# FILENAME: 00\_Getting\_Started

* GitHub Desktop - Git is a ***version control system*** and Github Desktop is a "point and click" version of Git designed to work with GitHub, which makes it easier to work on projects hosted on Github (it is annoying that the names are so similar, but Git GitHub GitHub Desktop!)

# FILENAME: 01\_Motivation

# Motivation,

## Or: Should I take this class (A: YES!)

Jobs and pay by language Python is #2 among languages in terms of high salary jobs:

Coding skills are valuable to your resume because computing power can make you more productive at tasks. You'll see an example of this on our first day in class. This is why knowing how to code will open more interview doors and possibly lead to improved job offers (and change the trajectory of your career at future steps).

A saw can be dangerous, useless, or useful - which it is depends on the context you use it in.

ML techniques, computing power, and big data can be dangerous, useless, or useful - which it is depends on the context you use it in. In particular, what are the relevant economics you're facing? A great coder might ignore the economics, but your previous classes will help you elevate the economics!

Adding this class to your Lehigh business education gives you a set of complementary skills: The ability to use high power datasets and analysis within a framework for problem solving.

## How to create value

One way:

1. Define your project: A clearly specified question with metrics for success and idea of impact. Always keep the big picture, and economic context, in mind!
2. Acquire data and clean it: Age old wisdom tells us that if the input is crap, the output will be... Thus, time spent on cleaning data is often more valuable than time spent on modeling.
3. Deliver the project conclusions to higher ups in the form of clear business recommendations. Writing should always be geared to the audience, and managers typically want bottom lines, whereas technical leads need more technical justification.

# FILENAME: 02\_Setup

1. Join the class's GitHub organization: Go to the coursesite for this class, and click the link to join the class's "GitHub organization".
   * Bonus: Play around and explore Jupyter Lab. [For a walkthough of what you're seeing, this page should help](https://jupyterlab.readthedocs.io/en/latest/user/interface.html).
   * **Mac users**: Your TA installed Git via Homebrew. Go to <https://brew.sh>, copy the home-brew address (they have a little copy paste icon which makes it easier for students), then type brew install git into the terminal and it'll install git. (There is no config necessary, type "y " when prompted with y/n?)

* If Mac users install Git via Xcode, you'll have to install Xcode and essentially waste 4gb of space you probably won't be using.
  1. Email your TA your github username and a screenshot of your computer where we can see GitHub Desktop and Jupyter Lab open.

These extensions will give you spell checking, code hints, and automatic formatting for your code 😍.

# FILENAME: 03a\_githubworkflow

# The Github Workflow

**Then use the 3 step github workflow below!**

This habit will help you avoid disasters, so that you get the positive features of Github without the headaches.

\_Being careful about these steps might seem pointless during solo projects, but I encourage you to practice these good habits now, so that when you do collaborative work, you're protected from mistakes.\_  
  
1. Make your coffee, open Github Desktop, and \*\*FETCH\*\* the project(s) you'll work on.   
1. Click "Fetch origin" to download any changes from the master repo on the Github servers. This is important, because if someone else changed the files while you were sleeping, you'll get the most updated files to work on.   
1. Start your work on your computer.   
  
If you don't "fetch" your project before you start, it's becomes easier to change a file someone else changed differently, creating a conflict. When this happens, you have to resolve the conflicting files before moving on.

Fetch the textbook repo everyday before and after class. I add lecture slides as the semester goes on.

* Now, you've got an up-to-date backup and teammates can see the changes and work with the latest files.

# FILENAME: 03\_github

***Note: I use Windows and will be of less help for Mac users. The TA, however, uses Mac and will be more helpful when issues stem from OS differences. GitHub, in particular,***

* + An *organization* account can be owned by multiple people, and typically holds repositories relevant to a group. The [LeDataSciFi organization](https://github.com/LeDataSciFi/) account is owned by me, and contains the textbook and all of our assignments.

Examples:

* The [awesome-python](https://github.com/vinta/awesome-python) repo is a "curated list of awesome Python frameworks, libraries, software and resources"

* + You and collaborators can download ("fetch") the remote repo to your computer. The folder and files on your computer are called the "local" version.

I think you can use these check boxes to track your progress, if you'd like.

1. **Download a file from GitHub.com:** Go to any repo, and click on any file. On the next page that opens, right click the "Raw" button and "Save Link As".
2. View commit history of the [LeDataSciFi.github.io](https://github.com/LeDataSciFi/LeDataSciFi.github.io) repository by clicking on the "commits" button on the repo home page. (You'll end up [here](https://github.com/LeDataSciFi/LeDataSciFi.github.io/commits/master).)
3. View the history of a file by clicking on the any, then clicking "History".

# FILENAME: 04\_Markdown

Markdown is a just way to give text in a readme or notebook formatting.

* Scroll down and look at the readme for [this repo](https://github.com/LeDataSciFi/ledatascifi-2023) or [this repo](09_gitignore). You can see that there is bolded and italics text, code, hyperlinks and headers for sectioning. There is a lot more you can do with Markdown, but that will give you a taste.

# FILENAME: 05\_jupyterlab

In fact, almost every page on this website is a Jupyter Notebook! Notice how there are headers, full text formatting, media inserts, and also code snippets and output. This means readers see output immediately after the relevant code, and makes understanding code much easier.

* is browser based

1. Find your **class notes folder** by navigating inside of the "file browser" (click on the folder icon on the left part of the screen)
2. Create a folder: right click in file explorer pane and select "New Folder"

This means that the effect of typing at the keyboard depends on which mode you are in. The two modes are

* + **Switch into edit mode by hitting Enter or double clicking in the cell.**

You can speed up youtube videos to save time. Click the "gear" icon in the lower right of the video, then open submenu "Playback speed".

(I don't use magic commanads much, but those that do find them very helpful.)

# FILENAME: 06\_python

This page is long, but important. It's structured as a walkthrough - you should run the code on your computer as you read it.

Research strongly indicates that active learning is the most effective way to learn new skills. That's why I linked to tutorials above.   
  
To the extent possible, I want you to get comfortable typing commands yourself rather than copy-pasting. This is slightly more painful in the beginning, but much better payoff in the long-run.

**Leaving GOOD comments in the code is important!** [Good, smart code](https://web.stanford.edu/~gentzkow/research/CodeAndData.xhtml#magicparlabel-1130) tries to reduce the use of comments by writing code so obvious that it is "self-documenting" (I'll explain why later).

The **identity operators** is and is not check whether the left side and the right side are the same object.

**WARNING: is and == are NOT the same!!!**\* Here is an example [borrowed from G4G](https://www.geeksforgeeks.org/difference-operator-python/).

**Parentheses:** You can (and certainly will at some point need to) check for the truth of statements involving many variables, and complex logic requests. You can dictate the order Python evaluates statements. So, for example,

### Built in data structures

* when a set is useful, when a dictionary is useful (as opposed to a list)

1. Returns the odd numbered elements (i.e. [8,6,7]).

...because in the second version the second print line isn't indented and therefore isn't part of the "if" statement

1. then, all the code of the function is indented and begins on on the next line

- \*\*Students new to functions usually want to end them with something like `print(answer)`.\*\* Don't! End functions with return, like `return answer`. If you use print, you can't use the function's output for anything else after it finishes. If you use return, you can save the output and use it (even if its just to print it).

See the tips in the [Jupyter Lab](05_jupyterlab#resources) page.

# FILENAME: 07a\_errors

Below, I try to fix a two line program, and you'll see the most common error types. There are more error types in python, and for more info, I'll refer you to

* [the official python documentation](https://docs.python.org/3/tutorial/errors.html) for more through coverage

Is pretty self explanatory but common, especially when you're writing code with multiple layers of for, if, def, try, etc...

1. [Tutorialsteach](https://www.tutorialsteacher.com/python/exception-handling-in-python)

# FILENAME: 07\_debugging

Much of this class will require delivery of ipynb files, and I would recommended using Jupyter exclusively at the beginning.

# FILENAME: 08\_libraries

The process of doing homeworks or working on projects will inevitably create files that you might not want to sync to the remote repo on GitHub.com.

1. You can add it later on github or on your computer. The file is a simple text file and should be named .gitignore

Editting it: Any text editor!

That's because it was already on the remote repo when you added it to the gitignore; **The gitignore file simply tells git to "not sync" those files any more. It doesn't mean "delete these files from the remote repo".**

Open terminal/powershell, **move to the current directory to the root folder of your repo,** and then run these commands in order:

# FILENAME: 10\_Golden\_1

The goal of this chapter is to give you a set of rules, guidance, and some hard-earned wisdom to help you write better programs and develop better projects. You'll code faster, the coding will be easier and result in code that is easier to understand, maintain, scale, debug, and repurpose.

It turns out that these issues are not unique to me. Economists, financiers, social scientists, and researchers that are trained outside of software engineering or computer science are often (especially early on in careers) unaware of solutions those fields solved for problems that social scientists are only now (at least to them) just facing. It's very common, for example:

* You want to change your sample for tests #4-#12 (in an analysis with 15 test), but the code that defines the sample has been copied and pasted throughout our project directory, and making the change requires updating dozens of files. Turns out, there were copy and paste errors, so those tests didn't even use the same sample!

# FILENAME: 10\_Golden\_2

And no, this isn't a joke - I've definitely seen "professional" researchers with project organized like this in one way or another.

# FILENAME: 10\_Golden\_3

A + B = your folders and files will be largely self documenting

A. "Pseudo code" is writing out the broad steps in plain language. I often (almost always for complicated tasks) do this on paper, then translate it to code as an outline (in the code's comments).

# FILENAME: 10\_Golden\_4

Let's put that into practice. First we need to agree on three things:

1. Definition: **The "unit" of a dataset refers the level of aggregation of variables in the dataset.** Examples:
   * Daily-data (or any time interval): Market returns

Rule 3 means that if the key is a county, and the unit is the county in 2010 (a cross section at a single point in time), then the variables you can have in that dataset are attributes of the county as of 2010.

So at some point you will combine datasets with different unit levels. Do your best to delay that (**4.B**) until the beginning of the analysis.

# FILENAME: 00\_Data\_Wrangling\_Intro

and introduce you to some seriously high powered finance datasets. What is data science without ***BIIIIG DATA***?

# FILENAME: 01a\_NumpyCS

**MY MAIN POINT HERE:** The files we deal with will get larger as the class moves forward. Typically, a file on the computer hard drive takes up *twice* as much space in memory (RAM).

This is why Python is a "high level" language - it is written at a level far from the computer's language (binary).

* Usually, your code is slow because of one or only a few lines of code. If the code **must** be speed up, we can identify the culprit, and apply one of a few fixes.

That's ok! This isn't a computer science class! Here are the takeaways of this page:

# FILENAME: 02a\_pandasIntro

So I'm going to try to reduce it to the most common operations, and make you aware of things you can do.

# FILENAME: 02b\_pandasVocab

* I try to make my data "tidy" at the start of analysis. Tidy data is quicker to analyze!
  + [This is a good description](https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html) of what "tidy" is, what "messy" often is, and how to tidy messy data. Focus not on the code, but the textual ideas and the data (shown as code comments)

# FILENAME: 02c\_commonFcns

# Common Functions/Methods

Some `pandas` methods are a called on the pandas module itself (e.g. `pd.merge`).

# FILENAME: 02e\_eda\_golden

What's below is some hard earned wisdom. Please accept this into your heart and go forth such that data never harms you as it has so many before you.

You can refine this as you go - you'll see more and more tips how to improve your EDA as we go.

{admonition} A thought not quite profound enough to be wisdom :class: tip Data cleaning, exploration, and analysis exist in a never ending feedback loop.

I do all of suggested steps here.

# FILENAME: 02f\_chains

# Using "method chains" to create more readable code

This page is optional but suggested. It shows you *way* to write code. But this is useful because (A) it's often easier to write and read and (B) I will use it in class often.

The material chains on this page is within the "v1.+" tabs below. ```

I didn't find an off the shelf data set to run our seminal analysis in [Chapter 2](../02/10_Golden_2), but I found another option.

In this example, we use a <data.world>'s function to download their data.

# FILENAME: 02g\_commontasks

**So lambda functions let us refer to an unnamed dataframe objects!**

It turns out that lambda functions are very useful in python programming, and not just within pandas. For example, some functions take functions as inputs, like [csnap()](#printing-inside-of-chains), `map()`, and `filter()`, and lambda functions lets us give them custom functions quickly.

`{admonition} The syntax of .pipe()python df.pipe(<'outside function'>, <'if the first parameter of the outside function isnt the df, ' 'the name of the parameter that is expecting the dataframe'>, <'any other parameters youd give the outside function'>

### 

* 1. Temp:Add style.format to the end of your table command. E.g.: df.describe().style.format("{:.2f}")

# FILENAME: 04a-dataviz

# Data Visualization

Data viz can be extremely powerful. There is an enormous amount of scholarship about how to create good visuals, and entire classes within our school about visualization itself.

# FILENAME: 04b-whyplot

A common data science work flow:

Notice: We're in an loop now. **Point being:**

* Remember, our goal is to understand new data, so that we can then use that data to learn something. But since you just got the data, asking good questions is hard - you don't know what's in the data in terms of problems or insights.

1. After you ask a question and answer it with a plot, ask a new question that follows-up on what you just learned.

Anscombe's quartet is four datasets with two variables which have identical means and standard deviation.

# FILENAME: 04c-makeplot

1. **Really** compare your code with the syntax in the documentation. **Understanding what each parameters does and needs is essential.**
2. Triple check for typos, unclosed parentheses and the like
3. [What chart should I use (with sns examples)](https://www.data-to-viz.com) and more help on [how can I make it](https://python-graph-gallery.com)
4. The seaborn tutorial page is excellent: <https://seaborn.pydata.org/tutorial.html>

## Syntax tips

With seaborn, I usually use this syntax that looks something like for graphing. (Delete the "<" and ">" and replace the inside with what you need.) Obviously, you'll see many examples in this chapter that deviate from this. Usually this is because you don't need to explicitly declare "data", or because "x" is just assumed as all variables in the dataset.

* + Q: Which? A: Which ever is easiest! panda's plotting functions are simple and good for early stage and some simple graphics (bar, "barh", scatter, and density), but seaborn has many more built in options, has simpler syntax, and is easier to use, IMO.

## "I swear the syntax is correct!"

After syntax errors, \*\*most graphing pain comes from insufficient data wrangling.\*\* Most plotting functions have assumptions about how the data is shaped. Data might be unwieldy but we can control it:

# FILENAME: 04d-whichplot

## Common plot functions

Below, if I call something like `df['variable'].<someplottype>` that means we are using `pandas` built in plotting methods. Else, we call `sns` to use `seaborn`.

The countplot/bar graph counts frequency of values (# of times that value exists) within a variable, and is best when there are fewer possible values or when the variable is categorical instead of numerical (e.g. the color of a car).

You will come across times where you think the relationship between and might on a third variable, , or maybe even a fourth variable . For example, age and income are related, but the relationship is different for college educated women than it is for high-school only men.

**Facets** allow you to present more info on a graph by designing a plot for a subset of the data, and quickly repeating it for other parts.

* + the categorical boxplot below does this for each sub group

1. The volume of apples picked at an orchard based on the type of apple (Granny Smith, Fuji, etcetera).

# FILENAME: 04e-visualEDA

Four functions come in handy as a starting point, and you should look at their documentation and the example galleries: [sns.displot](https://seaborn.pydata.org/generated/seaborn.displot.html), [sns.boxplot](https://seaborn.pydata.org/generated/seaborn.boxplot.html), [sns.catplot](https://seaborn.pydata.org/generated/seaborn.catplot.html), and the built in pandas plot function [df[<columnName>].plot()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.plot.html)

2. Better syntax help: Go the official seaborn page for your function and look at the examples to figure out what argument creates the change you want.

* Note 3: I like horizontal bars, because if you replace the industry code with the industry name, it's easier to read

To get a quick sense of relationships, I like to use I like to use pairplot and heatmap to get a quick since of relationships.

### 

# FILENAME: 04f-betterplots

- \*\*labels > legends! (so readers eyes don't have to dart back and forth)\*\*

## Transforming bad figures to good ones

1. Which series are the high leverage industries? Which are low?

Q3 It's a easy way to show info about additional variables of interest to a figure. {dropdown}

# FILENAME: 05a-otherskills

(Scraping and strings needs a whole new section for the textbook...)

# FILENAME: 05c\_missingdata

## Dealing with Missing Data

With firm level data that investment analysts deal with, the most common approach to missing data is to keep all valid observations (don't drop anything), and for each test you run, use all observations that have no missing values for all the variables in a given test. In the slides above, this is called "Complete-Case Analysis". - "Complete-Case Analysis" works well, as long as the fact that the variable is missing doesn't indicate a systematic difference between observations that are missing and those that aren't. For example, [I have research that characterizes innovation based on patent text](https://bowen.finance/bfh\_data/), which gives researchers a powerful tool to examine the impacts of firm innovation. However, this dataset will lead to missing values for any firm without patents. And firms without patents are systematically differant than firms with patents. - It is less common corporate finance to impute missing values. However, "deductive imputation" is common when the cost of doing so isn't high, like the height example above. - Interpolation is done in asset pricing when it's necessary to estimate the pricing of options or other derivatives that aren't actually traded. 1. `df.isnull().sum()` will report missing values by variable. 4. In general, when you can confidently deduce a value (my height this year is the same as last year because I'm a fully grown adult, mostly), go ahead. That is valid data. 2. With new datasets, look out for "missing values" that aren't missing. Some datasets use a certain number to indicate missing data (i.e. -99). Convert these to NaNs with `replace`.

### Pandas functions you might use to fill missing values

* fillna - any value (strings included) you want, back fill, forward, fill, and more

# FILENAME: 05d\_outliers

# Dealing with Outliers

Outliers can indicate a problem with your dataset (e.g. data errors to fix or encoding choices to understand[^negativeone]). And even if they are correct data, they might skew your analysis so that it's less useful! or example, [in the first plot on the "Role of Viz" page](04b-whyplot.html#summary-statistics-don-t-show-relationships), and outliers in dataset III and IV will cause your analysis ("What is the slope of the line describing how X and Y are related?") to give an answer that doesn't reflect a layman's sense of the "typical" relationship in those datasets.

[^negativeone]: In some datasets, a missing value might be represented by -1 or -99. If you don't know that and start running analysis on the "numbers", your analysis will be seriously wrong!

**The second consideration is whether to winsorize over the whole dataset or within subgroups?**

The latter is most commonly useful when the distribution of a variable changes over time. For example, the average firm does much more R&D now than in 2000. So the 99th percentile of R&D in 2020 and 2000 are very different.

If you winsorize R&D over both years at once, you'll chop off the lower values in 2000 and the upper values in year 2020. Perhaps it makes more sense to winsorize each year separately!

**First let's look at the min and max of each variable:**

When we winsorize by a group (here, the grouping is each year), the min no longer need equal the percentile threshold. Also notice that the 1st and 99th percentile are a little different. This is ok when we winsorize by subgroups.

# FILENAME: 00\_World\_Wide\_Data

# FILENAME: 01a\_openingAndParsing

You can right click on a page and "view source" to see the underlying HTML, XML, or JSON.

* Right click and view the source. See the structure? That's HTML!

### 

### 

# FILENAME: 01b\_spiders

## One API hit is cool, but do you know whats really cool?

In many web scraping projects, a lot of data needs to get scraped, over thousands (or millions) (or billions) of pages. It's unlikely that you can do this all in one session. (What if your WiFi disconnects, or Windows decides to do an update, or the webpage freezes you out for a period of time?)

It's important to save them in an organized way. There is no "one way", and the directory/storage scheme I choose depends on the job. The main thing is that you probably want two abilities ater the download:

How you achieve these is somewhat up to you but you basically have two choices (and these can work in tandom):

To find the 2008 10-K, you'd open up a master list of documents which contains variables with enough info to assemble the path to each file, and info about each file. Then you can query("form='10-K' & fyear=2008"), assemble the filename, and run your code.

Again, Greg Reda has a nice [walkthrough](https://nbviewer.jupyter.org/github/nealcaren/ScrapingData/blob/master/Notebooks/Bonus_Downloading.ipynb) discussing building a robust code to download a list, and incorporates many of the elements in code we've talked about.

# FILENAME: 01\_Intro\_to\_scraping

Plug and play APIs let you interact with a website without specifying the exact API requests to send to the server.

* The pandas\_datareader plug in for Yahoo stock prices is one version of this.

1. If the data is in the HTML code, you can scrape it using the approaches we will discuss. You can look at the HTML code for any webpage by right clicking and then selecting "View Page Source" (or similar, depends on the browser). After opening the HTML code, CTRL+F to look for some of the data. If the data is in the page, you can scrape it various ways, which we will cover.
2. Web pages are inconsistent - There's sometimes some manual clean up that has to happen even after you've gotten your data.

Now to contradict myself: Some of the packages above can't do things others can, or do them much slower, or the code is hard to write, read, and debug. Sometimes, you're holding a hammer but you need a screwdriver. What I'm saying is, if another package can easily do the job, use it. (Just realize that learning a new package comes with a fixed cost, so be sure you need that screwdriver before grabbing it.)

# FILENAME: 02b\_regex

3. Google+stackoverflow. If someone has done something similar, and found a solution, great!

Your eyeballs can easily do that, but once the job involves enough enough numbers, it makes sense to let your computer do it for you.

# FILENAME: 02c\_developing\_a\_regex

# Developing a regex

1. Think of the PATTERN you want to capture in general terms. "I want three letter words."

# FILENAME: 02d\_RegexApplication

1. [Clicking on the filename in the above link, then the "Raw" button will get you to this page](https://raw.githubusercontent.com/LeDataSciFi/ledatascifi-2023/main/community_codebook/near_regex.py). If you right click on the page and then select "Save Page As" (or similar, depending on browser), you can put it in the folder where it is needed.

At the bottom of the near\_regex.py file, I included examples. You don't need to keep these and if you do, it will cause your code to print stuff. This will look silly and be confusing. So when you copy near\_regex.py into a folder to use it, delete all the examples and stuff below the "return" line in the function.

## Demo

Let me start by showing you some examples. After these example, load the function and read the help documentation in it, and then we can do some practice.

If your document has line breaks, like a newline symbol ("\n") or return symbol ("\r"), which create paragraph breaks in a document, this this function \_won't\_ search both sides of it. If a document splits paragraphs like this, great: our function will do within-paragraph searching, \*\*as long as our cleaning process doesn't delete those symbols.\*\*

The cases\_matter parameter is pretty simple. If true, then it only reports a match if the document's case exactly match your search terms.

## 

Starting from "hey", it finds "jimmy" right away, but it doesn't stop. (That would be a "lazy" regex.) It keeps looking, since it can look up to three words away. It takes the largest mechanical match possible, so when it finds james, it deems "hey jimmy hi james" a match, and because of that, "hi james" is already used and can't be a match. So it reports 1 match, not 2. ```

1. changing them gives you BAD results: Either way too many non-results are included, or your search to returns too few results because a single hit encompasses several hits that should be separately counted
   * Going from 1 to 2 hits is probably more informative that from 99 hits to 100

# FILENAME: 00\_intro

# Data Science for Finance

The [Objectives page](../about/objectives) says that the rest of the class is about

* understanding the how "data analysis/ML/<<buzz word #51>>" fit into the bigger picture of producing and using our domain knowledge of from finance. to quote Prof Gunther: data < info < knowledge < wisdom

# FILENAME: 01a\_MLgonewrong

* + [Google Flu Trends](https://gking.harvard.edu/files/gking/files/0314policyforumff.pdf) consistently over predicted flu prevalence

# FILENAME: 01b\_model\_process

Modeling is NOT typing import sklearn and plowing into the data like Leroy Jenkins. Throughout the semester, you've seen that thinking about how to code a solution to a homework problem is usually easiest when you step back, and think about the challenge you're facing, and writing down some pseudocode.

Modeling (applying data science techniques to our finance problems) is that on steroids. Instead of needing to figure out how to writing code to solve a discrete problem (reshape the data to our analysis level, add a variable to our dataset), we need *to figure out what to do.*

## Design your tests

TBD - TODO. Map model to the empirical test you'll actually implement. Evaluate it for weaknesses.

## Estimate your model

We will talk in depth about a few models in class, but generally, these three steps always apply:

# FILENAME: 01c\_teams

2. Co-work on a task simultaneously: Persons A and B do a zoom share meeting and co code on person A's computer via screen share + remote control. Advantage: More brainpower, and good when the whole group is stuck.   
3. Separately attack the same task, then combine your answers: Persons A and B separately do part 1, and compare answers/approach, and put together a finalized solution to part 1. This creates duplicate and discarded work product, but will generate more ideas on getting to the solution.

If you forget to fetch/pull before you start (and someone made a change on the github repo since you last synced), or if someone is working at the same time (and pushes a change to the github repo that conflicts with a change you made), you are likely to receive a "Merge Conflict" notification from GH Desktop.

3. It's better to over communicate than under communicate, especially in our virtual world

## Branching Demo

Above, I mentioned that one way that multiple people can work in the same repo at the same time is by "branching". Rather than explaining it, let's let one of our TAs do a walk through on how this can work!

# FILENAME: 01d\_sharingBigFiles

1. Turn on sharing for the file by right clicking it and selecting get shareable link, but you don't need to give edit access.

# FILENAME: 01\_bigpicture

# The promise of ML

Machine learning can generate solutions to problems at scales that are cost prohibitive otherwise. For example, the earliest (and ongoing) waves of ML in the finance space includes

1. Investment choices - stocks, real estates (where to put factories, banks, etc)

# FILENAME: 02a\_basics

* **Regression** is the single most important tool at the econometrician's disposal
* **Regression analysis** is concerned with the description and evaluation of the relationship between a variable typically called the dependent variable, and one or more other variables, typically called the independent or explanatory variables.

# FILENAME: 02b\_mechanics

These next few pages use a classic dataset called "diamonds" to introduce the regression methods. In lectures, we will use finance oriented data.

## 

1. You can set up the model (the equation) more naturally. Simply tell it the name of your dataframe and then you can regress height on weight () by writing weight ~ height'. - You don't need to include the constantaor the coefficientb`
2. You can add more variables to the regression just as easily. To estimate , write this inside the function: y ~ X + Z

## 

Suppose that an ideal cut diamond doesn't just add a fixed dollar value to the diamond. Perhaps it also changes the value of having a larger diamond. You might say that

* A high quality cut is even more valuable for a larger diamond than it is for a small diamond. ("A great cut makes a diamond sparkle, but it's hard to see sparkle on a tiny diamond no matter what.")
* In other words, the effect of carats depends on the cut and visa versa

Thus: The return on carats is different (and higher) for better cut diamonds!

# FILENAME: 02d\_interpretingCoefs

Regressions of y on $N$ different variables takes the form

Usually, we encode one value as zero, and the other as one before we include it in the regression. This makes interpretation simple, as it just follows from the previous table, since a "1 unit change in X" simply means changing from the baseline group encoded as zero to the other group encoded as one.

The interpretation of $\beta\_{oneLevelOfACategoricalVariable}$ is the same as a binary variable (use the table above depending on if the model is using y or $\log y$), \*\*except that the it is capturing the jump from the "omitted group" (X=0 above) to whichever level that particular $\beta$ captures.\*\*

Suppose we model the price of a diamond as function of its cut and nothing else. [This is close to what we did previously](02b_mechanics.html#including-categorical-variables).

* + A 1% increase in size is associate with a 1.53% higher price **for non ideal diamonds**
  + A 1% increase in size is associate with a 1.71% higher price **for ideal diamonds**
  + *Economically, you might say that the value of a larger ring is even more valuable for better cut diamonds*
  + Mathematically: 1 carat diamonds that are ideal are 33% more expensive than non-ideal diamonds, but 2 carat ideal diamonds are 45% more expensive than non-deal diamonds
* This regression does NOT imply that taller players average fewer tackles. In the real world, what we call "independent variables" in the regression often change together. Taller players are likely to be heavier. So if a 1 inch increase typically comes with a weight gain, the *total impact* of height on tackles (i.e. *not* holding all other factors constant) will include the estimated impact of weight, which $\hat{\beta\_2} < 0 $

Suppose you estimate , and you want to focus on to capture how investments translate to profits. You've added some control variables X, but you're still worried that this regression will get the relationship wrong, because different industries have different profit margins for reasons that have nothing to do with investment levels.

In other words, you want to "control for industry". So you estimate , by including the firm's industry as a categorical control.

When you add industry to a regression as a categorical variable, it is called including "industry \*\*fixed effects\*\*".

This should go a decent way to solving your worry above.

Similarly, you might be worried that some years are at high points in the business cycle, and these years have concurrently high investment and profits simply because of the business cycle. This would cause to be positive *even if investment does not lead to profits.*

So you might estimate . This is often referred to as "year fixed effects", and it means that your estimate of removes the impact of years, and presumably, the business cycle.

Remember our weird result earlier? That better cut diamonds had lower average prices?

The answer to that puzzle is pretty simple: Better cut diamonds tend to be smaller, and size is the most important aspect of diamond price. You can click to show the model results below.

By adding carat size back to our model, we get the sensible result, that going from an Ideal cut to a Fair cut diamond (a big downgrade), as long as we compare similar sized diamonds ("control for diamond size"), is associated with a 31% decrease in price.

To which, I'd say that how "big" a coefficient is depends on the variable!

Also, notice that the new coefficent (0.98) is about 58% of the original coefficient (1.69).

# FILENAME: 02e\_statisticalSig

1. Practically, if the p-value is above 0.05, most researchers consider completely disregard the coefficient (and ignore the sign and the value). Because you can't say the coefficient is statistically distinguishable from zero, they basically the interpret the coefficient for that variable as being zero. Meaning: Ignore the sign, ignore the value, assume the coefficient is zero.

# FILENAME: 02f\_warnings

And suddenly, you see a article saying that 10 cups of coffee, and 2 bars of chocolate, and 3 glasses of wine a day **leads to** longer lives, or that breastfeeding for up to two years **causes** better outcomes.[^babies]

The "default" interpretation you should have of a regression is that you're seeing a correlation, not that X causes Y. You need to rule out some alternative possibilities first.

The most common methods that *can* to establish causality are:

Until you learn about the advance techniques above, focus on humility as you report regressions:

1. Our standard fill in the blank [interpretation sentence](02d_interpretingCoefs) calls the relationship an "association" and avoids the banned words below.
2. Emphasize in discussion of findings what you found (a statistical association) and didn't ("We acknowledge that this finding isn't causal." "One limitation of our study is that...")

However, it doesn't take ill intent: You, or friends, or strangers might find a false result and trumpet it due to **motivated reasoning, cognitive dissonance, or confirmation bias.** Analysis in many domains are fraught with these temptations; [the game above](https://fivethirtyeight.com/features/science-isnt-broken/#part1) has a political valence.

# FILENAME: 02\_reg

# Regression

We start our machine learning applications with regression for a few simple reasons:

* Regression is fundamental method for estimating the relationship between a variable ("y") that condition on many ("X") variables.
* *Note: The focus in this section is on RELATIONSHIP paradigm*
* Many issues that confront researchers have well understood solutions when regression is the model being used.
  + interaction terms between two X variables changes interpretation

# FILENAME: 03a\_ML\_obj\_and\_tradeoff

* Def: Is errors stemming from the model's assumptions in how it predicts the outcome variable. (It is the opposite of model accuracy.)

## 

# FILENAME: 03c\_ModelEval

### 

So we need a way to estimate the test error. The way we do that is by creating a **holdout sample** (step #2 in the [machine learning workflow](03_ML)) to test the model at step #6, after our model development process is completed.

**If you use the holdout during the iterative training/evaluation process (steps 3-5), it stops being a a holdout sample and effectively becomes part of the training set.**

After we create the holdout sample in step \#2, we enter the grey box where all the ML magic happens, and pre-process the data (step \#3).  
  
So, we are at step \#3. We have the holdout sample isolated on the side, and will develop our model on the training sample.   
  
If we fit the model on the training sample, and then examine its performance against the same sample, the error will be misleadingly low. (Duh! We fit the model on it!)   
  
- We estimate make predictions in the validation sample using our fit model, and measure the "accuracy" of the prediction.

# FILENAME: 03f\_leakage

# Data Leakage - Illustration

Data leakage is the one of the cardinal sins of ML.

# FILENAME: 03f\_leakage2

[The next pages](03c_ModelEval) of the book discusses the *main* (but not only) method to avoid data leakage and the [next section](04a_SKLearn) of the book will explain how *with code*, but for now, let's just state the following warning:

I know I already said that, and repetition is usually bad writing, but it must be said again. And again.

Is it too good to be true? One way to tell is to establish a baseline: Find best-in-class performance metrics for your problem. If you see dramatically better results, data leakage is a good candidate for why.

# FILENAME: 03\_ML

This chapter will **discuss** this modeling process backwards, under a framework mostly focused on "supervised" prediction problems.

Working backwards will help keep our focus on **why** we are doing certain steps and the big picture, rather than getting mired in the weeds, which can lead to [poor or disastrous analysis](01a_MLgonewrong).

# FILENAME: 04d\_crossval

Code wise, it *can* be this simple:

* **Grouped, time independent data**: If time can be ignored, and you have multiple observations for each unit ("groups"), your goal is likely to find out if a model trained on a particular set of groups generalizes well to the unseen groups:
  + Usually: If you have have a cross-sectional panel (multiple observations for each time), build a custom splitter to generate your folds.

The code below implements it, and can be used with appropriate adjustments for your setting.

## Code for rolling OOS testing (v1)

The code above works. It lacks some features, and until a classmate codes it up int a nice function, it requires copy-pasting all of the code I wrote, and adjusting it.

# FILENAME: 04e1\_preprocessing

1. Or model the missing values explicitly (e.g. in a regression, as an incremental intercept but with no impact on the slope).

**You should focus on the whys and hows of dealing with missing data rather than mechanics. (You can look up mechanics later.)** You should have some livecoding from the prior lecture showing imputation in pandas.

# FILENAME: 04e\_pipelines

Pipelines are just series of steps you perform on data in sklearn. (The sklearn [guide to them is here.](https://scikit-learn.org/stable/modules/compose.html))

I use [ColumnTransformer](https://scikit-learn.org/stable/auto_examples/compose/plot_column_transformer_mixed_types.html#sphx-glr-auto-examples-compose-plot-column-transformer-mixed-types-py) to assemble my preprocessing portion of my full pipeline, and it allows me to process different variables differently.

* When modelling, you should spend time interrogating model predictions, plotting and printing. Does the model struggle predicting certain observations? Does it excel at some?

# FILENAME: 04f\_optimizing\_a\_model

Lets load the data [we've been using.](04e_pipelines) (The code is here but hidden.)

Let's set up our model. For simplicity, this model just imputes missing values of (all) numerical variables and then OneHotEncodes the state variable. Here is

Now, look how many parameters this tiny three-step model (impute, encode, ridge regression) has!

The basic idea is to repeatedly run the pipeline through a cross-validation while changing the hyperparameters to improve your CV scores.

# FILENAME: 04h\_putting\_together

* Feature selection/reduction: Which X variables to include. Too many variables will lead to overfitting.
  + Common options: SelectFromModel, LassoCV, RFECV
* Gradient Boosting, discussed [here](https://www.kaggle.com/kashnitsky/topic-10-gradient-boosting) and [here](https://www.youtube.com/watch?v=yrTW5YTmFjw), and ensemble + stacked predictors
  + xgboost and lightGBM are the go to implementations, and HistGradientBoostingRegressor is the analogue in sk-learn
  + If you just sk-learn for gradient boosting, look for the "Hist" in the function name! (Newer, much faster.)

# FILENAME: 05a\_compounding

## 

Assume you have simple periodic (little ) returns in a variable called "ret" in df for various assets and you want to compound them to a longer period of time (e.g. monthly).

# FILENAME: 05a\_expanding

We need a dataset with firm, date, and the faily return. Let's build it:

The code in the next block is explained more thoroughly in `handouts/factor\_loading\_simple` in the textbook repo, because that file prints the status of the data throughout. Looking at this might help.

Which you choose is up to you, but in my testing the cumprod approach is 2.5x faster.

# FILENAME: 05a\_finapps

This chapter will work through a few common steps and analysis.

# FILENAME: 05a\_rolling

Let's create a variable containing the cumulative return over the last week. Let's stipulate that if we have less than fives days of returns for a firm at any point, we just use what we have.

# FILENAME: 05b\_capm

The psuedo code is relatively simple:

1. Load the market return premium and the risk free rate (the "factors" in CAPM).

2. This page uses Yahoo finance for stock returns.

The code in the next block is explained more thoroughly in `handouts/factor\_loading\_simple` in the textbook repo, because that file prints the status of the data throughout. Looking at this might help.

# FILENAME: 05c\_factorloadings

# Estimating the loadings on 3 factor model

Again, the pseudo code is simple:

## 

# FILENAME: 05c\_finpacks

1. View it as having an infinite supply of interns with an 8th grade IQ. Meaning:
   * Low cost way to brute force some tasks (brainstorm, rough drafts, outline, editing, starting code)
   * Responses often vague and lack specifics
2. Do not assume its output is correct or the truth
   * Meaning: It will confidently lie and make up things (so called "hallucinations" and "bullshiting")
   * Start up time on (coding) projects will decline drastically
   * Shift in emphasis to idea creation, editorial choices, truth checking
   * Typed essays decline in favor of oral evaluations, handwritten essays

# FILENAME: 06b\_nextsteps

There is no one path forward, but I think people learn skills best when the skill-building is the byproduct of doing something fun/exciting that scratches your curiosity.

Your projects don't have to be finance-related. (My initial python projects weren't!) Maybe it's about your love of some game (e.g. basketball), or tv/movies/books, or sports. Just pick something that sounds fun.

* A fantasy draft wizard, powered by data scraped from multiple sites and organized by ML techniques. ([This repo, written in R, can be inspiration](https://github.com/jsoslow2/Fantasy-Football-Models).)
* [The website resources](../about/resources.html#resources-tutorials-and-data) has lots of good stuff to explore

Not only are dashboards cool as hell, they are also good ways to show off your skills (e.g. put them on your resume) because they can be really useful for real-world decision makers in business and government settings.

[How do you think this amazing page was built?](https://www.perthirtysix.com/essay/nba-player-scoring-analysis) Or the interactive visualizations [538, NYT, WaPo, WSJ and others frequently publish](https://fivethirtyeight.com/features/science-isnt-broken/)? And look at all [these freaking awesome examples!](https://dash.gallery/Portal/)

The main hiccup: GitHub Pages can create a website out of a Jupyter notebook, but it's just a static image of the file after you ran the code.

There are many packages that are designed to produce full featured dashboards, apps, and widgets. My current rough pecking order:

# FILENAME: 06\_thefuture

# The Future

I hope you've enjoyed the class and learned a bunch! This class is only an introduction to what can be done; clearly, there are galaxies of material on data science broadly and its possible applications to solve important (finance related) problems. But you have the starting toolkit to explore that galaxy of possibility now.

- "Notes" for each chapter is a concise doc contain \_some\_ of what is in the video.   
 -

# FILENAME: gradeoverview

What I love about this is that it puts you on the precipice of truly large scale analysis.

Student groups write a proposal outlining a question the group is interested in and their plan to answer it. Over the course of the last month of class, we will work to break your question down into manageable subproblems and solve them one-by-one to conduct their analysis.

# FILENAME: structure\_and\_policies

* In addition, you will have as resources this website, your peers via a github discussion board, the whole of the internet, and, naturally, office hours. **Which is to say: You have my utmost support, and you will succeed if you put in the time!**

Seeing students in class is fun for me, and your attendance will help you (and your classmates!) learn the content in this course.

# FILENAME: tips

# FILENAME: asgn05\_measurerisk

* *What have I done to foresake you?*
* I define a regex pattern that looks for the word "detroit" near "loser". We talked about how to do this in class: Use NEAR\_regex. Let's call that pattern loser\_regex\_pattern.
* hits = len(re.findall(loser\_regex\_pattern,<clean\_10\_text>)) will count the number of times the document discusses losing and save it to hits.
* Save hits inside of the variable losers for that document (put it in the correct row!)
* **Manually check it by opening the 10-K on the browser - do your functions give you the same values you'd create if you did it by hand?**
* **So how should you define the regex pattern?**
  + You could just count how many times the risk word is used. But *be careful: You might need to make sure the topic is being discussed in the context (near) of risk. E.g. "Patent" is often talked about without invoking risks.*
  + So, use NEAR\_regex(). You give it a list like [string1, string2] and it will design a regex that looks for the strings being within some number of words of each other.
  + *Be careful: Sometimes the risk is discussed using a synonym or a partial word. E.g. the string "patents" is not the same to a computer as "patent".*
  + Advanced: What do you think [(drake|billie eilish), (grammy|grammies|oscar|oscars)] will do if used in the NEAR\_regex function? (Go ahead and try it out!)

1. Repeat the steps in step 3 above to define that risk a second way, but change the words and parameters you used.
2. Repeat the steps in step 3 above to define that risk a third way, but change the words and parameters you used.
3. Repeat the steps in step 3 above to define a second type of risk.
   * Open some 10-Ks manually and read them to verify if your guess for how to check actually results in hits.
   * If it doens't work like you think (misses obvious discussions of the risk, or finds non-discussions), tweak it.
4. Repeat the steps in #3 above to define a third type of risk.
   * Open some 10-Ks manually and read them to verify if your guess for how to check actually results in hits.
   * If it doens't work like you think (misses obvious discussions of the risk, or finds non-discussions), tweak it.
5. Now, loop over all the firms and for each, measure the risks in the corresponding 10-K. *Warning: Looping over rows in a pandas dataframe is a little different than normal! Look up how to do it!*

* **IMPORTANT! .describe() those five variables after you're done!**
  + You should have observations for most/all firms for your new measures. **Fix it - do not proceed!**
  + If any of your variables are always 0, it's meaningless. **Change it - do not proceed!**
  + If any of your variables are always really high, consider if your search thinks too many things are that risk. Searching for "risk" for example is too vague.

## Some more pointers on how to measure risk

* Risks you could look to measure include, but are not limited to: antitrust; litigation - e.g. patent, consumer, class action; real estate; inflation; commodity; supply chain; natural disasters; weather; employees (fraud, compensation, departure); changes in tax policy; currency rates; regulatory approval; reputation; refinancing;
* Prof. Kathleen Hanley [has a recent paper](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2792943) on risks. It focuses on financial firms, which isn't our sample, but nevertheless, it contains a long list of risks in Table 5 you might find interesting.

This assignment is about covid. What factors of firms do you think might make a firm more or less resilient to operating in a pandemic?

# FILENAME: howto\_do

# Homework guidelines

## Starting homework

1. **Click on the assignment link in the classmate team on GitHub.** When you follow the link and accept the assignment, GitHub will create a repo for you. The repo will be a private repo (only you, me, and the TA can see it) under the LeDataSciFi organization on GitHub.
2. The repo will contain instructions and a rubric, and possibly starter code/data.
3. Clone the repo to your computer using GitHub desktop and follow the instructions.

## Working on assignments / projects / taking notes

[Use the Github workflow!](../01/03a_githubworkflow)

\*\*Fetch early, commit frequently, push often!\*\*  
  
This habit will help you avoid disasters, so that you get the positive features of Github without the headaches.

## Submitting assignments

Submission is easy: Whatever is in your assignment repo (the online version, not your computer!!!) at the deadline is what will be graded.

Again: Whatever is in your assignment repo (the online version, not your computer!!!) at the deadline is what will be graded.

Shortly after the deadline, GitHub will automatically stop you from pushing edits to the repo! So commit and push your changes often, and do not wait until minutes before the deadline.

Your graders will download your repo on their computer, try to run your code, and evaluate it for accuracy and quality, based on the rubric (each assignment has a different rubric).

## Tips for better grades (+ workproduct + repos)

```{dropdown} **TIP #1:** Check out the rubric for the assignment Each assignment has different grading criterion, which you'll see in the rubric within each assignment's repo.

Generically and briefly, I'll say the important themes are:

* reproducibility (Your peers should be able to download the folder to their computer and execute the main analysis file and receive back the same results you generated)
* organization of the repo (can an outsider discern what is going on?)
* the README file, which should describe the repo to readers
* the main analysis file. More on this in the next tip...

```{dropdown} \*\*TIP #2:\*\* Before you push what you think are your final changes to the master repo...  
1. Delete all temporary and output files you generated, restart the kernal (and clear output),   
2. rerun the analysis/code (as though you just started from scratch),  
3. and save your notebook file when it's done running.   
  
This will clean up your folder for viewers (yay, professional work product!) and is a first pass at finding out if your code "works from scratch".  
  
Did it work? If not, then the code is not reproducible. Probably, you referenced a temporary file you created outside of the flow of the code, or accidentally put an input file in the temporary file folder. Whoops! Fix it!

``{dropdown} \*\*TIP #3:\*\* After you push what you think are your final changes to the master repo... 1. Go to the repo \_\*\*online\*\*\_. This is what your reviewers will see. The process of writing code often creates little files (like\_checkpoints`) that you don't want in the online repo, but that you can't or don't want to delete locally. The usual solution is to add them to gitignore and purge them from the master repo. (This is easy, and I'll describe how to do it elsewhere.)

1. **A 100% solution**: Download it to your computer (click the green button, then download the zip file) **to a different spot** (say a temporary folder on your desktop), and run the analysis again, there.

Did it work? If not, then the code is not reproducible outside the directory you built it in.

Why did it fail? Probably, the code included absolute path references like "C:\Users\DonBowen\Documents\project1\code\extra\_function.py". These are bad, because they only work on my computer. (You probably won't have "DonBowen" as a folder anywhere on your computer, I hope!)

Use relative path references instead. For example, if main\_analysis.py uses extra\_function.py, and main\_analysis.py is in the "project1" folder, then "code/extra\_function.py" will find the function on any computer.

```{dropdown} \*\*TIP #4:\*\* Make it easy for others to see the source code that executes the analysis as well as the report.   
The biggies:  
1. The README in the main directory is what they will see first when they open it. Make it professional and helpful! GitHub can format "Markdown" nicely, with headers, links, media, and more and will show visitors this.   
 - Create annotated links to documents graders need to access. For example, point them to your main Jupyter file, which GitHub will also render so that visitors see the output on the website without needing to run code!  
 - [GitHub](https://guides.github.com/features/wikis/#Formatting-a-readme) suggests including the project name, a clear and short description (what it does and why it is important), installation tips (list packages they might need to install), and usage instructions.  
2. Make sure your final analysis code shows outputs \_when viewed online\_! See Tip #2 above. This will help them grade faster (and faster graders are nicer graders).

{dropdown} \*\*TIP #5:\*\* Make it easy for others to run your code. - At the top of the code, load all packages you need - Under that, list parameters a user would set (you obviously choose these during your analysis, but it's nice to see them quickly in one place) - Under that, import external files. This makes it easy for someone to see which data files are required and edit the paths if necessary. Even though I said absolute path references are bad above, there are situations where you might not keep data in the repo itself. (For example: Huge data files or sensitive data subject to privacy issues.) When this is the case and you have to use absolute path references, you want those at the top of your code so people see them! - Again, see tip #3 above.

{dropdown} \*\*TIP #6:\*\* Make your work product (especially tables and figures) pretty. - Use good design concepts that we discuss early in the semester for figures and tables. - I google "markdown cheat sheet" often (sometimes with "github" or "jupyter" added to it depending on what I'm writing). [This is a good start cheat sheet](https://www.markdownguide.org/cheat-sheet/) and there is so much good stuff online.

# FILENAME: howto\_review

# How to do peer review

After assignments are submitted,

* You will be added to two classmate's assignment repos. Go to [github.com](www.github.com). On the left side of the page, under repositories, you'll see that you have access to assignments for two peers.
* Inside the assignment repos, there will be an answer key along with instructions for how you submit your reviews.

The peer reviews you do will be graded based on:

* Did you actually run the code to verify accuracy? (We do random audits to see if the code works.)
* Your review should be accurate and honest
* Did you give helpful, constructive, and nice feedback? (Looking for professionalism + community!)

# FILENAME: mid\_proj

# Midterm aka Assignment 5 - Our first real data science project

{admonition} Tips :class: tip 1. Read all instructions before starting. 1. Start early. Work on the components of the project in parallel with related class discussions. 1. RECHECK THESE INSTRUCTIONS BEFORE SUBMITTING

Per the [syllabus](../about/gradeoverview), this project is 10% if your overall grade, which is about 2x the weight of a typical assignment. It will probably take 2-3x the time of a typical assignment.  
  
\*\*Really fun news:\*\* This is a end-to-end data science project! You will be downloading a lot of files, parsing/exploring/cleaning those file, and then exploring the data.   
  
BUT: It will take time! If you start the day before it is due, YOU WILL NOT FINISH IT. If you start two days before it is do, you might finish it, but it will not be done well.

## Project Set Up

The nuts and bolts of the set up are:

* Basic question: What "types" of firms were hurt more or less by covid?
* Specific questions: What risk factors were associated with better/worse stock returns around the onset of covid?
  + This is called a "cross-sectional event study"
  + Expected minimum output: Scatterplot (x = some "risk factors", y = returns around March 2020) with regression lines; formatted well
  + Discussion of the economics linking the your risk factors to the returns is expected
  + Pro output: Regression tables, heatmaps, better scatterplots
* New data science technique: Textual analysis. We will estimate "risk factors" from the text of S&P 500 firm's 10-K filings.
  + More on this below
* Data needed:
  + Returns: Stock returns for S&P 500 firms can be pulled from Yahoo
  + Risk factors for each firm will be created from their 10-K filings.

So your main challenge... is to create variables that measure risks for each firm.

## Steps to complete the assignment

```{dropdown} 1. Start the assignment

* As usual, click the link I provide in the discussion board.
* But unlike before, the repo will be essentially empty. This is a start to finish project, so I'm letting you dictate the structure of the files.
* Clone this to your computer.

````{dropdown} 2. Edit \*\*.gitignore\*\*   
  
The `download\_text\_files` file will create a large data structure in a subfolder called `10k\_files/` with all the downloaded 10-K files. There will be several gigs of data in this folder. We don't want to save/push all these files to github!  
  
```{warning}  
So add this directory (`10k\_files/`) to your gitignore before you proceed!

````{dropdown} 3. Create \*\*download\_text\_files\*\*   
  
This file   
1. Should create a subfolder for inputs (`inputs/`). You should probably save the S&P500 list from the wikipedia page there.   
1. Should create another subfolder (`text\_files/`) to hold all the text files you download. Because scraping can generate large amounts of files, I usually put it in a dedicated input folder instead of the generic input folder we just made.  
  
\*\*Tips/recommendations\*\*  
  
1. Try to download just one 10-K at first. When you can successfully do that, try a few more, one at a time. Check the folders on your computer - did they download like you expected? Are the files correct? If yes, continue. If not, you have an error to fix.   
1. The website has really good info on "building a spider." Highly recommend!   
1.   
  
```{tip}  
When you are confident the program works,   
1. Delete your whole `text\_files/` and `input/` subfolders on your computer so you have a "fresh start"   
2. Rerun this from scratch.   
3. Rerun the file AGAIN (but don't delete the files you have). Does the file work after it's already been run, or partially completed it's work? Real spiders have to resume where they left off. You might need to make some conditional tweaks to the file to account for this. You don't want the code to actually re-download the data, but the code should still run without error!  
```

```{dropdown} 4. IMPORTANT: Create **screenshot.png**

It's not polite to upload so much data to GitHub. It takes up space on the server, and your collaborators/peer reviewers will have to download them all when they clone your repo.

That's why you edited the gitignore before doing all those downloads. If you did it correctly and check Github Desktop, you won't see any of the text files!

1. Now that your download\_text\_files is done running, push the repo. Even though your *computer* has a /text\_files/\* folder on it with many files and some hard drive space used, the repo in your browser doesn't show this at all! Good job!
2. **Create screenshot.png. The purpose of this is to upload proof of the files for your reviewers.**

Right click your text\_files folder so it shows the number of files inside of it, and take a screenshot showing this. Save it as screenshot.png inside your repo.

```{dropdown} 5. Download \*\*near\_regex.py\*\* from the community codebook into your repo  
  
This will be used in the next step.

````{dropdown} 6. Create **measure\_risk**

The basic idea is to measure risks by counting the number of times a given risk topic is discussed in the 10-K.

This file (broad steps)

1. Creates an output/ folder
2. Loads the initial dataset of sample firms saved inside of input/.
3. For each firm, load the corresponding 10-K and create (at least) 5 different risk measures, and save those new measurements to each of 5 new variables in that row.
   1. **Pick one risk type, and think of three ways to measure it.** For example, there are many ways you could try to measure "antitrust risk", so come up with 3 different ways to measure it from the text. You can try different terms, different combinations of terms, different limits on how close terms need to be, and more. Comparing these different ways might help you understand how your choices can improve or hurt the value of your measurement.
   2. **Pick a second risk type and create a single measure for it** (you only need to do one measurement on this risk type, but you can do more)
   3. **Pick a third risk type and create a single measure for it** (again, you only need to do one, but you can do more)
   4. Bonus measures - interesting variables you could also measure:
      * The total length of the document (# of words)
      * The # of unique words (similar to total length)
      * The "tone" of the document
4. Downloads 2019 accounting data (**2019 ccm\_cleaned.dta**) from the data folder in the class repo on S&P500 firms (possibly useful in analysis) and adds them to the dataset
5. Save the whole thing to output/sp500\_accting\_plus\_textrisks.csv

[There is a bunch more on this file/step here.](asgn05\_measurerisk)

When you are confident the program works, delete your whole `output/` folder on your computer so you have a "fresh start" and then rerun this from scratch.

````{dropdown} 7. Create \*\*explore\_ugly\*\* to see if your risk factors were associated with higher or lower returns around covid.  
  
Try to figure out how to do the analysis below, downloading and intergrating return measures. Play around in this file. No one will look at it. It's a safe space.  
  
If you find issues with your risk measurements or come up with improvements you think you should make, go back and work on the previous file more.   
  
You can and should use this file to figure out what you want to include in the final report and how you want it to appear.

````{dropdown} 8. Create **analysis\_report**

This is the main portion of your grade. It should be well formatted and clean in terms of text, code, and output. Don't show extraneous print statements. Treat it like a Word document that happens to have some code (but just enough to do the analysis and show outputs). I've included more thoughts in the next dropdown.

First compute the returns for the 3/9-3/13 week. This will give you a dataset with one row per firm, and one number per row (the return for that week). Then merge this into the analysis dataset. Rinse and repeat if you try for the other return measures I describe below.

1. Load output/sp500\_accting\_plus\_textrisks.csv
2. Explain and describe to readers your risk measurements
   * How were they measured? (Mechanical description)
   * Why did you choose them and what do you hope they capture? (Economic reasoning)
   * What are their statistical properties? (Do you have values for most/all firms, they should have variation within them, are they correlated with any accounting measures)
3. **Validation checks and discussion of the risk measurements** **This step (validating the measurement) is very important in production quality analysis!**\*
   * Discuss briefly whether these measurements are likely "valid" in the sense they capture what you hope.
   * Present some evidence they do capture your hopes. There are many ways to do this, and depend on the data you have and the risks you're measuring.
     + You might print out a few examples of matches.
     + One option is to show sentences that will correctly be caught by the search, and correctly not caught. And how easy is it for your search to find a sentence that matches the search but shouldn't.. (Hopefully: not too easy!) How easy is it for your search to miss a sentence that it should match...
     + One option is to output the list of firms that have high scores, or the industries that have high and low scores. Does the output make sense?
4. Describe the *final* sample for your tests, the set of observations where you have all the data you need.
   * This includes summary stats,the number of firms, and other things EDA would turn up
   * Are there any caveats about the sample and/or data? If so, mention them and briefly discuss possible issues they raise with the analysis.
5. Explore the correlation between your risk values and stock returns around key dates for the onset of covid.
   * Stock returns are in the class's data folder ("2019-2020-stock\_rets cleaned.zip")
   * Get the firm's returns for the week of Mar 9 - Mar 13, 2020 (the cumulative return for the week)
   * Bonus: repeat the analysis but use the cumulative returns from Feb 23-Mar 23 as the "collapse period"
   * Bonus: repeat the analysis but use Mar 24 as the "stimmy day" (stimulus was announced) ... how does this change your results, and is it doing so in a predictable way?
   * Bonus: repeat the analysis, but use firm accounting variables: Some of these probably indicate that a firm should be more [resilient to the crisis](https://privpapers.ssrn.com/sol3/papers.cfm?abstract_id=3597838)!
   * Present your findings visually and follow the lessons on effective visualization!
   * You should write brief summaries of your findings.
6. Bonus: Explore the risk-return relationship, but use regressions so that you can control for firm traits and market returns. Does this change your results?
   * Don't worry about printing these regressions out "pretty", just try them if you want!
7. Bonus: Use **alpha** as y, not returns, in your plots and/or regressions. This will likely change the results.
   * Step 1: Separately, for each firm, [estimate the beta and factor loadings](https://ledatascifi.github.io/ledatascifi-2022/content/05/05c_factorloadings.html) of each firm's returns in **2019**. Save that data.
   * Step 2: For firm i on date t, alpha(i,t) = ret(i,t) - beta(of firm i)*mkt\_return(t) - SMB(of firm i)*SMB\_port\_ret(t) - HML(of firm i)\*HML\_port\_ret(t)
     + *SMB\_port\_ret(t) is the return on the SMB portfolio on date t, which you can get from the Fama-French datasets!*
   * Just present the findings if you do this. Don't worry about explaining it - but it might make more sense in a few weeks!

If you want to do any regressions, let me know. I'll give you a few pointers.

````{dropdown} 9. Finalize and polish  
  
Unlike previous assignments, how clean your code and report are will factor into your grade. Additionally, your README file should be nice!  
  
\*\*Edit the readme file - it should be "publication ready"\*\*  
- Make the readme file informative and professional.   
- Inform readers of the order in which files should be run. And warn users that this folder will download X files of X MB or GB.  
- Change the title of it (not the filename, the title at the top)  
- Describe the purpose of this repo (what this repo is analyzing) and the key inputs  
- List any necessary packages (might a reader need to `pip install` anything?) or steps a visitor will need to run to make it work on their computer   
  
\*\*The `analysis\_report` file should be written and formatted like an executive report.\*\*   
- There is no "page expectation" or "page limit". Aim to provide sufficient analysis and explanation, but in a concise and clear way. Bullet points are fine in places, but you should have a few places with paragraph-style discussion, especially where you explain why you chose the specific risks, the way you defined them, and what issues you think they have (which points the way forward on "extensions").   
- In other words: You will be graded on how much this looks like a professional report. Just "dumping" endless printouts is not as valuable as well-tailored tables and figures. High quality and concise reporting is an A1 emphasis. \*\*Here, pretty, smart, and effective tables and visualizations will receive higher grades.\*\*   
- \*\*The teaching team will \_not\_ read your measure\_risk file other than to comment on code style.\*\* So:   
 1. Any details in that file on search terms and descriptive information on your text-based measures should be copied into your analysis file (with appropriate adjustments to suit how a report would be presented).   
 2. Make the measurement code easy to read, because we will grade the code style.

## Cheers!

**Give yourself a big round of applause at this point!**

Your code is probably very flexible and powerful at this point. If you have the appetite + a larger list of EDGAR files to download + a large enough hard drive + and time, then you could download more than 100GB of 10-K filings and run textual analysis across 20+ years of data for all publicly traded firms.

Seriously: You are in the ball park of pulling off any analysis you want that needs to harness the power of these filings. These four studies are variously provocative, great, and (in one case) mine:

* [Check this claim: Identifying changes in 10-K/Q filings can generate a 20% alpha](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1658471)
* [Prof. Hanley measured emerging risks in the financial sector](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2792943)
* [Build a unique list of competitors for each firm (really powerful!)](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1520062)
* [I used 10-K text to identify public rivals of startup firms](https://ssrn.com/abstract=3245839)

# FILENAME: project

# The final project

This project wraps up all the skills in the class so far:

* Ask an interesting question or highlight a problem to be solved
* Get the necessary data to address it, performing EDA as needed to clean it
* Analysis - question dependent
* Communicating your results: notebook + presentation + website

## A note on ambition

The goal of the project is not to simply take a pre-cleaned dataset and run a basic analysis. You should collect data, from one or many sources, and combine them into a usable, clean dataset.

Ambition is a non-trivial portion of the grade. Ambition in data acquisition, analysis methods, and website presentation will be considered and rewarded.

## Components

As with any (interesting) project, the path from now to completion will not be a straight line.

![](project\_\_timeline.jpg)

So, to keep us on track, the project will have several deliverable stages, according to the [schedule](../about/schedule). *Changes to that master schedule supersede any dates below.*

```{dropdown} **Initial proposals (15%)**

* General idea: The question/problem are you interested in, the data you need to acquire, the variables you'll use, and the plan for how you'll analyze it (what methods you'll try and why you think they apply to your problem), considerations about how data might impact that.
* **Treat this document as if it is public facing, and a proposal for which you would like funding. That is, the proposal document should be polished (both in visual formatting and editing) for external audiences.**
* Graded on: question viability, creativity, finance application, plan sketch, writing quality.
* Instructions for the proposals are [here](project_prop_template).

```{dropdown} \*\*Final proposals (5%)\*\*  
  
- I will provide feedback to your proposals to help you "right size" your goal and avoid some pitholes.   
- Graded on: The improvement from the prior version, how feedback was incorporated, and current status

```{dropdown} **Project status report (20%)**

* General idea: You've now acquired the key data and finished most of the data cleaning.
* Purpose: Needs to show progress and that you're on track!
* *Ideal deliverable*: A notebook file with nice data sections describing data source(s) and how you got/cleaned the data. This section could go straight into your final report if it's polished enough.
* **Actual deliverable** A notebook file that
  + describes (short bullet points) your data sources,
  + outlines (numbered list, broad steps, not minutia) how you acquired the data (for many groups, the downloading is in a separate file), got the data into python, and if you found any issues with the data you cleaned up (again, possibly a different file)
  + includes a bullet point list of your main observations from your EDA
  + shows your exploratory data analysis (EDA) (tables and figures and whatnot, does not need to be pretty or formatted)
* Graded on: Data you have, EDA shown and discussed

```{dropdown} \*\*Repo at submission (20%)\*\*  
   
On the due date (listed in the schedule), your repo should be cleaned and polished for publication. That means it should be cleaned of excess and random files, and that folders are sensible (data, temporary, code), the readme helps me/the TA/future visitors explore your repo easily. Your folder structure is up to you and will respond to the nature of your particular project, but I should be able to easily find  
  
- The readme should contain a link to the website built off this analysis  
- The code used to scrape and download data (and if you click-and-download anything, a link to the source) can be separate files, and the code used to load, clean, merge, and explore the data.   
- The code used to do the analysis  
- Your presentation file - \*\*it needs to be in this repo.\*\* If you use google slides, you should include them as a PDF in this folder / put a link to the slides in the readme.  
  
Graded on: Folder org, read me, code readability/structure  
  
Obvious caveats for grading: Form matters, check grammar, and cite work you build on. \*Plagiarism is not acceptable.\*\*

```{dropdown} **Website (20%)**

We will talk about this more later.

Obvious caveats for grading: Form matters, check grammar, and cite work you build on. \*Plagiarism is not acceptable.\*\*

```{dropdown} \*\*Presentation (20%)\*\*  
  
I'll discuss scheduling later.  
  
- You have 15 minutes  
- Everyone should contribute  
- There will be Q&A (from myself)  
- Teach your classmates and myself something! Strive for clarity and making something about it memorable.  
   
Method: You can present a powerpoint, a jupyter file, or [jupyter slides](https://medium.com/@mjspeck/presenting-code-using-jupyter-notebook-slides-a8a3c3b59d67) (nice!). I'll leave it up to your group to present in the manner you consider most effective for your project.   
  
Time: Each group will have up to 15 minutes to present your project, so build your presentation file accordingly. Try to avoid "speed talking" to make the time work. Less is more, usually. Sadly, 15 minutes won't be enough to show everything you did, so focus on big picture details rather than on the syntax of line 89 of your code.  
  
Content: A presentation's structure is tailored even more to its material than a report is, so what your slides show is up to you. Be creative, and have fun. Try to convey to myself and your peers why the question is interesting, describe \*\*plainly\*\* your approach and why it approach makes sense, what your main analytical findings are, and what you concluded from the exercise. You can even show/use your website during your presentation if you want.  
  
Enjoyment: Don't be afraid to "market yourselves"! If you did something impressive (tons and tons of data, or an impressive scraper, or a great model), find a way to tastefully show your classmates (and me) the cool stuff you did!  
  
Obvious caveats for grading: Form matters, check grammar, and cite work you build on. \*Plagiarism is not acceptable.\*\*

# FILENAME: project\_prop\_template

# Full instructions for the proposals

In the project repo, create a file called "proposal". It should cover two big things:

1. The research question. It should be precise (NOT VAGUE), the hypothesis clear, and the metrics well defined.
2. The necessary data. This should be realistically acquirable over our time frame. There are a lot of data resources [on the website](about/resources.html#resources-tutorials-and-data), including FRED, <ourworldindata.com>, and SEC's EDGAR.

The template below is just a template. You can adapt it. For example, sell me on the idea and why it's both interesting and feasible.

# Research Proposal: < Title >

By X, Y, and Z

## Research Question

This section should cover:

1. What do we want to know or what problems are we trying to solve? As in the midterm, you should list (1) the "bigger" question/debate/problem you're interested in, and also (2) the specific research question(s) you'll actually try to answer.

* The research question will be smaller in scope than the big picture question. But the answer to your specific research question should *shed light* on the bigger question (although it likely won't conclusively answer it).
* The answer to your specific research question should *shed light* on the bigger question (although it likely won't conclusively answer it).

1. If your project is about relationships, what are the hypotheses you're testing?
2. If your project is about prediction, what is your metrics of success? [(What are you maximizing?)](https://ledatascifi.github.io/ledatascifi-2022/content/05/03d_whatToMax.html) Can you find a baseline from prior work to give you a ball park to aim for?

## Necessary Data

This section should cover:

1. What does the final dataset need to look like (mostly dictated by the question and the availability of data):
   * What is an observation, e.g. a firm, or a firm-year, etc.
   * What is the sample period?
   * What are the sample conditions? (Years, restrictions you anticipate (e.g. exclude or require some industries)
   * What variables are absolutely necessary and what would you like to have if possible?
2. What data do we have and what data do we need?
3. How will we collect more data?
4. What are the raw inputs and how will you store them (the folder structure(s) for each input type).
5. Speculate at a high level (not specific code!) about how you'll transform the raw data into the final form.

Acknowledgment: We are effectively answering questions 1.1-1.3 and 2.1-2.3 from [DS100](https://www.textbook.ds100.org/ch/01/lifecycle_students_1.html) in this proposal.