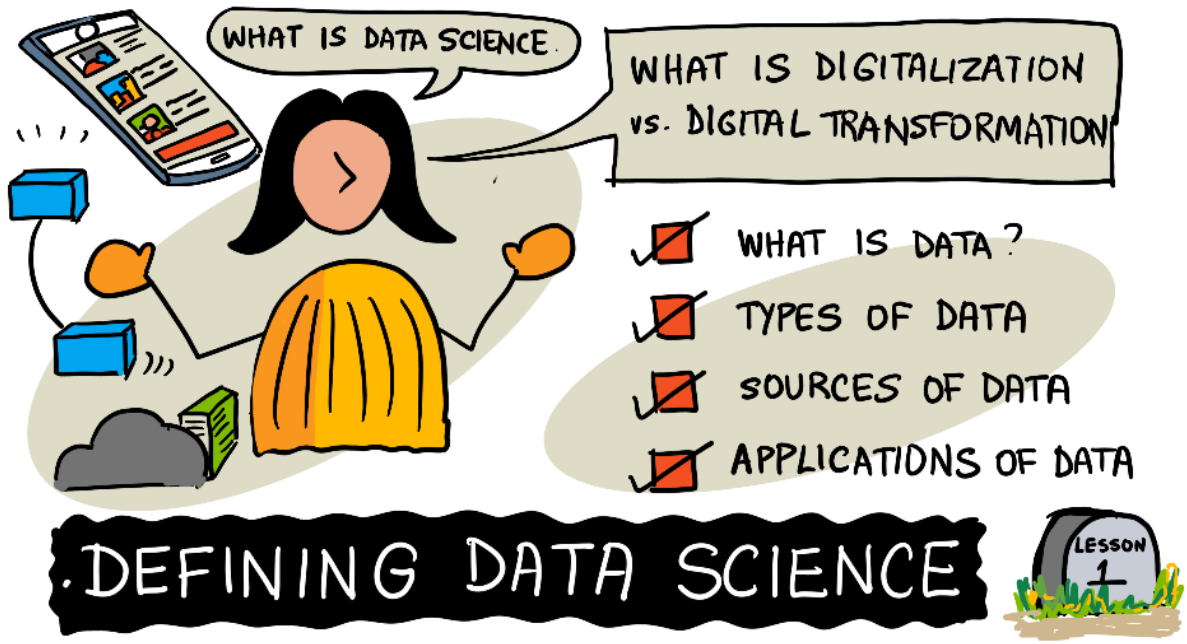
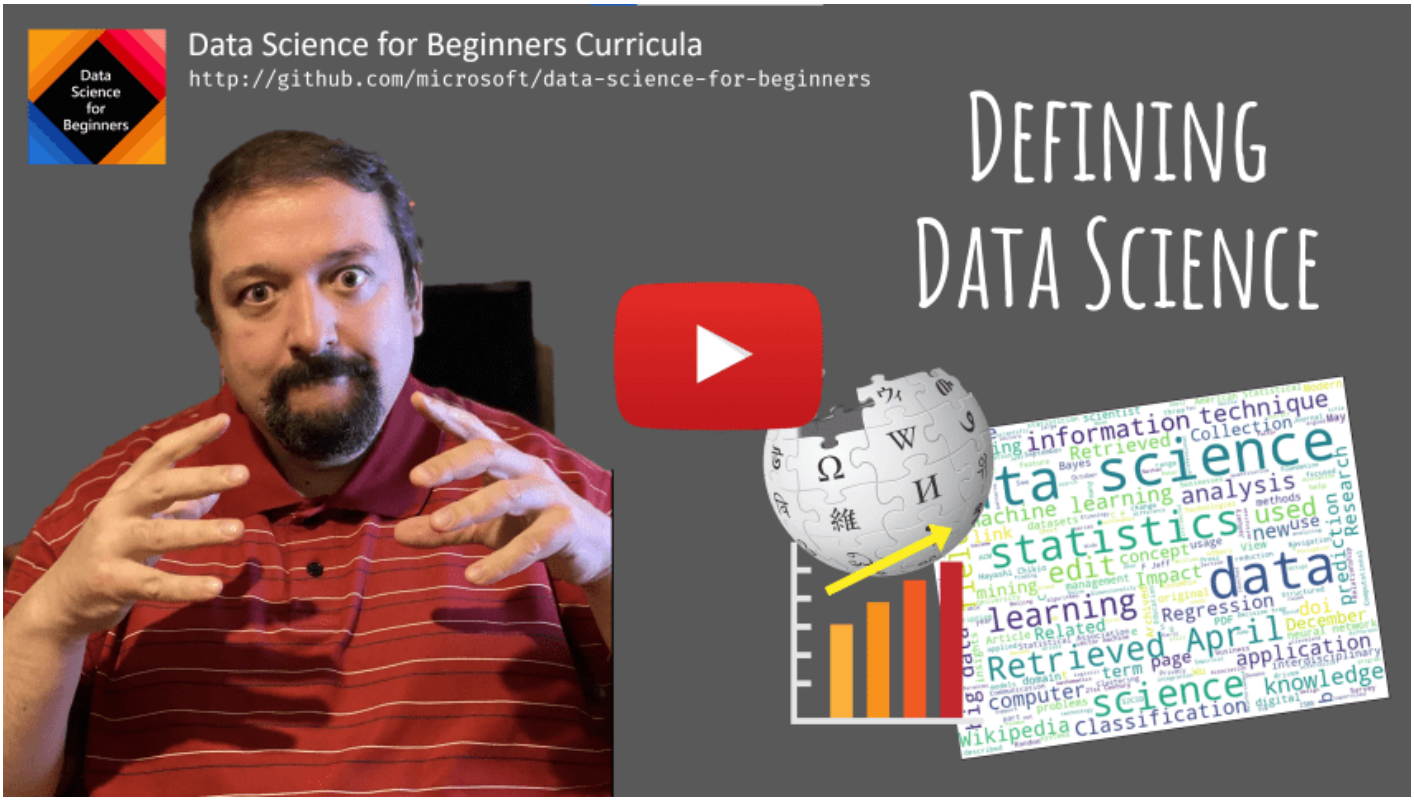
Defining Data Science



Defining Data Science - *Sketchnote by @nitya*



Pre-lecture quiz

What is Data?

In our everyday life, we are constantly surrounded by data. The text you are reading now is data, the list of phone numbers of your friends in your smartphone is data, as well as the current time displayed on your watch. As human beings, we naturally operate with data by counting the money we have or writing letters to our friends.

However, data became much more critical with the creation of computers. The primary role of computers is to perform computations, but they need data to operate on. Thus, we need to understand how computers store and process data.

With the emergence of the Internet, the role of computers as data handling devices increased. If you think of it, we now use computers more and more for data processing and communication, rather than actual computations. When we write an e-mail to a friend or search for some information on the Internet - we are essentially creating, storing, transmitting, and manipulating data.

> Can you remember the last time you have used computers to actually compute something?

What is Data Science?

In Wikipedia, **Data Science** is defined as *a scientific field that uses scientific methods to extract knowledge and insights from structured and unstructured data, and apply knowledge and actionable insights from data across a broad range of application domains*.

This definition highlights the following important aspects of data science:

- The main goal of data science is to **extract knowledge** from data, in order words - to **understand** data, find some hidden relationships and build a **model**.
- Data science uses **scientific methods**, such as probability and statistics. In fact, when the term *data science* was first introduced, some people argued that data science is just a new fancy name for statistics. Nowadays it has become evident that the field is much broader.
- Obtained knowledge should be applied to produce some **actionable insights**.
- We should be able to operate on both **structured** and **unstructured** data. We will come back to discuss different types of data later in the course.
- **Application domain** is an important concept, and data scientist often needs at least some degree of expertise in the problem domain.

> Another important aspect of Data Science is that it studies how data can be gathered, stored and operated upon using computers. While statistics gives us mathematical foundations, data science applies mathematical concepts to actually draw insights from data.

One of the ways (attributed to Jim Gray) to look at the data science is to consider it to be a separate paradigm of science:

- **Empyrical**, in which we rely mostly on observations and results of experiments
- **Theoretical**, where new concepts emerge from existing scientific knowledge
- **Computational**, where we discover new principles based on some computational experiments
- **Data-Driven**, based on discovering relationships and patterns in the data

Other Related Fields

Since data is a pervasive concept, data science itself is also a broad field, touching many other related disciplines.

Databases
: The most obvious thing to consider is **how to store** the data, i.e. how to structure them in a way that allows faster processing. There are different types of databases that store structured and unstructured data, which [we will consider in our course](../../2-Working-With-Data/README.md).

Big Data
: Often we need to store and process really large quantities of data with relatively simple structure. There are special approaches and tools to store that data in a distributed manner on a computer cluster, and process them efficiently.

Machine Learning
: One of the ways to understand the data is to **build a model** that will be able to predict desired outcome. Being able to learn such models from data is the area studied in **machine learning**. You may want to have a look at our [Machine Learning for Beginners] (https://github.com/microsoft/ML-For-Beginners/) Curriculum to get deeper into that field.

Artificial Intelligence
: As machine learning, artificial intelligence also relies on data, and it involves building high complexity models that will exhibit the behavior similar to a human being. Also, AI methods often allow us to turn unstructured data (eg. natural language) into structured by extracting some insights.

Visualization

Vast amounts of data are incomprehensible for a human being, but once we create useful visualizations - we can start making much more sense of data, and drawing some conclusions. Thus, it is important to know many ways to visualize information - something that we will cover in [Section 3](../../3-Data-Visualization/README.md) of our course. Related fields also include **Infographics**, and **Human-Computer Interaction** in general.

Types of Data

As we have already mentioned - data is everywhere, we just need to capture it in the right way! It is useful to distinguish between **structured** and **unstructured** data. The former are typically represented in some well-structured form, often as a table or number of tables, while latter is just a collection of files. Sometimes we can also talk about **semistructured** data, that have some sort of a structure that may vary greatly.

Structured	Semi-structured	Unstructured
List of people with their phone numbers	Wikipedia pages with links	Text of Encyclopaedia Britannica
Temperature in all rooms of a building at every minute for the last 20 years	Collection of scientific papers in JSON format with authors, data of publication, and abstract	File share with corporate documents
Data for age and gender of all people entering the building	Internet pages	Raw video feed from surveillance camera

Where to get Data

There are many possible sources of data, and it will be impossible to list all of them! However, let's mention some of the typical places where you can get data:

- **Structured**
  - **Internet of Things**, including data from different sensors, such as temperature or pressure sensors, provides a lot of useful data. For example, if an office building is equipped with IoT sensors, we can automatically control heating and lighting in order to minimize costs.
  - **Surveys** that we ask users after purchase of a good, or after visiting a web site.

    - **Analysis of behavior** can, for example, help us understand how deeply a user goes into a site, and what is the typical reason for leaving the site.
- **Unstructured**
    - **Texts** can be a rich source of insights, starting from overall **sentiment score**, up to extracting keywords and even some semantic meaning.
    - **Images** or **Video**. A video from surveillance camera can be used to estimate traffic on the road, and inform people about potential traffic jams.
    - Web server **Logs** can be used to understand which pages of our site are most visited, and for how long.
- Semi-structured
    - **Social Network** graph can be a great source of data about user personality and potential effectiveness in spreading information around.
    - When we have a bunch of photographs from a party, we can try to extract **Group Dynamics** data by building a graph of people taking pictures with each other.

By knowing different possible sources of data, you can try to think about different scenarios where data science techniques can be applied to know the situation better, and to improve business processes.

What you can do with Data

In Data Science, we focus on the following steps of data journey:

1) Data Acquisition

    First step is to collect the data. While in many cases it can be a straightforward process, like data coming to a database from web application, sometimes we need to use special techniques. For example, data from IoT sensors can be overwhelming, and it is a good practice to use buffering endpoints such as IoT Hub to collect all the data before further processing.

2) Data Storage

    Storing the data can be challenging, especially if we are talking about big data. When deciding how to store data, it makes sense to anticipate the way you would want later on to query them. There are several ways data can be stored:

    - Relational database stores a collection of tables, and uses a special language called SQL to query them. Typically, tables would be connected to each other using some schema. In many cases we need to convert the data from original form to fit the schema.
    - NoSQL database, such as CosmosDB, does not enforce schema on data, and allows storing more complex data, for example, hierarchical JSON documents or graphs. However, NoSQL database does not have rich querying capabilities of SQL, and cannot enforce referential integrity between data.

- Data Lake storage is used for large collections of data in raw form. Data lakes are often used with big data, where all data cannot fit into one machine, and has to be stored and processed by a cluster. Parquet is the data format that is often used in conjunction with big data.

3) Data Processing

   This is the most exciting part of data journey, which involved processing the data from its original form to the form that can be used for visualization/model training. When dealing with unstructured data such as text or images, we may need to use some AI techniques to extract **features** from the data, thus converting it to structured form.

4) Visualization / Human Insights

   Often to understand the data we need to visualize them. Having many different visualization techniques in our toolbox we can find the right view to make an insight. Often, data scientist needs to "play with data", visualizing it many times and looking for some relationships. Also, we may use techniques from statistics to test some hypotheses or prove correlation between different pieces of data.

5) Training predictive model

   Because the ultimate goal of data science is to be able to take decisions based on data, we may want to use the techniques of Machine Learning to build predictive model that will be able to solve our problem.

Of course, depending on the actual data some steps might be missing (eg., when we already have the data in the database, or when we do not need model training), or some steps might be repeated several times (such as data processing).

Digitalization and Digital Transformation

In the last decade, many businesses started to understand the importance of data when making business decisions. To apply data science principles to running a business one first needs to collect some data, i.e. somehow turn business processes into digital form. This is known as **digitalization**, and followed by using data science techniques to guide decisions it often leads to significant increase of productivity (or even business pivot), called **digital transformation**.

Let's consider an example. Suppose, we have a data science course (like this one), which we deliver online to students, and we want to use data science to improve it. How can we do it?

We can start with thinking "what can be digitized?". The simplest way would be to measure time it takes each student to complete each module, and the obtained knowledge (eg. by giving multiple-choice test at the end of each module). By averaging time-to-complete across all students, we can find out which modules cause the most problems to students, and work on simplifying them.

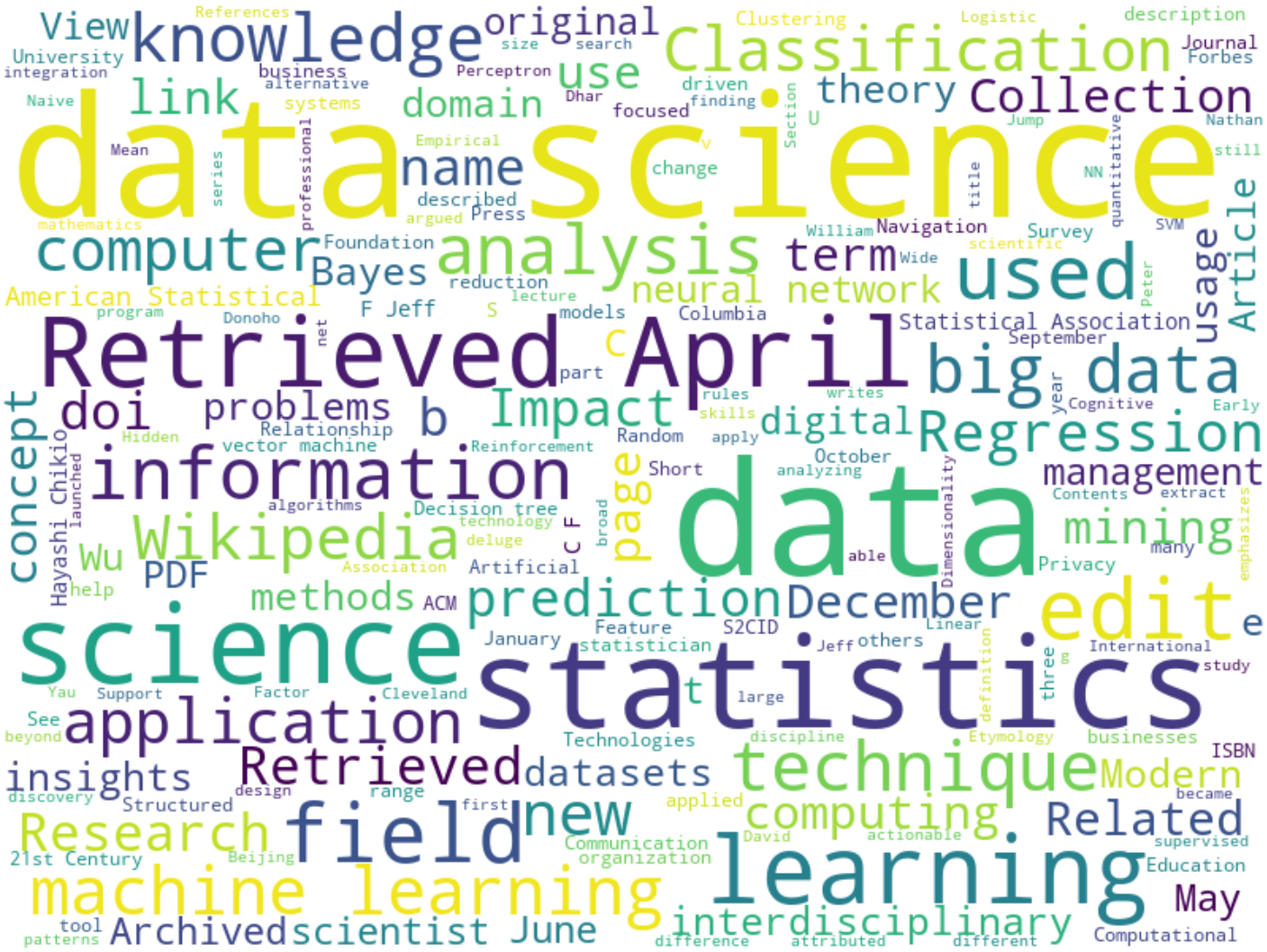
> You may argue that this approach is not ideal, because modules can be of different length. It is probably more fair to divide the time by the length of the module (in number of characters), and compare those values instead.

When we start analyzing results of multiple-choice tests, we can try to find out specific concepts that students understand poorly, and improve the content. To do that, we need to design tests in such a way that each question maps to a certain concept or chunk of knowledge.

If we want to get even more complicated, we can plot the time taken for each module against the age category of students. We might find out that for some age categories it takes inappropriately long time to complete the module, or students drop out at certain point. This can help us provide age recommendation for the module, and minimize people's dissatisfaction from wrong expectations.

🚀 Challenge

In this challenge, we will try to find concepts relevant to the field of Data Science by looking at texts. We will take Wikipedia article on Data Science, download and process the text, and then build a word cloud like this one:



Visit `notebook.ipynb` to read through the code. You can also run the code, and see how it performs all data transformations in real time.

> If you do not know how to run code in Jupyter Notebook, have a look at this article.

Post-lecture quiz

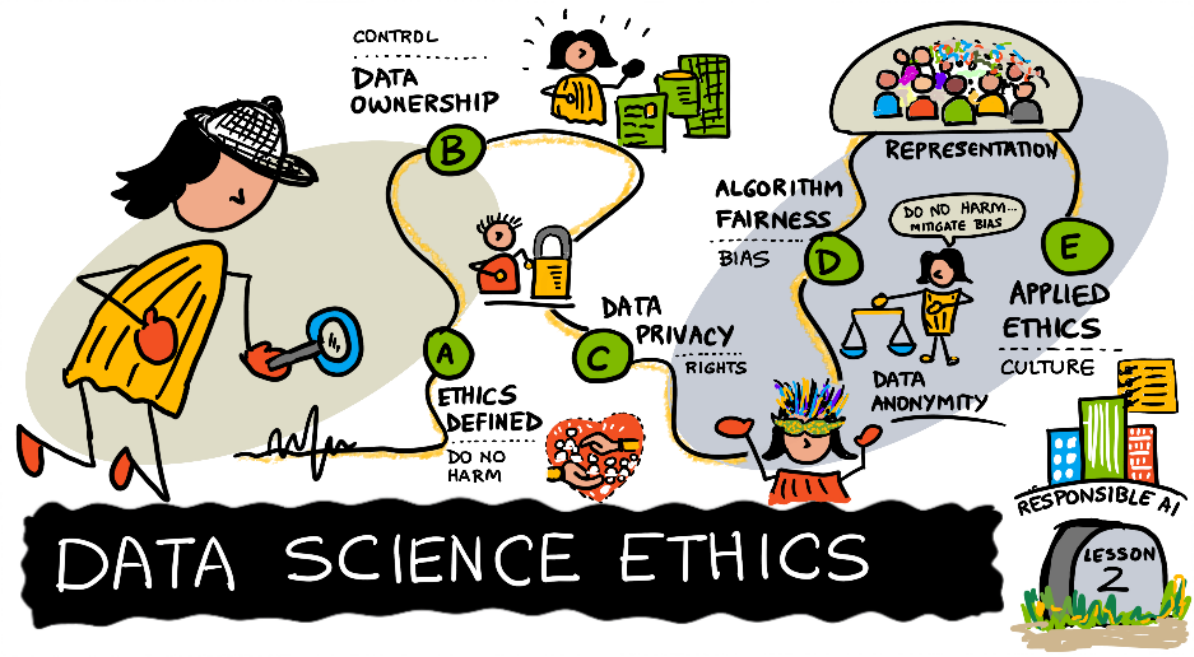
Assignments

- **Task 1**: Modify the code above to find out related concepts for the fields of **Big Data** and **Machine Learning**
- **Task 2**: Think About Data Science Scenarios

Credits

This lesson has been authored with ♥ by Dmitry Soshnikov

Introduction to Data Ethics



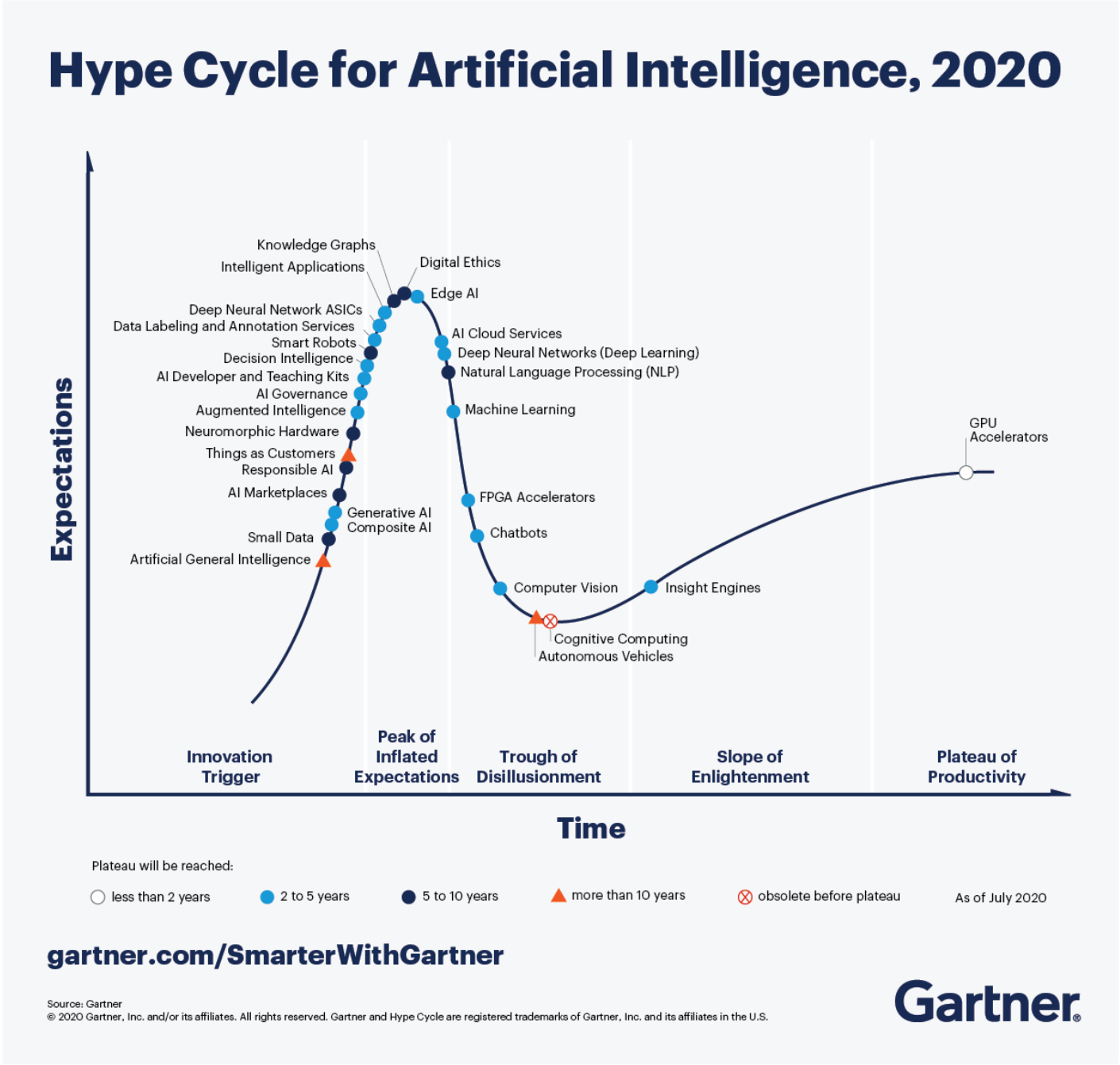
Data Science Ethics - *Sketchnote by @nitya*

We are all data citizens living in a datafied world.

Market trends tell us that by 2022, 1-in-3 large organizations will buy and sell their data through online Marketplaces and Exchanges. As **App Developers**, we'll find it easier and cheaper to integrate data-driven insights and algorithm-driven automation into daily user experiences. But as AI becomes pervasive, we'll also need to understand the potential harms caused by the weaponization of such algorithms at scale.

Trends also indicate that we will create and consume over 180 zettabytes of data by 2025. As **Data Scientists**, this gives us unprecedented levels of access to personal data. This means we can build behavioral profiles of users and influence decision-making in ways that create an illusion of free choice while potentially nudging users towards outcomes we prefer. It also raises broader questions on data privacy and user protections.

Data ethics are now *necessary guardrails* for data science and engineering, helping us minimize potential harms and unintended consequences from our data-driven actions. The Gartner Hype Cycle for AI identifies relevant trends in digital ethics, responsible AI ,and AI governances as key drivers for larger megatrends around *democratization* and *industrialization* of AI.



In this lesson, we'll explore the fascinating area of data ethics - from core concepts and challenges, to case studies and applied AI concepts like governance - that help establish an ethics culture in teams and organizations that work with data and AI.

Pre-lecture quiz 🎯

Basic Definitions

Let's start by understanding the basic terminology.

The word "ethics" comes from the Greek word "ethikos" (and its root "ethos") meaning *character or moral nature*.

Ethics is about the shared values and moral principles that govern our behavior in society. Ethics is based not on laws but on widely accepted norms of what is "right vs. wrong". However, ethical considerations can influence corporate governance initiatives and government regulations that create more incentives for compliance.

Data Ethics is a new branch of ethics that "studies and evaluates moral problems related to *data, algorithms and corresponding practices*". Here, **"data"** focuses on actions related to generation, recording, curation, processing dissemination, sharing ,and usage, **"algorithms"** focuses on AI, agents, machine learning ,and robots, and **"practices"** focuses on topics like responsible innovation, programming, hacking and ethics codes.

Applied Ethics is the practical application of moral considerations. It's the process of actively investigating ethical issues in the context of *real-world actions, products and processes*, and taking corrective measures to make that these remain aligned with our defined ethical values.

Ethics Culture is about *operationalizing* applied ethics to make sure that our ethical principles and practices are adopted in a consistent and scalable manner across the entire organization. Successful ethics cultures define organization-wide ethical principles, provide meaningful incentives for compliance, and reinforce ethics norms by encouraging and amplifying desired behaviors at every level of the organization.

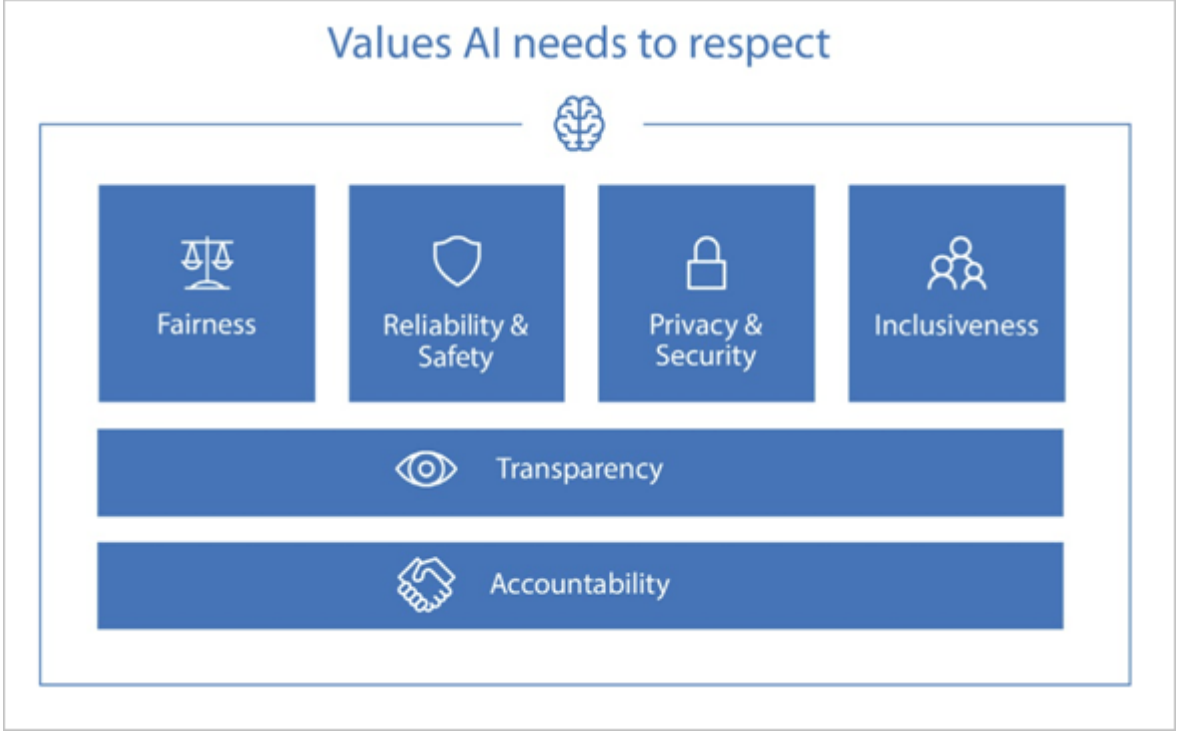
Ethics Concepts

In this section, we'll discuss concepts like **shared values** (principles) and **ethical challenges** (problems) for data ethics - and explore **case studies** that help you understand these concepts in real-world contexts.

1. Ethics Principles

Every data ethics strategy begins by defining *ethical principles* - the "shared values" that describe acceptable behaviors, and guide compliant actions, in our data & AI projects. You can define these at an individual or team level. However, most large organizations outline these in an *ethical AI* mission statement or framework that is defined at corporate levels and enforced consistently across all teams.

Example: Microsoft's Responsible AI mission statement reads: *"We are committed to the advancement of AI-driven by ethical principles that put people first"* - identifying 6 ethical principles in the framework below:



Let's briefly explore these principles. *Transparency* and *accountability* are foundational values that other principles built upon - so let's begin there:

- **Accountability** makes practitioners *responsible* for their data & AI operations, and compliance with these ethical principles.
- **Transparency** ensures that data and AI actions are *understandable* (interpretable) to users, explaining the what and why behind decisions.
- **Fairness** - focuses on ensuring AI treats *all people* fairly, addressing any systemic or implicit socio-technical biases in data and systems.
- **Reliability & Safety** - ensures that AI behaves *consistently* with defined values, minimizing potential harms or unintended consequences.
- **Privacy & Security** - is about understanding data lineage, and providing *data privacy and related protections* to users.
- **Inclusiveness** - is about designing AI solutions with intention, adapting them to meet a *broad range of human needs* & capabilities.

> 🚨 Think about what your data ethics mission statement could be. Explore ethical AI frameworks from other organizations - here are examples from IBM, Google ,and Facebook. What shared values do they have in common? How do these principles relate to the AI product or industry they operate in?

2. Ethics Challenges

Once we have ethical principles defined, the next step is to evaluate our data and AI actions to see if they align with those shared values. Think about your actions in two categories: *data collection* and

algorithm design.

With data collection, actions will likely involve **personal data** or personally identifiable information (PII) for identifiable living individuals. This includes diverse items of non-personal data that *collectively* identify an individual. Ethical challenges can relate to *data privacy*, *data ownership*, and related topics like *informed consent* and *intellectual property rights* for users.

With algorithm design, actions will involve collecting & curating **datasets**, then using them to train & deploy **data models** that predict outcomes or automate decisions in real-world contexts. Ethical challenges can arise from *dataset bias*, *data quality* issues, *unfairness* ,and *misrepresentation* in algorithms - including some issues that are systemic in nature.

In both cases, ethics challenges highlight areas where our actions may encounter conflict with our shared values. To detect, mitigate, minimize, or eliminate, these concerns - we need to ask moral "yes/no" questions related to our actions, then take corrective actions as needed. Let's take a look at some ethical challenges and the moral questions they raise:

2.1 Data Ownership

Data collection often involves personal data that can identify the data subjects. Data ownership is about *control* and *user rights* related to the creation, processing ,and dissemination of data.

The moral questions we need to ask are:

- Who owns the data? (user or organization)
- What rights do data subjects have? (ex: access, erasure, portability)
- What rights do organizations have? (ex: rectify malicious user reviews)

2.2 Informed Consent

Informed consent defines the act of users agreeing to an action (like data collection) with a *full understanding* of relevant facts including the purpose, potential risks ,and alternatives.

Questions to explore here are:

- Did the user (data subject) give permission for data capture and usage?
- Did the user understand the purpose for which that data was captured?
- Did the user understand the potential risks from their participation?

2.3 Intellectual Property

Intellectual property refers to intangible creations resulting from the human initiative, that may *have economic value* to individuals or businesses.

Questions to explore here are:

- Did the collected data have economic value to a user or business?
- Does the **user** have intellectual property here?
- Does the **organization** have intellectual property here?
- If these rights exist, how are we protecting them?

2.4 Data Privacy

Data privacy or information privacy refers to the preservation of user privacy and protection of user identity with respect to personally identifiable information.

Questions to explore here are:

- Is users' (personal) data secured against hacks and leaks?
- Is users' data accessible only to authorized users and contexts?
- Is users' anonymity preserved when data is shared or disseminated?
- Can a user be de-identified from anonymized datasets?

2.5 Right To Be Forgotten

The Right To Be Forgotten or Right to Erasure provides additional personal data protection to users. Specifically, it gives users the right to request deletion or removal of personal data from Internet searches and other locations, *under specific circumstances* - allowing them a fresh start online without past actions being held against them.

Questions to explore here are:

- Does the system allow data subjects to request erasure?
- Should the withdrawal of user consent trigger automated erasure?
- Was data collected without consent or by unlawful means?
- Are we compliant with government regulations for data privacy?

2.6 Dataset Bias

Dataset or Collection Bias is about selecting a *non-representative* subset of data for algorithm development, creating potential unfairness in result outcomes for diverse groups. Types of bias include selection or sampling bias, volunteer bias, and instrument bias.

Questions to explore here are:

- Did we recruit a representative set of data subjects?
- Did we test our collected or curated dataset for various biases?
- Can we mitigate or remove any discovered biases?

2.7 Data Quality

Data Quality looks at the validity of the curated dataset used to develop our algorithms, checking to see if features and records meet requirements for the level of accuracy and consistency needed for our AI purpose.

Questions to explore here are:

- Did we capture valid *features* for our use case?
- Was data captured *consistently* across diverse data sources?
- Is the dataset *complete* for diverse conditions or scenarios?
- Is information captured *accurately* in reflecting reality?

2.8 Algorithm Fairness

Algorithm Fairness checks to see if the algorithm design systematically discriminates against specific subgroups of data subjects leading to potential harms in *allocation* (where resources are denied or withheld from that group) and *quality of service* (where AI is not as accurate for some subgroups as it is for others).

Questions to explore here are:

- Did we evaluate model accuracy for diverse subgroups and conditions?
- Did we scrutinize the system for potential harms (e.g., stereotyping)?
- Can we revise data or retrain models to mitigate identified harms?

Explore resources like AI Fairness checklists to learn more.

2.9 Misrepresentation

Data Misrepresentation is about asking whether we are communicating insights from honestly reported data in a deceptive manner to support a desired narrative.

Questions to explore here are:

- Are we reporting incomplete or inaccurate data?
- Are we visualizing data in a manner that drives misleading conclusions?
- Are we using selective statistical techniques to manipulate outcomes?
- Are there alternative explanations that may offer a different conclusion?

2.10 Free Choice

The Illusion of Free Choice occurs when system "choice architectures" use decision-making algorithms to nudge people towards taking a preferred outcome while seeming to give them options

and control. These dark patterns can cause social and economic harm to users. Because user decisions impact behavior profiles, these actions potentially drive future choices that can amplify or extend the impact of these harms.

Questions to explore here are:

- Did the user understand the implications of making that choice?
- Was the user aware of (alternative) choices and the pros & cons of each?
- Can the user reverse an automated or influenced choice later?

3. Case Studies

To put these ethical challenges in real-world contexts, it helps to look at case studies that highlight the potential harms and consequences to individuals and society, when such ethics violations are overlooked.

Here are a few examples:

Ethics Challenge	Case Study
Informed Consent	1972 - Tuskegee Syphillis Study - African American men who participated in the study were promised free medical care *but deceived* by researchers who failed to inform subjects of their diagnosis or about availability of treatment. Many subjects died & partners or children were affected; the study lasted 40 years.
Data Privacy	2007 - The Netflix data prize provided researchers with *10M anonymized movie rankings from 50K customers* to help improve recommendation algorithms. However, researchers were able to correlate anonymized data with personally-identifiable data in *external datasets* (e.g., IMDb comments) - effectively "de-anonymizing" some Netflix subscribers.
Collection Bias	2013 - The City of Boston developed Street Bump, an app that let citizens report potholes, giving the city better roadway data to find and fix issues. However, people in lower income groups had less access to cars and phones, making their roadway issues invisible in this app. Developers worked with academics to *equitable access and digital divides* issues for fairness.
Algorithmic Fairness	2018 - The MIT Gender Shades Study evaluated the accuracy of gender classification AI products, exposing gaps in accuracy for women and persons of color. A 2019 Apple Card seemed to offer less credit to women than men. Both illustrated issues in algorithmic bias leading to socio-economic harms.

Ethics Challenge	Case Study
Data Misrepresentation	2020 - The Georgia Department of Public Health released COVID-19 charts that appeared to mislead citizens about trends in confirmed cases with non-chronological ordering on the x-axis. This illustrates misrepresentation through visualization tricks.
Illusion of free choice	2020 - Learning app ABCmouse paid $10M to settle an FTC complaint where parents were trapped into paying for subscriptions they couldn't cancel. This illustrates dark patterns in choice architectures, where users were nudged towards potentially harmful choices.
Data Privacy & User Rights	2021 - Facebook Data Breach exposed data from 530M users, resulting in a $5B settlement to the FTC. It however refused to notify users of the breach violating user rights around data transparency and access.

Want to explore more case studies? Check out these resources:

- Ethics Unwrapped - ethics dilemmas across diverse industries.
- Data Science Ethics course - landmark case studies explored.
- Where things have gone wrong - deon checklist with examples

> 🚨 Think about the case studies you've seen - have you experienced, or been affected by, a similar ethical challenge in your life? Can you think of at least one other case study that illustrates one of the ethical challenges we've discussed in this section?

Applied Ethics

We've talked about ethics concepts, challenges ,and case studies in real-world contexts. But how do we get started *applying* ethical principles and practices in our projects? And how do we *operationalize* these practices for better governance? Let's explore some real-world solutions:

1. Professional Codes

Professional Codes offer one option for organizations to "incentivize" members to support their ethical principles and mission statement. Codes are *moral guidelines* for professional behavior, helping employees or members make decisions that align with their organization's principles. They

are only as good as the voluntary compliance from members; however, many organizations offer additional rewards and penalties to motivate compliance from members.

Examples include:

- Oxford Munich Code of Ethics
- Data Science Association Code of Conduct (created 2013)
- ACM Code of Ethics and Professional Conduct (since 1993)

> 🚨 Do you belong to a professional engineering or data science organization? Explore their site to see if they define a professional code of ethics. What does this say about their ethical principles? How are they "incentivizing" members to follow the code?

2. Ethics Checklists

While professional codes define required *ethical behavior* from practitioners, they have known limitations in enforcement, particularly in large-scale projects. Instead, many data Science experts advocate for checklists, that can **connect principles to practices** in more deterministic and actionable ways.

Checklists convert questions into "yes/no" tasks that can be operationalized, allowing them to be tracked as part of standard product release workflows.

Examples include:

- Deon - a general-purpose data ethics checklist created from industry recommendations with a command-line tool for easy integration.
- Privacy Audit Checklist - provides general guidance for information handling practices from legal and social exposure perspectives.
- AI Fairness Checklist - created by AI practitioners to support the adoption and integration of fairness checks into AI development cycles.
- 22 questions for ethics in data and AI - more open-ended framework, structured for initial exploration of ethical issues in design, implementation, and organizational, contexts.

3. Ethics Regulations

Ethics is about defining shared values and doing the right thing *voluntarily*. **Compliance** is about *following the law* if and where defined. **Governance** broadly covers all the ways in which organizations operate to enforce ethical principles and comply with established laws.

Today, governance takes two forms within organizations. First, it's about defining **ethical AI** principles and establishing practices to operationalize adoption across all AI-related projects in the organization. Second, it's about complying with all government-mandated **data protection regulations** for regions it operates in.

Examples of data protection and privacy regulations:

- `1974` , US Privacy Act - regulates *federal govt.* collection, use ,and disclosure of personal information.
- `1996` , US Health Insurance Portability & Accountability Act (HIPAA) - protects personal health data.
- `1998` , US Children's Online Privacy Protection Act (COPPA) - protects data privacy of children under 13.
- `2018` , General Data Protection Regulation (GDPR) - provides user rights, data protection ,and privacy.
- `2018` , California Consumer Privacy Act (CCPA) gives consumers more *rights* over their (personal) data.
- `2021` , China's Personal Information Protection Law just passed, creating one of the strongest online data privacy regulations worldwide.

> 🚨 The European Union defined GDPR (General Data Protection Regulation) remains one of the most influential data privacy regulations today. Did you know it also defines 8 user rights to protect citizens' digital privacy and personal data? Learn about what these are, and why they matter.

4. Ethics Culture

Note that there remains an intangible gap between *compliance* (doing enough to meet "the letter of the law") and addressing systemic issues (like ossification, information asymmetry ,and distributional unfairness) that can speed up the weaponization of AI.

The latter requires collaborative approaches to defining ethics cultures that build emotional connections and consistent shared values *across organizations* in the industry. This calls for more formalized data ethics cultures in organizations - allowing *anyone* to pull the Andon cord (to raise ethics concerns early in the process) and making *ethical assessments* (e.g., in hiring) a core criteria team formation in AI projects.

Challenge 🚀

Post-lecture quiz 🎯

Review & Self Study

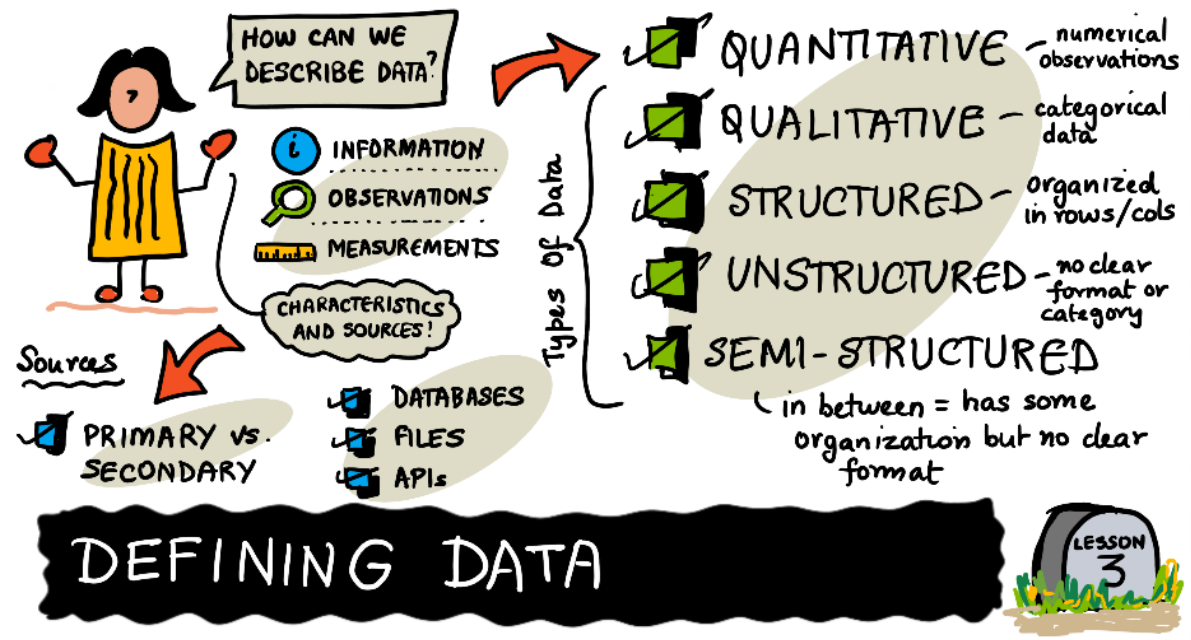
Courses and books help with understanding core ethics concepts and challenges, while case studies and tools help with applied ethics practices in real-world contexts. Here are a few resources to start with.

- Machine Learning For Beginners - lesson on Fairness, from Microsoft.
- Principles of Responsible AI - free learning path from Microsoft Learn.
- Ethics and Data Science - O'Reilly EBook (M. Loukides, H. Mason et. al)
- Data Science Ethics - online course from the University of Michigan.
- Ethics Unwrapped - case studies from the University of Texas.

Assignment

Assignment Title

Defining Data

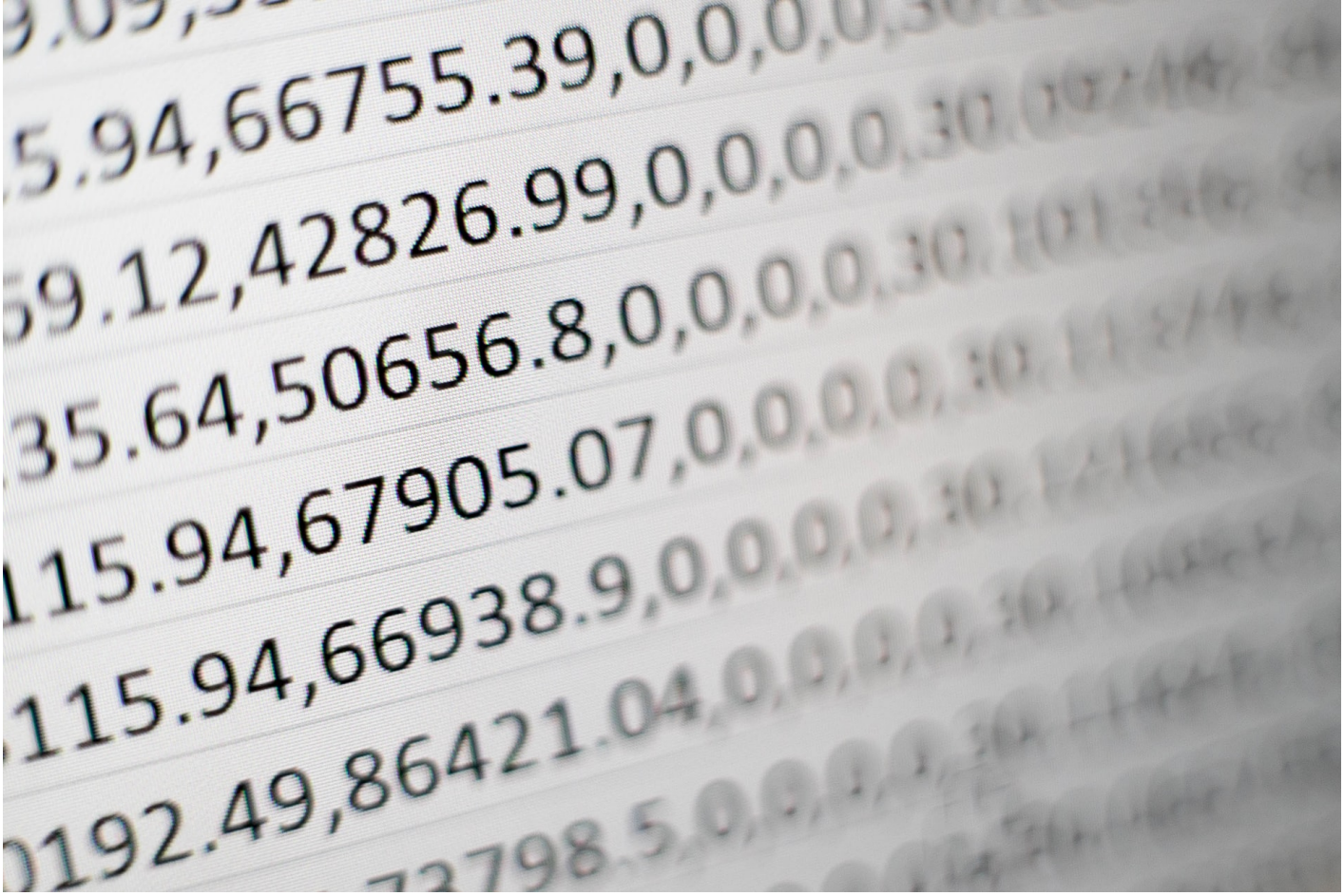


Defining Data - *Sketchnote by @nitya*

Data is facts, information, observations and measurements that are used to make discoveries and to support informed decisions. A data point is a single unit of data with in a dataset, which is collection of data points. Datasets may come in different formats and structures, and will usually be based on its source, or where the data came from. For example, a company's monthly earnings might be in a spreadsheet but hourly heart rate data from a smartwatch may be in JSON format. It's common for data scientists to work with different types of data within a dataset.

This lesson focuses on identifying and classifying data by its characteristics and its sources.

Pre-Lecture Quiz



> Source: Mika Baumister via Unsplash

How Data is Described

Raw data is data that has come from its source in its initial state and has not been analyzed or organized. In order to make sense of what is happening with a dataset, it needs to be organized into a format that can be understood by humans as well as the technology they may use to analyze it further. The structure of a dataset describes how it's organized and can be classified at structured, unstructured and semi-structured. These types of structure will vary, depending on the source but will ultimately fit in these three categories.

Quantitative Data

Quantitative data is numerical observations within a dataset and can typically be analyzed, measured and used mathematically. Some examples of quantitative data are: a country's population, a person's height or a company's quarterly earnings. With some additional analysis, quantitative data could be used to discover seasonal trends of the Air Quality Index (AQI) or estimate the probability of rush hour traffic on a typical work day.

Qualitative Data

Qualitative data, also known as categorical data is data that cannot be measured objectively like observations of quantitative data. It's generally various formats of subjective data that captures the quality of something, such as a product or process. Sometimes, qualitative data is numerical and wouldn't be typically used mathematically, like phone numbers or timestamps. Some examples of qualitative data are: video comments, the make and model of a car or your closest friends' favorite color. Qualitative data could be used to understand which products consumers like best or identifying popular keywords in job application resumes.

Structured Data

Structured data is data that is organized into rows and columns, where each row will have the same set of columns. Columns represent a value of a particular type and will be identified with a name describing what the value represents, while rows contain the actual values. Columns will often have a specific set of rules or restrictions on the values, to ensure that the values accurately represent the column. For example imagine a spreadsheet of customers where each row must have a phone number and the phone numbers never contain alphabetical characters. There may be rules applied on the phone number column to make sure it's never empty and only contains numbers.

A benefit of structured data is that it can be organized in such a way that it can be related to other structured data. However, because the data is designed to be organized in a specific way, making changes to its overall structure can take a lot of effort to do. For example, adding an email column to the customer spreadsheet that cannot be empty means you'll need figure out how you'll add these values to the existing rows of customers in the dataset.

Examples of structured data: spreadsheets, relational databases, phone numbers, bank statements

Unstructured Data

Unstructured data typically cannot be categorized into into rows or columns and doesn't contain a format or set of rules to follow. Because unstructured data has less restrictions on its structure it's easier to add new information in comparison to a structured dataset. If a sensor capturing data on barometric pressure every 2 minutes has received an update that now allows it to measure and record temperature, it doesn't require altering the existing data if it's unstructured. However, this may make analyzing or investigating this type of data take longer. For example, a scientist who wants to find the average temperature of the previous month from the sensors data, but discovers that the sensor recorded an "e" in some of its recorded data to note that it was broken instead of a typical number, which means the data is incomplete.

Examples of unstructured data: text files, text messages, video files

Semi-structured

Semi-structured data has features that make it a combination of structured and unstructured data. It doesn't typically conform to a format of rows and columns but is organized in a way that is considered structured and may follow a fixed format or set of rules. The structure will vary between sources, such as a well defined hierarchy to something more flexible that allows for easy integration of new information. Metadata are indicators that help decide how the data is organized and stored and will have various names, based on the type of data. Some common names for metadata are tags, elements, entities and attributes. For example, a typical email message will have a subject, body and a set of recipients and can be organized by whom or when it was sent.

Examples of unstructured data: HTML, CSV files, JavaScript Object Notation (JSON)

Sources of Data

A data source is the initial location of where the data was generated, or where it "lives" and will vary based on how and when it was collected. Data generated by its user(s) are known as primary data while secondary data comes from a source that has collected data for general use. For example, a group of scientists collecting observations in a rainforest would be considered primary and if they decide to share it with other scientists it would be considered secondary to those that use it.

Databases are a common source and rely on a database management system to host and maintain the data where users use commands called queries to explore the data. Files as data sources can be audio, image, and video files as well as spreadsheets like Excel. Internet sources are a common location for hosting data, where databases as well as files can be found. Application programming interfaces, also known as APIs allow programmers to create ways to share data with external users through the internet, while the process of web scraping extracts data from a web page. The lessons in Working with Data focuses on how to use various data sources.

Conclusion

In this lesson we have learned:

- What data is
- How data is described
- How data is classified and categorized
- Where data can be found

🚀 Challenge

Kaggle is an excellent source of open datasets. Use the dataset search tool to find some interesting datasets and classify 3-5 datasets with this criteria:

- Is the data quantitative or qualitative?
- Is the data structured, unstructured, or semi-structured?

Post-Lecture Quiz

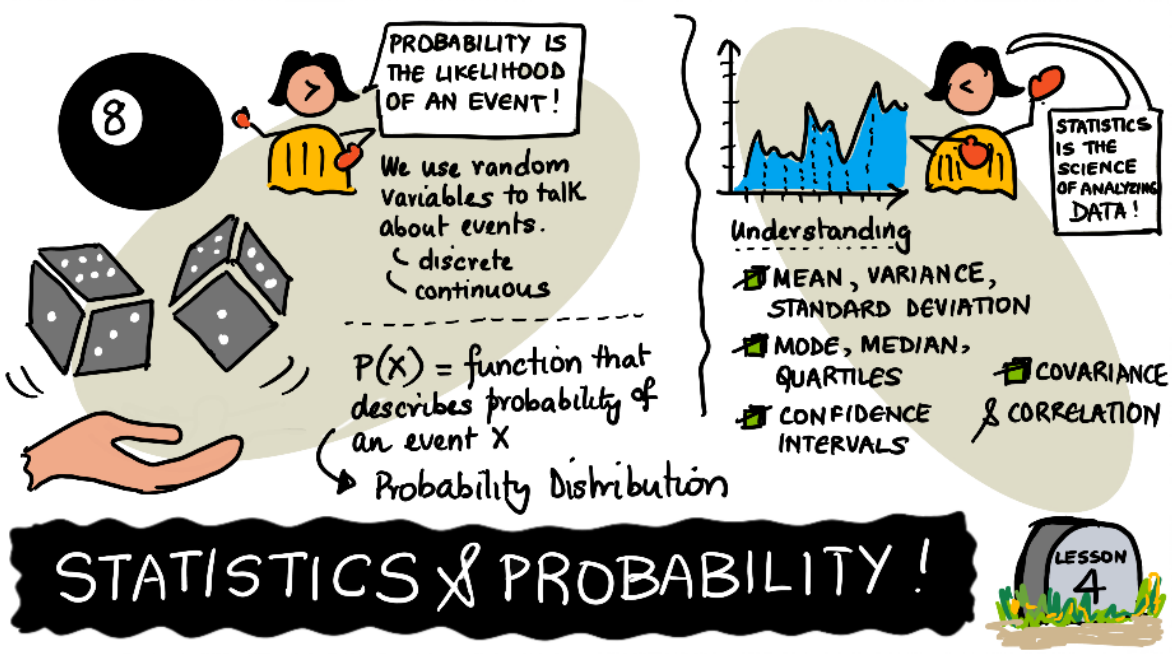
Review & Self Study

- This Microsoft Learn unit, titled Classify your Data has a detailed breakdown of structured, semi-structured, and unstructured data.

Assignment

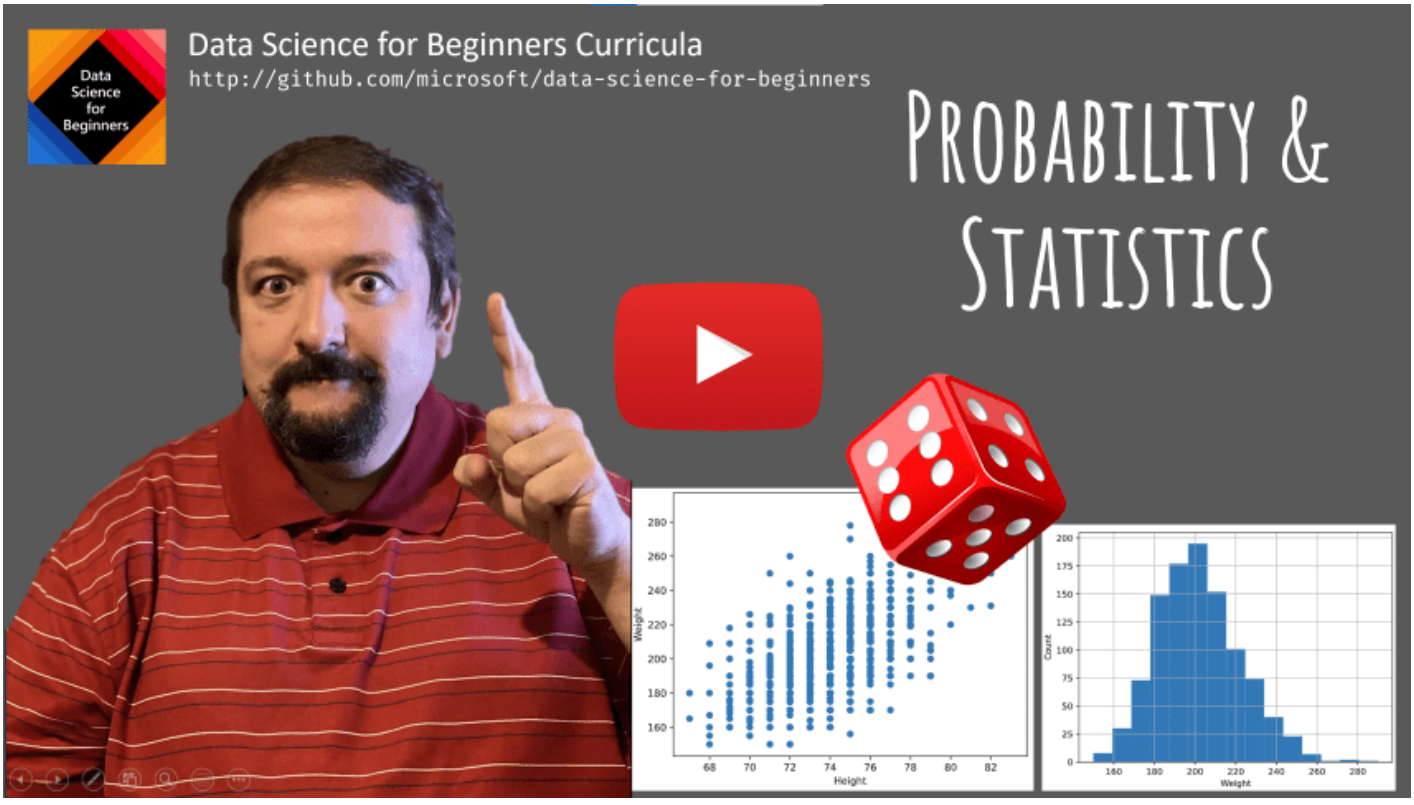
Classifying Datasets

A Brief Introduction to Statistics and Probability



Statistics and Probability - *Sketchnote by @nitya*

Statistics and Probability Theory are two highly related areas of Mathematics that are highly relevant to Data Science. It is possible to operate with data without deep knowledge of mathematics, but it is still better to know at least some basic concepts. Here we will present a short introduction that will help you get started.



Pre-lecture quiz

Probability and Random Variables

Probability is a number between 0 and 1 that expresses how probable an **event** is. It is defined as a number of positive outcomes (that lead to the event), divided by total number of outcomes, given that all outcomes are equally probable. For example, when we roll a dice, the probability that we get an even number is 3/6 = 0.5.

When we talk about events, we use **random variables**. For example, the random variable that represents a number obtained when rolling a dice would take values from 1 to 6. Set of numbers from 1 to 6 is called **sample space**. We can talk about the probability of a random variable taking a certain value, for example P(X=3)=1/6.

The random variable in previous example is called **discrete**, because it has a countable sample space, i.e. there are separate values that can be enumerated. There are cases when sample space is a range of real numbers, or the whole set of real numbers. Such variables are called **continuous**. A good example is the time when the bus arrives.

Probability Distribution

In the case of discrete random variables, it is easy to describe the probability of each event by a function P(X). For each value *s* from sample space *S* it will give a number from 0 to 1, such that the sum of all values of P(X=s) for all events would be 1.

The most well-known discrete distribution is **uniform distribution**, in which there is a sample space of N elements, with equal probability of 1/N for each of them.

It is more difficult to describe the probability distribution of a continuous variable, with values drawn from some interval [a,b], or the whole set of real numbers $\mathbb{R}$. Consider the case of bus arrival time. In fact, for each exact arrival time t, the probability of a bus arriving at exactly that time is 0!

> Now you know that events with 0 probability happen, and very often! At least each time when the bus arrives!

We can only talk about the probability of a variable falling in a given interval of values, eg. $P(t_1 \le X < t_2)$. In this case, probability distribution is described by a **probability density function** p(x), such that

$P(t_1 \le X < t_2) = \int_{t_1}^{t_2} p(x)dx$ A continuous analog of uniform distribution is called **continuous uniform**, which is defined on a finite interval. A probability that the value X falls into an interval of length l is proportional to l, and rises up to 1.

Another important distribution is **normal distribution**, which we will talk about in more detail below.

Mean, Variance and Standard Deviation

Suppose we draw a sequence of n samples of a random variable X: x_1, x_2, ..., x_n. We can define **mean** (or **arithmetic average**) value of the sequence in the traditional way as $(x_1+x_2+x_n)/n$. As we grow the size of the sample (i.e. take the limit with $n\to\infty$), we will obtain the mean (also called **expectation**) of the distribution. We will denote expectation by **E**(x).

> It can be demonstrated that for any discrete distribution with values $\{x_1, x_2, ..., x_N\}$ and corresponding probabilities p_1, p_2, ..., p_N, the expectation would equal to $E(X)=x_1p_1+x_2p_2+...+x_Np_N$.

To identify how far the values are spread, we can compute the variance $\sigma^2 = \sum(x_i - \mu)^2/n$, where μ is the mean of the sequence. The value σ is called **standard deviation**, and σ^2 is called a **variance**.

Mode, Median and Quartiles

Sometimes, mean does not adequately represent the "typical" value for data. For example, when there are a few extreme values that are completely out of range, they can affect the mean. Another good indication is a **median**, a value such that half of data points are lower than it, and another half - higher.

To help us understand the distribution of data, it is helpful to talk about **quartiles**:

- First quartile, or Q1, is a value, such that 25% of the data fall below it
- Third quartile, or Q3, is a value that 75% of the data fall below it

Graphically we can represent relationship between median and quartiles in a diagram called the **box plot**:

Here we also compute **inter-quartile range** IQR=Q3-Q1, and so-called **outliers** - values, that lie outside the boundaries [Q1-1.5*IQR,Q3+1.5*IQR].

For finite distribution that contains a small number of possible values, a good "typical" value is the one that appears the most frequently, which is called **mode**. It is often applied to categorical data, such as colors. Consider a situation when we have two groups of people - some that strongly prefer red, and others who prefer blue. If we code colors by numbers, the mean value for a favorite color would be somewhere in the orange-green spectrum, which does not indicate the actual preference on neither group. However, the mode would be either one of the colors, or both colors, if the number of people voting for them is equal (in this case we call the sample **multimodal**).

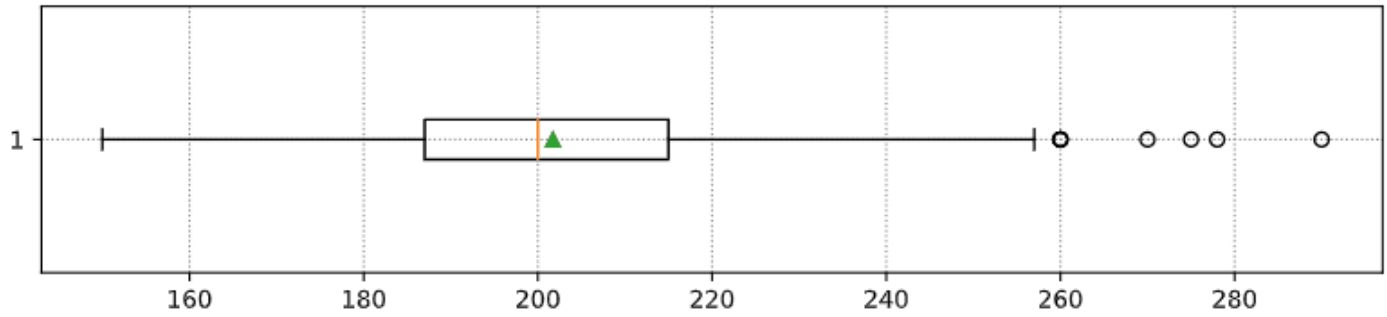
Real-world Data

When we analyze data from real life, they often are not random variables as such, in a sense that we do not perform experiments with unknown result. For example, consider a team of baseball players, and their body data, such as height, weight and age. Those numbers are not exactly random, but we can still apply the same mathematical concepts. For example, a sequence of people's weights can be considered to be a sequence of values drawn from some random variable. Below is the sequence of weights of actual baseball players from Major League Baseball, taken from this dataset (for your convenience, only first 20 values are shown):

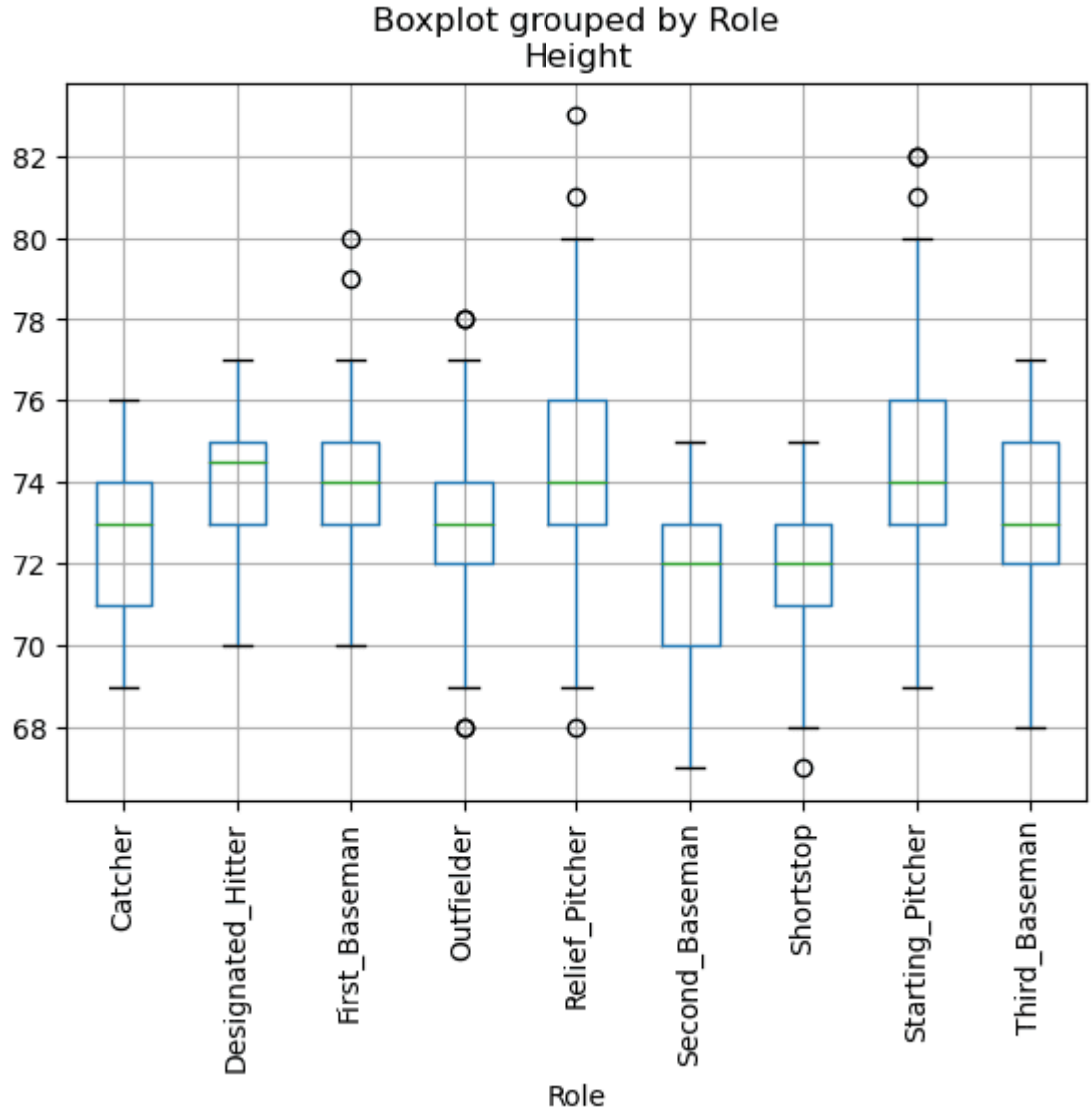
```
[180.0, 215.0, 210.0, 210.0, 188.0, 176.0, 209.0, 200.0, 231.0, 180.0, 188
```

> **Note**: To see the example of working with this dataset, have a look at the accompanying notebook. There are also a number of challenges throughout this lesson, and you may complete them by adding some code to that notebook. If you are not sure how to operate on data, do not worry - we will come back to working with data using Python at a later time. If you do not know how to run code in Jupyter Notebook, have a look at this article.

Here is the box plot showing mean, median and quartiles for our data:



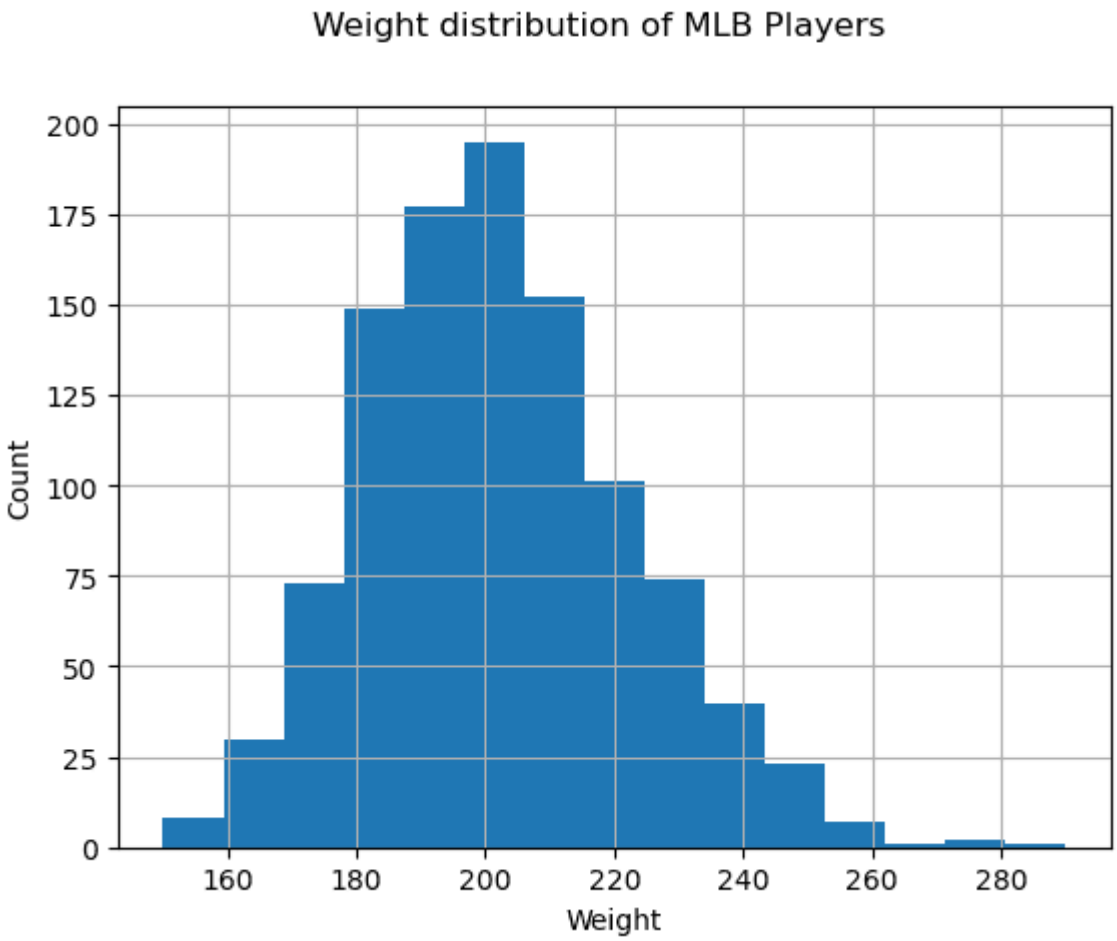
Since our data contains information about different player **roles**, we can also do the box plot by role - it will allow us to get the idea on how parameters values differ across roles. This time we will consider height:



This diagram suggests that, on average, height of first basemen is higher that height of second basemen. Later in this lesson we will learn how we can test this hypothesis more formally, and how to demonstrate that our data is statistically significant to show that.

> When working with real-world data, we assume that all data points are samples drawn from some probability distribution. This assumption allows us to apply machine learning techniques and build working predictive models.

To see what the distribution of our data is, we can plot a graph called a **histogram**. X-axis would contain a number of different weight intervals (so-called **bins**), and the vertical axis would show the number of times our random variable sample was inside a given interval.



From this histogram you can see that all values are centered around certain mean weight, and the further we go from that weight - the fewer weights of that value are encountered. I.e., it is very improbable that the weight of a baseball player would be very different from the mean weight. Variance of weights show the extent to which weights are likely to differ from the mean.

> If we take weights of other people, not from the baseball league, the distribution is likely to be different. However, the shape of the distribution will be the same, but mean and variance would change. So, if we train our model on baseball players, it is likely to give wrong results when applied to students of a university, because the underlying distribution is different.

Normal Distribution

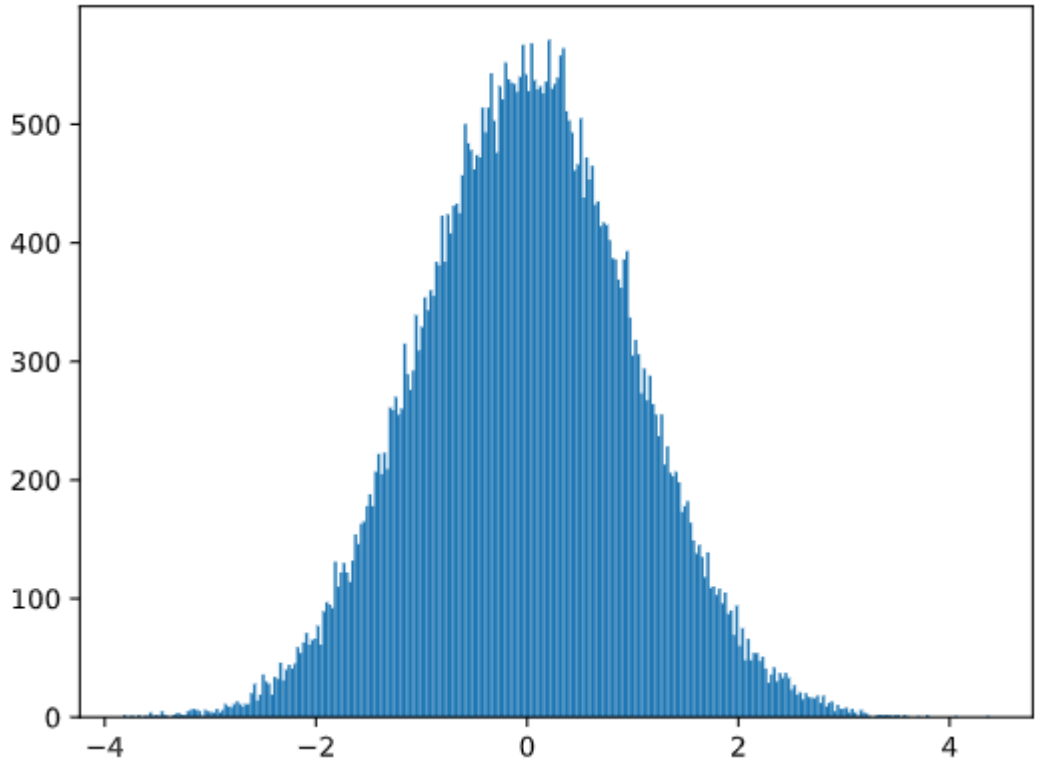
The distribution of weights that we have seen above is very typical, and many measurements from real world follow the same type of distribution, but with different mean and variance. This distribution

is called **normal distribution**, and it plays a very important role in statistics.

Using normal distribution is a correct way to generate random weights of potential baseball players. Once we know mean weight `mean` and standard deviation `std`, we can generate 1000 weight samples in the following way:

```python
samples = np.random.normal(mean,std,1000)
```

If we plot the histogram of the generated samples we will see the picture very similar to the one shown above. And if we increase the number of samples and the number of bins, we can generate a picture of a normal distribution that is more close to ideal:



Normal Distribution with mean=0 and std.dev=1

Confidence Intervals

When we talk about weights of baseball players, we assume that there is certain **random variable W** that corresponds to ideal probability distribution of weights of all baseball players (so-called **population**). Our sequence of weights corresponds to a subset of all baseball players that we call **sample**. An interesting question is, can we know the parameters of distribution of W, i.e. mean and variance of the population?

The easiest answer would be to calculate mean and variance of our sample. However, it could happen that our random sample does not accurately represent complete population. Thus it makes sense to

talk about **confidence interval**.

> **Confidence interval** is the estimation of true mean of the population given our sample, which is accurate is a certain probability (or **level of confidence**).

Suppose we have a sample $X_1, ..., X_n$ from our distribution. Each time we draw a sample from our distribution, we would end up with different mean value μ. Thus μ can be considered to be a random variable. A **confidence interval** with confidence p is a pair of values (L_p,R_p), such that $\mathbf{P}(L_p \le \mu \le R_p) = p$, i.e. a probability of measured mean value falling within the interval equals to p.

It does beyond our short intro to discuss in detail how those confidence intervals are calculated. Some more details can be found on Wikipedia. In short, we define the distribution of computed sample mean relative to the true mean of the population, which is called **student distribution**.

> **Interesting fact**: Student distribution is named after mathematician William Sealy Gosset, who published his paper under the pseudonym "Student". He worked in the Guinness brewery, and, according to one of the versions, his employer did not want general public to know that they were using statistical tests to determine the quality of raw materials.

If we want to estimate the mean μ of our population with confidence p, we need to take *(1-p)/2-th percentile* of a Student distribution A, which can either be taken from tables, or computer using some built-in functions of statistical software (eg. Python, R, etc.). Then the interval for μ would be given by $X \pm A*D/\sqrt{n}$, where X is the obtained mean of the sample, D is the standard deviation.

> **Note**: We also omit the discussion of an important concept of degrees of freedom, which is important in relation to Student distribution. You can refer to more complete books on statistics to understand this concept deeper.

An example of calculating confidence interval for weights and heights is given in the accompanying notebooks.

p	Weight mean
0.85	201.73±0.94

p	Weight mean
0.90	201.73±1.08
0.95	201.73±1.28

Notice that the higher is the confidence probability, the wider is the confidence interval.

Hypothesis Testing

In our baseball players dataset, there are different player roles, that can be summarized below (look at the accompanying notebook to see how this table can be calculated):

Role	Height	Weight	Count
Catcher	72.723684	204.328947	76
Designated_Hitter	74.222222	220.888889	18
First_Baseman	74.000000	213.109091	55
Outfielder	73.010309	199.113402	194
Relief_Pitcher	74.374603	203.517460	315
Second_Baseman	71.362069	184.344828	58
Shortstop	71.903846	182.923077	52
Starting_Pitcher	74.719457	205.163636	221
Third_Baseman	73.044444	200.955556	45

We can notice that the mean heights of first basemen is higher than that of second basemen. Thus, we may be tempted to conclude that **first basemen are higher than second basemen**.

> This statement is called **a hypothesis**, because we do not know whether the fact is actually true or not.

However, it is not always obvious whether we can make this conclusion. From the discussion above we know that each mean has an associated confidence interval, and thus this difference can just be a statistical error. We need some more formal way to test our hypothesis.

Let's compute confidence intervals separately for heights of first and second basemen:

Confidence	First Basemen	Second Basemen
0.85	73.62..74.38	71.04..71.69
0.90	73.56..74.44	70.99..71.73
0.95	73.47..74.53	70.92..71.81

We can see that under no confidence the intervals overlap. That proves our hypothesis that first basemen are higher than second basemen.

More formally, the problem we are solving is to see if **two probability distributions are the same**, or at least have the same parameters. Depending on the distribution, we need to use different tests for that. If we know that our distributions are normal, we can apply **Student t-test**.

In Student t-test, we compute so-called **t-value**, which indicates the difference between means, taking into account the variance. It is demonstrated that t-value follows **student distribution**, which allows us to get the threshold value for a given confidence level **p** (this can be computed, or looked up in the numerical tables). We then compare t-value to this threshold to approve or reject the hypothesis.

In Python, we can use the **SciPy** package, which includes `ttest_ind` function (in addition to many other useful statistical functions!). It computes the t-value for us, and also does the reverse lookup of confidence p-value, so that we can just look at the confidence to draw the conclusion.

For example, our comparison between heights of first and second basemen give us the following results:

```python
from scipy.stats import ttest_ind

tval, pval = ttest_ind(df.loc[df['Role']=='First_Baseman',['Height']], df.
print(f"T-value = {tval[0]:.2f}\nP-value: {pval[0]}")
```

```
T-value = 7.65
P-value: 9.137321189738925e-12
```

In our case, p-value is very low, meaning that there is strong evidence supporting that first basemen are taller.

There are also different other types of hypothesis that we might want to test, for example:

- To prove that a given sample follows some distribution. In our case we have assumed that heights are normally distributed, but that needs formal statistical verification.
- To prove that a mean value of a sample corresponds to some predefined value
- To compare means of a number of samples (eg. what is the difference in happiness levels among different age groups)

Law of Large Numbers and Central Limit Theorem

One of the reasons why normal distribution is so important is so-called **central limit theorem**. Suppose we have a large sample of independent N values X_1, ..., X_N, sampled from any distribution with mean μ and variance σ^2. Then, for sufficiently large N (in other words, when $N\to\infty$), the mean $\Sigma_i X_i$ would be normally distributed, with mean μ and variance σ^2/N.

> Another way to interpret the central limit theorem is to say that regardless of distribution, when you compute the mean of a sum of any random variable values you end up with normal distribution.

From the central limit theorem it also follows that, when $N\to\infty$, the probability of the sample mean to be equal to μ becomes 1. This is known as **the law of large numbers**.

Covariance and Correlation

One of the things Data Science does is finding relations between data. We say that two sequences **correlate** when they exhibit the similar behavior at the same time, i.e. they either rise/fall simultaneously, or one sequence rises when another one falls and vice versa. In other words, there seems to be some relation between two sequences.

> Correlation does not necessarily indicate causal relationship between two sequences; sometimes both variables can depend on some external cause, or it can be purely by chance

> the two sequences correlate. However, strong mathematical correlation is a good indication that two variables are somehow connected.

Mathematically, the main concept that shows the relation between two random variables is **covariance**, that is computed like this: Cov(X,Y) = **E**[(X-**E**(X))(Y-**E**(Y))]. We compute the deviation of both variables from their mean values, and then product of those deviations. If both variables deviate together, the product would always be a positive value, that would add up to positive covariance. If both variables deviate out-of-sync (i.e. one falls below average when another one rises above average), we will always get negative numbers, that will add up to negative covariance. If the deviations are not dependent, they will add up to roughly zero.

The absolute value of covariance does not tell us much on how large the correlation is, because it depends on the magnitude of actual values. To normalize it, we can divide covariance by standard deviation of both variables, to get **correlation**. The good thing is that correlation is always in the range of [-1,1], where 1 indicates strong positive correlation between values, -1 - strong negative correlation, and 0 - no correlation at all (variables are independent).

Example: We can compute correlation between weights and heights of baseball players from the dataset mentioned above:

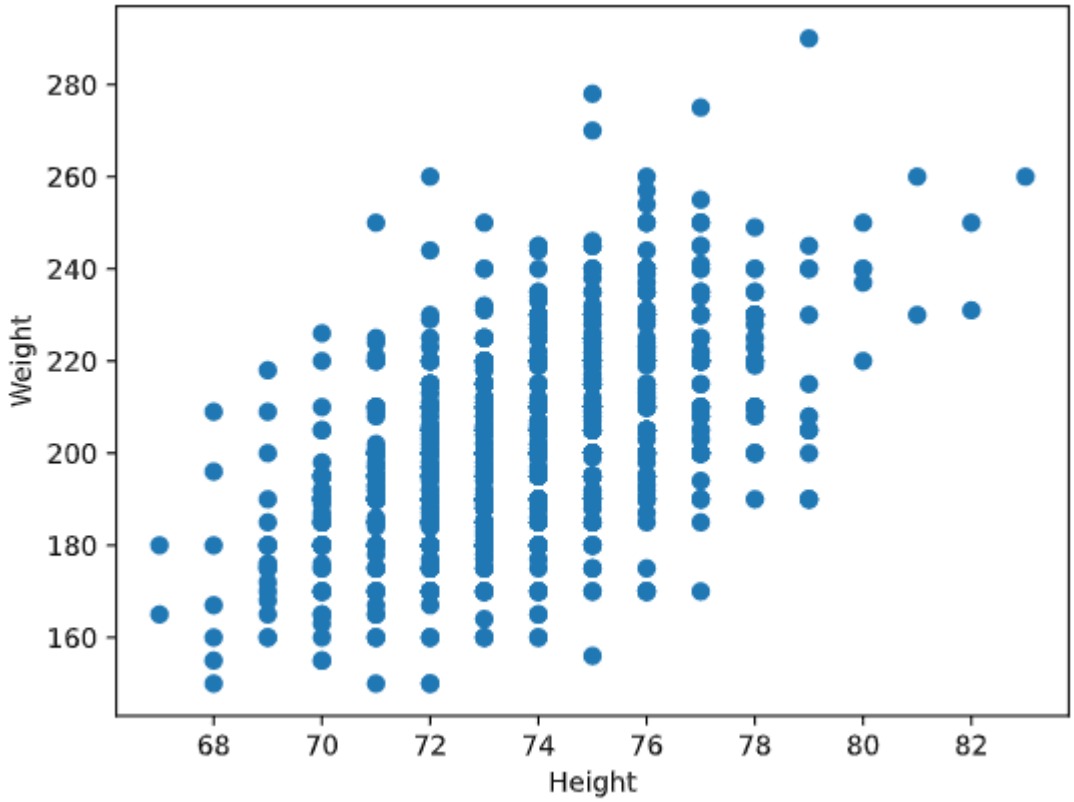
```python
print(np.corrcoef(weights,heights))
```

As a result, we get **correlation matrix** like this one:

```
array([[1.        , 0.52959196],
       [0.52959196, 1.        ]])
```

> Correlation matrix C can be computed for any number of input sequences S_1, ..., S_n. The value of C_{ij} is the correlation between S_i and S_j, and diagonal elements are always 1 (which is also self-correlation of S_i).

In our case, the value 0.53 indicates that there is some correlation between weight and height of a person. We can also make the scatter plot of one value against the other to see the relationship visually:



> More examples of correlation and covariance can be found in accompanying notebook.

Conclusion

In this section, we have learnt:

- basic statistical properties of data, such as mean, variance, mode and quartiles
- different distributions of random variables, including normal distribution
- how to find correlation between different properties
- how to use sound apparatus of math and statistics in order to prove some hypotheses,
- how to compute confidence intervals for random variable given data sample

While this is definitely not exhaustive list of topics that exist within probability and statistics, it should be enough to give you a good start into this course.

🚀 Challenge

Use the sample code in the notebook to test other hypothesis that:

1. First basemen and older that second basemen

2. First basemen and taller than third basemen
3. Shortstops are taller than second basemen

Post-lecture quiz

Review & Self Study

Probability and statistics is such a broad topic that it deserves its own course. If you are interested to go deeper into theory, you may want to continue reading some of the following books:

1. Carlos Fernanderz-Granda from New York University has great lecture notes Probability and Statistics for Data Science (available online)
2. Peter and Andrew Bruce. Practical Statistics for Data Scientists. [sample code in R].
3. James D. Miller. Statistics for Data Science [sample code in R]

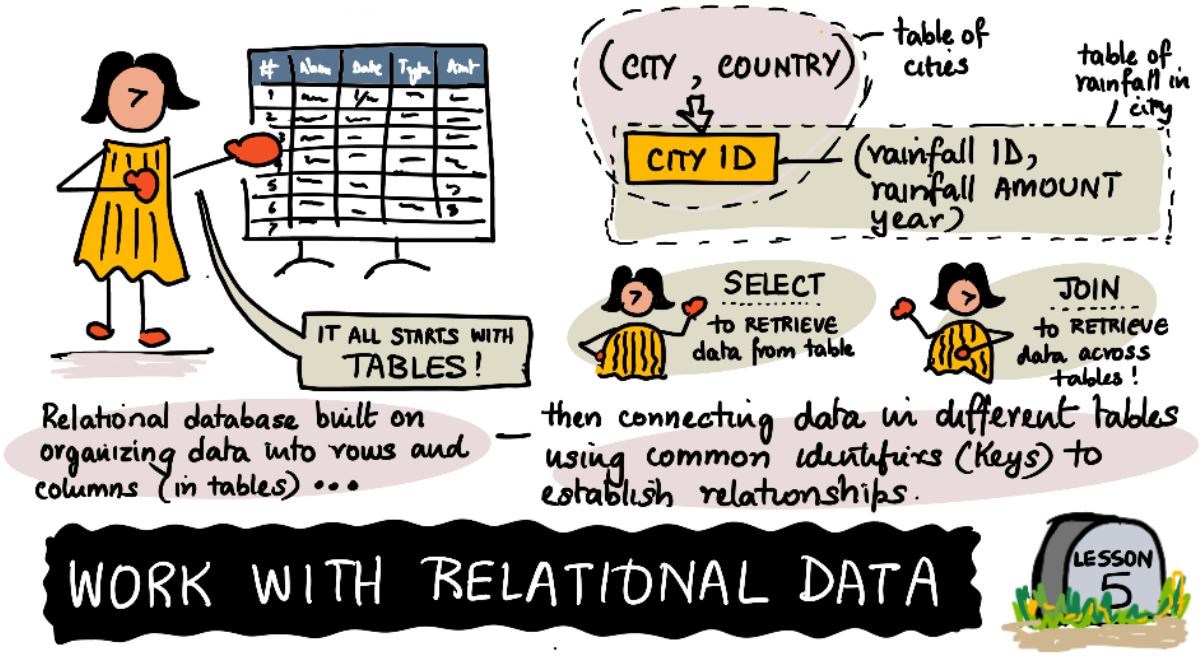
Assignment

Small Diabetes Study

Credits

This lesson has been authored with ♥ by Dmitry Soshnikov

Working with Data: Relational Databases



Working With Data: Relational Databases - *Sketchnote by @nitya*

Chances are you have used a spreadsheet in the past to store information. You had a set of rows and columns, where the rows contained the information (or data), and the columns described the information (sometimes called metadata). A relational database is built upon this core principle of columns and rows in tables, allowing you to have information spread across multiple tables. This allows you to work with more complex data, avoid duplication, and have flexibility in the way you explore the data. Let's explore the concepts of a relational database.

Pre-lecture quiz

It all starts with tables

A relational database has at its core tables. Just as with the spreadsheet, a table is a collection of columns and rows. The row contains the data or information we wish to work with, such as the name of a city or the amount of rainfall. The columns describe the data they store.

Let's begin our exploration by starting a table to store information about cities. We might start with their name and country. You could store this in a table as follows:

City	Country

City	Country
Tokyo	Japan
Atlanta	United States
Auckland	New Zealand

Notice the column names of **city**, **country** and **population** describe the data being stored, and each row has information about one city.

The shortcomings of a single table approach

Chances are, the table above seems relatively familiar to you. Let's start to add some additional data to our burgeoning database - annual rainfall (in millimeters). We'll focus on the years 2018, 2019 and 2020. If we were to add it for Tokyo, it might look something like this:

City	Country	Year	Amount
Tokyo	Japan	2020	1690
Tokyo	Japan	2019	1874
Tokyo	Japan	2018	1445

What do you notice about our table? You might notice we're duplicating the name and country of the city over and over. That could take up quite a bit of storage, and is largely unnecessary to have multiple copies of. After all, Tokyo has just the one name we're interested in.

OK, let's try something else. Let's add new columns for each year:

City	Country	2018	2019	2020
Tokyo	Japan	1445	1874	1690
Atlanta	United States	1779	1111	1683
Auckland	New Zealand	1386	942	1176

While this avoids the row duplication, it adds a couple of other challenges. We would need to modify the structure of our table each time there's a new year. Additionally, as our data grows having our years as columns will make it trickier to retrieve and calculate values.

This is why we need multiple tables and relationships. By breaking apart our data we can avoid duplication and have more flexibility in how we work with our data.

The concepts of relationships

Let's return to our data and determine how we want to split things up. We know we want to store the name and country for our cities, so this will probably work best in one table.

City	Country
Tokyo	Japan
Atlanta	United States
Auckland	New Zealand

But before we create the next table, we need to figure out how to reference each city. We need some form of an identifier, ID or (in technical database terms) a primary key. A primary key is a value used to identify one specific row in a table. While this could be based on a value itself (we could use the name of the city, for example), it should almost always be a number or other identifier. We don't want the id to ever change as it would break the relationship. You will find in most cases the primary key or id will be an auto-generated number.

> ✅ Primary key is frequently abbreviated as PK

cities

city_id	City	Country
1	Tokyo	Japan
2	Atlanta	United States

city_id	City	Country
3	Auckland	New Zealand

> ✅ You will notice we use the terms "id" and "primary key" interchangeably during this lesson. The concepts here apply to DataFrames, which you will explore later. DataFrames don't use the terminology of "primary key", however you will notice they behave much in the same way.

With our cities table created, let's store the rainfall. Rather than duplicating the full information about the city, we can use the id. We should also ensure the newly created table has an *id* column as well, as all tables should have an id or primary key.

rainfall

rainfall_id	city_id	Year	Amount
1	1	2018	1445
2	1	2019	1874
3	1	2020	1690
4	2	2018	1779
5	2	2019	1111
6	2	2020	1683
7	3	2018	1386
8	3	2019	942
9	3	2020	1176

Notice the **city_id** column inside the newly created **rainfall** table. This column contains values which reference the IDs in the **cities** table. In technical relational data terms, this is called a **foreign key**; it's

a primary key from another table. You can just think of it as a reference or a pointer. **city_id** 1 references Tokyo.

> [!NOTE] Foreign key is frequently abbreviated as FK

Retrieving the data

With our data separated into two tables, you may be wondering how we retrieve it. If we are using a relational database such as MySQL, SQL Server or Oracle, we can use a language called Structured Query Language or SQL. SQL (sometimes pronounced sequel) is a standard language used to retrieve and modify data in a relational database.

To retrieve data you use the command `SELECT`. At its core, you **select** the columns you want to see **from** the table they're contained in. If you wanted to display just the names of the cities, you could use the following:

```sql
SELECT city
FROM cities;

-- Output:
-- Tokyo
-- Atlanta
-- Auckland
```

`SELECT` is where you list the columns, and `FROM` is where you list the tables.

> [NOTE] SQL syntax is case-insensitive, meaning `select` and `SELECT` mean the same thing. However, depending on the type of database you are using the columns and tables might be case sensitive. As a result, it's a best practice to always treat everything in programming like it's case sensitive. When writing SQL queries common convention is to put the keywords in all upper-case letters.

The query above will display all cities. Let's imagine we only wanted to display cities in New Zealand. We need some form of a filter. The SQL keyword for this is `WHERE`, or "where something is true".

```sql
SELECT city
FROM cities
WHERE country = 'New Zealand';

-- Output:
-- Auckland
```

Joining data

Until now we've retrieved data from a single table. Now we want to bring the data together from both **cities** and **rainfall**. This is done by *joining* them together. You will effectively create a seam between the two tables, and match up the values from a column from each table.

In our example, we will match the **city_id** column in **rainfall** with the **city_id** column in **cities**. This will match the rainfall value with its respective city. The type of join we will perform is what's called an *inner* join, meaning if any rows don't match with anything from the other table they won't be displayed. In our case every city has rainfall, so everything will be displayed.

Let's retrieve the rainfall for 2019 for all our cities.

We're going to do this in steps. The first step is to join the data together by indicating the columns for the seam - **city_id** as highlighted before.

```sql
SELECT cities.city
    rainfall.amount
FROM cities
    INNER JOIN rainfall ON cities.city_id = rainfall.city_id
```

We have highlighted the two columns we want, and the fact we want to join the tables together by the **city_id**. Now we can add the `WHERE` statement to filter out only year 2019.

```sql
SELECT cities.city
    rainfall.amount
FROM cities
    INNER JOIN rainfall ON cities.city_id = rainfall.city_id
WHERE rainfall.year = 2019

-- Output
```

```
-- city     | amount
-- -------- | ------
-- Tokyo    | 1874
-- Atlanta  | 1111
-- Auckland |  942
```

Summary

Relational databases are centered around dividing information between multiple tables which is then brought back together for display and analysis. This provides a high degree of flexibility to perform calculations and otherwise manipulate data. You have seen the core concepts of a relational database, and how to perform a join between two tables.

🚀 Challenge

There are numerous relational databases available on the internet. You can explore the data by using the skills you've learned above.

Post-Lecture Quiz

Post-lecture quiz

Review & Self Study

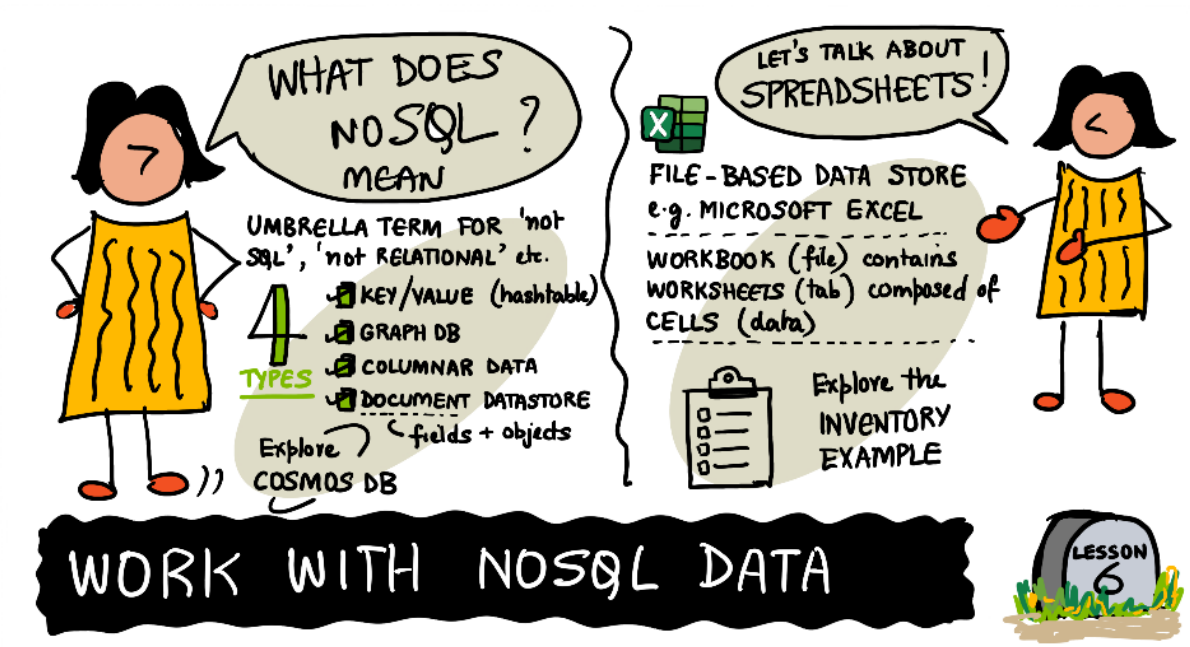
There are several resources available on Microsoft Learn for you to continue your exploration of SQL and relational database concepts

- Describe concepts of relational data
- Get Started Querying with Transact-SQL (Transact-SQL is a version of SQL)
- SQL content on Microsoft Learn

Assignment

Assignment Title

Working with Data: Non-Relational Data



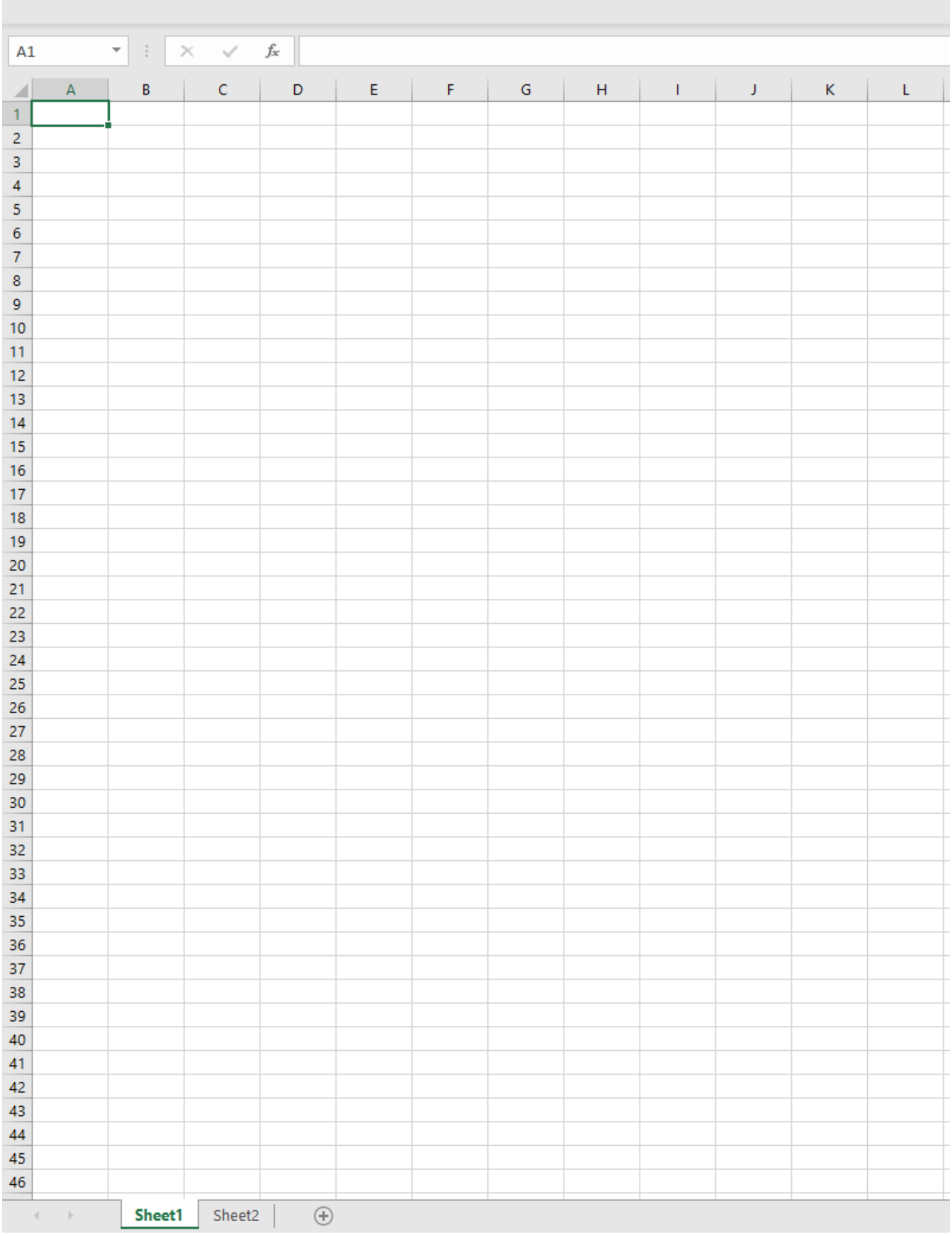
Working with NoSQL Data - *Sketchnote by @nitya*

Pre-Lecture Quiz

Data is not limited to relational databases. This lesson focuses on non-relational data and will cover the basics of spreadsheets and NoSQL.

Spreadsheets

Spreadsheets are a popular way to store and explore data because it requires less work to setup and get started. In this lesson you'll learn the basic components of a spreadsheet, as well as formulas and functions. The examples will be illustrated with Microsoft Excel, but most of the parts and topics will have similar names and steps in comparison to other spreadsheet software.



A spreadsheet is a file and will be accessible in the file system of a computer, device, or cloud based file system. The software itself may be browser based or an application that must be installed on a computer or downloaded as an app. In Excel these files are also defined as **workbooks** and this terminology will be used the remainder of this lesson.

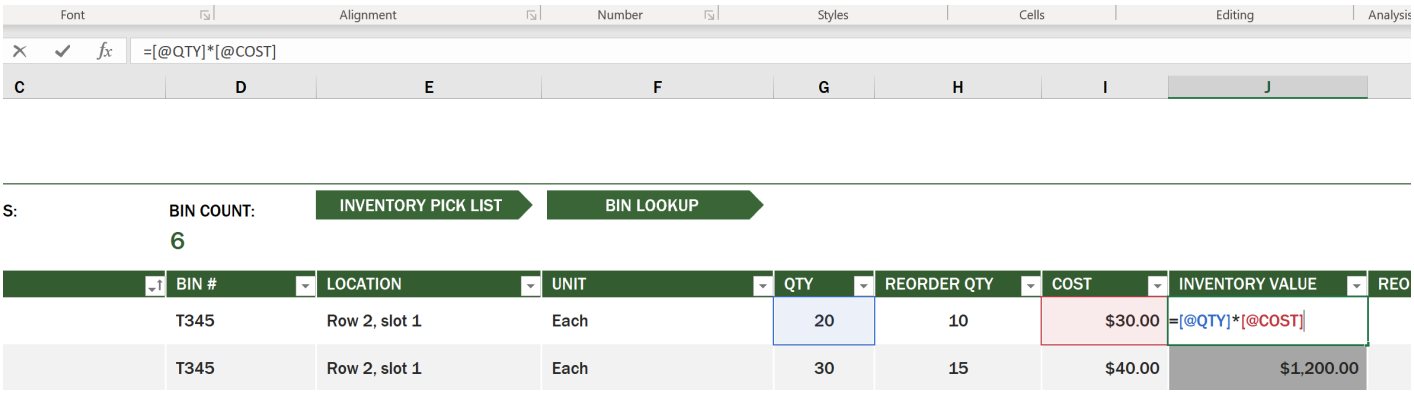
A workbook contains one or more **worksheets**, where each worksheet are labeled by tabs. Within a worksheet are rectangles called **cells**, which will contain the actual data. A cell is the intersection of a

row and column, where the columns are labeled with alphabetical characters and rows labeled numerically. Some spreadsheets will contain headers in the first few rows to describe the data in a cell.

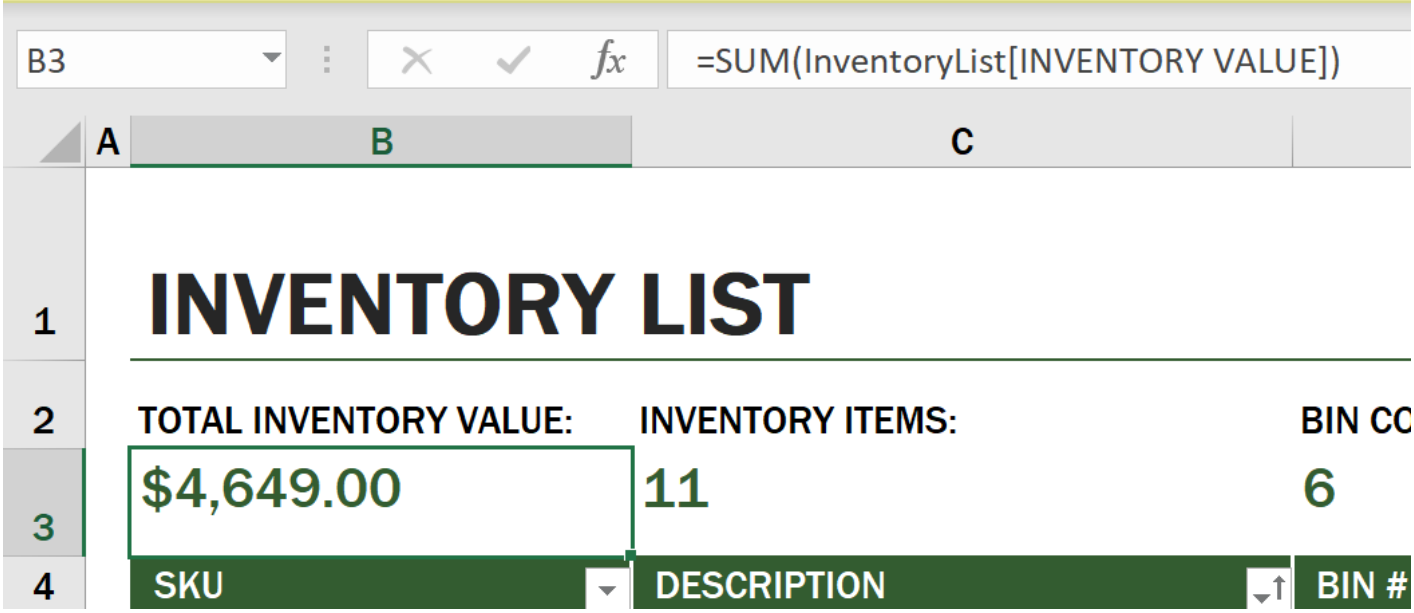
With these basic elements of an Excel workbook, we'll use and an example from Microsoft Templates focused on an inventory to walk through some additional parts of a spreadsheet.

Managing an Inventory

The spreadsheet file named "InventoryExample" is a formatted spreadsheet of items within an inventory that contains three worksheets, where the tabs are labeled "Inventory List", "Inventory Pick List" and "Bin Lookup". Row 4 of the Inventory List worksheet is the header, which describes the value of each cell in the header column.



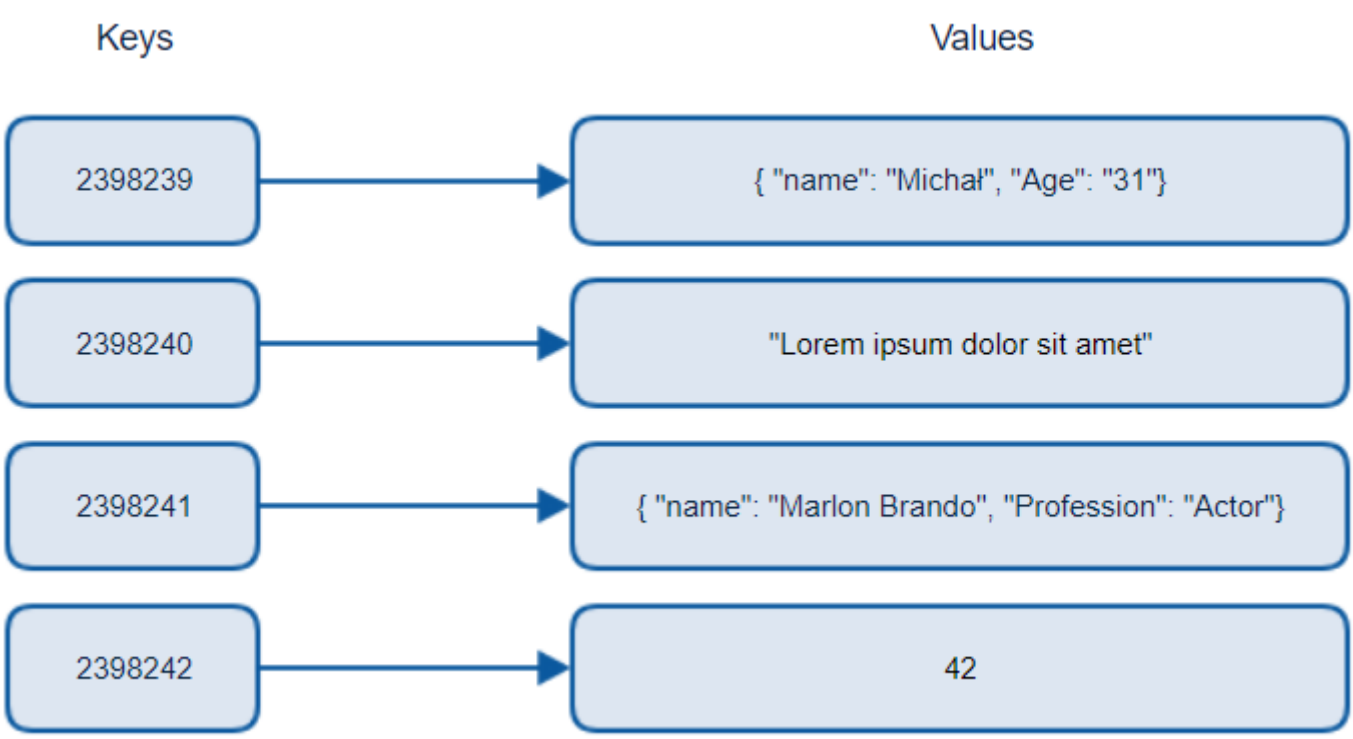
There are instances where a cell is dependent on the values of other cells to generate its value. The Inventory List spreadsheet keeps track of the cost of every item in its inventory, but what if we need to know the value of everything in the inventory? **Formulas** perform actions on cell data and is used to calculate the cost of the inventory in this example. This spreadsheet used a formula in the Inventory Value column to calculate the value of each item by multiplying the quantity under the QTY header and its costs by the cells under the COST header. Double clicking or highlighting a cell will show the formula. You'll notice that formulas start with an equals sign, followed by the calculation or operation.



We can use another formula to add all the values of Inventory Value together to get its total value. This could be calculated by adding each cell to generate the sum, but that can be a tedious task. Excel has **functions**, or predefined formulas to perform calculations on cell values. Functions require arguments, which are the required values used to perform these calculations. When functions require more than one argument, they will need to be listed in a particular order or the function may not calculate the correct value. This example uses the SUM function, and uses the values of on Inventory Value as the argument to add generate the total listed under row 3, column B (also referred to as B3).

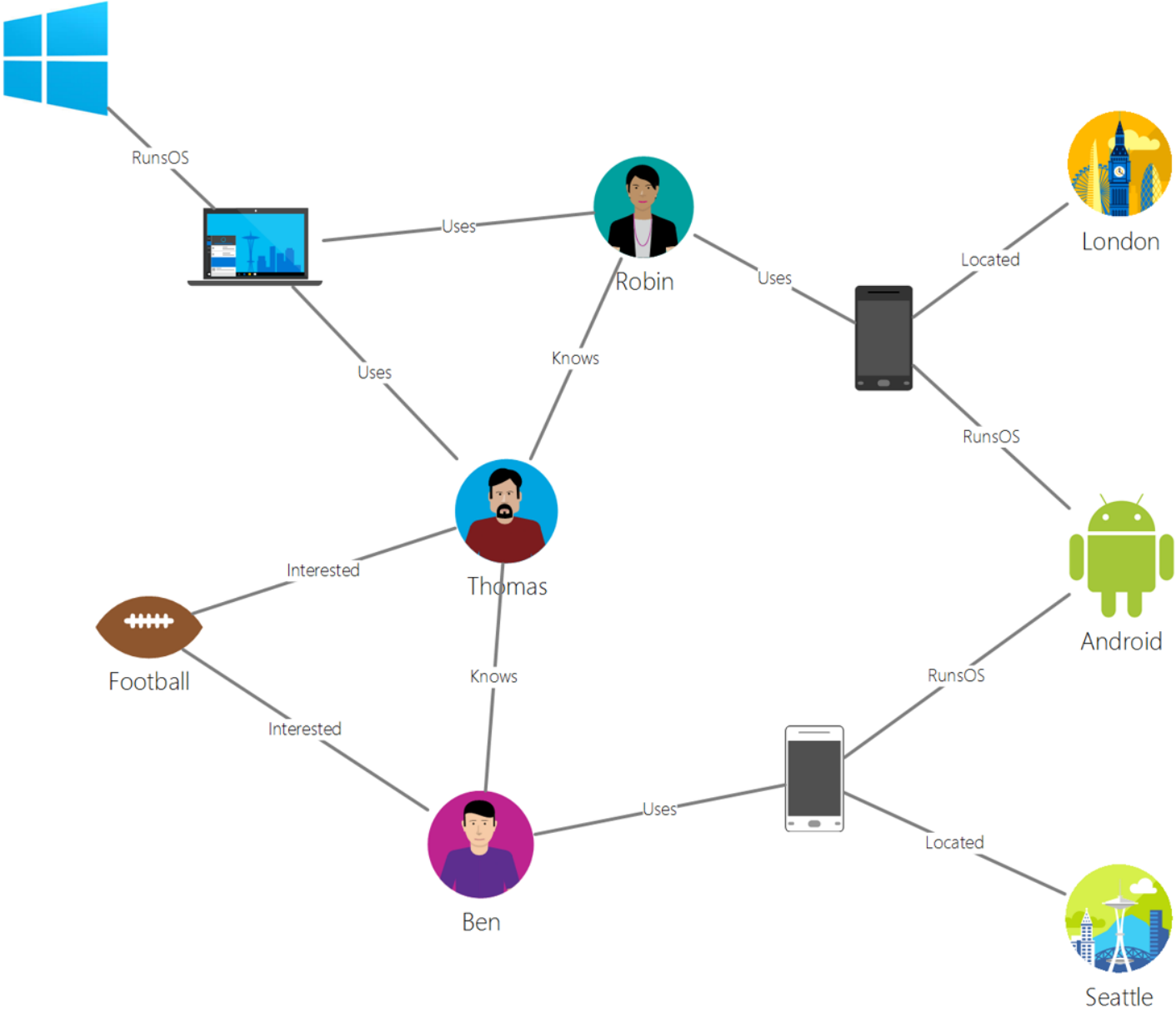
NoSQL

NoSQL is an umbrella term for the different ways to store non-relational data and can be interpreted as "non-SQL", "non-relational" or "not only SQL". These type of database systems can be categorized into 4 types.



> Source from Michał Białecki Blog

Key-value databases pair unique keys, which are a unique identifier associated with a value. These pairs are stored using a hash table with an appropriate hashing function.



> Source from Microsoft

Graph databases describe relationships in data and are represented as a collection of nodes and edges. A node represents an entity, something that exists in the real world such as a student or bank statement. Edges represent the relationship between two entities Each node and edge have properties that provides additional information about each node and edges.

CustomerID	Column Family: Identity
001	**First name:** Mu Bae **Last name:** Min
002	**First name:** Francisco **Last name:** Vila Nova **Suffix:** Jr.
003	**First name:** Lena **Last name:** Adamcyz **Title:** Dr.

CustomerID	Column Family: Contact Info
001	**Phone number:** 555-0100 **Email:** someone@example.com
002	**Email:** vilanova@contoso.com
003	**Phone number:** 555-0120

Columnar data stores organizes data into columns and rows like a relational data structure but each column is divided into groups called a column family, where the all the data under one column is related and can be retrieved and changed in one unit.

Document Data Stores with the Azure Cosmos DB

Document data stores build on the concept of a key-value data store and is made up of a series of fields and objects. This section will explore document databases with the Cosmos DB emulator.

A Cosmos DB database fits the definition of "Not Only SQL", where Cosmos DB's document database relies on SQL to query the data. The previous lesson on SQL covers the basics of the language, and we'll be able to apply some of the same queries to a document database here. We'll be using the Cosmos DB Emulator, which allows us to create and explore a document database locally on a computer. Read more about the Emulator here.

A document is a collection of fields and object values, where the fields describe what the object value represents. Below is an example of a document.

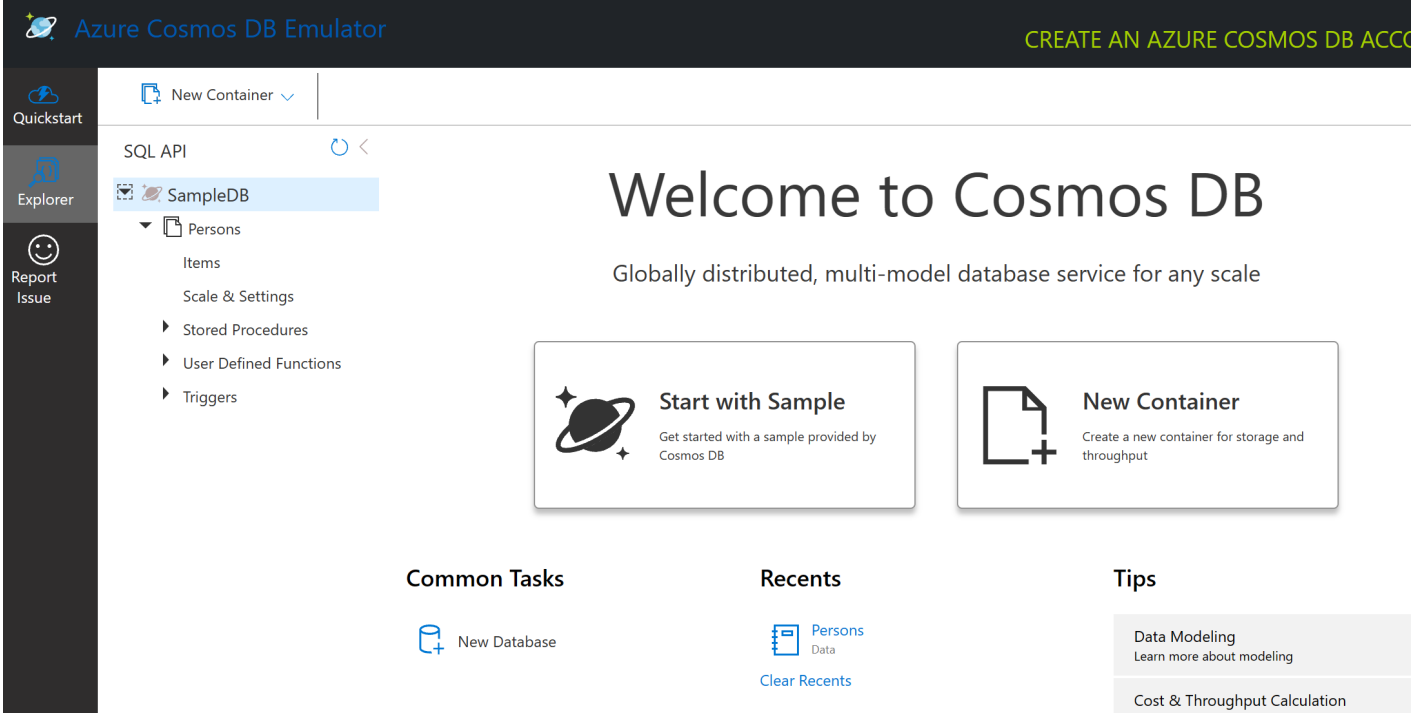
```json
{
    "firstname": "Eva",
    "age": 44,
    "id": "8c74a315-aebf-4a16-bb38-2430a9896ce5",
    "_rid": "bHwDAPQz8s0BAAAAAAAAAA==",
    "_self": "dbs/bHwDAA==/colls/bHwDAPQz8s0=/docs/bHwDAPQz8s0BAAAAAAAAAA==
    "_etag": "\"00000000-0000-0000-9f95-010a691e01d7\"",
    "_attachments": "attachments/",
    "_ts": 1630544034
}
```

The fields of interest in this document are: `firstname`, `id`, and `age`. The rest of the fields with the underscores were generated by Cosmos DB.

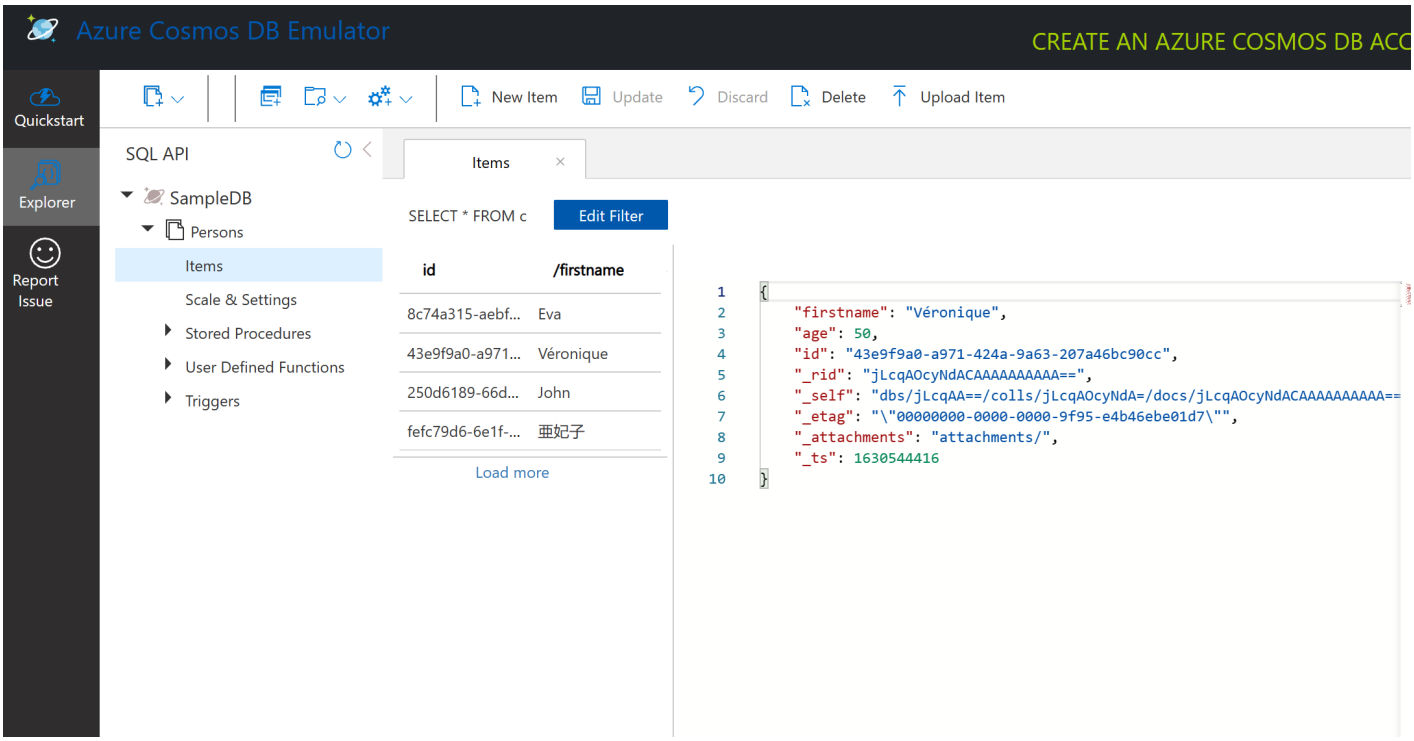
Exploring Data with the Cosmos DB Emulator

You can download and install the emulator for Windows here. Refer to this documentation for options on how to run the Emulator for macOS and Linux.

The Emulator launches a browser window, where the Explorer view allows you to explore documents.



If you're following along, click on "Start with Sample" to generate a sample database called SampleDB. If you expand Sample DB by clicking on the arrow you'll find a container called `Persons`, a container holds a collection of items, which are the documents within the container. You can explore the four individual documents under `Items`.

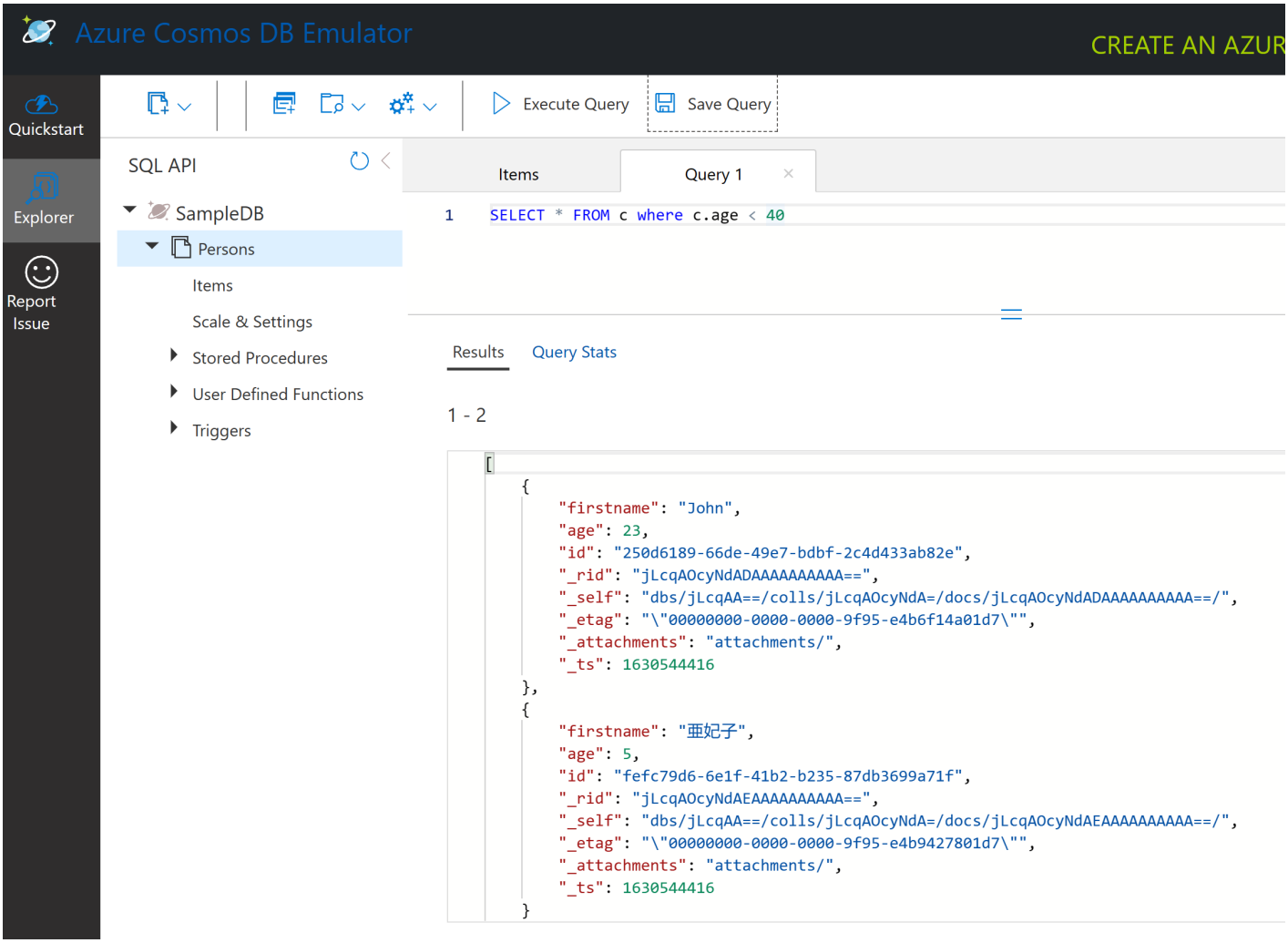


Querying Document Data with the Cosmos DB Emulator

We can also query the sample data by clicking on the new SQL Query button (second button from the left).

`SELECT * FROM c` returns all the documents in the container. Let's add a where clause and find everyone younger than 40.

```
SELECT * FROM c where c.age < 40
```



The query returns two documents, notice the age value for each document is less than 40.

JSON and Documents

If you're familiar with JavaScript Object Notation (JSON) you'll notice that documents look similar to JSON. There is a `PersonsData.json` file in this directory with more data that you may upload to the Persons container in the Emulator via the `Upload Item` button.

In most instances, APIs that return JSON data can be directly transferred and stored in document databases. Below is another document, it represents tweets from the Microsoft Twitter account that was retrieved using the Twitter API, then inserted into Cosmos DB.

```json
{
    "created_at": "2021-08-31T19:03:01.000Z",
    "id": "1432780985872142341",
    "text": "Blank slate. Like this tweet if you've ever painted in Micros
    "_rid": "dhAmAIUsA4oHAAAAAAAAAA==",
    "_self": "dbs/dhAmAA==/colls/dhAmAIUsA4o=/docs/dhAmAIUsA4oHAAAAAAAAAA=
    "_etag": "\"00000000-0000-0000-9f84-a0958ad901d7\"",
```

```
    "_attachments": "attachments/",
    "_ts": 1630537000
```

The fields of interest in this document are: `created_at` , `id` , and `text` .

🚀 Challenge

There is a `TwitterData.json` file that you can upload to the SampleDB database. It's recommended that you add it to a separate container. This can be done by:

1. Clicking the new container button in the top right
2. Selecting the existing database (SampleDB) creating a container id for the container
3. Setting the partition key to `/id`
4. Clicking OK (you can ignore rest of the information in this view as this is a small dataset running locally on your machine)
5. Open your new container and upload the Twitter Data file with `Upload Item` button

Try to run a few select queries to find the documents that have Microsoft in the text field. Hint: try to use the LIKE keyword

Post-Lecture Quiz

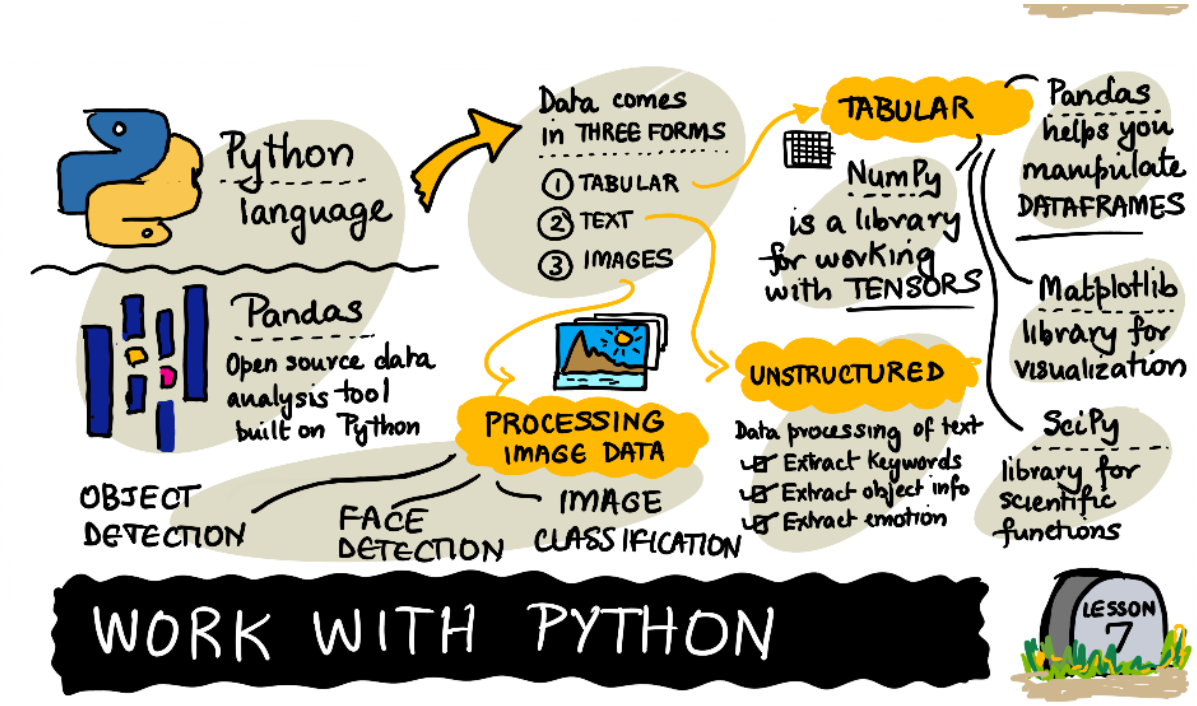
Review & Self Study

- There are some additional formatting and features added to this spreadsheet that this lesson does not cover. Microsoft has a large library of documentation and videos on Excel if you're interested in learning more.
- This architectural documentation details the characteristics in the different types of non-relational data: Non-relational Data and NoSQL
- Cosmos DB is a cloud based non-relational database that can also store the different NoSQL types mentioned in this lesson. Learn more about these types in this Cosmos DB Microsoft Learn Module

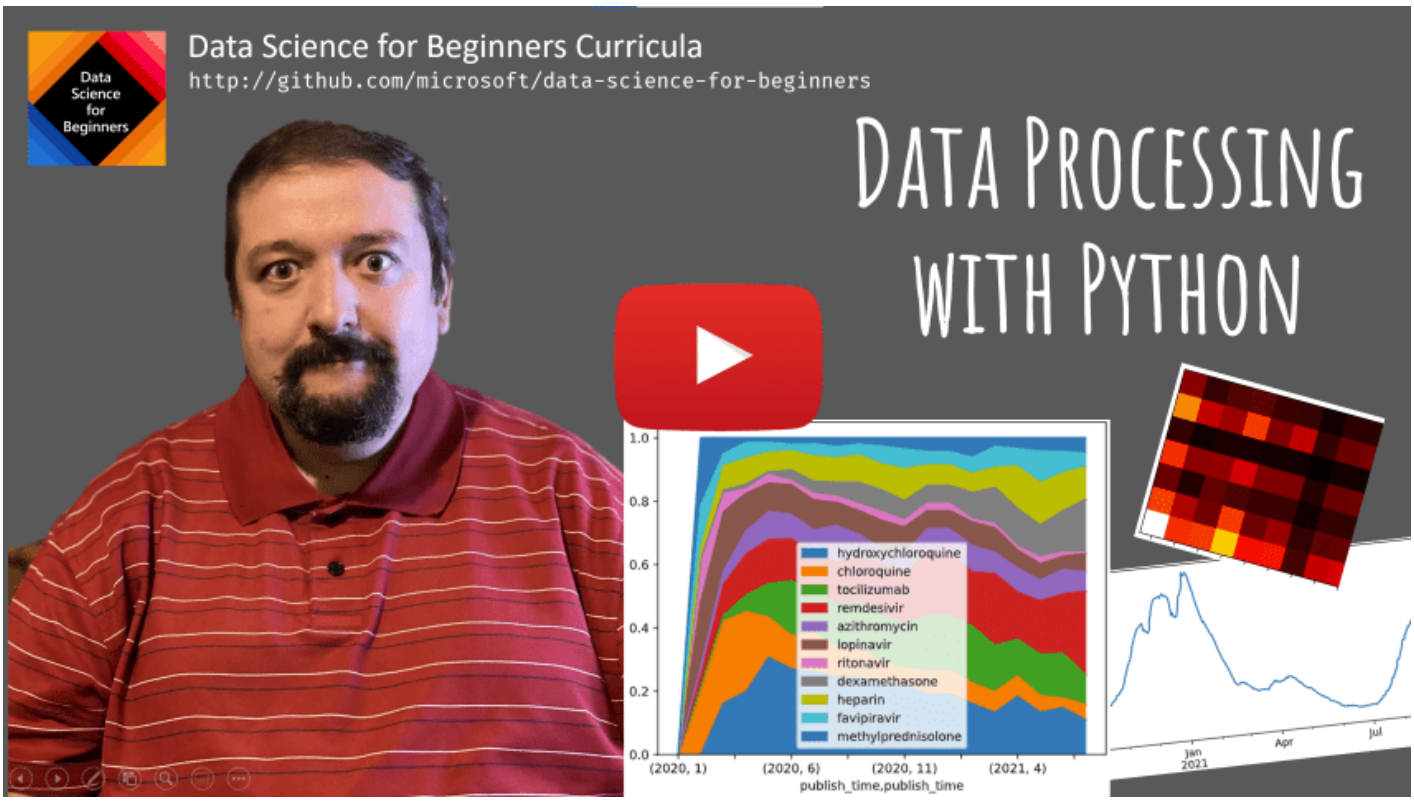
Assignment

Soda Profits

Working with Data: Python and the Pandas Library



Working With Python - *Sketchnote by @nitya*



While databases offer very efficient ways to store data and query them using query languages, the most flexible way of data processing is writing your own program to manipulate data. In many cases, doing a database query would be a more effective way. However in some cases when more complex data processing is needed, it cannot be done easily using SQL. Data processing can be programmed in any programming language, but there are certain languages that are higher level with respect to working with data. Data scientists typically prefer one of the following languages:

- **Python**, a general-purpose programming language, which is often considered one of the best options for beginners due to its simplicity. Python has a lot of additional libraries that can help you solve many practical problems, such as extracting your data from ZIP archive, or converting picture to grayscale. In addition to data science, Python is also often used for web development.
- **R** is a traditional toolbox developed with statistical data processing in mind. It also contains large repository of libraries (CRAN), making it a good choice for data processing. However, R is not a general-purpose programming language, and is rarely used outside of data science domain.
- **Julia** is another language developed specifically for data science. It is intended to give better performance than Python, making it a great tool for scientific experimentation.

In this lesson, we will focus on using Python for simple data processing. We will assume basic familiarity with the language. If you want a deeper tour of Python, you can refer to one of the following resources:

- Learn Python in a Fun Way with Turtle Graphics and Fractals - GitHub-based quick intro course into Python Programming
- Take your First Steps with Python Learning Path on Microsoft Learn

Data can come in many forms. In this lesson, we will consider three forms of data - **tabular data**, **text** and **images**.

We will focus on a few examples of data processing, instead of giving you full overview of all related libraries. This would allow you to get the main idea of what's possible, and leave you with understanding on where to find solutions to your problems when you need them.

> **Most useful advice**. When you need to perform certain operation on data that you do not know how to do, try searching for it in the internet. Stackoverflow usually contains a lot of useful code sample in Python for many typical tasks.

Pre-lecture quiz

Tabular Data and Dataframes

You have already met tabular data when we talked about relational databases. When you have a lot of data, and it is contained in many different linked tables, it definitely makes sense to use SQL for working with it. However, there are many cases when we have a table of data, and we need to gain some **understanding** or **insights** about this data, such as the distribution, correlation between values, etc. In data science, there are a lot of cases when we need to perform some transformations of the original data, followed by visualization. Both those steps can be easily done using Python.

There are two most useful libraries in Python that can help you deal with tabular data:

- **Pandas** allows you to manipulate so-called **Dataframes**, which are analogous to relational tables. You can have named columns, and perform different operations on row, columns and dataframes in general.
- **Numpy** is a library for working with **tensors**, i.e. multi-dimensional **arrays**. Array has values of the same underlying type, and it is simpler than dataframe, but it offers more mathematical operations, and creates less overhead.

There are also a couple of other libraries you should know about:

- **Matplotlib** is a library used for data visualization and plotting graphs
- **SciPy** is a library with some additional scientific functions. We have already come across this library when talking about probability and statistics

Here is a piece of code that you would typically use to import those libraries in the beginning of your Python program:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import ... # you need to specify exact sub-packages that you ne
```

Pandas is centered around a few basic concepts.

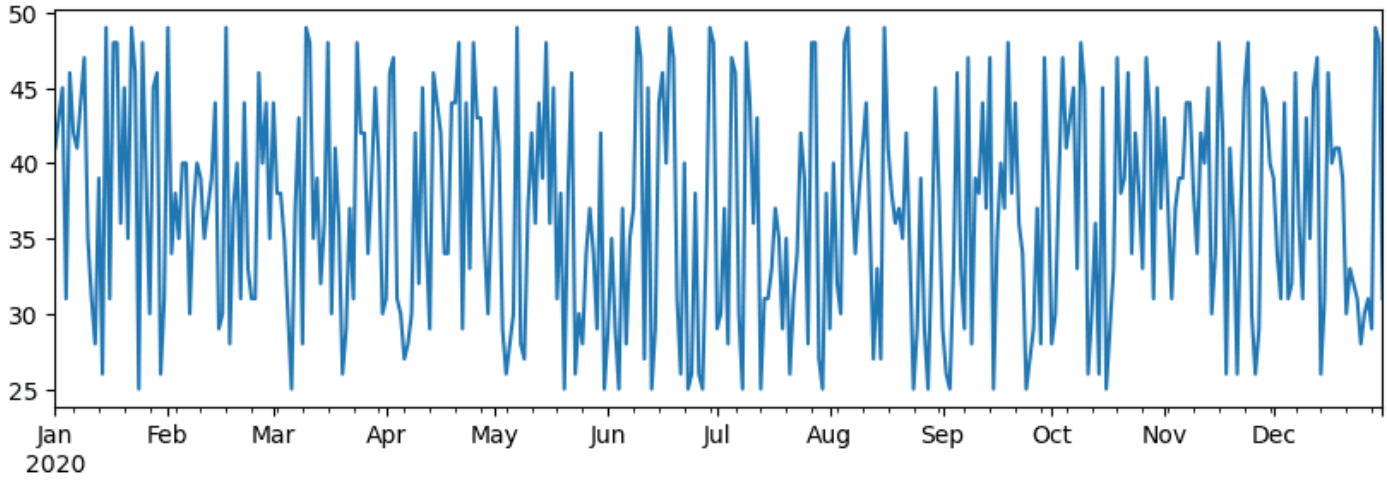
Series

Series is a sequence of values, similar to a list or numpy array. The main difference is that series also has and **index**, and when we operate on series (eg., add them), the index is taken into account. Index can be as simple as integer row number (it is the index used by default when creating a series from list or array), or it can have a complex structure, such as date interval.

> **Note**: There is some introductory Pandas code in the accompanying notebook `notebook.ipynb` . We only outline some the examples here, and you are definitely welcome to check out the full notebook.

Consider an example: we want to analyze sales of our ice-cream spot. Let's generate a series of sales numbers (number of items sold each day) for some time period:

```python
start_date = "Jan 1, 2020"
end_date = "Mar 31, 2020"
idx = pd.date_range(start_date,end_date)
print(f"Length of index is {len(idx)}")
items_sold = pd.Series(np.random.randint(25,50,size=len(idx)),index=idx)
items_sold.plot()
```

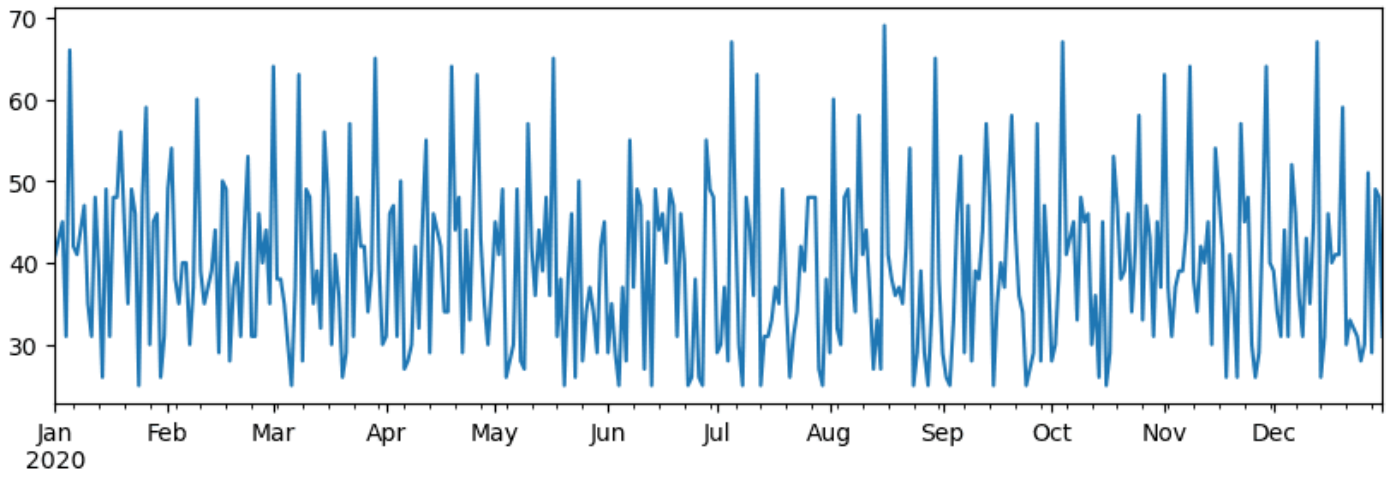


Now suppose that each week we are organizing a party for friends, and we take additional 10 packs of ice-cream for a party. We can create another series, indexed by week, to demonstrate that:

```python
additional_items = pd.Series(10,index=pd.date_range(start_date,end_date,fr
```

When we add two series together, we get total number:

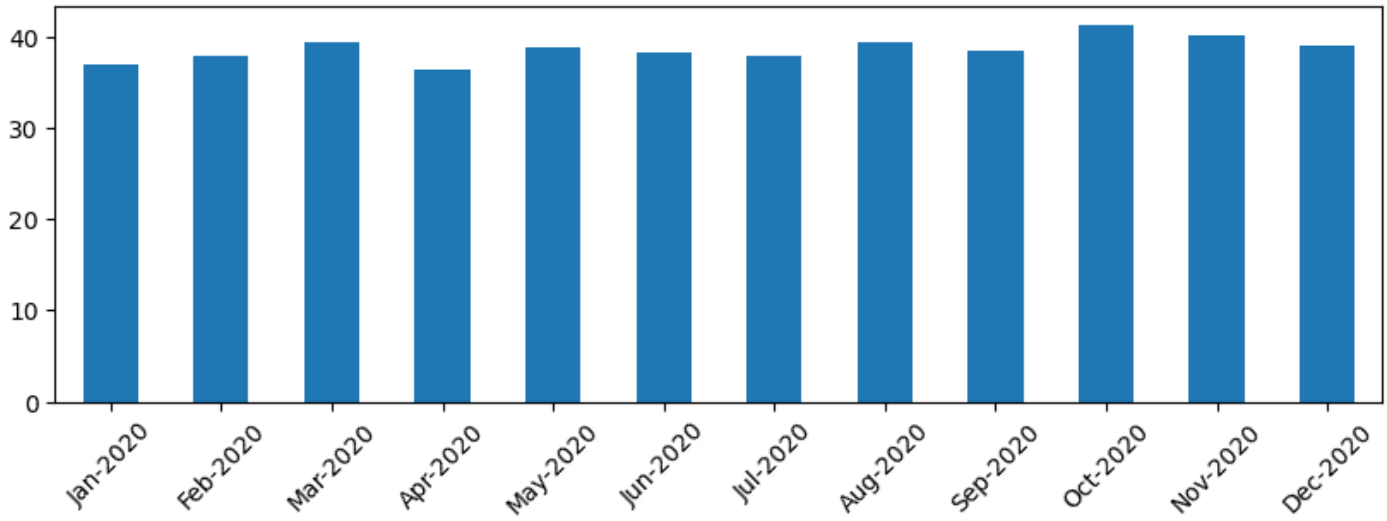
```python
total_items = items_sold.add(additional_items,fill_value=0)
total_items.plot()
```



> **Note** that we are not using simple syntax `total_items+additional_items`. If we did, we would have received a lot of `NaN` (Not a Number) values in the resulting series. This is because there are missing values for some of the index point in the `additional_items` series, and adding `Nan` to anything results in `NaN`. Thus we need to specify `fill_value` parameter during addition.

With time series, we can also **resample** the series with different time intervals. For example, suppose we want to compute mean sales volume monthly. We can use the following code:

```python
monthly = total_items.resample("1M").mean()
ax = monthly.plot(kind='bar')
```



DataFrame

A DataFrame is essentially a collection of series with the same index. We can combine several series together into a DataFrame:

```python
a = pd.Series(range(1,10))
b = pd.Series(["I","like","to","play","games","and","will","not","change"])
df = pd.DataFrame([a,b])
```

This will create a horizontal table like this: | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |---|---|---|---|---|---|---|---|---|---|---| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | 1 | I | like | to | use | Python | and | Pandas | very | much |

We can also use Series as columns, and specify column names using dictionary:

```python
df = pd.DataFrame({ 'A' : a, 'B' : b })
```

This will give us a table like this:

	A	B
0	1	I
1	2	like
2	3	to
3	4	use
4	5	Python
5	6	and
6	7	Pandas
7	8	very
8	9	much

Note that we can also get this table layout by transposing the previous table, eg. by writing

```python
df = pd.DataFrame([a,b]).T..rename(columns={ 0 : 'A', 1 : 'B' })
```

Here `.T` means the operation of transposing the DataFrame, i.e. changing rows and columns, and `rename` operation allows us to rename columns to match the previous example.

Here are a few most important operations we can perform on DataFrames:

Column selection. We can select individual columns by writing `df['A']` - this operation returns a Series. We can also select a subset of columns into another DataFrame by writing `df[['B','A']]` - this return another DataFrame.

Filtering only certain rows by criteria. For example, to leave only rows with column `A` greater than 5, we can write `df[df['A']>5]` .

> **Note**: The way filtering works is the following. The expression `df['A']<5` returns a boolean series, which indicates whether expression is `True` or `False` for each element of the original series `df['A']` . When boolean series is used as an index, it returns subset of rows in the DataFrame. Thus it is not possible to use arbitrary Python boolean expression, for example, writing `df[df['A']>5 and df['A']<7]` would be wrong. Instead, you should use special `&` operation on boolean series, writing `df[(df['A']>5) & (df['A']<7)]` (brackets are important here).

Creating new computable columns. We can easily create new computable columns for our DataFrame by using intuitive expression like this:

```python
df['DivA'] = df['A']-df['A'].mean()
```

This example calculates divergence of A from its mean value. What actually happens here is we are computing a series, and then assigning this series to the left-hand-side, creating another column. Thus, we cannot use any operations that are not compatible with series, for example, the code below is wrong:

```python
# Wrong code -> df['ADescr'] = "Low" if df['A'] < 5 else "Hi"
df['LenB'] = len(df['B']) # <- Wrong result
```

The latter example, while being syntactically correct, gives us wrong result, because it assigns the length of series `B` to all values in the column, and not the length of individual elements as we intended.

If we need to compute complex expressions like this, we can use `apply` function. The last example can be written as follows:

```python
df['LenB'] = df['B'].apply(lambda x : len(x))
# or
df['LenB'] = df['B'].apply(len)
```

After operations above, we will end up with the following DataFrame:

	A	B	DivA	LenB
0	1	I	-4.0	1
1	2	like	-3.0	4
2	3	to	-2.0	2
3	4	use	-1.0	3
4	5	Python	0.0	6
5	6	and	1.0	3
6	7	Pandas	2.0	6
7	8	very	3.0	4
8	9	much	4.0	4

Selecting rows based on numbers can be done using `iloc` construct. For example, to select first 5 rows from the DataFrame:

```python
df.iloc[:5]
```

Grouping is often used to get a result similar to *pivot tables* in Excel. Suppose that we want to compute mean value of column `A` for each given number of `LenB`. Then we can group our DataFrame by `LenB`, and call `mean`:

```python
df.groupby(by='LenB').mean()
```

If we need to compute mean and the number of elements in the group, then we can use more complex `aggregate` function:

```python
df.groupby(by='LenB') \
 .aggregate({ 'DivA' : len, 'A' : lambda x: x.mean() }) \
 .rename(columns={ 'DivA' : 'Count', 'A' : 'Mean'})
```

This gives us the following table:

LenB	Count	Mean
1	1	1.000000
2	1	3.000000
3	2	5.000000
4	3	6.333333
6	2	6.000000

Getting Data

We have seen how easy it is to construct Series and DataFrames from Python objects. However, data usually comes in the form of a text file, or an Excel table. Luckily, Pandas offers us a simple way to load data from disk. For example, reading CSV file is as simple as this:

```python
df = pd.read_csv('file.csv')
```

We will see more examples of loading data, including fetching it from external web sites, in the "Challenge" section

Printing and Plotting

A Data Scientist often has to explore the data, thus it is important to be able to visualize it. When DataFrame is big, many times we want just to make sure we are doing everything correctly by printing out the first few rows. This can be done by calling `df.head()`. If you are running it from Jupyter Notebook, it will print out the DataFrame in a nice tabular form.

We have also seen the usage of `plot` function to visualize some columns. While `plot` is very useful for many tasks, and supports many different graph types via `kind=` parameter, you can

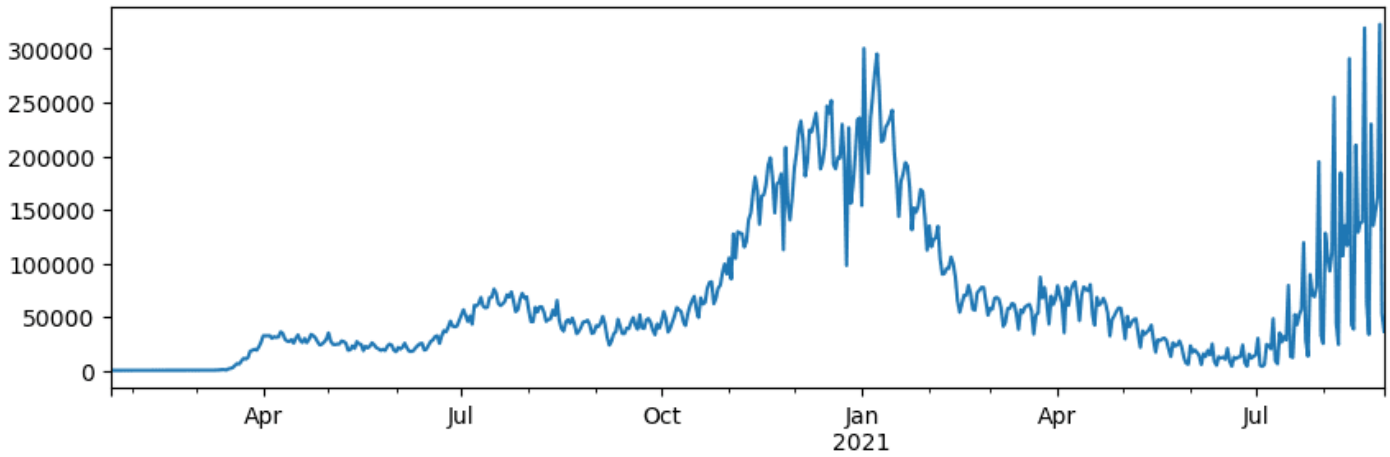
always use raw `matplotlib` library to plot something more complex. We will cover data visualization in detail in separate course lessons.

This overview covers most important concepts of Pandas, however, the library is very rich, and there is no limit to what you can do with it! Let's now apply this knowledge for solving specific problem.

🚀 Challenge 1: Analyzing COVID Spread

First problem we will focus on is modelling of epidemic spread of COVID-19. In order to do that, we will use the data on the number of infected individuals in different countries, provided by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. Dataset is available in this GitHub Repository.

Since we want to demonstrate how to deal with data, we invite you to open `notebook-covidspread.ipynb` and read it from top to bottom. You can also execute cells, and do some challenges that we have left for you at the end.



> If you do not know how to run code in Jupyter Notebook, have a look at this article.

Working with Unstructured Data

While data very often comes in tabular form, in some cases we need to deal with less structured data, for example, text or images. In this case, to apply data processing techniques we have seen above, we need to somehow **extract** structured data. Here are a few examples:

- Extracting keywords from text, and seeing how often those keywords appear
- Using neural networks to extract information about objects on the picture

- Getting information on emotions of people on video camera feed

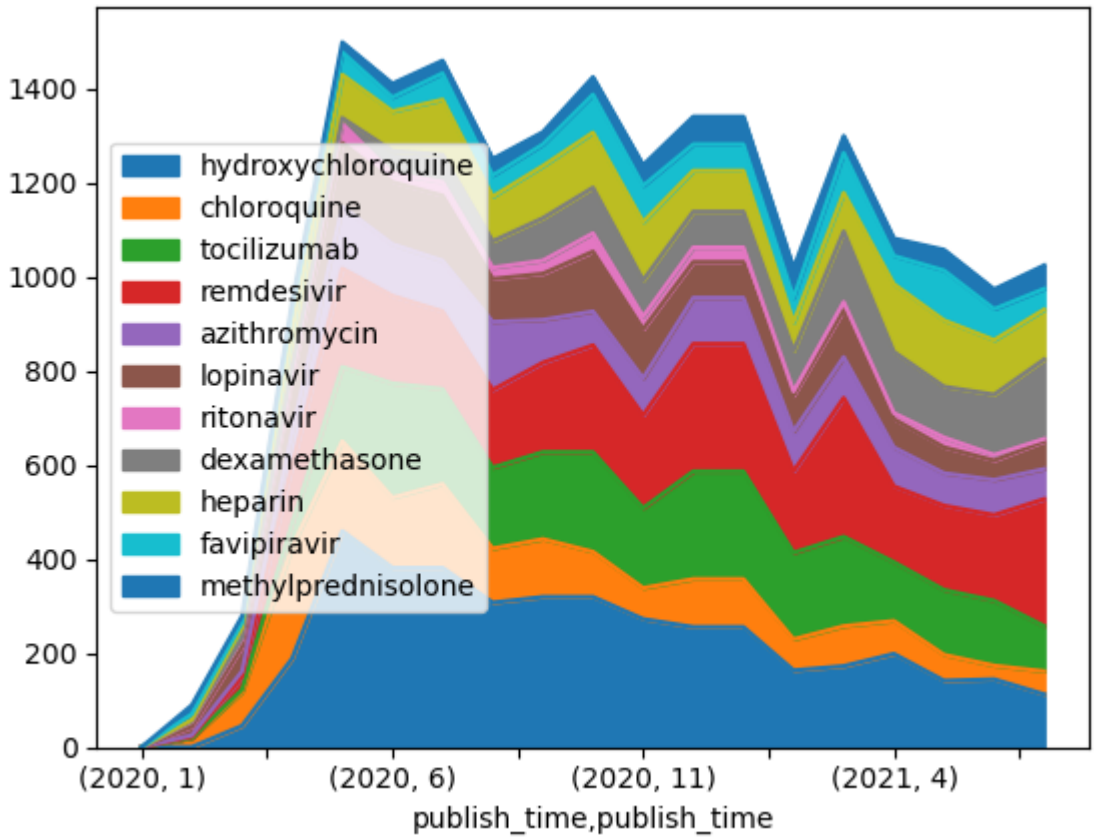
🚀 Challenge 2: Analyzing COVID Papers

In this challenge, we will continue with the topic of COVID pandemic, and focus on processing scientific papers on the subject. There is CORD-19 Dataset with more than 7000 (at the time of writing) papers on COVID, available with metadata and abstracts (and for about half of them there is also full text provided).

A full example of analyzing this dataset using Text Analytics for Health cognitive service is described in this blog post. We will discuss simplified version of this analysis.

> **NOTE**: We do not provide a copy of the dataset as part of this repository. You may first need to download the `metadata.csv` file from this dataset on Kaggle. Registration with Kaggle may be required. You may also download the dataset without registration from here, but it will include all full texts in addition to metadata file.

Open `notebook-papers.ipynb` and read it from top to bottom. You can also execute cells, and do some challenges that we have left for you at the end.



Processing Image Data

Recently, very powerful AI models have been developed that allow us to understand images. There are many tasks that can be solved using pre-trained neural networks, or cloud services. Some examples include:

- **Image Classification**, which can help you categorize the image into one of the pre-defined classes. You can easily train your own image classifiers using services such as Custom Vision
- **Object Detection** to detect different objects in the image. Services such as computer vision can detect a number of common objects, and you can train Custom Vision model to detect some specific objects of interest.
- **Face Detection**, including Age, Gender and Emotion detection. This can be done via Face API.

All those cloud services can be called using Python SDKs, and thus can be easily incorporated into your data exploration workflow.

Here are some examples of exploring data from Image data sources:

- In the blog post How to Learn Data Science without Coding we explore Instagram photos, trying to understand what makes people give more likes to a photo. We first extract as much information from pictures as possible using computer vision, and then use Azure Machine Learning AutoML to build interpretable model.
- In Facial Studies Workshop we use Face API to extract emotions on people on photographs from events, in order to try to understand what makes people happy.

Conclusion

Whether you already have structured or unstructured data, using Python you can perform all steps related to data processing and understanding. It is probably the most flexible way of data processing, and that is the reason the majority of data scientists use Python as their primary tool. Learning Python in depth is probably a good idea if you are serious about your data science journey!

Post-lecture quiz

Review & Self Study

Books

- Wes McKinney. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython

Online Resources

- Official 10 minutes to Pandas tutorial
- Documentation on Pandas Visualization

Learning Python

- Learn Python in a Fun Way with Turtle Graphics and Fractals
- Take your First Steps with Python Learning Path on Microsoft Learn

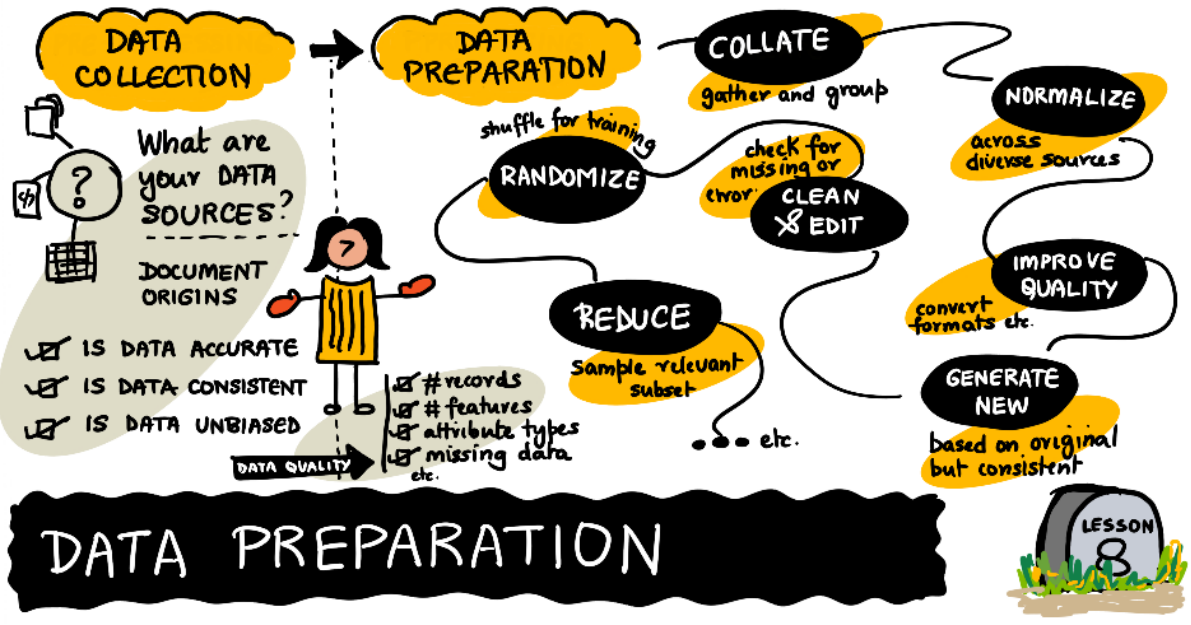
Assignment

Perform more detailed data study for the challenges above

Credits

This lesson has been authored with ♥ by Dmitry Soshnikov

Working with Data: Data Preparation



Data Preparation - *Sketchnote by @nitya*

Pre-Lecture Quiz

Depending on its source, raw data may contain some inconsistencies that will cause challenges in analysis and modeling. In other words, this data can be categorized as “dirty” and will need to be cleaned up. This lesson focuses on techniques for cleaning and transforming the data to handle challenges of missing, inaccurate, or incomplete data. Topics covered in this lesson will utilize Python and the Pandas library and will be demonstrated in the notebook within this directory.

The importance of cleaning data

- **Ease of use and reuse**: When data is properly organized and normalized it’s easier to search, use, and share with others.
- **Consistency**: Data science often requires working with more than one dataset, where datasets from different sources need to be joined together. Making sure that each individual data set has common standardization will ensure that the data is still useful when they are all merged into one dataset.
- **Model accuracy**: Data that has been cleaned improves the accuracy of models that rely on it.

Common cleaning goals and strategies

- **Exploring a dataset**: Data exploration, which is covered in a later lesson can help you discover data that needs to be cleaned up. Visually observing values within a dataset can set expectations of what that rest of it will look like, or provide an idea of the problems that can be resolved. Exploration can involve basic querying, visualizations, and sampling.
- **Formatting**: Depending on the source, data can have inconsistencies in how it’s presented. This can cause problems in searching for and representing the value, where it’s seen within the dataset but is not properly represented in visualizations or query results. Common formatting problems involve resolving whitespace, dates, and data types. Resolving formatting issues is typically up to the people who are using the data. For example, standards on how dates and numbers are presented can differ by country.

- **Duplications**: Data that has more than one occurrence can produce inaccurate results and usually should be removed. This can be a common occurrence when joining more two or more datasets together. However, there are instances where duplication in joined datasets contain pieces that can provide additional information and may need to be preserved.
- **Missing Data**: Missing data can cause inaccuracies as well as weak or biased results. Sometimes these can be resolved by a "reload" of the data, filling in the missing values with computation and code like Python, or simply just removing the value and corresponding data. There are numerous reasons for why data may be missing and the actions that are taken to resolve these missing values can be dependent on how and why they went missing in the first place.

🚀 Challenge

Give the exercises in the notebook a try!

Post-Lecture Quiz

Review & Self Study

There are many ways to discover and approach preparing your data for analysis and modeling and cleaning the data is an important step that is a "hands on" experience. Try these challenges from Kaggle to explore techniques that this lesson didn't cover.

- Data Cleaning Challenge: Parsing Dates
- Data Cleaning Challenge: Scale and Normalize Data

Assignment

Evaluating Data from a Form

Visualizing Quantities

Visualizing Quantities - *Sketchnote by @nitya*

In this lesson you will explore how to use one of the many available Python libraries to learn how to create interesting visualizations all around the concept of quantity. Using a cleaned dataset about the birds of Minnesota, you can learn many interesting facts about local wildlife.

Pre-lecture quiz

Observe wingspan with Matplotlib

An excellent library to create both simple and sophisticated plots and charts of various kinds is Matplotlib. In general terms, the process of plotting data using these libraries includes identifying the parts of your dataframe that you want to target, performing any transforms on that data necessary, assigning its x and y axis values, deciding what kind of plot to show, and then showing the plot. Matplotlib offers a large variety of visualizations, but for this lesson, let's focus on the ones most appropriate for visualizing quantity: line charts, scatterplots, and bar plots.

> ✅ Use the best chart to suit your data's structure and the story you want to tell.
>
> - To analyze trends over time: line
> - To compare values: bar, column, pie, scatterplot
> - To show how parts relate to a whole: pie

- To show distribution of data: scatterplot, bar
- To show trends: line, column
- To show relationships between values: line, scatterplot, bubble

If you have a dataset and need to discover how much of a given item is included, one of the first tasks you have at hand will be to inspect its values.

✅ There are very good 'cheat sheets' available for Matplotlib here and here.

Build a line plot about bird wingspan values

Open the `notebook.ipynb` file at the root of this lesson folder and add a cell.

> Note: the data is stored in the root of this repo in the `/data` folder.

```python
import pandas as pd
import matplotlib.pyplot as plt
birds = pd.read_csv('../../data/birds.csv')
birds.head()
```

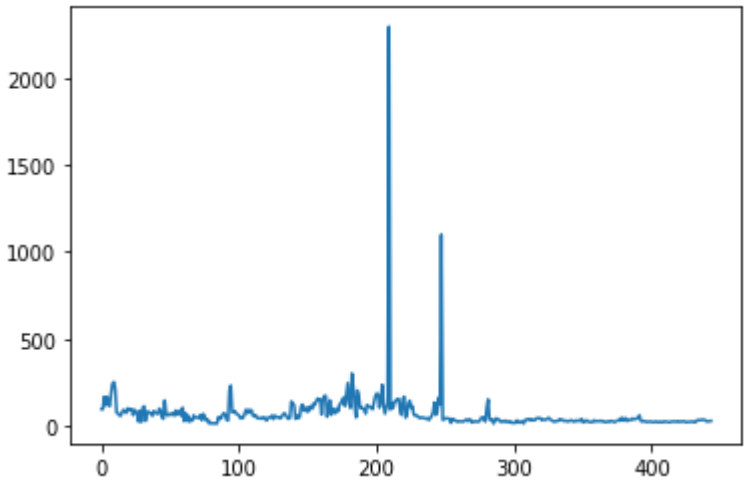
This data is a mix of text and numbers:

	Name	ScientificName	Category	Order	Family	Genus
0	Black-bellied whistling-duck	Dendrocygna autumnalis	Ducks/Geese/Waterfowl	Anseriformes	Anatidae	Dendrocygna
1	Fulvous whistling-duck	Dendrocygna bicolor	Ducks/Geese/Waterfowl	Anseriformes	Anatidae	Dendrocygna

	Name	ScientificName	Category	Order	Family	Genus
2	Snow goose	Anser caerulescens	Ducks/Geese/Waterfowl	Anseriformes	Anatidae	Anser
3	Ross's goose	Anser rossii	Ducks/Geese/Waterfowl	Anseriformes	Anatidae	Anser
4	Greater white-fronted goose	Anser albifrons	Ducks/Geese/Waterfowl	Anseriformes	Anatidae	Anser

Let's start by plotting some of the numeric data using a basic line plot. Suppose you wanted a view of the maximum wingspan for these interesting birds.

```python
wingspan = birds['MaxWingspan']
wingspan.plot()
```



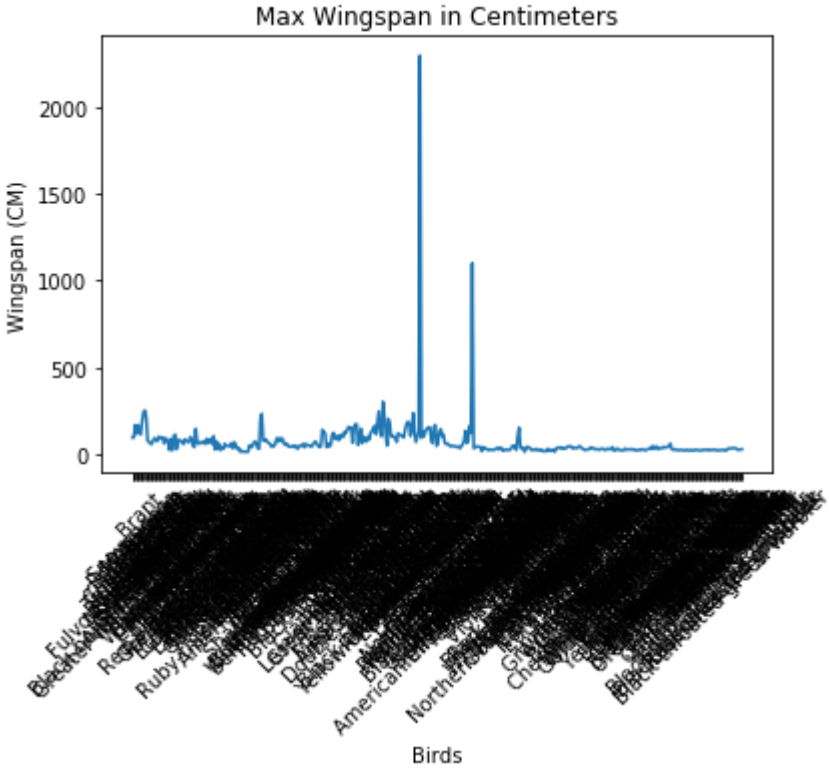
What do you notice immediately? There seems to be at least one outlier - that's quite a wingspan! A 2300 centimeter wingspan equals 23 meters - are there Pterodactyls roaming Minnesota? Let's investigate.

While you could do a quick sort in Excel to find those outliers, which are probably typos, continue the visualization process by working from within the plot.

Add labels to the x-axis to show what kind of birds are in question:

```
plt.title('Max Wingspan in Centimeters')
plt.ylabel('Wingspan (CM)')
```

```
plt.xlabel('Birds')
plt.xticks(rotation=45)
x = birds['Name']
y = birds['MaxWingspan']

plt.plot(x, y)

plt.show()
```



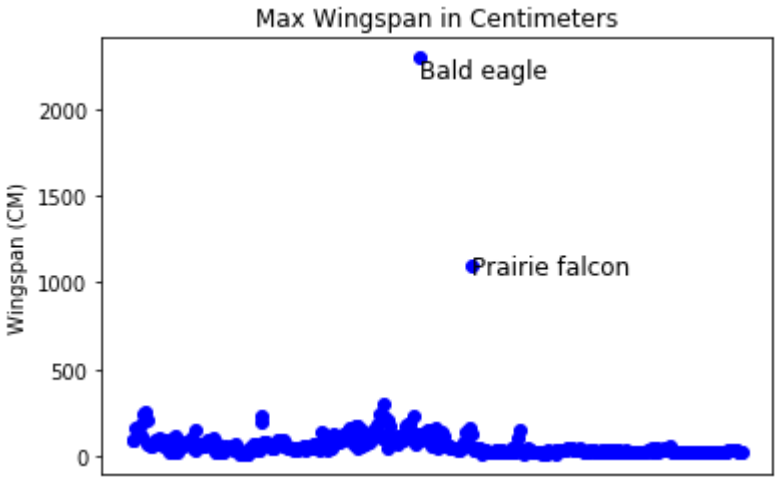
Even with the rotation of the labels set to 45 degrees, there are too many to read. Let's try a different strategy: label only those outliers and set the labels within the chart. You can use a scatter chart to make more room for the labeling:

python

```
plt.title('Max Wingspan in Centimeters')
plt.ylabel('Wingspan (CM)')
plt.tick_params(axis='both',which='both',labelbottom=False,bottom=False)

for i in range(len(birds)):
    x = birds['Name'][i]
    y = birds['MaxWingspan'][i]
    plt.plot(x, y, 'bo')
    if birds['MaxWingspan'][i] > 500:
        plt.text(x, y * (1 - 0.05), birds['Name'][i], fontsize=12)

plt.show()
```

What's going on here? You used `tick_params` to hide the bottom labels and then created a loop over your birds dataset. Plotting the chart with small round blue dots by using `bo`, you checked for any bird with a maximum wingspan over 500 and displayed their label next to the dot if so. You offset the labels a little on the y axis (`y * (1 - 0.05)`) and used the bird name as a label.

What did you discover?

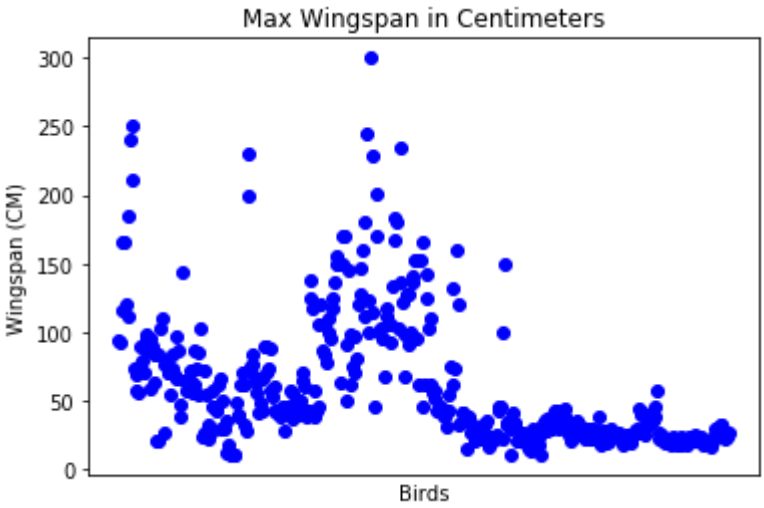


Filter your data

Both the Bald Eagle and the Prairie Falcon, while probably very large birds, appear to be mislabeled, with an extra `0` added to their maximum wingspan. It's unlikely that you'll meet a Bald Eagle with a 25 meter wingspan, but if so, please let us know! Let's create a new dataframe without those two outliers:

```python
plt.title('Max Wingspan in Centimeters')
plt.ylabel('Wingspan (CM)')
plt.xlabel('Birds')
plt.tick_params(axis='both',which='both',labelbottom=False,bottom=False)
for i in range(len(birds)):
    x = birds['Name'][i]
    y = birds['MaxWingspan'][i]
    if birds['Name'][i] not in ['Bald eagle', 'Prairie falcon']:
        plt.plot(x, y, 'bo')
plt.show()
```

By filtering out outliers, your data is now more cohesive and understandable.



Now that we have a cleaner dataset at least in terms of wingspan, let's discover more about these birds.

While line and scatter plots can display information about data values and their distributions, we want to think about the values inherent in this dataset. You could create visualizations to answer the following questions about quantity:

> How many categories of birds are there, and what are their numbers? How many birds are extinct, endangered, rare, or common? How many are there of the various genus and orders in Linnaeus's terminology?

Explore bar charts

Bar charts are practical when you need to show groupings of data. Let's explore the categories of birds that exist in this dataset to see which is the most common by number.

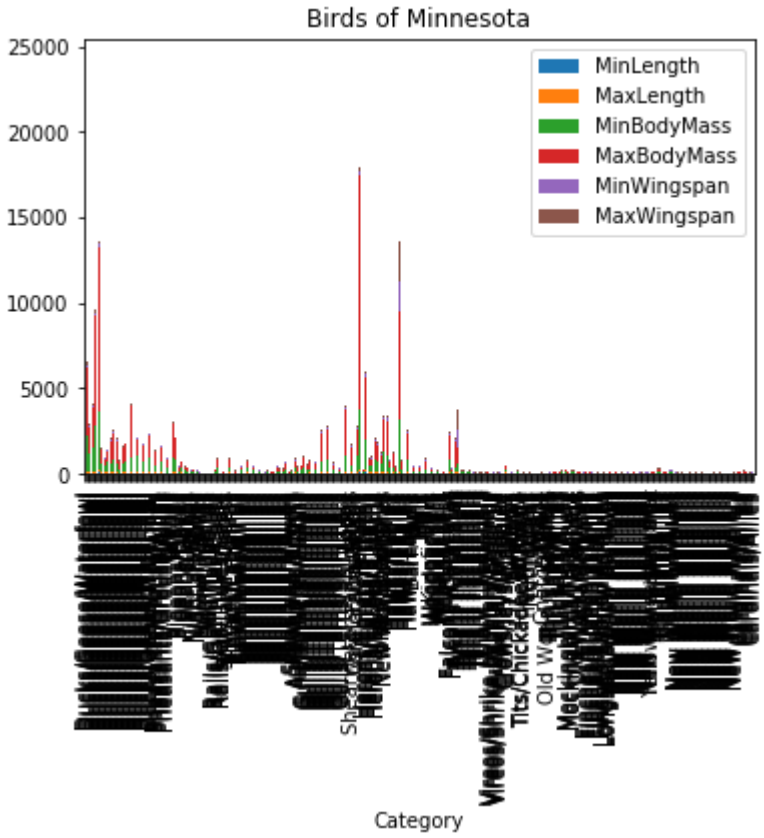
In the notebook file, create a basic bar chart

✅ Note, you can either filter out the two outlier birds we identified in the previous section, edit the typo in their wingspan, or leave them in for these exercises which do not depend on wingspan values.

If you want to create a bar chart, you can select the data you want to focus on. Bar charts can be created from raw data:

python

```
birds.plot(x='Category',
        kind='bar',
        stacked=True,
        title='Birds of Minnesota')
```



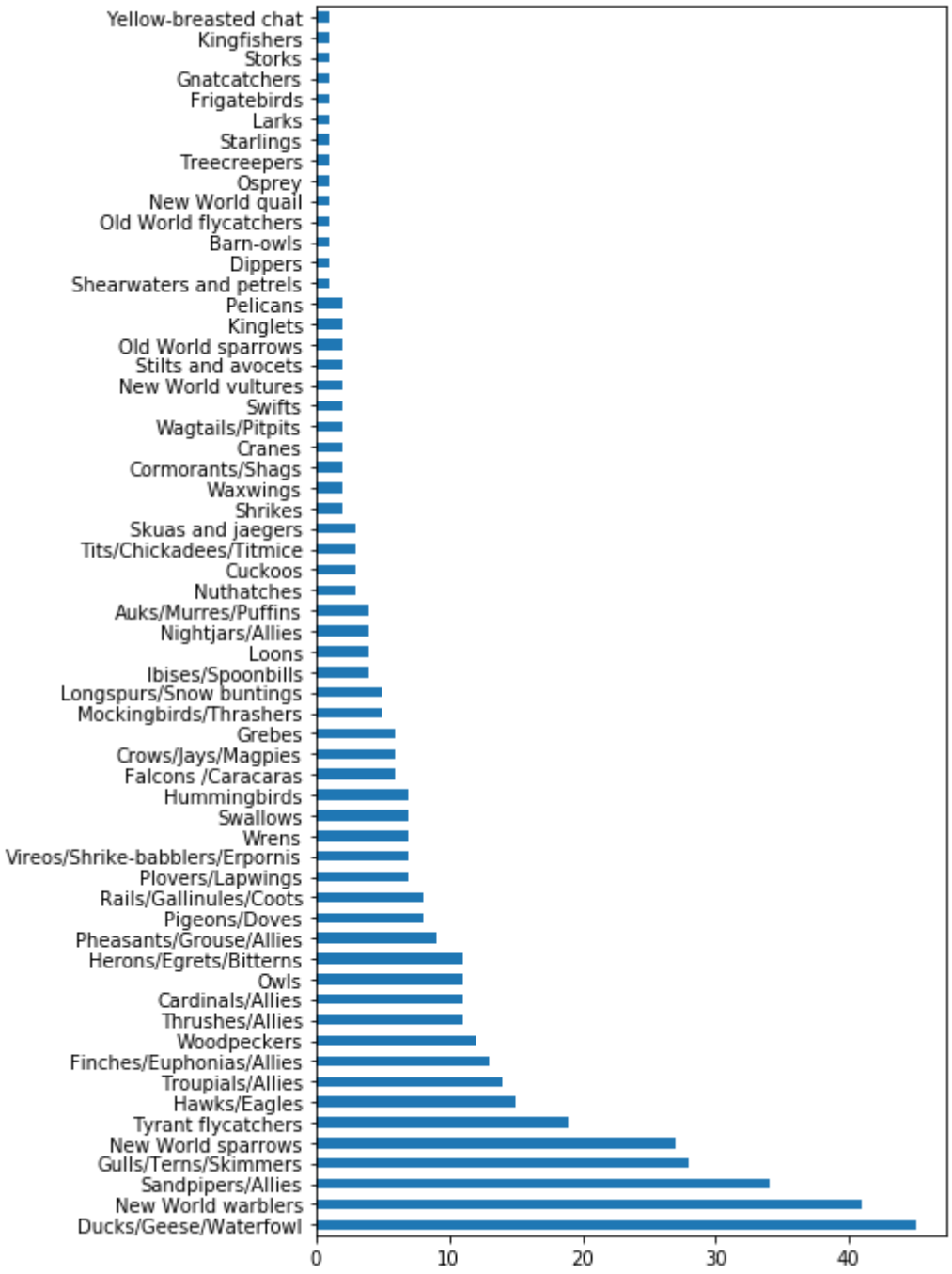
This bar chart, however, is unreadable because there is too much non-grouped data. You need to select only the data that you want to plot, so let's look at the length of birds based on their category.

Filter your data to include only the bird's category.

✅ Notice that that you use Pandas to manage the data, and then let Matplotlib do the charting.

Since there are many categories, you can display this chart vertically and tweak its height to account for all the data:

```python
category_count = birds.value_counts(birds['Category'].values, sort=True)
plt.rcParams['figure.figsize'] = [6, 12]
category_count.plot.barh()
```



This bar chart shows a good view of the number of birds in each category. In a blink of an eye, you see that the largest number of birds in this region are in the Ducks/Geese/Waterfowl category. Minnesota is the 'land of 10,000 lakes' so this isn't surprising!

✅ Try some other counts on this dataset. Does anything surprise you?

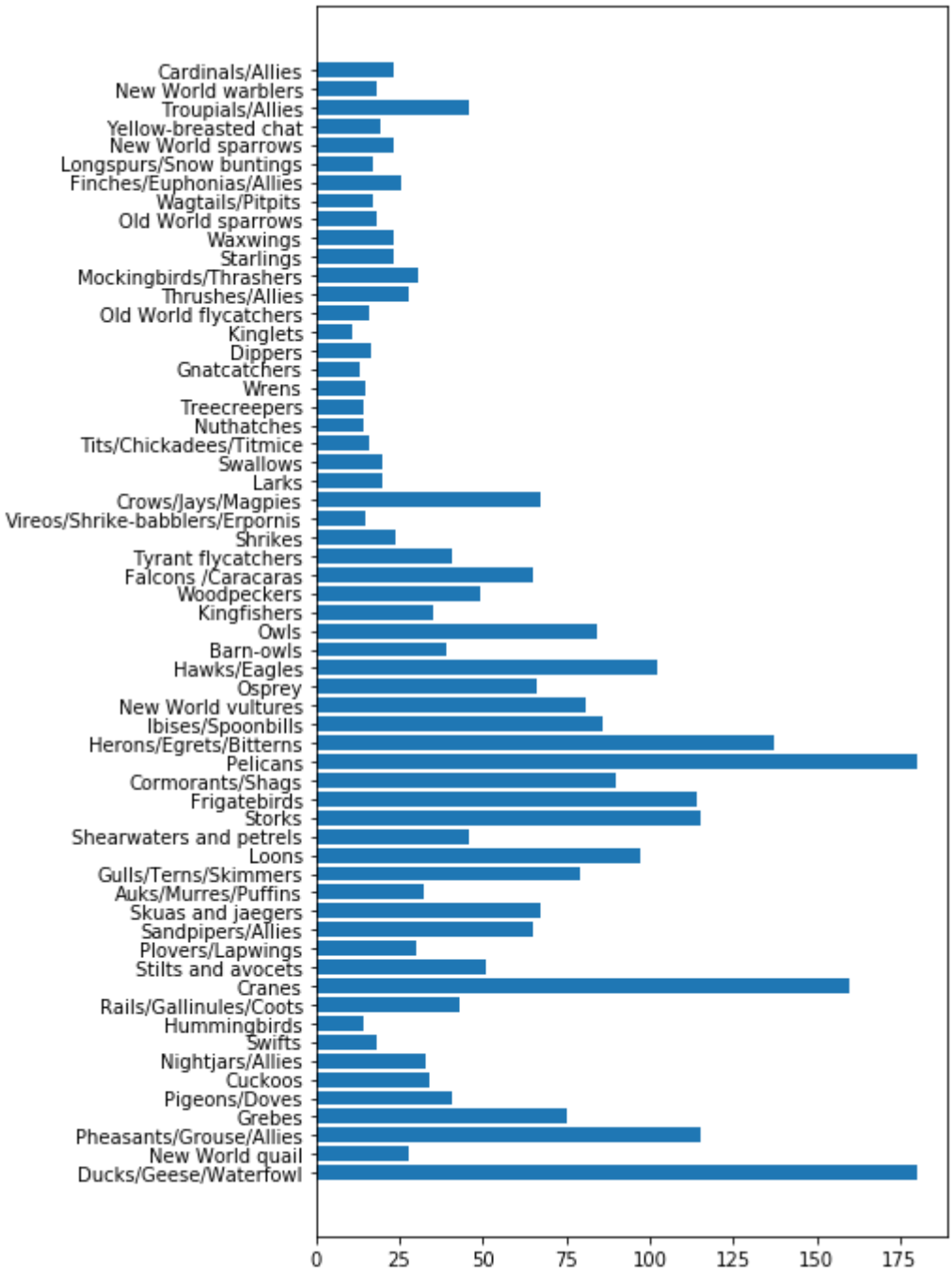
Comparing data

You can try different comparisons of grouped data by creating new axes. Try a comparison of the MaxLength of a bird, based on its category:

python

```
maxlength = birds['MaxLength']
plt.barh(y=birds['Category'], width=maxlength)
```

```
plt.rcParams['figure.figsize'] = [6, 12]
plt.show()
```



Nothing is surprising here: hummingbirds have the least MaxLength compared to Pelicans or Geese. It's good when data makes logical sense!

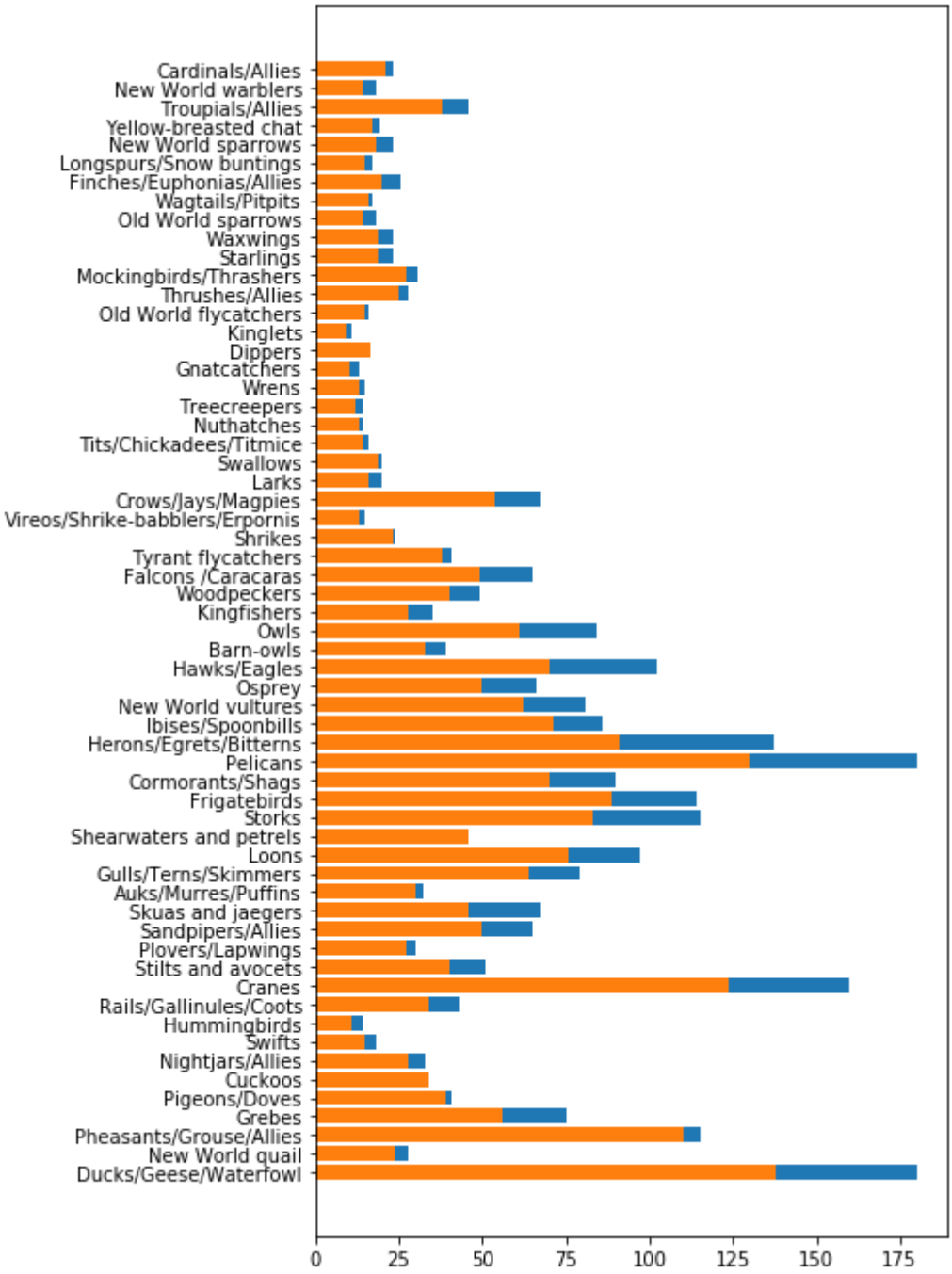
You can create more interesting visualizations of bar charts by superimposing data. Let's superimpose Minimum and Maximum Length on a given bird category:

python

```
minLength = birds['MinLength']
maxLength = birds['MaxLength']
category = birds['Category']

plt.barh(category, maxLength)
plt.barh(category, minLength)
```

```
plt.show()
```

In this plot, you can see the range per bird category of the Minimum Length and Maximum length. You can safely say that, given this data, the bigger the bird, the larger its length range. Fascinating!



🚀 Challenge

This bird dataset offers a wealth of information about different types of birds within a particular ecosystem. Search around the internet and see if you can find other bird-oriented datasets. Practice building charts and graphs around these birds to discover facts you didn't realize.

Post-lecture quiz

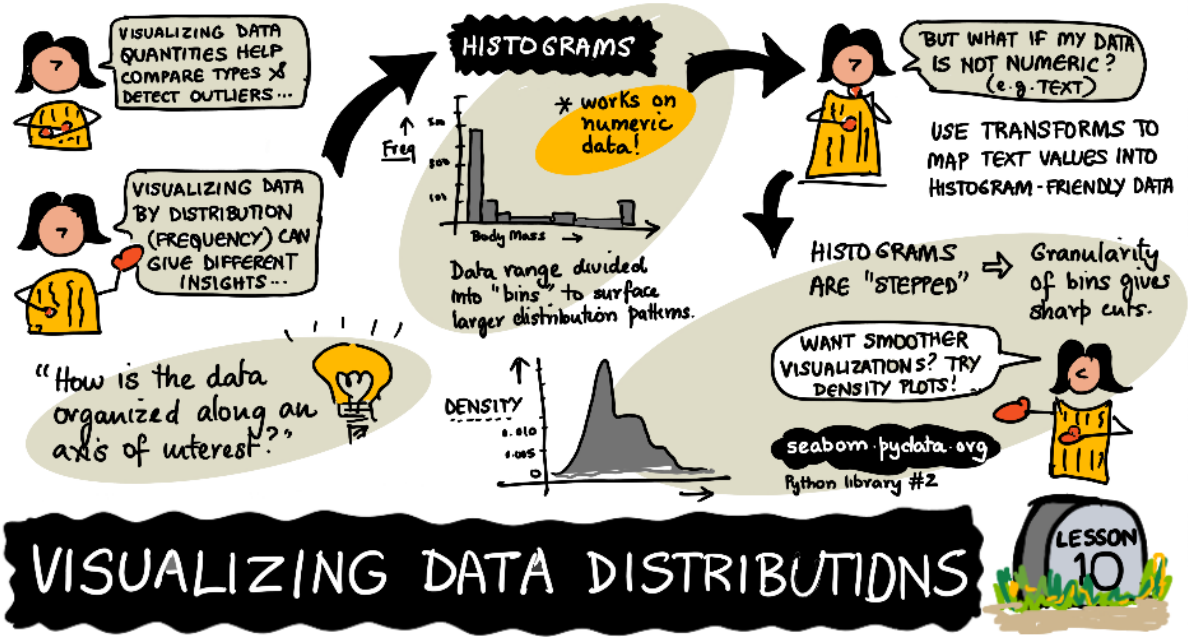
Review & Self Study

This first lesson has given you some information about how to use Matplotlib to visualize quantities. Do some research around other ways to work with datasets for visualization. Plotly is one that we won't cover in these lessons, so take a look at what it can offer.

Assignment

Lines, Scatters, and Bars

Visualizing Distributions



Visualizing Distributions - *Sketchnote by @nitya*

In the previous lesson, you learned some interesting facts about a dataset about the birds of Minnesota. You found some erroneous data by visualizing outliers and looked at the differences between bird categories by their maximum length.

Pre-lecture quiz

Explore the birds dataset

Another way to dig into data is by looking at its distribution, or how the data is organized along an axis. Perhaps, for example, you'd like to learn about the general distribution, for this dataset, of the maximum wingspan or maximum body mass for the birds of Minnesota.

Let's discover some facts about the distributions of data in this dataset. In the *notebook.ipynb* file at the root of this lesson folder, import Pandas, Matplotlib, and your data:

```python
import pandas as pd
import matplotlib.pyplot as plt
birds = pd.read_csv('../../data/birds.csv')
birds.head()
```

In general, you can quickly look at the way data is distributed by using a scatter plot as we did in the previous lesson:

```python
birds.plot(kind='scatter',x='MaxLength',y='Order',figsize=(12,8))

plt.title('Max Length per Order')
plt.ylabel('Order')
plt.xlabel('Max Length')

plt.show()
```

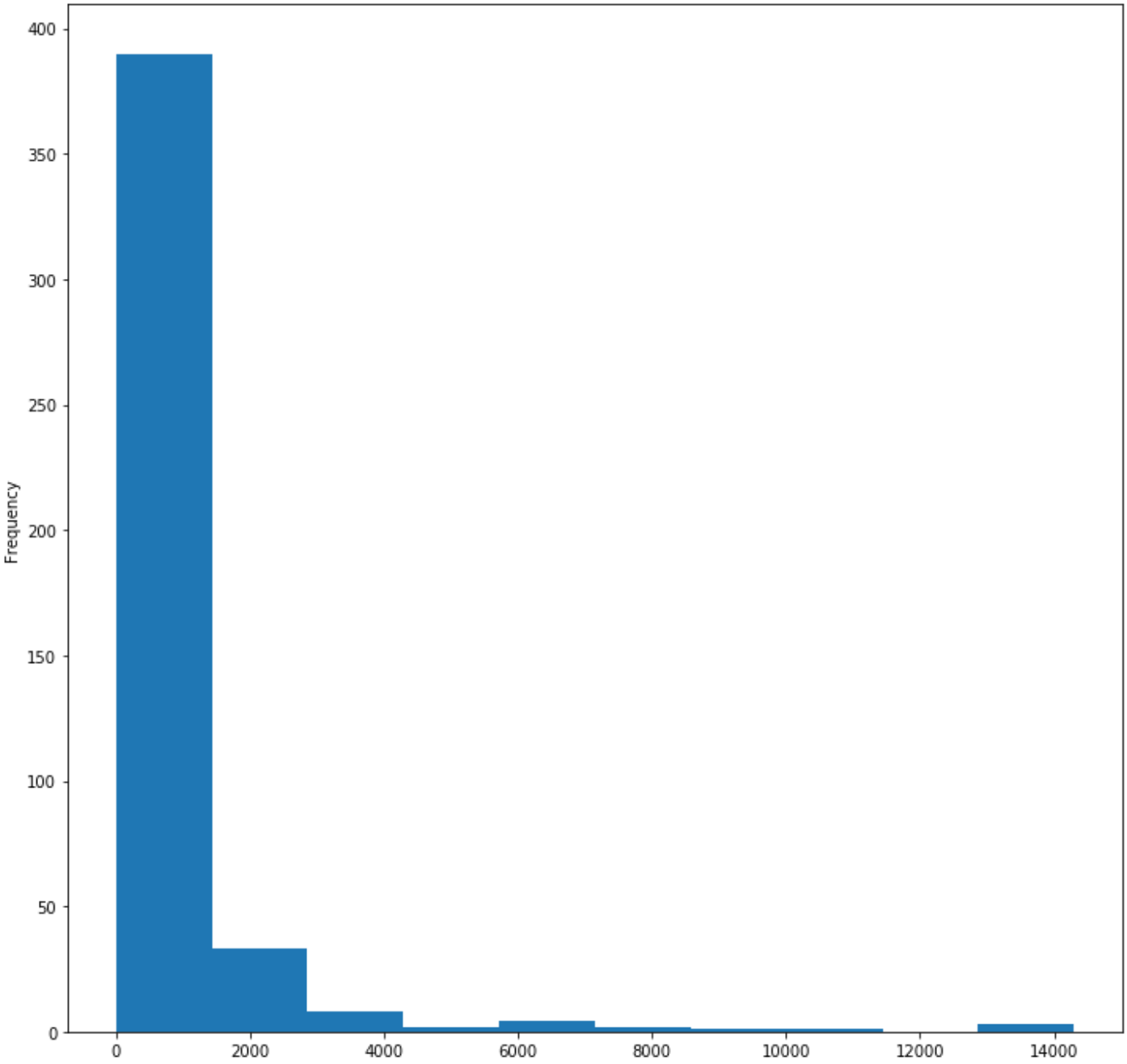
This gives an overview of the general distribution of body length per bird Order, but it is not the optimal way to display true distributions. That task is usually handled by creating a Histogram.

Working with histograms

Matplotlib offers very good ways to visualize data distribution using Histograms. This type of chart is like a bar chart where the distribution can be seen via a rise and fall of the bars. To build a histogram, you need numeric data. To build a Histogram, you can plot a chart defining the kind as 'hist' for

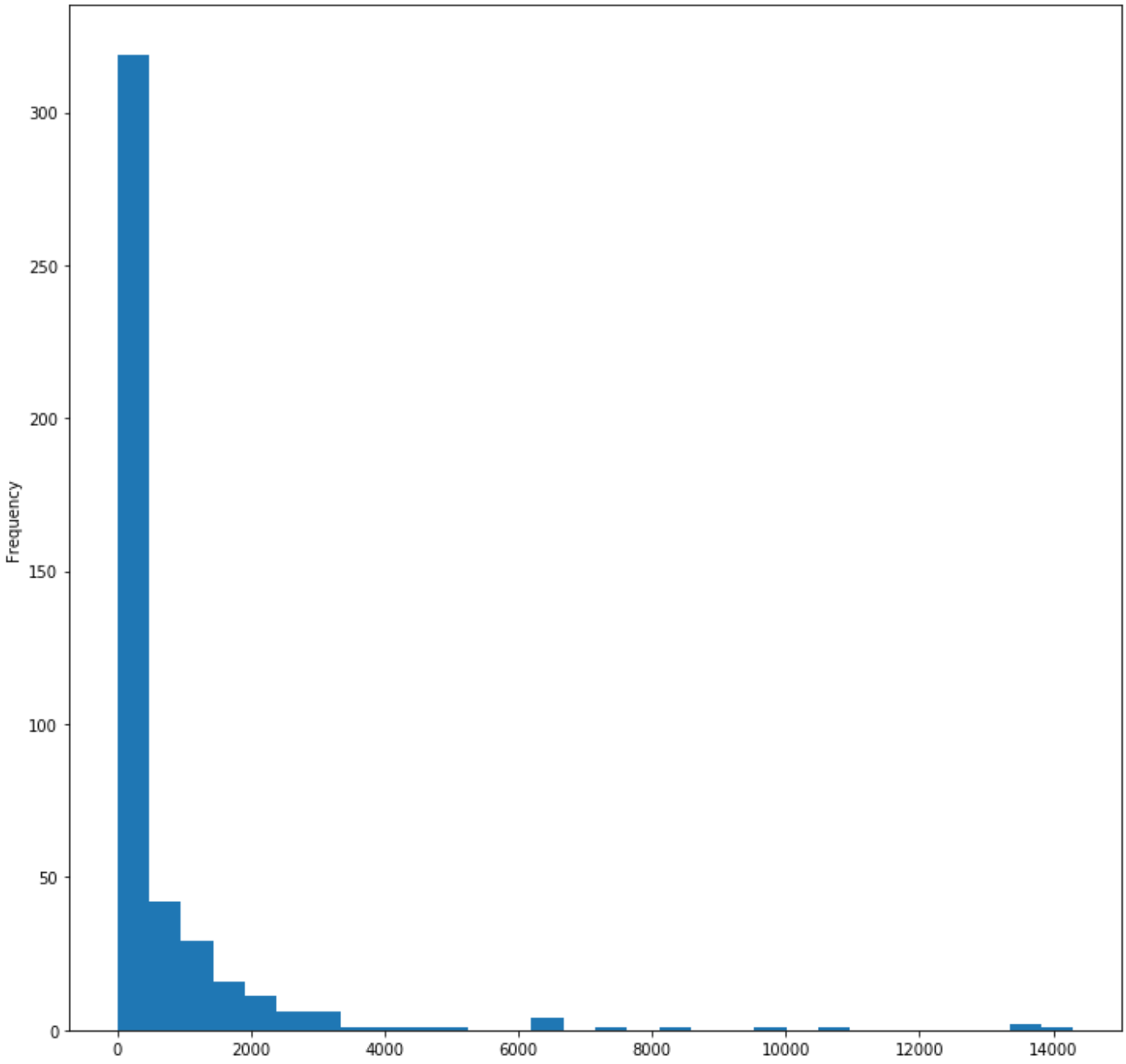
Histogram. This chart shows the distribution of MaxBodyMass for the entire dataset's range of numeric data. By dividing the array of data it is given into smaller bins, it can display the distribution of the data's values:

```python
birds['MaxBodyMass'].plot(kind = 'hist', bins = 10, figsize = (12,12))
plt.show()
```



As you can see, most of the 400+ birds in this dataset fall in the range of under 2000 for their Max Body Mass. Gain more insight into the data by changing the `bins` parameter to a higher number, something like 30:

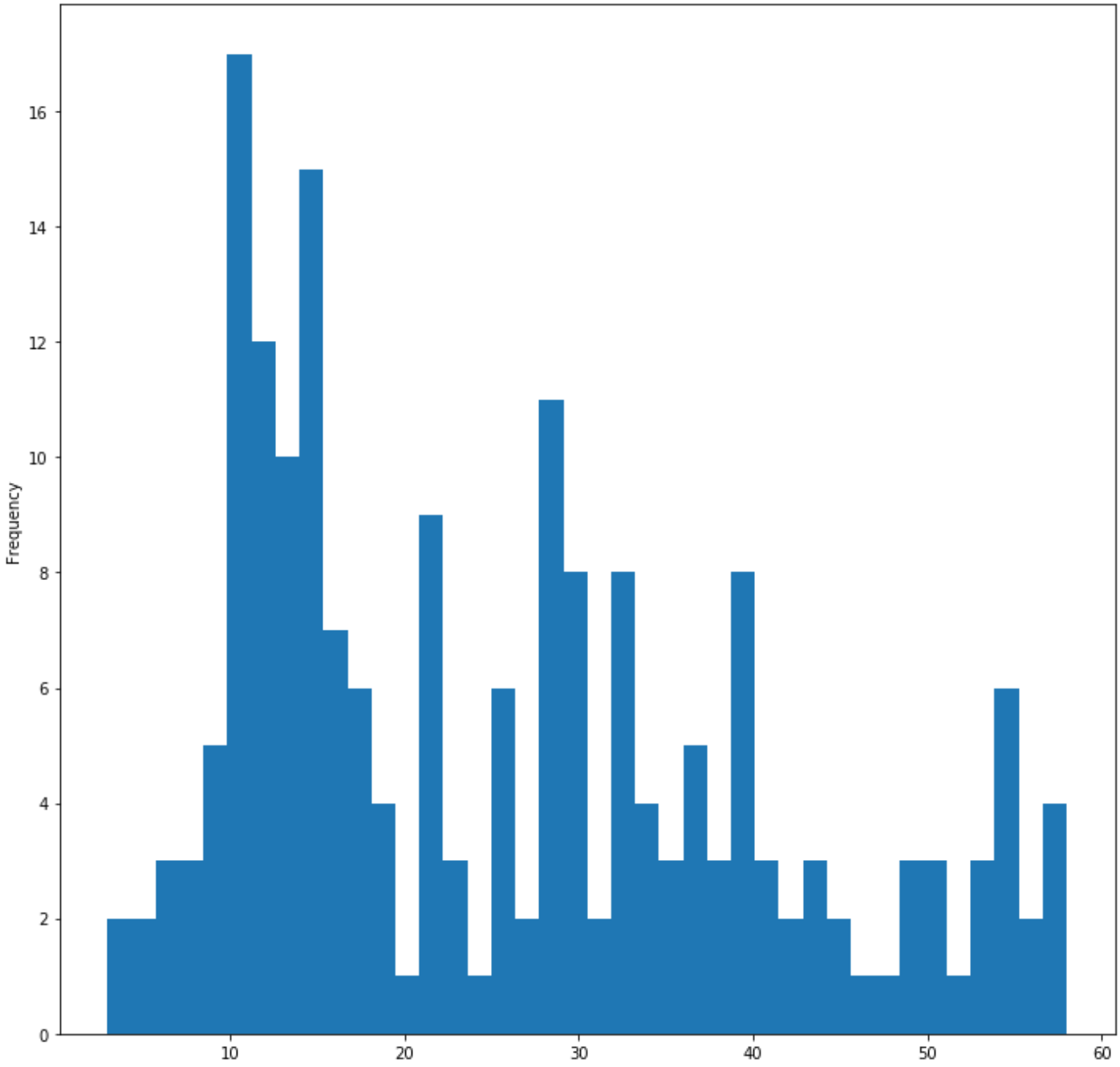
```python
birds['MaxBodyMass'].plot(kind = 'hist', bins = 30, figsize = (12,12))
plt.show()
```



This chart shows the distribution in a bit more granular fashion. A chart less skewed to the left could be created by ensuring that you only select data within a given range:

Filter your data to get only those birds whose body mass is under 60, and show 40 `bins` :

```python
filteredBirds = birds[(birds['MaxBodyMass'] > 1) & (birds['MaxBodyMass'] <
filteredBirds['MaxBodyMass'].plot(kind = 'hist',bins = 40,figsize = (12,12
plt.show()
```



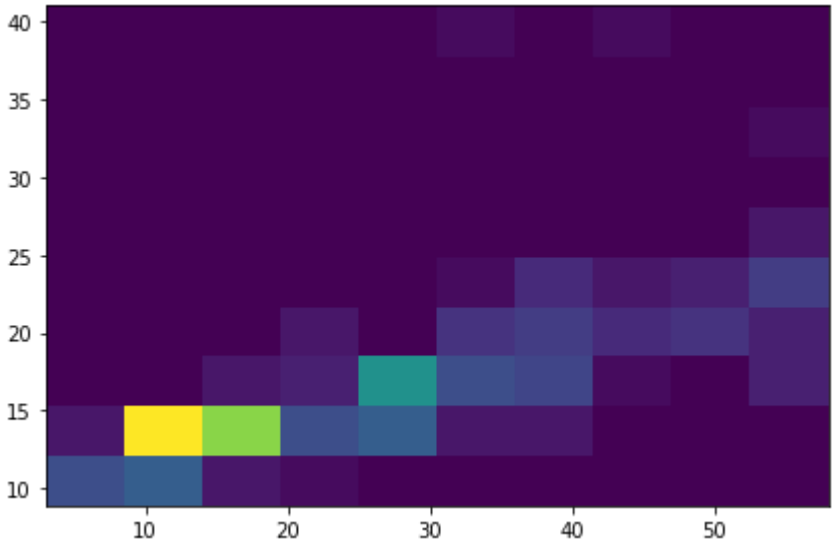
✅ Try some other filters and data points. To see the full distribution of the data, remove the `['MaxBodyMass']` filter to show labeled distributions.

The histogram offers some nice color and labeling enhancements to try as well:

Create a 2D histogram to compare the relationship between two distributions. Let's compare `MaxBodyMass` vs. `MaxLength`. Matplotlib offers a built-in way to show convergence using brighter colors:

```python
x = filteredBirds['MaxBodyMass']
y = filteredBirds['MaxLength']

fig, ax = plt.subplots(tight_layout=True)
hist = ax.hist2d(x, y)
```

There appears to be an expected correlation between these two elements along an expected axis, with one particularly strong point of convergence:



Histograms work well by default for numeric data. What if you need to see distributions according to text data?

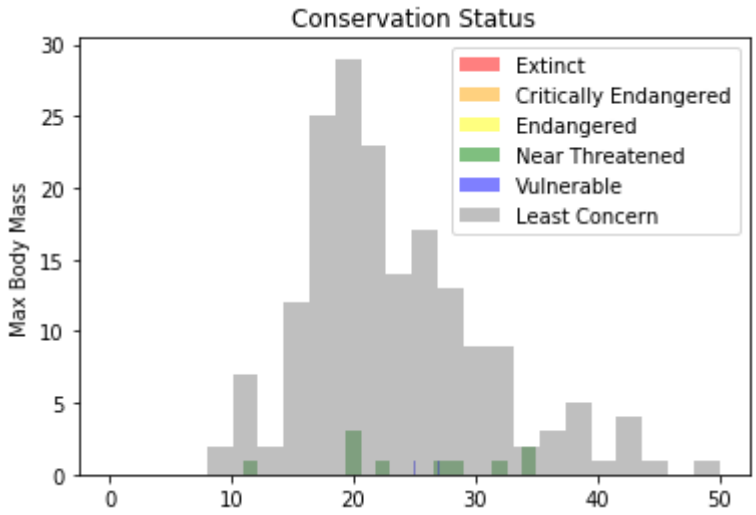
Explore the dataset for distributions using text data

This dataset also includes good information about the bird category and its genus, species, and family as well as its conservation status. Let's dig into this conservation information. What is the distribution of the birds according to their conservation status?

> ✅ In the dataset, several acronyms are used to describe conservation status. These acronyms come from the IUCN Red List Categories, an organization that catalogs species' status.
>
> - CR: Critically Endangered
> - EN: Endangered
> - EX: Extinct
> - LC: Least Concern
> - NT: Near Threatened
> - VU: Vulnerable

These are text-based values so you will need to do a transform to create a histogram. Using the filteredBirds dataframe, display its conservation status alongside its Minimum Wingspan. What do you see?

```python
x1 = filteredBirds.loc[filteredBirds.ConservationStatus=='EX', 'MinWingspan
x2 = filteredBirds.loc[filteredBirds.ConservationStatus=='CR', 'MinWingspan
x3 = filteredBirds.loc[filteredBirds.ConservationStatus=='EN', 'MinWingspan
x4 = filteredBirds.loc[filteredBirds.ConservationStatus=='NT', 'MinWingspan
x5 = filteredBirds.loc[filteredBirds.ConservationStatus=='VU', 'MinWingspan
x6 = filteredBirds.loc[filteredBirds.ConservationStatus=='LC', 'MinWingspan

kwargs = dict(alpha=0.5, bins=20)

plt.hist(x1, **kwargs, color='red', label='Extinct')
plt.hist(x2, **kwargs, color='orange', label='Critically Endangered')
plt.hist(x3, **kwargs, color='yellow', label='Endangered')
plt.hist(x4, **kwargs, color='green', label='Near Threatened')
plt.hist(x5, **kwargs, color='blue', label='Vulnerable')
plt.hist(x6, **kwargs, color='gray', label='Least Concern')

plt.gca().set(title='Conservation Status', ylabel='Max Body Mass')
plt.legend();
```



There doesn't seem to be a good correlation between minimum wingspan and conservation status. Test other elements of the dataset using this method. You can try different filters as well. Do you find any correlation?

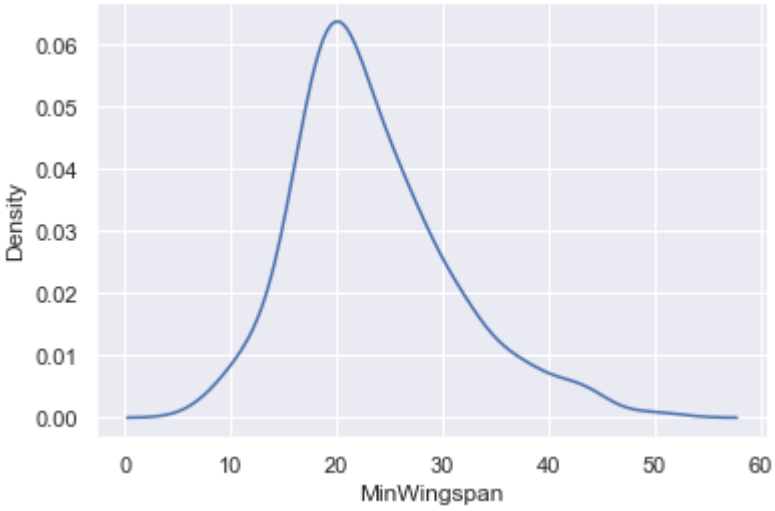
Density plots

You may have noticed that the histograms we have looked at so far are 'stepped' and do not flow smoothly in an arc. To show a smoother density chart, you can try a density plot.

To work with density plots, familiarize yourself with a new plotting library, Seaborn.

Loading Seaborn, try a basic density plot:

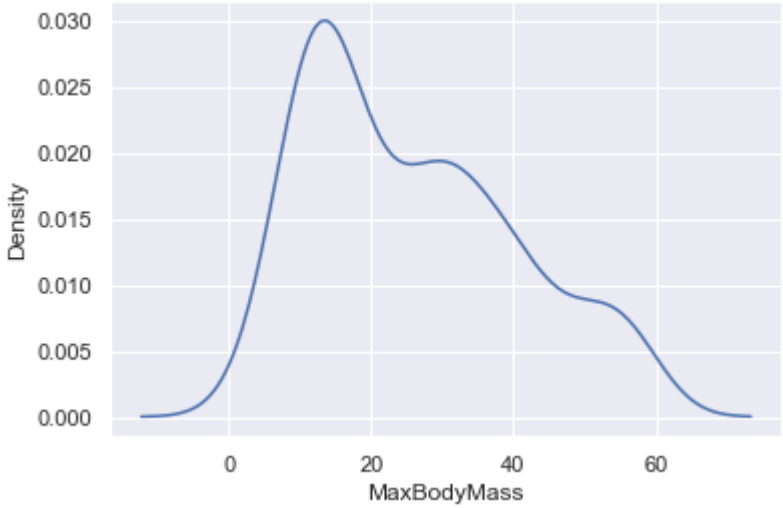
```python
import seaborn as sns
import matplotlib.pyplot as plt
sns.kdeplot(filteredBirds['MinWingspan'])
plt.show()
```



You can see how the plot echoes the previous one for Minimum Wingspan data; it's just a bit smoother. According to Seaborn's documentation, "Relative to a histogram, KDE can produce a plot that is less cluttered and more interpretable, especially when drawing multiple distributions. But it has the potential to introduce distortions if the underlying distribution is bounded or not smooth. Like a histogram, the quality of the representation also depends on the selection of good smoothing parameters." source In other words, outliers as always will make your charts behave badly.

If you wanted to revisit that jagged MaxBodyMass line in the second chart you built, you could smooth it out very well by recreating it using this method:

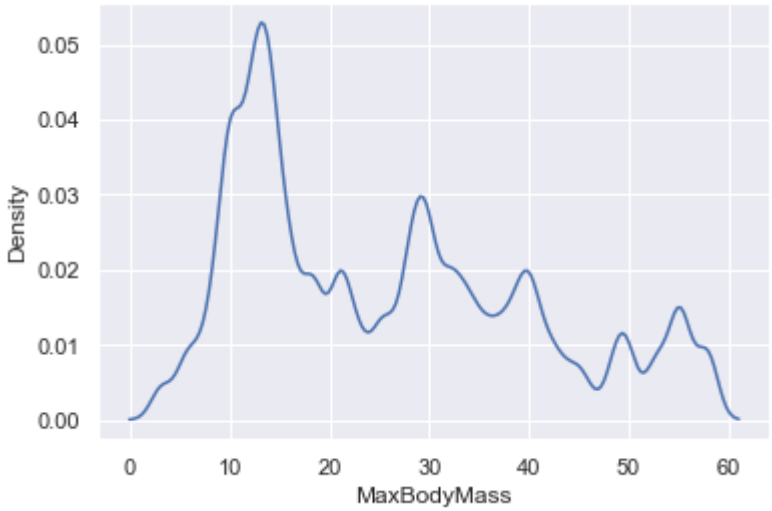
```python
sns.kdeplot(filteredBirds['MaxBodyMass'])
plt.show()
```



If you wanted a smooth, but not too smooth line, edit the `bw_adjust` parameter:

python

```
sns.kdeplot(filteredBirds['MaxBodyMass'], bw_adjust=.2)
plt.show()
```

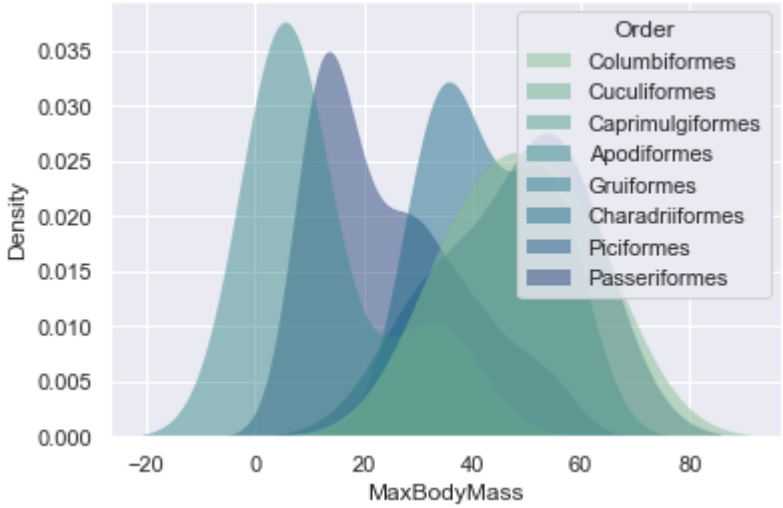


✅ Read about the parameters available for this type of plot and experiment!

This type of chart offers beautifully explanatory visualizations. With a few lines of code, for example, you can show the max body mass density per bird Order:

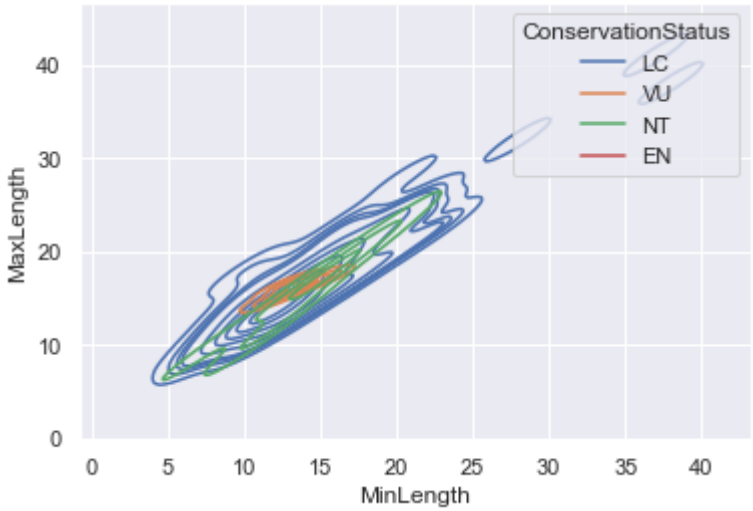
python

```
sns.kdeplot(
   data=filteredBirds, x="MaxBodyMass", hue="Order",
   fill=True, common_norm=False, palette="crest",
   alpha=.5, linewidth=0,
)
```



You can also map the density of several variables in one chart. Text the MaxLength and MinLength of a bird compared to their conservation status:

```python
sns.kdeplot(data=filteredBirds, x="MinLength", y="MaxLength", hue="Conserva
```



Perhaps it's worth researching whether the cluster of 'Vulnerable' birds according to their lengths is meaningful or not.

🚀 Challenge

Histograms are a more sophisticated type of chart than basic scatterplots, bar charts, or line charts. Go on a search on the internet to find good examples of the use of histograms. How are they used, what do they demonstrate, and in what fields or areas of inquiry do they tend to be used?

Post-lecture quiz

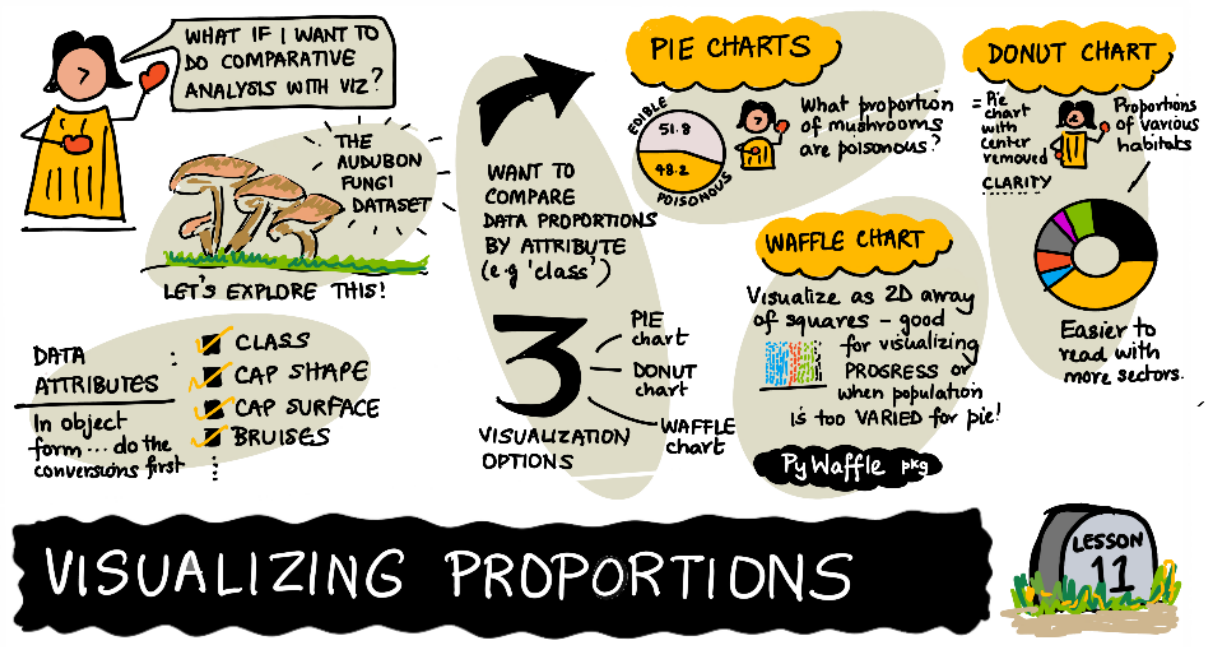
Review & Self Study

In this lesson, you used Matplotlib and started working with Seaborn to show more sophisticated charts. Do some research on `kdeplot` in Seaborn, a "continuous probability density curve in one or more dimensions". Read through the documentation to understand how it works.

Assignment

Apply your skills

Visualizing Proportions



Visualizing Proportions - *Sketchnote by @nitya*

In this lesson, you will use a different nature-focused dataset to visualize proportions, such as how many different types of fungi populate a given dataset about mushrooms. Let's explore these fascinating fungi using a dataset sourced from Audubon listing details about 23 species of gilled mushrooms in the Agaricus and Lepiota families. You will experiment with tasty visualizations such as:

- Pie charts 🥧
- Donut charts 🍩
- Waffle charts 🧇

> 💡 A very interesting project called Charticulator by Microsoft Research offers a free drag and drop interface for data visualizations. In one of their tutorials they also use this mushroom dataset! So you can explore the data and learn the library at the same time: Charticulator tutorial.

Pre-lecture quiz

Get to know your mushrooms 🍄

Mushrooms are very interesting. Let's import a dataset to study them:

```python
import pandas as pd
import matplotlib.pyplot as plt
mushrooms = pd.read_csv('../../data/mushrooms.csv')
mushrooms.head()
```

A table is printed out with some great data for analysis:

class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gil co
Poisonous	Convex	Smooth	Brown	Bruises	Pungent	Free	Close	Narrow	Bla
Edible	Convex	Smooth	Yellow	Bruises	Almond	Free	Close	Broad	Bla
Edible	Bell	Smooth	White	Bruises	Anise	Free	Close	Broad	Br
Poisonous	Convex	Scaly	White	Bruises	Pungent	Free	Close	Narrow	Br

Right away, you notice that all the data is textual. You will have to convert this data to be able to use it in a chart. Most of the data, in fact, is represented as an object:

```python
print(mushrooms.select_dtypes(["object"]).columns)
```

The output is:

```output
Index(['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
       'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
       'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
       'stalk-surface-below-ring', 'stalk-color-above-ring',
       'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
       'ring-type', 'spore-print-color', 'population', 'habitat'],
      dtype='object')
```

Take this data and convert the 'class' column to a category:

```python
cols = mushrooms.select_dtypes(["object"]).columns
mushrooms[cols] = mushrooms[cols].astype('category')
```

Now, if you print out the mushrooms data, you can see that it has been grouped into categories according to the poisonous/edible class:

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	st sh
class										
Edible	4208	4208	4208	4208	4208	4208	4208	4208	4208	4
Poisonous	3916	3916	3916	3916	3916	3916	3916	3916	3916	3

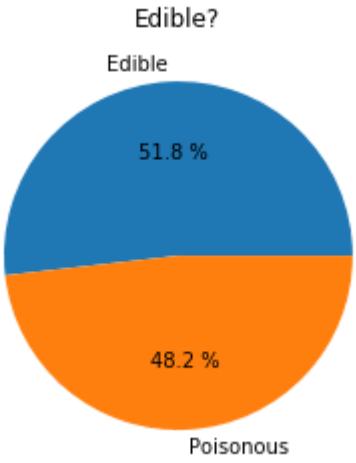
If you follow the order presented in this table to create your class category labels, you can build a pie chart:

Pie!

```python
labels=['Edible','Poisonous']
plt.pie(edibleclass['population'],labels=labels,autopct='%.1f %%')
```

```
plt.title('Edible?')
plt.show()
```

Voila, a pie chart showing the proportions of this data according to these two classes of mushrooms. It's quite important to get the order of the labels correct, especially here, so be sure to verify the order with which the label array is built!



Donuts!

A somewhat more visually interesting pie chart is a donut chart, which is a pie chart with a hole in the middle. Let's look at our data using this method.

Take a look at the various habitats where mushrooms grow:

python

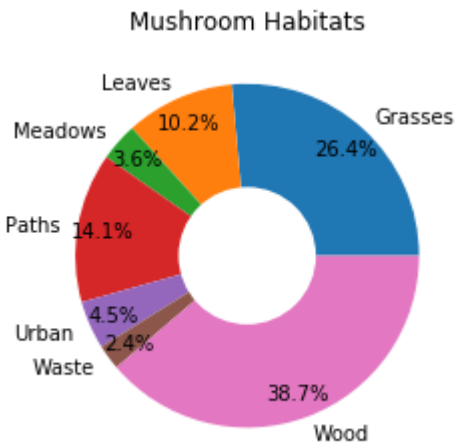
```
habitat=mushrooms.groupby(['habitat']).count()
habitat
```

Here, you are grouping your data by habitat. There are 7 listed, so use those as labels for your donut chart:

python

```
labels=['Grasses','Leaves','Meadows','Paths','Urban','Waste','Wood']

plt.pie(habitat['class'], labels=labels,
        autopct='%1.1f%%', pctdistance=0.85)

center_circle = plt.Circle((0, 0), 0.40, fc='white')
fig = plt.gcf()

fig.gca().add_artist(center_circle)
```

```
plt.title('Mushroom Habitats')

plt.show()
```



This code draws a chart and a center circle, then adds that center circle in the chart. Edit the width of the center circle by changing `0.40` to another value.

Donut charts can be tweaked in several ways to change the labels. The labels in particular can be highlighted for readability. Learn more in the docs.

Now that you know how to group your data and then display it as a pie or donut, you can explore other types of charts. Try a waffle chart, which is just a different way of exploring quantity.

Waffles!

A 'waffle' type chart is a different way to visualize quantities as a 2D array of squares. Try visualizing the different quantities of mushroom cap colors in this dataset. To do this, you need to install a helper library called PyWaffle and use Matplotlib:

python

```
pip install pywaffle
```

Select a segment of your data to group:

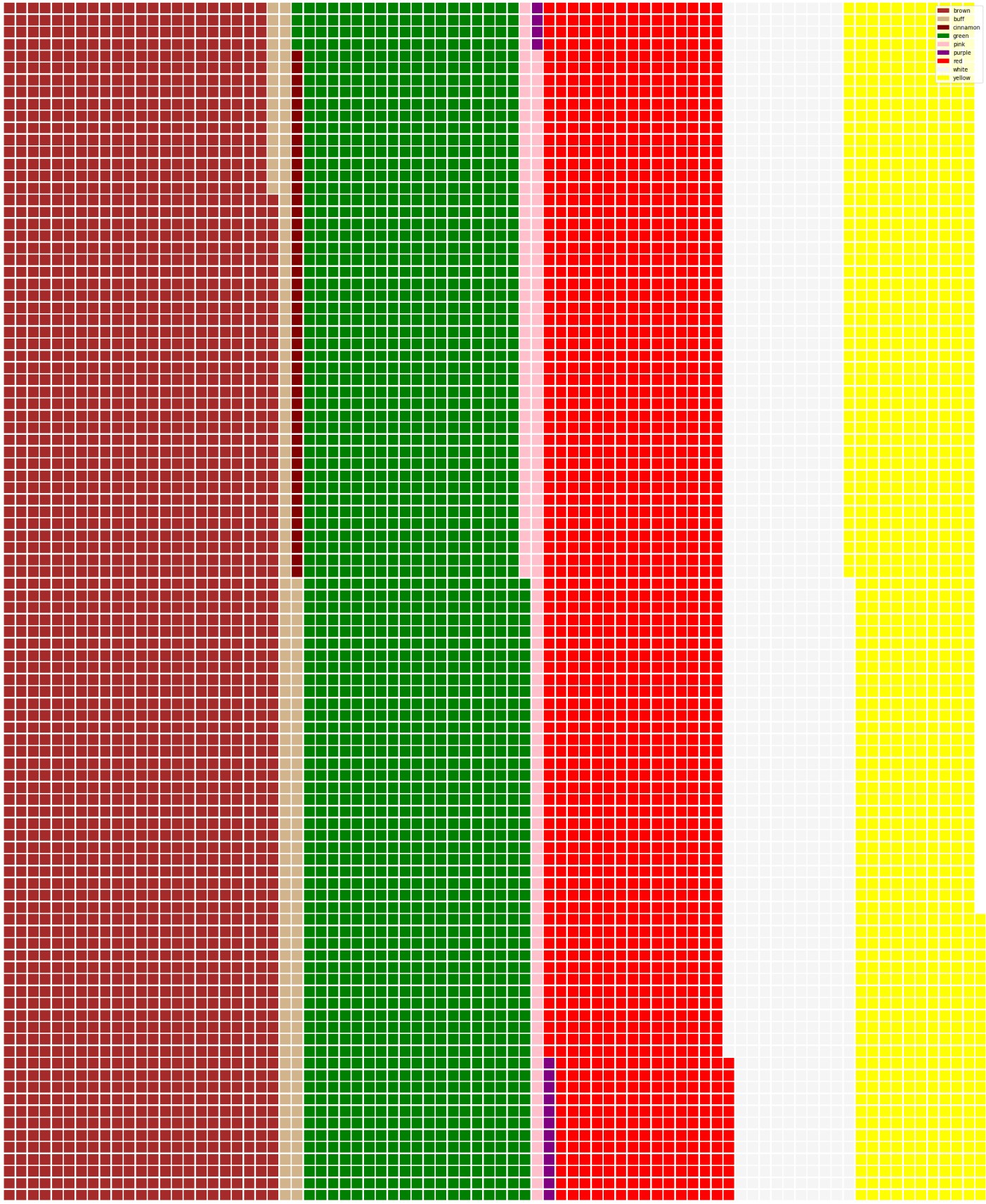
python

```
capcolor=mushrooms.groupby(['cap-color']).count()
capcolor
```

Create a waffle chart by creating labels and then grouping your data:

```python
import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle

data ={'color': ['brown', 'buff', 'cinnamon', 'green', 'pink', 'purple', '
    'amount': capcolor['class']
     }

df = pd.DataFrame(data)

fig = plt.figure(
    FigureClass = Waffle,
    rows = 100,
    values = df.amount,
    labels = list(df.color),
    figsize = (30,30),
    colors=["brown", "tan", "maroon", "green", "pink", "purple", "red", "wh
)
```

Using a waffle chart, you can plainly see the proportions of cap colors of this mushrooms dataset. Interestingly, there are many green-capped mushrooms!



✅ Pywaffle supports icons within the charts that use any icon available in Font Awesome. Do some experiments to create an even more interesting waffle chart using icons instead of squares.

In this lesson, you learned three ways to visualize proportions. First, you need to group your data into categories and then decide which is the best way to display the data - pie, donut, or waffle. All are delicious and gratify the user with an instant snapshot of a dataset.

🚀 Challenge

Try recreating these tasty charts in Charticulator.

Post-lecture quiz

Review & Self Study

Sometimes it's not obvious when to use a pie, donut, or waffle chart. Here are some articles to read on this topic:

https://www.beautiful.ai/blog/battle-of-the-charts-pie-chart-vs-donut-chart

https://medium.com/@hypsypops/pie-chart-vs-donut-chart-showdown-in-the-ring-5d24fd86a9ce

https://www.mit.edu/~mbarker/formula1/f1help/11-ch-c6.htm

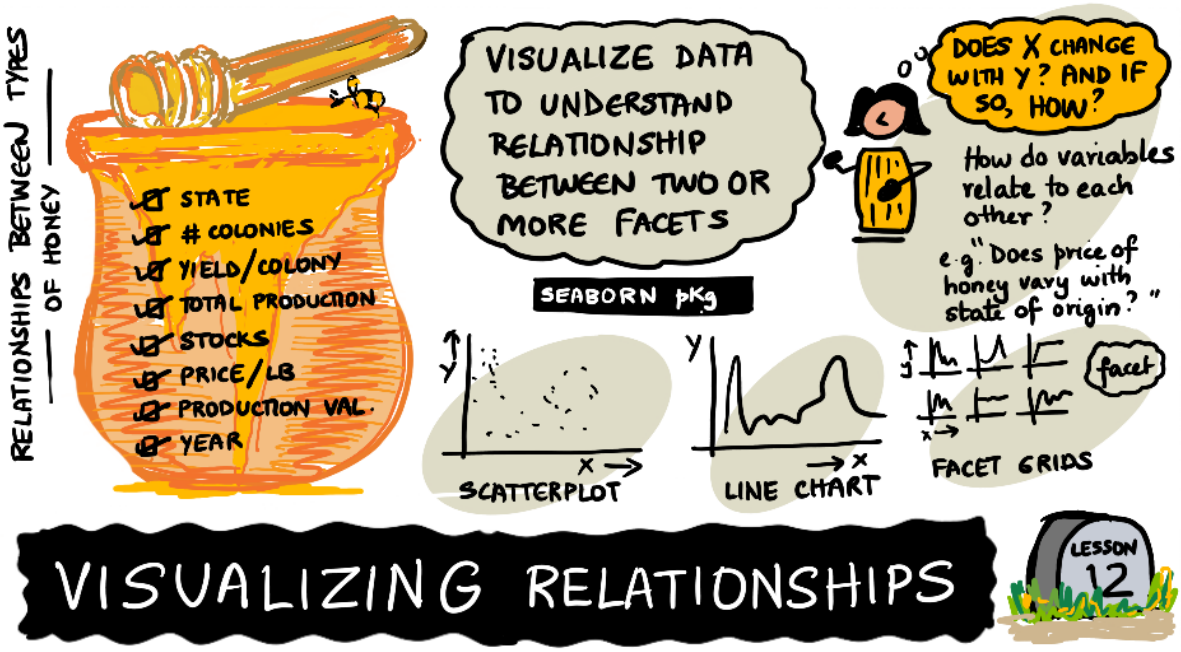
https://medium.datadriveninvestor.com/data-visualization-done-the-right-way-with-tableau-waffle-chart-fdf2a19be402

Do some research to find more information on this sticky decision.

Assignment

Try it in Excel

Visualizing Relationships: All About Honey



Visualizing Relationships - *Sketchnote by @nitya*

Continuing with the nature focus of our research, let's discover interesting visualizations to show the relationships between various types of honey, according to a dataset derived from the United States Department of Agriculture.

This dataset of about 600 items displays honey production in many U.S. states. So, for example, you can look at the number of colonies, yield per colony, total production, stocks, price per pound, and value of the honey produced in a given state from 1998-2012, with one row per year for each state.

It will be interesting to visualize the relationship between a given state's production per year and, for example, the price of honey in that state. Alternately, you could visualize the relationship between states' honey yield per colony. This year span covers the devastating 'CCD' or 'Colony Collapse Disorder' first seen in 2006 (http://npic.orst.edu/envir/ccd.html), so it is a poignant dataset to study. 🐝

Pre-lecture quiz

In this lesson, you can use Seaborn, which you use before, as a good library to visualize relationships between variables. Particularly interesting is the use of Seaborn's `relplot` function that allows scatter plots and line plots to quickly visualize 'statistical relationships', which allow the data scientist to better understand how variables relate to each other.

Scatterplots

Use a scatterplot to show how the price of honey has evolved, year over year, per state. Seaborn, using `relplot`, conveniently groups the state data and displays data points for both categorical and numeric data.

Let's start by importing the data and Seaborn:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
honey = pd.read_csv('../../data/honey.csv')
honey.head()
```

You notice that the honey data has several interesting columns, including year and price per pound. Let's explore this data, grouped by U.S. state:

state	numcol	yieldpercol	totalprod	stocks	priceperlb	prodvalue	year
AL	16000	71	1136000	159000	0.72	818000	1998
AZ	55000	60	3300000	1485000	0.64	2112000	1998
AR	53000	65	3445000	1688000	0.59	2033000	1998
CA	450000	83	37350000	12326000	0.62	23157000	1998
CO	27000	72	1944000	1594000	0.7	1361000	1998

Create a basic scatterplot to show the relationship between the price per pound of honey and its U.S. state of origin. Make the `y` axis tall enough to display all the states:

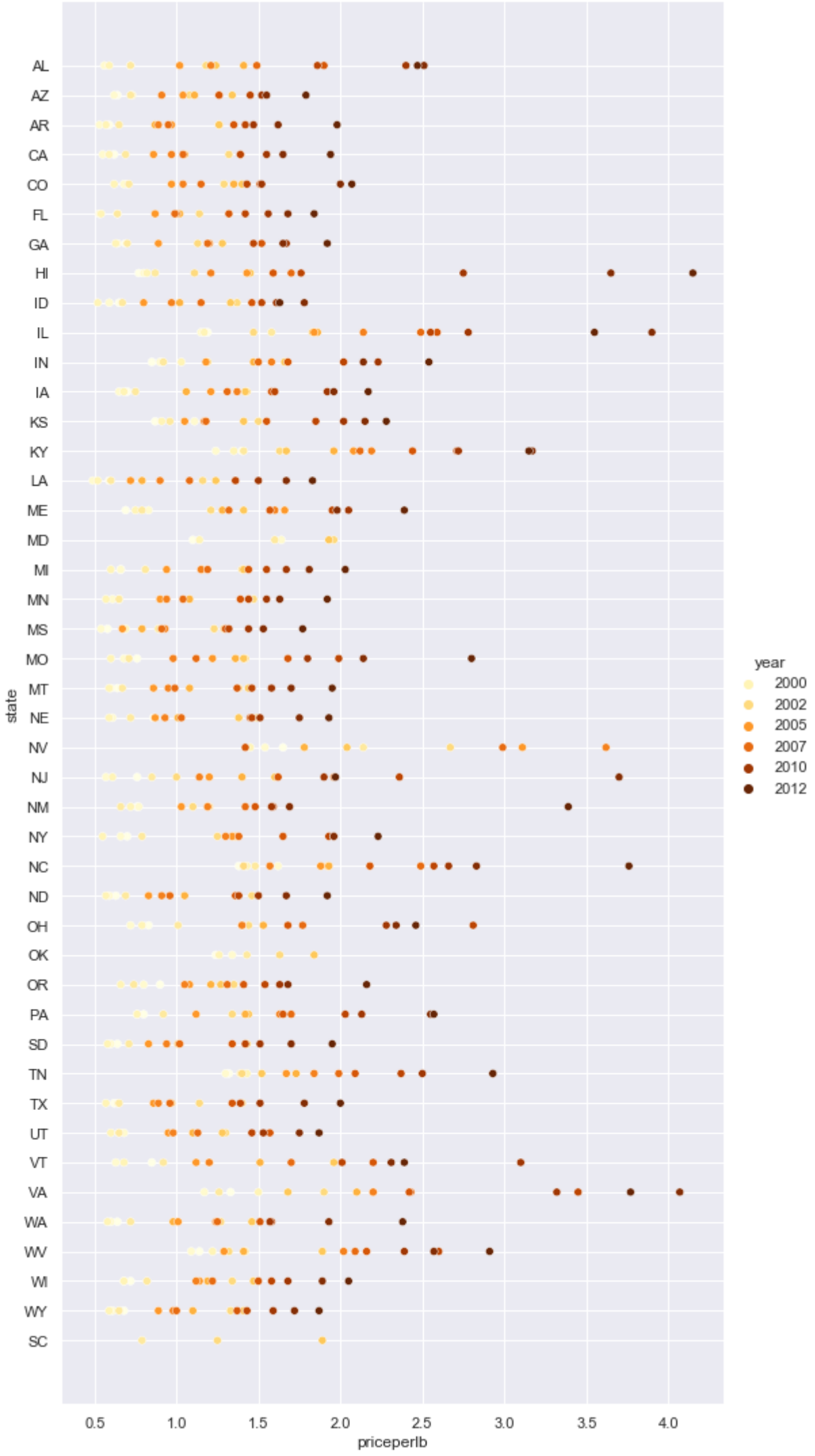
```python
sns.relplot(x="priceperlb", y="state", data=honey, height=15, aspect=.5);
```


AL
AZ
AR
CA
CO
FL
GA
HI
ID
IL
IN
IA
KS
KY
LA
ME
MD
MI
MN
MS
MO
MT
NE
NV
NJ
NM
NY
NC
ND
OH
OK
OR
PA
SD
TN
TX
UT
VT
VA
WA
WV
WI
WY
SC
state
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
priceperlb

Now, show the same data with a honey color scheme to show how the price evolves over the years. You can do this by adding a 'hue' parameter to show the change, year over year:

> ✅ Learn more about the color palettes you can use in Seaborn - try a beautiful rainbow color scheme!

```python
sns.relplot(x="priceperlb", y="state", hue="year", palette="YlOrBr", data=I
```


AL
AZ
AR
CA
CO
FL
GA
HI
ID
IL
IN
IA
KS
KY
LA
ME
MD
MI
MN
MS
MO
MT
NE
NV
NJ
NM
NY
NC
ND
OH
OK
OR
PA
SD
TN
TX
UT
VT
VA
WA
WV
WI
WY
SC
state
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
priceperlb
year
2000
2002
2005
2007
2010
2012

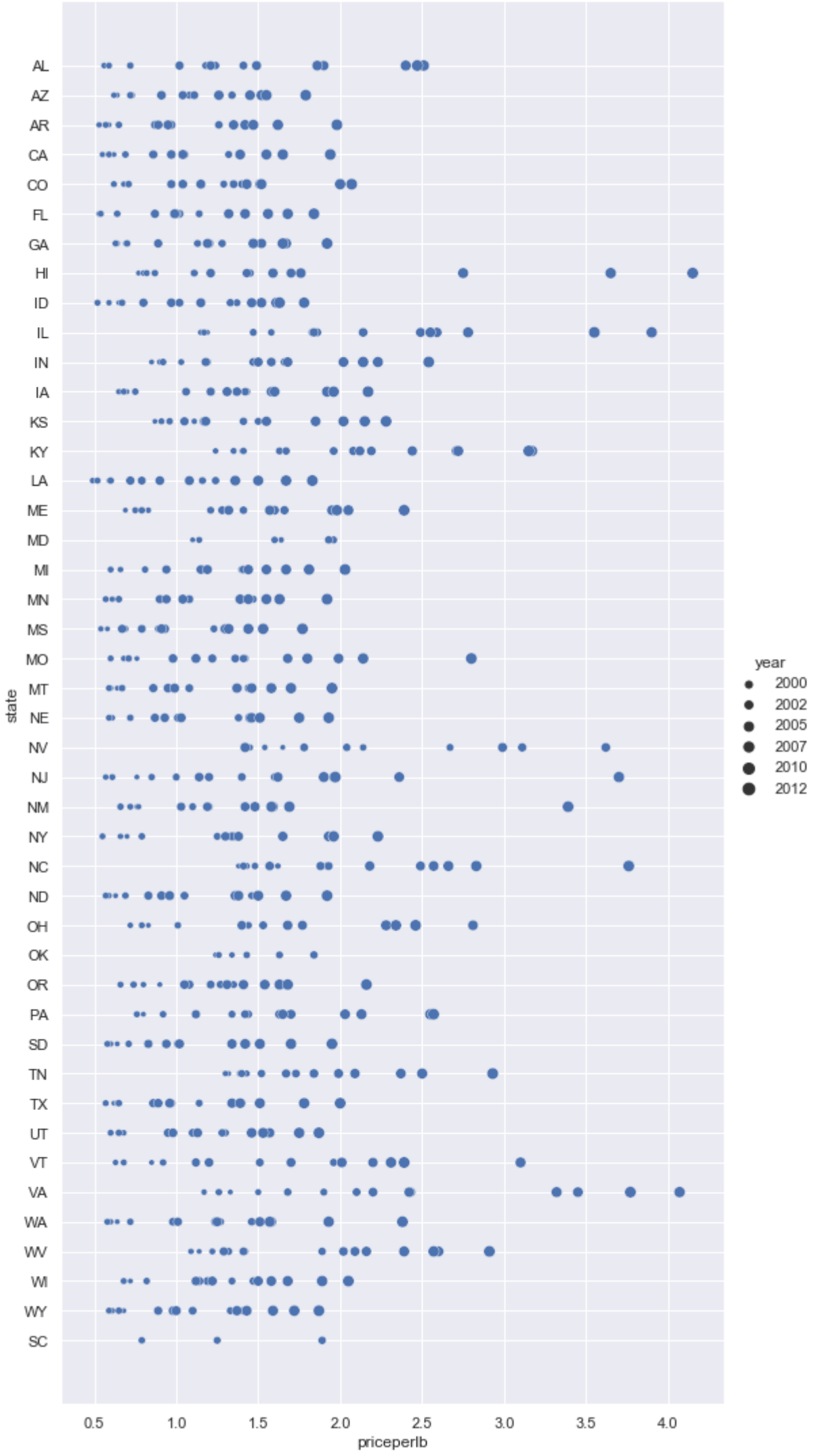
With this color scheme change, you can see that there's obviously a strong progression over the years in terms of honey price per pound. Indeed, if you look at a sample set in the data to verify (pick a given state, Arizona for example) you can see a pattern of price increases year over year, with few exceptions:

state	numcol	yieldpercol	totalprod	stocks	priceperlb	prodvalue	year
AZ	55000	60	3300000	1485000	0.64	2112000	1998
AZ	52000	62	3224000	1548000	0.62	1999000	1999
AZ	40000	59	2360000	1322000	0.73	1723000	2000
AZ	43000	59	2537000	1142000	0.72	1827000	2001
AZ	38000	63	2394000	1197000	1.08	2586000	2002
AZ	35000	72	2520000	983000	1.34	3377000	2003
AZ	32000	55	1760000	774000	1.11	1954000	2004
AZ	36000	50	1800000	720000	1.04	1872000	2005
AZ	30000	65	1950000	839000	0.91	1775000	2006
AZ	30000	64	1920000	902000	1.26	2419000	2007
AZ	25000	64	1600000	336000	1.26	2016000	2008
AZ	20000	52	1040000	562000	1.45	1508000	2009
AZ	24000	77	1848000	665000	1.52	2809000	2010
AZ	23000	53	1219000	427000	1.55	1889000	2011
AZ	22000	46	1012000	253000	1.79	1811000	2012

Another way to visualize this progression is to use size, rather than color. For colorblind users, this might be a better option. Edit your visualization to show an increase of price by an increase in dot circumference:

```python
sns.relplot(x="priceperlb", y="state", size="year", data=honey, height=15,
```

You can see the size of the dots gradually increasing.


AL
AZ
AR
CA
CO
FL
GA
HI
ID
IL
IN
IA
KS
KY
LA
ME
MD
MI
MN
MS
MO
MT
NE
NV
NJ
NM
NY
NC
ND
OH
OK
OR
PA
SD
TN
TX
UT
VT
VA
WA
WV
WI
WY
SC
state
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
priceperlb
year
2000
2002
2005
2007
2010
2012

Is this a simple case of supply and demand? Due to factors such as climate change and colony collapse, is there less honey available for purchase year over year, and thus the price increases?

To discover a correlation between some of the variables in this dataset, let's explore some line charts.

Line charts

Question: Is there a clear rise in price of honey per pound year over year? You can most easily discover that by creating a single line chart:

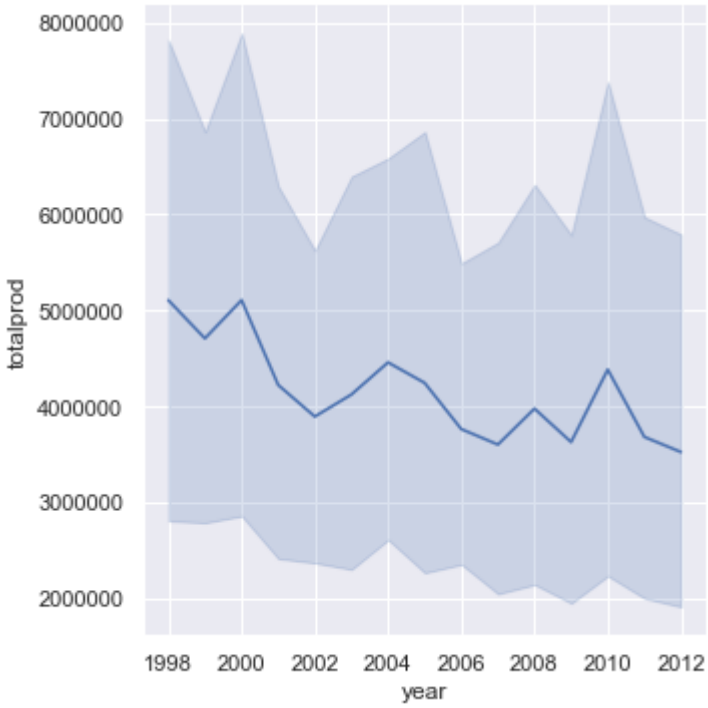
```python
sns.relplot(x="year", y="priceperlb", kind="line", data=honey);
```

Answer: Yes, with some exceptions around the year 2003:

✅ Because Seaborn is aggregating data around one line, it displays "the multiple measurements at each x value by plotting the mean and the 95% confidence interval around the mean". source. This time-consuming behavior can be disabled by adding `ci=None` .

Question: Well, in 2003 can we also see a spike in the honey supply? What if you look at total production year over year?

```python
sns.relplot(x="year", y="totalprod", kind="line", data=honey);
```



Answer: Not really. If you look at total production, it actually seems to have increased in that particular year, even though generally speaking the amount of honey being produced is in decline during these years.

Question: In that case, what could have caused that spike in the price of honey around 2003?

To discover this, you can explore a facet grid.

Facet grids

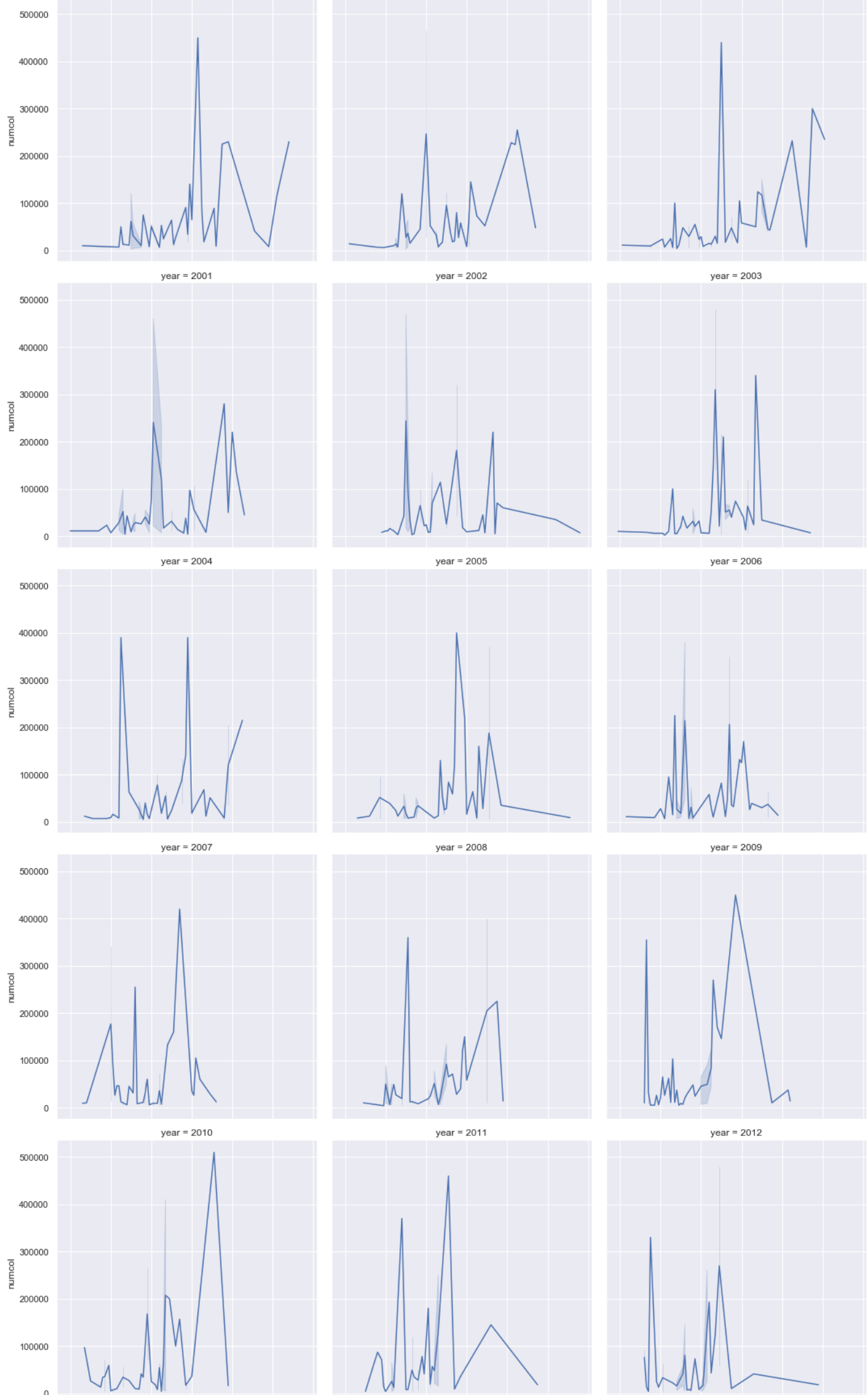
Facet grids take one facet of your dataset (in our case, you can choose 'year' to avoid having too many facets produced). Seaborn can then make a plot for each of those facets of your chosen x and y coordinates for more easy visual comparison. Does 2003 stand out in this type of comparison?

Create a facet grid by continuing to use `relplot` as recommended by Seaborn's documentation.

python

```
sns.relplot(
    data=honey,
    x="yieldpercol", y="numcol",
    col="year",
    col_wrap=3,
    kind="line"
```

In this visualization, you can compare the yield per colony and number of colonies year over year, side by side with a wrap set at 3 for the columns:


year = 2001
year = 2002
year = 2003
year = 2004
year = 2005
year = 2006
year = 2007
year = 2008
year = 2009
year = 2010
year = 2011
year = 2012
numcol
0
100000
200000
300000
400000
500000

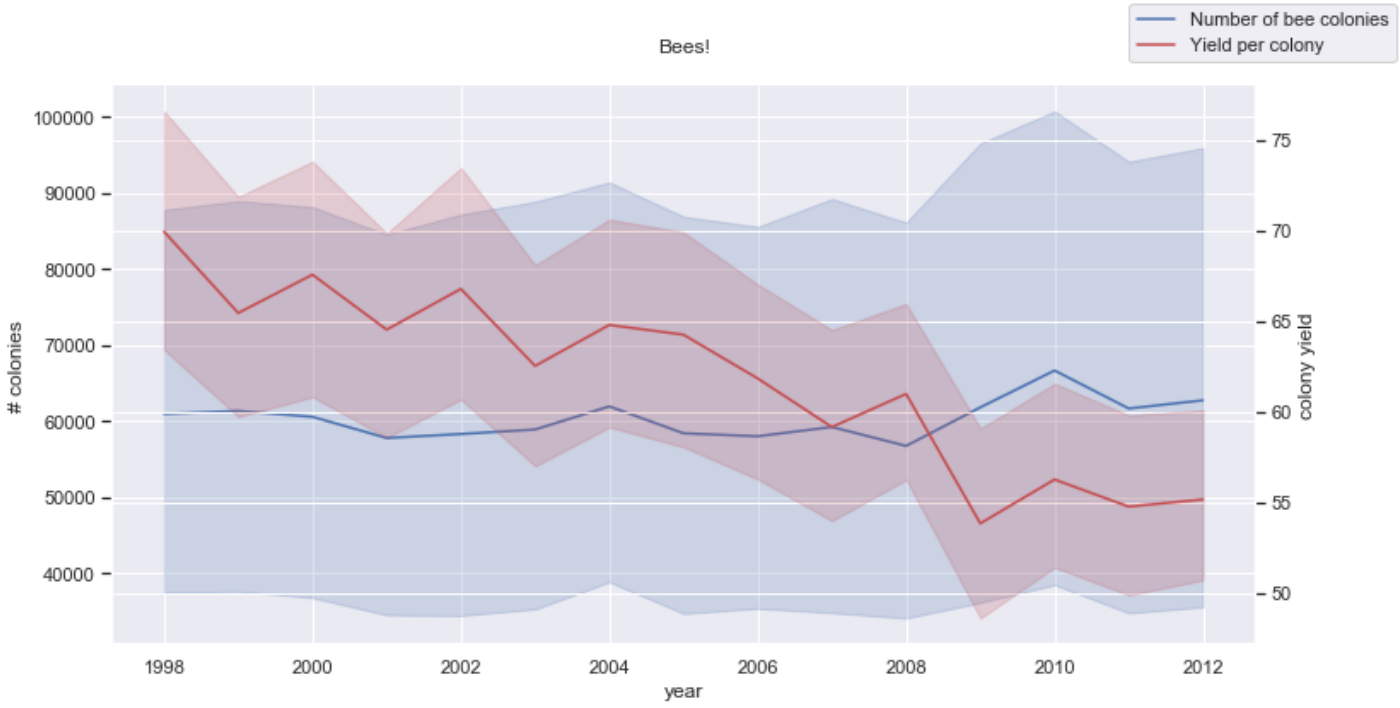
For this dataset, nothing particularly stands out with regards to the number of colonies and their yield, year over year and state over state. Is there a different way to look at finding a correlation between these two variables?

Dual-line Plots

Try a multiline plot by superimposing two lineplots on top of each other, using Seaborn's 'despine' to remove their top and right spines, and using `ax.twinx` derived from Matplotlib. Twinx allows a chart to share the x axis and display two y axes. So, display the yield per colony and number of colonies, superimposed:

python

```
fig, ax = plt.subplots(figsize=(12,6))
lineplot = sns.lineplot(x=honey['year'], y=honey['numcol'], data=honey,
                        label = 'Number of bee colonies', legend=False)
sns.despine()
plt.ylabel('# colonies')
plt.title('Honey Production Year over Year');

ax2 = ax.twinx()
lineplot2 = sns.lineplot(x=honey['year'], y=honey['yieldpercol'], ax=ax2, 
                         label ='Yield per colony', legend=False)
sns.despine(right=False)
plt.ylabel('colony yield')
ax.figure.legend();
```



While nothing jumps out to the eye around the year 2003, it does allow us to end this lesson on a little happier note: while there are overall a declining number of colonies, their numbers might seem to be stabilizing and their yield per colony is actually increasing, even with fewer bees.

Go, bees, go!

🐝❤️

🚀 Challenge

In this lesson, you learned a bit more about other uses of scatterplots and line grids, including facet grids. Challenge yourself to create a facet grid using a different dataset, maybe one you used prior to these lessons. Note how long they take to create and how you need to be careful about how many grids you need to draw using these techniques.

Post-lecture quiz

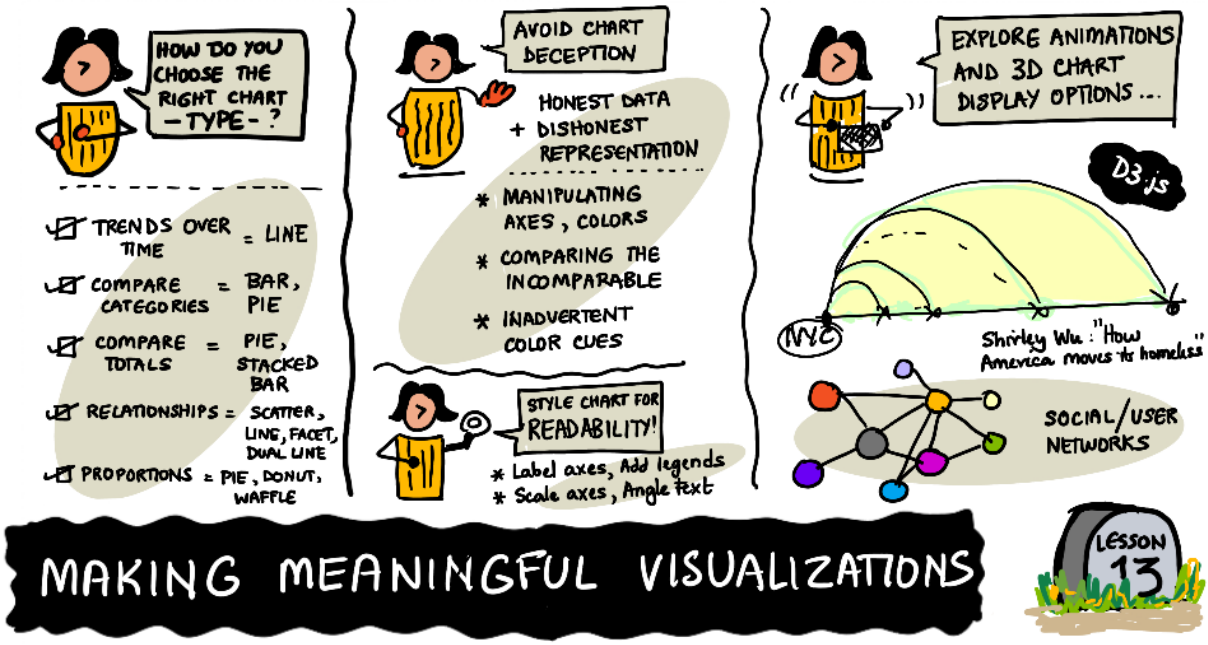
Review & Self Study

Line plots can be simple or quite complex. Do a bit of reading in the Seaborn documentation on the various ways you can build them. Try to enhance the line charts you built in this lesson with other methods listed in the docs.

Assignment

Dive into the beehive

Making Meaningful Visualizations



Meaningful Visualizations - *Sketchnote by @nitya*

> "If you torture the data long enough, it will confess to anything" -- Ronald Coase

One of the basic skills of a data scientist is the ability to create a meaningful data visualization that helps answer questions you might have. Prior to visualizing your data, you need to ensure that it has been cleaned and prepared, as you did in prior lessons. After that, you can start deciding how best to present the data.

In this lesson, you will review:

1. How to choose the right chart type
2. How to avoid deceptive charting
3. How to work with color
4. How to style your charts for readability
5. How to build animated or 3D charting solutions

6. How to build a creative visualization

Pre-Lecture Quiz

Choose the right chart type

In previous lessons, you experimented with building all kinds of interesting data visualizations using Matplotlib and Seaborn for charting. In general, you can select the right kind of chart for the question you are asking using this table:

You need to:	You should use:
Show data trends over time	Line
Compare categories	Bar, Pie
Compare totals	Pie, Stacked Bar
Show relationships	Scatter, Line, Facet, Dual Line
Show distributions	Scatter, Histogram, Box
Show proportions	Pie, Donut, Waffle

> ✅ Depending on the makeup of your data, you might need to convert it from text to numeric to get a given chart to support it.

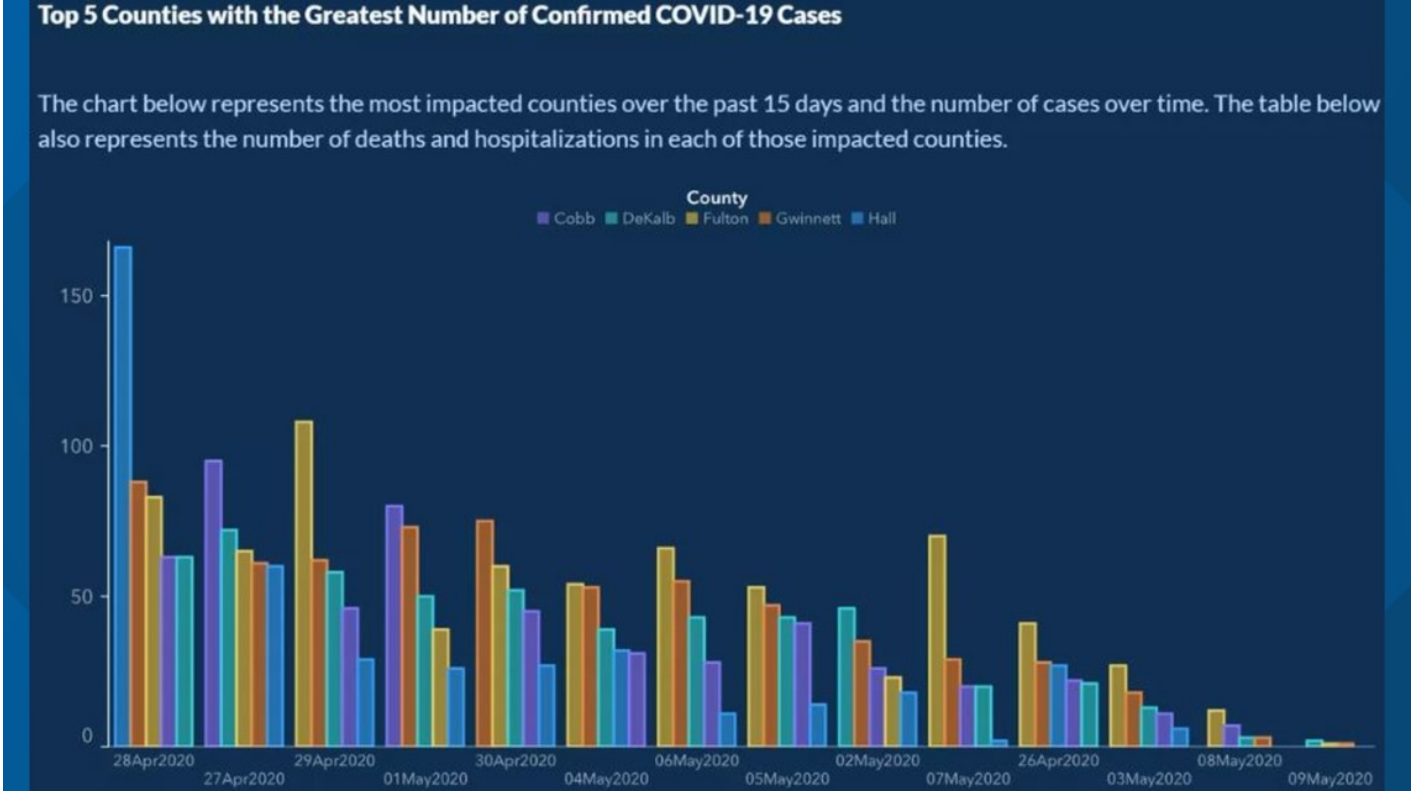
Avoid deception

Even if a data scientist is careful to choose the right chart for the right data, there are plenty of ways that data can be displayed in a way to prove a point, often at the cost of undermining the data itself. There are many examples of deceptive charts and infographics!

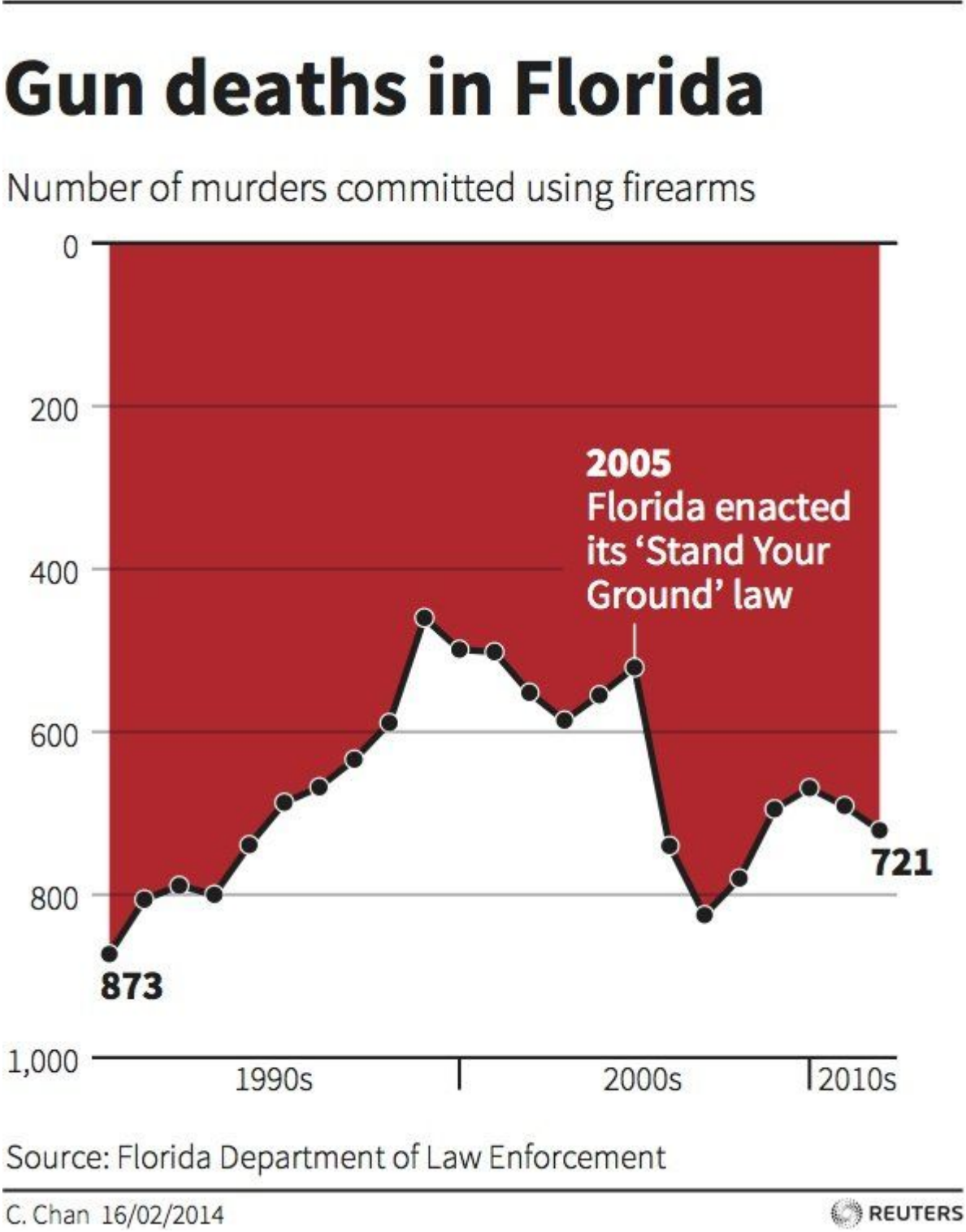
> 📽️ Click the image above for a conference talk about deceptive charts

This chart reverses the X axis to show the opposite of the truth, based on date:

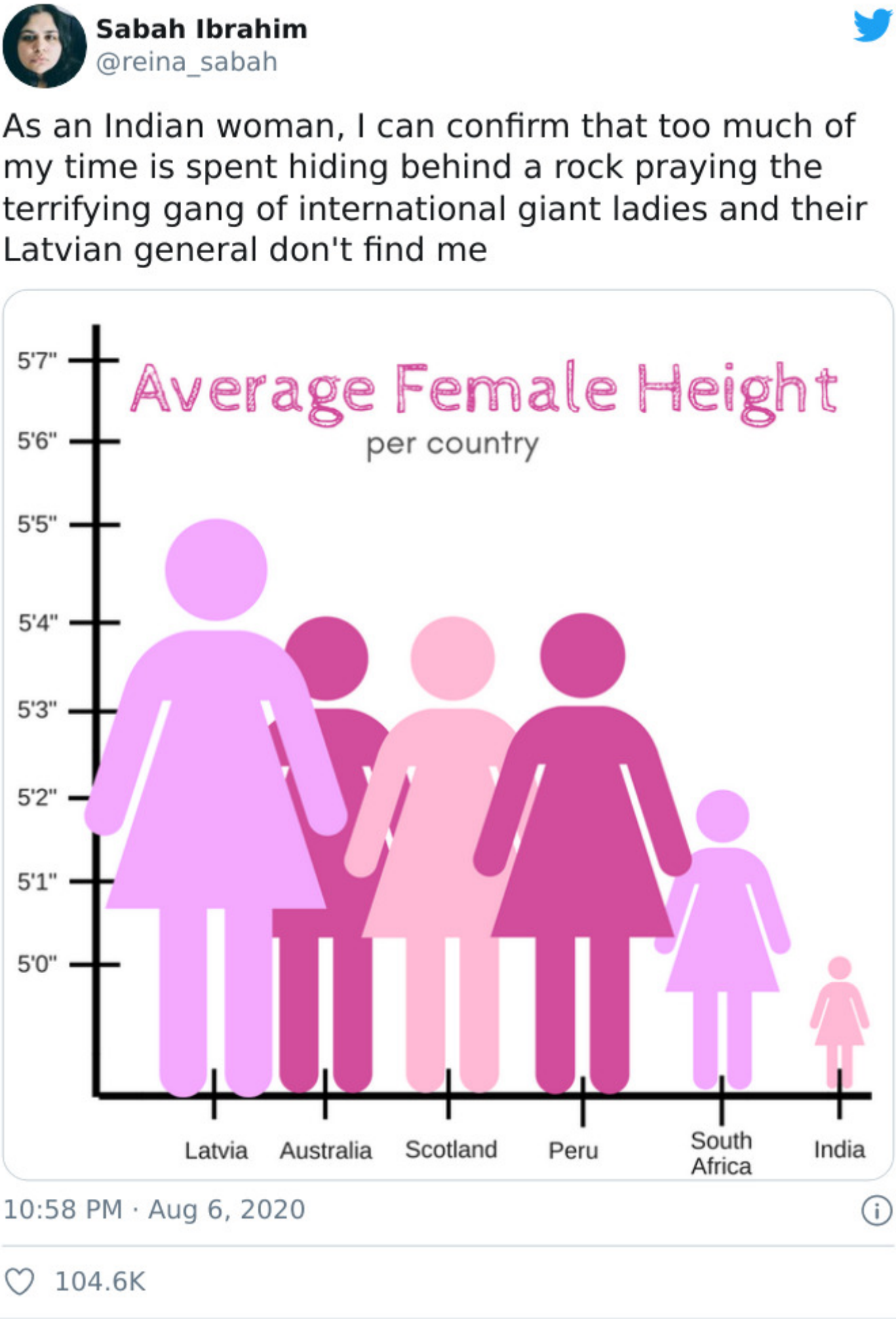
This chart is even more deceptive, as the eye is drawn to the right to conclude that, over time, COVID cases have declined in the various counties. In fact, if you look closely at the dates, you find that they have been rearranged to give that deceptive downward trend.



This notorious example uses color AND a flipped Y axis to deceive: instead of concluding that gun deaths spiked after the passage of gun-friendly legislation, in fact the eye is fooled to think that the opposite is true:



This strange chart shows how proportion can be manipulated, to hilarious effect:



Comparing the incomparable is yet another shady trick. There is a wonderful web site all about 'spurious correlations' displaying 'facts' correlating things like the divorce rate in Maine and the consumption of margarine. A Reddit group also collects the ugly uses of data.

It's important to understand how easily the eye can be fooled by deceptive charts. Even if the data scientist's intention is good, the choice of a bad type of chart, such as a pie chart showing too many categories, can be deceptive.

Color

You saw in the 'Florida gun violence' chart above how color can provide an additional layer of meaning to charts, especially ones not designed using libraries such as Matplotlib and Seaborn which come with various vetted color libraries and palettes. If you are making a chart by hand, do a little study of color theory

> ✅ Be aware, when designing charts, that accessibility is an important aspect of visualization. Some of your users might be color blind - does your chart display well for users with visual impairments?

Be careful when choosing colors for your chart, as color can convey meaning you might not intend. The 'pink ladies' in the 'height' chart above convey a distinctly 'feminine' ascribed meaning that adds to the bizarreness of the chart itself.

While color meaning might be different in different parts of the world, and tend to change in meaning according to their shade. Generally speaking, color meanings include:

Color	Meaning
red	power
blue	trust, loyalty
yellow	happiness, caution
green	ecology, luck, envy
purple	happiness
orange	vibrance

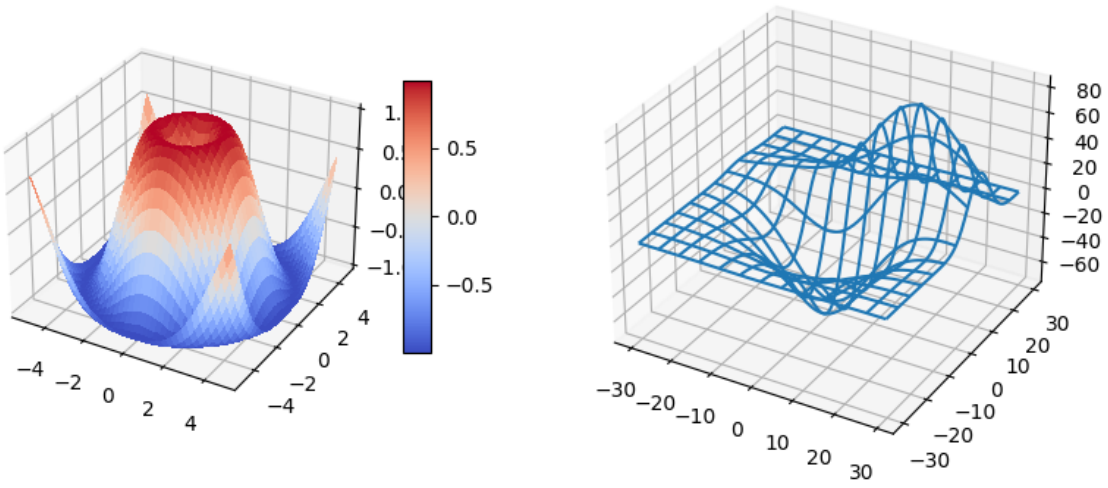
If you are tasked with building a chart with custom colors, ensure that your charts are both accessible and the color you choose coincides with the meaning you are trying to convey.

Styling your charts for readability

Charts are not meaningful if they are not readable! Take a moment to consider styling the width and height of your chart to scale well with your data. If one variable (such as all 50 states) need to be displayed, show them vertically on the Y axis if possible so as to avoid a horizontally-scrolling chart.

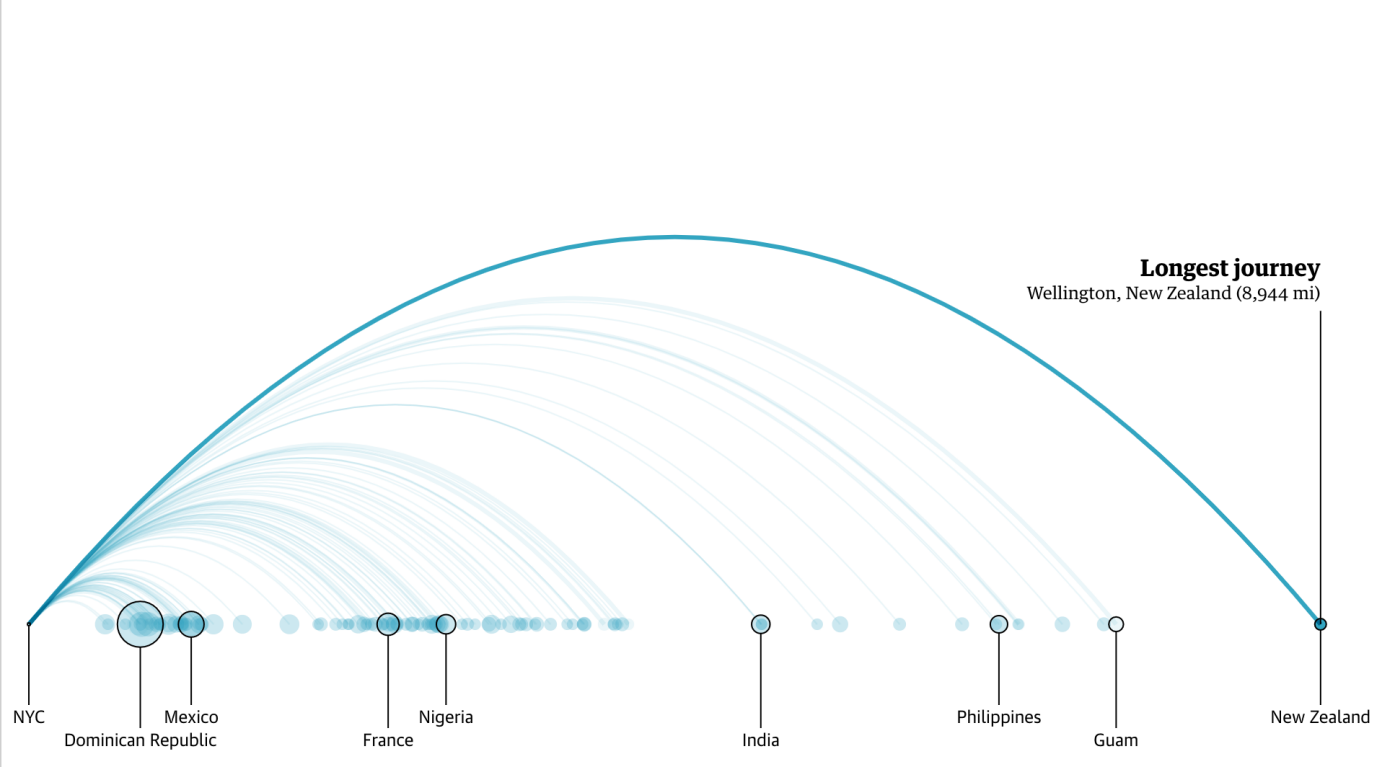
Label your axes, provide a legend if necessary, and offer tooltips for better comprehension of data.

If your data is textual and verbose on the X axis, you can angle the text for better readability. Matplotlib offers 3d plotting, if you data supports it. Sophisticated data visualizations can be produced using `mpl_toolkits.mplot3d` .



Animation and 3D chart display

Some of the best data visualizations today are animated. Shirley Wu has amazing ones done with D3, such as 'film flowers', where each flower is a visualization of a movie. Another example for the Guardian is 'bussed out', an interactive experience combining visualizations with Greensock and D3 plus a scrollytelling article format to show how NYC handles its homeless problem by bussing people out of the city.



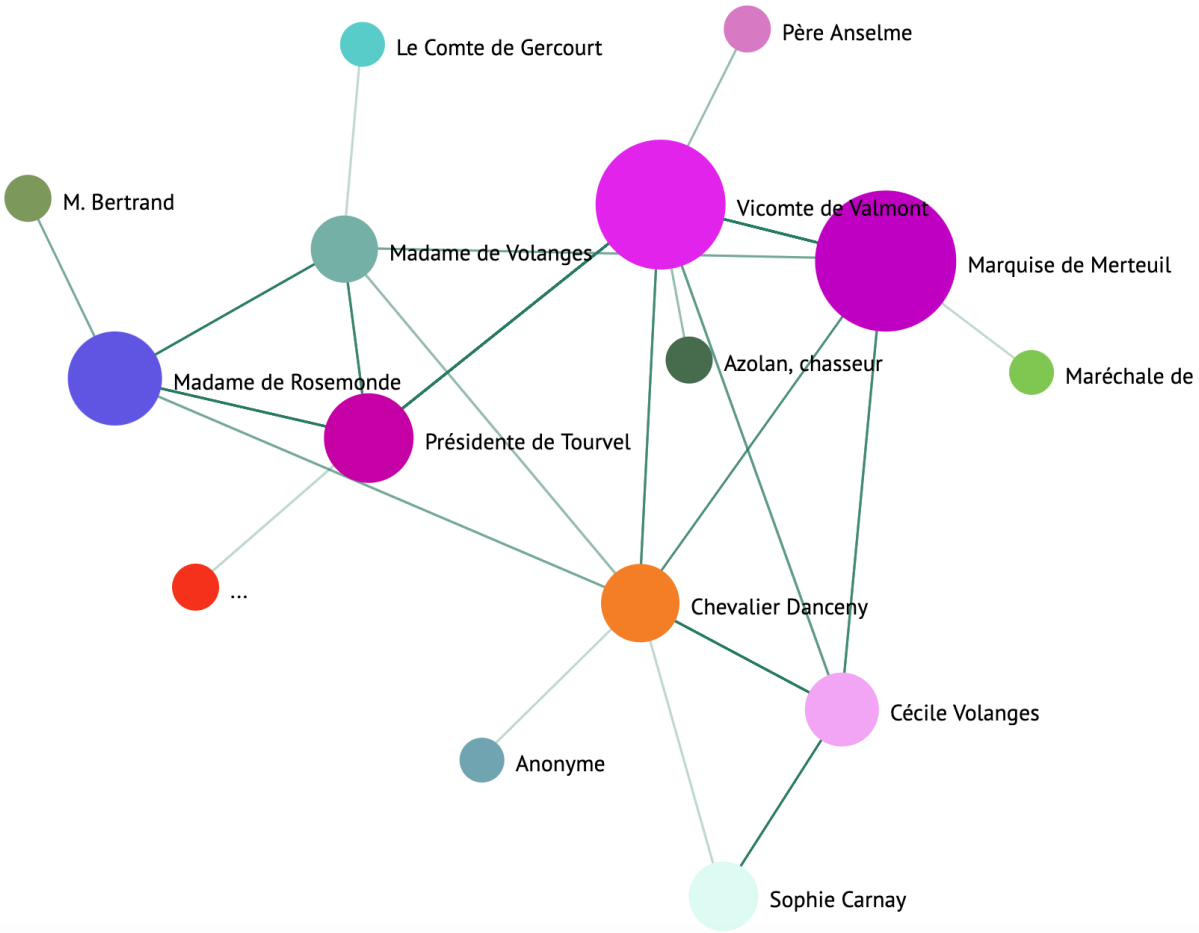
> "Bussed Out: How America Moves its Homeless" from the Guardian. Visualizations by Nadieh Bremer & Shirley Wu

While this lesson is insufficient to go into depth to teach these powerful visualization libraries, try your hand at D3 in a Vue.js app using a library to display a visualization of the book "Dangerous Liaisons" as an animated social network.

> "Les Liaisons Dangereuses" is an epistolary novel, or a novel presented as a series of letters. Written in 1782 by Choderlos de Laclos, it tells the story of the vicious, morally-bankrupt social maneuvers of two dueling protagonists of the French aristocracy in the late 18th century, the Vicomte de Valmont and the Marquise de Merteuil. Both meet their demise in the end but not without inflicting a great deal of social damage. The novel unfolds as a series of letters written to various people in their circles, plotting for revenge or simply to make trouble. Create a visualization of these letters to discover the major kingpins of the narrative, visually.

You will complete a web app that will display an animated view of this social network. It uses a library that was built to create a visual of a network using Vue.js and D3. When the app is running, you can

pull the nodes around on the screen to shuffle the data around.



Project: Build a chart to show a network using D3.js

> This lesson folder includes a `solution` folder where you can find the completed project, for your reference.

1. Follow the instructions in the README.md file in the starter folder's root. Make sure you have NPM and Node.js running on your machine before installing your project's dependencies.
2. Open the `starter/src` folder. You'll discover an `assets` folder where you can find a .json file with all the letters from the novel, numbered, with a 'to' and 'from' annotation.
3. Complete the code in `components/Nodes.vue` to enable the visualization. Look for the method called `createLinks()` and add the following nested loop.

Loop through the .json object to capture the 'to' and 'from' data for the letters and build up the `links` object so that the visualization library can consume it:

```javascript
  //loop through letters
        let f = 0;
        let t = 0;
        for (var i = 0; i < letters.length; i++) {
            for (var j = 0; j < characters.length; j++) {

              if (characters[j] == letters[i].from) {
                f = j;
              }
              if (characters[j] == letters[i].to) {
                t = j;
              }
          }
          this.links.push({ sid: f, tid: t });
        }
```

Run your app from the terminal (npm run serve) and enjoy the visualization!

🚀 Challenge

Take a tour of the internet to discover deceptive visualizations. How does the author fool the user, and is it intentional? Try correcting the visualizations to show how they should look.

Post-lecture quiz

Review & Self Study

Here are some articles to read about deceptive data visualization:

https://gizmodo.com/how-to-lie-with-data-visualization-1563576606

http://ixd.prattsi.org/2017/12/visual-lies-usability-in-deceptive-data-visualizations/

Take a look at these interest visualizations for historical assets and artifacts:

https://handbook.pubpub.org/

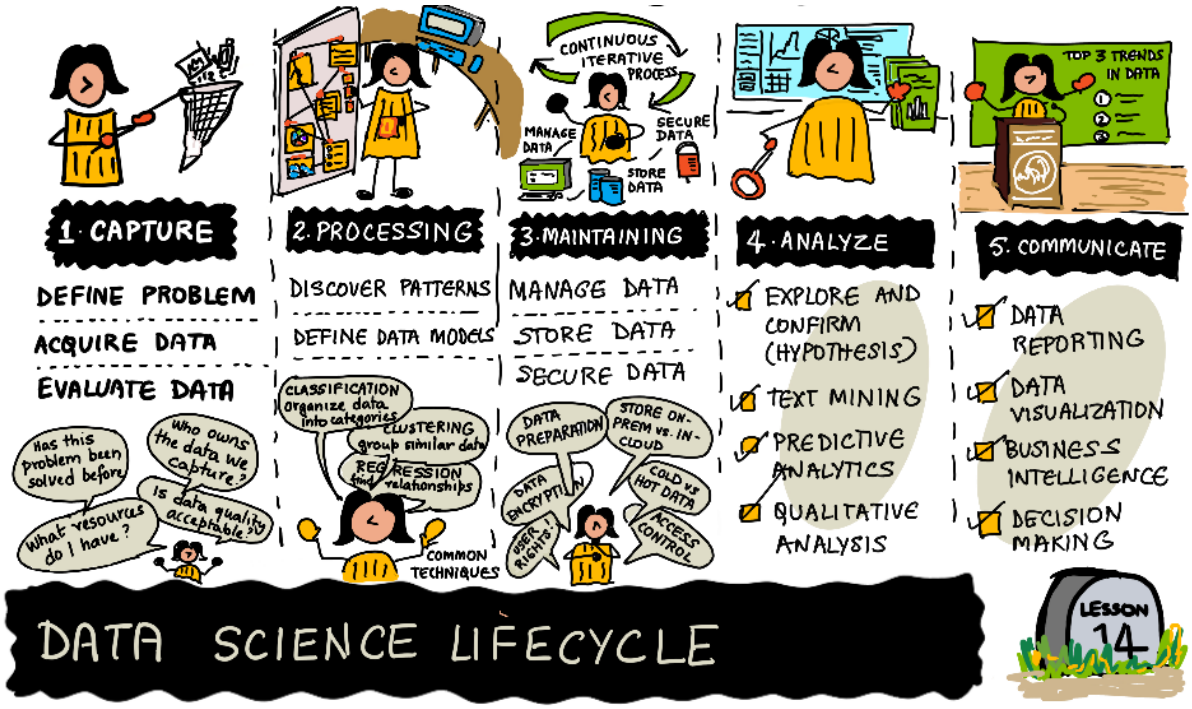
Look through this article on how animation can enhance your visualizations:

https://medium.com/@EvanSinar/use-animation-to-supercharge-data-visualization-cd905a882ad4

Assignment

Build your own custom visualization

Introduction to the Data Science Lifecycle



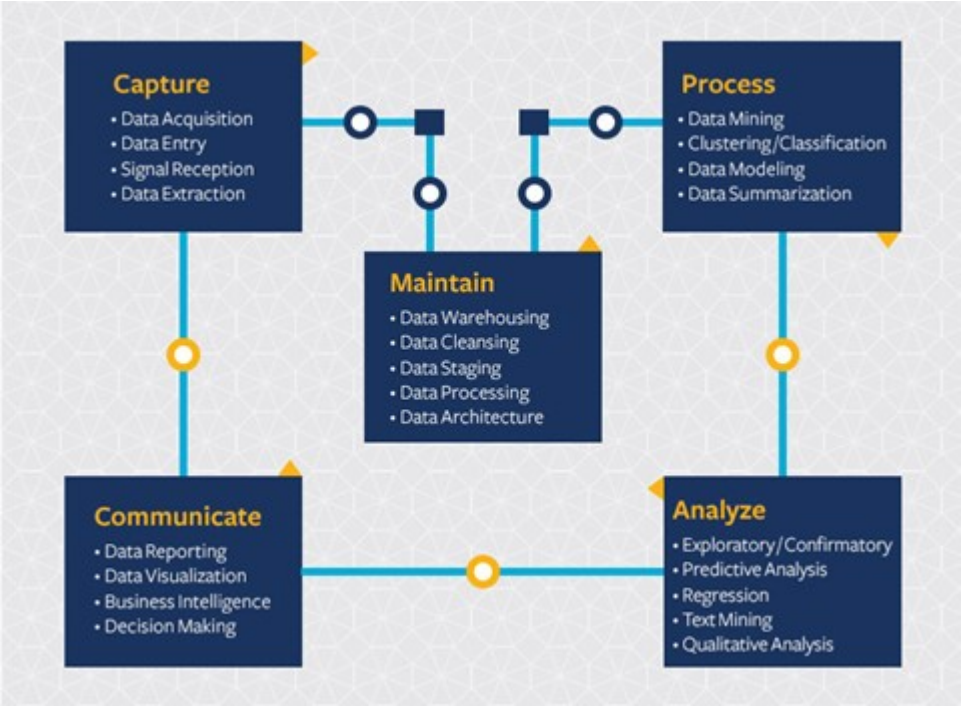
Introduction to the Data Science Lifecycle - *Sketchnote by @nitya*

Pre-Lecture Quiz

At this point you've probably come to the realization that data science is a process. This process can be broken down into 5 stages:

- Capturing
- Processing
- Analysis
- Communication
- Maintenance

This lesson focuses on 3 parts of the life cycle: capturing, processing and maintenance.



> Photo by Berkley School of Information

Capturing

The first stage of the lifecycle is very important as the next stages are dependent on it. It's practically two stages combined into one: acquiring the data and defining the purpose and problems that need to be addressed. Defining the goals of the project will require deeper context into the problem or question. First, we need to identify and acquire those who need their problem solved. These may be stakeholders in a business or sponsors of the project, who can help identify who or what will benefit from this project as well as what, and why they need it. A well-defined goal should be measurable and quantifiable to define an acceptable result.

Questions a data scientist may ask:

- Has this problem been approached before? What was discovered?
- Is the purpose and goal understood by all involved?
- Where is there ambiguity and how to reduce it?
- What are the constraints?
- What will the end result potentially look like?
- How much resources (time, people, computational) are available?

Next is identifying, collecting, then finally exploring the data needed to achieve these defined goals. At this step of acquisition, data scientists must also evaluate the quantity and quality of the data. This

requires some data exploration to confirm what has been acquired will support reaching the desired result.

Questions a data scientist may ask about the data:

- What data is already available to me?
- Who owns this data?
- What are the privacy concerns?
- Do I have enough to solve this problem?
- Is the data of acceptable quality for this problem?
- If I discover additional information through this data, should we consider changing or redefining the goals?

Processing

The processing stage of the lifecycle focuses on discovering patterns in the data as well as modeling. Some techniques used in the processing stage require statistical methods to uncover the patterns. Typically, this would be a tedious task for a human to do with a large data set and will rely on computers to do the heavy lifting to speed up the process. This stage is also where data science and machine learning will intersect. As you learned in the first lesson, machine learning is the process of building models to understand the data. Models are a representation of the relationship between variables in the data that help predict outcomes.

Common techniques used in this stage are covered in the ML for Beginners curriculum. Follow the links to learn more about them:

- Classification: Organizing data into categories for more efficient use.
- Clustering: Grouping data into similar groups.
- Regression: Determine the relationships between variables to predict or forecast values.

Maintaining

In the diagram of lifecycle, you may have noticed that maintenance sits between capturing and processing. Maintenance is an ongoing process of managing, storing and securing the data throughout the process of a project and should be taken into consideration throughout the entirety of the project.

Storing Data

Considerations of how and where the data is stored can influence the cost of its storage as well as performance of how fast the data can be accessed. Decisions like these are not likely to made by a data scientist alone but they may find themselves making choices on how to work with the data based on how it's stored.

Here's some aspects of modern data storage systems that can affect these choices:

On premise vs off premise vs public or private cloud On premise refers to hosting managing the data on your own equipment, like owning a server with hard drives that store the data, while off premise relies on equipment that you don't own, such as a data center. The public cloud is a popular choice for storing data that requires no knowledge of how or where exactly the data is stored, where public refers to a unified underlying infrastructure that is shared by all who use the cloud. Some organizations have strict security policies that require that they have complete access to the equipment where the data is hosted and will rely on a private cloud that provides its own cloud services. You'll learn more about data in the cloud in later lessons.

Cold vs hot data When training your models, you may require more training data. If you're content with your model, more data will arrive for a model to serve its purpose. In any case the cost of storing and accessing data will increase as you accumulate more of it. Separating rarely used data, known as cold data from frequently accessed hot data can be a cheaper data storage option through hardware or software services. If cold data needs to be accessed, it may take a little longer to retrieve in comparison to hot data.

Managing Data

As you work with data you may discover that some of the data needs to be cleaned using some of the techniques covered in the lesson focused on data preparation to build accurate models. When new data arrives, it will need some of the same applications to maintain consistency in quality. Some projects will involve use of an automated tool for cleansing, aggregation, and compression before the data is moved to its final location. Azure Data Factory is an example of one of these tools.

Securing the Data

One of the main goals of securing data is ensuring that those working it are in control of what is collected and in what context it is being used. Keeping data secure involves limiting access to only those who need it, adhering to local laws and regulations, as well as maintaining ethical standards, as covered in the ethics lesson.

Here's some things that a team may do with security in mind:

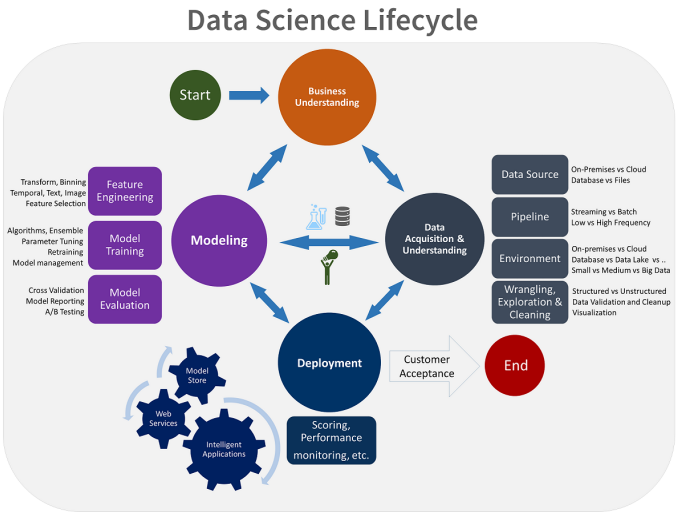
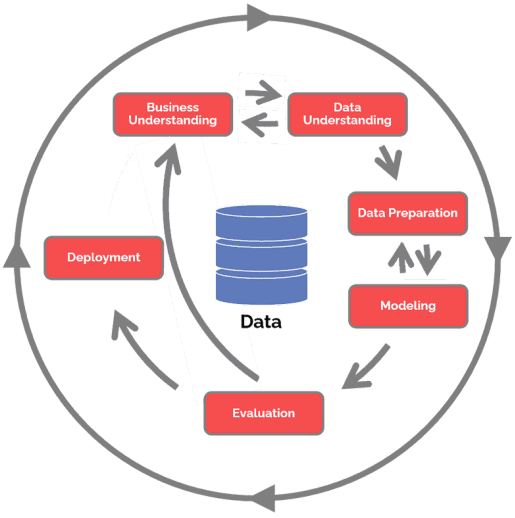
- Confirm that all data is encrypted
- Provide customers information on how their data is used

- Remove data access from those who have left the project
- Let only certain project members alter the data

🚀 Challenge

There are many versions of the Data Science Lifecycle, where each step may have different names and number of stages but will contain the same processes mentioned within this lesson.

Explore the Team Data Science Process lifecycle and the Cross-industry standard process for data mining. Name 3 similarities and differences between the two.

Team Data Science Process (TDSP)	Cross-industry standard process for data mining (CRISP-DM)
	
> Photo by Microsoft	> Photo by Data Science Process Alliance

Post-Lecture Quiz

Review & Self Study

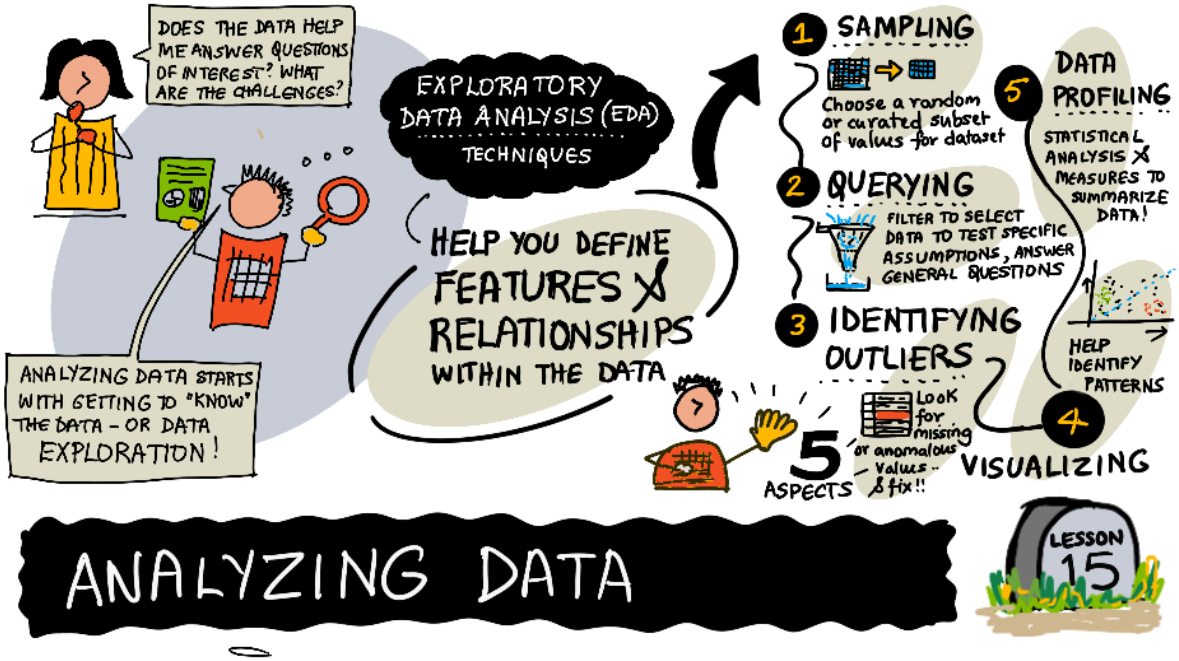
Applying the Data Science Lifecycle involves multiple roles and tasks, where some may focus on particular parts of each stage. The Team Data Science Process provides a few resources that explain the types of roles and tasks that someone may have in a project.

- Team Data Science Process roles and tasks
- Execute data science tasks: exploration, modeling, and deployment

Assignment

Assessing a Dataset

The Data Science Lifecycle: Analyzing



Data Science Lifecycle: Analyzing - *Sketchnote by @nitya*

Pre-Lecture Quiz

Pre-Lecture Quiz

Analyzing in the data lifecycle confirms that the data can answer the questions that are proposed or solving a particular problem. This step can also focus on confirming a model is correctly addressing these questions and problems. This lesson is focused on Exploratory Data Analysis or EDA, which are techniques for defining features and relationships within the data and can be used to prepare the data for modeling.

We'll be using an example dataset from Kaggle to show how this can be applied with Python and the Pandas library. This dataset contains a count of some common words found in emails, the sources of

these emails are anonymous. Use the notebook in this directory to follow along.

Exploratory Data Analysis

The capture phase of the lifecycle is where the data is acquired as well as the problems and questions at hand, but how do we know the data can help support the end result? Recall that a data scientist may ask the following questions when they acquire the data:

- Do I have enough data to solve this problem?
- Is the data of acceptable quality for this problem?
- If I discover additional information through this data, should we consider changing or redefining the goals? Exploratory Data Analysis is the process of getting to know that data and can be used to answer these questions, as well identify the challenges of working with the dataset. Let's focus on some of the techniques used to achieve this.

Data Profiling, Descriptive Statistics, and Pandas

How do we evaluate if we have enough data to solve this problem? Data profiling can summarize and gather some general overall information about our dataset through techniques of descriptive statistics. Data profiling helps us understand what is available to us, and descriptive statistics helps us understand how many things are available to us.

In a few of the previous lessons, we have used Pandas to provide some descriptive statistics with the `describe()` function. It provides the count, max and min values, mean, standard deviation and quantiles on the numerical data. Using descriptive statistics like the `describe()` function can help you assess how much you have and if you need more.

Sampling and Querying

Exploring everything in a large dataset can be very time consuming and a task that's usually left up to a computer to do. However, sampling is a helpful tool in understanding of the data and allows us to have a better understanding of what's in the dataset and what it represents. With a sample, you can apply probability and statistics to come to some general conclusions about your data. While there's no defined rule on how much data you should sample it's important to note that the more data you sample, the more precise of a generalization you can make of about data. Pandas has the `sample()` function in its library where you can pass an argument of how many random samples you'd like to receive and use.

General querying of the data can help you answer some general questions and theories you may have. In contrast to sampling, queries allow you to have control and focus on specific parts of the data you have questions about. The `query()` function in the Pandas library allows you to select columns and receive simple answers about the data through the rows retrieved.

Exploring with Visualizations

You don't have to wait until the data is thoroughly cleaned and analyzed to start creating visualizations. In fact, having a visual representation while exploring can help identify patterns, relationships, and problems in the data. Furthermore, visualizations provide a means of communication with those who are not involved with managing the data and can be an opportunity to share and clarify additional questions that were not addressed in the capture stage. Refer to the section on Visualizations to learn more about some popular ways to explore visually.

Exploring to identify inconsistencies

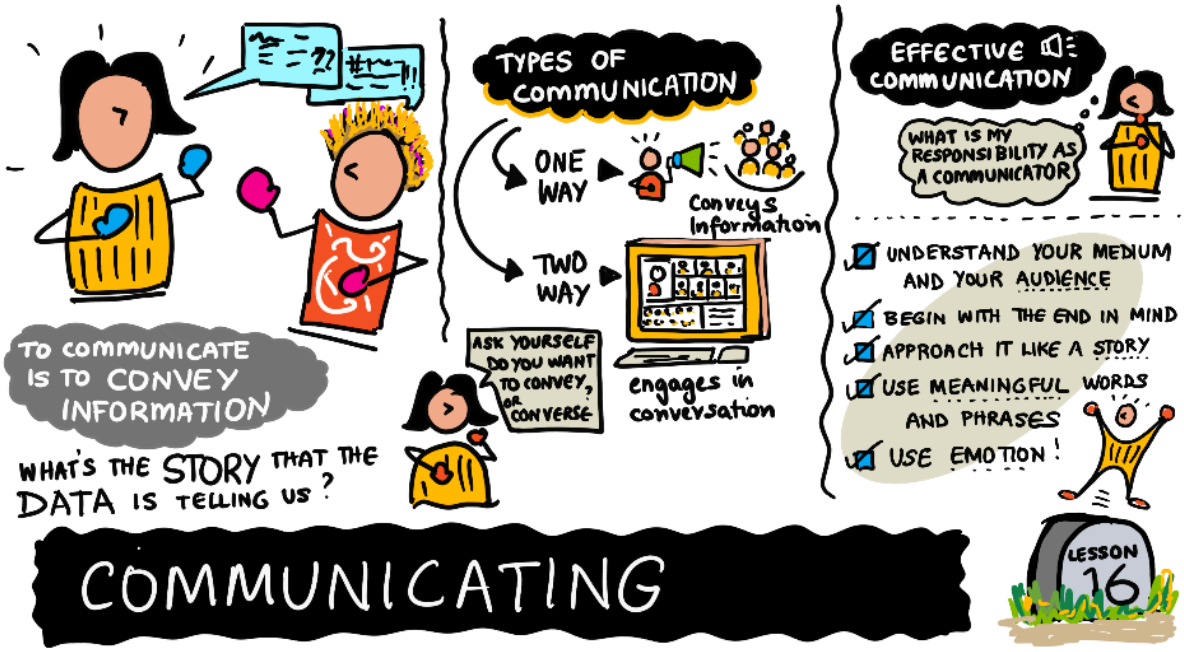
All the topics in this lesson can help identify missing or inconsistent values, but Pandas provides functions to check for some of these. isna() or isnull() can check for missing values. One important piece of exploring for these values within your data is to explore why they ended up that way in the first place. This can help you decide on what actions to take to resolve them.

Pre-Lecture Quiz

Assignment

Exploring for answers

The Data Science Lifecycle: Communication



Data Science Lifecycle: Communication - *Sketchnote by @nitya*

Pre-Lecture Quiz

Test your knowledge of what's to come with the Pre-Lecture Quiz above!

Introduction

What is Communication?

Let's start this lesson by defining what is means to communicate. **To communicate is to convey or exchange information.** Information can be ideas, thoughts, feelings, messages, covert signals, data – anything that a ***sender*** (someone sending information) wants a ***receiver*** (someone receiving information) to understand. In this lesson, we will refer to senders as communicators, and receivers as the audience.

Data Communication & Storytelling

We understand that when communicating, the aim is to convey or exchange information. But when communicating data, your aim shouldn't be to simply pass along numbers to your audience. Your aim should be to communicate a story that is informed by your data - effective data communication and storytelling go hand-in-hand. Your audience is more likely to remember a story you tell, than a

number you give. Later in this lesson, we will go over a few ways that you can use storytelling to communicate your data more effectively.

Types of Communication

Throughout this lesson two different types of communication will be discussed, One-Way Communication and Two-Way Communication.

One way communication happens when a sender sends information to a receiver, without any feedback or response. We see examples of one-way communication every day – in bulk/mass emails, when the news delivers the most recent stories, or even when a television commercial comes on and informs you about why their product is great. In each of these instances, the sender is not seeking an exchange of information. They are only seeking to convey or deliver information.

Two-way communication happens when all involved parties act as both senders and receivers. A sender will begin by communicating to a receiver, and the receiver will provide feedback or a response. Two-way communication is what we traditionally think of when we talk about communication. We usually think of people engaged in a conversation - either in person, or over a phone call, social media, or text message.

When communicating data, there will be cases where you will be using one-way communication (think about presenting at a conference, or to a large group where questions won't be asked directly after) and there will be cases where you will use two-way communication (think about using data to persuade a few stakeholders for buy-in, or to convince a teammate that time and effort should be spent building something new).

Effective Communication

Your Responsibilities as a communicator

When communicating, it is your job to make sure that your receiver(s) are taking away the information that you want them to take away. When you're communicating data, you don't just want your receivers to takeaway numbers, you want your receivers to takeaway a story that's informed by your data. A good data communicator is a good storyteller.

How do you tell a story with data? There are infinite ways – but below are 6 that we will talk about in this lesson.

1. Understand Your Audience, Your Medium, & Your Communication Method
2. Begin with the End in Mind
3. Approach it Like an Actual Story

4. Use Meaningful Words & Phrases
5. Use Emotion

Each of these strategies is explained in greater detail below.

1. Understand Your Audience, Your Channel & Your Communication Method

The way you communicate with family members is likely different than the way you communicate with your friends. You probably use different words and phrases that the people you're speaking to are more likely to understand. You should take the same approach when communicating data. Think about who you're communicating to. Think about their goals and the context that they have around the situation that you're explaining to them.

You can likely group the majority of your audience them within a category. In a *Harvard Business Review* article, "How to Tell a Story with Data," Dell Executive Strategist Jim Stikeleather identifies five categories of audiences.

- **Novice**: first exposure to the subject, but doesn't want oversimplification
- **Generalist**: aware of the topic, but looking for an overview understanding and major themes
- **Managerial**: in-depth, actionable understanding of intricacies and interrelationships with access to detail
- **Expert**: more exploration and discovery and less storytelling with great detail
- **Executive**: only has time to glean the significance and conclusions of weighted probabilities

These categories can inform the way you present data to your audience.

In addition to thinking about your audience's category, you should also consider the channel you're using to communicate with your audience. Your approach should be slightly different if you're writing a memo or email vs having a meeting or presenting at a conference.

On top of understanding your audience, knowing how you will be communicating with them (using one-way communication or two-way) is also critical.

If you are communicating with a majority Novice audience and you're using one-way communication, you must first educate the audience and give them proper context. Then you must present your data to them and tell them what your data means and why your data matters. In this instance, you may want to be laser focused on driving clarity, because your audience will not be able to ask you any direct questions.

If you are communicating with a majority Managerial audience and you're using two-way communication, you likely won't need to educate your audience or provide them with much context. You may be able to jump straight into discussing the data that you've collected and why it matters. In this scenario though, you should be focused on timing and controlling your presentation. When using

two-way communication (especially with a Managerial audience who is seeking an “actionable understanding of intricacies and interrelationships with access to detail”) questions may pop up during your interaction that may take the discussion in a direction that doesn’t relate to the story that you’re trying to tell. When this happens, you can take action and move the discussion back on track with your story.

2. Begin With The End In Mind

Beginning with the end in mind means understanding your intended takeaways for your audience before you start communicating with them. Being thoughtful about what you want your audience to takeaway ahead of time can help you craft a story that your audience can follow. Beginning with the end in mind is appropriate for both one-way communication and two-way communication.

How do you begin with the end in mind? Before communicating your data, write down your key takeaways. Then, every step of the way as you're preparing the story that you want to tell with your data, ask yourself, "How does this integrate into the story I'm telling?"

Be Aware – While starting with the end in mind is ideal, you don’t want to communicate only the data that supports your intended takeaways. Doing this is called Cherry-Picking, which happens when a communicator only communicates data that supports the point they are tying to make and ignores all other data.

If all the data that you collected clearly supports your intended takeaways, great. But if there is data that you collected that doesn’t support your takeaways, or even supports an argument against your key takeaways, you should communicate that data as well. If this happens, be upfront with your audience and let them know why you're choosing to stick with your story even though all the data doesn't necessarily support it.

3. Approach it Like an Actual Story

A traditional story happens in 5 Phases. You may have heard these phases expressed as Exposition, Rising Action, Climax, Falling Action, and Denouncement. Or the easier to remember Context, Conflict, Climax, Closure, Conclusion. When communicating your data and your story, you can take a similar approach.

You can begin with context, set the stage and make sure your audience is all on the same page. Then introduce the conflict. Why did you need to collect this data? What problems were you seeking to solve? After that, the climax. What is the data? What does the data mean? What solutions does the data tell us we need? Then you get to the closure, where you can reiterate the problem, and the proposed solution(s). Lastly, we come to the conclusion, where you can summarize your key takeaways and the next steps you recommend the team takes.

4. Use Meaningful Words & Phrases

If you and I were working together on a product, and I said to you "Our users take a long time to onboard onto our platform," how long would you estimate that "long time" to be? An hour? A week? It's hard to know. What if I said that to an entire audience? Everyone in the audience may end up with a different idea of how long users take to onboard onto our platform.

Instead, what if I said "Out users take, on average, 3 minutes to sign up and onboard onto our platform."

That messaging is more clear. When communicating data, it can be easy to think that everyone in your audience is thinking just like you. But that is not always the case. Driving clarity around your data and what it means is one of your responsibilities as a communicator. If the data or your story is not clear, your audience will have a hard time following, and it is less likely that they will understand your key takeaways.

You can communicate data more clearly when you use meaningful words and phrases, instead of vague ones. Below are a few examples.

- We had an *impressive* year!
  - One person could think a impressive means a 2% - 3% increase in revenue, and one person could think it means a 50% - 60% increase.
- Our users' success rates increased *dramatically*.
  - How large of an increase is a dramatic increase?
- This undertaking will require *significant* effort.
  - How much effort is significant?

Using vague words could be useful as an introduction to more data that's coming, or as a summary of the story that you've just told. But consider ensuring that every part of your presentation is clear for your audience.

5. Use Emotion

Emotion is key in storytelling. It's even more important when you're telling a story with data. When you're communicating data, everything is focused on the takeaways you want your audience to have. When you evoke an emotion for an audience it helps them empathize, and makes them more likely to take action. Emotion also increases the likelihood that an audience will remember your message.

You may have encountered this before with TV commercials. Some commercials are very somber, and use a sad emotion to connect with their audience and make the data that they're presenting really stand out. Or, some commercials are very upbeat and happy may make you associate their data with a happy feeling.

How do you use emotion when communicating data? Below are a couple of ways.

- Use Testimonials and Personal Stories
  - When collecting data, try to collect both quantitative and qualitative data, and integrate both types of data when you're communicating. If your data is primarily quantitative, seek stories from individuals to learn more about their experience with whatever your data is telling you.
- Use Imagery
  - Images help an audience see themselves in a situation. When you use images, you can push an audience toward the emotion that you feel they should have about your data.
- Use Color
  - Different colors evoke different emotions. Popular colors and the emotions they evoke are below. Be aware, that colors could have different meanings in different cultures.
    - Blue usually evokes emotions of peace and trust
    - Green is usually related to the nature and the environment
    - Red is usually passion and excitement
    - Yellow is usually optimism and happiness

Communication Case Study

Emerson is a Product Manager for a mobile app. Emerson has noticed that customers submit 42% more complaints and bug reports on the weekends. Emerson also noticed that customers who submit a complaint that goes unanswered after 48 hours are more 32% more likely to give the app a rating of 1 or 2 in the app store.

After doing research, Emerson has a couple of solutions that will address the issue. Emerson sets up a 30-minute meeting with the 3 company leads to communicate the data and the proposed solutions.

During this meeting, Emerson's goal is to have the company leads understand that the 2 solutions below can improve the app's rating, which will likely translate into higher revenue.

Solution 1. Hire customer service reps to work on weekends

Solution 2. Purchase a new customer service ticketing system where customer service reps can easily identify which complaints have been in the queue the longest – so they can tell which to address most immediately.

In the meeting, Emerson spends 5 minutes explaining why having a low rating on the app store is bad, 10 minutes explaining the research process and how the trends were identified, 10 minutes going through some of the recent customer complaints, and the last 5 minutes glossing over the 2 potential solutions.

Was this an effective way for Emerson to communicate during this meeting?

During the meeting, one company lead fixated on the 10 minutes of customer complaints that Emerson went through. After the meeting, these complaints were the only thing that this team lead remembered. Another company lead primarily focused on Emerson describing the research process. The third company lead did remember the solutions proposed by Emerson but wasn't sure how those solutions could be implemented.

In the situation above, you can see that there was a significant gap between what Emerson wanted the team leads to take away, and what they ended up taking away from the meeting. Below is another approach that Emerson could consider.

How could Emerson improve this approach? Context, Conflict, Climax, Closure, Conclusion **Context** - Emerson could spend the first 5 minutes introducing the entire situation and making sure that the team leads understand how the problems affect metrics that are critical to the company, like revenue.

It could be laid out this way: "Currently, our app's rating in the app store is a 2.5. Ratings in the app store are critical to App Store Optimization, which impacts how many users see our app in search, and how our app is viewed to perspective users. And ofcourse, the number of users we have is tied directly to revenue."

Conflict Emerson could then move to talk for the next 5 minutes or so on the conflict.

It could go like this: "Users submit 42% more complaints and bug reports on the weekends. Customers who submit a complaint that goes unanswered after 48 hours are more 32% less likely to give our app a rating over a 2 in the app store. Improving our app's rating in the app store to a 4 would improve our visibility by 20-30%, which I project would increase revenue by 10%." Of course, Emerson should be prepared to justify these numbers.

Climax After laying the groundwork, Emerson could then move to the Climax for 5 or so minutes.

Emerson could introduce the proposed solutions, lay out how those solutions will address the issues outlined, how those solutions could be implemented into existing workflows, how much the solutions cost, what the ROI of the solutions would be, and maybe even show some screenshots or wireframes of how the solutions would look if implemented. Emerson could also share testimonials from users who took over 48 hours to have their complaint addressed, and even a testimonial from a current customer service representative within the company who has comments on the current ticketing system.

Closure Now Emerson can spend 5 minutes restating the problems faced by the company, revisit the proposed solutions, and review why those solutions are the right ones.

Conclusion Because this is a meeting with a few stakeholders where two-way communication will be used, Emerson could then plan to leave 10 minutes for questions, to make sure that anything that was confusing to the team leads could be clarified before the meeting is over.

If Emerson took approach #2, it is much more likely that the team leads will take away from the meeting exactly what Emerson intended for them to take away – that the way complaints and bugs are handled could be improved, and there are 2 solutions that could be put in place to make that improvement happen. This approach would be a much more effective approach to communicating the data, and the story, that Emerson wants to communicate.

Conclusion

Summary of main points

- To communicate is to convey or exchange information.
- When communicating data, your aim shouldn't be to simply pass along numbers to your audience. Your aim should be to communicate a story that is informed by your data.
- There are 2 types of communication, One-Way Communication (information is communicated with no intention of a response) and Two-Way Communication (information is communicated back and forth.)
- There are many strategies you can use to telling a story with your data, 5 strategies we went over are:
    - Understand Your Audience, Your Medium, & Your Communication Method
    - Begin with the End in Mind
    - Approach it Like an Actual Story
    - Use Meaningful Words & Phrases
    - Use Emotion

Post-Lecture Quiz

Recommended Resources for Self Study

The Five C's of Storytelling - Articulate Persuasion

1.4 Your Responsibilities as a Communicator – Business Communication for Success (umn.edu)

How to Tell a Story with Data (hbr.org)

Two-Way Communication: 4 Tips for a More Engaged Workplace (yourthoughtpartner.com)

6 succinct steps to great data storytelling - BarnRaisers, LLC (barnraisersllc.com)

How to Tell a Story With Data | Lucidchart Blog

6 Cs of Effective Storytelling on Social Media | Cooler Insights

The Importance of Emotions In Presentations | Ethos3 - A Presentation Training and Design Agency

Data storytelling: linking emotions and rational decisions (toucantoco.com)

Emotional Advertising: How Brands Use Feelings to Get People to Buy (hubspot.com)

Choosing Colors for Your Presentation Slides | Think Outside The Slide

How To Present Data [10 Expert Tips] | ObservePoint

Microsoft Word - Persuasive Instructions.doc (tpsnva.org)

The Power of Story for Your Data (thinkhdi.com)

Common Mistakes in Data Presentation (perceptualedge.com)

Infographic: Here are 15 Common Data Fallacies to Avoid (visualcapitalist.com)

Cherry Picking: When People Ignore Evidence that They Dislike – Effectiviology

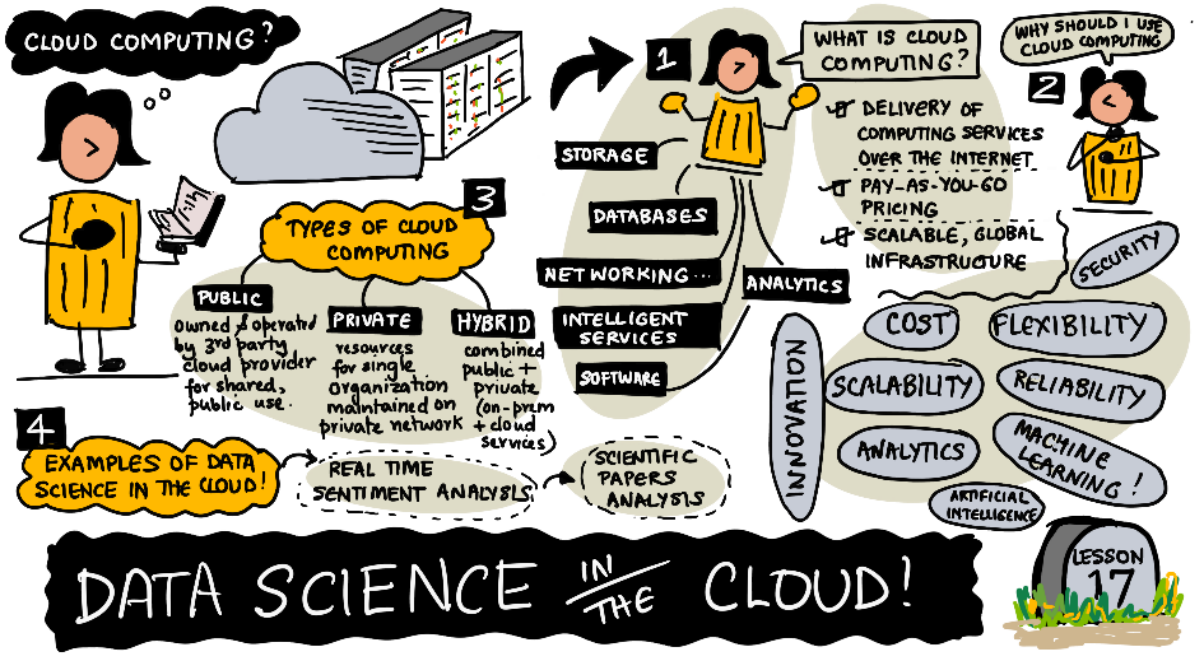
Tell Stories with Data: Communication in Data Science | by Sonali Verghese | Towards Data Science

1. Communicating Data - Communicating Data with Tableau [Book] (oreilly.com)

Assignment

Tell a story

Introduction to Data Science in the Cloud



Data Science In The Cloud: Introduction - *Sketchnote by @nitya*

In this lesson, you will learn the fundamental principles of the Cloud, then you will see why it can be interesting for you to use Cloud services to run your data science projects and we'll look at some examples of data science projects run in the Cloud.

Pre-Lecture Quiz

What is the Cloud?

The Cloud, or Cloud Computing, is the delivery of a wide range of pay-as-you-go computing services hosted on an infrastructure over the internet. Services include solutions such as storage, databases, networking, software, analytics, and intelligent services.

We usually differentiate the Public, Private and Hybrid clouds as follows:

- Public cloud: a public cloud is owned and operated by a third-party cloud service provider which delivers its computing resources over the Internet to the public.
- Private cloud: refers to cloud computing resources used exclusively by a single business or organization, with services and an infrastructure maintained on a private network.
- Hybrid cloud: the hybrid cloud is a system that combines public and private clouds. Users opt for an on-premises datacenter, while allowing data and applications to be run on one or more public clouds.

Most cloud computing services fall into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

- Infrastructure as a Service (IaaS): users rent an IT infrastructure such as servers and virtual machines (VMs), storage, networks, operating systems
- Platform as a Service (PaaS): users rent an environment for developing, testing, delivering, and managing software applications. Users don't need to worry about setting up or managing the underlying infrastructure of servers, storage, network, and databases needed for development.
- Software as a Service (SaaS): users get access to software applications over the Internet, on demand and typically on a subscription basis. Users don't need to worry about hosting and managing the software application, the underlying infrastructure or the maintenance, like software upgrades and security patching.

Some of the largest Cloud providers are Amazon Web Services, Google Cloud Platform and Microsoft Azure.

Why Choose the Cloud for Data Science?

Developers and IT professionals chose to work with the Cloud for many reasons, including the following:

- Innovation: you can power your applications by integrating innovative services created by Cloud providers directly into your apps.
- Flexibility: you only pay for the services that you need and can choose from a wide range of services. You typically pay as you go and adapt your services according to your evolving needs.
- Budget: you don't need to make initial investments to purchase hardware and software, set up and run on-site datacenters and you can just pay for what you use.
- Scalability: your resources can scale according to the needs of your project, which means that your apps can use more or less computing power, storage and bandwidth, by adapting to external factors at any given time.
- Productivity: you can focus on your business rather than spending time on tasks that can be managed by someone else, such as managing datacenters.
- Reliability: Cloud Computing offers several ways to continuously back up your data and you can set up disaster recovery plans to keep your business and services going, even in times of crisis.
- Security: you can benefit from policies, technologies and controls that strengthen the security of your project.

These are some of the most common reasons why people choose to use Cloud services. Now that we have a better understanding of what the Cloud is and what its main benefits are, let's look more specifically into the jobs of Data scientists and developers working with data, and how the Cloud can help them with several challenges they might face:

- Storing large amounts of data: instead of buying, managing and protecting big servers, you can store your data directly in the cloud, with solutions such as Azure Cosmos DB, Azure SQL Database and Azure Data Lake Storage.
- Performing Data Integration: data integration is an essential part of Data Science, that lets you make a transition from data collection to taking actions. With data integration services offered in the cloud, you can collect, transform and integrate data from various sources into a single data warehouse, with Data Factory.
- Processing data: processing vast amounts of data requires a lot of computing power, and not everyone has access to machines powerful enough for that, which is why many people choose to directly harness the cloud's huge computing power to run and deploy their solutions.
- Using data analytics services: cloud services like Azure Synapse Analytics, Azure Stream Analytics and Azure Databricks to help you turn your data into actionable insights.
- Using Machine Learning and data intelligence services: Instead of starting from scratch, you can use machine learning algorithms offered by the cloud provider, with services such as AzureML. You can also use cognitive services such as speech-to-text, text to speech, computer vision and more.

Examples of Data Science in the Cloud

Let's make this more tangible by looking at a couple of scenarios.

Real-time social media sentiment analysis

We'll start with a scenario commonly studied by people who start with machine learning: social media sentiment analysis in real time.

Let's say you run a news media website and you want to leverage live data to understand what content your readers could be interested in. To know more about that, you can build a program that performs real-time sentiment analysis of data from Twitter publications, on topics that are relevant to your readers.

The key indicators you will look at is the volume of tweets on specific topics (hashtags) and sentiment, which is established using analytics tools that perform sentiment analysis around the

specified topics.

The steps necessary to create this project are as follows:

- Create an event hub for streaming input, which will collect data from Twitter
- Configure and start a Twitter client application, which will call the Twitter Streaming APIs
- Create a Stream Analytics job
- Specify the job input and query
- Create an output sink and specify the job output
- Start the job

To view the full process, check out the documentation.

Scientific papers analysis

Let's take another example of a project created by Dmitry Soshnikov, one of the authors of this curriculum.

Dmitry created a tool that analyses COVID papers. By reviewing this project, you will see how you can create a tool that extracts knowledge from scientific papers, gains insights and helps researchers navigate through large collections of papers in an efficient way.

Let's see the different steps used for this:

- Extracting and pre-processing information with Text Analytics for Health
- Using Azure ML to parallelize the processing
- Storing and querying information with Cosmos DB
- Create an interactive dashboard for data exploration and visualization using Power BI

To see the full process, visit Dmitry's blog.

As you can see, we can leverage Cloud services in many ways to perform Data Science.

Footnote

Sources:

- https://azure.microsoft.com/overview/what-is-cloud-computing?ocid=AID3041109
- https://docs.microsoft.com/azure/stream-analytics/stream-analytics-twitter-sentiment-analysis-trends?ocid=AID3041109
- https://soshnikov.com/science/analyzing-medical-papers-with-azure-and-text-analytics-for-health/

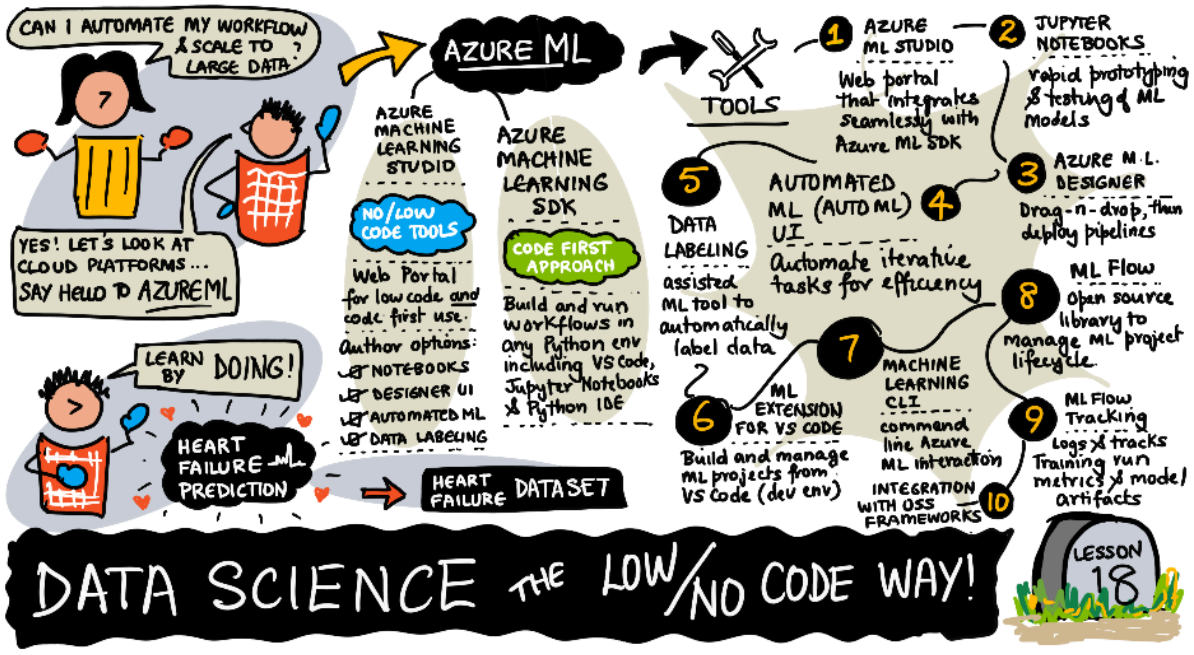
Post-Lecture Quiz

Post-lecture quiz

Assignment

Market Research

Data Science in the Cloud: The "Low code/No code" way



Data Science In The Cloud: Low Code - *Sketchnote by @nitya*

Table of contents:

Pre-Lecture quiz

1. Introduction

1.1 What is Azure Machine Learning?

The Azure cloud platform is more than 200 products and cloud services designed to help you bring new solutions to life. Data scientists expend a lot of effort exploring and pre-processing data, and trying various types of model-training algorithms to produce accurate models. These tasks are time consuming, and often make inefficient use of expensive compute hardware.

Azure ML is a cloud-based platform for building and operating machine learning solutions in Azure. It includes a wide range of features and capabilities that help data scientists prepare data, train models, publish predictive services, and monitor their usage. Most importantly, it helps them to increase their

efficiency by automating many of the time-consuming tasks associated with training models; and it enables them to use cloud-based compute resources that scale effectively, to handle large volumes of data while incurring costs only when actually used.

Azure ML provides all the tools developers and data scientists need for their machine learning workflows. These include:

- **Azure Machine Learning Studio**: it is a web portal in Azure Machine Learning for low-code and no-code options for model training, deployment, automation, tracking and asset management. The studio integrates with the Azure Machine Learning SDK for a seamless experience.
- **Jupyter Notebooks**: quickly prototype and test ML models.
- **Azure Machine Learning Designer**: allows to drag-n-drop modules to build experiments and then deploy pipelines in a low-code environment.
- **Automated machine learning UI (AutoML)** : automates iterative tasks of machine learning model development, allowing to build ML models with high scale, efficiency, and productivity, all while sustaining model quality.
- **Data Labelling**: an assisted ML tool to automatically label data.
- **Machine learning extension for Visual Studio Code**: provides a full-featured development environment for building and managing ML projects.
- **Machine learning CLI**: provides commands for managing Azure ML resources from the command line.
- **Integration with open-source frameworks** such as PyTorch, TensorFlow, scikit-learn and many more for training, deploying, and managing the end-to-end machine learning process.
- **MLflow**: It is an open-source library for managing the life cycle of your machine learning experiments. **MLFlow Tracking** is a component of MLflow that logs and tracks your training run metrics and model artifacts, irrespective of your experiment's environment.

1.2 The Heart Failure Prediction Project:

There is no doubt that making and building projects is the best to put your skills and knowledge to test. In this lesson, we are going to explore two different ways of building a data science project for the prediction of heart failure attacks in Azure ML Studio, through Low code/No code and through the Azure ML SDK as shown in the following schema:


project-schema

Each way has its own pros and cons. The Low code/No code way is easier to start with as it involves interacting with a GUI (Graphical User Interface), with no pior knowledge of code required. This method enables quick testing of the project's viability and to create POC (Proof Of Concept). However, as the project grows and things need to be production ready, it is not feasible to create resources through GUI. We need to programmatically automate everything, from the creation of

resources, to the deployment of a model. This is where knowing how to use the Azure ML SDK becomes crucial.

	Low code/No code	Azure ML SDK
Expertise in code	Not required	Required
Time to develop	Fast and easy	Depends on code expertise
Production ready	No	Yes

1.3 The Heart Failure Dataset:

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, accounting for 31% of all deaths worldwide. Environmental and behavioural risk factors such as use of tobacco, unhealthy diet and obesity, physical inactivity and harmful use of alcohol could be used as features for estimation models. Being able to estimate the probability of the development of a CVD could be of great use to prevent attacks in high risk people.

Kaggle has made a Heart Failure dataset publically available, that we are going to use for this project. You can download the dataset now. This is a tabular dataset with 13 columns (12 features and 1 target variable) and 299 rows.

	Variable name	Type	Description	Example
1	age	numerical	age of the patient	25
2	anaemia	boolean	Decrease of red blood cells or haemoglobin	0 or 1
3	creatinine_phosphokinase	numerical	Level of CPK enzyme in the blood	542
4	diabetes	boolean	If the patient has diabetes	university.degree
5	ejection_fraction	numerical	Percentage of blood leaving the heart on each contraction	45
6	high_blood_pressure	boolean	If the patient has hypertension	0 or 1
7	platelets	numerical	Platelets in the blood	149000

	Variable name	Type	Description	Example
8	serum_creatinine	numerical	Level of serum creatinine in the blood	0.5
9	serum_sodium	numerical	Level of serum sodium in the blood	jun
10	sex	boolean	woman or man	0 or 1
11	smoking	boolean	If the patient smokes	285
12	time	numerical	follow-up period (days)	4
--	----------	----------	----------	----------
21	DEATH_EVENT [Target]	boolean	if the patient dies during the follow-up period	0 or 1

Once you have the dataset, we can start the project in Azure.

2. Low code/No code training of a model in Azure ML Studio

2.1 Create an Azure ML workspace

To train a model in Azure ML you first need to create an Azure ML workspace. The workspace is the top-level resource for Azure Machine Learning, providing a centralized place to work with all the artifacts you create when you use Azure Machine Learning. The workspace keeps a history of all training runs, including logs, metrics, output, and a snapshot of your scripts. You use this information to determine which training run produces the best model. Learn more

It is recommended to use the most up-to-date browser that's compatible with your operating system. The following browsers are supported:

- Microsoft Edge (The new Microsoft Edge, latest version. Not Microsoft Edge legacy)
- Safari (latest version, Mac only)
- Chrome (latest version)

- Firefox (latest version)

To use Azure Machine Learning, create a workspace in your Azure subscription. You can then use this workspace to manage data, compute resources, code, models, and other artifacts related to your machine learning workloads.

> NOTE: Your Azure subscription will be charged a small amount for data storage as long as the Azure Machine Learning workspace exists in your subscription, so we recommend you to delete the Azure Machine Learning workspace when you are no longer using it.

1. Sign into the Azure portal using the Microsoft credentials associated with your Azure subscription.
2. Select **+ Create a resource**

   Search for Machine Learning and select the Machine Learning tile

   Click the create button

   Fill in the settings as follows:

   - Subscription: Your Azure subscription
   - Resource group: Create or select a resource group
   - Workspace name: Enter a unique name for your workspace
   - Region: Select the geographical region closest to you
   - Storage account: Note the default new storage account that will be created for your workspace
   - Key vault: Note the default new key vault that will be created for your workspace
   - Application insights: Note the default new application insights resource that will be created for your workspace
   - Container registry: None (one will be created automatically the first time you deploy a model to a container)

- Click the create + review and then on the create button

3. Wait for your workspace to be created (this can take a few minutes). Then go to it in the portal. You can find it through the Machine Learning Azure service.

4. On the Overview page for your workspace, launch Azure Machine Learning studio (or open a new browser tab and navigate to https://ml.azure.com), and sign into Azure Machine Learning studio using your Microsoft account. If prompted, select your Azure directory and subscription, and your Azure Machine Learning workspace.

5. In Azure Machine Learning studio, toggle the ☰ icon at the top left to view the various pages in the interface. You can use these pages to manage the resources in your workspace.

You can manage your workspace using the Azure portal, but for data scientists and Machine Learning operations engineers, Azure Machine Learning Studio provides a more focused user interface for managing workspace resources.

2.2 Compute Resources

Compute Resources are cloud-based resources on which you can run model training and data exploration processes. There are four kinds of compute resource you can create:

- **Compute Instances**: Development workstations that data scientists can use to work with data and models. This involves the creation of a Virtual Machine (VM) and launch a notebook instance. You can then train a model by calling a computer cluster from the notebook.
- **Compute Clusters**: Scalable clusters of VMs for on-demand processing of experiment code. You will need it when training a model. Compute clusters can also employ specialized GPU or CPU resources.
- **Inference Clusters**: Deployment targets for predictive services that use your trained models.
- **Attached Compute**: Links to existing Azure compute resources, such as Virtual Machines or Azure Databricks clusters.

2.2.1 Choosing the right options for your compute resources

Some key factors are to consider when creating a compute resource and those choices can be critical decisions to make.

Do you need CPU or GPU ?

A CPU (Central Processing Unit) is the electronic circuitry that executes instructions comprising a computer program. A GPU (Graphics Processing Unit) is a specialized electronic circuit that can execute graphics-related code at a very high rate.

The main difference between CPU and GPU architecture is that a CPU is designed to handle a wide-range of tasks quickly (as measured by CPU clock speed), but are limited in the concurrency of tasks that can be running. GPUs are designed for parallel computing and therfore are much better at deep learning tasks.

CPU	GPU
Less expensive	More expensive
Lower level of concurrency	Higher level of concurrency
Slower in training deep learning models	Optimal for deep learning

Cluster Size

Larger clusters are more expensive but will result in better responsiveness. Therefore, if you have time but not enough money, you should start with a small cluster. Conversely, if you have money but not much time, you should start with a larger cluster.

VM Size

Depending on your time and budgetary constraints, you can vary the size of your RAM, disk, number of cores and clock speed. Increasing all those parameters will be costlier, but will result in better performance.

Dedicated or Low-Priority Instances ?

A low-priority instance means that it is interruptible—essentially, Microsoft Azure can take those resources and assign them to another task, thus interrupting a job. A dedicated instance, or non-interruptible, means that the job will never be terminated without your permission. This is another consideration of time vs money, since interruptible instances are less expensive than dedicated ones.

2.2.2 Creating a compute cluster

In the Azure ML workspace that we created earlier, go to compute and you will be able to see the different compute resources we just discussed (i.e compute instances, compute clusters, inference clusters and attached compute). For this project, we are going to need a compute cluster for model training. In the Studio, Click on the "Compute" menu, then the "Compute cluster" tab and click on the "+ New" button to create a compute cluster.

1. Choose your options: Dedicated vs Low priority, CPU or GPU, VM size and core number (you can keep the default settings for this project).
2. Click on the Next button.

3. Give the cluster a compute name
4. Choose your options: Minimum/Maximum number of nodes, Idle seconds before scale down, SSH access. Note that if the minimum number of nodes is 0, you will save money when the cluster is idle. Note that the higher the number of maximum nodes, the shorter the training will be. The maximum number of nodes recommended is 3.
5. Click on the "Create" button. This step may take a few minutes.

Awesome! Now that we have a Compute cluster, we need to load the data to Azure ML Studio.

2.3 Loading the Dataset

1. In the Azure ML workspace that we created earlier, click on "Datasets" in the left menu and click on the "+ Create dataset" button to create a dataset. Choose the "From local files" option and select the Kaggle dataset we downloaded earlier.

2. Give your dataset a name, a type and a description. Click Next. Upload the data from files. Click Next.

3. In the Schema, change the data type to Boolean for the following features: anaemia, diabetes, high blood pressure, sex, smoking, and DEATH_EVENT. Click Next and Click Create.

Great! Now that the dataset is in place and the compute cluster is created, we can start the training of the model!

2.4 Low code/No Code training with AutoML

Traditional machine learning model development is resource-intensive, requires significant domain knowledge and time to produce and compare dozens of models. Automated machine learning (AutoML), is the process of automating the time-consuming, iterative tasks of machine learning

model development. It allows data scientists, analysts, and developers to build ML models with high scale, efficiency, and productivity, all while sustaining model quality. It reduces the time it takes to get production-ready ML models, with great ease and efficiency. Learn more

1. In the Azure ML workspace that we created earlier click on "Automated ML" in the left menu and select the dataset you just uploaded. Click Next.

   
   27

2. Enter a new experiment name, the target column (DEATH_EVENT) and the compute cluster we created. Click Next.

   
   28

3. Choose "Classification" and Click Finish. This step might take between 30 minutes to 1 hour, depending upon your compute cluster size.

   
   30

4. Once the run is complete, click on the "Automated ML" tab, click on your run, and click on the Algorithm in the "Best model summary" card.

   
   31

Here you can see a detailed description of the best model that AutoML generated. You can also explore other modes generated in the Models tab. Take a few minutes to explore the models in the Explanations (preview button). Once you have chosen the model you want to use (here we will chose the best model selected by autoML), we will see how we can deploy it.

3. Low code/No Code model deployment and endpoint consumption

3.1 Model deployment

The automated machine learning interface allows you to deploy the best model as a web service in a few steps. Deployment is the integration of the model so that it can make predictions based on new data and identify potential areas of opportunity. For this project, deployment to a web service means that medical applications will be able to consume the model to be able to make live predictions of their patients risk to get a heart attack.

In the best model description, click on the "Deploy" button.


deploy-1

15. Give it a name, a description, compute type (Azure Container Instance), enable authentication and click on Deploy. This step might take about 20 minutes to complete. The deployment process entails several steps including registering the model, generating resources, and configuring them for the web service. A status message appears under Deploy status. Select Refresh periodically to check the deployment status. It is deployed and running when the status is "Healthy".

16. Once it has been deployed, click on the Endpoint tab and click on the endpoint you just deployed. You can find here all the details you need to know about the endpoint.

Amazing! Now that we have a model deployed, we can start the consumption of the endpoint.

3.2 Endpoint consumption

Click on the "Consume" tab. Here you can find the REST endpoint and a python script in the consumption option. Take some time to read the python code. This script can be run directly from your local machine and will consume your endpoint.

Take a moment to check those 2 lines of code:

```python
url = 'http://98e3715f-xxxx-xxxx-xxxx-9ec22d57b796.centralus.azurecontaine
api_key = '' # Replace this with the API key for the web service
```

The `url` variable is the REST endpoint found in the consume tab and the `api_key` variable is the primary key also found in the consume tab (only in the case you have enabled authentication). This is how the script can consume the endpoint.

18. Running the scrip, you should see the following output:

```python
b'"{\\"result\\": [true]}"'
```

This means that the prediction of heart failure for the data given is true. This makes sense because if you look more closely at the data automatically generated in the script, everything is at 0 and false by default. You can change the data with the following input sample:

```python
data = {
    "data":
    [
        {
            'age': "0",
            'anaemia': "false",
            'creatinine_phosphokinase': "0",
            'diabetes': "false",
            'ejection_fraction': "0",
            'high_blood_pressure': "false",
            'platelets': "0",
            'serum_creatinine': "0",
            'serum_sodium': "0",
            'sex': "false",
            'smoking': "false",
            'time': "0",
        },
        {
            'age': "60",
            'anaemia': "false",
            'creatinine_phosphokinase': "500",
            'diabetes': "false",
            'ejection_fraction': "38",
            'high_blood_pressure': "false",
            'platelets': "260000",
            'serum_creatinine': "1.40",
            'serum_sodium': "137",
            'sex': "false",
            'smoking': "false",
            'time': "130",
        },
    ],
}
```

The script should return : `python b'"{\\"result\\": [true, false]}"'`

Congratulations! You just consumed the model deployed and trained it on Azure ML !

> NOTE: Once you are done with the project, don't forget to delete all the resources.

🚀 Challenge

Look closely at the model explanations and details that AutoML generated for the top models. Try to understand why the best model is better than the other ones. What algorithms were compared? What are the differences between them? Why is the best one performing better in this case?

Post-Lecture Quiz

Review & Self Study

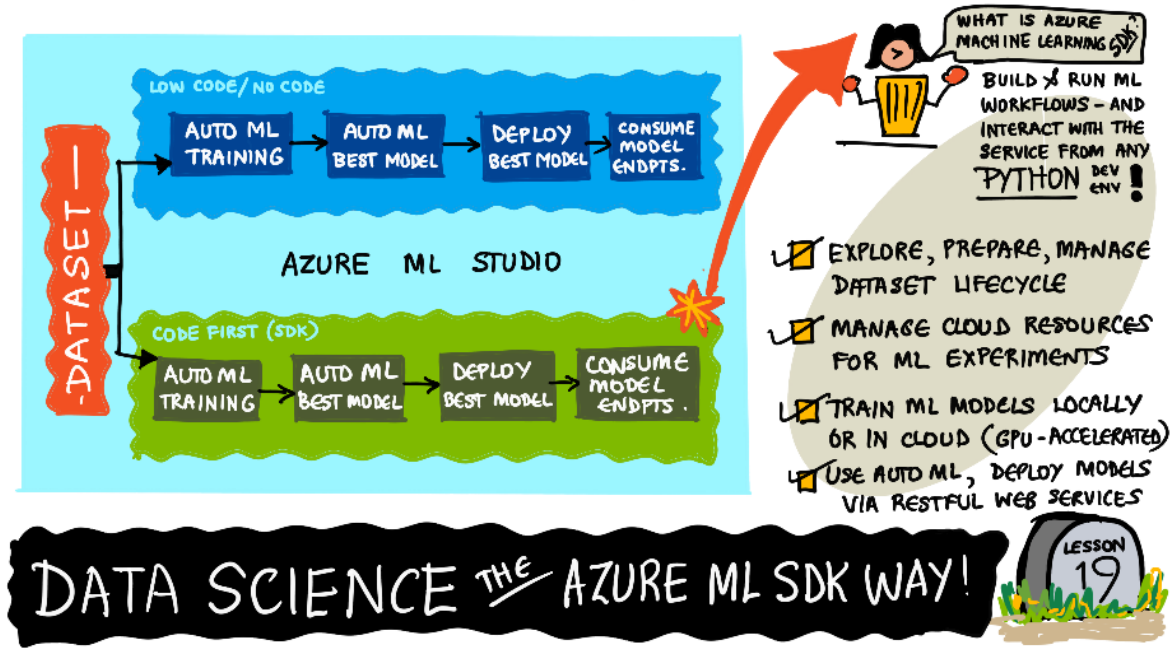
In this lesson, you learned how to train, deploy and consume a model to predict heart failure risk in a Low code/No code fashion in the cloud. If you have not done it yet, dive deeper into the model explanations that AutoML generated for the top models and try to understand why the best model is better than others.

You can go further into Low code/No code AutoML by reading this documentation.

Assignment

Low code/No code Data Science project on Azure ML

Data Science in the Cloud: The "Azure ML SDK" way



Data Science In The Cloud: Azure ML SDK - *Sketchnote by @nitya*

Table of contents:

Pre-Lecture Quiz

1. Introduction

1.1 What is Azure ML SDK?

Data scientists and AI developers use the Azure Machine Learning SDK to build and run machine learning workflows with the Azure Machine Learning service. You can interact with the service in any Python environment, including Jupyter Notebooks, Visual Studio Code, or your favorite Python IDE.

Key areas of the SDK include:

- Explore, prepare and manage the lifecycle of your datasets used in machine learning experiments.
- Manage cloud resources for monitoring, logging, and organizing your machine learning experiments.
- Train models either locally or by using cloud resources, including GPU-accelerated model training.
- Use automated machine learning, which accepts configuration parameters and training data. It automatically iterates through algorithms and hyperparameter settings to find the best model for running predictions.
- Deploy web services to convert your trained models into RESTful services that can be consumed in any application.

Learn more about the Azure Machine Learning SDK

In the previous lesson, we saw how to train, deploy and consume a model in a Low code/No code fashion. We used the Heart Failure dataset to generate and Heart failure prediction model. In this lesson, we are going to do the exact same thing but using the Azure Machine Learning SDK.

1.2 Heart failure prediction project and dataset introduction

Check here the Heart failure prediction project and dataset introduction.

2. Training a model with the Azure ML SDK

2.1 Create an Azure ML workspace

For simplicity, we are going to work on a jupyter notebook. This implies that you already have a Workspace and a compute instance. If you already have a Workspace, you can directly jump to the section 2.3 Notebook creation.

If not, please follow the instructions in the section **2.1 Create an Azure ML workspace** in the previous lesson to create a workspace.

2.2 Create a compute instance

In the Azure ML workspace that we created earlier, go to the compute menu and you will see the different compute resources available

Let's create a compute instance to provision a jupyter notebook.

1. Click on the + New button.
2. Give a name to your compute instance.
3. Choose your options: CPU or GPU, VM size and core number.
4. Click in the Create button.

Congratulations, you have just created a compute instance! We will use this compute instance to create a Notebook the Creating Notebooks section.

2.3 Loading the Dataset

Refer the previous lesson in the section **2.3 Loading the Dataset** if you have not uploaded the dataset yet.

2.4 Creating Notebooks

> NOTE: For the next step you can either create a new notebook from scratch, or you can upload the notebook we created in you Azure ML Studio. To upload it, simply click on the "Notebook" menu and upload the notebook.

Notebook are a really important part of the data science process. They can be used to Conduct Exploratory Data Analysis (EDA), call out to a computer cluster to train a model, call out to an inference cluster to deploy an endpoint.

To create a Notebook, we need a compute node that is serving out the jupyter notebook instance. Go back to the Azure ML workspace and click on Compute instances. In the list of compute instances you should see the compute instance we created earlier.

1. In the Applications section, click on the Jupyter option.
2. Tick the "Yes, I understand" box and click on the Continue button. 
3. This should open a new browser tab with your jupyter notebook instance as follow. Click on the "New" button to create a notebook.

Now that we have a Notebook, we can start training the model with Azure ML SDK.

2.5 Training a model

First of all, if you ever have a doubt, refer to the Azure ML SDK documentation. In contains all the necessary information to understand the modules we are going to see in this lesson.

2.5.1 Setup Workspace, experiment, compute cluster and dataset

You need to load the `workspace` from the configuration file using the following code:

```python
from azureml.core import Workspace
ws = Workspace.from_config()
```

This returns an object of type `Workspace` that represents the workspace. The you need to create an `experiment` using the following code:

```python
from azureml.core import Experiment
experiment_name = 'aml-experiment'
experiment = Experiment(ws, experiment_name)
```

To get or create an experiment from a workspace, you request the experiment using the experiment name. Experiment name must be 3-36 characters, start with a letter or a number, and can only contain letters, numbers, underscores, and dashes. If the experiment is not found in the workspace, a new experiment is created.

Now you need to create a compute cluster for the training using the following code. Note that this step can take a few minutes.

```python
from azureml.core.compute import AmlCompute

aml_name = "heart-f-cluster"
try:
    aml_compute = AmlCompute(ws, aml_name)
    print('Found existing AML compute context.')
except:
    print('Creating new AML compute context.')
    aml_config = AmlCompute.provisioning_configuration(vm_size = "Standard_
    aml_compute = AmlCompute.create(ws, name = aml_name, provisioning_conf
    aml_compute.wait_for_completion(show_output = True)

cts = ws.compute_targets
compute_target = cts[aml_name]
```

You can get the dataset from the workspace using the dataset name in the following way:

```python
dataset = ws.datasets['heart-failure-records']
df = dataset.to_pandas_dataframe()
df.describe()
```

2.5.2 AutoML Configuration and training

To set the AutoML configuration, use the AutoMLConfig class.

As described in the doc, there are a lot of parameters with which you can play with. For this project, we will use the following parameters:

- `experiment_timeout_minutes` : The maximum amount of time (in minutes) that the experiment is allowed to run before it is automatically stopped and results are automatically made available
- `max_concurrent_iterations` : The maximum number of concurrent training iterations allowed for the experiment.
- `primary_metric` : The primary metric used to determine the experiment's status.
- `compute_target` : The Azure Machine Learning compute target to run the Automated Machine Learning experiment on.
- `task` : The type of task to run. Values can be 'classification', 'regression', or 'forecasting' depending on the type of automated ML problem to solve.
- `training_data` : The training data to be used within the experiment. It should contain both training features and a label column (optionally a sample weights column).

- `label_column_name` : The name of the label column.
- `path` : The full path to the Azure Machine Learning project folder.
- `enable_early_stopping` : Whether to enable early termination if the score is not improving in the short term.
- `featurization` : Indicator for whether featurization step should be done automatically or not, or whether customized featurization should be used.
- `debug_log` : The log file to write debug information to.

```python
from azureml.train.automl import AutoMLConfig

project_folder = './aml-project'

automl_settings = {
    "experiment_timeout_minutes": 20,
    "max_concurrent_iterations": 3,
    "primary_metric" : 'AUC_weighted'
}

automl_config = AutoMLConfig(compute_target=compute_target,
                             task = "classification",
                             training_data=dataset,
                             label_column_name="DEATH_EVENT",
                             path = project_folder,
                             enable_early_stopping= True,
                             featurization= 'auto',
                             debug_log = "automl_errors.log",
                             **automl_settings
                            )
```

Now that you have your configuration set, you can train the model using the following code. This step can take up to an hour depending on your cluster size.

```python
remote_run = experiment.submit(automl_config)
```

You can run the RunDetails widget to show the different experiments.

```python
from azureml.widgets import RunDetails
RunDetails(remote_run).show()
```

3. Model deployment and endpoint consumption with the Azure ML SDK

3.1 Saving the best model

The `remote_run` an object of type AutoMLRun. This object contains the method `get_output()` which returns the best run and the corresponding fitted model.

```python
best_run, fitted_model = remote_run.get_output()
```

You can see the parameters used for the best model by just printing the fitted_model and see the properties of the best model by using the get_properties() method.

```python
best_run.get_properties()
```

Now register the model with the register_model method.

```python
model_name = best_run.properties['model_name']
script_file_name = 'inference/score.py'
best_run.download_file('outputs/scoring_file_v_1_0_0.py', 'inference/score
description = "aml heart failure project sdk"
model = best_run.register_model(model_name = model_name,
                                model_path = './outputs/',
                                description = description,
                                tags = None)
```

3.2 Model Deployment

Once the best model is saved, we can deploy it with the InferenceConfig class. InferenceConfig represents the configuration settings for a custom environment used for deployment. The AciWebservice class represents a machine learning model deployed as a web service endpoint on Azure Container Instances. A deployed service is created from a model, script, and associated files. The resulting web service is a load-balanced, HTTP endpoint with a REST API. You can send data to this API and receive the prediction returned by the model.

The model is deployed using the deploy method.

```python
from azureml.core.model import InferenceConfig, Model
from azureml.core.webservice import AciWebservice

inference_config = InferenceConfig(entry_script=script_file_name, environme

aciconfig = AciWebservice.deploy_configuration(cpu_cores = 1,
                                               memory_gb = 1,
                                               tags = {'type': "automl-hea
                                               description = 'Sample servi

aci_service_name = 'automl-hf-sdk'
aci_service = Model.deploy(ws, aci_service_name, [model], inference_config
aci_service.wait_for_deployment(True)
print(aci_service.state)
```

This step should take a few minutes.

3.3 Endpoint consumption

You consume your endpoint by creating a sample input:

```python
data = {
    "data":
    [
        {
            'age': "60",
            'anaemia': "false",
            'creatinine_phosphokinase': "500",
            'diabetes': "false",
            'ejection_fraction': "38",
            'high_blood_pressure': "false",
            'platelets': "260000",
            'serum_creatinine': "1.40",
            'serum_sodium': "137",
            'sex': "false",
            'smoking': "false",
            'time': "130",
        },
    ],
}
```

```
test_sample = str.encode(json.dumps(data))
```

And then you can send this input to your model for prediction :

```python
response = aci_service.run(input_data=test_sample)
response
```

This should output `'{"result": [false]}'` . This means that the patient input we sent to the endpoint generated the prediction `false` which means this person is not likely to have a heart attack.

Congratulations! You just consumed the model deployed and trained on Azure ML with the Azure ML SDK!

> NOTE: Once you are done with the project, don't forget to delete all the resources.

🚀 Challenge

There are many other things you can do through the SDK, unfortunately, we can not view them all in this lesson. But good news, learning how to skim through the SDK documentation can take you a long way on your own. Have a look at the Azure ML SDK documentation and find the `Pipeline` class that allows you to create pipelines. A Pipeline is a collection of steps which can be executed as a workflow.

HINT: Go to the SDK documentation and type keywords in the search bar like "Pipeline". You should have the `azureml.pipeline.core.Pipeline` class in the search results.

Post-lecture quiz

1. What is the reason for creating an AutoMLConfig?
    1. It is where the training and the testing data are split
    2. TRUE : It provides all the details of your AutoML experiment
    3. It is where you specify the model to be trained
2. Which of the following metrics is supported by Automated ML for a classification task?

1. TRUE : accuracy
2. r2_score
3. normalized_root_mean_error

3. What is NOT an advantage of using the SDK?
    1. It can be used to automate multiple tasks and runs
    2. It makes it easier to programmatically edit runs
    3. It can be used throught a Graphical User Interface

Review & Self Study

In this lesson, you learned how to train, deploy and consume a model to predict heart failure risk with the Azure ML SDK in the cloud. Check this documentation for further information about the Azure ML SDK. Try to create your own model with the Azure ML SDK.

Assignment

Data Science project using Azure ML SDK

Data Science in the Wild

Real-world applications of data science across industries.

Topics

1. Data Science in the Real World

Credits

Written with ❤️ by Nitya Narasimhan