ARGUMENTS FOR BUILDING YOUR OWN DATA VISUALIZATION PLATFORM FROM SCRATCH

Artem Seleznev

Big Data Analyst, Megafon

PyCon Belarus

What's a problem?

Commercial

- QlikView
- Klipfolio
- Tableau
- Power BI Pro

and etc…

Pricey!
Not invented here

What's a problem?

Commercial

- QlikView
- Klipfolio
- Tableau
- Power BI Pro

and etc…

Pricey!
Not invented here

Open Source

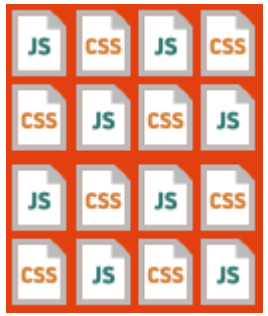
Repositories 3K

Recommended to use:

- Grafana
- Redash
- Metabase

What does it hide?

- Where is Python?



- Error of a group: **ConnectionError**

- Non-aggregated data

What does it hide?

- Where is Python?
- Error of a group: **ConnectionError** →
  - BrokenPipeError
  - ConnectionAbortedError
  - ConnectionRefusedError
  - ConnectionResetError
- Non-aggregated data

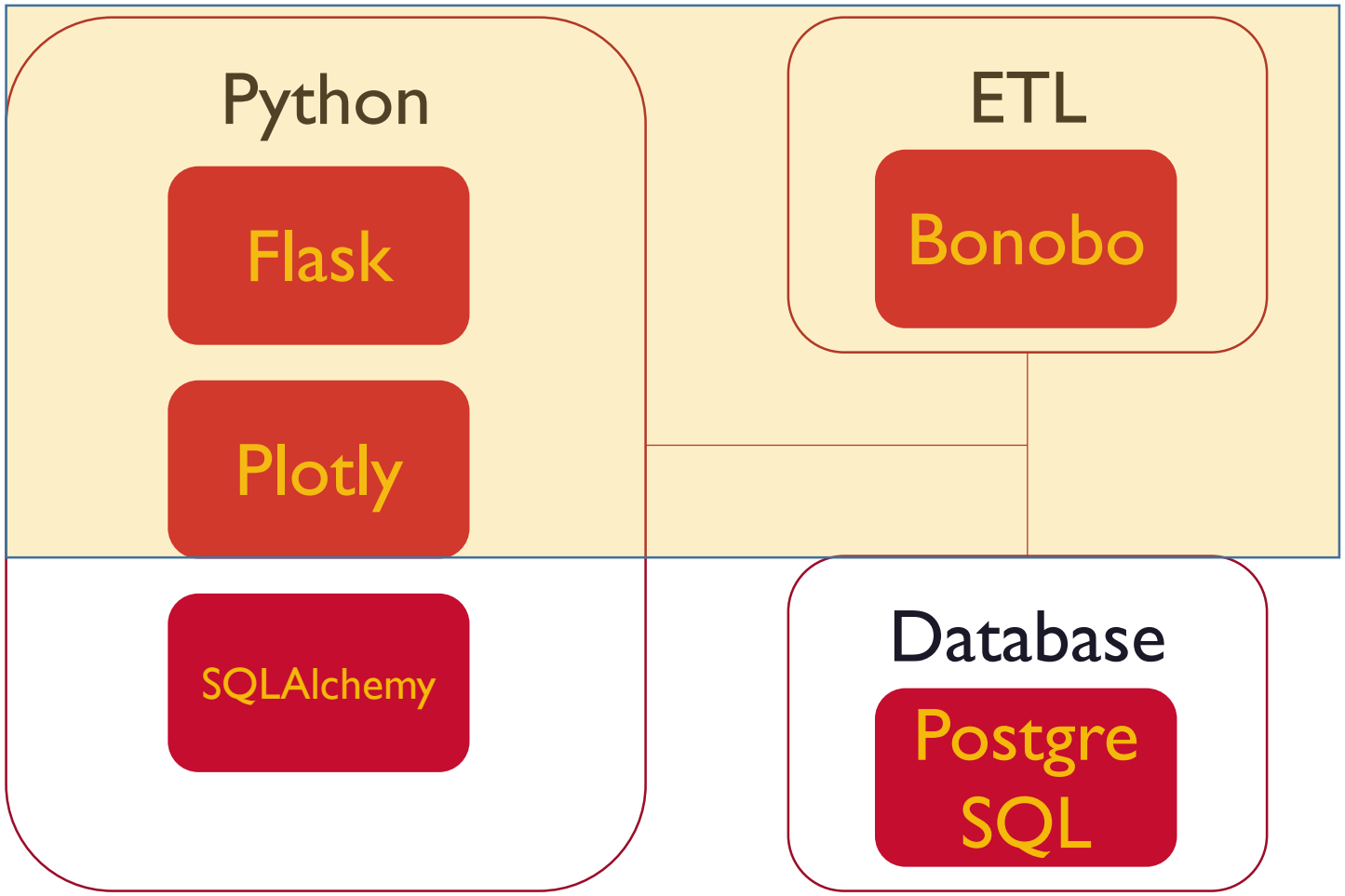
What does it hide?

- Where is Python?
- Error of a group: **ConnectionError**
- Non-aggregated data

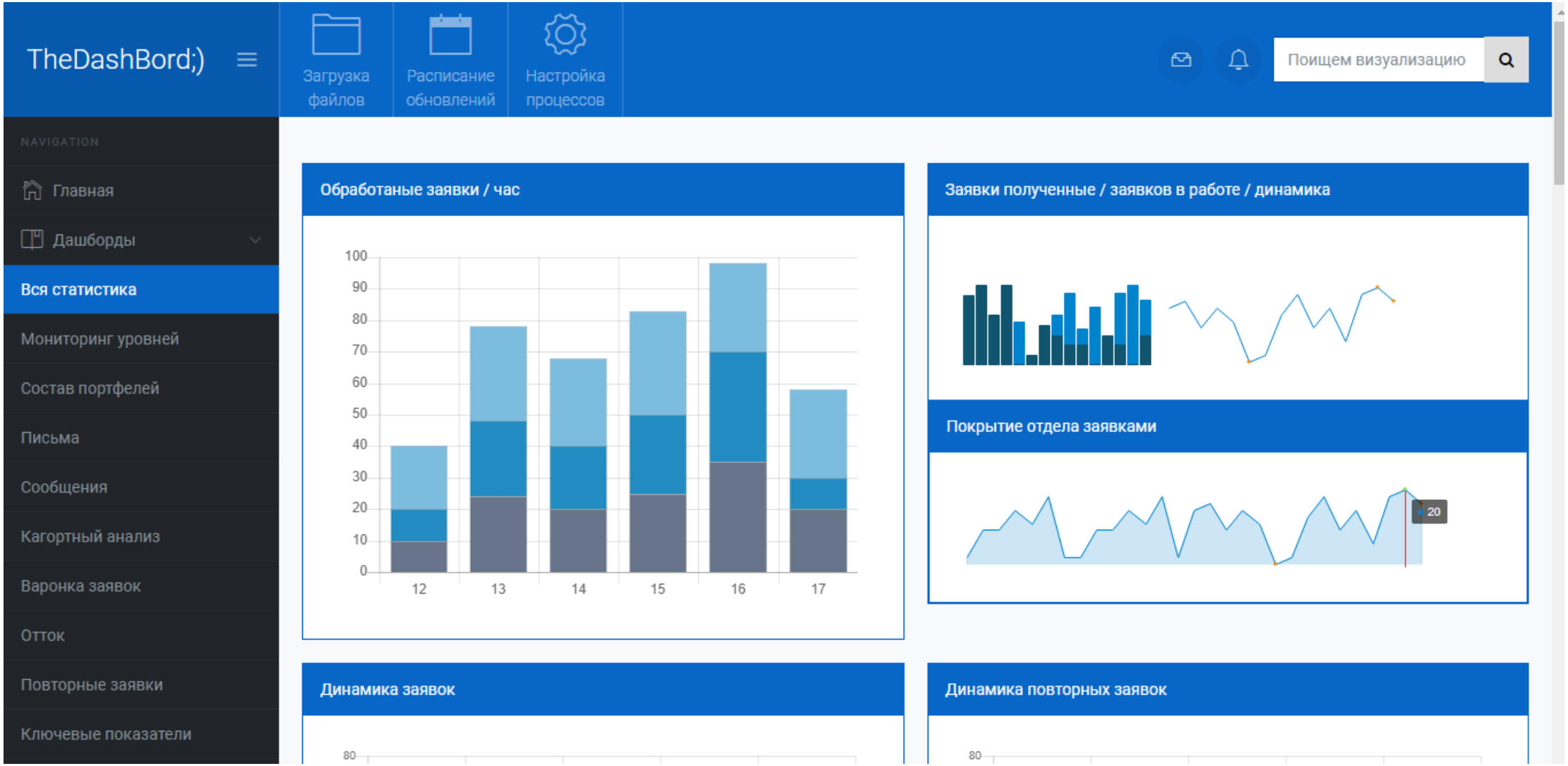
```
SELECT customer.id,
       count(customer.scoring)
FROM customer
```

```
SELECT customer.id,
       customer.scoring
FROM customer
```

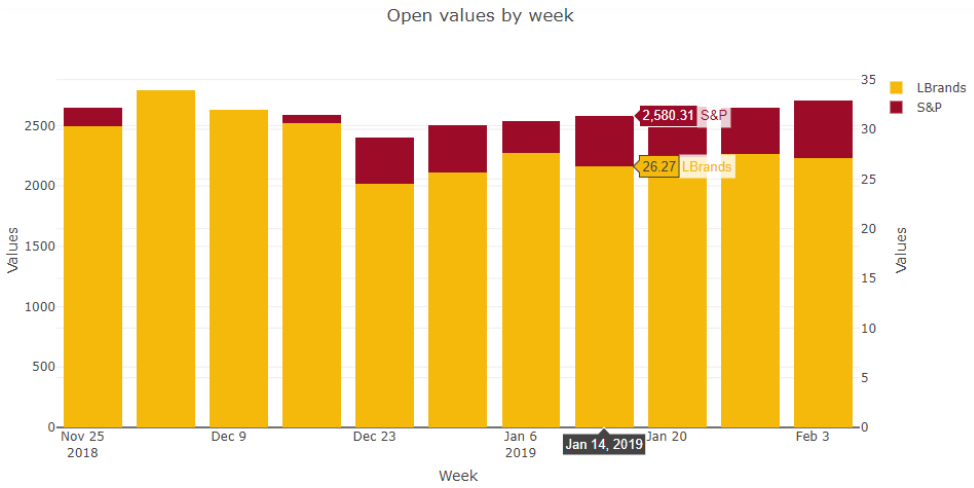
My aim (blocks)



My aim (as the result-dashboard)



Make an object




```
1  bar_data
```

```
{'type': 'bar', 'x': 0     2018-11-26
 1     2018-12-03
 2     2018-12-10
 3     2018-12-17
       ....
 Name: Date, dtype: object, 'y': 0     2649.97
 1     2790.50
 2     2630.86
 3     2590.75
       ....
 Name: Open, dtype: float64, 'marker': {'color': 'rgb(156,12,41)'}, 'name': 'S&P'}
```

```
1  bar_data
```

```
Figure(id = 'c820adfd-e2ea-4356-a2c2-d52502fb47c0', ‹‹‹
        above = [],
        aspect_scale = 1,
        background_fill_alpha = {'value': 1.0},
        below = [CategoricalAxis(id='b69a7358-72e1-4116-a3c7-
        border_fill_alpha = {'value': 1.0},
        ....
```

Make an object

```
years = data

data_2 = {'Categories' : categories,
        year[1] : grouped_data.\
        loc[(grouped_data["Year"]>=2018),"IncidntNum"]}

x = [ (category, year) for category in categories for year in years ]
counts = sum(zip(data_2),()) |

source = ColumnDataSource(data=dict(x=x, counts=counts))

TOOLS = "hover,save,pan,box_zoom,reset,wheel_zoom"

p = figure(x_range=FactorRange(*x),
           height=500,
           width = 700,
           title="S&P",
           toolbar_location=None,
           tools=TOOLS,
           y_axis_label = 'Values')

p.vbar(x='x',
       top='counts',
       width=0.9,
       source=source,
       line_color="white",
       fill_color=factor_cmap('x',
                              palette=bokeh.palettes.Category10[3],
                              factors=years, start=1, end=2))

show(p)
```

Hard way

Make an object

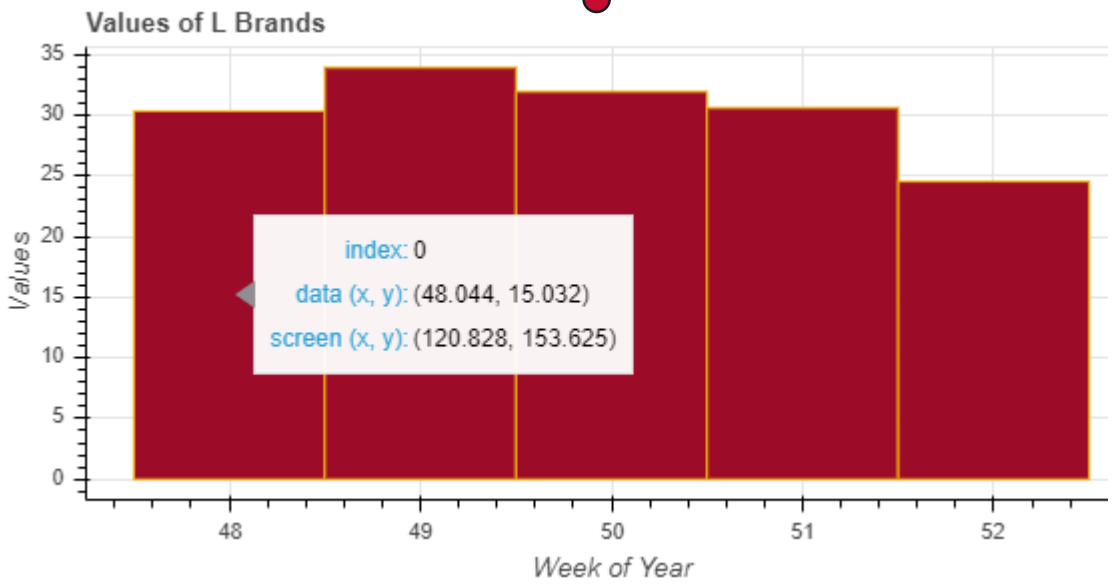
THX, Plotly!

It's JSON

```
{ 'data' : { 'type' :  'bar',
             'x‘ :  [2018-11-26, ..., 2019-02-04],
             'y‘ :  [2649.97, ..., 2706.49],
             'marker‘ :  { 'color‘ :  'rgb(156,12,41)‘ },
             'name‘ :  'S&P‘ },
 'layout‘ :  { 'title‘ :  'Open values by week ',
               'xaxis‘ : { 'title‘ : 'Week‘ },
               'yaxis‘ : { 'title‘ :  'Values',
                           'side‘ : 'left‘ },
               'yaxis2‘ :  { 'title‘ : 'Values', 'side‘ : 'right‘ },
               'barmode': 'stack'}}
```

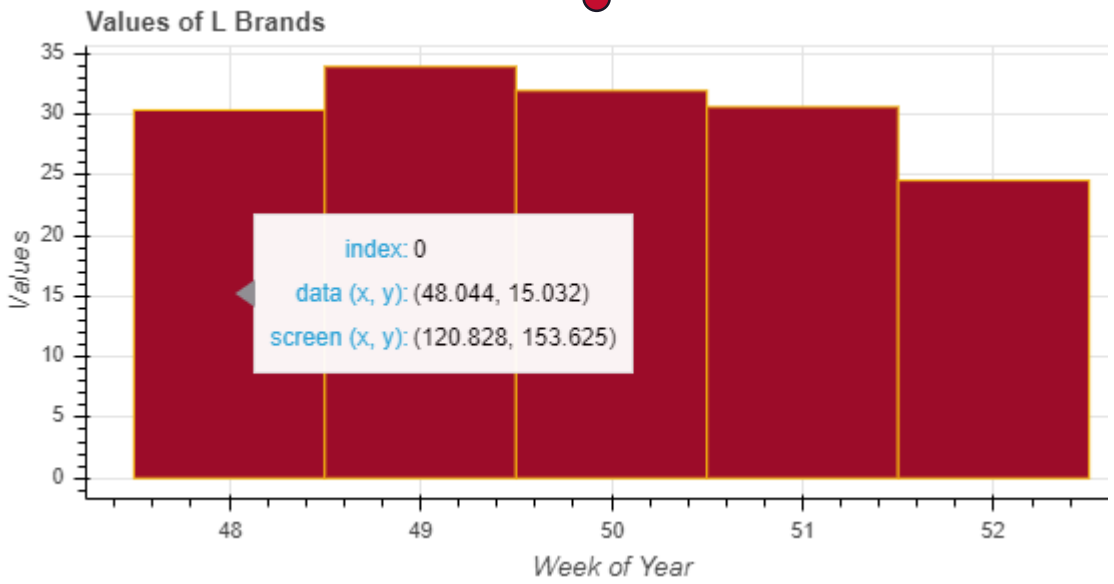
Change type according to an object behavior

```
if isinstance(object, list or tuple or dict):
    ...
```



Change type according to an object behavior

```
if isinstance(object, list or tuple or dict):
    ...
import collections

if isinstance(object, collections.Iterable):
    ...
```



```python
from functools import singledispatch
from collections import  abc
import numbers


class Viz:
    ...
    @singledispatch
    def disp_func(self, obj):
        return '{}'.format(repr(obj))

    @disp_func.register(str)
    def _(self, text):
        content = some_dict[text]
        return '{}'.format(content)

    @disp_func.register(numbers.Integral)
    def _(self, n):
        return n

    @disp_func.register(list)
    @disp_func.register(abc.MutableSequence)
    def _(self, seq):
        addline(seq)
```

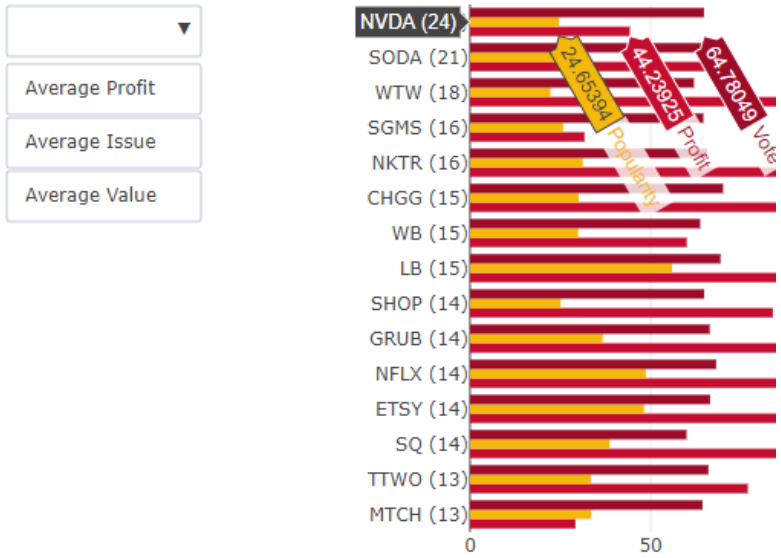
Do you use Python dispatcher ? (function.singledispatch)

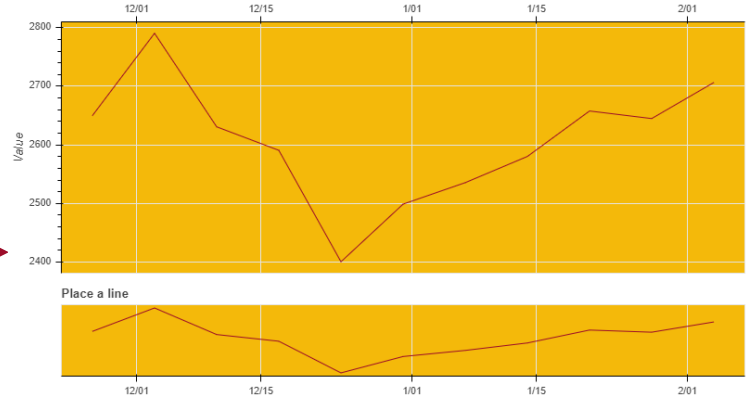
```python
from functools import singledispatch
from collections import  abc
import numbers


class Viz:
    ...
    @singledispatch
    def disp_func(self, obj):
        return '{}'.format(repr(obj))

    @disp_func.register(str)
    def _(self, text):
        content = some_dict[text]
        return '{}'.format(content)

    @disp_func.register(numbers.Integral)
    def _(self, n):
        return n

    @disp_func.register(list)
    @disp_func.register(abc.MutableSequence)
    def _(self, seq):
        addline(seq)
```

str(16) => ‘sixteen’

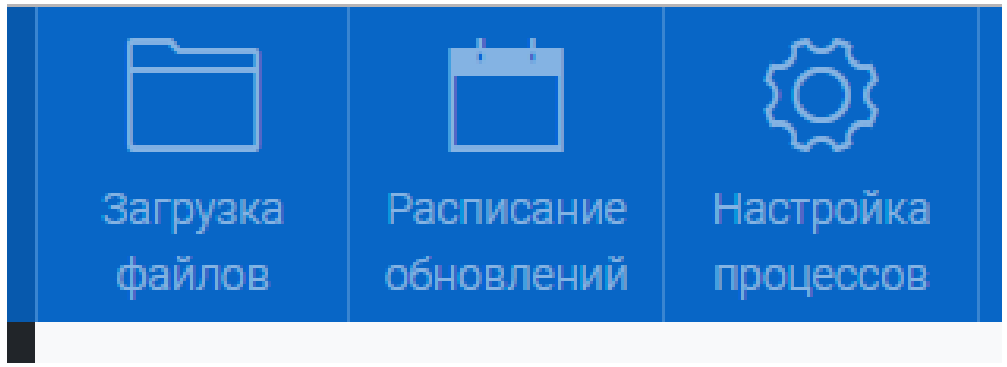
52

```python
from functools import singledispatch
from collections import  abc
import numbers


class Viz:
    ...
    @singledispatch
    def disp_func(self, obj):
        return '{}'.format(repr(obj))

    @disp_func.register(str)
    def _(self, text):
        content = some_dict[text]
        return '{}'.format(content)

    @disp_func.register(numbers.Integral)
    def _(self, n):
        return n

    @disp_func.register(list)
    @disp_func.register(abc.MutableSequence)
    def _(self, seq):
        addline(seq)
```





A little bit of ETL



Required framework

Luigi

```
import luigi
from luigi import Task
from luigi.contrib.sqla import CopyToTable

class ETL(CopyToTable, Task):
    …
    columns = [
        (["id", Integer], {"primary_key": True}),
        (["share", String], {})
    ]
    #define a table
    def process(self):
        SQL = "QUERY"
        def run(self):
            with psycopg2.connect(connect_str) as c:
                engine.execute(sql)
                 …
        def output(self):
            with psycopg2.connect(connect_str) as c:
                engine.execute(new_sql)
                 …
```

Bonobo

```
import bonobo
import bonobo_sqlalchemy

class ETL:
        …
        def get_etl(**options):
            graph = bonobo.Graph()
            graph.add_chain(bonobo_sqlalchemy.\
                            Select("QUERY"),
            … #your process
            bonobo_sqlalchemy.InsertOrUpdate('Out_Table')
            )

            return graph
```

Required framework

Luigi

```
import luigi
from luigi import Task
from luigi.contrib.sqla import CopyToTable

class ETL(CopyToTable, Task):
    …
    columns = [
        (["id", Integer], {"primary_key": True}),
        (["share", String], {})
    ]
    #define a table
    def process(self):
        SQL = "QUERY"
        def run(self):
            with psycopg2.connect(connect_str) as c:
                engine.execute(sql)
                …
        def output(self):
            with psycopg2.connect(connect_str) as c:
                engine.execute(new_sql)
                …
```

Bonobo

```
import bonobo
import bonobo_sqlalchemy

class ETL:
        …
        def get_etl(**options):
            graph = bonobo.Graph()
            graph.add_chain(bonobo_sqlalchemy.\
                            Select("QUERY"),
            … #your process
            bonobo_sqlalchemy.InsertOrUpdate('Out_Table')
            )

            return graph
```

ETL with Bonobo

```
def extract():
    ...
    result = engine.execute(sql)
    return result
```

```
def transform(result_sql):
    path = 'Some path to a folder'
    files = [one for one in listdir(path)]
    ...
```

```
def load(csv):
    ...
    data = data = genfromtxt(csv, delimiter=',',
                             skip_header=1,
converters={0: lambda s: str(s)})
    for i in data:
        record = Name(**{
            'col_name': i[n]
        })
        s.add(record)
    s.commit()
    ...
```

???

ETL with Bonobo

```python
def extract():
    ...
    result = engine.execute(sql)
    return result
```

```python
def transform(result_sql):
    path = 'Some path to a folder'
    files = [one for one in listdir(path)]
    ...
```

```python
def load(csv):
    ...
    data = data = genfromtxt(csv, delimiter=',',
                             skip_header=1,
converters={0: lambda s: str(s)})
    for i in data:
        record = Name(**{
            'col_name': i[n]
        })
        s.add(record)
    s.commit()
    ...
```

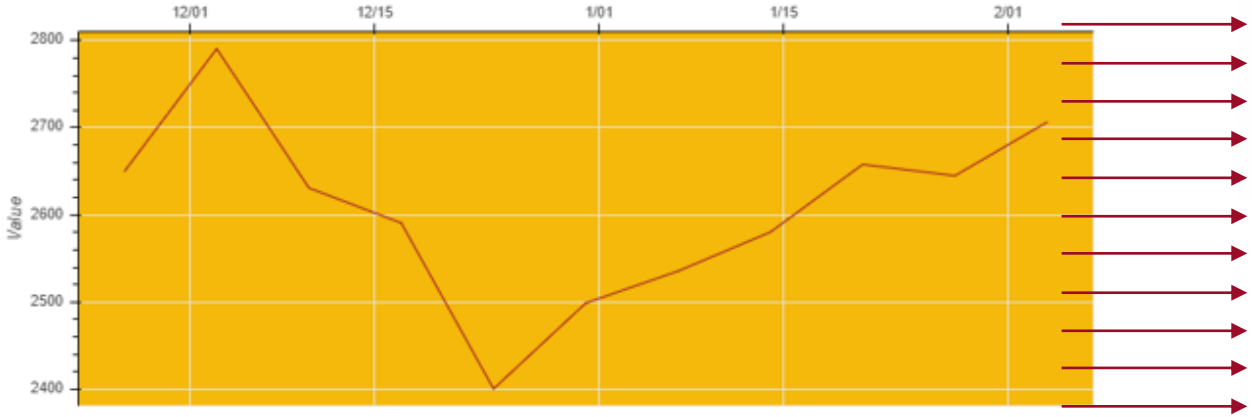
```python
import bonobo

graph = bonobo.Graph(
    result = engine.execute(sql),
    path = 'Some path to a folder'
    files = [one for one in listdir(path)]
    ...
    #execute files
    ...,
    data = data = genfromtxt(csv, delimiter=',',
                             skip_header=1,
                             converters={0:lambda s: str(s)})
    for i in data:
        record = Name(**{'col_name': i[n]})
        s.add(record)
    s.commit()

if __name__ == "__main__":
    bonobo.run(graph)
```

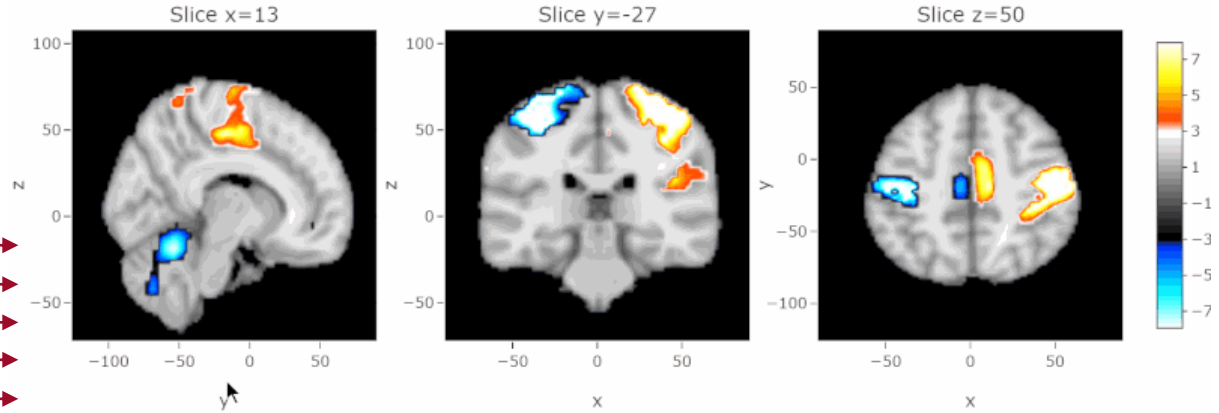
Summary

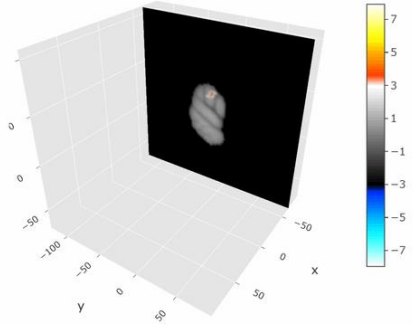
- Data visualization is a good way… thanks to Python classes!
- Make your own dashboard! (if you want)
- Python(Flask + Plotly) + DataBase + ETL(Bonobo)
  …etc (add what you want)
- Use docker
  => Install Portainer (https://www.portainer.io/) and manage it!

Summary

- Make more complicated objects….







Thank you!

fb: /seleznev.artem.info

telegram: @SeleznevArtem

If you are going to a hackathon

and

need teammates

Invite Me

PyCon Belarus