mendeleev Documentation

Release 0.14.0

Lukasz Mentel

Jul 09, 2023
## CONTENTS

1 Getting started 3
  1.1 Overview .................................................. 3
  1.2 Contributing ............................................ 3
  1.3 Citing .................................................... 3
  1.4 Related projects ......................................... 4
  1.5 Funding .................................................. 4

2 Installation .................................................. 5

3 Tutorials .................................................. 7
  3.1 Quick start ............................................... 7
  3.2 Bulk data access ........................................ 16
  3.3 Electronic configuration ................................ 23
  3.4 Ions ....................................................... 25
  3.5 Visualizing custom periodic tables ...................... 27
  3.6 Advanced visualization tutorial ......................... 29
  3.7 Jupyter notebooks ....................................... 32

4 Data ......................................................... 33
  4.1 Elements .................................................. 33
  4.2 Isotopes ................................................... 37
  4.3 Isotope Decay Modes .................................... 37

5 Accessing data ............................................... 39
  5.1 Individual Elements ..................................... 39
  5.2 Fetching data in bulk .................................. 39
  5.3 Computed properties .................................... 40
  5.4 Database session and engine ............................ 41

6 Electronegativities ........................................... 43
  6.1 Allen ..................................................... 43
  6.2 Allred and Rochow ....................................... 44
  6.3 Cottrell and Sutton ..................................... 44
  6.4 Ghosh ..................................................... 44
  6.5 Gordy ...................................................... 45
  6.6 Li and Xue ................................................ 45
  6.7 Martynov and Batsanov .................................. 45
  6.8 Mulliken .................................................. 46
  6.9 Nagle ...................................................... 46
  6.10 Pauling ................................................... 46
  6.11 Sanderson ............................................... 46
9.34 v0.2.12 (21-12-2016) ............................................................... 63
9.35 v0.2.11 (10-11-2016) ............................................................... 63
9.36 v0.2.10 (18-10-2016) ............................................................... 64
9.37 v0.2.9 (16-10-2016) ............................................................... 64
9.38 v0.2.8 (29-08-2016) ............................................................... 64
9.39 v0.2.7 (02-04-2016) ............................................................... 64
9.40 v0.2.6 (02-04-2016) ............................................................... 64
9.41 v0.2.5 (02-04-2016) ............................................................... 64
9.42 v0.2.4 (05-02-2016) ............................................................... 65
9.43 v0.2.3 (27-01-2016) ............................................................... 65
9.44 v0.2.2 (29-11-2015) ............................................................... 65
9.45 v0.2.1 (26-10-2015) ............................................................... 66
9.46 v0.2.0 (22-10-2015) ............................................................... 66
9.47 v0.1.0 (11-07-2015) ............................................................... 67

10 License  ............................................................... 69

11 Indices and tables ............................................................... 71

Bibliography ............................................................... 73

Python Module Index ............................................................... 79

Index ............................................................... 81
This package provides an API for accessing various properties of elements from the periodic table of elements.
GETTING STARTED

1.1 Overview

This package provides an API for accessing various properties of elements from the periodic table of elements. The repository is hosted on github.

1.2 Contributing

All contributions are welcome!

If you would like to suggest an improvement or report a bug or data inconsistency please consider creating an issue on github. I would be especially grateful for references to possible data updates and sources and recommendations of new data.

1.3 Citing

If you use mendeleev in a scientific publication, please consider citing the software as


Here's the reference in the BibLaTeX format

```latex
@software{mendeleev2014,
  author = {Mentel, Łukasz},
  title = {{mendeleev} -- A Python resource for properties of chemical elements, ions and isotopes},
  url = {https://github.com/lmmentel/mendeleev},
  version = {0.14.0},
  date = {2014--},
}
```

or the older BibTeX format
1.4 Related projects

periodictable
   This package provides a periodic table of the elements with support for mass, density and xray/neutron scattering information.

periodic
   Periodic is an open source simple python API/command line script for the periodic table.

1.5 Funding

This project is supported by the RCN (The Research Council of Norway) project number 239193.
The package can be installed using `pip`:

```bash
pip install mendeleev
```

You can also install the most recent version from the repository:

```bash
pip install git+https://github.com/lmmentel/mendeleev.git
```

If you use `conda` you can install the package from my anaconda channel by:

```bash
conda install -c lmmentel mendeleev=0.14.0
```
3.1 Quick start

This simple tutorial will illustrate the basic capabilities of the package.

3.1.1 Table of Contents

- Basic interactive usage
  - Getting single elements
  - Getting-list-of-elements
- Extended attributes
  - Oxidation states
  - Ionization energies
  - Isotopes
  - Ionic radii
  - Electronic configuration
- Useful functions for calculating properties
- Electronegativity
- CLI utility

3.1.2 Basic interactive usage

Getting single elements

The simplest way of accessing the elements is importing them directly from mendeleev by symbols

```
[1]: from mendeleev import Si, Fe, O
   print("Si's name: ", Si.name)
   print("Fe's atomic number:", Fe.atomic_number)
   print("O's atomic weight: ", O.atomic_weight)

Si's name: Silicon
Fe's atomic number: 26
O's atomic weight: 15.999
```
An alternative interface to the data is through the `element` function that returns a single `Element` object or a list of `Element` object depending on the arguments.

The function can be imported directly from the `mendeleev` package

```python
from mendeleev import element
```

The `element` method accepts unique identifiers: atomic number, atomic symbol or element’s name in English. To retrieve the entries on Silicon by symbol type

```python
si = element('Si')
```

```python
si
```

```python
Element(
    abundance_crust=282000.0,
    abundance_sea=2.2,
    annotation='',
    atomic_number=14,
    atomic_radius=110.0,
    atomic_radius_rahm=231.99999999999997,
    atomic_volume=12.1,
    atomic_weight=28.085,
    atomic_weight_uncertainty=None,
    block='p',
    c6=305.0,
    c6_gb=308.0,
    cas='7440-21-3',
    covalent_radius_bragg=117.0,
    covalent_radius_cordero=111.00000000000001,
    covalent_radius_pyykko=115.99999999999999,
    covalent_radius_pyykko_double=107.0,
    covalent_radius_pyykko_triple=102.0,
    cpk_color='#daa520',
    density=2.3296,
    description="Metalloid element belonging to group 14 of the periodic table. It is the second most abundant element in the Earth’s crust, making up 25.7% of it by weight. Chemically less reactive than carbon. First identified by Lavoisier in 1787 and first isolated in 1823 by Berzelius.",
    dipole_polarizability=37.3,
    dipole_polarizability_unc=0.7,
    discoverers='Jöns Berzelius',
    discovery_location='Sweden',
    discovery_year=1824,
    ec=<ElectronicConfiguration(conf="1s2 2s2 2p6 3s2 3p2")>,
    econf='[Ne] 3s2 3p2',
    electron_affinity=1.3895211,
    en_allen=11.33,
    en_ghosh=0.178503,
    en_pauling=1.9,
    evaporation_heat=383.0,
    fusion_heat=50.6,
    gas_basicity=814.1,
    geochemical_class='major',
```

(continues on next page)
glawe_number=85,
goldschmidt_class='litophile',
group=<Group(symbol=IVA, name=Carbon group)>,
group_id=14,
heat_of_formation=450.0,
ionic_radii=[IonicRadius(
    atomic_number=14,
    charge=4,
    coordination='IV',
    crystal_radius=40.0,
    econf='2p6',
    id=379,
    ionic_radius=26.0,
    most_reliable=True,
    origin='',
    spin='',
), IonicRadius(
    atomic_number=14,
    charge=4,
    coordination='VI',
    crystal_radius=54.0,
    econf='2p6',
    id=380,
    ionic_radius=40.0,
    most_reliable=True,
    origin='from r^3 vs V plots',
    spin='',
)],
is_monoisotopic=None,
is_radioactive=False,
isotopes=[<Isotope(Z=14, A=22, mass=22.0361(5), abundance=None)>, <Isotope(Z=14, A=23, mass=23.0257(5), abundance=None)>, <Isotope(Z=14, A=24, mass=24.01154(2), abundance=None)>, <Isotope(Z=14, A=25, mass=25.00411(1), abundance=None)>, <Isotope(Z=14, A=26, mass=25.992338(1), abundance=None)>, <Isotope(Z=14, A=27, mass=26.9867047(1), abundance=None)>, <Isotope(Z=14, A=28, mass=27.9769265344(6), abundance=92.254(4))>, <Isotope(Z=14, A=29, mass=28.9764946643(6), abundance=4.67(2))>, <Isotope(Z=14, A=30, mass=29.97377014(2), abundance=3.074(2))>, <Isotope(Z=14, A=31, mass=30.9736320(5), abundance=None)>, <Isotope(Z=14, A=32, mass=31.9741515(3), abundance=None)>, <Isotope(Z=14, A=33, mass=32.97797708, abundance=None)>, <Isotope(Z=14, A=34, mass=33.9785380(9), abundance=None)>, <Isotope(Z=14, A=35, mass=34.98455(4), abundance=None)>, <Isotope(Z=14, A=36, mass=35.98665(8), abundance=None)>, <Isotope(Z=14, A=37, mass=36.9929(1), abundance=None)>, <Isotope(Z=14, A=38, mass=37.9955(1), abundance=None)>, <Isotope(Z=14, A=39, mass=39.0025(1), abundance=None)>, <Isotope(Z=14, A=40, mass=40.0061(1), abundance=None)>, <Isotope(Z=14, A=41, mass=41.0142(3), abundance=None)>, <Isotope(Z=14, A=42, mass=42.0181(3), abundance=None)>, <Isotope(Z=14, A=43, mass=43.0261(4), abundance=None)>, <Isotope(Z=14, A=44, mass=44.0315(5), abundance=None)>, <Isotope(Z=14, A=45, mass=45.0398(6), abundance=None)>, jmol_color='#f0c8a0', lattice_constant=5.43, lattice_structure='DIA', mendeleev_number=88,
metallic_radius=117.0,
molar_heat_capacity=19.99,
molcas_gv_color='#f0c8a0',
metallic_radius_c12=138.0,
name='Silicon',
name_origin='Latin: silex, silicus, (flint).',
period=3,
pettifor_number=85,
phase_transitions=[14 Tm=1687.15 Tb=3538.15],
proton_affinity=837.0,
screening_constants=[<ScreeningConstant(Z=14, n=1, s=s, screening=0.4255)>, <ScreeningConstant(Z=14, n=2, s=p, screening=4.055)>, <ScreeningConstant(Z=14, n=2, s=s, screening=4.980)>, <ScreeningConstant(Z=14, n=3, s=p, screening=9.7148)>, <ScreeningConstant(Z=14, n=3, s=s, screening=9.0968)>],
sources='Makes up major portion of clay, granite, quartz (SiO2), and sand. Commercial production depends on a reaction between sand (SiO2) and carbon at a temperature of around 2200 °C.',
specific_heat_capacity=0.712,
symbol='Si',
thermal_conductivity=149.0,
uses='Used in glass as silicon dioxide (SiO2). Silicon carbide (SiC) is one of the hardest substances known and used in polishing. Also the crystalline form is used in semiconductors.',
vdw_radius=210.0,
vdw_radius_alvarez=219.0,
vdw_radius_batsanov=210.0,
vdw_radius_bondi=210.0,
vdw_radius_dreiding=426.99999999999994,
vdw_radius_mm3=229.0,
vdw_radius_rt=None,
vdw_radius_truhlar=None,
vdw_radius_uff=429.5,

Similarly to access the data by atomic number or element names type

[5]: al = element(13)
print(al.name)

Aluminum

[6]: o = element('Oxygen')
print(o.atomic_number)

8
Getting list of elements

The `element` method also accepts list or tuple of identifiers and then returns a list of `Element` objects.

```python
[7]: c, h, o = element(['C', 'Hydrogen', 8])
    print(c.name, h.name, o.name)
Carbon Hydrogen Oxygen
```

3.1.3 Extended attributes

Next to simple attributes returning `str`, `int` or `float`, there are extended attributes:

- `oxistates`, returns a list of oxidation states
- `ionenergies`, returns a dictionary of ionization energies
- `isotopes`, returns a list of `Isotope` objects
- `ionic_radii` returns a list of `IonicRadius` objects
- `ec`, electronic configuration object

Oxidation states

`oxistates` returns a list of most common oxidation states for a given element.

```python
[8]: fe = element('Fe')
    print(fe.oxistates)
[2, 3]
```

Ionization energies

The `ionenergies` returns a dictionary with ionization energies in eV as values and degrees of ionization as keys.

```python
[9]: o = element('O')
o.ionenergies
[9]: {1: 13.618054,
   2: 35.12111,
   3: 54.93554,
   4: 77.4135,
   5: 113.8989,
   6: 138.1189,
   7: 739.32679,
   8: 871.40985}
```
Isotopes

The `isotopes` attribute returns a list of `Isotope` objects with the following attributes per isotope:

- abundance
- abundance_uncertainty
- atomic_number
- discovery_year
- g_factor
- g_factor_uncertainty
- half_life
- half_life_uncertainty
- half_life_unit
- is_radioactive
- mass
- mass_number
- mass_uncertainty
- parity
- quadrupole_moment
- quadrupole_moment_uncertainty
- spin

```
for iso in Fe.isotopes:
    print('{0:4d} {1:4d} {2:10.5f} {3:8.2e} {4:} {5:}'.format(
        iso.atomic_number, iso.mass_number, iso.mass, iso.mass_uncertainty, iso.
        abundance, iso.is_radioactive))
```

```
# (continues on next page)
```
Accessing isotopes

Similarly to element function that can be used to fetch specific isotopes by:

- atomic_number and mass_number or
- symbol and mass_number

[11]: from mendeleev import isotope

[12]: isotope("Fe", mass_number=57)

[12]: <Isotope(Z=26, A=57, mass=56.9353920(3), abundance=2.12(3))>

[13]: # tritium
    isotope(1, 3)

[13]: <Isotope(Z=1, A=3, mass=3.01604928132(8), abundance=None)>

Radioactive isotopes can have multiple decay modes and that data is available as decay_modes attribute for each Isotope

[14]: isotope("Li", 11).decay_modes

[14]: [<IsotopeDecayMode(id=40, isotope_id=39, mode='B-', intensity=100.0)>,
    <IsotopeDecayMode(id=41, isotope_id=39, mode='B-n', intensity=86.3)>,
    <IsotopeDecayMode(id=42, isotope_id=39, mode='2n', intensity=4.1)>,
    <IsotopeDecayMode(id=43, isotope_id=39, mode='3n', intensity=1.9)>,
    <IsotopeDecayMode(id=44, isotope_id=39, mode='B-A', intensity=1.7)>,
    <IsotopeDecayMode(id=45, isotope_id=39, mode='B-d', intensity=0.013)>,
    <IsotopeDecayMode(id=46, isotope_id=39, mode='B-t', intensity=0.0093)>]
Ionic radii

Another composite attribute is `ionic_radii` which returns a list of `IonicRadius` object with the following attributes:

- **atomic_number**, atomic number of the ion
- **charge**, charge of the ion
- **econf**, electronic configuration of the ion
- **coordination**, coordination type of the ion
- **spin**, spin state of the ion (HS or LS)
- **crystal_radius**, crystal radius in pm
- **ionic_radius**, ionic radius in pm
- **origin**, source of the data
- **most_reliable**, recommended value, (see the original paper for more information)

```python
for ir in fe.ionic_radii:
    print(ir)
```

<table>
<thead>
<tr>
<th>charge</th>
<th>coordination</th>
<th>crystal_radius</th>
<th>ionic_radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>IV</td>
<td>77.000</td>
<td>63.000</td>
</tr>
<tr>
<td>2</td>
<td>IVSQ</td>
<td>78.000</td>
<td>64.000</td>
</tr>
<tr>
<td>2</td>
<td>VI</td>
<td>75.000</td>
<td>61.000</td>
</tr>
<tr>
<td>2</td>
<td>VI</td>
<td>92.000</td>
<td>78.000</td>
</tr>
<tr>
<td>2</td>
<td>VII</td>
<td>106.000</td>
<td>92.000</td>
</tr>
<tr>
<td>3</td>
<td>IV</td>
<td>63.000</td>
<td>49.000</td>
</tr>
<tr>
<td>3</td>
<td>V</td>
<td>72.000</td>
<td>58.000</td>
</tr>
<tr>
<td>3</td>
<td>VI</td>
<td>69.000</td>
<td>55.000</td>
</tr>
<tr>
<td>3</td>
<td>VI</td>
<td>78.500</td>
<td>64.500</td>
</tr>
<tr>
<td>3</td>
<td>VIII</td>
<td>92.000</td>
<td>78.000</td>
</tr>
<tr>
<td>4</td>
<td>VI</td>
<td>72.500</td>
<td>58.500</td>
</tr>
<tr>
<td>6</td>
<td>IV</td>
<td>39.000</td>
<td>25.000</td>
</tr>
</tbody>
</table>

### 3.1.4 Useful functions for calculating properties

Next to stored attributes there is a number of useful functions

```python
si = element('Si')
```

```python
# get the number of valence electrons
si.nvalence()
```

```python
4
```

```python
# calculate softness for an ion
si.softness(charge=2)
```

```python
0.058318712346158874
```

```python
# calculate hardness for an ion
si.hardness(charge=4)
```
Electronegativity

Currently there are 9 electronegativity scales implemented that can be accessed through the common `electronegativity` method, the scales are:

- allen
- allred-rochow
- cottrell-sutton
- ghosh
- gordy
- li-xue
- martynov-batsanov
- mulliken
- nagle
- pauling
- sanderson

More information can be found in the documentation.
3.1.5 CLI utility

For those who work in the terminal there is a simple command line interface (CLI) for printing the information about a given element. The script name is element.py and it accepts either the symbol or name of the element as an argument and prints the data about it. For example, to print the properties of silicon type

```

element.py Si
```

```
/usr/bin/sh: 1: element.py: not found
```

3.2 Bulk data access

This tutorial explains how to retrieve full tables from the database into pandas DataFrames.

3.2.1 The following tables are available from mendeleev

- elements
- ionicradii
- ionizationenergies
- oxidationstates
- groups
- series
- isotopes

All data is stored in a sqlite database that is shipped together with the package. You can interact directly with the database if you need more flexibility but for convenience mendeleev provides a few functions in the fetch module to retrieve data.

To fetch whole tables you can use fetch_table. The function can be imported from mendeleev.fetch

```
from mendeleev.fetch import fetch_table
```

To retrieve a table call the fetch_table with the table name as argument. Here we’ll get probably the most important table elements with basis data on each element

```
ptable = fetch_table('elements')
```

Now we can use pandas’ capabilities to work with the data.

```
ptable.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 118 entries, 0 to 117
Data columns (total 70 columns):
#   Column       Non-Null Count  Dtype
---  ------       --------------  -----  
0   annotation  118 non-null    object
1   atomic_number 118 non-null   int64
2   atomic_radius 90 non-null    float64
3   atomic_volume 91 non-null    float64
```

(continues on next page)
<table>
<thead>
<tr>
<th></th>
<th>Name</th>
<th>Non-null</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>block</td>
<td>118</td>
<td>object</td>
</tr>
<tr>
<td>5</td>
<td>boiling_point</td>
<td>96</td>
<td>float64</td>
</tr>
<tr>
<td>6</td>
<td>density</td>
<td>95</td>
<td>float64</td>
</tr>
<tr>
<td>7</td>
<td>description</td>
<td>109</td>
<td>object</td>
</tr>
<tr>
<td>8</td>
<td>dipole_polarizability</td>
<td>117</td>
<td>float64</td>
</tr>
<tr>
<td>9</td>
<td>electron_affinity</td>
<td>77</td>
<td>float64</td>
</tr>
<tr>
<td>10</td>
<td>electronic_configuration</td>
<td>118</td>
<td>object</td>
</tr>
<tr>
<td>11</td>
<td>evaporation_heat</td>
<td>88</td>
<td>float64</td>
</tr>
<tr>
<td>12</td>
<td>fusion_heat</td>
<td>75</td>
<td>float64</td>
</tr>
<tr>
<td>13</td>
<td>group_id</td>
<td>90</td>
<td>float64</td>
</tr>
<tr>
<td>14</td>
<td>lattice_constant</td>
<td>87</td>
<td>float64</td>
</tr>
<tr>
<td>15</td>
<td>lattice_structure</td>
<td>91</td>
<td>object</td>
</tr>
<tr>
<td>16</td>
<td>melting_point</td>
<td>100</td>
<td>float64</td>
</tr>
<tr>
<td>17</td>
<td>name</td>
<td>118</td>
<td>object</td>
</tr>
<tr>
<td>18</td>
<td>period</td>
<td>118</td>
<td>int64</td>
</tr>
<tr>
<td>19</td>
<td>series_id</td>
<td>118</td>
<td>int64</td>
</tr>
<tr>
<td>20</td>
<td>specific_heat</td>
<td>81</td>
<td>float64</td>
</tr>
<tr>
<td>21</td>
<td>symbol</td>
<td>118</td>
<td>object</td>
</tr>
<tr>
<td>22</td>
<td>thermal_conductivity</td>
<td>66</td>
<td>float64</td>
</tr>
<tr>
<td>23</td>
<td>vdw_radius</td>
<td>103</td>
<td>float64</td>
</tr>
<tr>
<td>24</td>
<td>covalent_radius_cordero</td>
<td>96</td>
<td>float64</td>
</tr>
<tr>
<td>25</td>
<td>covalent_radius_pyykko</td>
<td>118</td>
<td>float64</td>
</tr>
<tr>
<td>26</td>
<td>en_pauling</td>
<td>85</td>
<td>float64</td>
</tr>
<tr>
<td>27</td>
<td>en_allen</td>
<td>71</td>
<td>float64</td>
</tr>
<tr>
<td>28</td>
<td>jmol_color</td>
<td>109</td>
<td>object</td>
</tr>
<tr>
<td>29</td>
<td>cpk_color</td>
<td>103</td>
<td>object</td>
</tr>
<tr>
<td>30</td>
<td>proton_affinity</td>
<td>32</td>
<td>float64</td>
</tr>
<tr>
<td>31</td>
<td>gas_basicity</td>
<td>32</td>
<td>float64</td>
</tr>
<tr>
<td>32</td>
<td>heat_of_formation</td>
<td>89</td>
<td>float64</td>
</tr>
<tr>
<td>33</td>
<td>c6</td>
<td>43</td>
<td>float64</td>
</tr>
<tr>
<td>34</td>
<td>covalent_radius_bragg</td>
<td>37</td>
<td>float64</td>
</tr>
<tr>
<td>35</td>
<td>vdw_radius_bondi</td>
<td>28</td>
<td>float64</td>
</tr>
<tr>
<td>36</td>
<td>vdw_radius_truhlar</td>
<td>16</td>
<td>float64</td>
</tr>
<tr>
<td>37</td>
<td>vdw_radius_rt</td>
<td>9</td>
<td>float64</td>
</tr>
<tr>
<td>38</td>
<td>vdw_radius_batsanov</td>
<td>65</td>
<td>float64</td>
</tr>
<tr>
<td>39</td>
<td>vdw_radius_dreiding</td>
<td>21</td>
<td>float64</td>
</tr>
<tr>
<td>40</td>
<td>vdw_radius_uff</td>
<td>103</td>
<td>float64</td>
</tr>
<tr>
<td>41</td>
<td>vdw_radius_mm3</td>
<td>94</td>
<td>float64</td>
</tr>
<tr>
<td>42</td>
<td>abundance_crust</td>
<td>88</td>
<td>float64</td>
</tr>
<tr>
<td>43</td>
<td>abundance_sea</td>
<td>81</td>
<td>float64</td>
</tr>
<tr>
<td>44</td>
<td>molcas_gv_color</td>
<td>103</td>
<td>object</td>
</tr>
<tr>
<td>45</td>
<td>en_ghosh</td>
<td>103</td>
<td>float64</td>
</tr>
<tr>
<td>46</td>
<td>vdw_radius_alvarez</td>
<td>94</td>
<td>float64</td>
</tr>
<tr>
<td>47</td>
<td>c6_gb</td>
<td>86</td>
<td>float64</td>
</tr>
<tr>
<td>48</td>
<td>atomic_weight</td>
<td>118</td>
<td>float64</td>
</tr>
<tr>
<td>49</td>
<td>atomic_weight_uncertainty</td>
<td>74</td>
<td>float64</td>
</tr>
<tr>
<td>50</td>
<td>is_monoisotopic</td>
<td>21</td>
<td>object</td>
</tr>
<tr>
<td>51</td>
<td>is_radioactive</td>
<td>118</td>
<td>bool</td>
</tr>
<tr>
<td>52</td>
<td>cas</td>
<td>118</td>
<td>object</td>
</tr>
<tr>
<td>53</td>
<td>atomic_radius_rahm</td>
<td>96</td>
<td>float64</td>
</tr>
<tr>
<td>54</td>
<td>geochemical_class</td>
<td>76</td>
<td>object</td>
</tr>
<tr>
<td>55</td>
<td>goldschmidt_class</td>
<td>118</td>
<td>object</td>
</tr>
</tbody>
</table>

### 3.2. Bulk data access

(continues on next page)
For clarity let’s take only a subset of columns

```python
[5]: cols = ['atomic_number', 'symbol', 'atomic_radius', 'en_pauling', 'block', ...
       'vdw_radius_mm3']
```

```python
[6]: ptable[cols].head()
```

<table>
<thead>
<tr>
<th>atomic_number</th>
<th>symbol</th>
<th>atomic_radius</th>
<th>en_pauling</th>
<th>block</th>
<th>vdw_radius_mm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>H</td>
<td>25.0</td>
<td>2.20</td>
<td>s</td>
<td>162.0</td>
</tr>
<tr>
<td>1</td>
<td>He</td>
<td>120.0</td>
<td>NaN</td>
<td>s</td>
<td>153.0</td>
</tr>
<tr>
<td>2</td>
<td>Li</td>
<td>145.0</td>
<td>0.98</td>
<td>s</td>
<td>255.0</td>
</tr>
<tr>
<td>3</td>
<td>Be</td>
<td>105.0</td>
<td>1.57</td>
<td>s</td>
<td>223.0</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>85.0</td>
<td>2.04</td>
<td>p</td>
<td>215.0</td>
</tr>
</tbody>
</table>

It is quite easy now to get descriptive statistics on the data.

```python
[7]: ptable[cols].describe()
```

```
<table>
<thead>
<tr>
<th>atomic_number</th>
<th>atomic_radius</th>
<th>en_pauling</th>
<th>vdw_radius_mm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>118.000000</td>
<td>90.000000</td>
<td>85.000000</td>
</tr>
<tr>
<td>mean</td>
<td>59.500000</td>
<td>149.844444</td>
<td>1.748588</td>
</tr>
<tr>
<td>std</td>
<td>34.207699</td>
<td>40.079110</td>
<td>0.634442</td>
</tr>
<tr>
<td>min</td>
<td>1.000000</td>
<td>25.000000</td>
<td>0.700000</td>
</tr>
<tr>
<td>25%</td>
<td>30.250000</td>
<td>135.000000</td>
<td>1.240000</td>
</tr>
<tr>
<td>50%</td>
<td>59.500000</td>
<td>145.000000</td>
<td>1.700000</td>
</tr>
<tr>
<td>75%</td>
<td>88.750000</td>
<td>178.750000</td>
<td>2.160000</td>
</tr>
<tr>
<td>max</td>
<td>118.000000</td>
<td>260.000000</td>
<td>3.980000</td>
</tr>
</tbody>
</table>
```
3.2.2 Isotopes table

Let try and retrieve another table, namely isotopes

```
[8]: isotopes = fetch_table('isotopes', index_col='id')
```

```
[9]: isotopes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 1 to 406
Data columns (total 11 columns):
# Column      Non-Null Count  Dtype
--- -------- -------------- ----- 
 0 atomic_number  406 non-null   int64
 1 mass       377 non-null    float64
 2 abundance  288 non-null    float64
 3 mass_number  406 non-null    int64
 4 mass_uncertainty  377 non-null    float64
 5 is_radioactive  406 non-null     bool
 6 half_life  121 non-null    float64
 7 half_life_unit  85 non-null    object
 8 spin       323 non-null    float64
 9 g_factor   323 non-null    float64
10 quadrupole_moment  320 non-null    float64
dtypes: bool(1), float64(7), int64(2), object(1)
memory usage: 35.3+ KB
```

Merge the elements table with the isotopes

We can now perform SQL-like merge operation on two DataFrames and produce an outer join

```
[10]: import pandas as pd
```

```
[11]: merged = pd.merge(ptable[cols], isotopes, how='outer', on='atomic_number')
```

now we have the following columns in the merged DataFrame

```
[12]: merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 406 entries, 0 to 405
Data columns (total 16 columns):
# Column            Non-Null Count  Dtype
--- -------- -------------- ----- 
 0 atomic_number   406 non-null   int64
 1 symbol          406 non-null    object
 2 atomic_radius   328 non-null    float64
 3 en_pauling      313 non-null    float64
 4 block           406 non-null    object
 5 vdw_radius_mm3  350 non-null    float64
 6 mass            377 non-null    float64
 7 abundance      288 non-null    float64
 8 mass_number    406 non-null    int64
```

(continues on next page)
To display all the isotopes of Silicon

```
merged[merged['symbol'] == 'Si']
```

<table>
<thead>
<tr>
<th>atomic_number</th>
<th>symbol</th>
<th>atomic_radius</th>
<th>en_pauling</th>
<th>block</th>
<th>vdw_radius_mm3</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Si</td>
<td>110.0</td>
<td>1.9</td>
<td>p</td>
<td>229.0</td>
</tr>
<tr>
<td>29</td>
<td>Si</td>
<td>110.0</td>
<td>1.9</td>
<td>p</td>
<td>229.0</td>
</tr>
<tr>
<td>30</td>
<td>Si</td>
<td>110.0</td>
<td>1.9</td>
<td>p</td>
<td>229.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mass</th>
<th>abundance</th>
<th>mass_number</th>
<th>mass_uncertainty</th>
<th>is_radioactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.976927</td>
<td>0.92191</td>
<td>28</td>
<td>3.0000000e-09</td>
<td>False</td>
</tr>
<tr>
<td>28.976495</td>
<td>0.04699</td>
<td>29</td>
<td>3.0000000e-09</td>
<td>False</td>
</tr>
<tr>
<td>29.973770</td>
<td>0.03110</td>
<td>30</td>
<td>2.0000000e-08</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>half_life</th>
<th>half_life_unit</th>
<th>spin</th>
<th>g_factor</th>
<th>quadrupole_moment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NaN</td>
<td>None</td>
<td>0.5</td>
<td>-4.254995</td>
<td>0.00000</td>
</tr>
<tr>
<td>NaN</td>
<td>None</td>
<td>0.5</td>
<td>-4.254995</td>
<td>0.00000</td>
</tr>
<tr>
<td>NaN</td>
<td>None</td>
<td>0.0</td>
<td>0.000000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>
3.2.3 Ionic radii

The function to fetch ionic radii is called `fetch_ionic_radii` and can either fetch ionic or crystal radii depending on the radius argument.

```python
from mendeleev.fetch import fetch_ionic_radii

irs = fetch_ionic_radii(radius="ionic_radius")
irs.head(10)
```

<table>
<thead>
<tr>
<th>coordination</th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IIIPY</th>
<th>IV</th>
<th>IVPY</th>
<th>IVSQ</th>
<th>IX</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_number</td>
<td>1</td>
<td>1</td>
<td>-38.0</td>
<td>-18.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>59.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>NaN</td>
<td>NaN</td>
<td>16.0</td>
<td>NaN</td>
<td>27.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>NaN</td>
<td>NaN</td>
<td>1.0</td>
<td>NaN</td>
<td>11.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4</td>
<td>NaN</td>
<td>NaN</td>
<td>-8.0</td>
<td>NaN</td>
<td>15.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>146.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>NaN</td>
<td>NaN</td>
<td>-10.4</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>-2</td>
<td>NaN</td>
<td>135.0</td>
<td>136.0</td>
<td>NaN</td>
<td>138.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>-1</td>
<td>NaN</td>
<td>128.5</td>
<td>130.0</td>
<td>NaN</td>
<td>131.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>coordination</th>
<th>VI</th>
<th>VII</th>
<th>VIII</th>
<th>X</th>
<th>XI</th>
<th>XII</th>
<th>XIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomic_number</td>
<td>1</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>92.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>45.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>3</td>
<td>27.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>4</td>
<td>16.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>-3</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>3</td>
<td>16.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>13.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>-2</td>
<td>140.0</td>
<td>NaN</td>
<td>142.0</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>-1</td>
<td>133.0</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>

3.2.4 Ionization energies

To fetch ionization energies use `fetch_ionization_energies` that takes a degree (default is degree=1) argument that can either be a single integer or a list if integers to fetch multiple ionization energies.

```python
from mendeleev.fetch import fetch_ionization_energies

ies = fetch_ionization_energies(degree=2)
ies.head(10)
```

<table>
<thead>
<tr>
<th>atomic_number</th>
<th>IE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>54.417763</td>
</tr>
<tr>
<td>3</td>
<td>75.640094</td>
</tr>
<tr>
<td>4</td>
<td>18.211153</td>
</tr>
<tr>
<td>5</td>
<td>25.154830</td>
</tr>
</tbody>
</table>

(continues on next page)
### 3.2.5 Electronegativities

To fetch all data from electronegativity scales use `fetch_electronegativities`. This can take a few seconds since most of the values need to be computed.

```python
[26]: from mendeleev.fetch import fetch_electronegativities

[27]: ens = fetch_electronegativities()
ens.head(10)
```

```plaintext
<table>
<thead>
<tr>
<th>atomic_number</th>
<th>Allen</th>
<th>Allred-Rochow</th>
<th>Cottrell-Sutton</th>
<th>Ghosh</th>
<th>Gordy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13.610</td>
<td>0.000977</td>
<td>0.176777</td>
<td>0.263800</td>
<td>0.31250</td>
</tr>
<tr>
<td>2</td>
<td>24.590</td>
<td>0.000803</td>
<td>0.192241</td>
<td>0.442712</td>
<td>0.36957</td>
</tr>
<tr>
<td>3</td>
<td>5.392</td>
<td>0.000073</td>
<td>0.098866</td>
<td>0.105093</td>
<td>0.09774</td>
</tr>
<tr>
<td>4</td>
<td>9.323</td>
<td>0.000187</td>
<td>0.138267</td>
<td>0.144986</td>
<td>0.019118</td>
</tr>
<tr>
<td>5</td>
<td>12.130</td>
<td>0.000360</td>
<td>0.174895</td>
<td>0.184886</td>
<td>0.030588</td>
</tr>
<tr>
<td>6</td>
<td>15.050</td>
<td>0.000578</td>
<td>0.208167</td>
<td>0.224776</td>
<td>0.043333</td>
</tr>
<tr>
<td>7</td>
<td>18.130</td>
<td>0.000774</td>
<td>0.234371</td>
<td>0.264930</td>
<td>0.054930</td>
</tr>
<tr>
<td>8</td>
<td>21.360</td>
<td>0.001146</td>
<td>0.268742</td>
<td>0.304575</td>
<td>0.072222</td>
</tr>
<tr>
<td>9</td>
<td>24.800</td>
<td>0.001270</td>
<td>0.285044</td>
<td>0.344443</td>
<td>0.081250</td>
</tr>
<tr>
<td>10</td>
<td>28.310</td>
<td>0.001303</td>
<td>0.295488</td>
<td>0.384390</td>
<td>0.087313</td>
</tr>
</tbody>
</table>
```

(continues on next page)
### 3.3 Electronic configuration

The `ec` attribute is an object from the `ElectronicConfiguration` class that has additional methods for manipulating the configuration. Internally the configuration is represented as a `OrderedDict` from the `collections` module where tuples `(n, s)` (n is the principal quantum number and s is the subshell label) are used as keys and shell occupations are the values.

```python
[1]: from mendeleev import Si

[2]: Si.ec.conf

[2]: OrderedDict([(1, 's'), 2),
               (2, 's'), 2),
               (2, 'p'), 6),
               (3, 's'), 2),
               (3, 'p'), 2))
```

The occupation of different subshells can be accessed by supplying a proper key:

```python
[3]: Si.ec.conf[(1, 's')]

[3]: 2
```

to calculate the number of electrons per shell type:

```python
[4]: Si.ec.electrons_per_shell()

[4]: {'K': 2, 'L': 8, 'M': 4}
```

get the largest value of the principal quantum number.

---

<table>
<thead>
<tr>
<th>atomic_number</th>
<th>Martynov-Batsanov</th>
<th>Mulliken</th>
<th>Nagle</th>
<th>Pauling</th>
<th>Sanderson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.687605</td>
<td>6.799217</td>
<td>0.605388</td>
<td>2.20</td>
<td>2.187771</td>
</tr>
<tr>
<td>2</td>
<td>6.285107</td>
<td>12.293694</td>
<td>1.130639</td>
<td>NaN</td>
<td>1.000000</td>
</tr>
<tr>
<td>3</td>
<td>2.322007</td>
<td>2.695857</td>
<td>0.182650</td>
<td>0.98</td>
<td>0.048868</td>
</tr>
<tr>
<td>4</td>
<td>3.710381</td>
<td>4.661350</td>
<td>0.375615</td>
<td>1.57</td>
<td>0.126847</td>
</tr>
<tr>
<td>5</td>
<td>4.877958</td>
<td>4.149010</td>
<td>0.526974</td>
<td>2.04</td>
<td>0.254627</td>
</tr>
<tr>
<td>6</td>
<td>6.083300</td>
<td>5.630148</td>
<td>0.707393</td>
<td>2.55</td>
<td>0.427525</td>
</tr>
<tr>
<td>7</td>
<td>7.306768</td>
<td>7.267065</td>
<td>0.877498</td>
<td>3.04</td>
<td>0.577482</td>
</tr>
<tr>
<td>8</td>
<td>8.496136</td>
<td>6.809027</td>
<td>1.042218</td>
<td>3.44</td>
<td>0.941649</td>
</tr>
<tr>
<td>9</td>
<td>9.701808</td>
<td>8.711410</td>
<td>1.232373</td>
<td>3.98</td>
<td>1.017681</td>
</tr>
<tr>
<td>10</td>
<td>10.918389</td>
<td>NaN</td>
<td>1.443255</td>
<td>NaN</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
Get the largest value of azimuthal quantum number for a given value of principal quantum number:

```python
Si.ec.max_n()
```

Get the largest value of azimuthal quantum number for a given value of principal quantum number:

```python
Si.ec.max_l(n=3)
```

Find the large noble gas-like core configuration:

```python
Si.ec.get_largest_core()
```

Get the total number of electrons:

```python
Si.ec.ne()
```

Last subshell:

```python
Si.ec.last_subshell()
```

Get unpaired electrons:

```python
Si.ec.unpaired_electrons()
```

Remove electrons by ionizing returns a new configuration with an electron removed:

```python
ionized = Si.ec.ionize()
print(ionized)
```

We can check that it actually has less electrons:

```python
ionized.ne()
```

Spin occupations by subshell:

```python
Si.ec.spin_occupations()
```

Calculate the spin only magnetic moment:

```
OrderedDict([(1, 's'), {'pairs': 1, 'alpha': 1, 'beta': 1, 'unpaired': 0}),
(2, 's'), {'pairs': 1, 'alpha': 1, 'beta': 1, 'unpaired': 0}),
(2, 'p'), {'pairs': 3, 'alpha': 3, 'beta': 3, 'unpaired': 0}),
(3, 's'), {'pairs': 1, 'alpha': 1, 'beta': 1, 'unpaired': 0}),
(3, 'p'), {'pairs': 0, 'alpha': 2, 'beta': 0, 'unpaired': 2})
```
Calculate the screening constant using Slater's rules for 2s orbital

```python
Si.ec.slater_screening(n=2, o='s')
```

4.1499999999999995

3.3.1 Standalone use

You can use the `ElectronicConfiguration` as a standalone class and use all of the methods shown above.

```python
from mendeleev.econf import ElectronicConfiguration
```

```python
ec = ElectronicConfiguration("1s2 2s2 2p6 3s1")
```

Get the valence only configuration

```python
ec.get_valence()
```

```python
<ElectronicConfiguration(conf="3s1")>
```

3.4 Ions

You can use the `Ion` class to work with ions instead of elements. Ions can be created from elements and charge information.

```python
from mendeleev.ion import Ion
```

```python
fe_2 = Ion("Fe", 2)
```

You can access variety of properties of the ion

```python
fe_2.charge
```

2

```python
fe_2.electrons
```

24

```python
fe_2.Z
```

26

```python
fe_2.name
```

'Iron 2+ ion'

You can also print the unicode ion symbol
Ionic radii for this ion are available under radius attribute

```
[8]: fe_2.radius
[8]: [IonicRadius(
    atomic_number=26,
    charge=2,
    coordination='IV',
    crystal_radius=77.0,
    econf='3d6',
    id=149,
    ionic_radius=63.0,
    most_reliable=False,
    origin='',
    spin='HS',
),
 IonicRadius(
    atomic_number=26,
    charge=2,
    coordination='IVSQ',
    crystal_radius=78.0,
    econf='3d6',
    id=150,
    ionic_radius=64.0,
    most_reliable=False,
    origin='',
    spin='HS',
),
 IonicRadius(
    atomic_number=26,
    charge=2,
    coordination='VI',
    crystal_radius=75.0,
    econf='3d6',
    id=151,
    ionic_radius=61.0,
    most_reliable=False,
    origin='estimated',
    spin='LS',
),
 IonicRadius(
    atomic_number=26,
    charge=2,
    coordination='VI',
    crystal_radius=92.0,
    econf='3d6',
    id=152,
    ionic_radius=78.0,
    most_reliable=True,
    origin='from r^3 vs V plots',
),
```
Appropriate value of ionization energy and electron affinity are available under `ie` and `ea` attributes

```python
[9]: fe_2.ie
[9]: 30.651

[10]: fe_2.ea
[10]: 16.1992
```

---

### 3.5 Visualizing custom periodic tables

In this tutorial you’ll how to use `mendeleev` to create customized visualizations of the periodic table.

The most convenient method to use for this is `periodic_table` function from `mendeleev.vis` module.

```python
[1]: from mendeleev.vis import periodic_table

Make sure you have optional `vis` dependencies installed when installing `mendeleev`. If you are using `pip` install with

```bash
pip install mendeleev[vis]
```

To see the default visualization of the periodic table simply call the imported function

```python
[2]: periodic_table()
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

---

3.5. Visualizing custom periodic tables 27
mendeleev stores also two color schemes for atoms that are frequently used for visualizing molecular structures. One set is stored in the cpk_color column and refers to CPK coloring, another is stored in jmol_color column and is used by the Jmol program, finally there is also coloring scheme from MOLCAS GV program store in the molcas_gv_color attribute. They can be displayed either by hovering of the element to display a tooltip or used directly to color the element cells.

```
[3]: periodic_table(colorby='jmol_color', title="Jmol Colors")
```

```
[4]: periodic_table(colorby='cpk_color', title='CPK Colors')
```

```
[5]: periodic_table(colorby='molcas_gv_color', title='MOLCAS GV Colors')
```

## 3.5.1 Visualizing properties

Any of the properties in mendeleev can now be visualized and color coded. This means that the value of selected attribute will be visible on each element and also it is possible to use the attribute to color code the background of each element.

Let's first use the covalent_radius_pyykko and display the values with the default color coding by series

```
[6]: periodic_table(attribute='covalent_radius_pyykko', title="Covalent Radii of Pyykko")
```

Now let's use the same attribute but in addition color code by the actual values, by adding colorby='attribute' argument

```
[7]: periodic_table(attribute='covalent_radius_pyykko', colorby='attribute', title="Covalent Radii of Pyykko")
```

The colormap can also be customized using the cmap argument to any of the standard colormaps available in matplotlib

```
[8]: periodic_table(attribute='covalent_radius_pyykko', colorby='attribute', cmap='spring', title="Covalent Radii of Pyykko")
```

Let also see one of the more modern colormaps: viridis, plasma, inferno and magma.

```python
[9]: periodic_table(attribute='covalent_radius_pyykko', colorby='attribute',
                  cmap='inferno', title="Covalent Radii of Pyykko")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Let's try a different property: atomic_volume

```python
[10]: periodic_table(attribute='atomic_volume', colorby='attribute',
                    title='Atomic Volume')
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

```python
[11]: periodic_table(attribute='en_pauling', colorby='attribute',
                    title="Pauling's Electronegativity", cmap='viridis')
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

### 3.5.2 Wide 32-column version

The `periodic_table` function can also present the periodic table in the so-called wide format with the f-block between the s- and d-blocks resulting in 32 columns.

```python
[12]: periodic_table(height=600, width=1500, wide_layout=True)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

### 3.6 Advanced visualization tutorial

Next to the high level plotting function `mendeleev.vis.periodic_table`, mendeleev offers two lower level functions that give you more control over the result. There are two plotting backends supported:

1. `Plotly` (default)
2. `Bokeh`
3.6.1 Note

Depending on your environment being the classic jupyter notebook or jupyterlab you might have to do additional configuration steps, so if you’re not getting expected results see plotly of bokeh documentation.

3.6.2 Accessing lower level plotting functions

There are two plotting functions, one for each of the backends:

- periodic_table_plotly in mendeleev.vis.plotly
- periodic_table_bokeh in mendeleev.vis.bokeh

that you can use to customize the visualizations even further.

Both functions take the same keyword arguments as the periodic_table function but also require a DataFrame with periodic table data. That dataframe needs to have x and y columns for each element that play the role of coordinates. You can get the default data using the create_vis_dataframe function. Let’s start with an example using the plotly backend.

[1]: from mendeleev.vis import create_vis_dataframe, periodic_table_plotly

The function has only one required argument which is the data itself.

[2]: elements = create_vis_dataframe()
periodic_table_plotly(elements)

3.6.3 Custom coloring scheme

To apply a custom color scheme you can assign color to all the elements in the DataFrame. This can be done by creating a custom column in the DataFrame and then using colorby argument to specify which column contains colors. Let’s try to color the elements according to the block they belong to.

[3]: import seaborn as sns
from matplotlib import colors
blockcmap = {b : colors.rgb2hex(c) for b, c in zip(['s', 'p', 'd', 'f'], sns.color_palette('deep'))}

elements['block_color'] = elements['block'].map(blockcmap)
periodic_table_plotly(elements, colorby='block_color')
3.6.4 Custom properties

You can also visualize custom properties using pandas' awesome methods for manipulating data. For example let's consider the difference of electronegativity between every element and the Oxygen atom. To calculate the values we will use Allen scale this time and call our new value ENX-ENO.

```python
[4]: elements.loc[:, 'ENX-ENO'] = elements.loc[elements['symbol'] == 'O', 'en_allen'].values
    - elements.loc[:, 'en_allen']

periodic_table_plotly(elements, attribute='ENX-ENO', colorby='attribute',
                      cmap='viridis', title='Allen Electronegativity wrt. Oxygen')
```

As a second example let's consider a difference between the covalent_radius_slater and covalent_radius_pyykko values

```python
[5]: elements['cov_rad_diff'] = elements['atomic_radius'] - elements['covalent_radius_pyykko']

periodic_table_plotly(elements, attribute='cov_rad_diff', colorby='attribute',
                       title='Covalent Radii Difference', cmap='viridis')
```

3.6.5 Bokeh backend

We can also use the Bokeh backed in the same way but we need to take a few extra steps to render the result in a notebook

```python
[6]: from bokeh.plotting import show, output_notebook
    from mendeleev.vis import periodic_table_bokeh

First we need to enable notebook output

```python
[7]: output_notebook()
```

```python
[8]: fig = periodic_table_bokeh(elements)
    show(fig)
```

```python
[9]: fig = periodic_table_bokeh(elements, attribute="atomic_radius", colorby="attribute")
    show(fig)
```
3.7 Jupyter notebooks

All tutorials are available as Jupyter notebooks on binder where you can explore the examples interactively:

• Quick start
• Bulk data access
• Electronic Configuration
• Ions
• Visualizations
• Advanced visualizations
To find out how to fetch data in bulk, check out the documentation about *data access*.

### 4.1 Elements

The following data are currently available:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>abundance_crust</td>
<td>float</td>
<td>Abundance in the Earth’s crust</td>
<td>mg/kg</td>
<td>[23]</td>
</tr>
<tr>
<td>abundance_sea</td>
<td>float</td>
<td>Abundance in the seas</td>
<td>mg/L</td>
<td>[23]</td>
</tr>
<tr>
<td>annotation</td>
<td>str</td>
<td>Annotations regarding the data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic_number</td>
<td>int</td>
<td>Atomic number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic_radius</td>
<td>float</td>
<td>Atomic radius</td>
<td>pm</td>
<td>[53]</td>
</tr>
<tr>
<td>atomic_radius_rahm</td>
<td>float</td>
<td>Atomic radius by Rahm et al.</td>
<td>pm</td>
<td>[45, 46]</td>
</tr>
<tr>
<td>atomic_volume</td>
<td>float</td>
<td>Atomic volume</td>
<td>cm$^3$/mol</td>
<td></td>
</tr>
<tr>
<td>atomic_weight</td>
<td>float</td>
<td>Atomic weight</td>
<td></td>
<td>[35, 63]</td>
</tr>
<tr>
<td>atomic_weight_uncert</td>
<td>float</td>
<td>Atomic weight uncertainty</td>
<td></td>
<td>[35, 63]</td>
</tr>
<tr>
<td>block</td>
<td>str</td>
<td>Block in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>boiling_point</td>
<td>float</td>
<td>Boiling temperature</td>
<td>K</td>
<td>[22]</td>
</tr>
<tr>
<td>c6</td>
<td>float</td>
<td>C\textsubscript{6} dispersion coefficient in a.u.</td>
<td>a.u.</td>
<td>[13, 56]</td>
</tr>
<tr>
<td>c6_gb</td>
<td>float</td>
<td>C\textsubscript{6} dispersion coefficient in a.u. (Gould &amp; Bučko)</td>
<td>a.u.</td>
<td>[21]</td>
</tr>
<tr>
<td>cas</td>
<td>str</td>
<td>Chemical Abstracts Serice identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>covalent_radius_bragg</td>
<td>float</td>
<td>Covalent radius by Bragg</td>
<td>pm</td>
<td>[10]</td>
</tr>
<tr>
<td>covalent_radius_cordero</td>
<td>float</td>
<td>Covalent radius by Cerdero et al.</td>
<td>pm</td>
<td>[16]</td>
</tr>
<tr>
<td>covalent_radius_pyykko</td>
<td>float</td>
<td>Single bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[43]</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>cova-</td>
<td>float</td>
<td>Double bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[42]</td>
</tr>
<tr>
<td>lent_radius_pyykko_d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cova-</td>
<td>float</td>
<td>Triple bond covalent radius by Pyykko et al.</td>
<td>pm</td>
<td>[44]</td>
</tr>
<tr>
<td>lent_radius_pyykko_t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cpk_color</td>
<td>str</td>
<td>Element color in CPK convention</td>
<td>HEX</td>
<td>[61]</td>
</tr>
<tr>
<td>critical_pressure</td>
<td>float</td>
<td>Critical pressure</td>
<td>MPa</td>
<td>[22]</td>
</tr>
<tr>
<td>critical_temperature</td>
<td>float</td>
<td>Critical temperature</td>
<td>K</td>
<td>[22]</td>
</tr>
<tr>
<td>density</td>
<td>float</td>
<td>Density at 295K(°)</td>
<td>g/cm³</td>
<td>[23, 66]</td>
</tr>
<tr>
<td>description</td>
<td>str</td>
<td>Short description of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dipole_polarizability</td>
<td>float</td>
<td>Dipole polarizability</td>
<td>a.u.</td>
<td>[51]</td>
</tr>
<tr>
<td>dipole_polarizability_unc</td>
<td>float</td>
<td>Dipole polarizability uncertainty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discoverers</td>
<td>str</td>
<td>The discoverers of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discovery_location</td>
<td>str</td>
<td>The location where the element was discovered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discovery_year</td>
<td>int</td>
<td>The year the element was discovered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>electron_affinity</td>
<td>float</td>
<td>Electron affinity(¹)</td>
<td>eV</td>
<td>[6, 23]</td>
</tr>
<tr>
<td>electrons</td>
<td>int</td>
<td>Number of electrons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>electrophilicity</td>
<td>float</td>
<td>Electrophilicity index</td>
<td>eV</td>
<td>[39]</td>
</tr>
<tr>
<td>en_allen</td>
<td>float</td>
<td>Allen’s scale of electronegativity(²)</td>
<td>eV</td>
<td>[31, 32]</td>
</tr>
<tr>
<td>en_ghosh</td>
<td>float</td>
<td>Ghosh’s scale of electronegativity</td>
<td></td>
<td>[18]</td>
</tr>
<tr>
<td>en_mulliken</td>
<td>float</td>
<td>Mulliken’s scale of electronegativity</td>
<td>eV</td>
<td>[36]</td>
</tr>
<tr>
<td>en_pauling</td>
<td>float</td>
<td>Pauling’s scale of electronegativity</td>
<td></td>
<td>[23]</td>
</tr>
<tr>
<td>econf</td>
<td>str</td>
<td>Ground state electron configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>evaporation_heat</td>
<td>float</td>
<td>Evaporation heat</td>
<td>kJ/mol</td>
<td></td>
</tr>
<tr>
<td>fusion_heat</td>
<td>float</td>
<td>Fusion heat</td>
<td>kJ/mol</td>
<td></td>
</tr>
<tr>
<td>gas_basicity</td>
<td>float</td>
<td>Gas basicity</td>
<td>kJ/mol</td>
<td>[23]</td>
</tr>
<tr>
<td>geochemical_class</td>
<td>str</td>
<td>Geochemical classification</td>
<td></td>
<td>[59]</td>
</tr>
<tr>
<td>glawe_number</td>
<td>int</td>
<td>Glawe’s number (scale)</td>
<td></td>
<td>[19]</td>
</tr>
<tr>
<td>goldschmidt_class</td>
<td>str</td>
<td>Goldschmidt classification</td>
<td></td>
<td>[59, 60]</td>
</tr>
<tr>
<td>group</td>
<td>int</td>
<td>Group in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heat_of_formation</td>
<td>float</td>
<td>Heat of formation</td>
<td>kJ/mol</td>
<td>[23]</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Comment</td>
<td>Unit</td>
<td>Data Source</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>---------------------------------------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>inchi</td>
<td>str</td>
<td>International Chemical Identifier</td>
<td></td>
<td>[24]</td>
</tr>
<tr>
<td>ionenergy</td>
<td>tuple</td>
<td>Ionization energies</td>
<td>eV</td>
<td>[26]</td>
</tr>
<tr>
<td>ionic_radii</td>
<td>list</td>
<td>Ionic and crystal radii in pm(¹)</td>
<td>pm</td>
<td>[29, 52]</td>
</tr>
<tr>
<td>is_monoisotopic</td>
<td>bool</td>
<td>Is the element monoisotopic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_radioactive</td>
<td>bool</td>
<td>Is the element radioactive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>isotopes</td>
<td>list</td>
<td>Isotopes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmol_color</td>
<td>str</td>
<td>Element color in Jmol convention</td>
<td>HEX</td>
<td>[64]</td>
</tr>
<tr>
<td>lattice_constant</td>
<td>float</td>
<td>Lattice constant</td>
<td>Angstrom</td>
<td></td>
</tr>
<tr>
<td>lattice_structure</td>
<td>str</td>
<td>Lattice structure code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mass_number</td>
<td>int</td>
<td>Mass number (most abundant isotope)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>melting_point</td>
<td>float</td>
<td>Melting temperature</td>
<td>K</td>
<td>[22]</td>
</tr>
<tr>
<td>mendeleev_number</td>
<td>int</td>
<td>Mendeleev's number(⁵)</td>
<td></td>
<td>[41, 57]</td>
</tr>
<tr>
<td>metallic_radius</td>
<td>float</td>
<td>Single-bond metallic radius</td>
<td>pm</td>
<td>[1]</td>
</tr>
<tr>
<td>metallic_radius_c12</td>
<td>float</td>
<td>Metallic radius with 12 nearest neighbors</td>
<td>pm</td>
<td>[1]</td>
</tr>
<tr>
<td>molar_heat_capacity</td>
<td>float</td>
<td>Molar heat capacity @ 25 C, 1 bar</td>
<td>J/(mol K)</td>
<td>[23]</td>
</tr>
<tr>
<td>molcas_gv_color</td>
<td>str</td>
<td>Element color in MOCAS GV convention</td>
<td>HEX</td>
<td>[65]</td>
</tr>
<tr>
<td>name</td>
<td>str</td>
<td>Name in English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>name_origin</td>
<td>str</td>
<td>Origin of the name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neutrons</td>
<td>int</td>
<td>Number of neutrons (most abundant isotope)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oxistates</td>
<td>list</td>
<td>Commonly occurring oxidation states</td>
<td></td>
<td>[67]</td>
</tr>
<tr>
<td>nist_webbook_url</td>
<td>str</td>
<td>URL for the NIST Chemistry Web-Book</td>
<td></td>
<td>[38]</td>
</tr>
<tr>
<td>oxistates</td>
<td>list</td>
<td>Oxidation states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>period</td>
<td>int</td>
<td>Period in periodic table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pettifor_number</td>
<td>float</td>
<td>Pettifor scale</td>
<td></td>
<td>[41]</td>
</tr>
<tr>
<td>proton_affinity</td>
<td>float</td>
<td>Proton affinity</td>
<td>kJ/mol</td>
<td>[23]</td>
</tr>
<tr>
<td>protons</td>
<td>int</td>
<td>Number of protons</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sconst</td>
<td>float</td>
<td>Nuclear charge screening constants(⁶)</td>
<td></td>
<td>[14, 15]</td>
</tr>
</tbody>
</table>

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>series</td>
<td>int</td>
<td>Index to chemical series</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sources</td>
<td>str</td>
<td>Sources of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>specific_heat_capacity</td>
<td>float</td>
<td>Specific heat capacity @ 25 C, 1 bar</td>
<td>J/(g K)</td>
<td>[23]</td>
</tr>
<tr>
<td>symbol</td>
<td>str</td>
<td>Chemical symbol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thermal_conductivity</td>
<td>float</td>
<td>Thermal conductivity @25 C</td>
<td>W/(m K)</td>
<td></td>
</tr>
<tr>
<td>triple_point_pressure</td>
<td>float</td>
<td>Triple point pressure</td>
<td>kPa</td>
<td>[22]</td>
</tr>
<tr>
<td>triple_point_temperature</td>
<td>float</td>
<td>Triple point temperature</td>
<td>K</td>
<td>[22]</td>
</tr>
<tr>
<td>uses</td>
<td>str</td>
<td>Applications of the element</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vdw_radius</td>
<td>float</td>
<td>Van der Waals radius</td>
<td>pm</td>
<td>[23]</td>
</tr>
<tr>
<td>vdw_radius_alvarez</td>
<td>float</td>
<td>Van der Waals radius according to Alvarez(7)</td>
<td>pm</td>
<td>[5, 58]</td>
</tr>
<tr>
<td>vdw_radius_batsanov</td>
<td>float</td>
<td>Van der Waals radius according to Batsanov</td>
<td>pm</td>
<td>[8]</td>
</tr>
<tr>
<td>vdw_radius_bondi</td>
<td>float</td>
<td>Van der Waals radius according to Bondi</td>
<td>pm</td>
<td>[9]</td>
</tr>
<tr>
<td>vdw_radius_dreiding</td>
<td>float</td>
<td>Van der Waals radius from the DREIDING FF</td>
<td>pm</td>
<td>[34]</td>
</tr>
<tr>
<td>vdw_radius_mm3</td>
<td>float</td>
<td>Van der Waals radius from the MM3 FF</td>
<td>pm</td>
<td>[3]</td>
</tr>
<tr>
<td>vdw_radius_rt</td>
<td>float</td>
<td>Van der Waals radius according to Rowland and Taylor</td>
<td>pm</td>
<td>[48]</td>
</tr>
<tr>
<td>vdw_radius_truhlar</td>
<td>float</td>
<td>Van der Waals radius according to Truhlar</td>
<td>pm</td>
<td>[33]</td>
</tr>
<tr>
<td>vdw_radius_uff</td>
<td>float</td>
<td>Van der Waals radius from the UFF</td>
<td>pm</td>
<td>[47]</td>
</tr>
</tbody>
</table>

1 Atomic Weights
Atomic weights and their uncertainties were retrieved mainly from ref. [63]. For elements whose values were given as ranges the conventional atomic weights from Table 3 in ref. [35] were taken. For radioactive elements the standard approach was adopted where the weight is taken as the mass number of the most stable isotope. The data was obtained from CIAAW page on radioactive elements. In cases where two isotopes were specified the one with the smaller standard deviation was chosen. In case of Tc and Pm relative weights of their isotopes were used, for Tc isotope 98, and for Pm isotope 145 were taken from CIAAW.

2 Covalent Radius by Cordero et al.
In order to have a more homogeneous data for covalent radii taken from ref. [16] the values for 3 different valences for C, also the low and high spin values for Mn, Fe Co, were respectively averaged.

9 Densities
Density values for solids and liquids are always in units of grams per cubic centimeter and can be assumed to refer to temperatures near room temperature unless otherwise stated. Values for gases are the calculated ideal gas densities at 25°C and 101.325 kPa.
Original values for gasses are converted from g/L to g/cm³.
For elements where several allotropes exist, the density corresponding to the most abundant are reported (for full list refer to [23]), namely:
4.2 Isotopes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>abundance</td>
<td>float</td>
<td>Relative Abundance</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>abundance_uncertainty</td>
<td>float</td>
<td>Uncertainty of relative abundance</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>atomic_number</td>
<td>int</td>
<td>Atomic number</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>decay_modes</td>
<td>obj</td>
<td>Decay modes with intensities</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discovery_year</td>
<td>int</td>
<td>Year the isotope was discovered</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>g_factor</td>
<td>float</td>
<td>Nuclear g-factor</td>
<td></td>
<td>[55]</td>
</tr>
<tr>
<td>g_factor_uncertainty</td>
<td>float</td>
<td>Uncertainty of the nuclear g-factor</td>
<td></td>
<td>[55]</td>
</tr>
<tr>
<td>half_life</td>
<td>float</td>
<td>Half life of the isotope</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>half_life_uncertainty</td>
<td>float</td>
<td>Uncertainty of the half life</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>half_life_unit</td>
<td>str</td>
<td>Unit in which the half life is given</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>is_radioactive</td>
<td>bool</td>
<td>Is the isotope radioactive</td>
<td></td>
<td>[62]</td>
</tr>
<tr>
<td>mass</td>
<td>float</td>
<td>Atomic mass</td>
<td>Da</td>
<td>[62]</td>
</tr>
<tr>
<td>mass_number</td>
<td>int</td>
<td>Mass number of the isotope</td>
<td></td>
<td>[62]</td>
</tr>
<tr>
<td>mass_uncertainty</td>
<td>float</td>
<td>Uncertainty of the atomic mass</td>
<td>Da</td>
<td>[62]</td>
</tr>
<tr>
<td>parity</td>
<td>str</td>
<td>Parity, if present, it can be either + or -</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>quadrupole_moment</td>
<td>float</td>
<td>Nuclear electric quadrupole moment</td>
<td>b [100 fm^2]</td>
<td>[54]</td>
</tr>
<tr>
<td>quadrupole_moment_uncertainty</td>
<td>float</td>
<td>Nuclear electric quadrupole moment</td>
<td>b [100 fm^2]</td>
<td>[54]</td>
</tr>
<tr>
<td>spin</td>
<td>str</td>
<td>Nuclear spin quantum number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3 Isotope Decay Modes

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Comment</th>
<th>Unit</th>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>isotope_id</td>
<td>int</td>
<td>ID of the isotope</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>mode</td>
<td>str</td>
<td>ASCII symbol of the decay mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relation</td>
<td>str</td>
<td>Uncertainty of relative abundance</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>intensity</td>
<td>float</td>
<td>Intensity of the decay mode</td>
<td>%</td>
<td>[25]</td>
</tr>
<tr>
<td>is_allowed_not_observed</td>
<td>bool</td>
<td>If True decay mode is energetically allowed, but not experimentally observed</td>
<td></td>
<td>[25]</td>
</tr>
<tr>
<td>is_observed_intensity_unknown</td>
<td>bool</td>
<td>If True decay mode is observed, but its intensity is not experimentally known</td>
<td></td>
<td>[25]</td>
</tr>
</tbody>
</table>

The different modes in the table are stores as ASCII representations for compatibility. The table below provides explanation:

- Antimony (gray)
- Berkelium (form)
- Carbon (graphite)
- Phosphorus (white)
- Selenium (gray)
- Sulfur (rhombic)
- Tin (white)

For elements where experimental data is not available, theoretical estimates taken from [66] are used, namely for:

- Astatine
- Francium
- Einsteinium
- Fermium
- Mendelevium
- Nobelium
- Rutherfordium
- Dubnium
- Seaborgium
- Bohrium
nations of the symbols.

<table>
<thead>
<tr>
<th>ASCII</th>
<th>Unicode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>α</td>
<td>α emission</td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>proton emission</td>
</tr>
<tr>
<td>2p</td>
<td>2p</td>
<td>2-proton emission</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>neutron emission</td>
</tr>
<tr>
<td>2n</td>
<td>2n</td>
<td>2-neutron emission</td>
</tr>
<tr>
<td>EC</td>
<td>ε</td>
<td>electron capture</td>
</tr>
<tr>
<td>e+</td>
<td>e+</td>
<td>positron emission</td>
</tr>
<tr>
<td>B+</td>
<td>β⁺</td>
<td>β⁺ decay (β⁺ = ε⁺ + e⁺)</td>
</tr>
<tr>
<td>B⁻</td>
<td>β⁻</td>
<td>β⁻ decay</td>
</tr>
<tr>
<td>2B⁻</td>
<td>2β⁻</td>
<td>double β⁻ decay</td>
</tr>
<tr>
<td>2B⁺</td>
<td>2β⁺</td>
<td>double β⁺ decay</td>
</tr>
<tr>
<td>B⁻n</td>
<td>β⁻ n</td>
<td>β⁻-delayed neutron emission</td>
</tr>
<tr>
<td>B⁻2n</td>
<td>β⁻ 2n</td>
<td>β⁻-delayed 2-neutron emission</td>
</tr>
<tr>
<td>B⁻3n</td>
<td>β⁻ 3n</td>
<td>β⁻-delayed 3-neutron emission</td>
</tr>
<tr>
<td>B⁺p</td>
<td>β⁺ p</td>
<td>β⁺-delayed proton emission</td>
</tr>
<tr>
<td>B⁺2p</td>
<td>β⁺ 2p</td>
<td>β⁺-delayed 2-proton emission</td>
</tr>
<tr>
<td>B⁺3p</td>
<td>β⁺ 3p</td>
<td>β⁺-delayed 3-proton emission</td>
</tr>
<tr>
<td>B⁻A</td>
<td>β⁻ α</td>
<td>β⁻-delayed α emission</td>
</tr>
<tr>
<td>B⁺A</td>
<td>β⁺α</td>
<td>β⁺-delayed α emission</td>
</tr>
<tr>
<td>B⁻d</td>
<td>β⁻ d</td>
<td>β⁻-delayed deuteron emission</td>
</tr>
<tr>
<td>B⁻t</td>
<td>β⁻ t</td>
<td>β⁻-delayed triton emission</td>
</tr>
<tr>
<td>IT</td>
<td>IT</td>
<td>internal transition</td>
</tr>
<tr>
<td>SF</td>
<td>SF</td>
<td>spontaneous fission</td>
</tr>
<tr>
<td>B⁺SF</td>
<td>β⁺ SF</td>
<td>β⁺-delayed fission</td>
</tr>
<tr>
<td>B⁻SF</td>
<td>β⁻ SF</td>
<td>β⁻-delayed fission</td>
</tr>
<tr>
<td>24Ne</td>
<td>24Ne</td>
<td>heavy cluster emission</td>
</tr>
</tbody>
</table>

Data Footnotes

The screening constants were calculated according to the following formula

\[ \sigma_{n,l,m} = Z - n \cdot \zeta_{n,l,m} \]

where \( n \) is the principal quantum number, \( Z \) is the atomic number, \( \sigma_{n,l,m} \) is the screening constant, \( \zeta_{n,l,m} \) is the optimized exponent from [14, 15].

For elements Nb, Mo, Ru, Rh, Pd and Ag the exponent values corresponding to the ground state electronic configuration were taken (entries with superscript \( a \) in Table II in [15]).

For elements La, Pr, Nd and Pm two exponent were reported for 4f shell denoted 4f and 4f' in [15]. The value corresponding to 4f were used since according to the authors these are the dominant ones.

\(^7\) van der Waals radii according to Alvarez

The bulk of the radii data was taken from Ref. [5], but the radii for noble gasses were update according to the values in Ref. [58].
CHAPTER FIVE

ACCESSING DATA

5.1 Individual Elements

The easiest way to access individual elements is simply by importing them from the `mendeleev` directly using their symbols:

```python
>>> from mendeleev import H, C, O, Og
>>> [x.name for x in [H, C, O, Og]]
['Hydrogen', 'Carbon', 'Oxygen', 'Oganesson']
```

An alternative method of access is through the `element()` function that returns either a single `Element` instance or a tuple of those instances depending on the input. It provides a more flexible interface since it accepts element names, atomic numbers and symbols as well as their combinations.

5.2 Fetching data in bulk

If you want a whole set of data you can retrieve one of the tables from the database as pandas DataFrame through the `fetch_table()`. The following tables are available:

- elements
- groups
- ionicradii
- ionizationenergies
- isotopes
- oxidationstates
- screeningconstants
- series

```
fetched_table(table: str, **kwargs) → pandas.DataFrame
```

Return a table from the database as pandas.DataFrame

Parameters
- `table` – Name of the table from the database
- `kwargs` – A dictionary of keyword arguments to pass to the pandas.read_qsl()

Returns
Pandas DataFrame with the contents of the table
Return type
df (pandas.DataFrame)

Example

```python
>>> from mendeleev.fetch import fetch_table
>>> df = fetch_table('elements')
>>> type(df)
pandas.core.frame.DataFrame
```

fetch_ionization_energies(degree: List[int] | int = 1) → pandas.DataFrame

Fetch a pandas.DataFrame with ionization energies for all elements indexed by atomic number.

Parameters

degree – Degree of ionization, either as int or a list of ints. If a list is passed then the output will contain ionization energies corresponding to particular degrees in columns.

Returns

ionization energies, indexed by atomic number

Return type
df (pandas.DataFrame)

fetch_ionic_radii(radius: str = 'ionic_radius') → pandas.DataFrame

Fetch a pandas DataFrame with ionic radii for all the elements.

Parameters

radius – The radius to be returned either ionic_radius or crystal_radius

Returns

a table with atomic numbers, symbols and ionic radii for all coordination numbers

Return type
df (pandas.DataFrame)

5.3 Computed properties

Some properties need to be computed rather than directly retrieved from the database. Electronegativities

fetch_electronegativities(scales: List[str] = None) → pandas.DataFrame

Fetch electronegativity scales for all elements as pandas.DataFrame

Parameters

scales – list of scale names, defaults to all available scales

Returns

Pandas DataFrame with the contents of the table

Return type
df (pandas.DataFrame)
5.4 Database session and engine

For those who want to interact with the database through a layer of SQLAlchemy there are methods for getting the session or the engine:

```
get_session(dbpath: str = None) → Session
    Return the database session connection.

get_engine(dbpath: str = None) → Engine
    Return the db engine
```
Since electronegativity is a useful concept rather than a physical observable, several scales of electronegativity exist and some of them are available in mendeleev. Depending on the definition of a particular scale the values are either stored directly or recomputed on demand with appropriate formulas. The following scales are stored:

- Allen
- Ghosh
- Pauling

Moreover there are electronegativity scales that can be computed from their respective definition and the atomic properties available in mendeleev:

- Allred-Rochow
- Cottrell-Sutton
- Gordy
- Li and Xue
- Martynov and Batsanov
- Mulliken
- Nagle
- Sanderson

For a short overview on electronegativity see this presentation.

All the examples shown below are for Silicon:

```python
>>> from mendeleev import element
>>> Si = element('Si')
```

### 6.1 Allen

The electronegativity scale proposed by Allen in ref [2] is defined as:

\[ \chi_A = \frac{\sum_x n_x \varepsilon_x}{\sum_x n_x} \]

where: \( \varepsilon_x \) is the multiplet-averaged one-electron energy of the subshell \( x \) and \( n_x \) is the number of electrons in subshell \( x \) and the summation runs over the valence shell.

The values that are tabulated were obtained from refs. [31] and [32].
Example:

```python
>>> Si.en_allen
11.33
```
```
>>> Si.electronegativity('allen')
11.33
```

### 6.2 Allred and Rochow

The scale of Allred and Rochow [4] introduces the electronegativity as the electrostatic force exerted on the electron by the nuclear charge:

\[
\chi_{AR} = \frac{e^2 Z_{\text{eff}}}{r^2}
\]

where: \(Z_{\text{eff}}\) is the effective nuclear charge and \(r\) is the covalent radius.

Example:

```python
>>> Si.electronegativity('allred-rochow')
0.00028240190249702736
```

### 6.3 Cottrell and Sutton

The scale proposed by Cottrell and Sutton [17] is derived from the equation:

\[
\chi_{CS} = \sqrt{\frac{Z_{\text{eff}}}{r}}
\]

where: \(Z_{\text{eff}}\) is the effective nuclear charge and \(r\) is the covalent radius.

Example:

```python
>>> Si.electronegativity('cottrell-sutton')
0.18099342720014772
```

### 6.4 Ghosh

Ghosh [18] presented a scale of electronegativity based on the absolute radii of atoms computed as

\[
\chi_{GH} = a \cdot (1/R) + b
\]

where: \(R\) is the absolute atomic radius and \(a\) and \(b\) are empirical parameters.

Example:

```python
>>> Si.en_ghosh
0.178503
```
6.5 Gordy

Gordy’s scale [20] is based on the potential that measures the work necessary to achieve the charge separation, according to:

\[ \chi_G = \frac{eZ_{\text{eff}}}{r} \]

where: \( Z_{\text{eff}} \) is the effective nuclear charge and \( r \) is the covalent radius.

Example:

```python
>>> Si.electronegativity('gordy')
0.03275862068965517
```

6.6 Li and Xue

Li and Xue [27, 28] proposed a scale that takes into account different valence states and coordination environment of atoms and is calculated according to the following formula:

\[ \chi_{LX} = \frac{n^* \sqrt{I_j/Ry}}{r} \]

where: \( n^* \) is the effective principal quantum number, \( I_j \) is the \( j \)’th ionization energy in eV, \( Ry \) is the Rydberg constant in eV and \( r \) is either the crystal radius or ionic radius.

Example:

```python
>>> Si.en_li_xue(charge=4)
{u'IV': 13.16033405547733, u'VI': 9.748395596649873}
```

6.7 Martynov and Batsanov

Martynov and Batsanov [7] used the square root of the averaged valence ionization energy as a measure of electronegativity:

\[ \chi_{MB} = \sqrt{\frac{1}{n_v} \sum_{k=1}^{n_v} I_k} \]

where: \( n_v \) is the number of valence electrons and \( I_k \) is the \( k \) th ionization potential.

Example:

```python
>>> Si.en_martynov_batsanov()
5.0777041564076963
>>> Si.electronegativity(scale='martynov-batsanov')
5.0777041564076963
```
6.8 Mulliken

Mulliken scale [36] is defined as the arithmetic average of the ionization potential \((IP)\) and the electron affinity \((EA)\):

\[
\chi_M = \frac{IP + EA}{2}
\]

Example:

```python
>>> Si.en_mulliken()
4.0758415
>>> Si.electronegativity('mulliken')
4.0758415
```

6.9 Nagle

Nagle [37] derived his scale from the atomic dipole polarizability:

\[
\chi_N = \sqrt[3]{n \alpha}
\]

Example:

```python
>>> Si.electronegativity('nagle')
0.47505611644667534
```

6.10 Pauling

Pauling’s thermochemical scale was introduced in [40] as a relative scale based on electronegativity differences:

\[
\chi_A - \chi_B = \sqrt{E_d(AB) - \frac{1}{2}[E_d(AA) + E_d(BB)]}
\]

where: \(E_d(XY)\) is the bond dissociation energy of a diatomic \(XY\). The values available in mendeleev are taken from ref. [23].

Example:

```python
>>> Si.en_pauling
1.9
>>> Si.electronegativity('pauling')
1.9
```

6.11 Sanderson

Sanderson [49, 50] established his scale of electronegativity based on the stability ratio:

\[
\chi_S = \frac{\rho}{\rho_{ng}}
\]
where: \( \rho \) is the average electron density \( \rho = \frac{Z}{3\pi r^3/8} \), and \( \rho_{ng} \) is the average electron density of a hypothetical noble gas atom with charge \( Z \).

Example:

```python
>>> Si.en_sanderson()
0.3468157872145231
>>> Si.electronegativity()
0.3468157872145231
```

## 6.12 Fetching all electronegativities

If you want to fetch all the available scales for all elements you can use the `fetch_electronegativities` function, that collect all the values into a DataFrame.
Here you’ll find API documentation of the mendeleev’s modules.

<table>
<thead>
<tr>
<th>mendeleev.db</th>
<th>mendeleev.cli</th>
</tr>
</thead>
<tbody>
<tr>
<td>mendeleev.econf</td>
<td>Implementation of the abstraction for the electronic configuration object.</td>
</tr>
<tr>
<td>mendeleev.electronegativity</td>
<td>Electronegativity scale formulas.</td>
</tr>
<tr>
<td>mendeleev.fetch</td>
<td></td>
</tr>
<tr>
<td>mendeleev.mendelev</td>
<td></td>
</tr>
<tr>
<td>mendeleev.models</td>
<td>module specifying the database models</td>
</tr>
<tr>
<td>mendeleev.ion</td>
<td></td>
</tr>
<tr>
<td>mendeleev.vis.periodictable</td>
<td></td>
</tr>
<tr>
<td>mendeleev.vis.bokeh</td>
<td></td>
</tr>
<tr>
<td>mendeleev.vis.plotly</td>
<td></td>
</tr>
<tr>
<td>mendeleev.vis.seaborn</td>
<td></td>
</tr>
<tr>
<td>mendeleev.vis.utils</td>
<td></td>
</tr>
<tr>
<td>mendeleev.utils</td>
<td></td>
</tr>
</tbody>
</table>

### 7.1 mendeleev.db

#### Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_engine(dbpath)</code></td>
<td>Return the db engine</td>
</tr>
<tr>
<td><code>get_package_dbpath()</code></td>
<td>Return the default database path</td>
</tr>
<tr>
<td><code>get_session(dbpath)</code></td>
<td>Return the database session connection.</td>
</tr>
</tbody>
</table>
7.2 mendeleev.cli

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clielement()</td>
<td>CLI for convenient printing of properties for a given element</td>
</tr>
</tbody>
</table>

7.3 mendeleev.econf

Implementation of the abstraction for the electronic configuration object.

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_l(subshell)</td>
<td>Return the orbital angular momentum quantum number for a given subshell</td>
</tr>
<tr>
<td>get_spin_strings(sodict[, average])</td>
<td>Return spin strings as numpy arrays</td>
</tr>
<tr>
<td>print_spin_occupations(sodict[, average])</td>
<td>Return pretty format for the spin occupations</td>
</tr>
<tr>
<td>shell_capacity(shell)</td>
<td>Return the shell capacity (max number of electrons)</td>
</tr>
<tr>
<td>subshell_capacity(subshell)</td>
<td>Return the subshell capacity (max number of electrons)</td>
</tr>
<tr>
<td>subshell_degeneracy(subshell)</td>
<td>Return the degeneracy of a given subshell</td>
</tr>
</tbody>
</table>

Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElectronicConfiguration([conf, atomre, shellre])</td>
<td>Electronic configuration handler</td>
</tr>
</tbody>
</table>

7.4 mendeleev.electronegativity

Electronegativity scale formulas.
Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allred_rochow(zeff, radius)</td>
<td>Calculate the electronegativity of an atom according to the definition of Allred and Rochow</td>
</tr>
<tr>
<td>cottrell_sutton(zeff, radius)</td>
<td>Calculate the electronegativity of an atom according to the definition of Allred and Rochow</td>
</tr>
<tr>
<td>generic(zeff, radius[, rpow, apow])</td>
<td>Calculate the electronegativity from a general formula</td>
</tr>
<tr>
<td>gordy(zeff, radius)</td>
<td>Calculate the electronegativity of an atom according to the definition of Allred and Rochow</td>
</tr>
<tr>
<td>li_xue(ionization_energy, radius, valence_pqn)</td>
<td>Calculate the electronegativity of an atom according to the definition of Li and Xue</td>
</tr>
<tr>
<td>martynov_batsanov(ionization_energies)</td>
<td>Calculates the electronegativity value according to Martynov and Batsanov as the average of the ionization energies of the valence electrons</td>
</tr>
<tr>
<td>mulliken(ionization_energy, electron_affinity)</td>
<td>Return the absolute electronegativity (Mulliken scale).</td>
</tr>
<tr>
<td>nagle(nvalence, polarizability)</td>
<td>Calculate the electronegativity of an atom according to the definition of Nagle</td>
</tr>
<tr>
<td>sanderson(radius, noble_gas_radius)</td>
<td>Calculate Sanderson's electronegativity</td>
</tr>
</tbody>
</table>

7.5 mendeleev.fetch

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_plot_columns(elements)</td>
<td>Add columns needed for the creating the plots</td>
</tr>
<tr>
<td>fetch_electronegativities([scales])</td>
<td>Fetch electronegativity scales for all elements as pandas.DataFrame</td>
</tr>
<tr>
<td>fetch_ionic_radii([radius])</td>
<td>Fetch a pandas DataFrame with ionic radii for all the elements.</td>
</tr>
<tr>
<td>fetch_ionization_energies([degree])</td>
<td>Fetch a pandas DataFrame with ionization energies for all elements indexed by atomic number.</td>
</tr>
<tr>
<td>fetch_neutral_data()</td>
<td>Get extensive set of data from multiple database tables as pandas.DataFrame</td>
</tr>
<tr>
<td>fetch_table(table, **kwargs)</td>
<td>Return a table from the database as pandas.DataFrame</td>
</tr>
<tr>
<td>get_app_data()</td>
<td>write a file with the neutral data</td>
</tr>
<tr>
<td>get_zeff(an[, method])</td>
<td>A helper function to calculate the effective nuclear charge.</td>
</tr>
</tbody>
</table>
7.6 mendeleev.mendeleev

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deltaN(id1, id2[, charge1, charge2, ...])</td>
<td>Calculate the approximate fraction of transferred electrons between elements or ions $id1$ and $id2$ with charges $charge1$ and $charge2$ respectively according to the expression $\text{element}(ids)$ based on the type of the $ids$ identifier return either an Element object from the database, or a list of Element objects if the $ids$ is a list or a tuple of identifiers.</td>
</tr>
<tr>
<td>element(ids)</td>
<td>Based on the type of the $ids$ identifier return either an Element object from the database, or a list of Element objects if the $ids$ is a list or a tuple of identifiers.</td>
</tr>
<tr>
<td>get_all_elements()</td>
<td>Get all elements as a list of Element objects.</td>
</tr>
<tr>
<td>ids_to_attr(ids[, attr])</td>
<td>Convert the element ids: atomic numbers, symbols, element names or a combination of the above to a list of corresponding attributes.</td>
</tr>
<tr>
<td>isotope(symbol_or_atn, mass_number)</td>
<td>Get an Isotope based on the element symbol and mass number number or atomic number and mass number.</td>
</tr>
</tbody>
</table>

7.7 mendeleev.models

module specifying the database models

Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate_from_group(atomic_number, attr_name)</td>
<td>Evaluate a value $attribute$ for element by interpolation or extrapolation of the data points from elements from $group$.</td>
</tr>
<tr>
<td>fetch_attrs_for_group(attrs[, group])</td>
<td>A convenience function for getting a specified attribute for all the members of a given group.</td>
</tr>
<tr>
<td>with_uncertainty(value, uncertainty[, digits])</td>
<td>Format a value with uncertainty using scientific notation.</td>
</tr>
</tbody>
</table>
### Classes

<table>
<thead>
<tr>
<th><strong>Classes</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Element(**kwargs)</code></td>
<td>Chemical element.</td>
</tr>
<tr>
<td><code>Group(**kwargs)</code></td>
<td>Name of the group in the periodic table.</td>
</tr>
<tr>
<td><code>IonicRadius(**kwargs)</code></td>
<td>Effective ionic radii and crystal radii in pm retrieved from [1]_.</td>
</tr>
<tr>
<td><code>IonizationEnergy(**kwargs)</code></td>
<td>Ionization energy of an element</td>
</tr>
<tr>
<td><code>Isotope(**kwargs)</code></td>
<td><strong>param abundance</strong> Abundance of the isotope</td>
</tr>
<tr>
<td><code>IsotopeDecayMode(**kwargs)</code></td>
<td><strong>param mode</strong> ASCII symbol for the decay mode</td>
</tr>
<tr>
<td><code>OxidationState(**kwargs)</code></td>
<td>Oxidation states of an element</td>
</tr>
<tr>
<td><code>PhaseTransition(**kwargs)</code></td>
<td>Phase Transition Conditions</td>
</tr>
<tr>
<td><code>ScreeningConstant(**kwargs)</code></td>
<td>Nuclear screening constants from Clementi, E., &amp; Raimondi, D.</td>
</tr>
<tr>
<td><code>Series(**kwargs)</code></td>
<td>Name of the series in the periodic table.</td>
</tr>
</tbody>
</table>

#### 7.8 mendeleev.ion

**Classes**

- `Ion(label[, q])`: Class representing atomic ions

#### 7.9 mendeleev.vis.periodictable

**Functions**

- `periodic_table([attribute, height, width, ...])`: High level API for visualizing periodic tables.

#### 7.10 mendeleev.vis.bokeh

**Functions**

- `periodic_table_bokeh(elements[, attribute, ...])`: Use Bokeh backend to plot the periodic table.
### 7.11 mendeleev.vis.plotly

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create_annotation(row, attr[, size, ...])</code></td>
<td>Create an annotation from pandas series</td>
</tr>
<tr>
<td><code>create_tile(element, color[, x_offset, y_offset])</code></td>
<td>Create tile shape</td>
</tr>
<tr>
<td><code>periodic_table_plotly(elements[, attribute, ...])</code></td>
<td>Create a periodic table visualization with plotly.Figure</td>
</tr>
</tbody>
</table>

### 7.12 mendeleev.vis.seaborn

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heatmap(elements, prop[, style, figsize, ...])</code></td>
<td>Plot a heatmap of the given property</td>
</tr>
</tbody>
</table>

### 7.13 mendeleev.vis.utils

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_tile_coordinates(df[, x_coord, y_coord, ...])</code></td>
<td>Calculate coordinates for the tile centers</td>
</tr>
<tr>
<td><code>colormap_column(elements, column[, cmap, ...])</code></td>
<td>Return a new DataFrame with the same size (and index) as <code>elements</code> with a column <code>cmap</code> containing HEX color mapping from <code>cmap</code> colormap.</td>
</tr>
<tr>
<td><code>create_vis_dataframe([x_coord, y_coord, ...])</code></td>
<td>Base DataFrame for visualizations</td>
</tr>
</tbody>
</table>

### 7.14 mendeleev.utils

**Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>coeffs(a[, b])</code></td>
<td>Return stoichometric coefficients from oxidation states</td>
</tr>
<tr>
<td><code>n_effective(n[, source])</code></td>
<td>Effective principal quantum number</td>
</tr>
</tbody>
</table>
MENDELEEV CHANGELG

9.1 v0.14.0 (07.06.2023)

• Fix Mulliken electronegativity by @lmmentel in #116
• [FIX] Enable fetch of phase transition data by @lmmentel in #112

9.2 v0.13.1 (24.04.2023)

• Fix URL in references.bib by @paulromano in #108
• Fix import warning for declarative_base by @lmmentel in #109
• Add vis extra by @lmmentel in #110

9.3 v0.13.0 (11.04.2023)

• [MNT] Relax dependencies for sqlalchemy and pandas and drop python 3.7 by @lmmentel in #103
• Bump ipython from 7.34.0 to 8.10.0 by @dependabot in #104
• [MNT] Add API docs for vis module by @lmmentel in #105

9.4 v0.12.1 (28.11.2022)

• Add CodeQL workflow for GitHub code scanning by @lgtm-com in #89
• Fix number of valence electrons (#91) for Pd by @lmmentel in #92
• Add missing type hints by @lmmentel in #93
9.5 v0.12.0 (9.10.2022)

- Configure concurrency in github actions by @lmmentel in #82
- Fix abundancies for isotopes with one naturally occurring isotope by @lmmentel in #80
- Add IsotopeDecayMode model and data by @lmmentel in #84
- Update boiling and melting point data and add triple point and critical temperature and pressure, by @lmmentel in #88
- Include compatibility with python 3.11.

9.6 v0.11.0 (29.09.2022)

- Update data.rst by @Eben60 in #66
- Set discovery_location for Zinc to null by @lmmentel in #68
- Change “Oxidation states” to “Commonly occurring oxidation states” by @Eben60 in #69
- Add International Chemical Identifier property by @lmmentel in #76
- Update data for isotopes by @lmmentel in #74
- Update oxidation states and add method to fetch values by @lmmentel in #77
- Documentation fixes by @lmmentel in #78

9.7 v0.10.0 (17.07.2022)

- Corrected specific heat capacity values with CRC Handbook of Chemistry and Physics as the data source Issue #60
- Renamed specific_heat attribute to specific_heat_capacity PR #61 (for backwards compatibility specific_heat will still work)
- Added molar_heat_capacity property from CRC Handbook of Chemistry and Physics PR #61
- Corrected wrong units in the docs for specific_heat Issue #59
- Fixed usage of pytest.approx after api change PR #62
- Refactored format call to f-strings PR #62
- Updated locked dependencies to eliminate known vulnerabilities PR #63
- Added python 3.10 to CI workflows to increase test coverage PR #62
9.8 v0.9.0 (24.09.2021)

- Correct density data with *CRC Handbook of Chemistry and Physics* as the data source PR #39 that fixes issue #38.
- Fixed plotly based visualizations not rendering at https://mendeleev.readthedocs.io.
- Added DOI number.

9.9 v0.8.0 (22.08.2021)

- Enable visualizations of periodic tables with plotly as well as bokeh backends through mendeleev.vis.plotly.periodic_table_plotly and mendeleev.vis.bokeh.periodic_table_bokeh functions.
- Add mendeleev.vis.periodic_table function for convenient periodic table plotting wrapping both plotting backends.
- Refactored the mendeleev.vis module so it can be wasily extended with plotting backends.
- Add CITATION.cff file.

9.10 v0.7.0 (20.03.2021)

- Update ionic and crytal radii for III+ actinoids.
- Refactor electronegativity calculations for easier calculation and retrieval of the different scales.
- Add fetch.py module with methods for accessing bulk data.
- Add oxides methods to Element that returns possible oxides (Issue #17).
- Add tutorials on fetching data and electronic configuration.
- tables.py is renamed to models.py.
- Switch from pipenv to poetry for development.
- Switch from travis CI to github actions and extend testing matrix to Win and MacOS.
- Documentation udpate.

9.11 v0.6.1 (03.11.2020)

- Add electrophilicity index.
- Pin sqlalchemy version to prevent further issues with old versions, see Issue #22
9.12 v0.6.0 (10.04.2020)

- Add `Ion` class to handle atomic ions.
- Add Github templates for bug reports, feature requests and pull requests.
- Update the values of `atomic_radius_rahm` according to corrigendum, (PR #13).
- Switch the default documentation theme to material with `sphinx-material`.

9.13 v0.5.2 (29.01.2020)

- Fix a `UnicodeDecodeError` from README.md while installing on windows.
- Code quality improvements based on lgtm.com

9.14 v0.5.1 (26.08.2019)

- Fix issue #3, `get_table('elements')` throwing an error

9.15 v0.5.0 (25.08.2019)

- Migrate the package from bitbucket to github
- Add Pettifor scale: `pettifor_number` attribute
- Add Glawe scale: `glake_number` attribute
- Restore default printing of isotopic abundancies, fix issue #9
- Correct the oxidation states for Xe, fix issue #10

9.16 v0.4.5 (17.03.2018)

- Update dipole polarizability value to the latest recommended (2018)
- Fix issues/8/typeerror-on-some-of-the-element

9.17 v0.4.4 (10.12.2018)

- Fix issues/6/type-of-block-is-wrong
9.18  v0.4.3 (16-07-2018)

- Added mendeleev_number attribute to elements.
- Added footnotes to the data documentation.

9.19  v0.4.2 (26-12-2018)

- Fixed issue #3: encoding issue in econf.py.

9.20  v0.4.1 (03-12-2017)

- Corrected passing integers to the CLI script.
- Various documentation readability and structure improvements.

9.21  v0.4.0 (22-11-2017)

- The elements can now be directly imported from mendeleev by symbols.
- Added sphinxcontrib.bibtex extension to the docs to handle BibTeX style references to improve handling of the bibliographic entries.
- Added nbsphinx to include Jupyter Notebook tutorials in the docs.

9.22  v0.3.6 (17-09-2017)

- Added API documentation
- Corrected the sphinx configuration
- Updated the documentation

9.23  v0.3.5 (07-09-2017)

- Added a module with functions to scrape data from ciaaw.org
- Added new Element attributes, name_origin, uses and sources
- Added new Element attributes related to the discovery: discoverers, discovery_location, discovery_year
9.24 v0.3.4 (28-06-2017)

• Fixed python2.7 compatibility issue
• Added double and triple bond covalent radii from Pyykko
• Corrected minor error in the documentation
• Replaced lazy loading with eager in db queries

9.25 v0.3.3 (16-05-2017)

• Corrected the coordination of Br5+ ion in the ionic radii table

9.26 v0.3.2 (01-05-2017)

• Added metallic_radius
• Added Goldschmidt and geochemical classifications
• Corrected the docs configuration
• Added cas number attribute
• Added atomic radii by Rahm et al.
• Created a conda recipe
• Added a citation information to the readme
• Electronic configuration code was split into a separate module

9.27 v0.3.1 (25-01-2017)

• Added new properties of isotopes: spin, g_factor, quadrupole_moment

9.28 v0.3.0 (09-01-2017)

• Updates of the documentation and tutorials
• Added radioactive isotope half-lifes
9.29 v0.2.17 (08-01-2017)

• Extended the schema for isotopes with additional attributes and updated the values of abundancies, half lives and mass uncertainties.
• Updates to the tutorials and docs.

9.30 v0.2.16 (06-01-2017)

• Corrected the radioactive attribute of Th, Pa and U elements.

9.31 v0.2.15 (02-01-2017)

• Patched the sphinx configuration.

9.32 v0.2.14 (02-01-2017)

• Patched typos in README.

9.33 v0.2.13 (01-01-2017)

• Updated atomic weight with the newest IUPAC and CIAAW recommendations.
• Added is_radioactive and is_monoisotopic attributes.
• Updated the docs.

9.34 v0.2.12 (21-12-2016)

• Got rid of the scipy dependency.

9.35 v0.2.11 (10-11-2016)

• Updated the names and symbols of elements 113, 115, 117, 118.
• Updated the docs.
9.36 v0.2.10 (18-10-2016)

- Added the C6 coefficients from Gould and Bucko.
- Added van der Waals radii from Alvarez.

9.37 v0.2.9 (16-10-2016)

- Added a scale of electronegativities by Ghosh.

9.38 v0.2.8 (29-08-2016)

- Updated the electron affinity of Pb and Co.
- Updates of the docs.

9.39 v0.2.7 (02-04-2016)

- Maintenance.

9.40 v0.2.6 (02-04-2016)

- Mainly maintenance updates to docs, sphinx conf.py, setup.py, requirements.

9.41 v0.2.5 (02-04-2016)

### 9.41.1 Features added

- Added calculation of Martynov and Batsanov scale of electronegativity in en_martynov_batsanov method in the Element class
- Added abundance_crust and abundance_sea with element abundancies in the crust and seas
- Added molcas_gv_color attribute with MOLCAS GV colors

### 9.41.2 Bugs fixed

- Restored Python 3.x compatibility
9.42 v0.2.4 (05-02-2016)

9.42.1 Features added

- Extended and corrected the documentation and Jupyter notebook tutorials on basic usage electronegativities, plotting and tables

9.42.2 Bugs fixed

- Corrected raise to return when calling en_sanderson from electronegativity
- Fixed and tested the formula for calculating the Li and Xue scale of electronegativity in en_lie-xue

9.43 v0.2.3 (27-01-2016)

9.43.1 Features added

- Added new vdW radii: vdw_radius_batsanov, vdw_radius_bondi, vdw_radius_dreiding, vdw_radius_mm3, vdw_radius_rt, vdw_radius_truhlar, vdw_radius_uff
- Added an option to plot the long (wide) version of the periodic table in periodic_plot

9.43.2 Bugs fixed

- Typos in the docstrings

9.44 v0.2.2 (29-11-2015)

9.44.1 Features added

- Added new covalent radii: covalent_radius_bragg, covalent_radius_slater
- Added the c6 dispersion coefficients
- Added gas_basicity, proton_affinity and heat_of_formation
- Added periodic_plot function for producing bokeh <https://bokeh.org/> based plots of the periodic table
- Added jmol_color and cpk_color with different coloring schemes for atoms
9.44.2 Bug fixes

• Changed the series of elements 113, 114, 115, 116 to poor metals

9.45 v0.2.1 (26-10-2015)

9.45.1 Features added

• Extended the list of options for calculating Mulliken electronegativities in en_mulliken
• Added electrons_per_shell method
• Added a function to calculate linear interpolation of radii required for calculation of Sandersons electronegativity
• Added hybrid attributes electrons, protons, neutrons and mass_number

9.45.2 Bug fixes

• Changed the type of the melting_point from str to float

9.46 v0.2.0 (22-10-2015)

9.46.1 Features added

• Instead of covalent_radius added covalent_radius_2008 and covalent_radius_2009
• Instead of electronegativity added en_pauling and en_mulliken
• Added a method for getting ionic radii
• Improved the method for calculating the nuclear screening constants
• Added ElectronicConfiguration class initialized as Element attribute
• Added nuclear screening constants from Clementi and Raimondi
• Added a method to calculate the absolute softness, absolute hardness and absolute electronegativity
• Added get_table method to retrieve the tables as pandas DataFrames

9.46.2 Bug fixes

• Added missing electronic configurations
• Converted ionic radii from Angstrom to pico meters
9.47 v0.1.0 (11-07-2015)

First tagged version with the initial structure of the package and first version of the database and the python interface
The MIT License (MIT)

Copyright (c) 2015 Lukasz Mentel

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
CHAPTER

ELEVEN

INDICES AND TABLES

• genindex
• modindex
• search


Bibliography
mendeleev.cli, 50
mendeleev.db, 49
mendeleev.econf, 50
mendeleev.electronegativity, 50
mendeleev.fetch, 51
mendeleev.ion, 53
mendeleev.mendeleev, 52
mendeleev.models, 52
INDEX

F
fetch_electronegativities() (in module mendeleev.fetch), 40
fetch_ionic_radii() (in module mendeleev.fetch), 40
fetch_ionization_energies() (in module mendeleev.fetch), 40
fetch_table() (in module mendeleev.fetch), 39

G
get_engine() (in module mendeleev.db), 41
get_session() (in module mendeleev.db), 41

M
mendeleev.cli module, 50
mendeleev.db module, 49
mendeleev.ecnf module, 50
mendeleev.electronegativity module, 50
mendeleev.fetch module, 51
mendeleev.ion module, 53
mendeleev.mendeleev module, 52
mendeleev.models module, 52
mendeleev.utils module, 54
mendeleev.vis.bokeh module, 53
mendeleev.vis.periodictable module, 53
mendeleev.vis.plotly module, 54
mendeleev.vis.seaborn module, 54
mendeleev.vis.utils module, 54
module