jsonstat.py Documentation

Release 0.1.14

26fe

Aug 06, 2017

Contents

jsonstat.py is a library for reading the JSON-stat data format maintained and promoted by Xavier Badosa. The JSON-stat format is a JSON format for publishing dataset. JSON-stat is used by several institutions to publish statistical data.

Contents:

CHAPTER 1

Notebooks

Notebook: using jsonstat.py python library with jsonstat format version 1.

This Jupyter notebook shows the python library jsonstat.py in action. The JSON-stat is a simple lightweight JSON dissemination format. For more information about the format see the official site. This example shows how to explore the example data file oecd-canada from json-stat.org site. This file is compliant to the version 1 of jsonstat.

```
# all import here
from __future__ import print_function
import os
import pandas as ps # using panda to convert jsonstat dataset to pandas dataframe
import jsonstat     # import jsonstat.py package

import matplotlib as plt  # for plotting
```

```
%matplotlib inline
```

Download or use cached file oecd-canada.json. Caching file on disk permits to work off-line and to speed up the exploration of the data.

```
url = 'http://json-stat.org/samples/oecd-canada.json'
file_name = "oecd-canada.json"

file_path = os.path.abspath(os.path.join("..", "tests", "fixtures", "www.json-stat.org
↪", file_name))
if os.path.exists(file_path):
    print("using already downloaded file {}".format(file_path))
else:
    print("download file and storing on disk")
    jsonstat.download(url, file_name)
    file_path = file_name
```

```
using already downloaded file /Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/
↪tests/fixtures/www.json-stat.org/oecd-canada.json
```

Initialize JsonStatCollection from the file and print the list of dataset contained into the collection.

```
collection = jsonstat.from_file(file_path)
collection
```

Select the dataset named `oedc`. Oecd dataset has three dimensions (concept, area, year), and contains 432 values.

```
oecd = collection.dataset('oecd')
oecd
```

Shows some detailed info about dimensions

```
oecd.dimension('concept')
```

```
oecd.dimension('area')
```

```
oecd.dimension('year')
```

Accessing value in the dataset

Print the value in oecd dataset for area = IT and year = 2012

```
oecd.data(area='IT', year='2012')
```

```
JsonStatValue(idx=201, value=10.55546863, status=None)
```

```
oecd.value(area='IT', year='2012')
```

```
10.55546863
```

```
oecd.value(concept='unemployment rate',area='Australia',year='2004') # 5.39663128
```

```
5.39663128
```

```
oecd.value(concept='UNR',area='AU',year='2004')
```

```
5.39663128
```

Trasforming dataset into pandas DataFrame

```
df_oecd = oecd.to_data_frame('year', content='id')
df_oecd.head()
```

```
df_oecd['area'].describe() # area contains 36 values
```

```
count      432
unique      36
top         JP
freq        12
Name: area, dtype: object
```

Extract a subset of data in a pandas dataframe from the jsonstat dataset. We can trasform dataset freezing the dimension area to a specific country (Canada)

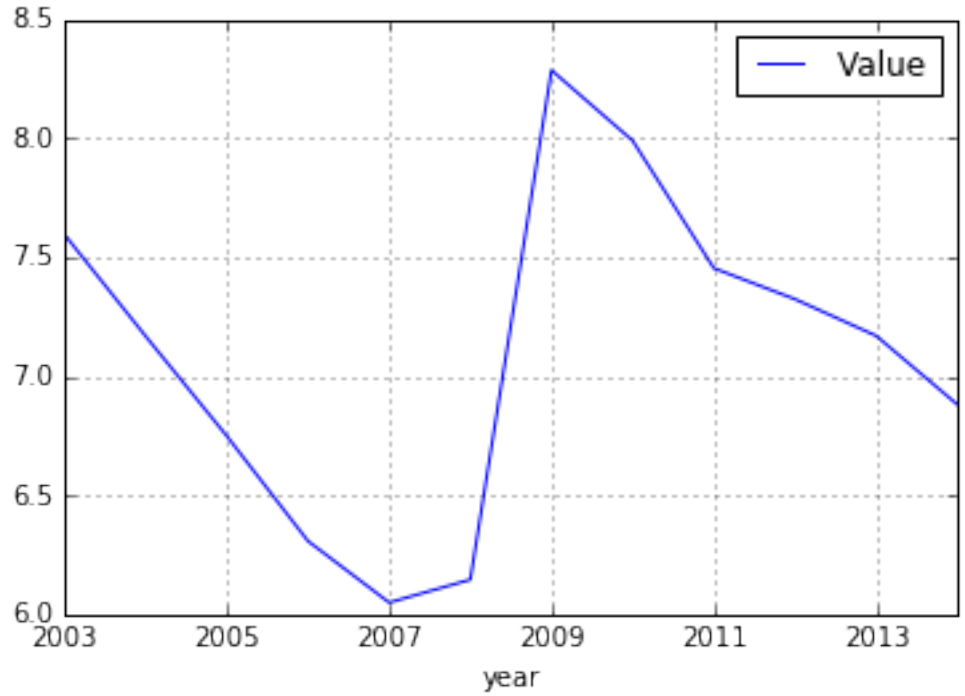
```
df_oecd_ca = oecd.to_data_frame('year', content='id', blocked_dims={'area':'CA'})
df_oecd_ca.tail()
```

```
df_oecd_ca['area'].describe()  # area contains only one value (CA)
```

```
count      12
unique      1
top        CA
freq       12
Name: area, dtype: object
```

```
df_oecd_ca.plot(grid=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x113980908>
```



Trasforming a dataset into a python list

```
oecd.to_table()[:5]
```

```
[['indicator', 'OECD countries, EU15 and total', '2003-2014', 'Value'],
 ['unemployment rate', 'Australia', '2003', 5.943826289],
 ['unemployment rate', 'Australia', '2004', 5.39663128],
 ['unemployment rate', 'Australia', '2005', 5.044790587],
 ['unemployment rate', 'Australia', '2006', 4.789362794]]
```

It is possible to trasform jsonstat data into table in different order

```
order = [i.did() for i in oecd.dimensions()]
order = order[::-1]  # reverse list
table = oecd.to_table(order=order)
table[:5]
```

```
[['indicator', 'OECD countries, EU15 and total', '2003-2014', 'Value'],
 ['unemployment rate', 'Australia', '2003', 5.943826289],
 ['unemployment rate', 'Austria', '2003', 4.278559338],
 ['unemployment rate', 'Belgium', '2003', 8.158333333],
 ['unemployment rate', 'Canada', '2003', 7.594616751]]
```

Notebook: using jsonstat.py python library with jsonstat format version 2.

This Jupyter notebook shows the python library jsonstat.py in action. The JSON-stat is a simple lightweight JSON dissemination format. For more information about the format see the official site.

In this notebook it is used the data file oecd-canada-col.json from json-stat.org site. This file is compliant to the version 2 of jsonstat. This notebook is equal to version 1. The only difference is the datasource.

```
# all import here
from __future__ import print_function
import os
import pandas as ps # using panda to convert jsonstat dataset to pandas dataframe
import jsonstat     # import jsonstat.py package

import matplotlib as plt  # for plotting
%matplotlib inline
```

Download or use cached file oecd-canada-col.json. Caching file on disk permits to work off-line and to speed up the exploration of the data.

```
url = 'http://json-stat.org/samples/oecd-canada-col.json'
file_name = "oecd-canada-col.json"

file_path = os.path.abspath(os.path.join("..", "tests", "fixtures", "www.json-stat.org
↪", file_name))
if os.path.exists(file_path):
    print("using already downloaded file {}".format(file_path))
else:
    print("download file and storing on disk")
    jsonstat.download(url, file_name)
    file_path = file_name
```

```
using already downloaded file /Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/
↪tests/fixtures/www.json-stat.org/oecd-canada-col.json
```

Initialize JsonStatCollection from the file and print the list of dataset contained into the collection.

```
collection = jsonstat.from_file(file_path)
collection
```

Select the firt dataset. Oecd dataset has three dimensions (concept, area, year), and contains 432 values.

```
oecd = collection.dataset(0)
oecd
```

```
oecd.dimension('concept')
```

```
oecd.dimension('area')
```

```
oecd.dimension('year')
```

Shows some detailed info about dimensions.

Accessing value in the dataset

Print the value in oecd dataset for area = IT and year = 2012

```
oecd.data(area='IT', year='2012')
```

```
JsonStatValue(idx=201, value=10.55546863, status=None)
```

```
oecd.value(area='IT', year='2012')
```

```
10.55546863
```

```
oecd.value(concept='unemployment rate',area='Australia',year='2004') # 5.39663128
```

```
5.39663128
```

```
oecd.value(concept='UNR',area='AU',year='2004')
```

```
5.39663128
```

Trasforming dataset into pandas DataFrame

```
df_oecd = oecd.to_data_frame('year', content='id')
df_oecd.head()
```

```
df_oecd['area'].describe() # area contains 36 values
```

```
count     432
unique     36
top        ES
freq       12
Name: area, dtype: object
```

Extract a subset of data in a pandas dataframe from the jsonstat dataset. We can trasform dataset freezing the dimension area to a specific country (Canada)

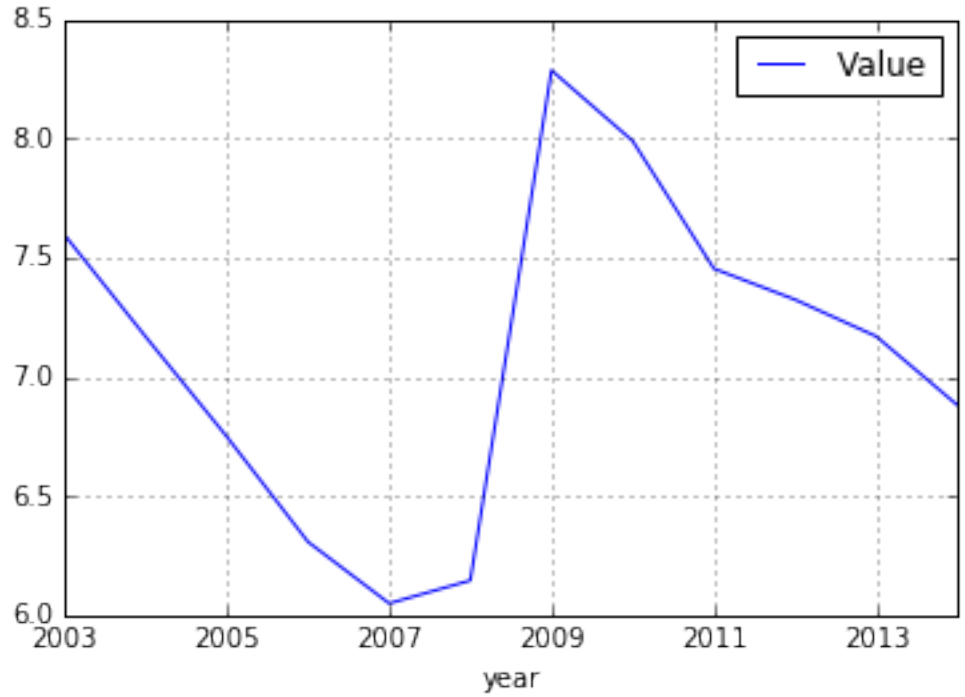
```
df_oecd_ca = oecd.to_data_frame('year', content='id', blocked_dims={'area':'CA'})
df_oecd_ca.tail()
```

```
df_oecd_ca['area'].describe()  # area contains only one value (CA)
```

```
count      12
unique      1
top        CA
freq       12
Name: area, dtype: object
```

```
df_oecd_ca.plot(grid=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x114298198>
```



Trasforming a dataset into a python list

```
oecd.to_table()[:5]
```

```
[['indicator', 'OECD countries, EU15 and total', '2003-2014', 'Value'],
 ['unemployment rate', 'Australia', '2003', 5.943826289],
 ['unemployment rate', 'Australia', '2004', 5.39663128],
 ['unemployment rate', 'Australia', '2005', 5.044790587],
 ['unemployment rate', 'Australia', '2006', 4.789362794]]
```

It is possible to trasform jsonstat data into table in different order

```
order = [i.did() for i in oecd.dimensions()]
order = order[::-1]  # reverse list
table = oecd.to_table(order=order)
table[:5]
```

```
[['indicator', 'OECD countries, EU15 and total', '2003-2014', 'Value'],
 ['unemployment rate', 'Australia', '2003', 5.943826289],
 ['unemployment rate', 'Austria', '2003', 4.278559338],
 ['unemployment rate', 'Belgium', '2003', 8.158333333],
 ['unemployment rate', 'Canada', '2003', 7.594616751]]
```

Notebook: using jsonstat.py with eurostat api

This Jupyter notebook shows the python library jsonstat.py in action. It shows how to explore dataset downloaded from a data provider. This notebook uses some datasets from Eurostat. Eurostat provides a rest api to download its datasets. You can find details about the api here It is possible to use a query builder for discovering the rest api parameters. The following image shows the query builder:

```
# all import here
from __future__ import print_function
import os
import pandas as pd
import jsonstat

import matplotlib as plt
%matplotlib inline
```

1 - Exploring data with one dimension (time) with size > 1

Following cell downloads a datataset from eurostat. If the file is already downloaded use the copy presents on the disk. Caching file is useful to avoid downloading dataset every time notebook runs. Caching can speed the development, and provides consistent results. You can see the raw data here

```
url_1 = 'http://ec.europa.eu/eurostat/wdds/rest/data/v1.1/json/en/nama_gdp_c?
↪precision=1&geo=IT&unit=EUR_HAB&indic_na=B1GM'
file_name_1 = "eurostat-name_gpd_c-geo_IT.json"

file_path_1 = os.path.abspath(os.path.join("..", "tests", "fixtures", "www.ec.europa.
↪eu_eurostat", file_name_1))
if os.path.exists(file_path_1):
    print("using already donwloaded file {}".format(file_path_1))
else:
    print("download file")
    jsonstat.download(url_1, file_name_1)
    file_path_1 = file_name_1
```

```
using already donwloaded file /Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/
↪tests/fixtures/www.ec.europa.eu_eurostat/eurostat-name_gpd_c-geo_IT.json
```

Initialize JsonStatCollection with eurostat data and print some info about the collection.

```
collection_1 = jsonstat.from_file(file_path_1)
collection_1
```

Previous collection contains only a dataset named ‘nama_gdp_c‘

```
nama_gdp_c_1 = collection_1.dataset('nama_gdp_c')
nama_gdp_c_1
```

All dimensions of the dataset ‘nama_gdp_c‘ are of size 1 with exception of time dimension. Let’s explore the time dimension.

```
nama_gdp_c_1.dimension('time')
```

Get value for year 2012.

```
nama_gdp_c_1.value(time='2012')
```

```
25700
```

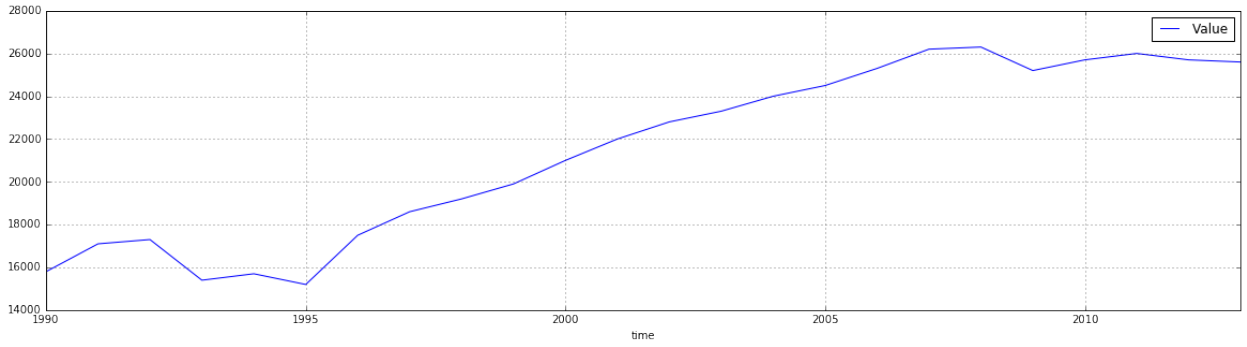
Convert the jsonstat data into a pandas dataframe.

```
df_1 = nama_gdp_c_1.to_data_frame('time', content='id')
df_1.tail()
```

Adding a simple plot

```
df_1 = df_1.dropna() # remove rows with NaN values
df_1.plot(grid=True, figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x114bc12b0>
```



2 - Exploring data with two dimensions (geo, time) with size > 1

Download or use the jsonstat file cached on disk. The cache is used to avoid internet download during the devolopment to make the things a bit faster. You can see the raw data here

```
url_2 = 'http://ec.europa.eu/eurostat/wdds/rest/data/v1.1/json/en/nama_gdp_c?
↪precision=1&geo=IT&geo=FR&unit=EUR_HAB&indic_na=B1GM'
file_name_2 = "eurostat-name_gpd_c-geo_IT_FR.json"

file_path_2 = os.path.abspath(os.path.join("..", "tests", "fixtures", "www.ec.europa.
↪eu_eurostat", file_name_2))
if os.path.exists(file_path_2):
```

```
    print("using alredy donwloaded file {}".format(file_path_2))
else:
    print("download file and storing on disk")
    jsonstat.download(url, file_name_2)
    file_path_2 = file_name_2
```

```
using alredy donwloaded file /Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/
↪tests/fixtures/www.ec.europa.eu_eurostat/eurostat-name_gpd_c-geo_IT_FR.json
```

```
collection_2 = jsonstat.from_file(file_path_2)
nama_gdp_c_2 = collection_2.dataset('nama_gdp_c')
nama_gdp_c_2
```

```
nama_gdp_c_2.dimension('geo')
```

```
nama_gdp_c_2.value(time='2012',geo='IT')
```

```
25700
```

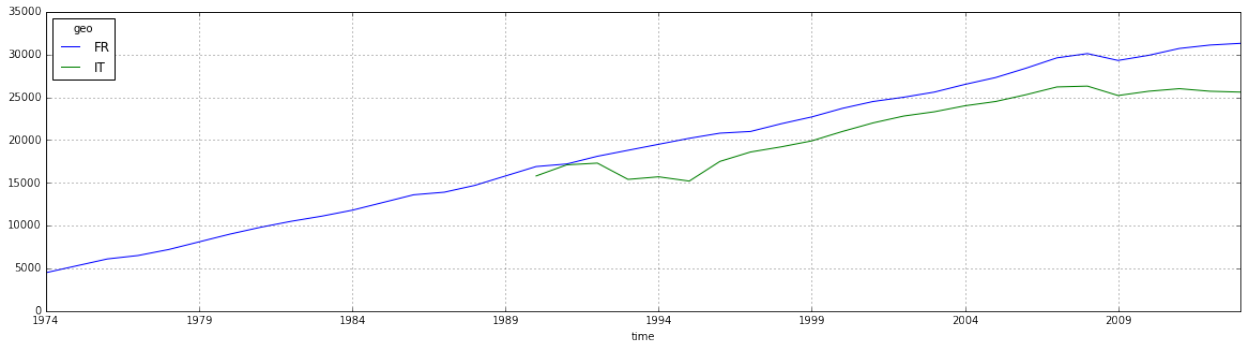
```
nama_gdp_c_2.value(time='2012',geo='FR')
```

```
31100
```

```
df_2 = nama_gdp_c_2.to_table(content='id',rtype=pd.DataFrame)
df_2.tail()
```

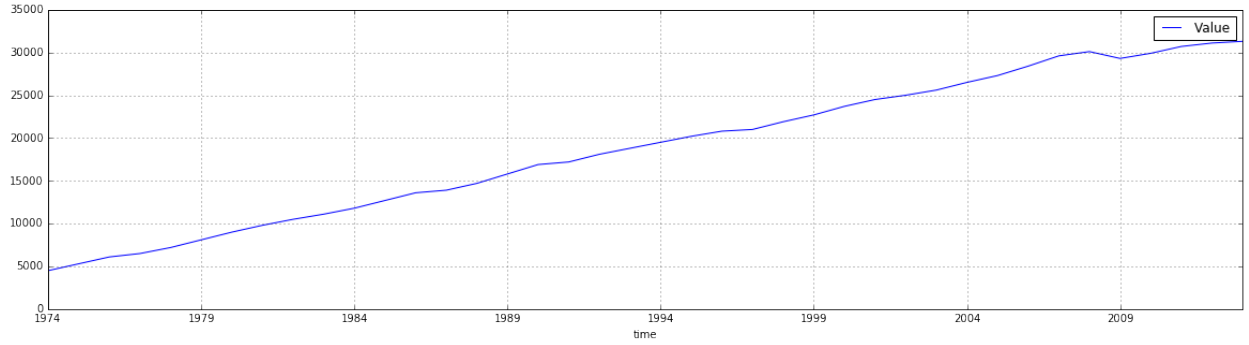
```
df_FR_IT = df_2.dropna()[['time', 'geo', 'Value']]
df_FR_IT = df_FR_IT.pivot('time', 'geo', 'Value')
df_FR_IT.plot(grid=True, figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x114c0f0b8>
```



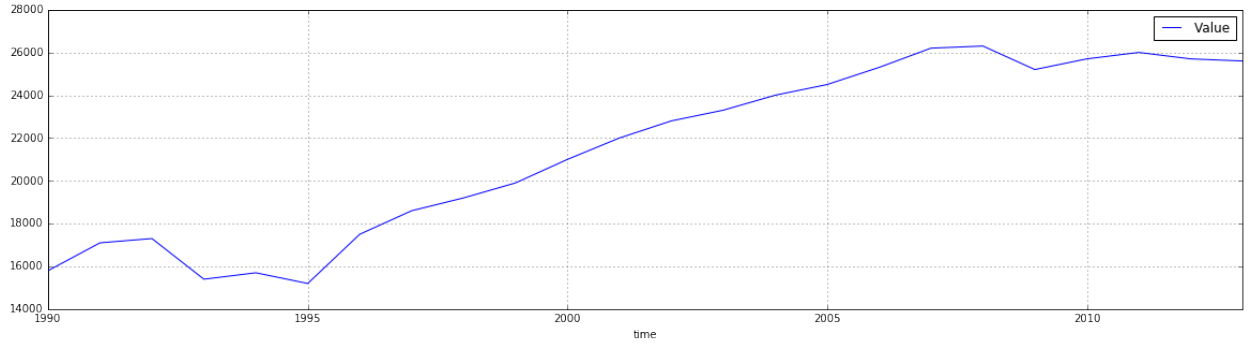

```
df_3 = nama_gdp_c_2.to_data_frame('time', content='id', blocked_dims={'geo':'FR'})
df_3 = df_3.dropna()
df_3.plot(grid=True,figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1178e7d30>
```




```
df_4 = nama_gdp_c_2.to_data_frame('time', content='id', blocked_dims={'geo':'IT'})
df_4 = df_4.dropna()
df_4.plot(grid=True,figsize=(20,5))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x117947630>
```



Notebook: using jsonstat.py to explore ISTAT data (house price index)

This Jupyter notebook shows how to use jsonstat.py python library to explore Istat data. Istat is Italian National Institute of Statistics. It publishs a rest api for querying italian statistics.

We starts importing some modules.

```
from __future__ import print_function
import os
import istat
from IPython.core.display import HTML
```

Step 1: using istat module to get a jsonstat collection

Following code sets a cache dir where to store json files download by Istat api. Storing file on disk speed up development, and assures consistent results over time. Anyway you can delete file to donwload a fresh copy.

```
cache_dir = os.path.abspath(os.path.join("..", "tmp", "istat_cached"))
istat.cache_dir(cache_dir)
print("cache_dir is '{}'".format(istat.cache_dir()))
```

```
cache_dir is '/Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/tmp/istat_cached'
```

Using istat api, we can shows the istat areas used to categorize the datasets

```
istat.areas()
```

Following code list all datasets contained into area `Prices`.

```
istat_area_prices = istat.area('Prices')
istat_area_prices.datasets()
```

List all dimension for dataset `DCSP_IPAB` (House price index)

```
istat_dataset_dcsp_ipab = istat_area_prices.dataset('DCSP_IPAB')
istat_dataset_dcsp_ipab
```

Finally from istat dataset we extracts data in jsonstat format by specifying dimensions we are interested.

```
spec = {
    "Territory": 1, "Index type": 18,
    # "Measure": 0, # "Purchases of dwelling": 0, # "Time and frequency": 0
}
# convert istat dataset into jsonstat collection and print some info
collection = istat_dataset_dcsp_ipab.getvalues(spec)
collection
```

The previous call is equivalent to call istat api with a "1,18,0,0,0" string of number. Below is the mapping from the number and dimensions:

dimension		
Territory	1	Italy
Type	18	house price index (base 2010=100) - quarterly data'
Measure	0	ALL
Purchase of dwelling	0	ALL
Time and frequency	0	ALL

```
json_stat_data = istat_dataset_dcsp_ipab.getvalues("1,18,0,0,0")
json_stat_data
```

step2: using jsonstat.py api.

Now we have a jsonstat collection, let expore it with the api of jsonstat.py

Print some info of one dataset contained into the above jsonstat collection

```
jsonstat_dataset = collection.dataset('IDMISURA1*IDTYPPURCH*IDTIME')
jsonstat_dataset
```

Print info about the dimensions to get an idea about the data

```
jsonstat_dataset.dimension('IDMISURA1')
```

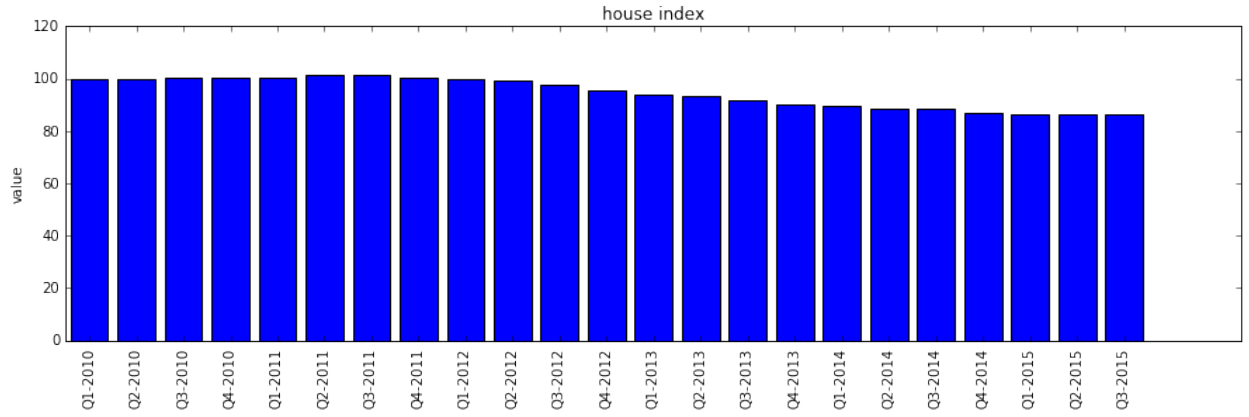
```
jsonstat_dataset.dimension('IDTYPPURCH')
```

```
jsonstat_dataset.dimension('IDTIME')
```

```
import pandas as pd
df = jsonstat_dataset.to_table(rtype=pd.DataFrame)
df.head()
```

```
filtered = df.loc[
    (df['Measure'] == 'index number') & (df['Purchases of dwellings'] == 'H1 - all␣
↪items'),
    ['Time and frequency', 'Value']
]
filtered.set_index('Time and frequency')
```

```
%matplotlib inline
import matplotlib.pyplot as plt

values = filtered['Value'].tolist()
labels = filtered['Time and frequency']

xs = [i + 0.1 for i, _ in enumerate(values)]
# bars are by default width 0.8, so we'll add 0.1 to the left coordinates
# so that each bar is centered

# plot bars with left x-coordinates [xs], heights [num_oscars]
plt.figure(figsize=(15,4))
plt.bar(xs, values)
plt.ylabel("value")
plt.title("house index")

# label x-axis with movie names at bar centers
plt.xticks([i + 0.5 for i, _ in enumerate(labels)], labels, rotation='vertical')
plt.show()
```



Notebook: using jsonstat.py to explore ISTAT data (unemployment)

This Jupyter notebook shows how to use jsonstat.py python library to explore Istat data. Istat is the Italian National Institute of Statistics. It publishes a rest api for browsing italian statistics. This api can return results in jsonstat format.

```
from __future__ import print_function
import os
import pandas as pd
from IPython.core.display import HTML
import matplotlib.pyplot as plt
%matplotlib inline

import istat
```

Using istat api

Next step is to set a cache dir where to store json files downloaded from Istat. Storing file on disk speeds up development, and assures consistent results over time. Eventually, you can delete donwloaded files to get a fresh copy.

```
cache_dir = os.path.abspath(os.path.join("..", "tmp", "istat_cached")) # you could␣
↪choice /tmp
istat.cache_dir(cache_dir)
print("cache_dir is '{}'".format(istat.cache_dir()))
```

```
cache_dir is '/Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/tmp/istat_cached'
```

List all istat areas

```
istat.areas()
```

List all datasets contained into area `LAB` (Labour)

```
istat_area_lab = istat.area('LAB')
istat_area_lab
```

List all dimension for dataset `DCCV_TAXDISOCCU` (Unemployment rate)

```
istat_dataset_taxdisoccu = istat_area_lab.dataset('DCCV_TAXDISOCCU')
istat_dataset_taxdisoccu
```

Extract data from dataset `DCCV_TAXDISOCCU`

```
spec = {
    "Territory": 0,                          # 1 Italy
    "Data type": 6,                          # (6:'unemployment rate')
    'Measure': 1,                            # 1 : 'percentage values'
    'Gender': 3,                             # 3 total
    'Age class':31,                          # 31:'15-74 years'
    'Highest level of education attained': 12, # 12:'total',
    'Citizenship': 3,                        # 3:'total')
    'Duration of unemployment': 3,           # 3:'total'
    'Time and frequency': 0                  # All
}

# convert istat dataset into jsonstat collection and print some info
collection = istat_dataset_taxdisoccu.getvalues(spec)
collection
```

Print some info of the only dataset contained into the above jsonstat collection

```
jsonstat_dataset = collection.dataset(0)
jsonstat_dataset
```

```
df_all = jsonstat_dataset.to_table(rtype=pd.DataFrame)
df_all.head()
```

```
df_all.pivot('Territory', 'Time and frequency', 'Value').head()
```

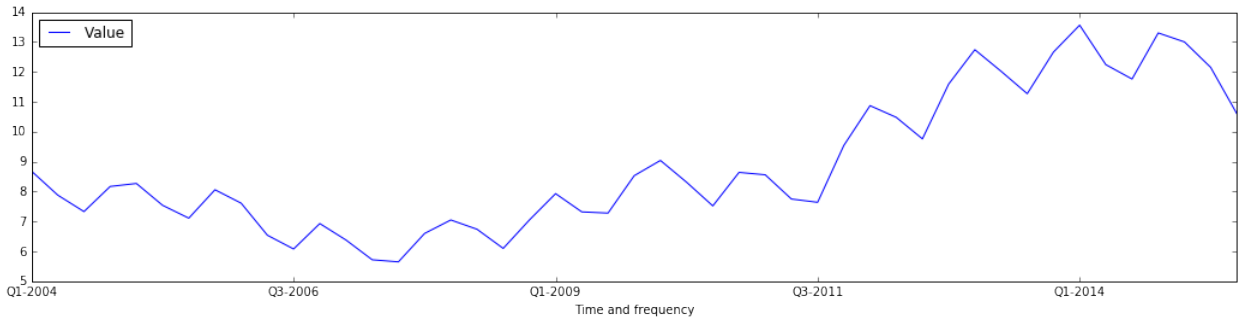
```
spec = {
    "Territory": 1,                             # 1 Italy
    "Data type": 6,                             # (6:'unemployment rate')
    'Measure': 1,
    'Gender': 3,
    'Age class':0,                              # all classes
    'Highest level of education attained': 12, # 12:'total',
    'Citizenship': 3,                           # 3:'total')
    'Duration of unemployment': 3,              #  3:'total')
    'Time and frequency': 0                     # All
}

# convert istat dataset into jsonstat collection and print some info
collection_2 = istat_dataset_taxdisoccu.getvalues(spec)
collection_2
```

```
df = collection_2.dataset(0).to_table(rtype=pd.DataFrame, blocked_dims={'IDCLASETA28':
↪'31'})
df.head(6)
```

```
df = df.dropna()
df = df[df['Time and frequency'].str.contains(r'^Q.*')]
# df = df.set_index('Time and frequency')
df.head(6)
```

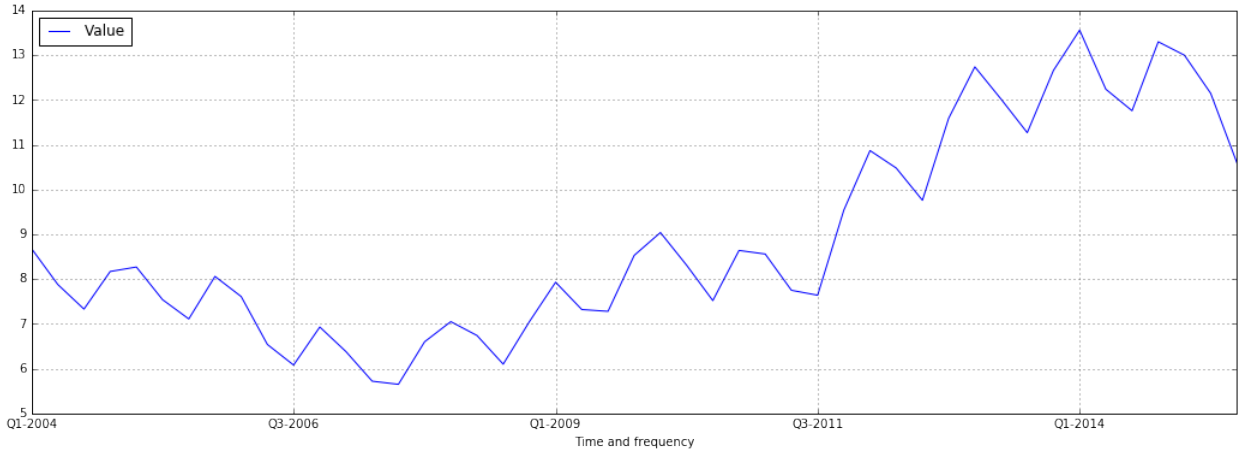
```
df.plot(x='Time and frequency',y='Value', figsize=(18,4))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1184b1908>
```




```
fig = plt.figure(figsize=(18,6))
ax = fig.add_subplot(111)
plt.grid(True)
df.plot(x='Time and frequency',y='Value', ax=ax, grid=True)
# kind='barh', , alpha=a, legend=False, color=customcmap,
# edgecolor='w', xlim=(0,max(df['population'])), title=ttl)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a898b70>
```




```
# plt.figure(figsize=(7,4))
# plt.plot(df['Time and frequency'],df['Value'], lw=1.5, label='1st')
# plt.plot(y[:,1], lw=1.5, label='2st')
# plt.plot(y,'ro')
# plt.grid(True)
# plt.legend(loc=0)
# plt.axis('tight')
# plt.xlabel('index')
# plt.ylabel('value')
# plt.title('a simple plot')
```

```
# forza lavoro
istat_forzlv = istat.dataset('LAB', 'DCCV_FORZLV')

spec = {
    "Territory": 'Italy',
    "Data type": 'number of labour force 15 years and more (thousands)',
↪               #
    'Measure':   'absolute values',
    'Gender':    'total',
    'Age class': '15 years and over',
    'Highest level of education attained': 'total',
    'Citizenship': 'total',
    'Time and frequency': 0
}

df_forzlv = istat_forzlv.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
df_forzlv = df_forzlv.dropna()
df_forzlv = df_forzlv[df_forzlv['Time and frequency'].str.contains(r'^Q.*')]
df_forzlv.tail(6)
```

```
istat_inattiv = istat.dataset('LAB', 'DCCV_INATTIV')
# HTML(istat_inattiv.info_dimensions_as_html())
```

```
spec = {
    "Territory": 'Italy',
    "Data type": 'number of inactive persons',
    'Measure':   'absolute values',
```

```
    'Gender':    'total',
    'Age class': '15 years and over',
    'Highest level of education attained': 'total',
    'Time and frequency': 0
}

df_inattiv = istat_inattiv.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
df_inattiv = df_inattiv.dropna()
df_inattiv = df_inattiv[df_inattiv['Time and frequency'].str.contains(r'^Q.*')]
df_inattiv.tail(6)
```

Notebook: using jsonstat.py to explore ISTAT data (unemployment)

This Jupyter notebook shows how to use jsonstat.py python library to explore Istat data. Istat is the Italian National Institute of Statistics. It publishs a rest api for browsing italian statistics. This api can return results in jsonstat format.

La forza lavoro e composta da occupati e disoccupati. La popolozione sopra i 15 anni e composta da forza lavoro ed inatttivi.

$$Popolozoine = ForzaLavoro + Inattivi$$

$$Forzalav = Occupati + Disoccupati$$

$$Inattivi = NonVoglioLavorare + Scoraggiati$$

Tasso disoccupazione = Disoccupati/Occupati

download dataset from Istat

```
from __future__ import print_function
import os
import pandas as pd
from IPython.core.display import HTML
import matplotlib.pyplot as plt
%matplotlib inline

import istat

# Next step is to set a cache dir where to store json files downloaded from Istat.
# Storing file on disk speeds up development, and assures consistent results over␣
↪time.
# Eventually, you can delete donwloaded files to get a fresh copy.

cache_dir = os.path.abspath(os.path.join("..", "tmp", "istat_cached"))
istat.cache_dir(cache_dir)
istat.lang(0)  # set italian language
print("cache_dir is '{}'".format(istat.cache_dir()))
```

```
cache_dir is '/Users/26fe_nas/gioprj.on_mac/prj.python/jsonstat.py/tmp/istat_cached'
```

```
# List all datasets contained into area `LAB` (Labour)
istat.area('LAB').datasets()
```

Download - numero occupati - numero disoccupati - forza lavoro - controllare che nroccupati + nrdisoccupati = forza lavoro

Download Occupati

```
# DCCV_OCCUPATIT
istat_occupatit = istat.dataset('LAB', 'DCCV_OCCUPATIT')
# HTML(istat_occupatit.info_dimensions_as_html(show_values=0))
```

```
spec = {
    'Territorio':               'Italia',
    'Sesso':                    'totale',
    'Classe di età':            '15 anni e più',
    'Titolo di studio':         'totale',
    'Cittadinanza':             'totale',
    'Ateco 2002' :              '0010 totale',
    'Ateco 2007' :              '0010 totale',
    'Posizione professionale': 'totale',
    'Profilo professionale':    'totale',
    'Professione 2001':         'totale',
    'Professione 2011':         'totale',
    'Regime orario':            'totale',
    'Carattere occupazione':    'totale',
    'Tempo e frequenza': 0
}

df_occupatit = istat_occupatit.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
df_occupatit[df_occupatit['Tempo e frequenza'].str.contains(r'^T.*')]
df_occupatit.tail(6)
```

```
df_occupatit.ix[192]
```

```
Tempo e frequenza    T3-2015
Value                22645.1
Name: 192, dtype: object
```

Download disoccupati

```
istat_disoccupt = istat.dataset('LAB', 'DCCV_DISOCCUPT')
istat_disoccupt
```

```
spec = {
    'Territorio':               'Italia',
    'Tipo dato' :               'numero di persone in cerca di occupazione 15 anni e␣
↪oltre (valori in migliaia)',
    'Misura':                   'valori assoluti',
    'Sesso':                    'totale',
    'Classe di età':            '15 anni e più',
    'Titolo di studio':         'totale',
    'Cittadinanza':             'totale',
    'Condizione professionale': 'totale',
    'Durata disoccupazione':    'totale',
    'Tempo e frequenza': 0
}

df_disoccupt = istat_disoccupt.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
df_disoccupt[df_disoccupt['Tempo e frequenza'].str.contains(r'^T.*')]
df_disoccupt.tail(6)
```

Download forza Lavoro

```
istat_forzlv = istat.dataset('LAB', 'DCCV_FORZLV')
istat_forzlv
```

```
spec = {
    'Territorio':       'Italia',
    'Tipo dato':        'numero di forze di lavoro15 anni e oltre (valori in migliaia)
↪',
    'Misura':           'valori assoluti',
    'Sesso':            'totale',
    'Classe di età':    '15 anni e più',
    'Titolo di studio': 'totale',
    'Cittadinanza':     'totale',
    'Tempo e frequenza': 0
}

df_forzlv = istat_forzlv.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
# df_forzlv
```

```
# df_forzlv = df_forzlv.dropna()
df_forzlv = df_forzlv[df_forzlv['Tempo e frequenza'].str.contains(r'^T.*')]
df_forzlv.tail(6)
```

Download inattivi

```
istat_inattiv = istat.dataset('LAB', 'DCCV_INATTIV')
istat.options.display.max_rows = 0
# HTML(istat_inattiv.info_dimensions_as_html(show_values=0))
istat_inattiv
```

```
spec = {
    'Territorio':        'Italia',
    'Tipo dato':         'numero di inattivi (valori in migliaia)',
    'Misura':            'valori assoluti',
    'Sesso':             'totale',
    'Classe di età':     '15 anni e più',
    'Titolo di studio':  'totale',
    'Cittadinanza' : 'totale',
    'Condizione professionale': 'totale',
    'Motivo inattività': 'totale',
    'Condizione dichiarata': 'totale',
    'Tempo e frequenza': 0
}

df_inattiv = istat_inattiv.getvalues(spec).dataset(0).to_table(rtype=pd.DataFrame)
# df_inattiv
```

```
df_inattiv = df_inattiv[df_inattiv['Tempo e frequenza'].str.contains(r'^T.*')]
df_inattiv.tail(6)
```

CHAPTER 2

Tutorial

The parrot module is a module about parrots.

Doctest example:

```
>>> 2 + 2
4
```

Test-Output example:

```
json_string = '''
{
    "label" : "concepts",
    "category" : {
        "index" : {
            "POP" : 0,
            "PERCENT" : 1
        },
        "label" : {
            "POP" : "population",
            "PERCENT" : "weight of age group in the population"
        },
        "unit" : {
            "POP" : {
                "label": "thousands of persons",
                "decimals": 1,
                "type" : "count",
                "base" : "people",
                "multiplier" : 3
            },
            "PERCENT" : {
                "label" : "%",
                "decimals": 1,
                "type" : "ratio",
                "base" : "per cent",
                "multiplier" : 0
```

```
                }
            }
        }
}
'''
print(2 + 2)
```

This would output:

```
4
```

CHAPTER 3

Api Reference

Jsonstat Module

jsonstat module contains classes and utility functions to parse jsonstat data format.

Utility functions

`jsonstat.`**`from_file`**(*filename*)

read a file containing a jsonstat format and return the appropriate object

Parameters **`filename`** – file containing a jsonstat

Returns a JsonStatCollection, JsonStatDataset or JsonStatDimension object

example

```
>>> import os, jsonstat
>>> filename = os.path.join(jsonstat._examples_dir, "www.json-stat.org", "oecd-
↪canada-col.json")
>>> o = jsonstat.from_file(filename)
>>> type(o)
<class 'jsonstat.collection.JsonStatCollection'>
```

`jsonstat.`**`from_url`**(*url*, *pathname=None*)

download an url and return the downloaded content.

see *`jsonstat.download()`* for how to use pathname parameter.

Parameters

- **`url`** – ex.: http://json-stat.org/samples/oecd-canada.json
- **`pathname`** – If `pathname` is defined the contents of the url

will be stored into the file `<cache_dir>/pathname` If `pathname` is None the filename will be automatic generated. If `pathname` is an absolute path cache_dir will be ignored.

Returns the contents of url

To set dir where to store downloaded file see *jsonstat.cache_dir()*. Cache expiration policy can be customized

example:

```
>>> import jsonstat
>>> # cache_dir = os.path.normpath(os.path.join(jsonstat.__fixtures_dir, "json-
↪stat.org"))
>>> # download external content into the /tmp dir so next downloads can be faster
>>> uri = 'http://json-stat.org/samples/oecd-canada.json'
>>> jsonstat.cache_dir("/tmp")
'/tmp'
>>> o = jsonstat.from_url(uri)
>>> print(o)
JsonstatCollection contains the following JsonStatDataSet:
+-----+----------+
| pos | dataset  |
+-----+----------+
| 0   | 'oecd'   |
| 1   | 'canada' |
+-----+----------+
```

jsonstat.**from_json**(*json_data*)

transform a json structure into jsonstat objects hierarchy

Parameters **json_data** – data structure (dictionary) representing a json

Returns a JsonStatCollection, JsonStatDataset or JsonStatDimension object

```
>>> import json, jsonstat
>>> from collections import OrderedDict
>>> json_string_v1 = '''{
...                       "oecd" : {
...                         "value": [1],
...                         "dimension" : {
...                           "id": ["one"],
...                           "size": [1],
...                           "one": { "category": { "index":{"2010":0 } } }
...                         }
...                       }
...                     }'''
>>> json_data = json.loads(json_string_v1, object_pairs_hook=OrderedDict)
>>> jsonstat.from_json(json_data)
JsonstatCollection contains the following JsonStatDataSet:
+-----+---------+
| pos | dataset |
+-----+---------+
| 0   | 'oecd'  |
+-----+---------+
```

jsonstat.**from_string**(*json_string*)

parse a jsonstat string and return the appropriate object

Parameters **json_string** – string containing a json

Returns a JsonStatCollection, JsonStatDataset or JsonStatDimension object

jsonstat.**cache_dir**(*cached_dir=u''*, *time_to_live=None*)

Manage the directory `cached_dir` where downloaded files are stored

without parameter return the `cached_dir` directory with a parameters set the directory

Parameters

- **cached_dir** –
- **time_to_live** –

`jsonstat.`**`download`**(*url*, *pathname=None*)
download a url and return the downloaded content‘‘

Parameters

- **url** – ex.: http://json-stat.org/samples/oecd-canada.json
- **pathname** – If `pathname` is defined the contents of the url

will be stored into the file `<cache_dir>/pathname` If `pathname` is None the filename will be automatic generated. If `pathname` is an absolute path cache_dir will be ignored.

Returns the contents of url

To set dir where to store downloaded file see *`jsonstat.cache_dir()`*. Cache expiration policy can be customized

JsonStatCollection

class `jsonstat.`**`JsonStatCollection`**
Represents a jsonstat collection.

It contains one or more datasets.

```
>>> import os, jsonstat
>>> filename = os.path.join(jsonstat._examples_dir, "www.json-stat.org", "oecd-
↪canada-col.json")
>>> collection = jsonstat.from_file(filename)
>>> len(collection)
2
>>> collection
JsonstatCollection contains the following JsonStatDataSet:
+-----+-------------------------------------------------------+
| pos | dataset                                               |
+-----+-------------------------------------------------------+
| 0   | 'Unemployment rate in the OECD countries 2003-2014' |
| 1   | 'Population by sex and age group. Canada. 2012'     |
+-----+-------------------------------------------------------+
```

`__len__`()
the number of dataset contained in this collection

`dataset`(*spec*)
select a dataset belonging to the collection

Parameters **spec** – can be

- the name of collection (string) for jsonstat v1
- an integer (for jsonstat v1 and v2)

Returns a JsonStatDataSet object

parsing

JsonStatCollection.**from_file**()
initialize this collection from a file It is better to use *jsonstat.from_file()*

Parameters **filename** – name containing a jsonstat

Returns itself to chain call

JsonStatCollection.**from_string**()
Initialize this collection from a string It is better to use *jsonstat.from_string()*

Parameters **json_string** – string containing a json

Returns itself to chain call

JsonStatCollection.**from_json**()
initialize this collection from a json structure It is better to use *jsonstat.from_json()*

Parameters **json_data** – data structure (dictionary) representing a json

Returns itself to chain call

JsonStatDataSet

class jsonstat.**JsonStatDataSet**(*name=None*)
Represents a JsonStat dataset

```
>>> import os, jsonstat
>>> filename = os.path.join(jsonstat._examples_dir, "www.json-stat.org", "oecd-
↪canada-col.json")
>>> dataset = jsonstat.from_file(filename).dataset(0)
>>> dataset.label
'Unemployment rate in the OECD countries 2003-2014'
>>> print(dataset)
name:   'Unemployment rate in the OECD countries 2003-2014'
label:  'Unemployment rate in the OECD countries 2003-2014'
size: 432
+-----+---------+--------------------------------+------+--------+
| pos | id      | label                          | size | role   |
+-----+---------+--------------------------------+------+--------+
| 0   | concept | indicator                      | 1    | metric |
| 1   | area    | OECD countries, EU15 and total | 36   | geo    |
| 2   | year    | 2003-2014                      | 12   | time   |
+-----+---------+--------------------------------+------+--------+
>>> dataset.dimension(1)
+-----+--------+--------------------------+
| pos | idx    | label                    |
+-----+--------+--------------------------+
| 0   | 'AU'   | 'Australia'              |
| 1   | 'AT'   | 'Austria'                |
| 2   | 'BE'   | 'Belgium'                |
| 3   | 'CA'   | 'Canada'                 |
| 4   | 'CL'   | 'Chile'                  |
| 5   | 'CZ'   | 'Czech Republic'         |
| 6   | 'DK'   | 'Denmark'                |
| 7   | 'EE'   | 'Estonia'                |
| 8   | 'FI'   | 'Finland'                |
| 9   | 'FR'   | 'France'                 |
| 10  | 'DE'   | 'Germany'                |
```

```
| 11  | 'GR'   | 'Greece'                  |
| 12  | 'HU'   | 'Hungary'                 |
| 13  | 'IS'   | 'Iceland'                 |
| 14  | 'IE'   | 'Ireland'                 |
| 15  | 'IL'   | 'Israel'                  |
| 16  | 'IT'   | 'Italy'                   |
| 17  | 'JP'   | 'Japan'                   |
| 18  | 'KR'   | 'Korea'                   |
| 19  | 'LU'   | 'Luxembourg'              |
| 20  | 'MX'   | 'Mexico'                  |
| 21  | 'NL'   | 'Netherlands'             |
| 22  | 'NZ'   | 'New Zealand'             |
| 23  | 'NO'   | 'Norway'                  |
| 24  | 'PL'   | 'Poland'                  |
| 25  | 'PT'   | 'Portugal'                |
| 26  | 'SK'   | 'Slovak Republic'         |
| 27  | 'SI'   | 'Slovenia'                |
| 28  | 'ES'   | 'Spain'                   |
| 29  | 'SE'   | 'Sweden'                  |
| 30  | 'CH'   | 'Switzerland'             |
| 31  | 'TR'   | 'Turkey'                  |
| 32  | 'UK'   | 'United Kingdom'          |
| 33  | 'US'   | 'United States'           |
| 34  | 'EU15' | 'Euro area (15 countries)' |
| 35  | 'OECD' | 'total'                   |
+-----+--------+---------------------------+
>>> dataset.data(0)
JsonStatValue(idx=0, value=5.943826289, status=None)
```

__init__(*name=None*)
Initialize an empty dataset.

Dataset could have a name (key) if we parse a jsonstat format version 1.

Parameters **name** – dataset name (for jsonstat v.1)

name()

Getter returns the name of the dataset

Type string

label()

Getter returns the label of the dataset

Type string

__len__()
returns the size of the dataset

dimensions

JsonStatDataSet.**dimension**(*spec*)
get a JsonStatDimension by spec

Parameters **spec** – spec can be: - (string) or id of the dimension - int position of dimension

Returns a JsonStatDimension

JsonStatDataSet.**dimensions**()
returns list of JsonStatDimension

JsonStatDataSet.**info_dimensions**()
print same info on dimensions on stdout

querying methods

JsonStatDataSet.**data**(**args*, ***kargs*)
Returns a JsonStatValue containings value and status about a datapoint The datapoint will be retrieved according the parameters

Parameters

- **args** –
  - data(<int>) where i is index into the
  - data(<list>) where lst = [i1,i2,i3,...]) each i indicate the dimension len(lst) == number of dimension
  - data(<dict>) where dict is {k1:v1, k2:v2, ...} dimension of size 1 can be ommitted
- **kargs** –
  - data(k1=v1,k2=v2,...) where **ki** are the id or label of dimension **vi** are the index or label of the category dimension of size 1 can be ommitted

Returns a JsonStatValue object

kargs { cat1:value1, ..., cati:valuei, ... } cati can be the id of the dimension or the label of dimension valuei can be the index or label of category ex.:{country:"AU", "year":"2014"}

```
>>> import os, jsonstat
>>> filename = os.path.join(jsonstat._examples_dir, "www.json-stat.org",
↪"oecd-canada-col.json")
>>> dataset = jsonstat.from_file(filename).dataset(0)
>>> dataset.data(0)
JsonStatValue(idx=0, value=5.943826289, status=None)
>>> dataset.data(concept='UNR', area='AU', year='2003')
JsonStatValue(idx=0, value=5.943826289, status=None)
>>> dataset.data(area='AU', year='2003')
JsonStatValue(idx=0, value=5.943826289, status=None)
>>> dataset.data({'area':'AU', 'year':'2003'})
JsonStatValue(idx=0, value=5.943826289, status=None)
```

JsonStatDataSet.**value**(**args*, ***kargs*)
get a value For the parameters see py:meth:*jsonstat.JsonStatDataSet.data*.

Returns value (typically a number)

JsonStatDataSet.**status**(**args*, ***kargs*)
get datapoint status

For the parameters see py:meth:*jsonstat.JsonStatDataSet.data*.

Returns status (typically a string)

transforming

JsonStatDataSet.**to_table**(*content=u'label'*, *order=None*, *rtype=<type 'list'>*, *blocked_dims={}*, *value_column=u'Value'*, *without_one_dimensions=False*)

Transforms a dataset into a table (a list of row)

table len is the size of dataset + 1 for headers

Parameters

- **content** – can be "label" or "id"
- **order** –
- **rtype** –
- **blocked_dims** –

Returns a list of row, first line is the header

JsonStatDataSet.**to_data_frame**(*index=None*, *content=u'label'*, *order=None*, *blocked_dims={}*, *value_column=u'Value'*)

Transform dataset to pandas data frame

extract_bidimensional("year", "country") generate the following dataframe: year | country 2010 | 1 2011 | 2 2012 | 3

Parameters

- **index** –
- **content** –
- **blocked_dims** –
- **order** –
- **value_column** –

Returns

parsing

JsonStatDataSet.**from_file**(*filename*)

read a jsonstat from a file and parse it to initialize this dataset.

It is better to use *jsonstat.from_file()*

Parameters **filename** – path of the file.

Returns itself to chain calls

JsonStatDataSet.**from_string**(*json_string*)

parse a string containing a jsonstat and initialize this dataset

It is better to use *jsonstat.from_string()*

Parameters **json_string** – string containing a jsonstat

Returns itself to chain calls

JsonStatDataSet.**from_json**(*json_data*)

parse a json structure and initialize this dataset

It is better to use py:meth:*jsonstat.from_json*

Parameters **json_data** – json structure

Returns itself to chain calls

JsonStatDimension

class jsonstat.**JsonStatDimension**(*did=None*, *size=None*, *pos=None*, *role=None*)

Represents a JsonStat Dimension. It is contained into a JsonStat Dataset.

```
>>> from jsonstat import JsonStatDimension
>>> json_string = '''{
...                     "label" : "concepts",
...                     "category" : {
...                        "index" : { "POP" : 0, "PERCENT" : 1 },
...                        "label" : { "POP" : "population",
...                                    "PERCENT" : "weight of age group in the␣
↪population" }
...                     }
...                   }
... '''
>>> dim = JsonStatDimension(did="concept", role="metric").from_string(json_string)
>>> len(dim)
2
>>> dim.category(0).index
'POP'
>>> dim.category('POP').label
'population'
>>> dim.category(1)
JsonStatCategory(label='weight of age group in the population', index='PERCENT',␣
↪pos=1)
>>> print(dim)
+-----+-----------+----------------------------------------+
| pos | idx       | label                                  |
+-----+-----------+----------------------------------------+
| 0   | 'POP'     | 'population'                           |
| 1   | 'PERCENT' | 'weight of age group in the population' |
+-----+-----------+----------------------------------------+
>>> json_string_dimension_sex = '''
... {
...     "label" : "sex",
...     "category" : {
...       "index" : {
...         "M" : 0,
...         "F" : 1,
...         "T" : 2
...       },
...       "label" : {
...         "M" : "men",
...         "F" : "women",
...         "T" : "total"
...       }
...     }
... }
... '''
>>> dim = JsonStatDimension(did="sex").from_string(json_string_dimension_sex)
>>> len(dim)
3
```

__init__(*did=None*, *size=None*, *pos=None*, *role=None*)
initialize a dimension

> **Warning:** this is an internal library function (it is not public api)

Parameters

- **did** – id of dimension
- **size** – size of dimension (nr of values)
- **pos** – position of dimension into the dataset
- **role** – of dimension

did()
id of this dimension

label()
label of this dimension

role()
role of this dimension (can be time, geo or metric)

pos()
position of this dimension with respect to the data set to which this dimension belongs

__len__()
size of this dimension

querying methods

JsonStatDimension.**category**(*spec*)
return JsonStatCategory according to spec

Parameters **spec** – can be index (string) or label (string) or a position (integer)

Returns a JsonStatCategory

parsing methods

JsonStatDimension.**from_string**(*json_string*)
parse a json string

Parameters **json_string** –

Returns itself to chain calls

JsonStatDimension.**from_json**(*json_data*)
Parse a json structure representing a dimension

From json-stat.org

> It is used to describe a particular dimension. The name of this object must be one of the strings in the id array. There must be one and only one dimension ID object for every dimension in the id array.

jsonschema for dimension is about:

```
"dimension": {
    "type": "object",
    "properties": {
        "version": {"$ref": "#/definitions/version"},
        "href": {"$ref": "#/definitions/href"},
        "class": {"type": "string", "enum": ["dimension"]},
        "label": {"type": "string"},
        "category": {"$ref": "#/definitions/category"},
        "note": {"type": "array"},
    },
    "additionalProperties": false
},
```

Parameters **json_data** –

Returns itself to chain call

Downloader helper

class jsonstat.**Downloader**(*cache_dir=u'./data'*, *time_to_live=None*)

Helper class to download json stat files.

It has a very simple cache mechanism

cache_dir()

download(*url*, *filename=None*, *time_to_live=None*)

Download url from internet.

Store the downloaded content into <cache_dir>/file. If <cache_dir>/file exists, it returns content from disk

Parameters

- **url** – page to be downloaded
- **filename** – filename where to store the content of url, None if we want not store
- **time_to_live** – how many seconds to store file on disk, None use default time_to_live, 0 don't use cached version if any

Returns the content of url (str type)

```
collection := {
                [ "version" ":" `string` ]
                [ "class" ":" "collection" ]
                [ "href" ":" `url` ]
                [ "updated": `date` ]
                link : {
                    item : [
                        ( dataset )+
                    ]
            }
dataset := {
       "version"   : <version>
       "class"     : "dataset",
       "href"      : <url>
       "label"     : <string>
       "id"        : [ <string>+]          # ex. "id" : ["metric", "time", "geo",
↪"sex"],
```

```
       "size"      : [ <int>, <int>, ... ]
       "role"      : roles of dimension
       "value"     : [<int>, <int> ]
       "status"    : status
       "dimension" : { <dimension_id> : dimension, ...}
       "link"      :
       }

dimension_id := <string>

# possible values of dimension are called categories
dimension := {
       "label" : <string>
       "class" : "dimension"
       "category: {
                 "index"       : dimension_index
                 "label"       : dimension_label
                 "child"       : dimension_child
                 "coordinates" :
                 "unit"        : dimension_unit
                  }
       }

dimension_index :=
                    { <cat1>:int, ....}      # { "2003" : 0, "2004" : 1, "2005" : 2,
↪"2006" : 3 }
                |
                    [ <cat1>, <cat2> ]   # [  2003, 2004 ]
```

dimension_label := { lbl1:idx1

Istat Module

This module contains helper class useful exploring the Italian Statistics Institute.

Utility Function

`istat.`**`cache_dir`**(*cache_dir=None*, *time_to_live=None*)
Manage the directory `cached_dir` where to store downloaded files

without parameter get the directory with a parameter set the directory :param time_to_live: :param cache_dir:

`istat.`**`areas`**()
returns a list of IstatArea objects representing all the area used to classify datasets

`istat.`**`area`**(*spec*)
returns a IstatArea object conforming to `spec`. :param spec: name of istat area

`istat.`**`dataset`**(*spec_area*, *spec_dataset*)
returns the IstatDataset identified by `spec_dataset` (name of the dataset) contained into the IstatArea identified by `spec_area` (name of the area) :param spec_area: name of istat area :param spec_dataset: name of istat dataset

IstatArea

class istat.**IstatArea**(*istat_helper*, *iid*, *cod*, *desc*)
Represents a Area. An Area contains a list of dataset. Instances of this class are build only by Istat class

cod
returns name of the area

dataset(*spec*)
get a instance of IstatDataset by spec :param spec: code of the dataset :return: IstatDataset instance

datasets()
Returns a list of IstatDataset

desc
returns name of the area

iid
returns the id of the area

info()
print some info about the area

IstatDataset

class istat.**IstatDataset**(*istat_helper*, *dataset*)

cod
returns the code of this dataset

dimension(*spec*)
Get dimension according to spec

Parameters **spec** – can be a int or a string

Returns an IstatDimension instance

dimensions()
Get list of dimensions

Returns list of IstatDimension

getvalues(*spec*, *rtype=<class jsonstat.collection.JsonStatCollection>*)
get values by dimensions

Parameters

- **spec** – it is a string for ex. "1,6,9,0,0"
- **type** –

Returns if type is JsonStatCollection return an istance of JsonStatCollection otherwise return a json structure representing the istat dataset

info_dimensions()
print info about dimensions of this dataset

name
returns the name of this dataset

nrdim()
returns the number of dimensions

IstatDimension

class istat.**IstatDimension**(*name*, *pos*, *json_data*)
Represents a IstatDimension (it is different from JsonStat Dimension

cod2desc(*spec*)
returns the description corresponding to the cod

desc2cod(*spec*)
returns the code corresponding to the description

name
the name of the istat dimension

pos
position into the dataset

CHAPTER 4

jsonstat.py

jsonstat.py is a library for reading the JSON-stat data format maintained and promoted by Xavier Badosa. The JSON-stat format is a JSON format for publishing dataset. JSON-stat is used by several institutions to publish statistical data. An incomplete list is:

- Eurostat that provide statistical information about the European Union (EU)
- Italian National Institute of Statistics Istat
- Central Statistics Office of Ireland
- United Nations Economic Commission for Europe (UNECE) statistical data are here
- Statistics Norway
- UK Office for national statistics see their blog post
- others...

jsonstat.py library tries to mimic as much is possible in python the json-stat Javascript Toolkit. One of the library objectives is to be helpful in exploring dataset using jupyter (ipython) notebooks.

For a fast overview of the feature you can start from this example notebook oecd-canada-jsonstat_v1.html You can also check out some of the jupyter example notebook from the example directory on github or from the documentation

As bonus **jsonstat.py** contains an useful classes to explore dataset published by Istat.

You can find useful another python library pyjstat by Miguel Expósito Martín concerning json-stat format.

This library is in beta status. I am actively working on it and hope to improve this project. For every comment feel free to contact me gf@26fe.com

You can find source at github , where you can open a ticket, if you wish.

You can find the generated documentation at readthedocs.

Installation

Pip will install all required dependencies. For installation:

pip install jsonstat.py

Usage

Simple Usage

There is a simple command line interface, so you can experiment to parse jsonstat file without write code:

```
# parsing collection
$ jsonstat info --cache_dir /tmp http://json-stat.org/samples/oecd-canada.json
downloaded file(s) are stored into '/tmp'
download 'http://json-stat.org/samples/oecd-canada.json'
Jsonsta    tCollection contains the following JsonStatDataSet:
+-----+----------+
| pos | dataset  |
+-----+----------+
| 0   | 'oecd'   |
| 1   | 'canada' |
+-----+----------+

# parsing dataset
$ jsonstat info --cache_dir /tmp  "http://ec.europa.eu/eurostat/wdds/rest/data/v2.1/
↪json/en/tesem120?sex=T&precision=1&age=TOTAL&s_adj=NSA"
downloaded file(s) are stored into '/tmp'
download 'http://ec.europa.eu/eurostat/wdds/rest/data/v2.1/json/en/tesem120?sex=T&
↪precision=1&age=TOTAL&s_adj=NSA'
name:   'Unemployment rate'
label:  'Unemployment rate'
size: 467
+-----+-------+-------+------+------+
| pos | id    | label | size | role |
+-----+-------+-------+------+------+
| 0   | s_adj | s_adj | 1    |      |
| 1   | age   | age   | 1    |      |
| 2   | sex   | sex   | 1    |      |
| 3   | geo   | geo   | 39   |      |
| 4   | time  | time  | 12   |      |
+-----+-------+-------+------+------+
```

code example:

```
url = 'http://json-stat.org/samples/oecd-canada.json'
collection = jsonstat.from_url(url)

# print list of dataset contained into the collection
print(collection)

# select the first dataset of the collection and print a short description
oecd = collection.dataset(0)
print(oecd)

# print description about each dimension of the dataset
for d in oecd.dimensions():
    print(d)

# print a datapoint contained into the dataset
```

```
print(oecd.value(area='IT', year='2012'))

# convert a dataset in pandas dataframe
df = oecd.to_data_frame('year')
```

For more python script examples see examples directory.

For jupyter (ipython) notebooks see examples-notebooks directory.

Support

This is an open source project, maintained in my spare time. Maybe a particular features or functions that you would like are missing. But things don't have to stay that way: you can contribute the project development yourself. Or notify me and ask to implement it.

Bug reports and feature requests should be submitted using the github issue tracker. Please provide a full traceback of any error you see and if possible a sample file. If you are unable to make a file publicly available then contact me at gf@26fe.com.

You can find support also on the google group.

How to Contribute Code

Any help will be greatly appreciated, just follow those steps:

1. Fork it. Start a new fork for each independent feature, don't try to fix all problems at the same time, it's easier for those who will review and merge your changes.
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Write your code. Add unit tests for your changes! If you added a whole new feature, or just improved something, you can be proud of it, so add yourself to the `AUTHORS` file :-) Update the docs!
4. Commit your changes (`git commit -am 'Added some feature'`)
5. Push to the branch (`git push origin my-new-feature`)
6. Create new Pull Request. Click on the large "pull request" button on your repository. Wait for your code to be reviewed, and, if you followed all theses steps, merged into the main repository.

License

jsonstat.py is provided under the LGPL license. See LICENSE file.

CHAPTER 5

Indices and tables

- genindex
- modindex

Python Module Index

i

j

Index

Symbols

A

C

D

F

G

I

J

L

N

P

R

S

T

V