NYC Taxi – Data exploration and cleaning

Author: Joan Borràs

1. Introduction

This document briefly explains the main issues encountered downloading and treating with the datasets and decisions we can made for further analysis.

The notebook "*data_exploration_cleaning*" shows how it has been implemented all the process.

2. Data sources

We have 3 main data sources:

- Yellow Taxi Trip Records:
    - Description: Data about yellow taxis in NYC, that includes include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances
    - URL: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page
    - Data dictionary: https://www1.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_yellow.pdf
- Taxi Zone Lookup Table:
    - Description: Table that groups different zones in NYC
    - URL: https://s3.amazonaws.com/nyc-tlc/misc/taxi+_zone_lookup.csv
- Taxi Zone Shapefile:
    - Description: Shepefile with the geographic polygons of the zones in NYC
    - URL: https://s3.amazonaws.com/nyc-tlc/misc/taxi_zones.zip

3. Data extraction

3.1. Memory issues

When loading the Yellow Taxi Trip Records we can suffer from memory problems, especially if we want to increase the periods to analyse. So, it has been decided to use Spark to load the Taxi datasets.

3.2. Downloading pipeline

A script helper (extraction.py) has been created that manages the data extraction from the URLs. The method saves the data in local folders, so it avoids always downloading the data from the website.

With the help of a configuration file (config.yml) we can specify the URLs, file names, local folders, etc. For the case of taxi data we can also specify the periods, so we can increase the months and years whenever we want.

NOTE: For the case of taxi data, even with Spark, we are suffering from some strange errors when we include all the months of 2017. We have added a parameter in config.yml (*sample_fraction*) to only work with a fraction of the dataset. Even with this we suffer from the following error, that for the lack of time it hasn't been furtherly investigated:

```
Output exceeds the size limit. Open the full output data in a text editor
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
    [... skipping hidden 1 frame]

c:\Users\joan.borras\sourcecode\NYCTaxi\notebooks\test.ipynb Cell 2' in <cell line: 27>()
     25 psdf_taxi = prep.clean_taxi_data(psdf_taxi, cfg)
---> 27 print(f"SPDF Taxi size: {psdf_taxi.count()}")

File c:\ProgramData\Anaconda3\envs\py39\lib\site-packages\pyspark\sql\dataframe.py:804, in DataFrame.count(self)
    795 """Returns the number of rows in this :class:`DataFrame`.
    796
    797 .. versionadded:: 1.3.0
   (...)
    802 2
    803 """
--> 804 return int(self._jdf.count())

File c:\ProgramData\Anaconda3\envs\py39\lib\site-packages\py4j\java_gateway.py:1321, in JavaMember.__call__(self, *args)
   1320 answer = self.gateway_client.send_command(command)
-> 1321 return_value = get_return_value(
   1322     answer, self.gateway_client, self.target_id, self.name)
   1324 for temp_arg in temp_args:

File c:\ProgramData\Anaconda3\envs\py39\lib\site-packages\pyspark\sql\utils.py:190, in capture_sql_exception.<locals>.deco(*a, **kw)
    189 try:

...
--> 438 self.socket.connect((self.java_address, self.java_port))
    439 self.stream = self.socket.makefile("rb")
    440 self.is_connected = True

ConnectionRefusedError: [WinError 10061] No se puede establecer una conexión ya que el equipo de destino denegó expresamente dicha conexión
```

By the way, working with a fraction of 10% of the dataset for the months 1, 3, 6, 9, 11 and 12 in 2017, worked perfectly. Reducing the fraction, and increasing the months, surprisingly still caused problems.

4. Main statistics

The following figures show the main statistics obtained from the taxi dataset. Previously we have added a new feature ("*duration_in_min*") which measures in minutes the duration of the trip, based on the "*tpep_pickup_datetime*" and "*tpep_dropoff_datetime*" features.

summary	passenger_count	trip_distance	RatecodeID	duration_in_min	payment_type	fare_amount	extra
count	575935	575935	575935	575935	575935	575935	575935
mean	1.623452299304609	2.900139043468437	1.0425846666724543	16.715698472917992	1.3329525033206873	12.934169550383297	0.3286682872199119
stddev	1.264168112625803	3.7249656428836073	0.4705459701182597	57.93918399674485	0.4907269525526679	11.437387074207802	0.4515549029997997
min	0	0.0	1	-50.8	1	-350.0	-4.5
max	9	221.0	99	1439.8	4	567.5	6.0

summary	MTA_tax	Improvement_surcharge	tip_amount	tolls_amount	total_amount	Congestion_Surcharge	Airport_fee
count	575935	575935	575935	575935	575935	0	0
mean	0.49744476373201835	0.2996359832267907	1.8444270620816878	0.31778537508571075	16.22605802739681	null	null
stddev	0.13428793872862688	0.01409659748960884	2.5888118242655973	1.7240316483446023	14.121522373352686	null	null
min	-0.5	-0.3	-6.0	-5.76	-350.3	null	null
max	97.75	1.0	153.0	666.02	687.32	null	null

summary	VendorID	RateCodeID	Store_and_fwd_flag	Payment_type
count	575935	575935	575935	575935
mean	1.5481538715306415	1.0425846666724543	null	1.3329525033206873
...				
min	1	1	N	1
max	2	99	Y	4

5. Check for errors

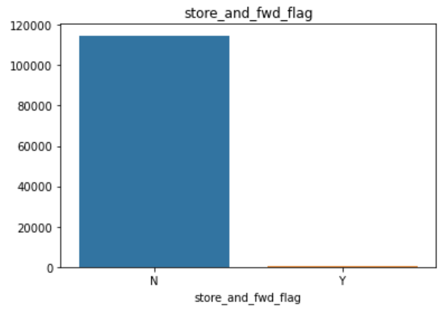
With the statistics observed above, and with some filtering analysis made at the notebebook, we can see several data inconsistency:

- We can see some features with negative values: *fare_amount, duration_in_min, extra, MTA_Tax, Improvement_surcharge, tip_amount, tolls_amount, total_amount.*
- The features *Passenger_count* and *trip_distance* have values equal to 0 but they should be above 0.
- We can see very high trip durations (maximum is at 24h). We could suspect that being exactly 24h the maximum value, it would be caused by some errors at the time stamp variables (for instance they could be specified as 00:00:00). But it has been checked and that this is not the case.
- The features “Congestion_Surcharge” and “Airport_fee” contain only null values.
- *RateCodeID* has 8 rows with values not between the available options (1 and 6). They will have to be removed from the dataset.
- The feature *extra* only accepts values of 0.5$ and 1$. It has been detected in 0.01% of cases that the values are not modulo of 0.5$.
- *MTA_Tax* should be always 0.5$ but it in 0.05% of cases we find different values.
- Adding all expenses should be equal to *total_amount*. But this is not always true in 0.17% of rows.
- *Improvement_surcharge* is different that 0.3$ in 0.07% of observations.
- 0.09% of observations have a pickup datetime greater than the drop off datetime.
- 0.02% of observations have the year different from the requested, and 0.19% have different months.

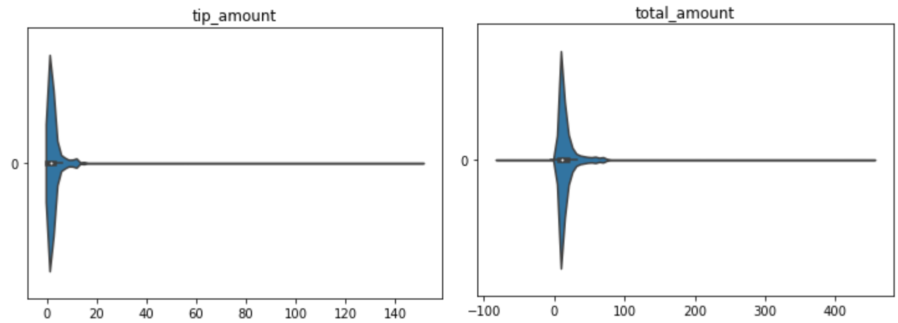
All the observations that do not accomplish the above conditions are removed by the “*clean_taxi_data*” function created at the module “*preprocessing.py*”.

It will have to be removed also those trips that are paid by cash, since they do not contain tip information. This represents the 31.84% of the total observations in the dataset.

We can also check the frequency of the category variables and discover how they are represented. Looking at the following example we see that *store_and_fwd_flag* are mainly equal to *N* (more charts are available at the notebook).



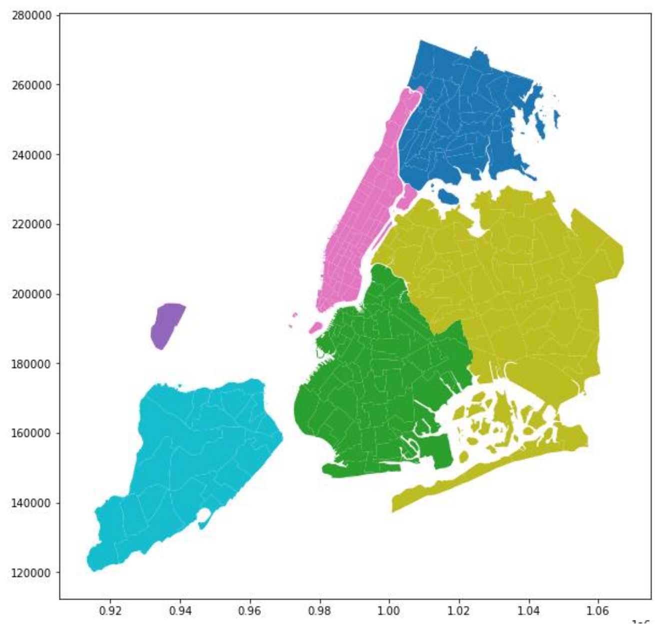
Checking the distribution of the numeric variables we can see a right-skewed distribution with extremely large values in some variables as shown below.



This could be caused by errors of the system, but they could also be feasible values. Before deciding if we remove or not those high values, we check the % of possible outliers applying the Mean and Standard Deviation Method. Seeing the results in the notebook, for the moment we decide to remove only the outliers for *duration_in_min* feature. In this case, it does not seem correct to have trips with 24 hours of duration. In other features like trip_distance we decide to not remove the outliers. The maximum value of *trip_distance* is 221 miles, which in this case, could be accepted.

Checking for ID duplicates, it has been discovered that LocationID in SHP has duplicates. In fact, the IDs of LocationID and OBJECTID are always the same, except for those that have duplicates. For this reason, we can conclude that we should use OBJECTID instead for the SHP dataframe.

Finally, we plot some geo maps, to see if the grouped areas are well identified with their boroughs. As we can see in the figure below, the boroughs seem to be correctly classified.



It appears to be a borough and a service zone with the name “Unknown”. With some domain knowledge of the region, it would be good to identify the reason.

6. Data cleaning

With the decisions made above, it has been created a function named *clean_taxi_data*, that performs 3 main steps:

- Removes all the observations with values not allowed for each feature
- Remove outliers for the feature *duration_in_min*
- Remove trips that are paid by cash

This method can be requested later for data exploration or modelling.