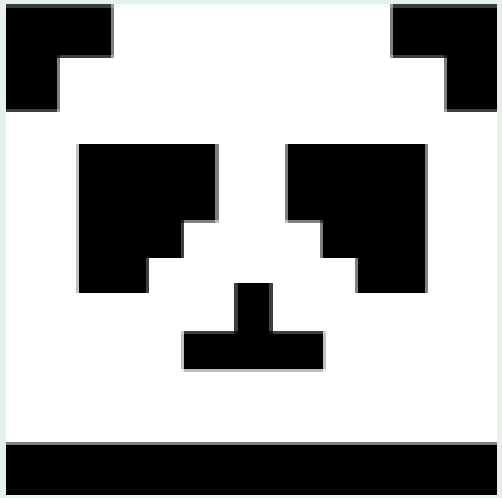


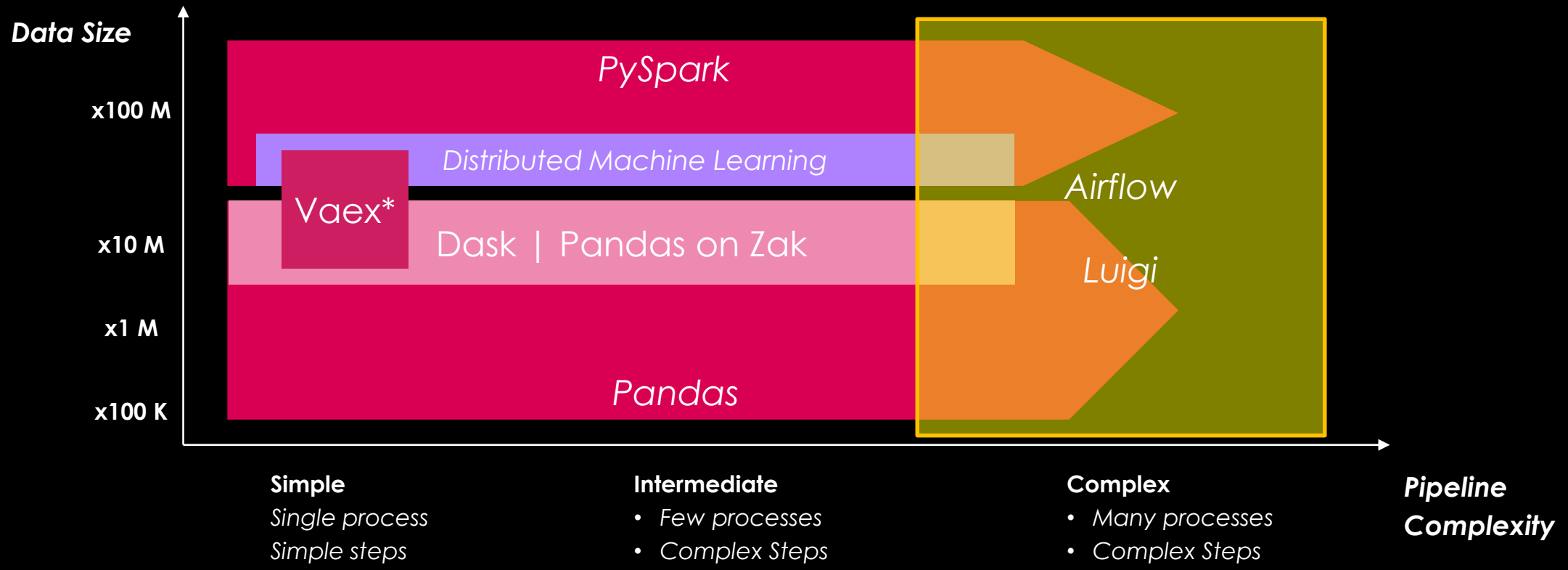
Modern pandas

Hervé Mignot
EQUANCY

PyParis

14 & 15 November 2018
EPITA, Paris

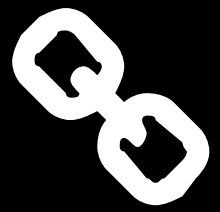
Building Pipelines with Python



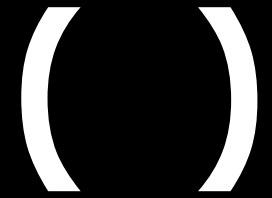
* See the slides presented at PyParis 2018 here: https://github.com/maartenbreddels/talk-pyparis-2018

Our tools

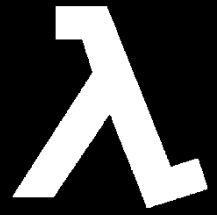
Using pandas *to build data transformation pipelines*



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

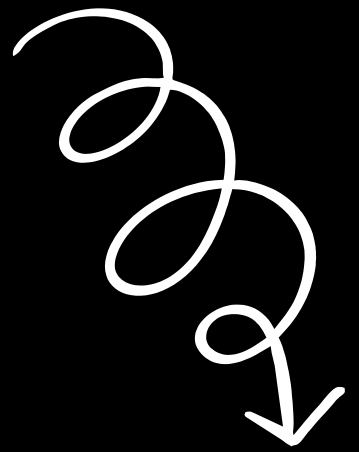
and many others classical pandas DataFrame methods

Hands-on!

kata

Our data set

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: '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 – 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: '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 – 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: '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 – 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

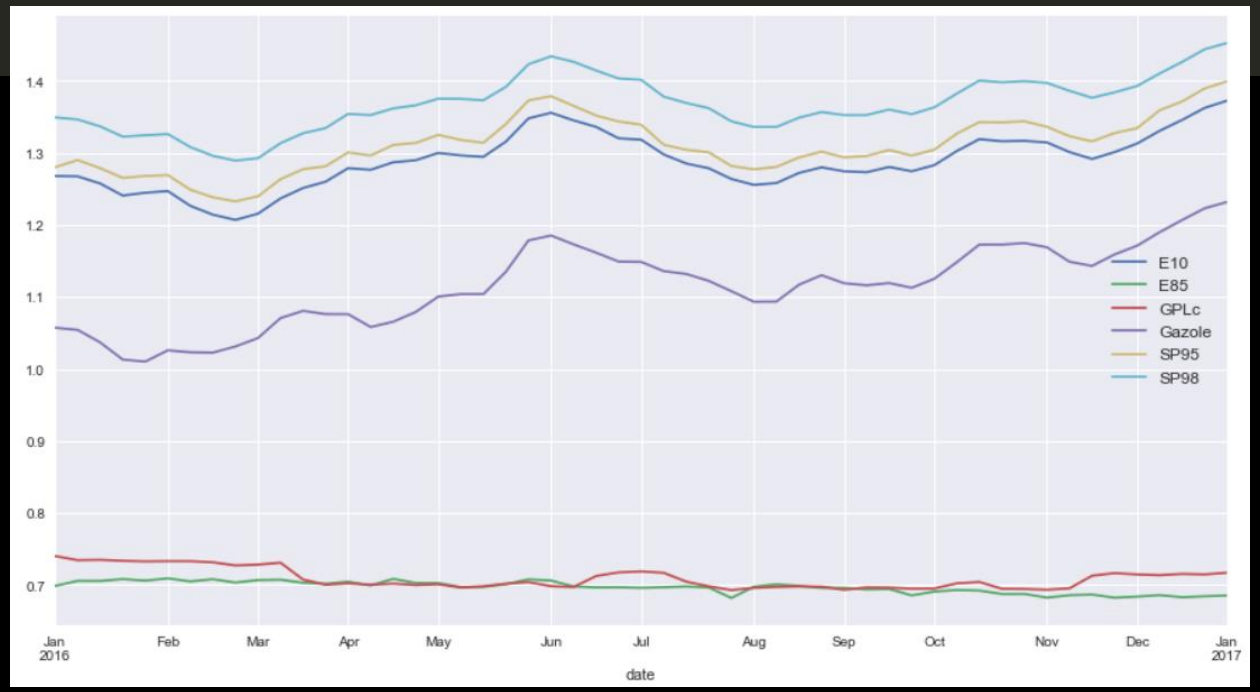
```
df = (pd.read_csv('./Prix2017.zip',
                  sep=';',
                  header=None,
                  dtype={1: str},
                  parse_dates = [5],
                 )
      # Rename columns
      .rename(columns={0: 'station_id', 1:'zip_code',
                       3: 'latitude', 4: '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)
)
```

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

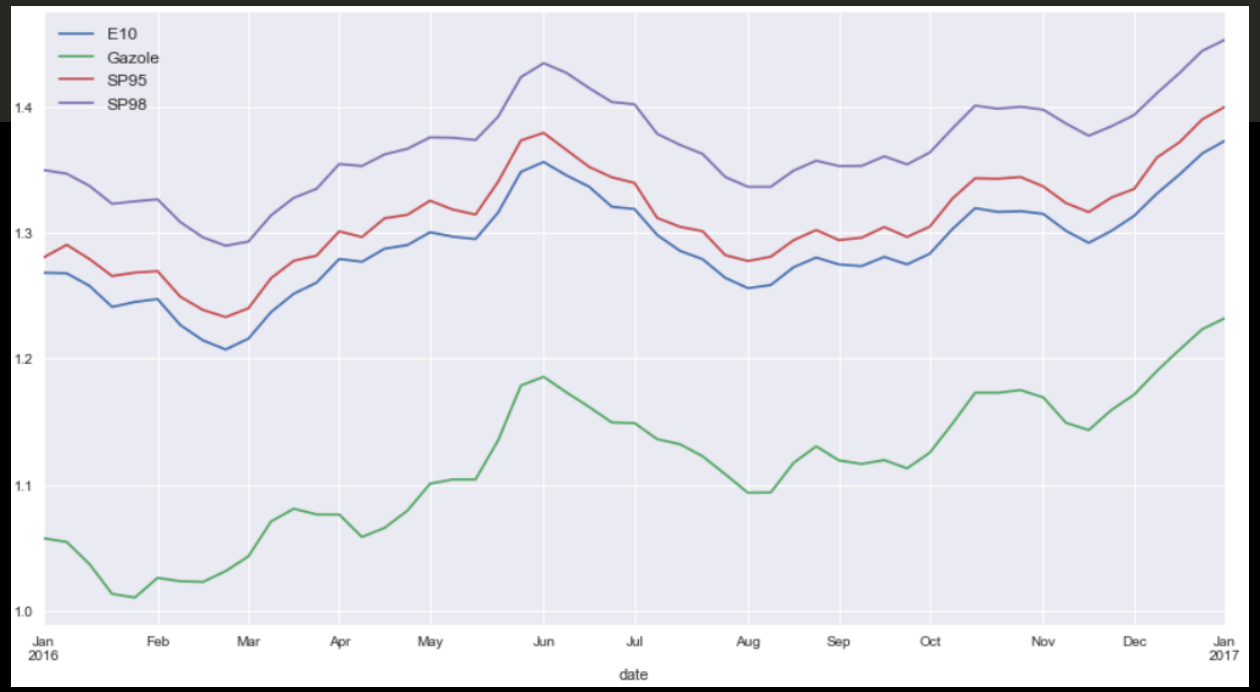
Charting prices evolutions

```
(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)
              )
```

equancy

Merci

Made with Pygments & Consolas
Pixel Pandas by Kira Chao

See you soon on...

modernpandas.io

Bansky

Made with Pygments & Consolas
Pixel Pandas by Kira Chao

equancy

47 rue de Chaillot
75116 Paris - FRANCE

www.equancy.com

Hervé Mignot

herve.mignot at equancy.com