

Different Methods to Handle Missing Values (non-time series data)

Refresh

Table of contents (linked)

[1. Reasons for missing data](#)

[2. Types of Missing data](#)

[2.1 Missing Completely at Random \(MCAR\)](#)

[2.2 Missing at Random \(MAR\)](#)

[2.3 Missing Not at Random \(MNAR\)](#)

[General guidelines to correctly specify the missing data mechanism](#)

[3. Detecting the Missing values](#)

[3.1 Using describe\(\) function to get all non-missing values](#)

[3.2 Using isnull\(\) function to get all missing values](#)

[3.3 Using Missingno library](#)

[3.3.1 Barplot](#)

[3.3.2 Matrix plot](#)

[Visualizing the locations of the missing data](#)

[3.3.3 Heat Map](#)

[3.3.4 Dendrogram](#)

[4. Treatment of missing data](#)

[4.1. Drop/Deletion](#)

[\(4.1.1\) List-wise deletion](#)

[\(4.1.2\) Pairwise deletion \(available-case analysis\)](#)

[\(4.1.3\) drop columns with at least one missing value](#)

[4.2. Imputation/Filling](#)

[\(4.2.1\) Fill missing values with specific guess \(arbitrary\) values](#)

[\(4.2.2\) Fill missing values with the mean/median/mode value\(s\)](#)

[\(4.2.3\) Fill missing values with the mean/median/mode value within groups](#)

[\(4.2.4\) End of distribution imputation](#)

[\(4.2.5\) Hot deck method](#)

[\(4.2.6\)fillna\(\) methods](#)

[\(4.2.6.1\) Impute missing values with ffill](#)

[\(4.2.6.2\) Impute missing values with bfill](#)

[\(4.3\) Imputation using Interpolation](#)

[\(4.3.1\) Linear interpolation](#)

[\(4.3.2\) Polynomial method](#)

[\(4.2.3\) cubic spline method](#)

[4.4. Imputation using ML methods](#)

[4.5. Full Analysis](#)

[5. Follow up techniques](#)

[REFERENCES](#)

Table of Contents:

1. Reasons for Missing data
2. Types of Missing data
 - MCAR
 - MAR
 - MNAR
3. Detection of Missing data
 - Numerically
 - Visually
4. Treatment for Missing data
 - Drop / Delete
 - Listwise (complete case) deletion
 - Pairwise (available case) deletion
 - Column deletion
 - Imputation
 - Simple
 - Fill values
 - Arbitrary
 - Mean/Median/Mode
 - ffill
 - bfill
 - Interpolate
 - Linear
 - Polynomial
 - Cubic spline
 - Advanced
 - ML methods

- Full Analysis

5. Effect of treatment/Follow up

Reasons for missing data

1. Figure out why the value is missing.

Is this value missing because

- it doesn't exist?

it doesn't make sense to try and guess what it might be. These values you probably do want to keep as NaN.

- it wasn't recorded
 - not collected/shared
 - deleted by mistake

you can try to guess what it might have been based on the other values in that column and row. This is called "imputation"

✓ Types of Missing data

Rubin (1976) classified missing data problems into three categories. In his theory every data point has some likelihood of being missing.

- The process that governs these probabilities is called the **missing data mechanism** or response mechanism.
- The model for the process is called the **missing data model** or response model.

Rubin's distinction is important for understanding why some methods will not work. His theory lays down the conditions under which a missing data method can provide valid statistical inferences.

1. Missing Completely at Random (MCAR)

- the probability of being missing is the same for all cases
- causes of the missing data are unrelated to the data.
 - Missingness is independent of data
 - The missing data are just a random subset of the data.
 - The missing values on a given variable are not associated with other variables or with the variable itself.
 - In other words, there is no particular reason for the missing values.
- examples:
 - a weighing scale that ran out of batteries.
 - a random sample of a population,
 - each member has the same chance of being included in the sample.
 - The (unobserved) data of members in the population that were not included in the sample are MCAR.
- When data are MCAR, the analysis performed on the data is unbiased;
- MCAR is often unrealistic for the data at hand, that is, data are rarely MCAR.

2. Missing at Random (MAR)

- the *probability of being missing* is the same only within groups defined by the observed data.
- MAR occurs when the missingness is not random, but where missingness can be fully accounted for by variables where there is complete information.
 - Missingness depends on the observed data
- MAR is a much broader class than MCAR.
- MAR is more general and more realistic than MCAR.
- example:
 - when placed on a soft surface, a weighing scale may produce more missing values than when placed on a hard surface.
 - Such data are thus not MCAR.
 - If, however, we know surface type and if we can assume MCAR within the type of surface, then the data are MAR.
 - a sample from a population, where the probability of selection depends on some known property.
- Modern missing data methods generally start from the MAR assumption.
 - MAR is an assumption that is impossible to verify statistically
- MAR will provide asymptotically unbiased estimates if the parameter is estimated with Full Information Maximum Likelihood
- **Techniques to use**

Imputation:

- a technique that involves replacing missing data with estimated values based on the observed data.
- There are several methods for imputation, such as
 - mean imputation,
 - hot-deck imputation, and
 - regression imputation.
- Imputation can help to reduce the bias and increase the power of statistical analyses.
- Imputation requires assumptions about the underlying data distribution and can introduce additional variability in the results.

Maximum Likelihood Estimation (MLE):

- a method that estimates the parameters of a statistical model by maximizing the likelihood of the observed data.
- MLE can be used to estimate the missing data and fit a model to the observed data simultaneously.
- MLE can provide unbiased and efficient estimates of the parameters of interest if the missing data mechanism is correctly specified.

Weighting:

- a technique that involves assigning weights to observations based on the probability of being missing.
- Weighting can help to account for the non-randomness of the missing data and reduce the bias in the estimates.

- Example: Inverse Probability Weighting (IPW), which weights the observed data by the inverse of the probability of being observed.

Model-based approaches:

- involve fitting models that account for missing data.
- Example: Expectation-Maximization (EM) algorithm, which iteratively estimates the missing data and fits a model to the observed data.
- Model-based approaches can be computationally intensive but can provide more accurate estimates of the parameters of interest.

- **Avoid the following techniques**

Complete case analysis:

- a technique that involves excluding observations with missing data from the analysis.
- can lead to biased results if the missing data is related to the outcome or predictor variables.

Pairwise deletion:

- a technique that involves using only the available data for each pair of variables in the analysis.
- can lead to biased results if the missing data is related to the outcome or predictor variables.

Listwise deletion:

- a technique that involves excluding all observations that have any missing data from the analysis.
- can lead to a loss of statistical power and biased estimates if the missing data is related to the outcome or predictor variables.

Mean imputation:

- a technique that involves replacing missing data with the mean of the observed data for that variable.
- can lead to biased estimates of the parameters of interest if the missing data is related to the outcome or predictor variables.

✓ 3. Missing Not at Random (MNAR)

- the probability of being missing varies for reasons that are unknown to us.
 - Missingness depends on unobserved data or the value of the missing data itself.
 - neither MCAR nor MAR holds,
 - also, called NMAR (not missing at random)
- example:
 - the weighing scale mechanism may wear out over time, producing more missing data as time progresses, but we may fail to note this.
 - If the heavier objects are measured later in time, then we obtain a distribution of the measurements that will be distorted.
 - Includes the possibility that the scale produces more missing values for the heavier objects, a situation that might be difficult to recognize and handle.

- MNAR in public opinion research occurs if those with weaker opinions respond less often.
- MNAR is the most complex case.
- Strategies to handle MNAR are to
 - find more data about the causes for the missingness, or
 - perform what-if analyses to see how sensitive the results are under various scenarios.

```
# display all outputs in a cell (https://rb.gy/lbz7hn)
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

```
# Setting your working directory
!pwd
!ls
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks/')

```

```
↗ /content
drive sample_data
```

```
# import required packages
import numpy as np
import pandas as pd
```

```
# import dataset
df = pd.read_csv('./data/aug_train1.csv') # ,parse_dates=['Date'] option for date columns
```

```
# No of rows and columns in the dataset
df.shape
```

```
# Looking at first 5 rows
#df.head()
```

```
↗ (19158, 14)
```

```
# Looking at a random sample of 5 rows
#df.sample(5)
```

```
# Better understanding of the dataset
df.info()
```

```
↗ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 19158 entries, 0 to 19157
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   enrollee_id            19158 non-null  int64
1   city                   19158 non-null  object
2   city_development_index 19130 non-null  float64
3   gender                 14650 non-null  object
4   relevent_experience     19158 non-null  object
5   enrolled_university    18772 non-null  object
6   education_level        18698 non-null  object
7   major_discipline       16345 non-null  object
8   experience              19093 non-null  object
```

```

9  company_size      13220 non-null object
10 company_type      13018 non-null object
11 last_new_job       18735 non-null object
12 training_hours     19158 non-null int64
13 target             19158 non-null int64
dtypes: float64(1), int64(3), object(10)
memory usage: 2.0+ MB

```

```

# display the variable names (column names)
df.columns

```

```

↵ Index(['enrollee_id', 'city', 'city_development_index', 'gender',
        'relevent_experience', 'enrolled_university', 'education_level',
        'major_discipline', 'experience', 'company_size', 'company_type',
        'last_new_job', 'training_hours', 'target'],
        dtype='object')

```

```

# A quick statistical view



```

```

df.describe()

```

```
↵
```

	enrollee_id	city_development_index	training_hours	target	
count	19158.000000	19130.000000	19158.000000	19158.000000	
mean	16875.358179	0.828840	65.366896	0.249348	
std	9616.292592	0.123391	60.058462	0.432647	
min	1.000000	0.448000	1.000000	0.000000	
25%	8554.250000	0.740000	23.000000	0.000000	
50%	16982.500000	0.903000	47.000000	0.000000	
75%	25169.750000	0.920000	88.000000	0.000000	
max	33380.000000	0.949000	336.000000	1.000000	

```

# Counting all the unique values the variable "gender" takes

```

```

df['gender'].value_counts()
print('\nTotal number of non missing values :', df['gender'].value_counts().sum())

```

```
↵
```

	count
gender	
Male	13221
Female	1238
Other	191

```

dtype: int64

```

```

Total number of non missing values : 14650

```

```

df['city_development_index'].value_counts()

```



city_development_index	count
0.920	5195
0.624	2700
0.910	1528
0.926	1335
0.698	679
...	...
0.649	4
0.807	4
0.781	3
0.625	3
0.664	1

93 rows × 1 columns

dtype: int64

General guidelines to correctly specify the **missing data mechanism**

Understand the missing data mechanism: To correctly specify the missing data mechanism, you need to have a good understanding of the data and the reasons why data are missing. This understanding can come from domain knowledge, data collection procedures, and data quality assessments.

Evaluate the patterns of missingness: Look for patterns in the missingness of the data. Are there any subgroups of the population that are more likely to have missing data? Are there any variables that are associated with the missingness of the data?

Use statistical methods: There are several statistical methods that can help you identify the missing data mechanism. For example, you can use graphical methods such as the missingness plot or the pattern of missingness by variable plot. You can also use statistical tests such as the Little's MCAR test or the likelihood ratio test for MAR.

Consider sensitivity analysis: It's important to assess the robustness of your analysis to different missing data mechanisms. You can do this by performing sensitivity analysis using different missing data assumptions and comparing the results.

Report the missing data mechanism: Once you have identified the missing data mechanism, report it clearly in your analysis. This will help others to understand your analysis and to assess the validity of your inferences.

✓ 3. Detecting the Missing values

1. Numerically using

- describe()
- isnull()

2. Graphically using missingno package

3.1.1 Using describe() function to get all non-missing values

The count property directly gives the count of non-NaN values in each column.

```
df.describe(include='all').loc['count']
```



	count
enrollee_id	19158.0
city	19158
city_development_index	19130.0
gender	14650
relevent_experience	19158
enrolled_university	18772
education_level	18698
major_discipline	16345
experience	19093
company_size	13220
company_type	13018
last_new_job	18735
training_hours	19158.0
target	19158.0

dtype: object

```
#counting values only for numeric columns
#df.numeric.describe() #not working
df.describe(include=[np.number]).loc['count']
```



	count
enrollee_id	19158.0
city_development_index	19130.0
training_hours	19158.0
target	19158.0

dtype: float64

```
# Including non-numerical columns
df.describe(exclude=[np.number]).loc['count']
```



	count
city	19158
gender	14650
relevent_experience	19158
enrolled_university	18772
education_level	18698
major_discipline	16345
experience	19093
company_size	13220
company_type	13018
last_new_job	18735

df type: object

✓ 3.1.2. Using isnull() function to get all missing values

notnull may be alternatively used to get all non-missing values

```
# counting the missing values under each column
```

```
df_mis=df.isnull().sum()
df_mis
```



	0
enrollee_id	0
city	0
city_development_index	28
gender	4508
relevent_experience	0
enrolled_university	386
education_level	460
major_discipline	2813
experience	65
company_size	5938
company_type	6140
last_new_job	423
training_hours	0
target	0

dtype: int64

```
df.isnull().mean()*100
df_mis_pcent=100*df.isnull().sum()/len(df)
print('\n\nMissing percentage:\n\n', df_mis_pcent)
```



0

enrollee_id	0.000000
city	0.000000
city_development_index	0.146153
gender	23.530640
relevent_experience	0.000000
enrolled_university	2.014824
education_level	2.401086
major_discipline	14.683161
experience	0.339284
company_size	30.994885
company_type	32.049274
last_new_job	2.207955
training_hours	0.000000
target	0.000000

dtype: float64

Missing percentage:

enrollee_id	0.000000
city	0.000000
city_development_index	0.146153
gender	23.530640
relevent_experience	0.000000
enrolled_university	2.014824
education_level	2.401086
major_discipline	14.683161
experience	0.339284
company_size	30.994885
company_type	32.049274
last_new_job	2.207955
training_hours	0.000000
target	0.000000

dtype: float64

```
# Make a table of missing values with %
df_mis_table=pd.concat([df_mis,df_mis_pcent],axis=1)
df_mis_table
```



	0	1	
enrollee_id	0	0.000000	
city	0	0.000000	
city_development_index	28	0.146153	
gender	4508	23.530640	
relevent_experience	0	0.000000	
enrolled_university	386	2.014824	
education_level	460	2.401086	
major_discipline	2813	14.683161	
experience	65	0.339284	
company_size	5938	30.994885	
company_type	6140	32.049274	
last_new_job	423	2.207955	
training_hours	0	0.000000	
target	0	0.000000	

Next steps:

[Generate code with df_mis_table](#)

[View recommended plots](#)

[New interactive sheet](#)

```
# Rename the columns
df_mis_table.rename(columns = {0 : 'Missing Values', 1 : '% of Total Values'})
```



	Missing Values	% of Total Values	
enrollee_id	0	0.000000	
city	0	0.000000	
city_development_index	28	0.146153	
gender	4508	23.530640	
relevent_experience	0	0.000000	
enrolled_university	386	2.014824	
education_level	460	2.401086	
major_discipline	2813	14.683161	
experience	65	0.339284	
company_size	5938	30.994885	
company_type	6140	32.049274	
last_new_job	423	2.207955	
training_hours	0	0.000000	
target	0	0.000000	

```
# checking the null values under a specific row
```

```
df.loc[10, :].isnull().sum()
```

```
↔ 5
```

```
# checking all null values in the dataset
```

```
df.isnull().sum().sum()
```

```
↔ 20761
```

```
## percent of data that is missing
```

```
df.isnull().sum().sum()/np.product(df.shape)*100
```

```
↔ 7.7405186941673
```

✓ 3.2. Using Missingno library

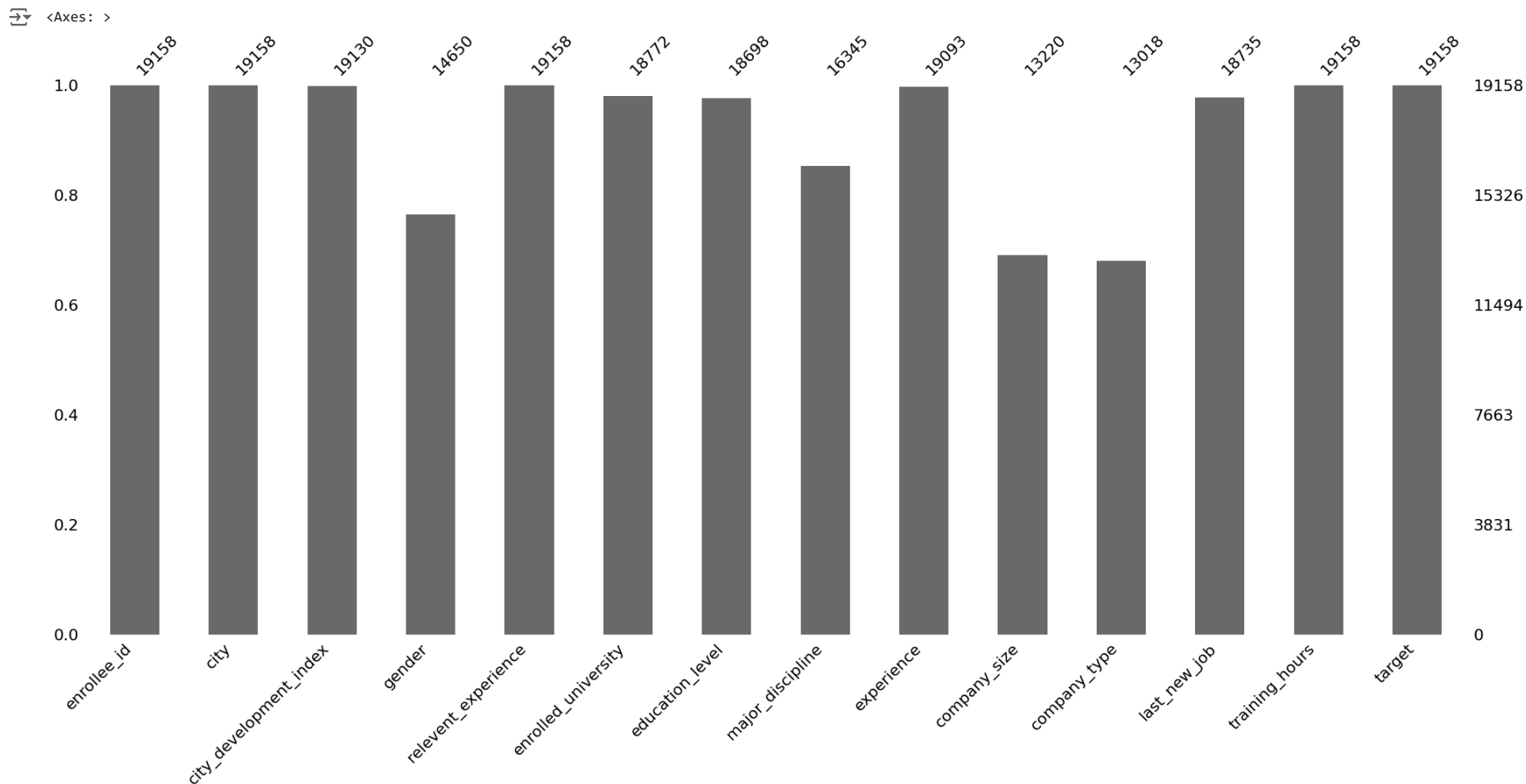
[Missingno library](#) is helpful for graphical analysis of missing values. There are four tools to do that.

1. barplot
2. matrix plot
3. heatmap
4. dendrogram

✓ 3.2.1 Barplot

```
import missingno as msno
```

```
msno.bar(df)
```



On the left side of the plot, the y-axis scale ranges from 0.0 to 1.0, where

- 1.0 represents 100% data completeness.
- If the bar is less than this, it indicates that we have missing values within that column.

On the right side of the plot, the scale is measured in **index values**.

- top right represents the *maximum number of rows* within the dataframe.

On the **top** of the plot, there are a series of numbers that represent the **total count of the non-null values** within that column.

✓ 3.2.2 Matrix plot

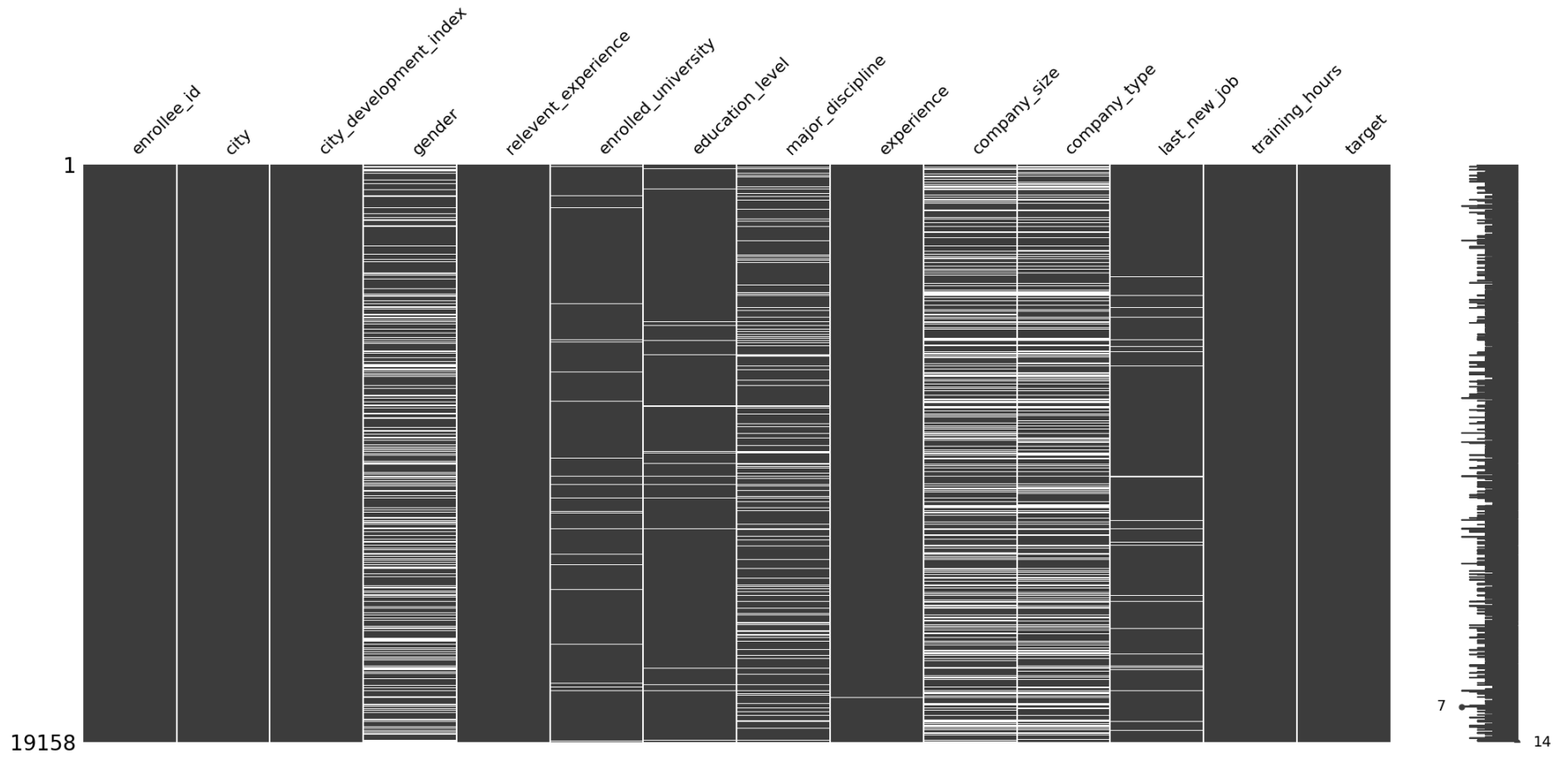
The matrix plot is a great tool if you are working with depth-related data or time-series data. It provides a colour fill for each column. When data is present, the plot is shaded in grey (or your colour of choice), and when it is absent the plot is displayed in white.

Visualizing the locations of the missing data

1. The plot appears white wherever there are missing values.
2. The **sparkline** on the right gives an idea of the general *shape of the completeness* of the data and points out the row with the minimum nullities and the total number of columns in a given dataset, at the bottom.

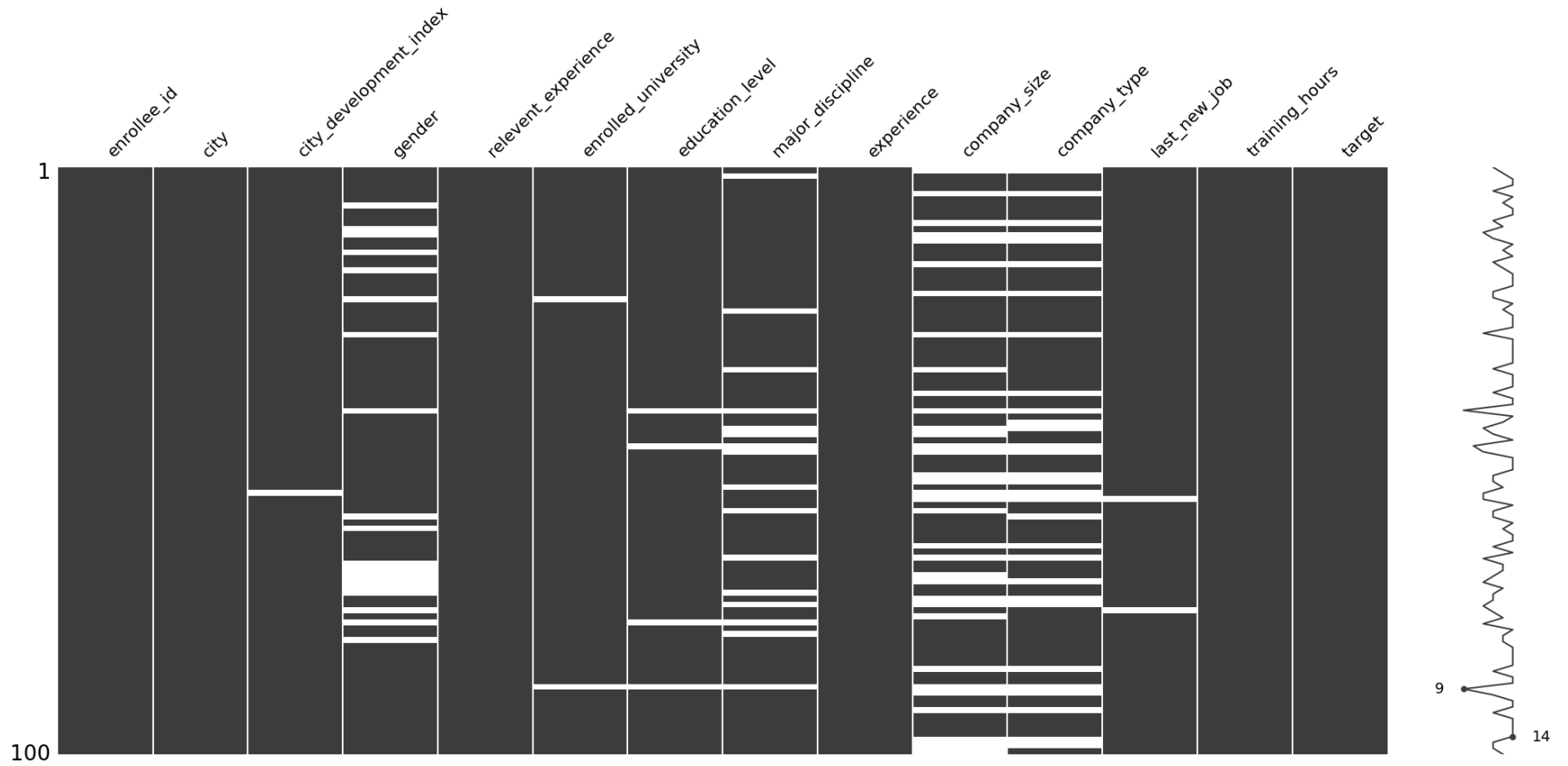
```
msno.matrix(df)
```


<Axes: >



```
msno.matrix(df.sample(100))
```

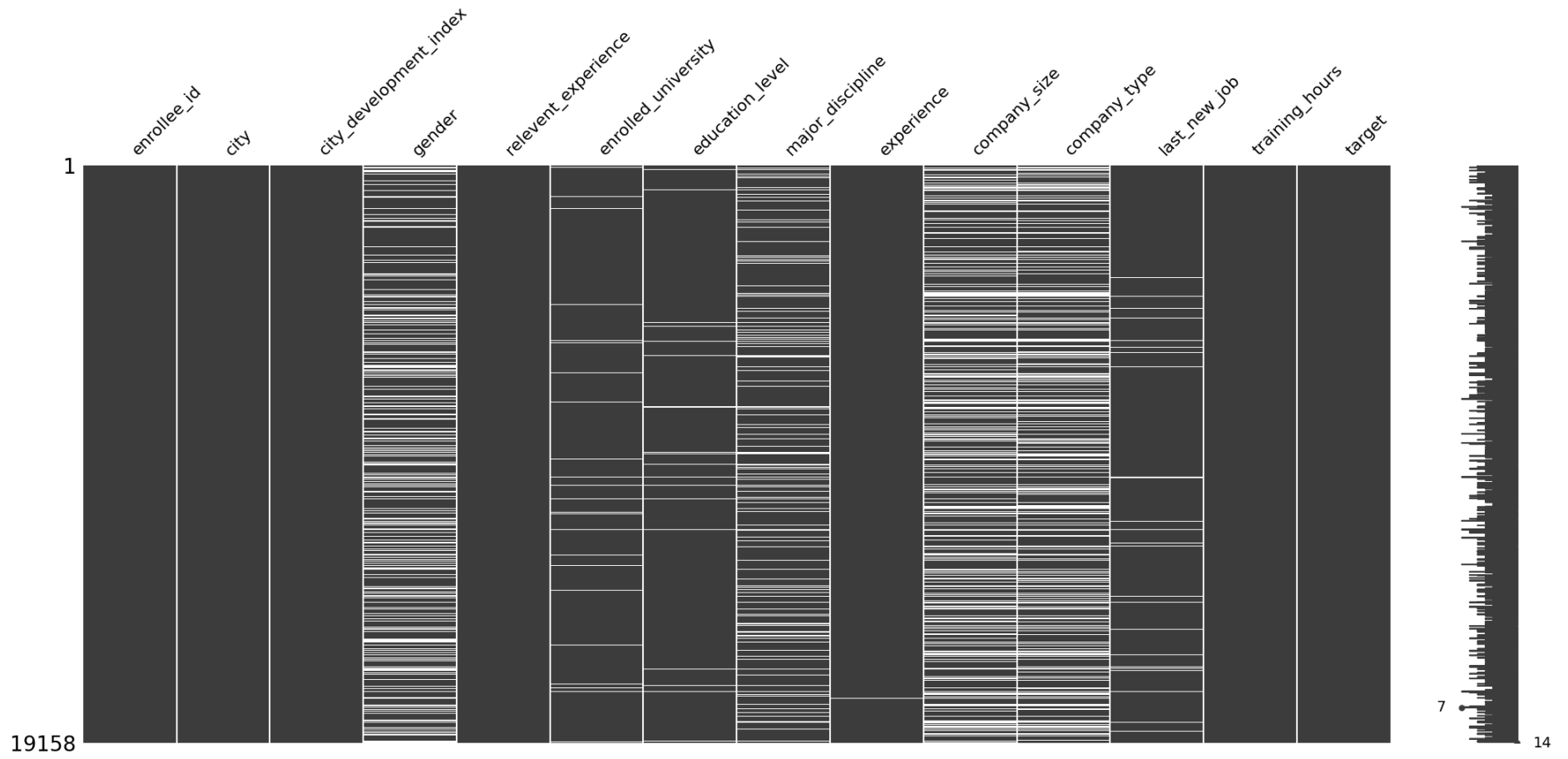
<Axes: >



3. When a row has a value in each column, the line will be at the maximum right position. As missing values start to increase within that row the line will move towards the left.

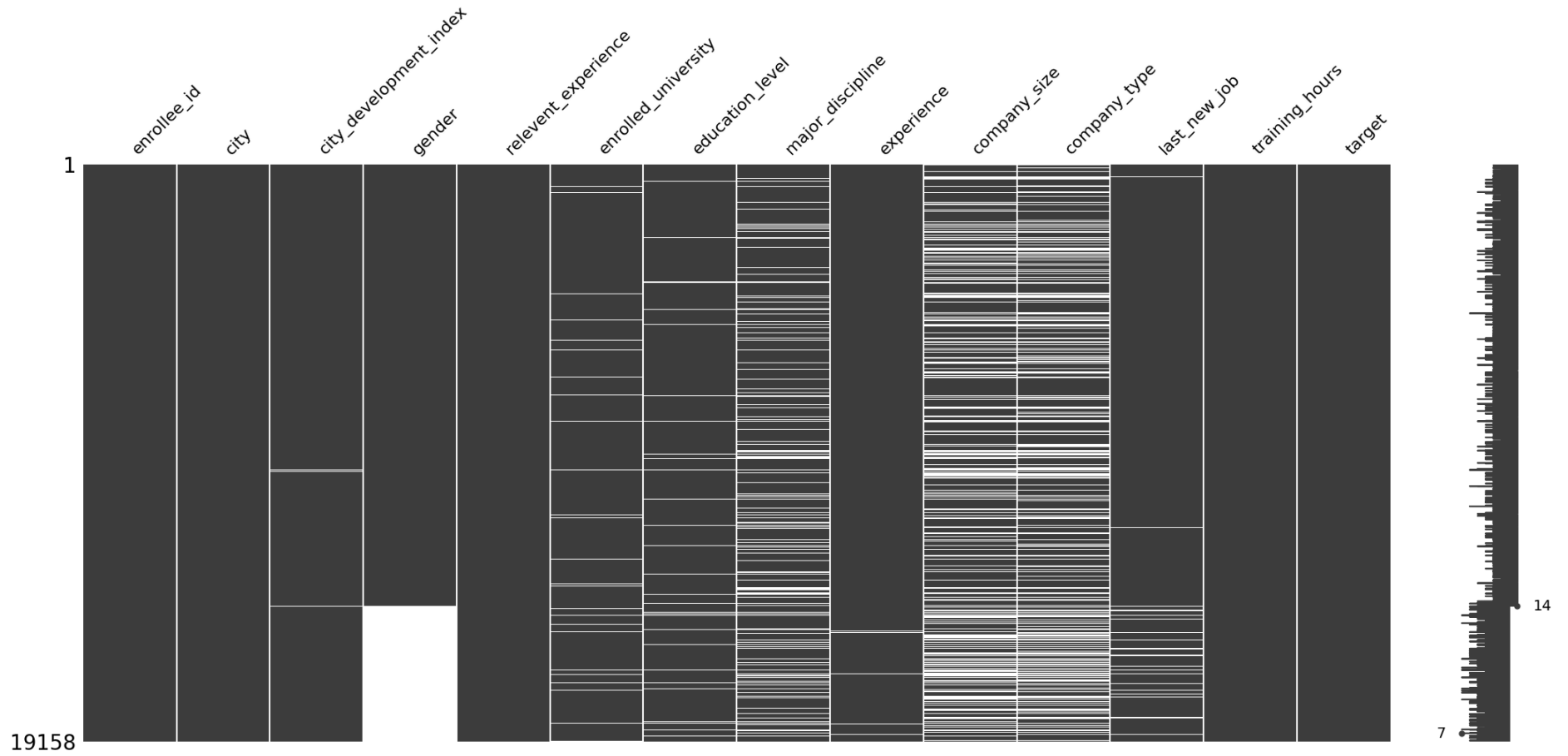
```
# Finding reason for missing data using matrix plot
msno.matrix(df)
```

<Axes: >



```
#sorted by Gender
sorted = df.sort_values('gender')
msno.matrix(sorted)
```

<Axes: >




3.2.3 Heat Map

The heatmap is used to identify correlations of the nullity between each of the different columns. In other words, it can be used to identify if there is a relationship in the presence of null values between each of the columns. The heatmap approach is more *suitable for smaller datasets*.

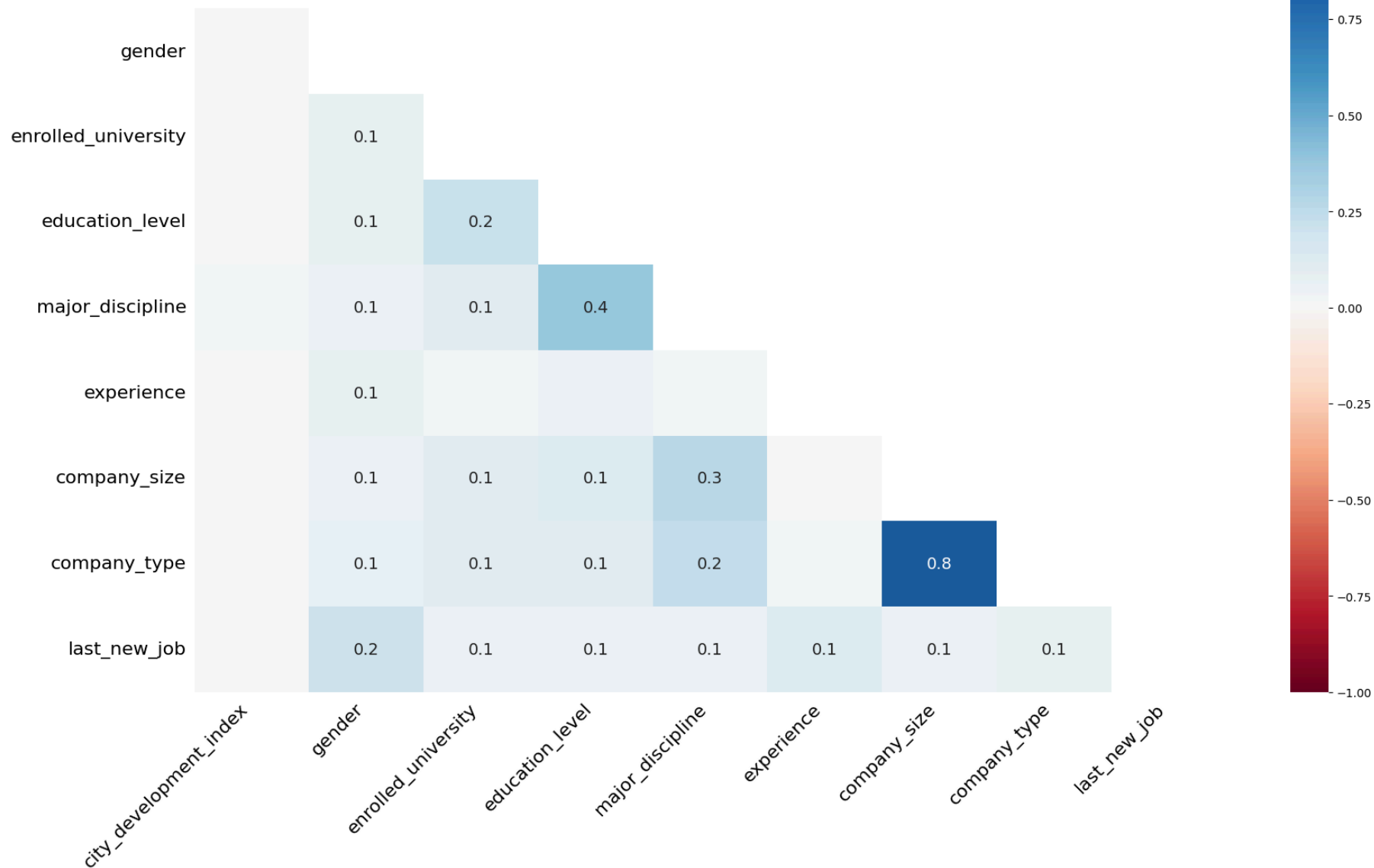
- Values close to $+1$ indicate that the presence of null values in one column is correlated with the presence of null values in another column.

- Values close to -1 indicate that the presence of null values in one column is anti-correlated with the presence of null values in another column. In other words, when null values are present in one column, there are data values present in the other column, and vice versa.
- Values close to 0 , indicate there is little to no relationship between the presence of null values in one column compared to another.
- There are a number of values that show as < -1 . This indicates that the correlation is very close to being 100% negative.

```
## Finding reason for missing data using a Heatmap  
msno.heatmap(df)
```

 <Axes: >

city_development_index



It is clear that there is no relation between the missingness in Gender and rest variables. The heatmap function shows that there are no strong correlations between missing values of different variables.

Low correlations further indicate that the data are MAR.

✓ 3.2.4 Dendrogram

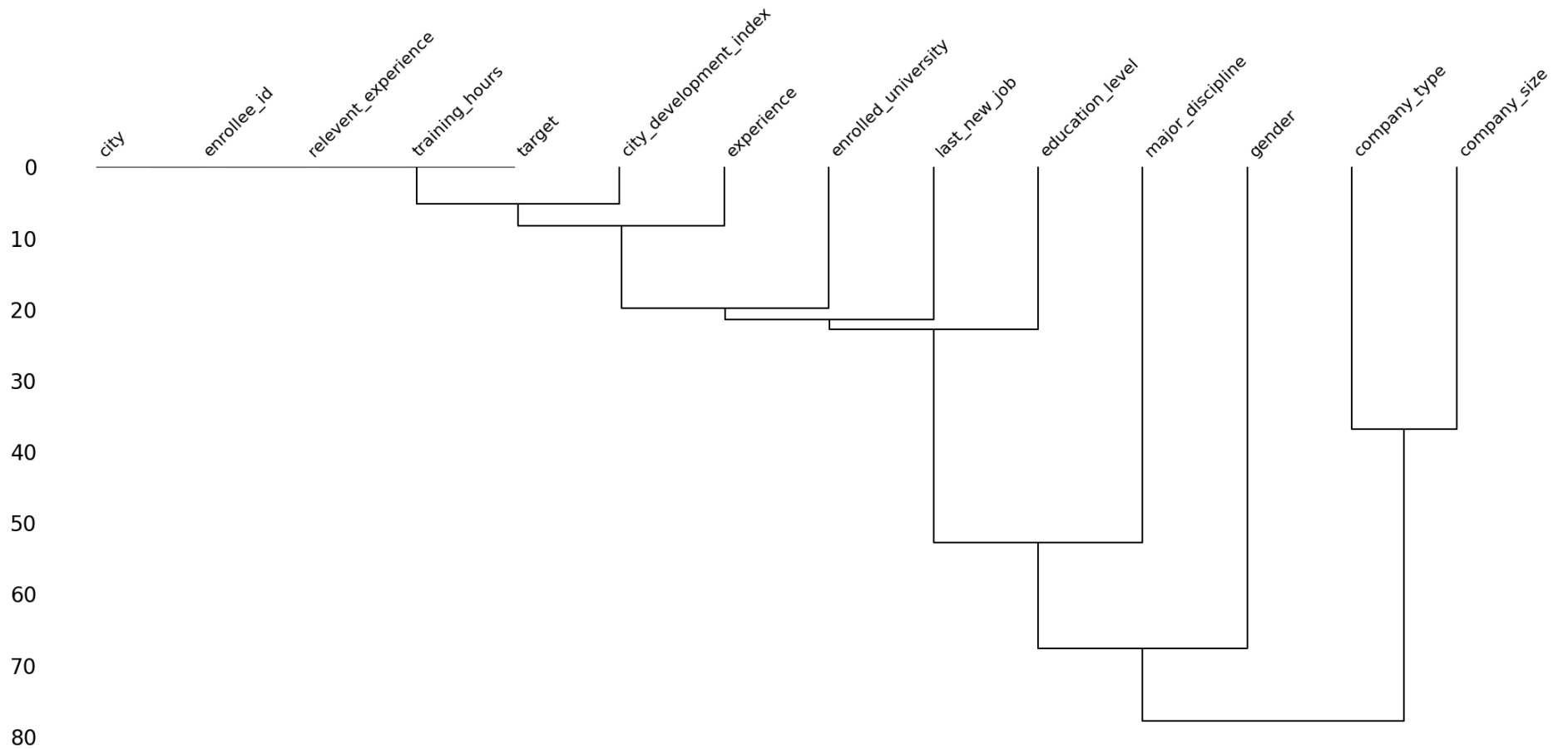
The dendrogram plot provides a tree-like graph generated through hierarchical clustering and groups together columns that have strong correlations in nullity.

If a number of columns are grouped together at level zero, then the presence of nulls in one of those columns is directly related to the presence or absence of nulls in the others columns.

The more separated the columns in the tree, the less likely the null values can be correlated between the columns.

```
## Finding reason for missing data using Dendrogram  
msno.dendrogram(df)
```

<Axes: >



Observations

1. city, enrollee_id, relevent_experience, training_hours and target are all grouped together at zero indicating that they are complete.
2. city_development_index is in the same larger branch suggesting that some of the missing values present within that column can be correlated with these five columns.

✓ Little's MCAR test

It is a statistical test used to assess whether the missing data in a dataset are missing completely at random or if there is a systematic pattern to the missingness. The test is named after *Donald R. Little*, who introduced it.

- The MCAR assumption is an important assumption in statistical analyses, particularly in techniques like multiple imputation.
- MCAR means that the probability of missingness is unrelated to the observed or unobserved data, and there are no systematic differences between missing and observed data.

How Little's MCAR test works:

Null Hypothesis (H0): The missing data are completely at random. Alternative Hypothesis (H1): The missing data are not completely at random; there is some systematic pattern or relationship with the observed data.

The test is based on a `chi-squared` statistic, The p-value associated with this statistic is used to assess the significance of the test.

- If the p-value is below a chosen significance level α , say 0.05, you may reject the null hypothesis and conclude that the missing data are not MCAR.

Keep in mind that the MCAR test has limitations, and it might not be sensitive to certain types of non-random missingness. If data are not missing completely at random, other imputation methods or adjustments to the analysis might be necessary.

Python implementation

- Perform Little's MCAR test using the `missingpy` library

```
#!/pip install statsmodels
#!/pip install scikit-learn==0.22.1
#!/pip install -U missingpy
```

```
🔄 Collecting scikit-learn==0.22.1
  Downloading scikit-learn-0.22.1.tar.gz (6.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 43.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==0.22.1) (1.26.4)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==0.22.1) (1.13.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==0.22.1) (1.4.2)
Building wheels for collected packages: scikit-learn
  error: subprocess-exited-with-error

  × python setup.py bdist_wheel did not run successfully.
  | exit code: 1
  | See above for output.

  note: This error originates from a subprocess, and is likely not a problem with pip.
Building wheel for scikit-learn (setup.py) ... error
ERROR: Failed building wheel for scikit-learn
Running setup.py clean for scikit-learn
Failed to build scikit-learn
ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (scikit-learn)
```

```
from missingpy import MissForest
import pandas as pd
```

```
# Sample data with missing values
data = {
```

```

'A': [1, 2, 3, 4, 5, 6],
'B': [10, 20, 30, None, 50, 60],
'C': [100, 200, 300, 400, 500, None]
}

```

```
df = pd.DataFrame(data)
```

```

# Fit MissForest model to impute missing values
imputer = MissForest()
imputed_data = imputer.fit_transform(df)

```

```

# Perform Little's MCAR test
mcar_test_result = imputer.is_missing_mcar()

```

```

# Print the result of Little's MCAR test
print("Little's MCAR Test Result:")
print(mcar_test_result)

```



ModuleNotFoundError Traceback (most recent call last)

[<ipython-input-48-f15d8243f7c5>](#) in <cell line: 1>()
 ----> 1 from missingpy import MissForest

```

2 import pandas as pd
3
4 # Sample data with missing values
5 data = {

```



1 frames

[/usr/local/lib/python3.10/dist-packages/missingpy/knnimpute.py](#) in <module>

```

11 from sklearn.utils.validation import check_is_fitted
12 from sklearn.utils.validation import FLOAT_DTYPES
---> 13 from sklearn.neighbors.base import _check_weights
14 from sklearn.neighbors.base import _get_weights
15

```

ModuleNotFoundError: No module named 'sklearn.neighbors.base'

 NOTE: If your import is failing due to a missing package, you can manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the "Open Examples" button below.

OPEN EXAMPLES

Next steps:

[Explain error](#)

```
from statsmodels.stats.missing import test_mcar
```



```
-----  
ModuleNotFoundError                                Traceback (most recent call last)  
<ipython-input-43-76b575149bda> in <cell line: 1>()  
----> 1 from statsmodels.stats.missing import test_mcar
```

ModuleNotFoundError: No module named 'statsmodels.stats.missing'

NOTE: If your import is failing due to a missing package, you can manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the "Open Examples" button below.

OPEN EXAMPLES

Next steps: [Explain error](#)

- missingpy library is used to perform imputation using the MissForest algorithm.
- is_missing_mcar() function is used to perform Little's MCAR test.

Remember that imputation is just one step in dealing with missing data. Depending on your analysis, you might also consider other imputation methods or handling missing data in a way that is appropriate for your specific context.

Manual Implementation of Little's MCAR Test

```

import numpy as np
import pandas as pd
from scipy.stats import chi2

def little_mcar_test(data):
    """
    Perform Little's MCAR test on a pandas dataframe.

    Parameters:
    data (DataFrame): The dataset to test for MCAR.

    Returns:
    dict: A dictionary containing the chi-square statistic, degrees of freedom, and p-value.
    """
    n = len(data)
    groups = []

    for col in data.columns:
        mask = data[col].isnull()
        if mask.any():
            groups.append(mask.astype(int).values.reshape(-1, 1))

    if len(groups) == 0:
        raise ValueError("No missing data found.")

    r = np.concatenate(groups, axis=1)
    group_stats = r.T @ r
    m = len(groups)
    df = (n - 1) * m
    chi2_stat = group_stats.trace()

    p_value = chi2.sf(chi2_stat, df)

    return {"chi2_stat": chi2_stat, "degrees_of_freedom": df, "p_value": p_value}

# Example usage with a DataFrame `df`:
result = little_mcar_test(df)
print(f"Chi-square statistic: {result['chi2_stat']}")
print(f"Degrees of freedom: {result['degrees_of_freedom']}")
print(f"P-value: {result['p_value']}")

```

```

🔗 Chi-square statistic: 20761
Degrees of freedom: 172413
P-value: 1.0

```

Interpreting the Results

- Chi-square statistic: This measures how much the observed missing data pattern deviates from what would be expected if the data were MCAR.
- Degrees of freedom: The number of independent pieces of information used to calculate the chi-square statistic.
- P-value: If this value is below a certain threshold (commonly 0.05), it suggests that the data are not MCAR.

4. Treatment of missing data

- Missing data reduces the representativeness of the sample and can therefore distort inferences about the population.
- There are three main approaches to handle missing data:
 - (1) Drop/delete—where samples with invalid data are discarded from further analysis
 - (2) Imputation—where values are filled in the place of missing data,
 - (3) Analysis—by directly applying methods unaffected by the missing values.

✓ 4.1. Drop/Delete Missing Values

- If you're in a hurry or don't have a reason to figure out why your values are missing, one option you have is to just remove any rows or columns that contain missing values.
- this approach is not recommended for important projects
- It's usually worth it to take the time to go through your dataset look at all the columns with missing values one-by-one

The first choice is to remove the missing values using `.drop` method of Pandas.

(4.1.1) List-wise deletion

- Complete case analysis (listwise deletion) is the default way of handling incomplete data in many statistical packages
- eliminates all cases with one or more missing values on the analysis variables.
- If the data are MCAR, listwise deletion produces
 - unbiased estimates of means, variances and regression weights.
 - standard errors and significance levels that are correct for the reduced subset of data, but that are often larger relative to all available data.
- If the data are not MCAR, listwise deletion can severely bias estimates of means, regression coefficients and correlations.
 - the bias in the estimated mean increases with the
 - difference between means of the observed and missing cases, and
 - proportion of the missing data.
 - bias of listwise deletion under MAR and MNAR is simulated by [Schafer and Graham](#).
- major advantage of complete case analysis is convenience.
- can provide better estimates than even the most sophisticated procedures.
- Very useful often in the context of regression analysis
- A disadvantage of listwise deletion is that it is potentially wasteful.
 - It is not uncommon in real life applications that more than half of the original sample is lost, especially if the number of variables is large.
 - a smaller subsample could seriously degrade the ability to detect the effects of interest.
 - can lead to nonsensical subsamples.
 - can introduce inconsistencies in reporting.

```
df.isna().sum()
```

```
df['city_development_index'].mean()
```

✓ (4.1.2) Pairwise deletion (available-case analysis)

- attempts to remedy the data loss problem of listwise deletion.
- Pairwise deletion is used when values are missing completely at random i.e MCAR.
- During Pairwise deletion, only the missing values are deleted.
- All operations in pandas like mean,sum etc intrinsically skip missing values.
- The method calculates the
 - means and (co)variances on all observed data.
 - the mean of variable X is based on all cases with observed data on X,
 - the mean of variable Y uses all cases with observed Y-values, and so on.
 - For the correlation and covariance, all data are taken on which both X and Y have non-missing scores.
 - Subsequently, the matrix of summary statistics are fed into a program for regression analysis, factor analysis or other modeling procedures.
- The method is simple, uses all available information and produces consistent estimates of mean, correlations and covariances under MCAR
- The estimates can be biased if the data are not MCAR.
- Some columns having no missing values are also affected by this.
- A major disadvantage of Listwise deletion is that a major chunk of data and hence a lot of information is lost.
- use it only when the number of missing values is very small.

```
df.dropna(subset=['gender'],how='any',inplace=True)
```

```
df['gender'].isnull().sum()
```

```
df.isna().sum()
```

✓ (4.1.3) drop columns with at least one missing value

- Columns that contain a lot of missing values, say more than 80%, and the feature is not significant, you can delete that feature.
- However, it is not a good methodology to remove a variable by deleting data.

```
df.dropna(axis=1)
```

✓ 4.2. Imputation/Filling Missing Values

- Specify what we want the NaN values to be replaced with.
- Check your variable type before filling
 - Categorical case- replace all the NaN values with appropriate value of the labels/factors
 - Numerical Case- replace all the NaN values with appropriate values like 0, mean, and others.
- There are two types

- Univariate : Using one variable (the same) for imputing the missing values
- Multivariate : Using multiple variables for imputing the missing values in one variable
- This may be done by two ways...
 - Using basic python/ pandas tools
 - Use `fillna()` function to fill in missing values
 - Using special packages like `sklearn`
 - Use `SimpleImputer()` function to fill in missing values

```
# get the index or location of the missing values
idx=df.index[df.gender.isna()].to_list()
```

✓ (4.2.1) Fill missing values with specific guess (arbitrary) values

- Applicable for numerical and categorical variables
- Useful when
 - missing data are **MNAR** type
- Advantage: Simple
- Disadvantages:
 - Changes the shape of the distribution
 - Changes variability in the data
 - Creates outliers
 - Changes in covariance/ Correlation

```
df.gender.loc[idx[:101]]='Male'
df.gender.loc[idx[:101]]
```

✓ (4.2.2) Fill missing values with the mean/median/mode value(s)

- Mean/Median values applicable for numerical data
- Mode value applicable for categorical data
- Useful when
 - missing data are **MCAR** type
 - percentage of missing data is less than 5%.
- Advantage: Simple
- Disadvantages:
 - Changes the shape of the distribution
 - Changes variability in the data
 - Creates outliers
 - Changes in covariance/ Correlation

```
fill_mean = df.city_development_index.fillna(df['city_development_index'].mean())
fill_mean
```

(4.2.3) Fill missing values with the mean/median/mode value within groups

- fill missing values in the group by the Mean/Median values for the same group
- Useful when
 - missing data are **MCAR** type
 - percentage of missing data is less than 5%.
- Advantage: Simple
- Disadvantages:
 - Changes the shape of the distribution
 - Changes variability in the data
 - Creates outliers
 - Changes in covariance/ Correlation

(4.2.4) End of distribution imputation

- Applicable for numerical variables
- Useful when
 - missing data are **MNAR** type
- If distribution of missing values is
 - Normal : fill missing values by $\mu+3\sigma$
 - Skewed : fill missing values by $Q1-1.5 \text{ IQR}$ or $Q3+1.5 \text{ IQR}$
- Advantage: Easy to apply
- Disadvantages:
 - Changes the shape of the distribution
 - Changes variability in the data
 - Creates outliers
 - Changes in covariance/ Correlation

(4.2.5) Hot deck method

Fill each missing value with an existing value from a similar case within the dataset

✓ (4.2.6) fillna() methods

1. fillna() is used for imputing missing values

- `ffill` - Replace NaNs with **last observed value**
- `bfill` - Replace NaNs with **next observed value**

```
df_ts = pd.read_csv('./data/city_day_aqi.csv', parse_dates=True, index_col='Date')
df_ts1=df_ts.copy(deep=True)
df_ts.head()
```

```
#Missing Values
df_ts_mis= pd.concat([df_ts.isnull().sum(), 100 * df_ts.isnull().sum() / len(df_ts)], axis=1)
df_ts_mis.rename(columns = {0 : 'Missing', 1 : '% of Total'})
```

✓ (4.2.6.1) Impute missing values with `ffill`

Replace all NA's with the value that comes directly after it (*last observed value*) in the same column, then replace all the remaining NA's with 0 for numerical data.

```
# Imputing Xylene value
df_ts['Xylene'][50:70]

df_ts.fillna(method='ffill', inplace=True)
df_ts['Xylene'][50:70]
```

✓ (4.2.6.2) Impute missing values with `bfill`

Replace all NA's with the value that comes directly before it in the same column, then replace all the remaining NA's with 0 for numerical data.

```
# Imputing PM10 value
df_ts['PM10'][1580:1600]

df_ts.fillna(method='bfill', inplace=True)
df_ts['PM10'][1580:1600]
```

✓ (4.3) Imputation using Interpolation

- imputing using backward fill and forward fill isn't the best solution to address the missing value problem.
- A better alternative would be to use interpolation methods, where the values are filled with incrementing or decrementing values.
- `interpolate()` interpolation methods
 - `method="linear"`
 - fill missing values with an increasing order between the previous and next observed values. (treats values as *equally spaced* by ignoring the index)