Storytelling with Data! in Altair

by Maisa de Oliveira Fraiz

Introduction

This project aims to replicate the exercises from Cole Nussbaumer Knaflic's book, "Storytelling with Data - Let's Practice!", using `Python Altair`. Our primary objective is to document the reasoning behind the modifications proposed by the author, while also highlighting the challenges that arise when transitioning from the book's Excel-based approach to programming in a different software environment.

`Altair` was selected for this project due to its declarative syntax, interactivity, grammar of graphics, and compatibility with web formatting tools, while within the user-friendly Python environment. Anticipated challenges include the comparatively smaller documentation and development community of `Altair` compared to more established libraries like `Matplotlib`, `Seaborn`, or `Plotly`,and seemingly straightforward tasks in Excel that may require multiple iterations to translate effectively into the language.

In addition to the broader objective, this notebook also serves as a personal journey of learning `Altair`, a syntax that was previously unfamiliar to me. By delving into it, I aim to widen my repertoire in the data visualization field, discovering new ways to create compelling visual representations.

The data for all exercises can be found in the book's official website:
https://www.storytellingwithdata.com/letspractice/downloads

Imports

These are the libraries necessary to run the code for this project.

In [1]:
```
# For data manipulation and visualization
import pandas as pd
import numpy as np
import altair as alt

# For animation in Chapter 6 - Exercise 6
import ipywidgets as widgets
from ipywidgets import interact
from IPython.display import clear_output

# For converting .ipynb into .html
import keyboard
import time
import nbconvert
import nbformat
```

And these are the versions used.

In [2]:
```
# Python version
! python --version
```

```
Python 3.11.6
```

In [3]:
```
# Library version

print("Pandas version: " + pd.__version__)
print("Numpy version: " + np.__version__)
print("Altair version: " + alt.__version__)
print("Ipywidgets version: " + widgets.__version__)
print("Nbconvert version: " + nbconvert.__version__)
print("Nbformat version: " + nbformat.__version__)
```

```
Pandas version: 2.1.2
Numpy version: 1.26.0
Altair version: 5.1.2
Ipywidgets version: 8.1.1
Nbconvert version: 7.11.0
Nbformat version: 5.9.2
```

Table of Contents

- Chapter 2
  - Exercise 1
  - Exercise 4
  - Exercise 5
- Chapter 3
  - Exercise 2
- Chapter 4
  - Exercise 2
  - Exercise 3
- Chapter 5
  - Exercise 4 (Inspired)
- Chapter 6
  - Exercise 6

Chapter 2 - Choose an effective visual

"*When I have some data I need to show, how do I do that in an effective way?*" - Cole Nussbaumer Knaflic

Exercise 2.1 - Improve this table

For this exercise, we will start with a simple table and work our way into transforming it into different types of commonly used visualizations.

Loading the data

The first problem with the Excel-to-Altair translation arises from the data itself, as it is polluted with titles and texts for readability in Excel. This, however, is not friendly when dealing with Python, so we should be careful when loading it. Alterations like this will happen in all subsequent exercises.

In [4]:
```
# Example of polluted loading

table = pd.read_excel(r"Data\2.1 EXERCISE.xlsx")
table
```

Out[4]:

	EXERCISE 2.1	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	FIG 2.1a	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	New client tier share	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	Tier	# of Accounts	% Accounts	Revenue ($M)	% Revenue
6	NaN	A	77	0.070772	4.675	0.25
7	NaN	A+	19	0.017463	3.927	0.21
8	NaN	B	338	0.310662	5.984	0.32
9	NaN	C	425	0.390625	2.805	0.15
10	NaN	D	24	0.022059	0.374	0.02

In [5]:
```
del table
```

In [6]:
```
# Right loading
table = pd.read_excel(r"Data\2.1 EXERCISE.xlsx", usecols = [1, 2, 3, 4, 5], head
table
```

Out[6]:

	Tier	# of Accounts	% Accounts	Revenue ($M)	% Revenue
0	A	77	0.070772	4.675	0.25
1	A+	19	0.017463	3.927	0.21
2	B	338	0.310662	5.984	0.32
3	C	425	0.390625	2.805	0.15
4	D	24	0.022059	0.374	0.02

Table

The initial changes recommended in the book focus on improving the table's readability itself. These changes include reordering the tiers, adding a row to show the total value, incorporating a category called "All others" to account for unmentioned values when the total percentage doesn't add up to 100%, and rounding the numbers while adjusting the percentage format as required.

The following code implements these modifications.

In [7]:
```
# Ordering the tiers

table = table.loc[[1, 0, 2, 3, 4]]
```

In [8]:
```
# Fixing the percentages

table['% Accounts'] = table['% Accounts'].apply(lambda x: x*100)
table['% Revenue'] = table['% Revenue'].apply(lambda x: x*100)
```

In [9]:
```
# Calculating and adding "All other" values

other_account_per = 100 - table['% Accounts'].sum()
other_revenue_per = 100 - table['% Revenue'].sum()

other_account_num = (other_account_per*table['# of Accounts'][0])/table['% Accou
other_revenue_num = (other_revenue_per*table['Revenue ($M)'][0])/table['% Revenu

table.loc[len(table)] = ["All other", other_account_num, other_account_per, othe
```

In [10]:
```
# Since we will not use rounded values or the total row for the graphs,
# we should create a new variable before making the following alterations

table_charts = table.copy()
```

In [11]:
```
# Adding total values row

table.loc[len(table)] = ["Total", table['# of Accounts'].sum(), table['% Account
                         table['Revenue ($M)'].sum(), table['% Revenue'].sum()]
```

In [12]:
```
# Rounding the numbers

table['% Accounts'] = table['% Accounts'].apply(lambda x: round(x))
table['Revenue ($M)'] = table['Revenue ($M)'].apply(lambda x: round(x, 1))
```

The new table is as follows:

In [13]:
```
table
```

Out[13]:

	Tier	# of Accounts	% Accounts	Revenue ($M)	% Revenue
1	A+	19.0	2	3.9	21.0
0	A	77.0	7	4.7	25.0
2	B	338.0	31	6.0	32.0
3	C	425.0	39	2.8	15.0
4	D	24.0	2	0.4	2.0
5	All other	205.0	19	0.9	5.0
6	Total	1088.0	100	18.7	100.0

or, for even better readability in `Python`:

In [14]:
```
table.set_index("Tier")
```

Out[14]:

Tier	# of Accounts	% Accounts	Revenue ($M)	% Revenue
A+	19.0	2	3.9	21.0
A	77.0	7	4.7	25.0
B	338.0	31	6.0	32.0
C	425.0	39	2.8	15.0
D	24.0	2	0.4	2.0
All other	205.0	19	0.9	5.0
Total	1088.0	100	18.7	100.0

Some changes were not implemented, such as colors of rows, alignment of text, and embedding graphs into the table, for lack of compatibility with the Pandas DataFrame format. The percentage symbol (%) next to the number in the percentage columns wasn't added since doing this in Python will transform the data from `int` to `string`, and therefore is not a recommended approach.

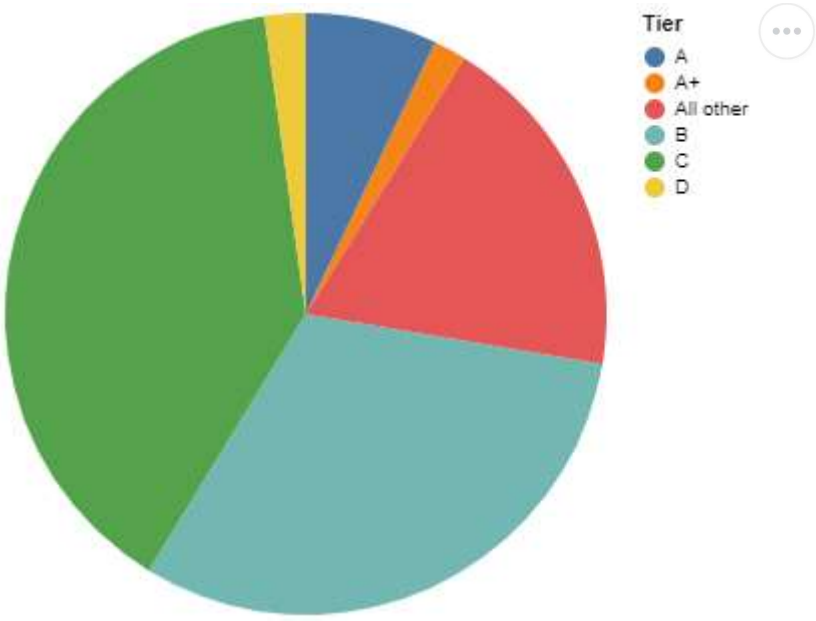
Pie chart

Considering that percentages depict a fraction of a whole, the next proposal is to employ a pie chart. Here is the default Altair graph version:

In [15]:
```
# Default pie chart

alt.Chart(table_charts).mark_arc().encode(
    theta = "% Accounts",
```

```
    color = alt.Color('Tier')
)
```

Out[15]:

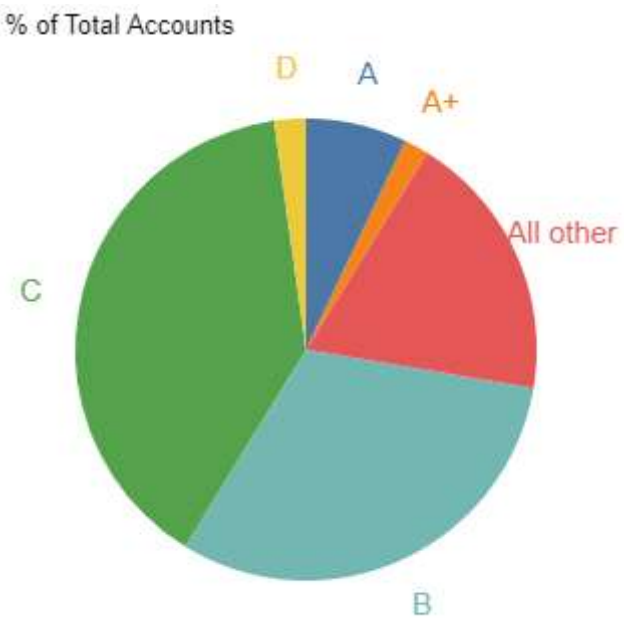


Some of the adjustments needed to bring it closer to the original include reordering the tiers, changing the labels position, altering the color palette, and adding an title.

In [16]:

```
## % of Accounts Pie Chart

# Creating a base chart with a title, aligned to the left and with normal font w
base = alt.Chart(
    table_charts,
    title = alt.Title(r"% of Total Accounts", anchor = 'start', fontWeight = 'nc
).encode(
    theta = alt.Theta("% Accounts:Q", stack = True), # Encoding the angle (theta
    color = alt.Color('Tier', legend = None), # Encoding color based on the 'Tie
    order = alt.Order(field = 'Tier') # Ordering the sectors of the pie chart ba
)

# Creating the pie chart with an outer radius of 115
pie = base.mark_arc(outerRadius = 115)

# Creating text labels for each sector of the pie chart
text = base.mark_text(radius = 140, size = 15).encode(text = alt.Text("Tier"))

# Combining the pie chart and text labels
acc_pie = pie + text
acc_pie
```

Out[16]:

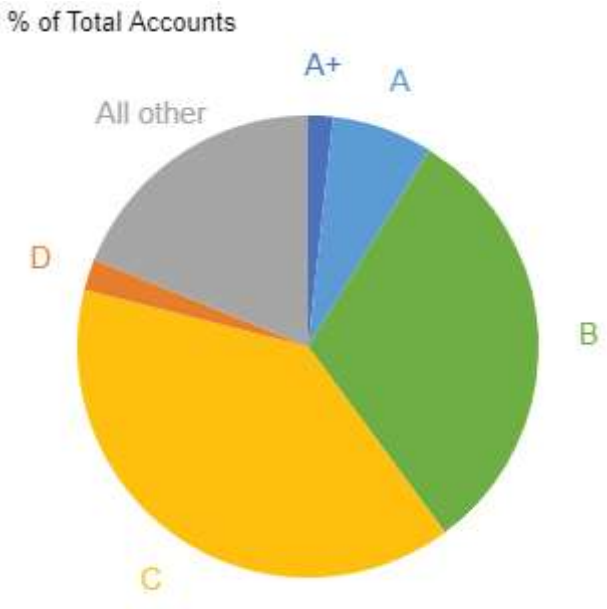


Not informing the data type for the field `order` makes it so Altair rearranges the `Tiers` alphabetically instead of using the order provided by dataframe. We can fix this by identifying `Tier` as Ordered (O).

In [17]:

```
## % of Accounts Pie Chart

base = alt.Chart(
    table_charts,
    title = alt.Title(r"% of Total Accounts", anchor = 'start', fontWeight = 'nc
).encode(
    theta = alt.Theta("% Accounts:Q", stack = True),
    color = alt.Color('Tier',
                      scale = alt.Scale(
                          range = ['#4d71bc', '#5d9bd4', '#6fae45', '#febf0f', '
                        ), # Setting custom colors for each sector of the pie ch
                      sort = None, # So that the colors don't follow the alphabe
                      legend = None
                      ),
    order = alt.Order(field = 'Tier:O'))


pie = base.mark_arc(outerRadius = 115)
text = base.mark_text(radius = 140, size = 15).encode(text = alt.Text("Tier"))


acc_pie = pie + text
acc_pie
```

Out[17]:



Initially, `offset` was used instead of `anchor`, manually specifying the title location in the x-axis by pixels.This produces a more replica-like result, as you define the texts to be exactly to the same place as the example. While this approach yields a result that closely mimics the example, we acknowledge that anchoring provides a faster and cleaner solution. The decision has been made to adopt anchoring for the remainder of this project, prioritizing efficiency and universality across all graphs, even if it means sacrificing pinpoint accuracy in text placement.

The HEX color code values of the palette from the book were acquired through the use of the online tool "Color Picker Online", which is freely accessible at https://imagecolorpicker.com/.

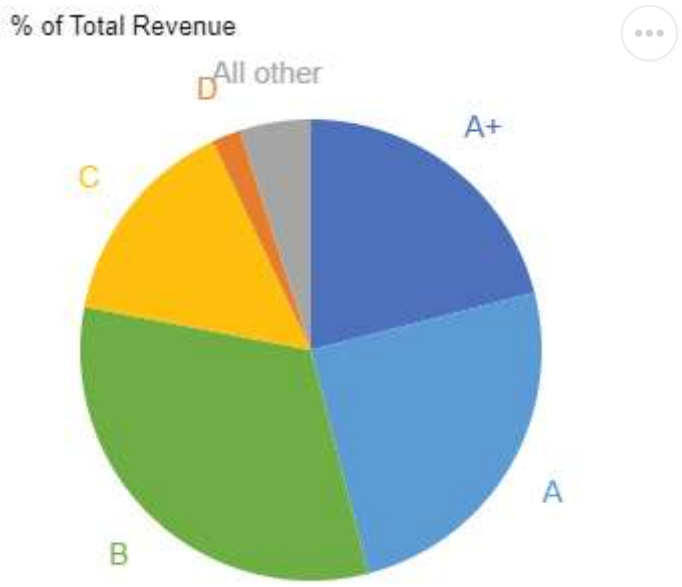
The pie chart above can now be easily modified to represent the percentage of total revenue.

In [18]:

```
# % of Revenue Pie Chart

base = alt.Chart(
    table_charts,
    title = alt.Title(r"% of Total Revenue", anchor = 'start', fontWeight = 'nor
).encode(
    theta = alt.Theta("% Revenue:Q", stack = True),
    color = alt.Color('Tier',
                      scale = alt.Scale(
                          range = ['#4d71bc', '#5d9bd4', '#6fae45', '#febf0f', '
                        ),
                      sort = None,
                      legend = None
                      ),
    order = alt.Order(field = 'Tier:O'))


pie = base.mark_arc(outerRadius = 115)
text = base.mark_text(radius = 140, size = 15).encode(text = alt.Text("Tier"))
```

```
rev_pie = pie + text
rev_pie
```

Out[18]:

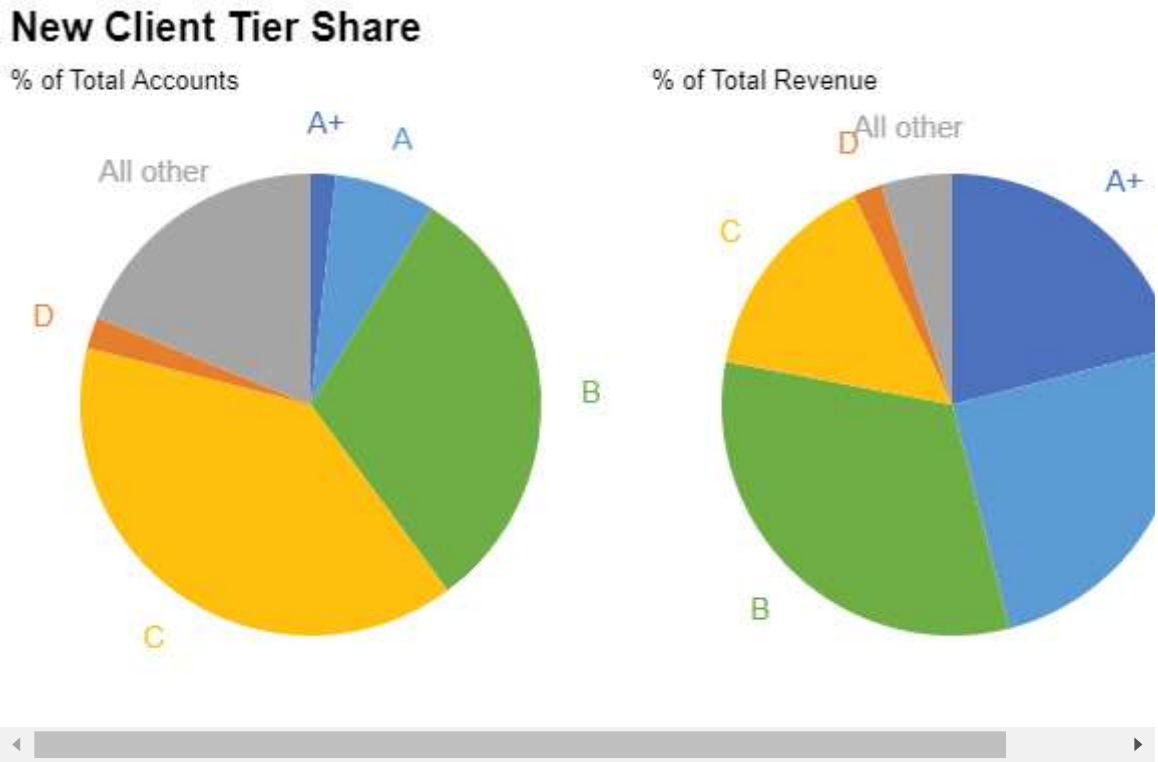


With both graphs available, we can add them next to each other and include a main title.

In [19]:

```
# Combining two pie charts using the vertical concatenation operator '|'
pies = acc_pie | rev_pie

# Setting properties for the combined pie charts
pies.properties(
    title = alt.Title('New Client Tier Share', offset=10, fontSize=20)  # Adding
)
```

Out[19]:



Visualization as depicted in the book:

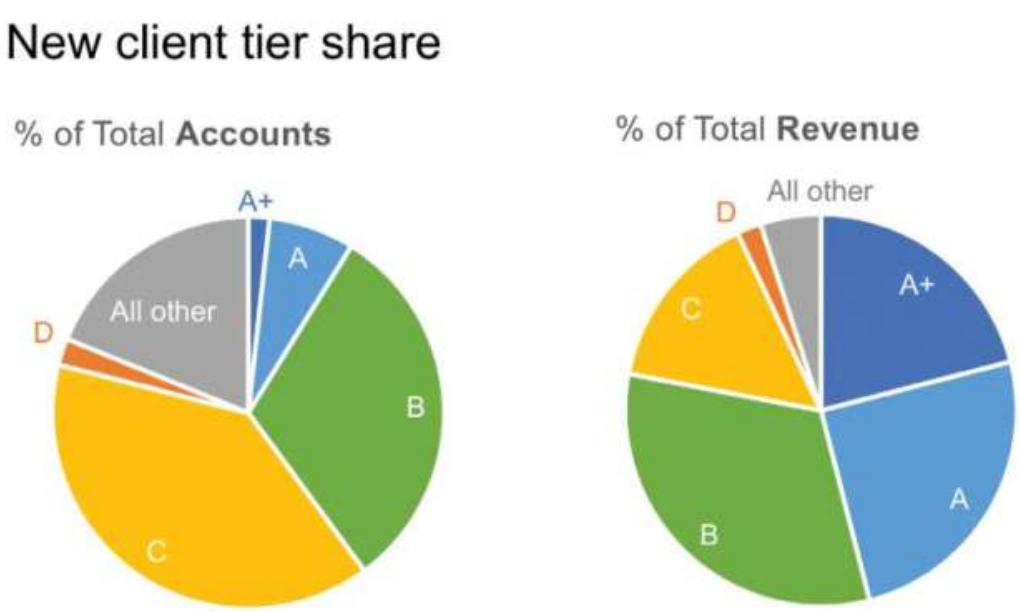


FIGURE 2.1e A pair of pies

Pie charts can present readability challenges, as the human eye struggles to differentiate the relative volumes of slices effectively. While adding data percentages next to the slices can enhance comprehension, it may also introduce unnecessary clutter to the visualization.

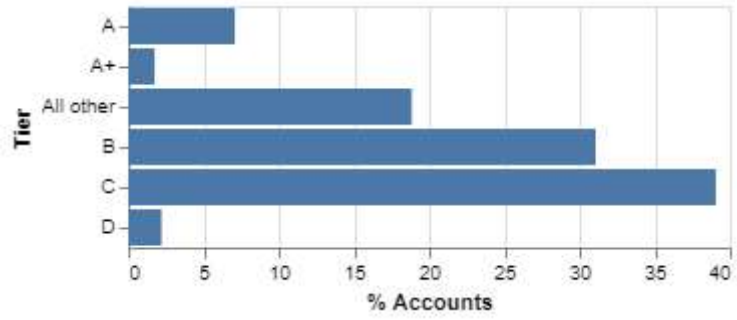
Bar chart

The next graph proposed to tackle is a horizontal bar chart. Since now the comparison does not involve angles and are aligned at the start point, discerning the segment's scale is easier.

This is the default representation in Altair:

In [20]:
```
# Default altair bar chart

alt.Chart(table_charts).mark_bar().encode(
    y = alt.Y('Tier'),
    x = alt.X('% Accounts'))
```

Out[20]:



The necessary adjustments involve placing the "Tier" label in the upper left corner, displaying values next to the bars instead of using an x-axis, and adding a title while rearranging the tiers.

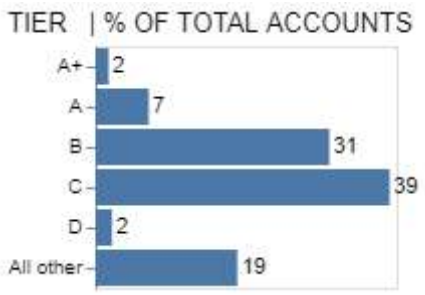
In [21]:
```
# Creating a base chart with a title, aligned to the left and with normal font w
base = alt.Chart(
    table_charts,
    title = alt.Title('TIER | % OF TOTAL ACCOUNTS', anchor = 'start', fontWeight
).mark_bar().encode(
    y = alt.Y('Tier', title = None),  # Encoding the 'Tier' field on the y-axis,
    x = alt.X('% Accounts', axis = None),  # Encoding the '% Accounts' field on
    order = alt.Order(field = 'Tier:O'),  # Ordering the bars based on the 'Tier
    text = alt.Text("% Accounts", format=".0f")  # Displaying the '% Accounts' v
)

# Creating the final bar chart by combining the bars and text labels
final_acc = base.mark_bar() + base.mark_text(align='left', dx=2)

# Displaying the final bar chart
final_acc
```

Out[21]:

Adding the `order` by `Tier:O` didn't had the same effect as it did on the pie chart. The compatible method for this case is adding a `sort` keyword in the axis to be sorted.

In [22]:
```
base = alt.Chart(
    table_charts,
    title = alt.Title('TIER    | % OF TOTAL ACCOUNTS      |', anchor = 'start', fo
).encode(
    y = alt.Y('Tier', sort = ["A+"], title = None),  # Encoding the 'Tier' fiela
    x = alt.X('% Accounts', axis = None),
    text = alt.Text("% Accounts", format = ".0f")
)

final_acc = (base.mark_bar() + base.mark_text(align = 'left', dx = 2)).propertie
final_acc
```

Out[22]:

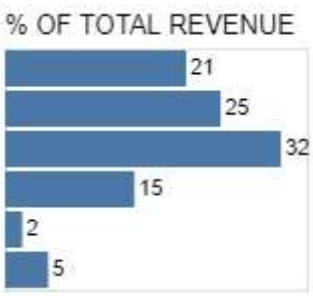


Now we do the same for the revenue column. In addition, the y-axis is removed so it isn't repeated when uniting the charts.

In [23]:
```
base = alt.Chart(
    table_charts,
    title = alt.Title('% OF TOTAL REVENUE', anchor = 'start', fontWeight = 'norm
).encode(
```

```
    y = alt.Y('Tier', sort = ["A+"]).axis(None),
    x = alt.X('% Revenue').axis(None),
    text = alt.Text("% Revenue", format = ".0f")
)

final_rev = (base.mark_bar() + base.mark_text(align = 'left', dx = 2)).propertie
final_rev
```

Out[23]:

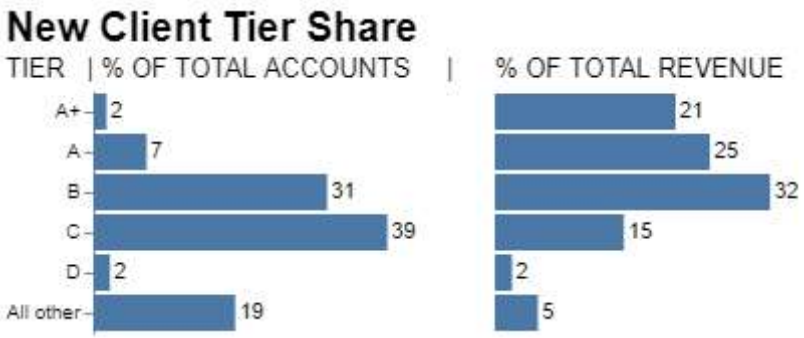


Similar to the pie chart, we can arrange these graphs side by side and include a main title.

In [24]:

```
# Combining two charts horizontally using the concatenation operator '|'
hor_bar = final_acc | final_rev

# Configuring the view of the combined chart, removing strokes
hor_bar.configure_view(stroke = None).properties(
    title = alt.Title('New Client Tier Share', anchor = 'start', fontSize = 20)
)
```

Out[24]:



Visualization as depicted in the book:

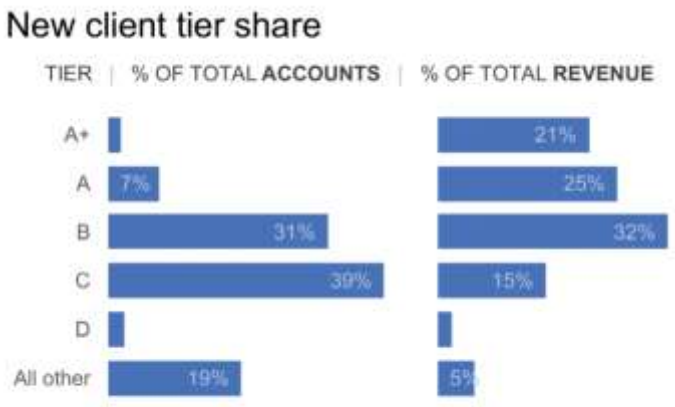


FIGURE 2.1f Two horizontal bar charts

In both the pie and bar chart, the labeling beside the value is not in the same position as the examples provided. This discrepancy arises from the fact that adjusting these labels to match the book's examples, with variations in positions (some inside and some outside of the pie), different colors, and even omitting some labels, would be a labor-intensive manual task in Altair. These adjustments are primarily for aesthetic purposes and do not significantly impact readability, in some cases even obscuring the information being presented.

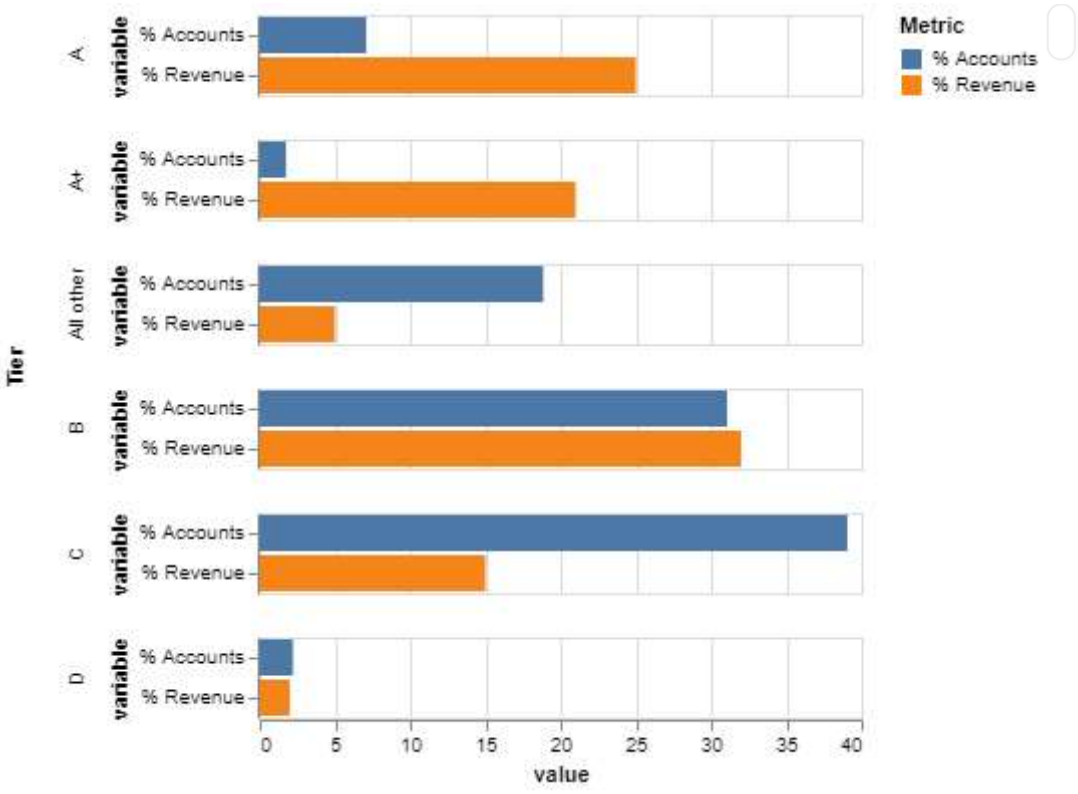
Examples of how to manually define labels will be presented in future exercises.

Horizontal dual series bar chart

The two graphs in the last visualization can be merged into a single grouped bar chart.

In [25]:
```
# Altair with default settings
alt.Chart(table_charts).mark_bar().encode(
    x = alt.X('value:Q'),  # Encoding the quantitative variable 'value' on the x
    y = alt.Y('variable:N'),  # Encoding the nominal variable 'variable' on the
    color = alt.Color(
        'variable:N',
        legend = alt.Legend(title = 'Metric')
    ),  # Encoding color based on 'variable' with legend title 'Metric'
    row = alt.Row('Tier:O')  # Faceting by rows based on the ordinal variable 'T
).transform_fold(
    fold = ['% Accounts', '% Revenue'],  # Transforming the data by folding the
    as_ = ['variable', 'value']  # Renaming the folded columns to 'variable' ana
)
```

Out[25]:



The necessary alterations involve removing the grid, adjusting label positions and reducing redundancy, adding a title and subtitle, and changing the color palette.

In [26]:
```
# Custom settings
merged_hor_bar = alt.Chart(
    table_charts,
    title = alt.Title('New client tier share', fontSize = 20)  # Adding a title
).mark_bar().encode(
    x = alt.X(
        'value:Q',
        axis = alt.Axis(
            title = "TIER |  % OF TOTAL ACCOUNTS vs REVENUE",  # Setting a custo
            grid = False, # Remove grid
            orient = 'top', # Put axis on top
            labelColor = "#888888",  # Setting the label color as gray
            titleColor = '#888888'  # Setting the title color as gray
        )
    ),
    y = alt.Y(
        'variable:N',
        axis = alt.Axis(title = None, labels = False, ticks = False)  # Removing
    ),
    color = alt.Color(
        'variable:N',
        legend = alt.Legend(title = 'Metric'),  # Adding a legend with a custom
        scale = alt.Scale(range = ['#b4c6e4', '#4871b7'])  # Setting a custom co
    ),
    row = alt.Row(
        'Tier:O',
        header = alt.Header(labelAngle = 0, labelAlign = "left"),  # Rotating ro
        title = None,
        sort = ['A+'],  # Sorting rows based on 'Tier'
        spacing = 10  # Adding spacing between rows
    )
).transform_fold(
    fold = ['% Accounts', '% Revenue'],  # Transforming the data by folding the
    as_ = ['variable', 'value']  # Renaming the folded columns to 'variable' ana
).properties(
    width = 200  # Setting the width of the chart
).configure_view(stroke = None)  # Removing the stroke from the view

merged_hor_bar
```

Out[26]:

Visualization as depicted in the book:

FIGURE 2.1g Horizontal dual series bar chart

Vertical bar chart

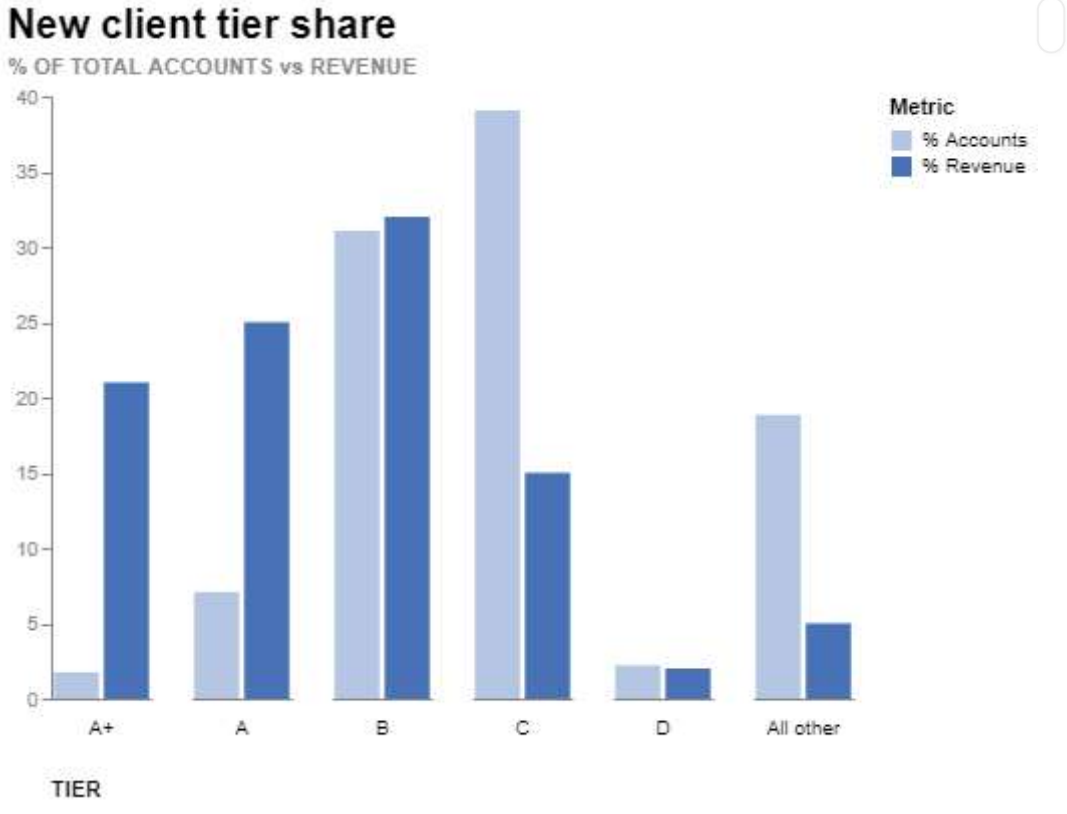
We should can modify the bar chart to be in a vertical orientation. This can be done by switching the y and x axis and the "Row" class to the "Column" class, as well as reorient the labels.

In [27]:

```
# Creating a vertical bar chart
vert_bar = alt.Chart(
    table_charts,
    title = alt.Title('New client tier share', fontSize = 20)  # Adding a title
).mark_bar().encode(
    y = alt.Y(
        'value:Q',
```

```
        axis = alt.Axis(
            title = "% OF TOTAL ACCOUNTS vs REVENUE",  # Setting a custom title
            titleAlign = 'left',  # Aligning the title to the left
            titleAngle = 0,  # Setting the title angle to 0 degrees
            titleAnchor = 'end',  # Anchoring the title to the end
            titleY = -10,  # Adjusting the title position
            grid = False,  # Turning off grid lines
            labelColor = "#888888",  # Setting the label color to gray
            titleColor = '#888888'  # Setting the title color to gray
        )
    ),
    x = alt.X(
        'variable:N',
        axis = alt.Axis(title = None, labels = False, ticks = False)  # Removing
    ),
    color = alt.Color(
        'variable:N',
        legend = alt.Legend(title = 'Metric'),  # Adding a legend with a custom
        scale = alt.Scale(range = ['#b4c6e4', '#4871b7'])  # Setting a custom co
    ),
    column = alt.Column(
        'Tier:O',
        header = alt.Header(labelOrient = 'bottom', titleOrient = "bottom", titl
        sort = ['A+'],  # Sorting columns based on 'Tier'
        title = 'TIER'  # Adding a title for the column
    )
).transform_fold(
    fold = ['% Accounts', '% Revenue'],  # Transforming the data by folding the
    as_ = ['variable', 'value']  # Renaming the folded columns to 'variable' ana
).properties(
    width = 50  # Setting the width of the chart
).configure_view(stroke = None)  # Removing the stroke from the view

vert_bar
```

Out[27]:



Visualization as depicted in the book:

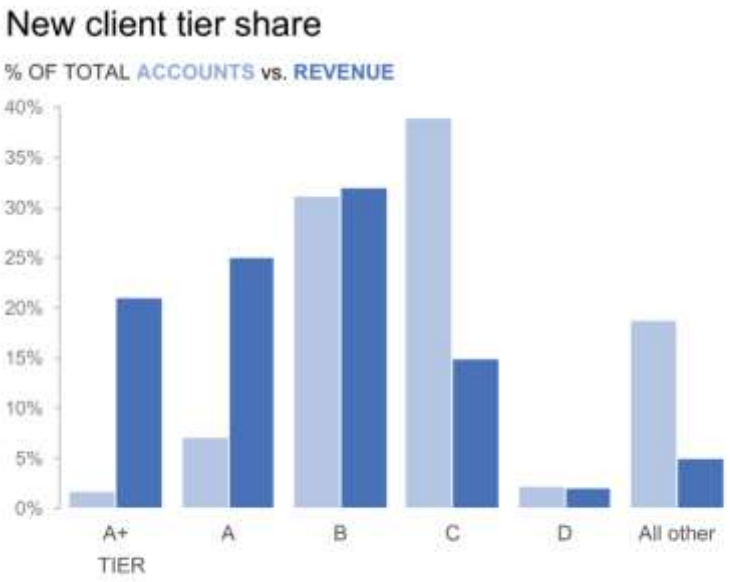


FIGURE 2.1h A vertical bar chart

It's worth noting that titles in Altair do not readily support the option of changing the colors of individual words within them. As a simple solution for the time being, we will retain the legend that effectively indicates which column corresponds to each word. Future exercises will delve into a more complicated way to tackle this challenge.

In the code above, we've utilized the `transform_fold` method to generate the grouped bar chart because our data is structured in the 'wide form', which is the standard Excel format. However, Altair (as well as other visualization languages) is inherently designed to work with 'long form' data. The `transform_fold` function automates this conversion within the chart, enabling us to create the graph. This approach can obscure the process, making it preferable to perform the data transformation before creating the visualizations.

In [28]:
```
# Transforms the data to the long-form format

melted_table = pd.melt(table_charts, id_vars = ['Tier'], var_name = 'Metric', va
melted_table
```

Out[28]:

	Tier	Metric	Value
0	A+	# of Accounts	19.000000
1	A	# of Accounts	77.000000
2	B	# of Accounts	338.000000
3	C	# of Accounts	425.000000
4	D	# of Accounts	24.000000
5	All other	# of Accounts	205.000000
6	A+	% Accounts	1.746324
7	A	% Accounts	7.077206
8	B	% Accounts	31.066176
9	C	% Accounts	39.062500
10	D	% Accounts	2.205882
11	All other	% Accounts	18.841912
12	A+	Revenue ($M)	3.927000
13	A	Revenue ($M)	4.675000
14	B	Revenue ($M)	5.984000
15	C	Revenue ($M)	2.805000
16	D	Revenue ($M)	0.374000
17	All other	Revenue ($M)	0.935000
18	A+	% Revenue	21.000000
19	A	% Revenue	25.000000
20	B	% Revenue	32.000000
21	C	% Revenue	15.000000
22	D	% Revenue	2.000000
23	All other	% Revenue	5.000000

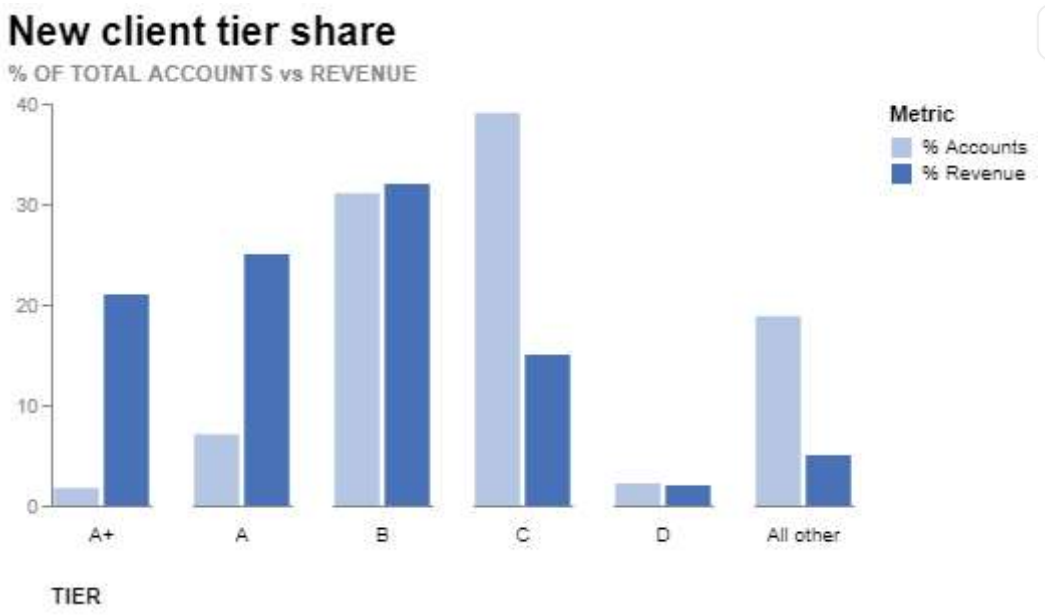
We can now use this table to remake the bar chart without the `transform_fold` method.

In [29]:

```
# Selecting specific rows from the melted table based on the 'Metric' column
selected_rows = melted_table[melted_table['Metric'].isin(['% Accounts', '% Reven

vert_bar2 = alt.Chart(
    selected_rows,
    title = alt.Title('New client tier share', fontSize = 20)  # Adding a title
).mark_bar().encode(
    y = alt.Y(
        'Value',
        axis = alt.Axis(
```

```
            title = "% OF TOTAL ACCOUNTS vs REVENUE",  # Setting a custom title
            titleAlign = 'left',  # Aligning the title to the left
            titleAngle = 0,  # Setting the title angle to 0 degrees
            titleAnchor = 'end',  # Anchoring the title to the end
            titleY = -10,  # Adjusting the title position
            grid = False,  # Turning off grid lines
            labelColor = "#888888",  # Setting the label color to gray
            titleColor = '#888888'  # Setting the title color to gray
        )
    ),
    x = alt.X(
        'Metric',
        axis = alt.Axis(title = None, labels = False, ticks = False)  # Removing
    ),
    color = alt.Color(
        'Metric',
        scale = alt.Scale(range = ['#b4c6e4', '#4871b7'])  # Setting a custom co
    ),
    column = alt.Column(
        'Tier',
        header = alt.Header(labelOrient='bottom', titleOrient="bottom", titleAnc
        sort = ['A+'],  # Sorting columns based on 'Tier'
        title = 'TIER'  # Adding a title for the column
    )
).properties(
    height = 200, width = 50  # Setting the height and width of the chart
).configure_view(stroke = None)  # Removing the stroke from the view

vert_bar2
```

Out[29]:



Bar chart with lines

The next proposed graph is an extension of the previous bar chart, featuring the addition of lines to accentuate the endpoints of the columns within the same tier.

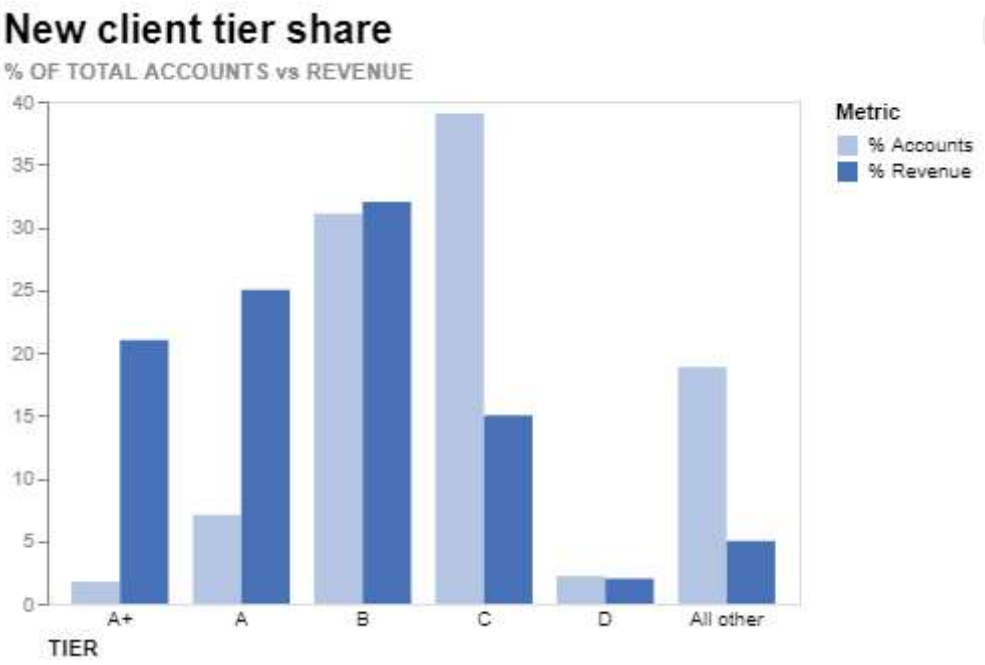
However, due to the nature of faceted charts, we encounter an error (*ValueError: Faceted charts cannot be layered. Instead, layer the charts before faceting*) when attempting to layer it. This issue arises because, in faceted charts, the x-axis structure is altered.

Now that we've transformed our data into long-format, we can work around this problem by creating our graph without using the 'column' method, and thereby, avoiding faceting. Instead of specifying 'x' as 'Metric,' 'y' as 'Value,' 'color' as 'Metric,' and 'column' as 'Tier,' we can redefine 'x' as 'Tier,' 'y' as 'Value,' 'color' as 'Metric,' and introduce 'XOffset' for controlling the horizontal positioning of data points within a group. In essence, 'column' primarily serves to define distinct x-axis categories, while 'XOffset' is employed to manage the horizontal placement of data points within a group.

The following chart incorporates the alterations we discussed and yields a graph that closely resembles the previous one.

In [30]:
```
bar = alt.Chart(
    selected_rows,
    title = alt.Title('New client tier share', fontSize = 20, anchor = 'start')
).mark_bar().encode(
    x = alt.X(
        'Tier',
        axis = alt.Axis(
            title = 'TIER',  # Setting a custom title for the x-axis
            labelAngle = 0,  # Setting the label angle to 0 degrees
            titleAnchor = "start",  # Anchoring the title to the start
            domain = False,  # Hiding the x-axis domain line
            ticks = False  # Hiding the x-axis ticks
        ),
        sort = ['A+']  # Sorting x-axis based on 'Tier'
    ),
    y = alt.Y(
        'Value',
        axis = alt.Axis(
            title = "% OF TOTAL ACCOUNTS vs REVENUE",  # Setting a custom title
            titleAlign = 'left',  # Aligning the title to the left
            titleAngle = 0,  # Setting the title angle to 0 degrees
            titleAnchor = 'end',  # Anchoring the title to the end
            titleY = -10,  # Adjusting the title position
            grid = False,  # Turning off grid lines
            labelColor = "#888888",  # Setting the label color to gray
            titleColor =' #888888'  # Setting the title color to gray
        )
    ),
    color = alt.Color(
        'Metric',
        scale = alt.Scale(range = ['#b4c6e4', '#4871b7'])  # Setting a custom co
    ),
    xOffset = 'Metric'  # Adjusting the x-offset
).properties(
    height = 250, width = 375  # Setting the height and width of the chart
)

bar.configure_view(stroke = None)  # Removing the stroke from the view

bar
```

Out[30]:



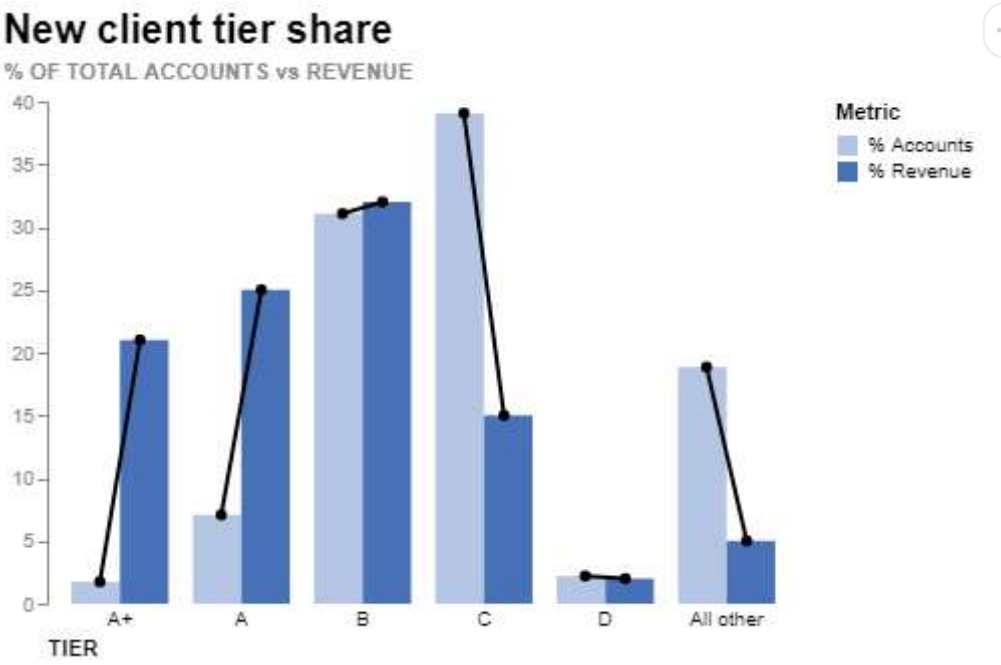
Now, we can layer the graph and introduce the lines. It's worth noting that creating the lines in Altair is not a straightforward task and a considerable amount of documentation searching was necessary to achieve it.

In [31]:

```
# x, y and y2 do not accept to be defined as "condition", so repetitive code is

# Create a vertical rule chart for ascending lines
rule_asc = alt.Chart(selected_rows).mark_rule(x2Offset = 10, xOffset = -10).encc
    x = alt.X('Tier', sort = ['A+']),  # X-axis encoding for 'Tier', sorted in a
    x2 = alt.X2('Tier'),  # End point of the rule line
    y = alt.Y('min(Value)'),  # Start point of the rule line
    y2 = alt.Y2('max(Value)'),  # End point of the rule line
    strokeWidth = alt.value(2),  # Set the stroke width of the rule line
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |  # Condition for specific tiers
        (alt.datum.Tier == 'A') |   # to determine opacity settings
        (alt.datum.Tier == 'B'),
        alt.value(1), alt.value(0)  # Opacity set to 1 if condition is met, else
    )
)

# Create a vertical rule chart for descending lines
rule_desc = alt.Chart(selected_rows).mark_rule(x2Offset = 10, xOffset = -10
).encode(
    x = alt.X('Tier', sort = ['A+']),
    x2 = alt.X2('Tier'),
    y = alt.Y('max(Value)'),
    y2 = alt.Y2('min(Value)'),
    strokeWidth = alt.value(2),
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |
        (alt.datum.Tier == 'A')  |
        (alt.datum.Tier == 'B'),
        alt.value(0), alt.value(1)
        )
    )

# Points of % Revenue where % Revenue > % Accounts
```

```
points1 = alt.Chart(selected_rows).mark_point(filled = True, xOffset = 10, color
    x = alt.X('Tier', sort = ['A+']),
    y = alt.Y('max(Value)'),
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |
        (alt.datum.Tier == 'A')  |
        (alt.datum.Tier == 'B'),
        alt.value(1), alt.value(0)
        )
    )

# Points of % Revenue where % Revenue < % Accounts
points2 = alt.Chart(selected_rows).mark_point(filled = True, xOffset = 10, color
    x = alt.X('Tier', sort = ['A+']),
    y = alt.Y('min(Value)'),
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |
        (alt.datum.Tier == 'A')  |
        (alt.datum.Tier == 'B'),
        alt.value(0), alt.value(1)
        )
    )

# Points of % Accounts where % Revenue < % Accounts
points3 = alt.Chart(selected_rows).mark_point(filled = True, xOffset = -10, colo
    x = alt.X('Tier', sort = ['A+']),
    y = alt.Y('max(Value)'),
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |
        (alt.datum.Tier == 'A')  |
        (alt.datum.Tier == 'B'),
        alt.value(0), alt.value(1)
        )
    )

# Points of % Revenue where % Revenue > % Accounts
points4 = alt.Chart(selected_rows).mark_point(filled = True, xOffset = -10, colo
    x = alt.X('Tier', sort = ['A+']),
    y = alt.Y('min(Value)'),
    opacity = alt.condition(
        (alt.datum.Tier == 'A+') |
        (alt.datum.Tier == 'A')  |
        (alt.datum.Tier == 'B'),
        alt.value(1), alt.value(0)
        )
    )

bar_point = bar + rule_asc + rule_desc + points1 + points2 + points3 + points4
bar_point.configure_view(stroke = None)
```

Out[31]:



Visualization as depicted in the book:

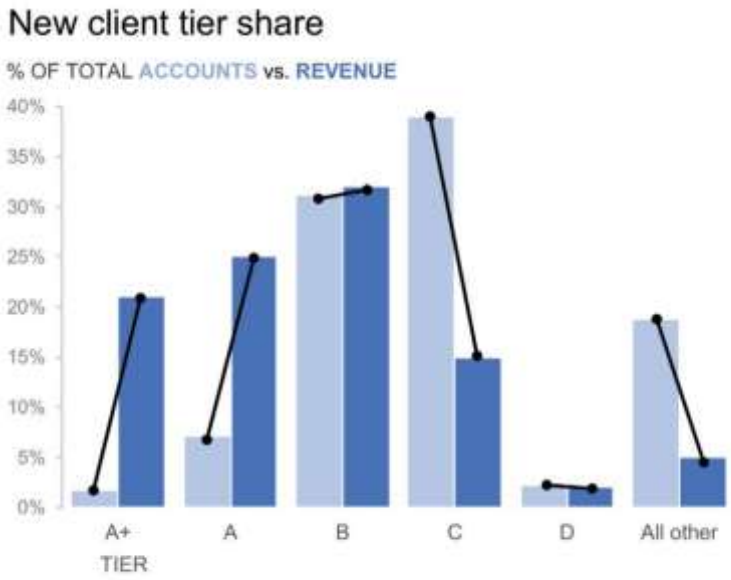


FIGURE 2.1i Let's draw some lines

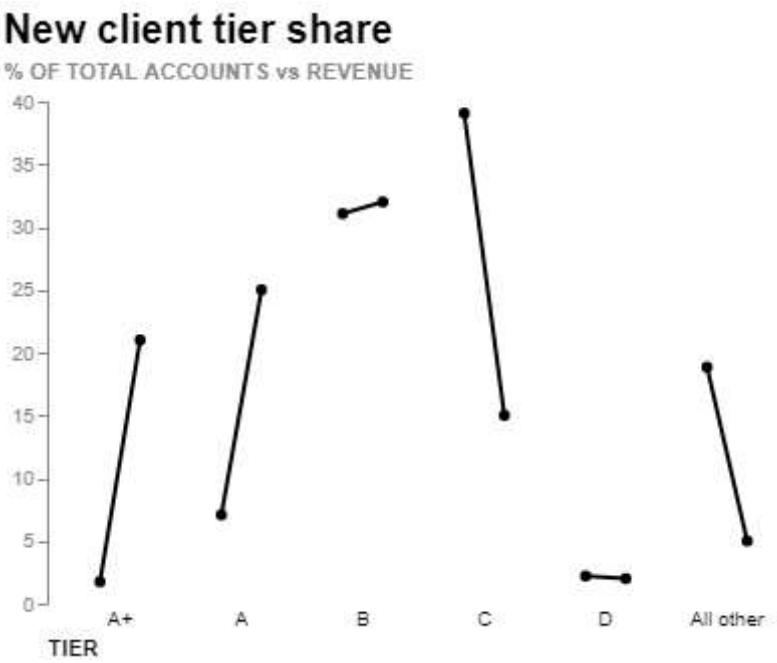
Lines only

With two types of visualizations displaying the same data, the book suggests to eliminate the bars altogether. This can be done without the need to program more graphs:

In [32]:

```
# Configure the legend to be disabled (hidden)
# The 'opacity' configuration is set to 0, making the bars transparent
point = bar_point.configure_mark(opacity = 0).configure_view(stroke = None).conf
point
```

Out[32]:



Visualization as depicted in the book:

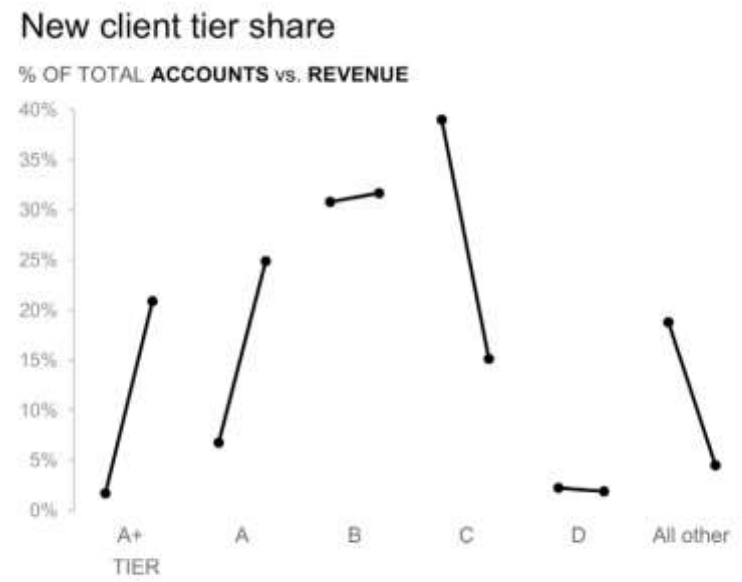


FIGURE 2.1j Take away the bars

Slope graph

At last, we can reassemble the lines to create a slope graph.

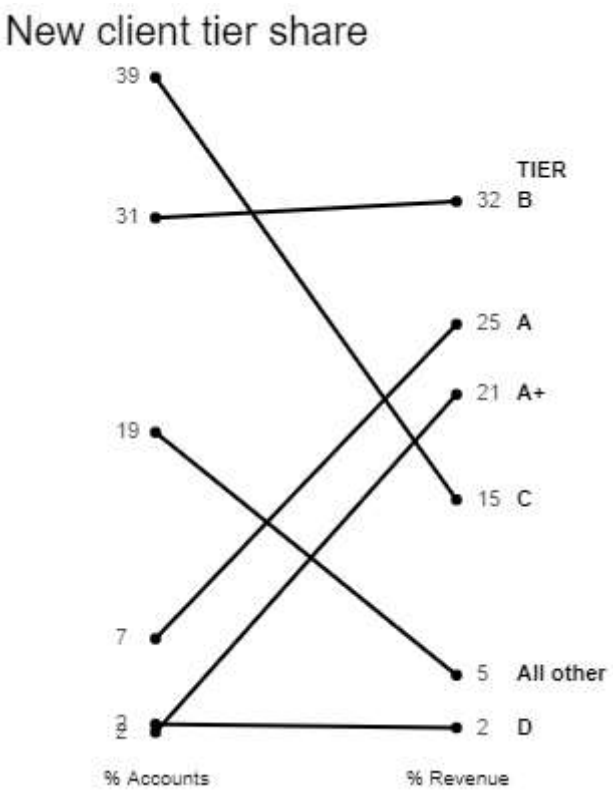
In [33]:

```
# Create base chart, setting the title
base = alt.Chart(
    selected_rows,
    title = alt.Title("New client tier share", anchor = 'start', fontWeight = 'n
)

# Line chart configuration
line = base.mark_line(
    point = True # The lines have points at the end
).encode(
    x = alt.X(
```

```
        'Metric',
        axis = alt.Axis(title = None, labelAngle = 0, domain = False, ticks = Fa
    ),
    y = alt.Y('Value', axis = None),
    color = alt.Color(
        'Tier',
        scale = alt.Scale(range = ['black']), # All Tier lines are black
        legend = None
        )
).properties(
    width = 300,
    height = 350
)

# Labels to the right of the slope
# These labels are for the Accounts
labels1 = base.mark_text(
    align = 'left',
    dx = 10
).encode(
    x = alt.X('Metric'),
    y = alt.Y('Value'),
    text = alt.Text('Value:Q', format = '.0f'),
    opacity = alt.condition(alt.datum.Metric == '% Accounts', alt.value(0), alt.
)

# Labels to the left of the slope
# These labels are for the Revenue
labels2 = base.mark_text(
    align ='left',
    dx = -20
).encode(
    x = alt.X('Metric'),
    y = alt.Y('Value'),
    text = alt.Text('Value:Q', format='.0f'),
    opacity = alt.condition(alt.datum.Metric == '% Accounts', alt.value(0.7), al
)

# Labels for the Tiers
tier_labels = base.mark_text(
    align = 'left',
    dx = 30,
    fontWeight = 'bold'
).encode(
    x = alt.X('Metric'),
    y = alt.Y('Value'),
    text = 'Tier',
    opacity = alt.condition(alt.datum.Metric == '% Accounts', alt.value(0), alt.
)

# Tier title
tier_title = alt.Chart(
    {"values": [{"text":  ['TIER']}]}
).mark_text(
            align = "left",
            dx = 105,
            dy = -120,
            fontWeight = 'bold'
).encode(
    text = "text:N")
```

```
slope = line + labels1 + labels2 + tier_labels + tier_title
slope.configure_view(stroke = None)
```

Out[33]:

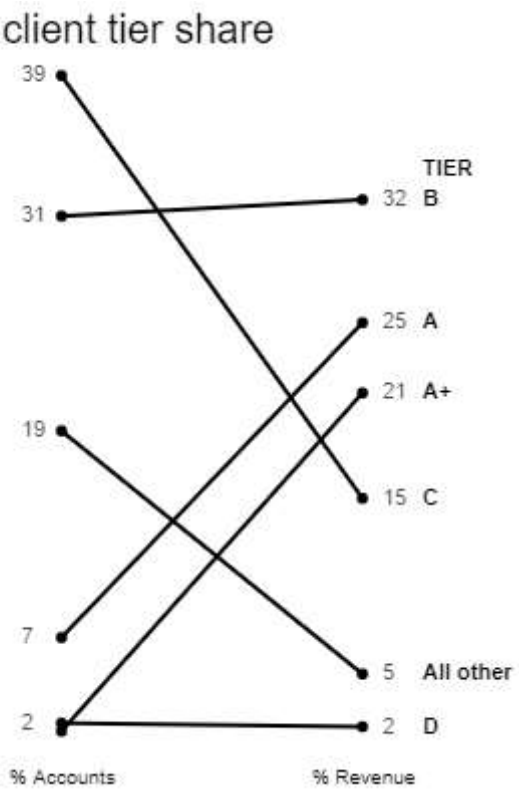


Notice how there are two numbers overlapping, the percentage of accounts of the A+ and D tiers. Since they are the same number when rounded, we can just eliminate one of the values display.

In [34]:

```
# Eliminate Tier A+ from display
label_condition = (alt.datum.Metric == '% Accounts') & (alt.datum.Tier != 'A+')


labels2 = base.mark_text(
    align ='left',
    dx = -20
).encode(
    x = alt.X('Metric'),
    y = alt.Y('Value'),
    text = alt.Text('Value:Q', format='.0f'),
    opacity = alt.condition(label_condition, alt.value(0.7), alt.value(0))
)

slope = line + labels1 + labels2 + tier_labels + tier_title
slope.configure_view(stroke = None)
```

Out[34]:



Visualization as depicted in the book:

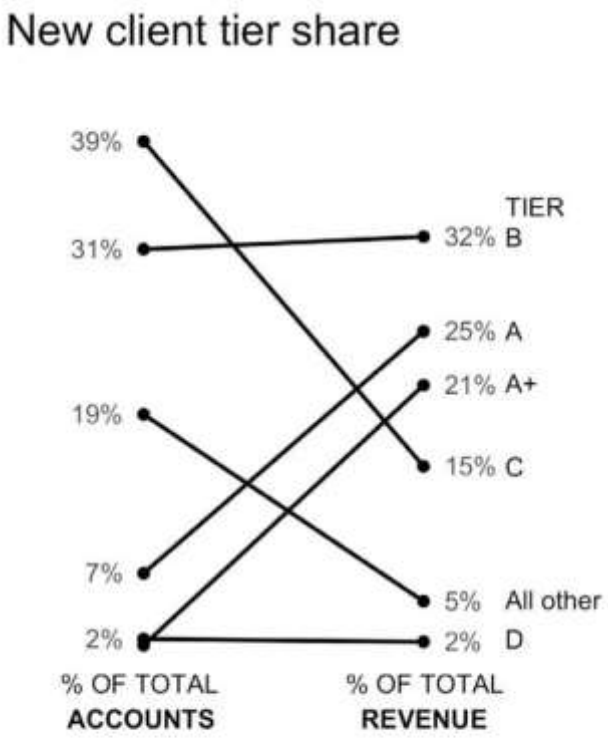


FIGURE 2.1k A slopegraph

Interactivity

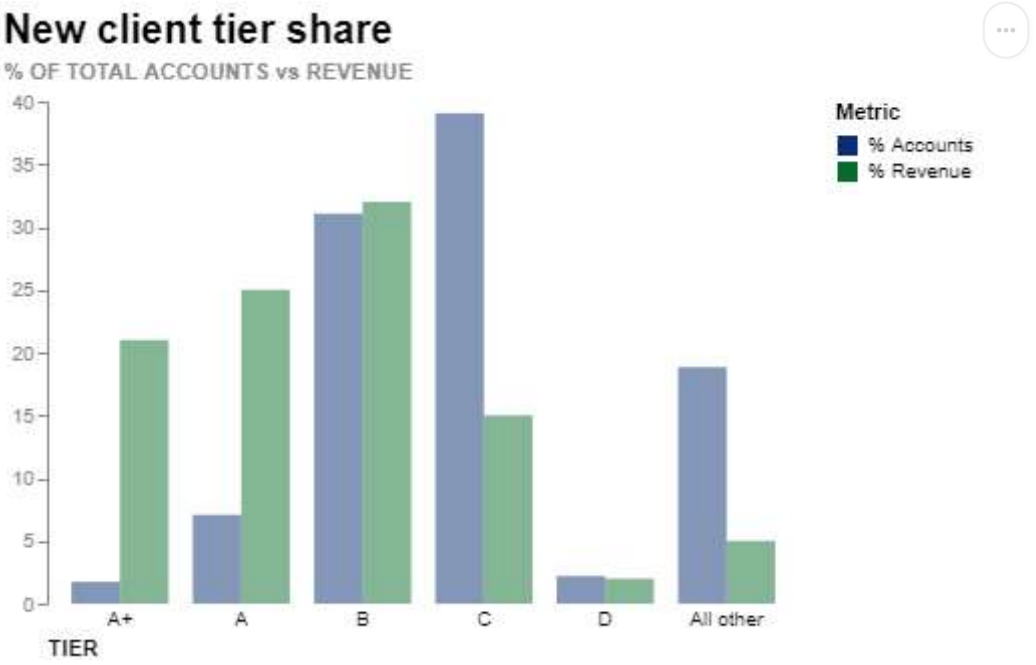
In this exercise, the selected graph for interactivity is the simple vertical bar chart, without the lines.

The chosen interactive features include a simple tooltip, revealing the precise values of each column upon hovering. Additionally, it shows the legend to provide further clarity about the corresponding data categories.

Finally, the columns are designed to highlight dynamically when the viewer hovers over them. Because of this feature, the color palette was changed, since the monochromatic version made the highlighted column and the not highlighted neighbor too similar.

In [35]:
```
# Selection for interactive points on hover
hover = alt.selection_point(on='mouseover', nearest=True, empty=False)

# Bar chart configuration with interactivity
bar_interactive = alt.Chart(
    selected_rows,
    title = alt.Title('New client tier share', fontSize = 20, anchor = 'start')
).mark_bar().encode(
    x = alt.X(
        'Tier',
        axis = alt.Axis(title = 'TIER', labelAngle = 0, titleAnchor = "start", d
        sort = ['A+']
    ),
    y = alt.Y('Value', axis = alt.Axis(
        title = "% OF TOTAL ACCOUNTS vs REVENUE",
        titleAlign = 'left',
        titleAngle = 0,
        titleAnchor = 'end',
        titleY = -10,
        grid = False,
        labelColor = "#888888",
        titleColor = '#888888'
    )),
    color = alt.Color('Metric', scale = alt.Scale(range = ['#0a2f73', '#096b2b']
    xOffset = 'Metric',
    opacity = alt.condition(hover, alt.value(1), alt.value(0.5)),  # Set opacity
    tooltip = ['Value:Q', 'Metric']  # Show tooltip with specified fields
).properties(
    height = 250, width = 375
).add_params(hover)  # Add the hover selection to the chart

# Configure view settings for the interactive bar chart
bar_interactive.configure_view(stroke=None)
```

Out[35]:



Exercise 2.4 - Practice in your tool

This exercise proposes to display the same data in six different formats, hand-drawn by the author in the theoretical exercise 2.3. The purpose of the activity is to practice in our own tool, and while C. Nussbaumer uses Excel, we will proceed with Altair.

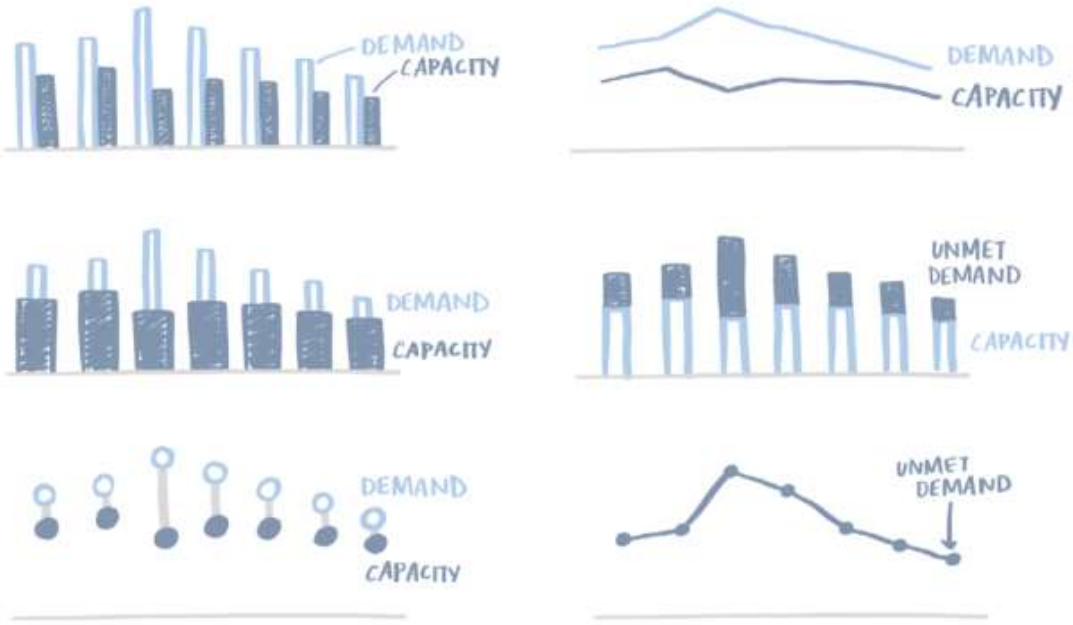


FIGURE 2.3b My data drawings

Loading the data

In [36]:

```
# Loading considering the NaN caused by Excel formatting
table = pd.read_excel(r"Data\2.4 EXERCISE.xlsx", usecols = [1, 2, 3], header = 4
table
```

Out[36]:

	DATE	CAPACITY	DEMAND
0	2019-04	29263	46193
1	2019-05	28037	49131
2	2019-06	21596	50124
3	2019-07	25895	48850
4	2019-08	25813	47602
5	2019-09	22427	43697
6	2019-10	23605	41058
7	2019-11	24263	37364
8	2019-12	24243	34364
9	2020-01	25533	34149
10	2020-02	24467	25573
11	2020-03	25194	25284

In the graphs for this exercise, we require the inclusion of the "unmet demand" column, which is currently absent from the dataset. To obtain this value, we can calculate the difference between demand and capacity for each date.

In [37]:
```
# Calculate Unmet Demand
table['UNMET DEMAND'] = table['DEMAND'] - table['CAPACITY']

# Show only the first five lines
table.head()
```

Out[37]:

	DATE	CAPACITY	DEMAND	UNMET DEMAND
0	2019-04	29263	46193	16930
1	2019-05	28037	49131	21094
2	2019-06	21596	50124	28528
3	2019-07	25895	48850	22955
4	2019-08	25813	47602	21789

Now we transform the data from the wide-format used in Excel to the long-format used in Altair.

In [38]:
```
# Transforming data into long-format

melted_table = pd.melt(table, id_vars = ['DATE'], var_name = 'Metric', value_nam
melted_table
```

Out[38]:

	DATE	Metric	Value
0	2019-04	CAPACITY	29263
1	2019-05	CAPACITY	28037
2	2019-06	CAPACITY	21596
3	2019-07	CAPACITY	25895
4	2019-08	CAPACITY	25813
5	2019-09	CAPACITY	22427
6	2019-10	CAPACITY	23605
7	2019-11	CAPACITY	24263
8	2019-12	CAPACITY	24243
9	2020-01	CAPACITY	25533
10	2020-02	CAPACITY	24467
11	2020-03	CAPACITY	25194
12	2019-04	DEMAND	46193
13	2019-05	DEMAND	49131
14	2019-06	DEMAND	50124
15	2019-07	DEMAND	48850
16	2019-08	DEMAND	47602
17	2019-09	DEMAND	43697
18	2019-10	DEMAND	41058
19	2019-11	DEMAND	37364
20	2019-12	DEMAND	34364
21	2020-01	DEMAND	34149
22	2020-02	DEMAND	25573
23	2020-03	DEMAND	25284
24	2019-04	UNMET DEMAND	16930
25	2019-05	UNMET DEMAND	21094
26	2019-06	UNMET DEMAND	28528
27	2019-07	UNMET DEMAND	22955
28	2019-08	UNMET DEMAND	21789
29	2019-09	UNMET DEMAND	21270
30	2019-10	UNMET DEMAND	17453
31	2019-11	UNMET DEMAND	13101
32	2019-12	UNMET DEMAND	10121

	DATE	Metric	Value
33	2020-01	UNMET DEMAND	8616
34	2020-02	UNMET DEMAND	1106
35	2020-03	UNMET DEMAND	90

To simplify the data transformation process in the graphs, we will deviate from the "yyyy-mm" format for the date. Instead, we will create two separate columns, one for the year and another for the abbreviated name of the month. This adjustment will streamline our visualization efforts by reducing the need for extensive data transformations within the graphs themselves.

In [39]:
```
# Transform the column into datetime format
melted_table['DATE'] = pd.to_datetime(melted_table['DATE'])

# Extracting year and month
melted_table['year'] = melted_table['DATE'].dt.year
melted_table['month'] = melted_table['DATE'].apply(lambda x: x.strftime('%b'))
```

In [40]:
```
# The DATE column is no longer useful

melted_table.drop('DATE', axis = 1, inplace = True)
melted_table.head()
```

Out[40]:

	Metric	Value	year	month
0	CAPACITY	29263	2019	Apr
1	CAPACITY	28037	2019	May
2	CAPACITY	21596	2019	Jun
3	CAPACITY	25895	2019	Jul
4	CAPACITY	25813	2019	Aug

To further avoid data manipulation within the chart code, we will create auxiliary tables.

In [41]:
```
# Making new sets of data

# Just 2019
table_2019  = melted_table[melted_table['year'].isin([2019])]
# Just Demand from 2019
demand_2019 = table_2019[table_2019['Metric'].isin(['DEMAND'])]
# Just Capacity from 2019
capacity_2019 = table_2019[table_2019['Metric'].isin(['CAPACITY'])]
# Just Unmet Demand from 2019
unmet_2019 = table_2019[table_2019['Metric'].isin(['UNMET DEMAND'])]
# Demand and Capacity from 2019
bar_table = table_2019[table_2019['Metric'].isin(['CAPACITY', 'DEMAND'])]
# Unmet Demand and Capacity from 2019
stacked_table = table_2019[table_2019["Metric"].isin(["CAPACITY", "UNMET DEMAND"
```

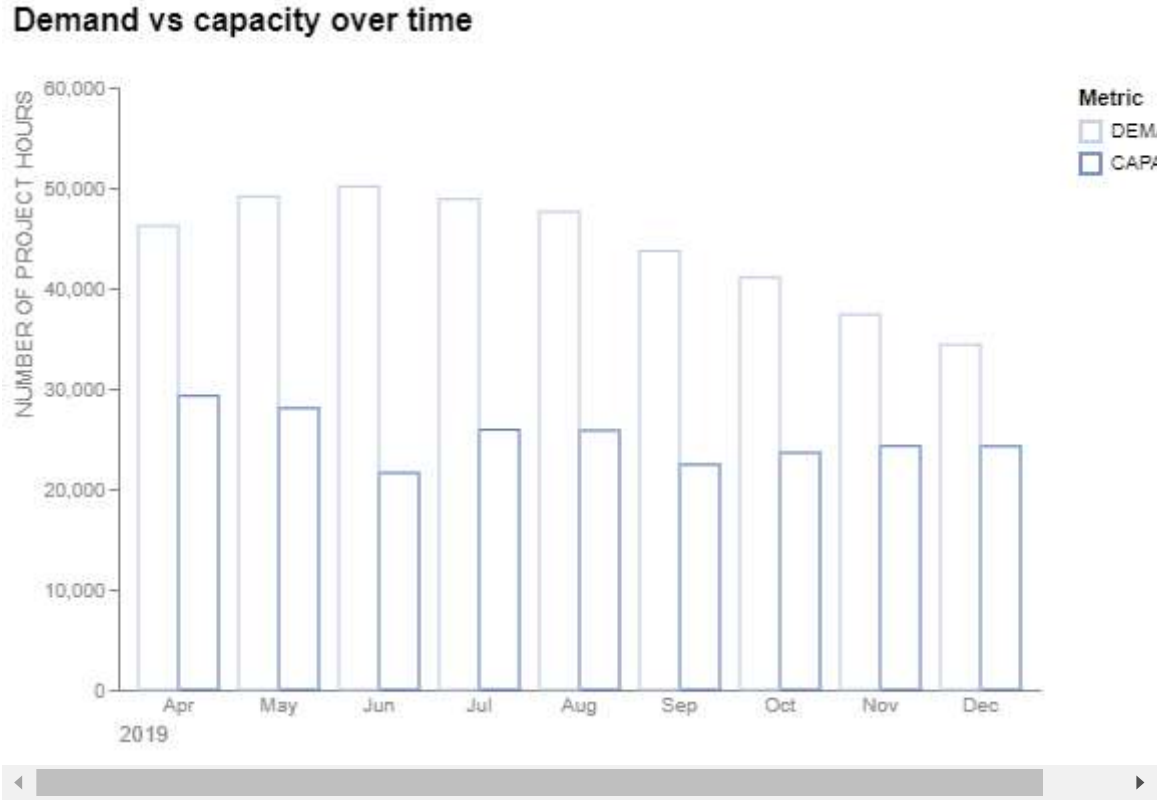
Bar chart

While the author deliberately filled the Capacity columns while leaving Demand only outlined in the attempt to visually distinguish between what can be fulfilled (Capacity) and the unmet portion of the requirement (Unmet Demand), `Altair` is not easily compatible with that choice.

The variable which dictates if the mark will be filled does not accept a condition as its value. Since the author itself admits the shortcomings of this approach ("*I find the outline plus the white space between the bars visually jarring*"), we chose to differentiate the data by color, as it is traditional.

In [42]:
```
# Unfilled version
alt.Chart(
    bar_table,
    title = alt.Title(
        "Demand vs capacity over time", anchor = "start", offset = 20, fontSize
    ),
).mark_bar(filled = False).encode( # Filled = False makes the bars unfilled
    y = alt.Y(
        "Value",
        axis = alt.Axis(
            grid = False, # Removes grid
            titleAnchor = "end",
            labelColor = "#888888", # Changes the label color to gray
            titleColor = "#888888", # Changes the title color to gray
            titleFontWeight = "normal"
        ),
        scale = alt.Scale(domain = [0, 60000]), # y-axis goes from 0 to 60000
        title = "NUMBER OF PROJECT HOURS"
    ),
    x = alt.X(
        "month",
        sort = None,
        axis = alt.Axis(
            labelAngle = 0, # Makes label horizontal
            titleAnchor = "start",
            labelColor = "#888888", # Changes label and title color to gray
            titleColor = "#888888",
            titleFontWeight = "normal",
            ticks = False # Removes ticks from axis
        ),
        title = "2019"
    ),
    color = alt.Color( # Sets colors based on Metric (Demand and Capacity)
        "Metric", scale = alt.Scale(range = ["#b4c6e4", "#4871b7"]), sort = "des
    ),
    xOffset = alt.XOffset("Metric", sort = "descending") # Sets offset on the x-
).configure_view(
    stroke = None
)  # Remove the chart border
```

Out[42]:

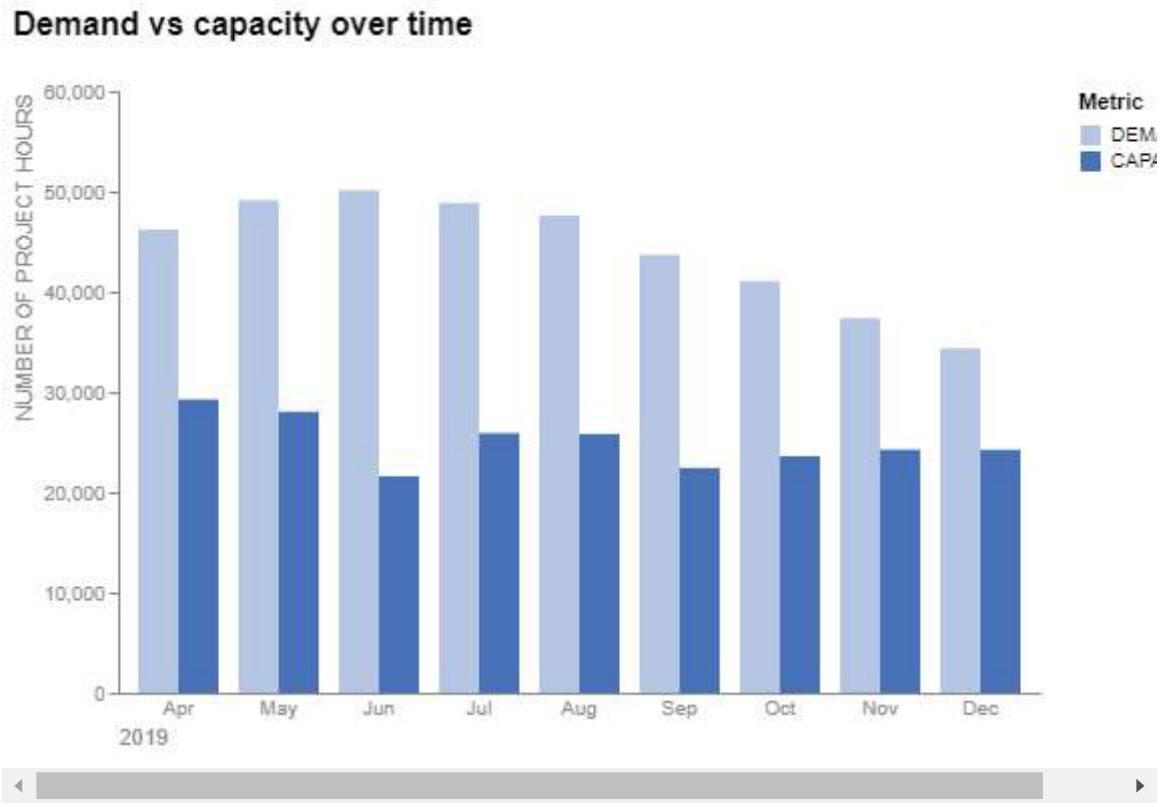


In [43]:

```
# Filled version
alt.Chart(
    bar_table,
    title = alt.Title(
        "Demand vs capacity over time", anchor = "start", offset = 20, fontSize
    ),
).mark_bar().encode( # Filled = True is default
    y = alt.Y(
        "Value",
        axis = alt.Axis(
            grid = False, # Removes grid
            titleAnchor = "end",
            labelColor = "#888888", # Changes the label color to gray
            titleColor = "#888888", # Changes the title color to gray
            titleFontWeight = "normal"
        ),
        scale = alt.Scale(domain = [0, 60000]), # y-axis goes from 0 to 60000
        title = "NUMBER OF PROJECT HOURS"
    ),
    x = alt.X(
        "month",
        sort = None,
        axis = alt.Axis(
            labelAngle = 0, # Makes label horizontal
            titleAnchor = "start",
            labelColor = "#888888", # Changes label and title color to gray
            titleColor = "#888888",
            titleFontWeight = "normal",
            ticks = False # Removes ticks from axis
        ),
        title = "2019"
    ),
    color = alt.Color( # Sets colors based on Metric (Demand and Capacity)
        "Metric", scale = alt.Scale(range = ["#b4c6e4", "#4871b7"]), sort = "des
    ),
```

```
    xOffset = alt.XOffset("Metric", sort = "descending") # Sets offset on the x-
).configure_view(
    stroke = None
)  # Remove the chart border
```

Out[43]:



Visualization as depicted in the book:

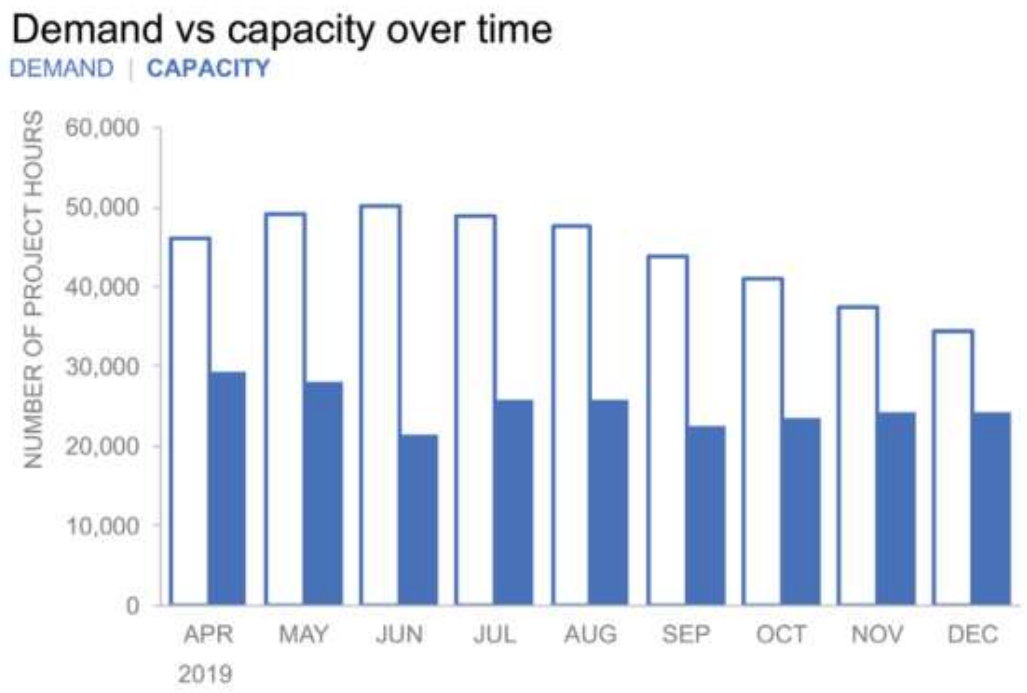


FIGURE 2.4a Basic bars

Line graph

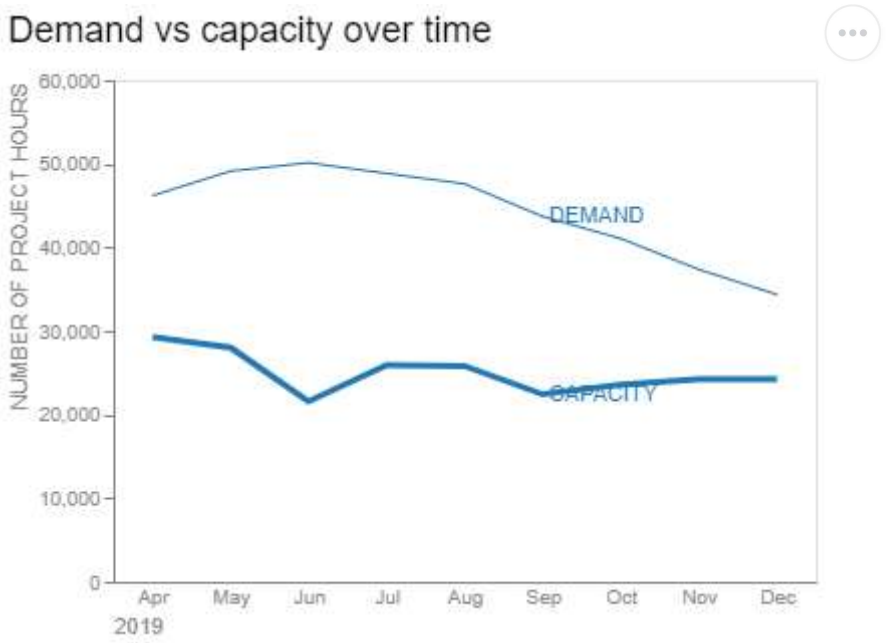
Cleaner than the bar chart, the next step was to convey the data using the line graph, with the labeling beside each line, along with the final value of the year. This helps the

viewer to visualize the difference between the capacity and the demand.

In [44]:
```
line = (
    alt.Chart(
        bar_table,
        title = alt.Title(
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10 # Offsets the title in the y-axis
        ),
    )
    .mark_line()  # Using a line mark for the chart
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,  # Disabling sorting for better time representation
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                labelColor = "#888888", # Set colors to gray
                titleColor = "#888888",
                titleFontWeight = "normal",
                ticks = False
            ),
            title = "2019"
        ),
        color = alt.Color(
            "Metric",
            scale = alt.Scale(range = ["#1f77b4", "#1f77b4"]),
            legend = None
        ),
        strokeWidth = alt.condition(
            "datum.Metric == 'CAPACITY'", alt.value(3), alt.value(1)
        )  # Adjusting line thickness based on the metric
    )
    .properties(width = 350, height = 250) # Set size of the graph
)

# Adding labels
label = (
    alt.Chart(bar_table)
    .mark_text(align = "left", dx = 3)
    .encode(
        x = alt.X("month", sort = None, aggregate = "max"),
        y = alt.Y("Value", aggregate = {"argmax": "month"}),
```

```
        text = alt.Text("Metric"), # The text itself is the Metric
        color = alt.Color("Metric", scale = alt.Scale(range = ["#1f77b4", "#1f77
    )
)

# Combining the line chart and labels
line + label
```

Out[44]:



As it is possible to notice, defining the label position as the maximum argument of the y-axis did not yield the intended result. This is because Altair is considering the values in an alphabetical order (making Sept the last month), even when setting `sort = None` in the x-axis.

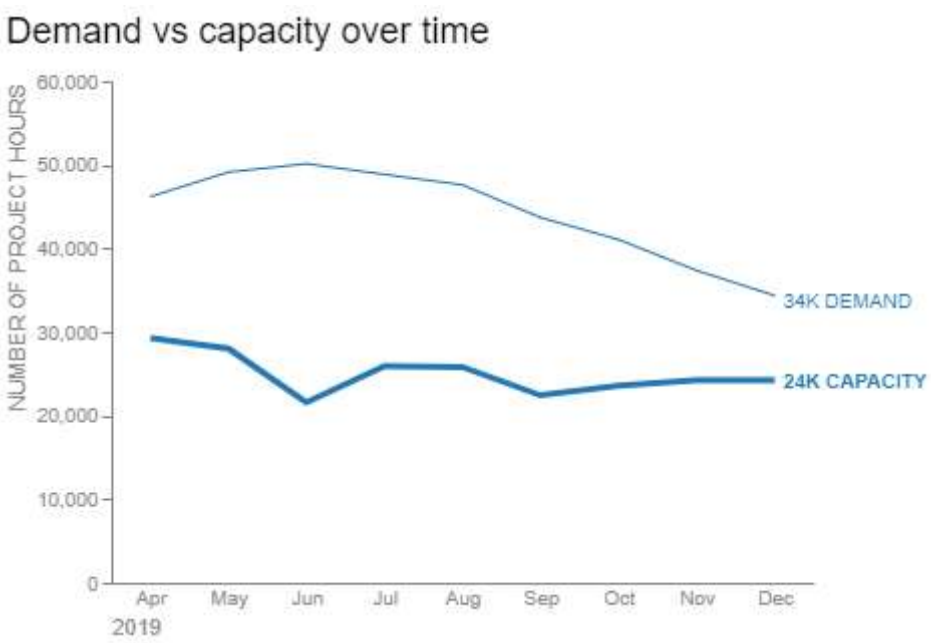
Since documentation fixing this issue was not found, the next approach was adding the label manually. This also assist the process of adding the value next to the metric.

In [45]:

```
# Demand label
label1 = alt.Chart({"values":
                    [{"text":  ['34K DEMAND']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 160, dy = -15,
                                color = '#1f77b4' # Color it blue
                                ).encode(text = "text:N")

# Capacity label
label2 = alt.Chart({"values":
                    [{"text":  ['24K CAPACITY']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 160, dy = 25,
                                color = '#1f77b4', # Color it blue
                                fontWeight = 'bold'
                                ).encode(text = "text:N")
```

```
line_final = line + label1 +  label2
line_final.configure_view(stroke = None)
```

Out[45]:



Visualization as depicted in the book:

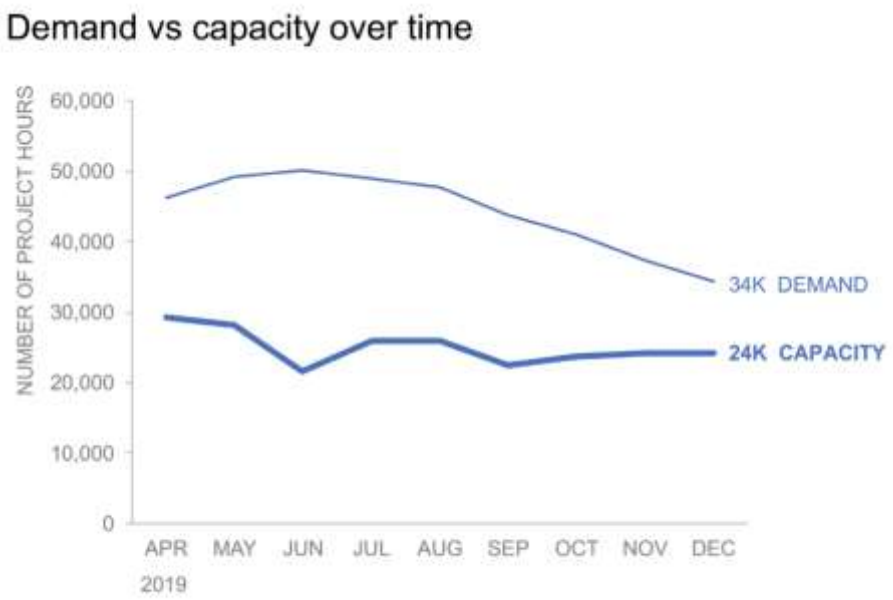


FIGURE 2.4b Line graph

Overlapping bars

The author now explores overlapping bars, wherein two bar graphs are positioned on top of each other, sharing the same axis. The Capacity data is displayed with transparency to prevent any potential confusion that might arise with a stacked bar chart.

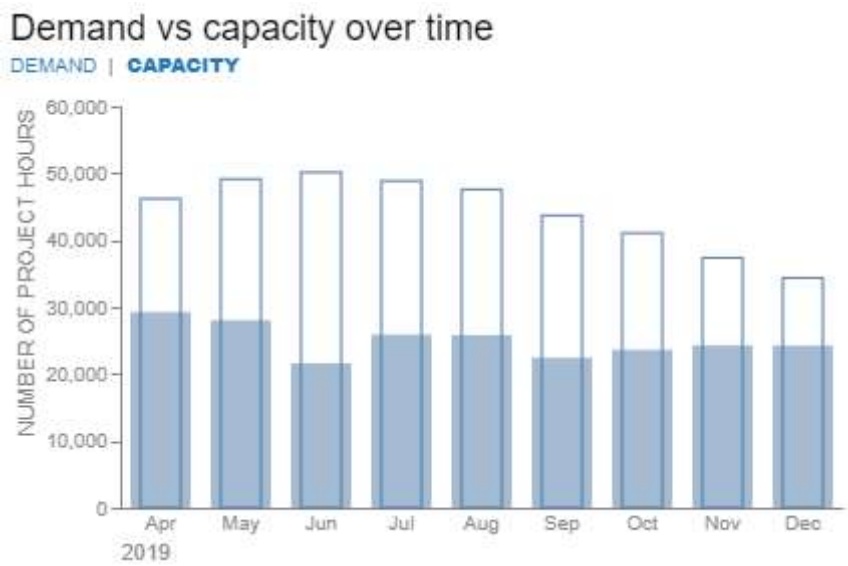
In this particular graph, our choice was to emulate the column labeling using a title with different colors, despite Altair not providing a straightforward method for such customization. Unlike previous examples where the default legend effectively distinguished colors, the current data distinction — "opaque" or "transparent" — is

better conveyed by utilizing normal or bold text in the title instead of relying on a legend with colors.

In [46]:
```
# Demand bar, with bigger spacing between them, unfilled
demand = (
    alt.Chart(
        demand_2019,
        width = alt.Step(40), # Defines the width of the bars (including distanc
        title = alt.Title(
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start"
        )
    )
    .mark_bar(filled = False) # Unfilled
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888", # Gray label and title
                titleColor = "#888888"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0, # Horizontal label
                titleAnchor = "start",
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False
            ),
            title = "2019"
        )
    )
)


# Capacity bar, bigger size and more transparency
capacity = (
    alt.Chart(capacity_2019)
    .mark_bar(size = 30) # Makes the bar thicker but keeps the distance the same
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal"
            ),
            scale = alt.Scale(domain = [0, 60000]),
```

```
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                ticks = False
            ),
            title = "2019"
        ),
        opacity = alt.value(0.5) # Makes the bar transparent
    )
)

# Labeling for subtitle
label1 = (
    alt.Chart({"values": [{"text": ["DEMAND  |"]}]})
    .mark_text(size = 10, align = "left", dx = -235, dy = -120, color = "#1f77b4
    .encode(text = "text:N")
)

label2 = (
    alt.Chart({"values": [{"text": ["CAPACITY"]}]})
    .mark_text(size = 10, align = "left", dx = -177, dy = -120, color = "#1f77b4
    .encode(text = "text:N")
)

overlap = capacity + demand + label1 + label2

# Sets space (padding) between bands
overlap.configure_scale(bandPaddingInner = 0.5).configure_view(stroke=None).prop
    height = 200
)
```

Out[46]:



Visualization as depicted in the book:

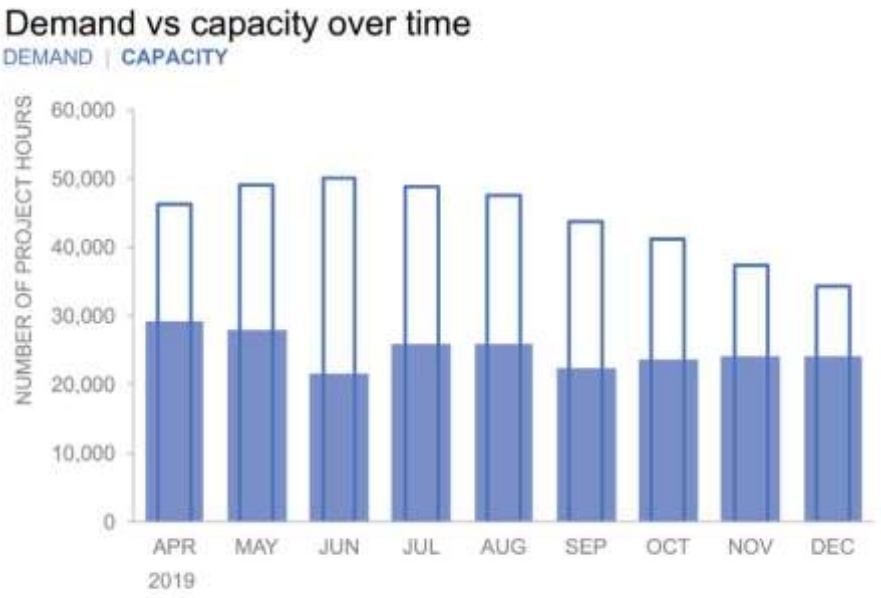


FIGURE 2.4c Overlapping bars

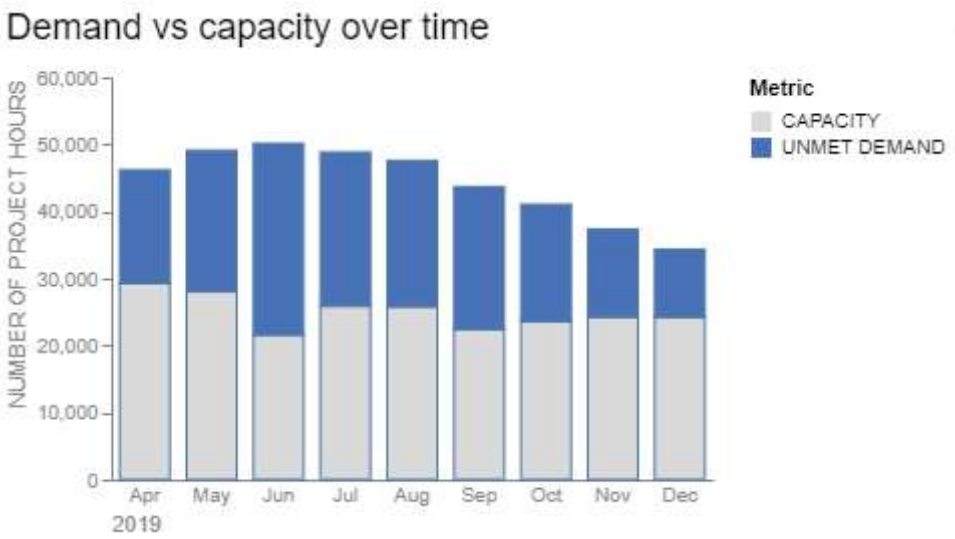
Stacked bars

In the stacked bars configuration, the Demand bar chart has been replaced with Unmet Demand (i.e., Demand - Capacity). This modification allows the stacking to represent the cumulative Demand value. Additionally, a color adjustment has been made, with Unmet Demand now rendered in a darker shade to emphasize its significance as a more meaningful metric.

In [47]:
```
# Stacked bar
bars = (
    alt.Chart(
        stacked_table,
        title = alt.Title(
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10 # Offsets title in the y-axis
        )
    )
    .mark_bar(size = 25)
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888", # Sets title and label to gray
                titleColor = "#888888",
                titleFontWeight = "normal"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
```

```
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                ticks = False
            ),
            title = "2019"
        ),
        color = alt.Color("Metric", scale = alt.Scale(range = ["#d9dad9", "#4871
        order = alt.Order("Metric", sort = "ascending") # Unmet demand on top
    )
)

# Border detail, makes the graph more visible
border = (
    alt.Chart(stacked_table)
    .mark_bar(size = 25, filled = False) # Makes an unfilled bar
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "start",
                labelColor = "#888888",
                titleColor = "#888888"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False
            ),
            title = "2019"
        ),
        order = alt.Order("Metric", sort = "ascending"),
    )
)

stacked = bars + border
stacked.configure_view(stroke = None).properties(width = 300, height = 200)
```

Out[47]:



Dot plot

For the next graph, a dot plot, the author reveals the challenges she had in Excel. To create the circles, she employed data markers from two line graphs, concealing the lines themselves. The region connecting the dots was achieved by employing a stacked bar of Unmet Demand, sitting on top of an transparent Capacity series.

This serves as a noteworthy example of the limitations of Excel when dealing with charts not inherently programmed into the tool. While certain graphs may be more straightforward in Excel, unconventional visualizations might demand intricate and obscure workarounds, while in Altair, where the approach to data visualization is more flexible, documentation for this graph was readily available.

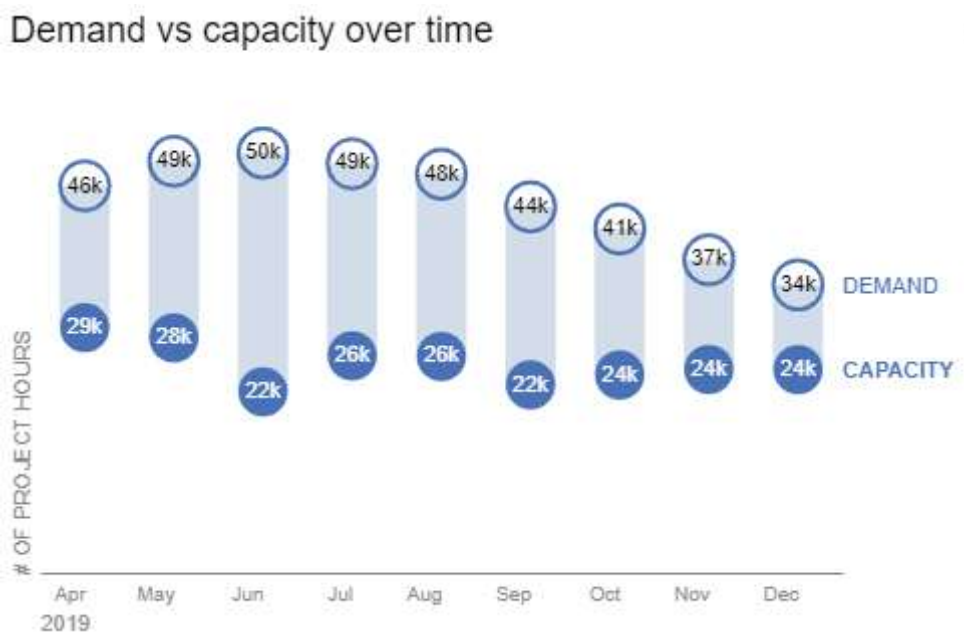
In [48]:

```
# Unfilled version
dots1 = (
    alt.Chart(
        bar_table,
        title = alt.Title(
            "Demand vs capacity over time", # Set title
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10
        )
    )
    .mark_circle(size = 600, opacity = 1) # Maximum opacity
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleFontWeight = "normal",
                titleColor = "#888888",
                titleAnchor = "start",
                ticks = False,
                labels = False, # Removes labels from axis
                domain = False # Removes line from axis
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "# OF PROJECT HOURS"
```

```
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                titleFontWeight = "normal",
                labelColor = "#888888",
                labelPadding = 10, # Makes label more distant to the axis
                titleColor = "#888888",
                ticks = False
            ),
            title = "2019"
        ),
        color = alt.Color("Metric", scale = alt.Scale(range = ["#4871b7"]), lege
    )
    .properties(width = 400, height = 250)
    .transform_filter(alt.datum.Metric == "CAPACITY") # Alternative way to filte
)

dots2 = ( # Same graph but to Demand
    alt.Chart(
        bar_table,
        title = alt.Title(
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10
        )
    )
    .mark_circle(size = 600, opacity = 1, filled = False) # Unfilled
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleFontWeight = "normal",
                titleColor = "#888888",
                titleAnchor = "start",
                ticks = False,
                labels = False,
                domain = False
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "# OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                titleFontWeight = "normal",
                labelColor = "#888888",
                labelPadding = 10, # Makes label more distant to axis
                titleColor = "#888888",
                ticks = False
            ),
```

```
            title = "2019"
        ),
        color = alt.Color("Metric", scale = alt.Scale(range = ["#b4c6e4"]), lege
    )
    .properties(width = 400, height = 250)
    .transform_filter(alt.datum.Metric == "DEMAND")
)

# Lines between dots
line = (
    alt.Chart(bar_table)
    .mark_line(strokeWidth = 25, opacity = 0.25) # Sets width and transparency
    .encode(x = alt.X("month", sort = None), y = "Value", detail = "month") # de
)                                                                            # in

# Text inside the dots
text = (
    alt.Chart(bar_table)
    .mark_text()
    .encode(
        x = alt.X("month", sort = None),
        y = "Value",
        text = alt.Text("Value:Q", format = ".2s"), # Formats 10000 as 10k
        color = alt.condition(
            alt.datum.Metric == "DEMAND", alt.value("black"), alt.value("white")
        )
    )
)

# Set legend for Metric
label1 = (
    alt.Chart({"values": [{"text": ["DEMAND"]}]})
    .mark_text(size = 11, align = "left", dx = 200, dy = -17, color = "#4871b7")
    .encode(text = "text:N")
)

label2 = (
    alt.Chart({"values": [{"text": ["CAPACITY"]}]})
    .mark_text(size = 11, align = "left", dx = 200, dy = 25, color = "#4871b7",
    .encode(text = "text:N")
)

dot_plot = line + dots1 + dots2 + text + label1 + label2
dot_plot.configure_view(stroke = None)
```

Out[48]:

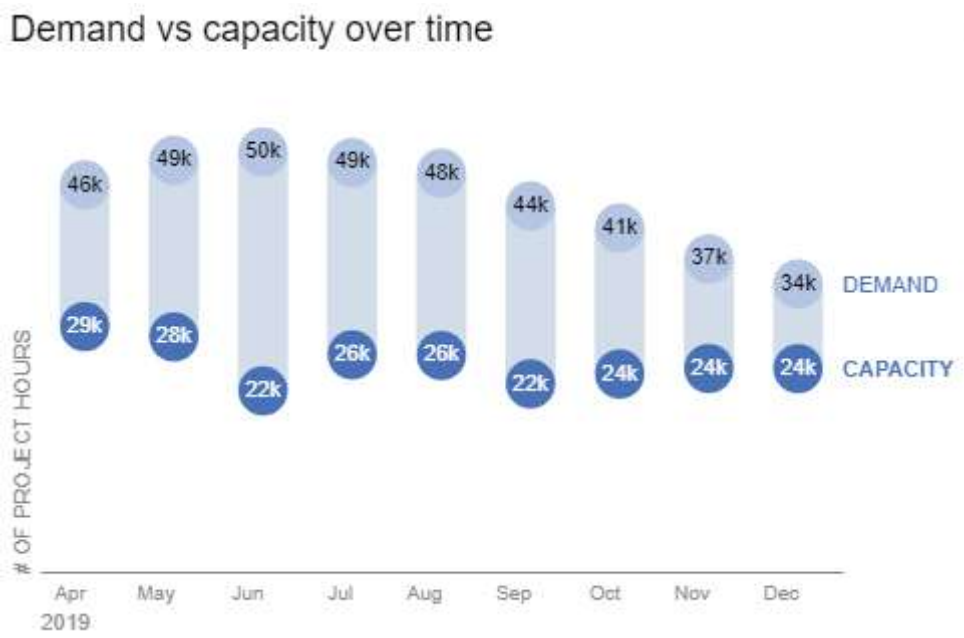


In [49]:

```
# Dot plot, filled-only version
dots = (
    alt.Chart(
        bar_table,
        title = alt.Title(
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10
        ),
    )
    .mark_circle(size = 600, opacity = 1) # Max opacity, filled by default
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleFontWeight = "normal",
                titleColor = "#888888",
                titleAnchor = "start",
                ticks = False,
                labels = False,
                domain = False
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "# OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                titleFontWeight = "normal",
                labelColor = "#888888",
                labelPadding = 10,
                titleColor = "#888888",
                ticks = False
            ),
```

```
            title = "2019"
        ),
        color = alt.Color(
            "Metric", scale = alt.Scale(range = ["#4871b7", "#b4c6e4"]), legend
        )
    )
    .properties(width = 400, height = 250)
)

dot_plot = line + dots + text + label1 + label2
dot_plot.configure_view(stroke = None)
```

Out[49]:



Visualization as depicted in the book:

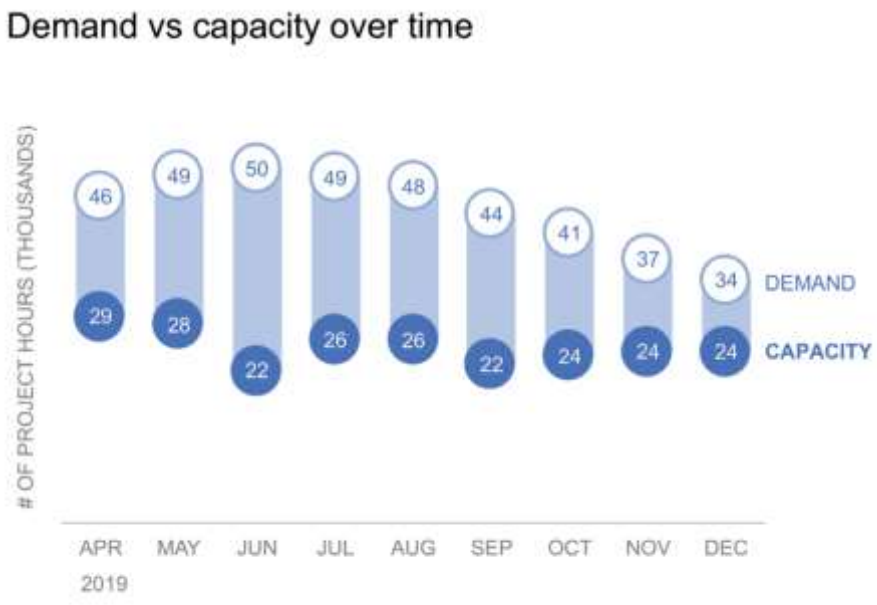


FIGURE 2.4e Dot plot

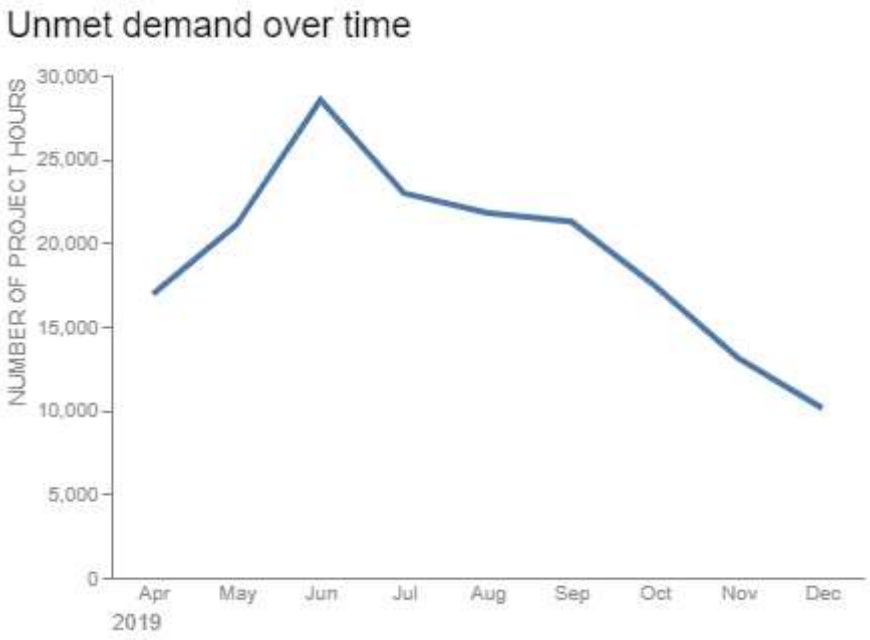
Graph the difference

For the final visualization, it was chosen a simple line plot representing the unmet demand. Although minimalist and clean, this choice occults data from the actual value of

demand and capacity.

In [50]:
```
alt.Chart(
    unmet_2019,
    title = alt.Title(
        "Unmet demand over time",
        fontSize = 18,
        fontWeight = "normal",
        anchor = "start",
        offset = 10
    ),
).mark_line().encode(
    y = alt.Y(
        "Value",
        axis = alt.Axis(
            grid = False,
            titleAnchor = "end",
            labelColor = "#888888",
            titleColor = "#888888",
            titleFontWeight = "normal"
        ),
        title = "NUMBER OF PROJECT HOURS"
    ),
    x = alt.X(
        "month",
        sort = None,
        axis = alt.Axis(
            labelAngle = 0,
            titleAnchor = "start",
            labelColor = "#888888",
            titleColor = "#888888",
            titleFontWeight = "normal",
            ticks = False
        ),
        title="2019",
    ),
    strokeWidth = alt.value(3) # Set thickness of the line
).properties(
    width = 375, height = 250
).configure_view(
    stroke = None
)
```

Out[50]:



Visualization as depicted in the book:

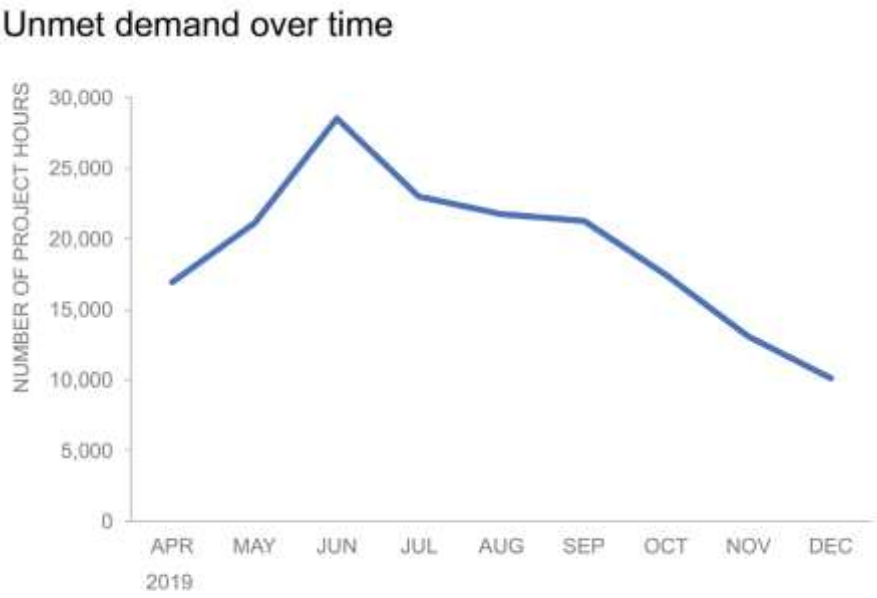


FIGURE 2.4f Graph the difference

Interactivity

The initial idea was to make an interactive version of the Overlapping graph where the opacity adjusts when the mouse in near the column, similarly to the one in Exercise 2.1. Despite the tooltip functioning correctly, the hover feature failed to identify all graphs in the layering. As a result, only the Demand bars were highlighted.

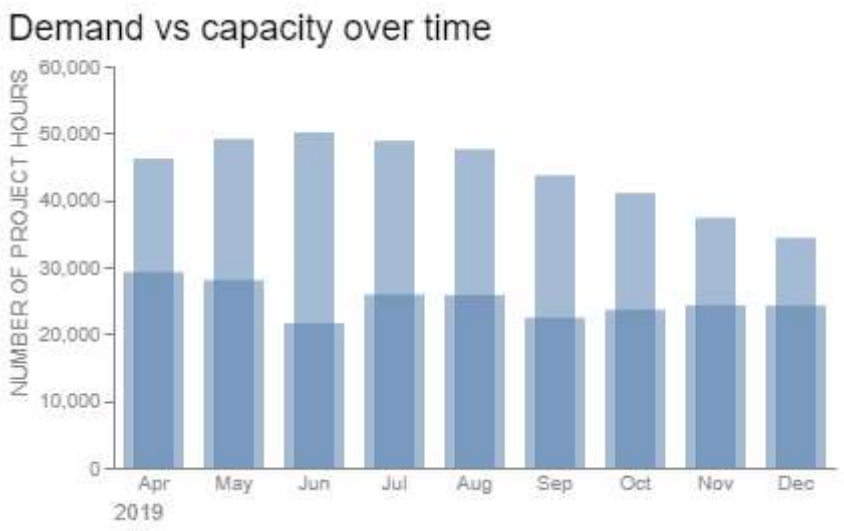
In [51]:

```
# Define hover
hover = alt.selection_point(on = "mouseover", nearest = True, empty = False)

demand = (
    alt.Chart(
        demand_2019,
        width = alt.Step(40),
        title = alt.Title(
```

```
            "Demand vs capacity over time",
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start"
        )
    )
    .mark_bar()
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888"
            ),
            scale=alt.Scale(domain = [0, 60000]),
            title="NUMBER OF PROJECT HOURS"
        ),
        x=alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False
            ),
            title = "2019"
        ),
        opacity = alt.condition(hover, alt.value(1), alt.value(0.5)),  # Set opa
        tooltip = ["Value:Q", "Metric"]
    )
    .add_params(hover)
)

# Capacity bar, bigger size and more transparency
capacity = (
    alt.Chart(capacity_2019)
    .mark_bar(size = 30)
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
```

```
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                ticks = False
            ),
            title = "2019"
        ),
        opacity = alt.condition(hover, alt.value(1), alt.value(0.5)),
        tooltip = ["Value:Q", "Metric"]
    )
    .add_params(hover)
)

overlap = demand + capacity

overlap.configure_scale(bandPaddingInner=0.5).configure_view(stroke=None).proper
    height=200
)
```

Out[51]:



Due to a lack of documentation about interactivity in layered charts, no solution to this problem was found.

The final interactive graph for this exercise was the line graph. Rather than keeping the "Capacity and Demand" graph distinct from the "Unmet Demand" one, we opted for a consolidated approach with three lines. Users can now choose the data to emphasize through a dropdown box. Additionally, each year features a tooltip marked by a point for enhanced clarity.

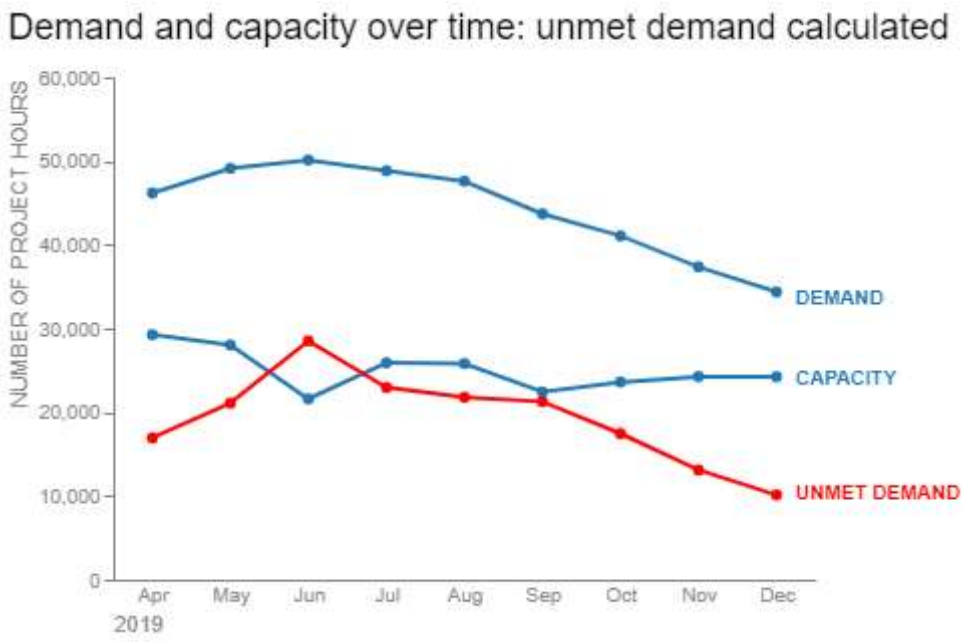
In [52]:

```
# Sorts so that the metrics in the tooltip aligns with the graphs order
custom_order = ["DEMAND", "CAPACITY", "UNMET DEMAND"]
sorted_metrics = [metric for metric in custom_order if metric in table_2019["Met

# Creates the dropdown
dropdown = alt.binding_select(options = list(sorted_metrics), name = "SELECT LIN
selection = alt.selection_point(fields = ["Metric"], bind = dropdown)

line = (
    alt.Chart(
        table_2019,
        title = alt.Title(
            "Demand and capacity over time: unmet demand calculated", # Changes
```

```
            fontSize = 18,
            fontWeight = "normal",
            anchor = "start",
            offset = 10
        ),
    )
    .mark_line(point = True)  # Create a point at each month
    .encode(
        y = alt.Y(
            "Value",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal"
            ),
            scale = alt.Scale(domain = [0, 60000]),
            title = "NUMBER OF PROJECT HOURS"
        ),
        x = alt.X(
            "month",
            sort = None,  # Disabling sorting for better time representation
            axis = alt.Axis(
                labelAngle = 0,
                titleAnchor = "start",
                labelColor = "#888888", # Set colors to gray
                titleColor = "#888888",
                titleFontWeight = "normal",
                ticks = False
            ),
            title = "2019"
        ),
        color = alt.Color(
            "Metric",
            scale = alt.Scale(range = ["#1f77b4", "#1f77b4", "red"]), # Unmet de
            legend = None
        ),
        opacity = alt.condition(
            selection, alt.value(1), alt.value(0.1)
        ),  # Adjusting opacity based on the dropdown
        tooltip = ["Metric", "month", "Value"] # Sets tooltip
    )
    .properties(width = 350, height = 250)
).add_params(selection)


# Demand label
label1 = alt.Chart({"values":
                    [{"text":  ['DEMAND']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 165, dy = -15,
                                color = '#1f77b4', # Color it blue
                                fontWeight = 700 # Bold font
                                ).encode(
                                    text = "text:N",
                                    opacity = alt.condition(
                                            selection,
```

```
                                        alt.value(1),
                                        alt.value(0)
                                        ) # Label disappears when any line i
                                    )

# Capacity label
label2 = alt.Chart({"values":
                    [{"text":  ['CAPACITY']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 165, dy = 25,
                                color = '#1f77b4', # Color it blue
                                fontWeight = 700
                                ).encode(
                                    text = "text:N",
                                    opacity = alt.condition(
                                            selection,
                                            alt.value(1),
                                            alt.value(0)
                                            ) # Label disappears when any line i
                                    )
# Unmet demand label
label3 = alt.Chart({"values":
                    [{"text":  ['UNMET DEMAND']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 165, dy = 82,
                                color = 'red', # Color it blue
                                fontWeight = 700
                                ).encode(
                                    text = "text:N",
                                    opacity = alt.condition(
                                            selection,
                                            alt.value(1),
                                            alt.value(0)
                                            ) # Label disappears when any line i
                                    )

line_final = line + label1 + label2 + label3
line_final.configure_view(stroke = None)
```

Out[52]:



SELECT LINE: DEMAND

Exercise 2.5 - How would you show this data?

This exercise shows a table and asks the viewer how would they show the data. We will replicate the visual answers given by the author.

Loading the data

In [53]:
```
# Loading considering the NaN caused by Excel formatting
table = pd.read_excel(r"Data\2.5 EXERCISE.xlsx", usecols = [1, 2], header = 5)
table
```

Out[53]:

	Year	Attrition Rate
0	2019	0.0910
1	2018	0.0820
2	2017	0.0450
3	2016	0.1230
4	2015	0.0560
5	2014	0.1510
6	2013	0.0700
7	2012	0.0100
8	2011	0.0200
9	2010	0.0970
10	AVG	0.0745

First, we will drop the "AVG" (Average) column, as it will not be a data point in our graphs. It is better to calculate it separately when needed.

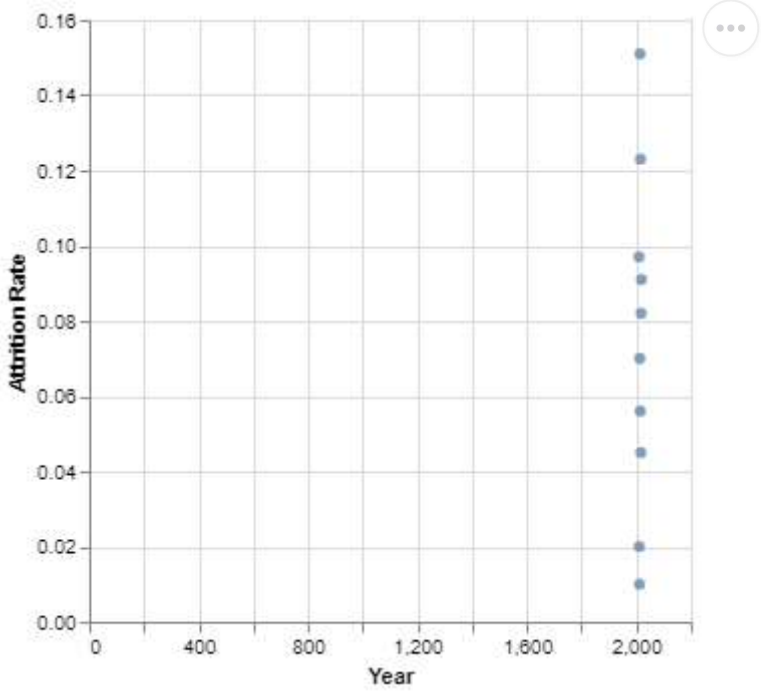
In [54]:
```
table.drop(10, inplace = True)
```

Dot plot

When attempting the first scatter plot, we realize that Altair incorrectly classifies the data type of the "Year" column. This can be fixed by specifying the correct date type (:O, as of, Ordinary).

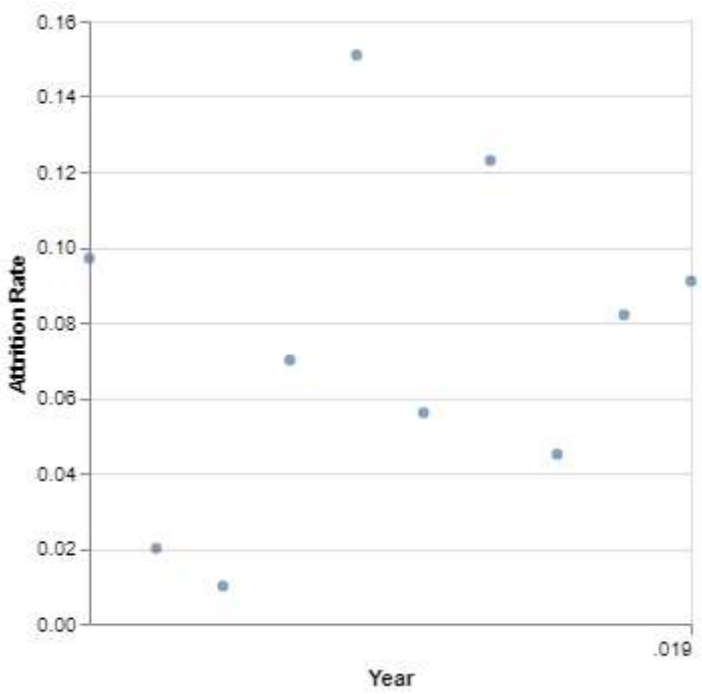
In [55]:
```
# Without data type

alt.Chart(table).mark_point(filled = True).encode(
    x = alt.X('Year'),
    y = alt.Y('Attrition Rate')
    )
```

Out[55]:



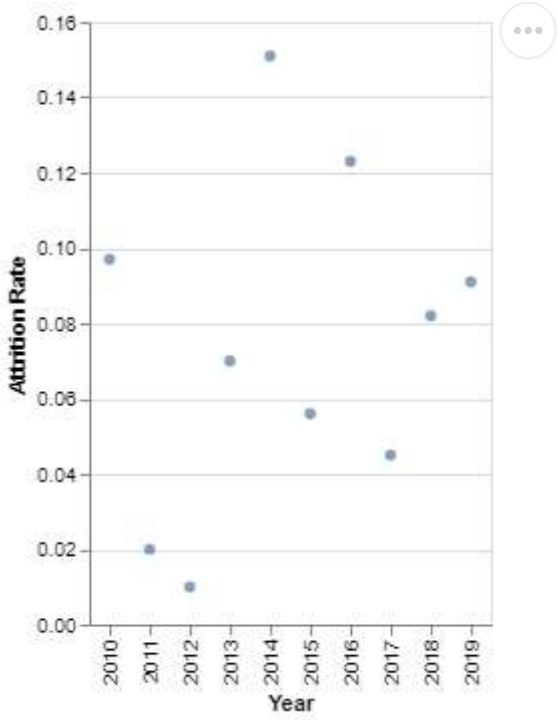
In [56]:
```
# With data type equals temporal

alt.Chart(table).mark_point(filled = True).encode(
    x = alt.X('Year:T'),
    y = alt.Y('Attrition Rate')
    )
```

Out[56]:



In [57]:
```
# With data type equals ordinal

alt.Chart(table).mark_point(filled = True).encode(
    x = alt.X('Year:O'),
    y = alt.Y('Attrition Rate')
    )
```

Out[57]:

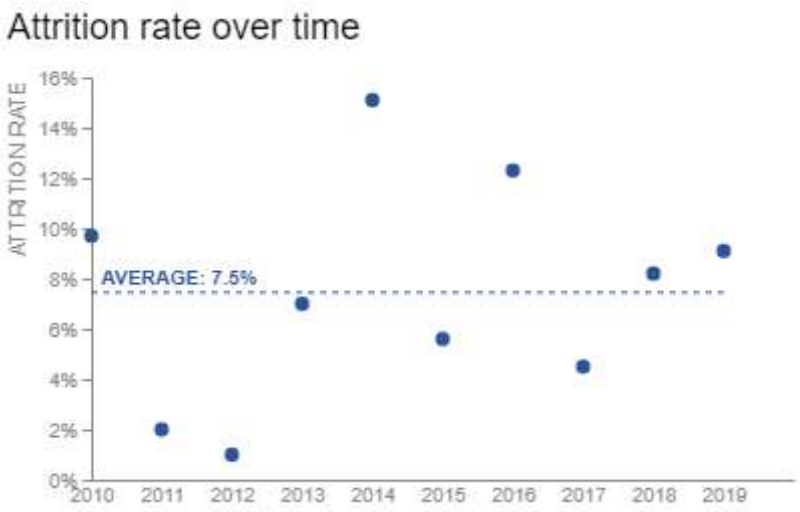


Initially, we will create a dot plot to visually represent the data over time, incorporating an average line to facilitate comparison.

In [58]:
```
# Create the base graph with title
base = alt.Chart(
    table,
    title = alt.Title(
        "Attrition rate over time",
```

```
        fontSize = 18,
        fontWeight = "normal",
        anchor = "start",
        offset = 10
    )
)

# Make the filled dots
dots = base.mark_point(filled = True, size = 50, color = "#2c549d").encode(
    x = alt.X(
        "Year:O",
        axis = alt.Axis(labelAngle = 0, labelColor = "#888888", ticks = False),
        title = None,
        scale = alt.Scale(align = 0) # Align the first dot with the y-axis (0 at
    ),
    y = alt.Y(
        "Attrition Rate",
        axis = alt.Axis(
            grid = False,
            titleAnchor = "end",
            labelColor = "#888888",
            titleColor = "#888888",
            tickCount = 9, # Set fixed number of ticks (intervals)
            format = "%", # y-axis data is a percentage
            titleFontWeight = "normal"
        ),
        title = "ATTRITION RATE"
    ),
    opacity = alt.value(1) # Maximum opacity
)

# Makes a line at the average value
# strokeDash defines how dotted is the line
rule = base.mark_rule(color = "#2c549d", strokeDash = [3, 3]).encode(
    x = alt.value(0), x2 = alt.value(315), y = "mean(Attrition Rate)"
)

# Text above the average line
label = (
    alt.Chart({"values": [{"text": ["AVERAGE: 7.5%"]}]})
    .mark_text(size = 10, align = "left", dx = -170, dy = 0, color = "#2c549d",
    .encode(text = "text:N")
)

final_dots = dots + rule + label
final_dots.properties(width=350, height=200).configure_view(stroke=None)
```

Out[58]:



Visualization as depicted in the book:

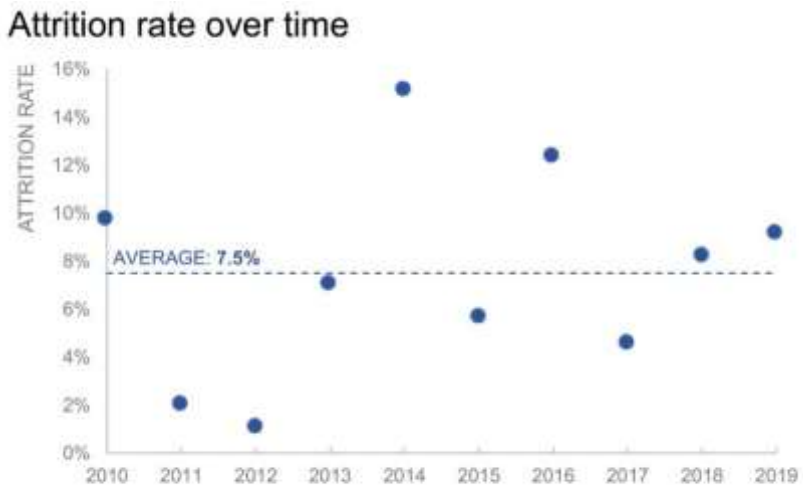


FIGURE 2.5b Dot plot

Line graph

Next, we will link the dots with a line, aiding in the comparison of value differences.

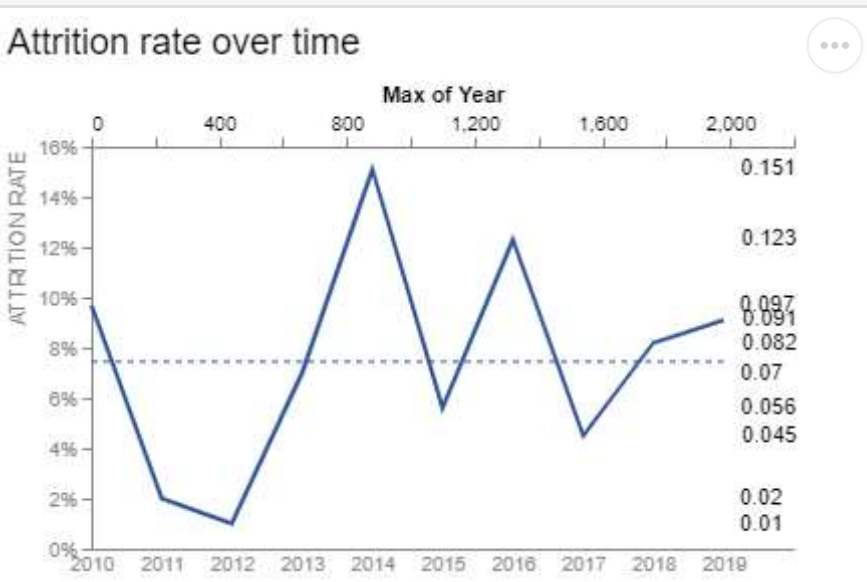
Once more, omitting the data type in the label specification causes the labels to accumulate on the right side of the graph.

In [59]:

```
line = base.mark_line(color = "#2c549d").encode(
    x = alt.X(
        "Year:O",
        axis = alt.Axis(labelAngle = 0, labelColor = "#888888", ticks = False),
        title = None,
        scale = alt.Scale(align = 0)
    ),
    y = alt.Y(
        "Attrition Rate",
        axis = alt.Axis(
            grid = False,
            titleAnchor = "end",
            labelColor = "#888888",
            titleColor = "#888888",
            tickCount = 9,
```

```
            format = "%",
            titleFontWeight = "normal"
        ),
        title = "ATTRITION RATE"
    )
)

# Label without specifying data type
label = base.mark_text(align = "left", dx = 3).encode(
    x = alt.X("Year", aggregate = "max"),
    y = alt.Y("Attrition Rate", aggregate = {"argmax": "Year"}),
    text = alt.Text("Attrition Rate")
)

final_line = line + rule + label

final_line.properties(width = 350, height = 200).configure_view(stroke = None)
```

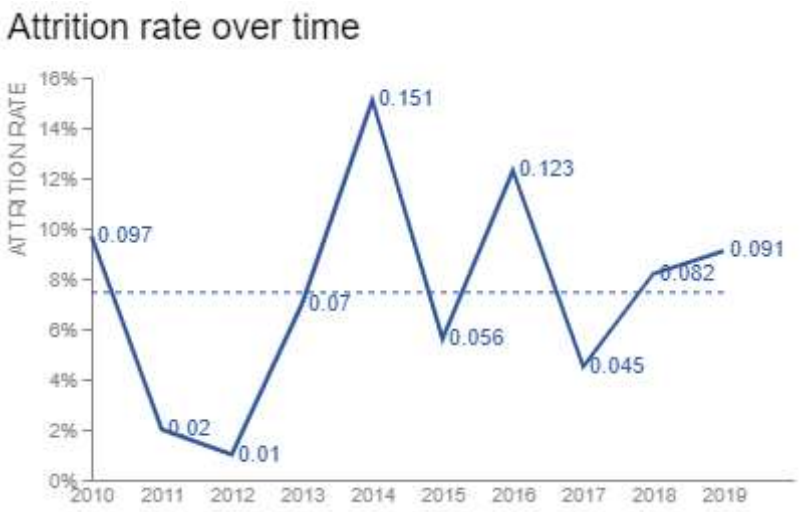
Out[59]:



In [60]:

```
# With data type
label = base.mark_text(align = "left", dx = 3, color = "#2c549d").encode(
    x = alt.X("Year:O", aggregate = "max"),
    y = alt.Y("Attrition Rate", aggregate = {"argmax": "Year"}),
    text = alt.Text("Attrition Rate")
)

final = line + rule + label

final.properties(width=350, height=200).configure_view(stroke=None)
```

Out[60]:



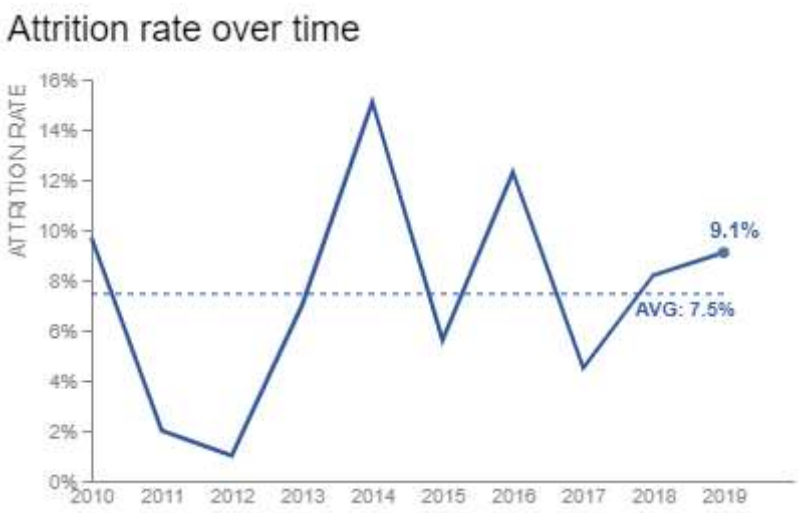
The default method for placing the end label appeared ineffective, as it failed to filter out 2019 as the maximum value in the Year column. This issue can be rectified by straightforwardly filtering the entire dataset to encompass only values where "Year == 2019".

In [61]:

```
# Filter 2019 manually

label = base.mark_text(align = 'left', dx = 3, color = '#2c549d', fontWeight = '
    x = alt.X('Year:O'),
    y = alt.Y('Attrition Rate'),
    text = alt.Text('Attrition Rate', format = ".1%"),
    xOffset = alt.value(-10), # Offsets slightly in the x and y-axis
    yOffset = alt.value(-10)
).transform_filter( # Filters
    alt.FieldEqualPredicate(field = 'Year', equal = 2019)
    )

# Label for the Average
label2 = alt.Chart({"values":
                    [{"text":  ['AVG: 7.5%']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 96, dy = 15,
                                color = '#2c549d',
                                fontWeight = 'bold'
                                ).encode(text = "text:N")


# Filled point at the end of the line
point = base.mark_point(filled = True).encode(
    x = alt.X('Year:O'),
    y = alt.Y('Attrition Rate'),
    opacity = alt.value(1)
).transform_filter(
    alt.FieldEqualPredicate(field = 'Year', equal = 2019)
    )

final_line = line + rule + label + label2 + point

final_line.properties(
    width = 350,
```

```
    height = 200
).configure_view(stroke = None)
```

Out[61]:



Visualization as depicted in the book:

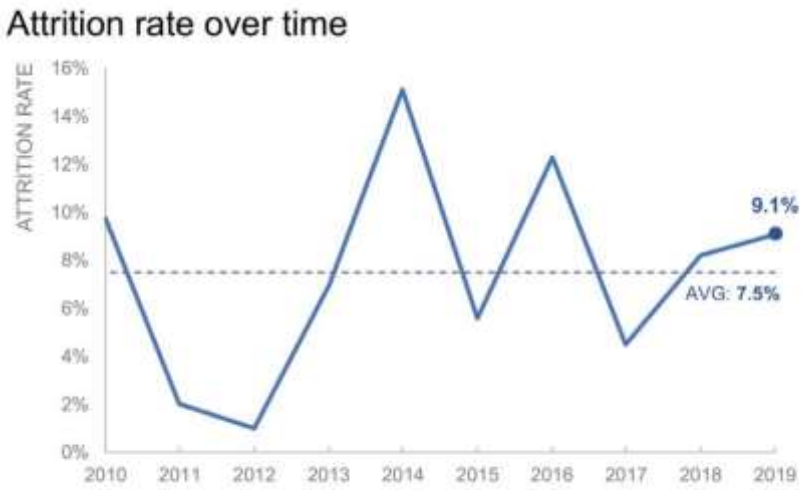


FIGURE 2.5c Line graph

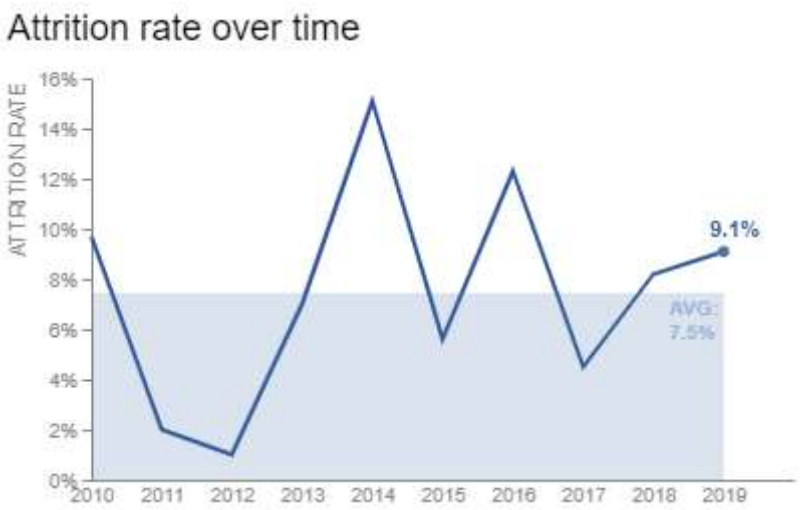
Coloring below the average line may help highlight values below it.

In [62]:

```
# Calculates average
avg = table['Attrition Rate'].mean()

# Creates a rectangle below the average line
rect = alt.Chart(pd.DataFrame({'y': [0], 'y2':[avg]})).mark_rect(
    opacity = 0.2
).encode(y = 'y', y2 = 'y2', x = alt.value(0), x2 = alt.value(315))

# Makes a different average label, in lighter color
label2 = alt.Chart({"values":
                    [{"text":  ['AVG:', '7.5%']}]
                    }
                    ).mark_text(size = 10,
                                align = "left",
                                dx = 113, dy = 15,
                                color = '#9fb5db',
                                fontWeight = 'bold'
                                ).encode(text = "text:N")
```

```
final_line2 = line + rect + label + label2 + point

final_line2.properties(
    width = 350,
    height = 200
).configure_view(stroke = None)
```

Out[62]:



Visualization as depicted in the book:

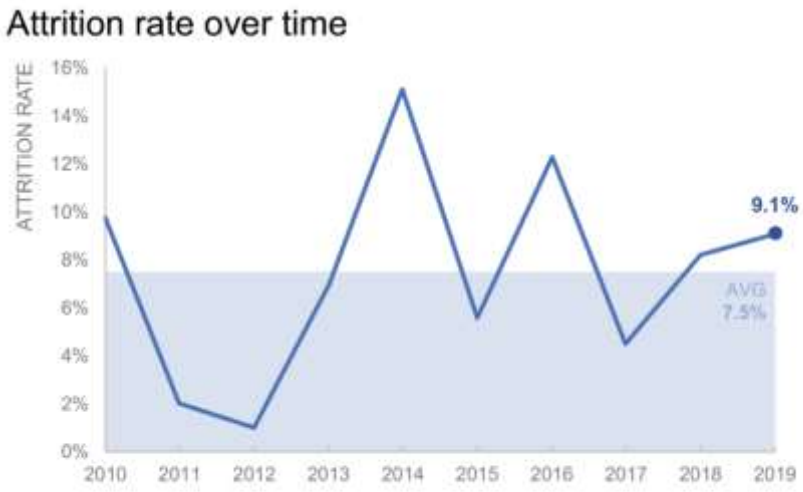


FIGURE 2.5d Line graph with shaded area depicting average

Area graph

An exploration using an area graph was undertaken; however, it conveys the impression that the area under the line holds significance, which is not the case for this dataset. This graph type may not be the most suitable choice for presenting this data.

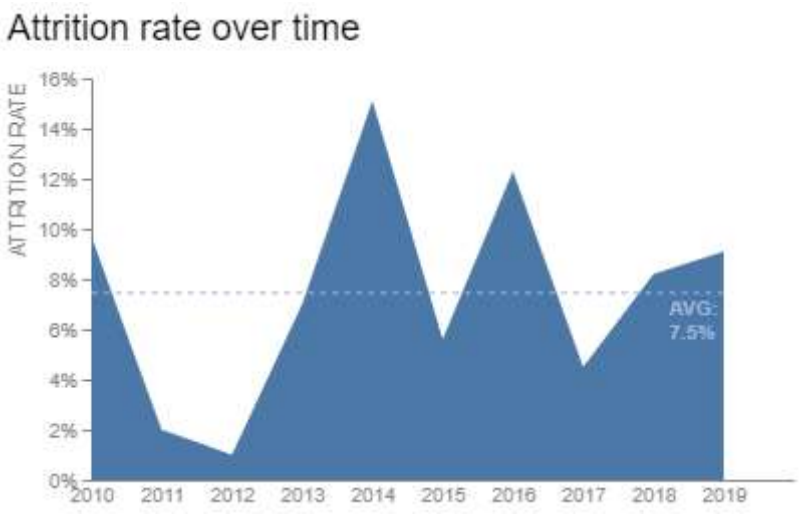
Also, the decision was made to stray from the example given and connect the area to the y-axis. It is unclear if the decision to have it separated for this graph only was on purpose or an error.

In [63]:

```
# Area graph
area = base.mark_area().encode(
    x = alt.X(
```

```
        "Year:O",
        axis = alt.Axis(labelAngle = 0, labelColor = "#888888", ticks = False),
        title = None,
        scale = alt.Scale(align = 0)
    ),
    y = alt.Y(
        "Attrition Rate",
        axis = alt.Axis(
            grid = False,
            titleAnchor = "end",
            labelColor = "#888888",
            titleColor = "#888888",
            tickCount = 9,
            format = "%",
            titleFontWeight = "normal"
        ),
        title = "ATTRITION RATE"
        )
)

# Creates a lighter rule to contrast with area graph
rule_light = base.mark_rule(color = "#9fb5db", strokeDash = [3, 3]).encode(
    x = alt.value(0), x2 = alt.value(315), y = "mean(Attrition Rate)"
)

final_area = area + rule_light + label2
final_area.properties(width = 350, height = 200).configure_view(stroke = None)
```

Out[63]:



Visualization as depicted in the book:

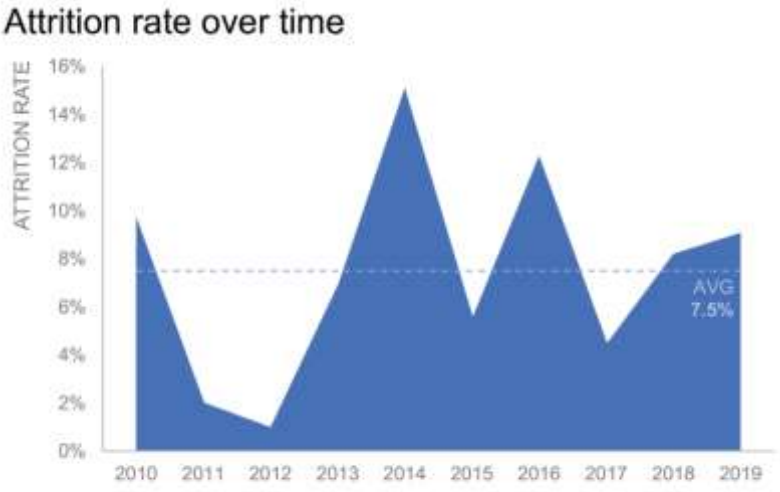


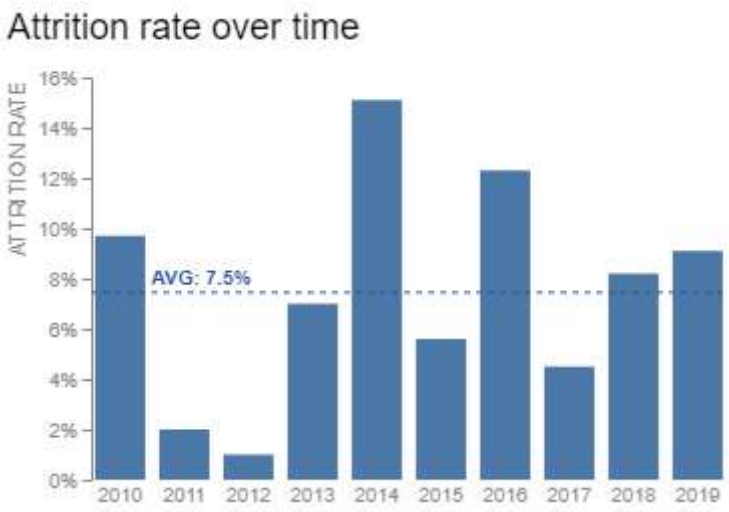
FIGURE 2.5e Area graph

Bar plot

Finally, we can do a classic bar plot.

In [64]:
```
# Bar plot
bar = base.mark_bar(size = 25).encode(
    x = alt.X(
        "Year:O",
        axis = alt.Axis(labelAngle = 0, labelColor = "#888888", ticks = False, d
        title = None,
        scale = alt.Scale(align = 0)
    ),
    y = alt.Y(
        "Attrition Rate",
        axis = alt.Axis(
            grid = False,
            titleAnchor = "end",
            labelColor = "#888888",
            titleColor = "#888888",
            tickCount = 9,
            format = "%",
            titleFontWeight = "normal"
        ),
        title = "ATTRITION RATE"
    ),
)

# Moves placement of average label
label = (
    alt.Chart({"values": [{"text": ["AVG: 7.5%"]}]})
    .mark_text(size = 10, align = "left", dx = -130, dy = 0, color = "#2c549d",
    .encode(text = "text:N")
)

final_bar = bar + rule + label
final_bar.properties(width = 320, height = 200).configure_view(stroke = None)
```

Out[64]:



Visualization as depicted in the book:

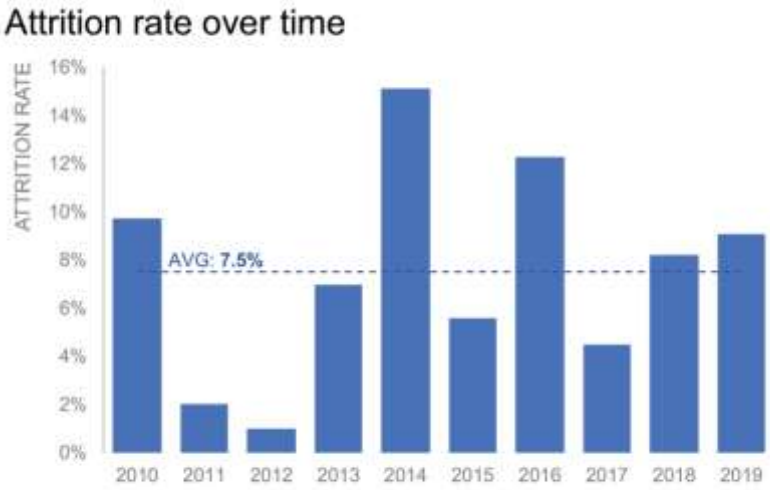


FIGURE 2.5f Bar graph

Interactive

[I can't think of anything cool now]

Chapter 3 - Identify and eliminate Cluster

"*This lesson is simple but the impact is huge: get rid of the stuff that doesn't need to be there*" - Cole Nussbaumer Knaflic

Exercise 3.2 - how can we tie words to the graph?

The main focus of this exercise is to apply the Gestalt Principles of Visual Perception to declutter graphs. For principles will be demonstrated, and each of them will be clarified through the visualization employing it.

Loading the data

In [65]:
```
# Loading considering the NaN caused by Excel formatting
table = pd.read_excel(r"Data\3.2 EXERCISE.xlsx", usecols = [1, 2, 3], header = 4
table
```

Out[65]:

	2019	Rate	# exits
0	JAN	0.0040	120
1	FEB	0.0010	30
2	MAR	0.0015	45
3	APR	0.0080	240
4	MAY	0.0030	90
5	JUN	0.0014	42
6	JUL	0.0044	132
7	AUG	0.0050	150
8	SEP	0.0022	66
9	OCT	0.0015	45
10	NOV	0.0005	15
11	DEC	0.0010	30

The column name for 2019 is currently an integer, which might pose issues in the future. To avoid potential complications, we will modify the column name to a string.

For example, trying to run the following code returns an error:

```
alt.Chart(table).mark_bar().encode(
        x = alt.X('2019'),
        y = alt.Y('Rate')
        )
ValueError: Dataframe contains invalid column name: 2019. Column names
must be strings
```

In [66]:
```
table.rename(columns = {2019:'Date'}, inplace = True)
```

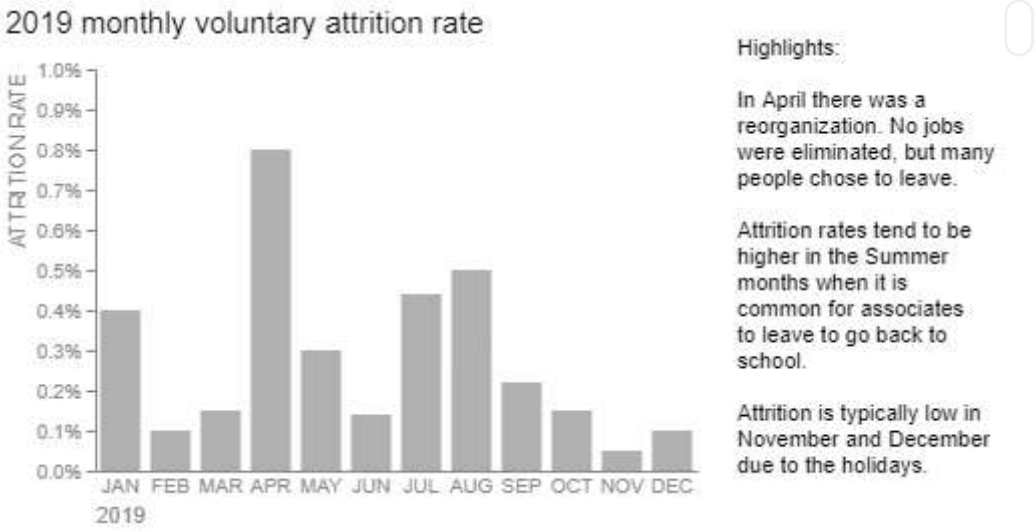
Cluttered graph

In [67]:
```
# Graph with cluttered text

bar = (
    alt.Chart(
        table,
        title = alt.Title(
            "2019 monthly voluntary attrition rate",
            fontSize = 15,
            anchor = "start",
            offset = 10,
            fontWeight = "normal"
        )
```

```
    )
    .mark_bar(size = 20, color = "#b0b0b0")
    .encode(
        x = alt.X(
            "Date",
            sort = None, # Avoids alphabetical order
            axis = alt.Axis(
                labelAngle = 0,
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False,
                titleAnchor = "start",
                titleFontWeight = "normal"
            ),
            title="2019"
        ),
        y = alt.Y(
            "Rate",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                format = "%", # y-axis is a percentage
                tickCount = 10 # Number of ticks (intervals) in axis
            ),
            scale = alt.Scale(domain = [0, 0.01]),
            title = "ATTRITION RATE"
        )
    )
    .properties(width = 300, height = 200)
)

# Text next to the graph
text = (
    alt.Chart(
        {
            "values": [
                {   # Each item is a line
                    "text": [
                        "Highlights:",
                        " ",
                        "In April there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                        " ",
                        "Attrition rates tend to be",
                        "higher in the Summer",
                        "months when it is",
                        "common for associates",
                        "to leave to go back to",
                        "school.",
                        " ",
                        "Attrition is typically low in",
                        "November and December",
                        "due to the holidays.",
                    ]
                }
```

```
            ]
        }
    )
    .mark_text(size = 11, align = "left", dy = -20, dx = -10) # Size and placeme
    .encode(text = "text:N")
)


# Using the | symbols makes it so Altair unites the bar and the text next to eac
final_cluttered = bar | text

final_cluttered.configure_view(stroke = None)
```

Out[67]:



Visualization as depicted in the book:

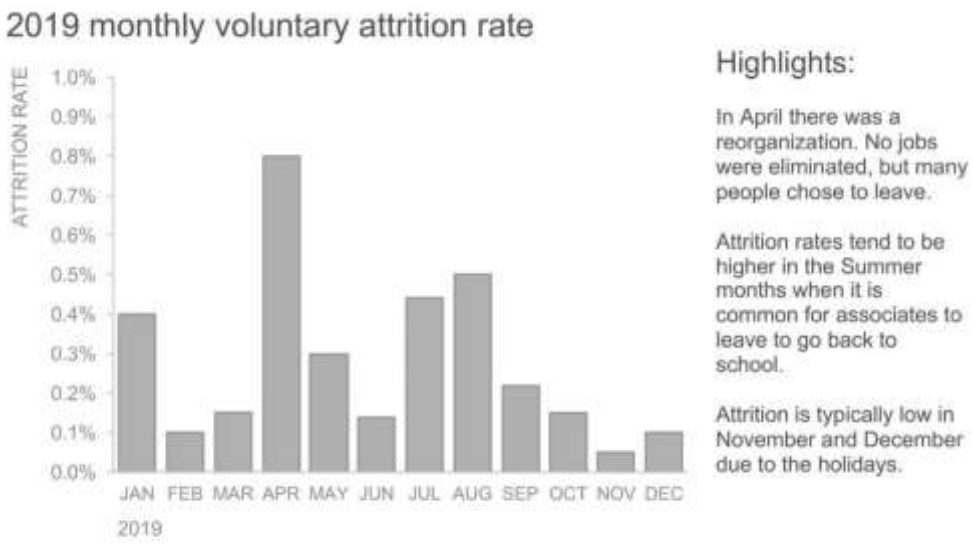


FIGURE 3.2a How can we visually tie the words to the graph?

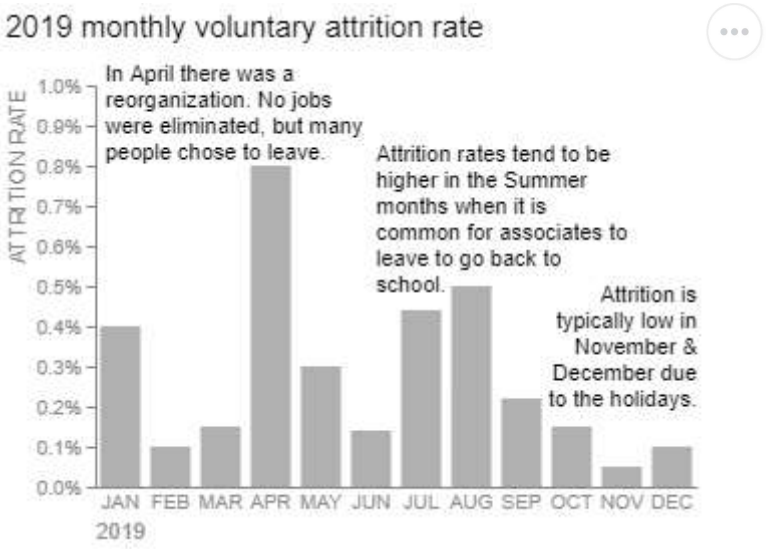
Proximity

The "Proximity Principle" says that we tend to associate objects close to each other as being part of a single group. To apply this is our graph, we bring the texts near the data they represent.

In [68]:
```
# The text now needs to be broken into parts

# First paragraph
text_april = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "In April there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -145, dy = -105)
    .encode(text="text:N")
)

# Second paragraph
text_summer = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition rates tend to be",
                        "higher in the Summer",
                        "months when it is",
                        "common for associates to",
                        "leave to go back to",
                        "school.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -10, dy = -65)
    .encode(text = "text:N")
)

# Third paragraph
text_nov_dec = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition is",
                        "typically low in",
                        "November &",
                        "December due",
                        "to the holidays.",
                    ]
                }
            ]
        }
```

```
        }
    )
    .mark_text(size = 11, align = "right", dx = 150, dy = 5)
    .encode(text = "text:N")
)

# Now we sum the graphs, so that the texts lie on top of the bar, instead of nex
final_prox = bar + text_april + text_summer + text_nov_dec

final_prox.configure_view(stroke = None)
```

Out[68]:



Visualization as depicted in the book:

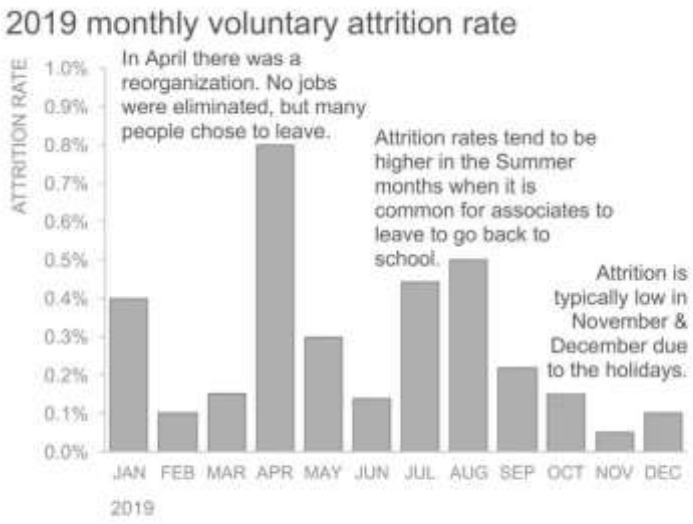


FIGURE 3.2b Proximity

Proximity with emphasis

We can enhance the visual impact by emphasizing the bars and keywords.

Given that Altair does not support bold text within regular content, a strategy is to introduce blank spaces in the text and create a distinct object for the bold keywords.

In [69]:

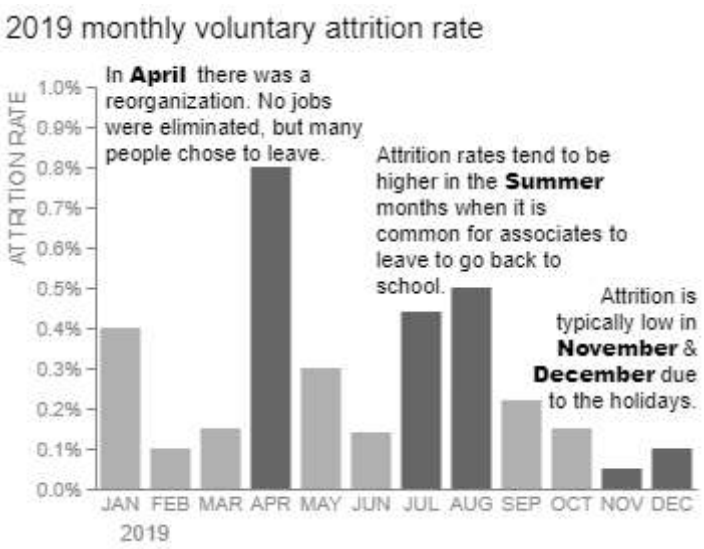
```
bar_highlight = (
    alt.Chart(
        table,
```

```
        title = alt.Title(
            "2019 monthly voluntary attrition rate",
            fontSize = 15,
            anchor = "start",
            offset = 10,
            fontWeight = "normal"
        ),
    )
    .mark_bar(size = 20)
    .encode(
        x = alt.X(
            "Date",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                titleX = 12,
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False,
                titleAnchor = "start",
                titleFontWeight = "normal"
            ),
            title = "2019"
        ),
        y = alt.Y(
            "Rate",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                format = "%",
                tickCount = 10,
            ),
            scale = alt.Scale(domain = [0, 0.01]),
            title = "ATTRITION RATE"
        ),
        color = alt.Color(
            "Date",
            sort = None,
            scale = alt.Scale(
                range = [
                    "#b0b0b0",
                    "#b0b0b0",
                    "#b0b0b0",
                    "#666666", # It was also possible to color by condition
                    "#b0b0b0", # where Date == [list of highlighted months]
                    "#b0b0b0",
                    "#666666",
                    "#666666",
                    "#b0b0b0",
                    "#b0b0b0",
                    "#666666",
                    "#666666"
                ]
            ),
            legend = None
        ),
    )
```

```
    .properties(width = 300, height = 200)
)

# First paragraph with blank space
text_april_blank = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "In            there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -145, dy = -105)
    .encode(text = "text:N")
)

# Second paragraph with blank space
text_summer_blank = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition rates tend to be",
                        "higher in the",
                        "months when it is",
                        "common for associates to",
                        "leave to go back to",
                        "school.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -10, dy = -65)
    .encode(text = "text:N")
)

# Third paragraph with blank space
text_nov_dec_blank = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition is",
                        "typically low in",
                        "&",
                        "due",
                        "to the holidays.",
                    ]
                }
            ]
        }
```

```
        }
    )
    .mark_text(size = 11, align = "right", dx = 150, dy = 5)
    .encode(text = "text:N")
)

# Bold "April" word
text_april_bold = (
    alt.Chart({"values": [{"text": ["April"]}]})
    .mark_text(size = 11, align = "left", dx = -133, dy = -105, fontWeight = 800
    .encode(text = "text:N")
)

# Bold "Summer" word
text_summer_bold = (
    alt.Chart({"values": [{"text": ["Summer"]}]})
    .mark_text(size = 11, align = "left", dx = 54, dy = -52, fontWeight = 800)
    .encode(text = "text:N")
)

# Bold "November" word
text_nov_bold = (
    alt.Chart({"values": [{"text": ["November"]}]})
    .mark_text(size = 11, align = "left", dx = 80, dy = 31, fontWeight = 800)
    .encode(text = "text:N")
)

# Bold "December" word
text_dec_bold = (
    alt.Chart({"values": [{"text": ["December"]}]})
    .mark_text(size = 11, align = "left", dx = 68, dy = 44, fontWeight = 800)
    .encode(text = "text:N")
)

# Adds everything
final_prox_emph = (
    bar_highlight
    + text_april_blank
    + text_april_bold
    + text_summer_blank
    + text_summer_bold
    + text_nov_dec_blank
    + text_nov_bold
    + text_dec_bold
)

final_prox_emph.configure_view(stroke=None)
```

Out[69]:



Visualization as depicted in the book:

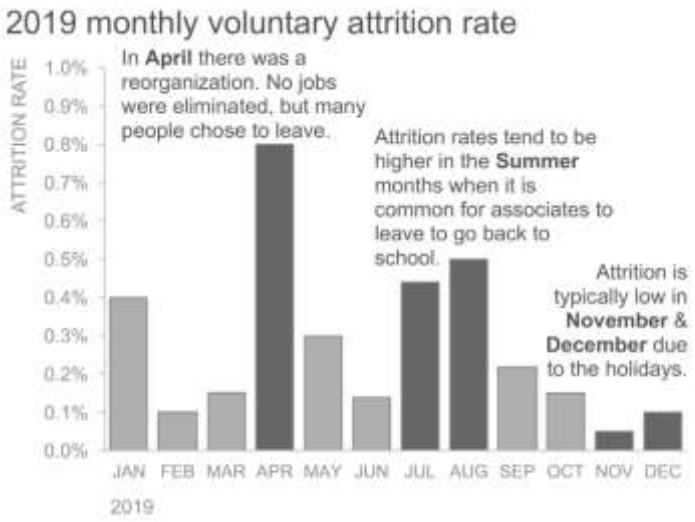


FIGURE 3.2c Proximity with emphasis

Similarity

The "Similarity Principle" pertains to our tendency to perceive objects as part of the same group when they share similar color, shape, or size. For this example, this means coloring the columns in the same shade as the chosen keywords.

In [70]:

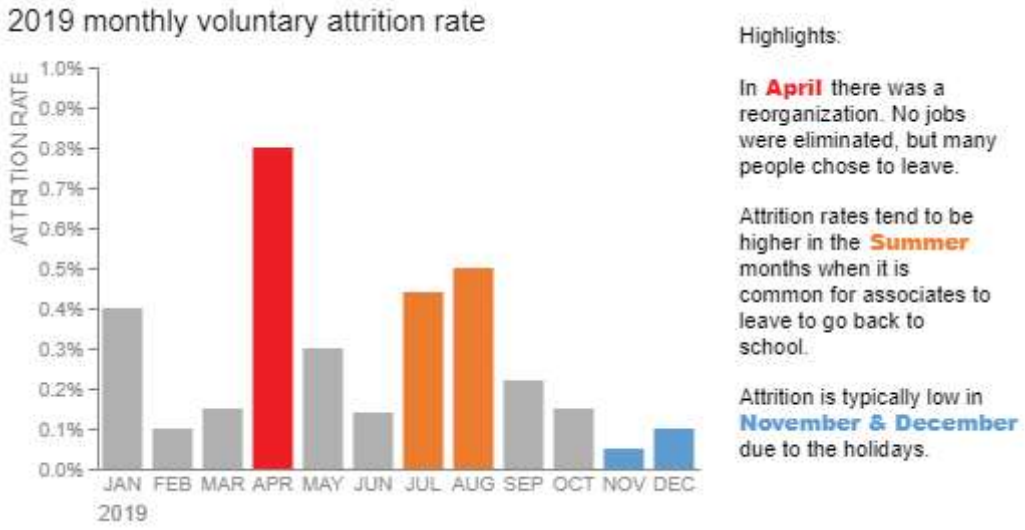
```
bar_highlight_color = (
    alt.Chart(
        table,
        title = alt.Title(
            "2019 monthly voluntary attrition rate",
            fontSize = 15,
            anchor = "start",
            offset = 10,
            fontWeight = "normal"
        ),
    )
    .mark_bar(size = 20)
    .encode(
        x = alt.X(
```

```
            "Date",
            sort = None,
            axis = alt.Axis(
                labelAngle = 0,
                labelColor = "#888888",
                titleColor = "#888888",
                ticks = False,
                titleAnchor = "start",
                titleFontWeight = "normal"
            ),
            title = "2019"
        ),
        y = alt.Y(
            "Rate",
            axis = alt.Axis(
                grid = False,
                titleAnchor = "end",
                labelColor = "#888888",
                titleColor = "#888888",
                titleFontWeight = "normal",
                format = "%",
                tickCount = 10
            ),
            scale = alt.Scale(domain = [0, 0.01]),
            title = "ATTRITION RATE"
        ),
        color = alt.Color(
            "Date",
            sort = None,
            scale = alt.Scale(
                range = [
                    "#b0b0b0",
                    "#b0b0b0", # Since there are multiple colors
                    "#b0b0b0", # set by condition would be harder
                    "#ed1e24",
                    "#b0b0b0",
                    "#b0b0b0",
                    "#ec7c30",
                    "#ec7c30",
                    "#b0b0b0",
                    "#b0b0b0",
                    "#5d9bd1",
                    "#5d9bd1",
                ]
            ),
            legend = None
        )
    )
    .properties(width = 300, height = 200)
)

# Blank texts with different position
text_april_blank2 = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Highlights:",
                        " ",
```

```
                        "In            there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dy = -25, dx = -10)
    .encode(text = "text:N")
)

text_summer_blank2 = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition rates tend to be",
                        "higher in the",
                        "months when it is",
                        "common for associates to",
                        "leave to go back to",
                        "school.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -10, dy = 65)
    .encode(text = "text:N")
)

text_nov_dec_blank2 = (
    alt.Chart(
        {
            "values": [
                {"text": ["Attrition is typically low in", " ", "due to the holi
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = -10, dy = 155)
    .encode(text = "text:N")
)

# Colored texts
text_april_color = (
    alt.Chart({"values": [{"text": ["April"]}]})
    .mark_text(size = 11, align = "left", dx = 3, dy = 1, fontWeight = 800, colo
    .encode(text="text:N")
)

text_summer_color = (
    alt.Chart({"values": [{"text": ["Summer"]}]})
    .mark_text(size = 11, align = "left", dx = 55, dy = 78, fontWeight = 800, co
    .encode(text="text:N")
)

text_nov_dec_color = (
```

```
    alt.Chart({"values": [{"text": ["November & December"]}]})
    .mark_text(size = 11, align = "left", dx = -10, dy = 168, fontWeight = 800,
    .encode(text = "text:N")
)

# While we could have used '&' to arrange the texts vertically,
# employing '+' provides greater flexibility in determining the layout of the te

final_sim = bar_highlight_color | (
    text_april_blank2
    + text_april_color
    + text_summer_blank2
    + text_summer_color
    + text_nov_dec_blank2
    + text_nov_dec_color
)

final_sim.configure_view(stroke=None)
```

Out[70]:



Visualization as depicted in the book:

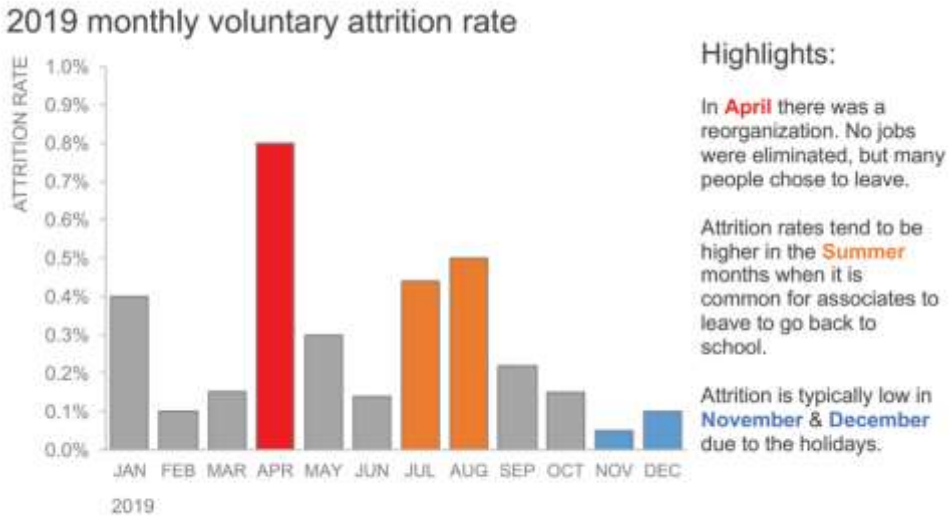


FIGURE 3.2d Similarity

Enclosure

The "Enclosure Principle" says simply that, when objects are enclosed together, we perceive them as belonging to the same group.

Attempting to combine charts using the expression `(bar | text) + rect_nov_dec + rect_summer + rect_april` results in an error:

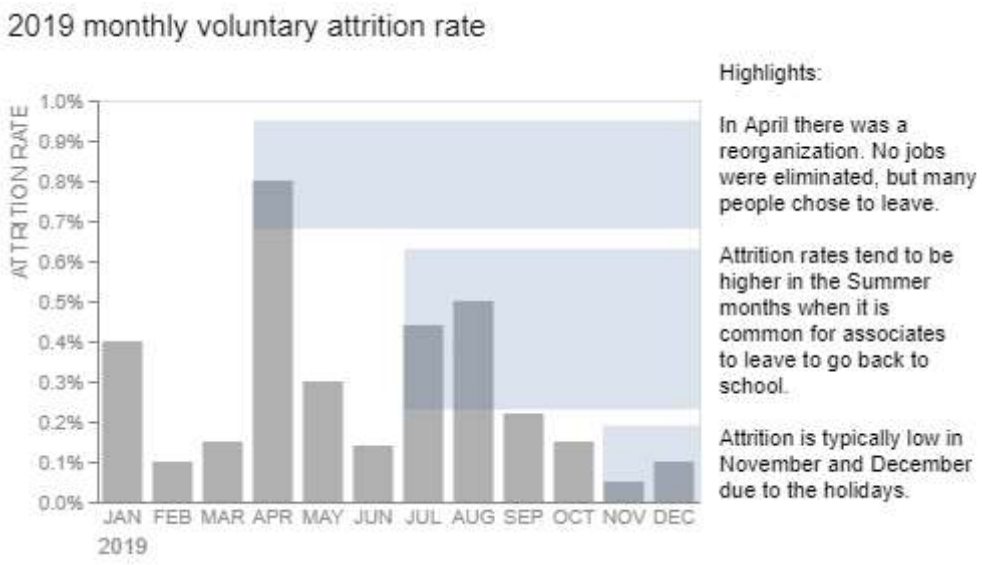
```
Concatenated charts cannot be layered. Instead, layer the charts before concatenating.
```

The most straightforward way to solve this is to add the text to the bar using `bar + text`, but doing so means assigning another position (dx, dy) to the text.

In [71]:
```
# Assign another position to text
text_enclosure = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Highlights:",
                        " ",
                        "In April there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                        " ",
                        "Attrition rates tend to be",
                        "higher in the Summer",
                        "months when it is",
                        "common for associates",
                        "to leave to go back to",
                        "school.",
                        " ",
                        "Attrition is typically low in",
                        "November and December",
                        "due to the holidays.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = 160, dy = -113)
    .encode(text = "text:N")
)

# Defines the rectangles that are going to enclose the text
rect_nov_dec = (
    alt.Chart(pd.DataFrame({"y": [0], "y2": [0.0019], "x": [10], "x2": [8.4]}))
    .mark_rect(opacity = 0.2) # Low opacity
    .encode(y = "y", y2 = "y2", x = alt.X("x", axis = None), x2 = "x2")
)

rect_summer = (
    alt.Chart(pd.DataFrame({"y": [0.0023], "y2": [0.0063], "x": [10], "x2": [5.1
    .mark_rect(opacity = 0.2)
    .encode(y = "y", y2 = "y2", x = alt.X("x", axis = None), x2 = "x2")
)

rect_april = (
```

```
    alt.Chart(pd.DataFrame({"y": [0.0068], "y2": [0.0095], "x": [10], "x2": [2.6
    .mark_rect(opacity = 0.2)
    .encode(y = "y", y2 = "y2", x = alt.X("x", axis = None), x2 = "x2")
)

bar + text_enclosure + rect_nov_dec + rect_summer + rect_april
```

Out[71]:

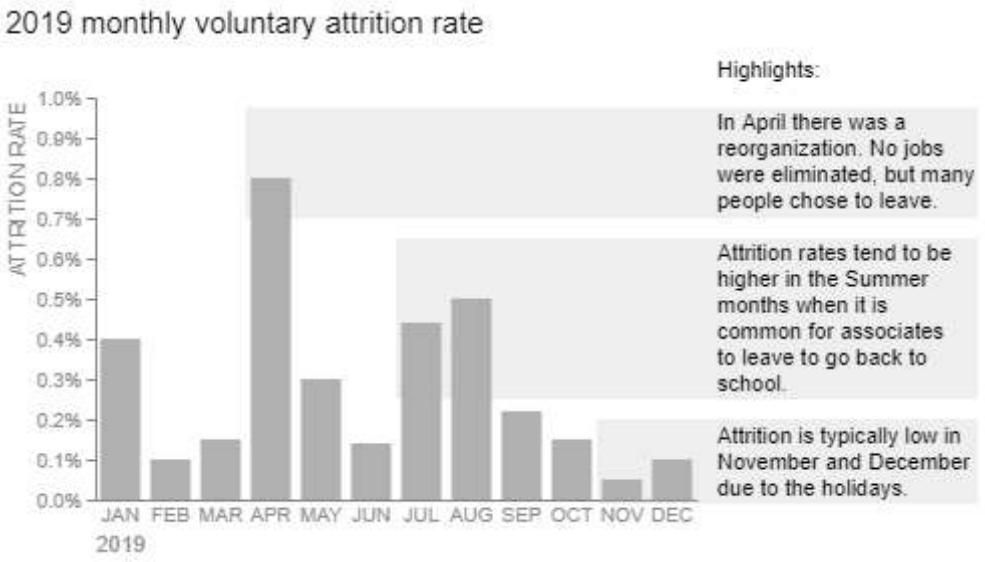


Utilizing a DataFrame to define the rectangles seems to prevent them from reaching the text section. As a next step, we will explicitly define the coordinates of the rectangles in pixels.

In [72]:

```
# Rectangles with position defined by pixels
rect_nov_dec = alt.Chart(pd.DataFrame({'values':[{}]})).mark_rect(opacity = 0.2,
   y = alt.value(5), y2 = alt.value(60), x = alt.value(75), x2 = alt.value(440)
)

rect_summer = alt.Chart(pd.DataFrame({'values':[{}]})).mark_rect(opacity = 0.2,
   y = alt.value(70), y2 = alt.value(150), x = alt.value(150), x2 = alt.value(44
)

rect_april = alt.Chart(pd.DataFrame({'values':[{}]})).mark_rect(opacity = 0.2, c
   y = alt.value(160), y2 = alt.value(202), x = alt.value(250), x2 = alt.value(4
)

# The text_enclosure and bar comes after the rectangles so that they sit on top
final_enc = rect_nov_dec + rect_summer + rect_april + bar + text_enclosure

final_enc.configure_view(stroke = None)
```

Out[72]:



Visualization as depicted in the book:

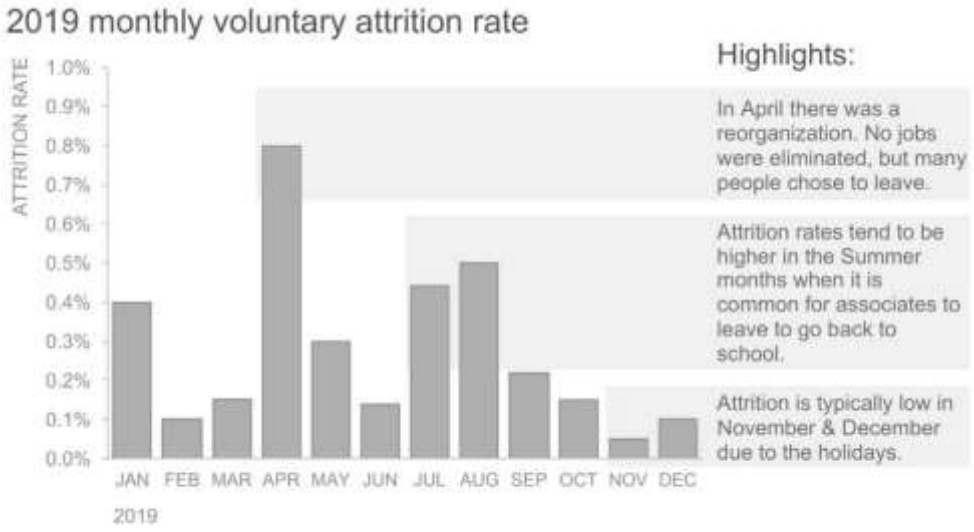


FIGURE 3.2e Enclosure

Enclosure with color differentiation

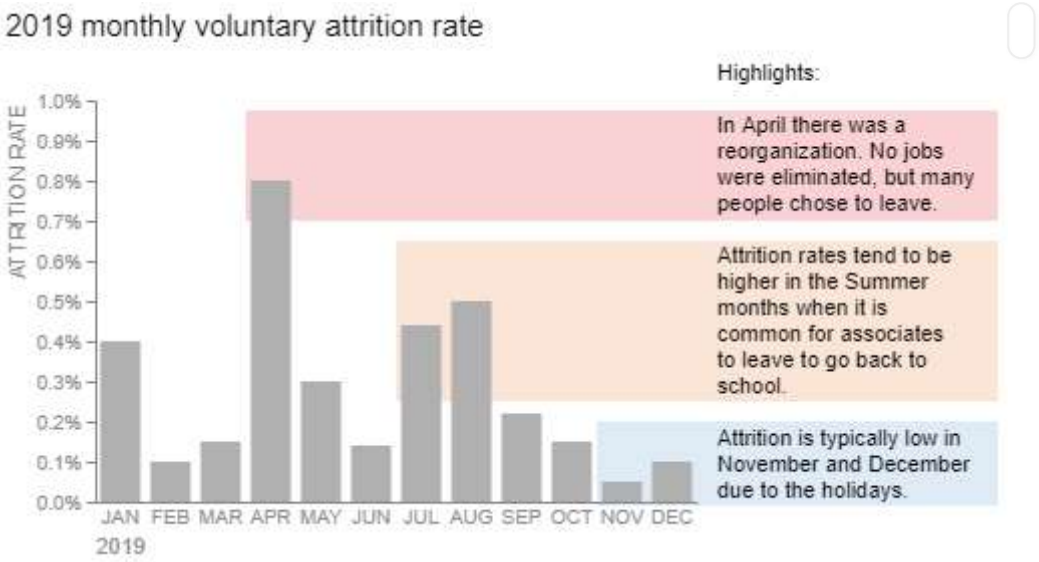
We can use color to emphasize the different enclosures.

In [73]:

```
# Colored rectangles
rect_nov_dec_color = (
    alt.Chart(pd.DataFrame({"values": [{}]}))
    .mark_rect(opacity = 0.2, color = "#ed1e24")
    .encode(y = alt.value(5), y2 = alt.value(60), x = alt.value(75), x2 = alt.va
)

rect_summer_color = (
    alt.Chart(pd.DataFrame({"values": [{}]}))
    .mark_rect(opacity = 0.2, color = "#ec7c30")
    .encode(y = alt.value(70), y2 = alt.value(150), x = alt.value(150), x2 = alt
)

rect_april_color = (
    alt.Chart(pd.DataFrame({"values": [{}]}))
    .mark_rect(opacity = 0.2, color = "#5d9bd1")
```

```
    .encode(y = alt.value(160), y2 = alt.value(202), x = alt.value(250), x2 = al
)

final_enc_color = rect_nov_dec_color + rect_summer_color + rect_april_color + ba

final_enc_color.configure_view(stroke = None)
```

Out[73]:



Visualization as depicted in the book:

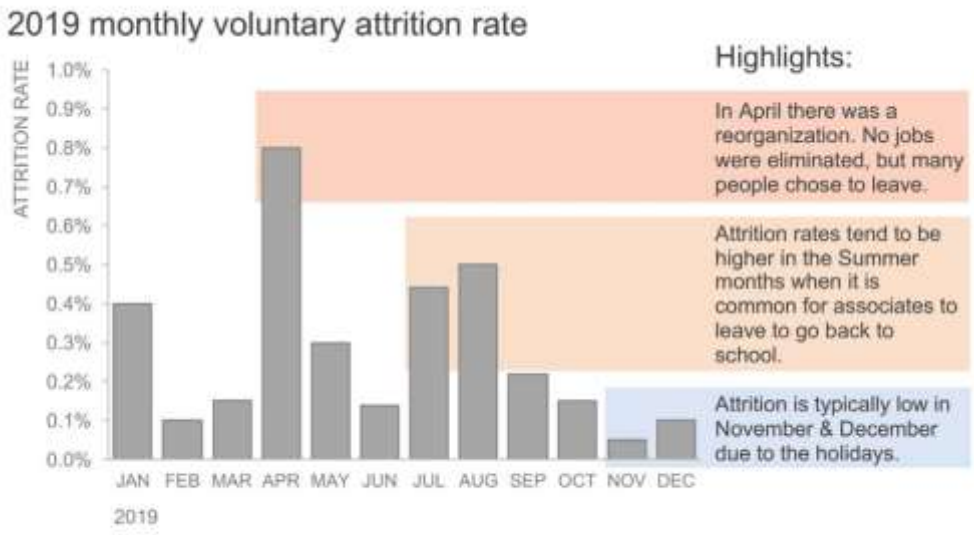


FIGURE 3.2f Enclosure with color differentiation

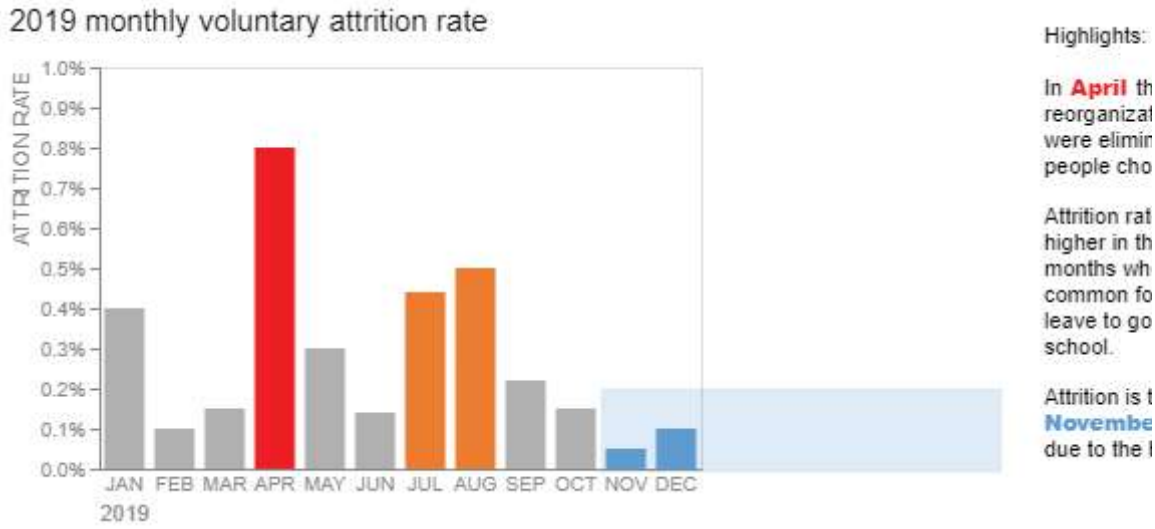
Enclosure + Similarity

We can make use of both Enclosure and Similarity principles. First, we will try to add already existing components.

In [74]:

```
bar_highlight_color + rect_april_color | (text_april_blank2 + text_april_color
                                + text_summer_blank2 + text_summer_color
                                + text_nov_dec_blank2 + text_nov_dec_color)
```

Out[74]:



In [75]:

```
bar_highlight_color | (text_april_blank2 + text_april_color
                      + text_summer_blank2 + text_summer_color
                      + text_nov_dec_blank2 + text_nov_dec_color) + rect_april_
```

Out[75]:

Since attempting to layer only already assigned variables seems to be ineffective, we will recreate the texts using alternative positions to enable their addition using the + symbol.

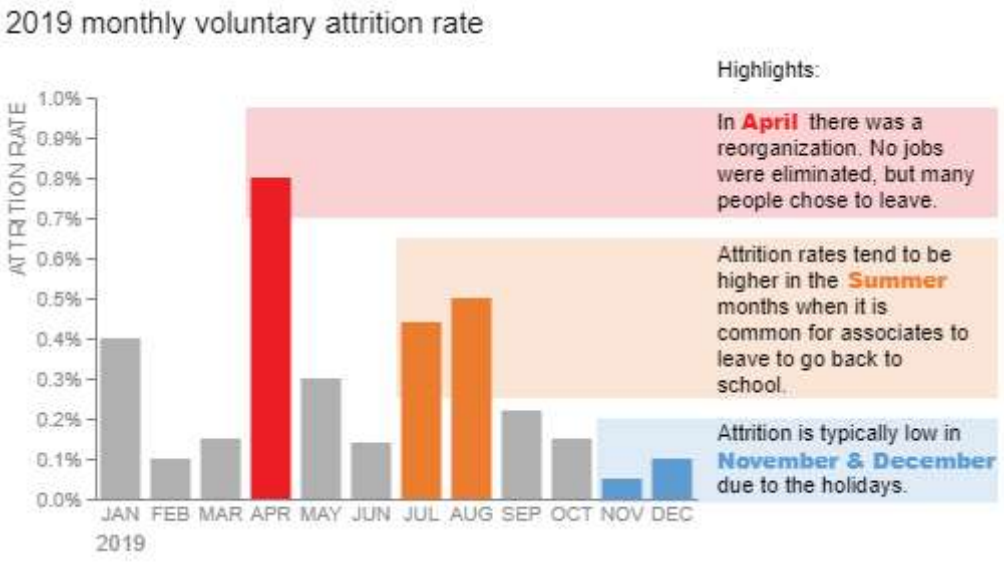
In [76]:

```
# Same texts, different dx and dy values

text_april_blank2_enclosure = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Highlights:",
                        " ",
                        "In            there was a",
                        "reorganization. No jobs",
                        "were eliminated, but many",
                        "people chose to leave.",
                    ]
                }
```

```
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = 160, dy = -113)
    .encode(text = "text:N")
)

text_summer_blank2_enclosure = (
    alt.Chart(
        {
            "values": [
                {
                    "text": [
                        "Attrition rates tend to be",
                        "higher in the",
                        "months when it is",
                        "common for associates to",
                        "leave to go back to",
                        "school.",
                    ]
                }
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = 160, dy = -21)
    .encode(text = "text:N")
)

text_nov_dec_blank2_enclosure = (
    alt.Chart(
        {
            "values": [
                {"text": ["Attrition is typically low in", " ", "due to the holi
            ]
        }
    )
    .mark_text(size = 11, align = "left", dx = 160, dy = 68)
    .encode(text = "text:N")
)

text_april_color_enclosure = (
    alt.Chart({"values": [{"text": ["April"]}]})
    .mark_text(size = 11, align = "left", dx = 172, dy = -87, fontWeight = 800,
    .encode(text = "text:N")
)

text_summer_color_enclosure = (
    alt.Chart({"values": [{"text": ["Summer"]}]})
    .mark_text(size = 11, align = "left", dx = 225, dy = -8, fontWeight = 800, c
    .encode(text = "text:N")
)

text_nov_dec_color_enclosure = (
    alt.Chart({"values": [{"text": ["November & December"]}]})
    .mark_text(size = 11, align = "left", dx = 160, dy = 82, fontWeight = 800, c
    .encode(text = "text:N")
)

final_enc_sim = (
    rect_nov_dec_color
```

```
    + rect_summer_color
    + rect_april_color
    + bar_highlight_color
    + text_april_blank2_enclosure
    + text_summer_blank2_enclosure
    + text_nov_dec_blank2_enclosure
    + text_april_color_enclosure
    + text_summer_color_enclosure
    + text_nov_dec_color_enclosure
)

final_enc_sim.configure_view(stroke=None)
```

Out[76]:



Visualization as depicted in the book:

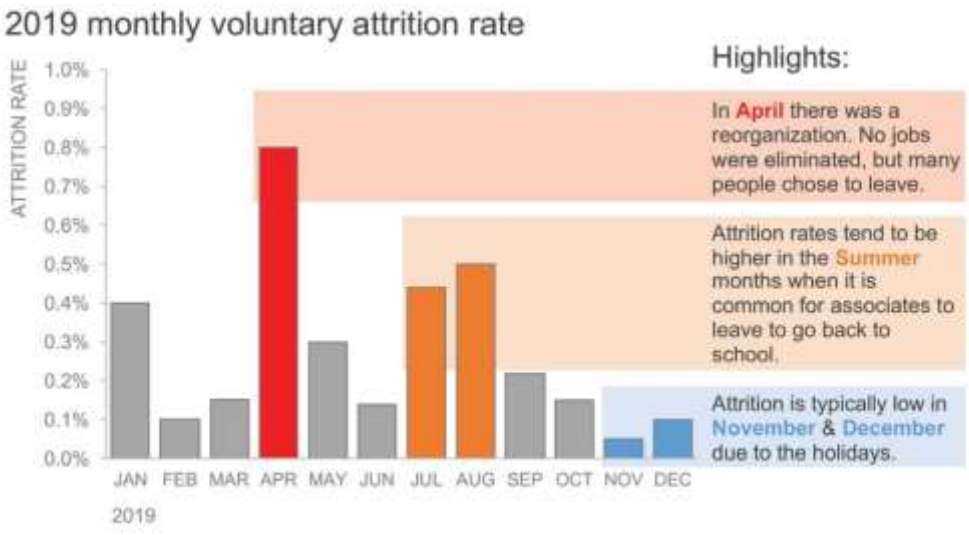


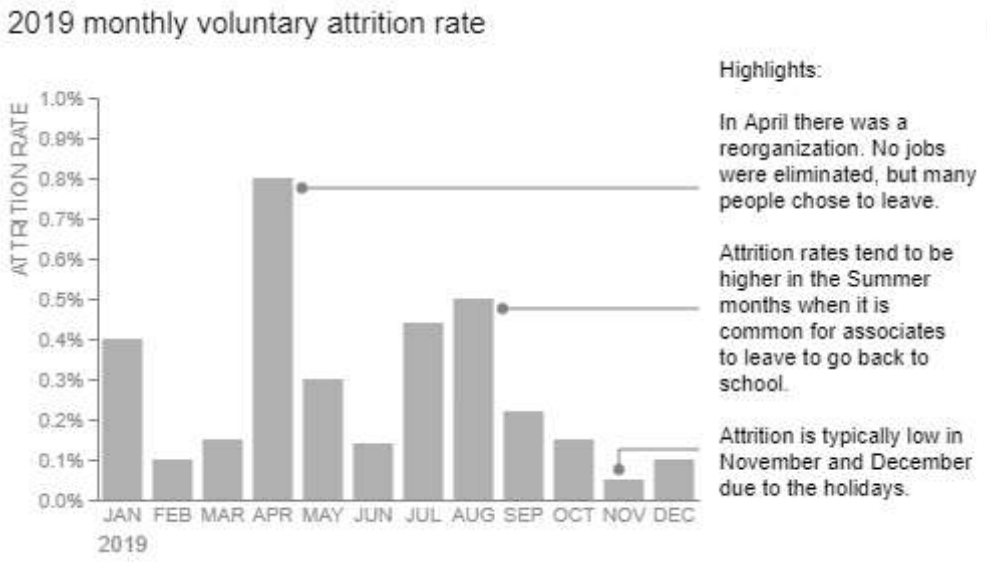
FIGURE 3.2g Enclosure plus similarity

Connection

The "Connection" relies on the fact that objects that are physically connected are often perceived as part of a single group. In this example, we will connect the texts and the data using a line.

In [77]:
```
# Rules connecting text with bar
rule_april = (
    alt.Chart()
    .mark_rule(point={"fill": "gray"}) # With a dot at the end
    .encode(
        x = alt.value(102),
        y = alt.value(45),
        x2 = alt.value(300),
        strokeWidth = alt.value(0.5)
    )
)

rule_summer = (
    alt.Chart()
    .mark_rule(point = {"fill": "gray"})
    .encode(
        x = alt.value(202),
        y = alt.value(105),
        x2 = alt.value(300),
        strokeWidth = alt.value(0.5)
    )
)

# Part one of the November/December rule
rule_nov_dec_1 = (
    alt.Chart()
    .mark_rule() # Without the dot
    .encode(
        x = alt.value(260),
        y = alt.value(175),
        x2 = alt.value(300),
        strokeWidth = alt.value(0.5)
    )
)

# Part two of the November/December rule
rule_nov_dec_2 = (
    alt.Chart()
    .mark_rule(point = {"fill": "gray"}) # With the dot
    .encode(
        x = alt.value(260),
        y = alt.value(185),
        y2 = alt.value(175),
        strokeWidth = alt.value(0.5)
    )
)

final_conn = (
    bar + text_enclosure + rule_april + rule_summer + rule_nov_dec_1 + rule_nov_
)

final_conn.configure_view(stroke = None)
```

Out[77]:



Visualization as depicted in the book:

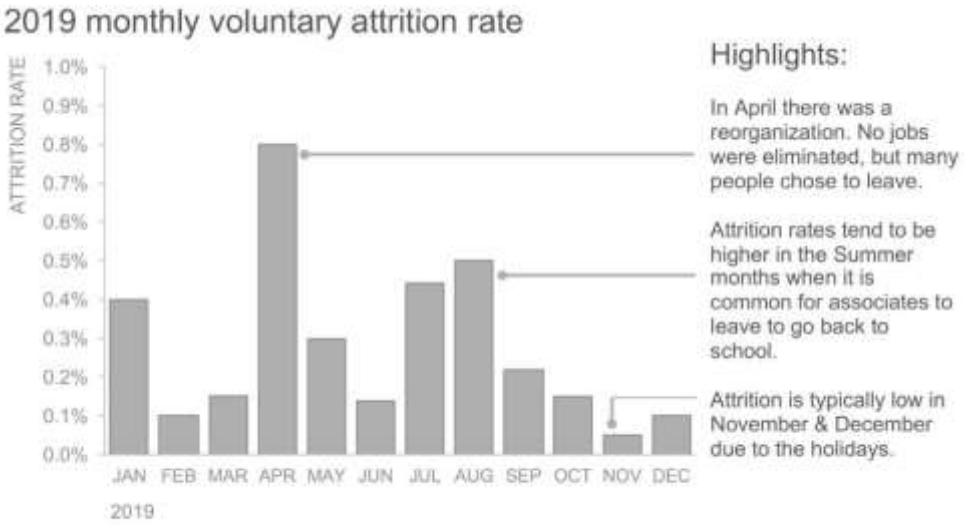


FIGURE 3.2h Connection

Connection + Similarity

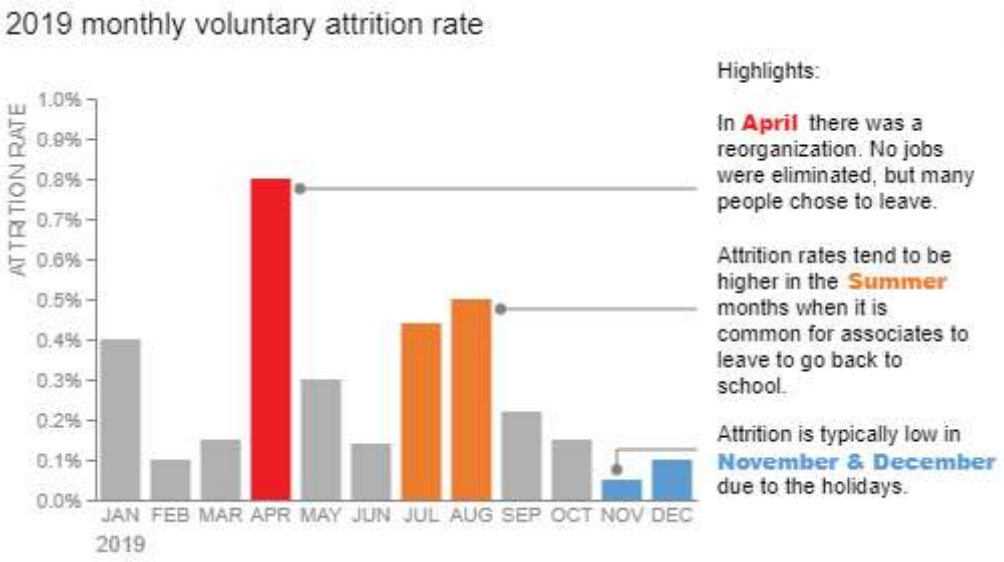
Now we can use the connection and the similarity principles to connect highlighted texts with colored data.

In [78]:

```
final_conn_sim = (
    bar_highlight_color
    + text_april_blank2_enclosure
    + text_summer_blank2_enclosure
    + text_nov_dec_blank2_enclosure
    + text_april_color_enclosure
    + text_summer_color_enclosure
    + text_nov_dec_color_enclosure
    + rule_april
    + rule_summer
    + rule_nov_dec_1
    + rule_nov_dec_2
)
```

```
final_conn_sim.configure_view(stroke=None)
```

Out[78]:



Visualization as depicted in the book:

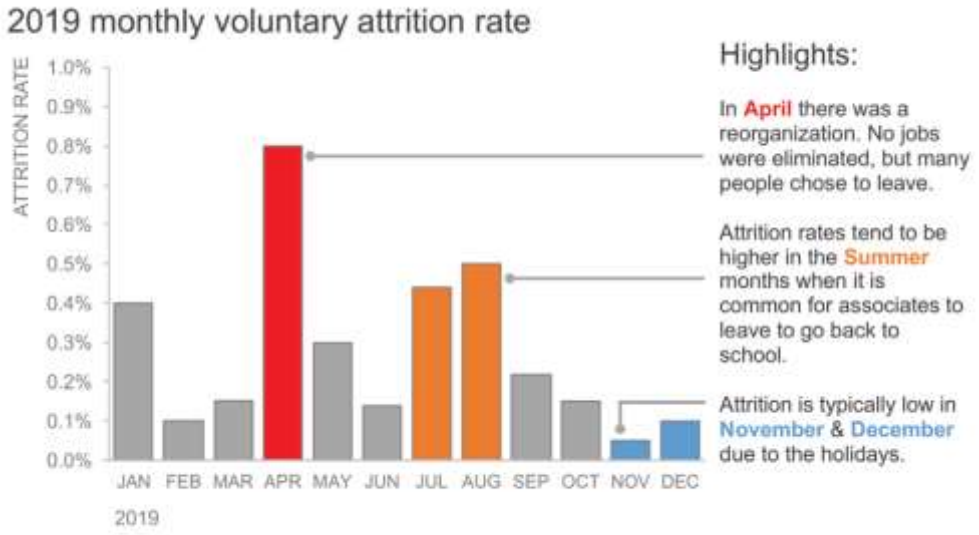


FIGURE 3.2i Connection plus similarity

Interactive

Cell to create the .html

This code was created by Søren Fuglede Jørgensen and can be found here. Please make sure any changes are saved before running this cell.

In [79]:

```
with open('index.ipynb') as nb_file:
    nb_contents = nb_file.read()

# Convert using the ordinary exporter
notebook = nbformat.reads(nb_contents, as_version=4)

# HTML Export
html_exporter = nbconvert.HTMLExporter()
body, res = html_exporter.from_notebook_node(notebook)
```

```
# Create a dict mapping all image attachments to their base64 representations
images = {}
for cell in notebook['cells']:
    if 'attachments' in cell:
        attachments = cell['attachments']
        for filename, attachment in attachments.items():
            for mime, base64 in attachment.items():
                images[f'attachment:{filename}'] = f'data:{mime};base64,{base64}

# Fix up the HTML and write it to disk
for src, base64 in images.items():
    body = body.replace(f'src="{src}"', f'src="{base64}"')

# Write HTML to file
with open('index.html', 'w') as html_output_file:
    html_output_file.write(body)
```