Ordenador con el que se ha hecho el experimento

Características:

- Intel® Core™ i9-12900H CPU @ 2.40GHz × 4
- 32 GiB RAM DDR4
- 1T SSD m.2
- Sistema Operativo: Ubuntu
- Empleamos el archivo mencionado en el manual. (DATOS DEL 2018)

Ya podemos intuir que no siendo SSD no se obtengan buenos tiempos.

NOTA IMPORTANTE:

- AMD (up to 2.4 GHz)
- 16 GB DDR3
- 1000 GB HDD con 600 GB libres
- Sistema Operativo: Windows

Crear un entorno virtual y usarlo en Jupyter

En el caso de crear un entorno virtual que use una versión concreta de Python y el entorno virtual en Jupyter notebook realizaremos los siguientes pasos:

- Crear el entorno virtual con la versión de python que se quiere: virtualenv venv --python=python3.8
- Activa el entorno
- Instala jupyter: pip install notebook
- Activa en el Kernel la opción del entorno: ipython kernel install --user --name=venv
- Ejecuta jupyter: jupyter notebook --allow-root
- Vete a Kernel --> Change Kernel --> venv (nombre entorno)

Librerías Vaex y Dask documentación

- Vaex

https://vaex.io/

https://vaex.io/docs/index.html

- Dask

https://www.dask.org/

https://tutorial.dask.org/

- Dataset taxis de Nueva York:

https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Pandas

In [4]:

```
%%time
import pandas as pd
```

```
CPU times: user 519 ms, sys: 1.21 s, total: 1.73 s
Wall time: 215 ms
```

Tiempos:

- Con 1TB HDD sobre Windows: 1.95 segundos (alguna vez hasta 12 segundos)
- Con 1TB SSD sobre Linux: 571 milisegundos
- Con 1TB SSD m.2 Linux: 519 ms

In [5]:

```
%%time
df_pandas = pd.read_csv("yellow_tripdata.csv")
df_pandas.head()
```

```
CPU times: user 5.99 s, sys: 1.14 s, total: 7.12 s
Wall time: 7.13 s
```

Out[5]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag
0	0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1	N
1	1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1	N
2	2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1	N
3	3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1	N
4	4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1	N

In [ ]:

```
# read_parquet AÑADIDO PARA LA NUEVA VERSIÓN:
```

In [6]:

```
%%time
df_pandas = pd.read_parquet("yellow_tripdata_2018-01.parquet")
df_pandas.head()
```

```
CPU times: user 2.64 s, sys: 1.58 s, total: 4.22 s
Wall time: 539 ms
```

Out[6]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1	N	41
1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1	N	239
2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1	N	262
3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1	N	140
4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1	N	246

In [7]:

```
# Transformar el dataframe para poder trabajar con él.
df_pandas.to_csv("yellow_tripdata_2.csv")
```

Tiempos:

- Con 1TB HDD sobre Windows: 1 minuto 6 segundos (a veces algo más)
- Con 1TB SSD sobre Linux: 56.6 segundos
- Con 250 SSD sobre linux: 21.6 segundos
- Con 1TB SSD m.2 Linux: 2.64 s

In [8]:

```
print('Number of Rows: ',len(df_pandas.index))
print("Number of Columns: " + str(len(df_pandas.axes[1])))
```

```
Number of Rows:  8760687
Number of Columns: 19
```

In [9]:

```
%%time
df_pandas.describe()
```

```
CPU times: user 2.3 s, sys: 99.2 ms, total: 2.4 s
Wall time: 2.4 s
```

Out[9]:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	ext
count	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+0
mean	1.560978e+00	1.606807e+00	2.804022e+00	1.039545e+00	1.644579e+02	1.627270e+02	1.310613e+00	1.224443e+01	3.246882e-0
std	4.962678e-01	1.258420e+00	6.412050e+01	4.450619e-01	6.635990e+01	7.031145e+01	4.817808e-01	1.168321e+01	4.502555e-0
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	-4.500000e+02	-4.469000e+0
25%	1.000000e+00	1.000000e+00	9.100000e-01	1.000000e+00	1.160000e+02	1.130000e+02	1.000000e+00	6.000000e+00	0.000000e+0
50%	2.000000e+00	1.000000e+00	1.550000e+00	1.000000e+00	1.620000e+02	1.620000e+02	1.000000e+00	9.000000e+00	0.000000e+0
75%	2.000000e+00	2.000000e+00	2.840000e+00	1.000000e+00	2.340000e+02	2.340000e+02	2.000000e+00	1.350000e+01	5.000000e-0
max	2.000000e+00	9.000000e+00	1.894838e+05	9.900000e+01	2.650000e+02	2.650000e+02	4.000000e+00	8.016000e+03	6.000000e+0

Tiempos:

- Con 1TB HDD sobre Windows: 11.7 segundos
- Con 1TB SSD sobre Linux: 7.76 segundos
- Con 250 SSD sobre linux: 4.81 segundos
- Con 1TB SSD m.2 Linux: 2.3 s

In [10]:

```
%%time
df_pandas.fare_amount.value_counts()
```

```
CPU times: user 44 ms, sys: 154 µs, total: 44.1 ms
Wall time: 42.7 ms
```

Out[10]:

```
6.00       473270
5.50       465207
6.50       461959
7.00       446414
5.00       433292
            ...
30.60           1
2409.00         1
168.88          1
201.50          1
33.96           1
Name: fare_amount, Length: 1714, dtype: int64
```

Tiempos:

- Con 1TB HDD sobre Windows: 555 milisegundos
- Con 1TB SSD sobre Linux: 313 milisegundos
- Con 250 SSD sobre linux: 213 milisegundos
- Con 1TB SSD m.2 Linux: 44 ms

In [11]:

```
%%time
# casi 9 millones de filas en este caso
len(df_pandas), df_pandas.shape
```

```
CPU times: user 18 µs, sys: 0 ns, total: 18 µs
Wall time: 20 µs
```

Out[11]: (8760687, (8760687, 19))

Tiempos:

- Con 1TB HDD sobre Windows: 0 nanosegundos
- Con 1TB SSD sobre Linux: 46.7 microsegundos
- Con 250 SSD sobre linux: 21 microsegundos
- Con 1TB SSD m.2 Linux: 18 microsegundos

In [12]:

```
%%time
df_pandas.tail()
```

```
CPU times: user 753 µs, sys: 74 µs, total: 827 µs
Wall time: 753 µs
```

Out[12]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag
8760682	2	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.80	1	N
8760683	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.60	1	N
8760684	2	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1	N
8760685	2	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.00	1	N
8760686	2	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.00	1	N

Tiempos:

- Con 1TB HDD sobre Windows: 0 nanosegundos
- Con 1TB SSD sobre Linux: 215 microsegundos
- Con 250 SSD sobre linux: 327 microsegundos
- Con 1TB SSD m.2 Linux: 753 microsegundos

DASK

In [14]:

```
# pip install dask
```

https://docs.dask.org/en/stable/install.html

In [15]:

```
%%time
import dask.dataframe as dd
```

```
CPU times: user 134 ms, sys: 3.96 ms, total: 138 ms
Wall time: 137 ms
```

Tiempos:

- Con 1TB HDD sobre Windows: 4.13 segundos (promedio 6 segundos, a veces 8 segundos)
- Con 1TB SSD sobre Linux: 846 milisegundos
- Con 250 SSD sobre linux: 792 milisegundos
- Con 1TB SSD m.2 Linux: 134 milisegundos

In [16]:

```
%%time
df_dask = dd.read_csv("yellow_tripdata.csv",
                      assume_missing=True)
df_dask.head()
```

```
CPU times: user 560 ms, sys: 116 ms, total: 677 ms
Wall time: 670 ms
```

Out[16]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag
0	0.0	1.0	2018-01-01 00:21:05	2018-01-01 00:24:23	1.0	0.5	1.0	N
1	1.0	1.0	2018-01-01 00:44:55	2018-01-01 01:03:05	1.0	2.7	1.0	N
2	2.0	1.0	2018-01-01 00:08:26	2018-01-01 00:14:21	2.0	0.8	1.0	N
3	3.0	1.0	2018-01-01 00:20:22	2018-01-01 00:52:51	1.0	10.2	1.0	N
4	4.0	1.0	2018-01-01 00:09:18	2018-01-01 00:27:06	2.0	2.5	1.0	N

Tiempos:

- Con 1TB HDD sobre Windows: 5.9 segundos
- Con 1TB SSD sobre Linux: 1.94 segundos
- Con 250 SSD sobre linux: 1.88 segundos
- Con 1TB SSD m.2 Linux: 550 milisegundos

In [ ]:

```
# Leer el archivo parquet:
```

In [17]:

```
%%time
df_dask = dd.read_parquet("yellow_tripdata_2018-01.parquet",
                          assume_missing=True)
df_dask.head()
```

```
CPU times: user 1.47 s, sys: 1.88 s, total: 3.34 s
Wall time: 3.4 s
```

Out[17]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1	N	41
1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1	N	239
2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1	N	262
3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1	N	140
4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1	N	246

In [18]:

```
%%time
df_dask.describe()
```

```
CPU times: user 56.7 ms, sys: 4.49 ms, total: 61.2 ms
Wall time: 59.9 ms
```

Out[18]:

Dask DataFrame Structure:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	extra
npartitions=1									
	float64	float64	float64	float64	float64	float64	float64	float64	float64
	...	...	...	...	...	...	...	...	...

Dask Name: describe-numeric, 68 graph layers

Tiempos:

- Con 1TB HDD sobre Windows: 465 milisegundos
- Con 1TB SSD sobre Linux: 298 milisegundos
- Con 250 SSD sobre linux: 220 milisegundos
- Con 1TB SSD m.2 Linux: 56.7 milisegundos

In [19]:

```
%%time
df_dask.describe().compute()
```

```
CPU times: user 9.43 s, sys: 8.58 s, total: 18 s
Wall time: 6.62 s
```

Out[19]:

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	ext
count	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+06	8.760687e+0
mean	1.560978e+00	1.606807e+00	2.804022e+00	1.039545e+00	1.644579e+02	1.627270e+02	1.310613e+00	1.224443e+01	3.246882e-0
std	4.962678e-01	1.258420e+00	6.412050e+01	4.450619e-01	6.635990e+01	7.031145e+01	4.817808e-01	1.168321e+01	4.502555e-0
min	1.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	-4.500000e+02	-4.469000e+0
25%	1.000000e+00	1.000000e+00	9.100000e-01	1.000000e+00	1.160000e+02	1.130000e+02	1.000000e+00	6.000000e+00	0.000000e+0
50%	2.000000e+00	1.000000e+00	1.550000e+00	1.000000e+00	1.620000e+02	1.620000e+02	1.000000e+00	9.000000e+00	0.000000e+0
75%	2.000000e+00	2.000000e+00	2.840000e+00	1.000000e+00	2.340000e+02	2.340000e+02	2.000000e+00	1.350000e+01	5.000000e-0
max	2.000000e+00	9.000000e+00	1.894838e+05	9.900000e+01	2.650000e+02	2.650000e+02	4.000000e+00	8.016000e+03	6.000000e+0

Tiempos:

- Con 1TB HDD sobre Windows: 50 segundos (57.5 segundos otra de las veces)
- Con 1TB SSD sobre Linux: 36 segundos
- Con 250 SSD sobre linux: 40.6 segundos
- Con 1TB SSD m.2 Linux: 9.43 segundos

In [20]:

```
%%time
df_dask.fare_amount.value_counts()
```

```
CPU times: user 2.08 ms, sys: 0 ns, total: 2.08 ms
Wall time: 1.72 ms
```

Out[20]:

```
Dask Series Structure:
npartitions=1
    int64
      ...
Name: fare_amount, dtype: int64
Dask Name: value-counts-agg, 4 graph layers
```

Tiempos:

- Con 1TB HDD sobre Windows: 5 milisegundos
- Con 1TB SSD sobre Linux: 3.25 milisegundos
- Con 250 SSD sobre linux: 1.65 milisegundos
- Con 1TB SSD m.2 Linux: 2.08 milisegundos

In [21]:

```
%%time
df_dask.fare_amount.value_counts().compute()
```

```
CPU times: user 149 ms, sys: 47.9 ms, total: 197 ms
Wall time: 196 ms
```

Out[21]:

```
6.00       473270
5.50       465207
6.50       461959
7.00       446414
5.00       433292
            ...
60.06           1
60.30           1
60.53           1
60.55           1
8016.00         1
Name: fare_amount, Length: 1714, dtype: int64
```

Tiempos:

- Con 1TB HDD sobre Windows: 13.1 segundos
- Con 1TB SSD sobre Linux: 5.45 segundos
- Con 250 SSD sobre linux: 12.5 segundos
- Con 1TB SSD m.2 Linux: 149 milisegundos

VAEX desde un CSV (1ª forma)

In [22]:

```
%%time
# https://pypi.org/project/vaex/
#pip install vaex
import vaex
```

```
CPU times: user 365 ms, sys: 42.5 ms, total: 408 ms
Wall time: 493 ms
```

Tiempos:

- Con 1TB HDD sobre Windows: 7.64 segundos (en otro intento fueron 14.7 segundos)
- Con 1TB SSD sobre Linux: 5.14 segundos
- Con 250 SSD sobre linux: 1.85 segundos
- Con 1TB SSD m.2 Linux: 365 milisegundos

In [23]:

```
%%time
# Necesito añadir convert=True para que me convierta .csv en .HDF5
# Default chunk_size for converting is 5 million rows
# which corresponds to around 1Gb memory on an example of NYC Taxi dataset.
df_vaex = vaex.from_csv("yellow_tripdata.csv",
                        convert=True, chunk_size = 5_000_000)

df_vaex
```

```
---------------------------------------------------------------------------
BufferError                               Traceback (most recent call last)
File ~/.local/lib/python3.8/site-packages/vaex/convert.py:95, in _from_csv_convert_and_read(filename_or_buffer, path_output, chunk_size, fs_options, fs, copy_index, progress, **kwargs)
     94 try:
---> 95     df.close()
     96     os.remove(df_path)

File ~/.local/lib/python3.8/site-packages/vaex/dataframe.py:2203, in DataFrame.close(self)
   2202 """Close any possible open file handles or other resources, the DataFrame will not be in a usable state afterwards."""
-> 2203 self.dataset.close()

File ~/.local/lib/python3.8/site-packages/vaex/hdf5/dataset.py:411, in Hdf5MemoryMapped.close(self)
    410 def close(self):
--> 411     super().close()
    412     self.h5file.close()

File ~/.local/lib/python3.8/site-packages/vaex/dataset_mmap.py:90, in DatasetMemoryMapped.close(self)
     89 for name, memmap in self.mapping_map.items():
---> 90     memmap.close()
     91 for name, file in self.file_map.items():

BufferError: cannot close exported pointers exist

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
File <timed exec>:4

File ~/.local/lib/python3.8/site-packages/vaex/__init__.py:595, in from_csv(filename_or_buffer, copy_index, chunk_size, convert, fs_options, progress, fs, **kwargs)
    593 import vaex.convert
    594 path_output = convert if isinstance(convert, str) else vaex.convert._convert_name(filename_or_buffer)
--> 595 vaex.convert.convert_csv(
    596     path_input=filename_or_buffer, fs_options_input=fs_options, fs_input=fs,
    597     path_output=path_output, fs_options_output=fs_options, fs_output=fs,
    598     chunk_size=chunk_size,
    599     copy_index=copy_index,
    600     progress=progress,
    601     **kwargs
    602 )
    603 return open(path_output, fs_options=fs_options, fs=fs)

File ~/.local/lib/python3.8/site-packages/vaex/convert.py:52, in convert_csv(path_input, fs_options_input, fs_input, path_output, fs_options_output, fs_output, progress, *args, **kwargs)
     50     kwargs['chunk_size'] = 5_000_000
     51     _from_csv_convert_and_read(path_input, path_output=path_output, fs_options=fs_options_input, fs=fs_
---> 52 cached_output(*args, **kwargs)

File ~/.local/lib/python3.8/site-packages/vaex/cache.py:427, in output_file.<locals>.wrapper1.<locals>.wrapper2(*args, **kwargs)
    425 if not vaex.file.exists(path_output, fs_options=fs_options_output_, fs=fs_output):
    426     log_info("file does not exist yet, running conversion %s → %s", path_output_meta, path_input, pa
--> 427     callable(*args, **kwargs)
    428     write_fingerprint()
    429     return value

File ~/.local/lib/python3.8/site-packages/vaex/convert.py:61, in convert_csv.<locals>.cached_output(*args, **kwargs)
     58 if 'chunk_size' not in kwargs:
     59     # make it memory efficient by default
     60     kwargs['chunk_size'] = 5_000_000
---> 61 _from_csv_convert_and_read(path_input, path_output=path_output, fs_options=fs_options_input, fs=fs_input, progress=progress, *args, **kwargs)

File ~/.local/lib/python3.8/site-packages/vaex/convert.py:98, in _from_csv_convert_and_read(filename_or_buffer, path_output, chunk_size, fs_options, fs, copy_index, progress, **kwargs)
     96     os.remove(df_path)
     97 except Exception as e:
---> 98     log.error("Could not close or delete intermediate file %s used to convert %s to single file: %s", (df_path, csv_path, path_output))

File /usr/lib/python3.8/logging/__init__.py:1475, in Logger.error(self, msg, *args, **kwargs)
   1466 """
   1467 Log 'msg % args' with severity 'ERROR'.
   1468
   (...)
   1472 logger.error("Houston, we have a %s", "major problem", exc_info=1)
   1473 """
   1474 if self.isEnabledFor(ERROR):
-> 1475     self._log(ERROR, msg, args, **kwargs)

File /usr/lib/python3.8/logging/__init__.py:1589, in Logger._log(self, level, msg, args, exc_info, extra, stack_info, stacklevel)
   1586     exc_info = sys.exc_info()
   1587 record = self.makeRecord(self.name, level, fn, lno, msg, args,
   1588                          exc_info, func, extra, sinfo)
-> 1589 self.handle(record)

File /usr/lib/python3.8/logging/__init__.py:1599, in Logger.handle(self, record)
   1592 """
   1593 Call the handlers for the specified record.
   1594
   1595 This method is used for unpickled records received from a socket, as
   1596 well as those created locally. Logger-level filtering is applied.
   1597 """
   1598 if (not self.disabled) and self.filter(record):
-> 1599     self.callHandlers(record)

File /usr/lib/python3.8/logging/__init__.py:1661, in Logger.callHandlers(self, record)
   1659 found = found + 1
   1660 if record.levelno >= hdlr.level:
-> 1661     hdlr.handle(record)
   1662 if not c.propagate:
   1663     c = None    #break out

File /usr/lib/python3.8/logging/__init__.py:954, in Handler.handle(self, record)
    952 self.acquire()
    953 try:
--> 954     self.emit(record)
    955 finally:
    956     self.release()

File ~/.local/lib/python3.8/site-packages/rich/logging.py:128, in RichHandler.emit(self, record)
    126 def emit(self, record: LogRecord) -> None:
    127     """Invoked by logging."""
--> 128     message = self.format(record)
    129     traceback = None
    130     if (
    131         self.rich_tracebacks
    132         and record.exc_info
    133         and record.exc_info != (None, None, None)
    134     ):

File /usr/lib/python3.8/logging/__init__.py:929, in Handler.format(self, record)
    927 else:
    928     fmt = _defaultFormatter
--> 929 return fmt.format(record)

File /usr/lib/python3.8/logging/__init__.py:668, in Formatter.format(self, record)
    655 def format(self, record):
    656     """
    657     Format the specified record as text.
    658
   (...)
    666     it is formatted using formatException() and appended to the message.
    667     """
--> 668     record.message = record.getMessage()
    669     if self.usesTime():
    670         record.asctime = self.formatTime(record, self.datefmt)

File /usr/lib/python3.8/logging/__init__.py:373, in LogRecord.getMessage(self)
    371 msg = str(self.msg)
    372 if self.args:
--> 373     msg = msg % self.args
    374 return msg

TypeError: not enough arguments for format string
```

Tiempos:

- Con 1TB HDD sobre Windows: 3 minutos 24 segundos (alguna vez mucho menos)
- Con 1TB SSD sobre Linux: 1 minuto 10 segundos
- Con 250 SSD sobre linux: 1 minuto 8 segundos

VAEX desde HDF5

In [24]:

```
%%time
# https://pypi.org/project/vaex/
# pip install vaex
import vaex
```

```
CPU times: user 13 µs, sys: 0 ns, total: 13 µs
Wall time: 21 µs
```

In [25]:

```
%%time
vaex_hdf5 = vaex.open("yellow_tripdata.csv.hdf5")
vaex_hdf5
```

```
CPU times: user 9.53 ms, sys: 0 ns, total: 9.53 ms
Wall time: 9.33 ms
```

Out[25]:

#	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
0	0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1
1	1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1
2	2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1
3	3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1
4	4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1
...	...	...	...	...	...	...	...
8,760,682	8760682	1	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.8	1
8,760,683	8760683	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.6	1
8,760,684	8760684	2	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1
8,760,685	8760685	2	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.0	1
8,760,686	8760686	2	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.0	1

Tiempos:

- Con 1TB HDD sobre Windows: 5.11 segundos (alguna ocasión fueron 57 milisegundos)
- Con 1TB SSD sobre Linux: 40.1 milisegundos
- Con 250 SSD sobre linux: 26.5 milisegundos
- Con 1TB SSD m.2 Linux: 9.53 milisegundos

In [26]:

```
%%time
vaex_hdf5.describe()
```

```
CPU times: user 8.1 s, sys: 65.8 ms, total: 8.17 s
Wall time: 916 ms
```

Out[26]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
data_type	int64	int64	string	string	int64	float64
count	8760687	8760687	8760687	8760687	8760687	8760687
NA	0	0	0	0	0	0
mean	4380343.0	1.560978037452999	--	--	1.6068074341658365	2.8040221012347804
std	2528992.49887	0.496268	--	--	1.25842	64.120494
min	0	1	--	--	0	0.0
max	8760686	2	--	--	9	189483.84

Tiempos:

- Con 1TB HDD sobre Windows: 9.56 segundos (a veces bastante más)
- Con 1TB SSD sobre Linux: 4.56 segundos
- Con 250 SSD sobre linux: 4.29 segundos
- Con 1TB SSD m.2 Linux: 8.1 segundos

In [27]:

```
%%time
vaex_hdf5.fare_amount.value_counts()
```

```
CPU times: user 373 ms, sys: 904 µs, total: 374 ms
Wall time: 40.7 ms
```

Out[27]:

```
6.00      473270
5.50      465207
6.50      461959
7.00      446414
5.00      433292
           ...
133.56         1
0.40           1
93.25          1
73.75          1
79.60          1
Length: 1714, dtype: int64
```

Tiempos:

- Con 1TB HDD sobre Windows: 1.62 segundos (No ejecuta correctamente alguna vez)
- Con 1TB SSD sobre Linux: 114 milisegundos
- Con 250 SSD sobre linux: 336 milisegundos
- Con 1TB SSD m.2 Linux: 373 milisegundos

VAEX (2ª forma)

In [29]:

```
%%time
# https://pypi.org/project/vaex/
# pip install vaex
import vaex
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 7.63 µs
```

Tiempos:

- Con 1TB HDD sobre Windows: 7.28 segundos
- Con 1TB SSD sobre Linux: 19.6 microsegundos
- Con 250 SSD sobre linux: 13.1 microsegundos

In [30]:

```
%%time
df_vaex_3 = vaex.from_csv("yellow_tripdata.csv")
df_vaex_3
```

```
CPU times: user 7.92 s, sys: 3.47 s, total: 11.4 s
Wall time: 10.8 s
```

Out[30]:

#	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
0	0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1
1	1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1
2	2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1
3	3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1
4	4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1
...	...	...	...	...	...	...	...
8,760,682	8760682	1	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.8	1
8,760,683	8760683	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.6	1
8,760,684	8760684	2	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1
8,760,685	8760685	2	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.0	1
8,760,686	8760686	2	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.0	1

Tiempos:

- Con 1TB HDD sobre Windows: 1 minuto 23 segundos
- Con 1TB SSD sobre Linux: 54.6 segundos
- Con 250 SSD sobre linux: 47.9 segundos
- Con 1TB SSD m.2 Linux: 7.92 segundos

In [35]:

```
%%time

df_vaex_3.describe()
```

```
CPU times: user 8.04 s, sys: 137 ms, total: 8.18 s
Wall time: 1.29 s
```

Out[35]:

	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
data_type	int64	int64	string	string	int64	float64
count	8760687	8760687	8760687	8760687	8760687	8760687
NA	0	0	0	0	0	0
mean	4380343.0	1.560978037452999	--	--	1.6068074341658365	2.8040221012347804
std	2528992.49887	0.496268	--	--	1.25842	64.120494
min	0	1	--	--	0	0.0
max	8760686	2	--	--	9	189483.84

In [31]:

```
%%time
df_vaex_3.export('yellow_tripdata_2018-01.csv.hdf5')
```

```
CPU times: user 1.36 s, sys: 954 ms, total: 2.31 s
Wall time: 2.07 s
```

Tiempos:

- Con 1TB HDD sobre Windows: 1 minuto 19 segundos (53.8 segundos en otro intento)
- Con 1TB SSD sobre Linux: 9.73 segundos
- Con 250 SSD sobre linux: 7.13 segundos
- Con 1TB SSD m.2 Linux: 1.36 segundos

In [32]:

```
%%time
df_vaex_4 = vaex.open('yellow_tripdata_2018-01.csv.hdf5')
df_vaex_4
```

```
CPU times: user 6.31 ms, sys: 607 µs, total: 6.92 ms
Wall time: 6.2 ms
```

Out[32]:

#	Unnamed: 0	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID
0	0	1	2018-01-01 00:21:05	2018-01-01 00:24:23	1	0.5	1
1	1	1	2018-01-01 00:44:55	2018-01-01 01:03:05	1	2.7	1
2	2	1	2018-01-01 00:08:26	2018-01-01 00:14:21	2	0.8	1
3	3	1	2018-01-01 00:20:22	2018-01-01 00:52:51	1	10.2	1
4	4	1	2018-01-01 00:09:18	2018-01-01 00:27:06	2	2.5	1
...	...	...	...	...	...	...	...
8,760,682	8760682	1	2018-01-31 23:21:35	2018-01-31 23:34:20	2	2.8	1
8,760,683	8760683	1	2018-01-31 23:35:51	2018-01-31 23:38:57	1	0.6	1
8,760,684	8760684	2	2018-01-31 23:28:00	2018-01-31 23:37:09	1	2.95	1
8,760,685	8760685	2	2018-01-31 23:24:40	2018-01-31 23:25:28	1	0.0	1
8,760,686	8760686	2	2018-01-31 23:28:16	2018-01-31 23:28:38	1	0.0	1

Tiempos:

- Con 1TB HDD sobre Windows: 9.28 segundos (5.83 segundos en otro intento, 84 milisegundos en otro)
- Con 1TB SSD sobre Linux: 61.1 milisegundos
- Con 250 SSD sobre linux: 142 milisegundos
- Con 1TB SSD m.2 Linux: 6.31 milisegundos

CONCLUSIÓN

Cuando se genera con Vaex el hdf5 lo ejecuta más rapido que sin él, sino tarda más que el resto.

PANDAS DASK VAEX Tiempo CSV CSV HDF5 CSV Read 5.99 s 560 ms 9.53 ms 7.92 s Describe 2.3 s 9.43 s 8.1 s 8.04 s