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!

No Python way

What's a problem?

Commercial

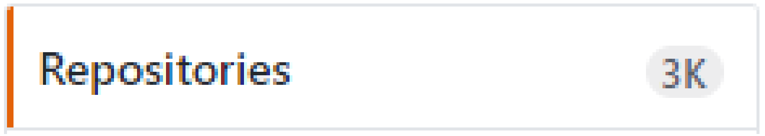
- ~~QlikView~~
- ~~Klipfolio~~
- ~~Tableau~~
- ~~Power BI Pro~~

~~and etc…~~

Pricey!

No Python way

Free of charge

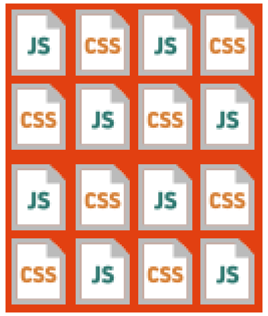

Repositories 3K

+

pitfalls

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** $\longrightarrow$
  - 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~~
```

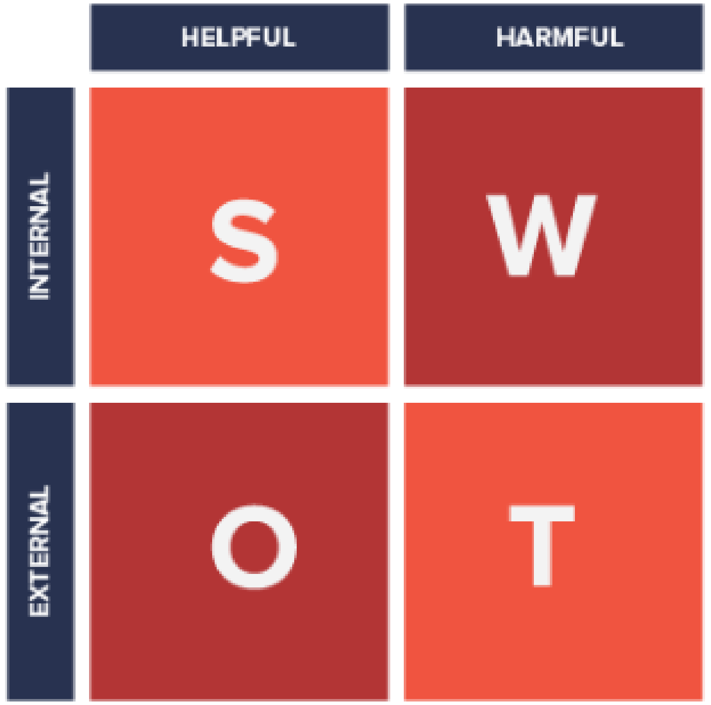
```
SELECT customer.id,
       customer.scoring
FROM customer
```

Advantages of your own visualization

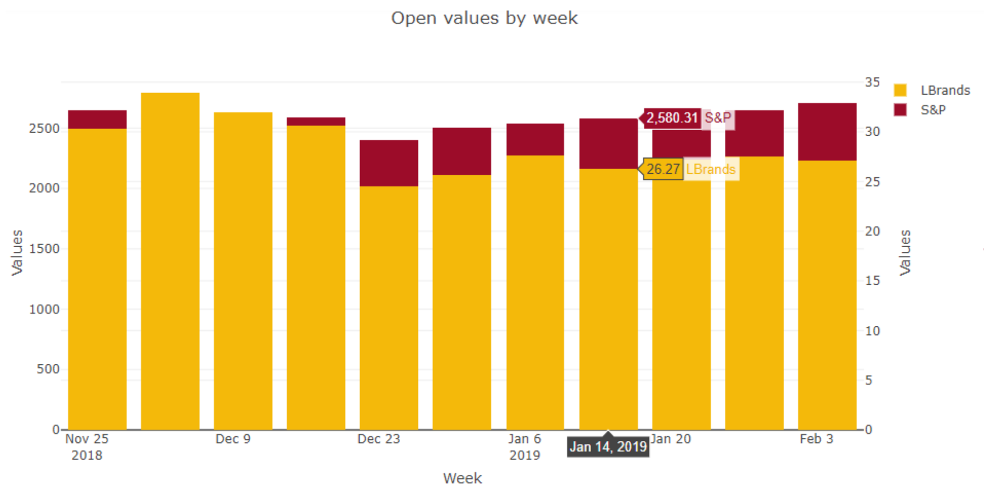
S Only necessary and helpful

O Improve Python
ETL (Luigi | Bonobo)
Docker

W Lots of VizLibs
(Plotly Dash, Bokeh, Pygal)



Make an object



```
bar_data
```

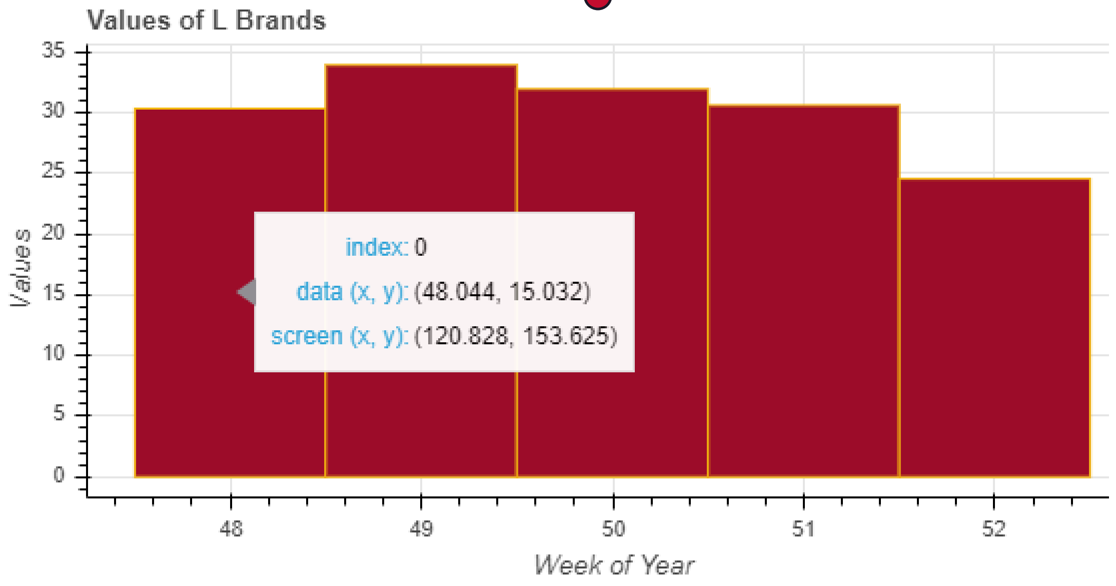
```
{'type': 'bar', 'x': 0     2018-11-26
 1     2018-12-03
 2     2018-12-10
 3     2018-12-17
        ....
 Name: Date, dtype: object, 'y': 0     2649.97
 1     2790.50
 2     2630.86
 3     2590.75
        ....
 Name: Open, dtype: float64, 'marker': {'color': 'rgb(156,12,41)'}, 'name': 'S&P'}
```

```
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},
        ....
```

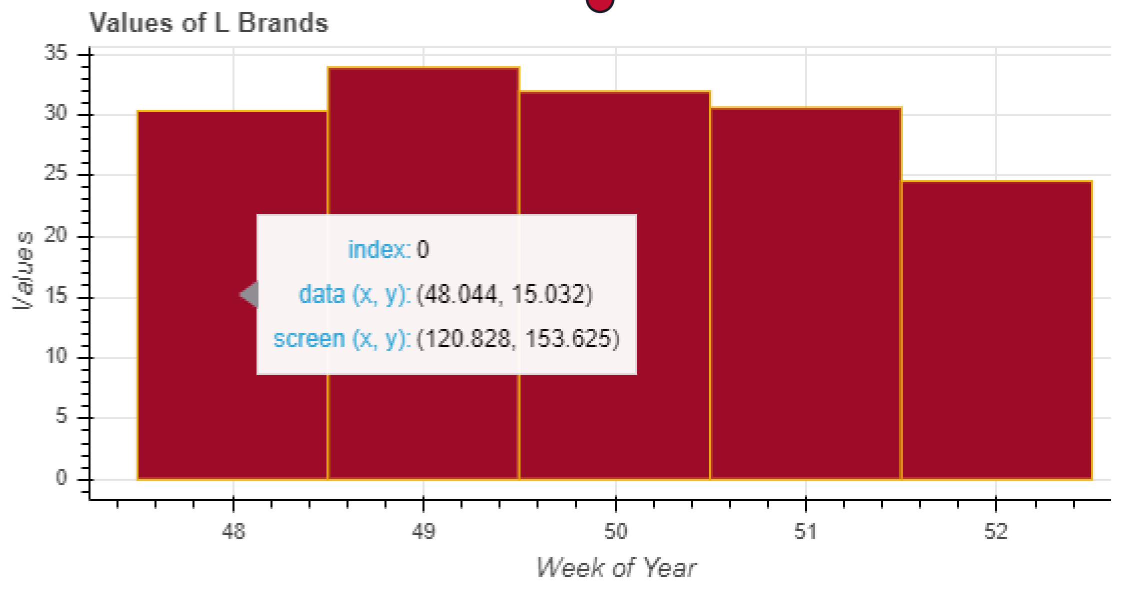
Change type according to an object behavior

```
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):
    ...
```



Do you use Python dispatcher ? (function.singledispatch)

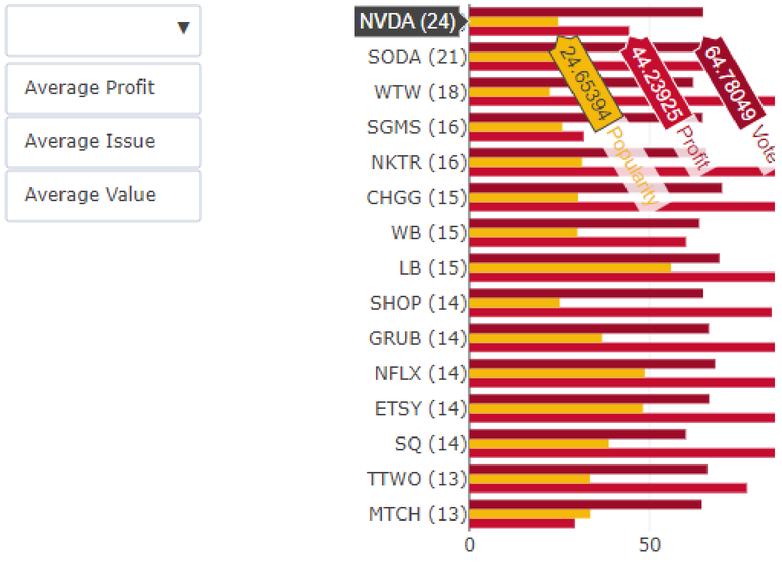
```
from functools import singledispatch
from collections import  abc
import numbers


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

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

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

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

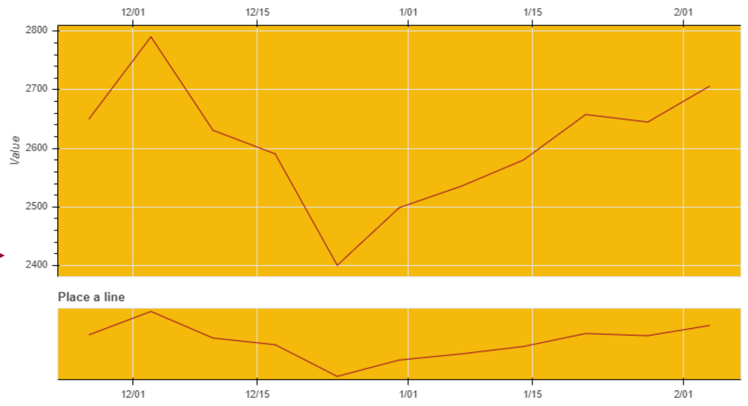
```python
from functools import singledispatch
from collections import  abc
import numbers


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

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

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

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

str(16) => 'sixteen'

52

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


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

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

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

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





SQL Constructor

```
from sqlalchemy import Integer, …
import sqlalchemy.ext.declarative.declarative_base as d
from sqlalchemy.orm import relationship

Base = d()

class Share(Base):
    __tablename__ = ‘share‘

    id = Column(Integer, primary_key=True)
    share_value = Column(Integer,
                         ForeignKey(“Values.share_id"))
    value = relationship(“Values")
```

```
class Values(Base):
    __tablename__ = ‘values‘

    id = Column(Integer, primary_key=True)
    share_id = Column(Integer)
    values = Column(Float)
    type = Column(String)
```

Where is __init__?

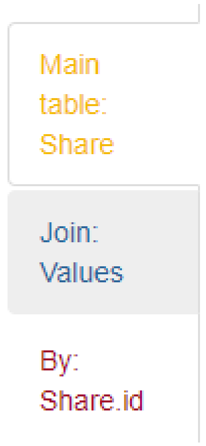
SQL Constructor

```
from sqlalchemy import Integer, …
import sqlalchemy.ext.declarative.declarative_base as d
from sqlalchemy.orm import relationship

Base = d()

class Share(Base):
    __tablename__ = ‘share‘

    id = Column(Integer, primary_key=True)
    share_value = Column(Integer,
                         ForeignKey(“Values.share_id"))
    value = relationship(“Values")
```

```
class Values(Base):
    __tablename__ = ‘values‘

    id = Column(Integer, primary_key=True)
    share_id = Column(Integer)
    values = Column(Float)
    type = Column(String)
```

```
def __init__(self, vars):
    #set from namedtuple
    for field in vars._fields:
        setattr(self,
                field,
                getattr(vars, field))
```

SQL Constructor


Main table: Share

Join: Values

By: Share.id

B

```
#make query
query = (session.query(Share, func.max(values.values)).
                outerjoin(Values, Values.share_id==Share.id).
                group_by(Share.id)
        )
```

Let's make creative!

C

```
#make query
sql = text("""SELECT share.id, max(values.values)
              FROM share
              LEFT JOIN values on values.share_id = share.id""")
result = engine.execute(sql)
```

A little bit ETL

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
```

A little bit ETL

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
```

Summary

- Data visualization is a good way thanks to Python classes!
- Make your own dashboard! (because of pitfalls)
- Actually, We can build a chain: Python + DataBase + ETL …etc (add what you want)
- If it's the chain we are able to use Docker

  => Install Portainer (https://www.portainer.io/) and manage it!

Thank you!

fb: /seleznev.artem.info

telegram: @SeleznevArtem

If you are going to a hackathon

and

need teammates

Invite Me

PyCon Belarus