path2insight Documentation

Release 1.1.0

Armel Levebvre and Jonathan de Bruin

May 03, 2020

Contents

CHAPTER 1

About

Path2Insight (p2i) is a modular and scalable python module which aims at offering a unified and comprehensive set of processing tools for analyzing file paths. P2i supports static file systems analysis without requiring access to the original physical storage. Basically, a scan of the storage's content exported as a text file suffices to explore the saved resources. There is also no need to access the content of the files as the p2i module import file paths as strings.

Once loaded, the file paths are stored in-memory as a python object enabling: preprocessing, text processing and descriptive analysis of folders and files.

Preprocessing: Sample, sort and select files based on multiple criteria (e.g. parent folder, depth).

Text processing: Chunk file paths into tokens (full path, stem and name), n-grams or complete paths with the help of several extensible tokenizers. Also, taggers offer the option to aggregate files based on their structure and content (which prepare paths for further analysis such as entity recognition or classification tasks).

Descriptive analysis: P2i implements counters for tokens, stems and extensions. It supports also statistical features such as X2 tests on the distribution of extensions, stems and names. Further, a representation of the complexity of the folder structure is facilitated by folder-depth analysis functionalities.

The table below shows how Path2Insight differ and complement functionalities offered by lower-level python modules (pathlib and os.path).

Functionality	P2i	Pathlib	os.path
Preprocessing	Pathlib + Sampling, sorting, selection	match, joinpath	Normcase, norm path
Descriptive statistics	Counters: stem, extension, name. Taggers. Tokenizers	os.stat	os.stat
Text processing	Pathlib + Tokens, n-grams, taggers, lower, upper,...	Stem, name, parent, extension drive, ...	Split
Access or modify information on the system	No, can be linked to additional metadata (datetimes, users) by joining on the full path	Yes, chmod, current folder. ...	Yes, user, size, datetimes, descriptors, ...

P2i is dependency free (only pathlib2 is required for Python 2.7 users), fast and scalable path processing toolkit. It is compliant with the major data analysis python modules such as pandas, scikit-learn, nltk and matplotlib to extent the

analytical possibilities of path2insight.

1.1 Example

Import the module and load a demo dataset with static file paths (or use *path2insight.walk* to collect from you file system).

```
>>> import path2insight
>>> from path2insight.datasets import load_ensembl

>>> filepaths = load_ensembl()
```

```
>>> path2insight.depth_counts(filepaths)
Counter({3: 1, 4: 11, 5: 39424, 6: 5543, 7: 2733, 8: 3388})
```

```
>>> path2insight.token_counts(filepaths).most_common(10)
[('txt', 31977),
 ('gene', 13798),
 ('ensembl', 12727),
 ('dm', 12500),
 ('homolog', 7380),
 ('fa', 5890),
 ('chromosome', 5011),
 ('feature', 4878),
 ('dna', 4608),
 ('90', 3404)]
```

```
>>> path2insight.extension_counts(filepaths).most_common(10)
[('.gz', 44427),
 ('', 3094),
 ('.bb', 847),
 ('.nsq', 349),
 ('.nin', 349),
 ('.nhr', 349),
 ('.tsv', 336),
 ('.psq', 250),
 ('.pin', 250),
 ('.phr', 250)]
```

```
>>> path2insight.select_re(filepaths, level5='micro.*')
[PosixFilePath('/Volumes/release-90/variation/VEP/microtus_ochrogaster_vep_90_MicOch1.
↪0.tar.gz'),
 PosixFilePath('/Volumes/release-90/variation/VEP/microtus_ochrogaster_refseq_vep_90_
↪MicOch1.0.tar.gz'),
 PosixFilePath('/Volumes/release-90/variation/VEP/microtus_ochrogaster_merged_vep_90_
↪MicOch1.0.tar.gz'),
 PosixFilePath('/Volumes/release-90/variation/VEP/microcebus_murinus_vep_90_Mmur_2.0.
↪tar.gz'),
 PosixFilePath('/Volumes/release-90/rdf/microtus_ochrogaster/microtus_ochrogaster_
↪xrefs.ttl.gz.graph'),
```

```
>>> path2insight.distance_on_token(filepaths[0:10])
array([[ 0.        ,  2.        ,  1.41421356,  3.        ,  3.        ],
       [ 2.        ,  0.        ,  2.44948974,  3.31662479,  3.31662479],
```

(continues on next page)

(continued from previous page)

```
       [ 1.41421356,  2.44948974,  0.        ,  3.        ,  3.        ],
       [ 3.        ,  3.31662479,  3.        ,  0.        ,  1.41421356],
       [ 3.        ,  3.31662479,  3.        ,  1.41421356,  0.        ]])
```

1.2 Cite

[1] A. Lefebvre and M. Spruit, “Designing laboratory forensics” in 18th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E 2019, 2019.

1.3 Authors

- Armel Lefebvre
- Jonathan de Bruin

CHAPTER 2

Installation and dependencies

Path2Insight is available on Pypi. This make it possible to install it with through:

```
pip install path2insight
```

To upgrade path2insight use

```
pip install --upgrade path2insight
```

It is also possible to install from source:

```
python setup.py install
```

Path2Insight is available for 3.6+. Path2Insight depends heavily on the pathlib module. This module is part of Python 3.4 or higher.

Some of the submodules of Path2Insight depend on other Python packages (numpy, pandas, sklearn, scipy, jellyfish). One can get a full installation by installing the packages in the *requirements-full.txt* file.

```
pip install -r requirements-full.txt
```

2.1 Docs

The documentation for Path2Insight is available on readthedocs. To generate the docs by yourself, install the following packages (available on pip):

- sphinx
- sphinx_rtd_theme
- nbsphinx

```
pip install -r docs/requirements-docs.txt
```

Navigate to the *docs/* folder and call *make html* or *make pdf* for a pdf version.

2.2 Development and Testing

Clone/fork the latest version of the Path2Insight from Github.

```
git clone https://github.com/OpenResearchIntelligence/path2insight
```

Install the module in the development mode.

```
python setup.py develop
```

Path2Insight makes use of PyTest for unit testing. Run the tests with the command:

```
pytest
```

CHAPTER 3

Getting started

Each file path in Path2Insight is converted into a *WindowsFilePath* or *PosixFilePath* object. Since Python 3.4, there is a module to analyze and manipulate file paths in Python named pathlib. Path2Insight extends the PurePath class in this module. Path2Insight has the classes PureWindowsPath and PurePosixPath. These two classes play an important role in the package and are the input of nearly each function.

As described before, there are objects for Windows file paths and objects for Posix (Linux, macOS) file paths. This is because the file paths on Windows and Posix are different. A typical (absolute) path in Windows looks like: `K:\Datasets\Climate\dataset_climate_change.csv`. A typical (absolute) path in Linux looks like: `/var/Datasets/Climate/dataset_climate_change.csv`. A typical (absolute) path in macOS looks like: `/Volumes/Datasets/Climate/dataset_climate_change.csv`. Windows file paths start with a drive letter and uses backslashes as separators. Posix file paths use a slash as root and and slashes as separators.

3.1 FilePath Objects

Load the `path2insight.WindowsFilePath` and `path2insight.PosixFilePath` objects.

[2]:
```
from path2insight import WindowsFilePath, PosixFilePath
```

Path2Insight provides the possibility to analyze the file path deeply. This can be done by converting the string-type file path into a WindowsFilePath-type or PosixFilePath-type object. One can make a WindowsFilePath object of the string-type file path by passing it as an argument to the class. It is also possible to provide pass the file path in parts.

[3]:
```
print(WindowsFilePath('K:\Datasets\Climate\dataset_climate_change.csv'))
print(WindowsFilePath('K:\\', 'Datasets', 'Climate', 'dataset_climate_change.csv'))
```

```
K:\Datasets\Climate\dataset_climate_change.csv
K:\Datasets\Climate\dataset_climate_change.csv
```

For a Posix type file path, one uses the PosixFilePath class.

[4]:
```
print(PosixFilePath('/var/Datasets/Climate/dataset_climate_change.csv'))
```

```
/var/Datasets/Climate/dataset_climate_change.csv
```

It is also possible to use relative file paths. This is done exactly the same way as with absolute file paths.

[5]:
```
print(WindowsFilePath('Climate', 'dataset_climate_change.csv'))
print(WindowsFilePath('..', 'Climate', 'dataset_climate_change.csv'))
```

```
Climate\dataset_climate_change.csv
..\Climate\dataset_climate_change.csv
```

To convert the WindowsFilePath or PosixFilePath back to

3.2 Methods and attributes

Converting the file path from a string into a WindowsFilePath or PosixFilePath object gives you large number of functionalities. One can split the file path into parts, get the extensions, lower or upper the stem. See the documentation of *WindowsFilePath* and *PosixFilePath* for all the attributes and methods. The following example shows some of the features.

[6]:
```
path = WindowsFilePath('K:\Datasets\Climate\dataset_climate_change.csv')
path.parts
# ('K:\\', 'Datasets', 'Climate', 'dataset_climate_change.csv')
path.drive
# 'K:'
path.lower()
# WindowsFilePath('k:/datasets/climate/dataset_climate_change.csv')
path.stem
# 'dataset_climate_change'
path.extension
# '.csv'
path.upper_stem()
# WindowsFilePath('K:/Datasets/Climate/DATASET_CLIMATE_CHANGE.csv')
path.name
# 'dataset_climate_change.csv'
path.name.upper();
# 'DATASET_CLIMATE_CHANGE.csv'
```

Note that some of the methods do return a WindowsFilePath object while other do not. It depends on your application what the preferred method is. Take a look at the following two examples:

[7]:
```
str(path).lower()
```

[7]:
```
'k:\\datasets\\climate\\dataset_climate_change.csv'
```

[8]:
```
str(path.lower())
```

[8]:
```
'k:\\datasets\\climate\\dataset_climate_change.csv'
```

They look the same, but the first one is a string while the second one is a WindowsFilePath object (see the cell below).

[9]:
```
print(type(str(path).lower()))
print(type(path.lower()))
```

```
<class 'str'>
<class 'path2insight.core.WindowsFilePath'>
```

The same for the when using parts. See below:

[10]:
```
# the following line return a tuple with parts
path.lower().parts
# ('k:\\', 'datasets', 'climate', 'dataset_climate_change.csv')
```

[10]:
```
('k:\\', 'datasets', 'climate', 'dataset_climate_change.csv')
```

[11]:
```
# while the following raises an error
'K:\Datasets\Climate\dataset_climate_change.csv'.lower().parts
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-37be527e591a> in <module>()
      1 # while the following raises an error
----> 2 'K:\Datasets\Climate\dataset_climate_change.csv'.lower().parts

AttributeError: 'str' object has no attribute 'parts'
```

There is a growing list of methods available for the WindowsFilePath and PosixFilePath objects. At the moment, nearly all methods and attributes available in the WindowsFilePath object are also available in the PosixFilePath class. The FilePath objects inherit methods and attributes from the PurePath object of the pathlib module (new in Python 3.4). See DOCUMENTATION_LINK for the reference for WindowsFilePath and PosixFilePath.

3.3 Collections of file paths

The previous section shows how to extract information from a single file path. Path2Insight is optimized to analyze large collections of file paths. One can analyze collections by using **list comprehensions**. The Python documentation has a clear section about this topic which you can find here. Path2Insight follows the Natural Language Toolkit (NLTK) with this API structure.

For this section we use a the following data:

[12]:
```
from path2insight import WindowsFilePath, PosixFilePath

import path2insight

paths = [
    'K:\Datasets\Climate\dataset_climate_change.csv',
    'K:\Datasets\Climate\dataset_energy_consumption.csv',
    'K:\Datasets\Climate\dataset_energy_consumption.xlsx',
    'K:\Datasets\Climate\climate_change.py',
    'K:\Datasets\Climate\README'
]
```

The first step is to convert the list of file paths into WindowsFilePath objects. This is done with a list comprehension (see cell below) or with the parse function in `path2insight.parse(windows_paths, 'windows')`.

[13]:
```
windows_paths = [WindowsFilePath(path) for path in paths]
windows_paths
```

[13]:
```
[WindowsFilePath('K:/Datasets/Climate/dataset_climate_change.csv'),
 WindowsFilePath('K:/Datasets/Climate/dataset_energy_consumption.csv'),
 WindowsFilePath('K:/Datasets/Climate/dataset_energy_consumption.xlsx'),
 WindowsFilePath('K:/Datasets/Climate/climate_change.py'),
 WindowsFilePath('K:/Datasets/Climate/README')]
```

By using list-comprehensions, one can extract information from the file paths. For example, extract the extension for each path:

[14]:
```
[path.extension for path in windows_paths]
```

[14]:
```
['.csv', '.csv', '.xlsx', '.py', '']
```

Or a more advanced example where the tokens of the file path stems are collected:

[15]:
```
[token for path in windows_paths for token in path.tokenize_stem()]
```

[15]:
```
['dataset',
 'climate',
 'change',
 'dataset',
 'energy',
 'consumption',
 'dataset',
 'energy',
 'consumption',
 'climate',
 'change',
 'README']
```

One can use the power of Python's Counter class to count the tokens.

[16]:
```
from collections import Counter

Counter([token for path in windows_paths for token in path.tokenize_stem()])
```

[16]:
```
Counter({'README': 1,
         'change': 2,
         'climate': 2,
         'consumption': 2,
         'dataset': 3,
         'energy': 2})
```

3.4 Selecting, sampling and sorting file paths

Path2Insight contains functions to handle collections of file paths (stored in lists). These functions make it easy to subset and select parts of the data. In this section shows some examples on using the file path handling functions. The functions described in this section cover sampling, subsetting and sorting.

First, load the demo dataset Ensembl

[17]:
```
from path2insight.datasets import load_ensembl

data = load_ensembl()
data[:5]
```

[17]:
```
[PosixFilePath('/Volumes/release-90/README'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/README.gene_trees.
↪xml_dumps.txt'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/MD5SUM'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/Compara.90.protein_
↪murinae.tree.phyloxml.xml.tar.gz'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/Compara.90.protein_
↪murinae.tree.orthoxml.xml.tar.gz')]
```

```
[18]: # sample 5 paths from the data
path2insight.sample(data, 5)
```

```
[18]: [PosixFilePath('/Volumes/release-90/mysql/ensembl_mart_90/lafricana_gene_ensembl__
↪homolog_jjaculus__dm.txt.gz'),
 PosixFilePath('/Volumes/release-90/mysql/ensembl_mart_90/gmorhua_gene_ensembl__
↪homolog_tguttata__dm.txt.gz'),
 PosixFilePath('/Volumes/release-90/mysql/ensembl_mart_90/mspreteij_gene_ensembl__
↪homolog_oprinceps__dm.txt.gz'),
 PosixFilePath('/Volumes/release-90/mysql/ensembl_mart_90/ttruncatus_gene_ensembl__
↪protein_feature_prints__dm.txt.gz'),
 PosixFilePath('/Volumes/release-90/gff3/mus_caroli/Mus_caroli.CAROLI_EIJ_v1.1.90.chr.
↪gff3.gz')]
```

```
[19]: # make a subset of paths with 'fasta' as the third level and 'dna' as the fifth level.
data_subset = path2insight.select(data, level3='fasta', level5='dna')
data_subset[:5]
```

```
[19]: [PosixFilePath('/Volumes/release-90/fasta/xiphophorus_maculatus/dna/Xiphophorus_
↪maculatus.Xipmac4.4.2.dna_sm.toplevel.fa.gz'),
 PosixFilePath('/Volumes/release-90/fasta/xiphophorus_maculatus/dna/Xiphophorus_
↪maculatus.Xipmac4.4.2.dna_sm.nonchromosomal.fa.gz'),
 PosixFilePath('/Volumes/release-90/fasta/xiphophorus_maculatus/dna/Xiphophorus_
↪maculatus.Xipmac4.4.2.dna_rm.toplevel.fa.gz'),
 PosixFilePath('/Volumes/release-90/fasta/xiphophorus_maculatus/dna/Xiphophorus_
↪maculatus.Xipmac4.4.2.dna_rm.nonchromosomal.fa.gz'),
 PosixFilePath('/Volumes/release-90/fasta/xiphophorus_maculatus/dna/Xiphophorus_
↪maculatus.Xipmac4.4.2.dna.toplevel.fa.gz')]
```

```
[20]: # default list sort method works also for the WindowsFilePath
# and PosixFilePath objects. This method sorts the data inplace.
data.sort()
```

```
[21]: # same as data.sort() but not inplace
data_sorted = path2insight.sort(data)
data_sorted[:5]
```

```
[21]: [PosixFilePath('/Volumes/release-90/README'),
 PosixFilePath('/Volumes/release-90/fasta/ciona_savignyi/dna_index/CHECKSUMS'),
 PosixFilePath('/Volumes/release-90/fasta/ciona_savignyi/dna_index/Ciona_savignyi.
↪CSAV2.0.dna.toplevel.fa.gz'),
 PosixFilePath('/Volumes/release-90/fasta/ciona_savignyi/dna_index/Ciona_savignyi.
↪CSAV2.0.dna.toplevel.fa.gz.fai'),
 PosixFilePath('/Volumes/release-90/fasta/ciona_savignyi/dna_index/Ciona_savignyi.
↪CSAV2.0.dna.toplevel.fa.gz.gzi')]
```

In the following example, the data is sorted on level 5 first and level 4 second.

```
[22]: data_sorted_advanced = path2insight.sort(data, level=[5, 4])
data_sorted_advanced[:5]
```

```
[22]: [PosixFilePath('/Volumes/release-90/README'),
 PosixFilePath('/Volumes/release-90/gff3/ailuropoda_melanoleuca/Ailuropoda_
↪melanoleuca.ailMel1.90.abinitio.gff3.gz'),
 PosixFilePath('/Volumes/release-90/gtf/ailuropoda_melanoleuca/Ailuropoda_melanoleuca.
↪ailMel1.90.abinitio.gtf.gz'),
 PosixFilePath('/Volumes/release-90/tsv/ailuropoda_melanoleuca/Ailuropoda_melanoleuca.
↪ailMel1.90.ena.tsv.gz'),
```

(continues on next page)

(continued from previous page)

```
 PosixFilePath('/Volumes/release-90/tsv/ailuropoda_melanoleuca/Ailuropoda_melanoleuca.
↪ailMel1.90.entrez.tsv.gz')]
```

CHAPTER 4

Basic insight

Path2Insight can provide insight in large collections of static file paths. This pages gives a short introduction in gaining basic insight in a collection of file paths. Topics are counting and comparing in file path collections.

[2]:
```
import path2insight
```

Path2Insight comes with sample collections of file paths. These collections/datasets were collected from the internet and were publicly available at the time of collection. For the examples on this page, the Ensembl dataset was used. Ensembl is a genome browser for vertebrate genomes. The data can be found at ftp://ftp.ensembl.org/pub/release-90/. The dataset contains 51100 filepaths.

[3]:
```
from path2insight.datasets import load_ensembl

data = load_ensembl()
```

The object `data` is a list with `PosixFilePath` objects. These `PosixFilePath` objects contain the file paths. The file paths are stored in `PosixFilePath` object because they were collected with a Posix device. To give you an indication of the first filepaths, we output the first 5 elements of the list:

[4]:
```
data[:5]
```

[4]:
```
[PosixFilePath('/Volumes/release-90/README'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/README.gene_trees.
↪xml_dumps.txt'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/MD5SUM'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/Compara.90.protein_
↪murinae.tree.phyloxml.xml.tar.gz'),
 PosixFilePath('/Volumes/release-90/xml/ensembl-compara/homologies/Compara.90.protein_
↪murinae.tree.orthoxml.xml.tar.gz')]
```

4.1 Counting

Counting is a very intuitive way to explore all kind of things. The same holds for file paths. In this section, we explore the collection of file paths based on some simple counts like the folder depth, the extension and the stem.

[5]:
```
path2insight.depth_counts(data)
```

[5]:
```
Counter({3: 1, 4: 11, 5: 39424, 6: 5543, 7: 2733, 8: 3388})
```

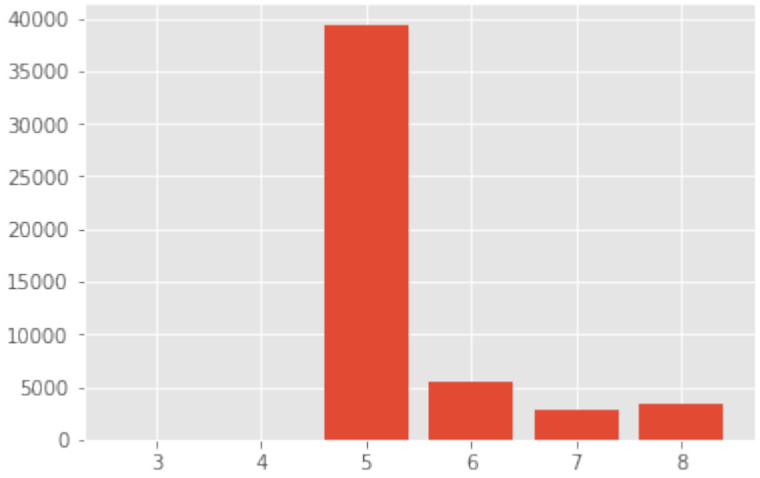
[6]:
```
path2insight.depth_counts(data, normalize=True)
```

[6]:
```
Counter({3: 1.9569471624266145e-05,
         4: 0.00021526418786692759,
         5: 0.7715068493150685,
         6: 0.10847358121330725,
         7: 0.05348336594911937,
         8: 0.0663013698630137})
```

It is easy to visualize the data with matplotlib or pandas:

[7]:
```
result = path2insight.depth_counts(data)

import matplotlib.pyplot as plt
plt.bar(*zip(*result.items()))

# another option:
# pandas.Series(result).plot.bar()
```

[7]:
```
<Container object of 6 artists>
```



One can see that a depth of 5 is very common. Keep in mind that we are starting at 3. The actual number is not so informative, the distribution is. This data server has a very flat distribution of folders and files.

It is also easy to compute the mean and variance of the depth.

[8]:
```
import numpy

depth = [path.depth for path in data]

print(numpy.mean(depth))
print(numpy.var(depth))
```

```
5.41409001957
0.747942371544
```

Extensions can be very informative. Large research data servers may contain more than hundred different file extensions. From that perspective, the file extension becomes very informative. Path2Insight contains functions to explore file extensions. File extension always start with a dot (.) in the Path2Insight. See Wikipedia for more information about definitions and remarks on filename extensions. https://en.wikipedia.org/wiki/Filename_extension

[9]:
```
path2insight.extension_counts(data)
```

[9]:
```
Counter({'': 3094,
         '.4_sauropsids_EPO': 1,
         '.7_sauropsids_EPO_LOW_COVERAGE': 1,
         '.8_primates_EPO': 1,
         '.bb': 847,
         '.fai': 69,
         '.graph': 198,
         '.gz': 44427,
         '.gzi': 69,
         '.maf': 2,
         '.md5': 1,
         '.nal': 3,
         '.nhr': 349,
         '.nin': 349,
         '.nsq': 349,
         '.ova': 1,
         '.pdf': 1,
         '.phr': 250,
         '.pin': 250,
         '.psq': 250,
         '.sh': 2,
         '.tar': 74,
         '.tbi': 72,
         '.tsv': 336,
         '.txt': 102,
         '.vcf': 2})
```

The number of extensions can be large. The functions like `extension_counts` return a python `collections.Counter` object. This object has useful properties and methods like `.most_common()`. Use the following code to output and order the 10 most used extensions:

[10]:
```
path2insight.extension_counts(data).most_common(10)
```

[10]:
```
[('.gz', 44427),
 ('', 3094),
 ('.bb', 847),
 ('.nsq', 349),
 ('.nin', 349),
 ('.nhr', 349),
 ('.tsv', 336),
 ('.psq', 250),
 ('.pin', 250),
 ('.phr', 250)]
```

Files can have multiple extensions. The number of extension occurrences is distributed as follows:

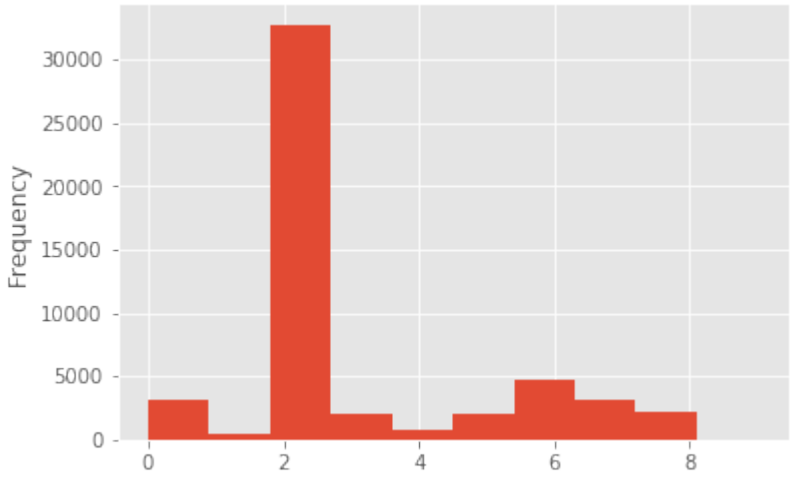
[11]:
```
number_of_extensions_list = [len(path.extensions) for path in data]
```

(continues on next page)

(continued from previous page)

```
pandas.Series(number_of_extensions_list).plot.hist()
```

```
[11]: <matplotlib.axes._subplots.AxesSubplot at 0x10d7a0748>
```



Path2insight can also be used to count other file path properties in file path collections. The following examples show how to count the stem occurrences and tokens. See the API reference for all counters.

```
[12]: path2insight.stem_counts(data).most_common(10)
```

```
[12]: [('CHECKSUMS', 2222),
       ('README', 762),
       ('meta.txt', 271),
       ('seq_region.txt', 267),
       ('meta_coord.txt', 267),
       ('coord_system.txt', 267),
       ('external_db.txt', 245),
       ('analysis_description.txt', 245),
       ('xref.txt', 244),
       ('unmapped_reason.txt', 244)]
```

```
[13]: # lowercase stems (no difference in counts)
path2insight.stem_counts([path.lower() for path in data]).most_common(10)
```

```
[13]: [('checksums', 2222),
       ('readme', 762),
       ('meta.txt', 271),
       ('seq_region.txt', 267),
       ('meta_coord.txt', 267),
       ('coord_system.txt', 267),
       ('external_db.txt', 245),
       ('analysis_description.txt', 245),
       ('xref.txt', 244),
       ('unmapped_reason.txt', 244)]
```

Counting tokens

```
[14]: data[100].tokenize()
```

```
[14]: ['Volumes',
       'release',
```

(continues on next page)

(continued from previous page)

```
 '90',
 'variation',
 'vcf',
 'mus',
 'musculus',
 'Mus',
 'musculus']
```

[15]:
```
path2insight.token_counts(data).most_common(10)
```

[15]:
```
[('txt', 31977),
 ('gene', 13798),
 ('ensembl', 12727),
 ('dm', 12500),
 ('homolog', 7380),
 ('fa', 5890),
 ('chromosome', 5011),
 ('feature', 4878),
 ('dna', 4608),
 ('90', 3404)]
```

4.2 Compare collections

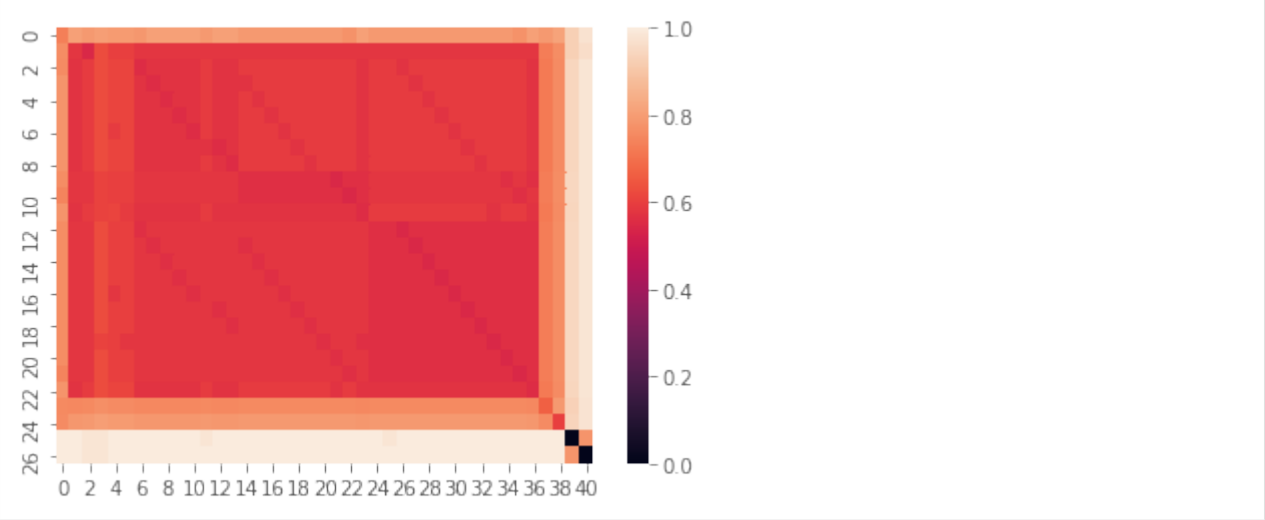
[16]:
```
# subset of data
data_tetraodon_nigroviridis = path2insight.select(
    data, level3='gff3', level4='tetraodon_nigroviridis')

# subset of data
data_taeniopygia_guttata = path2insight.select(
    data, level3='gff3', level4='taeniopygia_guttata')
```

4.2.1 String similarity

In the following example, the two datasets are compared on the string similarity of the file path stem. 0 is completely identical and 1 is completely distinct. It turns out that most of the file stems are partially similar. A few file stems are quite unique and 2 file paths are nearly (or exactly identical).

[17]:
```
m = path2insight.levenshtein_distance_stem(
    data_tetraodon_nigroviridis, data_taeniopygia_guttata,
    normalise=True
)

import seaborn as sns
sns.heatmap(m, vmin=0)
```

[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1192a20f0>
```



4.2.2 Hypothesis testing

Compare file extensions with the Chi-squared test. It turns out that the p-value is 0.0249.

[18]:
```
path2insight.extension_chisquare(
    data_tetraodon_nigroviridis, data_taeniopygia_guttata)
```

[18]:
```
Power_divergenceResult(statistic=5.0256410256410255, pvalue=0.024974679293054206)
```

CHAPTER 5

Advanced insight

Path2Insight can be used to gather advanced insight of the file system in terms of file paths. This section will show some examples on how to gather a more advanced insight. Think about insight in terms of comparing project structures.

UNDER CONTRUCTION

CHAPTER 6

Visualize

This page shows examples on how to visualize the file system.

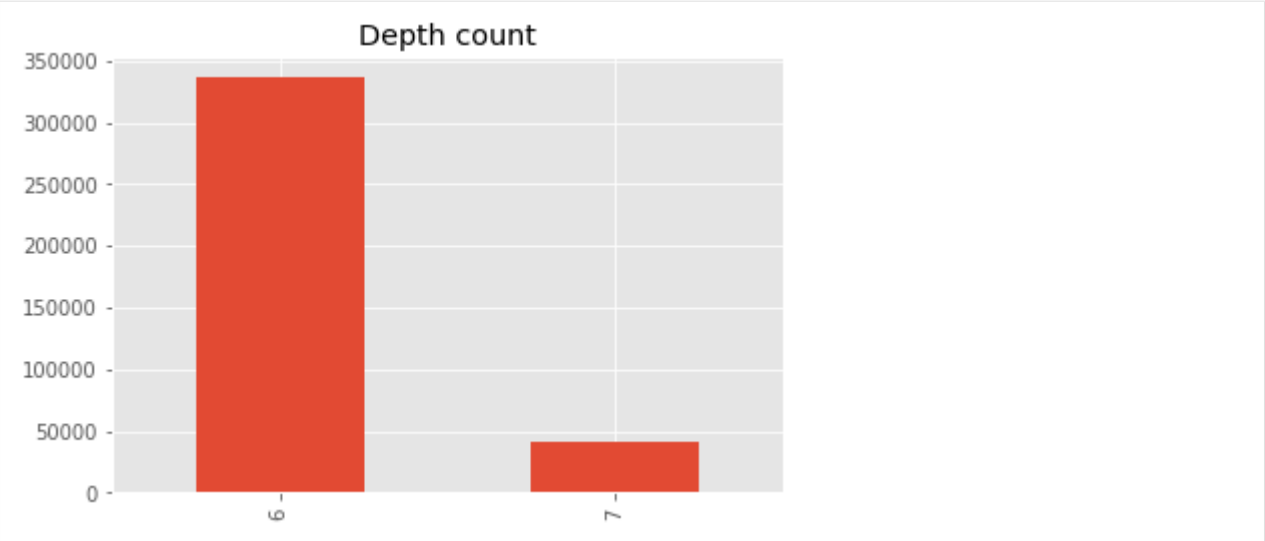
[2]:
```
%matplotlib inline

import matplotlib.pyplot as plt
plt.style.use('ggplot')
```

[3]:
```
import pandas
import numpy

import path2insight
from path2insight.datasets import load_pride

data = load_pride()
```

6.1 Example 1 (Simple aggregations)

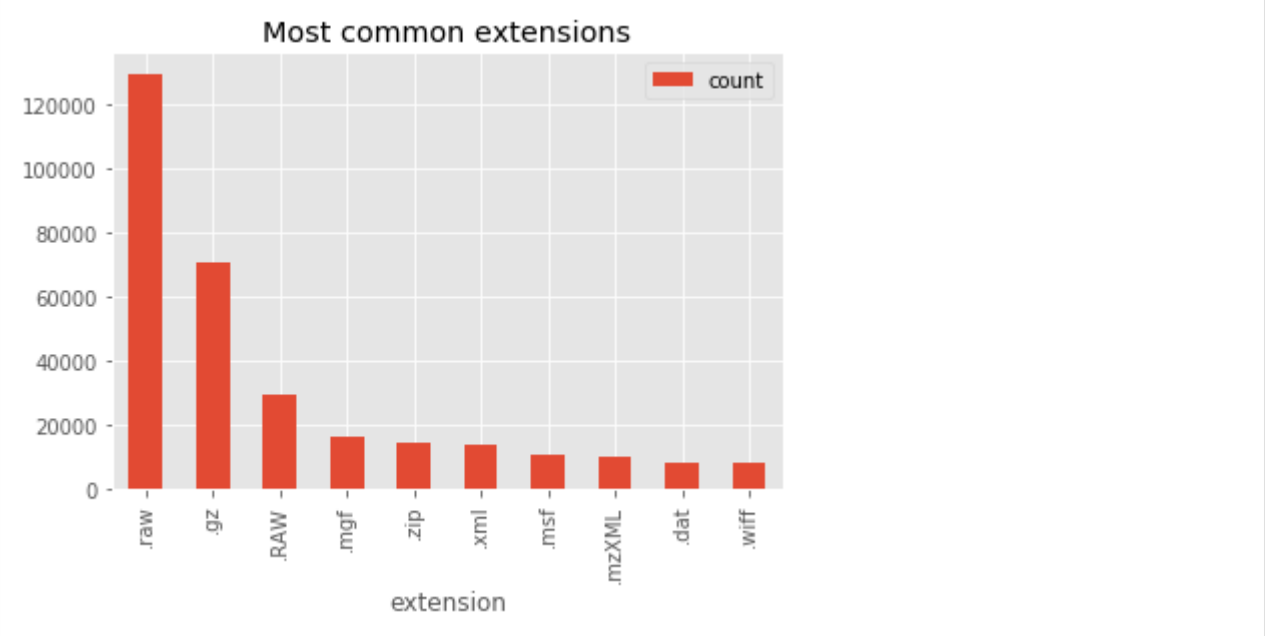
[4]:
```
import pandas

data_depth = path2insight.depth_counts(data)
pandas.Series(data_depth).plot.bar(title="Depth count")
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10b4905c0>
```



[5]:
```
data_top_extensions = path2insight.extension_counts(data).most_common(10)

# convert to dataframe
df_top_extensions = pandas.DataFrame(data_top_extensions,
                                     columns=['extension', 'count'])
df_top_extensions.set_index('extension', inplace=True)
df_top_extensions.plot.bar(title='Most common extensions')
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10ef05d68>
```



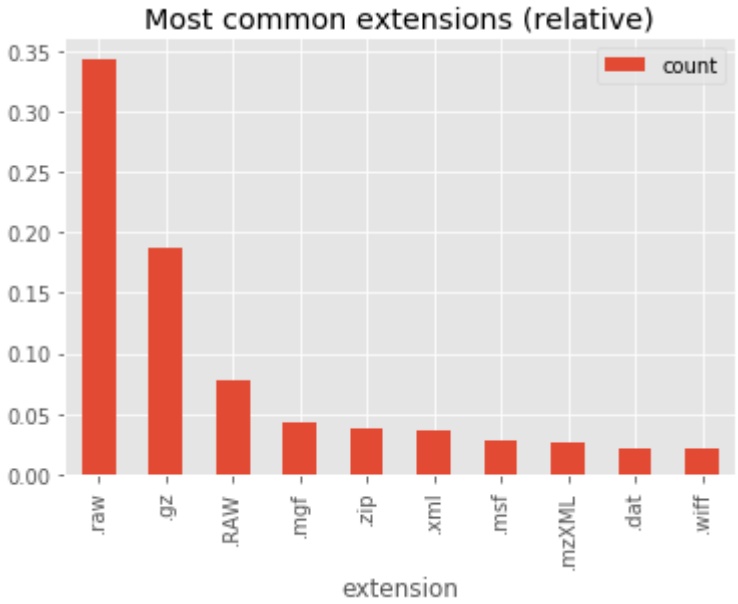
[6]:
```
data_top_extensions = path2insight.extension_counts(
    data, normalize=True).most_common(10)

# convert to dataframe
df_top_extensions = pandas.DataFrame(data_top_extensions,
                                     columns=['extension', 'count'])
```

(continues on next page)

(continued from previous page)

```
df_top_extensions.set_index('extension', inplace=True)
df_top_extensions.plot.bar(title='Most common extensions (relative)')
```

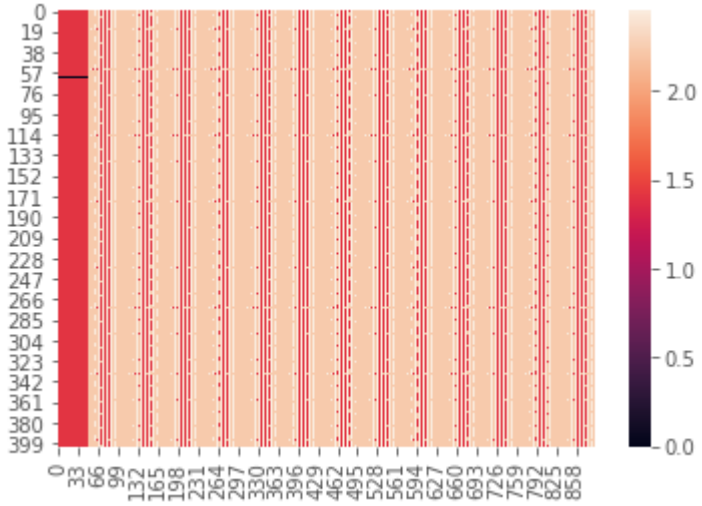
[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1146f0630>
```



6.2 Example 2 (Compare projects)

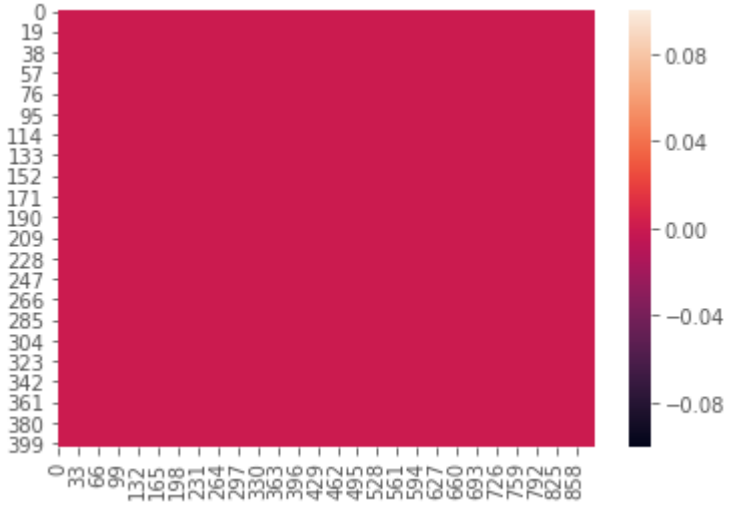
[7]:
```
import seaborn as sns

# select the two projects
project_PXD001787 = path2insight.select(data, level5='PXD001787')
project_PXD002010 = path2insight.select(data, level5='PXD002010')

# plot the similarity in tokens
m2 = path2insight.distance_on_extension(project_PXD001787, project_PXD002010)
sns.heatmap(m2, vmin=0)
```

[7]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2a21e4a8>
```



[8]:
```
import seaborn as sns

# select the two projects
project_PXD001787 = path2insight.select(data, level5='PXD001787')
project_PXD002010 = path2insight.select(data, level5='PXD002010')

# plot the similarity in tokens
m2 = path2insight.distance_on_depth(project_PXD001787, project_PXD002010)
sns.heatmap(m2, vmin=0)
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2a6462b0>
```



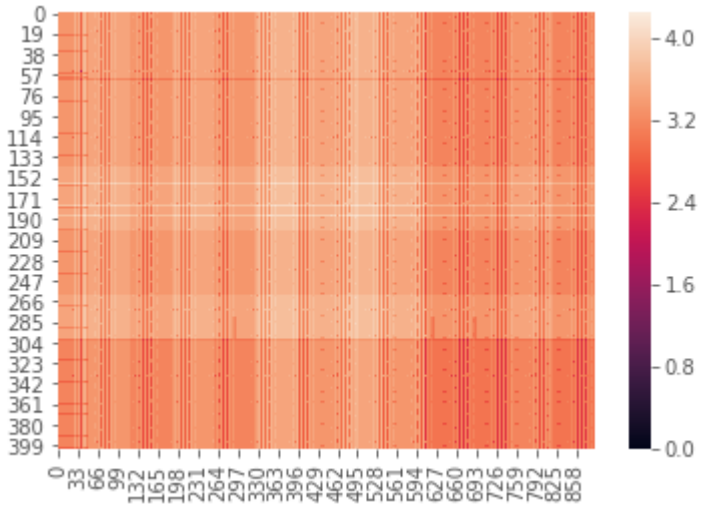
[9]:
```
import seaborn as sns

# select the two projects
project_PXD001787 = path2insight.select(data, level5='PXD001787')
project_PXD002010 = path2insight.select(data, level5='PXD002010')

# plot the similarity in tokens
m2 = path2insight.distance_on_token(project_PXD001787, project_PXD002010)
```

(continues on next page)

(continued from previous page)

```
sns.heatmap(m2, vmin=0)
```

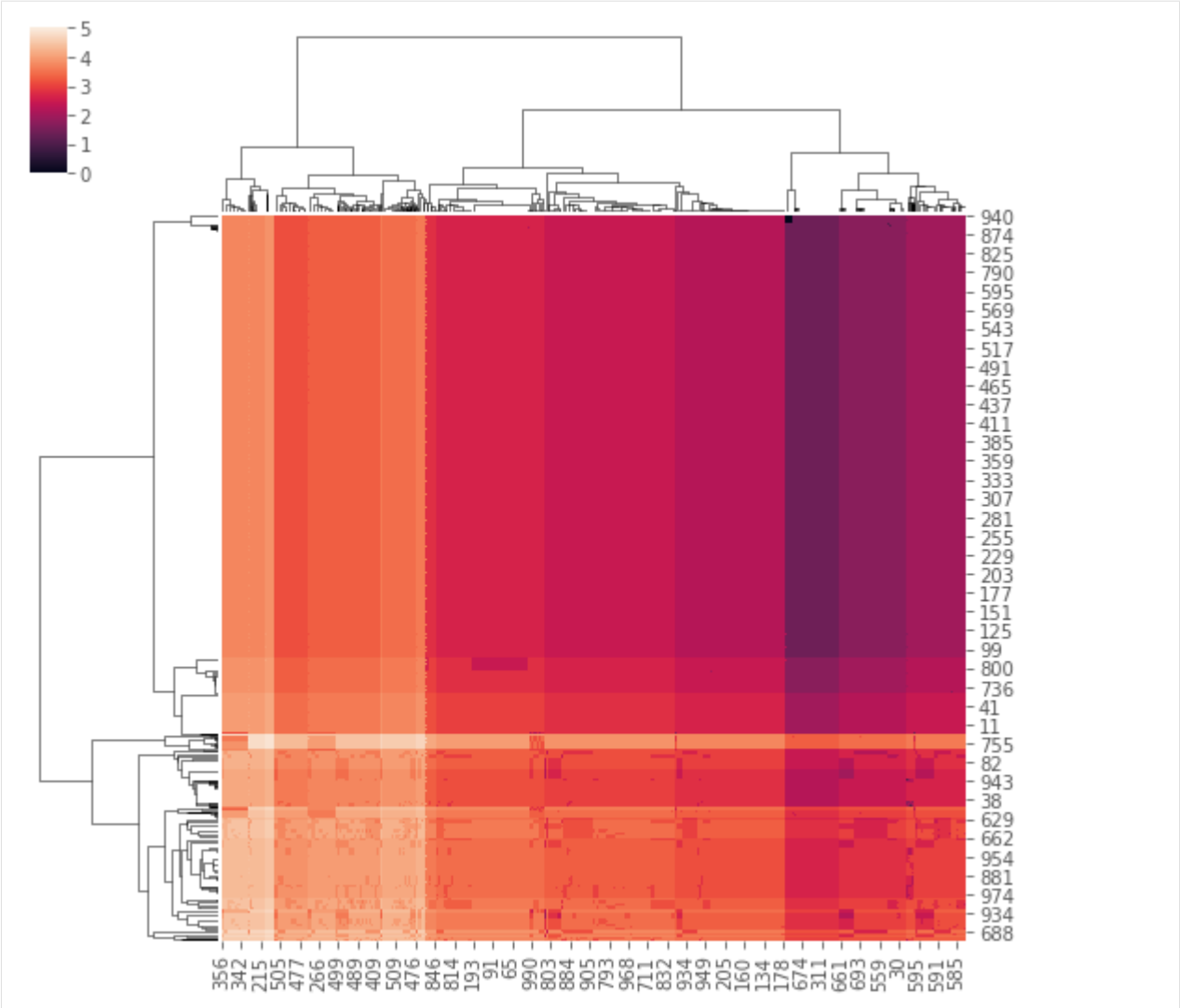
```
[9]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2bd5f748>
```



6.3 Example 3 (Clustering)

```
[10]: import seaborn as sns

m2 = path2insight.distance_on_token(data[0:1000], data[1000:2000])
sns.clustermap(m2, figsize=(9, 9))
```

```
[10]: <seaborn.matrix.ClusterGrid at 0x1146eb3c8>
```


5
4
3
2
1
0
940
874
825
790
595
569
543
517
491
465
437
411
385
359
333
307
281
255
229
203
177
151
125
99
800
736
41
11
755
82
943
38
629
662
954
881
974
934
688
356
342
215
505
477
266
499
489
409
509
476
846
814
193
91
65
990
803
884
905
793
968
711
832
934
949
205
160
134
178
674
311
661
693
559
30
595
591
585

CHAPTER 7

API Reference

7.1 FilePath Objects

class path2insight.**WindowsFilePath**(**args*)

Object to analyse Windows file or folder path.

The WindowsFilePath inherits from `pathlib.PureWindowsPath` in Python >3.4. See the documentation for all properties and methods.

Examples:

```
>>> p = WindowsFilePath("D://Documents/ProjectX/DEMO code.py")
>>> str(p)
'D:\Documents\ProjectX\DEMO code.py'
>>> p.lower_name().tokenize_stem()
['demo', 'code']
>>> p.extension
'.py'
```

See also:

https://docs.python.org/3/library/pathlib.html#methods-and-properties

anchor
The concatenation of the drive and root, or ‘’.

as_posix()
Return the string representation of the path with forward (/) slashes.

as_uri()
Return the path as a ‘file’ URI.

capitalize(**args*, ***kwargs*)
Apply string function ‘capitalize’ to filename parts.

capitalize_name(**args*, ***kwargs*)
Apply string function ‘capitalize’ to the name.

capitalize_stem(**args*, ***kwargs*)
Apply string function 'capitalize' to the stem.

casefold(**args*, ***kwargs*)
Apply string function 'casefold' to filename parts.

casefold_name(**args*, ***kwargs*)
Apply string function 'casefold' to the name.

casefold_stem(**args*, ***kwargs*)
Apply string function 'casefold' to the stem.

center(**args*, ***kwargs*)
Apply string function 'center' to filename parts.

center_name(**args*, ***kwargs*)
Apply string function 'center' to the name.

center_stem(**args*, ***kwargs*)
Apply string function 'center' to the stem.

count(**args*, ***kwargs*)
Apply string function 'count' to filename parts.

count_name(**args*, ***kwargs*)
Apply string function 'count' to the name.

count_stem(**args*, ***kwargs*)
Apply string function 'count' to the stem.

depth
Compute the depth of the path.

Example

```
>>> WindowsFilePath('R:/Armel/path2insight/demo.py').depth
3
```

drive
The drive prefix (letter or UNC path), if any.

encode(**args*, ***kwargs*)
Apply string function 'encode' to filename parts.

encode_name(**args*, ***kwargs*)
Apply string function 'encode' to the name.

encode_stem(**args*, ***kwargs*)
Apply string function 'encode' to the stem.

endswith(**args*, ***kwargs*)
Apply string function 'endswith' to filename parts.

endswith_name(**args*, ***kwargs*)
Apply string function 'endswith' to the name.

endswith_stem(**args*, ***kwargs*)
Apply string function 'endswith' to the stem.

expandtabs(**args*, ***kwargs*)
Apply string function 'expandtabs' to filename parts.

expandtabs_name(**args*, ***kwargs*)
Apply string function 'expandtabs' to the name.

expandtabs_stem(**args*, ***kwargs*)
Apply string function 'expandtabs' to the stem.

extension
Masked property from self.suffix

extensions
Masked property from self.suffixes

find(**args*, ***kwargs*)
Apply string function 'find' to filename parts.

find_name(**args*, ***kwargs*)
Apply string function 'find' to the name.

find_stem(**args*, ***kwargs*)
Apply string function 'find' to the stem.

format(**args*, ***kwargs*)
Apply string function 'format' to filename parts.

format_map(**args*, ***kwargs*)
Apply string function 'format_map' to filename parts.

format_map_name(**args*, ***kwargs*)
Apply string function 'format_map' to the name.

format_map_stem(**args*, ***kwargs*)
Apply string function 'format_map' to the stem.

format_name(**args*, ***kwargs*)
Apply string function 'format' to the name.

format_stem(**args*, ***kwargs*)
Apply string function 'format' to the stem.

index(**args*, ***kwargs*)
Apply string function 'index' to filename parts.

index_name(**args*, ***kwargs*)
Apply string function 'index' to the name.

index_stem(**args*, ***kwargs*)
Apply string function 'index' to the stem.

is_absolute()
True if the path is absolute (has both a root and, if applicable, a drive).

is_reserved()
Return True if the path contains one of the special names reserved by the system, if any.

isalnum(**args*, ***kwargs*)
Apply string function 'isalnum' to filename parts.

isalnum_name(**args*, ***kwargs*)
Apply string function 'isalnum' to the name.

isalnum_stem(**args*, ***kwargs*)
Apply string function 'isalnum' to the stem.

isalpha(**args*, ***kwargs*)
Apply string function 'isalpha' to filename parts.

isalpha_name(**args*, ***kwargs*)
Apply string function 'isalpha' to the name.

isalpha_stem(**args*, ***kwargs*)
Apply string function 'isalpha' to the stem.

isascii(**args*, ***kwargs*)
Apply string function 'isascii' to filename parts.

isascii_name(**args*, ***kwargs*)
Apply string function 'isascii' to the name.

isascii_stem(**args*, ***kwargs*)
Apply string function 'isascii' to the stem.

isdecimal(**args*, ***kwargs*)
Apply string function 'isdecimal' to filename parts.

isdecimal_name(**args*, ***kwargs*)
Apply string function 'isdecimal' to the name.

isdecimal_stem(**args*, ***kwargs*)
Apply string function 'isdecimal' to the stem.

isdigit(**args*, ***kwargs*)
Apply string function 'isdigit' to filename parts.

isdigit_name(**args*, ***kwargs*)
Apply string function 'isdigit' to the name.

isdigit_stem(**args*, ***kwargs*)
Apply string function 'isdigit' to the stem.

isidentifier(**args*, ***kwargs*)
Apply string function 'isidentifier' to filename parts.

isidentifier_name(**args*, ***kwargs*)
Apply string function 'isidentifier' to the name.

isidentifier_stem(**args*, ***kwargs*)
Apply string function 'isidentifier' to the stem.

islower(**args*, ***kwargs*)
Apply string function 'islower' to filename parts.

islower_name(**args*, ***kwargs*)
Apply string function 'islower' to the name.

islower_stem(**args*, ***kwargs*)
Apply string function 'islower' to the stem.

isnumeric(**args*, ***kwargs*)
Apply string function 'isnumeric' to filename parts.

isnumeric_name(**args*, ***kwargs*)
Apply string function 'isnumeric' to the name.

isnumeric_stem(**args*, ***kwargs*)
Apply string function 'isnumeric' to the stem.

isprintable(**args*, ***kwargs*)
Apply string function 'isprintable' to filename parts.

isprintable_name(**args*, ***kwargs*)
Apply string function 'isprintable' to the name.

isprintable_stem(**args*, ***kwargs*)
Apply string function 'isprintable' to the stem.

isspace(**args*, ***kwargs*)
Apply string function 'isspace' to filename parts.

isspace_name(**args*, ***kwargs*)
Apply string function 'isspace' to the name.

isspace_stem(**args*, ***kwargs*)
Apply string function 'isspace' to the stem.

istitle(**args*, ***kwargs*)
Apply string function 'istitle' to filename parts.

istitle_name(**args*, ***kwargs*)
Apply string function 'istitle' to the name.

istitle_stem(**args*, ***kwargs*)
Apply string function 'istitle' to the stem.

isupper(**args*, ***kwargs*)
Apply string function 'isupper' to filename parts.

isupper_name(**args*, ***kwargs*)
Apply string function 'isupper' to the name.

isupper_stem(**args*, ***kwargs*)
Apply string function 'isupper' to the stem.

join(**args*, ***kwargs*)
Apply string function 'join' to filename parts.

join_name(**args*, ***kwargs*)
Apply string function 'join' to the name.

join_stem(**args*, ***kwargs*)
Apply string function 'join' to the stem.

joinpath(**args*)
Combine this path with one or several arguments, and return a new path representing either a subpath (if all arguments are relative paths) or a totally different path (if one of the arguments is anchored).

ljust(**args*, ***kwargs*)
Apply string function 'ljust' to filename parts.

ljust_name(**args*, ***kwargs*)
Apply string function 'ljust' to the name.

ljust_stem(**args*, ***kwargs*)
Apply string function 'ljust' to the stem.

lower(**args*, ***kwargs*)
Apply string function 'lower' to filename parts.

lower_name(**args*, ***kwargs*)
Apply string function 'lower' to the name.

lower_stem(**args*, ***kwargs*)
Apply string function 'lower' to the stem.

lstrip(**args*, ***kwargs*)
Apply string function 'lstrip' to filename parts.

lstrip_name(**args*, ***kwargs*)
Apply string function 'lstrip' to the name.

lstrip_stem(**args*, ***kwargs*)
Apply string function 'lstrip' to the stem.

maketrans(**args*, ***kwargs*)
Apply string function 'maketrans' to filename parts.

maketrans_name(**args*, ***kwargs*)
Apply string function 'maketrans' to the name.

maketrans_stem(**args*, ***kwargs*)
Apply string function 'maketrans' to the stem.

match(*path_pattern*)
Return True if this path matches the given pattern.

name
The final path component, if any.

parent
The logical parent of the path.

parents
A sequence of this path's logical parents.

partition(**args*, ***kwargs*)
Apply string function 'partition' to filename parts.

partition_name(**args*, ***kwargs*)
Apply string function 'partition' to the name.

partition_stem(**args*, ***kwargs*)
Apply string function 'partition' to the stem.

parts
An object providing sequence-like access to the components in the filesystem path.

relative_to(**other*)
Return the relative path to another path identified by the passed arguments. If the operation is not possible (because this is not a subpath of the other path), raise ValueError.

replace(**args*, ***kwargs*)
Apply string function 'replace' to filename parts.

replace_name(**args*, ***kwargs*)
Apply string function 'replace' to the name.

replace_stem(**args*, ***kwargs*)
Apply string function 'replace' to the stem.

rfind(**args*, ***kwargs*)
Apply string function 'rfind' to filename parts.

rfind_name(**args*, ***kwargs*)
Apply string function 'rfind' to the name.

rfind_stem(**args*, ***kwargs*)
Apply string function 'rfind' to the stem.

rindex(**args*, ***kwargs*)
Apply string function 'rindex' to filename parts.

rindex_name(**args*, ***kwargs*)
Apply string function 'rindex' to the name.

rindex_stem(**args*, ***kwargs*)
Apply string function 'rindex' to the stem.

rjust(**args*, ***kwargs*)
Apply string function 'rjust' to filename parts.

rjust_name(**args*, ***kwargs*)
Apply string function 'rjust' to the name.

rjust_stem(**args*, ***kwargs*)
Apply string function 'rjust' to the stem.

root
The root of the path, if any.

rpartition(**args*, ***kwargs*)
Apply string function 'rpartition' to filename parts.

rpartition_name(**args*, ***kwargs*)
Apply string function 'rpartition' to the name.

rpartition_stem(**args*, ***kwargs*)
Apply string function 'rpartition' to the stem.

rsplit(**args*, ***kwargs*)
Apply string function 'rsplit' to filename parts.

rsplit_name(**args*, ***kwargs*)
Apply string function 'rsplit' to the name.

rsplit_stem(**args*, ***kwargs*)
Apply string function 'rsplit' to the stem.

rstrip(**args*, ***kwargs*)
Apply string function 'rstrip' to filename parts.

rstrip_name(**args*, ***kwargs*)
Apply string function 'rstrip' to the name.

rstrip_stem(**args*, ***kwargs*)
Apply string function 'rstrip' to the stem.

split(**args*, ***kwargs*)
Apply string function 'split' to filename parts.

split_name(**args*, ***kwargs*)
Apply string function 'split' to the name.

split_stem(**args*, ***kwargs*)
Apply string function 'split' to the stem.

splitlines(**args*, ***kwargs*)
Apply string function 'splitlines' to filename parts.

splitlines_name(**args*, ***kwargs*)
Apply string function 'splitlines' to the name.

splitlines_stem(**args*, ***kwargs*)
Apply string function 'splitlines' to the stem.

startswith(**args*, ***kwargs*)
Apply string function 'startswith' to filename parts.

startswith_name(**args*, ***kwargs*)
Apply string function 'startswith' to the name.

startswith_stem(**args*, ***kwargs*)
Apply string function 'startswith' to the stem.

stem
The final path component, minus its last suffix.

strip(**args*, ***kwargs*)
Apply string function 'strip' to filename parts.

strip_name(**args*, ***kwargs*)
Apply string function 'strip' to the name.

strip_stem(**args*, ***kwargs*)
Apply string function 'strip' to the stem.

suffix
The final component's last suffix, if any.

suffixes
A list of the final component's suffixes, if any.

swapcase(**args*, ***kwargs*)
Apply string function 'swapcase' to filename parts.

swapcase_name(**args*, ***kwargs*)
Apply string function 'swapcase' to the name.

swapcase_stem(**args*, ***kwargs*)
Apply string function 'swapcase' to the stem.

title(**args*, ***kwargs*)
Apply string function 'title' to filename parts.

title_name(**args*, ***kwargs*)
Apply string function 'title' to the name.

title_stem(**args*, ***kwargs*)
Apply string function 'title' to the stem.

tokenize(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*, *exclude_extension=True*)
Tokenise the name (without extension)

tokenize_name(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*)
Tokenise the name

tokenize_stem(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*)
Tokenise the name

translate(**args*, ***kwargs*)
Apply string function 'translate' to filename parts.

translate_name(**args*, ***kwargs*)
Apply string function 'translate' to the name.

translate_stem(**args*, ***kwargs*)
Apply string function 'translate' to the stem.

upper(**args*, ***kwargs*)
Apply string function 'upper' to filename parts.

upper_name(**args*, ***kwargs*)
Apply string function 'upper' to the name.

upper_stem(**args*, ***kwargs*)
Apply string function 'upper' to the stem.

with_name(*name*)
Return a new path with the file name changed.

with_suffix(*suffix*)
Return a new path with the file suffix changed. If the path has no suffix, add given suffix. If the given suffix is an empty string, remove the suffix from the path.

zfill(**args*, ***kwargs*)
Apply string function 'zfill' to filename parts.

zfill_name(**args*, ***kwargs*)
Apply string function 'zfill' to the name.

zfill_stem(**args*, ***kwargs*)
Apply string function 'zfill' to the stem.

class path2insight.**PosixFilePath**(**args*)
Object to analyse Posix file or folder path.

The WindowsFilePath inherits from pathlib.PureWindowsPath in Python >3.4. See https://docs.python.org/3/library/pathlib.html#methods-and-properties for all properties and methods.

anchor
The concatenation of the drive and root, or ''.

as_posix()
Return the string representation of the path with forward (/) slashes.

as_uri()
Return the path as a 'file' URI.

capitalize(**args*, ***kwargs*)
Apply string function 'capitalize' to filename parts.

capitalize_name(**args*, ***kwargs*)
Apply string function 'capitalize' to the name.

capitalize_stem(**args*, ***kwargs*)
Apply string function 'capitalize' to the stem.

casefold(**args*, ***kwargs*)
Apply string function 'casefold' to filename parts.

casefold_name(**args*, ***kwargs*)
Apply string function 'casefold' to the name.

casefold_stem(**args*, ***kwargs*)
Apply string function 'casefold' to the stem.

center(**args*, ***kwargs*)
Apply string function 'center' to filename parts.

center_name(**args*, ***kwargs*)
Apply string function 'center' to the name.

center_stem(**args*, ***kwargs*)
Apply string function 'center' to the stem.

count(**args*, ***kwargs*)
Apply string function 'count' to filename parts.

count_name(**args*, ***kwargs*)
Apply string function 'count' to the name.

count_stem(**args*, ***kwargs*)
Apply string function 'count' to the stem.

depth
Compute the depth of the path.

Example

```
>>> WindowsFilePath('R:/Armel/path2insight/demo.py').depth
3
```

drive
The drive prefix (letter or UNC path), if any.

encode(**args*, ***kwargs*)
Apply string function 'encode' to filename parts.

encode_name(**args*, ***kwargs*)
Apply string function 'encode' to the name.

encode_stem(**args*, ***kwargs*)
Apply string function 'encode' to the stem.

endswith(**args*, ***kwargs*)
Apply string function 'endswith' to filename parts.

endswith_name(**args*, ***kwargs*)
Apply string function 'endswith' to the name.

endswith_stem(**args*, ***kwargs*)
Apply string function 'endswith' to the stem.

expandtabs(**args*, ***kwargs*)
Apply string function 'expandtabs' to filename parts.

expandtabs_name(**args*, ***kwargs*)
Apply string function 'expandtabs' to the name.

expandtabs_stem(**args*, ***kwargs*)
Apply string function 'expandtabs' to the stem.

extension
Masked property from self.suffix

extensions
Masked property from self.suffixes

find(**args*, ***kwargs*)
Apply string function 'find' to filename parts.

find_name(**args*, ***kwargs*)
Apply string function 'find' to the name.

find_stem(**args*, ***kwargs*)
Apply string function 'find' to the stem.

format(**args*, ***kwargs*)
Apply string function 'format' to filename parts.

format_map(**args*, ***kwargs*)
Apply string function 'format_map' to filename parts.

format_map_name(**args*, ***kwargs*)
Apply string function 'format_map' to the name.

format_map_stem(**args*, ***kwargs*)
Apply string function 'format_map' to the stem.

format_name(**args*, ***kwargs*)
Apply string function 'format' to the name.

format_stem(**args*, ***kwargs*)
Apply string function 'format' to the stem.

index(**args*, ***kwargs*)
Apply string function 'index' to filename parts.

index_name(**args*, ***kwargs*)
Apply string function 'index' to the name.

index_stem(**args*, ***kwargs*)
Apply string function 'index' to the stem.

is_absolute()
True if the path is absolute (has both a root and, if applicable, a drive).

is_reserved()
Return True if the path contains one of the special names reserved by the system, if any.

isalnum(**args*, ***kwargs*)
Apply string function 'isalnum' to filename parts.

isalnum_name(**args*, ***kwargs*)
Apply string function 'isalnum' to the name.

isalnum_stem(**args*, ***kwargs*)
Apply string function 'isalnum' to the stem.

isalpha(**args*, ***kwargs*)
Apply string function 'isalpha' to filename parts.

isalpha_name(**args*, ***kwargs*)
Apply string function 'isalpha' to the name.

isalpha_stem(**args*, ***kwargs*)
Apply string function 'isalpha' to the stem.

isascii(**args*, ***kwargs*)
Apply string function 'isascii' to filename parts.

isascii_name(**args*, ***kwargs*)
Apply string function 'isascii' to the name.

isascii_stem(**args*, ***kwargs*)
Apply string function 'isascii' to the stem.

isdecimal(**args*, ***kwargs*)
Apply string function 'isdecimal' to filename parts.

isdecimal_name(**args*, ***kwargs*)
Apply string function 'isdecimal' to the name.

isdecimal_stem(**args*, ***kwargs*)
Apply string function 'isdecimal' to the stem.

isdigit(**args*, ***kwargs*)
Apply string function 'isdigit' to filename parts.

isdigit_name(**args*, ***kwargs*)
Apply string function 'isdigit' to the name.

isdigit_stem(**args*, ***kwargs*)
Apply string function 'isdigit' to the stem.

isidentifier(**args*, ***kwargs*)
Apply string function 'isidentifier' to filename parts.

isidentifier_name(**args*, ***kwargs*)
Apply string function 'isidentifier' to the name.

isidentifier_stem(**args*, ***kwargs*)
Apply string function 'isidentifier' to the stem.

islower(**args*, ***kwargs*)
Apply string function 'islower' to filename parts.

islower_name(**args*, ***kwargs*)
Apply string function 'islower' to the name.

islower_stem(**args*, ***kwargs*)
Apply string function 'islower' to the stem.

isnumeric(**args*, ***kwargs*)
Apply string function 'isnumeric' to filename parts.

isnumeric_name(**args*, ***kwargs*)
Apply string function 'isnumeric' to the name.

isnumeric_stem(**args*, ***kwargs*)
Apply string function 'isnumeric' to the stem.

isprintable(**args*, ***kwargs*)
Apply string function 'isprintable' to filename parts.

isprintable_name(**args*, ***kwargs*)
Apply string function 'isprintable' to the name.

isprintable_stem(**args*, ***kwargs*)
Apply string function 'isprintable' to the stem.

isspace(**args*, ***kwargs*)
Apply string function 'isspace' to filename parts.

isspace_name(**args*, ***kwargs*)
Apply string function 'isspace' to the name.

isspace_stem(**args*, ***kwargs*)
Apply string function 'isspace' to the stem.

istitle(**args*, ***kwargs*)
Apply string function 'istitle' to filename parts.

istitle_name(**args*, ***kwargs*)
Apply string function 'istitle' to the name.

istitle_stem(**args*, ***kwargs*)
Apply string function 'istitle' to the stem.

isupper(**args*, ***kwargs*)
Apply string function 'isupper' to filename parts.

isupper_name(**args*, ***kwargs*)
Apply string function 'isupper' to the name.

isupper_stem(**args*, ***kwargs*)
Apply string function 'isupper' to the stem.

join(**args*, ***kwargs*)
Apply string function 'join' to filename parts.

join_name(**args*, ***kwargs*)
Apply string function 'join' to the name.

join_stem(**args*, ***kwargs*)
Apply string function 'join' to the stem.

joinpath(**args*)
Combine this path with one or several arguments, and return a new path representing either a subpath (if all arguments are relative paths) or a totally different path (if one of the arguments is anchored).

ljust(**args*, ***kwargs*)
Apply string function 'ljust' to filename parts.

ljust_name(**args*, ***kwargs*)
Apply string function 'ljust' to the name.

ljust_stem(**args*, ***kwargs*)
Apply string function 'ljust' to the stem.

lower(**args*, ***kwargs*)
Apply string function 'lower' to filename parts.

lower_name(**args*, ***kwargs*)
Apply string function 'lower' to the name.

lower_stem(**args*, ***kwargs*)
Apply string function 'lower' to the stem.

lstrip(**args*, ***kwargs*)
Apply string function 'lstrip' to filename parts.

lstrip_name(**args*, ***kwargs*)
Apply string function 'lstrip' to the name.

lstrip_stem(**args*, ***kwargs*)
Apply string function 'lstrip' to the stem.

maketrans(**args*, ***kwargs*)
Apply string function 'maketrans' to filename parts.

maketrans_name(**args*, ***kwargs*)
Apply string function 'maketrans' to the name.

maketrans_stem(**args*, ***kwargs*)
Apply string function 'maketrans' to the stem.

match(*path_pattern*)
Return True if this path matches the given pattern.

name
The final path component, if any.

parent
The logical parent of the path.

parents
A sequence of this path's logical parents.

partition(**args*, ***kwargs*)
Apply string function 'partition' to filename parts.

partition_name(**args*, ***kwargs*)
Apply string function 'partition' to the name.

partition_stem(**args*, ***kwargs*)
Apply string function 'partition' to the stem.

parts
An object providing sequence-like access to the components in the filesystem path.

relative_to(**other*)
Return the relative path to another path identified by the passed arguments. If the operation is not possible (because this is not a subpath of the other path), raise ValueError.

replace(**args*, ***kwargs*)
Apply string function 'replace' to filename parts.

replace_name(**args*, ***kwargs*)
Apply string function 'replace' to the name.

replace_stem(**args*, ***kwargs*)
Apply string function 'replace' to the stem.

rfind(**args*, ***kwargs*)
Apply string function 'rfind' to filename parts.

rfind_name(**args*, ***kwargs*)
Apply string function 'rfind' to the name.

rfind_stem(**args*, ***kwargs*)
Apply string function 'rfind' to the stem.

rindex(**args*, ***kwargs*)
Apply string function 'rindex' to filename parts.

rindex_name(**args*, ***kwargs*)
Apply string function 'rindex' to the name.

rindex_stem(**args*, ***kwargs*)
Apply string function 'rindex' to the stem.

rjust(**args*, ***kwargs*)
Apply string function 'rjust' to filename parts.

rjust_name(**args*, ***kwargs*)
Apply string function 'rjust' to the name.

rjust_stem(**args*, ***kwargs*)
Apply string function 'rjust' to the stem.

root
The root of the path, if any.

rpartition(**args*, ***kwargs*)
Apply string function 'rpartition' to filename parts.

rpartition_name(**args*, ***kwargs*)
Apply string function 'rpartition' to the name.

rpartition_stem(**args*, ***kwargs*)
Apply string function 'rpartition' to the stem.

rsplit(**args*, ***kwargs*)
Apply string function 'rsplit' to filename parts.

rsplit_name(**args*, ***kwargs*)
Apply string function 'rsplit' to the name.

rsplit_stem(**args*, ***kwargs*)
Apply string function 'rsplit' to the stem.

rstrip(**args*, ***kwargs*)
Apply string function 'rstrip' to filename parts.

rstrip_name(**args*, ***kwargs*)
Apply string function 'rstrip' to the name.

rstrip_stem(**args*, ***kwargs*)
Apply string function 'rstrip' to the stem.

split(**args*, ***kwargs*)
Apply string function 'split' to filename parts.

split_name(**args*, ***kwargs*)
Apply string function 'split' to the name.

split_stem(**args*, ***kwargs*)
Apply string function 'split' to the stem.

splitlines(**args*, ***kwargs*)
Apply string function 'splitlines' to filename parts.

splitlines_name(**args*, ***kwargs*)
Apply string function 'splitlines' to the name.

splitlines_stem(**args*, ***kwargs*)
Apply string function 'splitlines' to the stem.

startswith(**args*, ***kwargs*)
Apply string function 'startswith' to filename parts.

startswith_name(**args*, ***kwargs*)
Apply string function 'startswith' to the name.

startswith_stem(**args*, ***kwargs*)
Apply string function 'startswith' to the stem.

stem
The final path component, minus its last suffix.

strip(**args*, ***kwargs*)
Apply string function 'strip' to filename parts.

strip_name(**args*, ***kwargs*)
Apply string function 'strip' to the name.

strip_stem(**args*, ***kwargs*)
Apply string function 'strip' to the stem.

suffix
The final component's last suffix, if any.

suffixes
A list of the final component's suffixes, if any.

swapcase(**args*, ***kwargs*)
Apply string function 'swapcase' to filename parts.

swapcase_name(**args*, ***kwargs*)
Apply string function 'swapcase' to the name.

swapcase_stem(**args*, ***kwargs*)
Apply string function 'swapcase' to the stem.

title(**args*, ***kwargs*)
Apply string function 'title' to filename parts.

title_name(**args*, ***kwargs*)
Apply string function 'title' to the name.

title_stem(**args*, ***kwargs*)
Apply string function 'title' to the stem.

tokenize(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*, *exclude_extension=True*)
Tokenise the name (without extension)

tokenize_name(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*)
Tokenise the name

tokenize_stem(*token_pattern='(?u)([a-zA-Z0-9\\:]+)(?=[^a-zA-Z0-9\\:]|$)'*)
Tokenise the name

translate(**args*, ***kwargs*)
Apply string function 'translate' to filename parts.

translate_name(**args*, ***kwargs*)
Apply string function 'translate' to the name.

translate_stem(**args*, ***kwargs*)
Apply string function 'translate' to the stem.

upper(**args*, ***kwargs*)
Apply string function 'upper' to filename parts.

upper_name(**args*, ***kwargs*)
Apply string function 'upper' to the name.

upper_stem(**args*, ***kwargs*)
Apply string function 'upper' to the stem.

with_name(*name*)
Return a new path with the file name changed.

with_suffix(*suffix*)
Return a new path with the file suffix changed. If the path has no suffix, add given suffix. If the given suffix is an empty string, remove the suffix from the path.

zfill(**args*, ***kwargs*)
Apply string function 'zfill' to filename parts.

zfill_name(**args*, ***kwargs*)
Apply string function 'zfill' to the name.

zfill_stem(**args*, ***kwargs*)
Apply string function 'zfill' to the stem.

7.2 Parsing

path2insight.**parse**(*obj*, *os_name=None*)
Parse (list of) file paths.

Parse a list with file paths into list of WindowsFilePath and PosixFilePath objects. This function can parse list, tuple, numpy.ndarray and pandas.Series. This is done with one of the following parsers: parse_from_pandas, parse_from_numpy or parse_from_list.

Example

```
>>> data = ['file1.xml', 'data/file1.txt', 'data/file2.txt']
>>> path2insight.parse(data, os_name='windows')
```

gives the same result as

```
>>> import pandas
>>> path2insight.parse(pandas.Series(data), os_name='windows')
```

Parameters

- **obj** (*list, tuple, numpy.ndarray, pandas.Series*) – An object such as a list, numpy array or pandas.Series with filepaths in the form of strings.
- **os_name** (*str*) – The operation system on with the filepaths are collected. The options are 'windows' or 'posix' for Windows and Posix system repectivily.

Returns Returns a list with WindowsFilePaths and PosixFilePaths.

Return_type list

path2insight.**parse_from_list**(*l*, *os_name=None*)
Parse a list with file paths.

See *path2insight.parse()* for additional information.

Parameters

- **obj** (*list*) – A list with filepaths in the form of strings.
- **os_name** (*str*) – The operation system on with the filepaths are collected. The options are 'windows' or 'posix' for Windows and Posix system repectivily.

Returns Returns a list with WindowsFilePaths and PosixFilePaths.

Return_type list

path2insight.**parse_from_numpy**(*np_object*, *os_name=None*)
Parse a numpy array with file paths.

See *path2insight.parse()* for additional information.

Parameters

- **obj** (*numpy.ndarray*) – A numpy.ndarray with filepaths in the form of strings.
- **os_name** (*str*) – The operation system on with the filepaths are collected. The options are 'windows' or 'posix' for Windows and Posix system repectivily.

Returns Returns a list with WindowsFilePaths and PosixFilePaths.

Return_type list

path2insight.**parse_from_pandas**(*pandas_object*, *os_name=None*)
Parse a series or dataframe with file paths.

See *path2insight.parse()* for additional information.

Parameters

- **obj** (*pandas.Series or pandas.DataFrame*) – An pandas.Series or pandas.DataFrame with filepaths in the form of strings.
- **os_name** (*str*) – The operation system on with the filepaths are collected. The options are 'windows' or 'posix' for Windows and Posix system repectivily.

Returns Returns a list with WindowsFilePaths and PosixFilePaths.

Return_type list

7.3 Handling

path2insight.**sort**(*paths*, *level=None*, *reverse=False*)
Sort a list of filepaths.

This function sorts a list of filepaths. The sorting can be based on parts of the (like folder of file) names. This is done with the key arguments.

Parameters

- **paths** (*list*) – A list of filepaths
- **level** (*(list of) int*) – List of positions which refer to the axis items. Default None.
- **reverse** (*bool*) – Reverse the sort.

Returns A sorted list.

Return_type list

Example

```
>>> path2insight.sort(data)
>>> path2insight.sort(data, key=1)
>>> path2insight.sort(data, key=1, reverse=True)
>>> path2insight.sort(data, key=[5, 4])
```

path2insight.**sample**(*data*, *n=None*)
Take a random sample of filepaths.

Parameters

- **paths** (*list*) – A list of filepaths
- **n** (*int, optional*) – The number of filepaths to return. If None, all filepaths are returned in a random order. Default None.

Returns A list with a sample of filepath.

Return_type list

path2insight.**select**(*paths*, ***kwargs*)

Select from a list of filepaths.

This function selects from a list of filepaths based on their part (like folder of file) names. This is done with the level arguments.

Parameters

- **paths** (*list*) – A list of filepaths
- **level0** (*(list of) str*) – The value(s) of the first level (root).
- **level1** (*(list of) str*) – The value(s) of the second level.
- **level*** (*(list of) str*) – The value(s) of the nth level.

Returns A list with the selection of matching filepath.

Return_type list

Note

One can use the value "*" to select all file paths. If a file path doesn't have a value on that level (because the level is higher than the number of parts), then the path is excluded from the selection. One can also use *True* instead of "*".

Example

Selection based on the name of a level.

```
>>> import path2insight
>>> data = [path2insight.WindowsFilePath("F:/data/file.txt"),
            path2insight.WindowsFilePath("F:/docs/file.xlsx"),
            path2insight.WindowsFilePath("F:/test/file.demo"),
            path2insight.WindowsFilePath("F:/README.txt")]
>>> path2insight.select(data, level1='data')
[path2insight.WindowsFilePath("F:/data/file.txt")]
```

Selection based on the existence of a level (wilcard). Path is only included when level exists.

```
>>> path2insight.select(data, level2='*')
[path2insight.WindowsFilePath("F:/data/file.txt"),
 path2insight.WindowsFilePath("F:/docs/file.xlsx"),
 path2insight.WindowsFilePath("F:/test/file.demo")]
```

path2insight.**select_re**(*paths*, ***kwargs*)

Select from a list of filepaths.

This function selects from a list of filepaths based on their part (like folder of file) names. This is done with the level arguments.

Parameters

- **paths** (*list*) – A list of filepaths
- **level0** (*(list of) str*) – The value(s) of the first level (root).

- **level1** (*(list of) str*) – The value(s) of the second level.
- **level*** (*(list of) str*) – The value(s) of the nth level.

Returns A list with the selection of matching filepath.

Return_type list

Example

Selection based on the name of a level.

```
>>> import path2insight
>>> data = [path2insight.WindowsFilePath("F:/data/file.txt"),
            path2insight.WindowsFilePath("F:/docs/file.xlsx"),
            path2insight.WindowsFilePath("F:/test/file.demo"),
            path2insight.WindowsFilePath("F:/README.txt")]
>>> path2insight.select(data, level1=r"[A-Z]")
[path2insight.WindowsFilePath("F:/README.txt")]
```

Select all paths starting with the letter d on the first level.

```
>>> path2insight.select(data, level1=r"^d")
[path2insight.WindowsFilePath("F:/data/file.txt"),
 path2insight.WindowsFilePath("F:/docs/file.xlsx")]
```

7.4 Explore

path2insight.explore.stats.**depth_counts**(*x*, *normalize=False*, *center=None*)

Count the filepath-depths.

This function counts the filepath depths of a list of filepaths. The function returns a Python collections.Counter object. This Counter object can be used to compute the most common depths or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to determine the depth of.
- **normalize** (*bool*) – Normalize the Counter result. Default False.
- **center** (*str, NoneType, callable*) – Method to correct the offset of the data. Options are 'mean' or callable. Default None.

Returns filepath depths counted

Return type collections.Counter

Example

```
>>> path2insight.depth_counts(list_of_filepaths)
Counter({5: 32, 6: 654, 7: 284, 8: 13, 9: 11, 10: 1, 11: 4, 13: 1})
```

Note

To get a Python dict, simply wrap the Counter object with dict().

path2insight.explore.stats.**drive_counts**(*x*, *lower=False*, *normalize=False*)
Count the drives of the paths.

This function counts the drives of a list of filepaths. The function returns a Python collections.Counter object. This Counter object can be used to compute the most common drives or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to count the stems of.
- **lower** (*boolean*) – Convert the drive to lower before counting.
- **normalize** (*bool*) – Normalize the Counter result. Default False.

Returns drives counted

Return type collections.Counter

Note

To get a Python dict, simply wrap the Counter object with dict().

path2insight.explore.stats.**extension_chisquare**(*x*, *y=None*, *lower=True*)
Calculates a one-way chi square test for file extensions.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with y.
- **y** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with x.
- **lower** (*boolean*) – Convert the extensions to lower before counting.

Returns The test result.

Return type scipy.stats.Power_divergenceResult

path2insight.explore.stats.**extension_counts**(*x*, *lower=False*, *normalize=False*)
Count the extensions of the filenames.

This function counts the name extensions of a list of filepaths. The function returns a Python collections.Counter object. This Counter object can be used to compute the most common extensions or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to determine the extension count of.
- **lower** (*boolean*) – Convert the extensions to lower before counting.
- **normalize** (*bool*) – Normalize the Counter result. Default False.

Returns extensions counted

Return type collections.Counter

Example

```
>>> path2insight.extension_counts(filepaths_list)
Counter({'.zip': 42, '.raw': 3, '.txt': 12, '.docx': 1})
```

(continues on next page)

(continued from previous page)

```
>>> path2insight.extension_counts(filepaths_list).most_common(3)
[('.zip', 42), ('.txt', 12), ('.raw', 3)]
```

Note

To get a Python `dict`, simply wrap the Counter object with `dict()`.

path2insight.explore.stats.**n_extension_counts**(*x*)
[CHANGE FUNCTION NAME]Count the number of extensions.

path2insight.explore.stats.**name_chisquare**(*x*, *y=None*, *lower=True*)
Calculates a one-way chi square test for file names.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with y.
- **y** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with x.
- **lower** (*boolean*) – Convert the extensions to lower before counting.

Returns The test result.

Return type scipy.stats.Power_divergenceResult

path2insight.explore.stats.**name_counts**(*x*, *lower=False*, *normalize=False*)
Count the names.

This function counts the names of a list of filepaths. The function returns a Python `collections.Counter` object. This Counter object can be used to compute the most common names or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to count the names of.
- **lower** (*boolean*) – Convert the filenames to lower before counting.
- **normalize** (*bool*) – Normalize the Counter result. Default False.

Returns names counted

Return type collections.Counter

Note

To get a Python `dict`, simply wrap the Counter object with `dict()`.

path2insight.explore.stats.**stem_chisquare**(*x*, *y=None*, *lower=True*)
Calculates a one-way chi square test for file name stems.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with y.
- **y** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to compare with x.
- **lower** (*boolean*) – Convert the extensions to lower before counting.

Returns The test result.

Return type scipy.stats.Power_divergenceResult

path2insight.explore.stats.**stem_counts**(*x*, *lower=False*, *normalize=False*)

Count the stems.

This function counts the stems of a list of filepaths. The function returns a Python `collections.Counter` object. This Counter object can be used to compute the most common stems or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to count the stems of.
- **lower** (*boolean*) – Convert the stems to lower before counting.
- **normalize** (*bool*) – Normalize the Counter result. Default False.

Returns stems counted

Return type collections.Counter

Note

To get a Python `dict`, simply wrap the Counter object with `dict()`.

path2insight.explore.stats.**token_counts**(*x*, *tokenizer=<function default_tokenizer>*, *lower=False*, *parents=False*, *stem=True*, *extension=False*, *normalize=False*)

Count the tokens in the paths.

This function counts the tokens of a list of filepaths. Use boolean settings to include the parents, stem and extension. The function returns a Python `collections.Counter` object. This Counter object can be used to compute the most common tokens or substract other Counter objects. For all options, see the Python documentation.

Parameters

- **x** (*list, tuple, array of WindowsFilePath or PosixFilePath objects*) – Paths to count the stems of.
- **parents** (*bool*) – tokenize the parents
- **stem** (*bool*) – tokenize the stem
- **extension** (*bool*) – tokenisze the extension
- **lower** (*boolean*) – Convert the filepath to lower before counting.
- **normalize** (*bool*) – Normalize the Counter result. Default False.

Returns drives counted

Return type collections.Counter

Example

```
>>> path2insight.token_counts(data).most_common(3)
[('FUNC001', 288), ('LTQ', 173), ('FUNCTNS', 96)]
```

Note

To get a Python `dict`, simply wrap the Counter object with `dict()`.

path2insight.explore.metrics.**distance_on_depth**(*x*, *y=None*, *metric='l2'*, *n_jobs=1*)

Compute the distance between filenames based on the depth.

The distance between filenames is computed based on the difference in depth between the filenames.

Parameters

- **x** (*list*) – A list of filepath objects
- **y** (*list*) – A list of filepath pbjects to compare x with. If y is None, the internal similarity of the filepaths in x are computed.
- **metric** (*string, or callable*) – The distance metric like 'cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html for all possible metrics. Default 'l2'.
- **n_jobs** (*int*) – The number of cores to use during the computation of the metric. Default 1.

Example

```
>>> from path2insight.explore import distance_on_depth
>>> import seaborn as sns
```

```
>>> d = distance_on_depth(DATASET)
>>> sns.heatmap(d)
```

Note

For visual inspection, the *heatmap* function in seaborn can be useful.

path2insight.explore.metrics.**distance_on_extension**(*x*, *y=None*, *tokenizer=None*, *metric='l2'*, *n_jobs=1*)

Compute the distance between filenames based on the extension.

The distance between filenames is computed based on the number of extensions that both filenames have in common.

Parameters

- **x** (*list*) – A list of filepath objects
- **y** (*list*) – A list of filepath pbjects to compare x with. If y is None, the internal similarity of the filepaths in x are computed.
- **metric** (*string, or callable*) – The distance metric like 'cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html for all possible metrics. Default 'l2'.
- **n_jobs** (*int*) – The number of cores to use during the computation of the metric. Default 1.

Example

```
>>> from path2insight.explore import distance_on_extension
>>> import seaborn as sns
```

```
>>> d = distance_on_extension(DATASET)
>>> sns.heatmap(d)
```

Note

For visual inspection, the *heatmap* function in seaborn can be useful.

path2insight.explore.metrics.**distance_on_token**(*x*, *y=None*, *tokenizer=None*, *metric='l2'*, *n_jobs=1*)

Compute the distance between filenames based on tokens.

The distance between filenames is computed based on the number of tokens that both filenames have in common.

Parameters

- **x** (*list*) – A list of filepath objects
- **y** (*list*) – A list of filepath pbjects to compare x with. If y is None, the internal similarity of the filepaths in x are computed.
- **tokenizer** (*callable*) – Not implemented yet.
- **metric** (*string, or callable*) – The distance metric like 'cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'. See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_distances.html for all possible metrics. Default 'l2'.
- **n_jobs** (*int*) – The number of cores to use during the computation of the metric. Default 1.

Example

```
>>> from path2insight.explore import distance_on_token
>>> import seaborn as sns
```

```
>>> d = distance_on_token(DATASET)
>>> sns.heatmap(d)
```

Note

For visual inspection, the *heatmap* function in seaborn can be useful.

7.5 Tokenizing

path2insight.tokenizers.tokenizers.**camel_splitter**(*x*)

Make tokens from camelCase strings

Parameters **x** (*WindowsFilePath, PosixFilePath, str*) – The filepath of string.

Returns list of tokens (strings)

Return_type list

path2insight.tokenizers.tokenizers.**default_tokenizer**(*x*)

Make tokens of a file path or string.

Parameters **x** (*WindowsFilePath, PosixFilePath, str*) – The filepath of string.

Returns list of tokens (strings)

Return_type list

path2insight.tokenizers.tokenizers.**path_tokenizer**(*x*)

Make parts of a file path.

Parameters **x** (*WindowsFilePath, PosixFilePath, str*) – The filepath .

Returns list of path parts (strings)

Return_type list

path2insight.tokenizers.tokenizers.**title_splitter**(*x*)

Make tokens from title formatted strings

Parameters **x** (*WindowsFilePath, PosixFilePath, str*) – The filepath of string.

Returns list of tokens (strings)

Return_type list

7.6 Tagging

This module implements a filepath tagger. The object structure of this filepath tagger is based on the tagger objects in the Natural Language Toolkit (NLTK).

class path2insight.explore.tagger.**BaseTypeTagger**(*tokenizer=None*, *tag_names=None*)

The base class for the type taggers.

Parameters

- **tokenizer** (*list*) – A tokenizer function to split the filepath parts.
- **tag_names** (*list*) – The names of the four tags that this tagger uses. The tags default tags are drive="DRV", folder="FLD", stem="STM" and extension="EXT".

tag(*x*)

Return a list with the parts/tokens and their tags.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list of lists for which each nested list is a 2-tuple of name and tag.

Return_type list

class path2insight.explore.tagger.**CompressionTagger**(*tags=...*, *na_tag=''*, *ignore_case=True*, *use_wildcards=True*)

Extension tagger for compression and archiving.

This tagger tags compressed file paths based on their extension. There are three different types of tags in this tagger. The tags are:

- ARCHIVE
- COMPRESSION
- ARCHIVE_AND_COMPRESSION

Parameters

- **tags** (*dict*) – A dict with the extensions to tag. The keys of the dict are the tags and the values of the dict are lists with extensions.
- **na_tag** (*str, None, object*) – The tag for an extension that is not in the tags dictonairy. Default ''.

Ignore_case bool Case-insensitive extension tagging. Default False.

Use_wildcards bool Use Unix shell-style wildcards like * and ?. Default True.

tag(*x*)

Return a list with the extension tag for each file path.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list with the tag(s) for each filepath.

Return_type list

class path2insight.explore.tagger.**DocumentTagger**(*tags=...*, *na_tag=''*, *ignore_case=True*, *use_wildcards=True*)

Extension tagger for documents.

This tagger tags compressed file paths based on their extension. There are three different types of tags in this tagger. The tags are:

- DOCUMENT
- PRESENTATION

Parameters

- **tags** (*dict*) – A dict with the extensions to tag. The keys of the dict are the tags and the values of the dict are lists with extensions.
- **na_tag** (*str, None, object*) – The tag for an extension that is not in the tags dictonairy. Default ''.

Ignore_case bool Case-insensitive extension tagging. Default False.

Use_wildcards bool Use Unix shell-style wildcards like * and ?. Default True.

tag(*x*)

Return a list with the extension tag for each file path.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list with the tag(s) for each filepath.

Return_type list

class path2insight.explore.tagger.**ExtensionTagger**(*tags={}*, *na_tag=''*, *ignore_case=False*, *use_wildcards=True*)

Extension tagger based on dict of tags.

Unix shell-style wildcards like * and ? are supported.

Parameters

- **tags** (*dict*) – A dict with the extensions to tag. The keys of the dict are the tags and the values of the dict are lists with extensions.
- **na_tag** (*str, None, object*) – The tag for an extension that is not in the tags dictonairy. Default ''.

Ignore_case bool Case-insensitive extension tagging. Default False.

Use_wildcards bool Use Unix shell-style wildcards like * and ?. Default True.

Note:

Use an OrderedDict in case of order prevelence.

tag(*x*)

Return a list with the extension tag for each file path.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list with the tag(s) for each filepath.

Return_type list

class path2insight.explore.tagger.**FolderTagger**
[EXPERIMENTAL] A tagger that assigns a FOLDER or FILE tag to each path.

```
>>> from path2insight.explore import FolderTagger
>>> folder_tagger = FolderTagger()
>>> list(folder_tagger.tag([WindowsFilePath('D:/armel/file.xyz')])
[(WindowsFilePath('D:/armel/file.xyz'), 'FILE')]
```

class path2insight.explore.tagger.**Tagger**
Base class for the taggers.

class path2insight.explore.tagger.**TokenTypeTagger**(*tokenizer=<function default_tokenizer>*, *tag_names=None*)
A tagger that tags each filepath part (and extension) with the following labels: drive (DRV), folder (FLD), stem (STEM) and extension (EXT).

Parameters

- **tokenizer** (*callable*) – A function that converts a filepath or string into tokens.
- **tag_names** (*list*) – The names of the four tags that this tagger uses. The tags default tags are drive="DRV", folder="FLD", stem="STM" and extension="EXT".

tag(*x*)
Return a list with the parts/tokens and their tags.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list of lists for which each nested list is a 2-tuple of name and tag.

Return_type list

class path2insight.explore.tagger.**TypeTagger**(*tag_names=None*)
A tagger that tags each filepath part (and extension) with the following labels: drive (DRV), folder (FLD), stem (STEM) and extension (EXT).

Parameters **tag_names** (*list*) – The names of the four tags that this tagger uses. The tags default tags are drive="DRV", folder="FLD", stem="STM" and extension="EXT".

tag(*x*)
Return a list with the parts/tokens and their tags.

Parameters **x** (*list*) – A list with WindowsFilePath and PosixFilePath objects.

Returns A list of lists for which each nested list is a 2-tuple of name and tag.

Return_type list

7.7 Datasets

7.7.1 Create

path2insight.collect.**walk**(*d*, *delay=None*, ***kwargs*)
Walk the file system like os.walk.

Function to collect file paths from the file system. This function is useful for collecting and sharing the file paths. The function is similar to os.walk.

Parameters

- **d** (*str*) – The path to the directory.
- **delay** (*(list of) str*) – Time delay between requesting paths.
- **kwargs** – Additional kwargs for os.walk.

Returns Return the file paths and folder paths. The function returns a tuples with (files, folders)

Return_type (list, list)

Example

Collect and share filepaths with pandas.

```
>>> import pandas as pd
>>> import path2insight
>>> files, folders = path2insight.walk('.')
>>> pd.DataFrame(files).to_csv("export_filepaths.csv", index=False)
```

7.7.2 Examples

Path2Insight comes with several datasets. These datasets are public and real datasets. The datasets are available through the submodule 'datasets'. See the example below:

```
from path2insight.datasets import load_pride
```

path2insight.datasets.external.**load_ensembl**(*nrows=None*, *skiprows=None*)

Load the filepaths of the Ensembl dataset (release 90).

"Ensembl is a genome browser for vertebrate genomes that supports research in comparative genomics, evolution, sequence variation and transcriptional regulation. Ensembl annotate genes, computes multiple alignments, predicts regulatory function and collects disease data. Ensembl tools include BLAST, BLAT, BioMart and the Variant Effect Predictor (VEP) for all supported species. (ensembl.org)"

The filepaths from release-90 of this dataset are loaded with this function. The data can be found at ftp://ftp.ensembl.org/pub/release-90/. The snapshot was taken on 16 November 2017 with a Linux device with the ftp://ftp.ensembl.org/pub/release-90/ as a mounted drive.

Parameters

- **nrows** – Number of rows of file to read. Useful for reading pieces of large files
- **skiprows** (*list-like or integer or callable, default None*) – Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file. See pandas.read_csv() for more information about this parameter.

Returns A list of PosixFilePaths of the PRIDE dataset.

Return_type list

path2insight.datasets.external.**load_pride**(*nrows=None*, *skiprows=None*)

Load the filepaths of the PRIDE proteomics archive.

"The PRIDE PRoteomics IDEntifications (PRIDE) database is a centralized, standards compliant, public data repository for proteomics data, including protein and peptide identifications, post-translational modifications and supporting spectral evidence. PRIDE is a core member in the ProteomeXchange (PX) consortium, which provides a single point for submitting mass spectrometry based proteomics data to public-domain repositories.

Datasets are submitted to PRIDE via ProteomeXchange and are handled by expert biocurators. (https://www.ebi.ac.uk/pride/archive/)"

The filepaths from of this dataset are loaded with this function. The data can be found at ftp://ftp.pride.ebi.ac.uk/pride/data/archive/. The snapshot was taken on 06 february 2018 with a Linux device with the ftp://ftp.pride.ebi.ac.uk/pride/data/archive/ as a mounted drive.

Parameters

- **nrows** – Number of rows of file to read. Useful for reading pieces of large files
- **skiprows** (*list-like or integer or callable, default None*) – Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file. See pandas.read_csv() for more information about this parameter.

Returns A list of PosixFilePaths of the PRIDE dataset.

Return_type list

7.8 Misc

path2insight.external.nltk.**bigrams**(*sequence*, ***kwargs*)

Return the bigrams generated from a sequence of items, as an iterator. For example:

```
>>> from path2insight.external.nltk import bigrams
>>> list(bigrams([1,2,3,4,5]))
[(1, 2), (2, 3), (3, 4), (4, 5)]
```

Use bigrams for a list version of this function.

Parameters **sequence** (*sequence or iter*) – the source data to be converted into bigrams

Return type iter(tuple)

path2insight.external.nltk.**everygrams**(*sequence*, *min_len=1*, *max_len=-1*, ***kwargs*)

Returns all possible ngrams generated from a sequence of items, as an iterator.

```
>>> sent = 'a b c'.split()
>>> list(everygrams(sent))
[('a',), ('b',), ('c',), ('a', 'b'), ('b', 'c'), ('a', 'b', 'c')]
>>> list(everygrams(sent, max_len=2))
[('a',), ('b',), ('c',), ('a', 'b'), ('b', 'c')]
```

Parameters

- **sequence** (*sequence or iter*) – the source data to be converted into trigrams
- **min_len** (*int*) – minimum length of the ngrams, aka. n-gram order/degree of ngram
- **max_len** (*int*) – maximum length of the ngrams (set to length of sequence by default)

Return type iter(tuple)

path2insight.external.nltk.**ngrams**(*sequence*, *n*, *pad_left=False*, *pad_right=False*, *left_pad_symbol=None*, *right_pad_symbol=None*)

Return the ngrams generated from a sequence of items, as an iterator. For example:

```
>>> from path2insight.external.nltk import ngrams
>>> list(ngrams([1,2,3,4,5], 3))
[(1, 2, 3), (2, 3, 4), (3, 4, 5)]
```

Wrap with list for a list version of this function. Set pad_left or pad_right to true in order to get additional ngrams:

```
>>> list(ngrams([1,2,3,4,5], 2, pad_right=True))
[(1, 2), (2, 3), (3, 4), (4, 5), (5, None)]
>>> list(ngrams([1,2,3,4,5], 2, pad_right=True, right_pad_symbol='</s>'))
[(1, 2), (2, 3), (3, 4), (4, 5), (5, '</s>')]
>>> list(ngrams([1,2,3,4,5], 2, pad_left=True, left_pad_symbol='<s>'))
[('<s>', 1), (1, 2), (2, 3), (3, 4), (4, 5)]
>>> list(ngrams([1,2,3,4,5], 2, pad_left=True, pad_right=True, left_pad_symbol='
↪<s>', right_pad_symbol='</s>'))
[('<s>', 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, '</s>')]
```

Parameters

- **sequence** (*sequence or iter*) – the source data to be converted into ngrams
- **n** (*int*) – the degree of the ngrams
- **pad_left** (*bool*) – whether the ngrams should be left-padded
- **pad_right** (*bool*) – whether the ngrams should be right-padded
- **left_pad_symbol** (*any*) – the symbol to use for left padding (default is None)
- **right_pad_symbol** (*any*) – the symbol to use for right padding (default is None)

Return type sequence or iter

path2insight.external.nltk.**pad_sequence**(*sequence*, *n*, *pad_left=False*, *pad_right=False*, *left_pad_symbol=None*, *right_pad_symbol=None*)

Returns a padded sequence of items before ngram extraction.

```
>>> list(pad_sequence([1,2,3,4,5], 2, pad_left=True, pad_right=True, left_pad_
↪symbol='<s>', right_pad_symbol='</s>'))
['<s>', 1, 2, 3, 4, 5, '</s>']
>>> list(pad_sequence([1,2,3,4,5], 2, pad_left=True, left_pad_symbol='<s>'))
['<s>', 1, 2, 3, 4, 5]
>>> list(pad_sequence([1,2,3,4,5], 2, pad_right=True, right_pad_symbol='</s>'))
[1, 2, 3, 4, 5, '</s>']
```

Parameters

- **sequence** (*sequence or iter*) – the source data to be padded
- **n** (*int*) – the degree of the ngrams
- **pad_left** (*bool*) – whether the ngrams should be left-padded
- **pad_right** (*bool*) – whether the ngrams should be right-padded
- **left_pad_symbol** (*any*) – the symbol to use for left padding (default is None)
- **right_pad_symbol** (*any*) – the symbol to use for right padding (default is None)

Return type sequence or iter

path2insight.external.nltk.**skipgrams**(*sequence*, *n*, *k*, ***kwargs*)

Returns all possible skipgrams generated from a sequence of items, as an iterator. Skipgrams are ngrams that allows tokens to be skipped. Refer to http://homepages.inf.ed.ac.uk/ballison/pdf/lrec_skipgrams.pdf

```
>>> sent = "Insurgents killed in ongoing fighting".split()
>>> list(skipgrams(sent, 2, 2))
[('Insurgents', 'killed'), ('Insurgents', 'in'), ('Insurgents', 'ongoing'), (
↪'killed', 'in'), ('killed', 'ongoing'), ('killed', 'fighting'), ('in', 'ongoing
↪'), ('in', 'fighting'), ('ongoing', 'fighting')]
>>> list(skipgrams(sent, 3, 2))
[('Insurgents', 'killed', 'in'), ('Insurgents', 'killed', 'ongoing'), ('Insurgents
↪', 'killed', 'fighting'), ('Insurgents', 'in', 'ongoing'), ('Insurgents', 'in',
↪'fighting'), ('Insurgents', 'ongoing', 'fighting'), ('killed', 'in', 'ongoing'),
↪ ('killed', 'in', 'fighting'), ('killed', 'ongoing', 'fighting'), ('in',
↪'ongoing', 'fighting')]
```

Parameters

- **sequence** (*sequence or iter*) – the source data to be converted into trigrams
- **n** (*int*) – the degree of the ngrams
- **k** (*int*) – the skip distance

Return type iter(tuple)

path2insight.external.nltk.**trigrams**(*sequence*, ***kwargs*)

Return the trigrams generated from a sequence of items, as an iterator. For example:

```
>>> from path2insight.external.nltk import trigrams
>>> list(trigrams([1,2,3,4,5]))
[(1, 2, 3), (2, 3, 4), (3, 4, 5)]
```

Use trigrams for a list version of this function.

Parameters **sequence** (*sequence or iter*) – the source data to be converted into trigrams

Return type iter(tuple)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

E

F

I

J

L

M

N

P

R

S

T

U

W

Z