In [3]:

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

In [4]:

```
dict1={'Name': ['Ramesh','Suresh','Sathish',np.nan],
      'Age':[30,33,np.nan,36],
      'City':[np.nan,'Hyd','Hyd','Blr']}
```

In [5]:

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

Out[5]:

	Name	Age	City
0	Ramesh	30.0	NaN
1	Suresh	33.0	Hyd
2	Sathish	NaN	Hyd
3	NaN	36.0	Blr

In [11]:

```
data.to_csv('data1.csv',index=False)
```

isnull

In [13]:

```
data.isnull()
```

Out[13]:

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

In [15]:

```
data.isnull().sum()
```

Out[15]:

```
Name     1
Age      1
City     1
dtype: int64
```

dropna

- drop the null values data based on rows and columns

In [17]:

```
# I want to exatrct the data with out Null values
data.dropna()
```

Out[17]:

	Name	Age	City
1	Suresh	33.0	Hyd

In [10]:

```
data
```

Out[10]:

	Name	Age	City
0	Ramesh	30.0	NaN
1	Suresh	33.0	Hyd
2	Sathish	NaN	Hyd
3	NaN	36.0	Blr

In [11]:

```
data.dropna(axis=1)
```

Out[11]:

0
1
2
3

In [19]:

```
dict1={'Name': ['Ramesh','Suresh','Sathish',np.nan],
       'Age':[30,33,np.nan,36],
       'City':[np.nan,'Hyd','Hyd','Blr'],
       'Company':['Google','Microsoft','Apple','Meta']}
data1=pd.DataFrame(dict1)
data1
```

Out[19]:

	Name	Age	City	Company
0	Ramesh	30.0	NaN	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	NaN	Hyd	Apple
3	NaN	36.0	Blr	Meta

In [21]:

```
data1.dropna(axis=1)
```

Out[21]:

	Company
0	Google
1	Microsoft
2	Apple
3	Meta

Fillna

In [25]:

```
data1.fillna(30)
```

Out[25]:

	Name	Age	City	Company
0	Ramesh	30.0	30	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	30.0	Hyd	Apple
3	30	36.0	Blr	Meta

In [27]:

```
# first select the column
# then apply Fill na
data1['Name'].fillna('Manish')
```

Out[27]:

```
0      Ramesh
1      Suresh
2     Sathish
3      Manish
Name: Name, dtype: object
```

Draw back: Filling with Random value is not a Good approach

- To avoid the we have some methods
    - backfill
    - bfill
    - ffill
    - pad

pad

In [39]:

```
data1.fillna(method='pad')
```

Out[39]:

	Name	Age	City	Company
0	Ramesh	30.0	NaN	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	33.0	Hyd	Apple
3	Sathish	36.0	Blr	Meta

In [23]:

```
import warnings
warnings.filterwarnings('ignore')
data1.fillna(method='pad')
```

Out[23]:

	Name	Age	City	Company
0	Ramesh	30.0	NaN	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	33.0	Hyd	Apple
3	Sathish	36.0	Blr	Meta

In [41]:

```
import warnings
warnings.filterwarnings('ignore')
data1.fillna(method='pad',axis=1)
```

Out[41]:

	Name	Age	City	Company
0	Ramesh	30.0	30.0	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	Sathish	Hyd	Apple
3	NaN	36.0	Blr	Meta

In [34]:

```
data1
```

Out[34]:

	Name	Age	City	Company
0	Ramesh	30.0	NaN	Google
1	Suresh	33.0	Hyd	Microsoft
2	Sathish	NaN	Hyd	Apple
3	NaN	36.0	Blr	Meta

In [ ]:

```
idea-1: filling with random value = df.fillname(<random value>)
idea-2: filling with rv based on column=df[<col>].fillname(<rv>)
idea-3: filling with some pattrn using a method: df.fillna(<method>)
idea-4: filling with m-m-m based on column=df[<col>].fillname(<m>or<m>or<m>)
idea-5: filling with avg value only selected neighbours
```

In [ ]:

```
```

Mean-Median-Mode

In [31]:

```
# step-1: select the column
# step-2: get the mean
# step-3: apply fill na

age_mean=data1['Age'].mean()
print('age_mean:',age_mean)
data1['Age'].fillna(age_mean)
```

```
age_mean: 33.0
```

```
Out[31]: 0    30.0
         1    33.0
         2    33.0
         3    36.0
         Name: Age, dtype: float64
```

```
In [47]: age_mean=data1['Age'].mean()
         age_mean
```

```
Out[47]: 33.0
```

```
In [51]: city_mode=data1['City'].mode()
         city_mode.values[0]
```

```
Out[51]: 'Hyd'
```

```
In [33]: city_mode=data1['City'].mode()
         data1['City'].fillna(city_mode)
```

```
Out[33]: 0    Hyd
         1    Hyd
         2    Hyd
         3    Blr
         Name: City, dtype: object
```

```
In [35]: age_mean  # fixed value
```

```
Out[35]: 33.0
```

```
In [37]: city_mode # series
```

```
Out[37]: 0    Hyd
         Name: City, dtype: object
```

```
In [41]: city_mode.values[0]
```

```
Out[41]: 'Hyd'
```

- in future sometime if we direct impute mode value we will get error
- mode is coming as series
- mean is coming as fixed value
- so first convert series to fixed value then apply it

```
In [53]: city_mode.values[0]
```

```
Out[53]: 'Hyd'
```

```
In [33]: city_mode=data1['City'].mode()
         data1['City'].fillna(city_mode.values[0])
```

```
Out[33]: 0    Hyd
         1    Hyd
         2    Hyd
         3    Blr
         Name: City, dtype: object
```

impute class

- under sklearn we have impute class is there

```
In [53]: from sklearn import impute
```

```
In [54]: dir(impute)
```

```
Out[54]: ['KNNImputer',
          'MissingIndicator',
          'SimpleImputer',
          '__all__',
          '__builtins__',
          '__cached__',
          '__doc__',
          '__file__',
          '__getattr__',
          '__loader__',
          '__name__',
          '__package__',
          '__path__',
          '__spec__',
          '_base',
          '_knn',
          'typing']
```

KNN Immputer

- K- Nearest Neighbours
- Where k = hyper parameter which means the user can change
- Idea: Instead of taking all the samples average
    - First fix the neighbours number i.e. k
    - Then find the neighbours using distance metric
    - Then take the average of those samepls to fill the missing value

```
In [43]: import numpy as np
         from sklearn.impute import KNNImputer
         X = [[1, 2, np.nan], [3, np.nan, 3], [np.nan, 60, 5], [8, 8, 7]]
         pd.DataFrame(X)
```

Out[43]:

	0	1	2
0	1.0	2.0	NaN
1	3.0	NaN	3.0
2	NaN	60.0	5.0
3	8.0	8.0	7.0

In [57]:

```
import numpy as np
from sklearn.impute import KNNImputer
X = [[1, 2, np.nan], [3, np.nan, 3], [np.nan, 60, 5], [8, 8, 7]]
imputer = KNNImputer(n_neighbors=2,weights='uniform')
imputer.fit_transform(X)
```

Out[57]:

```
array([[ 1. ,  2. ,  5. ],
       [ 3. , 31. ,  3. ],
       [ 5.5, 60. ,  5. ],
       [ 8. ,  8. ,  7. ]])
```

In [45]:

```
import numpy as np
from sklearn.impute import KNNImputer
X = [[1, 2, np.nan], [3, np.nan, 3], [np.nan, 60, 5], [8, 8, 7]]
imputer = KNNImputer(n_neighbors=2,weights='distance')
imputer.fit_transform(X)
```

Out[45]:

```
array([[ 1.        ,  2.        ,  3.93905505],
       [ 3.        , 31.        ,  3.        ],
       [ 3.25775362, 60.        ,  5.        ],
       [ 8.        ,  8.        ,  7.        ]])
```

In [ ]:

```

```

In [45]:

```
import numpy as np
from sklearn.impute import KNNImputer
X = [[1, 2, np.nan], [3, np.nan, 3], [np.nan, 60, 5], [8, 8, 7]]
imputer = KNNImputer(n_neighbors=2)
imputer.fit_transform(X)
```

Out[45]:

```
array([[ 1. ,  2. ,  5. ],
       [ 3. , 31. ,  3. ],
       [ 5.5, 60. ,  5. ],
       [ 8. ,  8. ,  7. ]])
```

In [ ]:

```
idea-1: filling with random value = df.fillname(<random value>)
idea-2: filling with rv based on column=df[<col>].fillname(<rv>)
idea-3: filling with some pattrn using a method: df.fillna(<method>)
idea-4: filling with m-m-m based on column=df[<col>].fillname(<m>or<m>or<m>)
idea-5: filling with avg value only selected neighbours
idea-6: filling with a value based on correlation with another columns
```