**Data-cleaning verification checklist**

This reading will give you a checklist of common problems you can refer to when doing your data cleaning verification, no matter what tool you are using. When it comes to data cleaning verification, there is no one-size-fits-all approach or a single checklist that can be universally applied to all projects. Each project has its own organization and data requirements that lead to a unique list of things to run through for verification.



Keep in mind, as you receive more data or a better understanding of the project goal(s), you might want to revisit some or all of these steps.

**Correct the most common problems**

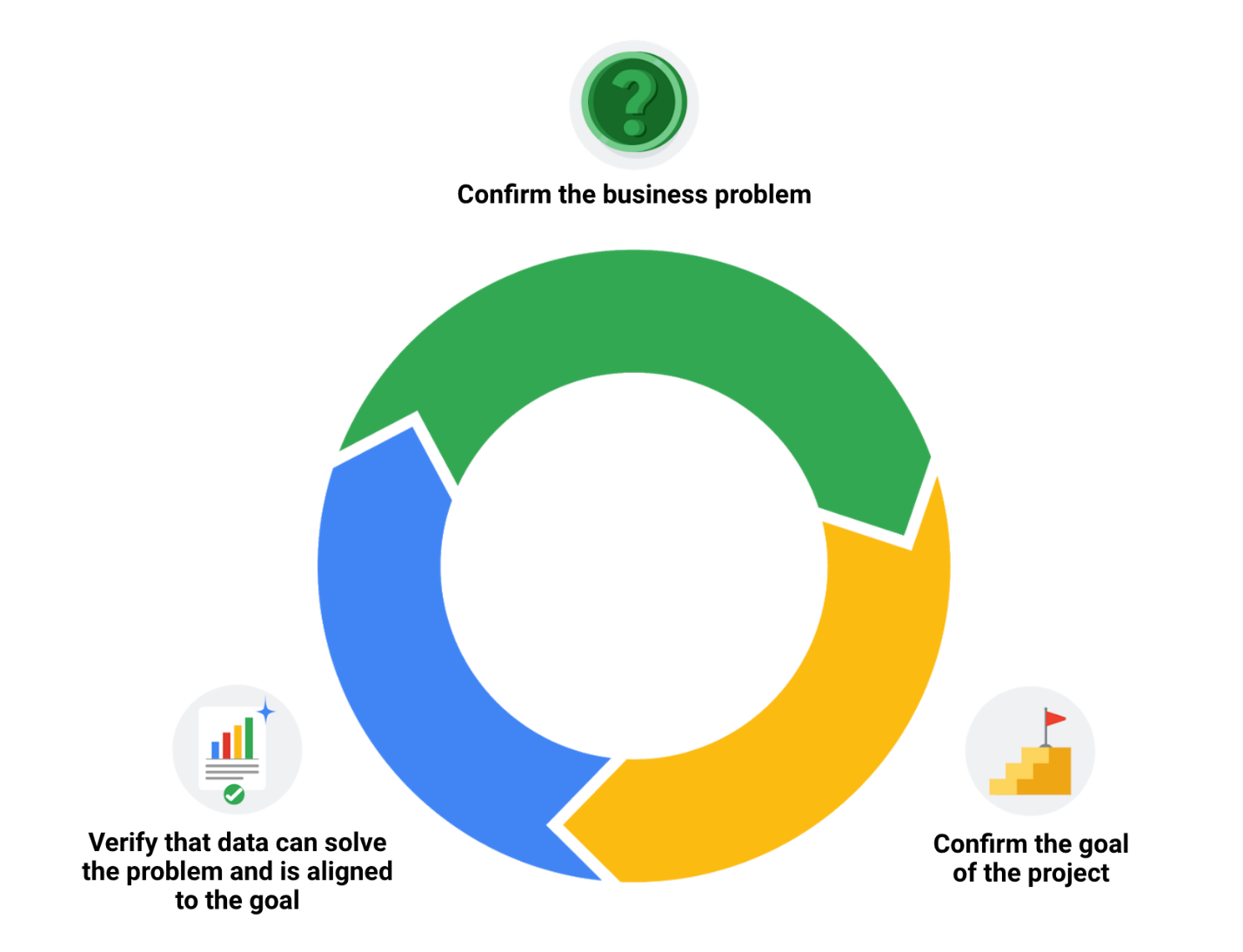
Make sure you identified the most common problems and corrected them, including:

* **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?

**Review the goal of your project**

Once you have finished these data cleaning tasks, it is a good idea to review the goal of your project and confirm that your data is still aligned with that goal. This is a continuous process that you will do throughout your project-- but here are three steps you can keep in mind while thinking about this:

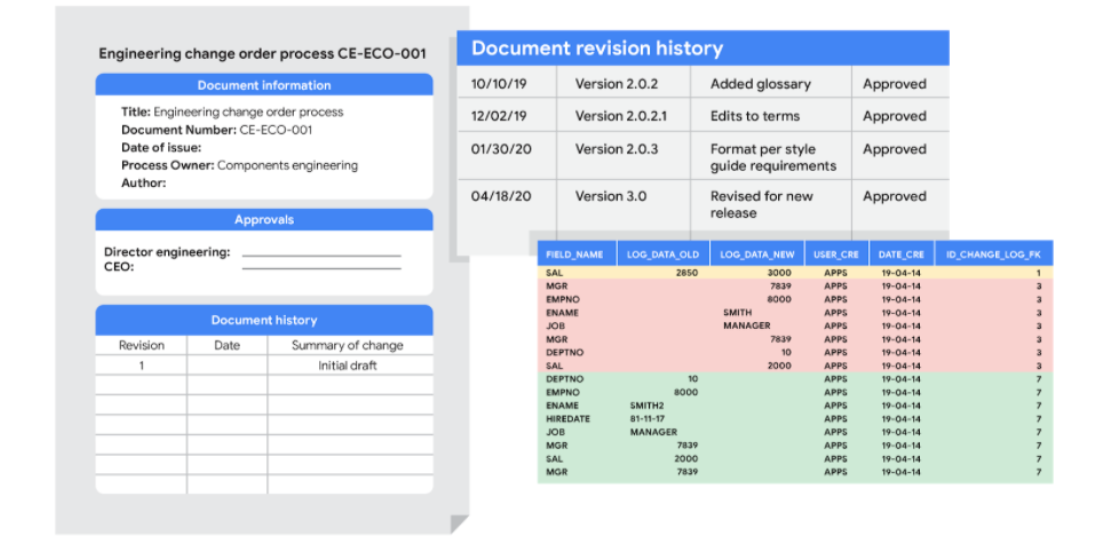
* Confirm the business problem
* Confirm the goal of the project
* Verify that data can solve the problem and is aligned to the goal



**Embrace changelogs**

What do engineers, writers, and data analysts have in common? Change.

Engineers use **engineering change orders** (ECOs) to keep track of new product design details and proposed changes to existing products. Writers use **document revision histories** to keep track of changes to document flow and edits. And data analysts use **changelogs** to keep track of data transformation and cleaning. Here are some examples of these:



**Automated version control takes you most of the way**

Most software applications have a kind of history tracking built in. For example, in Google sheets, you can check the version history of an entire sheet or an individual cell and go back to an earlier version. In Microsoft Excel, you can use a feature called **Track Changes**. And in BigQuery, you can view the history to check what has changed.

Here’s how it works:

|  |  |
| --- | --- |
| Google Sheets | 1. Right-click the cell and select **Show edit history**. 2. Click the left-arrow < or right arrow > to move backward and forward in the history as needed. |
| Microsoft Excel | 1. If Track Changes has been enabled for the spreadsheet: click **Review**.2. Under **Track Changes**, click the **Accept/Reject Changes** option to accept or reject any change made. |
| BigQuery | Bring up a previous version (without reverting to it) and figure out what changed by comparing it to the current version. |

**Changelogs take you down the last mile**

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:

* 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.

**Bonus tip**



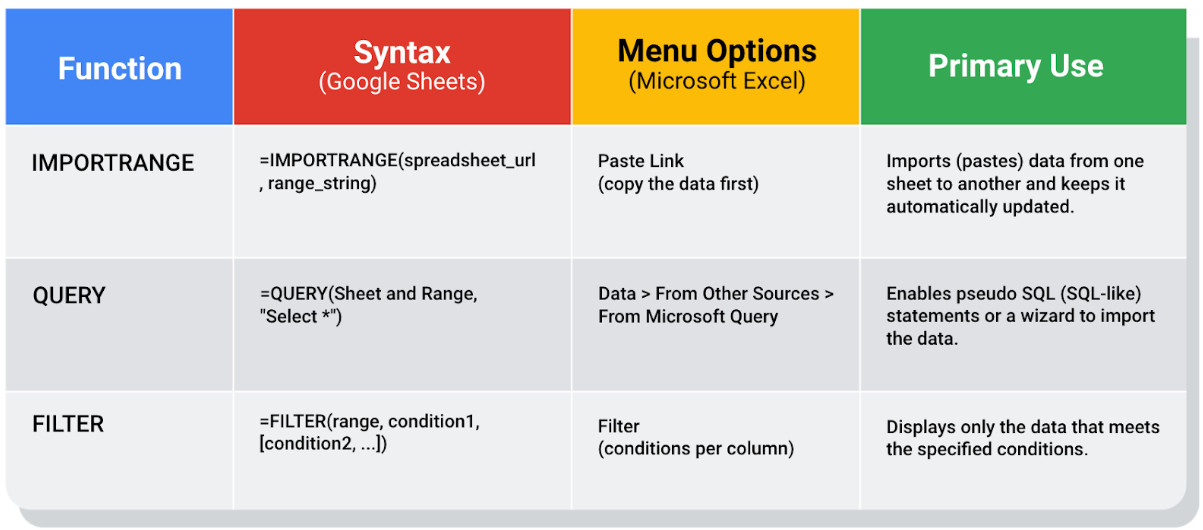
If an analyst is making changes to an existing SQL query that is shared across the company, the company most likely uses what is called a **version control system**. An example might be a query that pulls daily revenue to build a dashboard for senior management.

Here's 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.

**Advanced functions for speedy data cleaning**

In this reading, you will learn about some advanced functions that can help you speed up the data cleaning process in spreadsheets. Below is a table summarizing three functions and what they do:

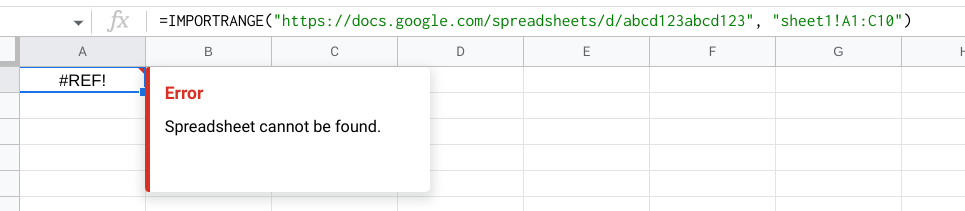


**Keeping data clean and in sync with a source**

The [**IMPORTRANGE**](https://support.google.com/docs/answer/3093340?hl=en) function in Google Sheets and the[**Paste Link**](https://professor-excel.com/how-to-paste-cell-links/) feature (a Paste Special option in Microsoft Excel) both allow you to insert data from one sheet to another. Using these on a large amount of data is more efficient than manual copying and pasting. They also reduce the chance of errors being introduced by copying and pasting the wrong data. They are also helpful for data cleaning because you can “cherry pick” the data you want to analyze and leave behind the data that isn’t relevant to your project. Basically, it is like canceling noise from your data so you can focus on what is most important to solve your problem. This functionality is also useful for day-to-day data monitoring; with it, you can build a tracking spreadsheet to share the relevant data with others. The data is synced with the data source so when the data is updated in the source file, the tracked data is also refreshed.

In Google Sheets, you can use the **IMPORTRANGE** function. It enables you to specify a range of cells in the other spreadsheet to duplicate in the spreadsheet you are working in. You must allow access to the spreadsheet containing the data the first time you import the data.

**The URL shown below is for syntax purposes only. Don't enter it in your own spreadsheet. Replace it with a URL to a spreadsheet you have created so you can control access to it by clicking the Allow access button.**



Refer to the [Google support page for IMPORTRANGE](https://support.google.com/docs/answer/3093340?hl=en) for the sample usage and syntax.

**Example of using IMPORTRANGE**

An analyst monitoring a fundraiser needs to track and ensure that matching funds are distributed. They use **IMPORTRANGE** to pull all the matching transactions into a spreadsheet containing all of the individual donations. This enables them to determine which donations eligible for matching funds still need to be processed. Because the total number of matching transactions increases daily, they simply need to change the range used by the function to import the most up-to-date data.

On Tuesday, they use the following to import the donor names and matched amounts:

**=IMPORTRANGE("https://docs.google.com/spreadsheets/d/abcd123abcd123", "sheet1!A1:C10", "Matched Funds!A1:B4001")**

On Wednesday, another 500 transactions were processed. They increase the range used by 500 to easily include the latest transactions when importing the data to the individual donor spreadsheet:

**=IMPORTRANGE("https://docs.google.com/spreadsheets/d/abcd123abcd123", "Matched Funds!A1:B4501")**

**Note: The above examples are for illustrative purposes only. Don't copy and paste them into your spreadsheet. To try it out yourself, you will need to substitute your own URL (and sheet name if you have multiple tabs) along with the range of cells in the spreadsheet that you have populated with data.**

**Pulling data from other data sources**

The [**QUERY**](https://support.google.com/docs/answer/3093343?hl=en) function is also useful when you want to pull data from another spreadsheet. The **QUERY** function's SQL-like ability can extract specific data within a spreadsheet. For a large amount of data, using the **QUERY** function is faster than filtering data manually. This is especially true when repeated filtering is required. For example, you could generate a list of all customers who bought your company’s products in a particular month using manual filtering. But if you also want to figure out customer growth month over month, you have to copy the filtered data to a new spreadsheet, filter the data for sales during the following month, and then copy those results for the analysis. With the **QUERY** function, you can get all the data for both months without a need to change your original dataset or copy results.

The **QUERY** function syntax is similar to **IMPORTRANGE**. You enter the sheet by name and the range of data that you want to query from, and then use the SQL **SELECT** command to select the specific columns. You can also add specific criteria after the **SELECT** statement by including a **WHERE** statement. But remember, all of the SQL code you use has to be placed between the quotes!

Google Sheets run the Google Visualization API Query Language across the data. Excel spreadsheets use a query wizard to guide you through the steps to connect to a data source and select the tables. In either case, you are able to be sure that the data imported is verified and clean based on the criteria in the query.

**Examples of using QUERY**

Check out the [Google support page for the QUERY function](https://support.google.com/docs/answer/3093343?hl=en) with sample usage, syntax, and examples you can download in a Google sheet.

Link to make a copy of the sheet: [QUERY examples](https://docs.google.com/spreadsheets/d/1815H5TCe91LLT6tD6FmxMHmeJAAkr4o5Q6rNpV6xiFk/copy)

**The solution**

Analysts can use SQL to pull a specific dataset into a spreadsheet. They can then use the **QUERY** function to create multiple tabs (views) of that dataset. For example, one tab could contain all the sales data for a particular month and another tab could contain all the sales data from a specific region. This solution illustrates how SQL and spreadsheets are used well together.

**Filtering data to get what you want**

The [**FILTER**](https://support.google.com/docs/answer/3093197?hl=en) function is fully internal to a spreadsheet and doesn’t require the use of a query language. The **FILTER** function lets you view only the rows (or columns) in the source data that meet your specified conditions. It makes it possible to pre-filter data before you analyze it.

The **FILTER** function might run faster than the **QUERY** function. But keep in mind, the **QUERY** function can be combined with other functions for more complex calculations. For example, the **QUERY** function can be used with other functions like **SUM** and **COUNT** to summarize data, but the **FILTER** function can't.

**Example of using FILTER**

Check out the [Google support page for the FILTER function](https://support.google.com/docs/answer/3093197?hl=en) with sample usage, syntax, and examples you can download in a Google sheet.

Link to make a copy of the sheet: [FILTER examples](https://docs.google.com/spreadsheets/d/1caULJLQvQuzBnCN7rO9utg0xSKrYms7wM0Ph7A2JXY4/copy)

**Terms and definitions for Course 4, Module 4**

**CASE:** A SQL statement that returns records that meet conditions by including an if/then statement in a query

**Changelog:** A file containing a chronologically ordered list of modifications made to a project

**COUNTA:** A spreadsheet function that counts the total number of values within a specified range

**Find and replace:** A tool that finds a specified search term and replaces it with something else

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

Mark as completed

Like

Dislike

Report an issue