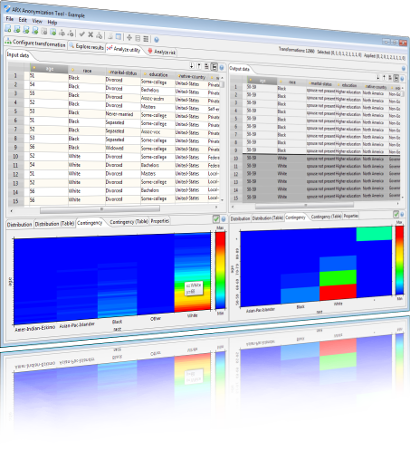
# What Is ARX:



## History of arx:

Arx is a cloud-based service that provides data anonymization and privacy-enhancing technologies to organisations. The history of ARX can be traced back to the early development of the arx algorithm for cryptography. Arx was originally designed to be a more efficient and secure way of implementing certain cryptographic operations, such as encryption and authentication. However, it was soon recognized that arx could also be used for data anonymization and de-identification, by replacing sensitive data with random or pseudonymous values while preserving the statistical properties of the original data. It uses the arx cryptographic primitive to protect data by replacing sensitive information with random or pseudonymous values while preserving the statistical properties of the original data.

Arx is a method of combining the basic bitwise operations of addition, rotation, and XOR to implement encryption and authentication. It was designed to be a more efficient and secure way of implementing certain cryptographic operations.

Arx module offers a range of anonymization techniques, including k-anonymity, L diversity,T-closeness and differential privacy, which help organisations comply with data protection regulations while preserving data utility. The service is accessed through an API that allows developers to easily integrate the anonymization capabilities of Arxaas into their existing applications. The service is also available as a software but it has limited use cases and cannot be customised. There is also a python module called ‘pyarxaas’ which helps users access the arx service right from their python IDE

Arxaas uses a scalable infrastructure and a cloud-based architecture, which makes it ideal for processing large volumes of data in a distributed manner. The service also provides security and compliance features such as role-based access control and audit logging, to ensure that sensitive data is handled in a secure and compliant manner. However, it’s python module pyarxaas does not support a lot of the additional functionality that the app provides

Overall, Arxaas provides a comprehensive solution for organisations looking to protect the privacy of their data, while also ensuring that they remain compliant with data protection regulations. It is used by different types of organisations, like hospitals, banks, and online stores, to protect their customers' privacy and comply with data protection laws. It is also used widely to make sure that the privacy of their customers is maintained

## Why do we need arx:

ARX might seem like a ‘nice-to-have’ for a lot of normal businesses dealing with the data however, with the concerns of data privacy and ethics rising the applications of the data can be far and wide. Here are some use cases which define the role of arx in an organisation:

1. Compliance: Many organisations are subject to data protection regulations, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA), that require them to protect the privacy of their customers' data. Arxaas provides privacy-enhancing technologies that help organisations comply with these regulations.
2. Data sharing: Organisations may need to share data with third parties, such as researchers or business partners, while ensuring that the data is anonymized and does not contain any personally identifiable information. Arxaas can help organisations anonymize their data so that it can be shared safely and securely.
3. Risk management: Organisations may want to reduce the risk of data breaches or other security incidents by de-identifying sensitive data. Arxaas can help organisations minimise the risk of data exposure by anonymizing sensitive data before it is processed or stored.
4. Data analysis: Organisations may want to use data for analysis or research purposes while protecting the privacy of individuals whose data is being used. Arxaas can help organisations preserve the utility of the data while ensuring that the privacy of individuals is protected.

One example of an organisation using Arxaas is the UK's National Health Service (NHS), which has used the service to anonymize patient data for research purposes. Another example is the German National Library of Science and Technology (TIB), which has used Arxaas to anonymize data on digital preservation and research data management. Additionally, several academic institutions have used Arxaas for research purposes, including the University of Copenhagen and the University of Vienna.

In summary, the Arxaas service can help organisations comply with data protection regulations, share data safely and securely, reduce the risk of data breaches, and preserve data utility while protecting individual privacy.

## Flavours of arx

To ensure that organisations and users across all domains can take advantage of their services, arx is available as a variety of services:

**ARXAAS**: is a cloud based service which can handle large amounts of data. Furthermore, the arxaas instance can be run locally using docker.

**Pyarxaas:** Pyarxaas is a python module for implementing functions for the arx, it supports a limited range of privacy models and does not have the additional functionality that the arx software does but it can be used to connect to the local arx instance and perform functions on the data

**API**: All the functionality of the arx is also available as an API which can be used to integrate to the surface

## Installation guide for arx:

### Installing ARXAAS:

The arxaas can be installed by installing the docker image and your system and running the local docker image

In order to run the local docker image, you have to make sure that the docker and docker desktop are installed.

Before installing docker on the system, the curl and sudo need to be installed or updated on the system

### Installing curl on Ubuntu 22.04

1. Update the system by running the following command:

| sudo apt get update |
| --- |

1. Installing curl on the system:

| sudo apt install curl |
| --- |

1. After the curl has been installed, it can be verified using the curl command

| $curl |
| --- |

### 

### Installing sudo on Ubuntu 22.04

First we update the database using:

| Sudo apt-get update |
| --- |

Then we install or update the sudo using:

| sudo apt-get -y install sudo |
| --- |

Now that the sudo has been installed, we can move on to installing docker for ubuntu

### Installing docker for Ubuntu 22.04

Before we install the docker engine on our system, it is crucial to update our packages and dependencies and that can be done by running the following command:

| sudo apt-get update |
| --- |

After updating the same, we install packages to allow apt to use a repository over HTTPS:

| sudo apt-get install \  ca-certificates \  curl \  gnupg \  lsb-release |
| --- |

Now we add the docker’s official gpg key

| sudo mkdir -m 0755 -p /etc/apt/keyrings curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg |
| --- |

Now we set up the repository

| echo \  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \  $(lsb\_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null |
| --- |

After that is done, we move on to installing the docker

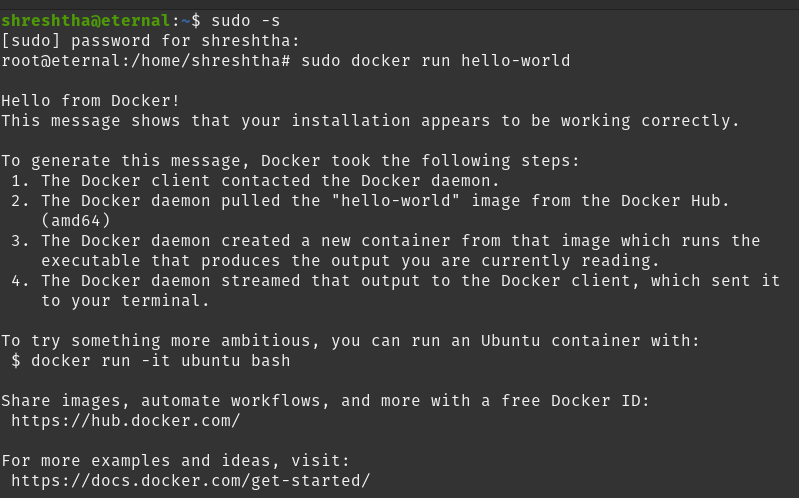


| sudo apt-get install docker-ce docker-ce-cli containerd.io docker-build-plugin docker-compose-plugin |
| --- |

The docker is now installed

We can verify the install by running ‘hello world’ using the command:

| sudo docker run hello-world |
| --- |



This is the output after running the hello world command which indicates that the docker has been running fine

After the docker has been installed, we can install the docker image of the pyarxaas

| docker pull navikt/arxaas |
| --- |

And then running

| docker run -p 8080:8080 navikt/arxaas |
| --- |

and run the image locally using the port 8080. Simply typing the following url in your browser after running the docker command would run your local docker instance

| http://localhost:8080/ |
| --- |

## 

This is what the arx local docker image looks like

## Installing pyarxaas traditionally:

The pyarxaas can be installed by cloning the github repository

| git clone https://github.com/navikt/pyarxaas |
| --- |

The pyarxaas can also be installed using the package manager pip.

### Installing pip for ubuntu:

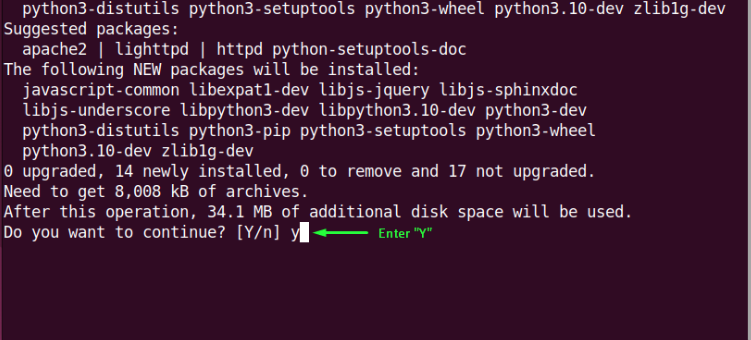
Firstly, run an upgrade on the system using:

| Sudo apt-get upgrade |
| --- |

After running the upgrade, we install the pip by using

| $ sudo apt install python3-pip |
| --- |

When prompted with the question of do you want to continue the installation, please answer yes;

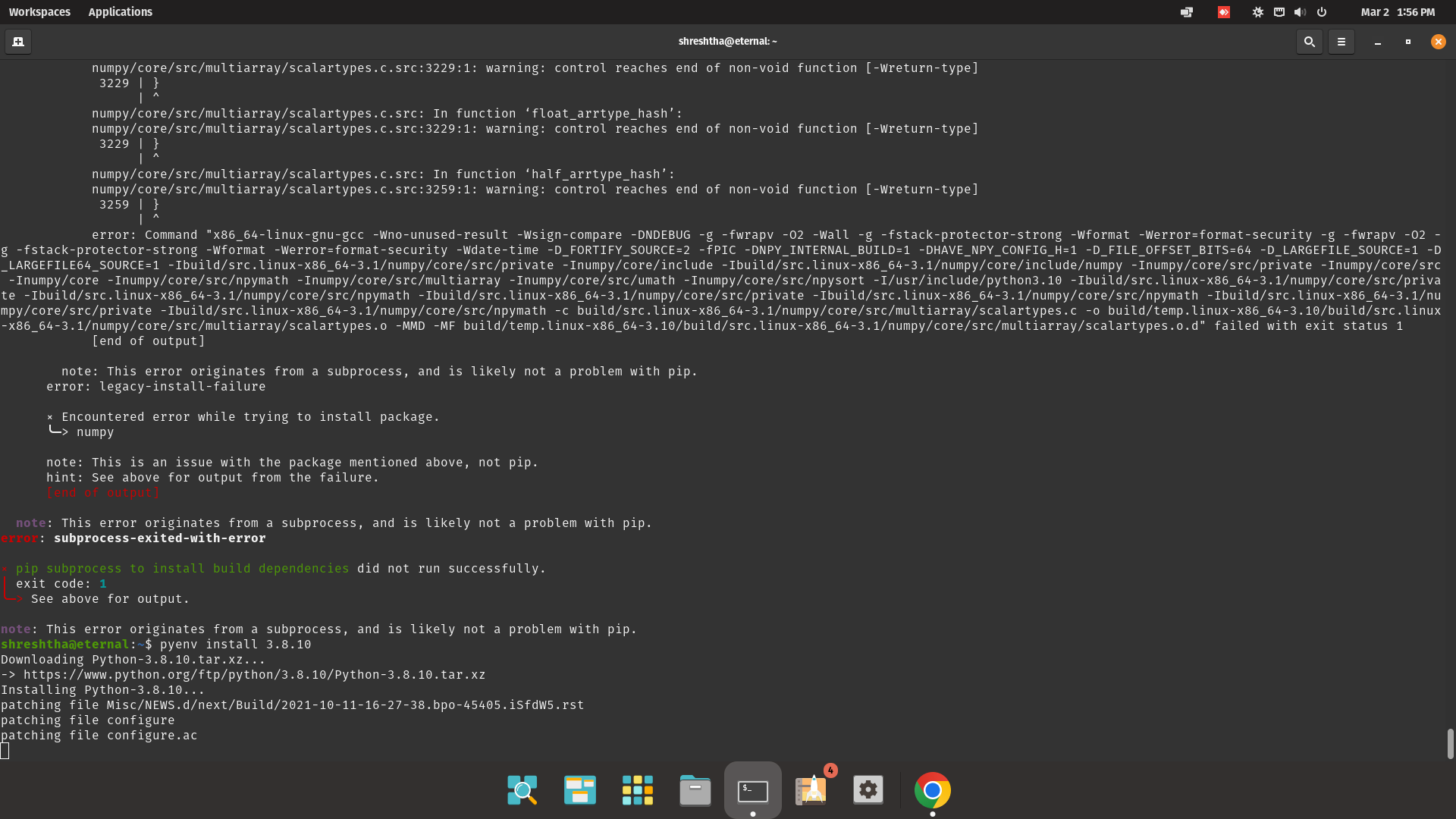


Verify if the pip has been installed successfully by checking the version of the pip

| $ pip3 --version |
| --- |

Now that the pip has been installed, we can start with the installation process of pyarxaas

One thing to note before installing pyarxaas is that without downgrading your system version of python from the latest (python 3.10 or 3.11) to the more older versions of python (3.8.10 or older) pyarxaas might not be installed and you might face errors such as



Hence we need a different environment of python. In order to do that, we can either downgrade the entire system (which is not recommended because it might break other systems dependencies which run on the current version of the python) or you can install multiple versions of python (using a virtual environment) or pyenv

### Installing pyenv for Ubuntu 22.04

1. Update the system and the dependencies using the command

| Sudo apt -get upgrade |
| --- |

1. After updating the system, we can now download the pyenv script and run it using

| $ curl https://pyenv.run | bash |
| --- |

1. After the installation is complete, we add the pyenv variable to the bash file by using the exec command

| export PATH="$HOME/.pyenv/bin:$PATH" && eval "$(pyenv init --path)" && echo -e 'if command -v pyenv 1>/dev/null 2>&1; then\n eval "$(pyenv init -)"\nfi' >> ~/.bashrc |
| --- |

The pyenv is now installed and can be verified by running ‘pyenv- - version’.

Note: In order to see the latest version of the pyenv and the changes that take place, you might need to restart the shell



### Installing python 3.8.10 using pyenv:

1)Installing a specific version of python using pyenv is fairly simple, we can do so using

| pyenv install -v 3.8.10 |
| --- |

2)Now that we have installed the version of the python, we can set it to global using

| Pyenv global 3.8.10  #we can check the version of the python using python —-version |
| --- |

## 

Thus, python 3.8.10 has been installed. With that being installed, we can now install pyarxaas

### Installing Pyarxaas for Ubuntu 22.04

1. Upgrading the dependencies using

| sudo apt-get upgrade |
| --- |

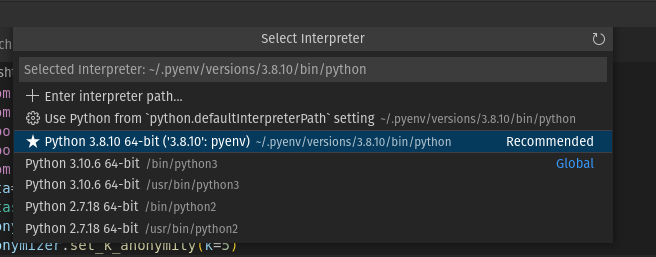
1. After we are done upgrading the system, we can now install pyarxaas using the following command

| Pip install pyarxaas |
| --- |

Upon being asked if you want to install pyarxaas please type yes. Once the arx has been successfully installed, we can verify the install using:

| Pyarxaas -version |
| --- |

After installing the pyarxaas and the python 3.8.10, we still have to change the python version interpreter. This can be done by going into vscode in the prompt and choosing your desired version of python to run



Thus, the kernel of python is ready to run our code

This works perfectly fine. However, please do note that if you choose a version of python that is not 3.8.10 to run the pyarxaas code, it will simply not run because we have installed the pyarxaas on python 3.8.10 version

## Troubleshooting common issues encountered while installing pyarxaas:

* **Curl Error**: While installing docker or pyenv, if your terminal throws in error and doesn’t recognize curl, you can simply update or reinstall the curl by using the command:

| sudo apt install curl/buster-backports |
| --- |

If this still does not work, you can check the path of curl to your system and manually specify the path where your curl is located by using:

| .bashrc  export PATH=/usr/curl/<Directory>/bin:$PATH |
| --- |

Hopefully, this should solve all the errors

* **Sudo error:** If you encounter a sudo error, it is best to install the sudo using sudo apt-get upgrade
* **Numpy/Pandas error:** Since pyarxaas is an older module with discontinued support, the chances of getting this error are high. This error usually occurs when the system has pandas and/or numpy installed which are not supported by pyarxaas. To solve them, simply open the terminal

| Python -version Import numpy Numpy -version #check for the version supported by pyarxaas #downgrade the same using: python -m pip install numpy==x.y.z |
| --- |

* **Pip error**: After installation of pip, pip errors should be unlikely. However, if you still face one of the pip errors, consider changing the version of pip from pip3 to pip in your commands and vice versa. E.g

| pip 3 install python |
| --- |

* **Docker error:** If the docker faces issues during installation, check if your machine supports virtualization by using

| sudo kvm-ok |
| --- |

If your computer supports the virtualization, then you might get an output like:



* **Python error:** For the errors corresponding to python during the installation, please check if the installation of python has been done properly by typing

| Python —-version |
| --- |

If the issue still persists, you might want to point to the path where the python is stored in your machine manually

* **No module named pyarxaas:** Having encountered this issue multiple times, the most common cause for this issue is not using the version of python where the pyarxaas was installed. Simply changing the version of python can resolve this issue

## What is pyarxaas

Pyarxaas is the python wrapper for accessing the arx functions on a local arx instance. It can be downloaded via github or using pip. It lets users convert their dataframe (it has pandas support) to an arx dataset. There are functions available for setting and creating hierarchy on the data, the risk analysis and using privacy models on the data. Much of the other arx software functionality is not available in the pyarxaas module but the module provides ease of access and easy integration with the python code

## Functions supported

### ARXaas:

Arxaas is used to connect to your local instance by passing the url to the local instance E.g

| arxaas = ARXaaS("http://localhost:8080/") |
| --- |

In the given code block, we created an object called arxaas which has the arx instance that will be used to perform functions on the data

### AttributeType

Attribute type is used to set the attributes to a given dataset. Usually the columns of a dataset fall into either of these categories:

**Identifying:** A column of a dataset is called identifying when it contains values that can be used to directly identify the attributes of a person. E.g. Name, E-mail, Aadhar card number etc

**Quasi Identifying:** Quasi Identifying columns are the columns which do not contain the identifying information directly but they can be used in combination with other data and or columns to re identify the person

Let us take an example to understand the quasi identifying columns a little better:

A dataset contains zip code, gender and items purchased. Now, the columns on their own might not be enough to identify the user who purchased the dataset but they can be used in combination. E.g zip code narrows down the area of the user and we can use the gender column to filter out the gender of the user. This dataset in conjunction with customer reviews or lets say order placed can be used to identify the user and hence reveals the sensitive personal information.

Thus, it is really important that quasi identifiers are dealt with properly in the data. When applying a privacy model to the data, it is crucial to check the list of quasi identifiers and the re identification

**Sensitive:** Sensitive attributes are the columns in your data which contain important and crucial information about the user which cannot be reveleade. E.g Bank account number, social security details etc etc. These columns should be removed from the data entirely

**Insensitive:** Insensitive columns are the columns which do not contain any identifying or sensitive information and they need not be anonymized. E.g when looking at the customer dataset insensitive information could be the brand he rejected or the number of items that he purchased

### Dataset

Dataset function takes in the pandas dataframe as an input and converts it to a dataset that is supported by pyarxaas containing information about generalisation, attribute types, privacy model etc

Usage:

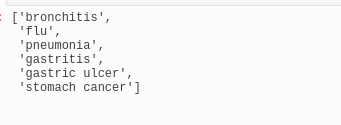
| dataset = Dataset.from\_pandas(data) |
| --- |

### Types of hierarchy supported

Hierarchy is nothing but the levels of generalisation defined when anonymizing the data.The pyarxaas module supports a variety of hierarchy types. You can either set your own hierarchy for generalisation or create one in pyarxaas. Here are some of the supported hierarchy types in pyarxaas:

**Order based Hierarchy:**  Order based hierarchy are suited for the categorical variables. I.e The value of variables can not be something other than the value from the specified list of values

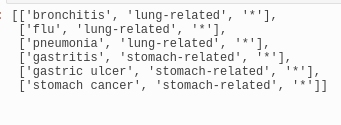
Here, we have a list of diseases for a patient dataset

****

This is how we set an order based hierarchy for this dataset

| **#create the builder**  **order\_based = OrderHierarchyBuilder()**  **#add groupings for the level order\_based.level(0)\  .add\_group(3, "lung-related")\  .add\_group(3, "stomach-related")**  **#set the hierarchy for those levels afterwards** |
| --- |

This is what the hierarchy looks like for the following diseases:



**Redaction based hierarchy:** Redaction based hierarchy are best suited for numeric but categorical values. E.g phone number or zip code. They take in a list and delete one number at a time from the attribute column until the privacy criteria is met

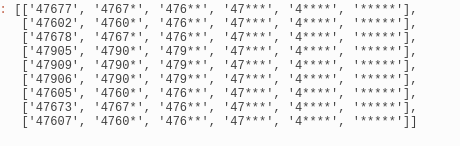
Example:

Here we have a list of zip codes



We can call the redaction based hierarchy by using:

| builder= RedactionHierarchyBuilder() # Create builder redaction\_hierarchy = arxaas.hierarchy(builder, zip codes) |
| --- |



**Interval based hierarchy:** Interval based hierarchy typically works well for the continuous numeric values such as age, height, weight, credit card number etc. The attribute column here gets divided into generalised level based instead of the actual numbers

Let's say we have a list of the age group for the customer data:



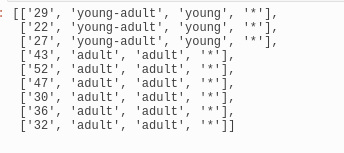
The interval based hierarchy for the same can be generated using:

| #creating the builder  interval\_based = IntervalHierarchyBuilder()  #adding the range based interval 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" |
| --- |

We can also add groupings to further specify the level:

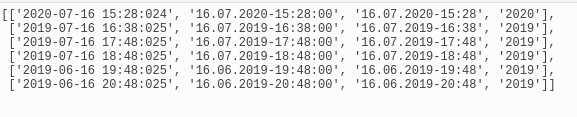
| interval\_based.level(0)\  .add\_group(2, "young")\  .add\_group(2, "adult"); |
| --- |

This is what the final hierarchy looks like:



**Date/Time based hierarchy :** Date based hierarchies are for the columns which follow the JavaSimpleDateformat and they are rarely used in practice. This is what they look like, the process of creating a date based hierarchy is fairly simple

| 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) |
| --- |



### privacy\_models

What are privacy models?

Privacy models refer to a set of techniques, methods, and frameworks that are used to protect the privacy of individuals in a data collection or analysis process. These models provide a structured approach to ensure that sensitive or personally identifiable information is not disclosed, while still allowing useful information to be extracted for research or analysis purposes.

Privacy models often involve a combination of statistical, cryptographic, and computational techniques to achieve their goals. They can be used to enforce different levels of privacy protection, depending on the specific needs and requirements of a particular use case or application.

The most common privacy models include k-Anonymity, l-Diversity, t-Closeness, and Differential Privacy. These models provide different levels of protection and are suitable for different types of data and analysis scenarios.

Overall, privacy models are essential for protecting individuals' privacy in an increasingly data-driven world, where personal information is often collected, processed, and shared without their knowledge or consent. They help ensure that sensitive information remains confidential and that individuals' privacy rights are respected.

What is a risk profile and how to generate one

**Privacy Models supported by arx:**

ARX supports the following privacy models:

**K-anonymity**:

K-Anonymity is a privacy model designed to protect the identity of individuals whose personal information is being used in a dataset. This model ensures that an individual cannot be re-identified from a dataset by ensuring that each record in the dataset is indistinguishable from at least k-1 other records.

* In other words, K-Anonymity is a technique that anonymizes data by grouping individuals with similar characteristics into clusters, where each cluster contains at least k individuals. By doing so, it makes it harder for an attacker to identify a particular individual in the dataset.
* For example, if a dataset contains age, gender, and zip code of individuals, the K-Anonymity model will group individuals who share the same age, gender, and zip code into a cluster. This cluster will contain at least k individuals, making it impossible to identify a specific individual from this group.
* K-Anonymity has become increasingly important in today's world where data breaches and privacy violations are becoming more common. It is used in many different applications, including healthcare, finance, and marketing, to protect sensitive personal information.

**L-diversity:**

* L-Diversity is a privacy model that ensures that sensitive information of individuals in a dataset is not revealed by adding diversity to the dataset. It aims to prevent attackers from identifying individuals by adding enough diversity to the dataset to make it difficult to link specific sensitive attributes to a particular individual.
* In other words, L-Diversity ensures that every group of individuals in the dataset is diverse enough in terms of sensitive attributes such as race, religion, or medical condition, so that it is not possible to link these attributes to a specific individual in the group.
* For example, let's say a dataset contains medical records of patients, including their age, gender, medical conditions, and zip code. If this dataset is not properly anonymized, an attacker may be able to identify individuals based on their medical conditions and zip code. To prevent this, L-Diversity can be applied by ensuring that every group of patients in the dataset contains a diverse range of medical conditions and zip codes.
* For instance, consider a group of patients with diabetes, in which all patients belong to the same age group and live in the same zip code area. By applying L-Diversity, the dataset could be modified to include a diverse range of age groups and zip codes within this group of patients with diabetes. This would make it more difficult for an attacker to identify an individual based on their medical condition, age, and zip code.
* L-Diversity is particularly useful in scenarios where sensitive attributes need to be protected while still allowing data to be used for analysis, research, or other purposes. It is used in many different applications, including healthcare, finance, and social research.

**T-closeness:**

* T-closeness is a privacy model that measures the degree to which a dataset preserves the privacy of individuals by ensuring that the distribution of sensitive attributes in the dataset is similar to their distribution in the general population. The aim is to prevent attackers from using background knowledge to link specific sensitive attributes to particular individuals in the dataset.
* In other words, T-closeness ensures that the distribution of sensitive attributes (such as age, race, or medical condition) in the dataset is not significantly different from the distribution of those attributes in the general population, to avoid revealing sensitive information about specific individuals.
* T-closeness can be measured using a distance metric that measures the difference between the distribution of a sensitive attribute in the dataset and its distribution in the general population. The goal is to minimise this distance, or "closeness", to protect the privacy of individuals in the dataset.
* For example, consider a dataset that contains medical records of patients, including their age, gender, medical condition, and zip code. If this dataset is not properly anonymized, an attacker may be able to identify individuals based on their medical condition and zip code. To prevent this, T-closeness can be applied by ensuring that the distribution of medical conditions and zip codes in the dataset is similar to their distribution in the general population.
* T-closeness has become increasingly important in today's world where data breaches and privacy violations are becoming more common. It is used in many different applications, including healthcare, finance, and marketing, to protect sensitive personal information.

**Differential Privacy :**

* Differential privacy is a privacy model that ensures that individual data points in a dataset remain private, even when the dataset is shared or analysed. It provides strong privacy guarantees by ensuring that any individual's data cannot be distinguished from any other individual's data, no matter how much background knowledge an attacker has.
* In other words, differential privacy is a technique that allows data to be used for analysis without revealing sensitive information about specific individuals. It does this by adding noise to the data before it is shared or analysed, which makes it difficult for an attacker to determine whether a particular individual is present in the dataset.
* Differential privacy is measured by a parameter called epsilon (ε), which represents the maximum amount of privacy loss that is allowed. A smaller epsilon value indicates stronger privacy protection.
* For example, consider a dataset that contains information about people's salaries. If the dataset is not properly anonymized, an attacker could potentially identify individuals based on their salary information. Differential privacy can be applied by adding random noise to the salary information, such that the noise makes it impossible to identify the salary of any specific individual.
* Differential privacy has become increasingly important in today's world where data privacy is a critical issue. It is used in many different applications, including healthcare, finance, and social research, to protect sensitive personal information while still allowing data to be used for analysis and research.

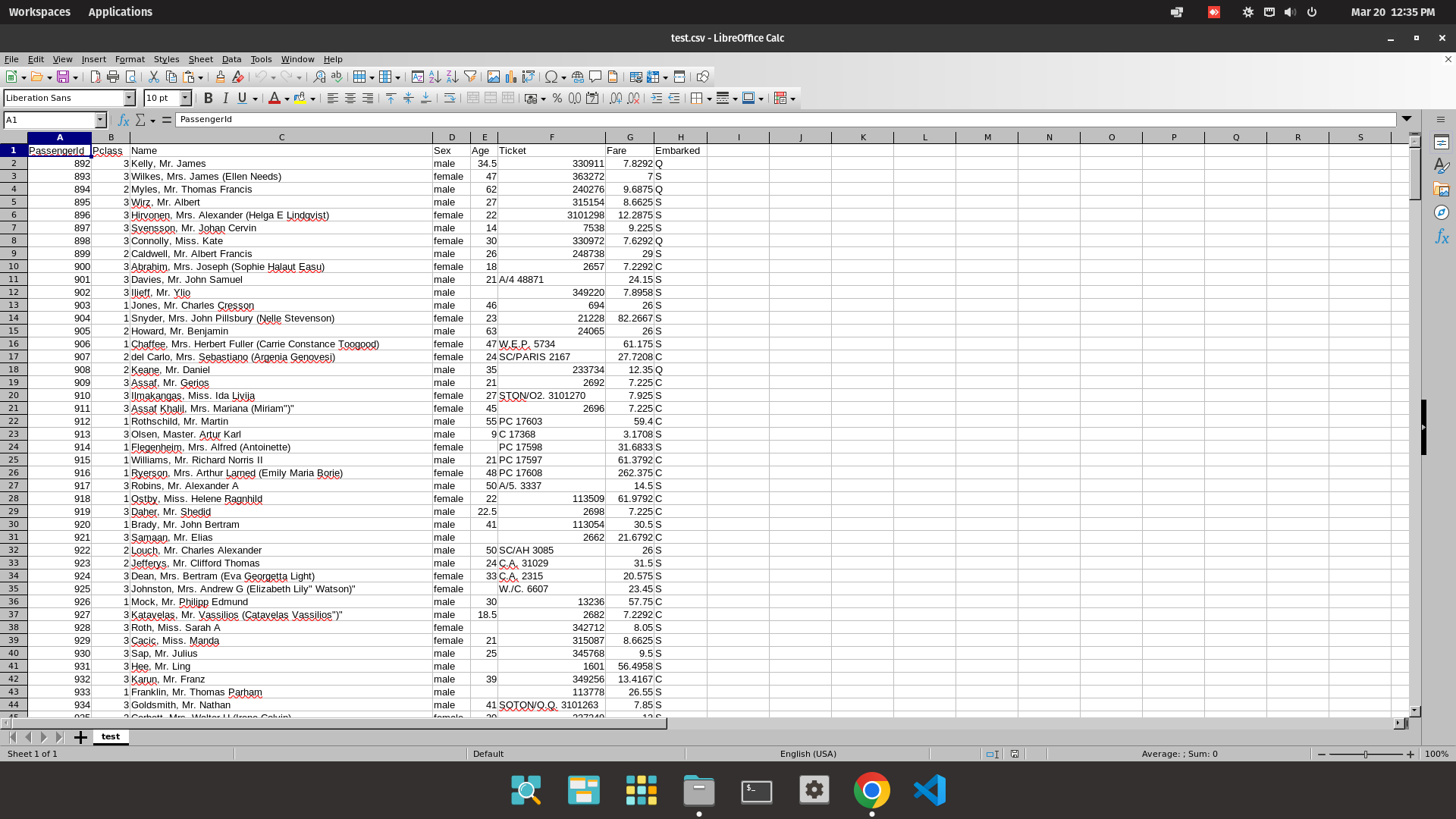
| #sample code to set the anonymization for the dataset  kanon = K Anonymity(2) ldiv = LDiversityDistinct(2, "disease") # in this example the dataset has a disease field anonymize\_result = arxaas.anonymize(dataset, [kanon, ldiv], 0.2) anonymized\_dataset = anonymize\_result.dataset |
| --- |

**End to End process of anonymizing data with pyarxaas:**

## 

## Exploring pyarxaas with the titanic dataset

The initial exploration of pyarxaas for my project was done on the popular titanic dataset. The result was done on the titanic dataset because the titanic dataset has a lot of columns like age, gender, number of siblings and parents travelling with the passenger, passenger ID, class, ticket and the name of the passenger. Since all of these are sensitive attributes and the dataset contains a variety of columns, it makes it excellent for the initial exploration. The dataset used here is downloaded from a popular platform for data science competitions known as **Kaggle.** You can simply download the two datasets called ‘testdata’ and ‘traindata’ in the form of zip files. After extracting them, this is what the data looks like:



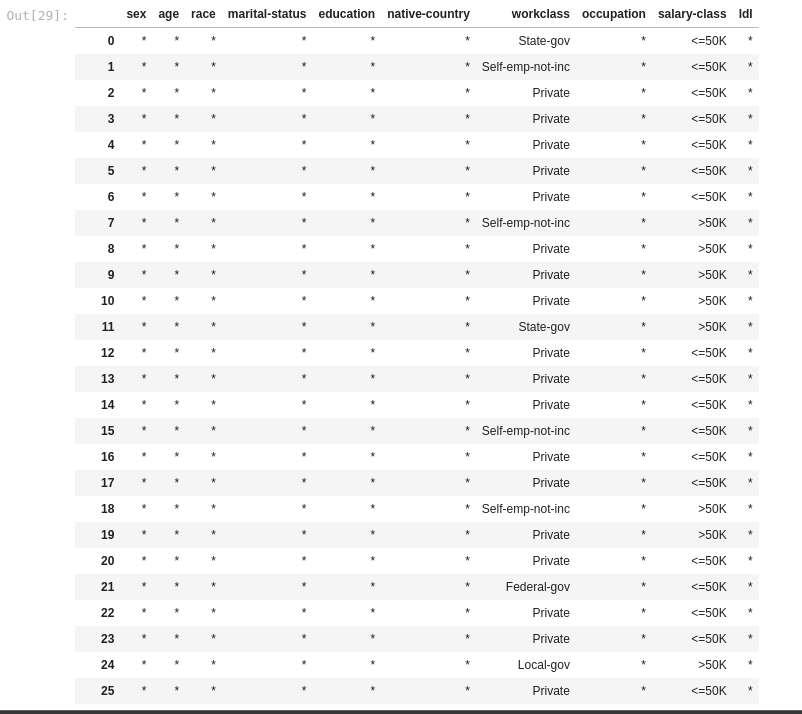
As shown in the figure, the dataset has name, gender, ticket, fare, passenger class and passenger ID as the columns. Proper understanding of these columns is necessary before the analysis.

**Analysing and anonymizing the data:**

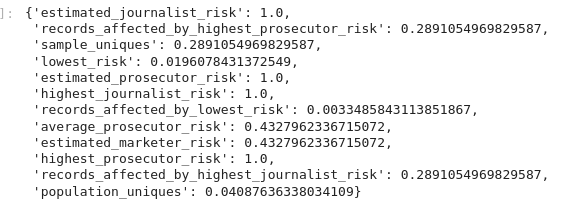
The given code reads the test file from the computer, generates hierarchies for each column used in the data and set those hierarchies while anonymizing the data for the same

| #The file given below takes the dataframe as a column input and sets the data. \ import numpy as np import pandas as pd import pyarxaas from pyarxaas import ARXaaS from pyarxaas.hierarchy import IntervalHierarchyBuilder, OrderHierarchyBuilder,RedactionHierarchyBuilder from pyarxaas import AttributeType from pyarxaas.privacy\_models import KAnonymity from pyarxaas import Dataset #running the arx instance locally from the object arxaas = ARXaaS("http://localhost:8080/") #loading the dataset data=pd.read\_csv("/home/shreshtha/Downloads/test.csv") #removing the NaN from the dataset data=data.dropna() #age and fare have float values and this will throw error while building hierarchies data['Age']=data['Age'].astype(int) data['Fare']=data['Fare'].astype(int) #data['Fare']=data('Fare').astype(int) def hierarchies(data):  #dropping the na's just in case  data=data.dropna()  #creating the dataset  dataset = Dataset.from\_pandas(data)  #setting the attributes as insesntive, sensitive and quasi identifying based on the values in the column  #here the fare sex and age and pclass and id are not identifying by themselves but combining them with other attributes might make them identifying  dataset.set\_attribute\_type(AttributeType.QUASIIDENTIFYING,'Fare','Sex','Age','Pclass','PassengerId','Ticket')  dataset.set\_attribute\_type(AttributeType.INSENSITIVE,'Embarked')  #setting the name as sensitive because seeing them on their own can disrupt the user privacy  dataset.set\_attribute\_type(AttributeType.SENSITIVE,'Name')  for column in dataset.columns:  #setting the hierarchies based on the column values  if column == 'Age':  #age can be grouped and shown generalised value as that still keeps the data statistically significant  hierarchy\_builder = IntervalHierarchyBuilder()  hierarchy\_builder.add\_interval(left=0, right=18, name='0-18')  hierarchy\_builder.add\_interval(left=19, right=30, name='19-30')  hierarchy\_builder.add\_interval(left=31, right=45, name='31-45')  hierarchy\_builder.add\_interval(left=46, right=60, name='46-60')  hierarchy = hierarchy\_builder.build()  dataset.set\_hierarchy(column, hierarchy) #this code would anonymize others based on generalised redaction hierarchy  elif column =='Ticket'or'PaseengerId':  builder= RedactionHierarchyBuilder()  ticket1=data['Ticket'].to\_list()  id=data['PassengerId'].to\_list()  ticket=arxaas.hierarchy(ticket,builder)  id1=arxaas.hierarchy(id,builder)  dataset.set\_hierarchy('Ticket',ticket)  dataset.set\_hierarchy('PassengerID',id)  elif column=='Fare' or 'Name':  redaction\_hierarchy=arxaas.hierarchy(builder,column)  fare\_list=data['Fare'].to\_list()  name=data['Name'].to\_list()  fare\_her=arxaas.hierarchy(fare\_list,redaction\_hierarchy)  name\_her=arxaas.hierarchy(name,redaction\_hierarchy)  dataset.set\_hierarchy('Fare',fare\_her)  dataset.set\_hierarchy("Name",name\_her)  return(dataset) #using the function to get the hierarchies out of the data dataset=hierarchies(data) #creating an arx-friendly dataset dataset=Dataset.from\_pandas((dataset)) #after the hierarchies are being set, calculating the risk profile of the dataset and ana;yzing it to get it ready for the anonymization risk\_profile = arxaas.risk\_profile(dataset) print(risk\_profile.re\_identification\_risk) print(risk\_profile.attacker\_success\_rate) print(risk\_profile.distribution\_of\_risk) #now that we have the risk profile, let us anonymize the data # importing the privacy\_models module from pyarxaas.privacy\_models import KAnonymity # creating a privacy\_models object kanon = KAnonymity(4) # specify the dataset as the first parameter, and privacy model list as the second parameter anonymize\_result = arxaas.anonymize(dataset, [kanon]) #looking at the risk profile after anonymization: # 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  # 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 metrics 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 anonymization metrics anon\_metrics = anonymize\_result.anonymization\_metrics |
| --- |

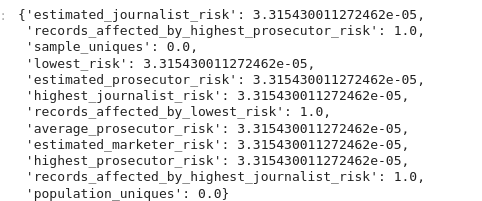
The dataset after anonymization:



Risk before anonymization:

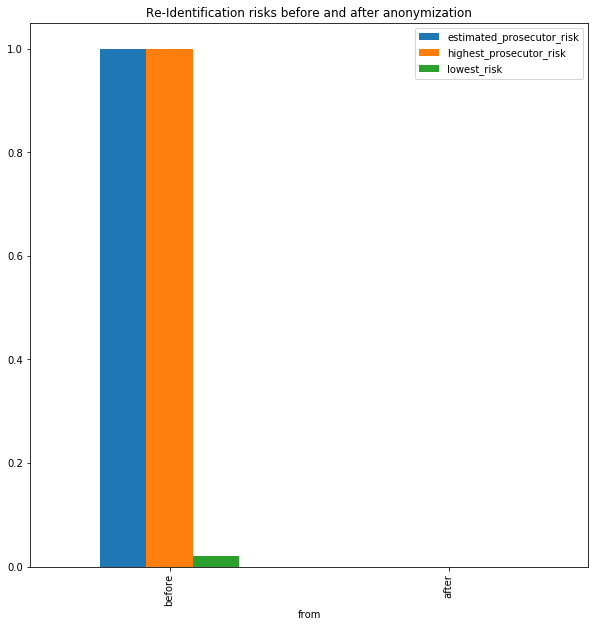


Risk after anonymization:



Thus, we can see that the risk before and after anonymization has reduced drastically while the data is statistically sound enough to generate the analysis

**The graph of accuracy after anonymization and before anonymization:**



## Creating the fake actual data

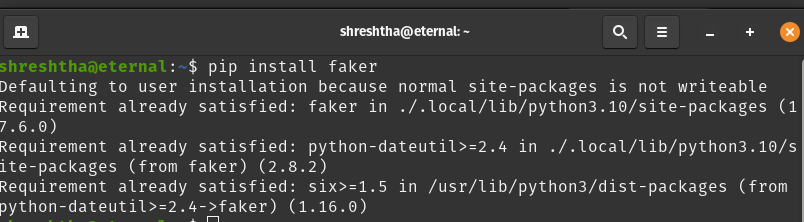
The data being used for further analysis from this point on will replicate the real life customer data very closely and will contain columns such as names, gender, email address and zip codes. The dataset is created using a python library called faker. Faker lets you create fake datasets in python with ease

### Installing faker:

Faker can be installed using the python library package manager called pip

| pip install Faker |
| --- |

Since i already have it installed on my local machine it does not show up, but if prompted with the question of do you want to install x mb of additional files? Please click yes



Thus, faker is installed and can now be used for anonymization. The following chapter will show how to create statistically significant fake random data using the faker library. This is a crucial part of the process as the faker library will help with testing of scripts and testing the enviournment

References:

<https://itslinuxfoss.com/install-use-pyenv-ubuntu/>

<https://pyarxaas.readthedocs.io/en/latest/user-guide/connect-to-arxaas.html#hierarchy-generation>

<https://docs.docker.com/engine/install/ubuntu/>