Image created with ChatGPT

Using Machine Learning to Predict the Next Command in Revit _ Part 2: Data Preprocessing Using PowerShell & Python with PyRevit

Let's BIM Together
728 followers

September 25, 2024

(Part 2 of a 7 series blog. ***Click here to go to Part 1****)*

The origin of this project is as puzzling as the chicken or the egg scenario; I'm unsure if I first encountered the dataset that inspired *AG Feeling Lucky* or if the idea came first. Either way, I found a dataset on ***Kaggle.com*** that contains Revit commands in sequence. This dataset also includes additional information, such as the view in which the command was entered and the time. For simplicity, the focus here will be on just the command sequences. Notice the command IDs that start with "*ID_*" such as *ID_REVIT_FILE_CLOSE*. These command IDs are unique identifiers used to reference specific commands in the *Revit API* which most user-initiated operations in Revit have.

https://www.kaggle.com/datasets/mohammadghafaripour/revitdata

Download the file called "*Data_Revit*". It has no extension, so rename it to "*Data_Revit.csv*".

> Sample of Data_Revit.csv

```
04-Mar-21,19:44:35,Ribbon,Purge (delete) unused families and types
ID_PURGE_UNUSED,3D View: 3D-DALUX-Full
04-Mar-21,19:44:46,Ribbon,Save the active project with a new name
ID_REVIT_FILE_SAVE_AS,3D View: 3D-DALUX-Full
04-Mar-21,19:58:49,Ribbon,Close the active project ID_REVIT_FILE_CLOSE,3D
View: 3D-DALUX-Full
04-Mar-21,20:04:24,Ribbon,Close the active project ID_REVIT_FILE_CLOSE,3D
View: 3D-DALUX-Full
04-Mar-21,20:04:31,Internal,ID_REVIT_MODEL_BROWSER_OPEN,3D View: 3D-DALUX-
Full
04-Mar-21,20:04:37,Ribbon,Open an existing project ID_REVIT_FILE_OPEN,3D
View: 3D-DALUX-Full
```

In ***ML.NET***, data must be in a columnar format (a text file with rows as lines and columns separated by commas) with inputs and a correct answer for those inputs. In this example, there will be five columns: the first four corresponding to the last four commands entered and the fifth as the correct answer we are trying to predict. We will achieve this data manipulation via a series of *PowerShell* scripts. A series is used instead of one script due to my relative newness to *PowerShell*; breaking tasks into smaller modules minimizes errors and builds a library of reusable modules.

PowerShell scripts are text files ending with the "*.ps1*" extension, we will be executing them by right-clicking the file and selecting "*Run with PowerShell*". To expedite the process, all *PowerShell* scripts in this blog must be run in the same folder as the data input files being manipulated.

Also, *ChatGPT* is really good at *PowerShell*. For *script 1_SCRAPE-COMMANDS.ps1* (which is discussed below), you can use the following prompt to generate it:

> Prompt

```
Generate a PowerShell script that reads a file called Data_Revit.csv,
extracts all the command IDs starting with 'ID_' using a regular
expression, and writes them to a new file called Data_Revit_CMD_Only.csv.
If the output file already exists, clear its content first; if it doesn't,
create a new file. For each match found, display the extracted command ID
in the console. Finally, after processing all lines, write the collected
command IDs to the output file.
```

> Script 1: 1_SCRAPE-COMMANDS.*ps1*

Link to source code: ***https://pastebin.com/yYr1Qc8y***

This script takes an input file, *Data_Revit.csv,* then reads only command IDs starting with "*ID_*" and saves it as "*Data_Revit_CMD_Only.csv*"

> Sample of Data_Revit_CMD_Only.csv

```
ID_PURGE_UNUSED
ID_REVIT_FILE_SAVE_AS
ID_REVIT_FILE_CLOSE
ID_REVIT_FILE_CLOSE
ID_REVIT_MODEL_BROWSER_OPEN
ID_REVIT_FILE_OPEN
```

Next, we need to transform this into a five-column format:

```
command 1, command 2, command 3, command 4, command 5
command 6, command 7, command 8, command 9, command 10
```

If the transformation to a five-column format is done immediately, a significant portion of the training data will be lost. As shown above, each line provides two answers with a set of four inputs. For example, command 5 uses commands 1, 2, 3, and 4 as inputs, and command 10 uses 6, 7, 8, and 9. This setup misses entries for predicting commands 6, 7, 8, or 9. A simple solution is to make four copies of the *Data_Revit_CMD_Only.csv* file, named as follows:

"*Data_Revit_CMD_Only-1.csv*"

"*Data_Revit_CMD_Only-2.csv*"

"*Data_Revit_CMD_Only-3.csv*"

"*Data_Revit_CMD_Only-4.csv*"

For these files, remove the first line from the first file, the first two lines from the second file, the first three lines from the third file, and the first four lines from the fourth file. Then, append the contents of each copied file, in order, to the previously created *Data_Revit_CMD_Only.csv*.

> Script 2: 2_LIST-TO-5-COL.*ps1*

Link to source code: ***https://pastebin.com/Wd8bF5WK***

This script takes *Data_Revit_CMD_Only.csv* and produces "*DATA-COOKED.txt*" in the desired five-column, comma-separated format. Note the use of a "*.txt*" extension which doesn't affect the data format.

> Sample of DATA-COOKED.txt

```
ID_CANCEL_EDITOR,ID_CANCEL_EDITOR,ID_ZOOM_ALL_ALL,ID_VIEWCUBE_SET_HOME,ID_
REVIT_FILE_SAVE_AS
ID_REVIT_FILE_CLOSE,ID_REVIT_MODEL_BROWSER_OPEN,ID_REVIT_FILE_OPEN,ID_REVI
T_FILE_OPEN,ID_REVIT_FILE_OPEN
ID_REVIT_FILE_SAVE_AS,ID_ZOOM_ALL_ALL,ID_PURGE_UNUSED,ID_PURGE_UNUSED,ID_P
URGE_UNUSED
```

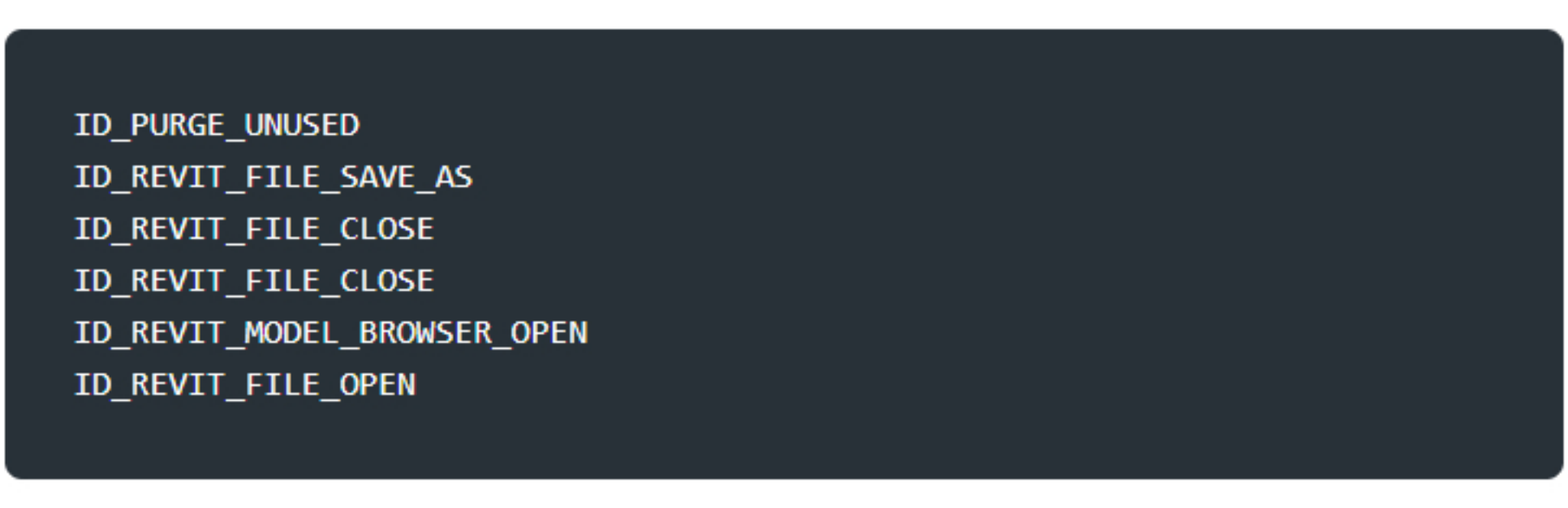
> Script 3: 3_5-COL-VALIDATE.ps1

Link to source code: ***https://pastebin.com/cQnJ1p2s***

Finally, validate the *DATA-COOKED.txt* file with this script which identifies lines without five columns. To test the validator, temporarily add a sixth column to the first line of *DATA-COOKED.txt* by appending "*,bad entry here*", resulting in:

```
ID_CANCEL_EDITOR,ID_CANCEL_EDITOR,ID_ZOOM_ALL_ALL,ID_VIEWCUBE_SET_HOME,ID_
REVIT_FILE_SAVE_AS,bad entry here
```

The validator confirms the issue we created and that the rest of the data is good.



Now, we have Revit command IDs sequenced in a five-column, comma-separated text file. However, typical Revit users may not recognize commands by their IDs, making it necessary to create another file with a list of command IDs and descriptions. When predicting the next command, *AG Feeling Lucky* will display the description instead of the ID. For example, if the prediction is "*ID_OBJECTS_ROOM_TAG*", the user will see "*Room Tag*".

We'll store this data in a two-column, comma-separated text file, with command IDs in the first column and descriptions in the second.

> Example

```
ID_ROOF_PICK_FACES,Roof By Face
ID_CREATE_SOFFIT_TB,Soffit
ID_LOAD_INTO_PROJECTS,Load As Group Into Open Projects
ID_LOAD_INTO_PROJECTS_REBAR_SHAPE,Load Into Project
ID_REPEAT_COMPONENT,Repeat Component
```

> Script 4: 1_GET-P-CMD.py

Link to source code: ***https://pastebin.com/yjZ2WVbj***

How do we achieve this? Using Python inside *PyRevit*! *1_GET-P-CMD.py* is a *PyRevit* extension script that prints all postable commands and their descriptions to a pyRevit console.

> Sample of PyRevit Console Output

```
ID_ROOF_PICK_FACES,Roof By Face
ID_CREATE_SOFFIT_TB,Soffit
ID_LOAD_INTO_PROJECTS,Load As Group Into Open Projects
ID_LOAD_INTO_PROJECTS_REBAR_SHAPE,Load Into Project
ID_REPEAT_COMPONENT,Repeat Component
```

Save this output as "*ID+DESCRIPTION-FORMATTED.txt*"

With the data in the desired format, we proceed with two files to the next step: *DATA-COOKED.txt* and *ID+DESCRIPTION-FORMATTED.txt*

Go to the next part

Go to the previous part

Download AG Feeling Lucky Plugin for Revit & Trainer/Analysis Console Application here:

https://letsbimtogether.com/blog.html