The Learning Hub

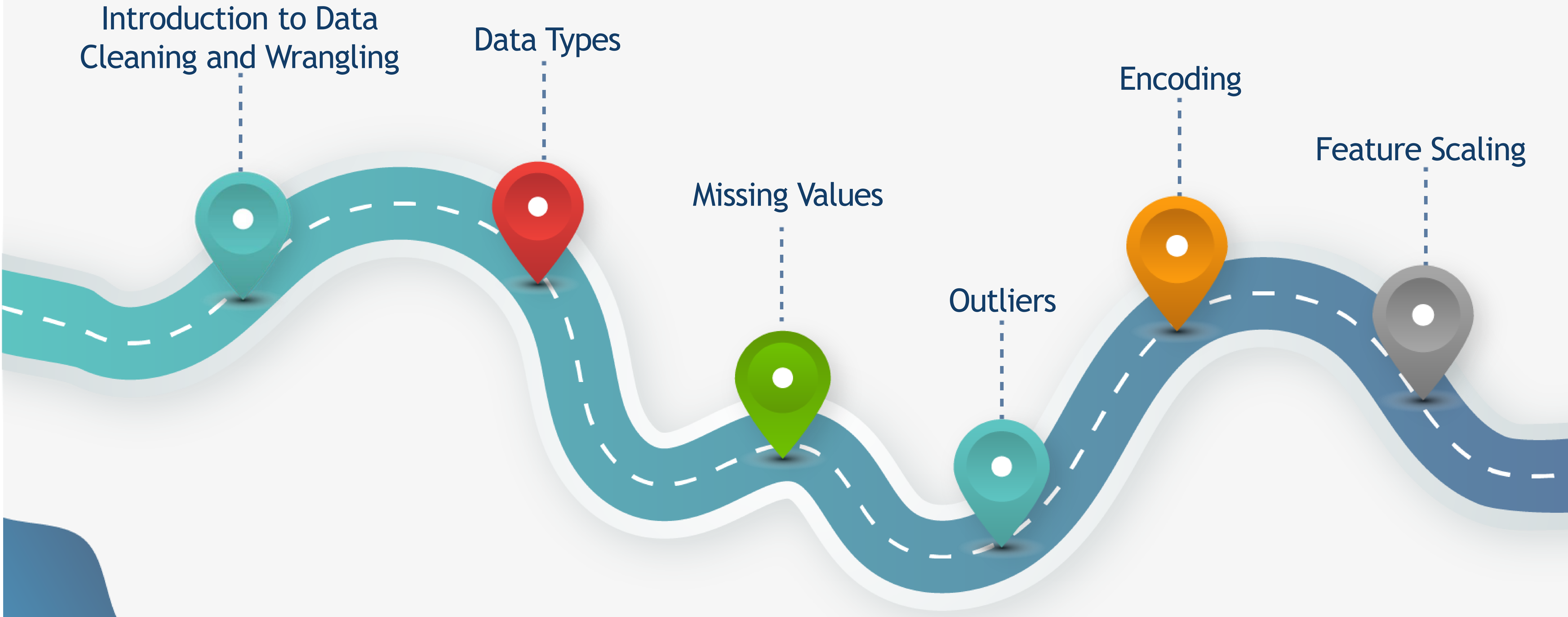
Data Science Advanced

Data Preprocessing

Education and Training Solutions

2023

The Learning Hub



Data Cleaning

Data Cleaning Definition

- **Data Cleaning (Data Cleansing)** is the process of finding, correcting, and structuring problematic data points within a data set, so that it's generally uniform and prepared for analysis.
- This includes **removing corrupt**, **irrelevant**, and **incomplete data** and **formatting** it into a language that computers can understand for optimal analysis.

Data Cleaning Definition

- Goal of **Data Cleaning** is to Identify and remove corrupt or irrelevant without deleting the necessary data to produce insights to ensure the best balance between data accuracy and minimal loss in sample size.
- It's important to remove these inconsistencies and format it into a language that computers can understand in order to increase the validity of the data set.

Data Cleaning Purposes

1. Removing outliers, duplicate and unnecessary values from the dataset that can potentially skew the results when analysing the data.
2. Imputing and changing missing values and standardizing the data format to enhance the quality and consistency.
3. Identifying duplicate values and standardizing systems of measurements.
4. Detect and fix structural errors and typos, and validate the data to make it easier to handle.

Data Cleaning Benefits

1. Errors Elimination.
2. Reduced costs associated with errors.
3. Enhances the integrity of data.
4. Ensures the highest quality of information for decision making.
5. Ensures ease of access to data.
6. Faster time to insights.

Data Wrangling Definition

- **Data Wrangling (Data Munging)** Is the process of transforming and mapping data from one format into another.

- **Purpose:** is to prepare the data in a way that makes it accessible for effective use further down the line, to enhance the quality and assure useful data.

Data Wrangling Purposes

1. Renaming columns and indices.
2. Detect and remove duplications.
3. Detect low variance feature.

Renaming Columns and Indices

- Pandas has a built-in function called **rename()** to change the column names. It's useful when you want to rename some selected columns.
  - Rename columns using a **dictionary mapping**.

```
# Rename columns
#   PassengerId  ->  Id
#   Pclass       ->  Classdf.rename(
columns=({ 'PassengerId': 'Id', 'Pclass': 'Class'}),
inplace=True,
`df.head()
```

Renaming Columns and Indices

- Instead of a dictionary mapping, we can also pass a function to the columns argument. For example, to convert column names into lowercase, we can pass a **str.lower function.**

```
df.rename(columns=str.lower).head()
```

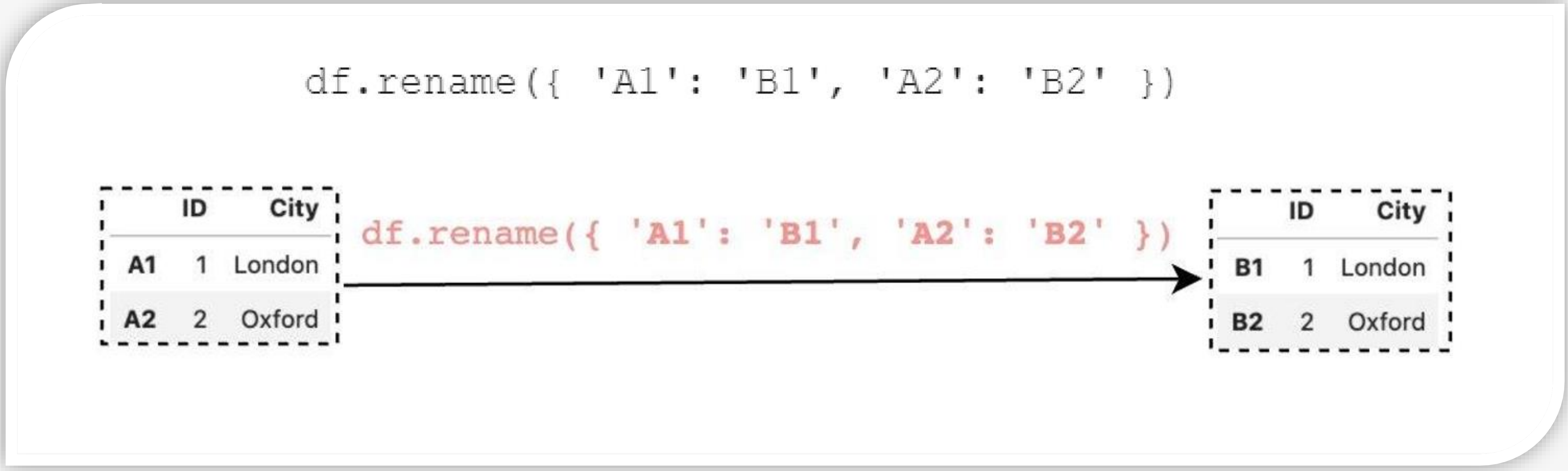
Renaming Columns and Indices

- **rename() function** can be used to rename index as well. Let's first create a Data Frame:

```
df = pd.DataFrame(
    { "ID": [1, 2], "City": ['London', 'Oxford']},
    index=['A1', 'A2'],
)
```

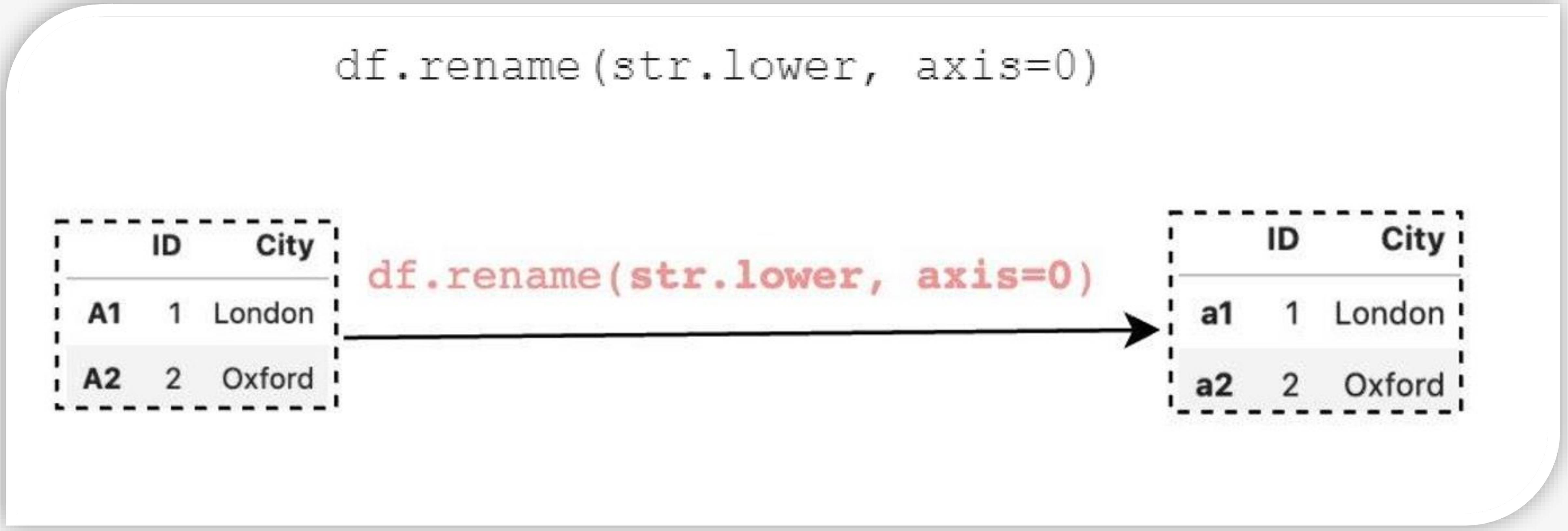
Renaming Columns and Indices

- We can use **dictionary mapping** to rename index:



Renaming Columns and Indices

- We can also pass a **function** and set the axis argument to **0 or 'index'.**



Detect and Remove Duplications

- Rows that have identical data are probably useless, if not dangerously misleading during model evaluation.
- A duplicate row in a row where each value in each column forth at row appears in identically the same order(same column values)in another row.
- Typically, this is not the case and machine learning algorithms will perform better by identifying and removing rows with duplicate data.
- From an algorithm evaluation perspective, duplicate rows will result in misleading performance.

Detect and Remove Duplications

The pandas function **duplicated()** will indicate whether a given row is duplicated or not.

- Rows are marked as **False** to indicate that it is not a duplicate.
- Rows are marked as **True** to indicate that it is a duplicate.

Rows of duplicated content in a dataset should probably be deleted from the dataset prior to modelling.

Detect and Remove Duplications

```
# locate rows of duplicate data
from pandas import read_csv
# define the location of the dataset
path =
'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
# load the dataset
df = read_csv(path, header=None)
# calculate duplicates
dups = df.duplicated()
# report if there are any duplicates

print(dups.any())
# list all duplicate rows
print(df[dups])
```

Detect and Remove Duplications

- There are many ways to achieve this, although Pandas provides the **drop_duplicates()** function that achieves exactly this.

```
# delete rows of duplicate data from the dataset
from pandas import read_csv
# define the location of the dataset
path = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
# load the dataset
df = read_csv(path, header=None)
print(df.shape)
# delete duplicate rows
df.drop_duplicates(inplace=True)
print(df.shape)
```

Detect and Remove Duplications

- Running the example first loads the dataset and reports the number of rows and columns.
- Next, the rows of duplicated data are identified and removed from the Data Frame. Then the shape of the Data Frame is reported to confirm the change.

```
(150, 5)
(147, 5)
```

Detect Low Variance Feature

- **Variance Threshold** is one of the simplest baseline approaches to feature selection. It deletes all features whose variance doesn't meet some threshold. By default, it removes all zero-variance features, i.e. features that have the same value in all samples

$$Var[X] = p(1-p)$$

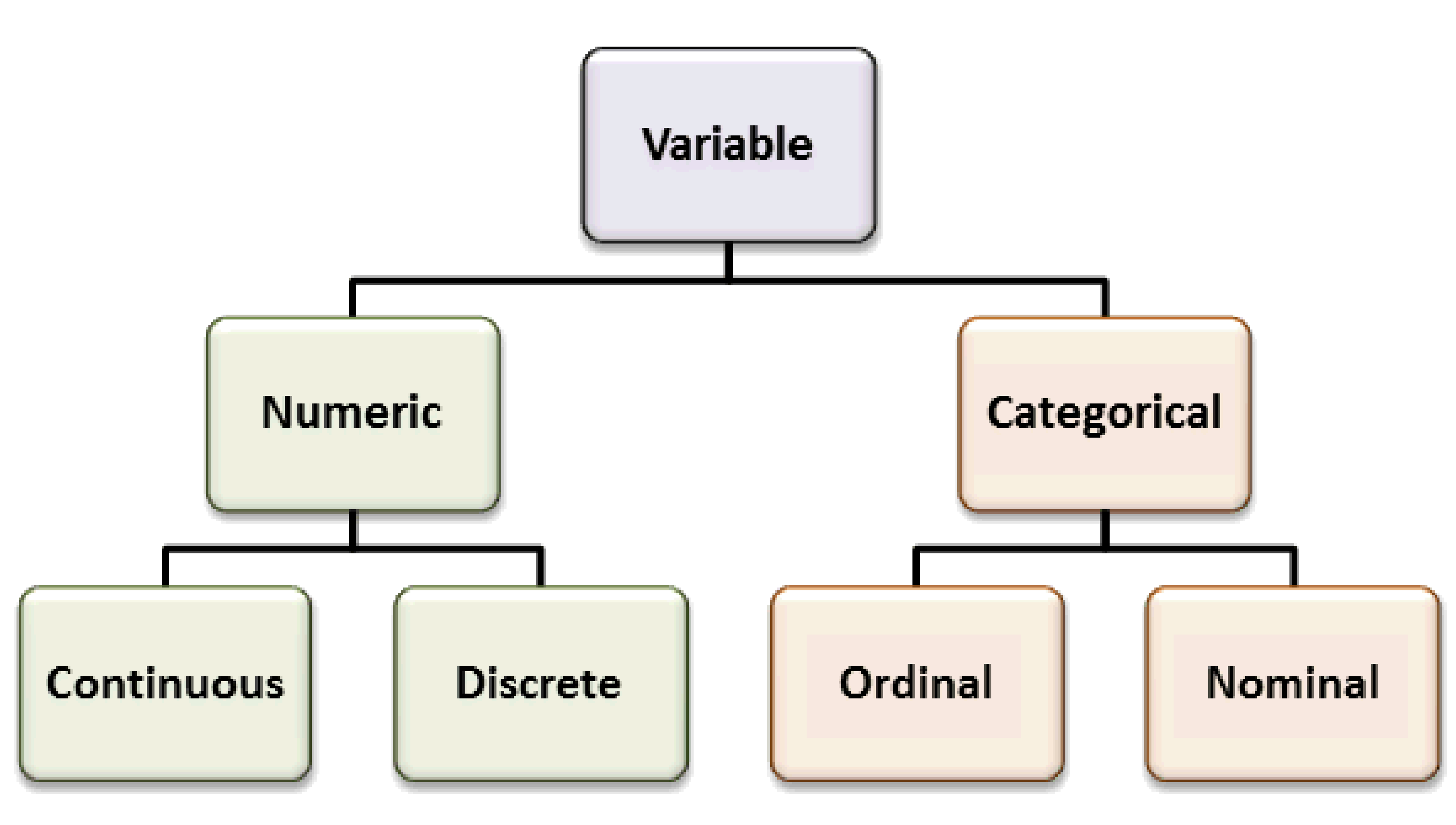
Detect Low Variance Feature

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)
array([[0, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1]])
```

Data Types

Data Types

- Data can come in many forms, but machine learning models rely on four primary data types. These include **numerical data**, **categorical data**, time series data, and text data.



Numeric Data

- **Continuous Variables** are numeric variables that have an infinite number of values between any two values.
  - Example: The length of a part or the date and time a payment is received.

- **Discreate Variables** Are numeric variables that have a countable number of values between any two values. A discrete variable is always numeric.
  - Examples:

    The number of customer complaints.

    The number of flaws or defects.

Categorical Data

- **Categorical Variables** is a type of data that consists of a finite number of categories, considered as gathered information that is divided into groups or categories with the aid of names or labels.
- Examples:
  - Gender (Male, Female).
  - Transportation choices (Car, Train, Subway, Bus, Plane).
  - Payment method (Cash, Credit Cards, Mobile Payments).
  - Marriage status (Single, Married).
  - Performance ratings (Poor, Fair, Average, Good, Excellent).

Categorical Data

1. **Ordinal Variables** the categories have an inherent Order or type of categorical data consisting of a set of orders or scales.

- Example: a list of patients consists of sugar level in the human body which can be classified into high, low, and medium classes.

2. **Nominal Variables** the categories do not have an inherent order or the type of categorical data consists of names or labels without any numerical

- Example: the name of the different departments in any organization such as the human resource department, research, and development department, accounts and billing department, etc.

Categorical vs Numeric Data

Category	Numeric
1. Gender 2. Neighbourhood 3. Scholarship 4. Hypertension 5. Diabetes	1. PatientID 2. AppointmentID 3. Age 4. Hypertension 5. Diabetes

Missing Values

Missing Values Definition

- **Missing Data** is defined as the absent values or the data that is not stored for some feature/s in the given dataset.
- **Missing Values types:**

1. **Missing Completely At Random (MCAR).**
2. **Missing At Random (MAR).**
3. **Missing Not At Random (MNAR).**

Missing Values Types

Table 1

Types of Missing Values

Types of missing values	Description	Possible causes
Missing completely at random	Missing data occur completely at random without being influenced by other data.	Consent withdrawal, omission of major exams, death, discontinued follow-up and serious adverse reactions.
Missing at random	Missing data occur at a specific time point in conjunction with participant dissatisfaction with study outcomes and ongoing participation	Refusal to continue measurements.
Not missing at random	Missing data occur when a patient who is not satisfied with study outcomes performs the required measurements on his own, before the scheduled measurement.	If a patient finds the results of self-measurement dissatisfactory in addition to dissatisfaction related to the study, the patient may refuse further measurements.

Standard Missing Values

- The missing values that Pandas can recognize and detect.
- In the third row, there's an empty cell.
- In the seventh row, there's an "NA" value.
- Pandas will detect both empty cells and "NA" types as **missing values.**

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

Non-Standard Missing Values

- The missing values have different formats.
- In the NUM_BEDROOMS column, there are four missing values.
- n/a, NA, --, na It's important to recognize these non-standard types of missing values for purposes of summarizing and transforming missing values.

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

Unexpected Missing Values

- For example, if our feature is expected to be a string, but there's a numeric type, then technically this is also a **missing value**.
- In the fourth row, there's the number 12.
- The response for Owner Occupied should clearly be a string (Y or N), so this numeric type should be a missing value.

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

Why Handling Missing Values?

- It is important to handle the missing values in the right way.
- Missing values in the dataset cause many machine learning algorithms to fail.
- Lead to building a biased machine learning model which will result in faulty results if the missing values are not handled correctly.
- Missing data can cause a reduction of precision in the statistical analysis.

Handling Missing Values Methods

1. Complete Case Analysis:

- This method aims to use the data of variables observed after removing all missing values.

- Although it has the advantage of the simplicity of analysis, decreasing the sample size and reducing statistical power are disadvantages because drawing statistical inferences becomes difficult during analysis.

Handling Missing Values Methods

2. Available Case Analysis:

This method deals with only the data available for each analysis. It allows a larger sample size than that used for complete case analysis. However, this approach causes sample sizes to vary between the variables used in the analysis.

Handling Missing Values Methods

3. Imputation Analysis:

- This method involves the replacement of values with substituted values computed from statistical analysis to Preserv data set completed without missing values for analysis. Imputations can be created by using either an explicit or an implicit modelling approach.

- It includes different methods of imputation by mean, median, probability, ratio, regression, predictive regression, and assumption of distribution.

Example of Missing Values

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

Example of Missing Values

1. Remove the Column:

- In this case let's remove the Age column from our dataset and check for accuracy.

- This approach can be used when there are many null values in the column.

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

updated_df.info()
```

Example of Missing Values

- After removing **Age column**.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pclass  891 non-null    int64
 1   Sex     891 non-null    int64
 2   SibSp   891 non-null    int64
 3   Parch   891 non-null    int64
 4   Fare    891 non-null    float64
dtypes: float64(1), int64(4)
memory usage: 34.9 KB
```

Example of Missing Values

- Accuracy after removing **Age column**.

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test =
train_test_split(updated_df,y,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))


0.7947761194029851
```

Example of Missing Values

2. Remove the Null rows:

- If there is a certain row with many missing values, then you can remove the entire row with all the features in that row.
- axis=0 is used to drop the row with `NaN` values.

```
updated_df = newdf.dropna(axis=0)


y1 = updated_df['Survived']
updated_df.drop("Survived",axis=1,inplace=True)
updated_df.info()
```

Example of Missing Values

- After removing **Null rows**.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Pclass 714 non-null    int64
 1   Sex    714 non-null    int64
 2   Age    714 non-null    float64
 3   SibSp  714 non-null    int64
 4   Parch  714 non-null    int64
 5   Fare   714 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 39.0 KB
```

Example of Missing Values

- Accuracy after removing **Null rows**.

```
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test =
train_test_split(updated_df,y1,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))


0.8232558139534883
```

Example of Missing Values

3. Fill missing values:

- The possible ways to fill the Missing Values are:

1. Filling the missing data with the **mean** or **median** value in case of a numerical variable.
2. Filling the missing data with the **mode** in case of a categorical value.

Example of Missing Values

- **fillna()** function can be used to fill the null values in the dataset.

```
updated_df = df
updated_df['Age']=updated_df['Age'].fillna(updated_df['Age'].mean())
updated_df.info()
```

Example of Missing Values

- After filling **Null values** in the **Age column** with **Mean**.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Survived  891 non-null    int64
 1   Pclass    891 non-null    int64
 2   Sex       891 non-null    int64
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64
 6   Fare      891 non-null    float64
dtypes: float64(2), int64(5)
memory usage: 48.9 KB
```

Example of Missing Values

- Accuracy after filling with **Mean**.

```
y1 = updated_df['Survived']
updated_df.drop("Survived",axis=1,inplace=True)
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test =
train_test_split(updated_df,y1,test_size=0.3)
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(metrics.accuracy_score(pred,y_test))


0.7798507462686567
```

The Learning Hub

Outliers

What is the Outlier?

- **Outlier** in statistics, an outlier is an observation point that is distant from other observations.
- Let's have a look at some examples. Suppose you have been asked to observe the performance of the Indian cricket team i.e. the Run made by each player and collect the data.

Players	Scores
Player1	500
Player2	350
Player3	10
Player4	300
Player5	450

What is the Outlier?

- Now that we know outliers can either be a **mistake** or just **variance**, how would you decide if they are important or not.
- Well, it is pretty simple if they are the result of a mistake, then we can ignore them, but if it is just variance in the data we would need to think a bit further. Before we try to understand whether to ignore the outliers or not, we need to know the **ways to identify them**.

Detecting Outliers Ways

- **Graphical Ways:**

1. Scatter Plots
2. Box Plots

- **Numerical Ways:**

1. Z-Score
2. IQR

Example of Outlier

- So, Let's get start. We will be using **Boston House Pricing Dataset** which is included in the sklearn dataset API. We will load the dataset and separate out the features and targets.

```
boston = load_boston()
x = boston.data
y = boston.target
columns = boston.feature_names

#create the dataframe
boston_df = pd.DataFrame(boston.data)
boston_df.columns = columns
boston_df.head()
```

Example of Outlier

- Features/independent variable will be used to look for any outlier. Looking at the data below, it s seems, we only have **numeric values**.

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Boston Housing Data

Example of Outlier

- There are two types of analysis we will follow to find the outliers: **Uni-variate(one variable outlier analysis)** and **Multi-variate(two or more variable outlier analysis).** When you will start coding and plotting the data, you will see yourself that how easy it was to detect the outlier. To keep things simple, we will start with the basic method of detecting outliers and slowly move on to the advance methods.

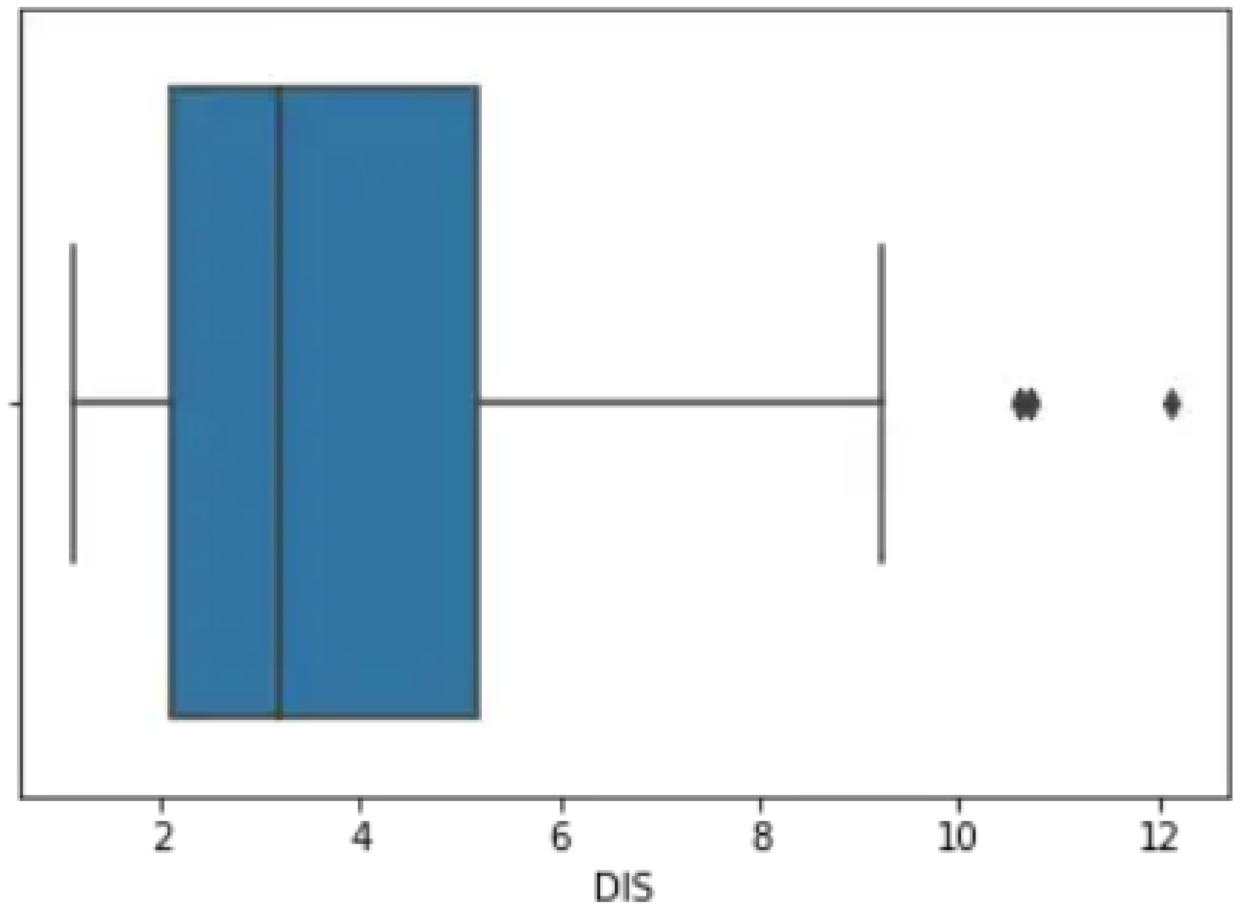
Discover outliers with Box Plot

- **Box Plot** is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points.

- Above definition suggests, that if there is an outlier it will plotted as point in boxplot but other population will be grouped together and display as boxes. Let's try and see it ourselves.

Discover outliers with Box Plot

```
import seaborn as sns
sns.boxplot(x=boston_df['DIS'])
```



Boxplot — Distance to Employment Center

Discover outliers with Box Plot

- Previous plot shows three points between **10 to 12, these are outliers** as there are not included in the box of other observation i.e. no where near the quartiles.

- Here we analysed **Uni-variate outlier** i.e. we used **DIS column** only to check the outlier. But we can do multivariate outlier analysis too. Can we do the multivariate analysis with Box plot? Well it depends, if you have a **categorical values** then you can use that with any continuous variable and do multivariate outlier analysis. As we do not have categorical value in our Boston Housing dataset, we might need to forget about using box plot for multivariate outlier analysis.

Discover outliers with Scatter Plot

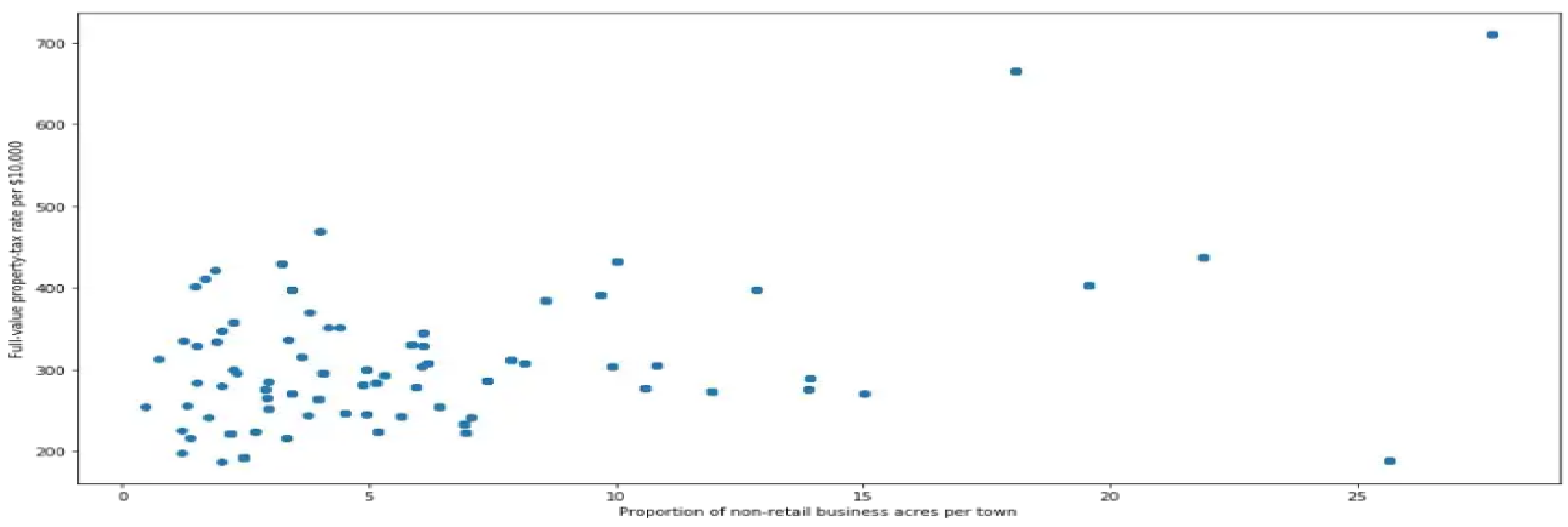
- **Scatter Plot** is a type of plot or mathematical diagram using Cartesian coordinates to display values for typically two variables for a set of data. The data are displayed as a **collection of points**, each having the value of **one variable** determining the position on the **horizontal** axis and the value of the **other variable** determining the position on the **vertical** axis.

- As the definition suggests, the scatter plot is the collection of points that shows values for **two variables**. We can try and draw scatter plot for two variables from our housing dataset.

Discover outliers with Scatter Plot

- Looking at the plot below, we can most of data points are lying bottom left side but there are points which are far from the population like top **right corner**.

```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(boston_df['INDUS'], boston_df['TAX'])
ax.set_xlabel('Proportion of non-retail business acres per town')
ax.set_ylabel('Full-value property-tax rate per $10,000')
plt.show()
```



Scatter plot — Proportion of non-retail business acres per town v/s Full value property tax

Discover outliers with Z-Score

- **Z-Score** is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured.

- The intuition behind Z-score is to describe any data point by finding their relationship with the Standard Deviation and Mean of the group of data points. Z-score is finding the distribution of data where **mean is 0** and **standard deviation is 1** i.e. normal distribution.

Discover outliers with Z-Score

- You must be wondering that, how does this help in identifying the outliers? Well, while calculating the Z-score we re-scale and centre the data and look for data points which are too far from zero. These data points which are way too far from zero will be treated as the outliers. In most of the cases **a threshold of 3 or -3** is used i.e. if the Z-score value is **greater than or less than 3 or -3 respectively, that data point will be identified as outliers.**

Discover outliers with Z-Score

- We will use **Z-score function** defined in **SciPy library** to detect the outliers.

```
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(boston_df))
print(z)
```

```
[[0.41771335 0.28482986 1.2879095  ... 1.45900038 0.44105193 1.0755623 ]
 [0.41526932 0.48772236 0.59338101 ... 0.30309415 0.44105193 0.49243937]
 [0.41527165 0.48772236 0.59338101 ... 0.30309415 0.39642699 1.2087274 ]
 ...
 [0.41137448 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.98304761]
 [0.40568883 0.48772236 0.11573841 ... 1.17646583 0.4032249  0.86530163]
 [0.41292893 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.66905833]]
```

Z-score of Boston Housing Data

Discover outliers with Z-Score

- Looking the code and the output above, it is difficult to say which data point is an outlier. Let's try and define a **threshold** to identify an outlier.

```
threshold = 3
print(np.where(z > 3))
```

This will give a result as below -

```
(array([ 55,  56,  57, 102, 141, 142, 152, 154, 155, 160, 162, 163, 199,
       200, 201, 202, 203, 204, 208, 209, 210, 211, 212, 216, 218, 219,
       220, 221, 222, 225, 234, 236, 256, 257, 262, 269, 273, 274, 276,
       277, 282, 283, 283, 284, 347, 351, 352, 353, 353, 354, 355, 356,
       357, 358, 363, 364, 364, 365, 367, 369, 370, 372, 373, 374, 374,
       380, 398, 404, 405, 406, 410, 410, 411, 412, 412, 414, 414, 415,
       416, 418, 418, 419, 423, 424, 425, 426, 427, 427, 429, 431, 436,
       437, 438, 445, 450, 454, 455, 456, 457, 466], dtype=int64), array([ 1,  1,  1, 11, 12,  3,  3,
3,  3,  3,  3,  3,  1,  1,  1,  1,  1,
        1,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  5,  3,  3,  1,  5,
        5,  3,  3,  3,  3,  3,  3,  1,  3,  1,  1,  7,  7,  1,  7,  7,  7,
        3,  3,  3,  3,  3,  5,  5,  5,  3,  3,  3, 12,  5, 12,  0,  0,  0,
        0,  5,  0, 11, 11, 11, 12,  0, 12, 11, 11,  0, 11, 11, 11, 11, 11,
       11,  0, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11],
      dtype=int64))
```

Data points where Z-scores is greater than 3

Discover outliers with Z-Score

- Don't be confused by the results. The first array contains the list of row numbers and second array respective column numbers, which mean **z[55][1] have a Z-score higher than 3.**

```
print(z[55][1])
3.375038763517309
```

So, the data point — 55th record on column ZN is an outlier.

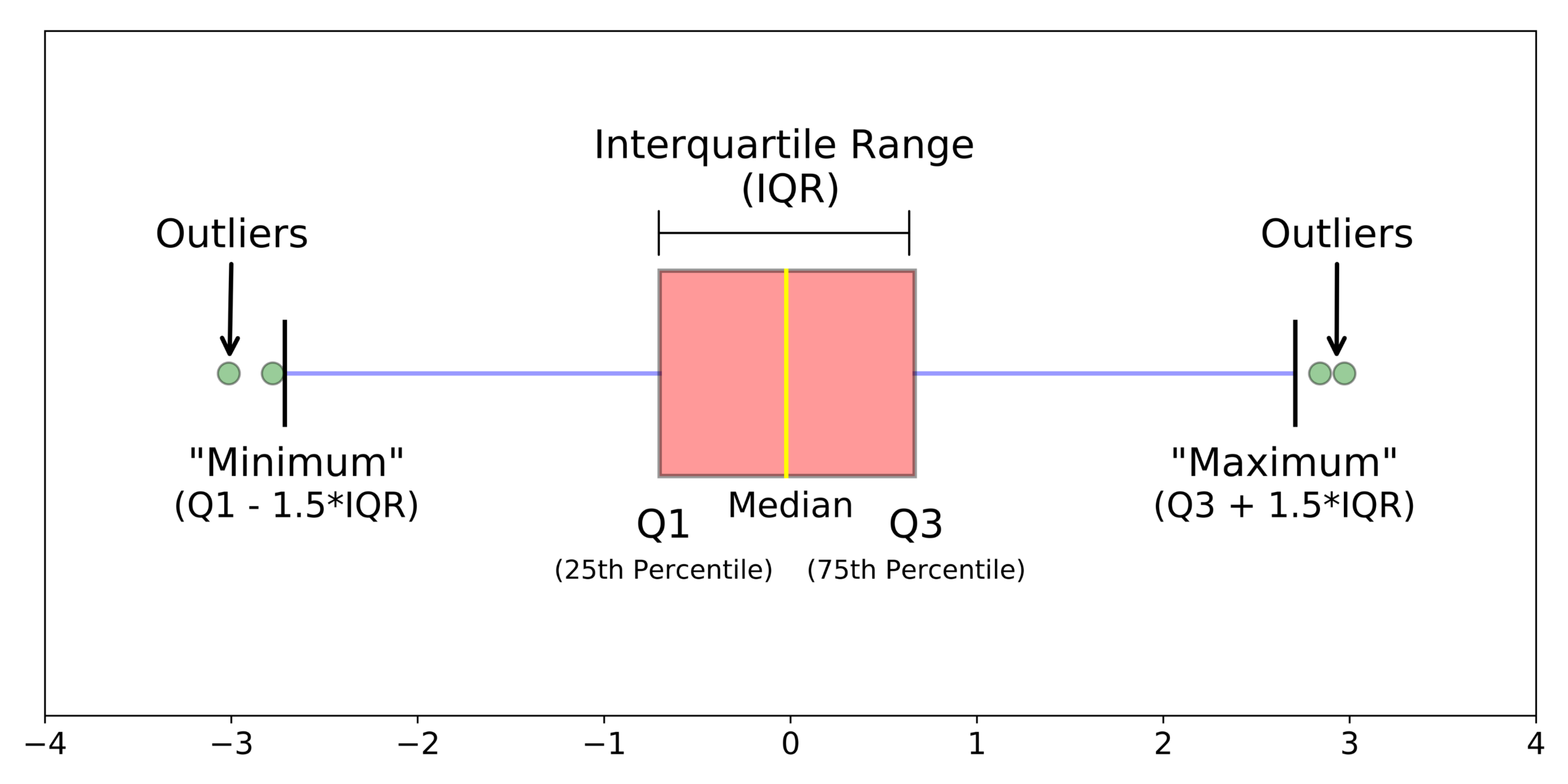
Discover outliers with IQR-Score

- **The interquartile range (IQR)** is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles, **IQR = Q3 − Q1**.

- In other words, the IQR is the first quartile subtracted from the third quartile; these quartiles can be clearly seen on a box plot on the data.

- IQR is somewhat similar to Z-score in terms of finding the distribution of data and then keeping some threshold to identify the outlier.

Discover outliers with IQR-Score



Discover outliers with IQR-Score

- Let's find out we can box plot uses IQR and how we can use it to find the list of outliers as we did using Z-score calculation. First we will calculate IQR.

```
Q1 = boston_df_o1.quantile(0.25)
Q3 = boston_df_o1.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

Here we will get IQR for each column.

```
CRIM         3.565378
ZN          12.500000
INDUS       12.910000
CHAS         0.000000
NOX          0.175000
RM           0.738000
AGE         49.050000
DIS          3.088250
RAD         20.000000
TAX        387.000000
PTRATIO      2.800000
B           20.847500
LSTAT       10.005000
dtype: float64
```

IQR for each column

Discover outliers with IQR-Score

- The data point where we have False that means these values are valid whereas True indicates presence of an outlier.

```
print(boston_df_o1 < (Q1 - 1.5 * IQR)) |(boston_df_o1 > (Q3 + 1.5 *
IQR))
```

4	False	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False	False	False	False	True	False
19	False	False	False	False	False	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False	False	False	False	False	False

Detecting outlier with IQR

Removing outliers with Z-Score

- We saw how one can detect the outlier using Z-score but now we want to remove or filter the outliers and get the clean data. This can be done with just one line code as we have already calculated the Z-score.

```
boston_df_o = boston_df_o[(z < 3).all(axis=1)]
```

```
boston_df.shape
```

```
(506, 13)
```

```
boston_df_o.shape
```

```
(415, 13)
```

With and without outlier size of the dataset

Removing outliers with IQR-Score

- Just like Z-score we can use previously calculated IQR score to filter out the outliers by keeping only valid values.

```
boston_df_out = boston_df_o1[~((boston_df_o1 < (Q1 - 1.5 * IQR)) |
(boston_df_o1 > (Q3 + 1.5 * IQR))).any(axis=1)]

boston_df_out.shape
```

Encoding

Encoding Categorical Data Types

1. **Label Encoding (Ordinal Encoding)** this type of encoding can be used when the data consists of ordinal variables, ordinal encoding converts each label into integer values and the encoded data represents the sequence of labels.

2. **One Hot Encoding** for categorical variables where there is no ordinal relationship, integer coding may be insufficient, at best, or misleading to the model at worst.
   - The one-hot encoding transform is available in the scikit-learn Python machine learning library via the OneHotEncoder class.

Encoding Categorical Data Types

In one hot encoding, for each level of a categorical feature, we create a new variable.

Each category is mapped with binary numbers (0 or 1). Here, **0 represents the absence, and 1 represents the presence of that category.**

One Hot Encoding Example

Index	Animal
0	Dog
1	Cat
2	Sheep
3	Horse
4	Lion

One-Hot code

Index	Dog	Cat	Sheep	Lion	Horse
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0

Label/Ordinal Encoding Example

	Degree
0	High school
1	Masters
2	Diploma
3	Bachelors
4	Bachelors
5	Masters
6	Phd
7	High school
8	High school

	Degree
0	1
1	4
2	2
3	3
4	3
5	4
6	5
7	1
8	1

Feature Scaling

Feature Scaling Techniques

1. **Normalization** is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

- Here's the formula for normalization:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

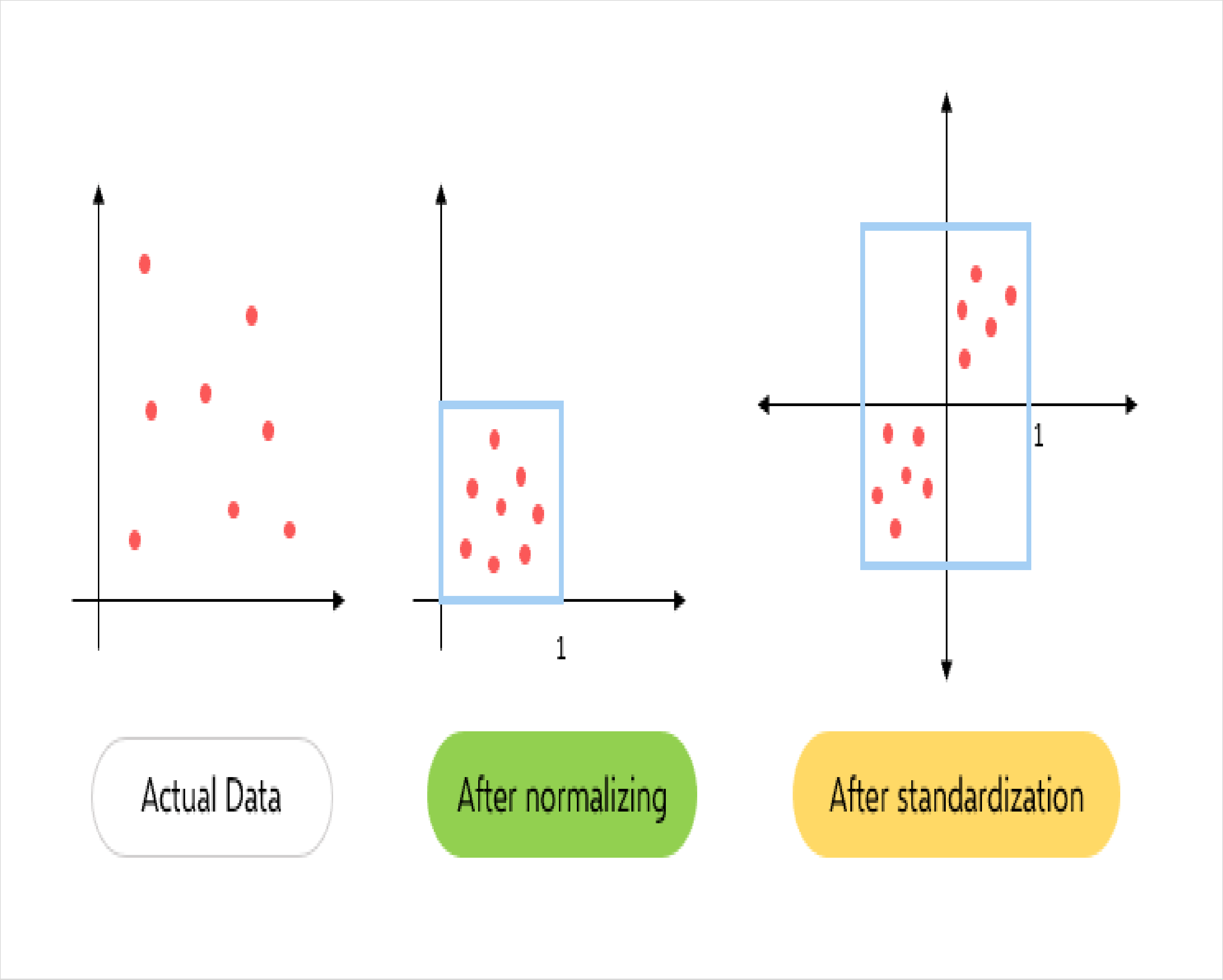
Feature Scaling Techniques

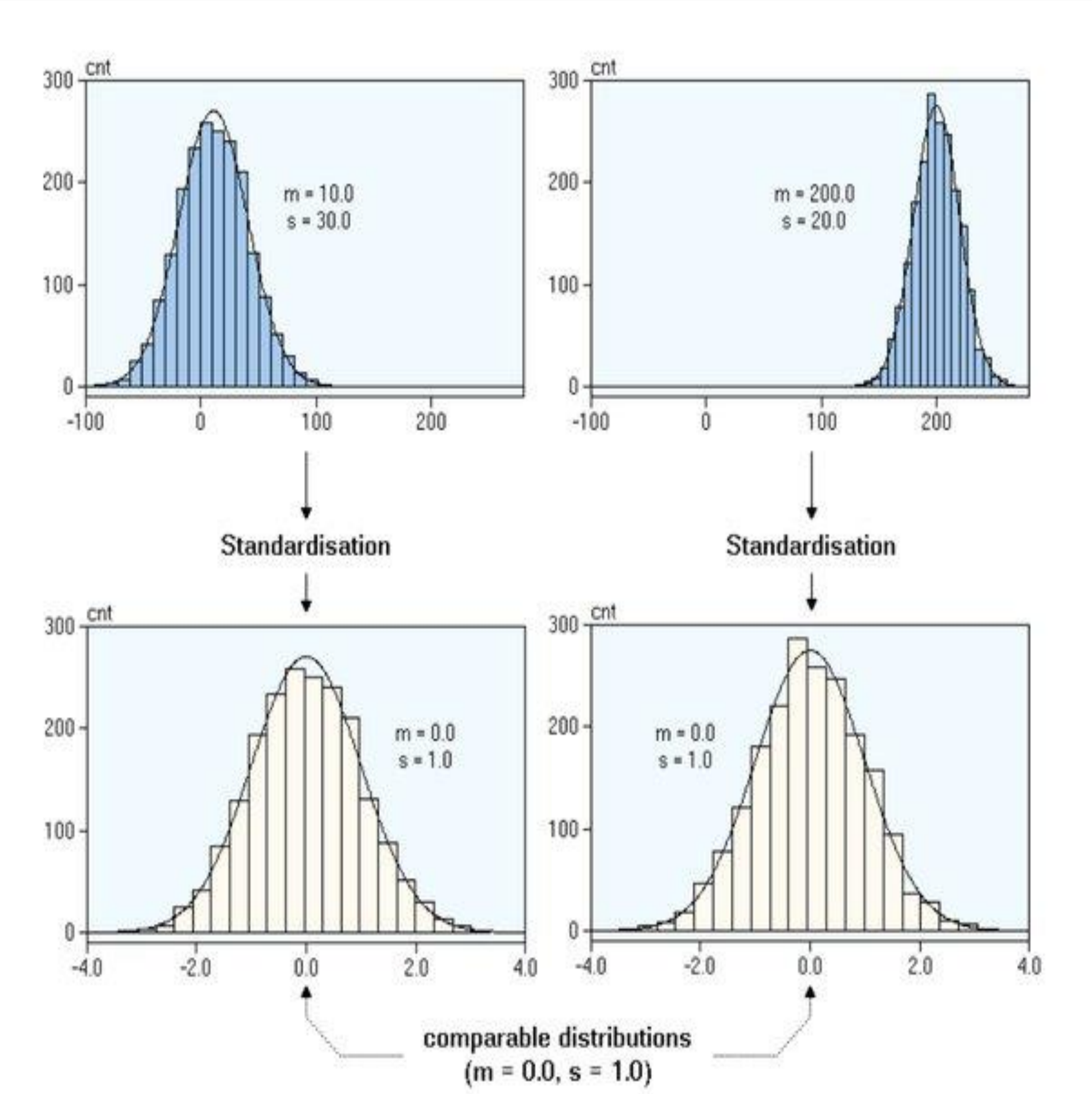
2. **Standardization** is another scaling technique where the values are cantered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

- Here's the formula for standardization:

$$X' = \frac{X - \mu}{\sigma}$$

Feature Scaling Techniques





Normalization using Sklearn

- To normalize your data, you need to import the **MinMaxScalar** from the sklearn library and apply it to our dataset.

```
# data normalization with sklearn
from sklearn.preprocessing import
MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(X_train)

# transform training data
X_train_norm =
norm.transform(X_train)

# transform testing dataabs
X_test_norm =
norm.transform(X_test)
```

Standardization using Sklearn

- To standardize your data, you need to import the **StandardScaler** from the sklearn library and apply it to our dataset.

```
# data standardization with  sklearn
from sklearn.preprocessing import StandardScaler


# copy of datasets
X_train_stand = X_train.copy()
X_test_stand = X_test.copy()


# numerical features
num_cols =
['Item_Weight','Item_Visibility','Item_MRP','Outlet_Establish
ment_Year']
```

Standardization using Sklearn

```
# apply standardization on numerical features
for i in num_cols:

    # fit on training data column
    scale = StandardScaler().fit(X_train_stand[[i]])

    # transform the training data column
    X_train_stand[i] = scale.transform(X_train_stand[[i]])

    # transform the testing data column
    X_test_stand[i] = scale.transform(X_test_stand[[i]])
```

References

- https://www.formpl.us/blog/categorical-data
- https://www.formpl.us/blog/categorical-numerical-data
- https://analyticsindiamag.com/a-complete-guide-to-categorical-data-encoding/
- Chapter-2 Data and It's Different Types | by Ashish Patel | ML Research Lab | Medium
- Ways to Detect and Remove the Outliers | by Natasha Sharma | Towards Data Science
- Different types of Encoding - AI ML Analytics (ai-ml-analytics.com)

THANK YOU