Introduction

> In data science, you will usually have missing data.

> The most time consuming part of a data science project is data cleaning and preparation.

> For example, an industrial application with sensors will have sensor data that is missing on certain days.

When and Why Is Data Missed?

- Let us consider an online survey for a product.

> Many a times, people do not share all the information related to them.

> Few people share their experience, but not how long they are using the product; few people share how long they are using the product, their experience but not their contact information.

> Thus, in some or the other way a part of data is always missing, and this is very common in real time.

> > Over time, individuals and businesses accumulate a lot of personal information! Eventually, information becomes outdated. For example, over 10 years you may change your address, or your name, and then change your address again!

> > Without cleaning and cleansing in the data science lifecycle or as a routine activity, the code for any purpose would simply not work.

What is Data Preparation?

> **Data preparation (also referred to as "data preprocessing") is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.**

What is Data Cleaning?

1. Data cleaning is the process of dealing with disordered data.
2. It is **the process of changing or eliminating missing or incomplete, duplicate, invalid or inconsistent data in any dataset to improve the data quality**.

> - The major requirement of data cleaning services is **to construct uniform and standardized datasets.**
> - **It improves the quality and accuracy of the data being fed to the machine learning algorithms that are aimed to solve a data science problem.**
> - **There's no such absolute way to describe the precise steps in the data cleaning process because the processes may vary from dataset to dataset.**

↳ *1 cell hidden*

Why Data Cleaning is Essential? (sometimes also known as data cleansing or data wrangling)

- Having clean data will ultimately increase overall productivity and permit the very best quality information in a decision-making problem.

1. Error-Free Data:

When multiple sources of data are combined there may be chances of so much error. Through Data Cleaning, errors can be removed from data.

> (Having clean data which is free from wrong and garbage values can help in performing analysis faster as well as efficiently.)

2. Data Quality:

Data Quality is the measure of the condition of dataset is according to the requirement or not.

> > For example, if we have imported phone numbers data of different customers, and in some places, we have added email addresses of customers in the data.
>
> > Particular types of data have unique restrictions. Data cleaning will help us simplify this process and avoid useless data values.

3. Accurate and Efficient:

Ensuring the data is close to the correct values. Even if the data is authentic and correct, it doesn't mean the data is accurate.

> > For example, the address of a customer is stored in the specified format, maybe it doesn't need to be in the right one. The email has an additional character or value that makes it incorrect or invalid.

4. Complete Data:

Completeness is the degree to which we should know all the required values.

> > It is nearly impossible to have all the info we need. Only known facts can be entered. We can try to complete data by re-doing the data gathering activities like approaching the clients again, re-interviewing people, etc.

> For example, we might need to enter every customer's contact information. But a number of them might not have email addresses. This may lead to incomplete data.

5. **Maintains Data Consistency:**

Data consistency refers to whether the same data kept at different places do or do not match.

> We can measure consistency by comparing two similar systems.

> For example, a customer's age might be 25, which is a valid value and also accurate, but it is also

Data Cleaning Methods:

A] Handling Missing Data

> Handling missing values is an essential part of data cleaning and preparation process because almost all data in real life comes with some missing values.

> There are several ways to find the missing or null values present in data. To handle the missing data:

1. Finding missing elements.
2. Filling missing values with some other value.

> It would not make sense to drop the column as that would throw away that metric for all rows. So, let's look at how to handle the missing data.

Let us create a DataFrame with Nan values (Missing Values)

> In Python, NumPy **NAN stands for Not a Number**.

> NaN is defined as a substitute for declaring values that are missing values in a DataFrame.

> It is initialized using numpy.nan or numpy.NaN.

> Missing values in a DataFrame are represented by NaN as a placeholder which is a floating-point number.

```
#import Libraries
import pandas as pd
import numpy as np

#Create a Sample DataFrame with missing values.
df = pd.DataFrame({
    'Name': ['Nik', 'Nik', 'Evan', 'Kyra', np.NaN],
    'Age': [33, 32, 40, 57, np.NaN],
    'Location': ['Toronto', 'London', 'New York', np.NaN, np.NaN] })

#Display the DataFrame
df
```

	Name	Age	Location
0	Nik	33.0	Toronto

```
#Write the DataFrame in a new csv file using df.to_csv()

#This will save the above DataFrame into your drive with the file name "w.csv"

df.to_csv("/content/drive/MyDrive/w.csv")
```

4	NaN	NaN	NaN

1) Finding missing elements in a DataFrame

There are four ways to find the null values, if present, in the dataset.

a) **Using isnull( ) function:** It is used to know the number of null values in a dataset. The below syntax returns true wherever the value is null in the dataset.

> Syntax:
>
> > data.isnull()

```
df.isnull()
```

	Name	Age	Location
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	True
4	True	True	True

b) **Using isna( ) function:** It is used to know the number of Na values in the dataset. It distplays the same result as isnull().

> Syntax:
>
> > data.isna()

```
df.isna()
```

	Name	Age	Location
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	True
4	True	True	True

```

```

b) **isnull().sum():** This function will give an integer value of the number of null values present in the dataset.

i.e. returns a Series containing the counts of missing items in each column. i.e. it gives the column-wise sum of the null values present in the dataset.

Note:

We can use, isna().sum(): It gives same output as isnull().sum()

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

```
Name        1
Age         1
Location    2
dtype: int64
```

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

```
Name        1
Age         1
Location    2
dtype: int64
```

2) Filling missing values in a DataFrame

fillna( ): This function will replace the null values in a DataFrame with the specified values.

```
# fillna() on all columns
a = df.fillna(0)

print(a)
```

```
    Name   Age  Location
0   Nik  33.0   Toronto
1   Nik  32.0    London
2  Evan  40.0  New York
3  Kyra  57.0         0
4     0   0.0         0
```

```
# fillna() on one column
b =  df['Name'].fillna(0)

print(b)
```

```
0     Nik
1     Nik
2    Evan
3    Kyra
4       0
Name: Name, dtype: object
```

```
# fillna() on multiple columns
c =  df[['Name','Age']].fillna(0)

print(c)
```

```
    Name   Age
0   Nik  33.0
1   Nik  32.0
2  Evan  40.0
3  Kyra  57.0
4     0   0.0
```

```
# fillna() on multiple columns with different values
d = df.fillna({'Name': 'Someone', 'Age': 25, 'Location': 'USA'})

print(d)
```

```
       Name   Age  Location
0      Nik  33.0   Toronto
1      Nik  32.0    London
2     Evan  40.0  New York
3     Kyra  57.0       USA
4  Someone  25.0       USA
```

3) Renaming Columns in a DataFrame

Renaming Single Columns

Pandas DataFrame.rename( ) accepts a dict(dictionary) as a parameter for columns you wanted to rename.

- So you just pass a dict with key-value pair.
- The key is an existing column you would like to rename and value would be your preferred new column name.

```
# Rename a Single Column
i = df.rename(columns = {'A':'X'})

print(i)
```

```
      Name   Age  Location
0      Nik  33.0   Toronto
1      Nik  32.0    London
2     Evan  40.0  New York
3     Kyra  57.0       NaN
4      NaN   NaN       NaN
```

Rename Multiple Columns

You can also use the same approach to rename multiple columns of Pandas DataFrame.

All you need to specify multiple columns you wanted to rename in a dictionary mapping manner.

```
# Rename multiple columns
j = df.rename(columns = {'A':'P','B':'Q','C':'R'})

print(j)
```

```
      Name   Age  Location
0      Nik  33.0   Toronto
1      Nik  32.0    London
2     Evan  40.0  New York
3     Kyra  57.0       NaN
4      NaN   NaN       NaN
```

B] Replacing Values

Pandas dataframe.replace( ) method is used to find a value on a DataFrame and replace it with another value on all columns & rows.

To replace values in a given column of pandas DataFrame, first, select the column you wanted to update values and use replace() method.

```
# Replace column value
e = df.replace('Nik','Nikon')
print(e)
```

```
      Name   Age  Location
0  Nikon  33.0   Toronto
1  Nikon  32.0    London
2   Evan  40.0  New York
3   Kyra  57.0       NaN
4    NaN   NaN       NaN
```

To find multiple values from a list and replace them with other values in a list.

```
f = df['Location'].replace('USA','California')

print(f)
```

```
0     Toronto
1      London
2    New York
3         NaN
4         NaN
Name: Location, dtype: object
```

C)Finding and Removing duplicate values in a DataFrame

i) duplicated( )

> The pandas.DataFrame.duplicated() method is used to find duplicate rows in a DataFrame. It returns a boolean series which identifies whether a row is duplicate or unique.

Use the subset parameter to specify if any columns should not be considered when looking for duplicates.

Syntax:

> **dataframe.duplicated(subset, keep)**

Parameters:

- ***subset:*** *Subset takes a column or list of column label. It's default value is none. After passing columns, it will consider them only for duplicates.*
- ***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*

```
import pandas as pd

#create DataFrame
df = pd.DataFrame({'team': ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'],
                   'points': [10, 10, 12, 12, 15, 17, 20, 20],
                   'assists': [5, 5, 7, 9, 12, 9, 6, 6]})
```

```
#view DataFrame
print(df)
```

```
  team  points  assists
0    A      10        5
1    A      10        5
2    A      12        7
3    A      12        9
4    B      15       12
5    B      17        9
6    B      20        6
7    B      20        6
```

```
#identify duplicate rows
df.duplicated()
```

```
0    False
1     True
2    False
3    False
4    False
5    False
6    False
7     True
dtype: bool
```

```
# By using drop_duplicates()

g = df.Name.duplicated()

print(g)
```

```
0    False
1     True
2    False
3    False
4    False
Name: Name, dtype: bool
```

```
#Display duplicate rows

a = df[df.duplicated()]

print(a)
```

```
  team  points  assists
1    A      10        5
7    B      20        6
```

```
#Find Duplicate Rows Across Specific Columns

#identify duplicate rows across 'team' and 'points' columns
a = df[df.duplicated(['team', 'points'])]

#view duplicate rows
print(a)
```

```
  team  points  assists
1    A      10        5
3    A      12        9
7    B      20        6
```

```
#To keep 2nd duplicate row, we set 'keep' = 'last'
duplicateRows = df[df.duplicated(keep='last')]

#view duplicate rows
print(duplicateRows)
```

```
  team  points  assists
0    A      10        5
6    B      20        6
```

ii) drop_duplicates( )

Pandas drop_duplicates() function helps the user to eliminate all the unwanted or duplicate rows of the Pandas Dataframe.

Syntax:

> **dataframe.drop_duplicates(subset, keep)**

Parameters:

- ***subset:*** *Subset takes a column or list of column label. It's default value is none. After passing columns, it will consider them only for duplicates.*
- ***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*

```
h = df.drop_duplicates(subset="assists", keep=False)

print(h)
```

```
  team  points  assists
2    A      12        7
4    B      15       12
```

D) Drop columns with Missing Data

dropna( ) is used to drop/remove missing values (NaN) from rows and columns.

Syntax:

> **df.dropna( )**

Parameters:

```
axis=0,         # axis takes int or string value for rows/columns. Input can be 0 or 1 for Integer and 'index' or 'columns' for S

how='any',      # The how parameter enables you to specify "how" the method will decide to drop a row from the DataFrame. There a
  * any: If how = 'any', dropna will drop the row if any of the values in that row are missing.
  * all: If how = 'all', dropna will drop the row only if all of the values in that row are missing.

thresh=None,    # thresh takes integer value which tells minimum amount of na values to drop.

subset=None,    # Which rows/columns to consider

inplace=False   # It is a boolean which makes the changes in data frame itself if True.
```

```
#Create a DataFrame with missing values:

import pandas as pd
import numpy as np

df=pd.DataFrame( {'A':[100, 90, np.nan, 95],
        'B': [30, 45, 56, np.nan],
        'C':[np.nan, 40, 80, 98]})

print(df)
```

```
     A     B     C
0  100.0  30.0   NaN
1   90.0  45.0  40.0
2    NaN  56.0  80.0
3   95.0   NaN  98.0
```

```
# Default drop rows that contains nan values
m = df.dropna()

print(m)
```

```
     A     B     C
1  90.0  45.0  40.0
```

```
# Default drop rows that contains nan values
m = df.dropna(axis=0)

print(m)
```

```
     A     B     C
1  90.0  45.0  40.0
```

```
# Drop all columns with NaN values
n =df.dropna(axis=1)

print(n)
```

```
Empty DataFrame
Columns: []
Index: [0, 1, 2, 3]
```

```
# Drop rows that has NaN values on selected columns
o = df.dropna(subset=['A','B'])

print(o)
```

```
     A     B     C
0  100.0  30.0   NaN
1   90.0  45.0  40.0
```

```
#Create a DataFrame with missing values:

import pandas as pd
import numpy as np

df=pd.DataFrame( {'A':[100, np.nan, np.nan, 95],
        'B': [np.nan, np.nan, np.nan, np.nan],
        'C':[np.nan, np.nan, 80, 98],
        'D':[50, 40, np.nan, 98],
        'E':[50, 40, 80, 98]})

print(df)
```

```
     A   B     C     D   E
0  100.0 NaN   NaN  50.0  50
1    NaN NaN   NaN  40.0  40
```

```
     2    NaN NaN  80.0   NaN  80
     3   95.0 NaN  98.0  98.0  98
```

```
#thresh parameter decides the minimum number of non-NAN values needed in a "ROW" not to drop.

p = df.dropna(thresh=3,axis=0)

print(p)
```

```
          A   B     C     D   E
     0  100.0 NaN   NaN  50.0  50
     3   95.0 NaN  98.0  98.0  98
```

```
#thresh parameter decides the minimum number of non-NAN values needed in a "ROW" not to drop.

p = df.dropna(thresh=3,axis=1)

print(p)
```

```
          D   E
     0  50.0  50
     1  40.0  40
     2   NaN  80
     3  98.0  98
```

```
# Drop columns that has atleast one Nan Values

k = df.dropna(how='any', axis=1)

print(k)
```

```
         E
     0  50
     1  40
     2  80
     3  98
```

```
# Drop columns that has all Nan Values
l = df.dropna(how='all', axis = 1)

print(l)
```

```
          A     C     D   E
     0  100.0   NaN  50.0  50
     1    NaN   NaN  40.0  40
     2    NaN  80.0   NaN  80
     3   95.0  98.0  98.0  98
```