Course 1: **Foundations: Data, Data, Everywhere**

Week-1

The Data Analysis process consists of steps:

1) **Ask:** Analysts should define the problem they are going to solve and decide on the project's successful outcome. To do so, they might **ask** different questions to org's authorities like managers, leaders, etc.

2) **Prepare: Preparing** the guidelines for the project like a timeline of the project, what type of data needed to be collected, giving limited access of the data to analysts, etc. Analyze the data needed and start working on collecting the data from the sources.

3) **Process:** Maintaining the privacy of the sources of data. Data Analysts should keep the data confidential and protect it effectively. Consent should be taken from the source to use their data. Also, they should be informed about how their data would be **collected, stored, managed, and protected.** Once everything above has been done properly, analysts should make sure it is correct, complete, and clean.

4) **Analyze:** Analysts analyze the data to find any patterns for a particular result. No matter the result, whether it is in favor of them or against, they should document every result.

5) **Share:** The results are shared with the higher authorities like managers, leaders for them to make appropriate decisions based on the results.

6) **Act:** The last stage of the process for the team of analysts was to work with leaders within their company and decide how best to implement changes and take actions based on the findings.

Successful analysis needs to be accurate, and fast enough to help decision-makers. So try asking yourself these questions about a project:

- What kind of results are needed?
- Who will be informed?
- Am I answering the question being asked?
- How quickly does a decision need to be made?

Week-2

Analytical Skills:

1. **Curiosity:**
   Eagerness to learn new things.

2. **Understanding the context:**
   The condition in which something exists or happens. Something which makes sense.

3. **Having a technical mindset:**
   The ability to break things down into smaller steps or pieces and work with them in an orderly or logical way.

4. **Data design:**
   The organization of the information(databases).

5. **Data Strategy:**
   The management of people(giving the right data to the people so that they can create solutions to the problems), processes(showing the right path that leads to the right decision), and tools(using right technologies or algorithms, etc) used in data analysis.

Aspects of analytical thinking:

Analytical thinking: Identifying or defining a problem and solving it using data in an organized, step-by-step manner.

1. **Visualization:**
   The graphical representation of information.

2. **Strategy:**
   Strategies can help DAs see what to achieve with the data available and they can get there. It also improves the quality and the usefulness of the data collected.

3. **Problem-orientation:**
   DAs use a problem-oriented approach to identify, describe and solve any problem. While building any project they should have the ultimate problem in mind which needs to be solved.

4. **Correlation:**
   Relation between two or more pieces. Correlation does not equal causation.

5. **Big-Picture and detailed-oriented thinking**
   Data analysts use a big-picture approach to zoom out and see possibilities and opportunities. Detail-oriented thinking is about figuring out all of the specifics that will help you execute a plan.

Exploring Core Analytical Skills:

What questions do DAs ask to solve the problem:

1. What is the root cause of the problem?
   Root cause: The reason why a problem occurs.
   If we can identify and get rid of the root cause, we can help to prevent the problem from occurring again.

2. Ask "Why?" five times to reveal the root cause
   For example, Let's say I want to make a blueberry pie. But I can't:
   - Why can't I make a blueberry pie? Because there are no blueberries at the store
   - Why are there no blueberries at the store? Because blueberry bushes didn't produce enough fruits.
   - Why didn't blueberry bushes produce fruits? Because birds eat them
   - Why do birds eat them? Birds chose blueberry over mulberries this season because mulberries didn't produce fruits this season.
   - Why didn't mulberries were not produced? Because late frost damaged the mulberry bushes and fruits, which is our root cause.

3. Where are gaps in our process?
   Gap Analysis: The method of examining and evaluating how a process works currently in order to get where you want it to be in the future.

4. What did we not consider before?
   This shows what information or what procedures we miss before.

Data-driven Decision making:
Using facts to guide business strategy

Week-3

Phases of the data life cycle:

1. **Planning:**
   This happens way before any analysis project takes place. The business decides on what data is needed, how the data will flow through the phases, and who would be responsible for it.

2. **Capture:**
   Data is collected from different sources and brought into the organization.

3. **Manage:**
   How and where the data is stored. The tools used to keep it safe and secure. Data cleansing is the most important part.

4. **Analyze:**
   Data is used to solve problems, make the right decisions and support business goals.

5. **Archive:**
   Storing data in a place where it is available but used very rarely.

6. **Destroy:**
   Important to protect the company's private information. In order to keep the information safe, the analyst uses secure data-erasure software for the digital files and a shredder for the paper files.

Stakeholders:
People who have invested time and resources into a project and are interested in the outcomes.

Finding a problem in the ASK phase:
Look at the current state and identify how it is different from the ideal state.

Redefinition of Data Analytics process steps:

1. **Ask:** You want to ask all of the right questions at the beginning of the engagement so that you better understand what your leaders and stakeholders need from this analysis.

2. **Prepare:** We need to be thinking about the type of data we need in order to answer the questions that we've set out to answer based on what we learned when we asked the right questions.

3. **Process:** This is where you get a chance to understand its structure, its quirks, its nuances, and you really get a chance to understand deeply what type of data you're going to be working with and understanding what potential that data has to answer all of your questions.

4. **Analyze:** This is the point where we have to take a step back and let the data speak for itself.

5. **Share:** The results are shared with the stakeholders like managers, leaders for them to make appropriate decisions based on the results.

6. **Act:** All of this work from asking the right questions to collecting your data, to analyzing and sharing, doesn't mean much of anything if we aren't taking action on what we've just learned.

Data Analytics Tools:

- **Spreadsheets:** Microsoft Excel, Google Sheets
- **Query languages for databases:** SQL
- **Visualization:** Tableau, Looker

Week-4

Data Analytics Tools

Spreadsheets:

- Column values can be sorted from the **Data** option up in the menu bar.

Week-5

Business Task: The question or problem data analysis answers for a business.

Business task types:

1. **Issue:** A topic or subject to investigate.

2. **Question:** Designed to discover information.

3. **Problem:** An obstacle or complication that needs to be worked out.

Fairness: Ensuring that your analysis does not create or reinforce bias.

Data analyst roles and job descriptions

Decoding the job description



	Data Analysts	Data Scientists	Data Specialists
Problem solving	Use existing tools and methods to solve problems with existing types of data	Invent new tools and models, ask open-ended questions, and collect new types of data	Use in-depth knowledge of databases as a tool to solve problems and manage data
Analysis	Analyze collected data to help stakeholders make better decisions	Analyze and interpret complex data to make business predictions	Organize large volumes of data for use in data analytics or business operations
Other relevant skills	• Database queries • Data visualization • Dashboards • Reports • Spreadsheets	• Advanced statistics • Machine learning • Deep learning • Data optimization • Programming	• Data manipulation • Information security • Data models • Scalability of data • Disaster recovery

Course 2: **Ask Questions to Make Data-Driven Decisions**

Week-1

Structured Thinking: The process of recognizing the current problem or situation, organizing available information, revealing gaps and opportunities, and identifying the options.

Six problem types:

1. Making predictions
   Eg Problem: How to determine the best advertising method for a target audience.

2. Categorizing things
   Eg Problem: How to improve customer satisfaction levels

3. Spotting something unusual
   Eg Problem: How to stay healthy

4. Identifying themes
   Eg Problem: How to improve user experience

5. Discovering connections
   Eg Problem: How to reduce waiting times.

6. Finding patterns
   Eg Problem: How to stop machines from breaking down.

Effective questions:

1. **Leading question:** These kinds of questions lead the answer in a specific way.
   Eg: These pancakes are delicious, aren't they?
   - This can be answered in a single way

2. **Closed-ended question:** Answers which can be answered with either yes or no.
   Eg: Did you enjoy growing up in India?
   - Yes

Ineffective question:

1. **Questions that are too vague and lacks context**
   Eg: Do you like chocolate or Vanilla?
   - Chocolate and vanilla can be of anything, either chocolate ice cream or vanilla flavor in coffee etc. This question lacks context.

Effective questions follow SMART methodology:

1. **Specific:** Specific questions are simple, significant and focused on a single topic or a few closely related topics.

   Are kids getting enough exercise these days? Instead of closed-ended question like this, ask
   What percentage of kids achieve the recommended 60 minutes of physical activity at least five days a week? Which is more specific and gives more information.

2. **Measurable:** Measurable questions can be quantified and assessed.

   Why did our recent video go viral? Which is an unmeasurable question
   How many times was our video shared on social channels the first week it was posted? Which is more measurable as we can come with a concrete number.

3. **Action-oriented:** Action-oriented questions encourage change. As the definition of problem-solving states “Recognizing the current state and figuring out the ideal state in the future”, action-oriented questions will help with problem-solving.

   How can we get our customers to recycle our product packaging?
   What design feature will make our packaging easier to recycle? Which requires action to be done.

4. **Relevant:** Relevant questions are important, are important and have significance to the problem you are going to solve.

   Why does it matter that Pine Barrens tree frogs started disappearing? This question will not help in solving(irrelevant) why the frogs got extinct, but instead if we ask
   What environmental factors changed in Durham, North Carolina between 1983 in 2004 that could cause Pine Barrens tree frogs to disappear from the Sandhills Regions?

5. **Time-bound:** Time-bound questions specify the time to be studied. For example, 1983-2004 in the previous question.

Week-2

Note:
Data: Collection of facts
Data Analysis: Revealing important patterns and insights from that data
Can help us make informed decisions.

Data Inspired decision making: Explores different data sources to find out what they have in common.
Algorithm: A process or set of rules to be followed for a specific task.

- Most of the data is meaningless until someone adds some interpretation and turns it into information and gains knowledge.

Qualitative and Quantitative Data:

- **Quantitative Data:** Specific and objective measures of numerical facts. Things that can be measured.
  Quantitative data can be visualized using **Charts** and **Graphs.**
- **Qualitative Data:** Subjective and explanatory measures of qualities and characteristics. Qualitative data is important to figure out why the numbers are the way they are.

  Example for above types of data:

- A local ice cream shop uses online platform's reviews to engage with their customers.
- The company uses those reviews to improve their business, if there are any negative reviews.
- Lets say, the customers are not satisfied by their ice creams and post negative comments on the product by simultaneously giving a low rating.
- To know why, the company's analyst starts asking questions, starting with measurable questions, which could be
  1) How many negative reviews are there?
  2) What is the average rating?
  3) How many of these use the same negative keywords?
- These numeric questions lead to more qualitative questions, such as **Why are customers unsatisfied? How can we improve their experience?**
- Quantitative conclusion the analyst made was that the customers used the word "frustrated" 17 times.
- Immediately, a qualitative question raises, which could be, why is the word used frequently?
- They then figured out that it was because the shop was running out of the popular flavors before the end of the day.
- Knowing this, the owner can order abundant supply to satisfy the customers.

<u>Reports and Dashboards:</u>

- Reports and Dashboards are two data presentation tools (used for visualizations).

Report: Static collection of data given to stakeholders periodically (weekly or monthly).

Pros:
- Giving snapshots of **high-level historical data**.
- They are easy to use and **quick to design**.
- Since they are static, they use **pre-cleaned and sorted** data.

Cons:
- They require **Continual maintenance**
- They are **visually less appealing.**

Dashboard: Monitors live, and incoming data.

Pros:
- Dynamic, automatic, and interactive.
- Easy stakeholder access
- Low maintenance

Cons:
- Labor-intensive design
- Can be confusing

- Potentially uncleaned data

Note: It is important to remember that changed data is pulled into dashboards automatically only if the data structure is the same. If the data structure changes, you have to update the dashboard design before the data can update live.

Pivot Table: A pivot table is a data summarization tool that is used in data processing. Pivot tables are used to summarize, sort, re-organize, group, count, total, or average data stored in a database.

Metrics:

- A metric is a single, quantifiable type of data that can be used for measurement.
- Metrics can also be combined into formulas in which we can plug our numerical data to calculate
  For example: In a company's sales data, if we want to find the revenue made by each sales person.
  Revenue = no.of sales * sales cost
- Most commonly used metric is **ROI (Return on Investment)**

  ROI = Net profit over time / Cost of investment
  ROI is used for companies to analyze how well their investments are doing.
- Metrics are used in marketing to help calculate customer retention rates, or a company's ability to keep its customers over time.
- **Metric goal:** A measurable goal set by the company and evaluated using metrics.

Mathematical thinking: Step-by-step breaking down of the problem to see relationships or patterns in the data and use this to analyze the problem.

The three (or Four) V words for big data:
When thinking about the benefits and challenges of big data, it helps to think about the three Vs: **volume, variety, and velocity. Volume** describes the amount of data. **Variety** describes the different kinds of data. **Velocity** describes how fast the data can be processed. Some data analysts also consider a fourth V: **veracity. Veracity** refers to the quality and reliability of the data. These are all important considerations related to processing huge, complex data sets.

Week-3

Structured Thinking (again): The process of recognizing the current problem or situation, organizing available information, revealing gaps or opportunities, and identifying problems.
In other words, the process is of being super prepared, creating a list of what needs to be delivered, the timelines for major tasks and activities and checkpoints (to make the team know where are we in the project)

The starting phase of structured thinking is the **problem domain** - the specific area of analysis which encompasses every activity affecting or affected by the problem. **In other words**, laying out a proper base with all the requirements and hypotheses before starting investigating (or analyzing) the data further (solving the problem) to prevent obstacles.

An example of an obstacle would be, let say if you're to predict the price of an apartment building based on 100's of variables in a dataset, if you miss out at least one important variable accidentally, example sq.ft, then you must re-do the entire process. Because missing crucial variables could lead to inaccurate conclusions.

Another way to practice structure thinking and avoid mistakes is using:
Scope of Work(SOW): An agreed-upon outline of the work you are going to do on a project.

For many businesses, this includes work details, schedules or reports to be shared with clients.

For data analysts, it is more technical, like data preparation, data cleaning, analysis of qualitative and quantitative datasets, analysis results or visuals.

The point of data analysis projects is to complete business tasks that are useful to the stakeholders. Creating an SOW helps to make sure that everyone involved, from analysts and engineers to managers and stakeholders, shares the understanding of what those business goals are, and the plan for accomplishing them.

As you ask more and more questions to clarify requirements, goals, data sources, stakeholders, and any other relevant info, an SOW helps you formalize it all by recording all the answers and details. Preparing to write an SOW is about asking questions to learn the necessary information about the project, but it's also about clarifying and defining what you're being asked to accomplish.

What is a good SOW:

There's no standard format for an SOW. They may differ significantly from one organization to another, or from project to project. However, they all have a few foundational pieces of content in common.

- **Deliverables:** What work is being done, and what things are being created as a result of this project? When the project is complete, what are you expected to deliver to the stakeholders? Be specific here. Will you collect data for this project? How much, or for how long?

Avoid vague statements. For example, "fixing traffic problems" doesn't specify the scope. This could mean anything from filling in a few potholes to building a new overpass. Be specific! Use numbers and aim for hard, measurable goals and objectives. For example: "Identify top 10

issues with traffic patterns within the city limits, and identify the top 3 solutions that are most cost-effective for reducing traffic congestion."

- **Milestones:** This is closely related to your timeline. What are the major milestones for progress in your project? How do you know when a given part of the project is considered complete?

Milestones can be identified by you, by stakeholders, or by other team members such as the Project Manager. Smaller examples might include incremental steps in a larger project like "Collect and process 50% of required data (100 survey responses)", but may also be larger examples like "complete initial data analysis report" or "deliver completed dashboard visualizations and analysis reports to stakeholders".

- **Timeline:** Your timeline will be closely tied to the milestones you create for your project. The timeline is a way of mapping expectations for how long each step of the process should take. The timeline should be specific enough to help all involved decide if a project is on schedule. When will the deliverables be completed? How long do you expect the project will take to complete? If all goes as planned, how long do you expect each component of the project will take? When can we expect to reach each milestone?

- **Reports:** Good SOWs also set boundaries for how and when you'll give status updates to stakeholders. How will you communicate progress with stakeholders and sponsors, and how often? Will progress be reported weekly? Monthly? When milestones are completed? What information will status reports contain?

At a minimum, any SOW should answer all the relevant questions in the above areas. Note that these areas may differ depending on the project. But at their core, the SOW document should always serve the same purpose by containing information that is specific, relevant, and accurate. If something changes in the project, your SOW should reflect those changes.

What is in and out of scope?

SOWs should also contain information specific to what is and isn't considered part of the project. The scope of your project is everything that you are expected to complete or accomplish, defined to a level of detail that doesn't leave any ambiguity or confusion about whether a given task or item is part of the project or not.

Notice how the previous example about studying traffic congestion defined its scope as the area within the city limits. This doesn't leave any room for confusion — stakeholders need only to refer to a map to tell if a stretch of road or intersection is part of the project or not. Defining requirements can be trickier than it sounds, so it's important to be as specific as possible in these documents, and to use quantitative statements whenever possible.

For example, assume that you're assigned to a project that involves studying the environmental effects of climate change on the coastline of a city: How do you define what parts of the coastline you are responsible for studying, and which parts you are not?

In this case, it would be important to define the area you're expected to study using GPS locations, or landmarks. Using specific, quantifiable statements will help ensure that everyone has a clear understanding of what's expected.

Contextualizing data and recognizing data bias:

A data analyst considers who, what, when, where, why, and how in order to put information into context.

To ensure the data is accurate and fair, we should make sure we start with an accurate representation of the population in the sample, collect the data in an objective way, and think through the "who, what, when, where, why, and how" of the data.

Week-4

Working effectively with stakeholders

Tips to communicate clearly, establish trust, and deliver your findings across groups:

Discuss goals. Stakeholder requests are often tied to a bigger project or goal. When they ask you for something, take the opportunity to learn more. Start a discussion. Ask about the kind of results the stakeholder wants. Sometimes, a quick chat about goals can help set expectations and plan the next steps.

Feel empowered to say "no." Let's say you are approached by a marketing director who has a "high-priority" project and needs data to back up their hypothesis. They ask you to produce the analysis and charts for a presentation by tomorrow morning. Maybe you realize their hypothesis isn't fully formed and you have helpful ideas about a better way to approach the analysis. Or maybe you realize it will take more time and effort to perform the analysis than estimated. Whatever the case may be, don't be afraid to push back when you need to.

Stakeholders don't always realize the time and effort that goes into collecting and analyzing data. They also might not know what they actually need. You can help stakeholders by asking about their goals and determining whether you can deliver what they need. If you can't, have the confidence to say "no," and provide a respectful explanation. If there's an option that would be more helpful, point the stakeholder toward those resources. If you find that you need to prioritize other projects first, discuss what you can prioritize and when. When your stakeholders understand what needs to be done and what can be accomplished in a given timeline, they will usually be comfortable resetting their expectations. You should feel empowered to say no-- just remember to give context so others understand why.

Plan for the unexpected. Before you start a project, make a list of potential roadblocks. Then, when you discuss project expectations and timelines with your stakeholders, give yourself some extra time for problem-solving at each stage of the process.

Know your project. Keep track of your discussions about the project over email or reports, and be ready to answer questions about how certain aspects are important for your organization. Get to know how your project connects to the rest of the company and get involved in providing the most insight possible. If you have a good understanding about why you are doing an analysis, it can help you connect your work with other goals and be more effective at solving larger problems.

Start with words <u>and</u> visuals. It is common for data analysts and stakeholders to interpret things in different ways while assuming the other is on the same page. This *illusion of agreement** has been historically identified as a cause of projects going back-and-forth a number of times before a direction is finally nailed down. To help avoid this, start with a description and a quick visual of what you are trying to convey. Stakeholders have many points of view and may prefer to absorb information in words or pictures. Work with them to make changes and improvements from there. The faster everyone agrees, the faster you can perform the first analysis to test the usefulness of the project, measure the feedback, learn from the data, and implement changes.

Communicate often. Your stakeholders will want regular updates on your projects. Share notes about project milestones, setbacks, and changes. Then use your notes to create a shareable report. Another great resource to use is a change-log, which you will learn more throughout the program. For now, just know that a change-log is a file containing a chronologically ordered list of modifications made to a project. Depending on the way you set it up, stakeholders can even pop in and view updates whenever they want.

Three thing to focus on that will help you stay on objective:

- Who are the primary and secondary stakeholders?
- Who is managing the data? (Who else is managing the data? to be precise)
- Where can you go for help?

<u>Clear Communication:</u>

Before you communicate, think about:

- Who your audience is
- What they already know
- What they need to know
- How you can communicate that effectively to them

Course 3: **Prepare data for exploration**

Week-1

Data Collection in our world:

How data is collected:

- Interviews
- Observations
- Forms
- Questionnaires
- Surveys
- Cookies

Data Collection Considerations:

- *How the data will be collected.*
  Decide if you will collect the data using your own resources or receive (and possibly purchase it) from another party. Data that you collect yourself is called first-party data.

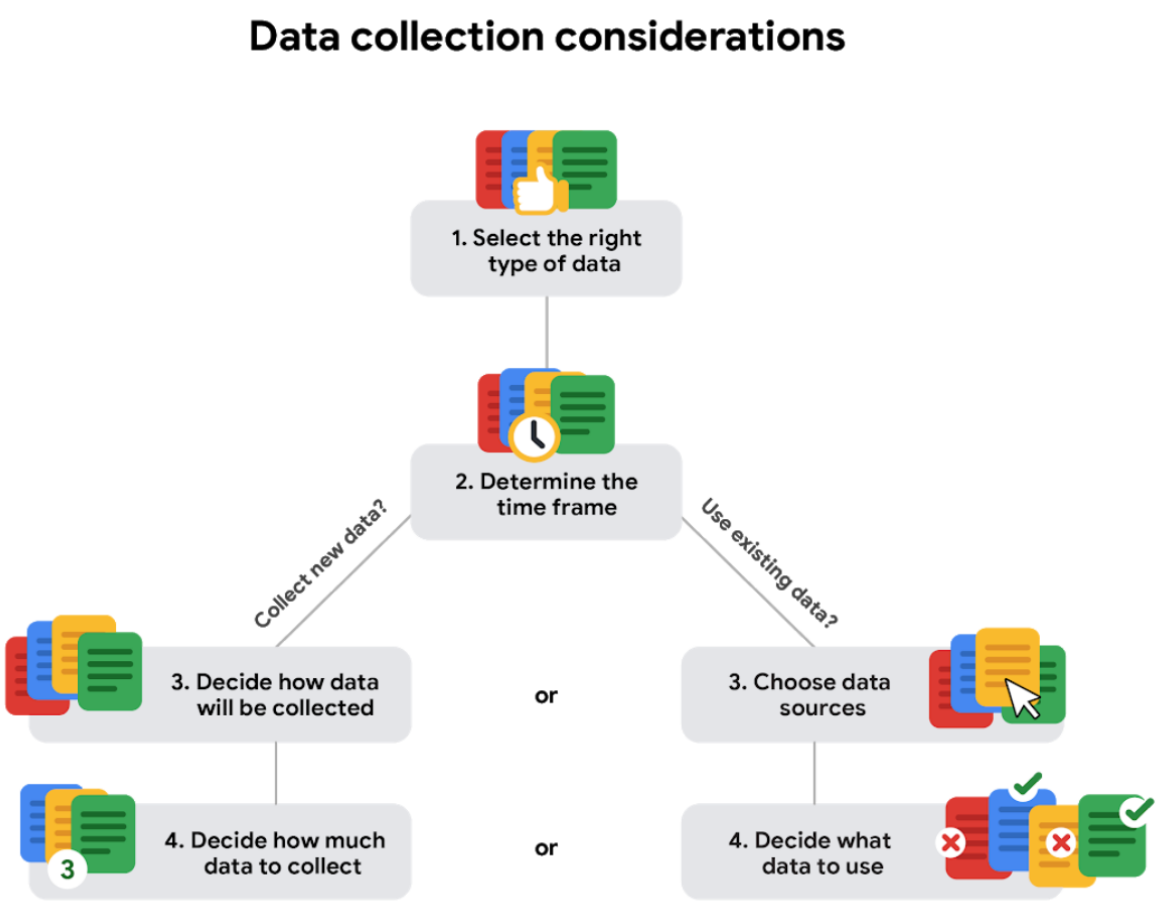
- *Choosing data sources*
  - **First-party data:** Data collected by an individual or group using their own resources.
  - **Second-party data:** Data collected by a group directly from its audience and then sold.
  - **Third-party data:** Data collected from outside sources who did not collect it directly.

- *Decide what data to use*

- *How much data to collect*
  If you are collecting your own data, make reasonable decisions about sample size. A random sample from existing data might be fine for some projects. Other projects might need more strategic data collection to focus on certain criteria. Each project has its own needs.

- *Select right data type*

- *Determine the time frame (how recent or old should the data be)*
  If you are collecting your own data, decide how long you will need to collect it, especially if you are tracking trends over a long period of time. If you need an immediate answer, you might not have time to collect new data. In this case, you would need to use historical data that already exists.



Data Formats:

Quantitative Data: Data which can be measured, counted or which contains numbers.

Discrete Data: Data that is counted and has a limited number of values.
Continuous Data: Data that is measured and can have almost any numeric value.
Example: Running time of a movie: 110.04544

Qualitative Data: Subjective and explanatory measures of qualities and characteristics

Nominal Data: A type of nominal data that is categorized without a set order.
Example: Did you like the movie? Answer can be Yes, No or not sure (No specific order to answer)
Ordinal Data: A type of qualitative data with a set order or scale
Example: Movie ratings from 1 to 5 stars or poor to excellent

Internal Data: Data that lives within a company's own systems.

External Data: Data that lives and is organized outside of an organization. It is useful when your analysis depends on as many sources as possible.

Structured Data: Data organized in a certain format such as rows and columns.
Example: Spreadsheets and Relational Databases
Unstructured Data: Data that is not organized in any easily identifiable manner
Example: Audio and Video files.

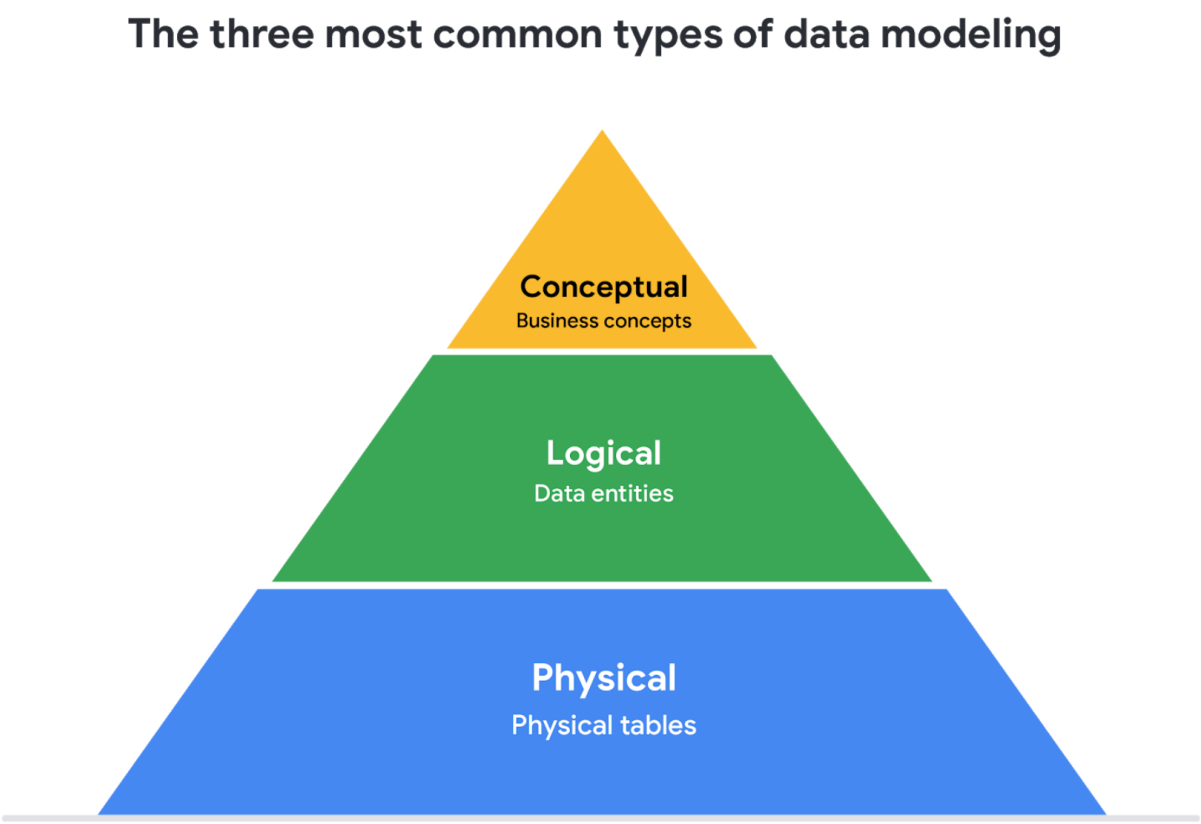
Data Model: A model that is used to organize data elements and how they relate to each other. It helps to keep data consistent and provide a map of how data is organized.
Data Elements: Pieces of information such as people's names, contact numbers and addresses.

Data Modeling:

- Data modeling is the process of creating diagrams that visually represent how data is organized and structured. These visual representations are called data models. You can think of data modeling as a blueprint of a house. At any point, there might be electricians, carpenters, and plumbers using that blueprint. Each one of these builders has a different relationship to the blueprint, but they all need it to understand the overall structure of the house. Data models are similar; different users might have different data needs, but the data model gives them an understanding of the structure as a whole.

Levels of data modeling: Each level of data modeling has a different level of detail.

1. **Conceptual data modeling** gives you a high-level view of your data structure, such as how you want data to interact across an organization.
2. **Logical data modeling** focuses on the technical details of the model such as relationships, attributes, and entities.
3. **Physical data modeling** should actually depict how the database was built. By this stage, you are laying out how each database will be put in place and how the databases, applications, and features will interact in specific detail.



Data modeling techniques:

- There are a lot of approaches when it comes to developing data models, but two common methods are the **Entity Relationship Diagram (ERD)** and the **Unified Modeling Language (UML)** diagram. ERDs are a visual way to understand the relationship between entities in the data model. UML diagrams are very detailed diagrams that describe the structure of a system by showing the system's entities, attributes, operations, and the relationships.

Data Analysis and Data modeling:

- Data modeling can help you explore the high-level details of your data and how it is related across the organization’s information systems. Data modeling sometimes requires data analysis to understand how the data is put together; that way, you know how to map the data. And finally, data models make it easier for everyone in your organization to understand and collaborate with you on your data. This is important for you and everyone on your team!

Data Types used in Spreadsheets:

- Numbers
- Text or string
- Boolean values

Wide Data: With wide data, every data subject has a single row with multiple columns to hold the values of various attributes of the subject.

Long Data: Data in which each row is a one-time point per subject, so each subject will have data in multiple rows.

Week-2

Introduction to bias, credibility, privacy, ethics, and access

Bias: A preference in favor of or against a person, group of people, or a thing.
Data Bias: A type of error that systematically skews the results in a certain direction.

As data analysts, we have to think about the bias and fairness right from collecting the data to presenting the conclusions to the stakeholders.

Sampling bias: A sample that isn't representative of the population as a whole. We can avoid this by randomly choosing samples and giving each part of the population an equal chance of being chosen.

Unbiased sampling: A sample that is representative of the population being measured.

There are some situations where the biased sample is useful. If you are looking for actionable data, biased data can narrow your focus. Example: To predict the future sales of a company, a data analyst might only consider the sales of the previous year, which is a biased sample.

Types of Bias:

- **Sampling bias:** A sample that isn't representative of the population as a whole. We can avoid this by randomly choosing samples and giving each part of the population an equal chance of being chosen.

- **Observer bias (experimenter/researcher bias):** The tendency for different people to observe things differently.
  Example: Scientists use observations a lot. They look at bacteria under the microscope. But when two scientists look through the same microscope, they might find different observations.

- **Interpretation bias:** The tendency to always interpret ambiguous situations in a positive or negative way.
  Example: A voicemail from the boss may seem angry to an employee but it may seem friendly to a stranger (employee's friend)

- **Confirmation bias:** The tendency to search for or interpret information in a way that confirms pre-existing beliefs.

Good Data Sources:

It's always better and efficient to work on good data taken from reliable sources. Following are the qualities required to identify good data (**ROCCC**)

- **Reliable:** With this kind of data, we can be sure that the data is accurate, complete, and contains unbiased information that's been vetted and proven fit for use.

- **Original:** To make sure we are dealing with good data, we should validate the data with the original source.

- **Comprehensive:** The best data sources contain all critical information used to either answer a question or find a solution.

- **Current:** The usefulness of data decreases as time passes.

- **Cited:** Citing makes the information we receive more credible.
  To determine if a data source is cited, we should ask the following questions:
  1) Who created this dataset?
  2) Is this dataset from a credible organization?

<u>Data Ethics</u>

Ethics: A set of principles to follow. In other words, well-founded standards of right and wrong that prescribe what humans ought to do, usually in terms of rights, obligations, benefits to society, fairness, or specific virtues.

Data Ethics: A well-founded standards of right and wrong that dictate how the data is collected, shared, and used.

Aspects of Data Ethics:

- **Ownership:** Individuals who own the raw data they provide, and they have primary control over its usage, how it's processed and how it's shared.

- **Transaction transparency:** All data processing activities and algorithms should be completely explainable and understood by the individual who provides their data.

- **Consent:** An individual's right to know explicit details about how and why their data will be used before agreeing to provide it.

- **Currency:** Individuals should be aware of financial transactions resulting from the use of their personal data and the scale of these transactions.
- **Privacy:** Preserving a data subject's information and activity any time a data transaction occurs.
  People should have:
    - Protection from unauthorized access to their private data.
    - Freedom from inappropriate use of their data.
    - The right to inspect, update, or correct their data.
    - Ability to give consent to use their data.
    - Legal right to access their data.

- **Openness (or Open data):** Openness refers to free access, usage, and sharing of data. But for data to be considered open, it has to:
    - Be available and accessible to the public as a complete dataset
    - Be provided under terms that allow it to be reused and redistributed
    - Allow universal participation so that anyone can use, reuse, and redistribute the data

Data Anonymization:

- **Personally identifiable information**, or PII, is information that can be used by itself or with other data to track down a person's identity.
- Data anonymization is the process of protecting people's private or sensitive data by eliminating that kind of information. Typically, data anonymization involves blanking, hashing, or masking personal information, often by using fixed-length codes to represent data columns, or hiding data with altered values.
- Organizations have a responsibility to protect their data and the personal information that data might contain. As a data analyst, you might be expected to understand what data needs to be anonymized, but you generally wouldn't be responsible for the data anonymization itself. A rare exception might be if you work with a copy of the data for testing or development purposes. In this case, you could be required to anonymize the data before you work with it.

What types of data should be anonymized?

- Healthcare and financial data are two of the most sensitive types of data. These industries rely a lot on data anonymization techniques. After all, the stakes are very high. That’s why data in these two industries usually goes through **de-identification**, which is a process used to wipe data clean of all personally identifying information.

- Data anonymization is used in just about every industry. That is why it is so important for data analysts to understand the basics. Here is a list of data that is often anonymized:
    - Telephone numbers
    - Names
    - License plates and License numbers
    - Social surety numbers
    - IP addresses
    - Medical records
    - Email addresses
    - Photographs
    - Account numbers

- For some people, it just makes sense that this type of data should be anonymized. For others, we have to be very specific about what needs to be anonymized. Imagine a world where we all had access to each other's addresses, account numbers, and other identifiable information. That would invade a lot of people's privacy and make the world less safe. Data anonymization is one of the ways we can keep data private and secure!

Week-3

Working with Databases

Database: A collection of data stored in a computer system. Databases let analysts manipulate, store, and process data. This helps them search through data a lot more efficiently to get the best insights.

Meta: Referencing back to itself.
Example: If a character in a book knows she's in a book.
If we are making a documentary about making a documentary.
If we analyze how we are analyzing our data.

Metadata: Data about data. Metadata tells where the data comes from, when and how it was created, and what it is about.

Relational Database: A database that contains a series of tables that can be connected to form relationships. Basically, they allow data analysts to organize and link data based on what the data has in common.

In a non-relational table, you will find all of the possible variables you might be interested in analyzing all grouped together. This can make it really hard to sort through. This is one reason why relational databases are so common in data analysis: they simplify a lot of analysis processes and make data easier to find and use across an entire database.

Primary key: An identifier that references a column in which each value is unique.
Foreign Key: A field within a table that is a primary key in another table.

Note: A table can only have one primary key, but it can have multiple foreign keys. These keys are what create the relationships between tables in a relational database, which helps organize and connect data across multiple tables in the database.

Normalized database: A database in which only related data is stored in each table. The main idea behind database normalization is that a table should be about a specific topic and only includes supporting related data. This minimizes the chances of having redundancy.
Redundancy: When the same piece of data is stored in two or more separate places.

Schema: A way of describing how something is organized.

Structured Query Language(SQL): Databases use a special language to communicate called a query language. Structured Query Language (SQL) is a type of query language that lets data analysts communicate with a database. So, a data analyst will use SQL to create a query to view the specific data that they want from within the larger set. In a relational database, data analysts can write queries to get data from the related tables. SQL is a powerful tool for working with databases.

Metadata (continued): Metadata is used in database management to help data analysts interpret the contents of the data within the database.

Three types of Metadata:

- **Descriptive metadata:** Metadata that describes a piece of data and can be used to identify it at a later point of time.
  Example: A book details like author, title of the book, ISBN of the book, etc

- **Structural metadata:** Metadata that indicates how a piece of data is organized and whether it is part of one, or more than one, data collection.
  Example: How the chapters of the book are organized into page numbers.

- **Administrative metadata:** Metadata that indicates the technical source of a digital asset.
  Example: Information about an image on the phone.

Characteristics of metadata:

- One of the most valuable things metadata does is putting data into **context.**

- Metadata creates a single source of truth by keeping things consistent and uniform.

- Metadata also makes data more reliable by making sure it's accurate, precise, relevant, and timely. This makes it easier for data analysts to identify the root cause of the problem.

- One of the ways data analysts make sure that the data is consistent and reliable is by metadata repository.

Metadata Repository: A database specifically created to store metadata

- Metadata repositories make it easier and faster to bring together multiple sources for data analysis.

They do this by:

- Describing the state and location of the metadata.

- Describing the structures of the tables inside.

- Describing how the data flows through the repository.

- Keeping track of who accesses the metadata and when.

Metadata is stored in a single, central location, and gives the company the standardized information about all of its data.

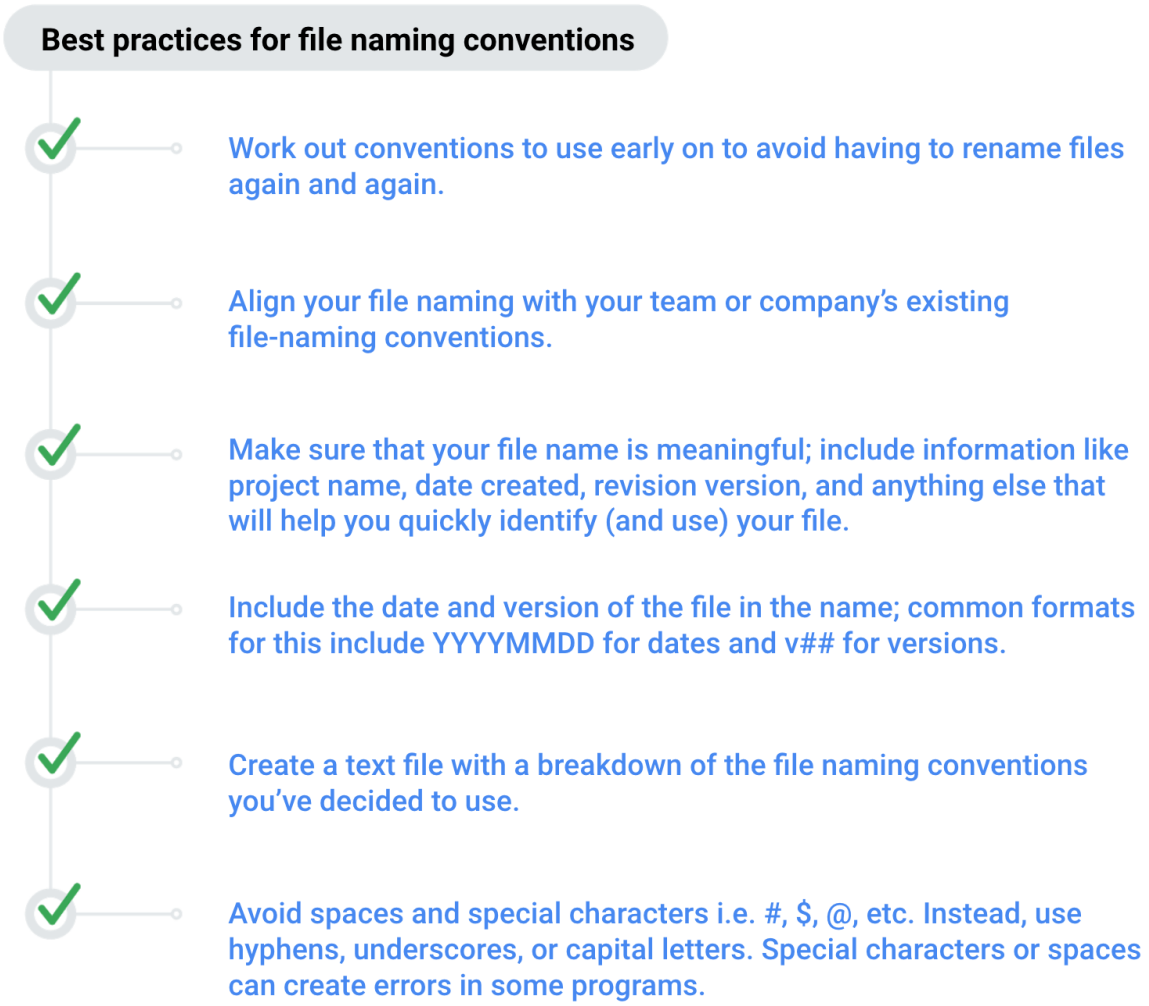
Data Governance: The process to ensure the formal management of a company's data assets.

Week-4

Organizing and protecting your data:

Benefits of organizing data:

- Makes it easier to find and use
- Helps you avoid mistakes during your analysis
- Helps you protect your data.



Data security means protecting data from unauthorized access or corruption by putting safety measures in place. Usually the purpose of data security is to keep unauthorized users from accessing or viewing sensitive data. Data analysts have to find a way to balance data security with their actual analysis needs. This can be tricky-- we want to keep our data safe and secure, but we also want to use it as soon as possible so that we can make meaningful and timely observations.

In order to do this, companies need to find ways to balance their data security measures with their data access needs.

Luckily, there are a few security measures that can help companies do just that. The two we will talk about here are encryption and tokenization.

Encryption uses a unique algorithm to alter data and make it unusable by users and applications that don't know the algorithm. This algorithm is saved as a "key" which can be used to reverse the encryption; so if you have the key, you can still use the data in its original form.

Tokenization replaces the data elements you want to protect with randomly generated data referred to as a “token.” The original data is stored in a separate location and mapped to the tokens. To access the complete original data, the user or application needs to have permission to use the tokenized data and the token mapping. This means that even if the tokenized data is hacked, the original data is still safe and secure in a separate location.

Course 4: **Process Data from Dirty to Clean**

Week-1

Data Integrity: The accuracy, completeness, consistency, and trustworthiness of data throughout its lifecycle.
It is very important to check all the above qualities before proceeding to start analysis, otherwise the analysis could be wrong.

Data integrity can be compromised in many ways, few of which are:

1. **Data Replication:** The process of storing data in multiple locations.
   If the data is replicated different times at different places, it could be out of sync. Due to which different analysts might work with different data which leads to inconsistency, in turn lack of integrity.

2. **Data Transfer:** The process of copying data from a storage device to memory, or from one computer to another. This might lead to data incompleteness which would not be of any use.

3. **Data manipulation:** The process of changing data to make it more organized and easier to read. DM is meant to make the data analysis process more efficient, but an error during the process might compromise efficiency.

Other threats to data integrity:

- Human error
- Viruses
- Malware
- Hacking
- System failures

Data constraints and examples

Data constraint	Definition	Examples
Data type	Values must be of a certain type: date, number, percentage, Boolean, etc.	If the data type is a date, a single number like 30 would fail the constraint and be invalid
Data range	Values must fall between predefined maximum and minimum values	If the data range is 10-20, a value of 30 would fail the constraint and be invalid
Mandatory	Values can't be left blank or empty	If age is mandatory, that value must be filled in
Unique	Values can't have a duplicate	Two people can't have the same phone number within the same service area
Regular expression (regex) patterns	Values must match a prescribed pattern	A phone number must match ###-###-#### (no other characters allowed)
Cross-field validation	Certain conditions for multiple fields must be satisfied	Values are percentages and values from multiple fields must add up to 100%
Primary-key	(Databases only) value must be unique per column	A database table can't have two rows with the same primary key value. A primary key is an identifier in a database that references a column in which each value is unique. More information about primary and

		foreign keys is provided later in the program.
Set-membership	(Databases only) values for a column must come from a set of discrete values	Value for a column must be set to Yes, No, or Not Applicable
Foreign-key	(Databases only) values for a column must be unique values coming from a column in another table	In a U.S. taxpayer database, the State column must be a valid state or territory with the set of acceptable values defined in a separate States table
Accuracy	The degree to which the data conforms to the actual entity being measured or described	If values for zip codes are validated by street location, the accuracy of the data goes up.
Completeness	The degree to which the data contains all desired components or measures	If data for personal profiles required hair and eye color, and both are collected, the data is complete.
Consistency	The degree to which the data is repeatable from different points of entry or collection	If a customer has the same address in the sales and repair databases, the data is consistent.

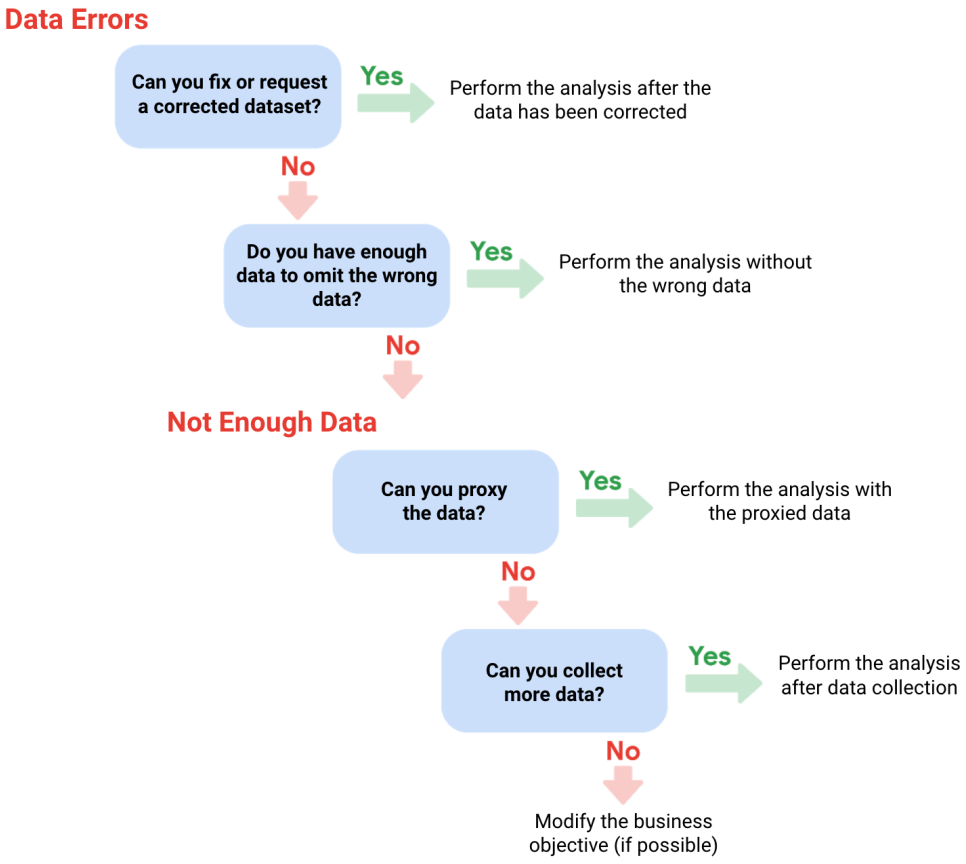
Dealing with insufficient data:

Types of insufficient data:

- Data from only one source
- Data that keeps updating
- Outdated data
- Geographically-limited data

Ways to address insufficient data:

- Identify trends with the data available
- Wait for more data if time allows
- Talk with stakeholders and adjust your objective
- Look for a new dataset



Sample Size: A part of a population that is representative of the whole population

Sampling Bias: A sample isn't representative of the population as a whole

Random Sampling: A way of selecting a sample from a population so that every possible type of the sample has an equal chance of being chosen.

Terms and Definitions:

Terminology	Definitions
Population	The entire group that you are interested in for your study. For example, if you are surveying people in your company, the population would be all the employees in your company.
Sample	A subset of your population. Just like a food sample, it is called a sample because it is only a taste. So if your company is too large to survey every individual, you can survey a representative sample of your population.
Margin of error	Since a sample is used to represent a population, the sample's results are expected to differ from what the result would have been if you had surveyed the entire population. This difference is called the margin of error. The smaller the margin of error, the closer the results of the sample are to what the result would have been if you had surveyed the entire population.
Confidence level	How confident you are in the survey results. For example, a 95% confidence level means that if you were to run the same survey 100 times, you would get similar results 95 of those 100 times. Confidence level is targeted before you start your study because it will affect how big your margin of error is at the end of your study.
Confidence interval	The range of possible values that the population's result would be at the confidence level of the study. This range is the sample result +/- the margin of error.

Statistical significance	The determination of whether your result could be due to random chance or not. The greater the significance, the less due to chance.

Check the reading “Calculating sample size”

Statistical Power: The probability of getting meaningful results from a test.

Hypothesis testing: A way to see if a survey or experiment has meaningful results.

The larger the sample size, the greater the chance that we will have statistically significant results on a test.

If a test is statistically significant, it means the results of the test are real and not an error caused by random chance.

Usually, you need a statistical power of at least 0.8 or 80% to consider your results statistically significant.

Confidence Level: The probability that your sample size accurately reflects the greater population. Having a 99% of confidence level is ideal, but most industries hope for at least 90-95% confidence level.

Sample size calculator: Sample size calculator by raosoft.com

Margin of error: The maximum that the sample results are expected to differ from those of the actual population. More technically, the margin of error defines a range of values below and above the average result for the sample. The average result for the entire population is expected to be within that range.

Margin of error in marketing

The margin of error is also important in marketing. Let’s use A/B testing as an example. A/B testing (or split testing) tests two variations of the same web page to determine which page is more successful in attracting user traffic and generating revenue. User traffic that gets monetized is known as the conversion rate. A/B testing allows marketers to test emails, ads, and landing pages to find the data behind what is working and what isn’t working. Marketers use the confidence interval (made up of the conversion rate and the margin of error) to understand the results.

For example, suppose you are conducting an A/B test to compare the effectiveness of two different email subject lines to entice people to open the email. You find that subject line A:

“Special offer just for you” resulted in a 5% open rate compared to subject line B: “Don’t miss this opportunity” at 3%.

Does that mean subject line A is better than subject line B? It depends on your margin of error. If the margin of error was 2%, then subject line A’s actual open rate is somewhere between 3% and 7%. Since the lower end of the range overlaps with subject line B’s results at 3%, you can’t conclude that there is a statistically significant difference between subject line A and B. Examining the margin of error is important when making conclusions based on your test results.

Week-2

Dirty data: Data that is incomplete, incorrect, and irrelevant to the problem you’re trying to solve. Data can be dirty if it has:

- Incorrect entries
- Duplicate values
- Blank fields
- Incompatible format (inconsistent data)
- Outdated

Clean data: Data that is complete, correct, and relevant to the problem you’re trying to solve.

Note:
Data Engineers: Transform data into a useful format for analysis and give it a reliable infrastructure.
Data warehousing specialists: Develop processes and procedures to effectively store and organize data.

Data Validation: A tool for checking the accuracy and quality of data before adding or importing it.

Clear formatting of different cells in a spreadsheet: **Format -> Clear formatting**

Cleaning data that is gathered from multiple sources has its own challenges. For example, **Merger:** An agreement that unites two organizations into a new one. All the data from these organizations should be combined using **data merging** (the process of combining two or more datasets into a single dataset). The data merging itself comes with a challenge, that is, when two different datasets are combined, they can be inconsistent and misaligned. Eg: they might have different sets of fields (columns) and different formats of data.

The following questions needs to be asked while merging data to avoid redundancy and confirm that the datasets are compatible:

1. Do I have all the data I need?
2. Does the data I need exist within these datasets?

3. Does the data need to be cleaned, or are they ready for me to use?
4. Are the datasets cleaned to the same standard?

Data cleaning features in spreadsheets:

1. **Conditional formatting:** A spreadsheet tool that changes how a cell appears when values meet specific conditions.
   Format -> Conditional formatting

2. **Remove Duplicates:** A tool that automatically searches for and eliminates duplicate entries from a spreadsheet.
   Data -> Remove duplicates

3. **Format consistency:** A tool that converts the incorrect format to a correct one. Eg: Dates. **Data -> Number -> Date**

4. **Text string:** A group of characters within a cell, most often composed of letters.

   Split: A tool that divides a text string around the specified character and puts each fragment into a new and separate cell.
   Specified text separator = delimiter (Eg: ,)
   Example: =SPLIT(B2,"-")

   Data -> Split text to columns (detects the delimiter automatically, of course we can mention a custom delimiter)

   Split text to columns also converts the numbers in text format to number.

5. **Concatenate:** A function that joins multiple text strings into a single string.

6. **Proper:** This function capitalizes the first letter **=PROPER(range)**

Functions used to optimize data cleaning process:

1. **COUNTIF:** A function that returns the number of cells that match a specified value.

   Syntax: =COUNTIF(range, "value") -> =COUNTIF(I2:I72, "<100")

2. **LEN:** A function that tells you the length of the text string by counting the number of characters it contains.

   Syntax: =LEN(range) -> =LEN(A2)

3. **LEFT :** A function that gives you a set number of characters from the left side of a text string.

Syntax: =LEFT(range, number of characters) -> =LEFT(A2, 5)
Example: =LEFT(F2,FIND("-",F2)-1)

RIGHT : A function that gives you a set number of characters from the right side of a text string.
Syntax: =RIGHT(range, number of characters) -> =RIGHT(A2, 4)
Example: =RIGHT(F2,LEN(F2)-FIND("-",F2))

4. **MID:** A function that gives you a segment from the middle of a text string.

   Syntax: =MID(range, reference starting point, number of characters) -> =MID(D2, 4,2)

5. **Concatenate:** A function that joins multiple text strings into a single string.

   Syntax: =CONCATENATE(item1, item2) -> =CONCATENATE(H2,I2)
   =CONCATENATE(G2&H2)
   Example: =CONCATENATE(PROPER(H2)&" "&PROPER(I2)&" "&PROPER(J2)&" "&PROPER(K2)&" "&PROPER(L2)&" "&PROPER(M2)&" "&PROPER(N2)&" "&PROPER(O2)&" "&PROPER(P2))

6. **TRIM:** A function that removes leading, trailing, and repeated spaces in data.

   Syntax: =TRIM(range) -> =TRIM(C2)

Note: Insert -> Pivot Table

VLOOKUP (Vertical LOOKUP): A function that searches for a certain value in a column to return a corresponding piece of information.

Syntax: =VLOOKUP(data to look up, 'where to look'!Range, column, false) -> false here says that we want an exact match
Example: =VLOOKUP(A2, 'Sheet 2'!A2:B31, 2, false)

Note: For visualization of numeric columns, go to **Insert -> Chart**

The above note is for plotting numeric data and checking for any outliers. This is called Plotting.

Data mapping: The process of matching fields from one data source to another. This is very important for the success of data migration, data integration, and lots of other data manipulation activities.

The first step to data mapping is to identify what data needs to be moved (tables or fields in the tables). We also need to define the format of the data once it reaches its destination.

Week-3

Data manipulation Language:

1. **INSERT**

   INSERT INTO customer_data.cusomter_address (customer_id, name, address, city)
   VALUES (“1234”,”anirudh”,”123 Boston”,”Boston”)

2. **UPDATE**

   UPDATE customer_data.customer_address
   SET address = “123 Allston”
   WHERE customer_id=”1234”

3. **CREATE TABLE IF NOT EXISTS**

4. **DROP IF EXISTS**

Cleaning string variables (data) using SQL:

- Including DISTINCT in the SELECT statement removes duplicates.

```
SELECT DISTINCT customer_id
FROM customer_data.customer_address
```

- To maintain the length of the string variables throughout the column, use **LENGTH**

```
SELECT
    country
FROM
    customer_data.customer_address
WHERE
    LENGTH(country) > 2
```

  Extract a substring from a string (start at position 5, extract 3 characters):

```
SELECT
    DISTINCT (customer_id)
```

```
FROM
    customer_data.customer_address
WHERE
    SUBSTR(country,1,2) = "US"
```

- To eliminate the extra spaces, use TRIM

```
SELECT
    DISTINCT customer_id
FROM
    customer_data.customer_address
WHERE
    TRIM(state) = "OH"
```

- To check if a column contains NULL value, use *IS NULL*

```
select * from automobile_data.car_info where num_of_doors IS NULL;
```

To replace NULL value with a certain value

```
select make, fuel_type, num_of_doors, body_style
from automobile_data.car_info
where make="dodge" AND fuel_type="gas" AND body_style="sedan";

update automobile_data.car_info
set num_of_doors="four"
where make="dodge" AND fuel_type="gas" AND body_style="sedan";
```

- To delete a column or row

```
DELETE
FROM
  cars.car_info
WHERE
  compression_ratio = 70;
```

- Mean imputation is a method in which erroneous values in a column are replaced by the mean (or average) of the other values in that column. This method maintains the dataset size, but some important statistics like variance and standard deviation tend to be minimized.

In SQL, the **AVG()** function returns the average value of a numeric column. To find the average price, make the following query containing the AVG() function:

```
SELECT
  AVG(price) AS average_price
FROM
  Cars.car_info;
```

Next, replace the 0 values in the price column with the rounded average value by making the following query:

```
UPDATE
  cars.car_info
SET
  price = 12978
WHERE
  price = 0;
```

- Data consistency means that there is consistency in the measurement of variables throughout the data tables. Discrepancies can create inaccurate, unreliable results. This leads to misinformed business decisions. Data inconsistencies are often overlooked and tricky to spot. One way data inconsistencies can occur within a table is if the values that are meant to be the same either are spelled differently or have different character length due to extra spaces. You can spot these types of inconsistencies by inspecting a column's unique values.

- **CAST()** is used to convert anything from one datatype to another (similar to typecasting, which is used to convert data from one datatype to another)

  SELECT
  CAST(purchase_prices AS FLOAT64)
  FROM
  table_name
  ORDER BY
  CAST(purchase_prices AS FLOAT64) DESC

- **CONCAT()** adds strings together to create a new text strings that can be used as an unique keys

  SELECT CONCAT(product, product_code) AS new_product_code
  FROM *table_name*
  WHERE product="couch"

- **COALESCE()** can be used to return non-null values in the list.

  SELECT COALESCE(arg1, arg2) -> arg1 is the column name in which nulls are present
  arg2 is the column name from which the value is to be replaced.

Week-4

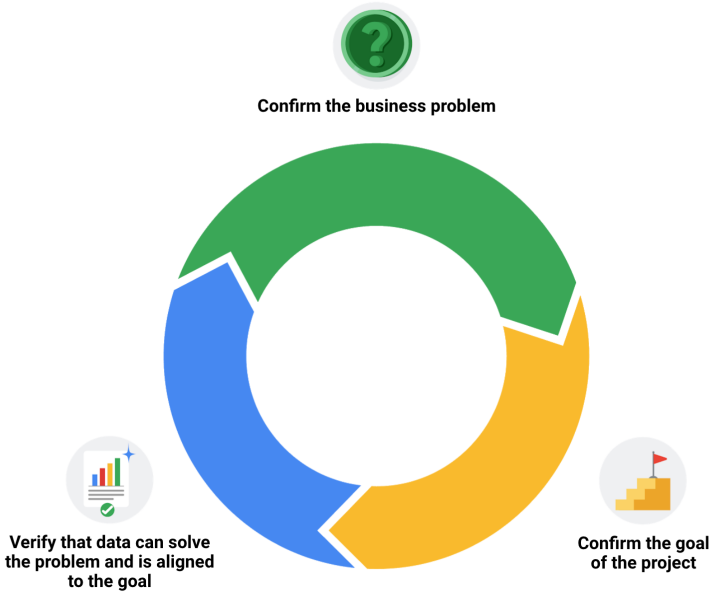
Verification: A process to confirm that a data cleaning effort was well- executed and the resulting data is accurate and reliable.

Changelog: A file containing a chronologically ordered list of modifications made to a project. It's usually organized by version and includes the date followed by a list of added, improved, and removed features.

First step in the verification process is to go back to the unclean data and compare it with the current data to check if the errors were cleaned properly.

See the big-picture when verifying the data:

- Consider the business problem
- Consider the goal
- Consider the data



Correct the most common problems

Make sure you identified the most common problems and corrected them, including:

- Sources of errors: Did you use the right tools and functions to find the source of the errors in your dataset?
- Null data: Did you search for NULLs using conditional formatting and filters?
- Misspelled words: Did you locate all misspellings?
- Mistyped numbers: Did you double-check that your numeric data has been entered correctly?
- Extra spaces and characters: Did you remove any extra spaces or characters using the TRIM function?
- Duplicates: Did you remove duplicates in spreadsheets using the Remove Duplicates function or DISTINCT in SQL?
- Mismatched data types: Did you check that numeric, date, and string data are typecast correctly?
- Messy (inconsistent) strings: Did you make sure that all of your strings are consistent and meaningful?
- Messy (inconsistent) date formats: Did you format the dates consistently throughout your dataset?
- Misleading variable labels (columns): Did you name your columns meaningfully?
- Truncated data: Did you check for truncated or missing data that needs correction?
- Business Logic: Did you check that the data makes sense given your knowledge of the business?

Sometimes, after the above verification process, there still might be an error that persists. Such errors can be either resolved manually or by using the spreadsheet functions (like TRIM(), Removing Duplicates). But in a few cases, using the above techniques might not be helpful and the error might still persist. In such cases we can use Pivot tables (A data summarization tool that is used in data processing).

For example: let's say we have a misspelled name in a spreadsheet. We can manually correct it but what if there are more than one of such errors. One way to solve this is using **Find and Replace**. The other way is to use pivot tables. **COUNTA** is a function that counts the total number of values within a specified range. If a similar error occurs in SQL, we can use **CASE** statement (It goes through one or more conditions and returns a value as soon as a condition is met.)

```
SELECT
    Customer_id,
    CASE
        WHEN first_name = “Tnoy” THEN “Tony”
        ELSE first_name
        END AS cleaned_name
FROM table_name
```

Documenting results and cleaning process:

Documentation: The process of tracking changes, additions, deletions and errors involved in your data cleaning effort.

Documentation:

- Recover data-cleaning errors
- Inform other users of changes
- Determine quality of data

Changelog: A file containing a chronologically ordered list of modifications made to a project. It's usually organized by version and includes the date followed by a list of added, improved, and removed features.

To check changelog in a spreadsheet **File -> Version History -> See version history**
To check a cell’s history **right click on the cell** and **choose Show edit history**

Every SQL software has its own changelog mechanism.

A changelog can build on your automated version history by giving you an even more detailed record of your work. This is where data analysts record all the changes they make to the data. Here is another way of looking at it. Version histories record *what* was done in a data change for a project, but don't tell us *why*. Changelogs are super useful for helping us understand the reasons changes have been made. Changelogs have no set format and you can even make your entries in a blank document. But if you are using a shared changelog, it is best to agree with other data analysts on the format of all your log entries.

Typically, a changelog records this type of information:

- Data, file, formula, query, or any other component that changed
- Description of what changed
- Date of the change
- Person who made the change
- Person who approved the change
- Version number
- Reason for the change

Let's say you made a change to a formula in a spreadsheet because you observed it in another report and you wanted your data to match and be consistent. If you found out later that the report was actually using the wrong formula, an automated version history would help you *undo* the change. But if you also recorded the reason for the change in a changelog, you could go back to the creators of the report and let them know about the incorrect formula. If the change happened a while ago, you might not remember who to follow up with. Fortunately, your changelog would have that information ready for you! By following up, you would ensure data integrity outside your project. You would also be showing personal integrity as someone who can be trusted with data. That is the power of a changelog!

Finally, a changelog is important for when lots of changes to a spreadsheet or query have been made. Imagine an analyst made four changes and the change they want to revert to is change #2. Instead of clicking the undo feature three times to undo change #2 (and losing changes #3 and #4), the analyst can undo just change #2 and keep all the other changes. Now, our example was for just 4 changes, but try to think about how important that changelog would be if there were hundreds of changes to keep track of.

A junior analyst probably only needs to know the above with one exception. If an analyst is making changes to an existing SQL query that is shared across the company, the company most likely uses what is called a version control system. An example might be a query that pulls daily revenue to build a dashboard for senior management.

Here is how a version control system affects a change to a query:

1. A company has official versions of important queries in their version control system.
2. An analyst makes sure the most up-to-date version of the query is the one they will change. This is called syncing
3. The analyst makes a change to the query.

4. The analyst might ask someone to review this change. This is called a code review and can be informally or formally done. An informal review could be as simple as asking a senior analyst to take a look at the change.
5. After a reviewer approves the change, the analyst submits the updated version of the query to a repository in the company's version control system. This is called a code commit. A best practice is to document exactly what the change was and why it was made in a comments area. Going back to our example of a query that pulls daily revenue, a comment might be: *Updated revenue to include revenue coming from the new product, Calypso*.
6. After the change is submitted, everyone else in the company will be able to access and use this new query when they sync to the most up-to-date queries stored in the version control system.
7. If the query has a problem or business needs change, the analyst can *undo* the change to the query using the version control system. The analyst can look at a chronological list of all changes made to the query and who made each change. Then, after finding their own change, the analyst can revert to the previous version.
8. The query is back to what it was before the analyst made the change. And everyone at the company sees this reverted, original query, too.

A changelog for a personal project may take any form desired. However, in a professional setting, particularly when collaborating with others, readability is important. In light of this, you can follow the guiding principles that make the interpretation of the changelog accessible to others:

- Changelogs are for humans, not machines, so write legibly
- Every version should have its own entry
- Group the same types of changes. For example, "Fixed" should be grouped separately from "Added"
- Each change gets its own line
- Versions should be ordered chronologically starting with the latest
- Release date of each version should be noted

Types of changes usually fall into one of the following categories. All the changes for each category should be grouped together.

- Added: new features introduced

- Changed: changes in existing functionality
- Deprecated: features about to be removed
- Removed: features that have been removed
- Fixed: bug fixes
- Security: lowering vulnerabilities

Keeping data clean and in sync with a source

- **IMPORTRANGE():** https://support.google.com/docs/answer/3093340?hl=en

  Sample Usage:
  ```
  IMPORTRANGE("https://docs.google.com/spreadsheets/d/abcd123abcd123", "sheet1!A1:C10")
  IMPORTRANGE(A2,"B2")
  ```

  Syntax:
  ```
  IMPORTRANGE(spreadsheet_url, range_string)
  ```
  - `spreadsheet_url` - The URL of the spreadsheet from where data will be imported.
    - The value for `spreadsheet_url` must either be enclosed in quotation marks or be a reference to a cell containing the URL of a spreadsheet.

  - `range_string` - A string, of the format `"[sheet_name!]range"` (e.g. `"Sheet1!A2:B6"` or `"A2:B6"`) specifying the range to import.
    - The `sheet_name` component of `range_string` is optional; by default `IMPORTRANGE` will import from the given range of the first sheet.
    - The value for `range_string` must either be enclosed in quotation marks or be a reference to a cell containing the appropriate text.

Pulling data from other data sources

- **QUERY():** https://support.google.com/docs/answer/3093343?hl=en

  Sample Usage:
  ```
  QUERY(A2:E6,"select avg(A) pivot B")
  QUERY(A2:E6,F2,FALSE)
  ```

Syntax:

```
QUERY(data, query, [headers])
```

- `data` - The range of cells to perform the query on.
    - Each column of `data` can only hold boolean, numeric (including date/time types) or string values.
    - In case of mixed data types in a single column, the majority data type determines the data type of the column for query purposes. Minority data types are considered null values.

- `query` - The query to perform, written in [the Google Visualization API Query Language]
    - The value for `query` must either be enclosed in quotation marks or be a reference to a cell containing the appropriate text.

- `headers` - [OPTIONAL] - The number of header rows at the top of `data`. If omitted or set to `-1`, the value is guessed based on the content of `data`.

Filtering data to get what you want

- **FILTER():** https://support.google.com/docs/answer/3093197?hl=en

Sample Usage:

```
FILTER(A2:B26, A2:A26 > 5, D2:D26 < 10)
FILTER(A2:C5, {TRUE; TRUE; FALSE; TRUE})
FILTER(A2:B10, NOT(ISBLANK(A2:A10)))
```

Syntax:

```
FILTER(range, condition1, [condition2, ...])
```

- `range` - The data to be filtered.
- `condition1` - A column or row containing true or false values corresponding to the first column or row of `range`, or an array formula evaluating to true or false.
- `condition2 ...` - [OPTIONAL] - Additional rows or columns containing boolean values `TRUE` or `FALSE` indicating whether the corresponding row or column in `range` should pass through `FILTER`. Can also contain array formula expressions which evaluate to such rows or columns. All conditions must be of the same type (row or column). Mixing row conditions and column conditions is not permitted.

- `condition` arguments must have exactly the same length as `range`.

Notes:

- `FILTER` can only be used to filter rows or columns at one time. In order to filter both rows and columns, use the return value of one `FILTER` function as `range` in another.
- If `FILTER` finds no values which satisfy the provided conditions, `#N/A` will be returned.

Week-5

Transferable skills: The skills used in the previous organization that can be transferred to a new organization while switching career paths.

Some of the transferable skills are for data analysts:

- Communication
- Problem-solving
- Teamwork

Problem
Action
Result

Course 5: **Analyze data to answer questions**

Week-1

Analysis: The process used to make sense of the data collected. The goal of analysis is to identify trends and relationships within data so you can accurately answer the question you're asking.

The four phases of analysis:

1. *Organize data*
2. *Format and adjust data*
3. *Get input from others:* When analyzing data, gaining input from others is important because it gives you a viewpoint you might not understand or have access to. On top of gaining input from other people, it's also important to seek out others' perspectives early. That way, if they predict any obstacles or challenges, you'll know beforehand.
4. *Transform data:* Identifying relationships and patterns between the data, and making calculations based on the data you have.

Keeping data organized with sorting and filters

The organization of datasets is really important for data analysts. Most of the datasets you will use will be organized as tables. Tables are helpful because they let you manipulate your data and categorize it. Having distinct categories and classifications lets you focus on, and differentiate between, your data quickly and easily.

Data analysts often need to format and adjust data when performing an analysis. Sorting and filtering are two ways you can keep things organized when you adjust the data to work with it. For example, a filter can help you find errors or outliers so you can fix or flag them before your analysis. Outliers are data points that are very different from similarly collected data and might not be reliable values. The benefit of filtering the data is that after you fix errors and identify outliers, you can remove the filter and return the data to its original form.

Sorting vs Filtering

Sorting is when you arrange data into a meaningful order to make it easier to understand, analyze, and visualize. It ranks your data based on a specific metric you choose. You can sort data in spreadsheets, SQL databases (when your dataset is too large for spreadsheets), and tables in documents.

Filtering is used when you are only interested in seeing data that meets a specific criteria, and hiding the rest. Filtering is really useful when you have lots of data. You can save time by zeroing in on the data that is really important or the data that has bugs or errors. Most spreadsheets and SQL databases allow you to filter your data in a variety of ways. Filtering gives you the ability to find what you are looking for without too much effort.

Sorting in a pivot table

Items in the row and column areas of a pivot table are sorted in ascending order by any custom list first. For example, if your list contains days of the week, the pivot table allows weekday and month names to sort like this: Monday, Tuesday, Wednesday, etc. rather than alphabetically like this: Friday, Monday, Saturday, etc.

If the items aren’t in a custom list, they will be sorted in ascending order by default. But, if you sort in descending order, you are setting up a rule that controls how the field is sorted even after new data fields are added.

Note:

- A data analyst uses database organization to decide which data is relevant to their analysis and which data types and variables are appropriate.

Sort Sheet: All of the data in a spreadsheet is sorted by the conditions of a single column, but the related information across each row stays together.

Sort Range: Nothing else on the spreadsheet gets rearranged but the specified cells.

SORT(): Syntax: SORT(range, column_number, **TRUE** for ascending or **FALSE** for descending)
SORT(A2:D6, 2, TRUE)

Customized sort order: When you sort data in a spreadsheet using multiple conditions.

Note: The difference between a menu sort function and a written SORT function is that a menu sort function rearranges the data, while the written SORT function changes it.

Sorting and Filtering in SQL

Sorting by WHERE clause
Filtering by ORDER BY clause (default is ascending, for descending mention **DESC**)

Week-2

Converting data from one type to another (Spreadsheets):

- Converting data might mean converting numbers into dates, strings, percentages, or currency.
- It also might mean changing the units of measurement (Fahrenheir to Celcius)
- Incorrectly formatted data can:
  - Lead to mistakes
  - Take time to fix
  - Affect stakeholder's decision-making
- To convert the numbers columns into some other data type from the spreadsheet **menu**, go to **Format -> Number -> (Choose the required format or type)**
- To convert Fahrenheit to Celcius, use **COVERT()**
  - **Syntax:** =CONVERT(cell_no, "F", "C")

Data Validation: Allows you to control what can or what can't be entered in your worksheet

Data Validation can:

- Add dropdown lists with predetermined options.
    - **Data -> Data Validation -> Criteria: List of items**
- Create custom checkboxes.
    - **Data -> Data Validation -> Criteria: Checkbox**
- Protect structured data and formulas.
    - **Data -> Data Validation -> On invalid data: Reject input**

Conditional Formatting: A spreadsheet tool that changes how cells appear when values meet specific conditions. Conditional Formatting can be used alongside with data validation. In the above dropdown case (of data validation) we can add specific color to each value in the dropdown.

Combining multiple datasets

Merging and multiple sources:

- Just like **CONCATENATE()** in spreadsheets combines two or more text strings together to form a single text string, SQL has **CONCAT()** which combines data from multiple columns.
    - Example:

```
select
    usertype,
    CONCAT(start_station_name, " to ", end_station_name) AS route,
    COUNT(*) AS num_trips,
    ROUND(AVG(CAST(tripduration AS int64)/60),2) AS duration
from
    bigquery-public-data.new_york.citibike_trips
group by
    start_station_name,
    end_station_name,
    usertype
order by
    num_trips DESC
LIMIT 10
```

Function	Usage	Example
CONCAT	A function that adds strings together to create new text strings that can be used as unique keys	CONCAT ('Google', '.com');
CONCAT_WS	A function that adds two or more strings together with a separator	CONCAT_WS (' . ', 'www', 'google', 'com') *The separator (being the period) gets input before and after Google when you run the SQL function
CONCAT with +	Adds two or more strings together using the + operator	'Google' + '.com'

Get support during analysis

Best practices for searching online:

- Thinking skills (Mental Model): Your thought process and the way you approach a problem.
- Data analytics terms
- Basic knowledge of tools

R: A programming language used for statistical analysis, data visualization, and other data analysis.

Week-3

VLOOKUP for data aggregation:

Aggregation: Collecting or gathering many separate pieces into a whole.
Data Aggregation: The process of gathering multiple sources in order to combine it into a single summarized collection.

- Data Aggregation helps data analysts to:
    - Identify trends
    - Gain insights
    - Make comparisons
- Data can also be aggregated over a given time period to provide the statistics such as:
    - Averages
    - Minimums
    - Maximums
    - Sums
- Functions are a big help in making data aggregation possible.
- **Subquery:** A query within a query

Preparing for VLOOKUP:

- **VLOOKUP(Vertical LOOKUP):** A function that searches for a certain value in a column to return a corresponding piece of information.
  Syntax: =VLOOKUP(search_key, range, index, [is_sorted])
- **VALUE:** A value is a function that converts a text string that represents a number to a numerical value. Syntax: **VALUE(range)**

Identifying common VLOOKUP errors:

- Not every data analyst works without facing any error or without requiring help from others. Identifying and solving such errors is called **troubleshooting.**
- **Troubleshooting questions:**
    - How should I prioritize these issues?
    - In a single sentence, what's the issue I'm facing?
    - What resources can help me solve the problem?
    - How can I stop this issue from happening in the future?
- VLOOKUP only returns the first match it finds.
- VLOOKUP can only look for data on the right (it can't look left).
- Let's say the first few rows of a VLOOKUP have returned the correct result. But when you drive the function down the column, problems start popping up. This is probably because the table array part of the function hasn't been locked or made absolute. An **absolute reference** is a reference that is locked so that rows and columns won't change when copied. This issue can be fixed by wrapping the table array in dollar signs. The dollar sign controls how the reference will be updated. They make sure that the corresponding part of the reference doesn't change.
- Something else that can throw off your VLOOKUP results are version control issues. In other words, a function worked perfectly at first, but then something in the spreadsheet it was referencing changed. For example, maybe a user inserted a column. So now the columns in your function no longer direct VLOOKUP to the right place. When something like this happens, it'll return an incorrect value. There are a few actions data analysts can take to ensure this doesn't happen. First, lock the spreadsheet. This stops other people

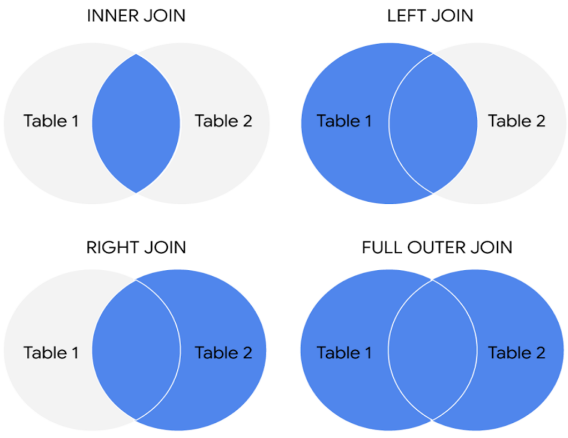
from making changes. To do this in Sheets, select Data, then Protected sheets and ranges. Next, choose what you want to protect. In this case, we want to protect the entire sheet. Then you can set permissions to either show a warning or restrict who can edit. Choose only you, then Done.
    - There will be times when other people need to work in the spreadsheet, so locking them out might make you pretty unpopular with your coworkers. When that's the case, you can use **MATCH**, which is a function used to locate the position of a specific lookup value and can help you with version control.
- **What if you get #N/A?** #N/A indicates that a matching value can't be returned as a result of the VLOOKUP. The error doesn't mean that anything is actually wrong with the data, but people might have questions if they see the error in a report. You can use the IFNA function to replace the #N/A error with something more descriptive, like "Does not exist."
  Syntax: IFNA(value, value_if_na) -> IFNA(#N/A, "Does not exist")
- **When do you need to use VLOOKUP?**
    - Populating data in a spreadsheet
    - Merging data from one spreadsheet with data in another

Use JOIN to aggregate data in SQL

JOIN: A SQL clause that is used to combine rows from two or more tables based on a related column.

Common JOINs:

- **Inner:** A function that returns records with matching values in both tables
- **Left:** A function that return all the records from the left table and only the matching records from the right table.
- **Right:** A function that returns all the records from the right table and only the matching records from the left table.
- **Outer:** A function that combines both the RIGHT and LEFT JOIN to return all the matching records in both tables.



Note:

The importance of Aliases

Aliases are used in SQL queries to create temporary names for a column or table. Aliases make referencing tables and columns in your SQL queries much simpler when you have table or column names that are too long or complex to make use of in queries. Imagine a table name like special_projects_customer_negotiation_mileages. That would be difficult to retype every time you use that table. With an alias, you can create a meaningful nickname that you can use for your analysis. In this case “special_projects_customer_negotiation_mileages” can be aliased to simply “mileage.” Instead of having to write out the long table name, you can use a meaningful nickname that you decide.

Basic syntax for aliasing

Aliasing is the process of using aliases. In SQL queries, aliases are implemented by making use of the AS command. The basic syntax for the AS command can be seen in the following query for aliasing a table:

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

```
SELECT column_name AS alias_name
FROM table_name;
```

Alternate syntax fro aliasing

If using AS results in an error when running a query because the SQL database you are working with doesn't support it, you can leave it out. In the previous examples, the alternate syntax for aliasing a table or column would be:

- FROM table_name alias_name
- SELECT column_name alias_name

The key takeaway is that queries can run with or without using AS for aliasing, but using AS has the benefit of making queries more readable. It helps to make aliases stand out more clearly.

COUNT in spreadsheets: Can be used to count the total number of numeric values within a specified range in spreadsheets.

COUNT in SQL: A query that returns the number of rows in a specified range.

COUNT DISTINCT: A query that only returns the distinct number of rows in a specified range.

Work with Subqueries:

Subquery: A SQL query that is listed inside a larger query.

Using subqueries to aggregate data:

HAVING: Allows you to add a filter to your query instead of the underlying table when you're working with aggregate functions.

CASE: Returns records with your conditions by allowing you to include if/then statements in your query.

There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses
- A subquery can have only one column specified in the SELECT clause. But if you want a subquery to compare multiple columns, those columns must be selected in the main query.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator which allows you to specify multiple values in a WHERE clause.
- A subquery can't be nested in a SET command. The SET command is used with UPDATE to specify which columns (and values) are to be updated in a table.

Example:

```
select
    warehouse.warehouse_id,
    CONCAT(warehouse.state, ": ", warehouse.warehouse_alias) as warehouse_name,
    COUNT(orders.order_id) as number_of_orders,
    (
        select COUNT(*)
        from warehouse_orders.orders as ordesr
    ) as total_number_of_orders,
    CASE
        when COUNT(orders.order_id)/(select COUNT(*) from warehouse_orders.orders as
orders) <=0.2
        then "fulfilled 0-20% of orders"
        when COUNT(orders.order_id)/(select COUNT(*) from warehouse_orders.orders as
orders) > 0.2
```

```
        and COUNT(orders.order_id)/(select COUNT(*) from warehouse_orders.orders as
orders) < 0.6
        then "fulfilled 20-60% of orders"
    else "fulfilled more than 60% of orders"
    end as fulfillment_summary

from warehouse_orders.warehouse as warehouse
left join
warehouse_orders.orders as orders on warehouse.warehouse_id = orders.warehouse_id
group by
    warehouse.warehouse_id,
    warehouse_name
having
    COUNT(orders.order_id)>0
```

Data Calculations:

- **SUM()**
- **AVERAGE()**

Conditional functions are functions that perform a specific task on an array, but only on those cells in the array that satisfy some defined criteria. They are usually identified with an IF suffix adjoined to the desired operation. They are frequently used when constructing more complex queries that cannot be accomplished using more basic functions.

- **COUNTIF():** Only counts the rows whose condition is met.
    - SYntax: =COUNTIF(range, condition)
      Example: =COUNTIF(C3;C15, “=1”)

- **SUMIF():** Only sums the values whose conditions are met.
    - Syntax: =SUMIF(range, condition, [sum_range])
      Example: =SUMIF(C3:C15,”>1”,D3:D15)

- **SUMIFS():** Same as SUMIF() but multiple conditions can be added
    - Syntax: =SUMIFS(sum_range, criteria_range1, criterion1, [criteria_range2, criterion2, ...])
      Example:

      Here, B1:B9 is sum_range, A1:A9 is first criterion_range, “Fuel” is first criteria, C1:C9 is the second criterion_range, “12/15/2020” is the second criteria.

You can add up to 127 conditions to a SUMIFS statement!

- **COUNTIFS():** Same as COUNTIF() but multiple conditions can be added.
    - Syntax: =COUNTIFS(criteria_range1, criterion1, [criteria_range2, criterion2, ...])
      Example:

      Here, A1:A9 is first criterion_range, “Coffee” is first criteria,
      C1:C9 is the second criterion_range, “12/15/2020” is the second criteria.

SUMPRODUCT(): A function that multiplies arrays and returns the sum of their products.

Syntax: =SUMPRODUCT(array1, [array 2]...)
Array: Collection of values in cells.

Example: =SUMPRODUCT(B:B, C:C)

Watch video on “Working with pivot tables” under Pivot...Pivot….Pivot

Pivot Tables: Pivot tables are a spreadsheet tool that lets you view data in multiple ways to find insights and trends.

Pivot tables allow you to make sense of large data sets by giving you tools to easily compare metrics, quickly perform calculations, and generate readable reports. You can create a pivot table to help you answer specific questions about your data. For example, if you were analyzing sales data, you could use pivot tables to answer questions like, “Which month had the most sales?” and “What products generated the most revenue this year?” When you need answers to questions about your data, pivot tables can help you cut through the clutter and focus on only the data you need.

You can perform a wide range of analysis tasks with your pivot tables to quickly draw meaningful insights from your data, including performing calculations, sorting, and filtering your data. Below is a list of online resources that will help you learn about performing basic calculations in pivot tables as well as resources for learning about sorting and filtering data in your pivot tables.

Performance Calculations

Microsoft Excel	Google Sheets
Calculate values in a pivot table: Microsoft Support's introduction to calculations in Excel pivot tables. This is a useful starting point if you are learning how to perform calculations with pivot tables specifically in Excel.	Create and use pivot tables: This guide is focused on using pivot tables in Google Sheets and it provides instructions for creating calculated fields. This is a quick how-to guide you can save and reference as a quick reminder on how to add calculated fields.
Pivot table calculated field example: This resource includes a detailed example of a pivot table being used for calculations. This step-by-step process demonstrates how calculated fields work, and provides you with some idea of how they can be used for analysis.	All about calculated field in pivot tables: This is a comprehensive guide to calculated fields for Google Sheets. If you are working with Sheets and are interested in learning more about pivot tables, this is a great resource.
Pivot table calculated fields: step-by-step tutorial: This tutorial for creating your own calculated fields in pivot tables is a really useful resource to save and bookmark for when you start to apply calculated fields to your own spreadsheets.	Pivot tables in Google Sheets: This beginner's guide covers the basics of pivot tables and calculated fields in Google Sheets and uses examples and how-to videos to help demonstrate these concepts.

Sort your data:

Microsoft Excel	Google Sheets
Sort data in a pivot table or PivotChart: This is a Microsoft Support how-to guide to sorting data in pivot tables. This is a useful reference if you are working with Excel and are interested in checking out how filtering will appear in Excel specifically.	Customize a pivot table: This guide from Google Support focuses on sorting pivot tables in Google Sheets. This is a useful, quick reference if you are working on sorting data in Sheets and need a step-by-step guide.
Pivot tables- Sorting data: This tutorial for sorting data in pivot tables includes an example with real data that demonstrates how sorting in Excel pivot tables works. This example is a great way to experience the entire process from start to finish.	How to sort pivot table columns: This detailed guide uses real data to demonstrate how the sorting process for Google Sheet pivot tables will work. This is a great resource if you need a slightly more detailed guide with screenshots of the actual Sheets environment.
How to sort a pivot table by value: This source uses an example to explain sorting by value in pivot tables. It includes a video, which is a useful guide if you need a demonstration of the process.	Pivot table ascending and descending order: This 1-minute beginner's guide is a great way to brush up on sorting in pivot tables if you are interested in a quick refresher.

Filter your data:

Microsoft Excel	Google Sheets
Filter data in a pivot table: This resource from the Microsoft Support page provides an explanation of filtering data in pivot tables in Excel. If you are working in Excel spreadsheets, this is a great resource to have bookmarked for quick reference.	Customize a pivot table: This is the Google Support page on filtering pivot table data. This is a useful resource if you are working with pivot tables in Google Sheets and need a quick resource to review the process.
How to filter Excel pivot table data: This how-to guide for filtering data in pivot tables demonstrates the filtering process in an Excel spreadsheet with data and includes tips and reminders for when you start using these tools on your own.	Filter multiple values in pivot table: This guide includes details about how to filter for multiple values in Google Sheet pivot tables. This resource expands some of the functionality that you have already learned and sets you up to create more complex filters in Google Sheets.

Format your data:

Microsoft Excel	Google Sheets
Design the layout and format of a PivotTable: This Microsoft Support article describes how to change the format of the PivotTable by applying a predefined style, banded rows, and conditional formatting.	Create and edit pivot tables: This Help Center article provides information about how to edit a pivot table to change its style, and group data.

Pivot tables are a powerful tool that you can use to quickly perform calculations and gain meaningful insights into your data directly from the spreadsheet file you are working in! By using pivot table tools to calculate, sort, and filter your data, you can immediately make high-level observations about your data that you can share with stakeholders in reports.

Calculations in SQL:

Note: 'not equals to' is denoted by "!=" or "<>" in SQL

GROUP BY: A command that groups rows that have the same values from a table into summary rows. It should be written at the end of the query.

EXTRACT(): Lets us pull one part of a date to use.

The Data validation process

Data Validation Process: Checking and rechecking the quality of your data so that it is complete, accurate, secure, and consistent.

Types of validation: Check its section under "The data validation process"

Practice Qwiklabs under "The data validation process" a lot

Using SQL with temporary tables

Temporary table: A database table that is created and exists temporarily on a database server. As data calculations become more complicated, there are many components to keep track of. This is similar to keeping track of tasks in daily life. Some people use sticky notes while others use checklists. In data science, a temporary table is just like a sticky note.
Temporary tables, or temp tables, store subsets of data from standard data tables for a certain period of time. When you end your SQL database session, they are automatically deleted. Temp tables allow you to run calculations in temporary data tables without needing to make modifications to the primary tables in your database.

- They are automatically deleted from the database when you end your SQL session.
- They can be used as a holding area for storing values if you are making a series of calculations. This is sometimes referred to as **pre-processing** of the data.
- They can collect the results of multiple, separate queries. This is sometimes referred to as data **staging**. Staging is useful if you need to perform a query on the collected data or merge the collected data.
- They can store a filtered subset of the database. You don't need to select and filter the data each time you work with it. In addition, using fewer SQL commands helps to keep your data clean.

The **WITH** clause is a type of temporary table that you can query from multiple times.

Example: **WITH** temp_table_name (

SELECT * **from** table_name
)

Queries for the above temporary table can be written right below it.

Select COUNT(*) **from** temp_table_name **where** condition

There are also other ways to create a temp table. Instead of using the WITH clause, you can use the **SELECT INTO** or the **CREATE TABLE** clauses.

The **SELECT INTO** clause copies data from one table into a new table, but doesn't add the new table to the database. It's useful if you want to make a copy of a table with a specific condition.

```
SELECT
	*
INTO
	new_table_name
FROM
	original_table_name
WHERE
	Condition...
```

The **CREATE TABLE** clause is a good option when several people need to access the same temp table. This statement adds the table into the database.

```
CREATE TABLE AS (
SELECT *
FROM
	original_table_name
WHERE
	Condition...
)
```

Best practices when working with temporary tables:

- **Global vs. local temporary tables:** Global temporary tables are made available to all database users and are deleted when all connections that use them have closed. Local temporary tables are made available only to the user whose query or connection established the temporary table. You will most likely be working with local temporary

tables. If you have created a local temporary table and are the only person using it, you can drop the temporary table after you are done using it.

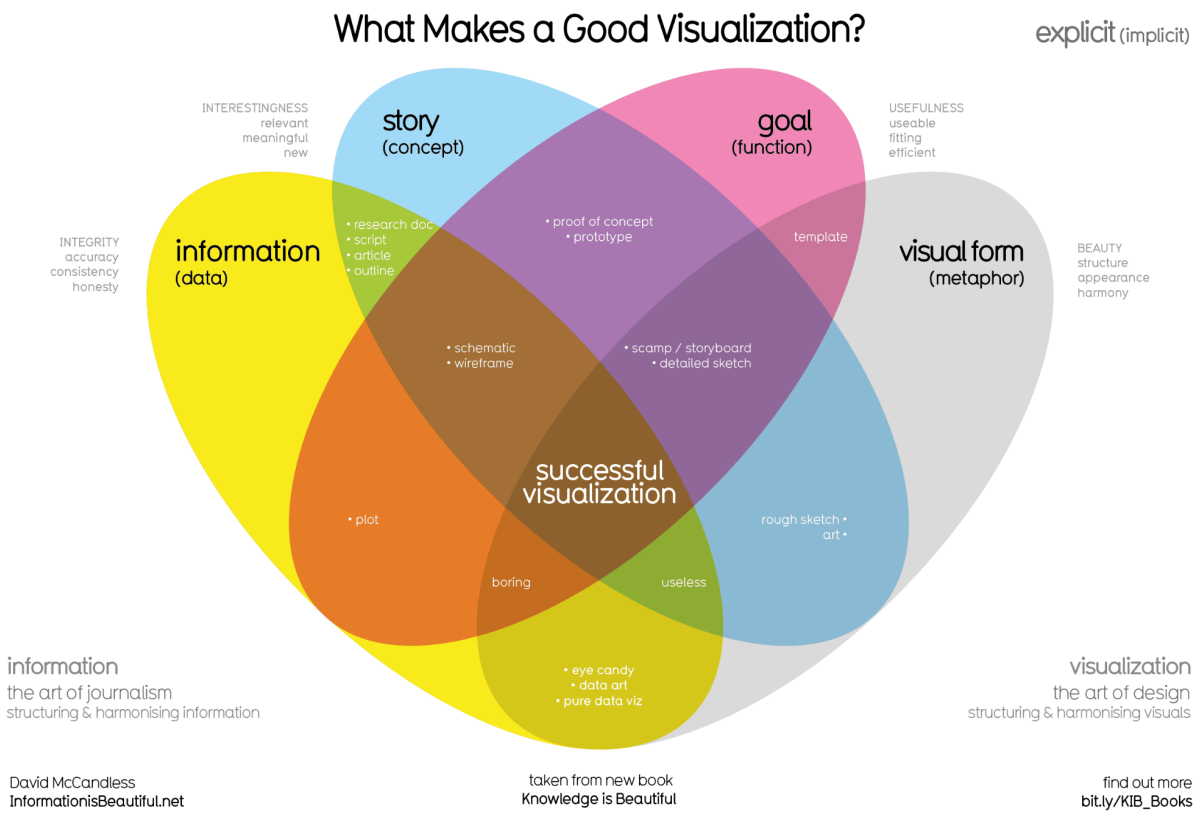
- **Dropping temporary tables after use:** Dropping a temporary table is a little different from deleting a temporary table. Dropping a temporary table not only removes the information contained in the rows of the table, but removes the table variable definitions (columns) themselves. Deleting a temporary table removes the rows of the table but leaves the table definition and columns ready to be used again. Although local temporary tables are dropped after you end your SQL session, it may not happen immediately. If a lot of processing is happening in the database, dropping your temporary tables after using them is a good practice to keep the database running smoothly.

Course 6: **Share Data Through the Art of Visualization**

Week-1

Data Visualization: The graphical representation and presentation of data.

According to David McCandless, the four key elements for a successful visualization are **the data, the story, the goal, and the visual form.**



A data visualization, sometimes referred to as a “data viz,” allows analysts to properly interpret data. A good way to think of data visualization is that it can be the difference between utter confusion and really grasping an issue. Creating effective data visualizations is a complex task; there is a lot of advice out there, and it can be difficult to grasp it all.

Pre-attentive attributes; Marks and Channels

Creating effective visuals means leveraging what we know about how the brain works, and then using specific visual elements to communicate the information effectively. Pre-attentive attributes are the elements of a data visualization that people recognize automatically without conscious effort. The essential, basic building blocks that make visuals immediately understandable are called marks and channels.

Marks

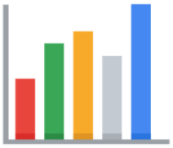
Marks are basic visual objects like points, lines, and shapes. Every mark can be broken down into four qualities:

1. **Position** - Where a specific mark is in space in relation to a scale or to other marks

2. **Size** - How big, small, long, or tall a mark is

3. **Shape** - Whether a specific object is given a shape that communicates something about it

4. **Color** - What color the mark is



Channels

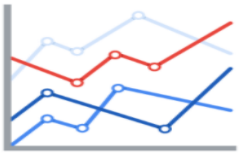
Channels are visual aspects or variables that represent characteristics of the data. Channels are basically marks that have been used to visualize data. Channels will vary in terms of how effective they are at communicating data based on three elements:

1. **Accuracy** - Are the channels helpful in accurately estimating the values being represented?

For example, color is very accurate when communicating categorical differences, like apples and oranges. But it is much less effective when distinguishing quantitative data like 5 from 5.5.

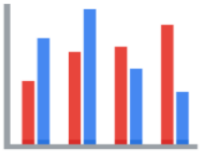
2. **Popout** - How easy is it to distinguish certain values from others?

There are many ways of drawing attention to specific parts of a visual, and many of them leverage pre-attentive attributes like line length, size, line width, shape, enclosure, hue, and intensity.



3. **Grouping** - How good is a channel at communicating groups that exist in the data?

Consider the proximity, similarity, enclosure, connectedness, and continuity of the channel.



But, remember: the more you emphasize different things, the less that emphasis counts. The more you emphasize one single thing, the more that counts.

Design Principles

Once you understand the pre-attentive attributes of data visualization, you can go on to design principles for creating effective visuals. These design principles are important to your work as a data analyst because they help you make sure that you are creating visualizations that communicate your data effectively to your audience. By keeping these rules in mind, you can plan and evaluate your data visualizations to decide if they are working for you and your goals. And, if they aren't, you can adjust them!

Principle	Description
Choose the right visual	One of the first things you have to decide is which visual will be the most effective for your audience. Sometimes, a simple table is the best visualization. Other times, you need a more complex visualization to illustrate your point.
Optimize the data-ink ratio	The data-ink entails focusing on the part of the visual that is essential to understanding the point of the chart. Try to minimize non-data ink like boxes around legends or shadows to optimize the data-ink ratio.
Use orientation effectively	Make sure the written components of the visual, like the labels on a bar chart, are easy to read. You can change the orientation of your visual to make it easier to read and understand.
Color	There are a lot of important considerations when thinking about using color in your visuals. These include using color consciously and meaningfully, staying consistent throughout your visuals, being considerate of what colors mean to different people, and using inclusive color scales that make sense for everyone viewing them.

Numbers of things	Think about how many elements you include in any visual. If your visualization uses lines, try to plot five or fewer. If that isn't possible, use color or hue to emphasize important lines. Also, when using visuals like pie charts, try to keep the number of segments to less than seven since too many elements can be distracting.

Avoiding misleading or deceptive charts

As you are considering what kind of visualization to create and how to design it, you will want to be sure that you are not creating misleading or deceptive charts. As you have been learning, data analysis provides people with insights and knowledge they can use to make decisions. So, it is important that the visualizations you create are communicating your data accurately and truthfully. Here are some common errors to avoid so that your visualizations aren't accidentally misleading:

What to avoid	Why
Cutting off the y-axis	Changing the scale on the y-axis can make the differences between different groups in your data seem more dramatic, even if the difference is actually quite small.
Misleading use of a dual y-axis	Using a dual y-axis without clearly labeling it in your data visualization can create extremely misleading charts.
Artificially limiting the scope of the data	If you only consider the part of the data that confirms your analysis, your visualizations will be misleading because they don't take all of the data into account.
Problematic choices in how data is binned or grouped	It is important to make sure that the way you are grouping data isn't misleading or misrepresenting your data and disguising important trends and insights.

Using part-to-whole visuals when the totals do not sum up appropriately	If you are using a part-to-whole visual like a pie chart to explain your data, the individual parts should add up to equal 100%. If they don't, your data visualization will be misleading.
Hiding trends in cumulative charts	Creating a cumulative chart can disguise more insightful trends by making the scale of the visualization too large to track any changes over time.
Artificially smoothing trends	Adding smooth trend lines between points in a scatter plot can make it easier to read that plot, but replacing the points with just the line can actually make it appear that the point is more connected over time than it actually was.

Examples of data visualizations:

- **Bar graphs:** They use size contrast to compare two or more values.
- **Line graphs:** They help your audience understand shifts or changes in your data.
- **Pie charts:** They show how much each part of something makes up the whole.
- **Maps:** They help organize data geographically.

Engage your audience

An important component of being a data analyst is the ability to communicate your findings in a way that will appeal to your audience. Data visualization has the ability to make complex (and even monotonous) information easily understood, and knowing how to utilize data visualization is a valuable skill to have. Your goal is always to help the audience have a conversation with the data so your visuals draw them into the conversation. This is especially true when you have to help your audience engage with a large amount of data, such as the flow of goods from one country to other parts of the world.

One of your biggest considerations when creating a data visualization is where you'd like your audience to focus.

- **Histogram:** A chart that shows how often data values fall into certain ranges.
- **Correlation charts:** Show relationships among data.
- **Causation:** Occurs when an action directly leads to an outcome.

Note: When two variables look like they are associated in some way, we might assume that one is dependent on the other. That implies causation, even if the variables are completely independent.

Correlation: In statistics is the measure of the degree to which two variables move in relationship to each other. An example of correlation is the idea that "As the temperature goes up, ice cream sales also go up." It is important to remember that correlation doesn't mean that one event causes another. But, it does indicate that they affect each other positively or negatively. If one variable goes up and the other variable also goes up, it is a positive correlation. If one variable goes up and the other variable goes down, it is a negative correlation.

Causation refers to the idea that an event leads to a specific outcome. For example, when lightning strikes, we hear the thunder (sound wave) caused by the air heating and cooling from the lightning strike. Lightning causes thunder.

Why is differentiating between correlation and causation important?

When you make conclusions from data analysis, you need to make sure that you don't assume a causal relationship between elements of your data when there is only a correlation. When your data shows that outdoor temperature and ice cream consumption both go up at the same time, it might be tempting to conclude that hot weather causes people to eat ice cream. But, a closer examination of the data would reveal that every change in temperature doesn't lead to a change in ice cream purchases. In addition, there might have been a sale on ice cream at the same time that the data was collected, which might not have been considered in your analysis.

Knowing the difference between correlation and causation is important when you make conclusions from your data since the stakes could be high. In the next two examples, we will illustrate high stakes to health and human services.

Cause of disease

For example, pellagra is a disease with symptoms of dizziness, sores, vomiting, and diarrhea. In the early 1900s, people thought that the disease was caused by unsanitary living conditions. Most people who got pellagra also lived in unsanitary environments. But, a closer examination of the data showed that pellagra was the result of a lack of niacin (Vitamin B3). Unsanitary conditions were related to pellagra because most people who couldn't afford to purchase niacin-rich foods also couldn't afford to live in more sanitary conditions. But, dirty living conditions turned out to be a correlation only.

Distribution of aid

Here is another example. Suppose you are working for a government agency that provides food stamps. You noticed from the agency's Google Analytics that people who qualify for food stamps are browsing the official website, but they are leaving the site without signing up for

benefits. You think that the people visiting the site are leaving because they aren't finding the information they need to sign up for food stamps. Google Analytics can help you find clues (correlations), like the same people coming back many times or how quickly people leave the page. One of those correlations might lead you to the actual cause, but you will need to collect additional data, like in a survey, to know exactly why people coming to the site aren't signing up for food stamps. Only then can you figure out how to increase the sign-up rate.

Static vs Dynamic Visualizations:

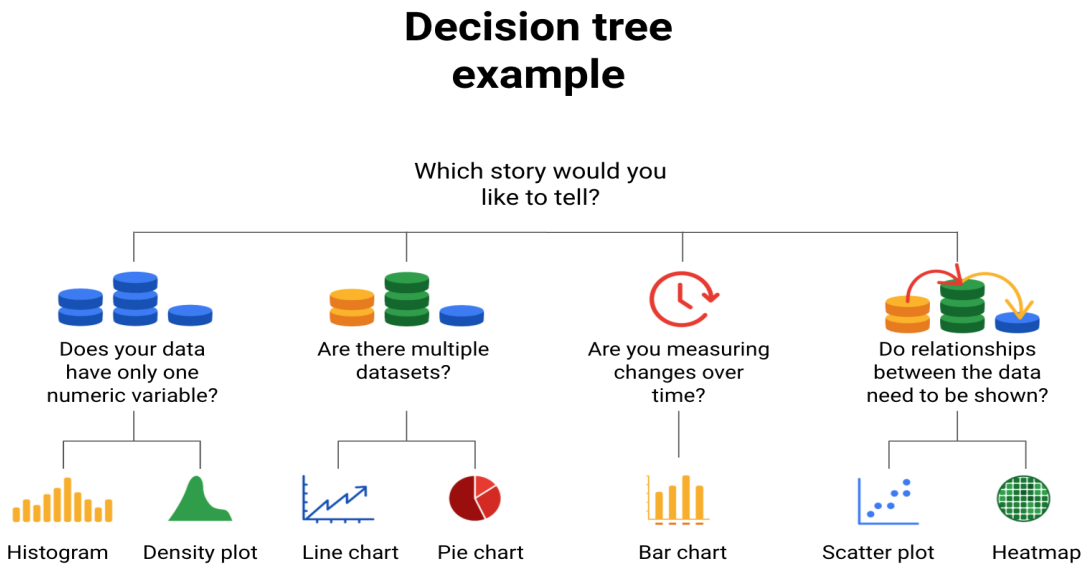
Static: Do not change over time unless they're edited.
Dynamic: Visualizations that are interactive and change over time.

Tableau: A business intelligence and analytics platform that helps people see, understand, and make decisions with data.

Check "The wonderful world of visualization" reading for different types of visualization tools

Data grows on decision trees

A decision tree is a decision-making tool that allows you, the data analyst, to make decisions based on key questions that you can ask yourself. Each question in the visualization decision tree will help you make a decision about critical features for your visualization. Below is an example of a basic decision tree to guide you towards making a data-driven decision about which visualization is the best way to tell your story. Please note that there are many different types of decision trees that vary in complexity, and can provide more in-depth decisions.



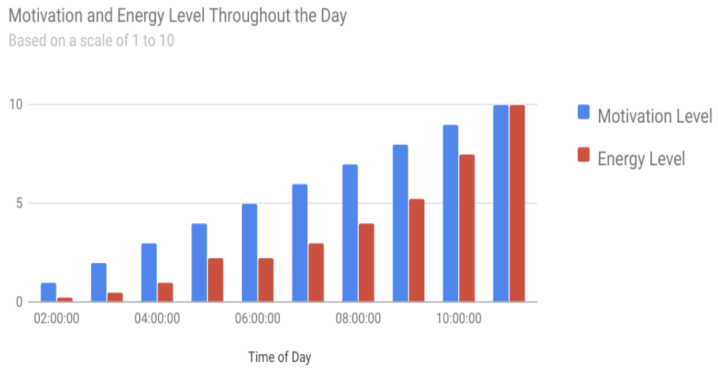
Design Data Visualizations

The elements of art:

- **Line:** Lines add visual form to your data and help build the structure for your visualization.
- **Shape:** Shapes create visual contrast that can help differentiate different data points, like different countries on a map visualization.
- **Color**
    - **Hue:** The hue is the color.
    - **Intensity:** Intensity is how bright or dull the color is
    - **Value:** The color's brightness or darkness
- **Space**
- **Movement**

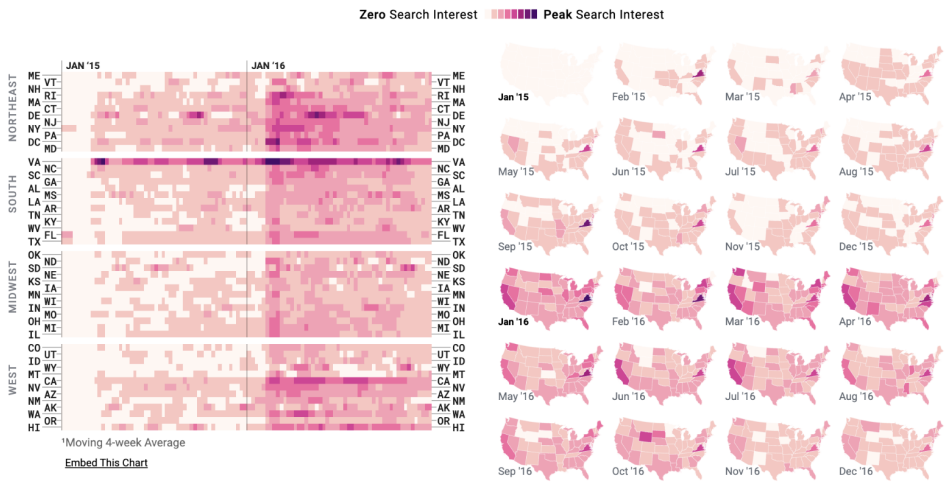
Nine basic principles of design:

- **Balance:** The design of a data visualization is balanced when the key visual elements, like color and shape, are distributed evenly. This doesn't mean that you need complete symmetry, but your visualization shouldn't have one side distracting from the other. If your data visualization is balanced, this could mean that the lines used to create the graphics are similar in length on both sides, or that the space between objects is equal. For example, teh column chart shown below is balanced; even though the columns are different heights and the chart isn't symmetrical, the colors, width, and spacing of the columns keep this data visualization balanced. The colors provide sufficient contrast to each other so that you can pay attention to both the motivation level and the energy level displayed.



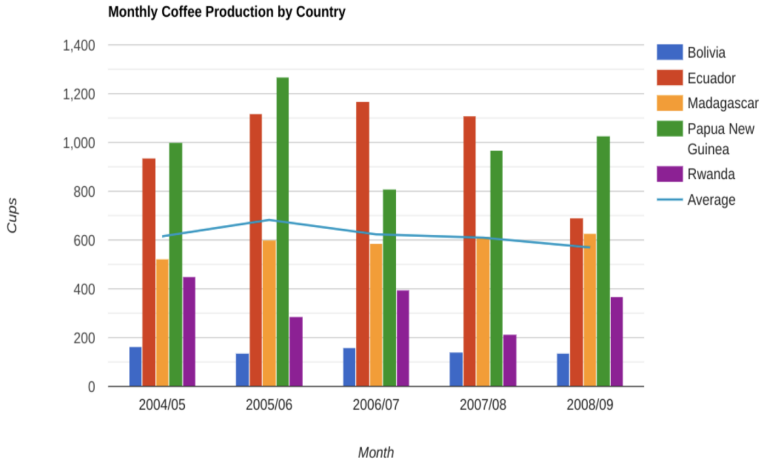
- **Emphasis:** Your data visualization should have a focal point, so that your audience knows where to concentrate. In other words, your visualizations should emphasize the most important data so that users recognize it first. Using color and value is one effective way to make this happen. By using contrasting colors, you can make certain that graphic elements—and the data shown in those elements—stand out.

For example, you will notice a heat map data visualization below. This heat map uses colors and value intensity to emphasize the states where search interest is highest. You can visually identify the increase in the search over time from low interest to high interest. This way, you are able to quickly grasp the key idea being presented without knowing the specific data values.



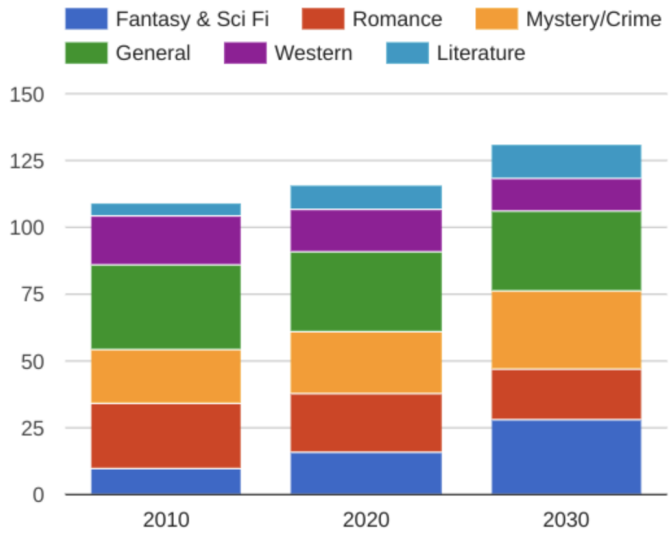
- **Movement:** Movement can refer to the path the viewer's eye travels as they look at a data visualization, or literal movement created by animations. Movement in data visualization should mimic the way people usually read. You can use lines and colors to pull the viewer's attention across the page.

  For example, notice how the average line in this combo chart (also shown below) draws your attention from left to right. Even though this example isn't moving, it still uses the movement principle to guide viewers' understanding of the data.



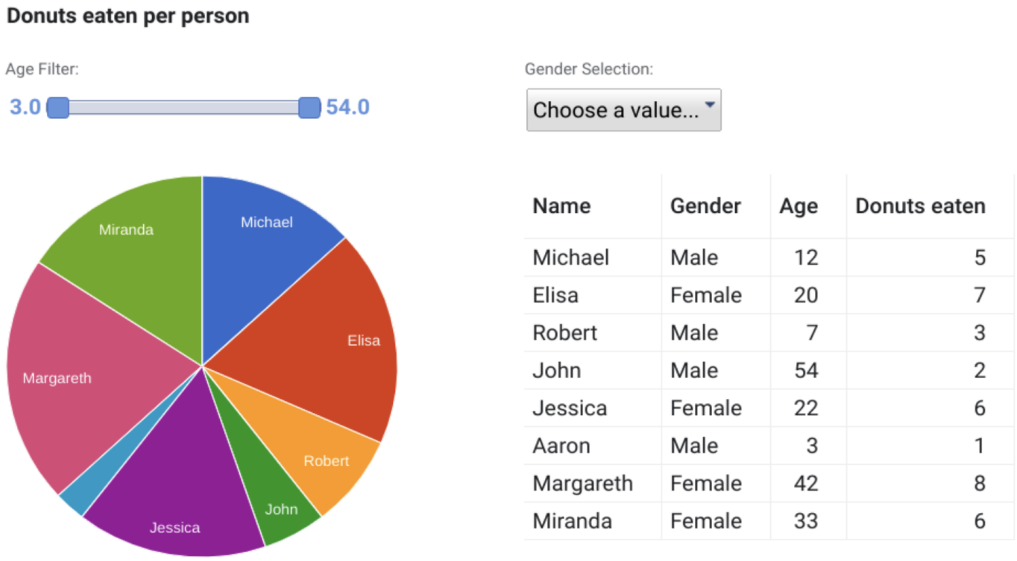
- **Pattern:** You can use similar shapes and colors to create patterns in your data visualization. This can be useful in a lot of different ways. For example, you can use patterns to highlight similarities between different data sets, or break up a pattern with a unique shape, color, or line to create more emphasis.

  In the example below, the different colored categories of this stacked column chart (also shown below) are a consistent pattern that makes it easier to compare book sales by genre in each column. Notice in the chart that the Fantasy & Sci Fi category (royal blue) is increasing over time even as the general category (green) is staying about the same.



- **Repetition:** Repeating chart types, shapes, or colors adds to the effectiveness of your visualization. Think about the book sales chart from the previous example: the repetition of the colors helps the audience understand that there are distinct sets of data. You may notice this repetition in all of the examples we have reviewed so far. Take some time to review each of the previous examples and notice the elements that are repeated to create a meaningful visual story.

- **Proportion:** Proportion is another way that you can demonstrate the importance of certain data. Using various colors and sizes helps demonstrate that you are calling attention to a specific visual over others. If you make one chart in a dashboard larger than the others, then you are calling attention to it. It is important to make sure that each chart accurately reflects and visualizes the relationship among the values in it. In this dashboard (also shown below), the slice sizes and colors of the pie chart compared to the data in the table help make the number of donuts eaten by each person the focal point



Name	Gender	Age	Donuts eaten
Michael	Male	12	5
Elisa	Female	20	7
Robert	Male	7	3
John	Male	54	2
Jessica	Female	22	6
Aaron	Male	3	1
Margareth	Female	42	8
Miranda	Female	33	6

These first six principles of design are key considerations that you can make while you are creating your data visualization. These next three principles are useful checks once your data visualization is finished. If you have applied the initial six principles thoughtfully, then you will probably recognize these next three principles within your visualizations already.

- **Rhythm:** This refers to creating a sense of movement or flow in your visualization. Rhythm is closely tied to the movement principle. If your finished design doesn't successfully create a flow, you might want to rearrange some of the elements to improve the rhythm.
- **Variety:** Your visualizations should have some variety in the chart types, lines, shapes, colors, and values you use. Variety keeps the audience engaged. But it is good to find balance since too much variety can confuse people. The variety you include should make your dashboards and other visualizations feel interesting and unified.
- **Unity:** The last principle is unity. This means that your final data visualization should be cohesive. If the visual is disjointed or not well organized, it will be confusing and overwhelming.

Data Visualization impact:

- Choosing visualizations for your findings can often come down to one question: **Which one will make it easier for the user to understand the point you're trying to make?**

- **Data Composition:** Combining the individual parts in a visualization and displaying them together as a whole. Example: Pie chart, etc

- Visual journalist **Dona Wong** proposes three essential elements for effective visualization:
    - **Clear meaning:** Good visualizations clearly communicate their intended insight.
    - **Sophisticated use of contrast:** Helps separating the important data from the rest using the visual context that our brains naturally look for.
    - **Refined execution:** Visuals with refined execution include deep attention to details using visual elements like line, shape, color, space, and movement.

Design Thinking and visualizations:

- **Design Thinking:** A process used to solve complex problems in a user-centric way.
- When you bring design thinking into the work, you're trying to identify alternate strategies for your visualizations that might not be clear right away.
- Five phases of design process: **(Check the video for definitions of each individual phase)**
    - Empathize: Thinking about the emotions and needs of the target audience for the data visualization.
    - Define: Figuring out exactly what your audience needs from the data.
    - Ideate: Generating ideas for data visualization.
    - Prototype: Putting visualizations together for testing and feedback.
    - Test: Showing prototype visualizations to people before stakeholders see them.
- Above phases can be seen as actions that will help you produce a user centered design in your visualization.

<u>Explore Visualization Consideration</u>

Headlines, subtitles, labels, and annotations help you turn your data visualizations into more meaningful displays. After all, you want to invite your audience into your presentation and keep them engaged. When you present a visualization, they should be able to process and understand the information you are trying to share in the first five seconds.

Your audience will be less likely to have questions about what you're sharing if you add:

- **Headline:** A line of words printed in large letters at the top of the visualization to communicate what data is being presented.
- **Subtitles:** Supports the headline by adding more context and description
- **Labels:** It is recommended to use labels directly on the diagram rather than using **legends** (Identifies the meaning of various elements in a data visualization).

Sometimes data visualizations can become too crowded or busy. When this happens, the audience can get confused or distracted by elements that aren't really necessary. The guidelines will help keep your data visualizations simple, and the style checks will help make your data visualizations more elegant.

Visualization components	Guidelines	Style checks
Headlines	- Content: Briefly describe the data - Length: Usually the width of the data frame - Position: Above the data	- Use brief language - Don’t use all caps - Don’t use italic - Don’t use acronyms - Don't use abbreviations - Don’t use humor or sarcasm
Subtitles	- Content: Clarify context for the data - Length: Same length or shorter than headline - Position: Directly below the headline	- Use smaller font size than the headline - Don’t use words that need definition - Don’t use all caps, bold, or italic - Don’t use acronyms or abbreviations
Labels	- Content: Replace the need for legends - Length: Usually <= 30 characters, unless for axes - Position: Next to data, or below or to the left of axes	- Use a few words only - Use thoughtful color-coding - Use callouts to point to the data - Don’t use all caps, bold, or italic
Annotations	- Content: Draws attention to certain data - Length: Varies, limited by available open space - Position: Immediately next to data annotated	- Don’t use all caps, bold, or italic - Don't use rotated text - Don’t distract viewers from the data

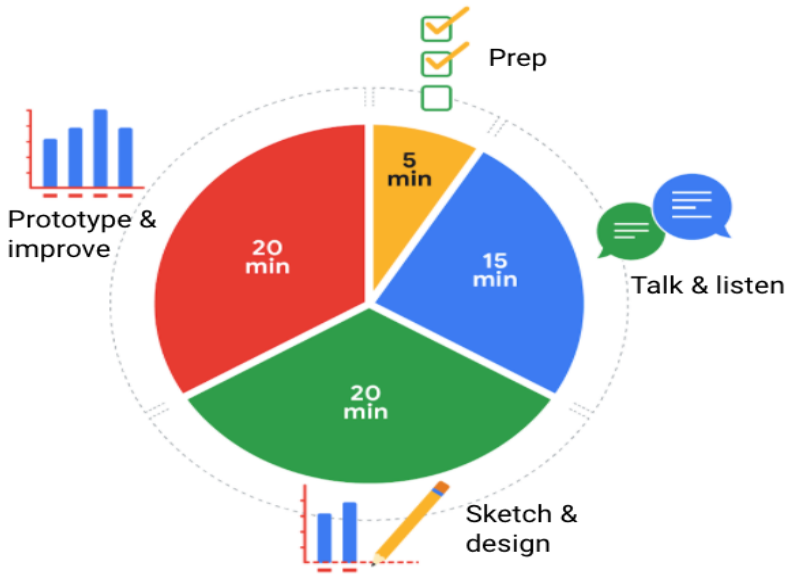
Accessible visualizations:

Ways to make data visualizations accessible:

- Labeling
- Text alternatives: Alternative text provides a textual alternative to the non-text content.
- Text-based format
- Distinguishing
- Simplify

Designing a chart in 60 minutes

Follow this high level 60-minute chart to guide your thinking whenever you begin working on a data visualization.



Prep (5 min): Create the mental and physical space necessary for an environment of comprehensive thinking. This means allowing yourself room to brainstorm *how* you want your data to appear while considering the amount and type of data that you have.

Talk and listen (15 min): Identify the object of your work by getting to the “ask behind the ask” and establishing expectations. Ask questions and really concentrate on feedback from stakeholders regarding your projects to help you hone how to lay out your data.

Sketch and design (20 min): Draft your approach to the problem. Define the timing and output of your work to get a clear and concise idea of what you are crafting.

Prototype and improve (20 min): Generate a visual solution and gauge its effectiveness at accurately communicating your data. Take your time and repeat the process until a final visual is produced. It is alright if you go through several visuals until you find the perfect fit.

Week-2

Get started with Tableau

Tableau: A business intelligence and analytics platform that helps people see, understand, and make decisions with data.

Practice visualizations from "Visualizations in spreadsheets and tableau" reading

Create visualizations in Tableau

The good, the bad, and the ugly

Diverging color palette: Displays two ranges of values using color intensity to show the magnitude of the number and the actual color to show which range the numbers from.

A diverging color palette in Tableau displays a value's magnitude by color intensity and a value's range by color hue.

Week-3

Use data to develop stories

Storytelling with data

Dashboard: A tool that organizes information, typically from multiple data sets, into one central location for tracking, analysis, and simple visualization through charts, graphs, and maps.

Dashboard filter: A tool for showing only the data that meets a specific criteria while hiding the rest.

"Numbers have an important story to tell. They rely on you to give them a clear and convincing voice" - Stephen Few

Data Storytelling: Communicating the meaning of a data set with visuals and a narrative that are customized for each particular audience.

Three data storytelling steps:

1. **Engage your audience:** Capturing and holding someone's interest and attention.
2. **Create compelling visuals:** Show the story of your data, not just tell it.
3. **Tell the story in an interesting narrative:** Clearly explain your insights from the data analysis.

<u>Speaking to your audience:</u>

To know your audience and engage them into your storytelling, **try answering these questions**:

- What role does this audience play?
- What is their stake in the project?
- What do they hope to get from the data insights I deliver?

Choose your primary message: Every single point of your story flows from this one key point, so it's better to keep it concise and direct.

Spotlighting: Scanning through the data to quickly identify the most important insights.

Use sticky notes or whiteboard to note all the insights from the analysis, and examine it.

<u>Use Tableau dashboards</u>

Dashboards and static insights:

Dashboards and static insights allow the client or user to be able to have control of their narrative. With dashboards, anyone can use the data and make their own determinations. A dashboard can be built on top of static insights or static data, which is data that doesn't change once it has been recorded. This allows you to tightly control the narrative and the way that your data is presented visually, whether via graphs, charts, etc.

In a business context, data visualizations are most useful when they are presented in a dashboard-style format to stakeholders. Dashboards put all the pertinent information in the same place, making it easier to understand the important takeaways. Many dashboards are also constantly updating to reflect new data, and some are even interactive. No matter what style of dashboard you choose, they can help you deliver the work you've done when creating visualizations.

Sharing data stories

Compelling presentation tips

- Create presentations that are logically organized, interesting, and communicate your key message clearly.
- The narrative you share with the stakeholders needs:
    - Characters: People affected by your story (stakeholders, etc)
    - Setting: Describes what's going on, how often it's happening, what tasks are involved, and other background information about the data project that describes the current situation.
    - Plot:
    - Big reveal
    - Aha moment
- **Check the video for detailed information about the above points**

Check the whole module under Sharing data stories

The art and science of an effective presentation

Presenting with a framework:

Framework: To make your data findings accessible to the audience, you'll need a framework to guide your presentation. The framework you choose gives your audience context to better understand your data. On top of that, it helps keep you focused on the most important information during your presentation. The framework for your presentation starts with your understanding of the business task.

By showcasing what business metrics you use. You can help your audience understand the impact your findings will have.

Weaving data into your presentation:

- Establish the initial hypothesis
    - **Hypothesis:** The theory you're trying to prove or disprove using data.
- Establishing the hypothesis early in the presentation will help the audience understand the data.
- **The McCandless method:**
    - Introduce the graphic by name
    - Answer obvious questions before they're asked
    - State the insight of your graphic.
    - Call out data to support that insight
    - Tell your audience why it matters

        - Present the possible business impact of the solution and clear actions stakeholders can take.

Note:
Business task: A question or problem your data analysis answers or tries to solve.

Identify presentation skills and practices

Proven presentation tips

Presentation tips:

- Channel your excitement
- Start with the broader ideas.
- Use the five-second rule
    - Wait five seconds after showing a data visualization
    - Ask if they understand it.
    - Give your audience another five seconds
    - Tell them the conclusion
- Preparation is key

Present like a pro

Your audience:

- Will not always see the steps you took to reach the conclusion
- Has a lot in their minds
- Is easily distracted.

How you speak:

- Keep your sentences short
- Build in intentional pauses
- Keep the pitch of your sentences level

Be mindful of nervous habits:

- Stay still and move with purpose
- Practice good posture
- Make positive eye contact

Caveats and limitations of data

Anticipate the question

- To anticipate the questions prior to the presentation, you should:
    - Understand the stakeholder's expectations
    - Make sure you have a clear understanding of the objective and what the stakeholders wanted when they asked you to take on this project.

- if you misunderstood your stakeholders' expectations or the project objectives, you won't be able to correctly anticipate or answer their questions.
- A great way to identify audience questions is through **The colleague test (Do a test-run of your presentation).**
- Start with zero assumptions.
    - Don't assume that your audience is already familiar with jargon, acronyms, past events, or other necessary background information.
- Work with your team to anticipate questions and draft responses.

Be prepared to consider any limitations of your data by:

- Critically analyzing the data
- Looking at the context
- Understanding the strengths and weakness of the tools

There are many things to consider before you begin asking and answering possible questions – like the objective, stakeholder expectations, and if there are any limitations. Make sure you have everything covered before you begin. The checklist below identifies ten tasks that you should engage in to be well prepared for your Q&A:

Before Presentation:

- Assemble and prepare your questions.
- Discuss your presentation with your manager, other analysts, or other friendly contacts in your organization.
- Ask a manager or other analysts what sort of questions were normally asked by your specific audience in the past.
- Seek comments, feedback, and questions on the deck or the document of your analysis.
- At least 24 hours ahead of the presentation, try and brainstorm tricky questions or unclear parts you may come across- this helps avoid surprises.
- It never hurts to practice what you will be presenting, to account for any missing information or simply to calm your nerves.

During Presentation:

- Be prepared to respond to the things that you find and effectively and accurately explain your findings.
- Address potential questions that may come up.
- Avoid having a single question derail a presentation and propose following-up offline.
- Put supplementary visualizations and content in the appendix to help answer questions.

Preparing for a presentation or a meeting doesn’t have to be intimidating. If you invest time into knowing your audience, crafting your notes, doing necessary research and organizing your data, then there is very little reason why your audience will not be engaged, even impressed.

Handling Objections

- **Types of objections:**
  - About the data
    - Where did you get the data?
    - What system did it come from?
    - What transformations happened to it?
    - How fresh and accurate is the data?

    You can include all this information in the beginning of the presentation to set up the data context
  - About your analysis
    - Is your analysis reproducible?
      You can maintain a change log for this to repeat the process.
    - Who did you get feedback from?
  - About your findings
    - Do these findings exist in previous time periods?
    - Did you control for the differences in the data?

- **Responding to possible objections:**
  - Communicate any assumptions
  - Explain why your analysis might be different than expected
  - Acknowledge that those objections are valid and require further investigation.

<u>Listen, Repond, and Include</u>

Q&A Best practices

- Tips to ensure you answer an audience member's question appropriately
  - Listen to the whole question
  - Repeat the question (if necessary)
  - Understand the context
  - Involve the whole audience
  - Keep your responses short and to the point

Note: Connor's important aspects to a presentation
- Define your purpose
- Keep it concise
- Have some logical flow to your presentation
- Make the visualization visually compelling
- How easy is it to understand?

Course 7: **Data Analysis with R Programming**

Week-1

The exciting world of programming

Introduction to the exciting world of programming

Computer programming: Giving instructions to a computer to perform an action or set of actions.

R programming is very useful for organizing, cleaning, and analyzing data.

R vs Python

Languages	**R**	**Python**
Common features	- Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support	- Open-source - Data stored in data frames - Formulas and functions readily available - Community for code development and support
Unique advantages	- Data manipulation, data visualization, and statistics packages - "Scalpel" approach to data: *find packages to do what you want with the data*	- Easy syntax for machine learning needs - Integrates with cloud platforms like Google Cloud, Amazon Web Services, and Azure
Unique challenges	- Inconsistent naming conventions make it harder for beginners to select the right functions - Methods for handling variables may be a little complex for beginners to understand	- Many more decisions for beginners to make about data input/output, structure, variables, packages, and objects - "Swiss army knife" approach to data: *figure out a way to do what you want with the data*

R has been used by professionals who have a statistical or research-oriented approach to solving problems; among them are scientists, statisticians, and engineers. Python has been used by professionals looking for solutions in the data itself, those who must heavily mine data for answers; among them are data scientists, machine learning specialists, and software developers.

Benefits of using programming languages with data:

- Clarify the steps of your analysis
- Saves time
- Reproduce and share your work

Spreadsheets, SQL, and R: a comparison

As a data analyst, there is a good chance you will work with SQL, R, and spreadsheets at some point in your career. Each tool has its own strengths and weaknesses, but they all make the data analysis process smoother and more efficient. There are two main things that all three have in common:

- **They all use filters:** for example, you can easily filter a dataset using any of these tools. In R, you can use the filter function. This performs the same task as a basic SELECT-FROM-WHERE SQL query. In a spreadsheet, you can create a filter using the menu options.
- **They all use functions:** In spreadsheets, you use functions in formulas, and in SQL, you include them in queries. In R, you will use functions in the code that is part of your analysis.

The table below presents key questions to explore a few more ways that these tools compare to each other.

Key question	**Spreadsheets**	**SQL**	**R**
What is it?	A program that uses rows and columns to organize data and allows for analysis and manipulation through formulas,	A database programming language used to communicate with databases to conduct an	A general purpose programming language used for statistical analysis, visualization,

	functions, and built-in features	analysis of data	and other data analysis
What is a primary advantage?	Includes a variety of visualization tools and features	Allows users to manipulate and reorganize data as needed to aid analysis	Provides an accessible language to organize, modify, and clean data frames, and create insightful data visualizations
Which datasets does it work best with?	Smaller datasets	Larger datasets	Larger datasets
What is the source of the data?	Entered manually or imported from an external source	Accessed from an external database	Loaded with R when installed, imported from your computer, or loaded from external sources
Where is the data from my analysis usually stored?	In a spreadsheet file on your computer	Inside tables in the accessed database	In an R file on your computer
Do I use formulas and functions?	Yes	Yes	Yes

Can I create visualizations?	Yes	Yes, by using an additional tool like a database management system (DBMS) or a business intelligence (BI) tool	Yes

Introduction to R

R: A programming language frequently used for statistical analysis, visualization, and other data analysis.

Situations where R can be used for data analysis:

- Reproducing your analysis
- Processing lots of data
- Creating data visualizations

Note: Packages are units of reproducible R code. Members of the R community create packages to keep track of the R functions that they write and reuse. Packages offer a helpful combination of code, reusable R functions, descriptive documentation, tests for checking your code, and sample data sets.

The **tidyverse** is a collection of packages in R with a common design philosophy for data manipulation, exploration, and visualization. For a lot of data analysts, the tidyverse is an essential tool.

Commands used in R:

- **install.packages("tidyverse"):** To install packages (tidyverse is the package's name)
- **library():** To load the libraries
- **<-** is an **assignment operator**
- **c(x,y,z….)** is to create **vectors,** where x,y,z… are the values in the vector.

Why RStudio?

One of your core tasks as an analyst will be converting raw data into insights that are accurate, useful, and interesting. That can be tricky to do when the raw data is complex. R and RStudio are

designed to handle large data sets, which spreadsheets might not be able to handle as well. RStudio also makes it easy to reproduce your work on different datasets. When you input your code, it's simple to just load a new dataset and run your scripts again. You can also create more detailed visualizations using RStudio.

When RStudio truly shines

When the data is spread across multiple categories or groups, it can be challenging to manage your analysis, visualize trends, and build graphics. And the more groups of data that you need to work with, the harder those tasks become. That's where RStudio comes in.

For example, imagine you are analyzing sales data for every city across an entire country. That is a lot of data from a lot of different groups–in this case, each city has its own group of data.

Here are a few ways RStudio could help in this situation:

- Using RStudio makes it easy to take a specific analysis step and perform it for each group using basic code. In this example, you could calculate the yearly average sales data for every city.
- RStudio also allows for flexible data visualization. You can visualize differences across the cities effectively using plotting features like facets.
- You can also use RStudio to automatically create an output of summary stats—or even your visualized plots—for each group.

Week-2

Understand Basic programming concepts

Programming with RStudio

- **Pipes** make a sequence of code easier to work with and read.

Programming fundamentals

- The basic concepts of R:
    - Functions: A body of reusable code used to perform specific tasks in R
    - Comments: Use an **#** to comment
    - Variables: A representation of a value in R that can be used later in the programming.
        - first_variable <- "It is an assignment operator (<-)"
    - Data Types
    - Vectors: A group of data elements of the same type stored in a sequence in R.
        - Vector_1 <- c(1,2,3,4)

    - Pipes: A tool in R for expressing a sequence of multiple operations, represented with “%>%”

Vectors and Lists in R

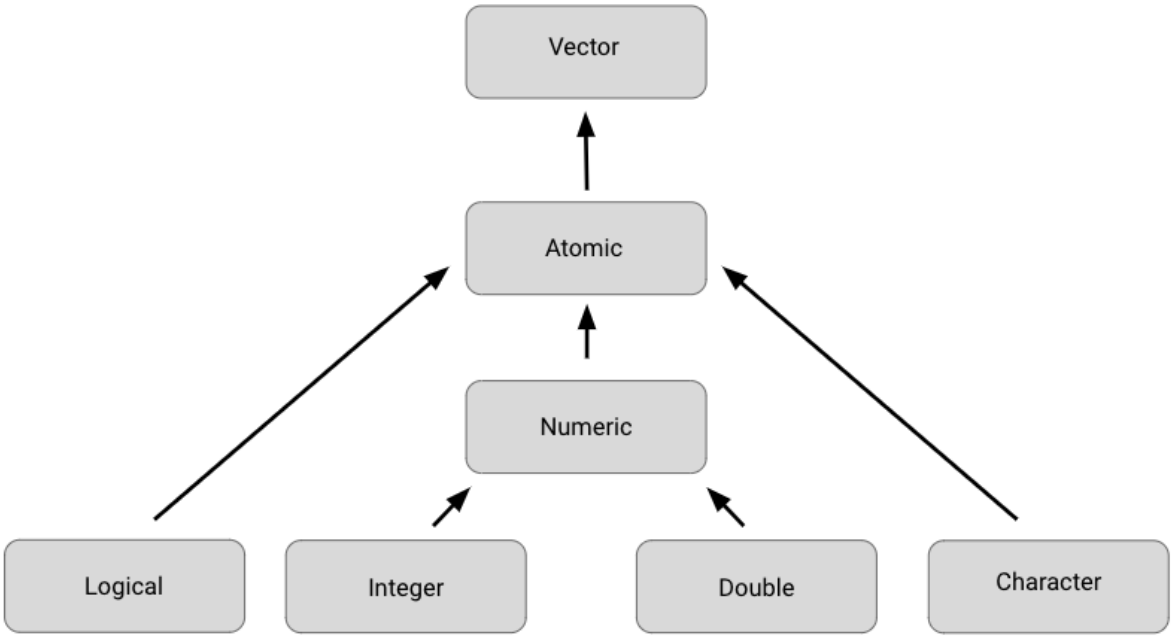
- In programming, a data structure is a format for organizing and storing data. Data structures are important to understand because you will work with them frequently when you use R for data analysis. The most common data structures in the R programming language include:
    - Vectors
    - Data Frames
    - Matrices
    - Arrays
- There are two types of vectors:
    - Atomic vectors
    - Lists

Atomic Vectors:

- There are six primary types of atomic vectors: logical, integer, double, character (which contains strings), complex, and raw. The last two–complex and raw–aren’t as common in data analysis, so we will focus on the first four. Together, integer and double vectors are known as numeric vectors because they both contain numbers. This table summarizes the four primary types:

Type	**Description**	**Example**
Logical	True/False	**TRUE**
Integer	Positive and negative whole values	**3**
Double	Decimal values	**101.175**
Character	String/character values	**“Coding”**

This diagram illustrates the hierarchy of relationships among these four main types of vectors:



Creating vectors:

- One way to create a vector is by using the c() function (called the “combine” function). The c() function in R combines multiple values into a vector. In R, this function is just the letter “c” followed by the values you want in your vector inside the parentheses, separated by a comma: c(x, y, z, …).
    - For example, you can use the c() function to store numeric data in a vector.
        - c(2.5, 48.5, 101.5)
    - To create a vector of integers using the c() function, you must place the letter "L" directly after each number.
        - c(1L, 5L, 15L)
    - You can also create a vector containing characters or logicals.
        - c(“Sara” , “Lisa” , “Anna”)
        - c(TRUE, FALSE, TRUE)

Determining the properties of vector:

- Every vector you create will have two key properties: type and length.
- You can determine what type of vector you are working with by using the **typeof()** function. Place the code for the vector inside the parentheses of the function. When you run the function, R will tell you the type. For example:
    - typeof(c(“a” , “b”)) returns “character”

- Notice that the output of the typeof function in this example is “character”. Similarly, if you use the typeof function on a vector with integer values, then the output will include “integer” instead:
    - typeof(c(1L , 3L)) returns integer
- You can determine the length of an existing vector–meaning the number of elements it contains–by using the **length()** function. In this example, we use an assignment operator to assign the vector to the variable *x*. Then, we apply the length() function to the variable. When we run the function, R tells us the length is 3.
    - x <- c(33.5, 57.75, 120.05)
      length(x)
      #> [1] 3
- In this example, R returns a value of FALSE because the vector does *not* contain characters, rather it contains logicals.
    - y <- c(TRUE, TRUE, FALSE)
      is.character(y)
      #> [1] FALSE

Naming Vectors:

- All types of vectors can be named. Names are useful for writing readable code and describing objects in R. You can name the elements of a vector with the names() function. As an example, let’s assign the variable x to a new vector with three elements.
    - x <- c(1, 3, 5)
    - You can use the names() function to assign a different name to each element of the vector.
    - names(x) <- c("a", "b", "c")
    - Now, when you run the code, R shows that the first element of the vector is named a, the second b, and the third c.
    - x
      #> a b c
      #> 1 3 5

Creating Lists:

- Lists are different from atomic vectors because their elements can be of any type—like dates, data frames, vectors, matrices, and more. Lists can even contain other lists.
- You can create a list with the **list()** function. Similar to the c() function, the list() function is just a list followed by the values you want in your list inside parentheses: **list(x, y, z, …)**. In this example, we create a list that contains four different kinds of elements: character ("a"), integer (1L), double (1.5), and logical (TRUE).
    - list("a", 1L, 1.5, TRUE)

Determining the structures of lists:

- If you want to find out what types of elements a list contains, you can use the **str()** function. To do so, place the code for the list inside the parentheses of the function. When you run the function, R will display the data structure of the list by describing its elements and their types.

- Let's apply the str() function to our first example of a list.

```
str(list("a", 1L, 1.5, TRUE))
```

- We run the function, then R tells us that the list contains four elements, and that the elements consist of four different types: character (chr), integer (int), number (num), and logical (logi).

    - #> List of 4
    - #> $: chr "a"
    - #> $: int 1
    - #> $: num 1.5
    - #> $: logi TRUE

Naming Lists:

- Lists, like vectors, can be named. You can name the elements of a list when you first create it with the list() function:

```
list("Chicago" = 1, "New York" = 2, "Los Angeles" = 3)
$Chicago

[1] 1

$`New York`

[1] 2

$`Los Angeles`

[1] 3
```

Dates and Times in R:

Loading Lubridates and Tidyverse packages:

Before you get started working with dates and times, you should load both **tidyverse and lubridate.** Lubridate is part of tidyverse.

First, open RStudio.

If you haven't already installed tidyverse, you can use the **install.packages()** function to do so:

- `install.packages("tidyverse")`

Next, load the tidyverse and lubridate packages using the **library()** function. First, load the core tidyverse to make it available in your current R session:

- `library(tidyverse)`

Then, load the lubridate package:

- `library(lubridate)`

Now you're ready to be introduced to the tools in the lubridate package.

Working with dates and times

This section covers the data types for dates and times in R and how to convert strings to date-time formats.

Types

In R, there are three types of data that refer to an instant in time:

- A date `("2016-08-16")`
- A time within a day (`"20:11:59 UTC"`)
- And a date-time. This is a date plus a time `("2018-03-31 18:15:48 UTC")`

The time is given in UTC, which stands for Universal Time Coordinated, more commonly called Universal Coordinated Time. This is the primary standard by which the world regulates clocks and time.

For example, to get the current date you can run the **today()** function. The date appears as year, month, and day.

```
today()
```

```
#> [1] "2021-01-20"
```

To get the current date-time you can run the **now()** function. Note that the time appears to the nearest second.

```
now()
```

```
#> [1] "2021-01-20 16:25:05 UTC"
```

When working with R, there are three ways you are likely to create date-time formats:

- From a string
- From an individual date
- From an existing date/time object

R creates dates in the standard yyyy-mm-dd format by default.

Let's go over each.

Converting from strings

Date/time data often comes as strings. You can convert strings into dates and date-times using the tools provided by lubridate. These tools automatically work out the date/time format. First, identify the order in which the year, month, and day appear in your dates. Then, arrange the letters *y*, *m*, and *d* in the same order. That gives you the name of the lubridate function that will parse your date. For example, for the date *2021-01-20,* you use the order *ymd*:

```
ymd("2021-01-20")
```

When you run the function, R returns the date in yyyy-mm-dd format.

```
#> [1] "2021-01-20"
```

It works the same way for any order. For example, month, day, and year. R still returns the date in yyyy-mm-dd format.

```
mdy("January 20th, 2021")
```

```
#> [1] "2021-01-20"
```

Or, day, month, and year. R still returns the date in yyyy-mm-dd format.

```
dmy("20-Jan-2021")
```

```
#> [1] "2021-01-20"
```

These functions also take unquoted numbers and convert them into the yyyy-mm-dd format.

```
ymd(20210120)
```

```
#> [1] "2021-01-20"
```

Creating date-time components

The ymd() function and its variations create dates. To create a date-time from a date*,* add an underscore and one or more of the letters *h*, *m*, and s (hours, minutes, seconds) to the name of the function:

```
ymd_hms("2021-01-20 20:11:59")
```

```
#> [1] "2021-01-20 20:11:59 UTC"
```

```
mdy_hm("01/20/2021 08:01")
```

```
#> [1] "2021-01-20 08:01:00 UTC"
```

Optional: Switching between existing date-time objects

Finally, you might want to switch between a date-time and a date.

You can use the function **as_date()** to convert a date-time to a date. For example, put the current date-time—now()—in the parentheses of the function.

```
as_date(now())
```

```
#> [1] "2021-01-20"
```

Data frames

Data frames are the most common way of storing and analyzing data in R, so it's important to understand what they are and how to create them. A **data frame** is a collection of columns–similar to a spreadsheet or SQL table. Each column has a name at the top that represents a variable, and includes one observation per row. Data frames help summarize data and organize it into a format that is easy to read and use.

There are a few key things to keep in mind when you are working with data frames:

- First, columns should be named.
- Second, data frames can include many different types of data, like numeric, logical, or character.
- Finally, elements in the same column should be of the same type.

You will learn more about data frames later on in the program, but this is a great starting point.

If you need to manually create a data frame in R, you can use the data.frame() function. The **data.frame()** function takes vectors as input. In the parentheses, enter the name of the column, followed by an equals sign, and then the vector you want to input for that column. In this example, the *x* column is a vector with elements 1, 2, 3, and the *y* column is a vector with elements 1.5, 5.5, 7.5.

```
data.frame(x = c(1, 2, 3) , y = c(1.5, 5.5, 7.5))
```

If you run the function, R displays the data frame in ordered rows and columns.

```
  x y
1  1 1.5
2  2 5.5
3  3 7.5
```

In most cases, you won't need to manually create a data frame yourself, as you will typically import data from another source, such as a .csv file, a relational database, or a software program.

Files

Let's go over how to create, copy, and delete files in R. For more information on working with files in R, check out R documentation: files. **R documentation** is a tool that helps you easily find and browse the documentation of almost all R packages on CRAN. It's a useful reference guide for functions in R code. Let's go through a few of the most useful functions for working with files.

Use the **dir.create** function to create a new folder, or directory, to hold your files. Place the name of the folder in the parentheses of the function.

```
dir.create ("destination_folder")
```

Use the **file.create()** function to create a blank file. Place the name and the type of the file in the parentheses of the function. Your file types will usually be something like .txt, .docx, or .csv.

```
file.create (“new_text_file.txt”)

file.create (“new_word_file.docx”)

file.create (“new_csv_file.csv”)
```

If the file is successfully created when you run the function, R will return a value of `TRUE` (if not, R will return `FALSE`).

```
file.create (“new_csv_file.csv”)
```

```
[1] TRUE
```

Copying a file can be done using the **file.copy()** function. In the parentheses, add the name of the file to be copied. Then, type a comma, and add the name of the destination folder that you want to copy the file to.

```
file.copy (“new_text_file.txt” , “destination_folder”)
```

If you check the Files pane in RStudio, a copy of the file appears in the relevant folder:

You can delete R files using the **unlink()** function. Enter the file’s name in the parentheses of the function.

```
unlink (“some_.file.csv”)
```

Matrices

A **matrix** is a two-dimensional collection of data elements. This means it has both rows and columns. By contrast, a vector is a one-dimensional sequence of data elements. But like vectors, matrices can only contain a single data type. For example, you can’t have both logicals and numerics in a matrix.

To create a matrix in R, you can use the **matrix()** function. The matrix() function has two main arguments that you enter in the parentheses. First, add a vector. The vector contains the values you want to place in the matrix. Next, add at least one matrix dimension. You can choose to specify the number of rows or the number of columns by using the code nrow = or ncol =.

For example, imagine you want to create a 2x3 (two rows by three columns) matrix containing the values 3-8. First, enter a vector containing that series of numbers: c(3:8). Then, enter a comma. Finally, enter nrow = 2 to specify the number of rows.

```
matrix(c(3:8), nrow = 2)
```

If you run the function, R displays a matrix with three columns and two rows (typically referred to as a “2x3”) that contain the numeric values 3, 4, 5, 6, 7, 8. R places the first value (3) of the vector in the uppermost row, and the leftmost column of the matrix, and continues the sequence from left to right.

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

[1,]    3    5    7

[2,]    4    6    8
```

You can also choose to specify the number of columns (ncol =) instead of the number of rows (nrow =).

```
matrix(c(3:8), ncol = 2)
```

When you run the function, R infers the number of rows automatically.

```
     [,1] [,2]

[1,]    3    6

[2,]    4    7

[3,]    5    8
```

<u>Exploring Coding in R</u>

Operators and Calculations

Operator: A symbol that names the type of operation or calculation to be performed in a formula.

Operators in R:

- **Arithmetic operators:** Used to complete math calculations
    - + (addition)
    - - (Subtraction)
    - * (Multiplication) and / (Division)

Logical Operators

Logical operators return a logical data type such as TRUE or FALSE.

There are three primary types of logical operators:

- AND (sometimes represented as & or && in R)
- OR (sometimes represented as | or || in R)
- NOT (!)

Review the summarized logical operators below.

AND operator "&"

- The AND operator takes two logical values. It returns **TRUE** only if both individual values are TRUE. This means that TRUE & TRUE evaluates to **TRUE**. However, FALSE & TRUE, TRUE & FALSE, and FALSE & FALSE all evaluate to **FALSE**.

If you run the the corresponding code in R, you get the following results:

```
> TRUE & TRUE
[1] TRUE
> TRUE & FALSE
[1] FALSE
> FALSE & TRUE
[1] FALSE
> FALSE & FALSE
[1] FALSE
```

You can illustrate this using the results of our comparisons. Imagine you create a variable x that is equal to 10.

```
x <- 10
```

To check if x is greater than 3 but less than 12, you can use x > 3 and x < 12 as the values of an "AND" expression.

```
x > 3 & x < 12
```

When you run the function, R returns the result TRUE.

[1] TRUE

The first part, **x > 3** will evaluate to **TRUE** since 10 is greater than 3. The second part, **x < 12** will also evaluate to **TRUE** since 10 is less than 12. So, since both values are TRUE, the result of the AND expression is **TRUE**. The number 10 lies between the numbers 3 and 12.

However, if you make x equal to 20, the expression **x > 3 & x < 12** will return a different result.

x <- 20
x > 3 & x < 12
[1] FALSE

- Although **x > 3** is **TRUE** (20 > 3), **x < 12** is **FALSE** (20 < 12). If one part of an AND expression is FALSE, the entire expression is FALSE (TRUE & FALSE = FALSE). So, R returns the result **FALSE**.

OR operator "|"

- The OR operator (|) works in a similar way to the AND operator (&). The main difference is that at least one of the values of the OR operation must be TRUE for the entire OR operation to evaluate to **TRUE**. This means that TRUE | TRUE, TRUE | FALSE, and FALSE | TRUE all evaluate to **TRUE**. When both values are FALSE, the result is **FALSE**.

If you write out the code, you get the following results:

> TRUE | TRUE

[1] TRUE

> TRUE | FALSE

[1] TRUE

> FALSE | TRUE

[1] TRUE

> FALSE | FALSE

[1] FALSE

For example, suppose you create a variable y equal to 7. To check if y is less than 8 or greater than 16, you can use the following expression:

y <- 7

y < 8 | y > 16

The comparison result is TRUE (7 is less than 8) | FALSE (7 is not greater than 16). Since only one value of an OR expression needs to be TRUE for the entire expression to be TRUE, R returns a result of TRUE.

[1] TRUE

Now, suppose y is 12. The expression y < 8 | y > 16 now evaluates to FALSE (12 < 8) | FALSE (12 > 16). Both comparisons are FALSE, so the result is **FALSE**.

y <- 12

y < 8 | y > 16

- **[1] FALSE**

NOT operator “!”

- The NOT operator (!) simply negates the logical value it applies to. In other words, !TRUE evaluates to **FALSE**, and !FALSE evaluates to **TRUE**.

When you run the code, you get the following results:

> !TRUE

[1] FALSE

> !FALSE

[1] TRUE

Just like the OR and AND operators, you can use the NOT operator in combination with logical operators. Zero is considered FALSE and non-zero numbers are taken as TRUE. The NOT operator evaluates to the opposite logical value.

Let's imagine you have a variable x that equals 2:

x <- 2

The NOT operation evaluates to FALSE because it takes the opposite logical value of a non-zero number (TRUE).

> !x

- **[1] FALSE**

Let's check out an example of how you might use logical operators to analyze data. Imagine you are working with the *airquality* dataset that is preloaded in RStudio. It contains data on daily air quality measurements in New York from May to September of 1973.

The data frame has six columns: *Ozone* (the ozone measurement), *Solar.R* (the solar measurement), *Wind* (the wind measurement), *Temp* (the temperature in Fahrenheit), and the *Month* and *Day* of these measurements (each row represents a specific month and day combination).

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4

AND example

Imagine you want to specify rows that are extremely sunny and windy, which you define as having a *Solar* measurement of over 150 and a *Wind* measurement of over 10.

In R, you can express this logical statement as **Solar.R > 150 & Wind > 10**.

Only the rows where *both* of these conditions are true fulfill the criteria:

	Ozone	Solar.R	Wind	Temp	Month	Day
1	18	313	11.5	62	5	4

OR example

Next, imagine you want to specify rows where it's extremely sunny or it's extremely windy, which you define as having a *Solar* measurement of over 150 or a *Wind* measurement of over 10.

In R, you can express this logical statement as **Solar.R > 150 | Wind > 10**.

All the rows where *either* of these conditions are true fulfill the criteria:

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	12	149	12.6	74	5	3
3	18	313	11.5	62	5	4

NOT example

Now, imagine you just want to focus on the weather measurements for days that aren't the first day of the month.

In R, you can express this logical statement as **Day != 1**.

The rows where this condition is true fulfill the criteria:

	Ozone	Solar.R	Wind	Temp	Month	Day
1	36	118	8.0	72	5	2
2	12	149	12.6	74	5	3
3	18	313	11.5	62	5	4

Finally, imagine you want to focus on scenarios that aren't extremely sunny and not extremely windy, based on your previous definitions of extremely sunny and extremely windy. In other words, the following statement should not be true: either a *Solar* measurement greater than 150 or a *Wind* measurement greater than 10.

Notice that this statement is the opposite of the OR statement used above. To express this statement in R, you can put an exclamation point (!) in front of the previous OR statement: **!(Solar.R > 150 | Wind > 10)**. R will apply the NOT operator to everything within the parentheses.

In this case, only one row fulfills the criteria:

	Ozone	Solar.R	Wind	Temp	Month	Day
1	36	118	8.0	72	5	2

Conditional Statements

A conditional statement is a declaration that if a certain condition holds, then a certain event must take place. For example, "*If* the temperature is above freezing, *then* I will go outside for a walk." If the first condition is true (the temperature is above freezing), then the second condition will occur (I will go for a walk). Conditional statements in R code have a similar logic.

Let's discuss how to create conditional statements in R using three related statements:

- **if()**
- **else()**
- **else if()**

if statement

The if statement sets a condition, and if the condition evaluates to TRUE, the R code associated with the if statement is executed.

In R, you place the code for the condition inside the parentheses of the if statement. The code that has to be executed if the condition is TRUE follows in curly braces (expr). Note that in this case, the second curly brace is placed on its own line of code and identifies the end of the code that you want to execute.

```
if (condition) {

 expr

}
```

For example, let's create a variable *x* equal to 4.

```
x <- 4
```

Next, let's create a conditional statement: if *x* is greater than 0, then R will print out the string "x is a positive number".

```
if (x > 0) {

  print("x is a positive number")

}
```

Since x = 4, the condition is true (4 > 0). Therefore, when you run the code, R prints out the string "x is a positive number".

```
[1] "x is a positive number"
```

But if you change x to a negative number, like -4, then the condition will be FALSE (-4 > 0). If you run the code, R will not execute the print statement. Instead, a blank line will appear as the result.

else statement

The else statement is used in combination with an if statement. This is how the code is structured in R:

```
if (condition) {

  expr1

} else {

 expr2

}
```

The code associated with the else statement gets executed whenever the condition of the if statement is *not* TRUE. In other words, if the condition is TRUE, then R will execute the code in the if statement (*expr1*); if the condition is *not* TRUE, then R will execute the code in the else statement (*expr2*).

Let's try an example. First, create a variable *x* equal to 7.

```
x <- 7
```

Next, let's set up the following conditions:

- If x is greater than 0, R will print "x is a positive number".
- If x is less than or equal to 0, R will print "x is either a negative number or zero".

In our code, the first condition (x > 0) will be part of the if statement. The second condition of x less than or equal to 0 is implied in the else statement. If x > 0, then R will print "x is a positive number". Otherwise, R will print "x is either a negative number or zero".

```
x <- 7

if (x > 0) {
```

```
 print ("x is a positive number")

} else {

 print ("x is either a negative number or zero")

}
```

Since 7 is greater than 0, the condition of the if statement is true. So, when you run the code, R prints out “x is a positive number”.

```
[1] "x is a positive number"
```

But if you make x equal to -7, the condition of the if statement is *not* true (-7 is not greater than 0). Therefore, R will execute the code in the else statement. When you run the code, R prints out “x is either a negative number or zero”.

```
x <- -7

if (x > 0) {

 print("x is a positive number")

} else {

 print ("x is either a negative number or zero")

}

[1] "x is either a negative number or zero"
```

else if statement

In some cases, you might want to customize your conditional statement even further by adding the else if statement. The else if statement comes in between the if statement and the else statement. This is the code structure:

```
if (condition1) {

 expr1

} else if (condition2) {

 expr2

} else {

 expr3

}
```

If the if condition (*condition1*) is met, then R executes the code in the first expression (*expr1*). If the if condition is not met, and the else if condition (*condition2*) is met, then R executes the code in the second expression (*expr2*). If neither of the two conditions are met, R executes the code in the third expression (*expr3*).

In our previous example, using only the if and else statements, R can only print "x is either a negative number or zero" if x equals 0 or x is less than zero. Imagine you want R to print the string "x is zero" if x equals 0. You need to add another condition using the else if statement.

Let's try an example. First, create a variable *x* equal to negative 1 ("-1").

```
x <- -1
```

Now, you want to set up the following conditions:

- If x is less than 0, print "x is a negative number"
- If x equals 0, print "x is zero"
- Otherwise, print "x is a positive number"

In the code, the first condition will be part of the if statement, the second condition will be part of the else if statement, and the third condition will be part of the else statement. If x < 0, then R

will print “x is a negative number”. If x = 0, then R will print “x is zero”. Otherwise, R will print “x is a positive number”.

```
x <- -1

if (x < 0) {

 print("x is a negative number")

} else if (x == 0) {

 print("x is zero")

} else {

 print("x is a positive number")

}
```

Since -1 is less than 0, the condition for the if statement evaluates to TRUE, and R prints “x is a negative number”.

```
[1] "x is a negative number"
```

If you make x equal to 0, R will first check the if condition (x < 0), and determine that it is FALSE. Then, R will evaluate the else if condition. This condition, x==0, is TRUE. So, in this case, R prints “x is zero”.

If you make x equal to 1, both the if condition and the else if condition evaluate to FALSE. So, R will execute the else statement and print “x is a positive number”.

As soon as R discovers a condition that evaluates to TRUE, R executes the corresponding code and ignores the rest.

R Sandbox

Step 1: Using `R packages`

`Packages` are a key part of working with `R.`They contain bundles of code called `functions` that allow you to perform a wide range of tasks in `R.` Some of them even contain datasets that you can use to practice the skills you have been learning throughout this course.

Some `packages` are installed by default, but many others can be downloaded from an external source such as the Comprehensive R Archive Network, or CRAN.

In this activity, you will be using a package called `tidyverse.` The `tidyverse` package is actually a collection of individual `packages` that can help you perform a wide variety of analysis tasks.

To install the `tidyverse` package, execute the code in the code chunk below by clicking on the green arrow button in the top right corner. When you execute a code chunk in RMarkdown, the output will appear in the .rmd area and your console.

```{r}
install.packages("tidyverse")
```

Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses, like this:

```{r}
library(tidyverse)
```

Installing and loading the `tidyverse` package may take a few minutes-- be sure to wait for it to finish running before moving on to the next steps!

Once the chunk above has finished running, you will get a report that summarizes what packages were loaded because you ran the `library()` function. The report will also let you know if there are any `functions` that have a conflict, but you don't need to worry about that for now.

Now that you have loaded an `R package,` you can start exploring some data.

Step 2: Viewing data

Many of the `tidyverse` packages contain sample datasets that you can use to practice your `R` skills. The `diamonds` dataset in the `ggplot2` package is a great example for previewing `R` functions.

Because you already loaded this package in the last step, the `diamonds` dataset is ready for you to use.

One common function you can use to preview the data is the `head()` function, which displays the columns and the first several rows of data. You can test out how the `head()` function works by running the chunk below:

```{r}
head(diamonds)
```

In addition to `head()` there are a number of other useful functions you can use to summarize or preview the data. For example, the `str()` and `glimpse()` functions will both return summaries of each column in your data arranged horizontally. You can try out these two functions by running the code chunks below:

```{r}
str(diamonds)
```

```{r}
glimpse(diamonds)
```

Another simple function that you may use regularly is the `colnames()` function. It returns a list of column names from your dataset. You can check out this function by running the code chunk below:

```{r}
colnames(diamonds)
```

After running the code chunk, you may have noticed a number in brackets. This number helps you count the number of columns in your dataset. If you have data with lots of columns and `colnames()` prints the results on multiple lines, each line will have a number in brackets at the start of the line indicating what number column that is! So, for example, "carat" is the first column in the `diamonds` dataset. On the second line, there is the number seven in brackets; "price" is the seventh column.

Step 3: Cleaning data

One of the most frequent tasks you will have to perform as an analyst is to clean and organize your data. `R` makes this easy! There are many functions you can use to help you perform important tasks easily and quickly.

For example, you might need to rename the columns, or variables, in your data. There is a function for that: `rename().` You can check out how it works in the chunk below:

```{r}
rename(diamonds, carat_new = carat)
```

Here, the function is being used to change the name of `carat` to `carat_new`. This is a pretty basic change, but `rename()` has many options that can help you do more complex changes across all of the variables in your data.

For example, you can rename more than one variable in the same `rename()` code. The code below demonstrates how:

```{r}
rename(diamonds, carat_new = carat, cut_new = cut)
```
Another handy function for summarizing your data is `summarize().` You can use it to generate a wide range of summary statistics for your data. For example, if you wanted to know what the mean for `carat` was in this dataset, you could run the code in the chunk below:

```{r}
summarize(diamonds, mean_carat = mean(carat))
```
These functions are a great way to get more familiar with your data and start making observations about it. But sometimes, previewing tables isn't enough to understand a dataset. Luckily, `R` has visualization tools built in.

Step 4: Visualizing data

With `R,` you can create data visualizations that are simple and easy to understand or complicated and beautiful just by changing a bit of code. `R` empowers you to present the same data in so many different ways, which can help you create new insights or highlight important data findings. One of the most commonly used visualization packages is the `ggplot2` package, which is loaded automatically when you install and load `tidyverse.` The `diamonds` dataset that you have been using so far is a `ggplot2` dataset.

To build a visualization with `ggplot2` you layer plot elements together with a `+` symbol. You will learn a lot more about using `ggplot2` later in the course, but here is a preview of how easy and flexible it is to make visuals using code:

```{r}
ggplot(data = diamonds, aes(x = carat, y = price)) +
  geom_point()
```
The code above takes the `diamonds` data, plots the carat column on the X-axis, the price column on the Y-axis, and represents the data as a scatter plot using the `geom_point()` command.

`ggplot2` makes it easy to modify or improve your visuals. For example, if you wanted to change the color of each point so that it represented another variable, such as the cut of the diamond, you can change the code like this:

```{r}
ggplot(data = diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point()
```

Wow, that's a busy visual! Sometimes when you are trying to represent many different aspects of your data in a visual, it can help to separate out some of the components. For example, you could create a different plot for each type of cut. `ggplot2` makes it easy to do this with the `facet_wrap()` function:

```{r}
ggplot(data = diamonds, aes(x = carat, y = price, color = cut)) +
  geom_point() +
  facet_wrap(~cut)
```

You will learn many other ways of working with `ggplot2` to make functional and beautiful visuals later on. For now, hopefully you understand that it is both flexible and powerful!

Step 5: Documentation

You have been working in an `R markdown` file, which allows you to put code and writing in the same place. Markdown is a simple language for adding formatting to text documents. For example, all of the section headers have been formatted by adding `##` to the beginning of the line. Markdown can be used to format the text in other ways, such as creating bulleted lists:

- So if you have a list of things
- Or want to write bullets for another reason
- You can do that using markdown

When you have written, executed, and documented your code in an `R markdown` document like this, you can use the `knit` button in the menu bar at the top of the editing pane to export your work to a beautiful, readable document for others.

Activity Wrap-up

You have had a chance to explore more `R` tools that you can start using on your own. You learned how to install and load `R packages`; functions for viewing, cleaning, and visualizing data; and using `R markdown`to export your work. Feel free to continue exploring these functions in the rmd file, or use this code in your own RStudio project space. As you practice on your own, consider how `R` is similar and different from the tools you have already learned in this program, and how you might start using it for your own data analysis projects. `R` provides a lot of flexibility and utility that can make it a key tool in a data analyst's tool kit.

Learning about R Packages

The gift that keeps on giving

- **Packages(R):** Units of reproducible R code

- **Packages include:**

    - Reusable R functions
    - Documentation about R functions
    - Sample datasets
    - Tests for checking your code.

- **CRAN(Comprehensive R Archive Network):** An online archive R packages, source code, manuals, and documentation.

Welcome to the tidyverse

- Packages offer a helpful combination of code, reusable R functions, descriptive documentation, tests for checking operability, and sample data sets.
- **Tidyverse(R):** A collection of packages in R with a common design philosophy for data manipulation, exploration, and visualization.
- Conflicts happen when packages have functions with the same names as other functions. This usually happens because an object in your workspace or a package you installed is masking a system object of the same name.
- 8 core tidyverse packages:
    - ggplot2
    - tibble: Deals with data frames
    - tidyr
    - readr
    - purrr: Deals with functions and vectors
    - dplyr
    - stringr: Deals with strings
    - forcats: Deals with factors
        - **Factors(R):** Factors store categorical data in R where the data values are limited and usually based on a finite group like country or year.
- **tidyverse_update()** allows us to check for any updates in the packages of tidyverse
- **update_packages()** will update all packages.
- **install.packages("package_name")** will only update one package
- Conflict notifications are just one type of message that can show up in the console. You might find warnings and error messages as well.

- **Read tidyverse vignettes:** A **vignette** is documentation that acts as a guide to an R package. A vignette shares details about the problem that the package is designed to

solve and how the included functions can help you solve it. The **browseVignettes** function allows you to read through vignettes of a loaded package.
- To check out vignettes for one specific package, type **browseVignettes("packagename")**

Explore the tidyverse

More on the tidyverse

- Four packages that are an essential part of an workflow for data analysts:
    - **Ggplot2:** Create a variety of data viz by applying different visual properties to the data variables in R
    - **dplyr:** Offers a consistent set of functions that help you complete some common data manipulation tasks.
    - **tidyr:** A package used for data cleaning to make tidy data.
    - **readr:** used for importing data
        - **read_csv()** imports a csv file.

Working with Pipes

- **Pipes:** A tool in R for expressing a sequence of multiple operations, represented with "%>%"
- When using pipes:
    - Add the pipe operator at the end of each line of the piped operation, except the last one.
    - Check your code after you've programmed your pipe.
    - Revisit the piped operation to check for parts of your code to fix.

Week-3

Explore Data and R

R Data Frames

- **Data Frame:** A collection of columns
- While using data frames:
    - Columns should be named
    - Data stored can be many different types, like numeric, factor, or character
    - Each column should contain the same number of data items.
- In tidyverse, **tibbles** are like streamlined data frames
- Tibbles:
    - Never change the data types of the inputs
    - Never change the names of your variables
    - Never create row names

    - Make printing easier
- **Tidy data(R):** A way of standardizing the organization of data within R.
- Tidy data standards:
    - Variables are organized into columns
    - Observations are organized into rows
    - Each value must have its own cell.

<u>Hands on activity: Create your own data frame</u>

DataFrame.rmd (Create your own dataframe)

Background for this activity
This activity is focused on creating and using data frames in `R`. A data frame is a collection of columns containing data, similar to a spreadsheet or SQL table. Data frames are one of the basic tools you will use to work with data in `R`. And you can create data frames from different data sources.

There are three common sources for data:

- A`package` with data that can be accessed by loading that `package`
- An external file like a spreadsheet or CSV that can be imported into `R`
- Data that has been generated from scratch using `R` code

Wherever data comes from, you will almost always want to store it in a data frame object to work with it. Now, you can start creating and exploring data frames with the code chunks in the RMD space. To interact with the code chunk, click the green arrow in the top-right corner of the chunk. The executed code will appear in the RMD space and your console.

Throughout this activity, you will also have the opportunity to practice writing your own code by making changes to the code chunks yourself. If you encounter an error or get stuck, you can always check the Lesson2_Dataframe_Solutions .rmd file in the Solutions folder under Week 3 for the complete, correct code.

Step 1: Load packages

Start by installing the required package; in this case, you will want to install `tidyverse`. If you have already installed and loaded `tidyverse` in this session, feel free to skip the code chunks in this step.

```{r}
install.packages("tidyverse")
```
Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses:

```{r}
library(tidyverse)
```

Step 2: Create data frame

Sometimes you will need to generate a data frame directly in `R`. There are a number of ways to do this; one of the most common is to create individual vectors of data and then combine them into a data frame using the `data.frame()` function.

Here's how this works. First, create a vector of names by inserting four names into this code block between the quotation marks and then run it:

```{r}
names <- c("Apple", "Orange", "Mango", "Pineapple")
```

Then create a vector of ages by adding four ages separated by commas to the code chunk below. Make sure you are inputting numeric values for the ages or you might get an error.

```{r}
age <- c(23, 32, 54 , 60)
```
With these two vectors, you can create a new data frame called `people`:

```{r}
people <- data.frame(names, age)
```
Now that you have this data frame, you can use some different functions to inspect it.

One common function you can use to preview the data is the `head()` function, which returns the columns and the first several rows of data. You can check out how the `head()` function works by running the chunk below:

```{r}
head(people)
```
In addition to `head()`, there are a number of other useful functions to summarize or preview your data. For example, the `str()` and `glimpse()` functions will both provide summaries of each column in your data arranged horizontally. You can check out these two functions in action by running the code chunks below:

```{r}
str(people)
```

```

```{r}
glimpse(people)
```
You can also use `colnames()` to get a list of the column names in your data set. Run the code chunk below to check out this function:

```{r}
colnames(people)
```
Now that you have a data frame, you can work with it using all of the tools in `R`. For example, you could use `mutate()` if you wanted to create a new variable that would capture each person's age in twenty years. The code chunk below creates that new variable:

```{r}
mutate(people, age_in_20 = age + 20)
```
<u>More about tibbles</u>

Tibbles

- Tibbles are a little different from standard data frames. A data frame is a collection of columns, like a spreadsheet or a SQL table. Tibbles are like streamlined data frames that are automatically set to pull up only the first 10 rows of a dataset, and only as many columns as can fit on the screen. This is really useful when you're working with large sets of data. Unlike data frames, tibbles never change the names of your variables, or the data types of your inputs. Overall, you can make more changes to data frames, but tibbles are easier to use. The tibble package is part of the core tidyverse. So, if you've already installed the tidyverse, you have what you need to start working with tibbles.

Creating tibbles

Now, let's go through an example of how to create a tibble in R. You can use the pre-loaded *diamonds* dataset that you're familiar with from earlier videos. As a reminder, the *diamonds* dataset includes information about different diamond qualities, like carat, cut, color, clarity, and more.

You can load the dataset with the data() function using the the following code:

library(tidyverse)

data(diamonds)
```

Then, let's add the data frame to our data viewer in RStudio with the View() function.

```
View(diamonds)
```

The dataset has 10 columns and thousands of rows. This image displays part of the data frame.

Now let's create a tibble from the same dataset. You can create a tibble from existing data with the **as_tibble()** function. Indicate what data you'd like to use in the parentheses of the function. In this case, you will use the word "diamonds."

```
as_tibble(diamonds)
```

When you run the function, you get a tibble of the *diamonds* dataset.

```
# A tibble: 53,940 x 10
   carat cut   color clarity depth table price     x     y     z
   <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1 0.23  Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
 2 0.21  Prem… E     SI1      59.8    61   326  3.89  3.84  2.31
 3 0.23  Good  E     VS1      56.9    65   327  4.05  4.07  2.31
 4 0.290 Prem… I     VS2      62.4    58   334  4.2   4.23  2.63
 5 0.31  Good  J     SI2      63.3    58   335  4.34  4.35  2.75
 6 0.24  Very… J     VVS2     62.8    57   336  3.94  3.96  2.48
 7 0.24  Very… I     VVS1     62.3    57   336  3.95  3.98  2.47
 8 0.26  Very… H     SI1      61.9    55   337  4.07  4.11  2.53
 9 0.22  Fair  E     VS2      65.1    61   337  3.87  3.78  2.49
10 0.23  Very… H     VS1      59.4    61   338  4     4.05  2.39
# … with 53,930 more rows
```

While RStudio's built-in data frame tool returns thousands of rows in the *diamonds* dataset, the tibble only returns the first 10 rows in a neatly organized table. That makes it easier to view and print.

Data-import basics

The readr package

In addition to using R's built-in datasets, it is also helpful to import data from other sources to use for practice or analysis. The readr package in R is a great tool for reading rectangular data. Rectangular data is data that fits nicely inside a rectangle of rows and columns, with each column referring to a single variable and each row referring to a single observation.

Here are some examples of file types that store rectangular data:

- **.csv (comma separated values):** a .csv file is a plain text file that contains a list of data. They mostly use commas to separate (or delimit) data, but sometimes they use other characters, like semicolons.
- **.tsv (tab separated values):** a .tsv file stores a data table in which the columns of data are separated by tabs. For example, a database table or spreadsheet data.
- **.fwf (fixed width files):** a .fwf file has a specific format that allows for the saving of textual data in an organized fashion.
- **.log:** a .log file is a computer-generated file that records events from operating systems and other software programs.

Base R also has functions for reading files, but the equivalent functions in readr are typically *much* faster. They also produce tibbles, which are easy to use and read.

The readr package is part of the core tidyverse. So, if you've already installed the tidyverse, you have what you need to start working with readr. If not, you can install the tidyverse now.

readr functions

The goal of readr is to provide a fast and friendly way to read rectangular data. readr supports several read_ functions. Each function refers to a specific file format.

- **read_csv()**: comma-separated values (.csv) files
- **read_tsv()**: tab-separated values files
- **read_delim()**: general delimited files
- **read_fwf()**: fixed-width files
- **read_table()**: tabular files where columns are separated by white-space
- **read_log()**: web log files

These functions all have similar syntax, so once you learn how to use one of them, you can apply your knowledge to the others. This reading will focus on the read_csv() function, since .csv files are one of the most common forms of data storage and you will work with them frequently.

In most cases, these functions will work automatically: you supply the path to a file, run the function, and you get a tibble that displays the data in the file. Behind the scenes, readr parses the overall file and specifies how each column should be converted from a character vector to the most appropriate data type.

Reading a .csv file with readr

The readr package comes with some sample files from built-in datasets that you can use for example code. To list the sample files, you can run the readr_example() function with no arguments.

readr_example()

[1] "challenge.csv" "epa78.txt" "example.log"

[4] "fwf-sample.txt" "massey-rating.txt" "mtcars.csv"

[7] "mtcars.csv.bz2" "mtcars.csv.zip"

The **"mtcars.csv"** file refers to the *mtcars* dataset that was mentioned earlier. Let's use the read_csv() function to read the **"mtcars.csv"** file, as an example. In the parentheses, you need to supply the path to the file. In this case, it's **"readr_example("mtcars.csv")**.

read_csv(readr_example("mtcars.csv"))

When you run the function, R prints out a column specification that gives the name and type of each column.

```
── Column specification ────────────────────────────────────────────────────────
cols(
  mpg = col_double(),
  cyl = col_double(),
  disp = col_double(),
  hp = col_double(),
  drat = col_double(),
  wt = col_double(),
  qsec = col_double(),
  vs = col_double(),
  am = col_double(),
  gear = col_double(),
  carb = col_double()
)
```

R also prints a tibble.

```
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1  21       6  160    110  3.9   2.62  16.5     0     1     4     4
 2  21       6  160    110  3.9   2.88  17.0     0     1     4     4
 3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
 4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
 5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
 6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
 7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
 8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
 9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# … with 22 more rows
```

Readxl package

To import spreadsheet data into R, you can use the readxl package. The readxl package makes it easy to transfer data from Excel into R. Readxl supports both the legacy .xls file format and the modern xml-based .xlsx file format.

The readxl package is part of the tidyverse but is not a *core* tidyverse package, so you need to load readxl in R by using the library() function.

library(readxl)

Reading a .csv file with readxl

Like the readr package, readxl comes with some sample files from built-in datasets that you can use for practice. You can run the code **readxl_example()** to see the list.

You can use the read_excel() function to read a spreadsheet file just like you used read_csv() function to read a .csv file. The code for reading the example file **"type-me.xlsx"** includes the path to the file in the parentheses of the function.

read_excel(readxl_example("type-me.xlsx"))

You can use the excel_sheets() function to list the names of the individual sheets.

excel_sheets(readxl_example("type-me.xlsx"))

[1] "logical_coercion" "numeric_coercion" "date_coercion" "text_coercion"

You can also specify a sheet by name or number. Just type "sheet =" followed by the name or number of the sheet. For example, you can use the sheet named "numeric_coercion" from the list above.

read_excel(readxl_example("type-me.xlsx"), sheet = "numeric_coercion")

When you run the function, R returns a tibble of the sheet.

```
# A tibble: 7 x 2
  `maybe numeric?` explanation
  <chr>            <chr>
1 NA               "empty"
2 TRUE             "boolean true"
3 FALSE            "boolean false"
4 40534            "datetime"
5 123456           "the string \"123456\""
6 123456           "the number 123456"
7 cabbage          "\"cabbage\""
```

Hand on activity: Importing and working with data

The scenario

In this scenario, you are a junior data analyst working for a hotel booking company. You have been asked to clean a .csv file that was created after querying a database to combine two different tables from different hotels. In order to learn more about this data, you are going to need to use functions to preview the data's structure, including its columns and rows. You will also need to use basic cleaning functions to prepare this data for analysis.

Step 1: Load packages

Start by installing your required package. If you have already installed and loaded `tidyverse` in this session, feel free to skip the code chunks in this step.

```{r}
install.packages("tidyverse")
```

Once a package is installed, we can load it by running the `library()` function with the package name inside the parentheses:

```{r}
library(tidyverse)
```

Step 2: Import data

One of the most common file types data analysts import into `R` is comma separated values files, or .csv files. The `tidyverse` library package `readr` has a number of functions for "reading in" or importing data, including .csv files and other external sources.

In the chunk below, use the `read_csv()` function to import data from a .csv in the project folder called "hotel_bookings.csv" and save it as a data frame called `bookings_df`.

If this line causes an error, copy in the line setwd("projects/Course 7/Week 3") before it.

The results will display as column specifications:
```{r}
bookings_df <- read_csv("hotel_bookings.csv")
```
Now that you have the `bookings_df`, you can work with it using all of the `R` functions you have learned so far.

Step 3: Inspect & clean data

One common function you can use to preview the data is the `head()` function, which returns the columns and first several rows of data. Check out the `head()` function by running the chunk below:

```{r}
head(bookings_df)
```
In addition to `head()` there are a number of other useful functions to summarize or preview your data frame. For example, the `str()` and function will provide summaries of each column in your data arranged horizontally. Check out the `str()` function by running the code chunk below:

```{r}
str(bookings_df)
```
To find out what columns you have in your data frame, try running the the `colnames()` function in the code chunk below:

```{r}
colnames(bookings_df)
```
If you want to create another data frame using `bookings_df` that focuses on the average daily rate, which is referred to as `adr` in the data frame, and `adults`, you can use the following code chunk to do that:

```{r}
new_df <- select(bookings_df, `adr`, adults)
new_df
```

To create new variables in your data frame, you can use the `mutate()` function. This will make changes to the data frame, but not to the original data set you imported. That source data will remain unchanged.

```{r}
mutate(new_df, total = `adr` / adults)
```

<u>Cleaning data</u>

Cleaning up with the basics

- **Here**
- **Skimr:** Makes summarizing data easy and allows you to skim through it more quickly.
- **Janitor:** Provides functions for data cleaning
- Some functions to get summary of the data:
    - **Skim_without_charts()**
    - **glimpse()**
    - **head()**
    - **select():** Used to select a subset of variables from a large dataset (particular columns). To exclude a single column and keep all the others, use **-**
- To rename a column name, use **rename()**
- To make columns consistent, use **rename_with()**
- To ensure there are only characters, numbers, and underscores in the column names, use **clean_names()**

Operators

In R, there are four main types of operators:

1. Arithmetic
2. Relational
3. Logical
4. Assignment

Logical operators

Logical operators allow you to combine logical values. Logical operators return a logical data type or boolean (TRUE or FALSE). As a refresher, we encountered logical operators in an earlier reading, “Logical Operators and Conditional Statements,” but let’s review them again.

The table below summarizes the logical operators in R.

Operator	Description
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR
!	Logical NOT

Element-wise logical AND (&) and OR (|)

You can illustrate logical AND (&) and OR (|) by comparing numerical values. Let's create a variable *x* that is equal to 10.

```
x <- 10
```

The AND operator returns TRUE only if *both* individual values are TRUE.

```
x > 2 & x < 12
```

[1] TRUE

10 is greater than 2 *and* 10 is less than 12. So, the operation evaluates to **TRUE**.

The OR operator (|) works in a similar way to the AND operator (&). The main difference is that just *one* of the values of the OR operation needs to be TRUE for the entire OR operation to evaluate to TRUE. Only if *both* values are FALSE will the entire OR operation evaluate to FALSE.

Let's try an example with the same variable (`x <- 10`):

```
x > 2 | x < 8
```

```
[1] TRUE
```

10 is greater than 2, but 10 is not less than 8. But since at least one of the values (10>2) is TRUE, the OR operation evaluates to TRUE.

Logical AND (&&) and OR (||)

The main difference between element-wise logical operators (&, |) and logical operators (&&, ||) is the way they apply to operations with vectors. The operations with double signs, AND (&&) and logical OR (||), only examine the *first* element of each vector. The operations with single signs, AND (&) and OR (|), examine all the elements of each vector.

For example, imagine you are working with two vectors that each contain three elements: **c(3, 5, 7)** and **c(2, 4, 6)**. The logical AND (&) will compare the first element of the first vector with the first element of the second vector (3&2), the second element with the second element (5&4), and the third element with the third element (7&6).

Let's check out this example in R code.

First, create two variables, *x* and *y*, to store the two vectors:

```
x <- c(3, 5, 7)
y <- c(2, 4, 6)
```

Then run the code with a single ampersand (&). The output is boolean (TRUE or FALSE).

```
x < 5 & y < 5
```

```
[1]  TRUE FALSE FALSE
```

When you compare each element of the two vectors, the output is TRUE, FALSE, FALSE. The first element of both *x* (3) and *y* (2) is less than 5, so this is TRUE. The second element of x is *not* less than 5 (it's equal to 5) but the second element of y is less than 5, so this is FALSE (because we used AND). The third element of both x and y is not less than 5, so this is also FALSE.

Now, let's run the same operation using the double ampersand (&&):

```
x < 5 && y < 5
```

```
[1] TRUE
```

In this case, R only compares the *first* elements of each vector: 3 and 2. So, the output is **TRUE** because 3 and 2 are both less than 5.
Depending on the type of work you do, you might make use of single sign operators more often than double sign operators. But it is helpful to know how all of the operators work regardless.

Organize your data

- To organize data:
    - **arrange():** To sort data in the columns. Default is in ascending order, but for descending order use **-** in front of column name
    - **group_by():** Group different rows together. Use **summarize()** function to perform aggregate functions
    - **filter()**

Hands-on activity: Cleaning data with R

The scenario

In this scenario, you are a junior data analyst working for a hotel booking company. You have been asked to clean a .csv file that was created after querying a database to combine two different tables from different hotels. In order to learn more about this data, you are going to need to use functions to preview the data's structure, including its columns and rows. You will also need to use basic cleaning functions to prepare this data for analysis.

Step 1: Load packages

In order to start cleaning your data, you will need to by install the required packages. If you have already installed and loaded `tidyverse`, `skimr`, and `janitor` in this session, feel free to skip the code chunks in this step.

```{r}
install.packages("tidyverse")
install.packages("skimr")
install.packages("janitor")
```

Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses:

```{r}
library(tidyverse)
library(skimr)
library(janitor)
```

Step 2: Import data

The data you have been asked to clean is currently an external .csv file. In order to view and clean it in `R`, you will need to import it. The `tidyverse` library `readr` package has a number of functions for "reading in" or importing data, including .csv files.

In the chunk below, you will use the `read_csv()` function to import data from a .csv file in the project folder called "hotel_bookings.csv" and save it as a data frame called `bookings_df`:

If this line causes an error, copy in the line setwd("projects/Course 7/Week 3") before it.

```{r}
bookings_df <- read_csv("hotel_bookings.csv")
```

Step 3: Getting to know your data

Before you start cleaning your data, take some time to explore it. You can use several functions that you are already familiar with to preview your data, including the `head()` function in the code chunk below:

```{r}
head(bookings_df)
```

You can also summarize or preview the data with the `str()` and `glimpse()` functions to get a better understanding of the data by running the code chunks below:

```{r}
str(bookings_df)
```

```{r}
glimpse(bookings_df)
```

You can also use `colnames()` to check the names of the columns in your data set. Run the code chunk below to find out the column names in this data set:

```{r}
colnames(bookings_df)
```

Some packages contain more advanced functions for summarizing and exploring your data. One example is the `skimr` package, which has a number of functions for this purpose. For example, the `skim_without_charts()` function provides a detailed summary of the data. Try running the code below:

```{r}
skim_without_charts(bookings_df)
```

Step 4: Cleaning your data

Based on the functions you have used so far, how would you describe your data in a brief to your stakeholder? Now, let's say you are primarily interested in the following variables: 'hotel',

'is_canceled', and 'lead_time'. Create a new data frame with just those columns, calling it `trimmed_df` by adding the variable names to this code chunk:

```{r}
trimmed_df <- bookings_df %>%
  select(hotel ,is_canceled ,lead_time )
trimmed_df
```

You might notice that some of the column names aren't very intuitive, so you will want to rename them to make them easier to understand. You might want to create the same exact data frame as above, but rename the variable 'hotel' to be named 'hotel_type' to be crystal clear on what the data is about

Fill in the space to the left of the '=' symbol with the new variable name:

```{r}
trimmed_df %>%
  select(hotel, is_canceled, lead_time) %>%
  rename(hotel_type = hotel)
```

Another common task is to either split or combine data in different columns. In this example, you can combine the arrival month and year into one column using the **unite()** function:

```{r}
example_df <- bookings_df %>%
  select(arrival_date_year, arrival_date_month) %>%
  unite(arrival_month_year, c("arrival_date_month", "arrival_date_year"), sep = " ")
example_df
```

Step 5: Another way of doing things

You can also use the`mutate()` function to make changes to your columns. Let's say you wanted to create a new column that summed up all the adults, children, and babies on a reservation for the total number of people. Modify the code chunk below to create that new column:

```{r}
example_df <- bookings_df %>%
  mutate(guests = adults+children+babies)

head(example_df)
```

Great. Now it's time to calculate some summary statistics! Calculate the total number of canceled bookings and the average lead time for booking - you'll want to start your code after the %>% symbol. Make a column called 'number_canceled' to represent the total number of canceled bookings. Then, make a column called 'average_lead_time' to represent the average lead time. Use the `summarize()` function to do this in the code chunk below:

```{r}

example_df <- bookings_df %>%
  summarize(number_canceled=sum(is_canceled), average_lead_time=mean(lead_time))


head(example_df)
```

Transforming data

- **separate():** Used to separate the columns based on the values
- **unite():** Used to combine the columns
- **mutate():** Used to create new columns

Wide to Long with tidyr

When organizing or tidying your data using R, you might need to convert wide data to long data or long to wide. Recall that this is what data in a wide format looks like in a spreadsheet:

	A	B	C	D	E	F	G	H	
1	Country Name	2010 [YR2010]	2011 [YR2011]	2012 [YR2012]	2013 [YR2013]	2014 [YR2014]	2015 [YR2015]	2016 [YR2016]	20
2	Antigua and Barb	88028	89253	90409	91516	92562	93566	94527	
3	Argentina	40788453	41261490	41733271	42202935	42669500	43131966	43590368	
4	Aruba	101669	102046	102560	103159	103774	104341	104872	
5	Bahamas, The	354942	359577	363584	367168	370633	374206	377931	
6	Barbados	282131	282987	283700	284296	284825	285324	285796	
7	Belize	322464	330237	338000	345715	353366	360933	368400	

Wide data has observations across several columns. Each column contains data from a different condition of the variable. In this example, different years.

Now check out the same data in a long format:

	A	B	C
1	Country Name	Year	Population
2	Antigua and Barb	2010	88028
3	Antigua and Barb	2011	89253
4	Antigua and Barb	2012	90409
5	Antigua and Barb	2013	91516
6	Antigua and Barb	2014	92562
7	Antigua and Barb	2015	93566
8	Antigua and Barb	2016	94527
9	Antigua and Barb	2017	95426
10	Antigua and Barb	2018	96286
11	Antigua and Barb	2019	97118
12	Argentina	2010	40788453

And, to review what you already learned about the difference, **long data** has all the observations in a single column, and variables in separate columns.

The pivot_longer() and pivot_wider() functions

There are compelling reasons to use both formats. But as an analyst, it is important to know how to tidy data when you need to. In R, you may have a data frame in a wide format that has several variables and conditions for each variable. It might feel a bit messy.

That's where **pivot_longer()** comes in. As part of the tidyr package, you can use this R function to lengthen the data in a data frame by increasing the number of rows and decreasing the number of columns. Similarly, if you want to convert your data to have more columns and fewer rows, you would use the **pivot_wider()** function.

Take a closer look at the data

Same data, different outcome

- **Anscombe's quartet:** Four datasets that have nearly identical summary statistics, but those summary statistics might be misleading. Data visualizations, especially for datasets like these, are so important. They help us discover things in our data that would otherwise remain hidden.

Working with biased data

The bias function **bias()** compares the actual outcome of the data with the predicted outcome to determine whether or not the model is biased.

Every data analyst will encounter an element of bias at some point in the data analysis process. That's why it's so important to understand how to identify and manage biased data whenever possible.

Addressing biased data with R

This scenario was shared by a quantitative analyst who collects data from people all over the world. They explain how they discovered bias in their data, and how they used R to address it:

"I work on a team that collects survey-like data. One of the tasks my team does is called a side-by-side comparison. For example, we might show users two ads side-by-side at the same time. In our survey, we ask which of the two ads they prefer. In one case, after many iterations, we were seeing consistent bias in favor of the first item. There was also a measurable decrease in the preference for an item if we swapped its position to second.

So we decided to add randomization to the position of the ads using R. We wanted to make sure that the items appeared in the first and second positions with similar frequencies. We used sample() to inject a randomization element into our R programming. In R, the sample() function allows you to take a random sample of elements from a data set. Adding this piece of code shuffled the rows in our data set randomly. So when we presented the ads to users, the positions of the ads were now random and controlled for bias. This made the survey more effective and the data more reliable."

The sample() function is just one of many functions and methods in R that you can use to address bias in your data. Depending on the kind of analysis you are conducting, you might need to incorporate some advanced processes in your programming.

Hands-in activity: Changing your data

The Scenario
In this scenario, you are a junior data analyst working for a hotel booking company. You have been asked to clean a .csv file that was created after querying a database to combine two different tables from different hotels. You have already performed some basic cleaning functions on this data; this activity will focus on using functions to conduct basic data manipulation.

Step 1: Load packages

Start by installing the required packages. If you have already installed and loaded `tidyverse`, `skimr`, and `janitor` in this session, feel free to skip the code chunks in this step. This may take a few minutes to run, and you may get a pop-up window asking if you want to proceed. Click yes to continue installing the packages.

```{r install packages}
install.packages("tidyverse")
install.packages("skimr")
install.packages("janitor")
```

Once a package is installed, you can load it by running the `library()` function with the package name inside the parentheses:

```{r load packages}
library(tidyverse)
library(skimr)
library(janitor)
```

Step 2: Import data

In the chunk below, you will use the `read_csv()` function to import data from a .csv in the project folder called "hotel_bookings.csv" and save it as a data frame called `hotel_bookings`.

If this line causes an error, copy in the line setwd("projects/Course 7/Week 3") before it.

Type the file name in the correct place to read it into your R console:

```{r load dataset}
hotel_bookings <- read_csv("")
```

Step 3: Getting to know your data

Like you have been doing in other examples, you are going to use summary functions to get to know your data. This time, you are going to complete the code chunks below in order to use these different functions. You can use the `head()` function to preview the columns and the first several rows of data. Finish the code chunk below and run it:

```{r head function}
head(hotel_bookings)
```

Now you know this dataset contains information on hotel bookings. Each booking is a row in the dataset, and each column contains information such as what type of hotel was booked, when the booking took place, and how far in advance the booking took place (the 'lead_time' column).

In addition to `head()` you can also use the `str()` and `glimpse()` functions to get summaries of each column in your data arranged horizontally. You can try these two functions by completing and running the code chunks below:

```{r str function}
str(hotel_bookings)
```

You can see the different column names and some sample values to the right of the colon.

```{r glimpse function}
glimpse(hotel_bookings)
```

You can also use `colnames()` to get the names of the columns in your dataset. Run the code chunk below to get the column names:

```{r colnames function}
colnames(hotel_bookings)
```

Manipulating your data

Let's say you want to arrange the data by most lead time to least lead time because you want to focus on bookings that were made far in advance. You decide you want to try using the `arrange()` function; input the correct column name after the comma and run this code chunk:

```{r arrange function}
arrange(hotel_bookings, -lead_time)
```

Now it is in the order you need. You can click on the different pages of results to see additional rows of data, too.

Notice that when you just run `arrange()` without saving your data to a new data frame, it does not alter the existing data frame. Check it out by running `head()` again to find out if the highest lead times are first:

```{r head function part two}
head(hotel_bookings)
```

This will be true of all the functions you will be using in this activity. If you wanted to create a new data frame that had those changes saved, you would use the assignment operator, <- , as written in the code chunk below to store the arranged data in a data frame named 'hotel_bookings_v2':

```{r new dataframe}
hotel_bookings_v2 <-
  arrange(hotel_bookings, desc(lead_time))
```

Run `head()`to check it out:

```{r new dataframe part two}
head(hotel_bookings_v2)
```

You can also find out the maximum and minimum lead times without sorting the whole dataset using the `arrange()` function. Try it out using the max() and min() functions below:

```{r}
**max(hotel_bookings$lead_time)**
```

```{r}
**min(hotel_bookings$lead_time)**
```

Remember, in this case, you need to specify which dataset and which column using the $ symbol between their names. Try running the below to see what happens if you forget one of those pieces:

```{r}
min(lead_time)
```
This is a common error that R users encounter. To correct this code chunk, you will need to add the data frame and the dollar sign in the appropriate places.

Now, let's say you just want to know what the average lead time for booking is because your boss asks you how early you should run promotions for hotel rooms. You can use the `mean()` function to answer that question since the average of a set of number is also the mean of the set of numbers:

```{r mean}
mean(hotel_bookings$lead_time)
```
You should get the same answer even if you use the v2 dataset that included the `arrange()` function. This is because the `arrange()` function doesn't change the values in the dataset; it just re-arranges them.

```{r mean part two}
mean(hotel_bookings_v2$lead_time)
```
You were able to report to your boss what the average lead time before booking is, but now they want to know what the average lead time before booking is for just city hotels. They want to focus the promotion they're running by targeting major cities.

You know that your first step will be creating a new dataset that only contains data about city hotels. You can do that using the `filter()` function, and name your new data frame 'hotel_bookings_city':

```{r filter}
 <-
  filter(hotel_bookings, hotel_bookings$hotel=="City Hotel")
```
Check out your new dataset:

```{r new dataset}
head(hotel_bookings_city)
```
You quickly check what the average lead time for this set of hotels is, just like you did for all of hotels before:

```{r average lead time city hotels}
mean(hotel_bookings_city$lead_time)
```
Now, your boss wants to know a lot more information about city hotels, including the maximum and minimum lead time. They are also interested in how they are different from resort hotels. You don't want to run each line of code over and over again, so you decide to use the `group_by()`and`summarize()` functions. You can also use the pipe operator to make your code easier to follow. You will store the new dataset in a data frame named 'hotel_summary':

```{r group and summarize}
hotel_summary <-
  hotel_bookings %>%
  group_by(hotel) %>%
  summarise(average_lead_time=mean(lead_time),
            min_lead_time=min(lead_time),
            max_lead_time=max(lead_time))
```
Check out your new dataset using head() again:

```{r}
head(hotel_summary)
```
Week-4

Create Data Visualizations in R

Visualizations basic in R and tidyverse

Some of the visualization packages:

- ggplot2 (gg means grammar of graphics)
- Plotly
- Lattice
- RGL
- Dygraphs
- Leaflet
- Highcharter
- Patchwork
- gganimate
- ggridges

The ggplot2 package lets you make high quality, customizable plots of your data. As a refresher, ggplot2 is based on the grammar of graphics, which is a system for describing and building data visualizations. The essential idea behind the grammar of graphics is that you can build any plot from the same basic components, like building blocks.

These building blocks include:

- A dataset
- A set of geoms: A geom refers to the geometric object used to represent your data. For example, you can use points to create a scatter plot, bars to create a bar chart, lines to create a line diagram, etc.
- A set of aesthetic attributes: An aesthetic is a visual property of an object in your plot. You can think of an aesthetic as a connection, or mapping, between a visual feature in your plot and a variable in your data. For example, in a scatterplot, aesthetics include things like the size, shape, color, or location (x-axis, y-axis) of your data points.

To create a plot with ggplot2, you first choose a dataset. Then, you determine how to visually organize your data on a coordinate system by choosing a geom to represent your data points and aesthetics to map your variables.

Benefits of ggplot2:

- Create different types of plots
- Customize the look and feel of plots
- Create high quality visuals
- Combine data manipulation and visualization

Core concepts in ggplot2:

- Aesthetics: A visual property of an object in your plot
- Geoms: The geometric object used to represent your data
- Facets: Let you display smaller groups, or subsets, of your data
- Labels and annotations: Let you customize your plot

<u>Getting started with ggplot2:</u>

- To construct a ggplot function, follow the below sequence:
    - Start with the ggplot function and choose a dataset to work with
    - Add a geom_function to display your data
    - Map the variables you want to plot in the argument of the aes function

  Example: ggplot(data=penguins)+geom_point(mapping=aes(x=fliiper_length_mm, y=body_mas_g))

Hand-on activity: Using ggplot

The Scenario

In this scenario, you are a junior data analyst working for a hotel booking company. You have cleaned and manipulated your data, and gotten some initial insights you would like to share. Now, you are going to create some simple data visualizations with the `ggplot2` package. You will use basic `ggplot2` syntax and troubleshoot some common errors you might encounter.

Step 1: Import your data

In the chunk below, you will use the `read_csv()` function to import data from a .csv in the project folder called "hotel_bookings.csv" and save it as a data frame called `hotel_bookings`:

If this line causes an error, copy in the line setwd("projects/Course 7/Week 4") before it.

```{r load data}
hotel_bookings <- read.csv("hotel_bookings.csv")
```
Step 2: Look at a sample of your data

Use the `head()` function to preview your data:

```{r examining your data}
head(hotel_bookings)
```
You can also use `colnames()` to get the names of all the columns in your data set. Run the code chunk below to find out the column names in this data set:

```{r look at column names}
colnames(hotel_bookings)
```
Step 3: Install and load the 'ggplot2' package

If you haven't already installed and loaded the `ggplot2` package, you will need to do that before you can use the `ggplot()` function.

Run the code chunk below to install and load `ggplot2`. This may take a few minutes.

```{r loading and installing ggplot2, echo=FALSE, message=FALSE}
install.packages('ggplot2')
library(ggplot2)
```
Step 4: Begin creating a plot

A stakeholder tells you, "I want to target people who book early, and I have a hypothesis that people with children have to book in advance."

When you start to explore the data, it doesn't show what you would expect. That is why you decide to create a visualization to see how true that statement is-- or isn't.

You can use `ggplot2` to do this. Try running the code below:

```{r creating a plot}
ggplot(data = hotel_bookings) +
  geom_point(mapping = aes(x = lead_time, y = children))
```

The geom_point() function uses points to create a scatterplot. Scatterplots are useful for showing the relationship between two numeric variables. In this case, the code maps the variable 'lead_time' to the x-axis and the variable 'children' to the y-axis.

On the x-axis, the plot shows how far in advance a booking is made, with the bookings furthest to the right happening the most in advance. On the y-axis it shows how many children there are in a party.

The plot reveals that your stakeholder's hypothesis is incorrect. You report back to your stakeholder that many of the advanced bookings are being made by people with 0 children.

Step 5: Try it on your own

Next, your stakeholder says that she wants to increase weekend bookings, an important source of revenue for the hotel. Your stakeholder wants to know what group of guests book the most weekend nights in order to target that group in a new marketing campaign. She suggests that guests without children book the most weekend nights. Is this true?

Try mapping 'stays_in_weekend_nights' on the x-axis and 'children' on the y-axis by filling out the remainder of the code below.

```{r}
ggplot(data = hotel_bookings) +
 geom_point(mapping = aes(x =children , y = stays_in_weekend_nights))
```

If you correctly enter this code, you should have a scatterplot with 'stays_in_weekend_nights' on the x-axis and 'children' on the y-axis.
What did you discover? Is your stakeholder correct?
What other types of plots could you use to show this relationship?
Remember, if you're having trouble filling out a code block, check the solutions document for this activity.

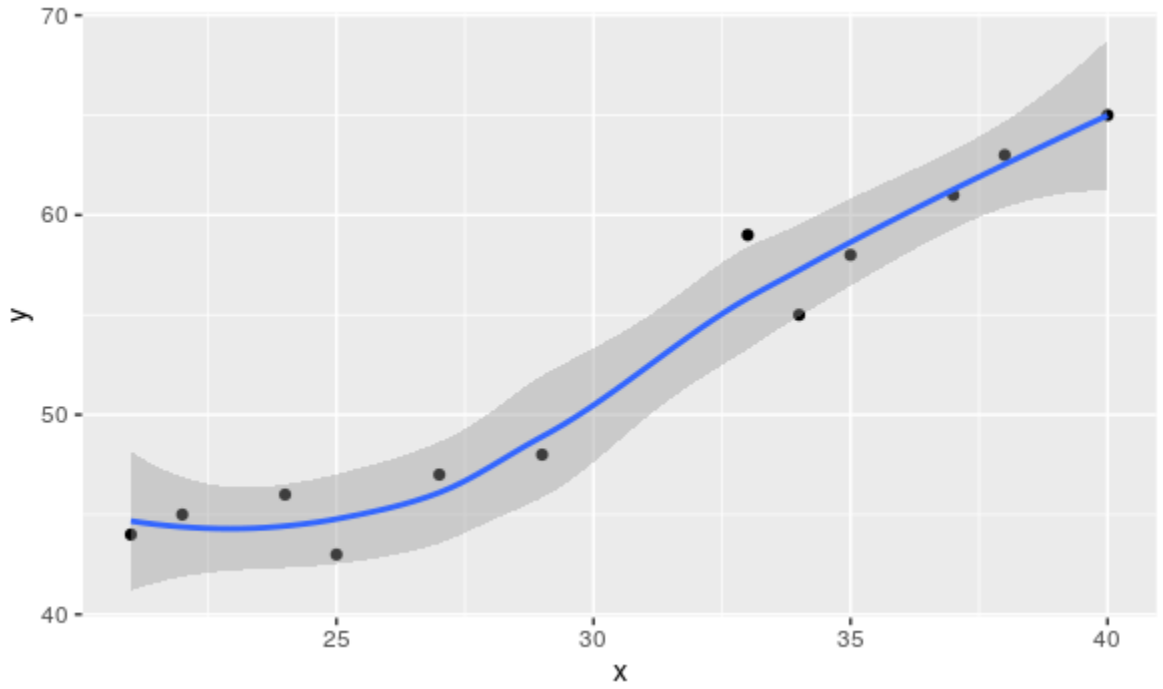
Smoothing

Sometimes it can be hard to understand trends in your data from scatter plots alone. **Smoothing** enables the detection of a data trend even when you can't easily notice a trend from the plotted data points. Ggplot2's smoothing functionality is helpful because it adds a smoothing line as another layer to a plot; the **smoothing line** helps the data to make sense to a casual observer.

Example code

```
ggplot(data, aes(x=distance,
y= dep_delay)) +
   geom_point() +
   geom_smooth()
```

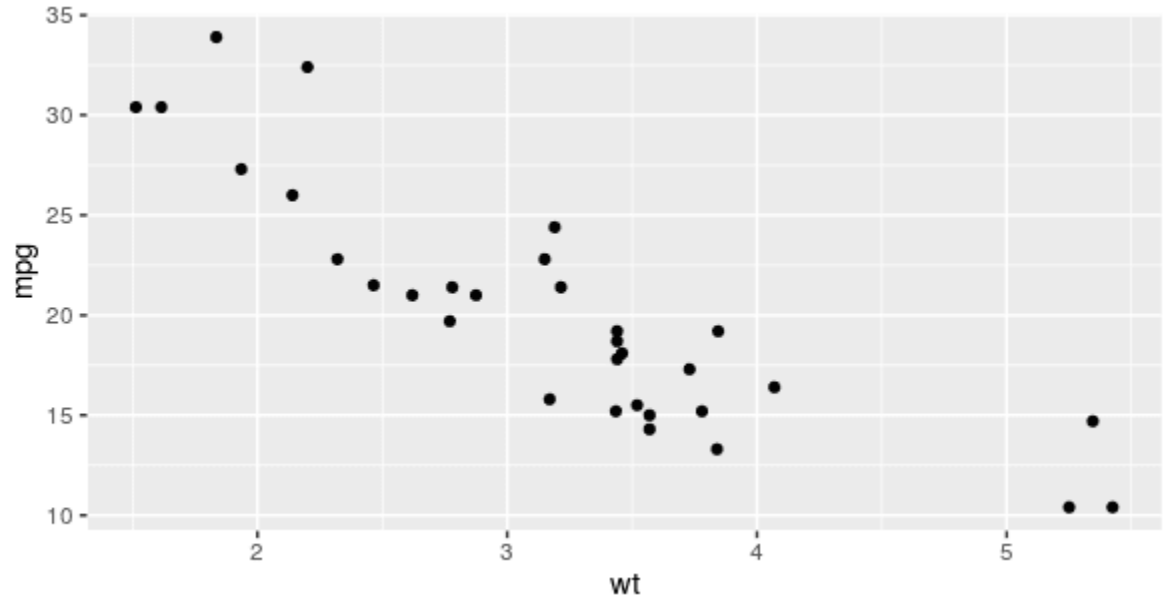
The example code creates a plot with a trend line similar to the blue line below.

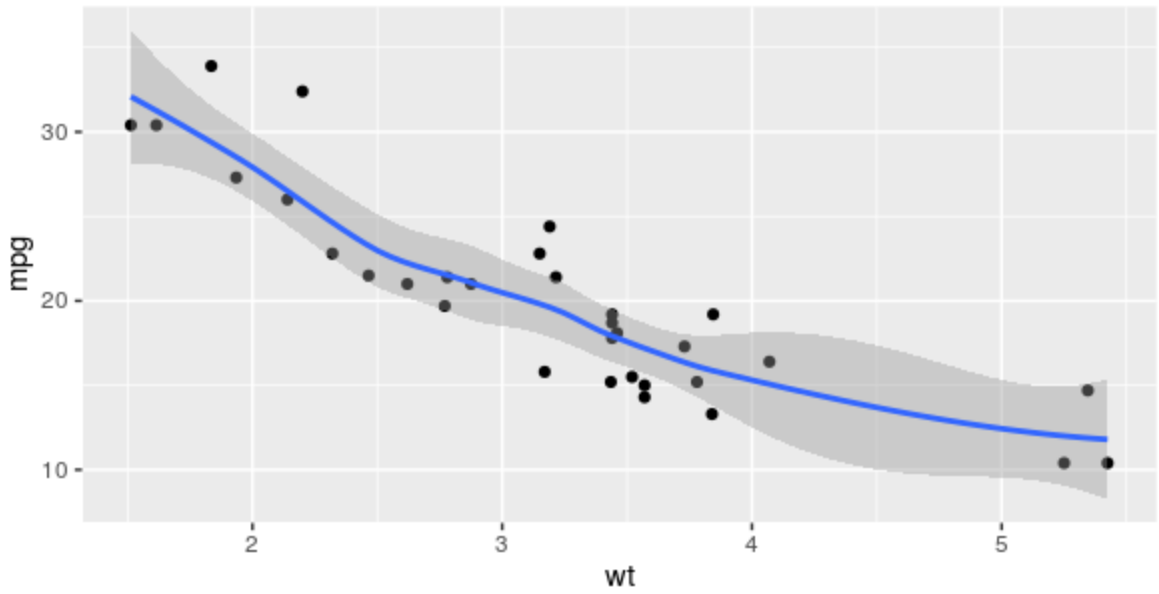


Two types of smoothing

Type of smoothing	Description	Example code
Loess smoothing	The loess smoothing process is best for smoothing plots with less than 1000 points.	`ggplot(data, aes(x=, y=))+ geom_point() + geom_smooth(method="loess")`
Gam smoothing	Gam smoothing, or generalized additive model smoothing, is useful for smoothing plots with a large number of points.	`ggplot(data, aes(x=, y=)) + geom_point() + geom_smooth(method="gam", formula = y ~s(x))`

The smoothing functionality in ggplot2 helps make data plots more readable, so you are better able to recognize data trends and make key insights. The first plot below is the data before smoothing, and the second plot below is the same data after smoothing.





Aesthetics and facets

- Facet functions
    - facet_wrap(): For one variable
    - facet_grid(): For two or more variables

Hands-on activity: Aesthetics and Visualizations

The Scenario
In this example, you are a junior data analyst working for the same hotel booking company from earlier. Last time, you created some simple visualizations with `ggplot2` to give your stakeholders quick insights into your data. Now, you are interested in creating visualizations that highlight different aspects of the data to present to your stakeholder. You are going to expand on what you have already learned about `ggplot2` and create new kinds of visualizations like bar charts.

Step 1: Import your data
If you haven't exited RStudio since importing this data last time, you can skip these steps. Rerunning these code chunks won't affect your console if you want to run them just in case, though.

Run the code below to read in the file 'hotel_bookings.csv' into a data frame:

If this line causes an error, copy in the line setwd("projects/Course 7/Week 4") before it.

```{r load data}
hotel_bookings <- read.csv("hotel_bookings.csv")
```

Step 2: Refresh Your Memory

By now, you are pretty familiar with this data set. But you can refresh your memory with the `head()` and `colnames()` functions. Run two code chunks below to get at a sample of the data and also preview all the column names:

```{r look at data}
head(hotel_bookings)
```

```{r look at column names}
colnames(hotel_bookings)
```

Step 3: Install and load the 'ggplot2' package (optional)

If you haven't already installed and loaded the `ggplot2` package, you will need to do that before you can use the `ggplot()` function. You only have to do this once though, not every time you call `ggplot()`.

You can also skip this step if you haven't closed your RStudio account since doing the last activity. If you aren't sure, you can run the code chunk and hit 'cancel' if the warning message pops up telling you that have already downloaded the `ggplot2` package.

Run the code chunk below to install and load `ggplot2`. This may take a few minutes!

```{r loading and installing ggplot2, echo=FALSE, message=FALSE}
install.packages('ggplot2')
library(ggplot2)
```

Step 4: Making a Bar Chart

Your stakeholder is interested in developing promotions based on different booking distributions, but first they need to know how many of the transactions are occurring for each different distribution type.

You can tell `ggplot()` what type of chart you want to create by using the `geom_` argument.

Previously, you used `geom_point` to make a scatter plot comparing lead time and number of children. Now, you will use `geom_bar` to make a bar chart in this code chunk:

```{r example chart}
ggplot(data = hotel_bookings) +
```

```
  geom_bar(mapping = aes(x = distribution_channel))
```

Step 5: Diving deeper into bar charts

After exploring your bar chart, your stakeholder has more questions. Now they want to know if the number of bookings for each distribution type is different depending on whether or not there was a deposit or what market segment they represent.

Try modifying the code below to answer the question about deposits by adding 'fill=deposit_type' after 'x = distribution_channel':

```{r pressure, echo=FALSE}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel, fill=deposit_type))
```

This code chunk also creates a bar chart with 'distribution_channel' on the x-axis and 'count' on the y axis. But it also includes data from 'deposit_type' column as color-coded sections of each bar. There is a legend explaining what each color represents on the right side of the visualization.

Now try adding 'fill=market_segment' to this code chunk instead of 'fill=deposit_type':

```{r pressure, echo=FALSE}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel, fill=market_segment))
```

This bar chart is similar to the previous chart, except that 'market_segment' data is being recorded in the color-coded sections of each bar.

Step 6: Facets galore

After reviewing the new charts, your stakeholder asks you to create separate charts for each deposit type and market segment to help them understand the differences more clearly.

You know that the `facet_` function can do this very quickly.

Add 'deposit_type' after the '~' symbol in the code chunk below to create a different chart for each deposit type:

```{r creating a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~deposit_type)
```

This code chunk creates three bar charts for 'no_deposit', non_refund', and 'refundable' deposit types. You notice that it's hard to read the x-axis labels here, so you add one piece of code at the end that rotates the text to 45 degrees to make it easier to read.

Try it out below:

```{r creating a plot with rotated labels}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~deposit_type) +
  theme(axis.text.x = element_text(angle = 45))
```

This code chunk creates a similar bar chart to the previous chunk, but now the labels on the x axis with the different distribution channels are clearer.

You can use the same syntax to create a different chart for each market segment:

```{r creating a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~market_segment) +
  theme(axis.text.x = element_text(angle = 45))
```

The `facet_grid` function does something similar. The main difference is that **`facet_grid` will include plots even if they are empty.** Run the code chunk below to check it out:

```{r creating a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_grid(~deposit_type) +
  theme(axis.text.x = element_text(angle = 45))
```

You should have three bar charts.

Now, you could put all of this in one chart and explore the differences by deposit type and market segment.

Run the code chunk below to find out; **notice how the ~ character is being used before the variables** that the chart is being split by:

```{r creating a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = distribution_channel)) +
  facet_wrap(~deposit_type~market_segment) +
  theme(axis.text.x = element_text(angle = 45))
```

These charts are probably overwhelming and too hard to read, but it can be useful if you are exploring your data through visualizations.

Filtering and Plotting

By this point you have likely downloaded at least a few packages into your R library. The tools in some of these packages can actually be combined and used together to become even more useful. This reading will share a few resources that will teach you how to use the filter function from **dplyr** to make the plots you create with **ggplot2** easier to read.

Example of filtering and plotting

Filtering your data before you plot it allows you to focus on specific subsets of your data and gain more targeted insights. To do this, just include the dplyr filter() function in your ggplot syntax.

Example code

```
data %>%
  filter(variable1 == "DS") %>%
  ggplot(aes(x = weight, y = variable2, colour = variable1)) +
  geom_point(alpha = 0.3,  position = position_jitter()) + stat_smooth(method = "lm")
```

Hands-on activity: Filters and Plots

The Scenario

As a junior data analyst for a hotel booking company, you have been asked to clean hotel booking data, create visualizations with `ggplot2` to gain insight into the data, and present different facets of the data through visualization. Now, you are going to build on the work you performed previously to apply filters to your data visualizations in `ggplot2`.

Step 1: Import your data

If you haven't exited RStudio since importing this data last time, you can skip these steps. Rerunning these code chunks won't affect your console if you want to run them just in case, though.

If this line causes an error, copy in the line setwd("projects/Course 7/Week 4") before it.

Run the code below to read in the file 'hotel_bookings.csv' into a data frame:

```{r load data}
hotel_bookings <- read.csv("hotel_bookings.csv")
```

Step 2: Refresh Your Memory

By now, you are pretty familiar with this data set. But you can refresh your memory with the `head()` and `colnames()` functions. Run two code chunks below to get at a sample of the data and also preview all the column names:

```{r look at data}
head(hotel_bookings)
```

```{r look at column names}
colnames(hotel_bookings)
```

Step 3: Install and load the 'ggplot2' package (optional)

If you haven't already installed and loaded the `ggplot2` package, you will need to do that before you can use the `ggplot()` function. You only have to do this once though, not every time you call `ggplot()`.

You can also skip this step if you haven't closed your RStudio account since doing the last activity. If you aren't sure, you can run the code chunk and hit 'cancel' if the warning message pops up telling you that you have already downloaded the `ggplot2` package.

Run the code chunk below to install and load `ggplot2`. This may take a few minutes!

```{r loading and installing ggplot2, echo=FALSE, message=FALSE}
install.packages('ggplot2')
library(ggplot2)
```

Step 4: Making many different charts

Earlier, you created a scatterplot to explore the relationship between booking lead time and guests traveling with children. As a refresher, here's the code:

```{r scatterplot}
ggplot(data = hotel_bookings) +
  geom_point(mapping = aes(x = lead_time, y = children))
```

```
Your stakeholder asked about the group of guests who typically make early bookings, and this plot showed that many of these guests do not have children.

Now, your stakeholder wants to run a family-friendly promotion targeting key market segments. She wants to know which market segments generate the largest number of bookings, and where these bookings are made (city hotels or resort hotels).

First, you decide to create a bar chart showing each hotel type and market segment. You use different colors to represent each market segment:

```{r bar chart}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = hotel, fill = market_segment))
```
The geom_bar() function uses bars to create a bar chart. The chart has 'hotel' on the x-axis and 'count' on the y-axis (note: if you don't specify a variable for the y-axis, the code defaults to 'count'). The code maps the 'fill' aesthetic to the variable 'market_segment' to generate color-coded sections inside each bar.

After creating this bar chart, you realize that it's difficult to compare the size of the market segments at the top of the bars. You want your stakeholder to be able to clearly compare each segment.

You decide to use the facet_wrap() function to create a separate plot for each market segment. In the parentheses of the facet_wrap() function, add the variable 'market_segment' after the tilde symbol (~):

```{r faceting a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = hotel)) +
  facet_wrap(~market_segment)
```
Now you have a separate bar chart for each market segment. Your stakeholder has a clearer idea of the size of each market segment, as well as the corresponding data for each hotel type.

Step 5: Filtering

For the next step, you will need to have the `tidyverse` package installed and loaded. You may see a pop-up asking if you want to install; if that's the case, click 'Install.' This may take a few minutes!

If you have already done this because you're using the `tidyverse` package on your own, you can skip this code chunk.
```

```{r install and download tidyverse}
install.packages('tidyverse')
library(tidyverse)
```
After considering all the data, your stakeholder decides to send the promotion to families that make online bookings for city hotels. The online segment is the fastest growing segment, and families tend to spend more at city hotels than other types of guests.

Your stakeholder asks if you can create a plot that shows the relationship between lead time and guests traveling with children for online bookings at city hotels. This will give her a better idea of the specific timing for the promotion.

You think about it, and realize you have all the tools you need to fulfill the request. You break it down into the following two steps: 1) filtering your data; 2) plotting your filtered data.

For the first step, you can use the `filter()` function to create a data set that only includes the data you want. Input 'City Hotel' in the first set of quotation marks and 'Online TA' in the second set of quotations marks to specify your criteria:

```{r filtering a dataset to just city hotels that are online TA}
onlineta_city_hotels <- filter(hotel_bookings,
                       (hotel=="City Hotel" &
                         hotel_bookings$market_segment=="Online TA"))
```

Note that you can use the '&' character to demonstrate that you want two different conditions to be true. Also, you can use the '$' character to specify which column in the data frame 'hotel_bookings' you are referencing (for example, 'market_segment').

You can use the`View`() function to check out your new data frame:

```{r View}
View(onlineta_city_hotels)
```
There is also another way to do this. You can use the pipe operator (%>%) to do this in steps!

You name this data frame `onlineta_city_hotels_v2`:

```{r filtering a dataset with the pipe}
onlineta_city_hotels_v2 <- hotel_bookings %>%
  filter(hotel=="City Hotel") %>%
  filter(market_segment=="Online TA")
```

Notice how in the code chunk above, the %>% symbol is used to note the logical steps of this code. First, it starts with the name of the data frame, `onlineta_city_hotels_v2`, AND THEN it tells `R` to start with the original data frame `hotel_bookings`. Then it tells it to filter on the 'hotel' column; finally, it tells it to filter on the 'market_segment' column.

This code chunk generates the same data frame by using the `View()` function:

```{r view second dataframe}
View(onlineta_city_hotels_v2)
```

Step 6: Use your new dataframe

You can use either of the data frames you created above for your new plots because they are the same.

Using the code for your previous scatterplot, replace `variable_name` in the code chunk below with either `onlineta_city_hotels` or `onlineta_city_hotels_v2` to plot the data your stakeholder requested:

```{r creating a plot part two}
ggplot(data = onlineta_city_hotels) +
  geom_point(mapping = aes(x = lead_time, y = children))
```

Based on your previous filter, this scatterplot shows data for online bookings for city hotels. The plot reveals that bookings with children tend to have a shorter lead time, and bookings with 3 children have a significantly shorter lead time (<200 days). So, promotions targeting families can be made closer to the valid booking dates.

Annotate and save visualizations

Annotation layer

To customize the look and feel of our plots, we use:

- Label
- Annotate

Annotate: To add notes to a document or diagram to explain or comment upon it.

Informative labels that can be added to the plots are:

- Titles
- Subtitles
- Captions

Check the code practice on RStudio Cloud

Check resources under “Drawing arrows and shapes in R”

Saving your visualizations

We can save our plots using two methods:

- Export option
- ggsave() function

Saving images without ggsave()

In most cases, ggsave() is the simplest way to save your plot. But there are situations when it might be best to save your plot by writing it directly to a graphics device. This reading will cover some of the different ways you can save images and plots without ggsave(), and includes additional resources to check out if you want to learn more.

To save images without using ggsave(), you can open a regular R graphics device like png() or pdf(); these will allow you to save your plot as a .png or .pdf file. You can also choose to print the plot and then close the device using dev.off().

Example of using png()	Example of using pdf()
png(file = "exampleplot.png", bg = "transparent") **plot(1:10)** **rect(1, 5, 3, 7, col = "white")** **dev.off()**	**pdf(file = "/Users/username/Desktop/example.pdf",** **width = 4,** **height = 4)** **plot(x = 1:10,** **y = 1:10)** **abline(v = 0)** **text(x = 0, y = 1, labels = "Random text")** **dev.off()**

Hands-on activity: Annotations and saving visualizations

The Scenario

As a junior data analyst for a hotel booking company, you have been creating visualizations in `R` with the `ggplot2` package to share insights about your data with stakeholders. After creating a series of visualizations using `ggplot()`, `ggplot2` aesthetics, and filters, your stakeholder asks you to add annotations to your visualizations to help explain your findings in a presentation. Luckily, `ggplot2` has annotation functions built in.

Step 1: Import your data

If you haven't exited RStudio since importing this data last time, you can skip these steps. Rerunning these code chunks won't affect your console if you want to run them just in case, though.

If this line causes an error, copy in the line setwd("projects/Course 7/Week 4") before it.

Run the code below to read in the file 'hotel_bookings.csv' into a data frame:

```{r load data}
hotel_bookings <- read.csv("hotel_bookings.csv")
```

Step 2: Refresh Your Memory

By now, you are pretty familiar with this data set. But you can refresh your memory with the `head()` and `colnames()` functions. Run two code chunks below to get at a sample of the data and also preview all the column names:

```{r look at data}
head(hotel_bookings)
```

```{r look at column names}
colnames(hotel_bookings)
```

Step 3: Install and load the 'ggplot2' and 'tidyverse' packages (optional)

If you haven't already installed and loaded the `ggplot2` package, you will need to do that before you can use the `ggplot()` function. You only have to do this once though, not every time you call `ggplot()`.

You can also skip this step if you haven't closed your RStudio account since doing the last activity. If you aren't sure, you can run the code chunk and hit 'cancel' if the warning message pops up telling you that you have already downloaded the `ggplot2` package.

Run the code chunk below to install and load `ggplot2`. This may take a few minutes!

```{r loading and installing ggplot2, echo=FALSE, message=FALSE}
install.packages('ggplot2')
library(ggplot2)
```

If you haven't installed and loaded tidyverse in this RStudio session, you can run the code chunk below. This may take a few minutes!

```{r install and download tidyverse}
install.packages('tidyverse')
library(tidyverse)
```

Step 4: Annotating your chart

Your stakeholder tells you that they would like you to share they visualization breaking down payment type by city because it will help inform how the company targets promotions in the future. They ask you to create a cleaned and labeled version and save it as a .png file for them to include in a presentation.

As a refresher, here is the chart you created before:

```{r faceting a plot}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel)
```

This creates two bar graphs: one for 'city_hotel' data and one for 'resort_hotel' data. The x axis is 'market_segment' and the y axis is 'count' for both charts.

In this visualization it is unclear where the data is from, what the main takeaway is, or even what the data is showing. To explain all of that, you can leverage annotations in `ggplot2`.

The first step will be adding a title; that is often the first thing people will pay attention to when they encounter a data visualization for the first time. To add a title, you will add `labs()` at the end of your `ggplot()` command and then input a title there. Add a descriptive title to the code chunk below:

```{r faceting a plot with a title}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
  labs(title="Number of Bookings of each market_segment for two hotels")
```

This code chunk will generate the same chart as before, but now it includes a title to explain the data visualization more clearly to your audience.

You also want to add another detail about what time period this data covers. To do this, you need to find out when the data is from.

You realize you can use the `min()` function on the year column in the data:

```{r earliest year}
min(hotel_bookings$arrival_date_year)
```
And the `max()` function:
```{r latest year}
max(hotel_bookings$arrival_date_year)
```
But you will need to save them as variables in order to easily use them in your labeling; the following code chunk creates two of those variables:
```{r latest date}
mindate <- min(hotel_bookings$arrival_date_year)
maxdate <- max(hotel_bookings$arrival_date_year)
```
Now, you will add in a subtitle using `subtitle=` in the `labs()` function. Then, you can use the **`paste0()` function to use your newly-created variables in your labels**. This is really handy, because if the data gets updated and there is more recent data added, you don't have to change the code below because the variables are dynamic:

```{r city bar chart with timeframe}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
  theme(axis.text.x = element_text(angle = 45)) +
  labs(title="Comparison of market segments by hotel type for hotel bookings",
       subtitle=paste0("Data from: ", mindate, " to ", maxdate))
```
This code chunk will add the subtitle 'Data from: 2015 to 2017' underneath the title you added earlier to the chart.

You realize that this chart is displaying the technical details a little too prominently. You don't want that to be the second thing people notice during the presentation. You decide to switch the `subtitle` to a `caption` which will appear in the bottom right corner instead.

```{r city bar chart with timeframe as caption}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
  theme(axis.text.x = element_text(angle = 45)) +
  labs(title="Comparison of market segments by hotel type for hotel bookings",
       caption=paste0("Data from: ", mindate, " to ", maxdate)
```

```
This code chunk makes a slight change to the visualization you created in the last chunk; now the "data from: 2015 to 2017" subtitle is in the bottom right corner.

**Now you want to clean up the x and y axis labels to make sure they are really clear. To do that, you can add to the `labs()` function and use `x=` and `y=`**. Feel free to change the text of the label and play around with it:

```{r city bar chart with x and y axis}
ggplot(data = hotel_bookings) +
  geom_bar(mapping = aes(x = market_segment)) +
  facet_wrap(~hotel) +
  theme(axis.text.x = element_text(angle = 45)) +
  labs(title="Comparison of market segments by hotel type for hotel bookings",
       caption=paste0("Data from: ", mindate, " to ", maxdate),
       x="Market Segment",
       y="Number of Bookings")
```
Now you have the data visualization from earlier, but now the x and y axis labels have been changed from 'market_segment' and 'count' to 'Market Segment' and 'Number of Bookings' so that the chart is clearer.

Step 5: Saving your chart

Now, it's time to save what you just created so you can easily share with stakeholders.

You can use the `ggsave()` function to do just that! It will save your image as a 7x7 at the file path you input by default, which makes it simple to export your plots from R.

The `ggsave()` function in the code chunk below will save the last plot that was generated, so if you ran something after running the code chunk above, run that code chunk again.

Then run the following code chunk to save that plot as a .png file named `hotel_booking_chart`, which makes it clear to your stakeholders what the .png file contains. Now you should be able to find this file in your 'Files' tab in the bottom right of your screen. Check it out!

```{r save your plot}
ggsave('hotel_booking_chart.png', width=7,
       height=7)
```
```

Week-5

Documentation and reports

Develop documentation and reports

R Markdown: A file format for making dynamic documents with R

Things covered in week-5:

- An overview of R Markdown
- How to install R Markdown in RStudio
- How to create an R Markdown document
- The structure and components of the document
- How to insert and edit pieces of code called chunks in your document.
- The process of exporting your document.

Overview of R Markdown

Markdown: A syntax for formatting plain text files.

Markdown formatting:

- To **italicize** a phrase or a word, use a **_single_ underscore or *asterisk*** in front and at the last of the word or phrase

R Notebook: Lets users run the code and show the graphs and charts that visualize the code.

R Markdown lets you convert your file into different formats:

- HTML, PDF, and Word documents
- Slide presentation
- Dashboard

Create R markdown documents

Structure of Markdown documents

YAML (Yet Another Markup Language): A language for data that translates it so it's readable.

Even more document elements

- Adding bullet points to the rmd file
- Another way to include a link (other than using **<>)**
- Adding images

Check out different output formats in R Markdown under "Output Formats in R Markdown"

Course 8: **Google Data Analytics Capstone: Complete a Case Study**

Week-1

What to include in a portfolio?

Even if you don't have previous data analytics work experience, you can still craft a great portfolio that represents your new skills and offers insight into who you are. Be sure to include the following in your portfolio:

- Biography: The main focus of your portfolio is to introduce yourself in a strong and memorable way. Write a concise and clear introduction of yourself. The goal is to capture your audience's interest and compel them to want to meet you to learn more.
- Contact page: Be sure to include a way for others to get in touch with you, whether it be via email, phone call (if you are comfortable), or social media handles (especially LinkedIn). You might find that your website has its own built-in contact form if you use common website builders.
- Resume: In previous readings, Adding Professional Skills to your Resume and Adding Soft Skills to your Resume, you learned how to craft a resume that reflects your skills and your experience. Be sure to include a resume somewhere in your portfolio.
- Accomplishments: You are not just limited to your past experiences. Any present career-worthy highlights you can think of should be included. This can be any certifications you have earned, data analytics events you have attended, or even blog posts you have published.
- An image of you (optional): Add a personal touch with your photo. All you need is a simple, clear photo that represents you well.

What to include in a case study?

During your interview process, you will very likely encounter the case study interview. In this interview, you will be provided with a business-related scenario where you analyze a problem and come up with the best solution. You will have a certain amount of time to solve this so it is best to be prepared for any scenario you are given. A great case study will include the following:

- Introduction: Make sure to state the purpose of the case study. This includes what the scenario is and an explanation on how it relates to a real-world obstacle. Feel free to note any assumptions or theories you might have depending on the information provided.
- Problems: You need to identify what the major problems are, explain how you have analyzed the problem, and present any facts you are using to support your findings.
- Solutions: Outline a solution that would alleviate the problem and have a few alternatives in mind to show that you have given the case study considerable thought. Don't forget to include pros and cons for each solution.
- Conclusion: End your presentation by summarizing key takeaways of all of the problem-solving you conducted, highlighting what you have learned from this.
- Next steps: Choose the best solution and propose recommendations for the client or business to take. Explain why you made your choice and how this will affect the scenario in a positive way. Be specific and include what needs to be done, who should enforce it, and when.

Career paths in data

	Data Analysts	Data Scientists	Data Specialists
Problem solving	Use existing tools and methods to solve problems with existing types of data	Invent new tools and models, ask open-ended questions, and collect new types of data	Use in-depth knowledge of databases as a tool to solve problems and manage data
Analysis	Analyze collected data to help stakeholders make better decisions	Analyze and interpret complex data to make business predictions	Organize large volumes of data for use in data analytics or business operations
Other relevant skills	• Database queries • Data visualization • Dashboards • Reports • Spreadsheets	• Advanced statistics • Machine learning • Deep learning • Data optimization • Programming	• Data manipulation • Information security • Data models • Scalability of data • Disaster recovery