```
import numpy as np
import pandas as pd
```

Basic Data Cleaning

- Data cleaning means fixing bad data in your data set.
- Bad data could be:
    - Empty cells
    - Data in wrong format
    - Wrong data
    - Duplicates

> **1. Removing Duplicates :**

- To discover duplicates, we can use the duplicated() method.
- The duplicated() method returns a Boolean values for each row.
- To remove duplicates, use the drop_duplicates() method.
- **Remember:** The (inplace = True) will make sure that the method does NOT return a new DataFrame, but it will remove all duplicates from the original DataFrame.

```
data = {
    'Id': [1,2,1,3,2,2,3],
    'Name': ['A','B','A','D','B','B','D'],
    'City': ['P','O','P','S','O','O','R'],
    'Year': [2013, 2023, 2013, 2025, 2023, 2023, 2017]
}

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

```
   Id Name City  Year
0   1    A    P  2013
1   2    B    O  2023
2   1    A    P  2013
3   3    D    S  2025
4   2    B    O  2023
5   2    B    O  2023
6   3    D    R  2017
```

```
# Find Duplicate rows in the DataFrame
# Returns True for duplicated rows and False otherwise.

print(df.duplicated())
```

```
0    False
1    False
2     True
3    False
4     True
```

```
5     True
6    False
dtype: bool
```

```
# Remove the duplicated rows

print(df.drop_duplicates())
# Returns DF with unique records only
```

```
   Id Name City  Year
0   1    A    P  2013
1   2    B    O  2023
3   3    D    S  2025
6   3    D    R  2017
```

2. Fixing Wrong Data:

- Wrong data may be data not in the default range.
- Or can also be an outlier.
- One way to fix wrong values is to replace them with something else.

```
# 1. Replacing the wrong data with some other value from the column.

data = {
    'Name': ['A','B','A','D','B','B','D'],
    'Duration': [30, 60, 45, 60, 30, 180, 30],
    'City': ['P','O','P','S','O','O','R'],
    'Year': [2013, 2023, 2013, 2025, 2023, 2023, 2017]
}
df = pd.DataFrame(data)
# In Duration, values are : 30, 45 OR 60, but 180 not lies in that.

# Replace 180 with other value
df.loc[5, 'Duration'] = 45

print(df)
```

```
  Name  Duration City  Year
0    A        30    P  2013
1    B        60    O  2023
2    A        45    P  2013
3    D        60    S  2025
4    B        30    O  2023
5    B        45    O  2023
6    D        30    R  2017
```

- For small number of wrong data, can replace them individually.
- But, for larger amount of wrong data, have to create some rules for replacing the wrong data.

```
# 2. Removing Rows
# So, no need to find what to replace with the wrong data.
# But, have to sure that, the wrong data must be very less.

data = {
    'Name': ['A','B','A','D','B','B','D'],
    'Duration': [30, 60, 45, 60, 30, 180, 30],
    'City': ['P','O','P','S','O','O','R'],
    'Year': [2013, 2023, 2013, 2025, 2023, 2023, 2017]
}
df = pd.DataFrame(data)

# Remove rows with Duration >60
for i in df.index:
    if df.loc[i, 'Duration'] > 60:
        df.drop(i, inplace=True)

print(df)
```

```
      Name  Duration City  Year
    0    A        30    P  2013
    1    B        60    O  2023
    2    A        45    P  2013
    3    D        60    S  2025
    4    B        30    O  2023
    6    D        30    R  2017
```

> ### 3. Cleaning Data of Wrong Format:

- Cells with data of wrong format can make it difficult, or even impossible, to analyze data.
- To fix it, you have two options:
    - remove the rows
    - convert all cells in the columns into the same format.

```
# 1. Convert to correct format

data = {
    'Name': ['A','B','A','D','B','B','D'],
    'Duration': [30, 60, 45, 60, 30, 180, 30],
    'City': ['P','O','P','S','O','O','R'],
    'Year': [2013, 2023, 2013, 2025, 2023, 2023, 2017],
    'Date': ['2020/12/01', '2020/12/02', '2020/12/03', '2020/12/04',
             '2020/12/05', np.nan, '2020/12/07']
}

# Data column contains 1 NULL value, which does not match with date format
df = pd.DataFrame(data)

print(df)
```

```
      Name  Duration City  Year        Date
    0    A        30    P  2013  2020/12/01
    1    B        60    O  2023  2020/12/02
    2    A        45    P  2013  2020/12/03
    3    D        60    S  2025  2020/12/04
    4    B        30    O  2023  2020/12/05
    5    B       180    O  2023         NaN
    6    D        30    R  2017  2020/12/07
```

```
# 1. Convert to same format

# Pandas has a to_datetime() method,
# to convert all cells in the 'Date' column into dates.
# So, all values from other format, converted to DateTime.

df['Date'] = pd.to_datetime(df['Date'])
print(df)
# Here, NaT = Not a Date, for null values
```

```
  Name  Duration City  Year       Date
0    A        30    P  2013 2020-12-01
1    B        60    O  2023 2020-12-02
2    A        45    P  2013 2020-12-03
3    D        60    S  2025 2020-12-04
4    B        30    O  2023 2020-12-05
5    B       180    O  2023        NaT
6    D        30    R  2017 2020-12-07
```

```
# 2. Remove rows with null values

df.dropna(subset=['Date'], inplace=True)

print(df)
# Thus, removes records with NULL values.
```

```
  Name  Duration City  Year       Date
0    A        30    P  2013 2020-12-01
1    B        60    O  2023 2020-12-02
2    A        45    P  2013 2020-12-03
3    D        60    S  2025 2020-12-04
4    B        30    O  2023 2020-12-05
6    D        30    R  2017 2020-12-07
```

> #### 4. Cleaning Empty Cells:

- Empty cells can potentially give you a wrong result when you analyze data.

```
# 1. Remove Rows
# Removing some rows in very large dataset, does not affect much

data = {
    'Name': ['A','B','A','D','B','B','D','G','H'],
    'Duration': [30, 60, 45, 60, 30, 45, 30, 45, 60],
    'City': ['P','O','P','S',np.nan,'O','R','S','R'],
    'Year': [2013, np.nan, 2013, 2025, 2023, 2023, 2017, 2013, 2018]
}

df = pd.DataFrame(data)

# Just remove the rows with null cell
df.dropna(inplace=True)

print(df)
```

```
  Name  Duration City    Year
0    A        30    P  2013.0
2    A        45    P  2013.0
3    D        60    S  2025.0
5    B        45    O  2023.0
6    D        30    R  2017.0
7    G        45    S  2013.0
8    H        60    R  2018.0
```

```
# 2. Replace Empty Values
# Fill another value in the place of empty cells
data = {
    'Name': ['A','B','A','D','B','B','D','G','H'],
    'Duration': [30, 60, 45, 60, 30, 45, 30, 45, 60],
    'City': ['P','O','P','S',np.nan,'O','R','S','R'],
    'Year': [2013, np.nan, 2013, 2025, 2023, 2023, 2017, 2013, 2018]
}

df = pd.DataFrame(data)

# Can fill different values for each columns of the DF
df['City'].fillna('T', inplace=True)
df['Year'].fillna(2018, inplace=True)

print(df)
```

```
  Name  Duration City    Year
0    A        30    P  2013.0
1    B        60    O  2018.0
2    A        45    P  2013.0
3    D        60    S  2025.0
4    B        30    T  2023.0
5    B        45    O  2023.0
6    D        30    R  2017.0
7    G        45    S  2013.0
8    H        60    R  2018.0
```

```
# 3. Replace Using Mean, Median, or Mode

data = {
    'Name': ['A','B','A','D','B','B','D','G','H'],
    'Age': [23, 25, 20, 19, 24, 27, np.nan, 30, 29],
    'Duration': [30, 60, 45, 60, 30, 45, 30, 45, 60],
    'City': ['P','O','P','S',np.nan,'O','R','S','R'],
    'Year': [2013, np.nan, 2013, 2025, 2023, 2023, 2017, 2013, 2018]
}
df = pd.DataFrame(data)

# Replace null in Age with Mean of the Age column
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Replace null in City with Mode of the City column
df['City'].fillna(df['City'].mode(), inplace=True)

# Replace null in Year with Median of the Year column
df['Year'].fillna(df['Year'].median(), inplace=True)

print(df)
```

```
  Name     Age  Duration City    Year
0    A  23.000        30    P  2013.0
1    B  25.000        60    O  2017.5
2    A  20.000        45    P  2013.0
3    D  19.000        60    S  2025.0
4    B  24.000        30  NaN  2023.0
5    B  27.000        45    O  2023.0
6    D  24.625        30    R  2017.0
7    G  30.000        45    S  2013.0
8    H  29.000        60    R  2018.0
```

Data Cleaning and Preprocessing

> **Handling missing data with dropna(), fillna():**

- Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways.
- **Syntax:**

```
DataFrameName.dropna(axis=0, how=’any’, thresh=None, subset=None, inplace=False)
```

  - axis - Input can be 0 or 1 for Integer and ‘index’ or ‘columns’ for String.
  - how - how takes string value of two kinds only (‘any’ or ‘all’). ‘any’ drops the row/column if ANY value is Null and ‘all’ drops only if ALL values are null.
  - thresh - thresh takes integer value which tells minimum amount of na values to drop.
  - subset - It’s an array which limits the dropping process to passed rows/columns through list.
  - inplace - It is a boolean which makes the changes in data frame permanent if True.

```
data = {
    'Id': [1,2,3,4,5,6,7],
    'Name': ['A','B','C','D','E','F','G'],
    'City': ['P','O',np.nan,'Q','R','H',np.nan],
    'Year': [2013, 2023, np.nan, np.nan, 2018, 2014, 2017]
}

df = pd.DataFrame(data)

print(df.isna().sum())  # --> Return how many null present in each column
```

```
Id      0
Name    0
City    2
Year    2
dtype: int64
```

```
# Drop all rows with null
# axis=0 : Drop rows having any null value
# axis=1 : Drop cols having any null value

# how='any': Drop evan for only one null as well
# how='all': Drop only if all values are null
print(df.dropna(axis=0, how='all'))
```

```
   Id Name City    Year
0   1    A    P  2013.0
1   2    B    O  2023.0
2   3    C  NaN     NaN
3   4    D    Q     NaN
4   5    E    R  2018.0
5   6    F    H  2014.0
6   7    G  NaN  2017.0
```

```
print(df.dropna(axis=0,thresh=3))
# thresh=int : Determine how many nou-null values must be present, otherwise drop.
# thresh=3: So, drop rows with less than 3 non-null values.
```

```
   Id Name City    Year
0   1    A    P  2013.0
1   2    B    O  2023.0
3   4    D    Q     NaN
4   5    E    R  2018.0
5   6    F    H  2014.0
6   7    G  NaN  2017.0
```

```
# Original DataFrame
print(df)
# So, can see that changes are not permanent to original DF
```

```
   Id Name City    Year
0   1    A    P  2013.0
1   2    B    O  2023.0
2   3    C  NaN     NaN
3   4    D    Q     NaN
4   5    E    R  2018.0
5   6    F    H  2014.0
6   7    G  NaN  2017.0
```

```
# inplace=True : Make changes permanent to the original DF
df.dropna(axis=0,inplace=True)
print(df)
```

```
   Id Name City    Year
0   1    A    P  2013.0
1   2    B    O  2023.0
4   5    E    R  2018.0
5   6    F    H  2014.0
```

> **fillna() :**

- fillna() method replaces the NULL values with a specified value.
- **Syntax:**

```
dataframe.fillna(value, method, axis, inplace, limit, downcast)
```

  - value - Required, Specifies the value to replace the NULL values with. This can also be values for the entire row or column. The value can be any data type: Number, String, Dictionary, Series, DataFrame
  - method - Optional, default None'. Specifies the method to use when replacing. 'backfill', 'bfill', 'pad', 'ffill', None
  - axis - Optional, default 0. The axis to fill the NULL values along. 0, 1, 'index', 'columns'
  - inplace - If True, make changes permanent to the original DF.
  - limit - Optional, default None. Specifies the maximum number of NULL values to fill (if method is specified).
  - downcast - Optional, a dictionary of values to fill for specific data types..

```
data = {
    'Id': [1,2,3,4,5,6,7],
    'Name': ['A','B','C','D','E','F','G'],
    'City': ['P','O',np.nan,'Q','R','H',np.nan],
    'Year': [2013, 2023, np.nan, np.nan, 2018, 2014, 2017]
}

df = pd.DataFrame(data)
df['City'].fillna('New_city', inplace=True)
# inplace=True : Makes changes permanent to the original df

print(df)
```

```
   Id Name      City    Year
0   1    A         P  2013.0
1   2    B         O  2023.0
2   3    C  New_city     NaN
3   4    D         Q     NaN
4   5    E         R  2018.0
5   6    F         H  2014.0
6   7    G  New_city  2017.0
```

> ### Removing duplicates with drop_duplicates()

- inplace=True : Make changes permanent to the original df.
- ignore_index : If True, starts indexing from 0 and original indes otherwise on 0.

```
data = {
    'Id': [1,2,1,3,2,2,3],
    'Name': ['A','B','A','D','B','B','D'],
    'City': ['P','O','P','S','O','O','R'],
    'Year': [2013, 2023, 2013, 2025, 2023, 2023, 2017]
}

df = pd.DataFrame(data)
print(df)
# DF contains many duplicate entries
```

```
   Id Name City  Year
0   1    A    P  2013
1   2    B    O  2023
2   1    A    P  2013
3   3    D    S  2025
4   2    B    O  2023
5   2    B    O  2023
6   3    D    R  2017
```

```
# Subset : Take list of columns, from which to consider duplicates
print(df.drop_duplicates(subset=['Id']))
# print(df.drop_duplicates())
# Returns unique rows only.
```

```
   Id Name City  Year
0   1    A    P  2013
1   2    B    O  2023
3   3    D    S  2025
```

keep= keep is to control how to consider duplicate value. It has only three distinct value and default is 'first'.

- If 'first', it considers first value as unique and rest of the same values as duplicate.
- If 'last', it considers last value as unique and rest of the same values as duplicate.
- If False, it consider all of the same values as duplicates

```
print(df.drop_duplicates(keep=False))
```

```
   Id Name City  Year
3   3    D    S  2025
6   3    D    R  2017
```

Data Transformation and Feature Engineering

> **Applying functions to DataFrame columns using apply() and map():**

- Pandas.apply() allows to pass a function and apply it on every single value of the Pandas series. i.e. all values in the given column
- **Syntax:**

```
s.apply(func, convert_dtype=True, args=())
```

  - func : Takes a function that needs to applied to all values in the Series.
  - convert_dtype: If True, Convert dtype as per the function's operation.
  - args=(): Additional arguments to pass to function instead of series.

```
data = {
    'Name': ['A','B','C','D','E','F'],
    'Marks': [56,78,90,34,62,78],
    'City': ['P','O','P','S','O','O']
}

df = pd.DataFrame(data)

def give(num):
    if num>60:
        return 'Pass'
    else:
        return 'Fail'

print(df['Marks'].apply(give, convert_dtype=True))
```

```
0    Fail
1    Pass
2    Pass
3    Fail
4    Pass
```

```
5    Pass
Name: Marks, dtype: object
```

> **map():**

- pandas.map() is used to map values from two series having one column same.

```
s1 = pd.Series([0,1,2,3,4,5])
s2 = pd.Series([21,24,26,32,34,31])

print(s1.map(s2))
```

```
0    21
1    24
2    26
3    32
4    34
5    31
dtype: int64
```

Working with Text Data

> **Handling text data in pandas (e.g., string methods):**

- Series provides various methods, for operating on Strings, using the str attribute.

1. **str.lower():** Method to convert a string's characters to lowercase.
2. **str.upper():** Method to convert a string's characters to uppercase.
3. **str.find():** Method is used to search a substring in each string present in a series.
4. **str.rfind():** Method is used to search a substring in each string present in a series from the Right side.
5. **str.findall():** Method is also used to find substrings or separators in each string in a series.
6. **str.isalpha():** Method is used to check if all characters in each string in series are alphabetic(a-z/A-Z).
7. **str.isdecimal():** Method is used to check whether all characters in a string are decimal.
8. **str.title():** Method to capitalize the first letter of every word in a string.
9. **str.len():** Method returns a count of the number of characters in a string.
10. **str.replace():** Method replaces a substring within a string with another value that the user provides.
11. **str.contains():** Method tests if pattern or regex is contained within a string of a Series or Index.
12. **str.extract():** Extract groups from the first match of regular expression pattern.
13. **str.startswith():** Method tests if the start of each string element matches a pattern.
14. **str.endswith():** Method tests if the end of each string element matches a pattern.
15. **str.isdigit():** Method is used to check if all characters in each string in series are digits.
16. **str.lstrip():** Method removes whitespace from the left side (beginning) of a string.

17. **str.rstrip():** Method removes whitespace from the right side (end) of a string.
18. **str.strip():** Method to remove leading and trailing whitespace from string.
19. **str.split():** Method splits a string value, based on an occurrence of a user-specified value.
20. **str.join():** Method is used to join all elements in list present in a series with passed delimiter.
21. **str.cat():** Method is used to concatenate strings to the passed caller series of string.
22. **str.repeat():** Method is used to repeat string values in the same position of passed series itself.
23. **str.get():** Method is used to get element at the passed position.
24. **str.partition():** Method splits the string only at the first occurrence unlike str.split().
25. **str.rpartition():** Method splits string only once and that too reversely. It works in a similar way like str.partition() and str.split()
26. **str.pad():** Method to add padding (whitespaces or other characters) to every string element in a series.
27. **str.swapcase():** Method to swap case of each string in a series.

```
data = {
    'Name': ['Abc','Bob','Cat','Dog','Ele','Fog'],
    'Marks': [56,78,90,34,62,78],
    'City': ['Pa','OpD','PsT','SpY','On','Om']
}

df = pd.DataFrame(data)

print(df)
```

```
  Name  Marks City
0  Abc     56   Pa
1  Bob     78  OpD
2  Cat     90  PsT
3  Dog     34  SpY
4  Ele     62   On
5  Fog     78   Om
```

```
# 1. str.lower() - Convert the string to lowercase chars
df['Name'] = df['Name'].str.lower()
print(df)
```

```
  Name  Marks City
0  abc     56   Pa
1  bob     78  OpD
2  cat     90  PsT
3  dog     34  SpY
4  ele     62   On
5  fog     78   Om
```

```
# 2. str.upper() - Convert the string to uppercase chars
df['City'] = df['City'].str.upper()
print(df)
```

```
  Name  Marks City
0  abc     56   PA
1  bob     78  OPD
```

```
2  cat     90  PST
3  dog     34  SPY
4  ele     62   ON
5  fog     78   OM
```

> ### Regular expressions for pattern matching:

```
import re

data = {
    'Name': ['Abc','Bob','Cat','Dog','Ele','Fog'],
    'Marks': [56,78,90,34,62,78],
    'City': ['Pa','OpD','PsT','SpY','On','Om']
}

df = pd.DataFrame(data)

pattern = '[O].*'

print(df[df.City.str.match(pattern)])
```

```
  Name  Marks City
1  Bob     78  OpD
4  Ele     62   On
5  Fog     78   Om
```