

Modern pandas

Hervé Mignot  
EQUANCY



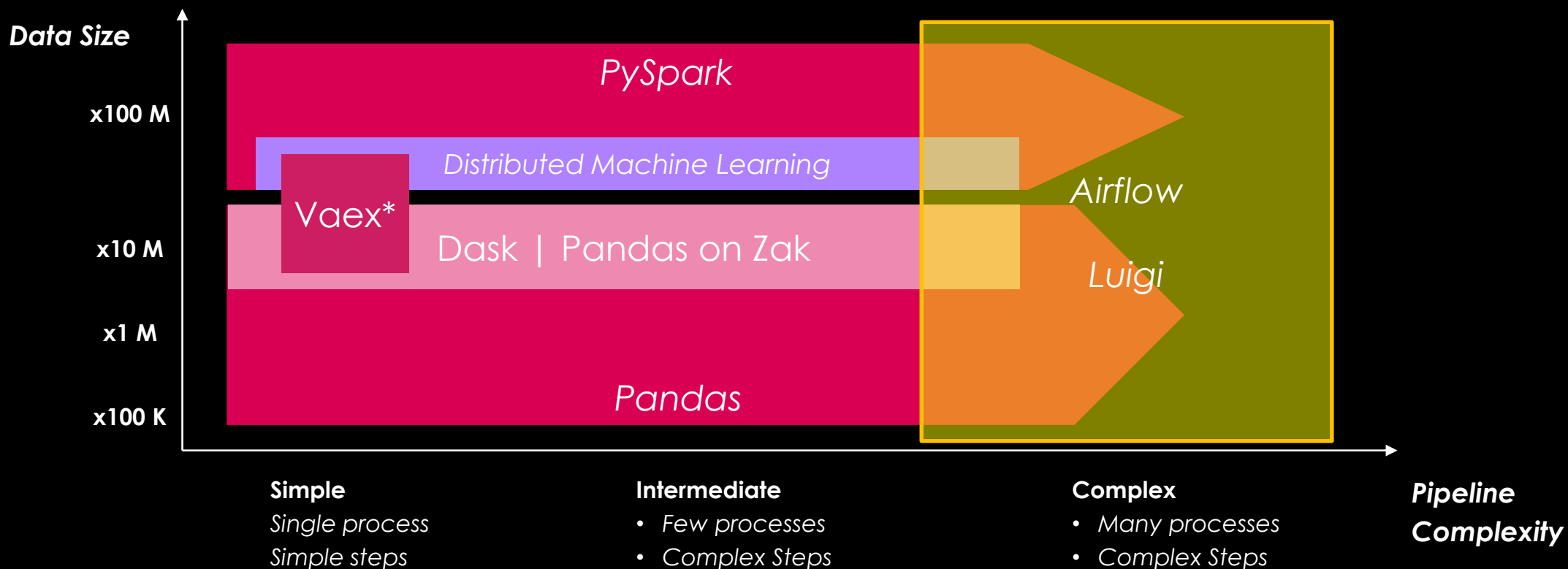
PyParis

14  
&  
15

November 2018  
EPITA, Paris



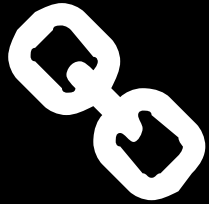
# Building Pipelines with Python



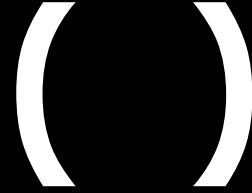
\* See the slides presented at PyParis 2018 here: <https://github.com/maartenbreddels/talk-pyparis-2018>

# Our tools

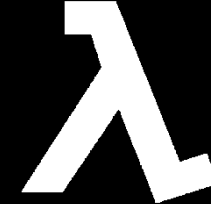
*Using pandas to build data transformation pipelines*



**Method Chaining**



**Brackets**



**lambda**

# Full credits to Tom Augspurger (@TomAugspurger)



<https://tomaugspurger.github.io/>

*Effective Pandas*

<https://leanpub.com/effective-pandas>

- ❑ *Effective Pandas*
- ❑ *Method Chaining*
- ❑ *Indexes*
- ❑ *Fast Pandas*
- ❑ *Tidy Data*
- ❑ *Visualization*
- ❑ *Time Series*



# Modern Pandas – Method Chaining

*Method chaining is composing functions application over an object.*

*Many data libraries API inspired from this functional programming pattern:*

- *dplyr (R)*
- *Apache Spark (Scala, Python, R)*
- ...

*Example (reading a csv file, renaming a column, taking the first 6 rows into a pandas dataframe) :*

```
df = pd.read_csv('myfile.csv').rename(columns={'old_col': 'new_col'}).head(6)
```

**vs.**

```
df = pd.read_csv('myfile.csv')  
df = df.rename(columns={'old_col': 'new_col'})  
df = df.head(6)
```



# Modern Pandas – Functions

**Method chaining is composing functions application over an object**

What?	Method
Compute columns	<code>.assign(col = val,           col = val,)</code>
Drop columns, rows	<code>.drop('val', axis=[0 1])</code> <code>.loc[condition for rows to be kept, list of columns]</code>
Call user defined function	<code>.pipe(fun, [args])</code>
Rename columns or index	<code>.rename(columns=mapper)</code> <code>.rename(mapper, axis=['columns'   'index'])</code>
Copy or replace	<code>.where(cond, other)</code>
Filter rows on “where expr”	<code>.query(where expr)</code> <code>.loc[dataframe expression using where expr]</code>
Drop missing values	<code>.dropna([subset=list])</code>
Sort against values	<code>.sort_values([subset=list])</code>



**and many others classical pandas DataFrame methods**

# Hands-on!



kata



# Our data set

The screenshot shows the website 'Le prix des carburants' with a navigation menu. The 'Données publiques' button is highlighted. Below the menu, the 'Données publiques' section is titled, followed by a paragraph explaining that the site provides open data on fuel prices. A list of URLs for instant, daily, and annual data is provided. A white arrow points from the text 'Les URL permettant de télécharger les données sont les suivantes :' to the list of URLs.

Le prix des carburants  
l'énergie est notre avenir, économisons-la

Données publiques

Le site gouvernemental des prix des carburants met à disposition de manière libre et gratuite (Open data) les données relatives aux prix des carburants. Ces données sont référencées sur la plate-forme des données publiques de l'État ([www.data.gouv.fr](http://www.data.gouv.fr)) et sont fournies sous la "Licence ouverte / Open licence". Les URL permettant de télécharger les données sont les suivantes :

- <https://donnees.roulez-eco.fr/opendata/instantane> - pour le flux instantané (-10 minutes)
- <https://donnees.roulez-eco.fr/opendata/jour> - pour le flux quotidien
- <https://donnees.roulez-eco.fr/opendata/annee> - pour le stock de l'année courante.

Il est possible de télécharger les données pour une date ou année souhaitée.

Ainsi pour les flux quotidiens il est possible de télécharger les données des 30 derniers jours en ajoutant la date au format AAAAMMJJ (année, mois, jour) en fin d'URL

<https://donnees.roulez-eco.fr/opendata/jour/AAAAMMJJ>

Les archives annuelles sont disponibles à partir de l'année 2007 aux URL suivantes :

- <https://donnees.roulez-eco.fr/opendata/annee/2017>
- <https://donnees.roulez-eco.fr/opendata/annee/2016>
- <https://donnees.roulez-eco.fr/opendata/annee/2015>
- <https://donnees.roulez-eco.fr/opendata/annee/2014>
- <https://donnees.roulez-eco.fr/opendata/annee/2013>
- <https://donnees.roulez-eco.fr/opendata/annee/2012>
- <https://donnees.roulez-eco.fr/opendata/annee/2011>
- <https://donnees.roulez-eco.fr/opendata/annee/2010>
- <https://donnees.roulez-eco.fr/opendata/annee/2009>
- <https://donnees.roulez-eco.fr/opendata/annee/2008>

<https://www.prix-carburants.gouv.fr/rubrique/opendata/>

0	1	2	3	4	5	6	7	8	9
0	1000001	1000	R	4620114.0	519791.0	2016-01-02T09:01:58	1.0	Gazole	1026.0
1	1000001	1000	R	4620114.0	519791.0	2016-01-04T10:01:35	1.0	Gazole	1026.0

<https://github.com/rvm-courses/GasPrices>





# Reading & preparing the data

```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                  )
      # Rename columns
      .rename(columns={0: 'station_id', 1: 'zip_code',
                       3: 'latitude', 4: 'longititude', 5: 'date',
                       7: 'gas_type', 8: 'price'})
      # Recompute columns
      .assign(
          price = lambda x: x['price'] / 1000,
          latitude = lambda x: x['latitude'] / 100000,
          longitude = lambda x: x.longitude / 100000,
      )
      # Drop columns
      .drop([2, 6,], axis=1)
      )
```



# Reading & preparing the data – 1/4

```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                  )
      # Rename columns
      .rename(columns={0: 'station_id', 1: 'zip_code',
                      3: 'latitude', 4: 'longititude', 5: 'date',
                      7: 'gas_type', 8: 'price'})
      # Recompute columns
      .assign(
          price = lambda x: x['price'] / 1000,
          latitude = lambda x: x['latitude'] / 100000,
          longitude = lambda x: x.longitude / 100000,
      )
      # Drop columns
      .drop([2, 6,], axis=1)
      )
```



## Reading & preparing the data – 2/4

```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                  )
      # Rename columns
      .rename(columns={0: 'station_id', 1: 'zip_code',
                      3: 'latitude', 4: 'longititude', 5: 'date',
                      7: 'gas_type', 8: 'price'})
      # Recompute columns
      .assign(
          price = lambda x: x['price'] / 1000,
          latitude = lambda x: x['latitude'] / 100000,
          longitude = lambda x: x.longitude / 100000,
      )
      # Drop columns
      .drop([2, 6,], axis=1)
      )
```



# Reading & preparing the data – 3/4

```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                  )
      # Rename columns
      .rename(columns={0: 'station_id', 1: 'zip_code',
                      3: 'latitude', 4: 'longitude', 5: 'date',
                      7: 'gas_type', 8: 'price'})
      # Recompute columns
      .assign(
          price = lambda x: x['price'] / 1000,
          latitude = lambda x: x['latitude'] / 100000,
          longitude = lambda x: x.longitude / 100000,
      )
      # Drop columns
      .drop([2, 6,], axis=1)
      )
```



# Reading & preparing the data – 4/4

```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                  )
      # Rename columns
      .rename(columns={0: 'station_id', 1: 'zip_code',
                      3: 'latitude', 4: 'longititude', 5: 'date',
                      7: 'gas_type', 8: 'price'})
      # Recompute columns
      .assign(
          price = lambda x: x['price'] / 1000,
          latitude = lambda x: x['latitude'] / 100000,
          longitude = lambda x: x.longitude / 100000,
      )
      # Drop columns
      .drop([2, 6,], axis=1)
      )
```



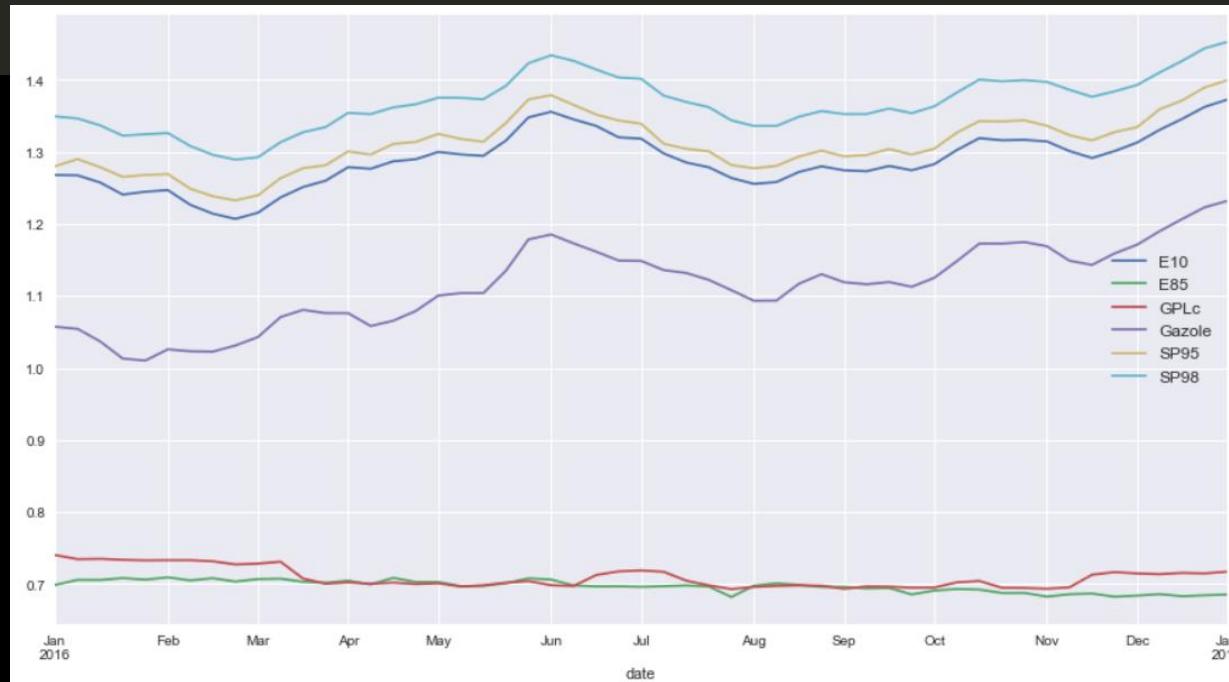
# Result

	station_id	zip_code	latitude	longitude	date	gas_type	price
0	1000001	01000	46.20114	5.19791	2016-01-02 09:01:58	Gazole	1.026
1	1000001	01000	46.20114	5.19791	2016-01-04 10:01:35	Gazole	1.026
2	1000001	01000	46.20114	5.19791	2016-01-04 12:01:15	Gazole	1.026
3	1000001	01000	46.20114	5.19791	2016-01-05 09:01:12	Gazole	1.026
4	1000001	01000	46.20114	5.19791	2016-01-07 08:01:13	Gazole	1.026



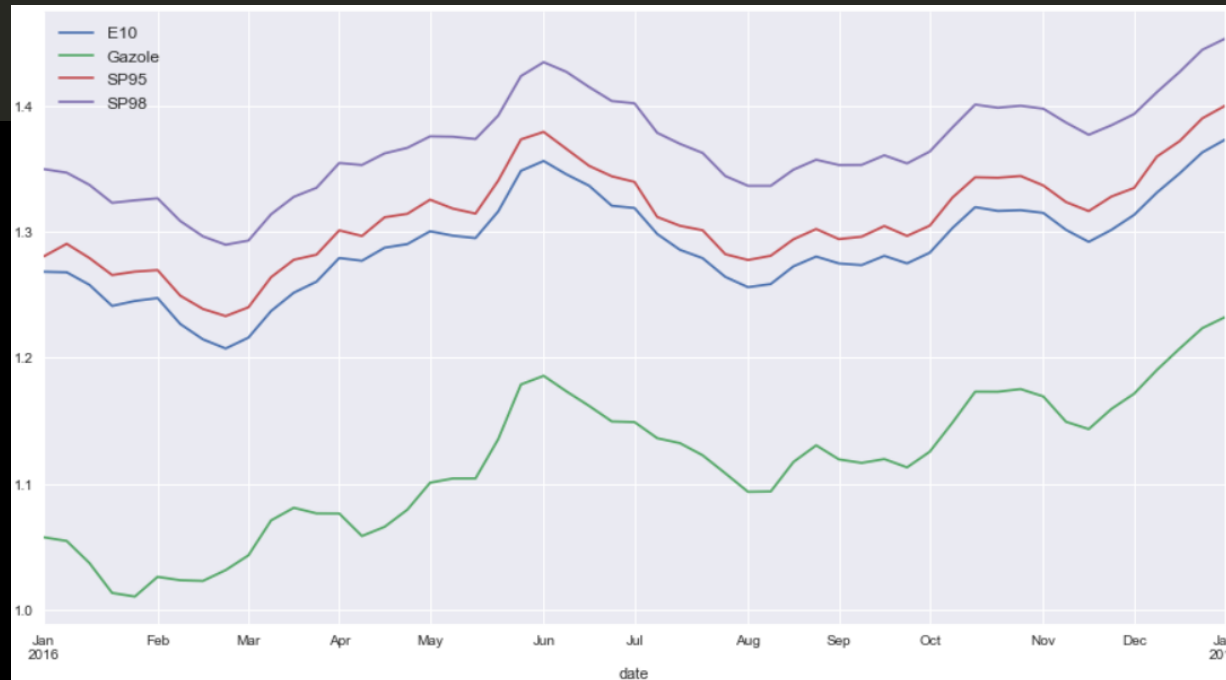
# Charting prices evolutions

```
(df
 .dropna(subset=['date'])
 .groupby(['gas_type', pd.Grouper(key='date', freq='1W')])
 ['price']
 .mean()
 .unstack(0)
 .rename_axis('Gas price changes', axis=1)
 .plot()
)
```



# Charting prices evolutions

```
(df
  .dropna(subset=['date'])
  .loc[df['gas_type'].isin(df['gas_type'].value_counts().index[:4])]
  .groupby(['gas_type', pd.Grouper(key='date', freq='1W')])
  ['price']
  .mean()
  .unstack(0)
  # .rename_axis('Gas price changes', axis=1)
  .plot()
)
```





# Data Quality – Chained Assertions

**Use assertions for testing constraints against data frames**

*engarde is a module defining a set of functions & decorators to check these*

*Defining methods (monkey patching) on `pd.DataFrame` allows chained assertions*

is_shape
none_missing
unique_index
within_range
within_set
has_dtypes
...

```
import engarde
# Adding a method to pandas data frame
pd.DataFrame.check_is_shape = engarde.checks.is_shape

stations_df = (pd.read_csv('./Stations2017.zip', sep='|', header=None, dtype={1: str},
                        names=['station_id', 'zip_code', 'type',
                              'latitude', 'longitude', 'address', 'city'],
                        )
               # Verify data frame structure
               .check_is_shape((None, 7))
               .assign(latitude = lambda x: x.latitude / 100000, longitude = lambda x: x.longitude / 100000)
               )
```



# Logging & debugging

## Encapsulate logging calls within pandas DataFrame methods

No module known, could be an addition to engarde (Tom Augspurger discussed logging)

```
import logging
...
def log_shape(df):
    logging.info('%s' % df.shape)
    return df

pd.DataFrame.log_shape = log_shape

stations_df = (pd.read_csv('./Stations2017.zip', sep='|', header=None, dtype={1: str},
                        names=['station_id', 'zip_code', 'type',
                              'latitude', 'longitude', 'address', 'city'],
                        )
              # Verify data frame structure
              .log_shape()
              .assign(latitude = lambda x: x.latitude / 100000, longitude = lambda x: x.longitude / 100000)
              )
```



Merci



See you soon...

modernpandas.io



Banksy



# equancy

---

47 rue de Chaillot  
75116 Paris - FRANCE

[www.equancy.com](http://www.equancy.com)



Hervé Mignot



[herve.mignot@equancy.com](mailto:herve.mignot@equancy.com)