R_intro_tdmstudio

October 20, 2025

1 R Introduction to TDM Studio

This is a brief presentation of the capabilities of ProQuest's TDM Studio, modeled in part on the introductory R materials in the Workbench and on Moacir de Sá Pereira's TDM Studio notebook.

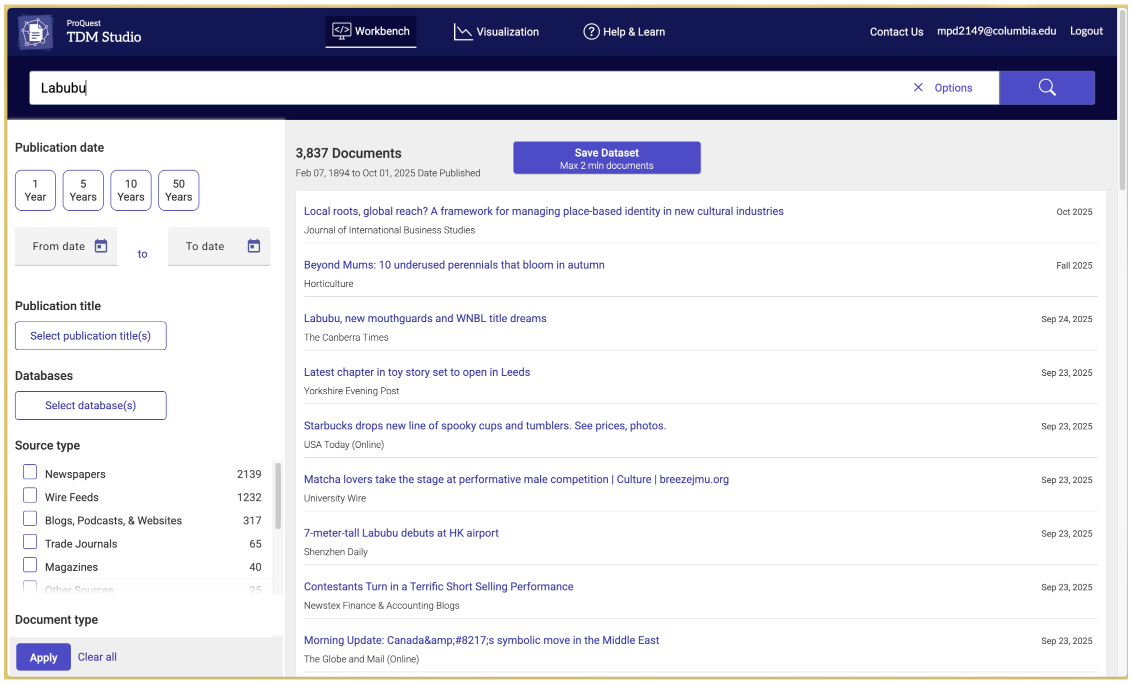
I'm using R here, but there are Python materials available in the Workbench and on the Columbia Data Club's GitHub repository.

There are three steps to using TDM Studio Workbench:

1. Search ProQuest using search tools
2. Filter the results and save them as a dataset
3. Read the dataset into your TDM Studio Jupyter notebook

1.1 Search ProQuest and Filter Results

You can use all the ProQuest field codes to search all the databases.



Labubu

3,837 Documents
Feb 07, 1894 to Oct 01, 2025 Date Published

Newspapers	2139
Wire Feeds	1232
Blogs, Podcasts, & Websites	317
Trade Journals	65
Magazines	40

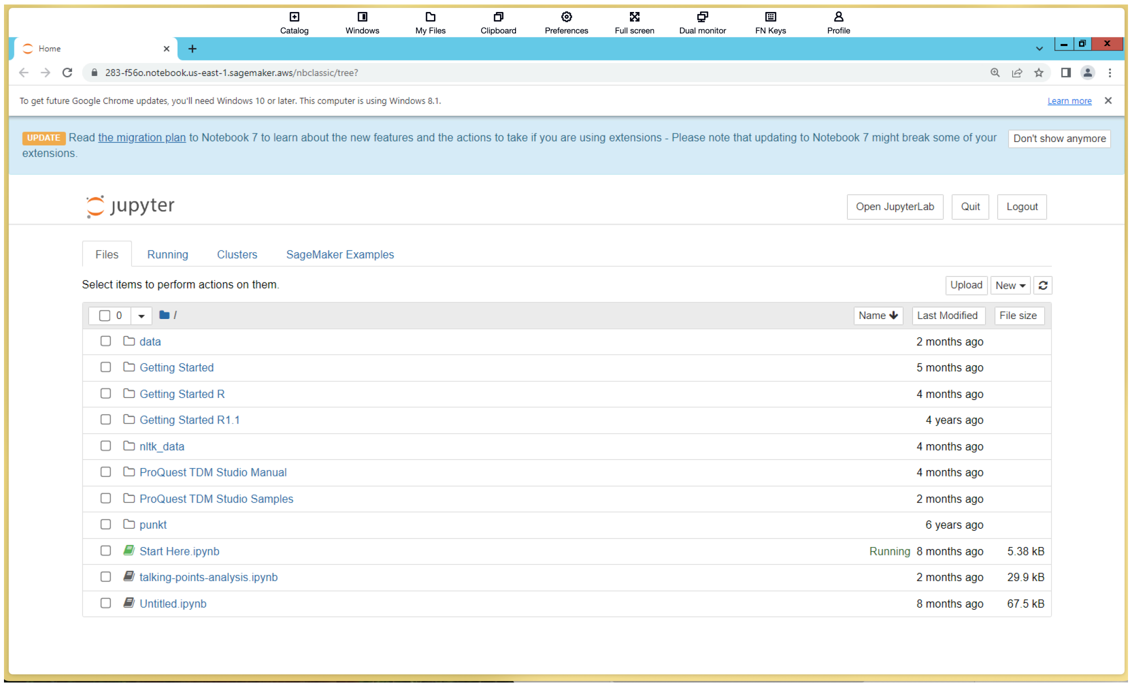
Local roots, global reach? A framework for managing place-based identity in new cultural industries Journal of International Business Studies	Oct 2025
Beyond Mums: 10 underused perennials that bloom in autumn Horticulture	Fall 2025
Labubu, new mouthguards and WNBL title dreams The Canberra Times	Sep 24, 2025
Latest chapter in toy story set to open in Leeds Yorkshire Evening Post	Sep 23, 2025
Starbucks drops new line of spooky cups and tumblers. See prices, photos. USA Today (Online)	Sep 23, 2025
Matcha lovers take the stage at performative male competition | Culture | breezejmu.org University Wire	Sep 23, 2025
7-meter-tall Labubu debuts at HK airport Shenzhen Daily	Sep 23, 2025
Contestants Turn in a Terrific Short Selling Performance Newstex Finance & Accounting Blogs	Sep 23, 2025
Morning Update: Canada&#8217;s symbolic move in the Middle East The Globe and Mail (Online)	Sep 23, 2025

Data sets can have no more than 2 million documents, and you can only have 10 datasets at once, but the actual limits on how many articles you can analyze is determined by your workbench's disk size.

Once the dataset is saved, in the background, TDM Studio starts copying it to your Workbench's `data/` folder.

1.2 Reading Data into your Jupyter Notebook

Simply put, your Workbench is a virtual Windows computer managed by ProQuest, and the only thing you can do on this computer is run Jupyter. Because it is a virtual computer, copying information and files to and from it is a convoluted process, but one you can get the hang of reasonably quickly.



283-f56o.notebook.us-east-1.sagemaker.aws/nbclassic/tree?

To get future Google Chrome updates, you'll need Windows 10 or later. This computer is using Windows 8.1. Learn more

UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions. Don't show anymore

Select items to perform actions on them.

Name		Last Modified	File size
data		2 months ago	
Getting Started		5 months ago	
Getting Started R		4 months ago	
Getting Started R1.1		4 years ago	
nltk_data		4 months ago	
ProQuest TDM Studio Manual		4 months ago	
ProQuest TDM Studio Samples		2 months ago	
punkt		6 years ago	
Start Here.ipynb	Running	8 months ago	5.38 kB
talking-points-analysis.ipynb		2 months ago	29.9 kB
Untitled.ipynb		8 months ago	67.5 kB

Your datasets appear in the data folder, and the format of each dataset is the same: a large number of XML files, one for each article in your dataset, each named with the unique ID ProQuest uses for the article.

1.3 Using the Terminal to Examine XML Structure

Let's navigate in a Terminal window to examine one of the XML files. From the root directory, press the `New` dropdown menu on the right and select Terminal. Enter these unix commands *one at a time* in Terminal, followed by return/enter, to navigate and inspect one of the XML files. Alternatively, follow along with the instructions in this PQ video.

[ ]:
```
# cd means change directory; ls means list files in current directory
cd Sagemaker/data
ls
cd [INSERT NAME OF DATASET]
ls
# use vim to inspect one XML document; exit vim using `:q!` command
```

```
vim [docid].xml
```

XML is a rich, text-based data format that nests information inside tags. It's somewhat comparable to a deeply structured JSON file or a deeply nested Python dictionary.

1.4 Using this Handout in TDM Studio

To use this handout in TDM Studio, we will first need to upload it to the Windows virtual machine. You may wish to follow along with this video prepared by the ProQuest team. Briefly, this is a two step process, outlined below:

1. In the jupyter notebook interface, click on "My Files" in the uppermost navigation window
2. Select Temporary Files and press "Upload file(s)"
3. Navigate to where you saved this handout saved on your local drive and select "Open"
4. When the file is uploaded, close the "My Files" dialog
5. In the Windows VM, move the file to the jupyter notebook application; navigate to where you want to store the file and press the "Upload" button on the right
6. Select `My Files/Temporary Files/R_intro_tdmstudio.ipynb` and press "Open"
7. Next to the file you just uploaded in the jupyter interface, you will see an "Upload" button; press it

Now you should be ready to go!

```
[ ]: # Set timezone env variable to UTC for our date extraction library
     Sys.setenv("TZ"="ECT/UTC")

     # Load libraries
     library(xml2);           # XML processing
     library(rvest);          # Cleanly strip HTML from document text
     library(parallel);       # Multi-core processing
     library(data.table);     # Fast reading/writing data
     library(dplyr);          # Data manipulation
     library(ggplot2);        # Visualizations
     library(clock);
```

Begin by making sure our output directory exists. This is the directory in which the output CSV will be created.

```
[ ]: if (!dir.exists("../output_files/")) {
         dir.create("../output_files/")
     }
```

1.5 Create Dataset

Create a list of XML files to process into a dataframe. Please replace the name `Nobel_Prize_in_literature_mentions_in_WaPo` with the name of your dataset.

```
[ ]: dataset_name = 'Nobel_Prize_in_literature_mentions_in_WaPo'
```

```
xml_files <- list.files(paste0('/home/ec2-user/SageMaker/data/', dataset_name),␣
↪pattern = "*xml$", full.names = TRUE)
print(paste("Files in dataset", length(xml_files)))
```

1.6 Visualizing the XML Structure

Run this command to print XML node structure as a list. This is just to help use see the XML structure and find data to extract.

```
[ ]: as_list(read_xml(xml_files[1]))
```

1.7 Extract Function

This creates the function to extract data from the XML structure, and can be altered to extract other XML nodes. This function will process a single XML file, extract the indicated data, and append it to a CSV file named after the dataset.

To add additional nodes, extract the contents of the node of interest into a variable, as in F.2, then add the variable to the list in F.3.

```
[ ]: extract <- function(filename, dataset) {

         # F.1
         # Read a single XML document
         doc_xml <- read_xml(filename)

         # F.2
         # Find and extract desired data
         Article_ID <- xml_find_all(doc_xml, "//GOID") %>% xml_text()
         Title <- xml_find_all(doc_xml, "//TitleAtt/Title") %>% xml_text()
         Date <- xml_find_all(doc_xml, "//NumericDate") %>% xml_text()

         # This section uses `tryCatch()` to test if a TextInfo node exists, and if␣
       ↪not, sets Text variable to NA
         tryCatch(
             error = function(cnd) Text <<- NA,
             Text <- xml_find_all(doc_xml, "//TextInfo") %>% xml_text() %>%␣
       ↪read_html() %>% html_text()
         )
         # F.3
         # Combine the data into a dataframe and return
         df <- as.data.table(cbind(Article_ID,Title,Date,Text))
         fwrite(df,paste0("../output_files/", dataset, ".csv"), append=T)
         return()
     }
```

1.8 Parallel Processing

Here we apply the function to our list of documents using the `mclapply()` function. The data from each document will be appended to a CSV file named as the dataset name. The CSV file can be found in the directory in which this notebook is run.

[ ]:
```
# Take system time
start <- proc.time()

# remove CSV if it already exists
if (file.exists(paste0("../output_files/", dataset_name, '.csv'))) {
    # invisible is used here to suppress printing boolean output.
    invisible(file.remove(paste0("../output_files/", dataset_name, '.csv')))
}
tryCatch(
    # invisible is used here to suppress printing output. Otherwise `mclapply`␣
↪would print a vector of all null
    invisible(mclapply(xml_files,extract,dataset=dataset_name,mc.cores =␣
↪detectCores())),
    warning = function(w) {
        message("A warning occurred and the csv may not have been created in␣
↪full.")
        message("This is often due to insufficient disc space to continue␣
↪writing the csv.")
    },
    error = function(e) {
        message("An error occurred and the csv may not have been created in␣
↪full.")
        message(paste("Error message: ", e))
    }
)

# Show elapsed time
difference <- proc.time() - start
print(paste("Creating the csv file took:", format(difference[3]), "seconds."))
```

1.9 Explore the Data

Prints out the first 3 entries to ensure the data is correct.

[ ]:
```
fread(file=paste0("../output_files/", dataset_name,'.csv'), nrows=3,␣
↪integer64='character')
```

1.10 Save data to dataframe

Since we already created a csv in the steps above, we are reusing it to create a dataframe object called `df`.

```
[ ]: df <- fread(file=paste0("../output_files/", dataset_name,'.csv'),␣
      ↪integer64='character')

     # peek at the dataframe that was created
     glimpse(df)
```

1.11 Create a column for word count

Let's use the `strsplit()` function of base R to create a new column in our data frame containing the word counts of the Text value for each document.

```
[ ]: df$word_count <- lengths(strsplit(df$Text, "\\s+"))
     df$Article_ID <- as.numeric(df$Article_ID) # retype Article_ID to be a double␣
      ↪so we get fewer warnings in R
     glimpse(df) # have another peek to make sure our new column was added
```

1.12 Plot to Histogram

This is a histogram of the documents per each date range bin. The number of bins is set to 10, but may be set otherwise in the cell below.

```
[ ]: # Set the number of date bins for the histograms. May be changed as needed.
     num_bins = 10
```

```
[ ]: # Set plot size
     options(repr.plot.width=14,repr.plot.height=10,repr.plot.res=200)

     df %>% mutate(Date = as.character(Date)) %>%
     mutate(Date = date_parse(Date)) %>%
     ggplot() +
         geom_histogram(aes(x=Date), fill='#5542a6', bins=num_bins) +
         labs(
             x="Date",
             y="Document Count",
             title="Documents by Date Bin"
         ) +
         theme(
             plot.title = element_text(size=22),
             axis.title = element_text(size=16),
             axis.text = element_text(size=12),
             panel.background = element_rect(fill='grey87')
         )
```

1.13 Word Count Plot

This is a histogram of the documents per word count bin. This could be helpful if you were looking to eliminate very brief articles, for example. The number of bins remains the same as in the above plot.

[ ]:
```
# Set plot size
options(repr.plot.width=14,repr.plot.height=10,repr.plot.res=200)

ggplot(df) + geom_histogram(aes(x=word_count), fill='#5542a6', bins=num_bins) +
    labs(
        x="Word Count",
        y="Document Count",
        title="Documents by Word Count"
    ) +
    theme(
        plot.title = element_text(size=22),
        axis.title = element_text(size=16),
        axis.text = element_text(size=12),
        panel.background = element_rect(fill='grey87')
    )
```

This brief tutorial shows the possibilities with TDM Studio. I have the full text of the articles as a column and can work with the text in a variety of fashions, including feeding each article into model of some sort to have it analyze the sentiment or extract named entities.