Share',
                          label='Gap'))
# one for the lollipop stick
lol1=info2 + geom_segment(aes(x = 'Location',xend = 'Location',
                              y = 0,yend = 'Gap'),#here
                          color = "lightgray")
# one for the lollipop head (just a dot)
lol1 = lol1 + geom_point(size=10)
# NO CHANGES (no realigment needed in Plotine)
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
lol1 = lol1 + theme_classic()
# NO X-AXIS
lol1 = lol1 + theme(axis_ticks= element_blank(),
                    axis_text_y = element_blank(),
                    axis_title_y = element_blank(),
                    axis_line_y = element_blank(),
                    axis_text_x = element_blank(),
                    axis_line_x = element_blank(),
                    axis_title_x = element_blank())
# NO CHANGES (title later)
lol1 = lol1 + scale_x_discrete(limits=rePOSITION)
# annotating threshold
lol1 = lol1 + geom_hline(yintercept=0, linetype = "dashed")
# for 'Gap' values
lol1 = lol1 + geom_text(show_legend = False,color='white',size=8)
# for 'Location' values.
lol1 = lol1 + geom_label(aes(label=rePOSITION),
                         color='black',size=9,
                         y=0, show_legend = False)
# coloring
lol1 = lol1 + scale_color_gray(0.6,0.2)
# legend position and frame
lol1 = lol1 + theme(legend_position = (0.8,0.4),
                    legend_background = element_rect(linetype="solid",
                                                     colour ="grey"))
# text with matplotlib
import matplotlib.pyplot as plt

fig = lol1.draw()
fig.text(x=0.1,y=-0.02,s=texts['SOURCE'])
plt.suptitle(texts['TITLE'], y=0.97, fontsize=14)
plt.title(texts['sTITLE'], fontsize=10)
```

4.3 Categorical II: Visualizing Representativity

Representativity is of interest when you want to know how much a particular population is present among others. Let me open a dataset with the residence and party of each member of the Washington House of Representatives for the 66th Legislature (Washington State Legislature, 2019). I can find the amount of representatives from the party or from the residence of the member. Let me try *Residence*, a nominal variable, so that I can get to see which cities are represented.

```
> linkREPS='raw/master/reps.csv'
> fullLink=paste0(linkRepo,linkREPS)
> reps=read.csv(fullLink, stringsAsFactors = F)
```

As before, I should start by preparing its frequency table as a data frame:

```
> ValuesAndCounts=table(reps$Residence,useNA = "ifany")
> values=names(ValuesAndCounts); counts=as.vector(ValuesAndCounts)
> FTrep=data.frame(Residence=values,Legislators=counts)
> FTrep=FTrep[order(FTrep$Legislators),]; row.names(FTrep)=NULL
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

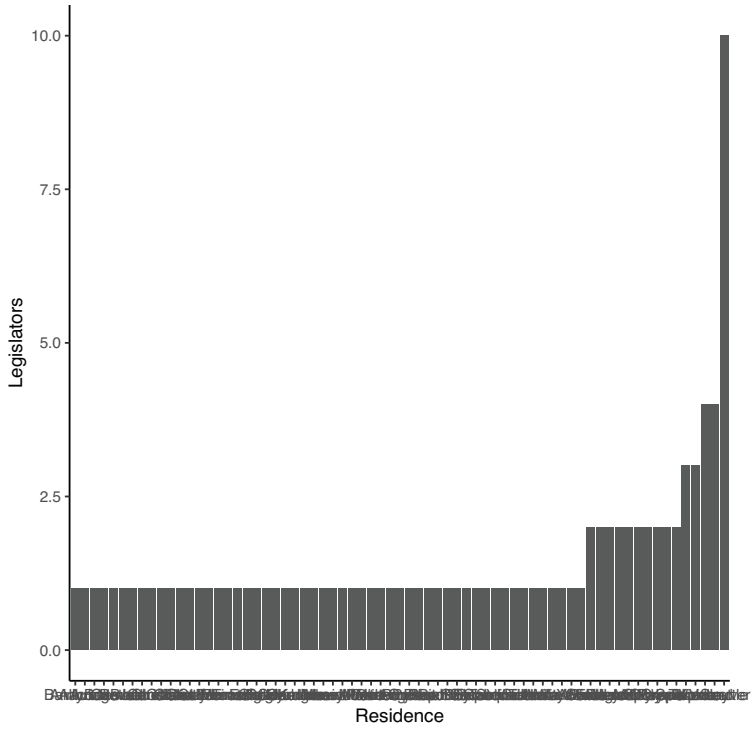


Figure 4.3 Bars for representivity

Using bars to show how sixty-nine cities are represented. The plot has not gone through major editing.

Source: Washington House of Representatives for (66th Legislature) (Washington State Legislature, 2019).

Let me make a basic barplot of the object `FTrep`:

```
> info3=  ggplot(FTrep,aes(x=Residence,y=Legislators)) + theme_classic()
> barFT2= info3+geom_bar(stat='identity')
> barFT2=barFT2 + scale_x_discrete(limits=FTrep$Residence)
```

The State House has sixty-nine cities represented in the 66th legislature (Washington has 212 cities according to the 2010 census (Wikimedia Foundation, 2019)), so you have this many bars in Figure 4.3. It has been shown that this many bars can cause annoying distractions to readers, a phenonenon called *The Moiré Effect* (Saveljev et al., 2018). So when too many bars are present, we need to find a better solution. This better way is the lollipop. Let's do it step by step:

1. Prepare texts for titles and caption:

```
> # text for titles
> texts_2=list(TITLE="Cities represented by Legislators",
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
+            sTITLE="WA State House of Representative (2019-2021)",
+            SOURCE="Source: Washington State Legislature.")
```

2. Prepare text for tick labels. I am saving these values so I can alter the labels of ticks in each axis later.

```
> # text for tick labels
> rePOSITION_2=FTrep$Residence
> CountToShow=FTrep$Legislators
```

3. Prepare a lollipop using `FTrep` object. I will not use the `label` aesthetics here, but I will use it later.

```
> #base
> info4 = ggplot(FTrep, aes(x=Residence,
+                           y=Legislators))
> # Lollipop stick
> lol2= info4 + geom_segment(aes(y = 0,
+                                yend = Legislators,
+                                x = Residence,
+                                xend = Residence),
+                            color = "black")
> # Lollipop head
> lol2= lol2 + geom_point(size=1.5)
```

4. Add the usual classic theme, the titles and align the titles text.

```
> # theme:
> lol2 = lol2 + theme_classic()
> # titles:
> lol2 = lol2 + labs(title=texts_2$TITLE,
+                    subtitle = texts_2$sTITLE,
+                    caption = texts_2$SOURCE)
> # adjustments: alignment
> lol2 = lol2 + theme(plot.title= element_text(hjust= 0.5),
+                     plot.subtitle= element_text(hjust= 0.5),
+                     plot.caption= element_text(hjust= 0))
```

5. Here, you can make the repositioning of lollipops:

```
> # repositioning
> lol2 = lol2 + scale_x_discrete(limits=rePOSITION_2)
```

6. You have by now a lollipop that will resemble Figure 4.3. So a good improvement will be to flip the plot:

```
> #flipping
> lol2 = lol2 + coord_flip()
```

7. After the plot is flipped, be careful with horizontal and vertical controls. Most of what used to be the horizontal axis is now the vertical one, and vice versa. Let's get rid of the elements in the vertical:

```
> # Vertical axis changes
> lol2 = lol2 + theme(axis.title.y=element_blank(),
+                     axis.text.y=element_blank(),
+                     axis.ticks.y=element_blank(),
+                     axis.line.y = element_blank())
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

8. On the horizantal axis, I will customize the ticks with the `CountToShow` object I created in step 2; I will also get rid of the horizontal line, and customize the lines of the major grid (the ones for the ticks I requested):

```
> # Horizontal axis changes
> lol2 = lol2 + scale_y_discrete(limits=CountToShow)
> lol2 = lol2 + theme(axis.line.x = element_blank())
> lol2 = lol2 + theme(panel.grid.major.x =
+                          element_line(color = "grey60",
+                                       linetype = "dashed"))
```

9. I erased the tick labels because I will write the city name next to the lollipop head. Notice I will add the `label` aesthetics. I could have done this in the step 3. above, but I just wanted to show how to add an aesthetics in a particular element, instead of having the element inherit it.

```
> # annotations: text near dot
> lol2 = lol2 + geom_text(aes(label=Residence),# its own aes
+                          hjust = 0,# left justified
+                          nudge_y = 0.1,#move a little to the right
+                          size=2)
```

   Notice that `nudge_y` will move to the right instead of pushing it towards the top (what the logic of the function suggests). This is a case where flipping did not change orientation.

Figure 4.4 is a major improvement from Figure 4.3, and required some more detail than Figure 4.2. Notice I had to do some work on the text of both axes. This was needed on the horizontal axis because if you do not write the name of the city next to the dot, it will be very uncomfortable to read as a tick text on the vertical axis. I also decided to use the same approach I followed for Figure 3.31, so that the grid can serve to signal the value of each dot, and so I have room for the city name.

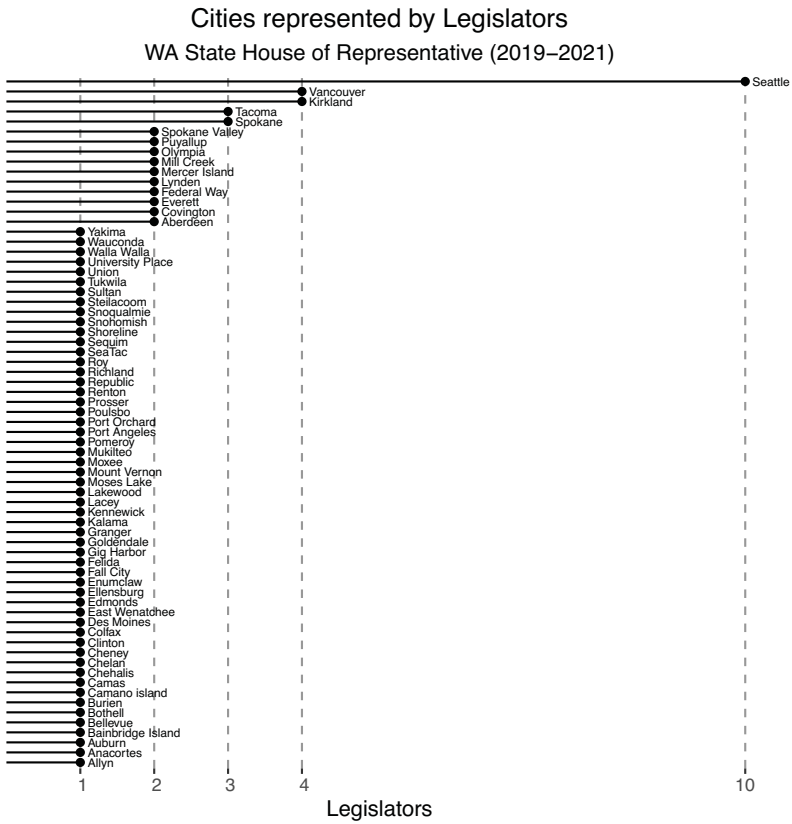
In order to re create Figure 4.4 in **Python**, I first need to create the frequency table:

```
linkRepo='https://github.com/resourcesbookvisual/data/'
linkREPS='raw/master/reps.csv'
fullLink=linkRepo+linkREPS
reps=pd.read_csv(fullLink)
# Frequency table
FTrep = pd.value_counts(reps.Residence,
                        ascending=True,
                        dropna=False).reset_index()
FTrep.columns = ['Residence','Legislators']
```

Then, this code will re-create Figure 4.4:

```
# text for titles
texts_2={'TITLE':"Cities represented by Legislators",
         'sTITLE':"WA State House of Representative (2019-2021)",
         'SOURCE':"Source: Washington State Legislature."}
# text for tick labels
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press



Source: Washington State Legislature.

Figure 4.4 Lollipop for representivity

Using Lollipop plot to show how sixty nine cities are represented. The plot has been flipped and gone through major editing.

Source: Washington House of Representatives for (66th Legislature) (Washington State Legislature, 2019).

```
rePOSITION_2=FTrep.Residence
CountToShow=FTrep.Legislators
#base
info4 = ggplot(FTrep, aes(x='Residence',
                          y='Legislators'))
# Lollipop stick
lol2= info4 + geom_segment(aes(y = 0,
                               yend = 'Legislators',
                               x = 'Residence',
                               xend = 'Residence'),
                           color = "black")
# Lollipop head
lol2= lol2 + geom_point(size=2)
# theme:
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
lol2 = lol2 + theme_classic()
# titles and aligment later in matplotlib
# repositioning
lol2 = lol2 + scale_x_discrete(limits=rePOSITION_2)
#flipping
lol2 = lol2 + coord_flip()
# Vertical axis changes
lol2 = lol2 + theme(axis_title_y=element_blank(),
                    axis_text_y=element_blank(),
                    axis_ticks_major_y = element_blank(),
                    axis_line_y = element_blank())
# Horizontal axis changes
lol2 = lol2 + scale_y_continuous(breaks=CountToShow)
lol2 = lol2 + theme(axis_line_x = element_blank())
lol2 = lol2 + theme(panel_grid_major_x =
                    element_line(color = "silver",
                                 linetype = "dashed"))
# annotations: text near dot
lol2 = lol2 + geom_text(aes(label='Residence'),
                        ha = 'left',
                        nudge_y = 0.1,
                        size=4.5)
```

As before, *matplotlib* will be needed for the titles and source:

```
import matplotlib.pyplot as plt

fig = lol2.draw()
fig.text(x=0.1,y=-0.02,s=texts_2['SOURCE'])
plt.suptitle(texts_2['TITLE'], y=0.97, fontsize=14)
plt.title(texts_2['sTITLE'], fontsize=10)
```

4.4 Categorical III: Diversity and Inequality

Frequency tables for nominal variables do not normally use cumulative values, as the order of frequency table rows can be arbitrary. However, the fact that you can order by frequency instead of alphabetic order of the categories allows you to represent how diverse or unequal a distribution can be.

This time, I will focus on the distribution of crimes in Seattle. Since inequality means that a few take most of the whole, a plot can inform us if every crime matters, or which are the main crimes. I have prepared a dataset using the Seattle Open Data Portal[2]. Let me open the file and prepare the frequency table for the variable *cat*(which I organized from the column `Crime.Subcategory`):

```
> linkRepo='https://github.com/resourcesbookvisual/data/'
> linkCRI='raw/master/crime.csv'
> fullLink=paste0(linkRepo,linkCRI)
> crime=read.csv(fullLink,stringsAsFactors = F)
> # preparing frequency table
```

[2] https://data.seattle.gov/Public-Safety/Crime-Data/4fs7-3vj5

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> library(questionr)
> #rename missing values
> crime$crimecat[is.na(crime$crimecat)]='UNcategorized'
> FTcri=freq(crime$crimecat, sort = "dec", total = F,
+            valid = F,digits = 3,cum = T)
> #result:
> FTcri
```

```
                              n      %    %cum
THEFT                    170946 34.210  34.210
CAR PROWL                142447 28.507  62.716
BURGLARY                  76630 15.335  78.052
AGGRAVATED ASSAULT        21315  4.266  82.317
NARCOTIC                  16864  3.375  85.692
ROBBERY                   16832  3.368  89.061
TRESPASS                  15919  3.186  92.246
DUI                       12205  2.442  94.689
FAMILY OFFENSE-NONVIOLENT  6601  1.321  96.010
SEX OFFENSE-OTHER          6050  1.211  97.221
WEAPON                     4751  0.951  98.171
PROSTITUTION               3555  0.711  98.883
RAPE                       1859  0.372  99.255
LIQUOR LAW VIOLATION       1619  0.324  99.579
ARSON                      1040  0.208  99.787
DISORDERLY CONDUCT          268  0.054  99.841
HOMICIDE                    267  0.053  99.894
UNcategorized               262  0.052  99.946
PORNOGRAPHY                 166  0.033  99.980
LOITERING                    85  0.017  99.997
GAMBLE                       17  0.003 100.000
```

The object `FTcri` was easily created with help from the package *questionr* (Barnier et al., 2018). I will send the row names as a data frame column:

```
> FTcri$Crimes=row.names(FTcri)
> row.names(FTcri)=NULL
```

From the frequency table you realize that the first ones from the top account for the most trouble in the city. It is important to recall that the **mode** is the value with the highest frequency; however, that does not illustrate the unequal distribution of counts. A good arithmetical strategy can be to compute the **Herfindahl-Hirschman** index (HH-index) (Woerheide, 1993):

```
> # sum of the squares of the percents:
> (HH=sum((FTcri$`%`/100)**2))
```

```
[1] 0.22801
```

When the HH-index is zero, this means we can expect an equal distribution or a very high diversity, but as soon as the index is above 0.25, you can expect an unequal distribution or less diversity. Then, from the value computed, you expect there to be clear signs of inequality, which confirms what we saw in the frequency table. I can also tell you that if you want to know *how many*

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

categories seem to stand out from the rest, you could take the inverse of the HH-index:

```
> # Inverse Simpson or Laakso-Taagepera Index
> 1/HH
```

```
[1] 4.385772
```

The value obtained suggests that there are no more than four groups that are salient from the rest. This value is a measure of *salient presence* in the group, and is known as the Inverse Simpson Index (Simpson, 1949) and also as the Laakso-Taagepera Index of Effective Number of Parties (Laakso and Taagepera, 1979). Now I have this information, let me find a way to visually represent it.

Distribution with One Vertical Axis

I will make a barplot, but I know that I should highlight four bars. Let me prepare a palette with two colors, so that the top crimes are highlighted:

```
> # palette for highlighting the top crimes
> bigCrimes=FTcri$Crimes[1:4]
> TOPS = ifelse(FTcri$Crimes %in% bigCrimes, 'black', 'grey60')
```

I have the colors that will help differentiate the four top crimes from the rest; now, let's use this and the frequency table to prepare the visual:

1. The usual first step:

```
> # base
> info5=ggplot(FTcri,aes(x=Crimes,y=`%cum`)) + theme_classic()
```

2. I will next add a reference line. I can do this as long as I know what the valid values in `%cum` are. However, my decision to have the reference at **80** is based on the Pareto Principle (Clayton, 2018; Juran, 2019) which proposes that 80 percent of an event is caused by a minority (the 20 percent). Notice that I am doing this so the bars are drawn on top of the line.

```
> # horizontal reference line
> annot1= info5 + geom_hline(yintercept = 80, linetype='dashed')
```

3. Adding the bars. Notice I am making thin bars to avoid *The Moiré Effect* (Saveljev et al., 2018). This is the first time I will use my **TOPS** palette:

```
> # creating bars, applying palette to bar border
> cumBar1= annot1 + geom_bar(stat = 'identity',
+                            fill='white',
+                            colour=TOPS, #border
+                            width = 0.2) # thining
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

4. Now, I should force the order of the bars (otherwise bars will appear in the alphabetic order of crimes):

```
> # reordering bars
> cumBar1=cumBar1+ scale_x_discrete(limits=FTcri$Crimes)
```

5. Another change of defaults, this time for the vertical axis. The intention is to see the Pareto value and other relevant values:

```
> # showing some y axis tick values
> cumBar1=cumBar1 + scale_y_continuous(breaks = c(20,50,80,100))
```

6. The last step for creating a basic plot will be to make sure the tick labels for the top crimes are different from the rest:

```
> # applying palette to text
> cumBar1=cumBar1 + theme(axis.text.x=element_text(angle = 45,
+                                                  hjust = 1,
+                                                  colour=TOPS))
```

The code above is not complete, as there are titles and other elements missing or in need of modification; also, you need to get rid of elements that do not add much (ticks, for instance). I think it would be a good exercise for you to finish the plot, which you can do with the codes given so far. The resulting visual is shown in Figure 4.5:

The **Python** version is shown next. First, the opening of the data and the preparation of the frequency table:

```
import pandas as pd

#link to data
linkRepo='https://github.com/resourcesbookvisual/data/'
linkFile='raw/master/crime.csv'
fullLink=linkRepo+linkFile
crime=pd.read_csv(fullLink)

# Frequency table
FTcri = pd.value_counts(crime.cat,
                        ascending=False,
                        dropna=False).reset_index()
FTcri.columns = ['Crimes','Counts']
```

Then, I add the cumulative percent and rename the missing value. Notice that `cumsum` and `sum` are *Pandas* functions that are applied to each column.

```
# adding column
FTcri['CumPercent']=100*FTcri.Counts.cumsum()/FTcri.Counts.sum()
# renaming missing values
FTcri['Crimes'].fillna('UNCATEGORIZED', inplace=True)
```

Here, I prepare the palette. I first prepared a condition using the `isin` function from *Pandas*, and then used the function `where` from **numpy**:

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

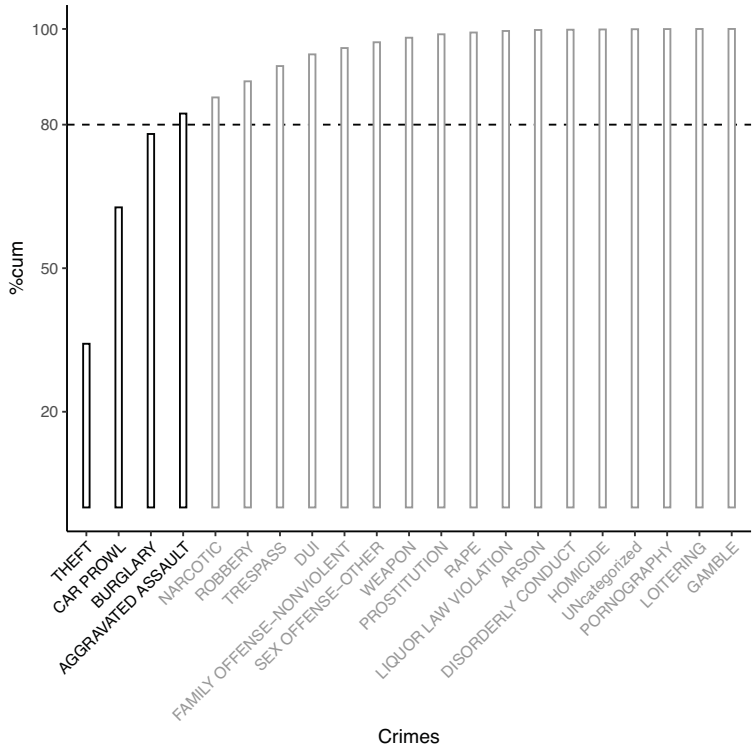


Figure 4.5 Cumulative bars for diversity

Using cumulative bars to show how few crimes are the biggest concern in Seattle. Data from Seattle Open Data Portal (City of Seattle, 2019).

```
import numpy as np

condition=FTcri.Crimes.isin(FTcri.Crimes[0:4])
TOPS =tuple(np.where(condition, 'black', 'silver'))
```

Then, I follow the same steps I took for creating Figure 4.5. The coding strategy is almost the same.

```
from plotnine import *

#1
info5=ggplot(FTcri,aes(x='Crimes',y='CumPercent')) + theme_classic()
#2
annot1=info5 + geom_hline(yintercept = 80, linetype='dashed')
#3
cumBar1=annot1 + geom_bar(stat = 'identity',
                          fill='white',
                          color=TOPS,
                          width = 0.2)
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
#4
cumBar1=cumBar1+ scale_x_discrete(limits=FTcri.Crimes)
#5
cumBar1=cumBar1+ scale_y_continuous(breaks = (20,50,80,100))
#6a
cumBar1=cumBar1+ theme(axis_text_x=element_text(rotation=45,
                                                ha='right'))
```

The code above did not use the palette `PLOTS` for the tick labels, as Plotnine does not have that functionality. So, I have to use *matplotlib* again. In this case, after I get the whole figure into *matplotlib*, I get the plotting area (`ax`) using `subplot` to recover each tick label to color it.

```
# 6b
import matplotlib.pyplot as plt

fig = cumBar1.draw()

ax=plt.subplot() # the plot area
# coloring each label
for aTick, aColor in zip(ax.get_xticklabels(), TOPS):
    aTick.set_color(aColor)
```

Distribution with Two Vertical Axes

Using two vertical axis is not always a good idea, as each element in the plot will have two measurements. However, if the plot is familiar in your community, it is worth producing it. The *Pareto Chart* is a good choice for categorical data. It is a modified barplot that depicts the group of categories that represent the biggest concern for the organization (Wilkinson, 2006). The plot uses bars in decreasing order, and a line element to represent the cumulative percent.

Let me first prepare a *Pareto Chart* using R. It can be in fact a long process but using the library **ggQC** (Grey, 2018) it can be very easy:

```
> library(ggQC)
> info6=ggplot(FTcri, aes(x=Crimes, y=n)) + theme_classic()
> paret=info6 + stat_pareto()
```

I will not show the result from the code above, but I recommend you do this. When you do, you will realize there is still a lot of work to be done. Fortunately, the library *ggQC* is easily integrated with *ggplot*. Let me now offer you one version, not necessarily complete, for our case:

```
> #base
> info6=ggplot(FTcri, aes(x=Crimes, y=n)) + theme_classic()
> #horizontal reference line at  80%
> info6=info6 + geom_hline(yintercept = sum(FTcri$n)*0.80,
+                          linetype = "dashed",color='grey90')
> #vertical reference line at  4th bar
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> info6=info6 + geom_vline(xintercept = 4,
+                          linetype = "dashed",color='grey90')
> #order of the bars
> info6=info6 + scale_x_discrete(limits=FTcri$Crimes)
> #angle for x ticks labels, to ease visibility
> info6=info6 + theme(axis.text.x =element_text(angle=45,hjust = 1))
> #add the Pareto, but shringk the dots, and recolor bar
> paret=info6 + stat_pareto(point.size = 0.5,bars.fill = "grey")
```

The previous code will produce a nice plot, but the default values of the secondary axis values are not the ones I want. So, I will change them with some extra manipulation. For that, I simply type `stat_pareto` in the **R** console to identify its default configuration in order to know how to change the name and the breaks of the secondary axis. Notice that the primary y-axis will produce, by default, labels with a scientific format if the values are big. Add this layer to accomplish the last changes:

```
> paret=paret +
+       scale_y_continuous(
+         sec.axis=sec_axis(trans = ~./(max(.)*0.95)*100,
+                           name="Cumulative %",
+                           breaks=c(20,50,80,100)),
+         labels=function(x) format(x,scientific=F))
```

The result of the `paret` object can be seen in Figure 4.6. There are elements missing, but we have the tools to finish it, by adding more layers using the *ggplot* functions. I also recommend altering the order of the layers in case you want to see what should or should not be moved.

Python can produce a visual similar to Figure 4.6 with the help of the *Pareto Chart* library, created by Lee (2013), and modified by Jesus (2019) (please install it before running the code). The basic functionality is limited, so I have to use *matplotlib* to add the reference lines. Notice that the vertical line value was very easy to produce, compared to the computation needed in *ggplot*.

```
from paretochart import pareto

#using pareto
ax=pareto(FTcri.Counts, FTcri.Crimes,
       line_args=('k'), #'k' is 'black'
       data_kw={'color': 'grey'})

#reference lines using matplotlib
plt.axvline(x=3,ls='--',c='silver',lw=1)
plt.axhline(y=80,ls='--',c='silver',lw=1)

#modifying tick labels
plt.setp(ax[1].get_xticklabels(), ha="right",rotation=45)
plt.show()
```

Notice in the code above that I saved the *Pareto Chart* in `ax` (a tuple with three elements), then I added layers to the plot using `plt`. Finally, the ticks of the horizontal are modified (`ax[1]`).

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

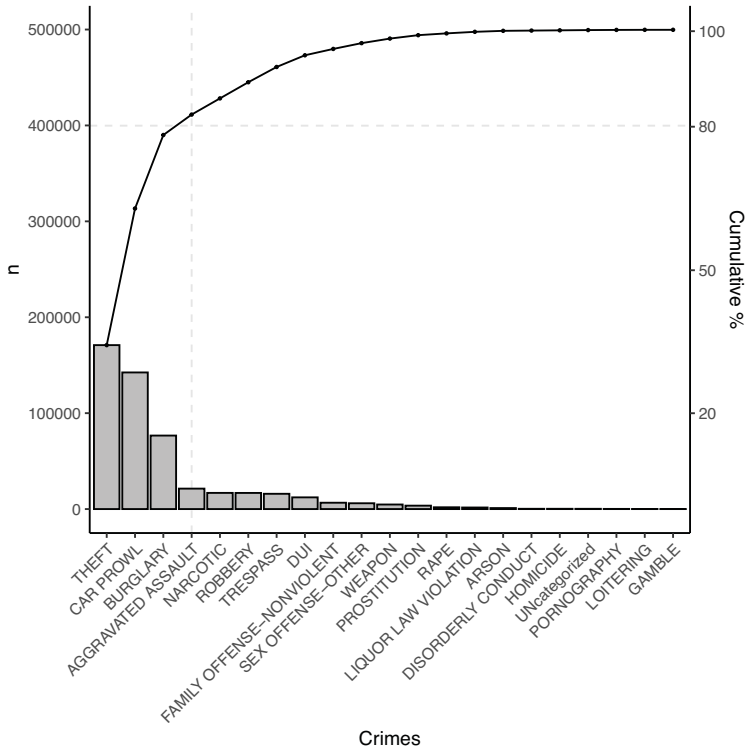


Figure 4.6 *Pareto Chart*

The intersection between the 80 percent value of the secondary axis, and the fourth bar is shown. The fourth bar was selected based on the Inverse Simpson Index (Simpson, 1949). Data from Seattle Open Data Portal (City of Seattle, 2019).

4.5 Categorical IV: Symmetry and Position

The previous examples used nominal data. In those cases, bars were reordered because they did not need to respect any sequence. In ordinal data, you could reorder, but since these data do have order, you should consider highlighting their own properties. For example, take a look at the categories present in the variable `High.Grade` from the data frame `eduwa`.

```
> table(eduwa$High.Grade,useNA = "ifany")
```

```
  1  10  11  12  13   2   3   4   5   6   7   8   9  KG  PK
  6   7   5 757   9  16  19  45 755 266  11 427  15   7  82
```

This represents the highest grade offered per school. However, ordinal data usually needs to be formatted, as most programs will have recognized it as

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

text or nominal by default. This is the case with this variable. Let's make the change with the command `ordered`:

```
> #get levels
> levelsHG=names(table(eduwa$High.Grade))
> #reorder levels
> ordLabels=c(levelsHG[c(15,14)],
+   sort(as.numeric(levelsHG[c(1:13)])))
> #apply that to the column
> eduwa$High.Grade=ordered(eduwa$High.Grade,levels=ordLabels)
```

The process to format ordinal data requires a little more work in **Python** than in **R** (five steps instead of three)[3]:

```
from pandas.api.types import CategoricalDtype

# get the levels
levels=eduwa['High.Grade'].value_counts().index.sort_values()
#reorder levels
ordLabels=levels[-2:].tolist()[::-1]+sorted(map(int,levels[:-2]))
# turn previous result into a list of strings
ordLabels=list(map(str,ordLabels))
# use that list of levels to create ordinal format
HGtype= CategoricalDtype(categories=ordLabels,ordered=True)
# apply that format to the column
eduwa['High.Grade']= eduwa['High.Grade'].astype(HGtype)
```

Since I have an ordinal column, I should consider that I can highlight:

- Position. Order gives you position, then you can speak in terms of "greater than," "less than," and the like. So you can try computing quartiles (which include the median). When you explore positions, you may find **outliers** – unusual values that are very far from the rest. These are not values to be disregarded, but values whose presence you should try to understand.
- Symmetry. If you have a middle value, you can see how similar the distribution to left of the middle is to the distribution to the right.

A key statistic in ordinal data is the median, which can be easily identified in the frequency table using the cumulative percent (the category that accumulates half or less of the counts):

```
> library(questionr)
> FThg=freq(eduwa$High.Grade, total=F,valid=F,digits=3,cum=T)
> # some changes to FThg:
> FThg$MaxOffer=row.names(FThg)
> row.names(FThg)=NULL
> FThg=FThg[c(4,1:3)]
> names(FThg)[2:4]=c("Counts","Percent","CumPercent")
> FThg
```

[3] The third step in **Python** was done by default in **R** the second step, and the fourth step in **Python** is included in the last step in R.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
   MaxOffer Counts Percent CumPercent
1        PK     82   3.379      3.379
2        KG      7   0.288      3.667
3         1      6   0.247      3.914
4         2     16   0.659      4.574
5         3     19   0.783      5.356
6         4     45   1.854      7.211
7         5    755  31.108     38.319
8         6    266  10.960     49.279
9         7     11   0.453     49.732
10        8    427  17.594     67.326
11        9     15   0.618     67.944
12       10      7   0.288     68.232
13       11      5   0.206     68.438
14       12    757  31.191     99.629
15       13      9   0.371    100.000
```

You can get the same table (without the percent column) using Python:

```
# Frequency table
FThg = pd.value_counts(eduwa['High.Grade'],
                       ascending=False,sort=False,
                       dropna=False).reset_index()
FThg.columns = ['MaxOffer','Counts']
# adding column
FThg['CumPercent']=100*FThg.Counts.cumsum()/FThg.Counts.sum()
```

From the frequency table, you know:

- The first quartile: 25 percent of the public Schools offer at most 5th grade.
- The median or secon quartile: 50 percent of the public Schools offer at most 8th grade.
- The third quartile: 75 percent of the public Schools offer at most 12th grade (25 percent of the schools offer at least 12th grade).

The median and the other positional values do not belong to the nominal data statistics realm, but all the statistics from nominal data can be used in ordinal ones. Let me get the median of this ordinal in **R** using function **Median** from the package **DescTools**[4] by Signorell et al. (2019):

```
> library(DescTools)
> medianHG=as.vector(Median(eduwa$High.Grade))
> # then
> medianHG
```

```
[1] "8"
```

You need to prepare a function in **R** to compute the other quartiles.[5] Since **Python** does not have a median function for ordinal data, let me give an example on how to prepare one to get all quartiles:

[4] The function **median** in **R** only works with numeric data.
[5] *DescTools* can not compute other positional values for ordinals.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
# function for quartiles in ordinal data
def Quart_Pos(cumFT,q=1): # q can be 1,2 or 3.
    position=0
    for percent in cumFT:
        if percent ≤ 25*q: position +=1
        else: break
    return position # returns a position

# applying function
medianHG=FThg.MaxOffer[Quart_Pos(FThg.CumPercent,2)]
```

My **Python** function gives you back the position of the quartile requested (**2** gives me the median). So, I used that value to get the grade that occupies that position.

Having the median will allow me to explore the ordinal characteristics. Let me first make a barplot with the median highlighted:

```
> # color to highlight median
> colCondition=ifelse(ordLabels==medianHG,'black','grey')
> #
> # usual
> info7=ggplot(FThg, aes(MaxOffer,Counts)) + theme_classic()
> barFThg=info7 + geom_bar(stat='identity',
+                          fill=colCondition)
> barFThg=barFThg + scale_x_discrete(limits=ordLabels)
```

As usual, the **Python** version using *plotnine* is almost identical:

```
# color to highlight median
condition=[medianHG==test for test in ordLabels]
colCondition=tuple(np.where(condition, 'black', 'silver'))

# usual
info7=ggplot(FThg, aes('MaxOffer','Counts')) + theme_classic()
barFThg=info7 + geom_bar(stat='identity',fill=colCondition)
barFThg=barFThg + scale_x_discrete(limits=ordLabels)
```

Figure 4.7 renders the object `barFThg`. Can you find signs of symmetry in that plot?

You can clearly see the lack of symmetry: the distribution to the left of the median (`'Grade 8'`) is not mirroring the distribution to the right. Also, from the frequency table `FThg`, we know the big grey bars are the first and third quartiles. However, you can not directly identify positional values in barplots without the support of frequency tables.

Your next option is the **boxplot** (Spear, 1969; Tukey, 1977). Boxplots have the advantage of not requiring a frequency table as input; however, they are not that common to understand if you are not familiar with quartiles. Also, **R** and **Python** require a numeric version of the variable. So, let me start by creating a numeric version of the ordinal column:

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

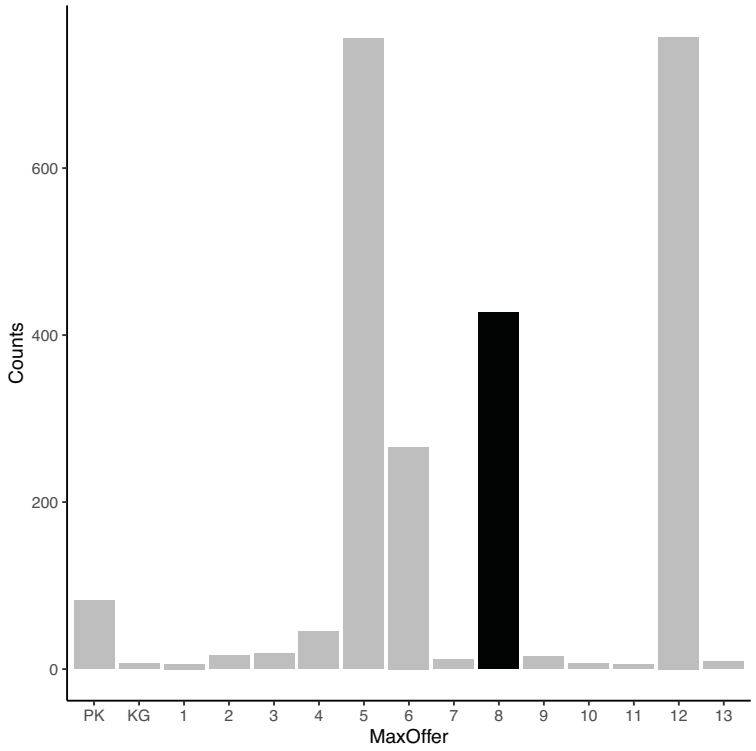


Figure 4.7 Ordinal barplot

The median is highlighted.

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

```
> # from ordinal to numeric
> eduwa$High.Grade.Num=as.numeric(eduwa$High.Grade)
```

You can now create a boxplot with this new variable. The result will be shown in Figure 4.8.

```
> info8=ggplot(eduwa,aes(x=0,y=High.Grade.Num)) + theme_classic()
> boxHG =info8 + geom_boxplot() + coord_flip()
> boxHG = boxHG + scale_y_continuous(labels=ordLabels,breaks=1:15)
```

This will be the **Python** version to get Figure 4.8:

```
# from ordinal to numeric
eduwa['High.Grade.Num']=eduwa['High.Grade'].cat.codes

info8=ggplot(eduwa,aes(x=0,y='High.Grade.Num')) + theme_classic()
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

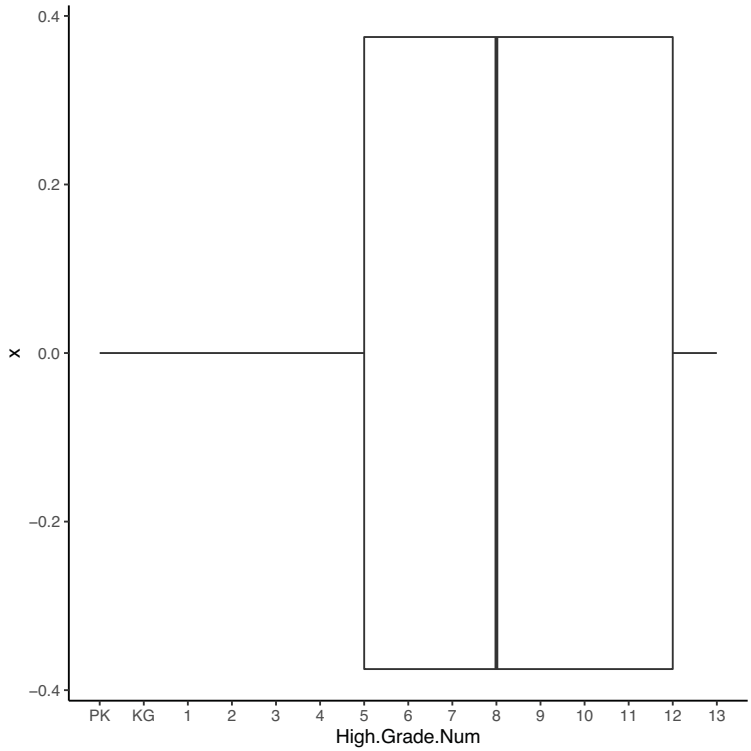


Figure 4.8 Boxplot

The median is highlighted.

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

```
boxHG =info8+ geom_boxplot()
boxHG =boxHG + coord_flip()
boxHG = boxHG + scale_y_continuous(labels=ordLabels,
                                   breaks=list(range(0, 15)))
```

Boxplots can confirm that you have symmetry when the distance of each *whisker* to the *box* is the same, and when the thick line (the median) is in the middle of the box. Since you have a *left tail*, you can say this variable has *negative symmetry*.

Boxplots do not replace barplots. The boxplot is hiding the peaks you can see in bars. However, you can reveal that if you combine the boxplot with a **violin plot**:

```
> info8=ggplot(eduwa,aes(x=0,y=High.Grade.Num)) + theme_classic()
> # first the violin:
> vio1 =info8 + geom_violin(width=1.4, # play with this value
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
+                              fill="black", color=NA)
> #now the bxplot
> boxHG2 =vio1 + geom_boxplot(width=0.2,# play with this value
+                               fill='white',
+                               color='grey',
+                               fatten=4) #thicker median
> # flipping
> boxHG2 = boxHG2 + coord_flip()
> # right order of tick labels
> boxHG2 = boxHG2 + scale_y_continuous(labels=ordLabels,
+                                      breaks=1:15)
> # erase unneeded elements.
> boxHG2 = boxHG2 + theme(axis.ticks = element_blank(),
+                         axis.text.y = element_blank(),
+                         axis.title.y = element_blank(),
+                         axis.line.y  = element_blank())
```

The **Python** version for the Violin plot:

```
theBreaks=list(range(0, 15))

info8=ggplot(eduwa,aes(x=0,y='High.Grade.Num')) + theme_classic()
vio1 =info8 + geom_violin(width=1.4,
                          fill="black", color=None)
boxHG2 = vio1 + geom_boxplot(width=0.2,
                             fill='white',
                             color='silver',
                             fatten=4)
boxHG2 = boxHG2 + coord_flip()
boxHG2 = boxHG2 + scale_y_continuous(labels=ordLabels,
                                     breaks=theBreaks)
boxHG2 = boxHG2 + theme(axis_ticks = element_blank(),
                        axis_text_y = element_blank(),
                        axis_title_y = element_blank(),
                        axis_line_y  = element_blank())
```

Then, the object `boxHG2` is represented in Figure 4.9.

Figures 4.8 and 4.9 have a negative asymmetry, but there is no presence of outliers. If there were, the boxplot would identify them with dots. In future visuals, we may see some outliers. Those boxplots are also a good example of the difference between `labels` and `limits` of an axis. Previously, I could customize the axis just by indicating the values I wanted, but in all those cases we just dealt with numeric values. In this case, I needed to indicate the numeric values and the labels each tick will have. Also, pay attention to the data structure needed for each argument. In R, `labels` and `breaks`, use *vectors*; while in **Python**, both arguments need *lists*.

4.6 Numerical I: Dispersion

Bars or similar elements have allowed you to see the dispersion of a variable; however, I have not paid so much attention to this property in nominal or ordinal variables as generally they do not take that many values, while

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

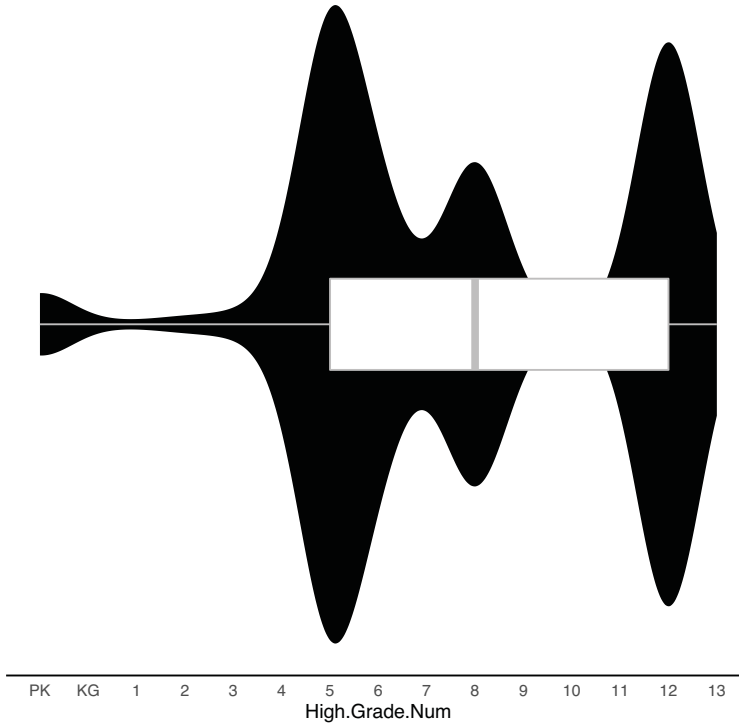


Figure 4.9 Box and Violin plot combined

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

numerical do. Let me use a barplot to show you the dispersion in the variable *Reduced.Lunch*, which informs how many kids there are in each school that have lunch for a reduced price, simply using a barplot:

```
> info9= ggplot(eduwa,aes(x = Reduced.Lunch)) + theme_classic()
> disp1= info9 + geom_bar()
```

Figure 4.10 has 172 bars, as there are that many different values, a lot more than what you find in categorical data. This prevents bars from being an option; so we need to find an alternative. Keeping in mind that when your purpose is to inform about dispersion, it is because you want to show that:

- most values are well represented by central values (median / mean). A boxplot can help, as its central box is showing the dispersion in the neighborhood of the median;

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

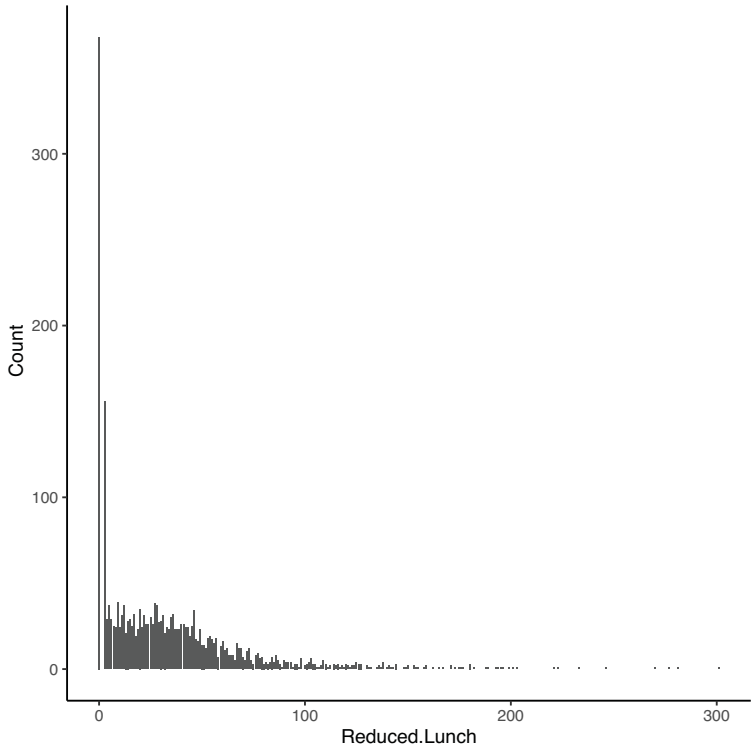


Figure 4.10 Dispersion using bars

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

- the dispersion pattern allows for outliers. A boxplot will automatically illustrate of that;
- the dispersion around the central values (median/mean).

You can see from what I have just mentioned that a first option can be a **boxplot**. However, before plotting, I recommend you get information on your data:

- Get the basic statistics:

```
> (statVals=summary(eduwa$Reduced.Lunch)) # just values
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00    5.00   25.50   33.53   47.00  301.00     131
```

```
> # In R, you need to get the standard deviation (NOT in Python):
> statVals['std']=sd(eduwa$Reduced.Lunch,na.rm = T)
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

When readers know the median and the mean, most will believe these are representative values of the population. You need to show how representative these actually are by showing the dispersion of the data. Notice that the command `summary` produced a *table* structure, which produces "named" values (there is a name for each value); this is how I added the standard deviation to `statVals`, which is a measure of dispersion relative to the mean.

- Turn the basic statistics into values you can use later:

```
> (statVals=ceiling(statVals))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.     std    NA's
      0       5      26      34      47     301      37     131
```

The function `ceiling` will round up the values. This is an optional step, which I may use for annotating.

- Thresholds to detect outliers:
  - Distance between quartiles (the measure of dispersion for the median):

```
> IQR=statVals['3rd Qu.']-statVals['1st Qu.']
```

  - Compute the upper and lower thresholds to detect outliers:

```
> statVals['upper']=1.5*IQR + statVals['3rd Qu.']
> statVals['lower']=statVals['1st Qu.']-1.5*IQR
```

Any value above or below will be considered an outlier.

- Prepare annotations for the plot:
  - Values for axis:

```
> # no 'lower' this time
> axisKeys=c('Min.','1st Qu.','Median','Mean',
+            '3rd Qu.','upper','Max.')
> myTicks=as.vector(statVals[axisKeys])
```

The object `myTicks` is a vector, a structure needed for the axis in *ggplot*.

  - Compute the share of values considered as outliers:

```
> # Share of values considered outliers:
> theVariable=eduwa$Reduced.Lunch
> theVariable = theVariable[!is.na(theVariable)]
> countOutliersUp=sum(theVariable>statVals['upper'])
> shareOut=ceiling(countOutliersUp/length(theVariable)*100)
> # message using the value computed:
> labelOutliers=paste0("Outliers:\n", shareOut,"% of data")
```

The **Python** code for the previous computations is shown next:

```
# get stats as DICT: count, mean,std,min,q1,q2,q3,max
statVals=eduwa['Reduced.Lunch'].describe().to_dict()

# Turn into DICT of integers (rounding up)
from math import ceil
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
statVals={key: ceil(val) for key, val in statVals.items()}

# Thresholds to detect outliers:

## distance between quartiles
IQR=statVals['75%']-statVals['25%']
## Thresholds:
statVals['upper']=1.5*IQR + statVals['75%']
statVals['lower']=statVals['25%'] - 1.5*IQR

#prepare annotations:
axisKeys=['min','25%','50%','mean','75%','upper','max']
myTicks = {axKey: statVals[axKey] for axKey in axisKeys}

# Share of values considered outliers:
theVariable=eduwa['Reduced.Lunch']
theVariable = theVariable.dropna()
countOutliersUp=sum(theVariable>statVals['upper'])
shareOut=ceil(countOutliersUp/len(theVariable)*100)
# message using the value computed:
labelOutliers="Outliers:\n" + str(shareOut) + "% of data"
```

Next, let me produce an annotated boxplot in R:

```
> # x=0, 0 is the position of line that goes accross the boxplot
> info10= ggplot(eduwa,aes(x=0,y=Reduced.Lunch)) + theme_classic()
> #
> # Changing defaults
> ## axis 'breaks'
> info10= info10 + scale_y_continuous(breaks=myTicks)
> #
> ## this is the x-axis limits, useful for annotation positions
> info10= info10 + xlim(c(-0.25,0.3)) + coord_flip()
> #
> # changing width of boxplot
> disp2= info10 + geom_boxplot(width=0.25,outlier.alpha = 0.2)
> #
> # ANNOTATING
> ## Standard deviation:
> ## this is a segment showing one standard deviation interval
> disp2=disp2 + annotate("pointrange",
+                        x=0.15, y=statVals['Mean'],
+                        ymin = (statVals['Mean']+5)-statVals['std'],
+                        ymax = (statVals['Mean']+5)+statVals['std'],
+                        colour = "gray80", size = 1)
> ## mean
> ### the line
> disp2=disp2 + geom_hline(yintercept = statVals['Mean'],
+                          linetype='dotted')
> #
> ### the text: notice I add '5', to move text
> disp2=disp2 + annotate(geom="text", fill='white',
+                        x=0.2, y=statVals['Mean']+5,
+                        label="Mean",angle=90,size=3)
> ## median
> ### the line
> disp2=disp2 + geom_hline(yintercept = statVals['Median'],
+                          linetype='dotted')
> #
> ### the text: notice I substract '5', to move text
> disp2=disp2 + annotate(geom="text",
+                        x=-0.2, y=statVals['Median']-5,
+                        label="Median",angle=90,size=3)
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> ## outliers
> ### the line
> disp2=disp2 + geom_hline(yintercept = statVals['upper'],
+                          linetype='dashed',color='grey50')
> #
> ### the text
> disp2=disp2 + annotate(geom="label",
+                        x=0.1, y=statVals['Max.'],
+                        label=labelOutliers,size=5,hjust=1,
+                        color='grey50')
> # erasing
> disp2=disp2 + theme(axis.ticks.y = element_blank(),
+                     axis.line.y = element_blank(),
+                     axis.text.y = element_blank(),
+                     axis.title.y = element_blank())
```

The object `disp2` is shown in Figure 4.11. This plot includes statistical information which is appropriate in case your audience has some basic training. You are clearly showing that the mean and the median are not at the center of the distribution, and that the amount of outliers is probably worth considering.

You can see the **Python** version of Figure 4.11 below. Notice that I am using dictionaries in Python; I have done that before to use *keys* instead of indexes (numbers). I followed a similar approach in **R** with the table structure. This is optional, but I think it improves not only readability but it gets easier to remember:

```
info10= ggplot(eduwa,aes(x=0,y = 'Reduced.Lunch')) + theme_classic()
info10=info10 + scale_y_continuous(breaks =list(myTicks.values()))
info10=info10 +xlim(-0.25,0.3) +coord_flip()
disp2= info10 + geom_boxplot(width=0.25,outlier_alpha = 0.2)

# annotating
## standard deviation
disp2=disp2 + annotate("pointrange",
                       x=0.15, y=statVals['mean'],
                       ymin = (statVals['mean']+5)-statVals['std'],
                       ymax = (statVals['mean']+5)+statVals['std'],
                       colour = "silver", size = 1)
## mean
disp2=disp2 + geom_hline(yintercept = statVals['mean'],
                         linetype='dotted')
disp2= disp2 + annotate(geom="text",
                        x=0.2, y=statVals['mean']+5,
                        label="Mean",angle=90,size=10)
## median
disp2=disp2 + geom_hline(yintercept = statVals['50%'],
                         linetype='dotted')
disp2= disp2 + annotate(geom="text",
                        x=-0.2, y=statVals['50%']-5,
                        label="Median",angle=90,size=10)

## outliers
disp2=disp2 + geom_hline(yintercept = statVals['upper'],
                         linetype='dashed',color='silver')
disp2=disp2 + annotate(geom="label",
                       x=0.1,y=statVals['max'],
                       label=labelOutliers,size=12,ha='right',
                       color='silver')
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

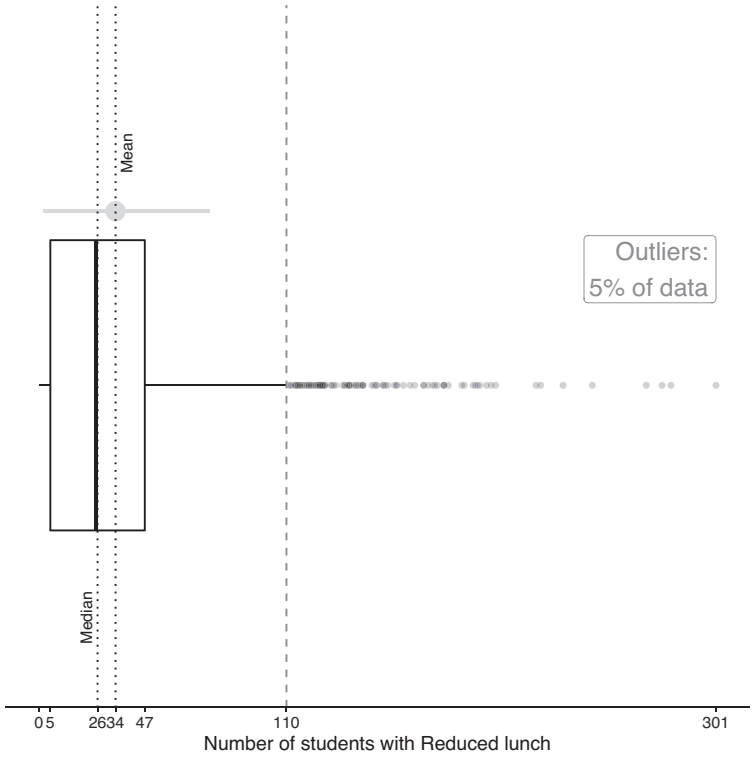


Figure 4.11 Dispersion using a boxplot

Data Source: The Common Core of Data from the US Department of Education, available at `https://nces.ed.gov/ccd/`.

```
# erasing
disp2=disp2 + theme(axis_ticks_major_y = element_blank(),
                    axis_line_y = element_blank(),
                    axis_text_y = element_blank(),
                    axis_title_y = element_blank())
```

Keep in mind that the features we have been highlighting in the previous subsections can all be applied to numeric data, but dispersion and shape feature, coming next, are more natural to numeric data.

4.7 Numerical II: Shape

Shape and dispersion combined allow for a comprehensive analysis of numeric data. Shape of data distribution can resemble several well-known statistical distributions. However, most of the time you will be interpreting data comparing

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

them to a distribution known as the *normal distribution*. A normal distribution is bell-shaped and symmetrical, and, in that situation, its mean and median are the same. However, even if symmetry is accomplished, you need to find out if it has an interesting relationship between its mean (M) and its standard deviation (STD):

- Approximately 68 percent of the data fall within 1 STD of the M.
- Approximately 95 percent of the data fall within 2 STDs of the M.
- Approximately 99.7 percent of the data fall within 3 STDs of the M.

When symmetry is satisfied, but the above conditions are not met, you explore the *kurtosis*: if the bell is too flat, the kurtosis becomes negative and you have a platycurtik shape; if the bell is thin, you have a leptokurtic shape. The normal distribution is mesokurtic.

Boxplots do not show shape very well; so, you should use the **histogram** for this feature. The histogram is very similar to a barplot, as the height of the bars also represents a count or a percent; however, the bars in a histogram are contiguous, and each bar width represents an interval of numbers, instead of a category.

Next, let me explore the shape of the variable on reduced lunch. I have to complete two basic steps:

a) Compute the input elements to create a frequency table of the data intervals:

1. Decide the starting value and the width of the interval. In our case it will be `10`.

```
> theStart=statVals['Min.']
> width=10
```

2. Since the maximum value of the data may not be a multiple of the width selected, I will need to select the smallest multiple of `10` as the new maximum value.

```
> oldMax=statVals['Max.']
> newMax=ifelse(oldMax%%width<width,
+               oldMax+(width-oldMax%%width),oldMax)
```

3. With the previous information, I will produce all the interval breaks.

```
> TheBreaks=seq(theStart,newMax,width)
```

4. The object `TheBreaks` will be used in the histogram. You can cut the variable with `TheBreaks` to produce a frequency table, but I only need the maximum frequency:

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
> intervals=cut(eduwa$Reduced.Lunch,
+               breaks=TheBreaks,include.lowest = T)
> topCount=max(table(intervals))
```

I need the `topCount` to determine the values on the vertical axis.

5. Compute the vertical axis values:

```
> widthY=50
> top_Y=ifelse(topCount%%widthY<widthY,
+              topCount+widthY-topCount%%widthY,topCount)
> vertiVals=seq(0,top_Y,widthY)
```

The **Python** code that computes the same is shown next:

```
#1
theStart=statVals['min']
width=10
#2
oldMax=statVals['max']
reminderMax=oldMax%width
newMax= oldMax+(width-reminderMax) if reminderMax<width else oldMax
#3
TheBreaks=list(range(theStart,newMax+width,width))
#4
intervals=pd.cut(eduwa['Reduced.Lunch'],
              bins=TheBreaks,include_lowest=True)
topCount=intervals.value_counts().max()
#5
widthY=50
reminderY=topCount%widthY
top_Y=topCount+widthY-reminderY if reminderY<widthY else topCount
vertiVals=list(range(0,top_Y+widthY,widthY))
```

a) Compute the normal curve for the data[6]. This will let you know how far the actual shape is from the normal one. These are the steps:

1. Set the input parameters:

```
> N = sum(!is.na(eduwa$Reduced.Lunch)) # number of elements
> MEAN = as.vector(statVals['Mean'])
> STD = as.vector(statVals['std'])
```

2. Create the function, modifying output to resemble counts:

```
> NormalHist=function(x) dnorm(x,mean =MEAN,sd =STD)*N*width
```

The previous function looks like this in **Python**:

```
#1
N = statVals['count']
MEAN = statVals['mean']
STD = statVals['std']
#2
```

[6] The function for the normal curve requires the input of mean and the standard deviation.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
def NormalHist(x,m=MEAN,s=STD,n=N,w=width):
    import scipy.stats as stats
    return stats.norm.pdf(x, m, s)*n*w
```

Then, I can create my histogram like this:

```
> info11= ggplot(eduwa, aes(x = Reduced.Lunch)) +
+         theme_classic() +
+         ylab("Number of students with Reduced lunch")
> # the histogram
> disp3= info11 + geom_histogram(binwidth = width,
+                                #start of first bar
+                                boundary=theStart,
+                                fill='white',color='grey60')
> # the normal curve
> disp3= disp3 + stat_function(fun = NormalHist,
+                  color = "black", size = 1,linetype='dashed')
> # the vertical axis values
> disp3= disp3 + scale_y_continuous(breaks = vertiVals)
```

The result can be seen in Figure 4.12.

The **Python** version for Figure 4.12 is:

```
info11= ggplot(eduwa, aes(x = 'Reduced.Lunch')) + theme_classic()
disp3= info11 + geom_histogram(binwidth = width,
                               boundary=theStart,
                               fill='white',color='silver')
disp3= disp3 + stat_function(fun = NormalHist,
                 color = "black", size = 1,linetype='dashed')

disp3= disp3 + scale_y_continuous(breaks = vertiVals)
```

4.8 Numerical III: Numerical Inequality

Some people would like to know how much a particular share of schools are receiving a particular share of benefits. In situations like this, some people want to see a **Lorenz Curve** (Lorenz, 1905). This curve is always compared to a diagonal: the closer to the diagonal, the more equal the distribution of the variable. Most of the time, the curve is accompanied by the **Gini Index** (Gini, 1912, 1921).

Plotting a Lorenz curve and computing the Gini index can be done by learning particular formulas. You can save some time when producing a Lorenz curve by using the library *gglorenz* (Chen, 2018) in R, which combines perfectly with *ggplot*; and you can also get the Gini Index using the library *DescTools* (Signorell et al., 2019), previously introduced, which has a function **Gini**.

```
> library(gglorenz)
> #for titles
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

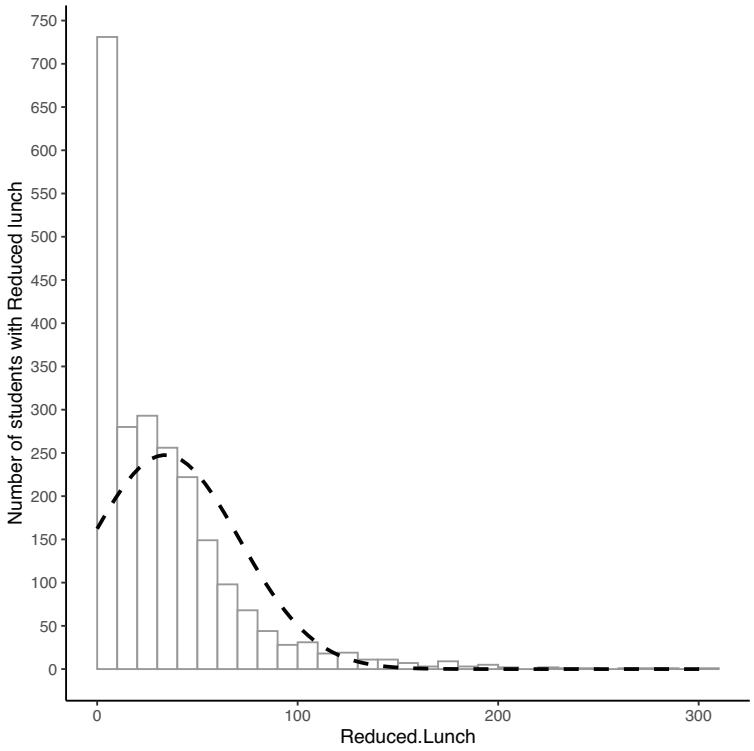


Figure 4.12 Shape using a histogram and normal curve

The normal curve has been created using the mean and standard deviation of the variable.

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

```
> HorizontalTitle="Percent of Schools by benefit received"
> VerticalTitle="Cummulative percent of benefit"
> plotTitle="How much receives\nthe 20% that receives most?"
> sourceText="Source: US Department of Education"
> # text for annotation
> ## computing
> gini=Gini(eduwa$Reduced.Lunch,na.rm = T)
> ## pasting message
> GINItext=paste("Gini:",round(gini,3))
> # plot Lorenz curve
> lorenz=info11 + stat_lorenz()
> #
> # diagonal
> lorenz= lorenz + geom_abline(linetype = "dashed")
> #
> # annotations
> ## vertical and horizontal lines
> lorenz= lorenz + geom_vline(xintercept = 0.8,
+                             color='grey80',
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
+                              lty='dotted') +
+                   geom_hline(yintercept = 0.5,
+                              color='grey80',
+                              lty='dotted')
> #
> # changing default axis tick values, positions and aspect
> lorenz= lorenz + scale_y_continuous(breaks = c(0,0.5,0.8),
+                                     position = 'right') + #position
+                   scale_x_continuous(breaks = c(0,0.5,0.8)) +
+                   coord_fixed() #aspect
> # annotating: adding Gini Index value
> lorenz= lorenz + annotate(geom="text",
+                           x=0.4, y=0.25,size=3,
+                           label=GINItext)
> # texts
> lorenz= lorenz +  labs(x = HorizontalTitle,
+                        y = VerticalTitle,
+                        title = plotTitle,
+                        caption = sourceText)
```

The object `lorenz` is represented in Figure 4.13.

You should have realized by now that for the categorical case I, represented inequality using *Pareto Charts* plus the Herfindahl-Hirschman Index, and for the numerical (or continuous case), I have used the Lorenz Curve and the Gini Index.

You can also prepare a **Python** version of Figure 4.13. However, this time I can not use *plotnine* (unless I want to implement it from scratch or adapt the code from `stat_lorenz()`). I recommend the use of the **ineqpy** library. Once you have installed it,[7] you need first to create a data frame where one column is the variable and another one is the counts of that variable (like a frequency table). However, our variable is not a frequency table, so I just create a column of *ones* instead:

```
import ineqpy as ineq

#new data frame (one column)
dfTest=pd.DataFrame(eduwa['Reduced.Lunch'].dropna())
## add a columns, where each value is number '1':
dfTest['count']=np.ones(dfTest.size)
```

The `dfTest` data frame will now be converted into a *survey* object:

```
# data frame to survey object
dfIneq = ineq.api.Survey(dfTest, weights='count')
```

The function from *ineqpy* can now be applied to the object **dfIneq** to create a **Python** version of Figure 4.13:

```
#for titles
txtLz={'HorizontalTitle':"Percent of Schools by benefit received",
       'VerticalTitle':"Cummulative percent of benefit",
```

[7] Use: pip install git+https://github.com/mmngreco/IneqPy.gittoinstallit.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

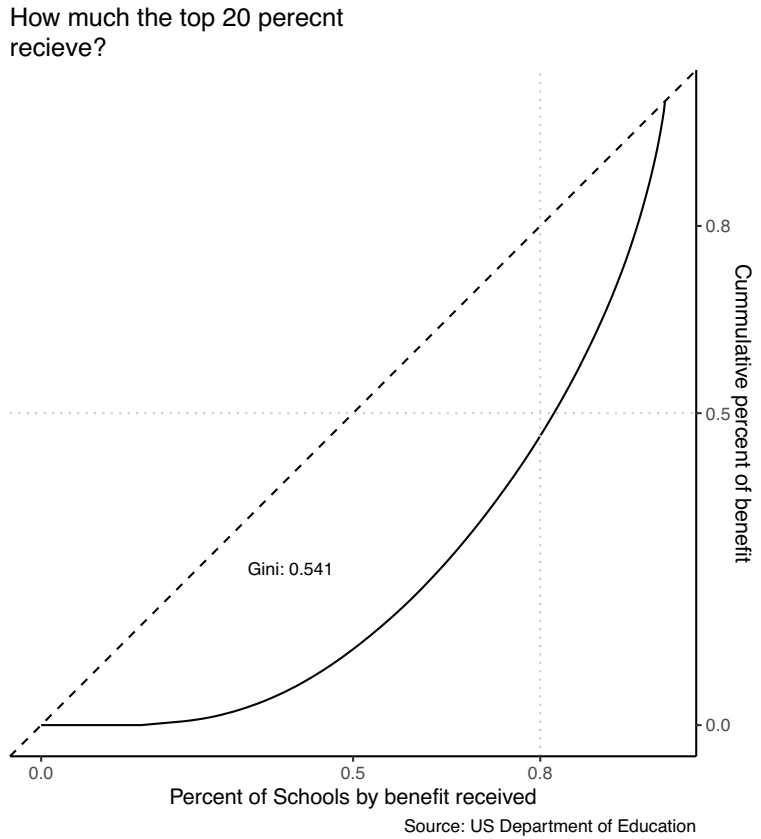


Figure 4.13 Lorenz curve for reduced lunch

You can see that the top 20% of schools that receive reduced lunches represent more then 50% of the whole system.

Data Source: The Common Core of Data from the US Department of Education, available at https://nces.ed.gov/ccd/.

```
        'plotTitle':"How much receives\nthe 20% that receives most?",
        'sourceText':"Source: US Department of Education"}

# text for annotation
## computing
gini=dfIneq.gini('Reduced.Lunch')
## pasting message (number to text before pasting)
GINItext='Gini:' + str(gini.round(3))

# plot diagonal (automatic) and Lorenz curve (in that order)
symbols=['k--','k-"] # color (black) and line type (respect order).
ax1=dfIneq.lorenz('Reduced.Lunch').plot(legend=False,style=symbols)

# annotations
## vertical and horizontal lines (matplotlib)
```

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press

```
plt.axvline(x=0.8,ls=':',c='silver',lw=1)
plt.axhline(y=0.5,ls=':',c='silver',lw=1)

# changing default axis tick values, positions and aspect
plt.yticks((0,0.5,0.8))
plt.xticks((0,0.5,0.8))
ax1.yaxis.set_label_position("right")
ax1.yaxis.set_ticks_position("right")
plt.axes().set_aspect('equal')

# annotating: adding Gini Index value
plt.text(0.4, 0.25,GINItext, fontsize=8)

# texts
plt.title(txtLz['plotTitle'])
plt.ylabel(txtLz['VerticalTitle'])
plt.xlabel(txtLz['HorizontalTitle'])
```

I hope these topics serve you well for plotting one variable. It is time now to pay attention to pairs of variables.

https://doi.org/10.1017/9781108625425.006 Published online by Cambridge University Press