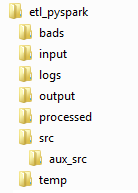
# ETL Process considerations

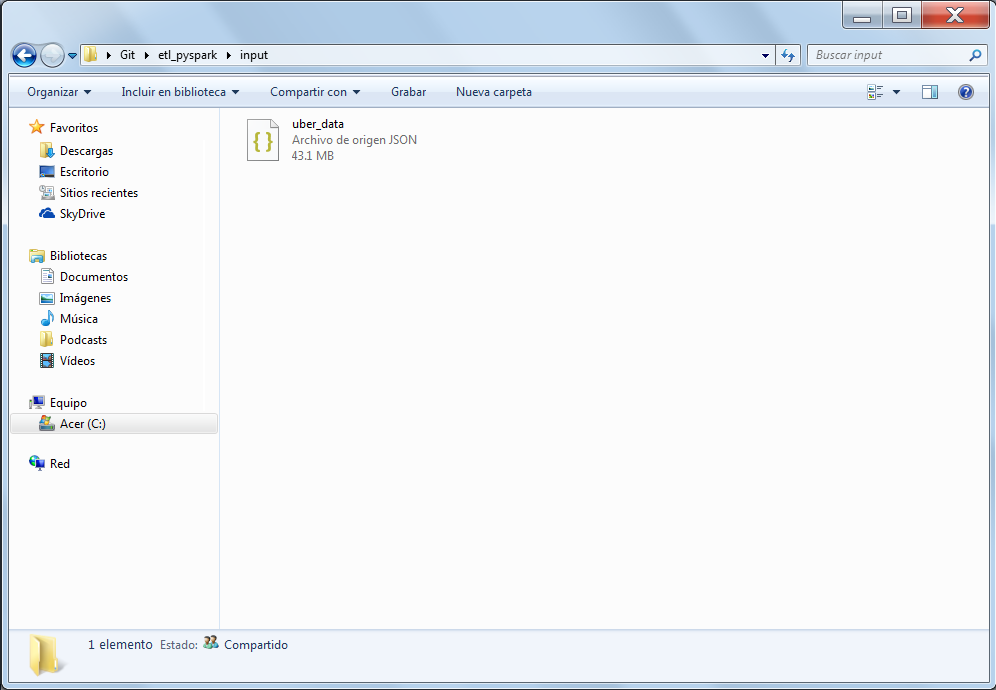
## Directories and files

For the ETL process to work correctly, the following files and directories must be present, this is after having installed all the prerequisites. In the directory where the project will be created (etl\_pyspark), the following directories must exist.



### Input directory

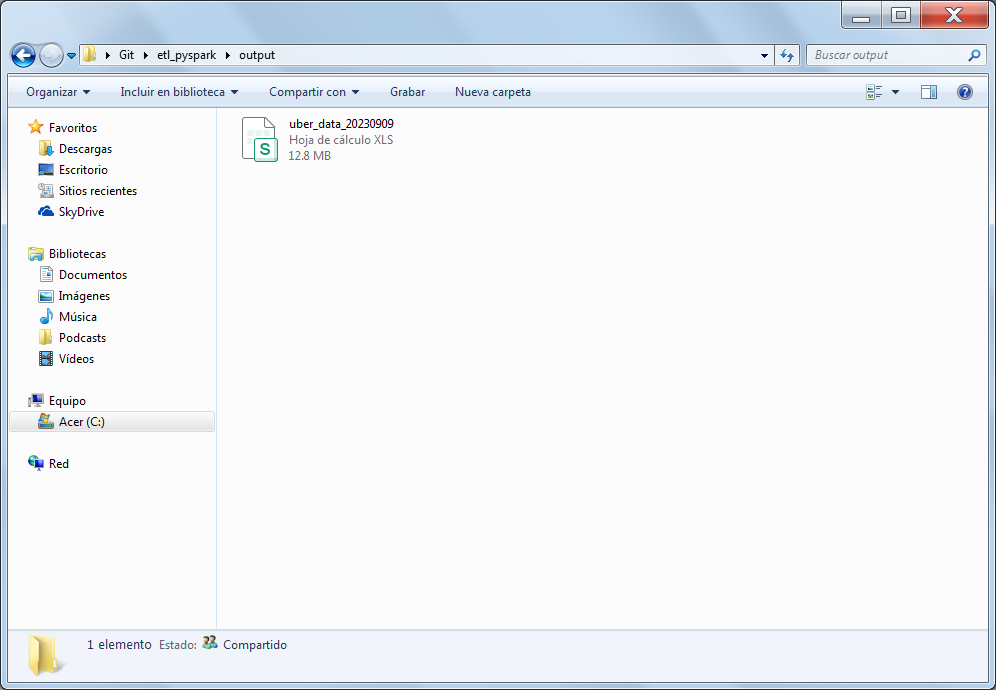
In the input directory is the uber\_data.json file that will be uploaded to the tables in the SQL server. The file used for this project is based on the file uber\_data.csv in the <https://github.com/darshilparmar/uber-etl-pipeline-data-engineering-project/tree/main/data> project.



### Output directory

The output directory is used to save all the files that are generated in each of the extract, transform and load processes. When the extraction process is run, the uber\_data\_yyyymmdd.csv file is generated with the same records as the uber\_data.json file.

Remember that the date in the name of the files created is the day the process was executed.



In the following link you can see the output log of the Extract process that creates the csv file described.



This file is the basis for creating the files in the following transform and load stages. With the exception of files that load to the following tables:

CAT\_TARIFA

CAT\_TIPO\_PAGO

This is because these catalogs are created based on a file found in the following link

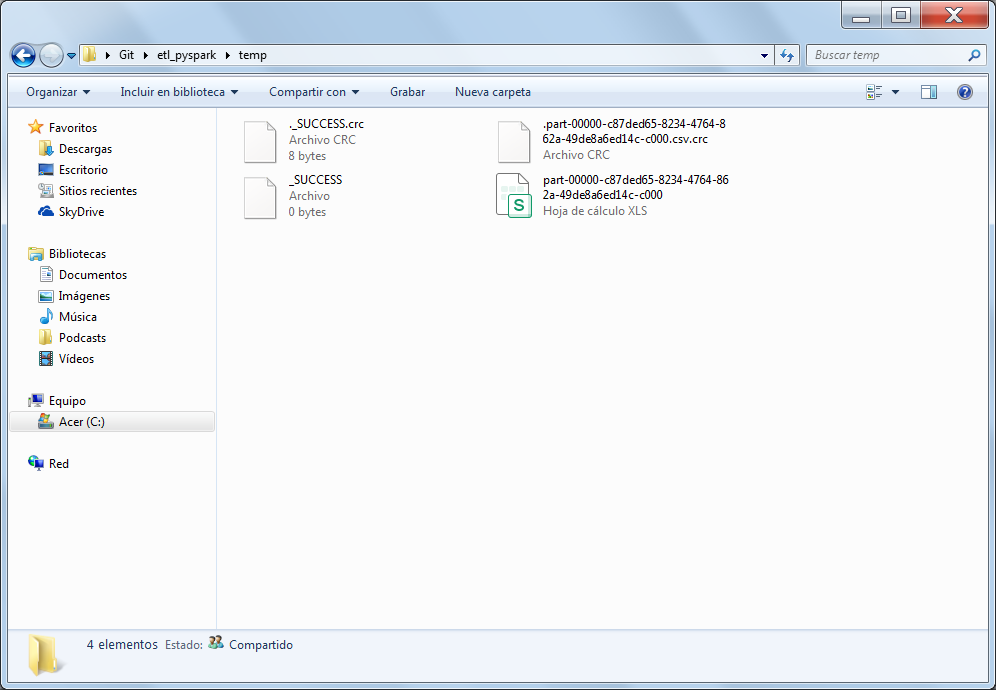
<https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf>

In the case of the following table, there was no value associated with its description, so the values were invented.

CAT\_PROVEEDOR

### Temp directory

The temporary directory is used to create the csv files. This is because in pyspark, when creating a file, only the path where you want to create the file is passed, the process creates four files of which one of them is the file with the information in csv format, as you can see next.



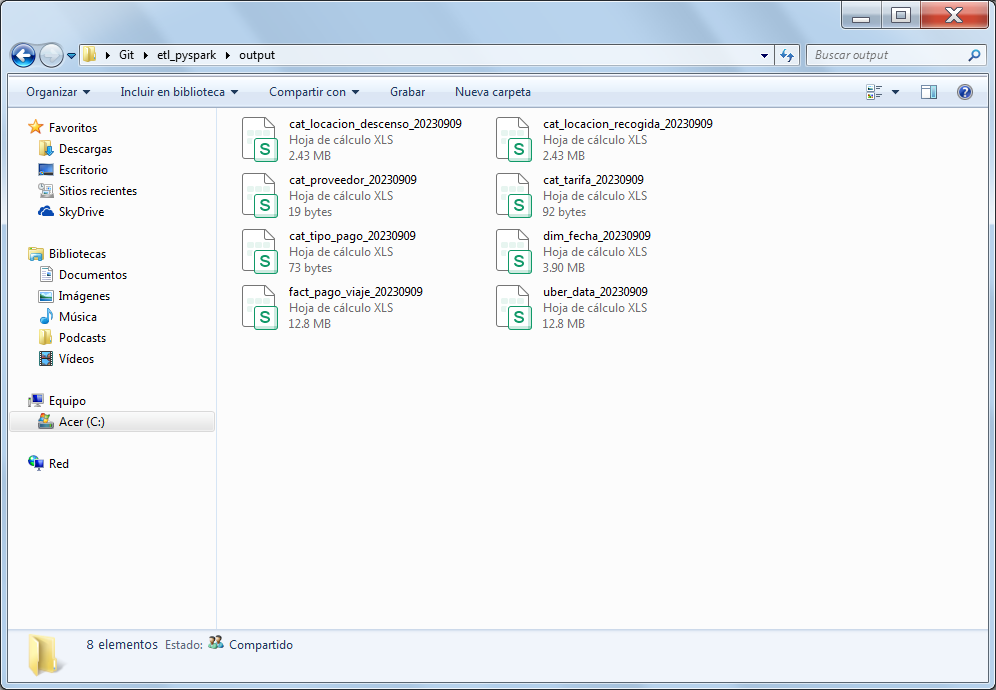
When one of the processes creates a file, the next steps are to change the name and directory of the file with the csv format to the output directory. When one of the processes creates a file, the next step is to change the name and directory of the file with the csv format to the output directory.

When the file has already been renamed and moved to the output directory, the files that are in the temporary directory are deleted, since it must always be empty for the following files that will be created.

This can be seen in the output log of the transform process, since the process generates several files.

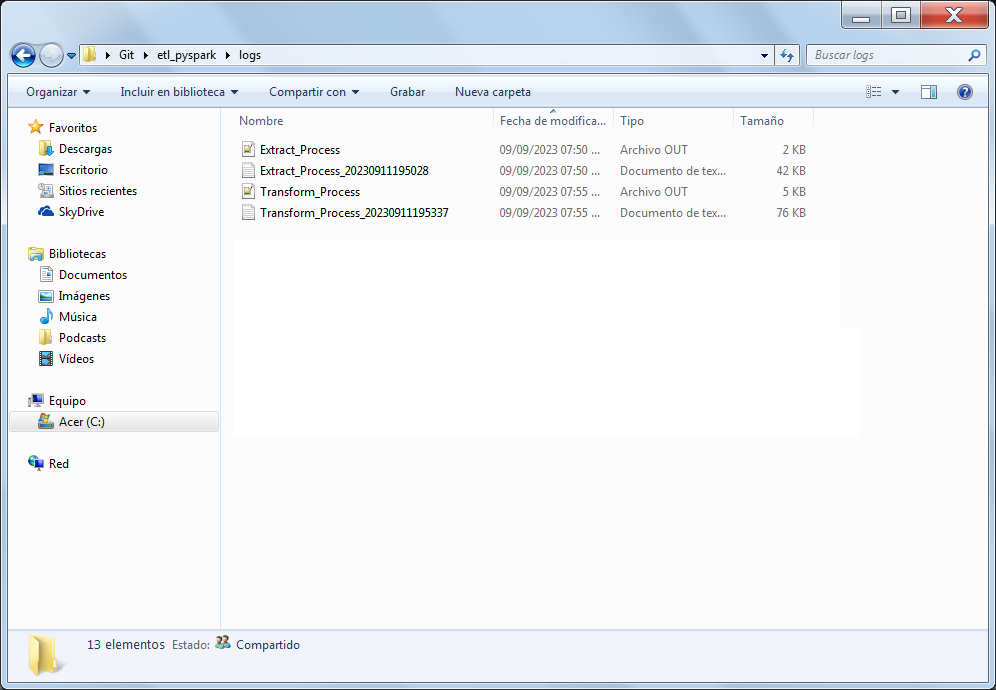


This can be seen in the output log of the transform process, since the process generates several files.



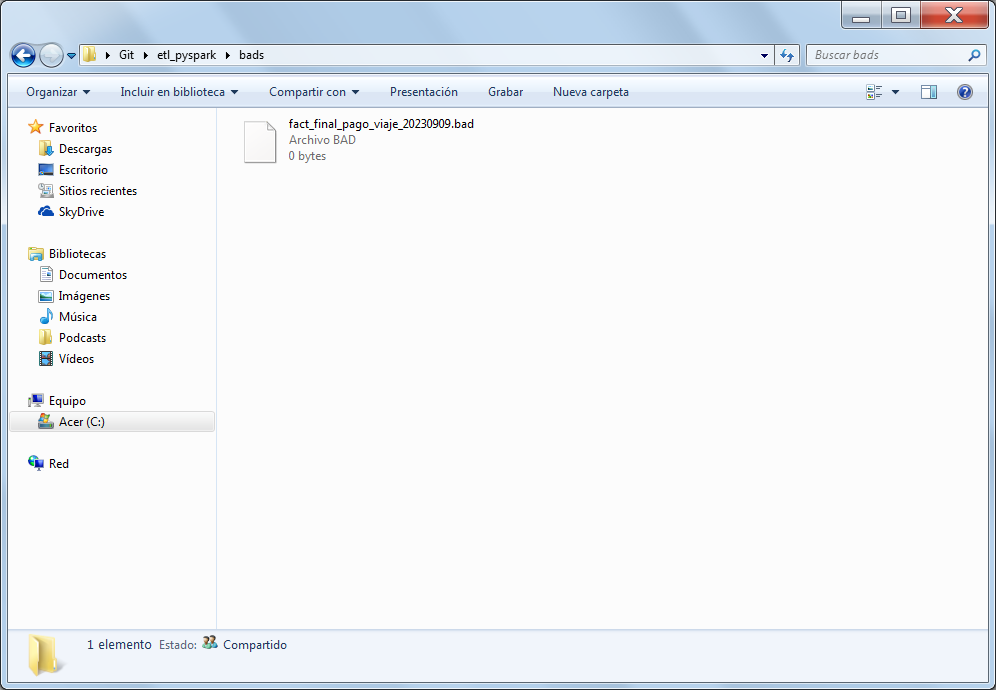
### Logs directory

The logs directory is used to create the log files. Either the files that each process generates automatically or the files with the console output logs.



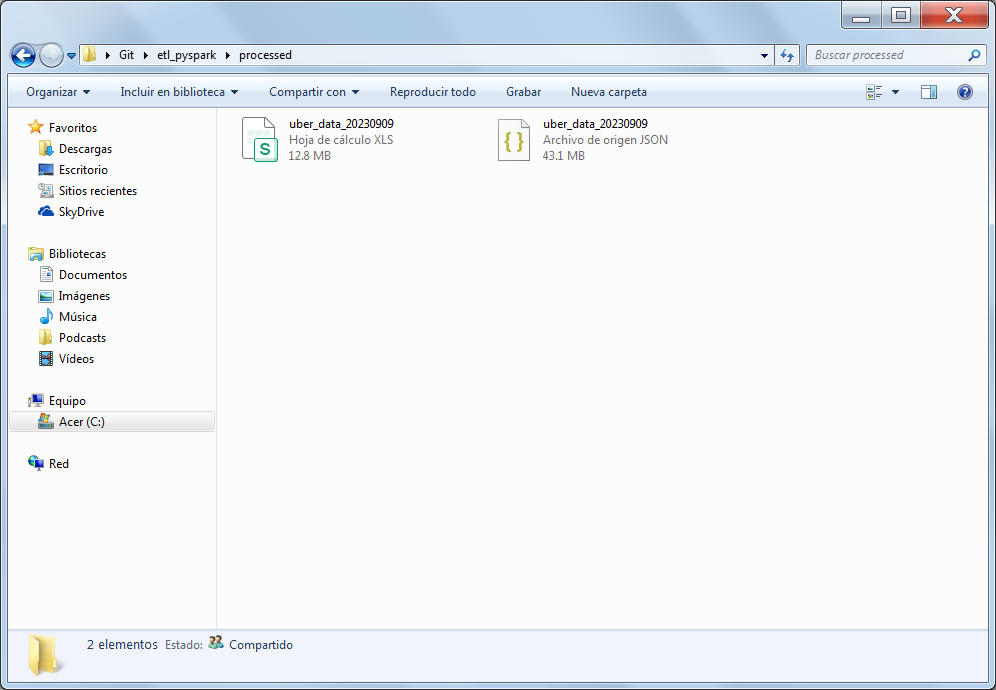
### Bads directory

The bads directory is used to create a bad file. This file is used to validate that the load to the FACT\_PAGO\_VIAJE table doesn’t have the IDs that were generated in the other tables with a null value. If everything is correct, the file should be generated without data.



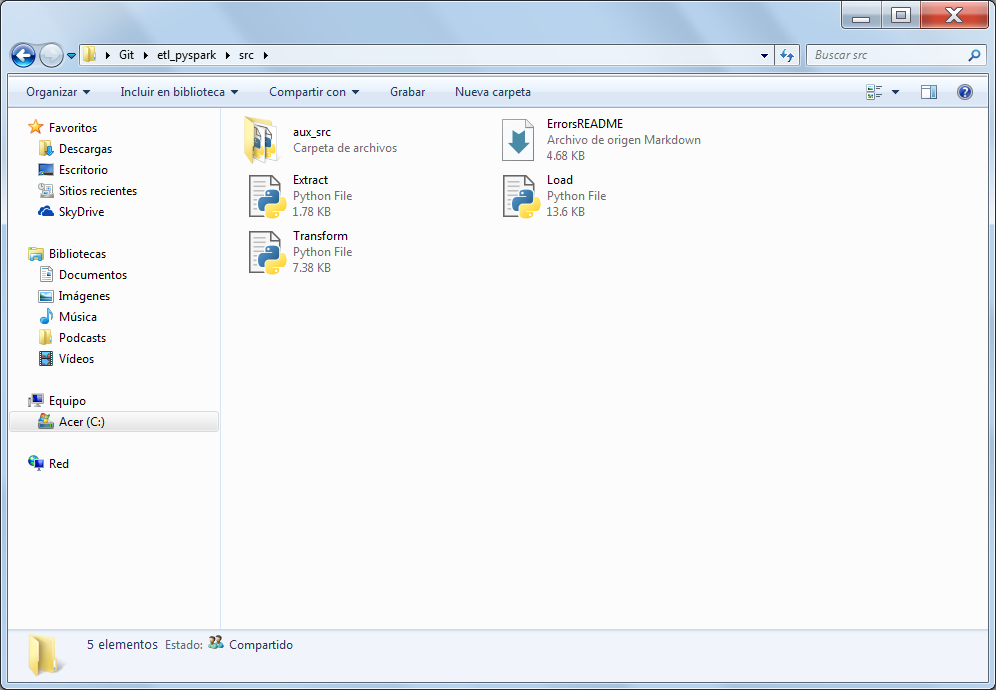
### Processed directory

The processed directory is used to copy the uber\_data.json and uber\_data\_yyyymmdd.csv files. When the tables have already been loaded.



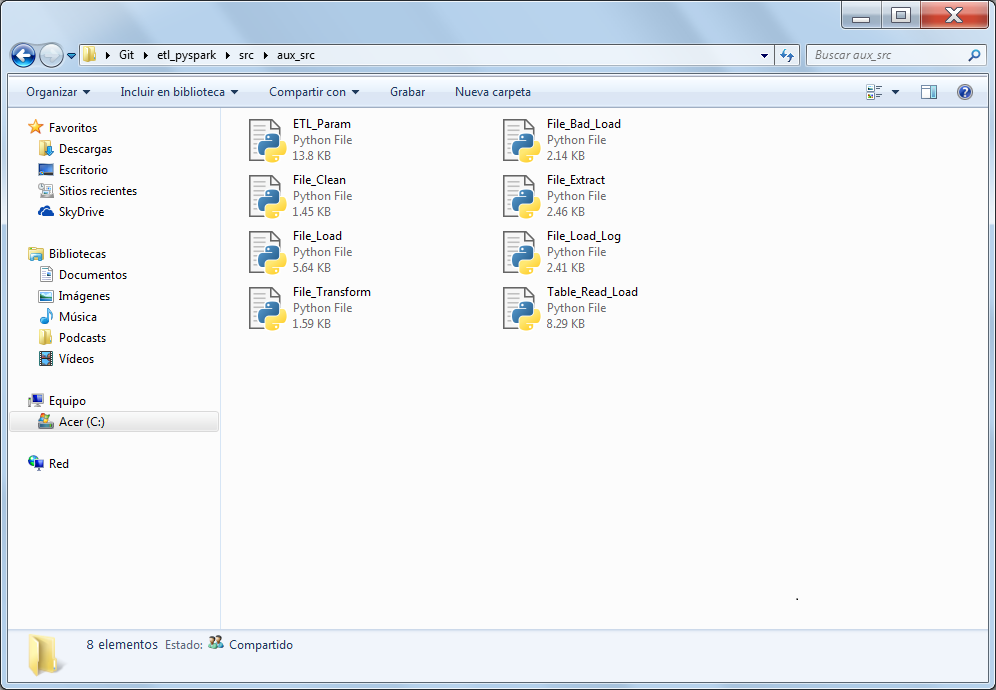
### Src directory

The src directory contains the Python files that run the Extract, Transform, and Load processes. Also, a file with some runtime errors. Also, a file with some runtime errors.



### Aux\_src directory

The aux\_src directory contains the Python files that help with the execution of the Extract, Transform and Load processes.



### Extract.py

The Extract.py file executes the process that creates the uber\_data\_yyyymmdd.csv file in the output path. This file is the basis for generating the other files for loading into the tables.

To achieve this, run the next functions from the File\_Extract.py file:

deleteFile()

createFile()

Finally, gets the next parameter from the ETL\_Param.py file.



### File\_Extract.py

The File\_Extract.py file helps to create the csv file from the uber\_data.json file. This file gets the next parameters from the ETL\_Param.py file.



Creates the csv file in the tmp directory, renames it and moves it to the output directory and deletes the files that are generated in addition to the csv file in the temp directory.

### Transform.py

The Transform.py file executes the process that creates the next files in the output path.



To achieve this, run the next functions from the File\_Transform.py file:

deleteFile()

createFile()

Finally, gets the next parameters from the ETL\_Param.py file.



### File\_Transform.py

The File\_Transform.py file helps to create the csv files from the uber\_data\_yyyymmd d.csv file. This file gets the next parameters from the ETL\_Param.py file.



Creates the csv files in the tmp directory, renames them and moves them to the output directory and deletes the files that are generated in addition to the csv file in the temp directory.

### Load.py

The Load.py file runs the process that loads the following files into the tables in SQL Server and prepares the directories for new execution.



To achieve this, run the next functions from the File\_Load.py file:

loadTable()

readTables()

deleteFile()

To achieve this, run the next function from the File\_Load\_Bad.py file:

createBadFile()

To achieve this, run the next function from the File\_Load\_Log.py file:

loadLogTable()

To achieve this, run the next function from the File\_Clean\_Path.py file:

cleanPaths()

Finally, gets the next parameters from the ETL\_Param.py file and defined queries from the File\_Load\_Table\_Read.py file.



### File\_Load.py

The File\_Load.py file helps to load the csv files into the SQL server tables and creates the csv files that validate the information in the image tables with the final tables, for later loading into the final tables. This file gets the next parameters from the ETL\_Param.py file and .env file.



When a table is loaded, the file generates a log csv file with a record that will be used to load the log table.

Creates the csv files in the tmp directory, renames them and moves them to the output directory and deletes the files that are generated in addition to the csv file in the temp directory.

### File\_Load\_Bad.py

The file File\_Load\_Bad.py generates a file that validates that the IDs of the catalogs and the dimensional table do not have null values, this file is created in the temporary directory and then renamed and moved to the bads directory. This file gets the next parameters from the ETL\_Param.py file.



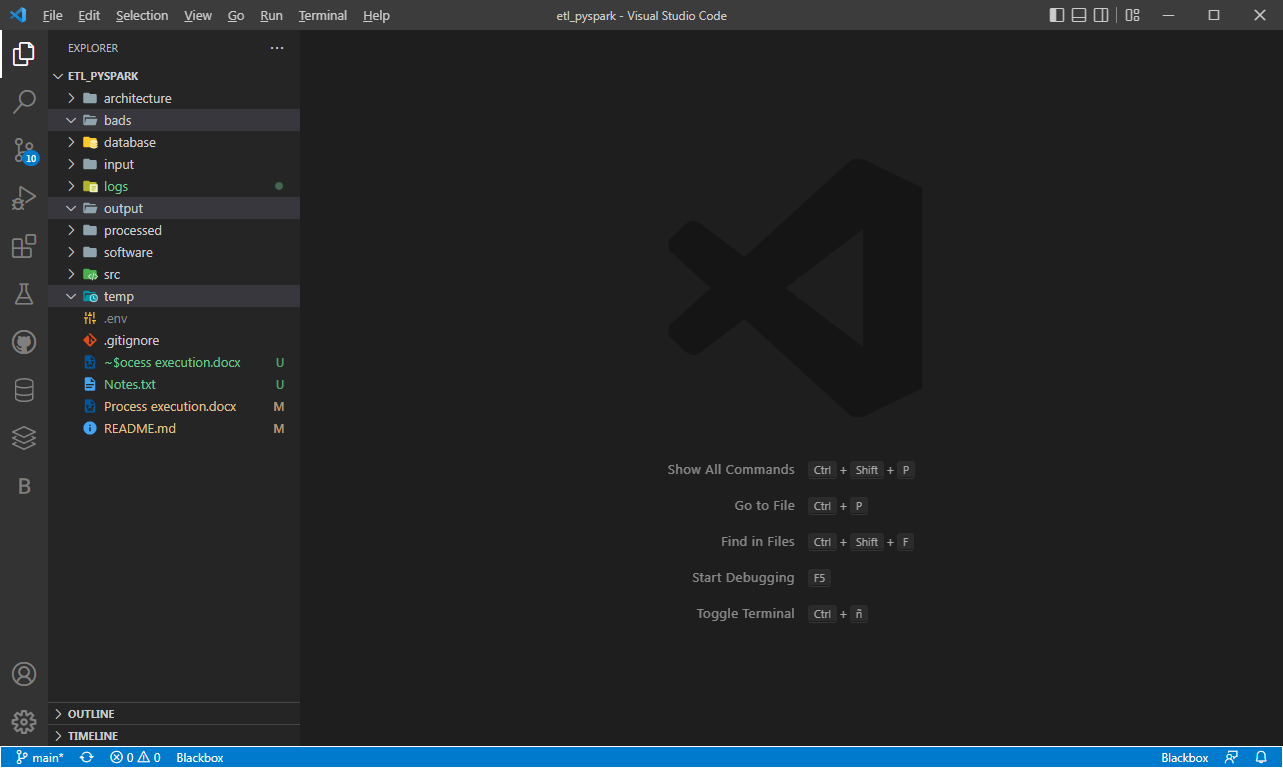
### File\_Load\_Log.py

The File\_Load\_Log.py file loads the log csv file generated in the File\_Load.py file into the log table. This file gets the next parameters from the ETL\_Param.py file and .env file.



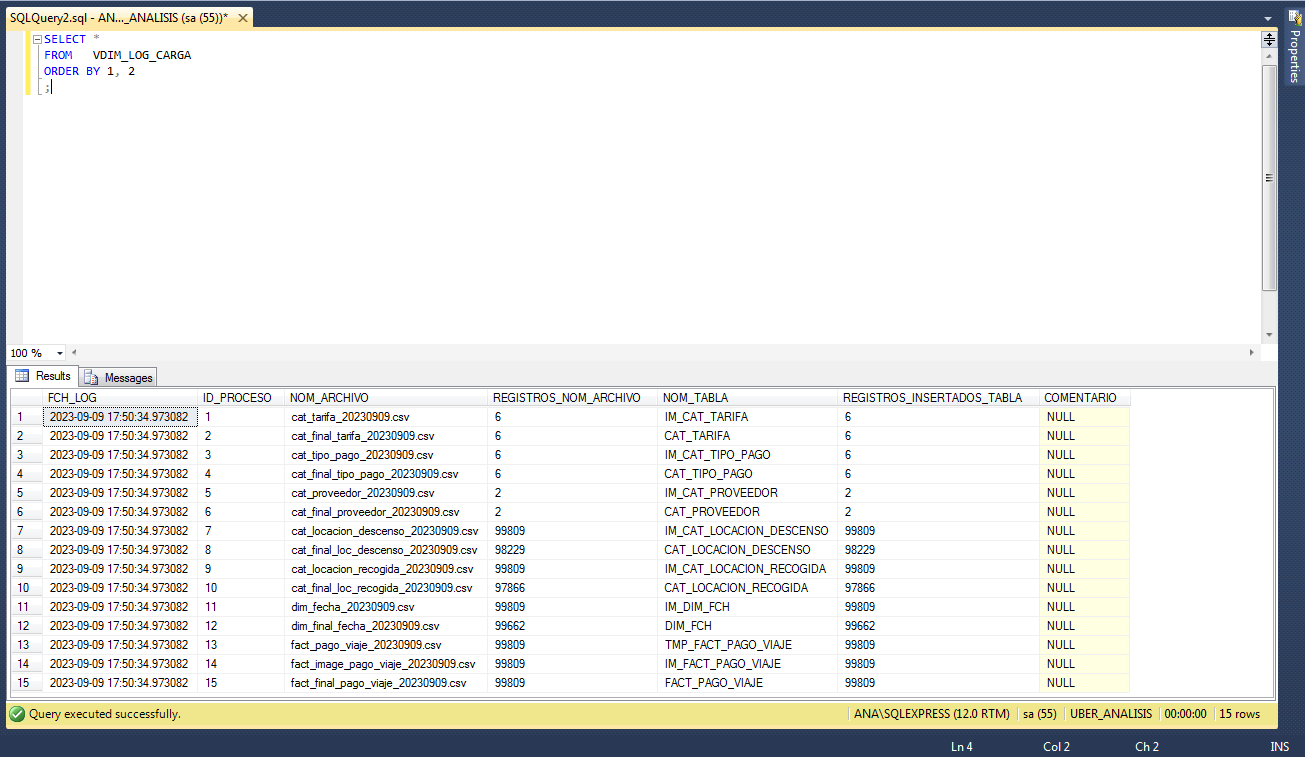
### File\_Clean\_Path.py

The File\_Clean\_Path.py file is the last file to be executed, copies the uber\_data.json and uber\_data\_yyyymmdd.csv files to the processed directory and cleans the bads, temp and output directories.

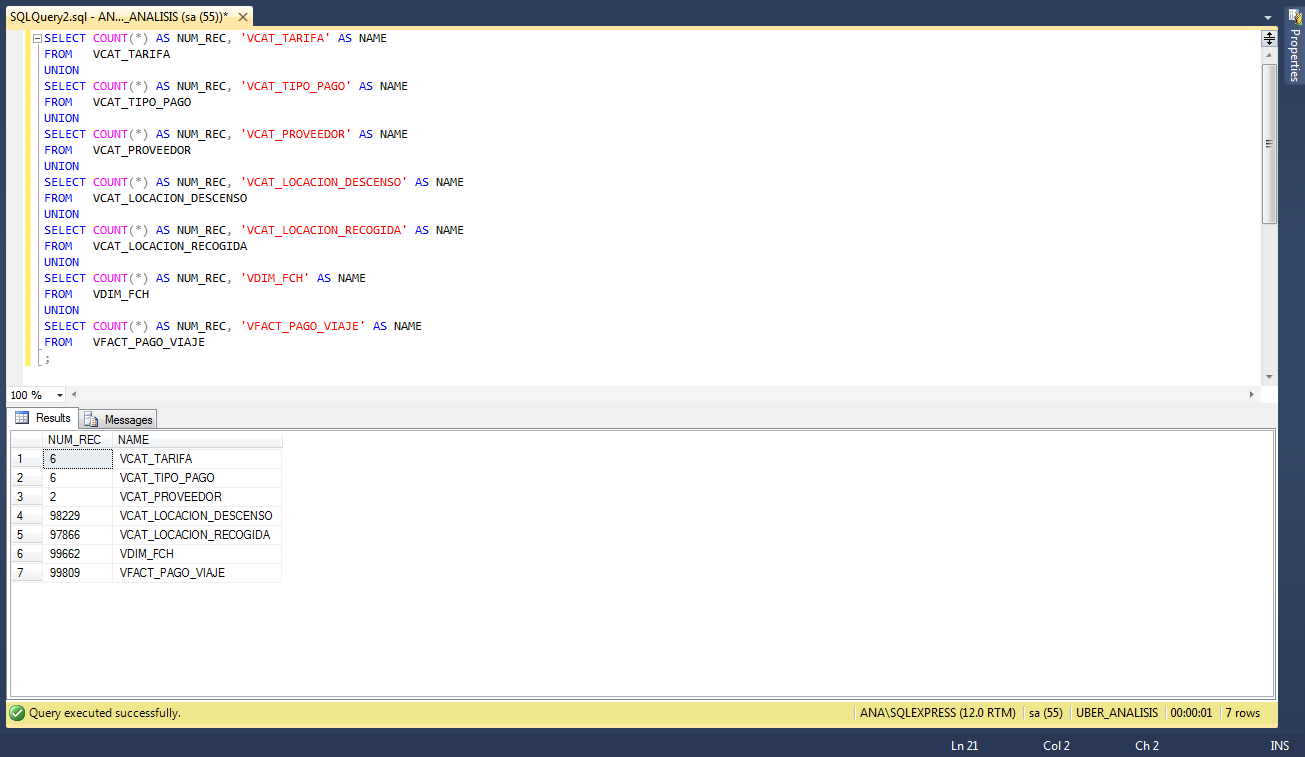


### Database

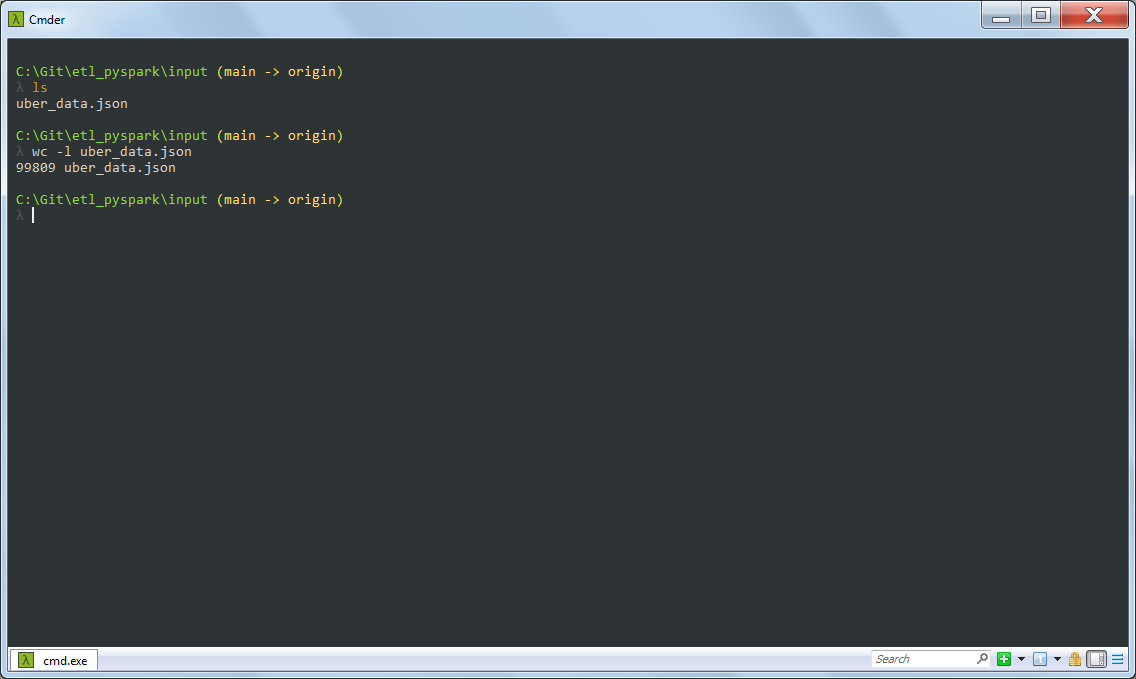
When the process is complete, you can review the information in the database model views. The VDIM\_LOG\_CARGA view contains the records loaded into all tables, The REGISTROS\_NOM\_ARCHIVO and REGISTROS\_INSERTADOS\_TABLA columns must be the same records, with that it is validated that there is no loss of information between the files and tables.



If we count each view, we can see that the numbers returned by the counts are the same as those in the VDIM\_LOG\_LOAD view.



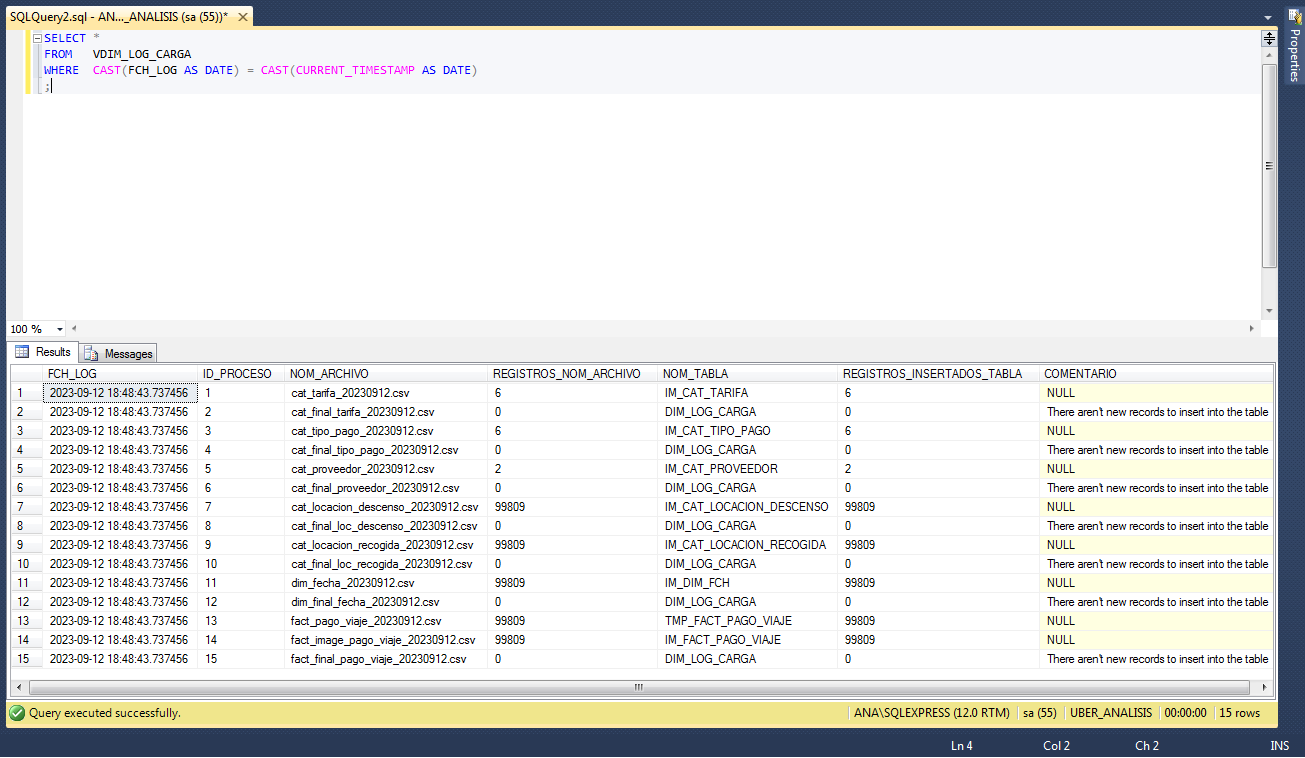
When validating the records in the json file, you can see that they are the same ones (99809 records) that were loaded in the FACT\_TYPE\_VOYAGE table, so there is no loss of information in the ETL process.



This process was done as if they sent loading information every day. Therefore, validations were made so that the tables did not have duplicate values, only the new ones were uploaded.

This is why if the process were executed several times even on the same day, the information loaded into the tables will not have inconsistencies in the data.

Next, you can see how when you run the process again and how the source of the information is still the same file, the information was not loaded again, saving processing time, and in the log table you can see how the records loaded to the final tables were zero.



In the output log of the loading process also indicates that the process does not load information to the final tables, since the files that should load the information were generated without records.

