IOTA ACADEMY

Become Job Ready...

Missing Data

- Missing data is defined as the values or data that is not stored (or not present) for some variable/s in the given dataset.

Let's show a few convenient methods to deal with Missing Data in pandas:

- `NaN` is the default missing value marker for reasons of computational speed and convenience.
- In many cases, however, the Python None will arise and we wish to also consider that "missing" or "not available" or "NA".

In [40]:
```
import numpy as np
import pandas as pd
```

In [ ]:

In [41]:
```
# Import the dataset
df = pd.read_csv("salary.csv")
df
# Below is a sample of the missing data
```

Out[41]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	NaN
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	NaN
7	8	Male	14000.0	Australia	NaN
8	9	Female	15000.0	NaN	NaN
9	10	Male	33000.0	Australia	NaN

1.detecting missing values:-

pandas provides the **isna()** ,**isnull()** and **notna()** functions

a) isna()/isnull()

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy.NaN, gets mapped to True values.

Both isna() and isnull() do the same thing

- **Syntax:** dataframe.isna()

In [42]:
```
# check in individual column
df["Company"].isna()
```

Out[42]:
```
0    False
1     True
2    False
3    False
4    False
5    False
6     True
7     True
8     True
9     True
Name: Company, dtype: bool
```

In [43]:
```
# You can also count the number of missing values present inside each column using isna().sum()
df["Company"].isna().sum()
# There are 5 NaN values in the "Company" column
```

Out[43]:
```
5
```

in below example isna() returns a data frame consisting of true and false values. A true value indicates data is null or missing, while a false one indicates that data is not null and not missing.

In [44]:
```
# Check in the whole DataFrame
df.isna()
```

Out[44]:

	ID	Gender	Salary	Country	Company
0	False	False	False	False	False
1	False	False	False	False	True
2	False	False	False	False	False
3	False	True	True	False	False
4	False	False	True	False	False
5	False	False	False	True	False
6	False	True	False	False	True
7	False	False	False	False	True
8	False	False	False	True	True
9	False	False	False	False	True

In [45]:
```
df.isna().sum()  # Returns the number of missing values in each column.
```

Out[45]:
```
ID         0
Gender     2
Salary     2
Country    2
Company    5
dtype: int64
```

b) notna()

Detect existing (non-missing) values.

Return a boolean same-sized object indicating if the values are not NA. Non-missing values get mapped to True.

- **Syntax:** DataFrame.notna()

In [46]:
```
df["Salary"].notna()
```

Out[46]:
```
0     True
1     True
2     True
3    False
4    False
5     True
6     True
7     True
8     True
9     True
Name: Salary, dtype: bool
```

In [47]:
```
df["Salary"].notna().sum()
```

Out[47]:
```
8
```

Datetimes

For datetime64[ns] types, NaT represents missing values.

In [48]:
```
df1= df.copy()
df1
```

Out[48]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	NaN
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	NaN
7	8	Male	14000.0	Australia	NaN
8	9	Female	15000.0	NaN	NaN
9	10	Male	33000.0	Australia	NaN

In [49]:
```
df1["Hiring Date"] = pd.Timestamp("20221009")
```

In [50]:
```
df1
```

Out[50]:

	ID	Gender	Salary	Country	Company	Hiring Date
0	1	Male	15000.0	India	Google	2022-10-09
1	2	Female	45000.0	China	NaN	2022-10-09
2	3	Female	25000.0	India	Google	2022-10-09
3	4	NaN	NaN	Australia	Google	2022-10-09
4	5	Male	NaN	India	Google	2022-10-09
5	6	Male	54000.0	NaN	Alibaba	2022-10-09
6	7	NaN	74000.0	China	NaN	2022-10-09
7	8	Male	14000.0	Australia	NaN	2022-10-09
8	9	Female	15000.0	NaN	NaN	2022-10-09
9	10	Male	33000.0	Australia	NaN	2022-10-09

In [51]:
```
# Inserting missing data
df1.iloc[[0,4],[5]] = np.nan
df1
```

Out[51]:

	ID	Gender	Salary	Country	Company	Hiring Date
0	1	Male	15000.0	India	Google	NaT
1	2	Female	45000.0	China	NaN	2022-10-09
2	3	Female	25000.0	India	Google	2022-10-09
3	4	NaN	NaN	Australia	Google	2022-10-09
4	5	Male	NaN	India	Google	NaT
5	6	Male	54000.0	NaN	Alibaba	2022-10-09
6	7	NaN	74000.0	China	NaN	2022-10-09
7	8	Male	14000.0	Australia	NaN	2022-10-09
8	9	Female	15000.0	NaN	NaN	2022-10-09
9	10	Male	33000.0	Australia	NaN	2022-10-09

2.Filling missing values:

fillna()

The fillna() method replaces the NULL values with a specified value.

- **Syntax:** dataframe.fillna(value, method, axis, inplace, limit, downcast)

  The fillna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the fillna() method does the replacing in the original DataFrame instead.

In [52]:
```
df
```

Out[52]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	NaN
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	NaN
7	8	Male	14000.0	Australia	NaN
8	9	Female	15000.0	NaN	NaN
9	10	Male	33000.0	Australia	NaN

In [53]:
```
# Replace NA with a scalar value
df.fillna(0)  # All missing values are filled with zeros
```

Out[53]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	0
2	3	Female	25000.0	India	Google
3	4	0	0.0	Australia	Google
4	5	Male	0.0	India	Google
5	6	Male	54000.0	0	Alibaba
6	7	0	74000.0	China	0
7	8	Male	14000.0	Australia	0
8	9	Female	15000.0	0	0
9	10	Male	33000.0	Australia	0

In [54]:
```
df # inplace = False, Changes will not appear in the original DataFrame.
```

Out[54]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	NaN
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	NaN
7	8	Male	14000.0	Australia	NaN
8	9	Female	15000.0	NaN	NaN
9	10	Male	33000.0	Australia	NaN

In [55]:
```
df["Company"].fillna("missing",inplace=True) # We can also make changes in any individual column too.
```

In [56]:
```
df
```

Out[56]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	missing
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	missing
7	8	Male	14000.0	Australia	missing
8	9	Female	15000.0	NaN	missing
9	10	Male	33000.0	Australia	missing

In [57]:
```
df["Salary"].fillna(df["Salary"].mean())  # fill with mean value
```

Out[57]:
```
0    15000.0
1    45000.0
2    25000.0
3    34375.0
4    34375.0
5    54000.0
6    74000.0
7    14000.0
8    15000.0
9    33000.0
Name: Salary, dtype: float64
```

Fill gaps forward or backward

Method **Action**

pad / ffill - Fill values forward

bfill / backfill - Fill values backward

In [58]:
```
df
```

Out[58]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	missing
2	3	Female	25000.0	India	Google
3	4	NaN	NaN	Australia	Google
4	5	Male	NaN	India	Google
5	6	Male	54000.0	NaN	Alibaba
6	7	NaN	74000.0	China	missing
7	8	Male	14000.0	Australia	missing
8	9	Female	15000.0	NaN	missing
9	10	Male	33000.0	Australia	missing

In [59]:
```
df.fillna(method="ffill")
```

Out[59]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	missing
2	3	Female	25000.0	India	Google
3	4	Female	25000.0	Australia	Google
4	5	Male	25000.0	India	Google
5	6	Male	54000.0	India	Alibaba
6	7	Male	74000.0	China	missing
7	8	Male	14000.0	Australia	missing
8	9	Female	15000.0	Australia	missing
9	10	Male	33000.0	Australia	missing

In [60]:
```
df.fillna(method="bfill")
```

Out[60]:

	ID	Gender	Salary	Country	Company
0	1	Male	15000.0	India	Google
1	2	Female	45000.0	China	missing
2	3	Female	25000.0	India	Google
3	4	Male	54000.0	Australia	Google
4	5	Male	54000.0	India	Google
5	6	Male	54000.0	China	Alibaba
6	7	Male	74000.0	China	missing
7	8	Male	14000.0	Australia	missing
8	9	Female	15000.0	Australia	missing
9	10	Male	33000.0	Australia	missing

3. Dropping axis labels with missing data:

dropna()

- **Syntax:** dataframe.dropna(axis, how, thresh, subset, inplace)

  ```
  - You may wish to simply exclude labels from a data set which refer to missing data. To do this, use dropna():
  ```

In [61]:
```
df = pd.DataFrame({"name": ['Alfred', 'Batman', 'Catwoman'],
                   "toy": [np.nan, 'Batmobile', 'Bullwhip'],
                   "born": [pd.NaT, pd.Timestamp("1940-04-25"),
                            pd.NaT]})

df
```

Out[61]:

	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

In [62]:
```
# Drop the rows where at least one element is missing.

df.dropna()   # default axis is 0 (rows)
              # default inplace = False
```

Out[62]:

	name	toy	born
1	Batman	Batmobile	1940-04-25

In [63]:
```
# Drop the columns where at least one element is missing.
df.dropna(axis=1)
```

Out[63]:

	name
0	Alfred
1	Batman
2	Catwoman

In [64]:
```
# Example
df.dropna(axis="columns")
```

Out[64]:

	name
0	Alfred
1	Batman
2	Catwoman

In [65]:
```
# Drop the rows where all elements are missing.
df.dropna(how="all")
```

Out[65]:

	name	toy	born
0	Alfred	NaN	NaT
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

In [66]:
```
# Keep only the rows with at least 2 non-NA values.

df.dropna(thresh=2)
```

Out[66]:

	name	toy	born
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

In [67]:
```
# Define in which columns to look for missing values.

df.dropna(subset=['name', 'toy'])
```

Out[67]:

	name	toy	born
1	Batman	Batmobile	1940-04-25
2	Catwoman	Bullwhip	NaT

Great Job!