In [4]:
```
import pandas as pd
import numpy as np
```

In [5]:
```
dict1= {'Names':['Aravind','Samar',np.nan,'Siri'],
        'Age':[np.nan,21,32,43],
        'City':['Hyd','Blr','Chennai',np.nan]}

dict1
```

Out[5]:
```
{'Names': ['Aravind', 'Samar', nan, 'Siri'],
 'Age': [nan, 21, 32, 43],
 'City': ['Hyd', 'Blr', 'Chennai', nan]}
```

In [7]:
```
d1=pd.DataFrame(dict1)
```

In [8]:
```
d1.dtypes
```

Out[8]:
```
Names     object
Age      float64
City      object
dtype: object
```

In [9]:
```
dict2= {'Names':['Aravind','Samar',None,'Siri'],
        'Age':[np.nan,21,32,43],
        'City':['Hyd','Blr','Chennai',None]}

pd.DataFrame(dict2)
```

Out[9]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	None	32.0	Chennai
3	Siri	43.0	None

In [10]:
```
d1.isnull()
```

Out[10]:

	Names	Age	City
0	False	True	False
1	False	False	False
2	True	False	False
3	False	False	True

In [11]:
```
d1.isnull().sum()
```

Out[11]:
```
Names    1
Age      1
City     1
dtype: int64
```

In [17]:
```
d1.isnull().sum()*100/len(d1)
```

Out[17]:
```
Names    25.0
Age      25.0
City     25.0
dtype: float64
```

- np.nan: not a number this applicable for numerical columns only

- Generally data has some empty rows
- It is a data problem
- If you read that kind of data you will see Null
- You need to understand it is a data corrupted or really a Null values are there

Method-1

Fill with some random value

Method name: fill na

In [18]:
```
d1.fillna(40)
```

Out[18]:

	Names	Age	City
0	Aravind	40.0	Hyd
1	Samar	21.0	Blr
2	40	32.0	Chennai
3	Siri	43.0	40

Method-2

- We can fill the values w.r.t columns also

In [21]:
```
d1['Age'].fillna(40)

# it will not updated
# inplace=False
```

Out[21]:
```
0    40.0
1    21.0
2    32.0
3    43.0
Name: Age, dtype: float64
```

In [22]:
```
d1
```

Out[22]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	NaN	32.0	Chennai
3	Siri	43.0	NaN

In [23]:
```
d1['Age'].fillna(40,inplace=True)
```

In [24]:
```
d1
```

Out[24]:

	Names	Age	City
0	Aravind	40.0	Hyd
1	Samar	21.0	Blr
2	NaN	32.0	Chennai
3	Siri	43.0	NaN

In [25]:
```
dict1= {'Names':['Aravind','Samar',np.nan,'Siri'],
        'Age':[np.nan,21,32,43],
        'City':['Hyd','Blr','Chennai',np.nan]}

d1=pd.DataFrame(dict1)
```

Method-3

- bfill
- ffill
- pad
- backfill

In [27]:
```
d1.fillna(method='bfill')
# It will fill with below value or next value
# Column1: Names index 3 is missing values it is filled with index 4 value
# Column2: Age index 1 is missing values it is filled with index 2 value
# Column3: City index 4 is missing values it should be fill with next value
#          but we dont have next value
```

```
C:\Users\omkar\AppData\Local\Temp\ipykernel_30684\1914124427.py:1: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  d1.fillna(method='bfill')
```

Out[27]:

	Names	Age	City
0	Aravind	21.0	Hyd
1	Samar	21.0	Blr
2	Siri	32.0	Chennai
3	Siri	43.0	NaN

In [28]:
```
d1.fillna(method='bfill',axis=1)
# axis=1 means columns
# so Nan values fill with next column values
```

```
C:\Users\omkar\AppData\Local\Temp\ipykernel_30684\981496262.py:1: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  d1.fillna(method='bfill',axis=1)
```

Out[28]:

	Names	Age	City
0	Aravind	Hyd	Hyd
1	Samar	21.0	Blr
2	32.0	32.0	Chennai
3	Siri	43.0	NaN

In [29]:
```
d1
```

Out[29]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	NaN	32.0	Chennai
3	Siri	43.0	NaN

In [30]:
```
d1.fillna(method='backfill')
```

```
C:\Users\omkar\AppData\Local\Temp\ipykernel_30684\907203477.py:1: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  d1.fillna(method='backfill')
```

Out[30]:

	Names	Age	City
0	Aravind	21.0	Hyd
1	Samar	21.0	Blr
2	Siri	32.0	Chennai
3	Siri	43.0	NaN

In [31]:
```
d1.fillna(method='pad')
```

```
C:\Users\omkar\AppData\Local\Temp\ipykernel_30684\3282249208.py:1: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  d1.fillna(method='pad')
```

Out[31]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	Samar	32.0	Chennai
3	Siri	43.0	Chennai

In [32]:
```
d1.fillna(method='ffill')
```

```
C:\Users\omkar\AppData\Local\Temp\ipykernel_30684\4088926743.py:1: FutureWarning:
DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  d1.fillna(method='ffill')
```

Out[32]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	Samar	32.0	Chennai
3	Siri	43.0	Chennai

- backfill and bill fill with Next value
- pad and ffill fill with previous value
- but it will change based on axis

Method-4

- mean
  - Numerical values can fill with mean value
  - but mean affect by outliers
  - if we dont have outliers it is best one
- medain
  - Numerical values can fill with median value
  - we know that median value does not affect with outliers
  - so if outliers are there we can go with median
- mode

- Mode is useful for categorical data

In [35]:
```
dict1= {'Names':['Aravind','Samar',np.nan,'Siri'],
        'Age':[np.nan,21,32,43],
        'City':['Hyd','Blr','Chennai',np.nan]}

d1=pd.DataFrame(dict1)
d1
```

Out[35]:

	Names	Age	City
0	Aravind	NaN	Hyd
1	Samar	21.0	Blr
2	NaN	32.0	Chennai
3	Siri	43.0	NaN

In [34]:
```
# we already know we can fill with some values based on specific column using fi
# d1.fillna(<random_number>)
# d1 represnts all the columns
age_mean=d1['Age'].mean()
d1['Age'].fillna(age_mean)
```

Out[34]:
```
0    32.0
1    21.0
2    32.0
3    43.0
Name: Age, dtype: float64
```

In [36]:
```
age_median=d1['Age'].median()
d1['Age'].fillna(age_median)
```

Out[36]:
```
0    32.0
1    21.0
2    32.0
3    43.0
Name: Age, dtype: float64
```

In [38]:
```
d1['City'].mode()
# In this use case we are getting mode equal
# But in realtime we will get clear mode value
```

Out[38]:
```
0        Blr
1    Chennai
2        Hyd
Name: City, dtype: object
```

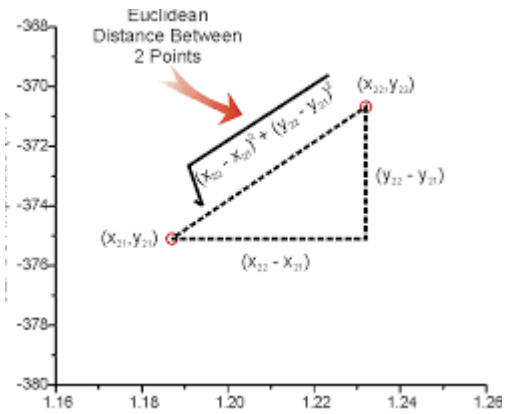
- Fill with random value
- Fill with random value on speecific value
- Fill with methods
  - bfill
  - ffill

    - pad
    - backfill
- Fill with
    - Mean
    - Medain
    - Mode

Method-5

KNN imputer

- KNN: K neartest Neighbours
- K is a hyper paramter means user can choose
- It is distance metric: **Euclidian distance**
- KNN imputer will take the mean of the neigbours value
- The neibhours value can provided by using value =k
- It is under sklearn package
- Under sklearn we have Imputer method
- Under imputer method we have KnnImputer
- Applicable only for Numerical data



In [40]:
```
from sklearn.impute import KNNImputer
KI=KNNImputer()
KI.fit_transform(d1[['Age']])
```

Out[40]:
```
array([[32.],
       [21.],
       [32.],
       [43.]])
```

In [ ]: