

# Lecture 1: Introduction to Data Science

LSE ME314: Introduction to Data Science and Machine Learning (<https://github.com/me314-lse>)

2025-07-14

**Daniel de Kadt**

# 오징어 게임 2



456억. 동심의 게임은 끝나지 않았다

ONLY ON **NETFLIX** | 12월 26일 공개

제작: 넷플릭스

# What Questions Can We Ask?

Imagine you're a member of Netflix's c-suite. What kind of questions might you have about Squid Game?

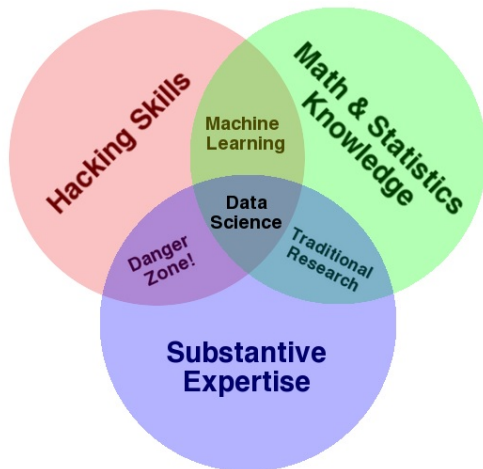
# What Questions Can We Ask?

Imagine you're a member of Netflix's c-suite. What kind of questions might you have about Squid Game?

- Did viewers like Squid Game?
- Which viewers most liked Squid Game?
- What types of viewers engaged with Squid Game?
- Did Squid Game increase Netflix viewership?
- What was the \$ value of Squid Game?
- Who should we advertise or push Squid Game to?
- How many \$ should we spend on advertising Squid Game?
- Should we invest in Season 4?
- What can we learn from Squid Game about the types of shows that are successful?

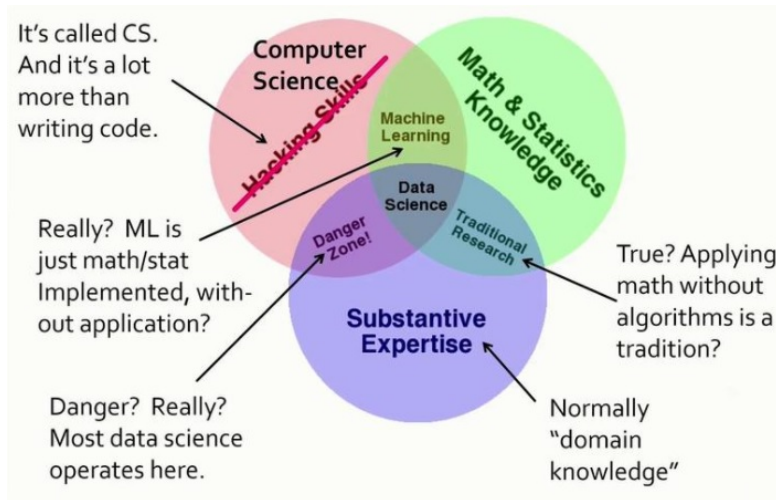
What is Data Science?

# What is Data Science?



Source: Drew Conway, [Data Science Venn Diagram](#)

# What is Data Science?



Source: Jeffrey Ullman, [What Is Data Science? A Turing Award Winner Shares His View](#)

# What is Data Science?

What's missing from this venn diagram?



# What is Data Science?

What's missing from this venn diagram?

## 1. **Data:**

- Typically at scale
- Is not value- or decision-free
- Must be domain-specific

# What is Data Science?

What's missing from this venn diagram?

## 1. **Data:**

- Typically at scale
- Is not value- or decision-free
- Must be domain-specific

## 2. **Science:**

- Principles, practices, and ethics
- Norms (that may or may not be good)
- Always domain-specific

# What is Data Science?

What's missing from this venn diagram?

## 1. Data:

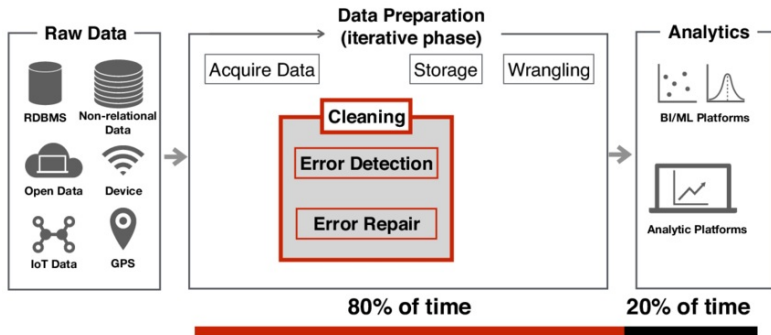
- Typically at scale
- Is not value- or decision-free
- Must be domain-specific

## 2. Science:

- Principles, practices, and ethics
- Norms (that may or may not be good)
- Always domain-specific

Let's try that again: Data science is the marriage of **scientific skills** to **data** (at scale), all of which are *a/ways* domain-specific.

# Data Science Landscape



*Note:* We will spend most of our time on the 20%, but some of lectures 1-3 will cover the 80%.

Source: Larysa Visengeriyeva, [Data Science Workflow](#)

# Data Science Landscape

## 1. Engineers

- Backend data engineers
- Analytics engineers
- Machine learning engineers

# Data Science Landscape

## 1. Engineers

- Backend data engineers
- Analytics engineers
- Machine learning engineers

## 2. Scientists

- Data scientists
- Machine learning scientists
- Research scientists
- Behavioral scientists
- Economists
- UX researchers

# Data Science Landscape

## 1. Engineers

- Backend data engineers
- Analytics engineers
- Machine learning engineers

## 2. Scientists

- Data scientists
- Machine learning scientists
- Research scientists
- Behavioral scientists
- Economists
- UX researchers

## 3. Analysts

- Business intelligence analysts
- Data analysts

# Data Science Landscape – Examples

## About the Role:

We're hiring a **Data Analyst** for a well-established retail business that supports **leading UK household brands**. This is their **first permanent data hire**, with a chance to take ownership of reporting and analytics and drive data-led decisions across the company. You'll work closely with stakeholders across **finance, operations, and leadership**, building on existing clean datasets and dashboards, and helping embed a culture of data-driven decision-making.

## Key Responsibilities:

- Own and manage company-wide reporting and dashboards
- Work closely with teams to understand and meet their data needs
- Deliver clear, actionable insights to non-technical stakeholders
- Maintain high data accuracy and integrity
- Support strategic decisions with reliable analysis

## Tech Requirements:

- **SQL** - essential
- **Any BI tool** - e.g., Power BI, Tableau, QuickSight
- Strong Excel skills

Example recruitment listing for “Data Analyst” position.

Source: LinkedIn, 10 July 2025.



# Data Science Landscape – Examples

## THE ROLE

As a **Data Engineer**, you will play a pivotal role in designing and building scalable data solutions that directly influence both internal operations and value creation across a world-leading portfolio. This is a unique opportunity to work in a hands-on, high-impact role shaping greenfield data platforms and embedding data strategy within ambitious software businesses.

Specifically, you can expect to be involved in the following:

- Designing and developing full-stack data pipelines and platforms using modern tools such as dbt, Airflow, and cloud infrastructure
- Cleansing, enriching and modelling data to generate commercial insights and power C-level dashboards
- Delivering scalable solutions that support internal use cases and extend directly to portfolio companies
- Working on project-based data initiatives, including AI/ML integrations for predictive modelling and cloud-based productisation
- Collaborating with commercial teams (RevOps, Finance, Sales) to turn messy data into meaningful business value
- Building internal data products and helping define the next generation of services across the wider platform

## SKILLS AND EXPERIENCE

The successful **Data Engineer** will have the following skills and experience:

- 5+ years in data engineering, with experience in commercial environments
- Proven ability to manage the full data lifecycle—from ingestion to transformation to delivery
- Comfortable working closely with commercial functions (e.g. finance, sales) and building tools that serve business needs
- Background in startups or scale-ups with high adaptability and a hands-on approach
- Experience with modern data tools (e.g. dbt, Airflow, CI/CD) and at least one cloud platform (AWS, GCP, Azure)

Example recruitment listing for “Data Engineer” position.

Source: LinkedIn, 10 July 2025.

# Data Science Landscape – Examples

## RESPONSIBILITIES:

- Work on designing and building advanced ML models - including propensity and clustering
- Apply machine learning, statistical, and econometric methods to a range of other customer data science problems
- Working closely within a team to understand, troubleshoot, and maintain machine learning models
- Maintain and enhance existing ML models, ensuring all models are deployed in production are monitored.
- Stay updated with the latest developments in ML/AI, and related fields to keep the company at the forefront of technological advancements

## REQUIREMENTS:

- PhD Degree in Computer Science, Artificial Intelligence, Mathematics, Statistics or related fields.
- Proven experience in clustering and propensity
- Strong coding skills in Python, R or MATLAB
- Excellent communication skills
- Knowledge of databases and SQL

Example recruitment listing for “Data Scientist (Machine Learning)” position.

Source: LinkedIn, 10 July 2025.

# Data Science Landscape – Examples

## What You'll Do

- Design and implement causal-inference frameworks (quasi-experimental, synthetic control) to quantify marketing uplift and ROI
- Build predictive and optimisation models to guide budget allocation across channels
- Translate complex statistical findings into concise presentations and dashboards for senior non-technical stakeholders
- Partner with marketing, data engineering and product teams to integrate models into live reporting pipelines

## Must-Haves

- Master's degree in Statistics, Econometrics, Data Science or related quantitative field
- Hands-on experience with causal-inference methods, including synthetic control techniques
- Deep conceptual understanding of statistical theory
- Excellent verbal and written communication-comfortable presenting to C-suite
- Demonstrable stakeholder engagement (beyond backend coding)
- Right to work in the UK

## Desirable

- 3+ years' experience in data-driven marketing, analytics consultancy or similar
- Familiarity with quasi-experimental designs and A/B testing frameworks
- Background in software design, data structures or production-grade ML engineering

## Technical Toolbox

- **Modelling & data:** Python or R, SQL
- **Containerisation:** Docker
- **Cloud platforms:** AWS, GCP or Azure

Example recruitment listing for “Data Scientist (Causal Inference)” position.

Source: LinkedIn, 10 July 2025.

# Data Science Landscape – Examples

## Role:

- Lead full-lifecycle data science projects within the Competitions domain — from discovery and modelling to deployment and business adoption.
- Design and implement models to forecast participation, optimise pricing and prize strategies, and segment audiences based on engagement patterns.
- Work closely with stakeholders in Consumer Promotions, Revenue Management, and Legal to ensure models are commercially effective and compliant.
- Embed experimentation, A/B testing, and uplift modelling into how the team runs campaigns and measures impact.
- Support the team with clear, explainable insights that can inform creative, operational, and compliance decisions.
- Mentor analysts and junior data scientists, and contribute to the wider data science capability across the business.
- Write clean, well-documented Python and SQL code, and collaborate with engineering teams to productionise models effectively.

## What We're Looking For

- 3+ years in applied data science roles, ideally with a strong commercial and product focus.
- Proven experience using data science to influence behaviour and drive performance in one or more of the following:
  - Consumer competitions, prize draws, or gaming
  - Revenue/yield management or dynamic pricing
  - Digital media or entertainment
- Expertise in Python and SQL, with a strong grounding in statistical and machine learning techniques (e.g. scikit-learn, XGBoost, statsmodels).
- Experience working with modern cloud data stacks (e.g. Snowflake, BigQuery, Redshift) and transformation tools like dbt.
- Familiarity with A/B testing, causal inference, and techniques for forecasting or audience

Example recruitment listing for “Data Scientist (Full Stack)” position.

Source: LinkedIn, 10 July 2025.

# Data Science Problems

Recall, this course will focus on the **scientists** part of the landscape (the 20%). What kind of problems do those people try to solve?

# Data Science Problems

Recall, this course will focus on the **scientists** part of the landscape (the 20%). What kind of problems do those people try to solve?

1. Descriptive problems:

- Did viewers like Squid Game?
- Can we group our Squid Game viewers into different types?

# Data Science Problems

Recall, this course will focus on the **scientists** part of the landscape (the 20%). What kind of problems do those people try to solve?

1. Descriptive problems:

- Did viewers like Squid Game?
- Can we group our Squid Game viewers into different types?

2. Predictive problems:

- Which viewers are likely to enjoy Squid Game?
- What other shows are Squid Game viewers likely to enjoy?

# Data Science Problems

Recall, this course will focus on the **scientists** part of the landscape (the 20%). What kind of problems do those people try to solve?

1. Descriptive problems:

- Did viewers like Squid Game?
- Can we group our Squid Game viewers into different types?

2. Predictive problems:

- Which viewers are likely to enjoy Squid Game?
- What other shows are Squid Game viewers likely to enjoy?

3. Causal problems:

- Did Squid Game cause viewers to stay or churn?
- Was Squid Game worth the investment?



# Data Science Problems

Recall, this course will focus on the **scientists** part of the landscape (the 20%). What kind of problems do those people try to solve?

1. Descriptive problems:

- Did viewers like Squid Game?
- Can we group our Squid Game viewers into different types?

2. Predictive problems:

- Which viewers are likely to enjoy Squid Game?
- What other shows are Squid Game viewers likely to enjoy?

3. Causal problems:

- Did Squid Game cause viewers to stay or churn?
- Was Squid Game worth the investment?

**Good data scientists** scope their problem, implement the right kinds of *solutions*, and deliver *meaningful learning*.

# Data Science Solutions: Measurement and Regression

## Measurement:

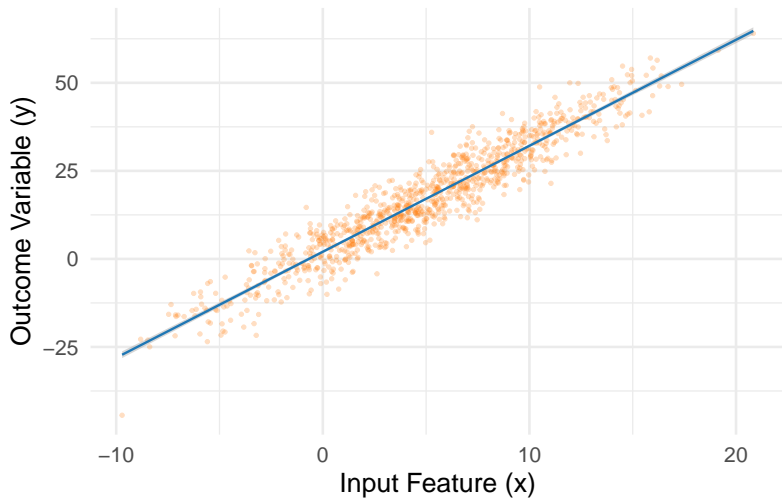
- You define a concept and develop a valid and reliable measure
- Examples → surveys, psychometrics, and measurement models

## Regression:

- The foundational tool for statistical comparison
- You connect an output variable  $Y$  to input features via a chosen functional ('parametric') form
- Examples → ordinary least squares ('linear regression'), generalized linear models, and generalized additive models

# Data Science Solutions: Measurement and Regression

## Linear Regression



# Data Science Solutions: Machine Learning Models

## Supervised learning:

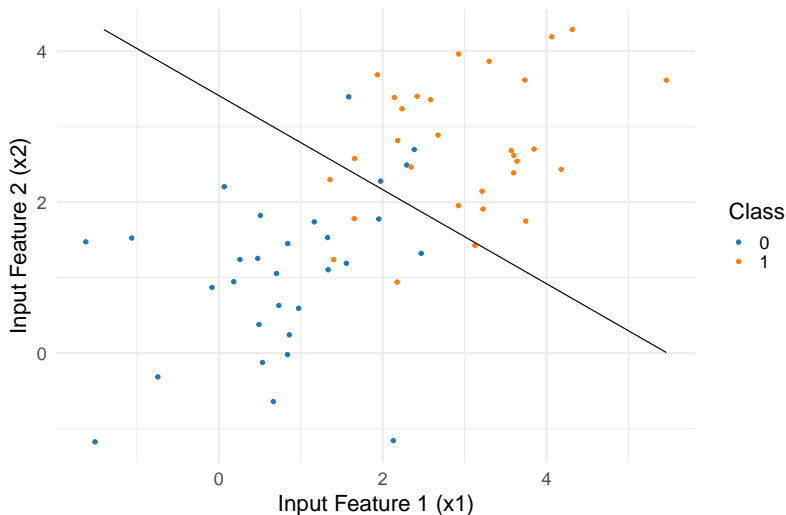
- You fit a model on some ('training') data, then apply the model to unseen ('out of sample') data
- You provide a set of output labels, and the model assigns those output labels to new inputs
- Examples → linear regression ('LPM'), logistic regression, decision trees, random forests, and neural networks

## Unsupervised learning:

- You provide a set of inputs, and the model tries to find patterns in those inputs
- No training required (but may use cross-validation or other out-of-sample techniques)
- Example → clustering, dimension reduction, topic modeling, word embeddings

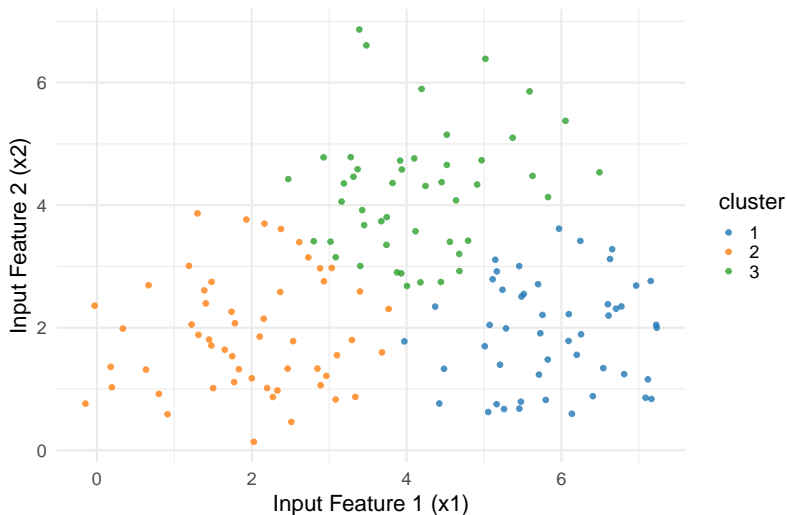
# Data Science Solutions: Machine Learning Models

## Logistic Regression as Classifier



# Data Science Solutions: Machine Learning Models

K-Means Clustering ( $k = 3$ )



# Data Science Solutions: Causal Designs

## Experiments (“A/B tests”):

- You control assignment of some causal feature (A or B)
- Examples → classical, clustered, and cross-overs

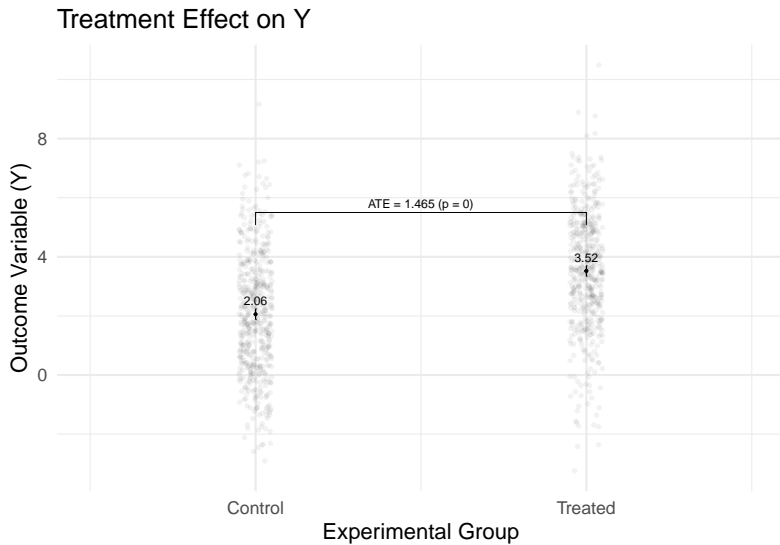
## Observational designs:

- You assume something about the assignment mechanism
- Examples → selection on observables, instrumental variables, and regression discontinuity (kinda)

## Model-based analyses:

- You assume something about your outcome variable
- Examples → regression discontinuity (kinda), difference-in-differences, synthetic control

# Data Science Solutions: Causal Designs





# Data Science Learnings

What kind of learnings might we look for from any given problem and solution?

# Data Science Learnings

What kind of learnings might we look for from any given problem and solution?

1. Retrospection:

- Example: "This is what happened"
- Useful for understanding past events and planning for the future.

# Data Science Learnings

What kind of learnings might we look for from any given problem and solution?

1. Retrospection:

- Example: “This is what happened”
- Useful for understanding past events and planning for the future.

2. Predictions:

- Example: “This is what we expect to happen”
- Useful for targeting and forecasting.

# Data Science Learnings

What kind of learnings might we look for from any given problem and solution?

1. Retrospection:

- Example: "This is what happened"
- Useful for understanding past events and planning for the future.

2. Predictions:

- Example: "This is what we expect to happen"
- Useful for targeting and forecasting.

3. Explanations:

- Example: "This is why/how something happened"
- Useful for understanding what's going on.

# Data Science Learnings

What kind of learnings might we look for from any given problem and solution?

1. Retrospection:

- Example: “This is what happened”
- Useful for understanding past events and planning for the future.

2. Predictions:

- Example: “This is what we expect to happen”
- Useful for targeting and forecasting.

3. Explanations:

- Example: “This is why/how something happened”
- Useful for understanding what’s going on.

**Good data scientists** think carefully about the *type of learnings* they hope to extract from any given project.

# Computational Landscape

# The Command Line

We will spend a lot of this class 'talking to' our computers.

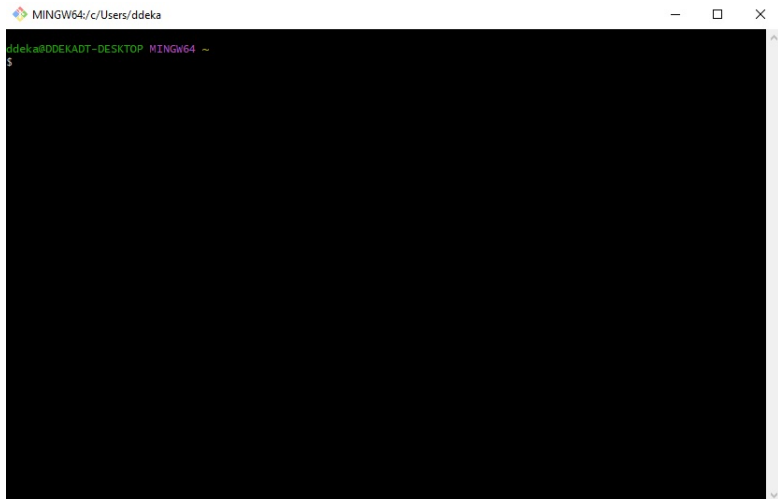
One way to do that directly is via the 'command line':

In Mac or Linux, this is called the 'Terminal'.

In Windows, use Git Bash or PowerShell (not command prompt).

We can use the command line to navigate around our folders, execute Git commands, execute scripts, and interact with our files.

# The Command Line





# The Command Line

Useful commands (Linux/Unix/MacOS):

- `pwd`: Print working directory
- `ls`: list folders and files in the working directory
- `cd`: Change directory using relative filepaths
  - `cd ..` goes up one folder level
- Other helpful commands include `mkdir`, `rmdir`, `rm`, and `touch`
- Can also use `git` in the command line (more later!)

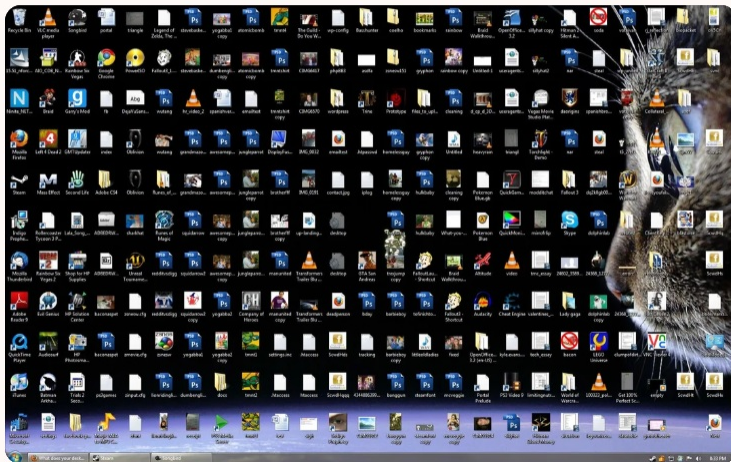
# Folders and Files



r/mildlyinfuriating • 6 yr. ago  
[deleted]



## People whose desktops look like this



# Folders and Files

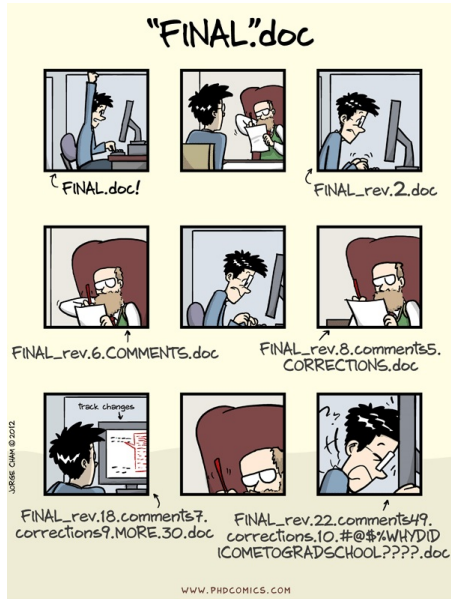
📁 > Search Results in Dropbox (Personal) > project_template > code > R >			
Name	Date modified	Type	Size
📁 functions	10/1/2024 8:02 PM	File folder	
📄 .gitignore	5/9/2024 9:47 AM	Text Document	1 KB
📄 _targets.R	5/9/2024 9:47 AM	R File	3 KB
📄 MY560_reproducible_pipelines_project_te...	5/9/2024 9:47 AM	R HTML File	4 KB

# Folders and Files

Some general rules to follow:

- For each project you work on, create a new folder to house all files related to that project
- If using Git/GitHub (more later!), each repository = a project = a top-level folder (with optional sub-folders)
- You can have too much structure, but more > less
- Use descriptive names for folders and files, and avoid spaces in file names:
  - Folders: `scripts`, `data`, `results`, `figures`, `docs`, `code`, `notebooks`
  - Files: `data_raw.csv`, `data_clean.csv`, `runner.py`, `cleaning.R`, `results.txt`, `figures.png`, `report.qmd`
  - Avoid hardcoding dates or versions where possible (sometimes necessary, but better inside the file)

# Folder and Files



\*\*\*Footnote: Thanks to Matt from UCLA for this comic idea!

# Programming for Data Science

# Programming Languages: Overview

A programming language has:

1. Primitives:

- Data types: Numbers, strings, etc.
- Operators: Mathematical, logical, etc.
- Control flow: If-else statements, loops, etc.

2. Syntax:

- Words: Individual tokens
- Phrases: How to put tokens together
- Context: How to assign meaning to tokens

3. Semantics:

- What code 'means' to your computer

# Programming Languages: Overview

A programming language has:

1. Primitives:

- Data types: Numbers, strings, etc.
- Operators: Mathematical, logical, etc.
- Control flow: If-else statements, loops, etc.

2. Syntax:

- Words: Individual tokens
- Phrases: How to put tokens together
- Context: How to assign meaning to tokens

3. Semantics:

- What code 'means' to your computer

Computer programme: A set or series of instructions that a computer is asked to perform (aka an algorithm).



# Programming Languages: Overview

A programming language has:

1. Primitives:

- Data types: Numbers, strings, etc.
- Operators: Mathematical, logical, etc.
- Control flow: If-else statements, loops, etc.

2. Syntax:

- Words: Individual tokens
- Phrases: How to put tokens together
- Context: How to assign meaning to tokens

3. Semantics:

- What code 'means' to your computer

Computer programme: A set or series of instructions that a computer is asked to perform (aka an algorithm).

Remember: Your intended meaning *may not be the same* as the computer's interpreted meaning!

# Programming Languages: Differences

Some points of difference:

- Low level (e.g. assembly or C++) vs. high level (e.g. Python, R, Julia)
- Functional programming vs. object oriented programming (OOP)

# Functional Programming vs OOP

Two paradigms:

---

## Functional Programming

Organizes code around *functions*

Functions take inputs and return outputs

Emphasizes vectorization and immutability

Functions are often pure (no side effects)

---

## (OOP)

Organizes code around *objects*

Objects belong to classes with attached methods

Emphasizes encapsulation, inheritance, modularity

Data and behavior are bundled in objects

---

## R: Introduction

High level programming language targeted at statistical computing

Functional programming emphasis, but can be used in an object-oriented fashion

Uses 1-based indexing (e.g. `x[1]` is the first element of `x`)

Open source, strong community, focused on econometrics and statistics

Well suited to one-off analyses - often too unstable for production

Installs from <https://www.r-project.org/> (current version will be something like R 4.X.X)

## R: Objects and Assignment

Everything in R is an object. The assignment operator (`<-`) binds objects on the right to names on the left.

Can also use `=` but conventionally use `<-` for assignments, and `=` for arguments in functions.

## R: Objects and Assignment

Everything in R is an object. The assignment operator ( $\leftarrow$ ) binds objects on the right to names on the left.

Can also use  $=$  but conventionally use  $\leftarrow$  for assignments, and  $=$  for arguments in functions.

```
# Assign the object c(1,2,3) to the name x
x <- c(1, 2, 3)
x
```

```
[1] 1 2 3
```

## R: Objects and Assignment

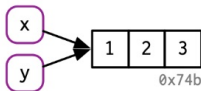
Everything in R is an object. The assignment operator ( $\leftarrow$ ) binds objects on the right to names on the left.

Can also use  $=$  but conventionally use  $\leftarrow$  for assignments, and  $=$  for arguments in functions.

```
# Assign the object c(1,2,3) to the name x
x <- c(1, 2, 3)
x
```

```
[1] 1 2 3
```

```
# Assign the object c(1,2,3) to the name y, via x
y <- x
```



# R: Functions

Everything in R is an object, but things happen via function calls:

- Functions are first-class objects
- Functions take arguments and return values
- Many of these functions are built-in (`{base}`)
- Sometimes you don't realise you are using one (but you are)
- You can also write your own functions (you should)



## R: Functions

Everything in R is an object, but things happen via function calls:

- Functions are first-class objects
- Functions take arguments and return values
- Many of these functions are built-in (`{base}`)
- Sometimes you don't realise you are using one (but you are)
- You can also write your own functions (you should)

In R we say that `print()` is a function, and the text in parentheses is the argument to the function:

```
print("Hello, world!")
```

```
[1] "Hello, world!"
```

## R: Packages

R is often most useful when we use packages. Those include custom functions:

- Packages are denoted by {}, e.g. {base} or {dplyr} or {ggplot2}
- Functions can be masked by various packages (naming conflicts)
- To refer to specific packages with ::, e.g. dplyr::filter() will call filter() from the {dplyr} package, even if there are other functions called filter()

Packages are hosted on the Comprehensive R Archive Network (CRAN) and usually well documented

Install packages with `install.packages("PACKAGE_NAME")`.

Load them with `library(PACKAGE_NAME)` or `require(PACKAGE_NAME)`.

# Python: Introduction

High level programming language for general computing

Object-oriented emphasis, but can be used in a functional fashion (we will mostly do this)

Uses 0-based indexing (e.g. `x[0]` is the first element of `x`)

Open source, strong community, focused on machine learning

Versatile and “robust” - good for production

Natively installed on Mac, but you might want to update. For Windows, installs from <https://www.python.org/> or via command line (current version will be something like Python 3.X.X)

# Python: Objects and Assignment

Like R, everything in python is an object.

The assignment operator (=) binds a variable (left) to an object (right) in memory.

```
# Assign the object 5 to the variable x
x = 5
print(x)
```

5

Again, the variable x points to the object 5, so could y

# Python: Functions

As with R, functions play an important role in python:

- Functions are first-class objects
- Functions take arguments and return values
- Many useful functions are built-in
- You can write your own functions

```
print("Hello, world!")
```

Hello, world!

# Python: Packages and Modules

In python, modules are python files (.py) containing code you can reuse.

Packages are collections of (usually related) modules.

Install packages with `pip install package_name` in the command line, or using a package manager like conda

Import and use them in your code with `import PACKAGE_NAME` or alias with `import PACKAGE_NAME as ALIAS`

Masking problems apply here too, so avoid confusion by aliasing on import then using `ALIAS.function()`

# Python vs. R: Operators and Mathematical Functions

Description	Python	R
Addition	+	+
Subtraction	-	-
Division	/	/
Multiplication	*	*
Exponentiation	**	^
Modulus (remainder)	%	%%
Integer (floor) division	//	%/%

## Python vs. R: Logical Operators

Purpose	Python	R
Less than	<	<
Greater than or equal to	>=	>=
Is equal to	==	==
Not equal to	!=	!=
Logical AND	and	&
Logical OR	or	
Membership test	in	%in%



# Python vs. R: OOP vs. Functional Programming

**Problem:** Take an input and double it.

Functional programming solution in R:

```
# Define function that takes x and doubles it
double <- function(x) {
  x * 2
}
```

```
# Use the function for an input:
double(4)
```

```
[1] 8
```

# Python vs. R: OOP vs. Functional Programming

**Problem:** Take an input and double it.

Object-oriented programming solution in python:

```
# Create a class called Number with attached methods:
class Number:
    def __init__(self, x):
        self.x = x
    # Method to double the object
    def double(self):
        return self.x * 2

# Create an instance of the Number class
num = Number(4)
# Use the method to double the object num
num.double()
```

# Python vs. R: OOP vs. Functional Programming

**Problem:** Take an input and double it.

But python can also be used functionally:

```
# Define function that takes x and doubles it
def double(x):
    return x * 2

# Use the function for an input:
print(double(4))
```

# Python vs. R: Conventions

Python has strong conventions and 'best' practices:

- Consult the **PEP 8** style guide
- People talk about code being `pythonic` or not

# Python vs. R: Conventions

Python has strong conventions and 'best' practices:

- Consult the **PEP 8** style guide
- People talk about code being *pythonic* or not

R has far fewer conventions:

- Consult **R for Data Science (2e)** for the closest approximation to PEP 8
- People talk about code being either *tidy* or *base* (*not* based) - this is very different to what *pythonic* means

## Aside: SQL and Database Languages

We won't learn any, but you should know about Structured Query Language (SQL)

SQL comes in many flavours, but broadly it is used to query and manipulate databases

Reasonably easy to learn, but very powerful

We will learn `{dplyr}` in R which often mocks SQL by design.

```
SELECT first_name,  
       last_name,  
       email  
FROM employees  
WHERE department = 'Sales'
```

# Markup Languages: Markdown and Dynamic Documents

Markdown: Plain text with formatting.

Can be used in notebook fashion, combining chunks of R ('Rmarkdown') or python ('Jupyter') code with text – these are called dynamic documents.

We will use .qmd files that can combine both R and python code with text

Your seminar/class exercises will be run from .qmd files.

# Markup Languages: Markdown and Dynamic Documents

## ### 6. Exercise

Let us create some: ``a`` should be a float, ``b`` a integer and ``string`` a string.

> Run Cell | Run Next Cell | Run Above

```
```{python, eval=FALSE}
```

```
a = ____
```

```
print(type(a))
```

```
b = ____
```

```
print(type(b))
```

```
string = "____"
```

```
print(type(string))
```

```
```
```

Lists are used to store multiple items, not necessarily of the same type, in a single variable.

- \* Ordered
- \* can contain arbitrary objects
- \* Elements can be accessed by index
- \* Allow duplicates
- \* Can be nested
- \* Mutable
- \* Dynamic

> Run Cell | Run Next Cell | Run Above

```
```{python}
```

```
my_list = ["dog", "cat", 1, 1.521, True]
```

```
my_list
```

```
```
```



# Integrated Development Environments (IDEs)

One can just write code in a text editor (e.g. notepad), but why would you. . .

# Integrated Development Environments (IDEs)

One can just write code in a text editor (e.g. notepad), but why would you. . .

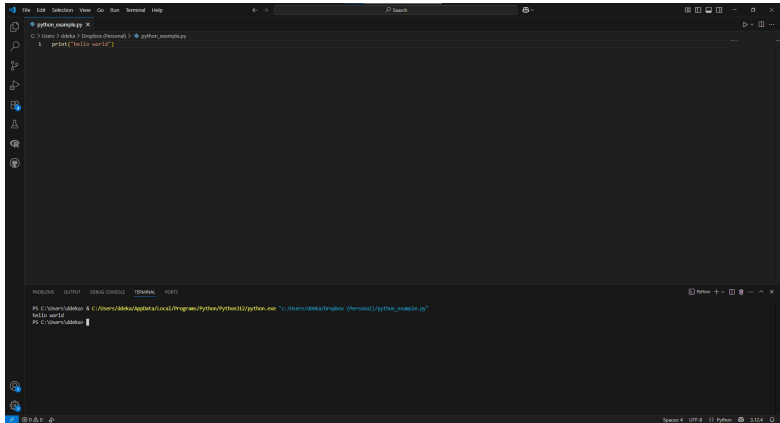
Instead, use an IDE which provides comfort and productivity tools.

Examples (there are many):

- RStudio: For R and Rmarkdown
- PyCharm: For python
- **VScode**: For many languages and dynamic documents



# Integrated Development Environments (IDEs)



# Using VSCode

Some machines may have VSCode pre-installed

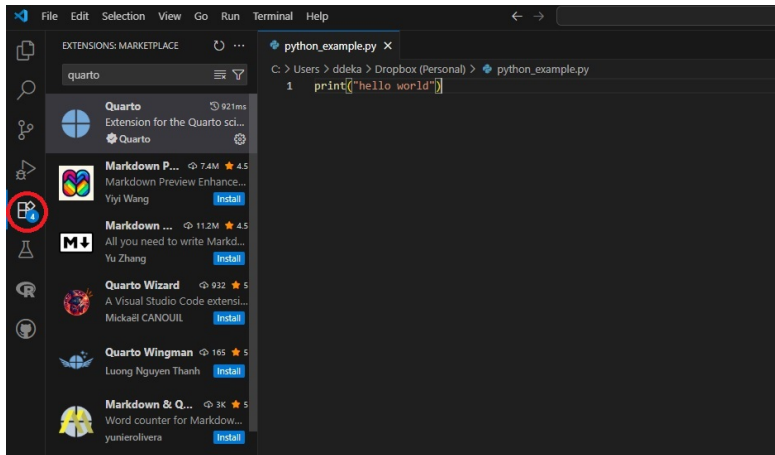
If not, install from: <https://code.visualstudio.com/>

VSCode supports many 'extensions' which you can install from within the IDE:

- R extensions
- python extensions
- Quarto extension
- GitHub Pull Requests extension

You may need be prompted to install some packages or modules by VSCode (e.g. {languageserver} for R)

# Using VSCode: Extensions



## Version Control

# Git



git

--distributed-is-the-new-centralized

🔍 Type / to search entire site...

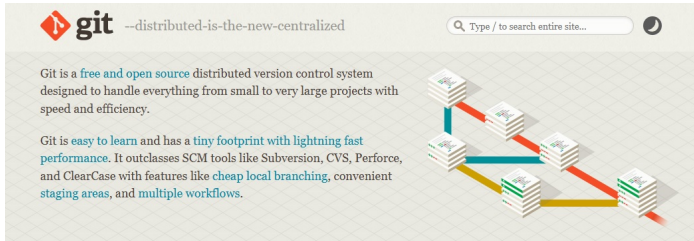


Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.



# Git



The screenshot shows the Git website homepage. At the top left is the Git logo (a red octocat) followed by the text "git --distributed-is-the-new-centralized". To the right is a search bar with the placeholder text "Type / to search entire site..." and a dark mode toggle icon. Below the header, there are two paragraphs of text. The first paragraph describes Git as a "free and open source" distributed version control system. The second paragraph describes Git as "easy to learn" and having a "tiny footprint with lightning fast performance". To the right of the text is a diagram illustrating Git's distributed nature, showing multiple stacks of files (represented as books) connected by colored lines (red, blue, yellow) representing different branches or commits.

**git** --distributed-is-the-new-centralized

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.

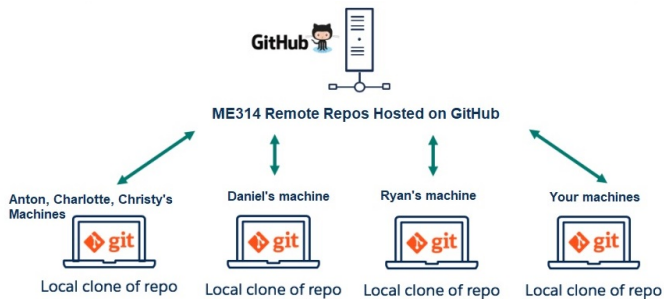
Version control system designed to (collaboratively) manage files.

Git “tracks” files and any changes made to it over time – you can always see *who* made *what* changes *when*.

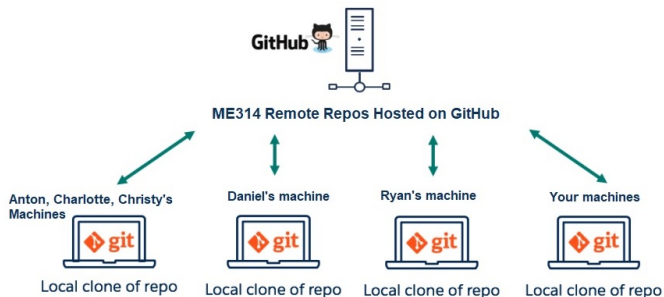
If something bad happens, can revert to previous versions.



# Git: Repositories



# Git: Repositories



Files are typically grouped into repositories – a folder in which all files are tracked (unless ignored)

This can either occur locally or in distributed fashion (over multiple machines/servers)

Git is an indisputably necessary part of any professional data science environment

# GitHub in ME314


GitHub is a service for managing Git, collaboration, and version control

We will use GitHub (and thus Git) extensively in this class

The course is hosted on GitHub:

1. Organization page: <https://github.com/me314-lse>.
2. Lecture slide repo: <https://github.com/me314-lse/lectures>.
3. Seminars each have their own repo:
  - a. Seminar 1
  - b. Seminar 2
  - c. Etc.

# GitHub in ME314

**ME314: Introduction to Data Science and Machine Learning**  
Home of ME314: Introduction to Data Science and Machine Learning, at the London School of Economics and Political Science.  
AI, 1 follower   United Kingdom

ME314: Introduction to Data Science and Machine Learning

Welcome to ME314! This course will be taught primarily through this github organization. Every day the lecture slides will be uploaded to the lectures repo as .pdf files. In the afternoon, you will clone into a new seminar\_day# repo, which will contain materials for the day's seminar activities.

Course Description

This course is an intensive and hands-on introduction to data science and machine learning, designed for students and professionals working with data in the social sciences, public policy, business, and beyond. It equips students with the practical tools and conceptual foundations needed to analyse complex datasets, build credible empirical arguments, and extract insight from both structured and unstructured data. A core theme of the course is using data science to bring clarity to ambiguity: transforming open-ended questions and messy data into structured, reproducible, and impactful analyses. Students will develop highly sought-after technical and analytical skills that enable them to work independently on real-world data problems—skills that are applicable across academic research, policy analysis, and data-driven roles in industry, and which also provide a strong foundation for more specialised training.

The course is divided into three parts: Fundamentals, Foundations, and Frontiers. In Fundamentals, students build core skills in data wrangling, visualization, and statistical reasoning, up to and including regression. Foundations focuses on rigorous analysis, covering how to infer causal relationships—including running experiments and distinguishing correlation from causation—as well as key machine learning techniques used for classification, clustering, and scaling. Frontiers introduces cutting-edge tools and challenges, including text-as-data, sourcing data from the web, unstructured data, and large language models, alongside a capstone lecture on how theoretical models of behaviour and strategy—such as rational choice and game theory—can help data scientists decide which questions to ask, design better studies, and interpret what empirical patterns actually mean.

Each topic in the course will be reinforced through hands-on programming in both R and Python using real-world datasets. We will also integrate generative AI tools alongside pdf-latex coding, reflecting how these technologies are increasingly used in professional and research settings. Rather than relying on these tools to do the work, students will learn how to use them effectively and critically—evaluating whether outputs are accurate, appropriate, and aligned with their analytical goals. Students will learn the skills required to be discerning users of generative AI, capable of integrating these increasingly important tools into rigorous, real-world data workflows.

First Week Office Hours

To assist with any computational troubles early in the class, the GTAs will hold office hours on Monday, Tuesday, and Wednesday of the first week of class in room CBG 3.19:

- Monday 14 July: 4-5pm
- Tuesday 15 July: 9-10am, 1-2pm, 5-6pm
- Wednesday 16 July: 9-10am, 1-2pm

Teaching Team

Dr Ryan Hubert, Dr Daniel de Kadt, Christy Coulton, Anton Korneika, and Charlotte Kubacka

View as: **Public**

You are viewing the EACOME and pinned repositories as a public user.  
You can pin repositories visible to anyone.  
You can hide the tasks we've suggested on this page and bring them back later.

Discussions

Set up discussions to engage with your community!  
[Turn on discussions](#)

Repositories

github


Updated 2 days ago

lectures

Updated 2 weeks ago

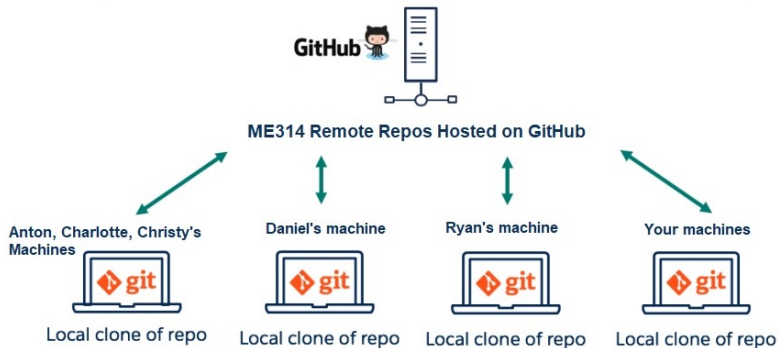
Create new repository   Import

People



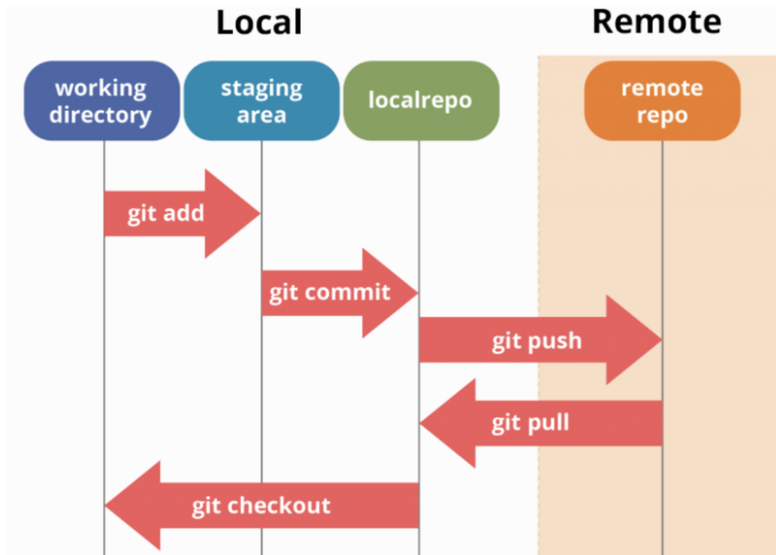
Invite someone

# GitHub in ME314



Source: Adapted from sfdc technie, [GitHub](#)

# GitHub: Workflow Basics



Source: Molly Nemerever, [Git](#), [GitHub](#), & [Workflow Fundamentals](#)

# GitHub: Workflow Commands

For our purposes, the key commands we will use throughout are:

1. `git clone`: Save a repo from a remote server to a local folder
2. `git pull`: Pull files from the remote server to the local folder
3. `git status`: Check the status of the local repo
4. `git add`: Stage files, preparing them for commit
5. `git commit`: Commit changes to the local repo, with a message
6. `git push`: Push changes and message from the local repo to the remote server

To use, navigate to a local repo on your computer, and use these commands either in the command line (e.g. through Git Bash), or through an IDE (VScode, or GitHub Desktop).

We recommend using the command line while you are learning ('learn to fly!').

# GitHub: Advanced Usage

In a development setting, you will also need to understand and use:

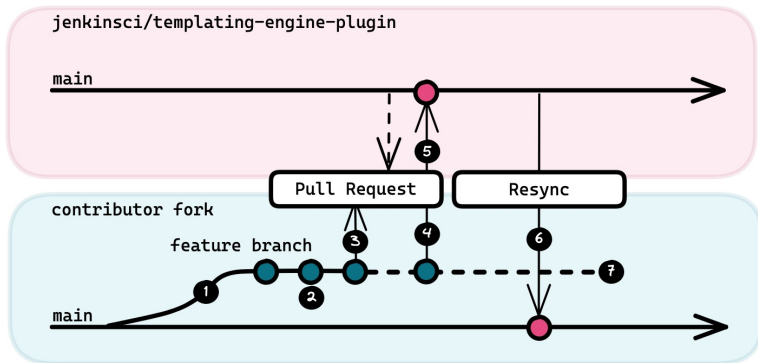
- Fork: Make your own version of a different repository (you're *forking* off, not *cloning*)
- Branch: A parallel version of the code originating at a particular point
- Merge: Combining branches together to maintain single codebase
- Pull request: GitHub based request to merge a branch or a fork into other code

You won't use these in this course, but they are important for professional data science work.

A great site for practice: <https://learngitbranching.js.org/>



# GitHub: Advanced Usage



Source: Jenkins Template Engine, **Fork-Based Contribution Model**

# Summary and Next Steps

## Today we covered:

1. What data science is, what kinds of problems data scientists solve with what solutions
2. What programming languages are, and the very basics of R and python
3. What version control is, and how to use Git and GitHub

This afternoon you will practice 2 and 3.

## Tomorrow:

1. What is data?
2. Different kinds of data
3. Working with data in R