PyAaaS Documentation

Release 0.5.7

OsloMET-Group-8

Oct 08, 2020

Contents:

PyARXaaS is a Python wrapper package for ARXaaS. It provides user-friendly abstractions for the APIs exposed ARXaaS. Github link

- For quick-start see *Quick Start*
- For more in-depth information about the API see *API Documentation* .

CHAPTER 1

How it works

PyARXaaS is a simple pure Python client package that provides abstractions for interacting with a ARXaaS instance. The package supports analyzing re-identification risks and anonymizing tabular datasets containing sensitive personal data.

CHAPTER 2

Features

- *ARXaaS* class for configuration and calling actions.
- *Dataset* class for encapsulating and configuring a dataset
- *Privacy Model* classes for configuring the Privacy Models to use in anonymization.
- Easy integration with pandas DataFrames

CHAPTER 3

Simple Use Case

Quick overview of how to get started using the package:

```
# import dependencies
from pyarxaas import ARXaaS
from pyarxaas.privacy_models import KAnonymity
from pyarxaas import AttributeType
from pyarxaas import Dataset
import pandas as pd

arxaas = ARXaaS(url) # url contains url to AaaS web service

df = pd.read_csv("data.csv")

# create Dataset
dataset = Dataset.from_pandas(df)


# set attribute type
dataset.set_attributes(AttributeType.QUASIIDENTIFYING, 'name', 'gender')
dataset.set_attribute(AttributeType.IDENTIFYING, 'id')

# get the risk profle of the dataset
risk_profile = arxaas.risk_profile(dataset)

# get risk metrics
re_indentifiation_risk = risk_profile.re_identification_risk
distribution_of_risk = risk_profile.distribution_of_risk
```

CHAPTER 4

Licensing

PyARXaaS is distributed under the MIT license. See LICENCE

4.1 User Guide

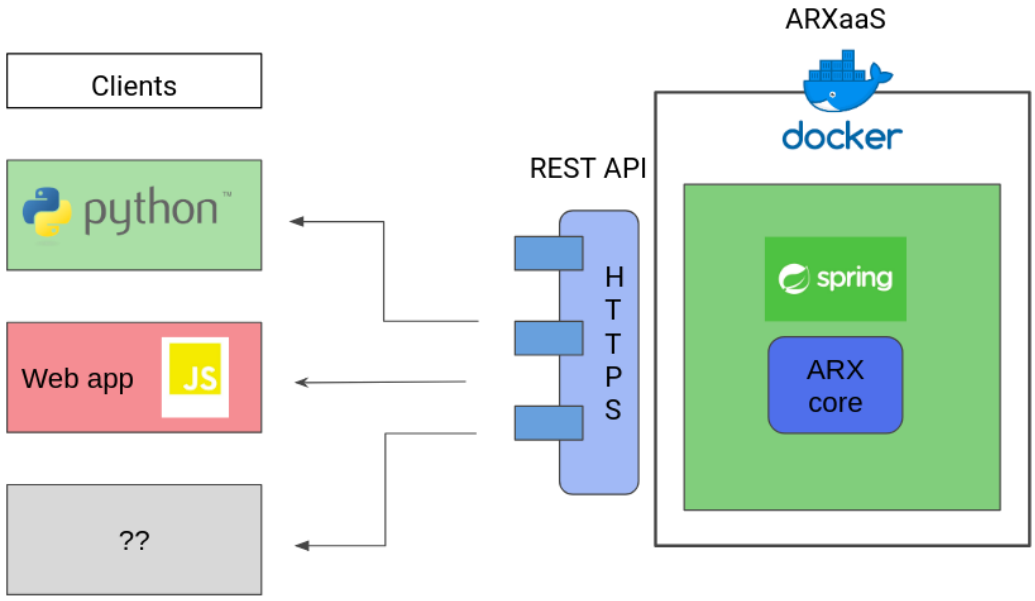
4.1.1 Introduction

PyARXaaS is the Python client for the ARXaaS service. You can read more about ARXaaS here . PyARXaaS is similar to projects like PyGithub. It tries to make make the integration of the risk analysis and de-identification functionality of ARXaaS as easy and intuitive as possible. The main user group of the package are data scientist that would be familiar and accustomed to work with data in Python. This user group would prefer not to have to work in a GUI tool such as ARX. The philosophy of the ARXaaS project has been to make a microservice component with the de-identification functionality, and building clients that make integration easy and seamless, instead of trying to integrate the ARX GUI into the work flow.

The ARX project makes their core library available by making it open source, and this library is what the team have used as the building foundation for the service. ARX is the industry leader in the de-identification space, and the team is grateful that the project remains open source, making projects such as this possible.

Bellow is a coarse overview of the ARXaaS ecosystem. PyARXaaS is the first and main client priority. But other clients can be developed as the need arises.

Architecture overview



4.1.2 Installing PyARXaaS

Note: PyARXaas requires python 3.6 and up. Python download

PyARXaaS is available on PyPI PyPI link

Pip install

Open the command-line interface and write:

```
pip install pyarxaas
```

(Optional) Setup virtual environment

Mac/Linux

1. Create a new directory

```
mkdir pyarxaas-project
```

2. Change currenct directory to 'pyarxaas-project'

```
cd pyarxaas-project
```

3. Create a virtual environment

```
python3 -m venv c:\path\to\myenv
```

4. activate the virtual environment

```
source venv/bin/activate
```

Windows (Windows Powershell)

1. Create a new directory

```
mkdir pyarxaas-project
```

2. Change currenct directory to 'pyarxaas-project'

```
cd .\pyarxaas-project
```

3. Create a virtual environment

```
python -m venv c:\path\to\myenv
```

4. activate the virtual environment

```
.\Scripts\activate
```

Get the Source Code

```
git clone https://github.com/oslomet-arx-as-a-service/PyARXaaS.git
```

4.1.3 Quick Start

This page gives a good introduction in how to get started with PyARXaaS

First, make sure that:

- PyARXaaS is installed
- PyARXaaS is up-to-date
- You have a tabular dataset to use
- **You hava a running ARXaaS instance to connect to.**
  - Instructions on how to run ARXaaS can be found here: https://github.com/oslomet-arx-as-a-service/ARXaaS/blob/master/README.md
- If you are going to anonymize a dataset, you need to have the required hierarchies. See anonymize section for more information

Let's get started with some simple examples.

Analyze the risk of a dataset

Analyze the risk of a dataset using PyARXaaS is very simple.

1. **Begin by importing the Dataset class and pandas which we are going to use to create a** *Dataset*

```
from pyarxaas import Dataset
import pandas as pd
```

Then we create a Dataset from a local csv file

Note: The dataset in this example contains the columns/fields **id, name, gender**

```
dataframe = pd.read_csv("data.csv", sep=";")
# create Dataset
dataset = Dataset.from_pandas(dataframe)
```

The Dataset class encapsulates the raw data, attribute types of the dataset fields and hierarchies

2. **Then we set the *Attribute Type* for the Dataset fields.**

```
# import the attribute_type module
 from pyarxaas import AttributeType

# set attribute type
dataset.set_attribute_type(AttributeType.QUASIIDENTIFYING, 'name', 'gender')
dataset.set_attribute_type(AttributeType.IDENTIFYING, 'id')
```

3. **To make a call to the ARXaaS instance we need to make a instance of the *ARXaaS* class.**

   The AaaS connector class needs a url to the ARXaaS instance. In this example we have ARXaaS running locally.

```
# import the ARXaaS class
from pyarxaas import ARXaaS

# establishing a connection to the ARXaaS service using a URL
arxaas = ARXaaS("http://localhost:8080")
```

4. **After the *ARXaaS* object is created we can use it to call the ARXaaS instance to make a *RiskProfile* for our Dataset.**

```
# get the risk profle of the dataset
risk_profile = arxaas.risk_profile(dataset)
```

   The *RiskProfile* contains two properties; re-indentification risks and distributed risks. The two properties contains the different risks and the distribution of risks for the *Dataset*.

```
# get risk metrics as a dictionary
re_indentifiation_risk = risk_profile.re_identification_risk
distribution_of_risk = risk_profile.distribution_of_risk

# get risk metrivs as pandas.DataFrame
re_i_risk_df = risk_profile.distribution_of_risk_dataframe()
dist_risk_df = risk_profile.distribution_of_risk_dataframe()
```

Anonymize a dataset

Anonymizing a dataset using PyARXaaS.

1. **Begin by importing the Dataset class and pandas which we are going to use to create a Dataset**

```
from pyarxaas import Dataset
import pandas as pd
```

2. **Same as when in analyze we set the attribute type for the dataset fields**:

```
# import the attribute_type module
 from pyarxaas import AttributeType

# set attribute type
dataset.set_attributes(AttributeType.QUASIIDENTIFYING, 'name', 'gender')
dataset.set_attributes(AttributeType.IDENTIFYING, 'id')
```

3. **In addition to setting attribute types we need to provide Transformation Models known as hierarchies for the dataset fields/columns with type AttributeType.QUASIIDENTIFYING** Hierarchies can be added as pandas.DataFrame objects:

```
# importing the hierarchies from a local csv file. Specify the file path as the first␣
↪parameter
id_hierarchy = pd.read_csv("id_hierarchy.csv", header=None)
name_hierarchy = pd.read_csv("name_hierarchy.csv", header=None)

# setting the imported csv file. Specify the column name as the fist parameter, and␣
↪the hierarchy as the second parameter
dataset.set_hierarchy('id', id_hierarchy)
dataset.set_hierarchy('name', name_hierarchy)
```

4. **When anonymizing we need to supply a *Privacy Model* for ARXaaS to run on the dataset. You can read more about the models here** ARX Privacy Models

```
# importing the privacy_models module
from pyarxaas.privacy_models import KAnonymity

# creating a privacy_models object
kanon = KAnonymity(4)
```

5. **To make a call to the ARXaaS instance we need to make a instance of the AaaS class. The AaaS connector class needs a url to the ARXaaS instance. In this example we have ARXaaS running locally.**

```
# import the aaas module
from pyarxaas import ARXaaS

# establishing a connection to the ARXaaS service using the URL
arxaas = ARXaaS("http://localhost:8080")
```

6. **After the *ARXaaS* object is created we can use it to call the ARXaaS instance. Back if the anonymization is successful we receive an *AnonymizeResult***

```
# specify the dataset as the first parameter, and privacy model list as the␣
↪second paramter
anonymize_result = arxxaas.anonymize(dataset, [kanon])
```

AnonymizeResult contains the new *Dataset*, the *RiskProfile* for the new , the *Dataset*, the anonymization status for the *Dataset* and *AnonymizeMetrics* which contains metrics regarding the anonymzation performed on the dataset.

```
# get the new dataset
anonymized_dataset = anonymize_result.dataset
anon_dataframe = anonymized_dataset.to_dataframe()

# get the risk profile for the new dataset
anon_risk_profile = anonymize_result.risk_profile
```

(continues on next page)

(continued from previous page)

```
# get risk metrics as a dictionary
re_indentifiation_risk = anon_risk_profile.re_identification_risk
distribution_of_risk = anon_risk_profile.distribution_of_risk

# get risk metrivs as pandas.DataFrame
re_i_risk_df = anon_risk_profile.distribution_of_risk_dataframe()
dist_risk_df = anon_risk_profile.distribution_of_risk_dataframe()

# get the anonymiztion metrics
anon_metrics = anonymize_result.anonymization_metrics
```

4.1.4 Using the Dataset class

The *Dataset* class is the represents a tabular dataset containing continuous or categorical attributes. Additionally each attribute has a *Attribute Type* describing the re-identification risk and sensitivity associated with the attribute.

In the case where a attribute is Quasi-identifying a hierarchy object can be added (Read more about hierarchies here).

Dataset contains

- Tabular dataset
- *Attribute Type* for the dataset fields/attributes
- (optional) hierarchies for the quasi-identifying attributes

Construction

A *Dataset* object can be made from a pandas.DataFrame or a python dict using the constructor class methods.

From Python dictionary

```
data_dict = {"id": [1,2,3], "name": ["Mike", "Max", "Larry"]}
new_dataset = Dataset.from_dict(data_dict)
```

From pandas.DataFrame

```
dataframe = pd.read_csv("data.csv", sep=";")
new_dataset = Dataset.from_pandas(dataframe)
```

Covert Dataset to other types

The Dataset class has convenient methods for converting the tabular dataset back to usefull datastructures

To pandas.DataFrame Note: When you create a pandas.DataFrame from a Dataset only the tabular data is included. The *Attribute Type* information and hierarchies are lost.

```
data_dict = {"id": [1,2,3], "name": ["Mike", "Max", "Larry"]}
new_dataset = Dataset.from_dict(data_dict)
dataframe = new_dataset.to_dataframe()
#    id   name
#0   1   Mike
#1   2    Max
#2   3  Larry
```

Mutation

Attribute type

The default *Attribute Type* for attributes in a Dataset is *Attribute Type*.QUASIIDENTIFYING. The default is set to quasi-identifying so that new users will error on the safe side. You can change the type of a attribute with the set_attribute_type() method.:

```
from pyarxaas import AttributeType
new_dataset.set_attribute_type(AttributeType.IDENTIFYING, "id")
```

Above we have changed the *Attribute Type* of the *Dataset* to *Attribute Type*.IDENTIFYING. This signals that the *id* attribute is a directly identifying attribute in this *Dataset*. *id* will be treated as such if anonymization is applied to the *Dataset*.

Read more about the different Attribute types here: *Attribute Type*

It is possible to pass *n* attributes following the *Attribute Type* parameter to set the attribute type to all the attribute.

```
# Here id and name are marked as insensitive attributes
new_dataset.set_attribute_type(AttributeType.INSENSITIVE, "id", "name")
```

Hierarchies

Hierarchy also referred to as *generalization hierarchies* represented either as pandas.DataFrames or a regular Python list, are the strategies ARXaaS will use when attempting to anonymize the dataset. Read more about them *Creating Hierarchies*.

Setting a hierarchy on a Dataset attribute

```
id_hierarchy = [["1", "*"], ["2", "*"], ["3", "*"]]
dataset.set_hierarchy("id", id_hierarchy)
```

You can also set several hierarchies in one call with the .set_hierarchies(hierarchies) method.

```
id_hierarchy = [["1", "*"], ["2", "*"], ["3", "*"]]
job_hierarchy = [["plumber", "manual-labour", "*"],
                 ["hairdresser", "service-industry", "*"]]
hierarchies = {"id": id_hierarchy, "job": job_hierarchy}
dataset.set_hierarchies(hierarchies)
```

4.1.5 Connecting to and using ARXaaS

Calls to ARXaaS is made through the *ARXaaS* class. ARXaaS implements methods for the following functionality:

- Anonymize a *Dataset* object
- Analyze re-identification risk for a *Dataset* object
- Create generalization hierarchies (See: *Creating Hierarchies*)

Creating

When creating a instance of the ARXaaS class you need to pass a full url to the service running.

Example

```
from pyarxaas import ARXaaS
arxaas = ARXaaS(https://localhost:8080)
```

Risk Profile

Re-identfification risk for prosecutor, journalist and markteter attack models can be obtained using the ARXaaS risk_profile method. The method takes a *Dataset* object and returns a *RiskProfile*. See *Using the Dataset class* for more on the Dataset class. More in depth information on re-identificaiton risk ARX | risk analysis

Example

```
risk_profile = arxaas.risk_profile(dataset)
```

Risk profile contains different properties containg analytics on the dataset re-identification risk. Most important is the re-identification risk property.

```
# create risk profile ...
risks = risk_profile.re_identification_risk
```

The property contains a mapping of risk => value. What is a acceptable risk depends entirely on the context of the dataset.

Anonymization

Anonymizing a dataset is as simple as passing a *Dataset* containing the neccessary hierarchies, a sequence of *Privacy Model* to use and optionally a suppersion limit to the anonymize() method. The method, if succesfull returns a *AnonymizeResult* object containing the new dataset.

Example

```
kanon = KAnonymity(2)
ldiv = LDiversityDistinct(2, "disease") # in this example the dataset has a disease␣
↪field
anonymize_result = arxaas.anonymize(dataset, [kanon, ldiv], 0.2)
anonymized_dataset = anonymize_result.dataset
```

Hierarchy Generation

Generalizaiton hierarchies are a important part of anonymization. ARXaaS contains a hierarchy() method. It takes a configured *Hierarchy Builders* object and a dataset column represented as a common Python list. It returns a 2D list structure containing a new hierarchy.

Example making a redaction hierarchy

```
redaction_builder = RedactionHierarchyBuilder()
zipcodes = [47677, 47602, 47678, 47905, 47909, 47906, 47605, 47673, 47607]
zipcode_hierarchy = arxaas.hiearchy(redaction_builder, zipcodes)
```

4.1.6 Creating Hierarchies

After creating a *Dataset* from some data source, you can set the hierarchies ARXaaS will use when attempting to anonymize the Dataset. ARXaaS currently only support value generalization hierarchies. Read more about different transformation models in ARX documentation.

Hierarchy Building

ARXaaS offers an endpoint to use the ARX library hierarchy generation functionality. PyARXaaS implements abstractions to make this process as easy and intuitive as possible.

Hierarchy generation that ARX offers falls into four different categories:

- Redaction based hierarchies
- Interval based hierarchies
- Order based hierarchies
- Date based hierarchies

ARXaaS and PyARXaaS currently only support Redaction, Interval and Order based hierarchy generation. In PyARXaaS all the hierarchy builders are importable from the *pyaaas.hierarchy* package

Redaction based hierarchies

Redaction based hierarchies are hierarchies suited best for categorical but numeric values. Attributes such as zipcodes are a prime candidate. The hierarchy strategy is to delete one number at a time from the attribute column until the privacy model criteria is meet. The hierarchy builder can be configured to start deleting from either direction, but will default to RIGHT_TO_LEFT. Redaction based hierarchies are the hierarchies with the least effort to create.

Example In this example we will use a list of zipcodes representing a column from a hypothetical dataset. The list could be generated from any source. Hierarchy building works on list of strings or numbers.

```
zipcodes = [47677, 47602, 47678, 47905, 47909, 47906, 47605, 47673, 47607]
```

We will then import the redaction hierarchy builder class

```
from pyarxaas.hierarchy import RedactionHierarchyBuilder
```

The *RedactionHierarchyBuilder* class is a simple class and all configuration is optional. When instantiating the class the user can pass in parameters to configure how the resulting hierarchy should be built. See *RedactionHierarchyBuilder* for more on the parameters.

Creating a simple redaction hierarchy

```
redaction_builder = RedactionHierarchyBuilder() # Create builder
```

The builder defines a template to build the resulting hierarchy from. Now that we have the list to create a hierarchy for and a builder to build it with can call ARXaaS to make the hierarchy.

```
from pyarxaas import ARXaaS
# establishing a connection to the ARXaaS service using a url, in this case ARXaaS is␣
↪running locally on port 8080
arxaas = ARXaaS("http://localhost:8080")
```

With the connection to ARXaaS established we can create the hierarchy.

```
redaction_hierarchy = arxaas.hierarchy(redaction_based, zipcodes) # pass builder and␣
↪column to arxaas
```

The resulting hierarchy looks like this:

```
[['47677', '4767*', '476**', '47***', '4****', '*****'],
['47602', '4760*', '476**', '47***', '4****', '*****'],
['47678', '4767*', '476**', '47***', '4****', '*****'],
['47905', '4790*', '479**', '47***', '4****', '*****'],
['47909', '4790*', '479**', '47***', '4****', '*****'],
['47906', '4790*', '479**', '47***', '4****', '*****'],
['47605', '4760*', '476**', '47***', '4****', '*****'],
['47673', '4767*', '476**', '47***', '4****', '*****'],
['47607', '4760*', '476**', '47***', '4****', '*****']]
```

Interval based hierarchies

Interval based hierarchies are well suited for continuous numeric values. Attributes such as age, income or credit score are typically generalized with a interval based hierarchy. The Interval based hierarchy builder requires the user to specify intervals in which to generalize values in the attribute. Optionally these intervals can be labeled. In addition intervals can be grouped upwards using levels and groups to create a deeper hierarchy.

Example In this example we will use a list of ages representing a column from a hypothetical dataset.

```
ages = [29, 22, 27, 43, 52, 47, 30, 36, 32]
```

We import the *IntervalHierarchyBuilder* class from the hierarchy package.

```
from pyarxaas.hierarchy import IntervalHierarchyBuilder
```

Then we instantiate the builder. *IntervalHierarchyBuilder* takes no constructor arguments.

```
interval_based = IntervalHierarchyBuilder()
```

Add intervals to the builder. The intervals must be continuous(without gaps)

```
interval_based.add_interval(0,18, "child")
interval_based.add_interval(18,30, "young-adult")
interval_based.add_interval(30,60, "adult")
interval_based.add_interval(60,120, "old")
```

(Optionally) Add groupings. Groupings are added to a specific level and are order based according to the interval order.

```
interval_based.level(0)\
    .add_group(2, "young")\
    .add_group(2, "adult")
```

Call the ARXaaS service to create the hierarchy

```
interval_hierarchy = arxaas.hierarchy(interval_based, ages)
```

The hierarchy looks like this:

```
[['29', 'young-adult', 'young', '*'],
 ['22', 'young-adult', 'young', '*'],
 ['27', 'young-adult', 'young', '*'],
 ['43', 'adult', 'adult', '*'],
 ['52', 'adult', 'adult', '*'],
 ['47', 'adult', 'adult', '*'],
```

(continues on next page)

(continued from previous page)

```
 ['30', 'adult', 'adult', '*'],
 ['36', 'adult', 'adult', '*'],
 ['32', 'adult', 'adult', '*']]
```

Order based hierarchy

OrderHierarchyBuilder are suited for categorical attributes. Attributes such as country, education level and employment status.

Order based hierarchies are built using groupings with optional labeling. This means that grouping is completed on the list of values as it is. This means the list has to be sorted according to some ordering before a hierarchy can be made. On the positive side. Order based hierarchies are usually very reusable depending on the domain.

In this example we will use a column of diseases.

```
diseases = ['bronchitis',
            'flu',
            'gastric ulcer',
            'gastritis',
            'pneumonia',
            'stomach cancer']
```

In this case we will sort the diseases according to the disease location; *lung-disease* or *stomach-disease*. But this sorting can be as sophistical as the user wants.

```
unique_diseases[2], unique_diseases[4] = unique_diseases[4], unique_diseases[2]
unique_diseases

#['bronchitis',
# 'flu',
# 'pneumonia',
# 'gastritis',
# 'gastric ulcer',
# 'stomach cancer']
```

Import *OrderHierarchyBuilder*

```
from pyarxaas.hierarchy import OrderHierarchyBuilder
```

Create instance to use.

```
order_based = OrderHierarchyBuilder()
```

Group the values. Note that the groups are applied to the values as they are ordered in the list. Adding labels are optional, if labels are not set the resulting field will be a concatenation of the values included in the group.

```
order_based.level(0)\
    .add_group(3, "lung-related")\
    .add_group(3, "stomach-related")
```

Call the ARXaaS service to create the hierarchy

```
order_hierarchy = arxaas.hierarchy(order_based, diseases)
```

The resulting hierarchy looks like this:

```
[['bronchitis', 'lung-related', '*'],
 ['flu', 'lung-related', '*'],
 ['pneumonia', 'lung-related', '*'],
 ['gastritis', 'stomach-related', '*'],
 ['gastric ulcer', 'stomach-related', '*'],
 ['stomach cancer', 'stomach-related', '*']]
```

Date based hierarchy

date_hierarchy_builder are suited for date and timestamp attributes. The date values must be formated according to Java SimpleDateFormat Link to SimpleDateFormat documentation.

In this example we will use a column of timestamps.

```
timestamps = ["2020-07-16 15:28:024",
   "2019-07-16 16:38:025",
   "2019-07-16 17:48:025",
   "2019-07-16 18:48:025",
   "2019-06-16 19:48:025",
   "2019-06-16 20:48:025"]
```

Import date_hierarchy_builder.

```
from pyarxaas.hierarchy import DateHierarchyBuilder
```

Create instance to use.

```
date_based = DateHierarchyBuilder("yyyy-MM-dd HH:mm:SSS",
                      DateHierarchyBuilder.Granularity.SECOND_MINUTE_HOUR_DAY_MONTH_
↪YEAR,
                      DateHierarchyBuilder.Granularity.MINUTE_HOUR_DAY_MONTH_YEAR,
                      DateHierarchyBuilder.Granularity.YEAR)
```

Call the ARXaaS service to create the hierarchy

```
timestamp_hierarchy = arxaas.hierarchy(date_based, timestamps)
```

The resulting hierarchy looks like this:

```
[['2020-07-16 15:28:024', '16.07.2020-15:28:00', '16.07.2020-15:28', '2020'],
 ['2019-07-16 16:38:025', '16.07.2019-16:38:00', '16.07.2019-16:38', '2019'],
 ['2019-07-16 17:48:025', '16.07.2019-17:48:00', '16.07.2019-17:48', '2019'],
 ['2019-07-16 18:48:025', '16.07.2019-18:48:00', '16.07.2019-18:48', '2019'],
 ['2019-06-16 19:48:025', '16.06.2019-19:48:00', '16.06.2019-19:48', '2019'],
 ['2019-06-16 20:48:025', '16.06.2019-20:48:00', '16.06.2019-20:48', '2019']]
```

4.2 API Documentation

4.2.1 ARXaaS

AaaS encapsulates the connection the service

class pyarxaas.arxaas.**ARXaaS**(*url: str*, *connector=<class 'pyarxaas.arxaas_connector.ARXaaSConnector'>*, *client=None*)

Represents the connection to ARXaaS. All public methods result in a call to the service.

anonymize(*dataset: pyarxaas.models.dataset.dataset.Dataset*, *privacy_models*, *suppression_limit: float = None*) → pyarxaas.models.anonymize_result.AnonymizeResult

Attempt to anonymize a dataset with provided privacy models

Parameters

- **dataset** – Dataset to be anonymized
- **privacy_models** – privacy models to be used in the anonymization
- **suppression_limit** – suppression limit to be used in the anonymization

Returns Dataset with anonymized data

hierarchy(*redaction_builder*, *column*)

Creates a value generalization hierarchy with the passed in builder for the passed in column.

Parameters

- **redaction_builder** – a Hierarchy builder instance
- **column** – a list of values

Returns list[list] containing the created hierarchy

risk_profile(*dataset: pyarxaas.models.dataset.dataset.Dataset*) → pyarxaas.models.risk_profile.RiskProfile

Creates a risk profile for a provided Dataset

RiskProfile contains:

- re-identifiaction risks
- distributed risk

Parameters **dataset** – Dataset to create a risk profile for

Returns RiskProfile

4.2.2 Dataset

Dataset encapsulates a dataset in to be analyzed or anonymized using ARXaaS

class pyarxaas.models.dataset.**Dataset**(*data: list*, *attribute_types: collections.abc.Mapping = None*)

Understand tabular data containing personal data.

describe()

Prints a description of the Dataset to stdout

Returns None

classmethod from_dict(*dictionary*)

Create Dataset from a python dictionary

Parameters **dictionary** – Mapping object to create Dataset from

Returns Dataset

classmethod from_pandas(*dataframe: pandas.core.frame.DataFrame*)

Create a Dataset from a pandas DataFrame

Parameters **dataframe** – pandas Dataframe

Returns Dataset

set_attribute_type(*attribute_type: pyarxaas.models.attribute_type.AttributeType*, **attributes*)
Set AttributeType for a collection of attributes

Parameters

- **attributes** – collection of attributes in the dataset
- **attribute_type** – AttributeType for the attributes

Returns None

set_hierarchy(*attribute*, *hierarchy*)
Set hierarchy for a attribute in the Dataset

Parameters

- **attribute** – attribute in the Dataset
- **hierarchy** – to be applied to the attribute

Returns None

to_dataframe() → pandas.core.frame.DataFrame
Create pandas DataFrame of the Dataset

Returns pandas.DataFrame

4.2.3 AnonymizeResult

AnonymizeResult encapsulates the response from a ARXaaS anonymize call

class pyarxaas.models.anonymize_result.**AnonymizeResult**(*dataset: pyarxaas.models.dataset.dataset.Dataset*, *risk_profile: pyarxaas.models.risk_profile.RiskProfile*, *anonymization_metrics: pyarxaas.models.anonymization_metrics.AnonymizationM*
anonymization_status)
Understands the result of a anonymization process

anonymization_metrics
AnonymizationMetrics about the anonymization process. Contains data on hierarchy level used and privacy model configuration

Returns AnonymizationMetrics

anonymization_status
Anonymization status for the new Dataset

Returns str

dataset
Dataset created from the anonymization

Returns Dataset

risk_profile
RiskProfile asscocciated with the new Dataset

Returns RiskProfile

4.2.4 RiskProfile

RiskProfile encapsulates the re-identification risks associated with a given Dataset

RiskProfile contains two main properties

Re-Identification Risks

- Lowest prosecutor re-identification risk.
- Individuals affected by lowest risk.
- Highest prosecutor re-identification risk.
- Individuals affected by highest risk.
- Average prosecutor re-identification risk.
- Fraction of unique records.
- Attacker success rate against the re-identification risk.
- Population Model name
- Quasi-identifiers

Distribution of Risks

The distribution of re-identification risks amongst the records of the dataset

class pyarxaas.models.risk_profile.**RiskProfile**(*metrics: collections.abc.Mapping*)
Represents the re-identification risks associated with a Dataset

attacker_success_rate
Attacker success rates against re-identification for a given Dataset

Returns dict containing the attacker success rate.

distribution_of_risk
Distribution of risk for a given Dataset

Returns dict containing the distribution of risks in a given Dataset

distribution_of_risk_dataframe() → pandas.core.frame.DataFrame
Distribution of risk as a pandas.DataFrame

Returns pandas.DataFrame

population_model
Population model used to analyze a given Dataset

Returns The Population model name used to analyze a given Dataset

quasi_identifiers
Quasi-identifiers for a given Dataset

Returns dict containing a list of all the quasi-identifying attribute in a a given Dataset

re_identification_risk
Re-identification risk metrics for a given Dataset

Returns dict containing re-identification metrics

re_identification_risk_dataframe() → pandas.core.frame.DataFrame
Re-identification risk as a pandas.DataFrame

Returns pandas.Dataframe with risk metrics

4.2.5 Attribute Type

1. Identifying attributes are associated with a high risk of re-identification. They will be removed from the dataset. Typical examples are names or Social Security Numbers.
2. Quasi-identifying attributes can in combination be used for re-identification attacks. They will be transformed. Typical examples are gender, date of birth and ZIP codes.
3. Sensitive attributes encode properties with which individuals are not willing to be linked with. As such, they might be of interest to an attacker and, if disclosed, could cause harm to data subjects. They will be kept unmodified but may be subject to further constraints, such as t-closeness or l-diversity. Typical examples are diagnoses.
4. Insensitive attributes are not associated with privacy risks. They will be kept unmodified.

AttributeType respresents data regarding the identification implication for a filed in a dataset

class pyarxaas.models.attribute_type.**AttributeType**
An enumeration.

4.2.6 AnonymizeMetrics

AnonymizeResult encapsulates the response from a ARXaaS anonymize call

class pyarxaas.models.anonymization_metrics.**AnonymizationMetrics**(*metrics: collections.abc.Mapping*)
Understand metrics from a anonymization process

attribute_generalization
Property for getting the attribute generalization

Returns Returns a dict containing all the quasi-identifying attributes and the transformation level used to anonymize the dataset.

elapsed_time
Property for getting the elapsed time for the anonymization process

Returns Returns the elapsed time in miliseconds

privacy_models
Property for getting the privacy models with the configurations used in the anonymization process.

Returns Returns a dict containing all the privacy models with the configuration used.

4.2.7 Privacy Model

If you are interrested in the theory behind each privacy models, you can read about it here: https://arx.deidentifier.org/overview/privacy-criteria/

class pyarxaas.privacy_models.**KAnonymity**(*k*)
Configuration class for K-Anonymity

Parameters **k** – Value of K to anonymize the dataset. K must have a value of 2 or higher to take effect.

class pyarxaas.privacy_models.**LDiversityDistinct**(*l*, *column_name*)
Configuration class for Distinct L-Diversity

Parameters

- **l** – Value of L to anonymize the dataset based on a column or dataset field that has a sensitive attribute. L must have a value of 2 or higher to take effect.
- **column_name** – Column or dataset field that has a sensitive attribute type.

class pyarxaas.privacy_models.**LDiversityGrassbergerEntropy**(*l*, *column_name*)
Configuration class for Grassberger Entropy L-Diversity

Parameters

- **l** – Value of L to anonymize the dataset based on a column or dataset field that has a sensitive attribute. L must have a value of 2 or higher to take effect.
- **column_name** – Column or dataset field that has a sensitive attribute type.

class pyarxaas.privacy_models.**LDiversityRecursive**(*l*, *c*, *column_name*)
Configuration class for Recursive L-Diversity

Parameters

- **l** – Value of L to anonymize the dataset based on a column or dataset field that has a sensitive attribute. L must have a value of 2 or higher to take effect.
- **c** – Value of C to anonymize the dataset based on a column or dataset field that has a sensitive attribute. c must have a value of 0.00001 or higher to take effect.
- **column_name** – Column or dataset field that has a sensitive attribute type.

class pyarxaas.privacy_models.**LDiversityShannonEntropy**(*l*, *column_name*)
Configuration class for Shannon Entropy L-Diversity

Parameters

- **l** – Value of L to anonymize the dataset based on a column or dataset field that has a sensitive attribute. L must have a value of 2 or higher to take effect.
- **column_name** – Column or dataset field that has a sensitive attribute type.

class pyarxaas.privacy_models.**PrivacyModel**
Documentation of the privacy models implemented in the ARXaaS service and the definition of the parameters each privacy model takes.

class pyarxaas.privacy_models.**TClosenessEqualDistance**(*t*, *column_name*)
Configuration class for Equal Distance T-Closeness

Parameters

- **t** – Value of T to anonymize the dataset based on a column or dataset field that has a sensitive attribute. T must have a value between 0.000001 to 1.0
- **column_name** – Column or dataset field that has a sensitive attribute type.

class pyarxaas.privacy_models.**TClosenessOrderedDistance**(*t*, *column_name*)
Configuration class for Ordered Distance T-Closeness

Parameters

- **t** – Value of T to anonymize the dataset based on a column or dataset field that has a sensitive attribute. T must have a value between 0.000001 to 1.0
- **column_name** – Column or dataset field that has a sensitive attribute type.

4.2.8 Hierarchy Builders

IntervalHierarchyBuilder

IntervalHierarchyBuilder represents the recipe to construct a hierarchy from in combination with a attribute list

class pyarxaas.hierarchy.interval_builder.interval_hierarchy_builder.**IntervalHierarchyBuild**

Represents a specific strategy for creating a value generalization hierarchy

add_interval(*from_n*, *to_n*, *label: str = None*)

Add a interval to the builder. from_n is inclusive, to_n is exclusive

Parameters

- **from_n** – create interval inclusive from this value
- **to_n** – create interval to this value
- **label** – (optional) set a string label for the interval

Returns None

level(*level*) → pyarxaas.hierarchy.level.Level

Get a level in the hierarchy

Parameters **level** – int representing the level

Returns Level

OrderHierarchyBuilder

OrderHierarchyBuilder represents the recipe to construct a hierarchy from in combination with a attribute list

class pyarxaas.hierarchy.order_builder.order_hierarchy_builder.**OrderHierarchyBuilder**

level(*level*) → pyarxaas.hierarchy.level.Level

Get a level in the hierarchy

Parameters **level** – int representing the level

Returns Level

RedactionHierarchyBuilder

RedactionHierarchyBuilder represents the recipe to construct a hierarchy from in combination with a attribute list

class pyarxaas.hierarchy.redaction_hierarchy_builder.**RedactionHierarchyBuilder**(*padding_char: str = ' ', redaction_char: str = '*', padding_order: pyarxaas.hierarchy.redaction_hierarchy_builder.RedactionHierarchyBuilder.Order = <Order.RIGHT_TO_LEFT: 'RIGHT_TO_LEFT'>, redaction_order: pyarxaas.hierarchy.redaction_hierarchy_builder.RedactionHierarchyBuilder.Order = <Order.RIGHT_TO_LEFT: 'RIGHT_TO_LEFT'>*)

Understands building redaction based hierarchies

class Order

An enumeration.

4.3 Example Notebooks

4.3.1 Example analyzation and anonymization of sensitive dataset

[1]:
```
from pyarxaas import ARXaaS
from pyarxaas.privacy_models import KAnonymity, LDiversityDistinct
from pyarxaas import AttributeType
from pyarxaas import Dataset
import pandas as pd
```

Create ARXaaS connection

[2]:
```
arxaas = ARXaaS("http://localhost:8080/") # connecting to online service
```

fetch sensitive data

[3]:
```
data_df = pd.read_csv("../data/data2.csv", sep=";")
```

[4]:
```
data_df
```

[4]:
```
   zipcode  age  salary         disease
0    47677   29       3   gastric ulcer
1    47602   22       4       gastritis
2    47678   27       5  stomach cancer
3    47905   43       6       gastritis
4    47909   52      11             flu
5    47906   47       8      bronchitis
6    47605   30       7      bronchitis
7    47673   36       9       pneumonia
8    47607   32      10  stomach cancer
```

Create Dataset

[6]:
```
dataset = Dataset.from_pandas(data_df)
```

Set the AttributeType for the dataset fields

[7]:
```
dataset.set_attribute_type(AttributeType.IDENTIFYING, 'salary')
```

Set Generalization Hierarchies

Note that if the hierarchy does not have a header row in the csv file, please set header=None in read_csv() or the first row will be interpreted as a header and ARXaaS will throw an exception for the missing hierarchy data.

[8]:
```
zipcode_hierarchy = pd.read_csv("../data/data2_zipcode_hierarchy.csv", sep=";",␣
↪header=None)
age_hierarchy = pd.read_csv("../data/data2_age_hierarchy.csv", sep=";", header=None)
disease_hierarchy = pd.read_csv("../data/data2_disease_hierarchy.csv", sep=";",␣
↪header=None)
```

[9]:
```
zipcode_hierarchy
```

[9]:
```
       0      1      2      3      4      5
0  47677  4767*  476**  47***  4****  *****
1  47602  4760*  476**  47***  4****  *****
2  47678  4767*  476**  47***  4****  *****
3  47905  4790*  479**  47***  4****  *****
4  47909  4790*  479**  47***  4****  *****
5  47906  4790*  479**  47***  4****  *****
6  47605  4760*  476**  47***  4****  *****
7  47673  4767*  476**  47***  4****  *****
8  47607  4760*  476**  47***  4****  *****
```

[10]:
```
dataset.set_hierarchy('age', age_hierarchy)
dataset.set_hierarchy("zipcode", zipcode_hierarchy)
dataset.set_hierarchy("disease", disease_hierarchy)
```

Create Privacy Models

[11]:
```
kanon = KAnonymity(4)
```

Create Risk Profile

[13]:
```
risk_profile = arxaas.risk_profile(dataset)
```

[14]:
```
risk_profile.re_identification_risk
```

[14]:
```
{'estimated_journalist_risk': 1.0,
 'records_affected_by_highest_prosecutor_risk': 1.0,
 'sample_uniques': 1.0,
 'lowest_risk': 1.0,
 'estimated_prosecutor_risk': 1.0,
 'highest_journalist_risk': 1.0,
 'records_affected_by_lowest_risk': 1.0,
 'average_prosecutor_risk': 1.0,
 'estimated_marketer_risk': 1.0,
 'highest_prosecutor_risk': 1.0,
 'records_affected_by_highest_journalist_risk': 1.0,
 'population_uniques': 1.0}
```

[15]:
```
risk_profile.distribution_of_risk_dataframe().head()
```

[15]:
```
    interval  recordsWithMaxmalRiskWithinInterval  \
0   ]50,100]                                  1.0
1  ]33.4,50]                                  0.0
2  ]25,33.4]                                  0.0
3    ]20,25]                                  0.0
4  ]16.7,20]                                  0.0

   recordsWithRiskWithinInteval
0                           1.0
1                           0.0
2                           0.0
3                           0.0
4                           0.0
```

[ ]:
```

```

Anonymize

[17]:
```
anon_result = arxaas.anonymize(dataset, [kanon])
```

[18]:
```
anon_result.dataset.to_dataframe()
```

[18]:
```
  zipcode age salary                disease
0   47***   *      *        stomach disease
1   47***   *      *        stomach disease
2   47***   *      *        stomach disease
3   47***   *      *        stomach disease
```

(continues on next page)

(continued from previous page)

```
4   47***   *       *  respiratory infection
5   47***   *       *  respiratory infection
6   47***   *       *  respiratory infection
7   47***   *       *  respiratory infection
8   47***   *       *         stomach disease
```

Anonymization Status

Anonymization status describes if ARXaaS was able to anonymize the dataset to comply with the provided Privacy Models.

[19]:
```
anon_result.anonymization_status
```

[19]:
```
'ANONYMOUS'
```

RiskProfile for the anonymized dataset

[20]:
```
anon_rp = anon_result.risk_profile
```

[21]:
```
anon_rp.re_identification_risk
```

[21]:
```
{'estimated_journalist_risk': 0.25,
 'records_affected_by_highest_prosecutor_risk': 0.4444444444444444,
 'sample_uniques': 0.0,
 'lowest_risk': 0.2,
 'estimated_prosecutor_risk': 0.25,
 'highest_journalist_risk': 0.25,
 'records_affected_by_lowest_risk': 0.5555555555555556,
 'average_prosecutor_risk': 0.2222222222222222,
 'estimated_marketer_risk': 0.2222222222222222,
 'highest_prosecutor_risk': 0.25,
 'records_affected_by_highest_journalist_risk': 0.4444444444444444,
 'population_uniques': 0.0}
```

[22]:
```
anon_rp.distribution_of_risk_dataframe().head(10)
```

[22]:
```
      interval  recordsWithMaxmalRiskWithinInterval  \
0     ]50,100]                             1.000000
1    ]33.4,50]                             1.000000
2    ]25,33.4]                             1.000000
3      ]20,25]                             1.000000
4    ]16.7,20]                             0.555556
5  ]14.3,16.7]                             0.000000
6  ]12.5,14.3]                             0.000000
7    ]10,12.5]                             0.000000
8       ]9,10]                             0.000000
9        ]8,9]                             0.000000

   recordsWithRiskWithinInteval
0                      0.000000
1                      0.000000
2                      0.000000
3                      0.444444
4                      0.555556
```

(continues on next page)

(continued from previous page)

```
5                      0.000000
6                      0.000000
7                      0.000000
8                      0.000000
9                      0.000000
```

[ ]:

4.3.2 Hierachy generation using PyARXaaS

[1]:
```
from pyarxaas import ARXaaS
from pyarxaas.privacy_models import KAnonymity, LDiversityDistinct
from pyarxaas import AttributeType
from pyarxaas import Dataset
from pyarxaas.hierarchy import IntervalHierarchyBuilder, RedactionHierarchyBuilder,␣
↪OrderHierarchyBuilder, DateHierarchyBuilder
import pandas as pd
```

Create connection to ARXaaS

[2]:
```
arxaas = ARXaaS("http://localhost:8080")
```

Fetch data

[3]:
```
data_df = pd.read_csv("../data/data2.csv", sep=";")
```

[4]:
```
data_df
```

[4]:

	zipcode	age	salary	disease
0	47677	29	3	gastric ulcer
1	47602	22	4	gastritis
2	47678	27	5	stomach cancer
3	47905	43	6	gastritis
4	47909	52	11	flu
5	47906	47	8	bronchitis
6	47605	30	7	bronchitis
7	47673	36	9	pneumonia
8	47607	32	10	stomach cancer

Create Redaction based hierarchy

Redaction based hierarchies are hierarchies suited best for categorical but numeric values. Attributes such as zipcodes are a prime canditate. The hierarchy strategy is to delete one number at the time from the attribute column until the privacy model criteria is meet. The hierchy builder can be configured to start deleting from either direction, but will default to RIGHT_TO_LEFT. Redaction hierarchies are the least effort hierarchy to create.

1. Extract column to create hierarchy from

[5]:
```
zipcodes = data_df["zipcode"].tolist()
zipcodes
```

[5]:
```
[47677, 47602, 47678, 47905, 47909, 47906, 47605, 47673, 47607]
```

2. Create hierarchy builder to use

Here we are specifying a character to use and the order the redaction should follow.

[6]:
```
redaction_based = RedactionHierarchyBuilder(redaction_char="",
                                            redaction_order=RedactionHierarchyBuilder.
↪Order.LEFT_TO_RIGHT)
```

3. Call the ARXaaS service to create the hierarchy

[7]:
```
redaction_hierarchy = arxaas.hierarchy(redaction_based, zipcodes)
```

[8]:
```
redaction_hierarchy
```

[8]:
```
[['47677', '7677', '677', '77', '7', ''],
 ['47602', '7602', '602', '02', '2', ''],
 ['47678', '7678', '678', '78', '8', ''],
 ['47905', '7905', '905', '05', '5', ''],
 ['47909', '7909', '909', '09', '9', ''],
 ['47906', '7906', '906', '06', '6', ''],
 ['47605', '7605', '605', '05', '5', ''],
 ['47673', '7673', '673', '73', '3', ''],
 ['47607', '7607', '607', '07', '7', '']]
```

Redaction hiearchy without configuration

[9]:
```
no_config_redaction_based = RedactionHierarchyBuilder() # Create builder
redaction_hierarchy = arxaas.hierarchy(no_config_redaction_based, zipcodes) # pass␣
↪builder and column to arxaas
redaction_hierarchy
```

[9]:
```
[['47677', '4767*', '476**', '47***', '4****', '*****'],
 ['47602', '4760*', '476**', '47***', '4****', '*****'],
 ['47678', '4767*', '476**', '47***', '4****', '*****'],
 ['47905', '4790*', '479**', '47***', '4****', '*****'],
 ['47909', '4790*', '479**', '47***', '4****', '*****'],
 ['47906', '4790*', '479**', '47***', '4****', '*****'],
 ['47605', '4760*', '476**', '47***', '4****', '*****'],
 ['47673', '4767*', '476**', '47***', '4****', '*****'],
 ['47607', '4760*', '476**', '47***', '4****', '*****']]
```

Create interval based hierarchy

Interval based hierarchies are well suited for continous numeric values. Attributes such as age, income or credit score are typical generalized with a interval hierarchy. The Interval hiearchy builder requires the user to specify intervals in

which to generalize values in the attribute into. Optionally these intervals can be labeled. In addition intervals can be grouped upwards using levels and groups to create a deeper hierarchy

1. Extract column to create hierarchy from

[10]:
```
column = data_df["age"].tolist()
column
```

[10]:
```
[29, 22, 27, 43, 52, 47, 30, 36, 32]
```

2. Create hierarchy builder to use

[11]:
```
interval_based = IntervalHierarchyBuilder()
```

3. Add intervals to the builder. The intervals must be continous(without gaps)

[12]:
```
interval_based.add_interval(0,18, "child")
interval_based.add_interval(18,30, "young-adult")
interval_based.add_interval(30,60, "adult")
interval_based.add_interval(60,120, "old")
```

4. (Optionally) Add groupings. Groupings are added to a specific level and are order based according to the interval order

[13]:
```
interval_based.level(0)\
    .add_group(2, "young")\
    .add_group(2, "adult");
```

3. Call the ARXaaS service to create the hierarchy

[14]:
```
interval_hierarchy = arxaas.hierarchy(interval_based, column)
```

[15]:
```
interval_hierarchy
```

[15]:
```
[['29', 'young-adult', 'young', '*'],
 ['22', 'young-adult', 'young', '*'],
 ['27', 'young-adult', 'young', '*'],
 ['43', 'adult', 'adult', '*'],
 ['52', 'adult', 'adult', '*'],
 ['47', 'adult', 'adult', '*'],
 ['30', 'adult', 'adult', '*'],
 ['36', 'adult', 'adult', '*'],
 ['32', 'adult', 'adult', '*']]
```

Create Order based hierarchy

Order based hierarchies are suited for categorical attributes. Attributes such as country, education level and employment status

1. Extract column to create hierarchy from

[16]:
```
diseases = data_df["disease"].tolist()
```

2. Strip to uniques

[17]:
```
unique_diseases = set(diseases)
unique_diseases = list(unique_diseases)
unique_diseases.sort()
unique_diseases
```

[17]:
```
['bronchitis',
 'flu',
 'gastric ulcer',
 'gastritis',
 'pneumonia',
 'stomach cancer']
```

3. Order column values

As this is a categorical attribute ARXaaS has no way of knowing how to group the values except for the ordering of the values.

[18]:
```
unique_diseases[2], unique_diseases[4] = unique_diseases[4], unique_diseases[2]
unique_diseases
```

[18]:
```
['bronchitis',
 'flu',
 'pneumonia',
 'gastritis',
 'gastric ulcer',
 'stomach cancer']
```

2. Create hierarchy builder to use

[19]:
```
order_based = OrderHierarchyBuilder()
```

3. Group the values

Note that the groups are applied to the values as they are ordered in the list. Adding labels are optional, if labels are not set the resulting field will be a concatination of the values included in the group.

[20]:
```
order_based.level(0)\
    .add_group(3, "lung-related")\
    .add_group(3, "stomach-related")
```

[20]:
```
Level(level=0, groups={Group(grouping=3, label=lung-related): None, Group(grouping=3,␣
↪label=stomach-related): None})
```

3. Call the ARXaaS service to create the hierarchy

[21]:
```
order_hierarchy = arxaas.hierarchy(order_based, unique_diseases)
```

[22]:
```
order_hierarchy
```

[22]:
```
[['bronchitis', 'lung-related', '*'],
 ['flu', 'lung-related', '*'],
 ['pneumonia', 'lung-related', '*'],
 ['gastritis', 'stomach-related', '*'],
 ['gastric ulcer', 'stomach-related', '*'],
 ['stomach cancer', 'stomach-related', '*']]
```

Create Date based hierarchy

Date based hierarchies are used for date values that follow the Java SimpleDateFormat

[26]:
```
dates = ["2020-07-16 15:28:024",
         "2019-07-16 16:38:025",
         "2019-07-16 17:48:025",
         "2019-07-16 18:48:025",
         "2019-06-16 19:48:025",
         "2019-06-16 20:48:025"]
```

1. Create the builder

the first parameter to the builder is the date_format. The date format specifies how ARXaaS should handle and parse the date strings. The format should follow Java SimpleDateFormat formating. link: https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

[31]:
```
date_based = DateHierarchyBuilder("yyyy-MM-dd HH:mm:SSS",
                            DateHierarchyBuilder.Granularity.SECOND_MINUTE_HOUR_DAY_
↪MONTH_YEAR,
                            DateHierarchyBuilder.Granularity.MINUTE_HOUR_DAY_MONTH_YEAR,
                            DateHierarchyBuilder.Granularity.YEAR)
```

[32]:
```
date_hierarchy = arxaas.hierarchy(date_based , dates)
```

[33]:
```
date_hierarchy
```

[33]:
```
[['2020-07-16 15:28:024', '16.07.2020-15:28:00', '16.07.2020-15:28', '2020'],
 ['2019-07-16 16:38:025', '16.07.2019-16:38:00', '16.07.2019-16:38', '2019'],
 ['2019-07-16 17:48:025', '16.07.2019-17:48:00', '16.07.2019-17:48', '2019'],
 ['2019-07-16 18:48:025', '16.07.2019-18:48:00', '16.07.2019-18:48', '2019'],
 ['2019-06-16 19:48:025', '16.06.2019-19:48:00', '16.06.2019-19:48', '2019'],
 ['2019-06-16 20:48:025', '16.06.2019-20:48:00', '16.06.2019-20:48', '2019']]
```

Example anonymization

[23]:
```
dataset = Dataset.from_pandas(data_df)
```

[24]:
```
dataset.set_attribute_type(AttributeType.IDENTIFYING, "salary")
```

[25]:
```
dataset.describe()
```

```
data:
  headers:
    ['zipcode', 'age', 'salary', 'disease']
rows:
    [47677, 29, 3, 'gastric ulcer']
    [47602, 22, 4, 'gastritis']
    [47678, 27, 5, 'stomach cancer']
    [47905, 43, 6, 'gastritis']
    [47909, 52, 11, 'flu']
    ...
attributes:
  field_name=zipcode, type=QUASIIDENTIFYING, hierarchy=None
  field_name=age, type=QUASIIDENTIFYING, hierarchy=None
  field_name=salary, type=IDENTIFYING, hierarchy=None
  field_name=disease, type=QUASIIDENTIFYING, hierarchy=None
```

[26]:
```
dataset.set_hierarchy("age", interval_hierarchy)
```

[27]:
```
dataset.set_hierarchy("zipcode", redaction_hierarchy)
```

[28]:
```
dataset.set_hierarchy("disease", order_hierarchy)
```

[29]:
```
anon_result = arxaas.anonymize(dataset=dataset, privacy_models=[KAnonymity(2)])
```

[30]:
```
anon_result.dataset.to_dataframe()
```

[30]:

	zipcode	age	salary	disease
0	47***	young-adult	*	stomach-related
1	47***	young-adult	*	stomach-related
2	47***	young-adult	*	stomach-related
3	47***	adult	*	stomach-related
4	47***	adult	*	lung-related
5	47***	adult	*	lung-related
6	47***	adult	*	lung-related
7	47***	adult	*	lung-related
8	47***	adult	*	stomach-related

[ ]:

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P

Q

R

S

T