mendeleev Documentation

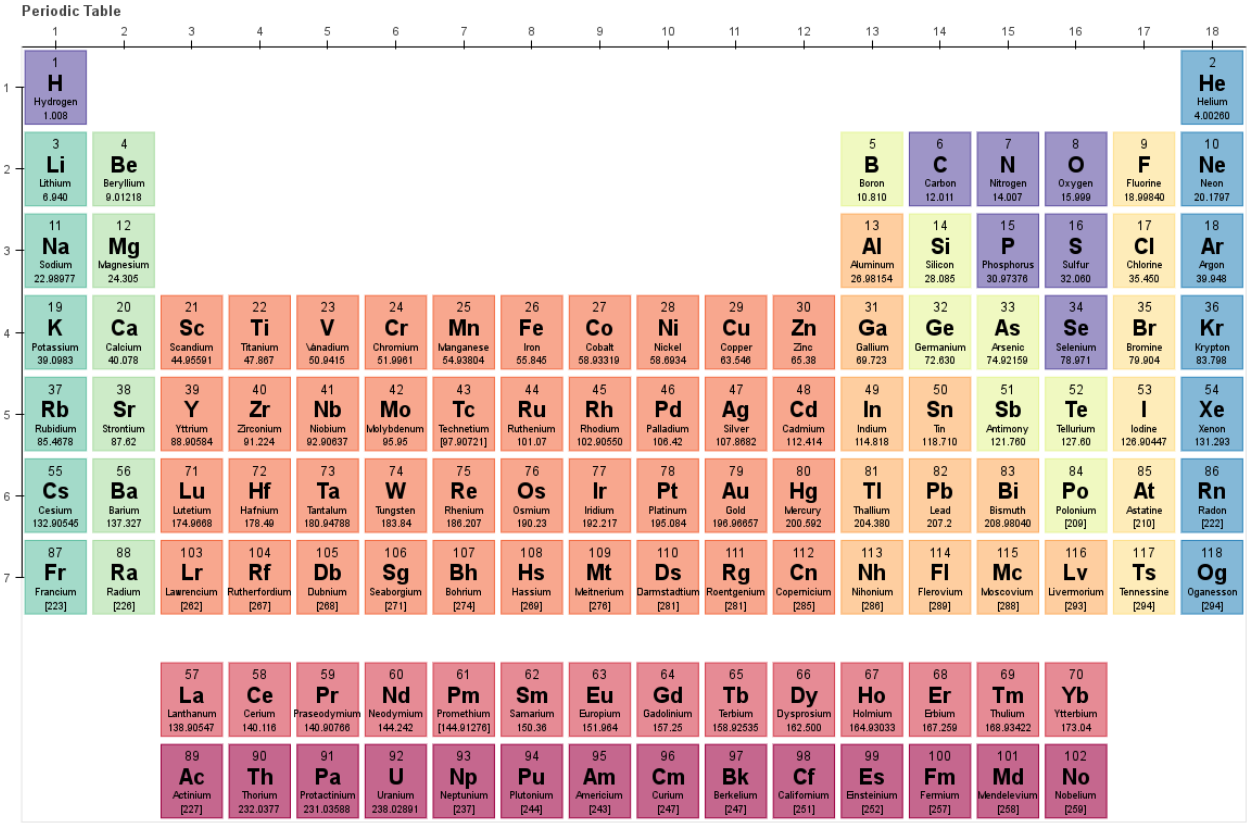
Release 0.14.0

Lukasz Mentel

Jun 07, 2023

CONTENTS

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



CHAPTER

ONE

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

L. M. Mentel, *mendeleev* - A Python resource for properties of chemical elements, ions and isotopes. , 2014– . Available at: https://github.com/lmmentel/mendeleev.

Here's the reference in the BibLaTeX format

```
@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

```
@misc{mendeleev2014,
   auhor = {Mentel, Łukasz},
   title = {mendeleev} -- A Python resource for properties of chemical elements, ions␣
↪and isotopes, ver. 0.14.0},
   howpublished = {\url{https://github.com/lmmentel/mendeleev}},
   year  = {2014--},
}
```

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.

CHAPTER

TWO

INSTALLATION

The package can be installed using pip

```
pip install mendeleev
```

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

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

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

```
conda install -c lmmentel mendeleev=0.14.0
```

CHAPTER

THREE

TUTORIALS

3.1 Quick start

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

3.1.1 Table of Contents

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

[2]:
```
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

[3]:
```
si = element('Si')
```

[4]:
```
si
```

[4]:
```
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)

(continued from previous 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=25.9923338(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=30.97536320(5), abundance=None)>, <Isotope(Z=14, A=32, mass=31.9741515(3),␣
↪abundance=None)>, <Isotope(Z=14, A=33, mass=32.9779770(8), 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,
```

(continues on next page)

(continued from previous page)

```
        metallic_radius=117.0,
        metallic_radius_c12=138.0,
        molar_heat_capacity=19.99,
        molcas_gv_color='#f0c8a0',
        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=    4.0550)>,
↪<ScreeningConstant(Z=  14, n=  2, s=s, screening=    4.9800)>, <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

[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

[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

[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

[ ]:

[10]:
```
print("{0:^4s} {1:^4s} {2:^10s} {3:8s} {4:6s} {5:5s}\n{6}".format("AN", "MN", "Mass",␣
↪"Unc.", "Abu.", "Rad.", "-" * 42))
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))
```

```
 AN   MN     Mass    Unc.     Abu.   Rad.
------------------------------------------
  26   45   45.01547 3.04e-04 None True
  26   46   46.00130 3.22e-04 None True
  26   47   46.99235 5.37e-04 None True
  26   48   47.98067 9.90e-05 None True
  26   49   48.97343 2.60e-05 None True
  26   50   49.96299 9.00e-06 None True
  26   51   50.95686 1.50e-06 None True
  26   52   51.94811 1.92e-07 None True
  26   53   52.94531 1.79e-06 None True
  26   54   53.93961 3.68e-07 5.845 False
  26   55   54.93829 3.30e-07 None True
  26   56   55.93494 2.87e-07 91.754 False
  26   57   56.93539 2.87e-07 2.119 False
```

(continues on next page)

(continued from previous page)

```
  26   58   57.93327 3.39e-07 0.282 False
  26   59   58.93487 3.54e-07 None True
  26   60   59.93407 3.66e-06 None True
  26   61   60.93675 2.80e-06 None True
  26   62   61.93679 3.00e-06 None True
  26   63   62.94027 4.62e-06 None True
  26   64   63.94099 5.39e-06 None True
  26   65   64.94502 5.49e-06 None True
  26   66   65.94625 4.40e-06 None True
  26   67   66.95093 4.10e-06 None True
  26   68   67.95288 2.07e-04 None True
  26   69   68.95792 2.15e-04 None True
  26   70   69.96040 3.22e-04 None True
  26   71   70.96572 4.29e-04 None True
  26   72   71.96860 5.37e-04 None True
  26   73   72.97425 5.37e-04 None True
  26   74   73.97782 5.37e-04 None True
  26   75   74.98422 6.44e-04 None True
  26   76   75.98863 6.44e-04 None True
```

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` attrobute 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)

[15]:
```
for ir in fe.ionic_radii:
    print(ir)
```

```
charge=   2, coordination=IV   , crystal_radius=77.000, ionic_radius=63.000
charge=   2, coordination=IVSQ , crystal_radius=78.000, ionic_radius=64.000
charge=   2, coordination=VI   , crystal_radius=75.000, ionic_radius=61.000
charge=   2, coordination=VI   , crystal_radius=92.000, ionic_radius=78.000
charge=   2, coordination=VIII , crystal_radius=106.000, ionic_radius=92.000
charge=   3, coordination=IV   , crystal_radius=63.000, ionic_radius=49.000
charge=   3, coordination=V    , crystal_radius=72.000, ionic_radius=58.000
charge=   3, coordination=VI   , crystal_radius=69.000, ionic_radius=55.000
charge=   3, coordination=VI   , crystal_radius=78.500, ionic_radius=64.500
charge=   3, coordination=VIII , crystal_radius=92.000, ionic_radius=78.000
charge=   4, coordination=VI   , crystal_radius=72.500, ionic_radius=58.500
charge=   6, coordination=IV   , crystal_radius=39.000, ionic_radius=25.000
```

3.1.4 Useful functions for calculating properties

Next to stored attributes there is a number of useful functions

[16]:
```
si = element('Si')
```

[17]:
```
# get the number of valence electrons
si.nvalence()
```

[17]:
```
4
```

[18]:
```
# calculate softness for an ion
si.softness(charge=2)
```

[18]:
```
0.058318712346158874
```

[19]:
```
# calcualte hardness for an ion
si.hardness(charge=4)
```

[19]:
```
60.812605
```

[20]:
```
# calculate the effective nuclear charge for a subshell using Slater's rules
si.zeff(n=3, o='s')
```

[20]:
```
4.149999999999999
```

[21]:
```
# calculate the effective nuclear charge for a subshell using Clemneti's and Raimondi's␣
↪exponents
si.zeff(n=3, o='s', method='clementi')
```

[21]:
```
4.9032
```

Electronegativity

Currently there are 9 electronagativity scales implemented that can me accessed though 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.

[22]:
```
si.electronegativity(scale='pauling')
```

[22]:
```
1.9
```

[23]:
```
si.electronegativity(scale='allen')
```

[23]:
```
11.33
```

[24]:
```
# calculate mulliken electronegativity for a neutral atom or ion
si.electronegativity(scale="mulliken", charge=1)
```

[24]:
```
12.248764000000001
```

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

[25]:
```
!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`

[1]:
```
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

[3]:
```
ptable = fetch_table('elements')
```

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

[4]:
```
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)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
 56  metallic_radius                56 non-null     float64
 57  metallic_radius_c12            63 non-null     float64
 58  covalent_radius_pyykko_double  108 non-null    float64
 59  covalent_radius_pyykko_triple  80 non-null     float64
 60  discoverers                    118 non-null    object
 61  discovery_year                 105 non-null    float64
 62  discovery_location             106 non-null    object
 63  name_origin                    118 non-null    object
 64  sources                        118 non-null    object
 65  uses                           112 non-null    object
 66  mendeleev_number               118 non-null    int64
 67  dipole_polarizability_unc      117 non-null    float64
 68  pettifor_number                103 non-null    float64
 69  glawe_number                   103 non-null    float64
dtypes: bool(1), float64(46), int64(4), object(19)
memory usage: 63.8+ KB
```

For clarity let's take only a subset of columns

[5]:
```
cols = ['atomic_number', 'symbol', 'atomic_radius', 'en_pauling', 'block',␣
↪'vdw_radius_mm3']
```

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

[6]:

	atomic_number	symbol	atomic_radius	en_pauling	block	vdw_radius_mm3
0	1	H	25.0	2.20	s	162.0
1	2	He	120.0	NaN	s	153.0
2	3	Li	145.0	0.98	s	255.0
3	4	Be	105.0	1.57	s	223.0
4	5	B	85.0	2.04	p	215.0

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

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

[7]:

	atomic_number	atomic_radius	en_pauling	vdw_radius_mm3
count	118.000000	90.000000	85.000000	94.000000
mean	59.500000	149.844444	1.748588	248.468085
std	34.207699	40.079110	0.634442	36.017828
min	1.000000	25.000000	0.700000	153.000000
25%	30.250000	135.000000	1.240000	229.000000
50%	59.500000	145.000000	1.700000	244.000000
75%	88.750000	178.750000	2.160000	269.250000
max	118.000000	260.000000	3.980000	364.000000

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)

(continued from previous page)

```
 9   mass_uncertainty   377 non-null    float64
 10  is_radioactive     406 non-null    bool
 11  half_life          121 non-null    float64
 12  half_life_unit     85 non-null     object
 13  spin               323 non-null    float64
 14  g_factor           323 non-null    float64
 15  quadrupole_moment  320 non-null    float64
dtypes: bool(1), float64(10), int64(2), object(3)
memory usage: 51.1+ KB
```

[13]:
```
merged.head()
```

[13]:
```
   atomic_number symbol  atomic_radius  en_pauling block  vdw_radius_mm3  \
0              1      H           25.0         2.2     s           162.0
1              1      H           25.0         2.2     s           162.0
2              1      H           25.0         2.2     s           162.0
3              2     He          120.0         NaN     s           153.0
4              2     He          120.0         NaN     s           153.0

       mass  abundance  mass_number  mass_uncertainty  is_radioactive  \
0  1.007825   0.999720            1      6.000000e-10           False
1  2.014102   0.000280            2      8.000000e-10           False
2       NaN        NaN            3               NaN            True
3  3.016029   0.000002            3      2.000000e-08           False
4  4.002603   0.999998            4      4.000000e-10           False

   half_life half_life_unit  spin  g_factor  quadrupole_moment
0        NaN            None   0.5  5.585695            0.00000
1        NaN            None   1.0  0.857438            0.00286
2        NaN            None   0.5  5.957994            0.00000
3        NaN            None   0.5 -4.254995            0.00000
4        NaN            None   0.0  0.000000            0.00000
```

To display all the isotopes of Silicon

[14]:
```
merged[merged['symbol'] == 'Si']
```

[14]:
```
    atomic_number symbol  atomic_radius  en_pauling block  vdw_radius_mm3  \
28             14     Si          110.0         1.9     p           229.0
29             14     Si          110.0         1.9     p           229.0
30             14     Si          110.0         1.9     p           229.0

         mass  abundance  mass_number  mass_uncertainty  is_radioactive  \
28  27.976927    0.92191           28      3.000000e-09           False
29  28.976495    0.04699           29      3.000000e-09           False
30  29.973770    0.03110           30      2.000000e-08           False

    half_life half_life_unit  spin  g_factor  quadrupole_moment
28        NaN            None   0.0   0.00000                0.0
29        NaN            None   0.5  -1.11058                0.0
30        NaN            None   0.0   0.00000                0.0
```

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.

[2]:
```
from mendeleev.fetch import fetch_ionic_radii
```

[3]:
```
irs = fetch_ionic_radii(radius="ionic_radius")
irs.head(10)
```

[3]:
```
coordination             I     II    III  IIIPY     IV  IVPY  IVSQ  IX   V  \
atomic_number charge
1                1     -38.0  -18.0    NaN    NaN    NaN   NaN   NaN NaN NaN
3                1       NaN    NaN    NaN    NaN   59.0   NaN   NaN NaN NaN
4                2       NaN    NaN   16.0    NaN   27.0   NaN   NaN NaN NaN
5                3       NaN    NaN    1.0    NaN   11.0   NaN   NaN NaN NaN
6                4       NaN    NaN   -8.0    NaN   15.0   NaN   NaN NaN NaN
7               -3       NaN    NaN    NaN    NaN  146.0   NaN   NaN NaN NaN
                 3       NaN    NaN    NaN    NaN    NaN   NaN   NaN NaN NaN
                 5       NaN    NaN  -10.4    NaN    NaN   NaN   NaN NaN NaN
8               -2       NaN  135.0  136.0    NaN  138.0   NaN   NaN NaN NaN
9               -1       NaN  128.5  130.0    NaN  131.0   NaN   NaN NaN NaN

coordination               VI  VII   VIII   X  XI  XII  XIV
atomic_number charge
1                1        NaN  NaN    NaN NaN NaN  NaN  NaN
3                1       76.0  NaN   92.0 NaN NaN  NaN  NaN
4                2       45.0  NaN    NaN NaN NaN  NaN  NaN
5                3       27.0  NaN    NaN NaN NaN  NaN  NaN
6                4       16.0  NaN    NaN NaN NaN  NaN  NaN
7               -3        NaN  NaN    NaN NaN NaN  NaN  NaN
                 3       16.0  NaN    NaN NaN NaN  NaN  NaN
                 5       13.0  NaN    NaN NaN NaN  NaN  NaN
8               -2      140.0  NaN  142.0 NaN NaN  NaN  NaN
9               -1      133.0  NaN    NaN NaN NaN  NaN  NaN
```

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.

[17]:
```
from mendeleev.fetch import fetch_ionization_energies
```

[25]:
```
ies = fetch_ionization_energies(degree=2)
ies.head(10)
```

[25]:
```
                     IE2
atomic_number
1                    NaN
2              54.417763
3              75.640094
4              18.211153
5              25.154830
```

(continues on next page)

(continued from previous page)

```
6              24.384500
7              29.601250
8              35.121110
9              34.970810
10             40.962960
```

[24]:
```
ies_multiple = fetch_ionization_energies(degree=[1, 3, 5])
ies_multiple.head(10)
```

[24]:
```
                     IE1         IE3         IE5
atomic_number
1              13.598434         NaN         NaN
2              24.587388         NaN         NaN
3               5.391715  122.454354         NaN
4               9.322699  153.896198         NaN
5               8.298019   37.930580  340.226008
6              11.260296   47.887780  392.090500
7              14.534130   47.445300   97.890130
8              13.618054   54.935540  113.898900
9              17.422820   62.708000  114.249000
10             21.564540   63.423310  126.247000
```

3.2.5 Electronegativities

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

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

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

[27]:
```
                 Allen  Allred-Rochow  Cottrell-Sutton     Ghosh     Gordy  \
atomic_number
1               13.610       0.000977         0.176777  0.263800  0.031250
2               24.590       0.000803         0.192241  0.442712  0.036957
3                5.392       0.000073         0.098866  0.105093  0.009774
4                9.323       0.000187         0.138267  0.144986  0.019118
5               12.130       0.000360         0.174895  0.184886  0.030588
6               15.050       0.000578         0.208167  0.224776  0.043333
7               18.130       0.000774         0.234371  0.264930  0.054930
8               21.360       0.001146         0.268742  0.304575  0.072222
9               24.800       0.001270         0.285044  0.344443  0.081250
10              28.310       0.001303         0.295488  0.384390  0.087313

                                                        Li-Xue  \
atomic_number
1               {('I', ''): -3.540721753312244, ('II', ''): -2...
2                                                           {}
3               {('IV', ''): 1.7160634314550876, ('VI', ''): 1...
4                                                           {}
```

(continues on next page)

(continued from previous page)

```
5                                                                {}
6                                                                {}
7                                                                {}
8                                                                {}
9                                                                {}
10                                                               {}

               Martynov-Batsanov   Mulliken     Nagle  Pauling  Sanderson
atomic_number
1                       3.687605   6.799217  0.605388     2.20   2.187771
2                       6.285107  12.293694  1.130639      NaN   1.000000
3                       2.322007   2.695857  0.182650     0.98   0.048868
4                       3.710381   4.661350  0.375615     1.57   0.126847
5                       4.877958   4.149010  0.526974     2.04   0.254627
6                       6.083300   5.630148  0.707393     2.55   0.427525
7                       7.306768   7.267065  0.877498     3.04   0.577482
8                       8.496136   6.809027  1.042218     3.44   0.941649
9                       9.701808   8.711410  1.232373     3.98   1.017681
10                     10.918389        NaN  1.443255      NaN   1.000000
```

3.3 Electronic configuration

`ec` attribute is an object from the `ElectronicConfiguration` class that has additional method 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

[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 access supplying a proper key

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

[3]:
```
2
```

to calculate the number of electrons per shell type

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

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

get the largest value of the pricipal quantum number

[5]:
```
Si.ec.max_n()
```
[5]:
```
3
```

Get the largest value of azimutal quantum number for a given value of principal quantum number

[6]:
```
Si.ec.max_l(n=3)
```
[6]:
```
'p'
```

Find the large noble gas-like core configuration

[7]:
```
Si.ec.get_largest_core()
```
[7]:
```
('Ne', <ElectronicConfiguration(conf="1s2 2s2 2p6")>)
```

Get the total number of electrons

[8]:
```
Si.ec.ne()
```
[8]:
```
14
```

Last subshell

[9]:
```
Si.ec.last_subshell()
```
[9]:
```
((3, 'p'), 2)
```

Get unpaired electrons

[10]:
```
Si.ec.unpaired_electrons()
```
[10]:
```
2
```

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

[11]:
```
ionized = Si.ec.ionize()
print(ionized)
```
```
1s2 2s2 2p6 3s2 3p1
```

We can check that it actually has less electrons:

[12]:
```
ionized.ne()
```
[12]:
```
13
```

Spin occupations by subshell

[13]:
```
Si.ec.spin_occupations()
```
[13]:
```
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 spin only magnetic moment

```
[14]: Si.ec.spin_only_magnetic_moment()
[14]: 2.8284271247461903
```

Calculate the screening constant using Slater's rules for `2s` orbital

```
[15]: Si.ec.slater_screening(n=2, o='s')
[15]: 4.1499999999999995
```

3.3.1 Standalone use

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

```
[16]: from mendeleev.econf import ElectronicConfiguration
```

```
[17]: ec = ElectronicConfiguration("1s2 2s2 2p6 3s1")
```

Get the valence only configuration

```
[18]: ec.get_valence()
[18]: <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.

```
[1]: from mendeleev.ion import Ion
```

```
[2]: fe_2 = Ion("Fe", 2)
```

You can access variety of properties of the ion

```
[3]: fe_2.charge
[3]: 2
```

```
[4]: fe_2.electrons
[4]: 24
```

```
[5]: fe_2.Z
[5]: 26
```

```
[6]: fe_2.name
[6]: 'Iron 2+ ion'
```

you can also print the unicode ion symbol

[7]:
```
fe_2.unicode_ion_symbol()
```

[7]:
```
'Fe²'
```

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, ',
```

(continues on next page)

(continued from previous page)

```
        spin='HS',
),
IonicRadius(
        atomic_number=26,
        charge=2,
        coordination='VIII',
        crystal_radius=106.0,
        econf='3d6',
        id=153,
        ionic_radius=92.0,
        most_reliable=False,
        origin='calculated, ',
        spin='HS',
)]
```

Appropriate value of ionization energy and electron affinity are available under `ie` and `ea` attributes

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

[9]:
```
30.651
```

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

[10]:
```
16.1992
```

compute ionic potential

[11]:
```
fe_2.ionic_potential()
```

[11]:
```
0.02564102564102564
```

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.

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

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

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

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

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

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

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")
```

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

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

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

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

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

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")
```

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

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")
```

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

The color map can aslo be csutomized 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")
```

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

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

[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

Lets try a different property: `atomic_volume`

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

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

[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.

[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 the 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)
```

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

3.6.3 Custom coloring scheme

To apply a custom color scheme you can assign color to all the elments 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')
```

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

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`.

[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')
```

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

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

[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')
```

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

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

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

First we need to enable notebook output

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

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

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

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

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

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

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

CHAPTER

FOUR

DATA

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:

Name	Type	Comment	Unit	Data Source
abundance_crust	float	Abundance in the Earth's crust	mg/kg	[23]
abundance_sea	float	Abundance in the seas	mg/L	[23]
annotation	str	Annotations regarding the data		
atomic_number	int	Atomic number		
atomic_radius	float	Atomic radius	pm	[53]
atomic_radius_rahm	float	Atomic radius by Rahm et al.	pm	[45, 46]
atomic_volume	float	Atomic volume	cm^3/mol	
atomic_weight	float	Atomic weight([1])		[35, 63]
atomic_weight_uncert	float	Atomic weight uncertainty([Page 36, 1])		[35, 63]
block	str	Block in periodic table		
boiling_point	float	Boiling temperature	K	[22]
c6	float	C_6 dispersion coefficient in a.u.	a.u.	[13, 56]
c6_gb	float	C_6 dispersion coefficient in a.u. (Gould & Bučko)	a.u.	[21]
cas	str	Chemical Abstracts Serice identifier		
covalent_radius_bragg	float	Covalent radius by Bragg	pm	[10]
covalent_radius_cordero	float	Covalent radius by Cerdero et al.([2])	pm	[16]
covalent_radius_pyykko	float	Single bond covalent radius by Pyykko et al.	pm	[43]

continues on next page

Table 1 – continued from previous page

Name	Type	Comment	Unit	Data Source
covalent_radius_pyykko_d	float	Double bond covalent radius by Pyykko et al.	pm	[42]
covalent_radius_pyykko_t	float	Triple bond covalent radius by Pyykko et al.	pm	[44]
cpk_color	str	Element color in CPK convention	HEX	[61]
critical_pressure	float	Critical pressure	MPa	[22]
critical_temperature	float	Critical temperature	K	[22]
density	float	Density at 295K([9])	g/cm^3	[23, 66]
description	str	Short description of the element		
dipole_polarizability	float	Dipole polarizability	a.u.	[51]
dipole_polarizability_	float	Dipole polarizability uncertainty	a.u.	[51]
discoverers	str	The discoverers of the element		
discovery_location	str	The location where the element was discovered		
discovery_year	int	The year the element was discovered		
electron_affinity	float	Electron affinity([3])	eV	[6, 23]
electrons	int	Number of electrons		
electrophilicity	float	Electrophilicity index	eV	[39]
en_allen	float	Allen's scale of electronegativity([4])	eV	[31, 32]
en_ghosh	float	Ghosh's scale of electronegativity		[18]
en_mulliken	float	Mulliken's scale of electronegativity	eV	[36]
en_pauling	float	Pauling's scale of electronegativity		[23]
econf	str	Ground state electron configuration		
evaporation_heat	float	Evaporation heat	kJ/mol	
fusion_heat	float	Fusion heat	kJ/mol	
gas_basicity	float	Gas basicity	kJ/mol	[23]
geochemical_class	str	Geochemical classification		[59]
glawe_number	int	Glawe's number (scale)		[19]
goldschmidt_class	str	Goldschmidt classification		[59, 60]
group	int	Group in periodic table		
heat_of_formation	float	Heat of formation	kJ/mol	[23]

continues on next page

Table 1 – continued from previous page

Name	Type	Comment	Unit	Data Source
inchi	str	International Chemical Identifier		[24]
ionenergy	tuple	Ionization energies	eV	[26]
ionic_radii	list	Ionic and crystal radii in pm([8])	pm	[29, 52]
is_monoisotopic	bool	Is the element monoisotopic		
is_radioactive	bool	Is the element radioactive		
isotopes	list	Isotopes		
jmol_color	str	Element color in Jmol convention	HEX	[64]
lattice_constant	float	Lattice constant	Angstrom	
lattice_structure	str	Lattice structure code		
mass_number	int	Mass number (most abundant isotope)		
melting_point	float	Melting temperature	K	[22]
mendeleev_number	int	Mendeleev's number([5])		[41, 57]
metallic_radius	float	Single-bond metallic radius	pm	[1]
metallic_radius_c12	float	Metallic radius with 12 nearest neighbors	pm	[1]
molar_heat_capacity	float	Molar heat capacity @ 25 C, 1 bar	J/(mol K)	[23]
molcas_gv_color	str	Element color in MOCAS GV convention	HEX	[65]
name	str	Name in English		
name_origin	str	Origin of the name		
neutrons	int	Number of neutrons (most abundant isotope)		
oxistates	list	Commonly occurring oxidation states		[67]
nist_webbook_url	str	URL for the NIST Chemistry WebBook		[38]
oxistates	list	Oxidation states		
period	int	Period in periodic table		
pettifor_number	float	Pettifor scale		[41]
proton_affinity	float	Proton affinity	kJ/mol	[23]
protons	int	Number of protons		
sconst	float	Nuclear charge screening constants([6])		[14, 15]

continues on next page

Table 1 – continued from previous page

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

[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/cm^3.

For elements where several allotropes exist, the density corresponding to the most abundand are reported (for full list refer to [23]), namely:

4.2 Isotopes

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

4.3 Isotope Decay Modes

Name	Type	Comment	Unit	Data Source
isotope_id	int	ID of the isotope		
mode	str	ASCII symbol of the decay mode		[25]
relation	str	Uncertainty of relative abundance		[25]
intensity	float	Intensity of the decay mode	%	[25]
is_allowed_not_observed	bool	If *True* decay mode is energetically allowed, but not experimentally observed		[25]
is_observed_intensity_unk	bool	If *True* decay mode is observed, but its intensity is not experimentally known		[25]

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

- 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
- Lawrencium
- Rutherfordium
- Dubnium
- Seaborgium
- Bohrium

nations of the symbols.

ASCII	Unicode	Description
A	α	α emission
p	p	proton emission
2p	2p	2-proton emission
n	n	neutron emission
2n	2n	2-neutron emission
EC	ϵ	electron capture
e+	e^+	positron emission
B+	β^+	β^+ decay ($\beta^+ = \epsilon + e^+$)
B-	β^-	β^- decay
2B-	$2\beta^-$	double β^- decay
2B+	$2\beta^+$	double β^+ decay
B-n	β^- n	β^--delayed neutron emission
B-2n	β^- 2n	β^--delayed 2-neutron emission
B-3n	β^- 3n	β^--delayed 3-neutron emission
B+p	β^+ p	β^+-delayed proton emission
B+2p	β^+ 2p	β^+-delayed 2-proton emission
B+3p	β^+ 3p	β^+-delayed 3-proton emission
B-A	$\beta^-\alpha$	β^--delayed α emission
B+A	$\beta^+\alpha$	β^+-delayed α emission
B-d	β^- d	β^--delayed deuteron emission
B-t	β^- t	β^--delayed triton emission
IT	IT	internal transition
SF	SF	spontaneous fission
B+SF	β^+ SF	β^+-delayed fission
B-SF	β^- SF	β^--delayed fission
24Ne	24Ne	heavy cluster emission

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:

```
>>> 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

fetch_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

```
>>> 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 particalr 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 how 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

CHAPTER

SIX

ELECTRONEGATIVITIES

Since electronegativity is 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:

```
>>> 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: ε_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:

```
>>> 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_{eff} is the effective nuclear charge and r is the covalent radius.

Example:

```
>>> 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_{eff} is the effective nuclear charge and r is the covalent radius.

Example:

```
>>> 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:

```
>>> 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_{eff} is the effective nuclear charge and r is the covalent radius.

Example:

```
>>> 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:

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

```
>>> 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:

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

6.9 Nagle

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

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

Example:

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

6.10 Pauling

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

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

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

Example:

```
>>> 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:

$$\chi_S = \frac{\rho}{\rho_{\text{ng}}}$$

where: ρ is the average electron density $\rho = \frac{Z}{4\pi r^3/3}$, and ρ_{ng} is the average electron density of a hypothetical noble gas atom with charge Z.

Example:

```
>>> 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`.

CHAPTER

SEVEN

API REFERENCE

Here you'll find API documentation of the mendeleev's modules.

mendeleev.db	
mendeleev.cli	
mendeleev.econf	Implementation of the abstraction for the elctronic configuration object.
mendeleev.electronegativity	Electronegativity scale formulas.
mendeleev.fetch	
mendeleev.mendeleev	
mendeleev.models	module specifying the database models
mendeleev.ion	
mendeleev.vis.periodictable	
mendeleev.vis.bokeh	
mendeleev.vis.plotly	
mendeleev.vis.seaborn	
mendeleev.vis.utils	
mendeleev.utils	

7.1 mendeleev.db

Functions

get_engine([dbpath])	Return the db engine
get_package_dbpath()	Return the default database path
get_session([dbpath])	Return the database session connection.

7.2 mendeleev.cli

Functions

clielement()	CLI for convenient printing of properties for a given element

7.3 mendeleev.econf

Implementation of the abstraction for the elctronic configuration object.

Functions

get_l(subshell)	Return the orbital angular momentum quantum number for a given subshell
get_spin_strings(sodict[, average])	spin strings as numpy arrays
print_spin_occupations(sodict[, average])	Pretty format for the spin occupations
shell_capactity(shell)	Return the shell capacity (max number of electrons)
subshell_capacity(subshell)	Return the subshell capacity (max number of electrons)
subshell_degeneracy(subshell)	Return the degeneracy of a given subshell

Classes

ElectronicConfiguration([conf, atomre, shellre])	Electronic configuration handler

7.4 mendeleev.electronegativity

Electronegativity scale formulas.

Functions

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

7.5 mendeleev.fetch

Functions

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

7.6 mendeleev.mendeleev

Functions

deltaN(id1, id2[, charge1, charge2, ...])	Calculate the approximate fraction of transferred electrons between elements or ions *id1* and *id2* with charges *charge1* and *charge2* respectively according to the expression
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.
get_all_elements()	Get all elements as a list
ids_to_attr(ids[, attr])	Convert the element ids: atomic numbers, symbols, element names or a combination of the above to a list of corresponding attributes.
isotope(symbol_or_atn, mass_number)	Get an Isotope based on the element symbol and mass number or atomic number and mass number.

7.7 mendeleev.models

module specifying the database models

Functions

estimate_from_group(atomic_number, attr_name)	Evaluate a value *attribute* for element by interpolation or extrapolation of the data points from elements from *group*.
fetch_attrs_for_group(attrs[, group])	A convenience function for getting a specified attribute for all the memebers of a given group.
with_uncertainty(value, uncertainty[, digits])	Format a value with uncertainty using scientific notation.

Classes

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

7.8 mendeleev.ion

Classes

Ion(label[, q])	Class representating 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

create_annotation(row, attr[, size, ...])	Create an annotation from pandas series
create_tile(element, color[, x_offset, y_offset])	Create tile shape
periodic_table_plotly(elements[, attribute, ...])	Create a periodic table visualization with plotly.Figure

7.12 mendeleev.vis.seaborn

Functions

heatmap(elements, prop[, style, figsize, ...])	Plot a heatmap of the given property

7.13 mendeleev.vis.utils

Functions

add_tile_coordinates(df[, x_coord, y_coord, ...])	Calculate coordinates for the tile centers
colormap_column(elements, column[, cmap, ...])	Return a new DataFrame with the same size (and index) as *elements* with a column *cmap* containing HEX color mapping from *cmap* colormap.
create_vis_dataframe([x_coord, y_coord, ...])	Base DataFrame for visualizations

7.14 mendeleev.utils

Functions

coeffs(a[, b])	Return stoichometric coefficients from oxidation states
n_effective(n[, source])	Effective principal quantum number

CHAPTER

EIGHT

BIBLIOGRAPHY

CHAPTER

NINE

MENDELEEV CHANGELOG

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: `glawe_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 lifes 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

CHAPTER

TEN

LICENSE

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Kyle & laby tables of physical & chemical constants. (2017). 3.7.5 atomic radii. [Online; accessed 30-April-2017]. URL: http://www.kayelaby.npl.co.uk/chemistry/3_7/3_7_5.html.

[2] Leland C Allen. Electronegativity is the average one-electron energy of the valence-shell electrons in ground-state free atoms. *Journal of the American Chemical Society*, 111(25):9003–9014, 1989. doi:10.1021/ja00207a003.

[3] Norman L. Allinger, Xuefeng Zhou, and John Bergsma. Molecular mechanics parameters. *Journal of Molecular Structure: THEOCHEM*, 312(1):69–83, jan 1994. URL: http://linkinghub.elsevier.com/retrieve/pii/S0166128009800080, doi:10.1016/S0166-1280(09)80008-0.

[4] A Louis Allred and E G Rochow. A scale of electronegativity based on electrostatic force. *Journal of Inorganic and Nuclear Chemistry*, 5(4):264–268, jan 1958. URL: http://linkinghub.elsevier.com/retrieve/pii/0022190258800032, doi:10.1016/0022-1902(58)80003-2.

[5] Santiago Alvarez. A cartography of the van der Waals territories. *Dalton Transactions*, 42(24):8617, 2013. doi:10.1039/c3dt50599e.

[6] T. Andersen. Atomic negative ions: structure, dynamics and collisions. *Physics Reports*, 394(4-5):157–313, may 2004. URL: http://linkinghub.elsevier.com/retrieve/pii/S0370157304000316, doi:10.1016/j.physrep.2004.01.001.

[7] Stepan S Batsanov. Dielectric Methods of Studying the Chemical Bond and the Concept of Electronegativity. *Russian Chemical Reviews*, 51(7):684–697, jul 1982. URL: http://stacks.iop.org/0036-021X/51/i=7/a=R08?key=crossref.14f2fec4c742d81d9efd6ad10be9ac6a, doi:10.1070/RC1982v051n07ABEH002900.

[8] Stepan S Batsanov. Van der Waals radii of elements. *Inorganic materials*, 37(9):871–885, 2001. URL: http://www.springerlink.com/index/wh8425p357657518.pdf, doi:10.1023/A:1011625728803.

[9] A Bondi. van der Waals Volumes and Radii. *The Journal of Physical Chemistry*, 68(3):441–451, 1964. URL: http://pubs.acs.org/doi/abs/10.1021/j100785a001, doi:10.1021/j100785a001.

[10] W. Lawrence Bragg. The arrangement of atoms in crystals. *Philosophical Magazine*, 40(236):169–189, aug 1920. URL: http://www.tandfonline.com/doi/abs/10.1080/14786440808636111, doi:10.1080/14786440808636111.

[11] Xiaolin Chen and Chuangang Ning. Accurate electron affinity of Co and fine-structure splittings of Co\$^-\$ via slow-electron velocity-map imaging. *Physical Review A*, 93(5):052508, may 2016. URL: http://link.aps.org/doi/10.1103/PhysRevA.93.052508, doi:10.1103/PhysRevA.93.052508.

[12] Xiaolin Chen and Chuangang Ning. Accurate electron affinity of Pb and isotope shifts of binding energies of Pb-. *The Journal of Chemical Physics*, 145(8):084303, aug 2016. URL: http://scitation.aip.org/content/aip/journal/jcp/145/8/10.1063/1.4961654, doi:10.1063/1.4961654.

[13] X Chu and Alexander Dalgarno. Linear response time-dependent density functional theory for van der Waals coefficients. *The Journal of chemical physics*, 121(9):4083–8, sep 2004. URL: http://www.ncbi.nlm.nih.gov/pubmed/15332953, doi:10.1063/1.1779576.

[14] Enrico Clementi and D L Raimondi. Atomic Screening Constants from SCF Functions. *The Journal of Chemical Physics*, 38(11):2686, 1963. URL: http://scitation.aip.org/content/aip/journal/jcp/38/11/10.1063/1.1733573, doi:10.1063/1.1733573.

[15] Enrico Clementi, D L Raimondi, and William P Reinhardt. Atomic Screening Constants from SCF Functions. II. Atoms with 37 to 86 Electrons. *The Journal of Chemical Physics*, 47(4):1300, 1967. URL: http://scitation.aip.org/content/aip/journal/jcp/47/4/10.1063/1.1712084, doi:10.1063/1.1712084.

[16] Beatriz Cordero, Verónica Gómez, Ana E Platero-Prats, Marc Revés, Jorge Echeverría, Eduard Cremades, Flavia Barragán, and Santiago Alvarez. Covalent radii revisited. *Dalton Transactions*, pages 2832, 2008. URL: http://xlink.rsc.org/?DOI=b801115j, doi:10.1039/b801115j.

[17] T. L. Cottrell and L. E. Sutton. Covalency, Electrovalency and Electronegativity. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 207(1088):49–63, jun 1951. URL: http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.1951.0098, doi:10.1098/rspa.1951.0098.

[18] Dulal C Ghosh. A NEW SCALE OF ELECTRONEGATIVITY BASED ON ABSOLUTE RADII OF ATOMS. *Journal of Theoretical and Computational Chemistry*, 04(01):21–33, mar 2005. URL: http://www.worldscientific.com/doi/abs/10.1142/S0219633605001556, doi:10.1142/S0219633605001556.

[19] Henning Glawe, Antonio Sanna, E K U Gross, and Miguel A L Marques. The optimal one dimensional periodic table: a modified Pettifor chemical scale from data mining. *New Journal of Physics*, 18(9):093011, sep 2016. URL: http://stacks.iop.org/1367-2630/18/i=9/a=093011?key=crossref.32680d846cd18e9182d769c453b6099e, doi:10.1088/1367-2630/18/9/093011.

[20] Walter Gordy. A New Method of Determining Electronegativity from Other Atomic Properties. *Physical Review*, 69(11-12):604–607, jun 1946. URL: http://link.aps.org/doi/10.1103/PhysRev.69.604, doi:10.1103/PhysRev.69.604.

[21] Tim Gould and Tomáš Bučko. C6 Coefficients and Dipole Polarizabilities for All Atoms and Many Ions in Rows 1-6 of the Periodic Table. *Journal of Chemical Theory and Computation*, 12(8):3603–3613, aug 2016. URL: http://pubs.acs.org/doi/abs/10.1021/acs.jctc.6b00361, doi:10.1021/acs.jctc.6b00361.

[22] W.M. Haynes. *CRC Handbook of Chemistry and Physics*. CRC Press, 97th edition, 2016. ISBN 9781498754293. URL: https://books.google.no/books?id=VVezDAAAQBAJ.

[23] William M Haynes. *CRC Handbook of Chemistry and Physics*. 100 Key Points. CRC Press, London, 95th edition, 2014. ISBN 9781482208689. URL: https://books.google.no/books?id=bNDMBQAAQBAJ.

[24] Stephen R. Heller, Alan McNaught, Igor Pletnev, Stephen Stein, and Dmitrii Tchekhovskoi. Inchi, the iupac international chemical identifier. *Journal of Cheminformatics*, 7:23, 2015. [Online; accessed 25-September-22], IUPAC link: http://www.iupac.org/inchi/. URL: https://doi.org/10.1186/s13321-015-0068-4, doi:10.1186/s13321-015-0068-4.

[25] F.G. Kondev, M. Wang, W.J. Huang, S. Naimi, and G. Audi. The NUBASE2020 evaluation of nuclear physics properties \ast . *Chinese Physics C*, 45(3):030001, mar 2021. Data file: https://www-nds.iaea.org/amdc/ame2020/nubase_3.mas20.txt. URL: https://doi.org/10.1088/1674-1137/abddae, doi:10.1088/1674-1137/abddae.

[26] A Kramida, Yu Ralchenko, J Reader, and and NIST ASD Team. Nist atomic spectra database (ver. 5.3), national institute of standards and technology, gaithersburg, md. 2015. [Online; accessed 13-April-2015]. URL: http://physics.nist.gov/asd.

[27] Keyan Li and Dongfeng Xue. Estimation of Electronegativity Values of Elements in Different Valence States. *The Journal of Physical Chemistry A*, 110(39):11332–11337, oct 2006. URL: http://pubs.acs.org/doi/abs/10.1021/jp062886k, doi:10.1021/jp062886k.

[28] KeYan Li and DongFeng Xue. New development of concept of electronegativity. *Chinese Science Bulletin*, 54(2):328–334, jan 2009. URL: http://link.springer.com/10.1007/s11434-008-0578-9, doi:10.1007/s11434-008-0578-9.

[29] Daniel Lundberg and Ingmar Persson. The size of actinoid(iii) ions - structural analysis vs. common misinterpretations. *Coordination Chemistry Reviews*, 318:131–134, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0010854515300862, doi:https://doi.org/10.1016/j.ccr.2016.04.003.

[30] Zhihong Luo, Xiaolin Chen, Jiaming Li, and Chuangang Ning. Precision measurement of the electron affinity of niobium. *Physical Review A*, 93(2):020501, feb 2016. URL: http://link.aps.org/doi/10.1103/PhysRevA.93.020501, doi:10.1103/PhysRevA.93.020501.

[31] Joseph B Mann, Terry L Meek, and Leland C Allen. Configuration Energies of the Main Group Elements. *Journal of the American Chemical Society*, 122(12):2780–2783, mar 2000. URL: http://pubs.acs.org/doi/abs/10.1021/ja992866e, doi:10.1021/ja992866e.

[32] Joseph B Mann, Terry L Meek, Eugene T Knight, Joseph F Capitani, and Leland C Allen. Configuration Energies of the d-Block Elements. *Journal of the American Chemical Society*, 122(21):5132–5137, may 2000. URL: http://pubs.acs.org/doi/abs/10.1021/ja9928677, doi:10.1021/ja9928677.

[33] Manjeera Mantina, Adam C Chamberlin, Rosendo Valero, Christopher J Cramer, and Donald G Truhlar. Consistent van der Waals Radii for the Whole Main Group. *The Journal of Physical Chemistry A*, 113(19):5806–5812, may 2009. URL: http://pubs.acs.org/doi/abs/10.1021/jp8111556, doi:10.1021/jp8111556.

[34] Stephen L. Mayo, Barry D. Olafson, and William A Goddard III. DREIDING: a generic force field for molecular simulations. *The Journal of Physical Chemistry*, 94(26):8897–8909, dec 1990. URL: http://pubs.acs.org/doi/abs/10.1021/j100389a010, doi:10.1021/j100389a010.

[35] Juris Meija, Tyler B. Coplen, Michael Berglund, Willi A. Brand, Paul De Bièvre, Manfred Gröning, Norman E. Holden, Johanna Irrgeher, Robert D. Loss, Thomas Walczyk, and Thomas Prohaska. Atomic weights of the elements 2013 (IUPAC Technical Report). *Pure and Applied Chemistry*, 88(3):265–291, jan 2016. URL: http://www.degruyter.com/view/j/pac.2016.88.issue-3/pac-2015-0305/pac-2015-0305.xml, doi:10.1515/pac-2015-0305.

[36] Robert S Mulliken. A New Electroaffinity Scale; Together with Data on Valence States and on Valence Ionization Potentials and Electron Affinities. *The Journal of Chemical Physics*, 2(11):782, 1934. URL: http://scitation.aip.org/content/aip/journal/jcp/2/11/10.1063/1.1749394, doi:10.1063/1.1749394.

[37] Jeffrey K. Nagle. Atomic polarizability and electronegativity. *Journal of the American Chemical Society*, 112(12):4741–4747, jun 1990. URL: http://pubs.acs.org/doi/abs/10.1021/ja00168a019, doi:10.1021/ja00168a019.

[38] National Institute of Standards and Technology. Nist chemistry webbook, standard reference database number 69. [Online; accessed 27-September-2022]. URL: https://webbook.nist.gov/chemistry/.

[39] Robert G. Parr, László v. Szentpály, and Shubin Liu. Electrophilicity index. *Journal of the American Chemical Society*, 121(9):1922–1924, 1999. URL: https://doi.org/10.1021/ja983494x, arXiv:https://doi.org/10.1021/ja983494x, doi:10.1021/ja983494x.

[40] Linus Pauling. THE NATURE OF THE CHEMICAL BOND. IV. THE ENERGY OF SINGLE BONDS AND THE RELATIVE ELECTRONEGATIVITY OF ATOMS. *Journal of the American Chemical Society*, 54(9):3570–3582, sep 1932. URL: http://pubs.acs.org/doi/abs/10.1021/ja01348a011, doi:10.1021/ja01348a011.

[41] D G Pettifor. A chemical scale for crystal-structure maps. *Solid State Communications*, 51(1):31–34, jul 1984. URL: http://www.sciencedirect.com/science/article/pii/0038109884907658, doi:10.1016/0038-1098(84)90765-8.

[42] Pekka Pyykkö and Michiko Atsumi. Molecular Double-Bond Covalent Radii for Elements Li-E112. *Chemistry - A European Journal*, 15(46):12770–12779, nov 2009. URL: http://doi.wiley.com/10.1002/chem.200901472, doi:10.1002/chem.200901472.

[43] Pekka Pyykkö and Michiko Atsumi. Molecular Single-Bond Covalent Radii for Elements 1-118. *Chemistry - A European Journal*, 15(1):186–197, jan 2009. URL: http://doi.wiley.com/10.1002/chem.200800987, doi:10.1002/chem.200800987.

[44] Pekka Pyykkö, Sebastian Riedel, and Michael Patzschke. Triple-Bond Covalent Radii. *Chemistry - A European Journal*, 11(12):3511–3520, jun 2005. URL: http://doi.wiley.com/10.1002/chem.200401299, doi:10.1002/chem.200401299.

[45] Martin Rahm, Roald Hoffmann, and N. W. Ashcroft. Atomic and Ionic Radii of Elements 1-96. *Chemistry - A European Journal*, 22(41):14625–14632, oct 2016. URL: http://doi.wiley.com/10.1002/chem.201602949, doi:10.1002/chem.201602949.

[46] Martin Rahm, Roald Hoffmann, and N. W. Ashcroft. Corrigendum: Atomic and Ionic Radii of Elements 1-96. *Chemistry - A European Journal*, 23(16):4017–4017, mar 2017. URL: http://doi.wiley.com/10.1002/chem.201700610, doi:10.1002/chem.201700610.

[47] A K Rappe, C. J. Casewit, K. S. Colwell, William A Goddard III, and W. M. Skiff. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American Chemical Society*, 114(25):10024–10035, dec 1992. URL: http://pubs.acs.org/doi/abs/10.1021/ja00051a040, doi:10.1021/ja00051a040.

[48] R Scott Rowland and Robin Taylor. Intermolecular Nonbonded Contact Distances in Organic Crystal Structures: Comparison with Distances Expected from van der Waals Radii. *The Journal of Physical Chemistry*, 100(18):7384–7391, 1996. doi:10.1021/jp953141+.

[49] R T Sanderson. An Interpretation of Bond Lengths and a Classification of Bonds. *Science*, 114(2973):670–672, dec 1951. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.114.2973.670, doi:10.1126/science.114.2973.670.

[50] R T Sanderson. An Explanation of Chemical Variations within Periodic Major Groups. *Journal of the American Chemical Society*, 74(19):4792–4794, oct 1952. URL: http://pubs.acs.org/doi/abs/10.1021/ja01139a020, doi:10.1021/ja01139a020.

[51] Peter Schwerdtfeger and Jeffrey K. Nagle. 2018 Table of static dipole polarizabilities of the neutral elements in the periodic table. *Molecular Physics*, 0(0):1–26, oct 2018. URL: https://doi.org/00268976.2018.1535143https://www.tandfonline.com/doi/full/10.1080/00268976.2018.1535143, doi:10.1080/00268976.2018.1535143.

[52] R. D. Shannon. Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. *Acta Crystallographica Section A*, 32(5):751–767, 1976. doi:10.1107/S0567739476001551.

[53] John C Slater. Atomic Radii in Crystals. *The Journal of Chemical Physics*, 41(10):3199, 1964. URL: http://scitation.aip.org/content/aip/journal/jcp/41/10/10.1063/1.1725697, doi:10.1063/1.1725697.

[54] N Stone. Table of nuclear quadrupole moments, international atomic energy agency, indc(nds)-650. December 2013. URL: https://www-nds.iaea.org/publications/indc/indc-nds-0650.pdf.

[55] N Stone. Table of nuclear magnetic dipole and electric quadrupole moments, international atomic energy agency, indc(nds)-0658. February 2014. URL: https://www-nds.iaea.org/publications/indc/indc-nds-0658.pdf.

[56] K T Tang, J M Norbeck, and P R Certain. Upper and lower bounds of two- and three-body dipole, quadrupole, and octupole van der Waals coefficients for hydrogen, noble gas, and alkali atom interactions. *The Journal of Chemical Physics*, 64(7):3063, 1976. URL: http://scitation.aip.org/content/aip/journal/jcp/64/7/10.1063/1.432569, doi:10.1063/1.432569.

[57] P. Villars, K. Cenzual, J. Daams, Y. Chen, and S. Iwata. Data-driven atomic environment prediction for binaries using the Mendeleev number. *Journal of Alloys and Compounds*, 367(1-2):167–175, mar 2004. URL: http://linkinghub.elsevier.com/retrieve/pii/S0925838803008004, doi:10.1016/j.jallcom.2003.08.060.

[58] Jürgen Vogt and Santiago Alvarez. van der Waals Radii of Noble Gases. *Inorganic Chemistry*, 53(17):9260–9266, sep 2014. URL: http://pubs.acs.org/doi/abs/10.1021/ic501364h, doi:10.1021/ic501364h.

[59] W M White. *Geochemistry*. Wiley, 2013. ISBN 9781118485255. URL: https://books.google.no/books?id=QPH1nY8WztkC.

[60] Wikipedia. Goldschmidt classification — wikipedia, the free encyclopedia. [Online; accessed 30-April-2017]. URL: https://en.wikipedia.org/w/index.php?title=Goldschmidt_classification&oldid=775842423.

[61] Wikipedia. Cpk coloring — wikipedia, the free encyclopedia. 2017. [Online; accessed 5-October-2017]. URL: https://en.wikipedia.org/w/index.php?title=CPK_coloring&oldid=802098372.

[62] IUPAC-CIAAW. Atomic masses. [Online; accessed 25-September-2022, data file: https://ciaaw.org/data/IUPAC-atomic-masses.csv, last updated: 17 March 2021]. URL: https://ciaaw.org/atomic-masses.htm.

[63] IUPAC-CIAAW. Standard atomic weights. [Online; accessed 1-January-2017]. URL: http://www.ciaaw.org/atomic-weights.htm.

[64] Jmol Team. Jmol colors. [Online; accessed 5-October-2017]. URL: http://jmol.sourceforge.net/jscolors/#color_U.

[65] MOLCAS Team. Molcas gv colors. [Online; accessed 5-October-2017]. URL: http://www.molcas.org/GV/.

[66] Wikipedia contributors. List of chemical elements — Wikipedia, the free encyclopedia. 2021. [Online; accessed 30-August-2021]. URL: https://en.wikipedia.org/w/index.php?title=List_of_chemical_elements&oldid=1039678864.

[67] Wikipedia contributors. Oxidation state — Wikipedia, the free encyclopedia. 2022. [Online; accessed 28-September-2022]. URL: https://en.wikipedia.org/w/index.php?title=Oxidation_state&oldid=1102394064.

PYTHON MODULE INDEX

m

INDEX