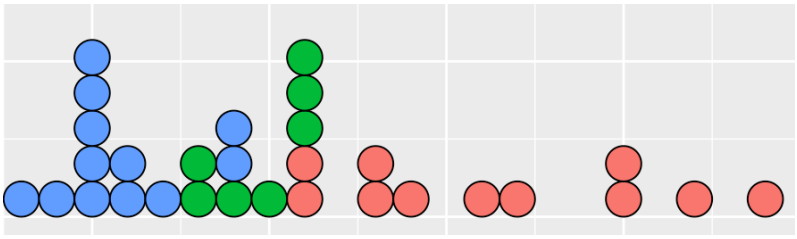


Measuring Social Impact Primer

Andre Luis Rossi de Oliveira
Gregory Brooks
Jacob Sorenson
Samuel Elzinga

SIMLAB

Center for Social Impact

Table of Contents

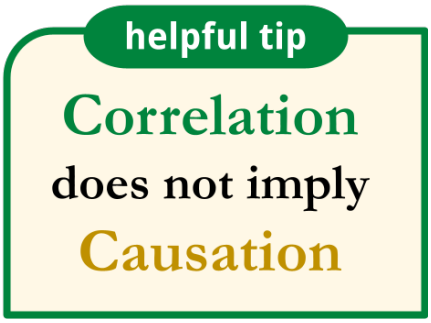
❖ Measuring social impact

In this primer, we discuss how to measure social impact. But before we can delve into the appropriate models and techniques for that, we need to answer two fundamental questions:

(a) What kind of measurement are we talking about?

(b) What do we mean by social impact?

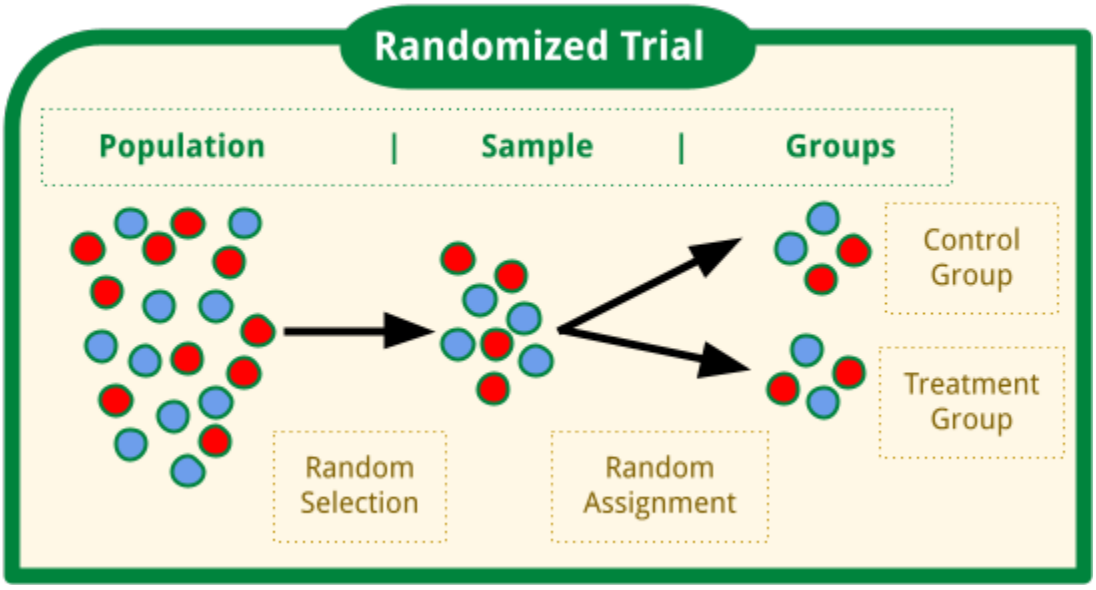
Let's start with question **(b)**. Social impact exists or happens when some type of intervention leads to changes in one or more features of society, including cultural practices, economic phenomena, and political processes. For example, an organization will have social impact when it provides a product or service to members of the community that gives them access to better jobs, helps them improve their education, or increases their participation in the community. Notice that social impact is associated with the idea of causality: we must be able to show that the intervention caused the outcome; it is not enough to find correlation between the intervention and the outcome.



This brings us to question **(a)**. In order to measure causal relationships, we need to employ statistical techniques that rule out, to the maximum extent possible, other possible explanations for the outcomes we observe. That is not to say that qualitative measures are not important or helpful, but only that by themselves they cannot establish causality. In this primer, we focus on quantitative methods to measure social impact.

The gold standard to establish causal relationships is a randomized trial or experiment. First, a sample that is representative of the population of interest is obtained (random selection). Then, sample units are randomly assigned to either a treatment group or a control group (randomized assignment). Members of the treatment group are exposed to the intervention, while those in the control group are

not. In general, when the sample size is large enough there are no systematic differences in observed and unobserved characteristics between the treatment and control groups at baseline, on average. In other words, the two groups are similar in every way.



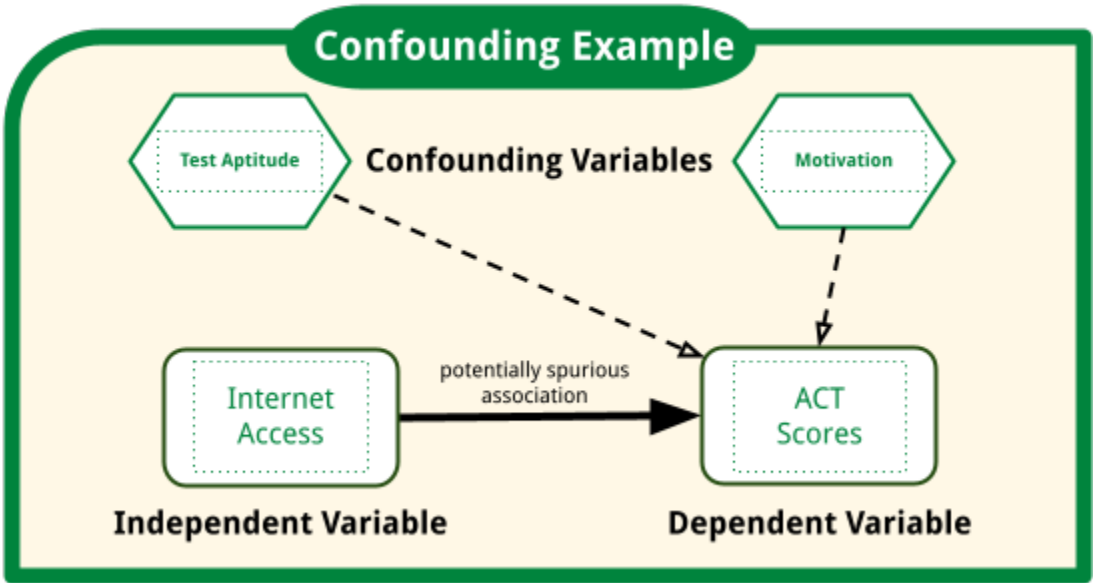
To estimate the impact of an intervention based on data from a randomized trial, we can simply compute the difference between the average values of the outcome variable for the treatment and control groups. If that difference is statistically significant, we have established a causal link between the intervention and the outcome, since randomized assignment eliminates other observed and unobserved factors that might plausibly explain it.

In contrast, in observational studies treatment assignment is not under experimental control, and differences in outcomes between the treatment and control groups may be due to reasons other than the effect of the treatment. In this primer, we discuss conditions under which causality can be established in observational studies and the limitations imposed by those conditions. We also introduce statistical methods that can be used to estimate social impact when only observational data is available, provide R code to run those methods, and illustrate their use with applications.

❖ Mimicking randomized trials

When the observations in our data set are not subject to manipulation or intervention by the researcher, the treated and the non-treated cases may not be comparable at baseline. The treated cases form the treatment group, and the non-treated are part of what we will call the comparison group (analogous to the control group in randomized trials) from now on.

Even though differences between members of the two groups that are accurately measured before treatment can often be removed, there is always the possibility that significant differences were not accounted for, invalidating any causality analysis. For instance, let's say we calculated ACT scores for students with and without access to broadband at home and found a positive and statistically significant difference. We also studied the compositions of the two groups in terms of age, gender, ethnicity, GPA and several other variables we were able to observe. Even if the two groups were similarly distributed with respect to those observed measures, we still would not be able to say that access to broadband at home had a causal effect on ACT scores, for there might be unobserved measures that can account for that, including motivation, aptitude for taking tests and other idiosyncratic factors.



Therefore, the best we can do is identify and include in our model as many measurable confounding features as possible to eliminate their influence on the outcome. The main methods to do that are stratification, weighing and matching. In this primer, we focus on matching by *propensity score*, a technique that uses a single indicator instead of all the covariates (features) separately to match treated and non-treated units. After the measured sources of bias are removed, we can run a *sensitivity analysis* to gauge how much residual bias would need to be present to invalidate the causal relationship, if one is detected.

❖ Matching and treatment effects

The main goal of a matching procedure is to reduce bias in the estimated treatment effect, which requires the treatment and comparison groups to be as similar as possible. This is achieved by matching each treated unit to one or more distinct comparison units based on some distance metric.

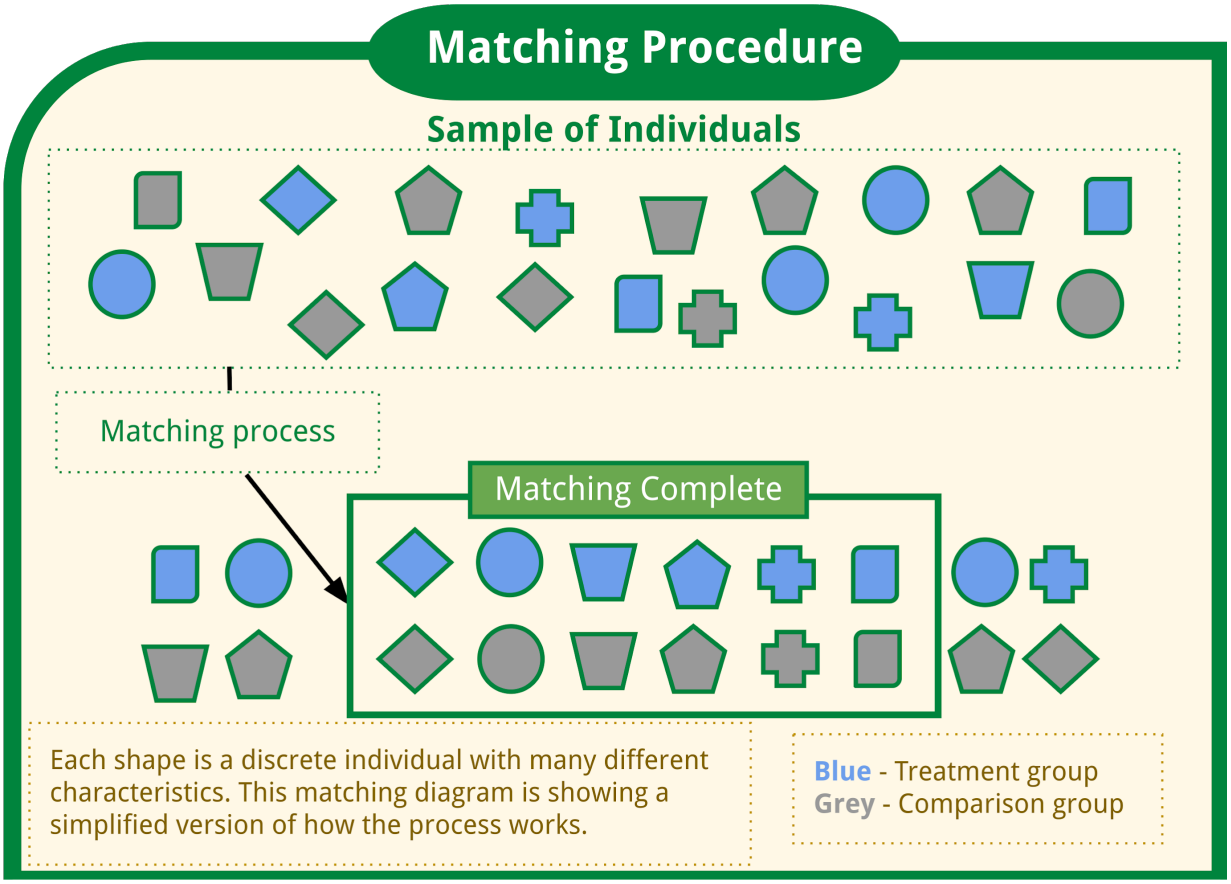
Matching methods have four key steps (Stuart, 2010):

1. Select a distance measure to determine whether a case is a good match for another;
2. Select and implement a matching method using that distance measure;
3. Use a series of indicators to assess the quality of the match. If needed, repeat steps 1 and 2 until a suitable match is obtained.
4. Estimate the treatment effect and analyze the outcome.

Step 1 actually has two parts. First, we must select the covariates to be used in the matching process. Recall that the problem we are trying to solve in observational studies is the presence of confounders, that is variables other than the treatment that might affect the outcome. Therefore, we want to include as many covariates in our analysis that are related to both treatment assignment and outcomes as possible, thereby

reducing the likelihood of leaving out confounders. On the other hand, variables that may have been affected by the treatment should not be included. The second part of **Step 1** is the selection of a distance measure that takes the covariates as arguments. There are several possibilities, but we will work only with the absolute value of the propensity score, to be discussed in more detail below.

In **Step 2**, we choose the matching method to be employed. There is a very large number of methods available, including greedy, optimal, full and genetic matching. A full description of these methods is outside the scope of this primer, but we will provide more detail about those we use in our application.



After we have a match, it is time to assess how similar the treatment and comparison groups are. This is done in **Step 3**, when different indicators are used to measure *covariate balance*, that is the similarity of the distributions of the covariates in the

matched treatment and comparison groups. It is also a good idea to compare those distributions before matching and check how much improvement matching has achieved. Graphical analysis is also helpful. If the desired balance is not achieved with one matching procedure, others should be tried.

Once we are satisfied with the matched data set, we can proceed to **Step 4** and estimate the effect of treatment on the outcome. This is typically done by running regressions on the matched samples, but other statistical techniques can be employed. The two main estimands of interest are the **Average Treatment Effect (ATE)** and the **Average Treatment Effect on the Treated (ATT)**.

The causal effect of the treatment on an individual is defined as the difference between her response (the value of the outcome variable for that individual) when she is treated and her response when she is not treated. Of course, in the real world we can only observe one of these responses, depending on whether the individual was treated or not, so the causal effect is not observable. We can, however, posit that these two responses are random variables with certain distributions, calculate their expected values, and define the **ATE** as the difference between these expected values.

Similarly, if we are only interested in the impact of the treatment on the treated, we focus on the difference between the conditional expected values of the responses (conditioned on having received treatment), known as the **ATT**.

It is now just a matter of finding a way to estimate the **ATE** and the **ATT**, since we typically only have data on a sample of the population. We will have more to say about this in the application section.

Once we have estimated the **ATE** and/or the **ATT** (and possibly other measures), it is time to analyze the results. This usually involves determining if the effect of the treatment is statistically significant, the magnitude of the effect, and discussing the implications of the results for business decisions and policy making.

❖ Application: Advising and student success

In the previous sections, we painted the process of measuring social impact with a broad brush. The idea was to not overwhelm the reader with technical details. Instead, we will go over any indispensable methodological details in this section as we put into practice the techniques we presented above. We will use a statistical program called R to perform all the graphical and statistical analysis.

We start with a brief discussion of a "results chain" (also called "theory of change") for the problem we tackle in this application, namely the impact of academic advising on student performance. A results chain "is a description of how an intervention is supposed to deliver the desired results. It describes the causal logic of how and why a particular program, program modality, or design innovation will reach its intended outcomes" (Gertler et al, 2016).

That discussion is followed by a brief description of the data set and a set of graphical resources to visualize the data. The visualization stage is very important, for it helps us detect data issues like missing values and outliers and identify possible relationships between the variables in our model. The 'ggplot' function from R will be instrumental in this regard. The appendix contains a description of some of the most important 'ggplot' features.

T tests, median tests and other types of tests can be used to perform a preliminary investigation of the differences between the treatment and comparison groups and how they relate to the outcome. More elaborate techniques as multiple regression can also be used. However, any results obtained at this stage may be spurious, since we have not yet dealt with the biases that can invalidate causal statements.

In the last part of the application, we follow the four steps of the matching process presented above with the ultimate goal of, having obtained a satisfactory match, estimate the treatment effect.

➢ Results Chain

The basic elements of a results chain are the following (Gertler et al, 2016):

- Inputs: Resources at the disposal of the project.
- Activities: Actions that convert inputs into outputs.
- Outputs: The tangible goods and services that project activities produce.
- Outcomes: Short to medium term results likely to be achieved once the beneficiary population uses project outputs.
- Final outcomes: Long term or final results achieved.

The first three elements are under the control of the project and concern its implementation. The last two elements are the results of the project and are not under its direct control. They depend on the behavior of the project's beneficiaries and external conditions.

Academic advising uses a variety of inputs to foster student success, including staff, facilities, databases, software, and other resources. Through activities like in-person advising sessions, contacting students through e-mail and other media, informing students of the resources available to them, and career development training, academic advising transforms inputs into outputs.

Activities associated with academic advising generate a variety of outputs, including course schedules, information about classes, graduation plans and referrals. These are valuable assets that help students navigate their programs.

Outcomes are short to medium term results students are likely to achieve on account of the outputs. They include better grades and retention rates, shorter graduation time, and improved mental health. When these outcomes are attained over a long period of time, they are placed into the category of final outcomes. Examples are better jobs or careers, job promotions, involvement in the community, upgraded housing, and improved overall quality of life.

The discussion of academic advising in this primer focuses on one outcome and one activity, namely students' average grade and visits to advisors in a semester, respectively. Nevertheless, the results chain developed above can be used to perform a much wider analysis of academic advising.

➢ The Data Set

The data set used in this application contains information on several student characteristics, including gender, age and ethnicity, academic measures such as courses taken, GPA, course grade and major, and features of student visits to advisors, including dates and times, reason for visit and advising center. The data covers the 2018-2019 school year. There are many entries per student. For instance, a student may have taken many courses and paid many visits to an advisor in a given semester. Each course/visit combination corresponds to a separate entry. The screenshot below shows a snippet of the entries for a single student (id # 21639)[1], where the variable 'advtimei' is the start time of the visit.

	id	advtimei	Course	Age	Gender
1	21639	2019-08-15 11:46:00	CJ 4990 X01	23	M
2	21639	2019-08-08 12:49:00	CJ 4990 X01	23	M
3	21639	2019-08-08 10:28:00	CJ 4990 X01	23	M
4	21639	2019-08-15 11:46:00	CJ 4990 X01	23	M
5	21639	2019-08-08 12:49:00	CJ 4990 X01	23	M
6	21639	2019-08-08 10:28:00	CJ 4990 X01	23	M
7	21639	2018-10-29 09:06:00	CJ 4200 601	22	M
8	21639	2018-10-29 09:06:00	CJ 4250 X01	22	M
9	21639	2018-10-29 09:06:00	CJ 4200 601	22	M

[1] The data has been anonymized.

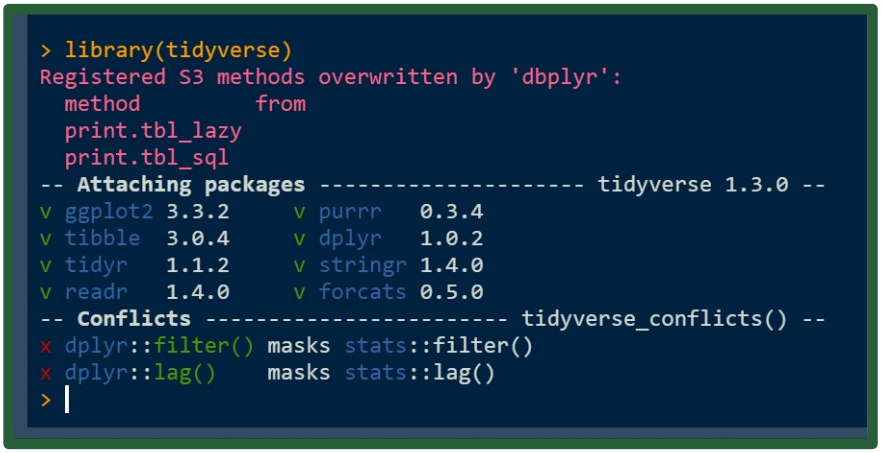
There are many ways to load a data set into R. If the data is already in R format (.Rdata), we use the `load` function as follows[2]:

```
load("C:/path/data.2018_19.RData")
```

In the example above, the name of the data set is "data.2018_19.Rdata". The **path** will be different, depending on where you stored the file.

The next step is to add the package "tidyverse", which includes "ggplot" as well as other tools for data analysis and processing:

```
library(tidyverse)
```



Once you run this snippet, it may take a minute to install all the packages, so wait until the console stops spitting out information.

The Tidyverse also allows us to import data from a **.csv** file and other formats, with the **readr** package. For example, we could run `read_csv("yourfilehere.csv")`. to load the file "yourfilehere.csv". This is useful because we don't always have our data in .RData format. Another useful package is **readxl**, which can be used to import and load Excel spreadsheets (remember that you would have to load the package first using `library(readxl)`).

[2] This is the syntax for file paths in Windows.

As mentioned above, our original data set has several entries per student. In order to study the distribution of the data by student characteristics, we need to create a new data set that has only one entry per student and preserves the variables that measure those characteristics. This can easily be done in R with the distinct function, as shown below[3]:

```
data.2018_19_distinct <- data.2018_19_filter %>% distinct(id, keep_all = TRUE)
```

The `distinct` function comes from the "dplyr" package, which is a part of the "tidyverse." It keeps only the unique/distinct rows from a data frame. If there are duplicate rows, only the first row is preserved. The `.keep_all` argument is used to retain all other variables in the output data frame.

➢ Data Visualization

Before we carry out any type of statistical analysis, it is important to explore and learn about the structure of the information in our data set. We do that through a series of plots of different types that throw light on different aspects of the data.

The first plots below are just bar plots that show how the data is distributed according to one or more features. We illustrate this process by providing the code to plot two student characteristics: gender and age. Let's start with 'Gender', a categorical variable (contains a finite number of categories or distinct groups).

```
Genderdist.2018_19 <- data.2018_19 %>% count(Gender)

Genderdist.2018_19 <- mutate(Genderdist.2018_19, prop = 100*(n / sum(n)))
```

[3] The pipe operator `%>%` will be discussed in the next section.

```
ggplot(Genderdist.2018_19, mapping = aes(x = Gender, y = prop, fill = Gender, alpha = I(0.3))) + geom_col() + xlab('Gender') + ylab('%') + theme(legend.position = "none") + scale_fill_manual(values = c("green", "red"))
```

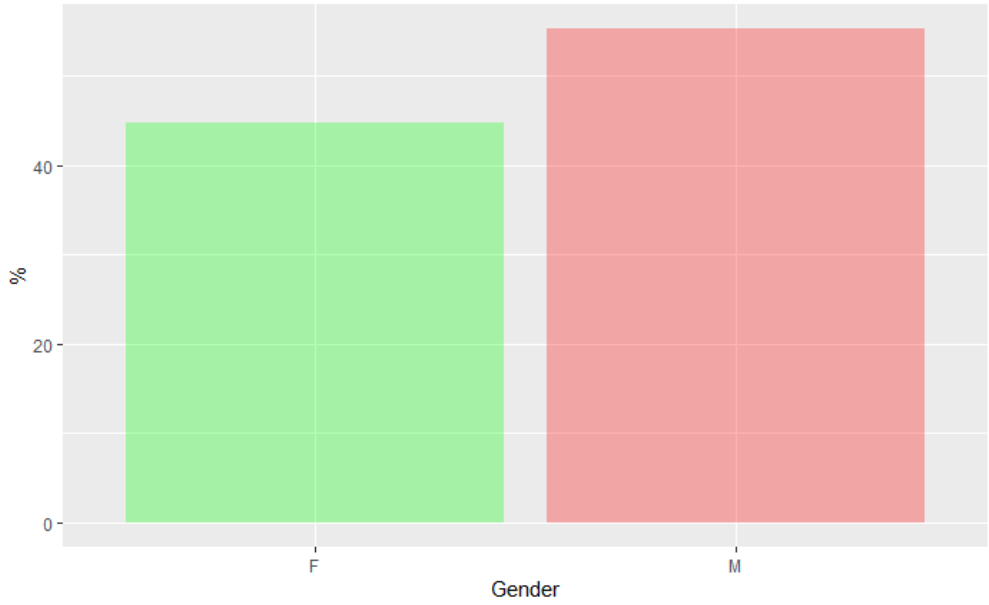
The first line of code creates a data frame we named `Genderdist.2018_19` (we could have given it many other names) using `<-`. The pipe operator `%>%` from the "dplyr" package simplifies the command syntax by inputting the object to its left into the function to its right. In the code above, `data.2018_19 %>% count(Gender)` is the same as `count(data.2018_19, Gender)`. The `count` function counts the number of occurrences of the variable 'Gender' (it is important to use capital G in our code since that is how the variable name is encoded in the data and R is case-sensitive) in `data.2018_19`.

The second line of code uses the `mutate` function from the "dplyr" package to create a new variable 'prop' that gives the proportion of the observations falling into each of the two categories of the variable 'Gender': male or female. The first argument of the mutate function is the data frame, and the second is the definition of the new variable.

Our goal is to create a bar chart that shows the relative proportion of males and females in the 2018-2019 data set. We can use the "ggplot" package to do that, as in the third line of code[4].

The geometric object (*geom* for short) `geom_col()` creates the bar plot we want. It expects the *y* values it is supposed to plot to have already been calculated. That's why we created the 'prop' variable earlier and set `y = prop` in the `mapping` argument. The output of the `ggplot` code is the picture below:

[4] See appendix for details.



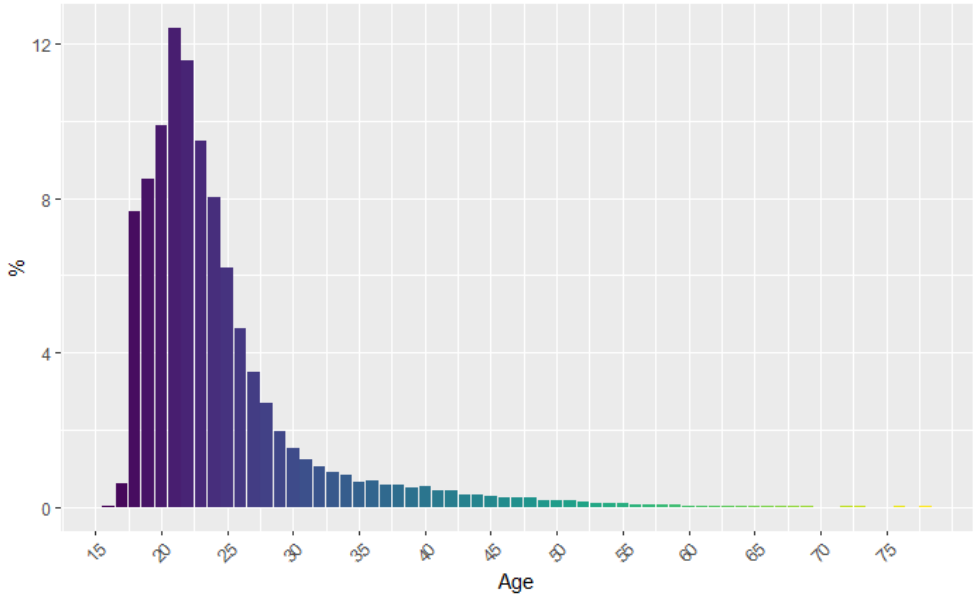
Notice that almost 45% of the students in the 2018-19 academic year were female and a little over 55% were male.

The variable 'age' is continuous, meaning it is a numeric variable that can take an infinite number of values, but we can use the same *geom* to plot it, namely `geom_col()`.

```
Agedist.2018_19 <- data.2018_19 %>% count(Age)

Agedist.2018_19 <- mutate(Agedist.2018_19, prop = 100*(n / sum(n)))

ggplot(Agedist.2018_19, mapping = aes(x = Age, y = prop, fill = Age)) +
geom_col() + xlab('Age') + ylab('%') + theme(legend.position = "none",
axis.text.x = element_text(angle = 45, hjust = 1)) + scale_x_continuous
(breaks = c(15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75)) +
scale_fill_viridis()
```

There are two main differences between the code above and the one used for 'Gender': `scale_x_continuous (breaks = c(15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75)` and `scale_fill_viridis()`. The first layer adds breaks to the plot to facilitate visualization, and the second calls a color scheme from a package called "viridis" (you need to install the package first).

The `ggplot` code above generates a picture with the distribution of UVU's student population by age in the 2018-19 academic year:



We can see that the only age groups that have a share higher than 4% are those comprised of students 18 to 26 years old.

The next step is to highlight some aspects of the interaction between UVU advisors and students. Let's say we want to know how visits to advisors in a certain college or school were distributed by major. We could write a chunk of code similar to those used for the variables 'Gender' and 'age" and then reuse it for each college/school, but this would be inefficient. There is a better way to do this in R, which requires the creation of objects called functions. The example below shows how to do it.

```
Majordist <- function(var) {
df <- data.2018_19 %>% filter(GROUP == var) %>% na.omit() %>% count(HMAJOR)
df <- mutate(df, prop = 100*(n / sum(n)))
df <- df %>% arrange(-prop)
```

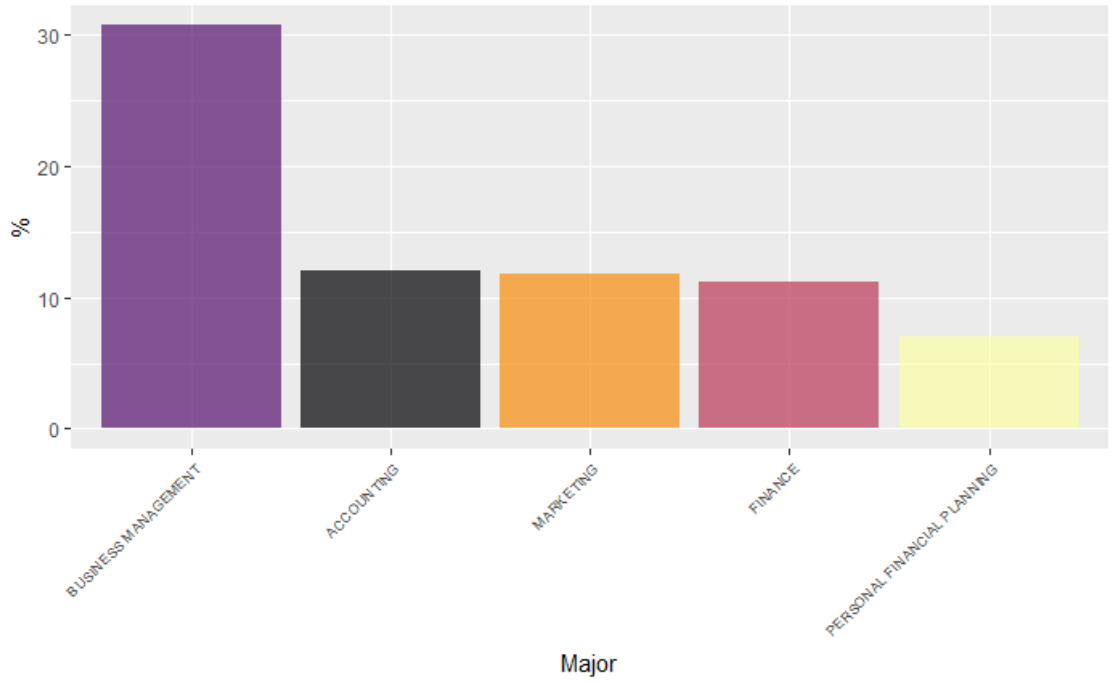
```
df %>% filter(row_number() <= 5) %>% ggplot(mapping = aes(x = reorder(HMAJOR, - prop), y = prop, fill = HMAJOR, alpha = I(0.7))) + geom_col() + xlab('Major') + ylab('%') + theme(legend.position = "none", axis.text.x = element_text(angle = 45, hjust = 1, size = 7)) + scale_fill_viridis(discrete = TRUE, option = "inferno")

}

Majordist("WOODBURY SCHOOL OF BUSINESS")
```

The first thing to pay attention to is the syntax for writing functions in R. Notice that the command `function` is followed by parentheses and one or more arguments inside them. In our example, the function assigned to the object `Majordist` (through `<-`) has a single argument named `var` (we could have used any other name of our choice). After the parentheses it is necessary to write the curly bracket `{`. The code that comes after `{` is the content, i.e., what the function is supposed to do. Don't forget to close the contents of the function with the curly bracket `}`. This tells R that the next line of code is not part of the definition of the function.

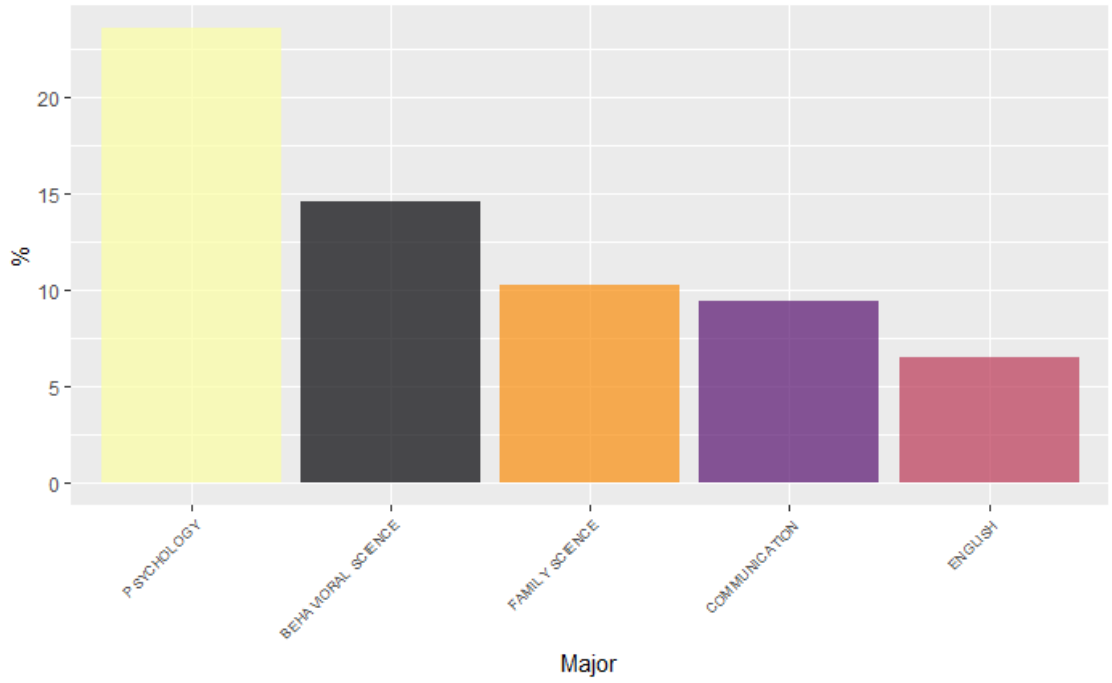
The function in our example is supposed to create a plot. If you pay close attention to the code inside the curly brackets, you will find the first two lines familiar. The first one counts the number of occurrences of the variable 'HMAJOR' (which records the student's major), while the second creates a variable that gives the proportion of the observations pertaining to each of the possible majors. However, the `filter(GROUP == var)` function in the first line after `{` is new and requires explaining. Part of the "dplyr" package, it keeps only the observations in the data set that satisfy the criterion `GROUP == var`. For instance, if var is "Woodbury School of Business", then only the observations in the data set such that the variable 'GROUP' takes the value "Woodbury School of Business" are used.

The third line of code after `{` makes use of the `arrange` command. Its purpose is to sort the data in the order of the argument it is given. Since there is a negative sign in front of the variable 'prop', the data is sorted in the inverse order of 'prop'.

The last part of the code should be familiar. The 'ggplot' funtion with the `geom_col()` layer generates a bar plot. Notice that the `filter(row_number() <= 5)` command keeps only the five majors with the highest shares. All that is left to do now is use the function. We call it by writing `Majordist("WOODBURY SCHOOL OF BUSINESS")`. This runs the function with `var` replaced by `("WOODBURY SCHOOL OF BUSINESS")`. Here is the output:



If we run `Majordist("Humanities and Social Science")`, we get the distribution of visits to advisors by majors in the college of Humanities and Social Science:



There are other plots, as well as descriptive statistics, that we can generate to illuminate and scrutinize the data set, but we will not pursue them here. Instead, we will start investigating the relationship between the treatment and outcome variables. In general, we might be interested in studying more than one outcome, but in this primer, we consider only one, namely semester grade point average, which is obtained from letter grades using a numerical scale.

Recall that the main purpose of our study is to identify the impact, if any, of visits to advisors on student success. There are many ways to measure those visits, but we will use just one: whether a student has seen an advisor or not. To do that, we create a dummy variable whose value is equal to 1 if the student has seen an advisor in a given semester and 0 otherwise. Here is the code, where we use `mutate` to create the variable 'advdummy'.

```
data.2018_19_id <- data.2018_19 %>% mutate(advdummy = ifelse(sumduration
> 0, 1, 0))
```

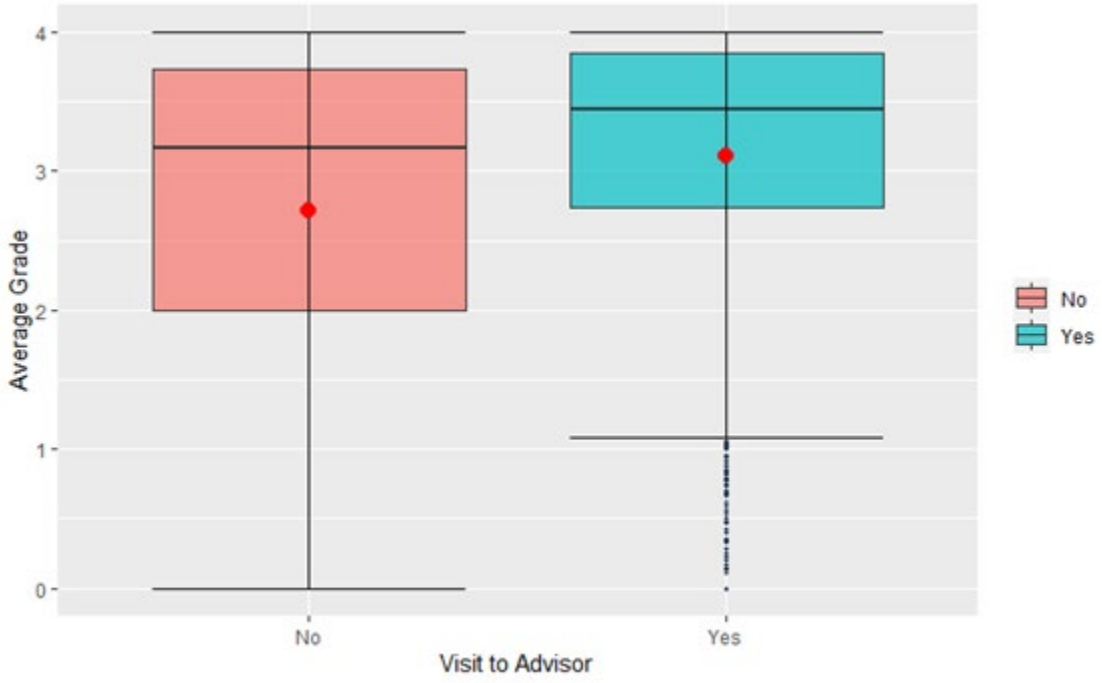
The argument of the function `mutate` contains another function called `ifelse`, which gives the following instruction: "if the variable 'sumduration' is greater than 0, 'advdummy' is equal to 1; otherwise 'advdummy' is equal to 0. In case you are wondering, 'sumduration' is a variable that gives the total time spent by a student in advising sessions in a given semester. You probably realized that we are using a different data set named `data.2018_19_id` instead of `data.2018_19`. You don't need to worry about that. We just had to apply a few transformations to the original data set to get it into the appropriate format.

We are now ready to explore the relationship between 'advdummy' and the variable that measures semester grade point average, which we call 'avggrade'. The first thing we can do is generate box plots to compare the distribution of 'avggrade' for the treatment and the comparison groups. In our study, the treatment group consists of students who did visit with an advisor in a given semester, and the comparison group comprises those who didn't. The code below creates the box plots:

```
data.2018_19_id %>% filter(semester == "Fall") %>% ggplot(aes(x = advdummy, y = avggrade, fill = advdummy)) + geom_boxplot(alpha = 0.7, outlier.colour = "#1F3552", outlier.shape = 20) + scale_y_continuous(name = "Average Grade") + scale_x_discrete(name = "Visit to Advisor") + theme(legend.title=element_blank()) + stat_boxplot(geom = 'errorbar') + stat_summary(fun = "mean", geom = "point", shape = 20, size = 5, color = "red", fill = "red")
```

The first thing to notice is that, by using `filter(semester == "Fall")`, we are restricting the data to the Fall 2018 semester. The `geom_boxplot()` layer generates the box plots. The lower and upper hinges of each box correspond to the first and third quartiles (the 25th and 75th percentiles). The solid line inside the box is the median. The upper whisker extends from the hinge to the largest value no further than 1.5*IQR from the hinge (where IQR is the inter-quartile range, or distance between the first and third quartiles). The lower whisker extends from the hinge to the smallest value at most

1.5*IQR from the hinge. Data beyond the end of the whiskers are called "outlying" points and are plotted individually. The `stat_boxplot(geom = 'errorbar')` argument adds whiskers, and the `stat_summary()` argument adds red points indicating means.



We can see that students who paid a visit to an advisor had higher average and median scores in Fall 2018. Moreover, the entire distribution of average grades for those who visited with an advisor is higher than that of students who didn't, which is an indication that visits to advisors have a positive impact on grades.

➢ Preliminary estimation of treatment effects

The box plots provided evidence of a possible relationship between visits to advisors and grades, but we need to investigate this further with the help of other statistical methods. A simple method is the t-test, which in this context assesses whether the averages of the treatment and comparison groups are statistically the same. The code is concise:

```
with(data.2018_19_id, t.test(avggrade ~ advdummy))
```

Another way to run the test would be to write the code `t.test(data.2018_19_id$avggrade ~ data.2018_19_id$advdummy))`, but to use the `$` sign to refer to variables is cumbersome. The `with()` command tells R which data set to use and frees us from having to use `$`. Here are the t-test results:

Value of t statistic	-29.854
Degrees of freedom	42,470
P-value	0.000000
Mean visited	3.094068
Mean did not visit	2.810079

Since the p-value is lower than 0.05, we reject the null hypothesis that the difference in means is equal to zero. In other words, the two means are statistically different.

A more sophisticated technique is called linear regression analysis. It is not within the scope of this primer to discuss this method, but we can think of it as trying to estimate a linear relationship between one variable, called the dependent variable, and other variables that can help explain its behavior. The latter are called explanatory variables. In our study, the dependent variable is the outcome 'avggrade', and the explanatory variable we are mainly interested in is the covariate 'advdummy'. The other explanatory variables are the covariates GPA, age, and dummies for gender, ethnicity, residency, class standing and generation (first or not).

The code to run a regression is very simple, as can be seen below:

```
lm_avggrade <- lm(avggrade ~ advdummy + GPA + gender + age + resident + ethnicity + class + firstgen, data = data.2018_19_id, na.action = na.omit)

summary(lm_avggrade)
```

The first two lines of code run the regression. The function for that is `lm()`. The function call requires, at a minimum, a formula and a data set. In our code, the formula `avggrade ~ advdummy + GPA + gender + age + resident + ethnicity + class + firstgen` specifies that the dependent variable is 'avggrade', and that the explanatory variables are those to the right of the ~ symbol. The argument `na.action = na.omit` instructs R to ignore observations with missing values. Notice that we saved the outcome of the regression into an object named `lm_avggrade`. The `summary(lm_avggrade)` command simply prints a summary of the results inside of it, which we display in the table below.

Variable	Coefficient	St. error	T statistic	P-value
Intercept	0.194074	0.057312	3.386	0.000709
Visit	0.196305	0.007811	25.132	< 2e-16
GPA	0.800732	0.005444	147.08	< 2e-16
Gender	-0.10285	0.007625	-13.487	< 2e-16
Age	0.002765	0.000652	4.239	2.25E-05
Resident	-0.01424	0.012913	-1.103	0.27021
Asian	0.089182	0.060579	1.472	0.140983
Black	0.042388	0.063923	0.663	0.507266
Hispanic	0.039959	0.052205	0.765	0.444023
Multi-racial	0.077376	0.055224	1.401	0.161178

Native Hawaiian/Pacific Islander	-0.04555	0.066676	-0.683	0.494494
Non-resident Alien	0.006361	0.058088	0.11	0.912799
Ethnicity Unknown	0.101143	0.062055	1.63	0.103128
White	0.136669	0.051247	2.667	0.007658
Junior	0.115038	0.012001	9.586	< 2e-16
Senior	0.182312	0.01185	15.386	< 2e-16
Sophomore	0.047211	0.011389	4.145	3.40E-05
Unknown Generation	-0.11339	0.011201	-10.124	< 2e-16
First Generation	-0.04813	0.008723	-5.517	3.46E-08

Notice that the 'visit' variable is significant (p-value less than 0.05) and positive. This means there is statistical evidence that visiting an advisor in a given semester has a positive impact on students' grades in that semester. Among the other explanatory variables, GPA, gender, age, and all the class standing and generation dummies are statistically significant. The residency dummy and all the ethnicity dummies, with the exception of 'white', are not significant.

Regression analysis is a step forward in our journey to include as many measurable confounding features as possible in our model. As explained in the discussion about the limits of observational studies, this is necessary to eliminate as many potential influences on the outcome other than the treatment as possible. It can be shown, however, that in many instances linear regression is not very effective at doing that. In the next section, we explore a technique called matching by propensity

score, which paired with linear regression does a much better job of controlling for those biases.

➢ Matching by propensity score

The idea of matching methods is to match each treated unit to one or more comparison units based on some distance metric that standardizes the covariates. One possibility would be to compare units one covariate at a time. For instance, people would be first matched on age. Then those in the same age group would be matched on gender, the resulting groups matched on ethnicity and so on. This process has two problems: as the number of covariates increases, it becomes increasingly complex and might result in a large number of non-matched units.

The propensity score resolves this problem. It collects covariate information into a single metric that can then be used to match the data. Formally, the propensity score is defined as the conditional probability of assignment to a treatment given the observed covariates (Rosenbaum and Rubin, 1983). When our sample consists of individuals, we can interpret the propensity score as the likelihood that an individual with certain age, ethnicity, gender and other characteristics would receive treatment.

The propensity score is typically not known and needs to be estimated. There are several ways to do that, but in this primer, we use logistic regression. It is beyond the scope of this primer to give a detailed description of this method. For our purposes, it suffices to know that, just like linear regression, its goal is to estimate a relationship between a dependent variable and explanatory variables. The main differences are that in logistic regression the dependent variable is discrete, meaning it has a finite number of possible values, and the output of the regression is a number greater than or equal to 0 and less than or equal to 1. In our application, the dependent variable can be equal to 0 or 1 (we call it binary), and we are trying to estimate the probability of treatment given the covariates, which must be a number between 0 and 1.

The following code runs a logistic regression to estimate the propensity score:

```
ps_adv <- glm(advdummy ~ GPA + gender + age + resident + ethnicity +
class + firstgen, family = binomial(), data = data.2018_19_id_nomiss)

summary(ps_adv)
```

The `glm()` function is similar to the `lm()` function we used for linear regression. In fact, it can also be used to run linear regressions. In this particular application, the option `family = binomial()` tells R it should estimate a logistic regression model. The other commands have already appeared before. The output (not reproduced here) is similar to that generated by `lm()`, containing coefficients, standard errors, P-values and other statistics.

The next step is to use the propensity score estimates to match the treatment group to a comparison group. There are several packages is R that accomplish that. We will use one of the most comprehensive and popular, called MatchIt. As usual, we have to install it first and then load it by calling `library (MatchIt)`.

The matching method we discuss here matches a treated unit to a comparison unit that is closest in terms of a distance metric based on the propensity score. In other words, it pairs observations with similar propensity scores. The function to implement this method, called `matchit`, can estimate the propensity score in the background, rendering the `glm` estimation we performed above unnecessary. Nevertheless, it is a good exercise to estimate the propensity score separately, for reasons that will soon become apparent.

The code to run what has a technical name of "greedy one-to-one matching with replacement and no caliper" is:

```
advising_match <- matchit(advdummy ~ GPA + gender + age + resident +
ethnicity + class + firstgen, data = data.2018_19_id_nomiss, distance =
```

```
data.2018_19_id_nomiss$logitPScores, method = "nearest", ratio = 1, replace = T, m.order = "largest")
```

Notice first that we are saving the outcome of the matching process in an object called `advising_match`. The formula inside parentheses tells R to do the matching based on the treatment variable 'advdummy' using the covariates 'GPA', 'gender', 'age', 'resident', 'ethnicity', 'class' and 'firstgen'. The option `distance = data.2018_19_id_nomiss$logitPScores` specifies that the log of the propensity score instead of the propensity score itself should be used as the distance metric. This typically produces better matches, but it is a technicality we will not get into. Had we not run a logistic regression earlier to estimate propensity scores, we would not have been able to use this option. The command to generate the log of the propensity scores is `data.2018_19_id_nomiss$logitPScores <- log(fitted(ps_adv)/(1-fitted(ps_adv)))`, where `fitted(ps_adv)` gives us the estimated propensity scores from the object `ps_adv` (remember that this object stores the output of the `glm` function we used above to run the logistic regression).

We will not go over the `m.order = "largest"` option, but the other options should be self-explanatory: `method = "nearest"` indicates we are matching units that are closest in terms of the distance metric; `ratio = 1` means each treated unit is matched to one comparison unit; and `replace = T` instructs R to perform matching with replacement. This is important because our data set has more treated than comparison observations. If we don't sample with replacement, not all treated units will be included in the matched data set.

Once we have a match, we need to check if it has generated an acceptable covariate balance, i.e. if the covariates have similar post-match distributions. After all, we want the treated and comparison units to be as similar as possible. There are many statistics that can help us determine whether the post-match distributions are close enough, but we will focus on just one: the standardized mean difference (SMD), defined

as the weighted difference in the means of a covariate between the treatment and comparison groups divided by the standard deviation of the treatment group. The closer to zero the SMD the better that covariate's balance.

There is a package in R called "cobalt" that makes it very easy to calculate the SMDs (as well as other covariate balance measures). After we install it and call it using `library (cobalt)`, we run the code below, where `binary = "std"` gives the instruction to compute SMDs and `un = TRUE`, to include the pre-match values of SMDs:

```
bal.tab(advising_match, binary = "std", un = TRUE)
```

The output is summarized in the following table:

Variable	Full sample	Matched sample
Propensity score	0.2484	0.0000
GPA	0.0741	-0.0198
Gender (female)	0.0008	0.0027
Age	0.0115	0.0039
Resident	-0.0538	-0.0001
Asian	0.0156	0.0058
Black	0.0141	0.0023
Hispanic	-0.0007	0.0011
Multi-racial	0.0042	0.0172
Native Hawaiian/Pacific Islander	0.02	-0.0072
Non-resident Alien	0.0912	0.0061
Ethnicity Unknown	-0.007	0.0024

White	-0.0515	-0.0099
Junior	0.0547	-0.0057
Senior	0.137	0.0043
Sophomore	-0.0376	0.0061
Unknown Generation	-0.0495	0.003
First Generation	-0.0243	0.0129
# treated	35,364	35,364
# comparison	20,870	14,459

We can see that the matching procedure improved covariate balance significantly. Typically, we are happy with an STD lower than 0.1 (Stuart et al., 2013). Notice also that all the treated units were matched, while 14,459 of the 20,870 comparison units were matched.

➢ Estimating treatment effects with the matched sample

The final step in our journey to determine the impact of advising on student performance is to estimate treatment effects. There are many measures used in the literature to do that, but we will only consider the simplest method: run a linear regression on the matched data set. The code below accomplishes that:

```
adv_data <- match.data(advising_match)
```

```
lm_match_cov <- lm(avggrade ~ advdummy + GPA + gender + age + resident
+ ethnicity + class + firstgen, data = adv_data)
```

```
summary(lm_match_cov)
```

The only part of the code that is new to us is the first line. The `match.data()` function, which comes from the MatchIt package, extracts the matched data set from the `advising_match` object we created earlier. It excludes unmatched units from the original data and adds information produced by the matching procedure, including a variable called distance that stores the propensity score estimates.

The results of this new linear regression can be found in the table below:

Variable	Coefficient	St. error	T statistic	P-value
Intercept	0.222845	0.058515	3.808	0.00014
Visit	0.195639	0.008595	22.762	< 2e-16
GPA	0.797561	0.005709	139.706	< 2e-16
Gender	-0.09697	0.007921	-12.241	< 2e-16
Age	0.00199	0.000682	2.919	0.00351
Resident	-0.0104	0.013363	-0.779	0.43622
Asian	0.081493	0.061491	1.325	0.18508
Black	0.035364	0.065097	0.543	0.58696
Hispanic	0.041043	0.052874	0.776	0.43761
Multi-racial	0.086845	0.056114	1.548	0.12171

Native Hawaiian/Pacific Islander	-0.03181	0.067405	-0.472	0.63696
Non-resident Alien	0.010867	0.058719	0.185	0.85318
Ethnicity Unknown	0.108663	0.063521	1.711	0.08715
White	0.13795	0.051849	2.661	0.0078
Junior	0.106172	0.0126	8.426	< 2e-16
Senior	0.167148	0.012417	13.461	< 2e-16
Sophomore	0.03522	0.01208	2.916	0.00355
Unknown Generation	-0.10465	0.011784	-8.881	< 2e-16
First Generation	-0.04301	0.009073	-4.741	2.14E-06

Our original regression findings are confirmed. The variable 'visits' still has a positive and highly significant impact on students' average semester grades ('avggrade'). The magnitudes of the coefficients of the other variables are similar to those we obtained earlier, and the same coefficients are significant.

The propensity score matching analysis thus confirms that visits to advisors in a given semester have a positive impact on student performance in that same semester. In other words, we obtained evidence that the treatment effects are statistically significant and positive.

This is not the end of the story. We should run a sensitivity analysis to assess the amount of bias that would need to exist to invalidate the conclusion that there is a causal

relationship between advising and grades. That analysis, however, will have to wait for a future workshop.

❖ References

Gertler, P., S. Martinez, P. Premand. L. Rawlings, and C. Vermeersch. (2016) **Impact Evaluation in Practice**. Washington, D.C.: World Bank Group, 2nd edition.

Rosenbaum, P. R., and D. B. Rubin. (1983) "The Central Role of the Propensity Score in Observational Studies for Causal Effects." Biometrika, Vol. 70, No. 1, pp. 41-55.

Stuart, E. A. (2010) "Matching Methods for Causal Inference: A Review and a Look Forward." Statistical Science, Vol. 25, No. 1, pp. 1-21.

Stuart, E. A., B. Lee, and F. Leacy. (2013) "Prognostic Score-based Balance Measures Can Be a Useful Diagnostic for Propensity Score Methods in Comparative Effectiveness Research." *Journal of Clinical Epidemiology*, Vol. 66, No. 8, pp. S84-S90.

❖Appendix

➢ Getting Started with Base R

➢What is R?

Before you can make data visualizations in *ggplot*, you'll need to understand some basics of the R programming language. R is popular in academia and industry for a number of reasons:

- It's open source, with an active community of developers who continually extend its functionality via packages.
- It can run on Windows, MacOS, and UNIX.
- It includes robust array of tools for data storage, handling, manipulation.
- It's powerful environment for statistical analysis and graphical presentation.

According to the PYPL Index, R is the 7th most popular programming language worldwide. Additionally, R has packages that allow users to run Python code, SQL queries, interact with their Google Drive, and more!

➢Learning R

R is a little difficult to learn, especially for first time programmers, because it has a long history starting in 1976. This age plays into the coding process since any given task may have multiple solutions, with functions that still work even if they are deprecated, meaning they are not the newest or most acceptable method.

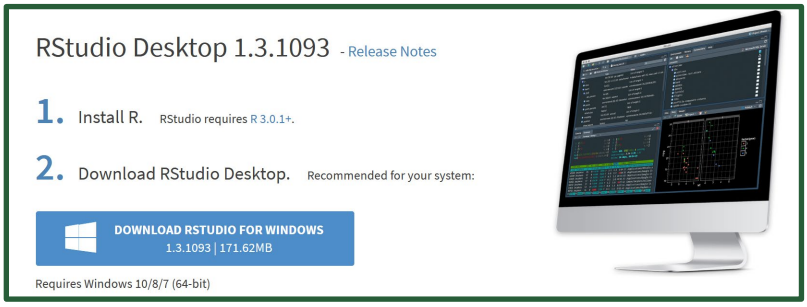
R can be interpreted as both *functional* and *object-oriented* depending on what perspective you employ. This introduction will use the object-oriented programming approach.

Luckily, there are a number of free resources, like the book *R for Data Science*, available online. This introduction will explain functions and syntax as they come up in

the case study, as well as a few basics in following sections[5]. Code will `look like this` throughout, along with RStudio screenshots.

➢Installing RStudio

Integrated developer environments (IDEs) make programming easier, since they organize tools into a graphical user interface (GUI). RStudio[6] is an open source IDE that will allow us to create plots, run code, and manage our data.



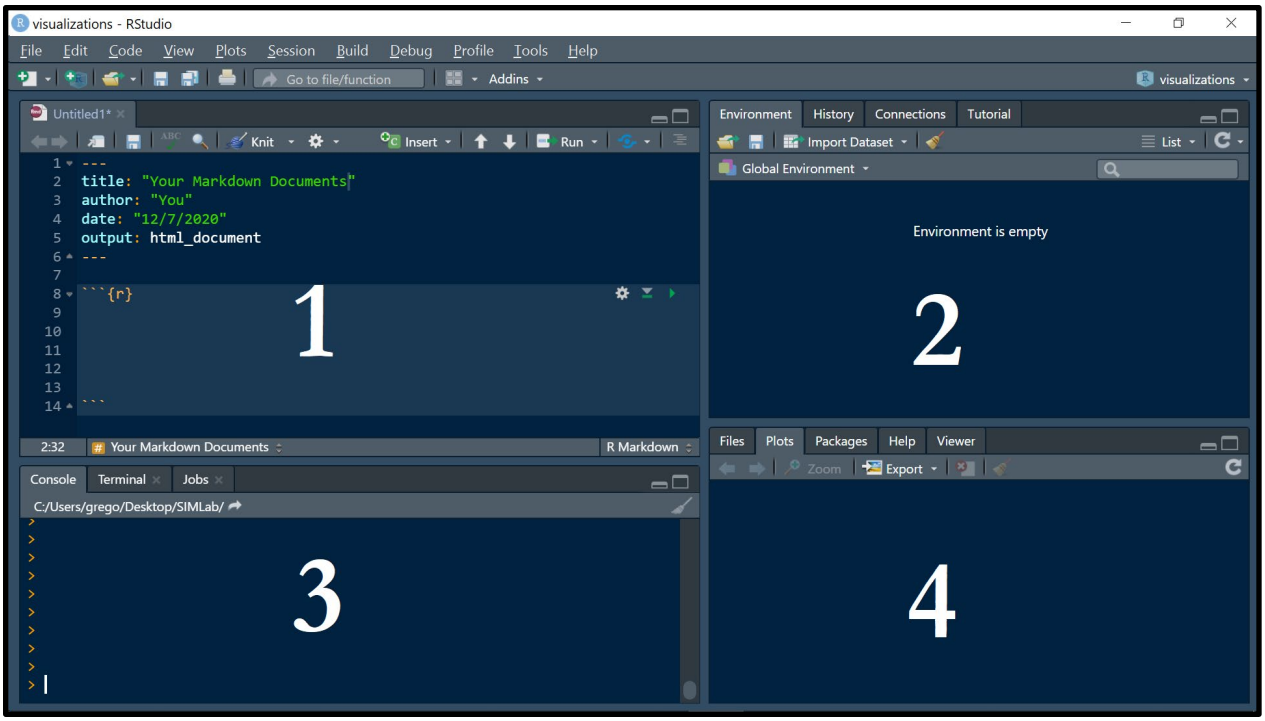
Go to the RStudio website download page (here), and follow the steps. If you do not have all installed, then click on the link in Step 1. Once RStudio is downloaded, run the program!

There are several panes in RStudio, including ones for displaying plots or downloading packages (4), a console (3), an environment pane that helps you manage objects and data (2), as well as the main project pane (1):

---

[5] The information here was parsed from *Advanced R* by Hadley Wickham, who was also the creator of the Tidyverse package, any work by Hadley is a great place to learn.. Explanations also adapted from this Standford presentation by Brandon Steward and Patrick Lam

[6] Their website also offers great tutorials and cheat sheets for learning R



The console displays code you are running, but you can also type commands there. The console can be useful for doing tasks you don't need to save in the code file. For example, typing `install.packages(tidyverse)` in the console downloads the "tidyverse" ecosystem of packages. You can then use that functions from that package by including `library(tidyverse)` at the beginning your code.

■ The Power of R Packages

"In R, the fundamental unit of shareable code is the package. A package bundles together code, data, documentation, and tests, and is easy to share with others. As of June 2019, there were over 14,000 packages available on the **C**omprehensive **R** **A**rchive **N**etwork, or CRAN, the public clearing house for R packages. This huge variety of packages is one of the reasons that R is so successful: the chances are that someone has already solved a problem that you're working on, and you can benefit from their work by downloading their package."

- quote from *R packages* by Hadley Wickham & Jenny Bryan

- **Customizing Appearance**

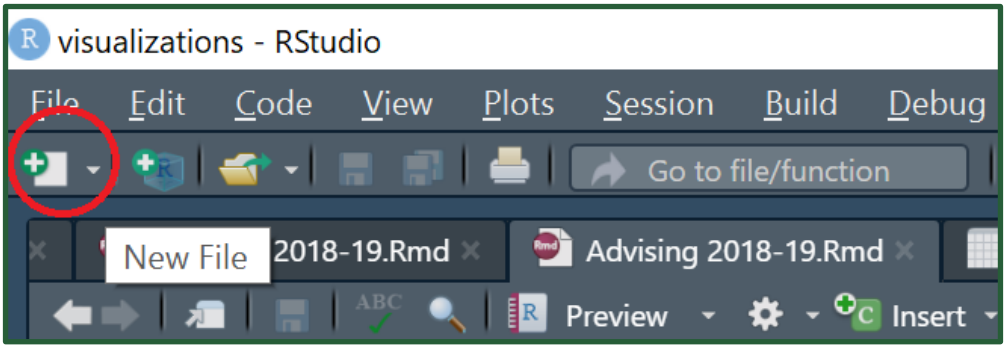
The screenshots in this document may not look exactly like RStudio out-of-the box because there are many options for customizing appearance. Using RStudio can also be streamlined by using keyboard shortcuts for common tasks, see list here.

➢Creating an R Markdown Document

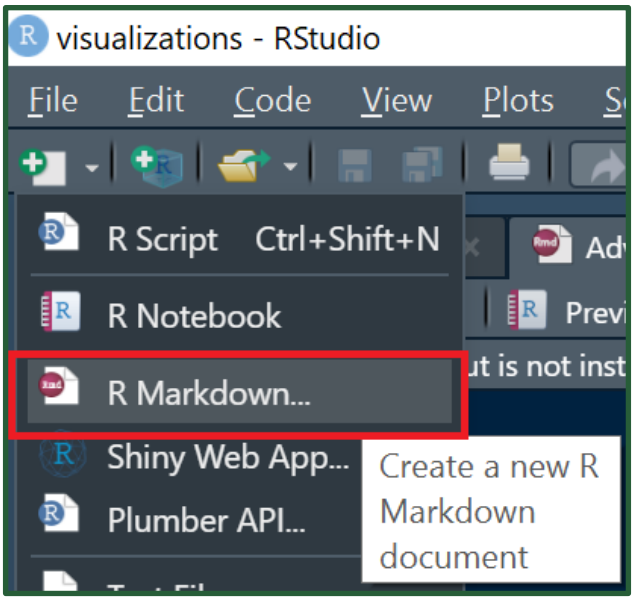
Deciding on the type of R file is important, depending on your goals for your project. If you want to write notes about your analyses in the same file as the code you run, or include samples of code, you may want to use an R Markdown document.

R Markdown is useful because the code, analyses, and plots created can be exported in a number of formats - .pdf, .docx, etc. It also aids in creating reproducible research by documenting the choices you make in your analysis.

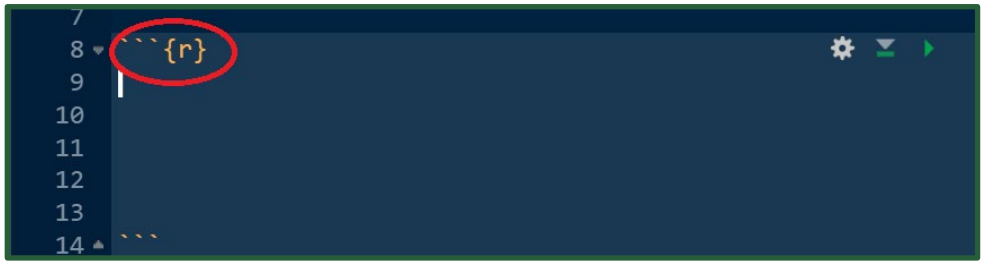
The following are steps for creating an R Markdown Document in RStudio:



Click the new file icon, a blank page with a green plus sign, and select R Markdown from the drop down list:



Once the document is created, it will have information about R Markdown that you can delete. Your document's first *chunk* will look like the picture below. Chunks can be labelled, and are a good way of organizing your code. You can run all the code in a chunk by pressing the green triangle on the far right, or hide chunks by pressing the grey triangle on the left.



➢ Using ggplot

➢ Getting Started

➢ *ggplot* is a package in R that utilizes a "grammar of graphics"[7] to create data visualizations. This framework is defined by its

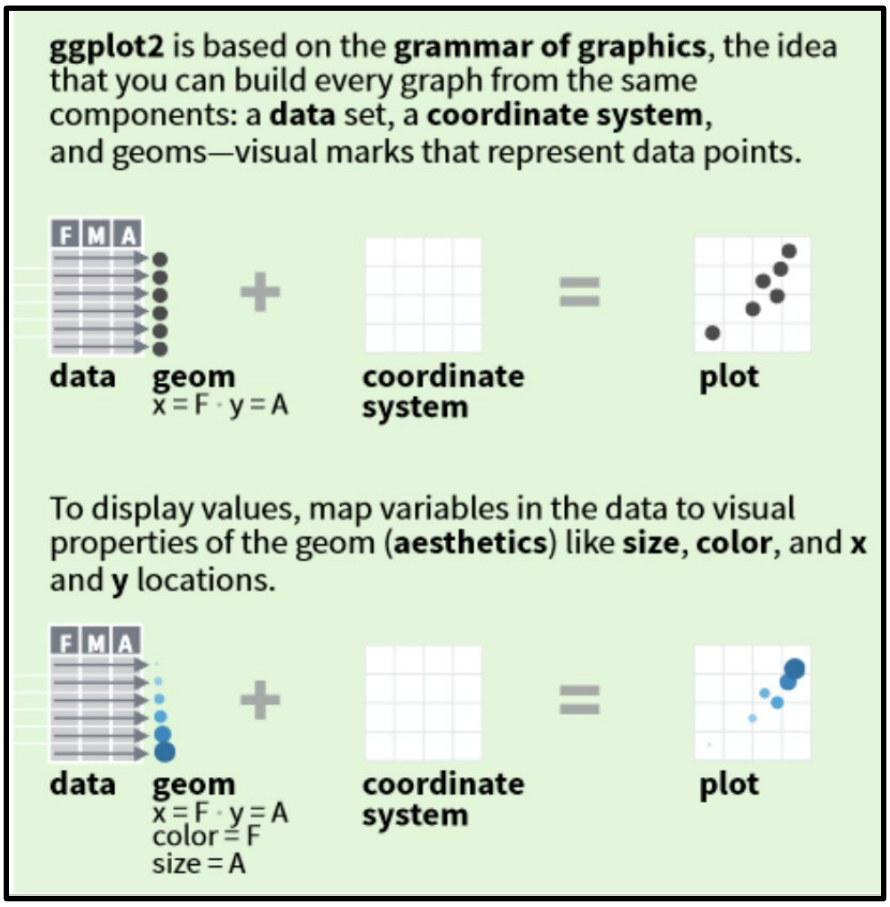
[7] *The Grammar of Graphics*, pg. 405, 2005, Leland Wilkinson.

structured and layered method for describing data. *ggplot* was developed as a robust, open source tool that "concisely describes components of a graphic."

➢ *ggplot* is one of *many* packages that follow the Tidyverse philosophy. *ggplot* exists in an "ecosystem" of other packages that all follow the same principles, where the code is consistent across the board, so learning one helps you learn the others. The code is meant to be readable, tidy, functional, and highly descriptive so as not to obscure how the underlying data is being used.

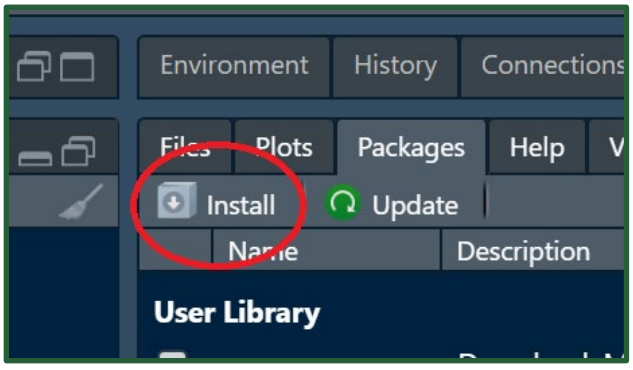
ggplot2 allows us to build data visualizations. There are many ways of using *ggplot*, especially within the wider framework of the Tidyverse. *ggplot* builds a graph by converting an underlying dataset into plots on a coordinate system. Then layers called geoms are added to visualize the data in the form you choose. There are hundreds of options for customization, as well as libraries that have preset themes and aesthetics.

Here's an infographic from the *ggplot* cheatsheet:

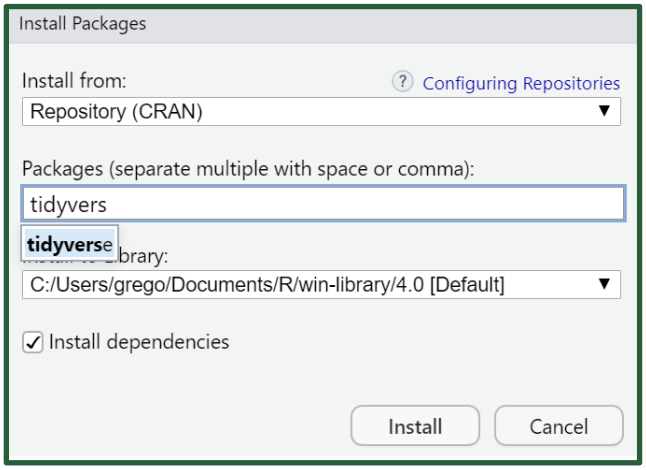


➢ Installing ggplot

- We can use a function: install.packages("ggplot")
- Or we can use an integrated development environment (IDE), like RStudio where installing packages is user-friendly:
  - Click on the install button (bottom right corner)



- Then this dialogue box will pop up, where you type in the name of the package, and click install.
- Installation can take some time, since the Tidyverse is multiple discrete packages.



- `library(tidyverse)` — adding this library to your code gives you *ggplot* as well as many other packages, like *dplyr*, *lubridate*, *readr*, and *tidyr*, that can be used in tandem.

- **Basic Functions** (full list here)
  `ggplot()` — This creates a new *ggplot* object based on the data we provide. We'll often use a `data.frame()` for this function. A data frame holds our variables, and is the "fundamental data structure" used in R.

  This is the basic format below:

  ```
  ggplot(data = NULL, mapping = aes())
  ```

  Your data frame will replace the `NULL`, and your visual choices will go into the `aes()` argument, where it is "mapped" onto the chart.

  Here's an example from a case study we'll get to later:

  ```
  mapping = aes(x = Gender, y = prop, fill = Gender,
  alpha = I(0.3))
  ```

Notice the different options, like `alpha` which determines the transparency. `x` and `y` determine the different axes of our chart, in this case they are objects, but they could also be strings. **Like this**:

```
mapping = aes(x = "Males", y = "Total")
```

- **`Geom_`** — This adds the actual geometric item that represents our data. There are many different kinds, including scatter plots, bar charts, and maps. We will cover these in more detail in **Case Study 1**.
    - Common functions we use:
        - `geom_col()`
        - `geom_boxplot()`
        - `geom_point()`

- **`Position_`** — Allows you to prevent overlapping geoms, like making points in a scatter plot "jitter" slightly for better visibility.
- **`Aes_`** — Short for aesthetics, lets you customize some visual details of the plot, like the transparency of a bar chart geom, or the color of points in a scatter plot.
- **`Scales_`** — This function family is important to make the plot look exactly how you want in terms of axis labels, legends, and titles. There are also color scales, positioning options, limits and more.
    - Common functions we use:
        - `ggtitle()`
        - `lims()`
        - `scale_x_continuous()`
- **`Facet_grid()`** & **`Facet_wrap()`** — This is a different way of visualizing discrete variables, kind of like a visual "subset" of your plot. Labels of each facet are created with `labeller()`.
- **`Theme_`** — This function family is similar to scales, except that it works with non-data details. Using `theme()` & `element_` lets you

quickly get rid of a legend you do not need, for example, or alter the margins.

- **ggsave()**—Handy for saving your charts so they can be inserted into a presentation. The default resolution is a little low, only 300 dpi, so you can use this function to save your chart at a higher resolution fit for a report.
