

Chapter 2. Data Is the First Step

This chapter provides an overview of the use cases and datasets used in the book while also providing information on where to find data sources for further study and practice. You'll also learn about data types, and the difference between batch and streaming data. You'll get hands-on practice with data preprocessing using Google's free browser-based open source Jupyter Notebook. The chapter concludes with a section on using GitHub to create a data repository for the selected projects used in the book.

Overview of Use Cases and Datasets Used in the Book

Hopefully, you picked up our book to learn ML not from a math-first or algorithm-first approach but from a project-based approach. The use cases we've chosen are designed to teach you ML using actual, real-world data across different sectors. There are use cases for healthcare, retail, energy, telecommunications, and finance. The use case on customer churn can be applied to any sector. Each of the use case projects can stand on its own if you have some data preprocessing experience, so feel free to skip ahead to what you need to learn to upskill yourself. **Table 2-1** shows each section, its use case, sector, and whether it is no-code or low-code.

Table 2-1. List of use cases by industry sector and coding type

Section	Use case	Sector	Type
1	Product pricing	Retail	N/A
2	Heart disease	Healthcare	Low-code data preprocessing
3	Marketing campaign	Energy	No-code (AutoML)
4	Advertising media channel sales	Insurance	No-code (AutoML)
5	Fraud detection	Financial	No-code (AutoML)
6	Power plant production prediction	Energy	Low-code (BigQuery ML)
7	Customer churn prediction	Telecommunications	Low-code (scikit-learn and Keras)
8	Improve custom model performance	Automotive	Custom-code (scikit-learn, Keras, BigQuery ML)

1. Retail: Product Pricing

This section begins with a use case designed to illustrate the role of data in decision making. In this use case, you are in charge of marketing for a company that makes umbrellas, and the *business goal* is to increase sales. If you reduce the selling price of your existing umbrellas, can you predict how many umbrellas you will sell? **Figure 2-1** shows the data elements that may impact a price reduction strategy to increase sales.

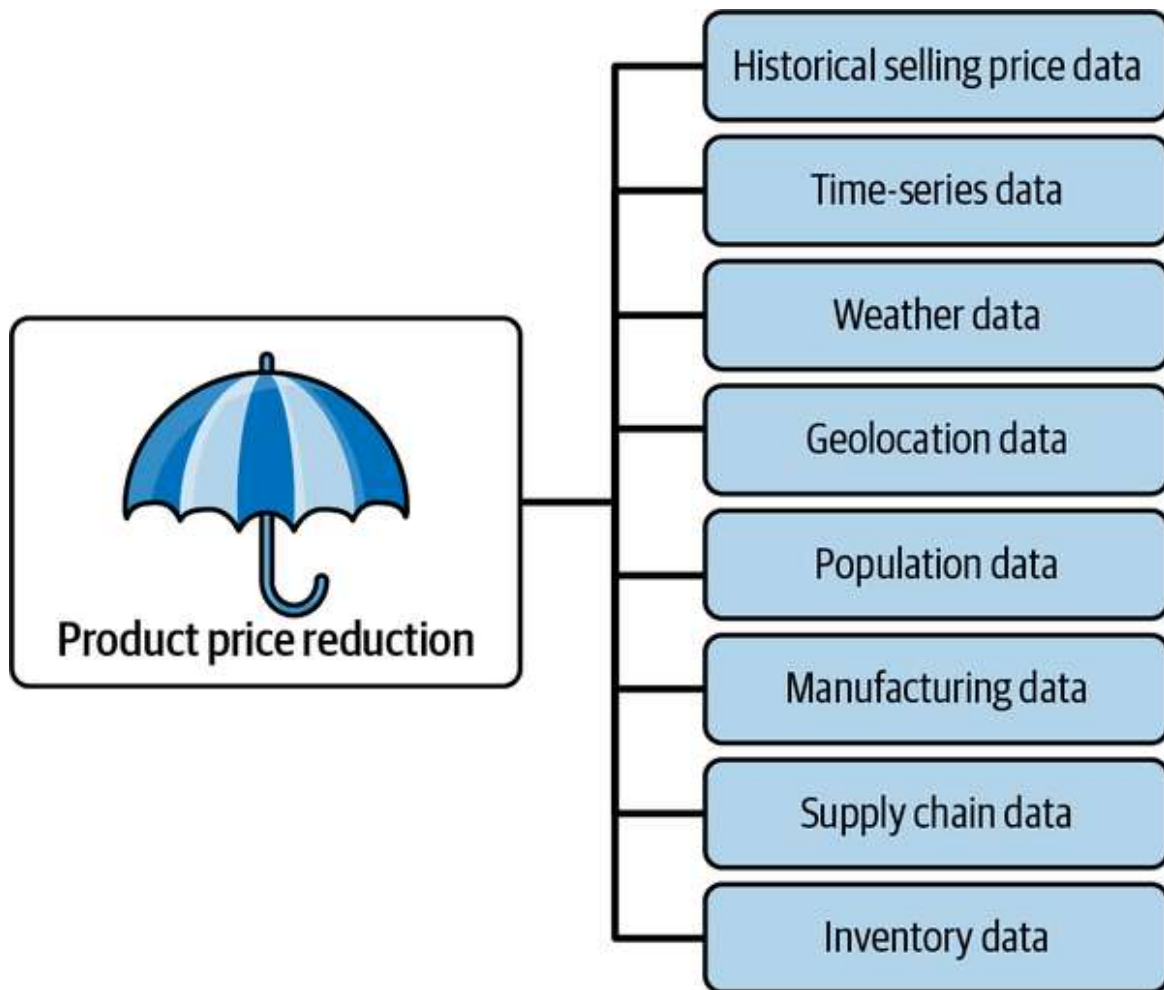


Figure 2-1. Data elements that impact a price reduction strategy to increase sales.

2. Healthcare: Heart Disease Campaign

In this one, you are a healthcare consultant and are given data on heart disease mortality for populations over the age of 35 in the United States. The goal is to analyze the heart disease mortality data and suggest a possible use case in a heart disease prevention campaign. For example, one

possible use case would be to track trends in heart disease mortality over time or to develop and validate models for predicting heart disease mortality. This dataset is dirty. Some fields have missing values. One field is missing. In working through these issues, you learn to import data into a Python Jupyter Notebook, analyze it, and fix dirty elements. **Figure 2-2** shows the data elements that contribute to your analysis.

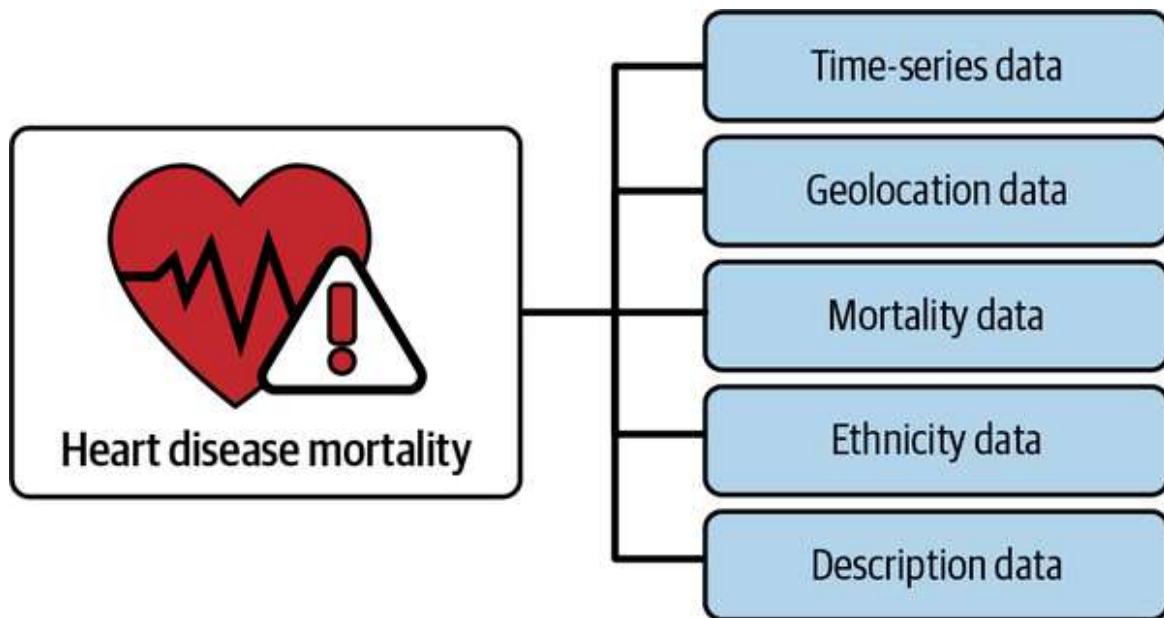


Figure 2-2. Data elements for a heart disease mortality use case.

3. Energy: Utility Campaign

Here, you are a business analyst working for a utility company. You are tasked with developing a marketing and outreach program that targets communities with high electrical energy consumption. The data has already been preprocessed. You do not have an ML background or any programming knowledge. You elect to use AutoML as your ML framework. **Figure 2-3** shows the data elements that contribute to your model.

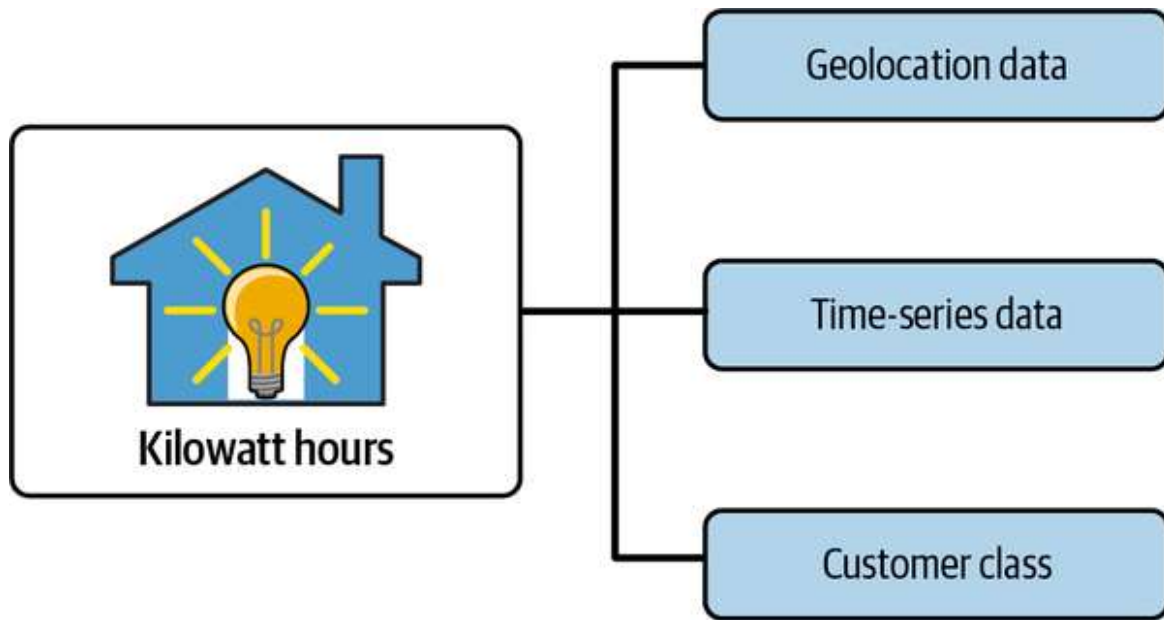


Figure 2-3. Data elements that contribute to the utility energy campaign.

4. Insurance: Advertising Media Channel Sales Prediction

In this section, you work on a team charged with developing a media strategy for an insurance company. The team wants to develop an ML model to predict sales based on advertising spend in various media channels. You are tasked with performing exploratory data analysis and with building and training the model. You do not have an ML background or any programming knowledge. You elect to use AutoML as your ML framework. **Figure 2-4** shows the data elements that contribute to your model.

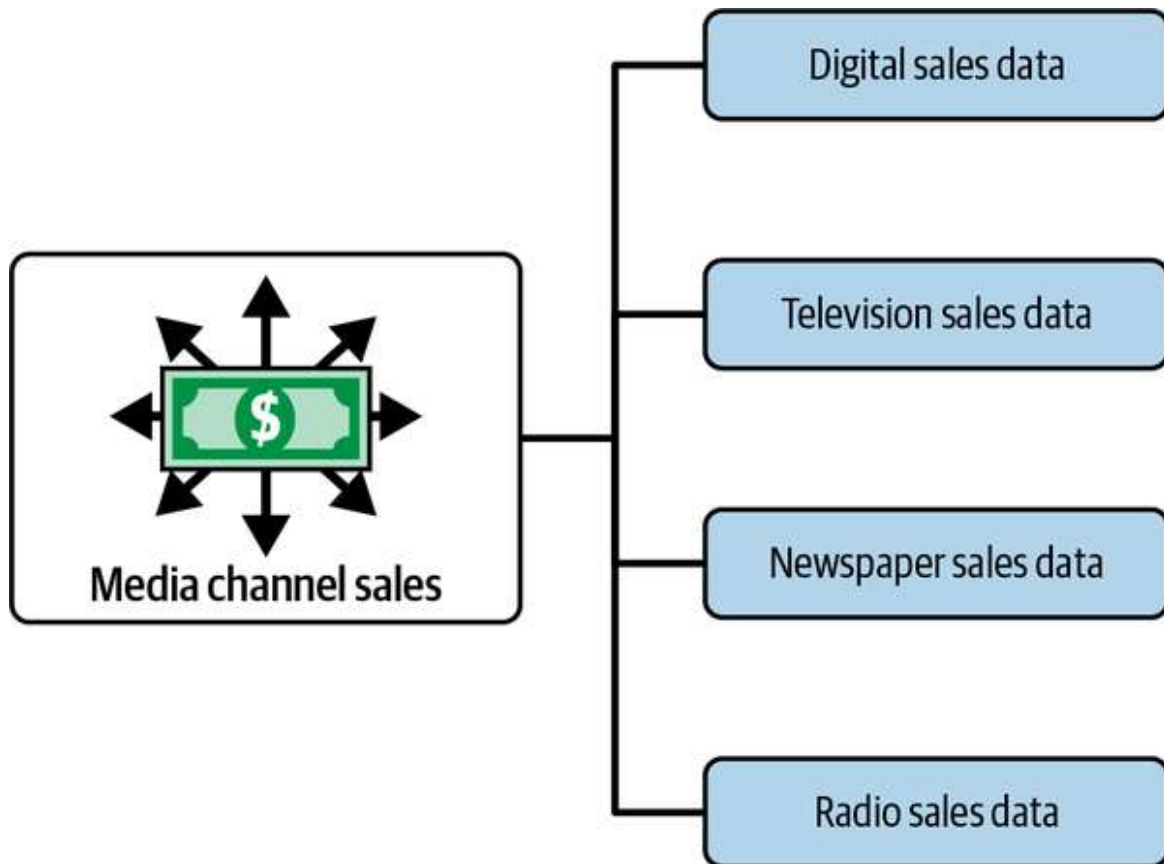


Figure 2-4. Data elements that contribute to media channel sales prediction.

5. Financial: Fraud Detection

Your goal in this project is to build a model to predict whether a financial transaction is fraudulent or legitimate. Your new company is a mobile payment service that serves hundreds of thousands of users. Fraudulent transactions are fairly rare and are usually caught by other protections. However, the unfortunate truth is that some of these are slipping through the cracks and negatively impacting your users. The dataset in this section consists of transaction data that has been simulated to replicate user behavior and fraudulent transactions. You do not have an ML background or any programming knowledge. You elect to use AutoML as your ML framework. **Figure 2-5** shows the data elements that contribute to your model.

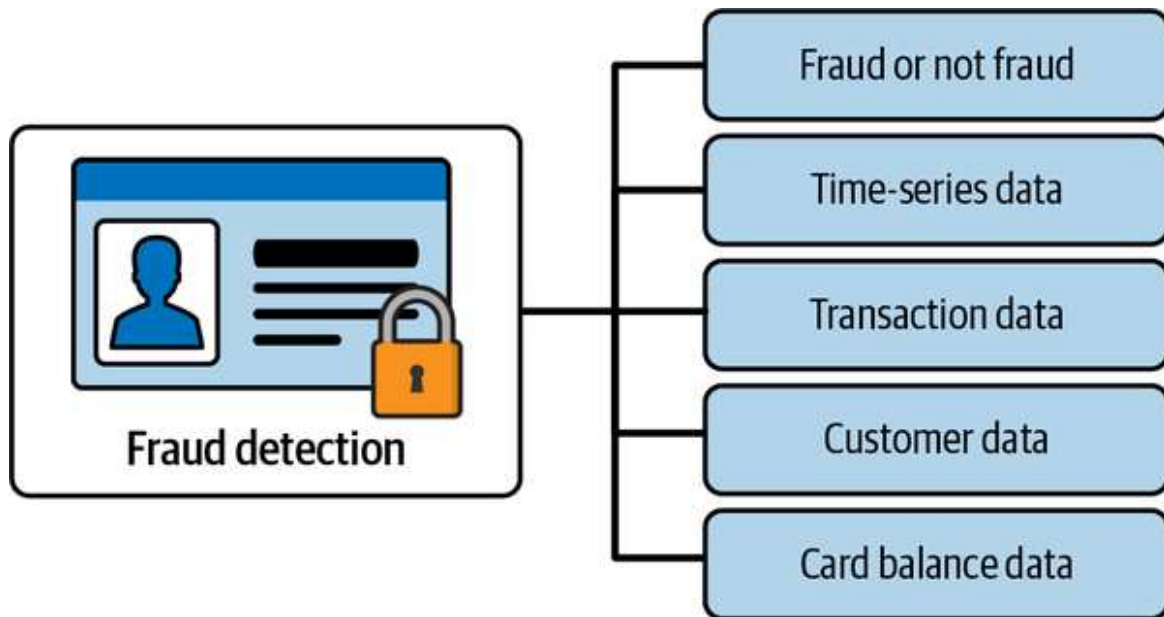


Figure 2-5. Data elements that contribute to a fraud detection model.

6. Energy: Power Production Prediction

Your goal in this project will be to predict the net hourly electrical energy output for a combined cycle power plant (CCPP) given the weather conditions near the plant at the time. The dataset in this section contains data points collected from a CCPP over a six-year period (2006–2011) when the power plant was set to work with a full load. The data is aggregated per hour, though the exact hour for the recorded weather conditions and energy production is not supplied in the dataset. From a practical viewpoint, this means that you will not be able to treat the data as sequence or time-series data, where you use information from previous records to predict future records. You have some Structured Query Language (SQL) knowledge from working with databases. You elect to use Google's BigQuery Machine Learning as your ML framework. **Figure 2-6** shows the data elements that contribute to your model.

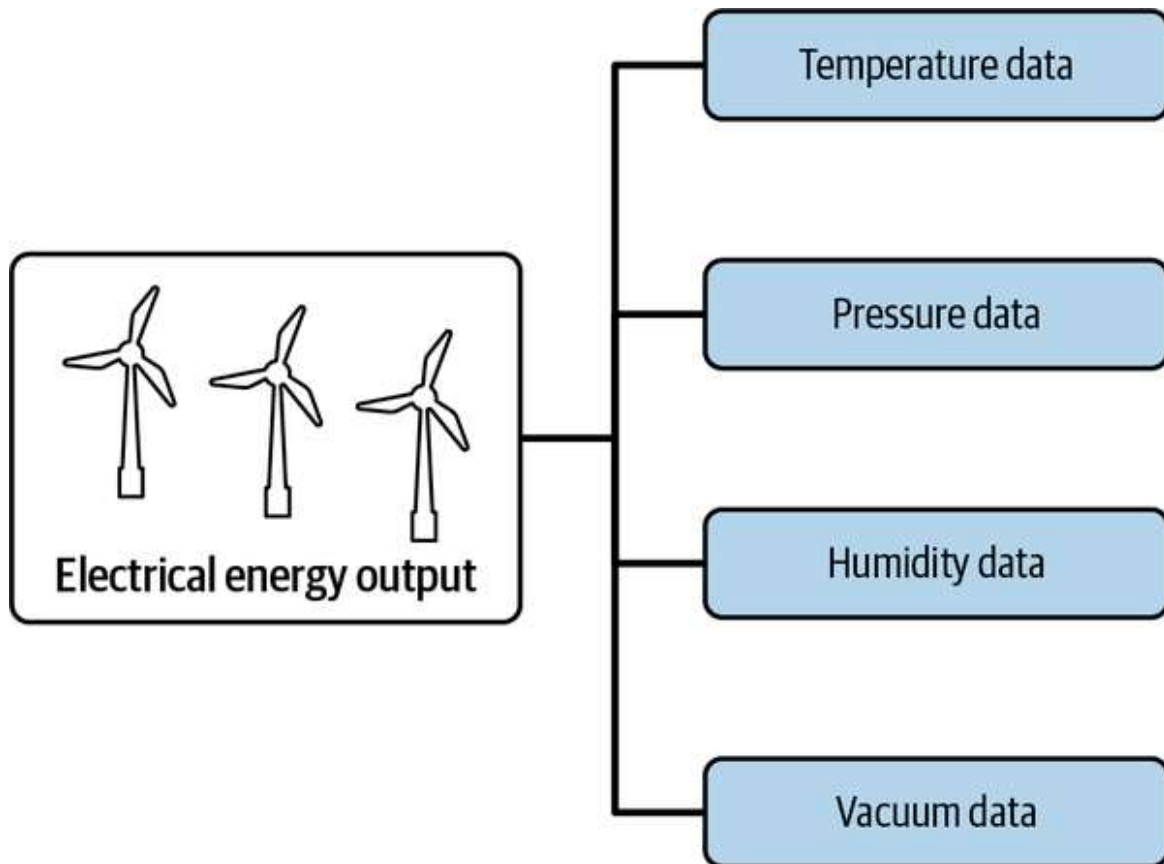


Figure 2-6. Data elements that contribute to the electrical energy output model.

7. Telecommunications: Customer Churn Prediction

Your goal in this project will be to predict customer churn for a telecommunications company. *Customer churn* is defined as the *attrition rate* for customers, or in other words, the rate of customers that choose to stop using services. Telecommunications companies often sell their products at a monthly rate or via annual contracts, so *churn* here will represent when a customer cancels their subscription or contract in the following month. The dataset contains both numeric variables and categorical variables, where the variable takes on a value from a discrete set of possibilities. You have some Python knowledge and find AutoML very powerful, yet are looking to learn low-code solutions that allow you to have a bit more control over your model. You elect to use scikit-learn and Keras as ML frameworks. **Figure 2-7** shows the data elements that contribute to your model.

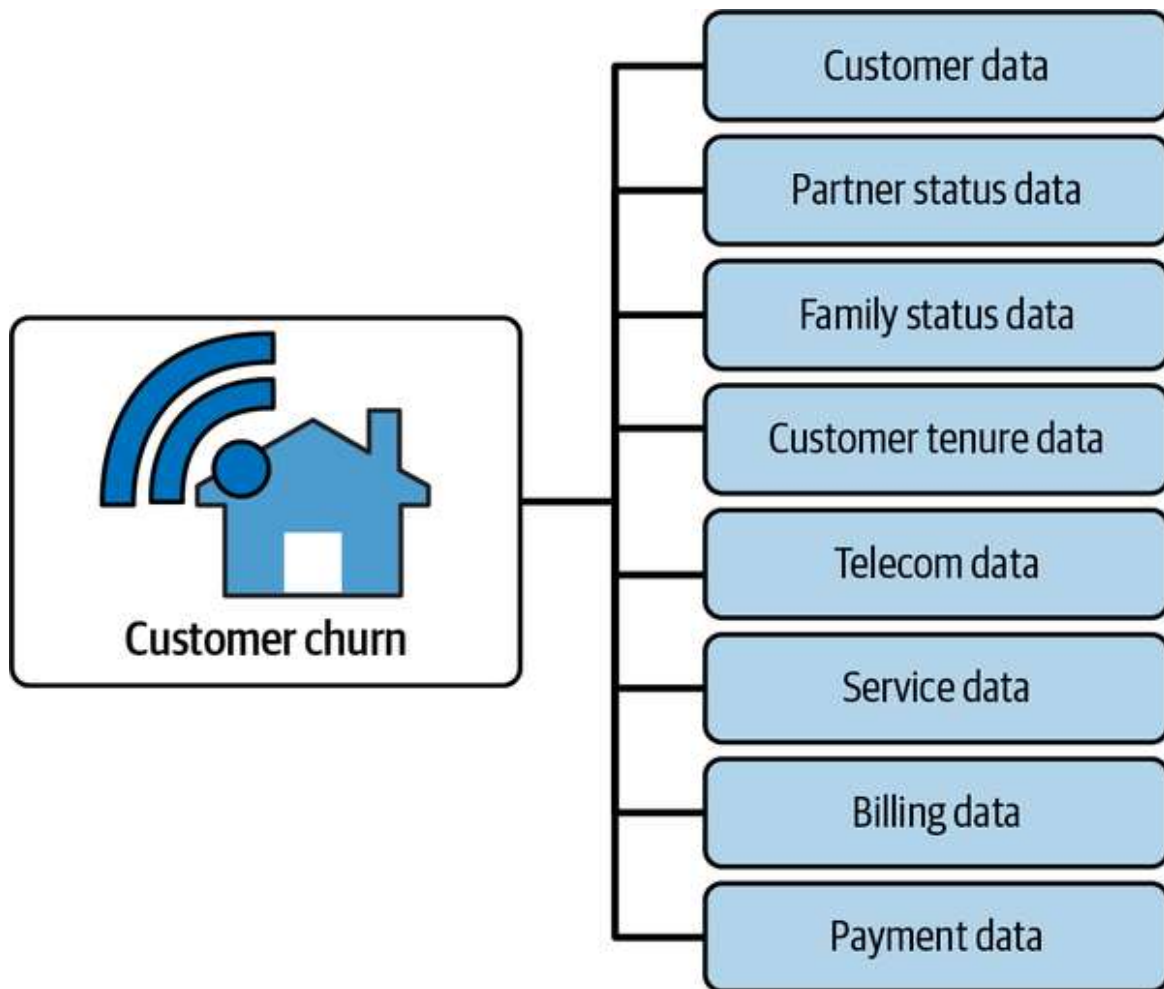


Figure 2-7. Data elements that contribute to the customer churn model.

8. Automotive: Improve Custom Model Performance

Your goal in this project (as a newer member of an ML team) will be to improve the performance of an ML model trained to predict the auction price of used cars. The initial model is a linear regression model in scikit-learn and does not quite meet your business goals. You will ultimately explore using tools in scikit-learn, Keras, and BigQuery ML to improve your model performance. The training, validation, and testing datasets used for training the linear regression model have been supplied to you as CSV files. These datasets have been cleaned (missing and incorrect values have been remedied appropriately), and the code that was used to build the scikit-learn linear regression model has also been provided. **Figure 2-8** shows the data elements that contribute to your model.

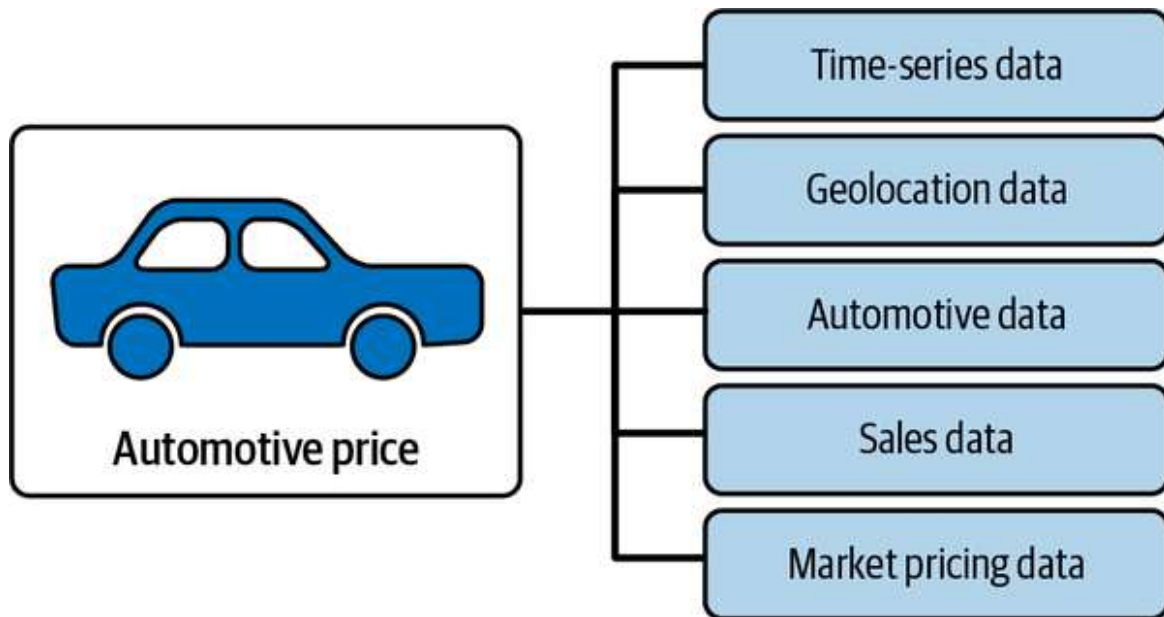


Figure 2-8. Data elements that contribute to the automotive pricing model.

Data and File Types

Data is really the first step, so let's go over some basic terminology and concepts around data. If you are already familiar with the differences between quantitative and qualitative data; between structured, semistructured, and unstructured data; and batch and streaming data, then skip to [“An Overview of GitHub and Google’s Colab”](#), where you can start creating the Jupyter Notebook in GitHub.

Quantitative and Qualitative Data

In data analysis, you work with two types of data: quantitative and qualitative. If it can be counted or measured, and given a numerical value, it's quantitative data. Quantitative data can tell you how many, how much, or how often—for example, how many people visited the website to view the product catalog? How much revenue did the company make this fiscal year? How often do the machines that manufacture your umbrella handles break?

Unlike quantitative data, qualitative data cannot be measured or counted and can include almost any non-numerical data. It's descriptive, expressed

in terms of language rather than numbers. Why is this distinction important in ML? If you have qualitative data, then you need to preprocess it so that it *becomes* quantitative—that is because you cannot feed qualitative data into an ML model. You will learn how to handle some qualitative data in subsequent chapters.

Structured, Unstructured, and Semistructured Data

Data can be grouped into three buckets: structured, unstructured, and semistructured.

Structured data is information that has been formatted and transformed into a well-defined data model. A data model is a way of organizing and structuring data so that it can be easily understood and manipulated. Data models are used in a variety of applications, including databases, software applications, and data warehouses. Structured data is well organized.

Table 2-2 shows the schema and data type used in **Chapter 4**’s Advertising Media Channel Sales Prediction use case. Note that there is a column name and column type. There are four columns of numeric (quantitative) data that feed into the AutoML model.

*Table 2-2. Schema and field value information for the advertising dataset from **Chapter 4***

Column name	Column type	Notes about field values
Digital	Numeric	Budget of advertisements in digital
Newspaper	Numeric	Budget of advertisements in newspaper
Radio	Numeric	Budget of advertisements in radio
TV	Numeric	Budget of advertisements in TV

Here are some examples of structured data:

- Customer records
- Product inventory
- Financial data
- Transaction logs
- Website analytics data
- Log files

Unstructured data is data that is not structured or tabular or formatted in a specific way. Here are some examples of unstructured data:

- Social media posts
- Chats (text)
- Videos
- Photos
- Web pages
- Audio files

Semistructured data is a type of structured data that lies between structured and unstructured data. It doesn't have a tabular data model but can include tags and semantic markers for records and fields in a dataset.

Semistructured data is, essentially, a combination of structured and unstructured. Videos may contain meta tags that relate to the date or location, but the information within has no structure.

Here are some examples of semistructured data:

- CSV, XML, JSON files
- HTML

- Email (Emails are considered semistructured data because they have some structure, but not as much as structured data. Emails typically contain a header, a body, and attachments. The header contains information about the sender, recipient, and date of the message. The body of the message contains the text of the message.)

Figure 2-9 compares unstructured, semistructured, and structured data.

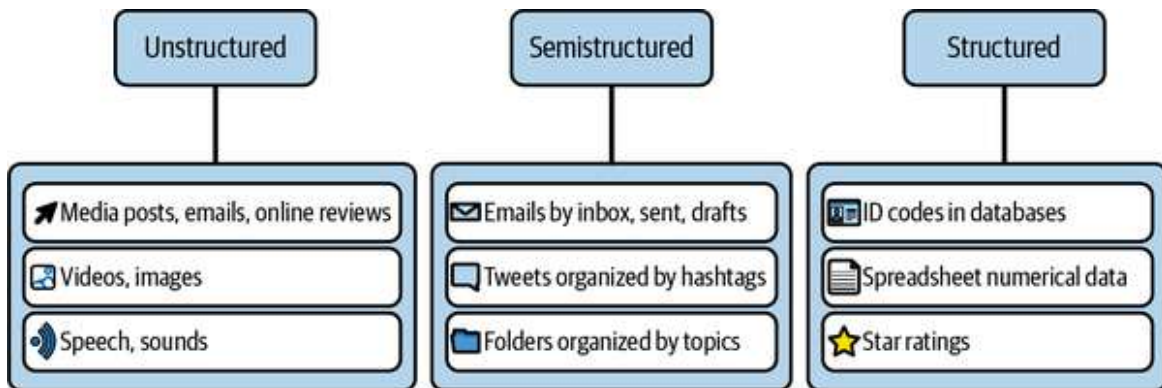


Figure 2-9. Unstructured, semistructured, and structured data examples.

Data File Types

You just learned about the different types of data, and several file types were mentioned. There are many different types of data file formats, each with its own purpose. Table 2-3 shows some of the most common data file types.

Table 2-3. Common data file types

Common data file types	Common file extensions
Text files are files that contain plain text. They are typically used to store documents, such as letters, reports, and code.	Some common text file extensions include .txt, .csv, .tsv, .log, and .json.
Spreadsheet files are files that contain data in a tabular format. They are typically used to store financial data, sales data, and other tabular data.	Some common spreadsheet file extensions include .xls, .xlsx, and .csv.
Image files are files that contain images. They are typically used to store photos, graphics, and other visual content.	Some common image file extensions include .jpg, .png, and .gif.
Audio files are files that contain audio recordings. They are typically used to store music, podcasts, and other audio content.	Some common audio file extensions include .mp3, .wav, and .ogg.
Video files are files that contain video recordings. They are typically used to store movies, TV shows, and other video content.	Some common video file extensions include .mp4, .avi, and .mov.
Webpage files are files that contain webpages. They are typically used to store HTML code, CSS code, and JavaScript code.	Some common webpage file extensions include .html, .htm, and .php.

How Data Is Processed

There are two main modes of how data is processed: batch processing and real-time processing. Batch processing is a mode of data processing where

data is collected over a period of time and then processed at a later time. This is a common mode of data processing for large datasets, as it can be more efficient to process the data in batches than to process it in real time. Real-time processing is a mode of data processing where data is processed as soon as it is collected. This is a common mode of data processing for applications where the data needs to be processed quickly, such as fraud detection or stock trading.

The frequency of how data is processed can also vary. Continuous processing is a mode of data processing where data is processed continuously, as it is collected. This is a common mode of data processing for applications where the data needs to be processed in real time. Periodic processing is a mode of data processing where data is processed at regular intervals. This is a common mode of data processing for applications where the data does not need to be processed in real time, such as financial reporting.

The mode and frequency of how data is processed depends on the specific needs of the application. For example, an application that needs to process large datasets may use batch processing, while an application that needs to process data in real time may use real-time processing. **Table 2-4** summarizes the different modes and frequencies of data processing.

Table 2-4. Summary of the different modes and frequencies of data processing

Mode	Frequency	Description
Batch processing	Intermittent	Data is collected over a period of time and then processed at a later time.
Real-time processing	Continuous	Data is processed as soon as it is collected.
Periodic processing	Intermittent	Data is processed at regular intervals.

Batch data and streaming data are two different types of data that are processed differently.

- *Batch* data is data that is collected over a period of time and then processed at a later time.
- *Streaming* data is data that is processed as it is received.

Batch data requires data to be collected in batches before it can be processed, stored, analyzed, and fed into an ML model.

Streaming data flows in continuously and can be processed, stored, analyzed, and acted on as soon as it is generated. Streaming data can come from a wide variety of distributed sources in many different formats. Simply stated, streaming data is data that is generated continuously and in real time. This type of data can be used to train ML models that can make predictions in real time. For example, a streaming data model could be used to detect fraud or predict customer churn.

An Overview of GitHub and Google's Colab

This section talks about how to set up a Jupyter Notebook and GitHub project repository. The GitHub repository can hold your datasets and the low-code project notebooks you create—such as the Jupyter Notebooks mentioned in this book.

Use GitHub to Create a Data Repository for Your Projects

GitHub is a code repository where you store your Jupyter notebooks and experimental raw data for free. Let's get started!

1. Sign up for a new GitHub account

GitHub offers personal accounts for individuals and organizations. When you create a personal account, it serves as your identity on *GitHub.com*.

When you create a personal account, you must select a billing plan for the account.

2. Set up your project's GitHub repo

To set up your first GitHub repo, see the full steps in the “Use GitHub to Create a Data Repository for Your Projects” page in Chapter 2 of the book’s [GitHub repo](#). You can also refer to [GitHub documentation](#) on how to create a repo.

Type a short, memorable name for your repository; for example, low-code book projects. A description is optional, but in this exercise, enter **Low-code AI book projects**. Choose a repository visibility—in this case, the default is Public, which means anyone on the internet can see this repository. [Figure 2-10](#) shows what your setup should look like.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * gstripling00 / Repository name * low-code book projects

⚠ Your new repository will be created as low-code-book-projects.

Great repository names are short and memorable. Need inspiration? How about [legendary-sniffle?](#)

Description (optional)
Low-code AI book projects

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Figure 2-10. Create a new repository page.

Have GitHub create a *README.md* file. This is where you can write a long description for your project. Keep the other defaults: *.gitignore* lets you choose which files not to track, and a license tells others what they can and can’t do with your code. Lastly, GitHub reminds you that you are creating a public repository in your personal account. When done, click “Create repository.” [Figure 2-11](#) shows what the page should look like.

Initialize this repository with:

☒ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

 You are creating a public repository in your personal account.

Create repository

Figure 2-11. Initialize the repo settings.

After clicking “Create repository,” the repo page appears, as shown in [Figure 2-12](#).

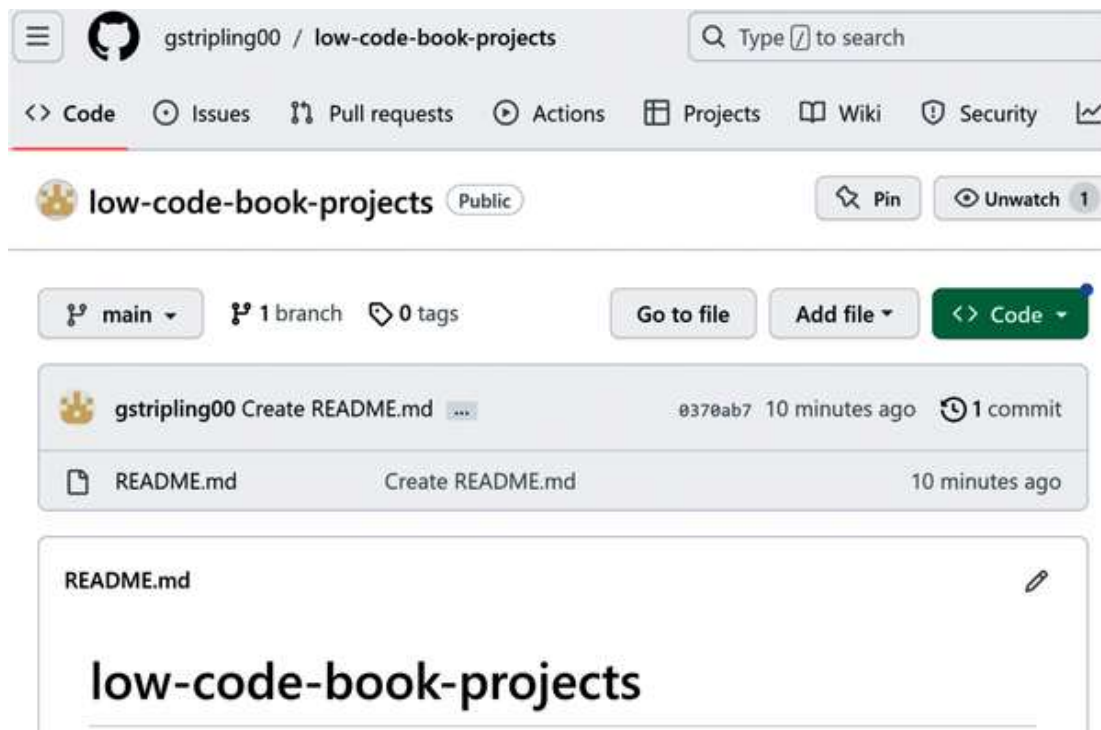


Figure 2-12. Your GitHub repo page.

NOTE

In the next section, you will create a Jupyter Notebook in Google's Colaboratory. You will save the notebook file from Colab into GitHub, which will create a file under the *Main* branch. Creating a file in GitHub is a great way to improve collaboration on a project. It provides a number of features that can help teams work more effectively together:

Version control

GitHub tracks changes to files. This means that everyone who has access to the file can see the changes that have been made, and they can revert to a previous version if necessary.

Pull requests

Pull requests allow collaborators to propose changes to a file. This gives everyone a chance to review the changes before they are merged into the main branch.

Issues

Issues can be used to track bugs or feature requests. This allows everyone to collaborate on solving problems and adding new features.

Comments

Comments can be added to files to provide feedback or ask questions. This allows for a more collaborative way of working on code.

Using Google's Colaboratory for Low-Code AI Projects

Years ago, if you wanted to learn Python, you had to download the Python interpreter and install it on your computer. This could be a daunting task for beginners, as it required knowledge of how to install software and configure your computer. Today, there are many ways to learn Python without having to install anything on your computer. You can use online IDEs (integrated development environments) that allow you to write and run Python code in a web browser. You can also use cloud-based Python environments that provide you with access to a Python interpreter and all the libraries you need to get started.

These online and cloud-based resources make it easier than ever to learn Python, regardless of your level of experience or technical expertise. Here are some of the benefits of using online and cloud-based resources to learn Python:

No installation required

You can start learning Python right away, without having to download or install any software.

Access from anywhere

You can use online and cloud-based resources to learn Python from anywhere, as long as you have an internet connection.

Affordable

Online and cloud-based resources are often free or very affordable.

Easy to use

Online and cloud-based resources are designed to be easy to use, even for beginners.

You build your low-code Python Jupyter Notebook using Google's Colaboratory, or Colab. Colab is a hosted Jupyter Notebook service that requires no setup to use, while providing access to computing resources, including graphical processing units (GPUs). Colab runs in your web browser and allows you to write and execute Python code. Colab notebooks are stored in Google Drive and can be shared similarly to how you share Google Docs or Sheets.

Google Colaboratory is free to use, and there is no need to sign up for any accounts or pay for any subscriptions. You can share your notebooks with others and work on projects together.

1. Create a Colaboratory Python Jupyter Notebook

Go to **Colab** to create a new Python Jupyter notebook. **Figure 2-13** shows the home screen.

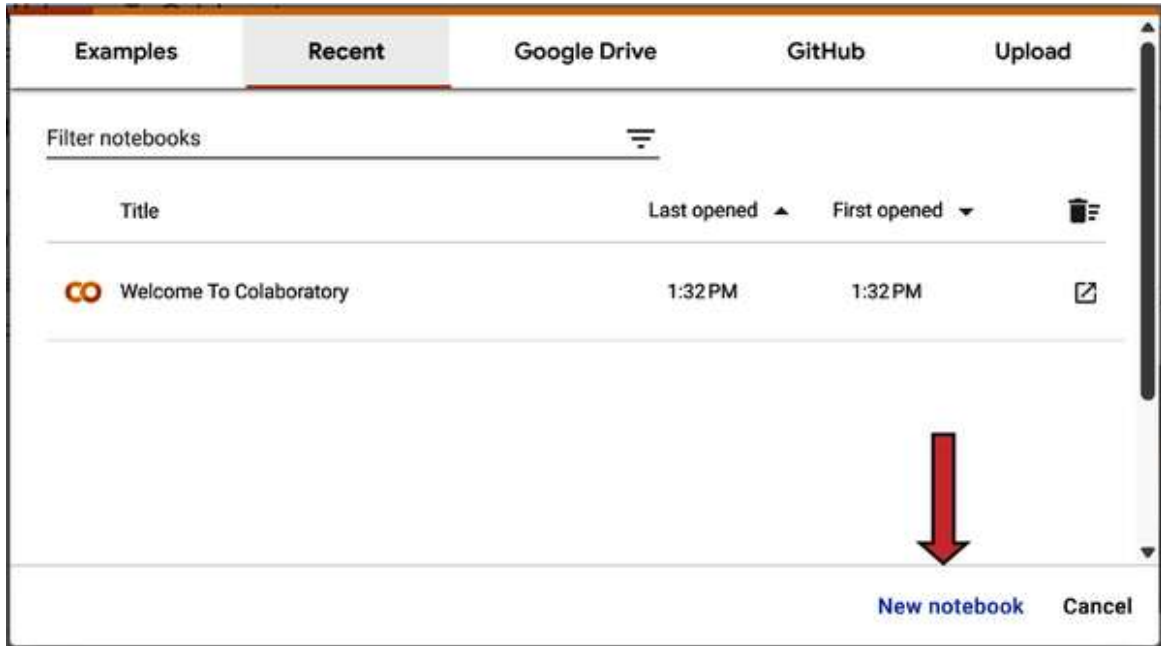


Figure 2-13. Google Colab home page.

Title the notebook in the title bar as shown in **Figure 2-14** (A) and expand to show the table of contents (B). Then click the + Code button (C) to add a cell to hold your code. The + Text button allows you to add text, such as documentation.

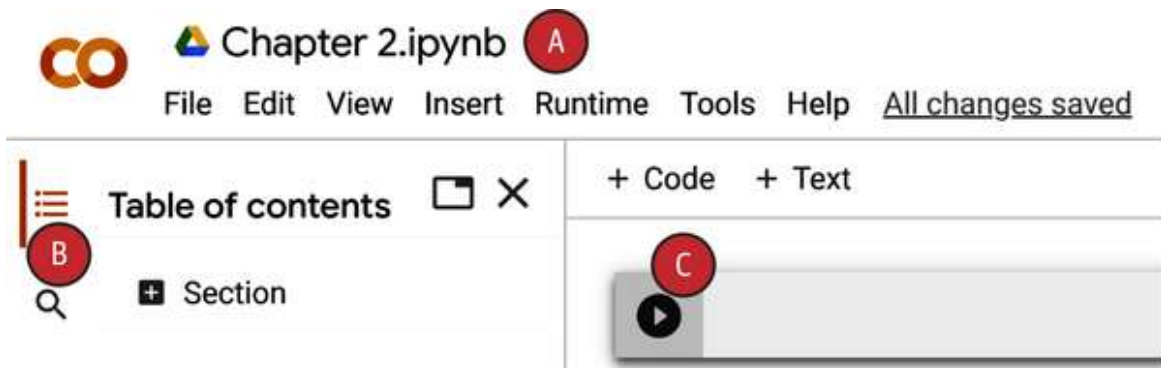


Figure 2-14. Title notebook and add a new cell code.

2. Import libraries and dataset using Pandas

Once you have added the code cell, you need to import any libraries you will need. In this simple example, you'll just import Pandas. Type **import pandas as pd** into the cell and run it by clicking the arrow, as shown in [Figure 2-15](#).



Figure 2-15. Code to import Pandas.

The Pandas library is used for data analysis. Typically, when you import a library, you want to provide a way to use it without having to write out the words *Pandas* each time. Thus, the *pd* is a short-hand name (or alias) for Pandas. This alias is generally used by convention to shorten the module and submodule names.

The dataset is from the [data.gov](#) website. It is entitled “[Heart Disease Mortality Data Among US Adults](#)” ([Figure 2-16](#)).

The screenshot shows a web interface for a data catalog. At the top, there is a blue header with the text "DATA CATALOG" and navigation links for "Datasets" and "Organizations". Below the header, a breadcrumb trail indicates the location: "U.S. Department of... / Centers for Disease...". A "Contact Data.gov" button is visible in the top right. The main content area features the U.S. Department of Health & Human Services logo on the left, which includes a circular seal with an eagle and the text "U.S. Department of Health & Human Services" and "Federal". To the right of the logo, the title "Heart Disease Mortality Data Among US Adults (35+) by State/Territory and County – 2018-2020" is displayed, followed by "Metadata Updated: November 2, 2022". A paragraph of text provides details: "2018 to 2020, 3-year average. Rates are age-standardized. County rates are spatially smoothed. The data can be viewed by gender and race/ethnicity. Data source: National Vital Statistics System. Additional data, maps, and methodology can be viewed on the Interactive Atlas of Heart Disease and Stroke <http://www.cdc.gov/dhds/maps/atlas>". Below this, an "Access & Use Information" section contains two items: "Public: This dataset is intended for public access and use." and "License: See this page for license information."

Figure 2-16. Heart disease mortality data among US adults by region.

Scroll down the page until you get to the section shown in **Figure 2-17**. Now, there are two ways you can import the file into your Jupyter Notebook. You can download the file to your desktop and then import it, or you can use the URL. Let's use the URL method. Click on the Comma Separated Values File shown in **Figure 2-17**, which takes you to the URL download shown in **Figure 2-18**.

Downloads & Resources






	Comma Separated Values File	Download
	RDF File	Download
	JSON File	Download
	XML File	Download
	Landing Page	Visit page

Figure 2-17. Downloads and resources page.

DATA CATALOG

[Home](#) / [U.S. Department of Health...](#) / [Heart Disease Mortality...](#)

Comma Separated Values File

URL <https://data.cdc.gov/api/views/jiwm-ppbh/rows.csv?accessType=DOWNLOAD>

Figure 2-18. Comma separated values file URL link.

Copy the URL shown in [Figure 2-18](#) from the website. Then, go to your Google Colab notebook and type in the code shown in [Figure 2-19](#) into a new cell (A). Run the cell by clicking the arrow (B).



The screenshot shows a Google Colab notebook interface. At the top, there are tabs for '+ Code' and '+ Text', and a status bar showing 'RAM' and 'Disk' usage. Below the tabs, there are two code cells. The first cell, labeled 'A', contains the following code:

```
1 import pandas as pd
2 url = 'https://data.cdc.gov/api/views/jiwm-ppbh/rows.csv?accessType=DOWNLOAD'
3 heart_df = pd.read_csv(url, index_col=0)
```

 The second cell, labeled 'B', is the run button for the first cell, indicated by a red circle with a white arrow.

Figure 2-19. Code to read the URL into a Pandas DataFrame.

You have written code to import the dataset into a Pandas DataFrame. A Pandas DataFrame is a two-dimensional data structure that is used to store data in a table format. It is similar to a spreadsheet.

Now you add code to show the first five rows (or *head*) of the DataFrame. Add a new cell, type **heart_df.head()** into the cell, and run the cell. The code and output are shown in [Figure 2-20](#).

```
heart_df.head()
```

GeographicLevel	DataSource	Class	Topic	Data_Value	Data_Value_Unit	Data_Value_Type
County	NVSS	Cardiovascular Diseases	Heart Disease Mortality	182.4	per 100,000 population	Age-adjusted, Spatially Smoothed, 3-year Avera...
County	NVSS	Cardiovascular Diseases	Heart Disease Mortality	172.6	per 100,000 population	Age-adjusted, Spatially Smoothed, 3-year Avera...
County	NVSS	Cardiovascular Diseases	Heart Disease Mortality	255.6	per 100,000 population	Age-adjusted, Spatially Smoothed, 3-year Avera...
County	NVSS	Cardiovascular Diseases	Heart Disease Mortality	343.4	per 100,000 population	Age-adjusted, Spatially Smoothed, 3-year Avera...
County	NVSS	Cardiovascular Diseases	Heart Disease Mortality	NaN	per 100,000 population	Age-adjusted, Spatially Smoothed, 3-year Avera...

Figure 2-20. First five rows of the DataFrame. Some columns were removed for the sake of readability.

Add a new code cell. Type **heart_df.info()** and run the cell to see information on the DataFrame. The `.info()` method gives you information on your dataset. The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values). **Figure 2-21** shows the output. Exact values may differ depending on when data is downloaded.

```
heart_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 59094 entries, 2019 to 2019
Data columns (total 19 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   LocationAbbr                          59094 non-null  object
1   LocationDesc                          59094 non-null  object
2   GeographicLevel                       59094 non-null  object
3   DataSource                            59094 non-null  object
4   Class                                59094 non-null  object
5   Topic                                59094 non-null  object
6   Data_Value                           33087 non-null  float64
7   Data_Value_Unit                      59094 non-null  object
8   Data_Value_Type                      59094 non-null  object
9   Data_Value_Footnote_Symbol           26007 non-null  object
10  Data_Value_Footnote                   26007 non-null  object
11  StratificationCategory1               59094 non-null  object
12  Stratification1                       59094 non-null  object
13  StratificationCategory2               59094 non-null  object
14  Stratification2                       59094 non-null  object
15  TopicID                              59094 non-null  object
16  LocationID                            59094 non-null  int64
17  Y_lat                                59076 non-null  float64
18  X_lon                                59076 non-null  float64
dtypes: float64(3), int64(1), object(15)
memory usage: 9.0+ MB
```

Figure 2-21. DataFrame information output.

From what the `.info()` output shows, you have 15 string object columns (which is qualitative data) and 4 numeric columns (quantitative data). Think of `int64` as a number without a decimal (for example, 25) and `float64` as a number with a decimal (25.5).

3. Data validation

As a best practice, validate any data you import from a URL—especially if you have a CSV file format to compare it with. If the dataset page had listed more metadata about the data, such as the number of columns and the

column names, you could have avoided the steps to follow. But alas, it is the nature of working with data!

Now, return to the data.gov page and download the CSV file to your computer. You are going to validate that the file you have downloaded matches the file you imported from the URL.

You do this by uploading the downloaded file to your Colab notebook and then reading that file into a Pandas DataFrame. Expand the table of contents in your [Chapter 2](#) notebook by selecting the folder shown in [Figure 2-22](#) (A). Then, to upload a file, select the up arrow folder (B).

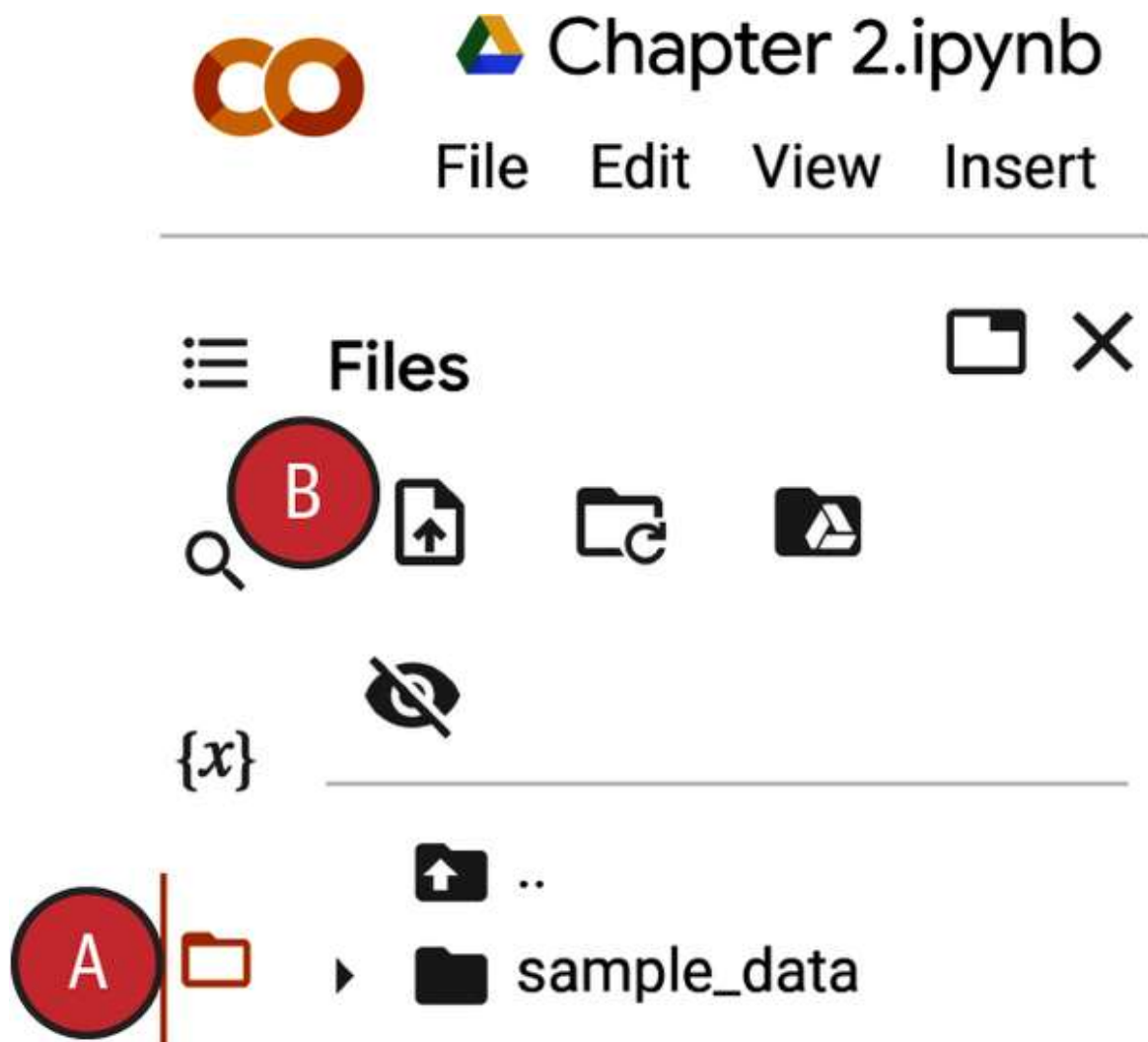


Figure 2-22. Upload file to your Colab notebook.

As you upload the file, you will see the warning message shown in **Figure 2-23**. This warning basically states that any file you upload will not be saved if the runtime is terminated (which can happen if you close out of Colab). Note that runtime provides the program with the environment it needs to run.

Warning

Ensure that your files are saved elsewhere. This runtime's files will be deleted when this runtime is terminated.

[More info](#)

OK

Figure 2-23. Warning message that any uploaded files are not permanently saved.

Refresh your notebook browser tab after the upload and expand it to see the table of contents. Your screen should look as shown in **Figure 2-24**.

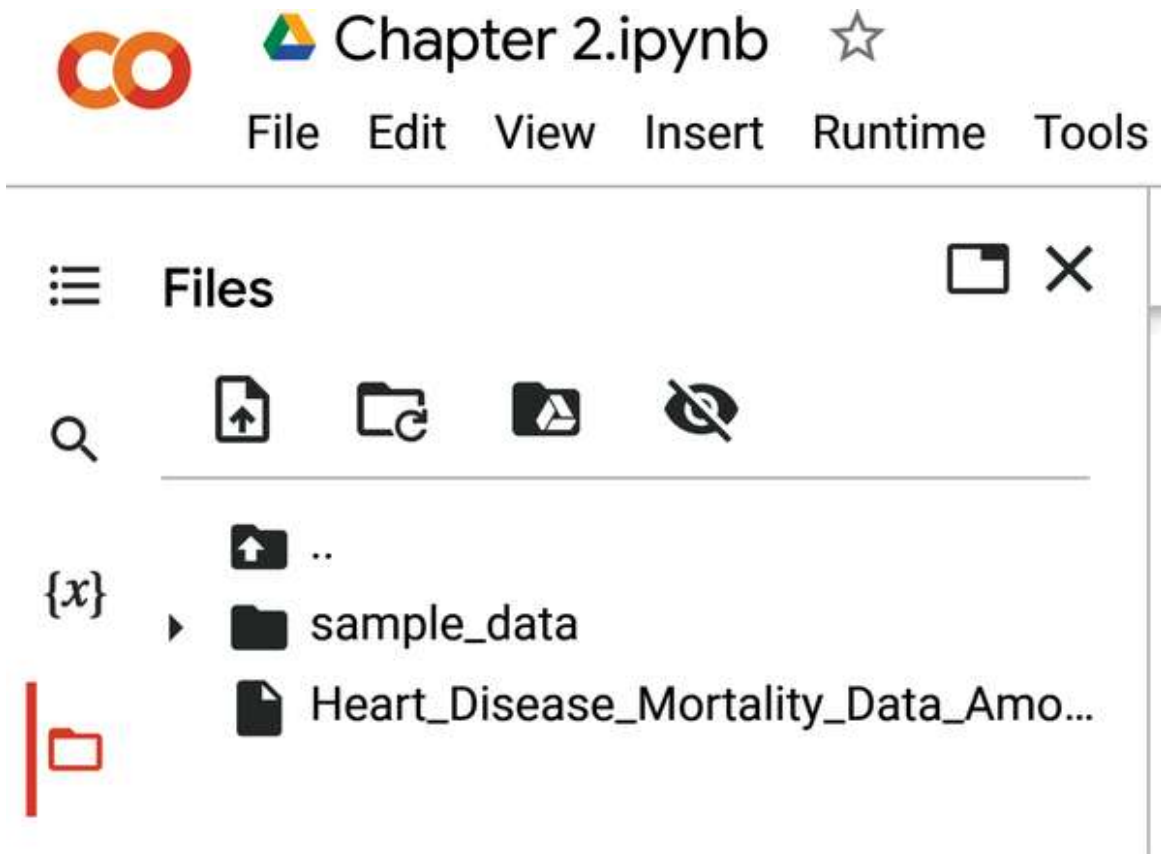


Figure 2-24. Table of contents that shows uploaded file.

Note how long the filename is—go ahead and rename it by right-clicking on the file and renaming it *heart.csv*. Your screen should look as shown in [Figure 2-25](#).

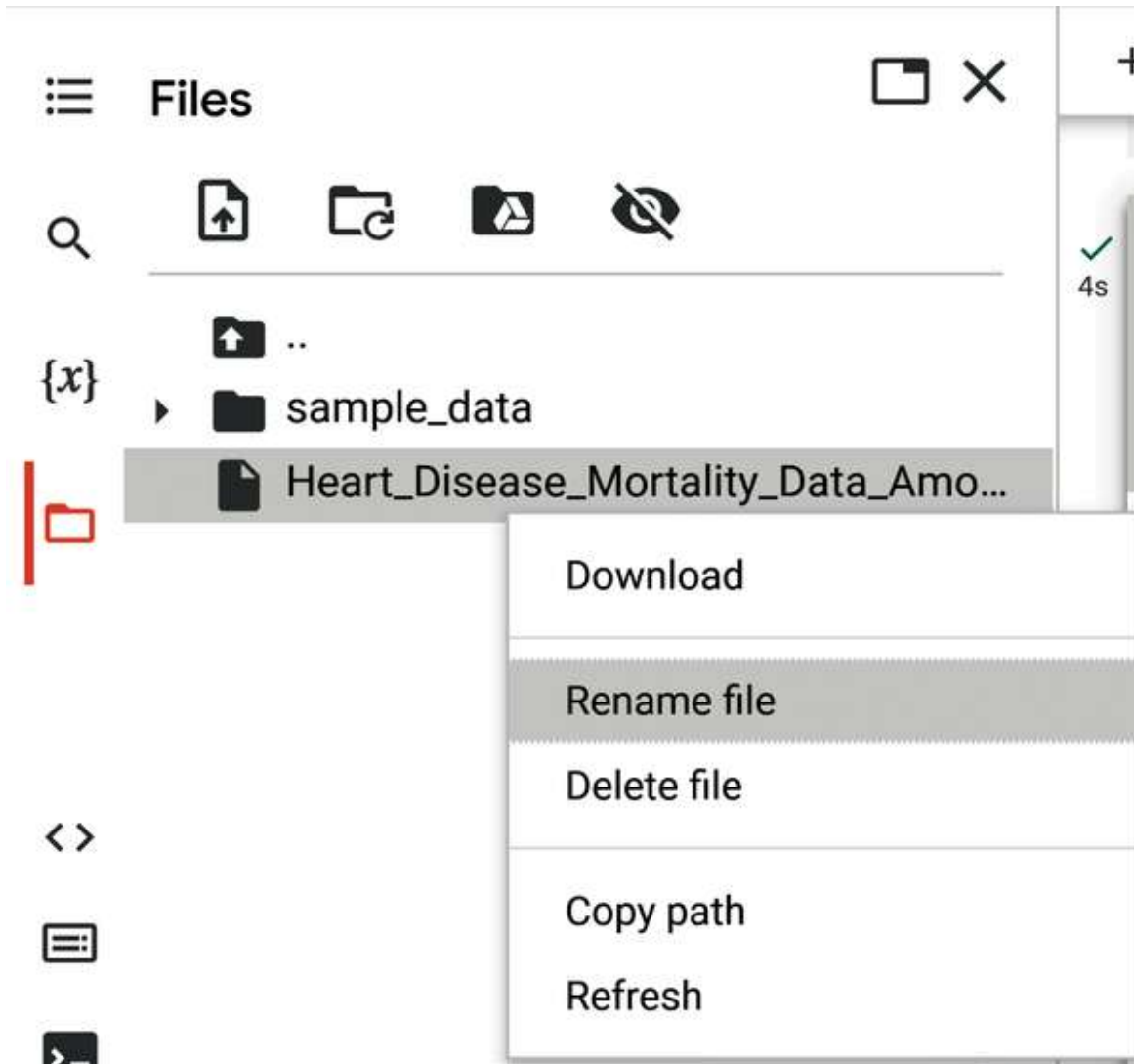


Figure 2-25. Select “Rename file” option.

Your screen should look as shown in [Figure 2-26](#) after renaming the file.

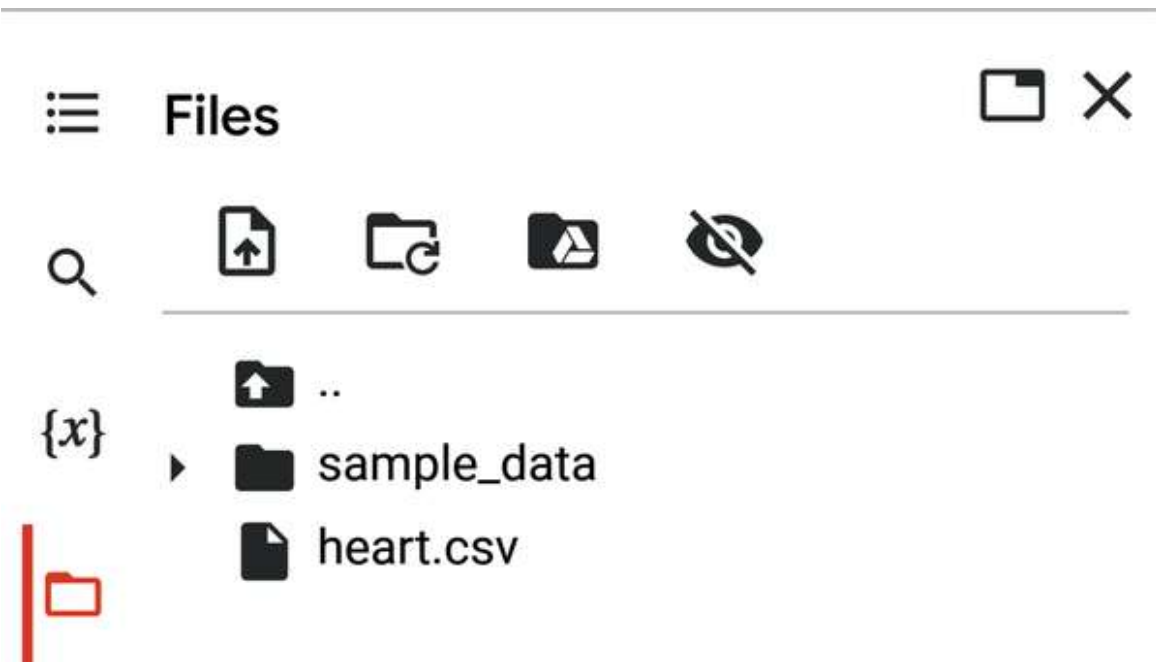


Figure 2-26. File renamed to *heart.csv*.

So, you've renamed your file to *heart.csv*. Now you need to copy the path of the file, as shown in [Figure 2-27](#). Why? You will need the exact location to input as a parameter in the Pandas `read.csv` method. Right-click on the *heart.csv* file to get the path.

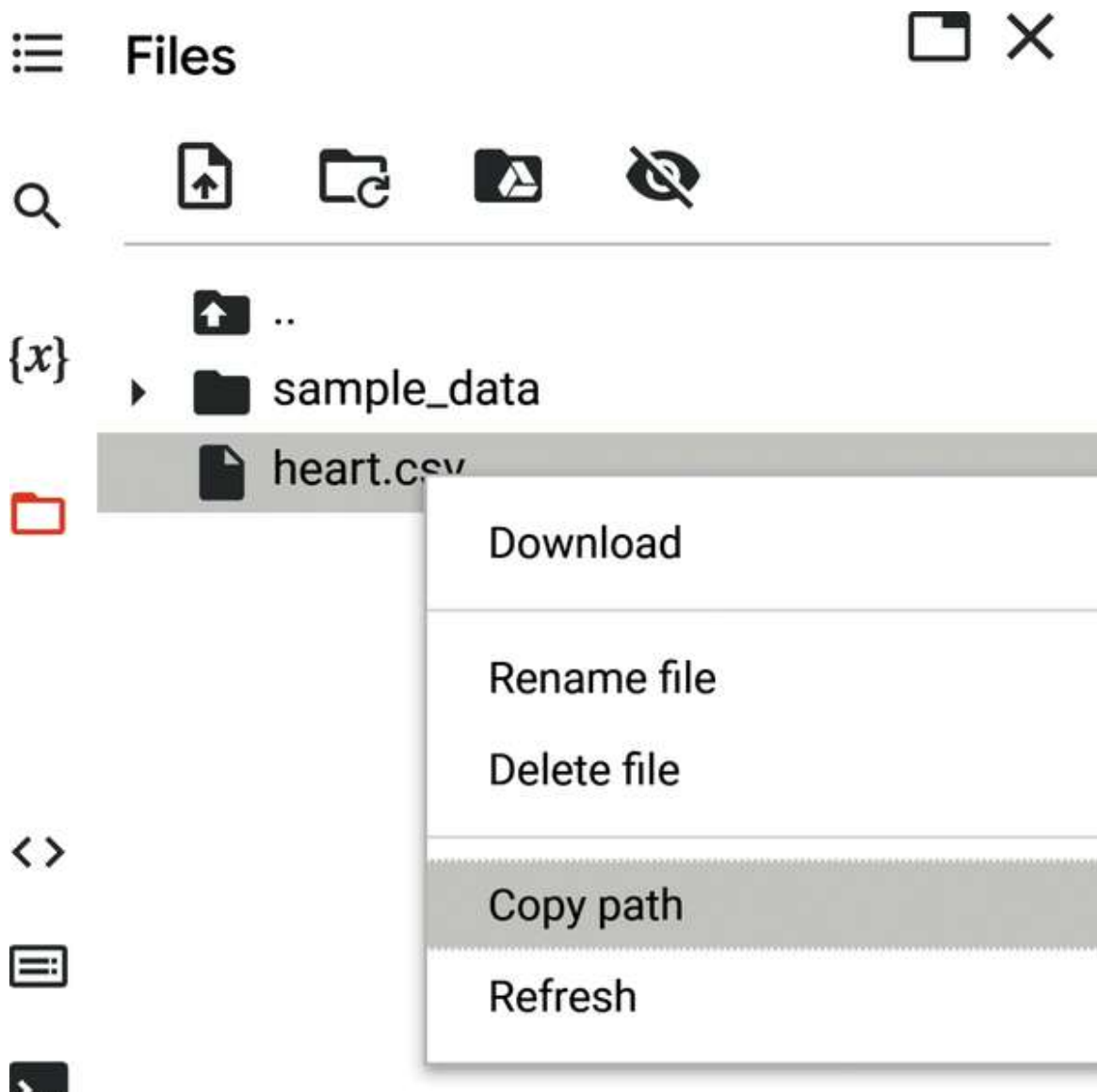


Figure 2-27. Copying the path of the file.

Add a code cell and type in the following code. Make sure to paste the path to the file between the two single quote marks around `/content/heart.csv`. Run the cell and review the output:

```
heart_df = pd.read_csv('/content/heart.csv', error_bad_lines=False,  
                        engine="python")  
heart_df.head()
```

Type `heart_df.info()` into a new cell. Run the cell to see DataFrame information. Compare the total columns from Figures 2-21 to 2-28.

Figure 2-21 has 19 columns and Figure 2-28 has 20 columns. This means that by uploading the file, a Year column was added. The Year datatype is not a numeric value—it has numbers, but those numbers are meant to rank and order, not calculate on. Figure 2-28 indicates you have your first case of dirty data for our machine learning use case.

▶ Data columns (total 20 columns):

#	Column	Non-Null Count	Dtype
0	Year	41071 non-null	int64
1	LocationAbbr	41071 non-null	object
2	LocationDesc	41071 non-null	object
3	GeographicLevel	41071 non-null	object
4	DataSource	41071 non-null	object
5	Class	41071 non-null	object
6	Topic	41071 non-null	object
7	Data_Value	23316 non-null	float64
8	Data_Value_Unit	41071 non-null	object
9	Data_Value_Type	41071 non-null	object
10	Data_Value_Footnote_Symbol	17755 non-null	object
11	Data_Value_Footnote	17755 non-null	object
12	StratificationCategory1	41071 non-null	object
13	Stratification1	41071 non-null	object
14	StratificationCategory2	41071 non-null	object
15	Stratification2	41071 non-null	object
16	TopicID	41071 non-null	object
17	LocationID	41071 non-null	int64
18	Y_lat	41071 non-null	float64
19	X_lon	41071 non-null	float64

dtypes: float64(3), int64(2), object(15)

Figure 2-28. Year column showing as *int64*.

Type `heart_df.isnull().sum()` into a new cell as shown in Figure 2-29. Run the cell. Are there any null values? A *null value* is a value that indicates the absence of a value. You have your second case of dirty data! Three of your columns have null values. You will learn how to deal with missing values in a subsequent chapter. Since you really don't have a use case yet, you may be wondering whether you really need all of these

features. Note that features are the input data that is used to train a model. The model then uses the features to make predictions. You'll learn about feature selection in a subsequent chapter.



```
heart_df.isnull().sum()
```

```
Year      0
LocationAbbr  0
LocationDesc  0
GeographicLevel  0
DataSource  0
Class      0
Topic      0
Data_Value 17755
Data_Value_Unit  0
Data_Value_Type  0
Data_Value_Footnote_Symbol 23316
Data_Value_Footnote 23316
StratificationCategory1  0
Stratification1  0
StratificationCategory2  0
Stratification2  0
TopicID      0
LocationID    0
Y_lat        0
X_lon        0
dtype: int64
```

Figure 2-29. `IsNull` output showing the number of null cells in the three columns with null values.

What are your data quality issues?

Missing values

Most algorithms do not accept missing values. Therefore, when we see missing values in our dataset, there may be a tendency to just “drop all the rows” with missing values. However, there are various ways you can deal with missing values, as explained in the following note.

NOTE

There are two main approaches to handling missing values in a dataset in ML: deletion or imputation.

Deletion involves removing the rows or columns with missing values from the dataset. This can be done by dropping all rows with missing values, dropping all columns with missing values, or dropping rows or columns with a certain threshold of missing values.

Imputation involves filling in the missing values with estimates. There are many different imputation techniques, including (1) mean imputation, which replaces missing values with the mean of the observed values for that variable; (2) median imputation, which replaces missing values with the median of the observed values for that variable; (3) mode imputation, which replaces missing values with the most frequent value of the variable; and (4) regression imputation, which uses a regression model to predict the missing values based on the observed values of other variables.

Although Pandas will fill in the blank space with *NaN* (*not a number*), we should handle them in some way. More on that later in the book.

Data type incorrect

Year is shown as an int64 data type and should be a string object—where you would need to handle it as a qualitative, categorical feature.

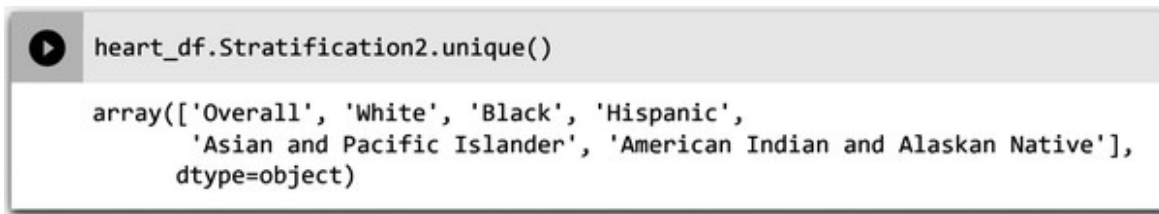
Categorical columns

There are quite a number of string object features—which are not numeric. You cannot feed values like this into an ML model. These features need to be *one-hot encoded*. You’ll see this in a subsequent chapter.

4. A little bit of exploratory data analysis

Before concluding this section, let’s look at some simple ways to explore the data. Want to see all of the unique values in the feature

Stratification2? Type **heart_df.Stratification2.unique()** into a new cell as shown in [Figure 2-30](#). Run the cell.



```
heart_df.Stratification2.unique()

array(['Overall', 'White', 'Black', 'Hispanic',
       'Asian and Pacific Islander', 'American Indian and Alaskan Native'],
      dtype=object)
```

Figure 2-30. Code to show unique values of column Stratification2.

Let’s use Seaborn’s violin plot to visualize this feature. Seaborn is a Python library for making statistical graphics. You can write all of this in the same cell, as shown in [Figure 2-31](#). This code uses the Data_Value feature as the *x* and the Stratification2 as the *y*. Note that the Data_Value feature is the count of heart disease in each region and by group, as shown in [Figure 2-32](#).



```
import seaborn as sns  
sns.violinplot(x="Data_Value", y="Stratification2", data=heart_df)
```

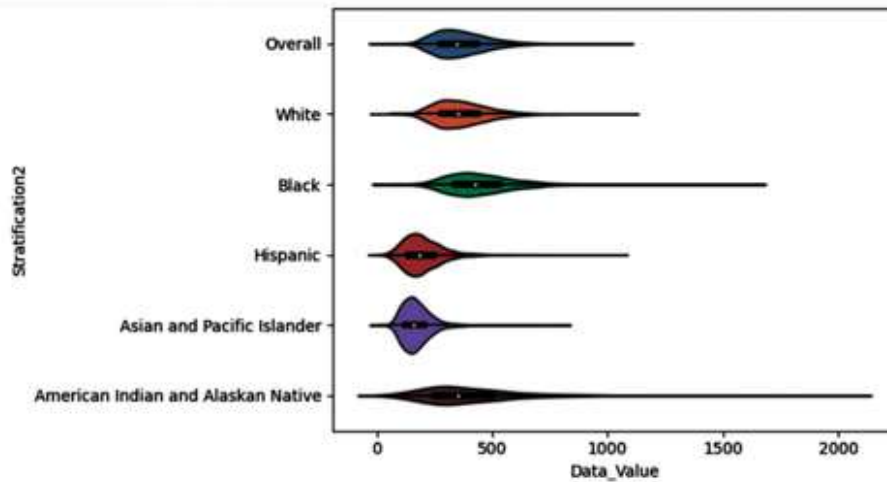


Figure 2-31. Violin plot of *Stratification2* column.

The “long tails” indicate there are more outliers in the data for that particular feature. The large shape of the violin body indicates how many heart disease cases are distributed by race.

```
sns.displot(heart_df['Data_Value'])
```

<seaborn.axisgrid.FacetGrid at 0x7ff879aa0e80>

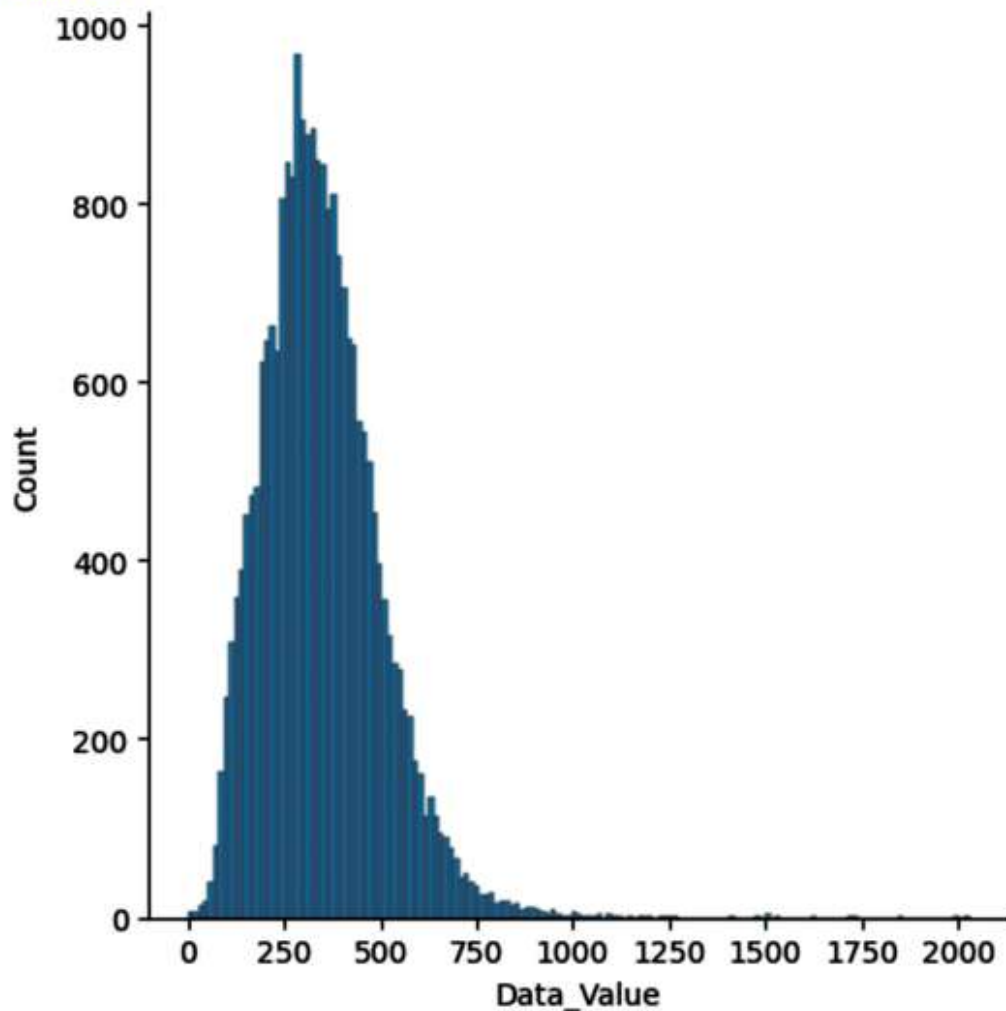


Figure 2-32. Distribution plot of heart disease count.

NOTE

Seaborn's violin plot is a good way to visualize the distribution of a univariate set of data.

It is used to visualize the distribution of numerical data. Unlike a box plot that can only show summary statistics, violin plots depict summary statistics and the density of each variable.

In later chapters, you'll perform more data analysis and learn data preprocessing. For now, save your notebook to GitHub. On the Colab menu, click File > Save a copy in GitHub, as shown in [Figure 2-33](#).

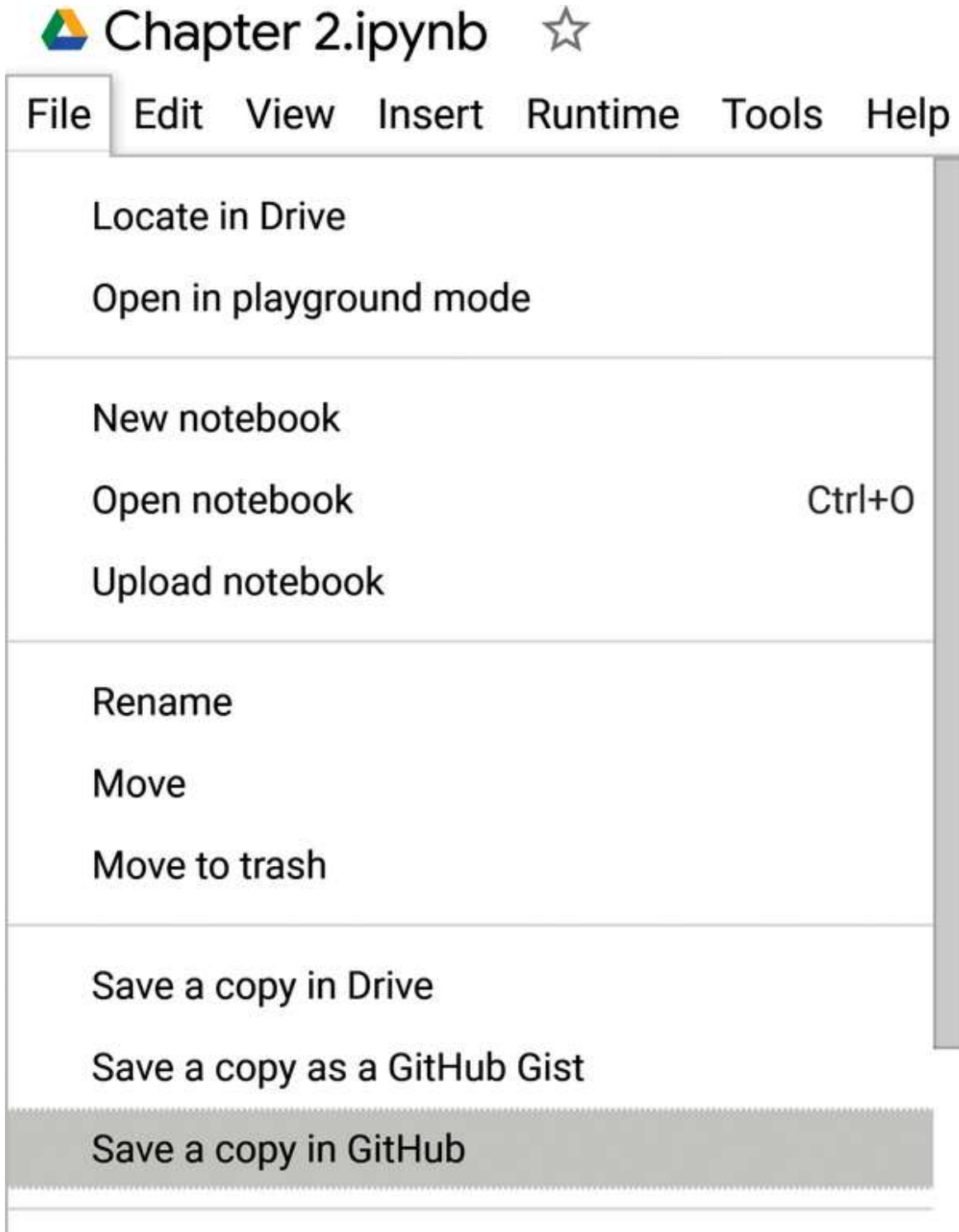


Figure 2-33. How to save a file copy of the notebook in GitHub.

Figure 2-34 shows the setup. From the drop-down under Repository, select the repo to save your notebook to. Under Branch, select Notebooks (you created this earlier). Keep the default “Include a link to Colaboratory.” This will show a link in GitHub that allows you to open your notebook directly from GitHub.

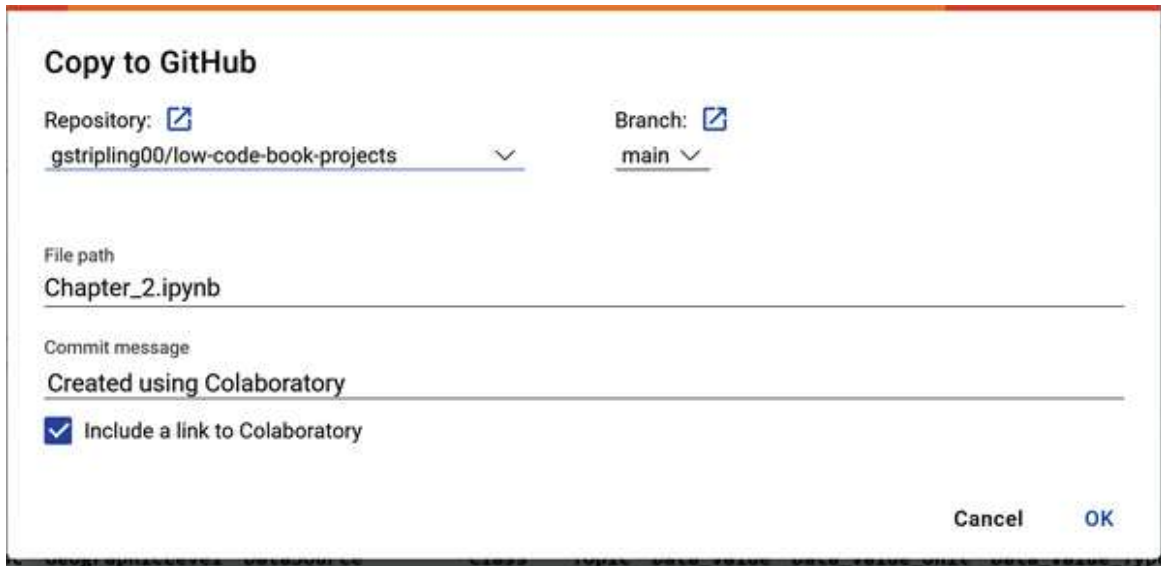




Figure 2-34. Colab’s GitHub export window.

After Colab copies the notebook to GitHub, it takes you directly to GitHub, as shown in Figure 2-35.

 **gstripling00** Created using Colaboratory

1 contributor

2542 lines (2542 sloc) | 131 KB

 [Open in Colab](#)

```
In [ ]: import pandas as pd
```

```
In [ ]: # Next, you read the dataset into a Pandas dataframe.

url = ''
heart_df= pd.read_csv(url,index_col=0)
```

```
In [ ]: # Next, you read the dataset into a Pandas dataframe.

url = 'https://data.cdc.gov/api/views/jiwm-ppbh/rows.csv?accessType=DOWNLOAD'
heart_df= pd.read_csv(url,index_col=0)
```

```
In [ ]: heart_df.head()
```

Figure 2-35. Notebook copied to GitHub.

Refresh the screen in the repo. You should see your Colab Jupyter notebook, as shown in **Figure 2-36**.

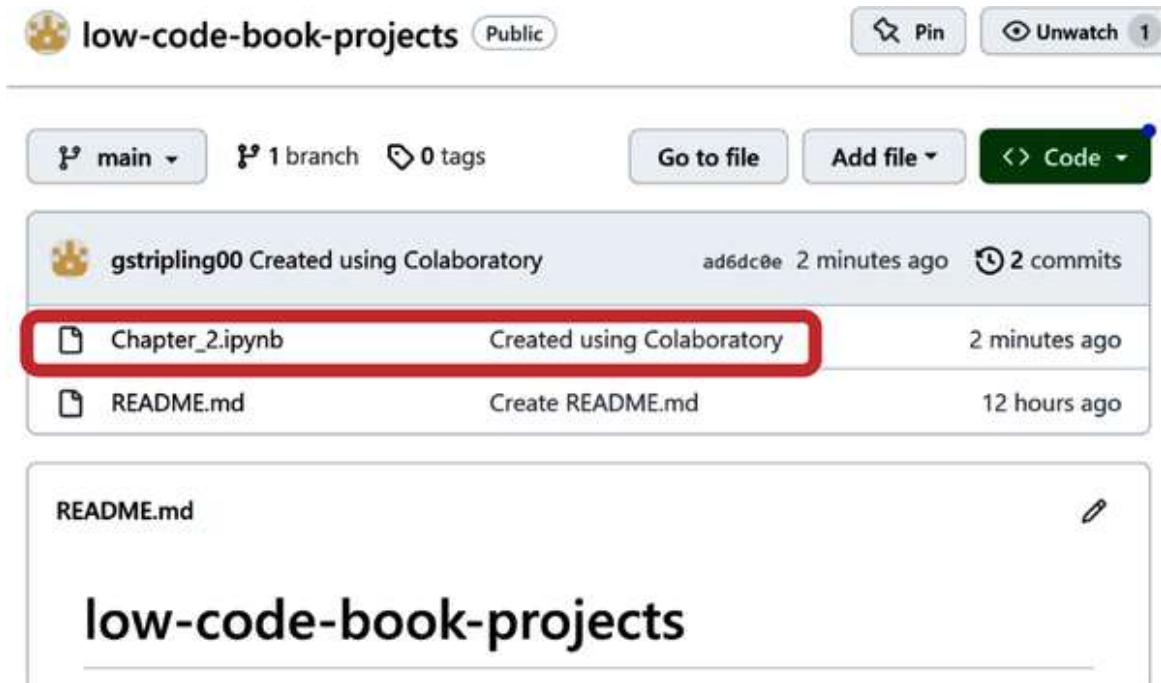


Figure 2-36. Copy of Colab Jupyter notebook in GitHub.

Summary

This chapter provided an overview of the use cases and datasets used in the book. You learned about data types and the difference between structured, semistructured, and unstructured and batch and streaming data. You got hands-on practice with a free browser-based Python Jupyter Notebook and GitHub. You discovered that dirty data is tricky and can impact data type and data ingestion into an ML model.

In the next chapter, you'll learn about ML frameworks and you'll get to work with AutoML. You are given a preprocessed dataset, and all you'll have to do is upload the dataset into the AutoML framework and you will be able to build and train a predictive model without writing a single line of code.