Data Pre-Processing/Data Wrangling

```
pandas helps here
```

1- Data wrangling steps

1. loading the raw data
2. tiding up
3. filteration
4. deletion
5. assembeling
6. grouping
7. spliting
8. dealing missing values
9. removing duplicates
10. much more to make it Clean data ## 2- Data analysis ## 3- Data visualization ## 4- Data reporting ## 5- Projects winning

Data Wrangling contents

1. handling missing values
2. data formating
3. data normalization
    A. scaling
    B. centralization
4. data binning
    A. for groups of data
5. making dummiesof catagorical data
    A. catagorical to numerical

PANDAS vs MS EXCEL

PANDAS	MS EXCEL
data frame	worksheet
series	columns
index	row heading
row	row
NaN	empty cell

Let's Practice!

In [1]:
```
#install libararies if required
```

In [2]:
```
#import libararies
```

```
import pandas as pd
import seaborn as sns
import numpy as np
```

In [3]:
```
kashti= sns.load_dataset('titanic')
kashti.head()
ks1= kashti
ks2= kashti
```

In [4]:
```
#simple math operations
# (kashti['age']*10).head()
```

In [5]:
```
#simple math operations
# (kashti['age']-10).head()
```

Dealing with missing values

- found as NaN,N/A,0 or empty cell

> steps ;

1. try to recollect values
2. remove the entire row or column of missing value if it don't effect much the data.
3. replace missing with
   A. average of entire variable or similar
   B. frequency or mode replacement
   C. replace based on other functions(data sampler knows)
   D. ML algorithm helps
   E. leave it as it is
      a. bcz its better not to lose data
      b. less accurate

In [6]:
```
#where are missing values
kashti.isnull().sum()
```

Out[6]:
```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [7]:
```
kashti.shape
```

Out[7]:
```
(891, 15)
```

In [8]:
```
#use of drop.na function
kashti.dropna(subset=["deck"], axis=0, inplace=True)
#remove sepecific subset
#inplace=True modifies change in dataframe

kashti.shape
```

Out[8]:
```
(203, 15)
```

In [9]:
```
kashti.isnull().sum()
```

Out[9]:
```
survived        0
pclass          0
sex             0
age            19
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck            0
embark_town     2
alive           0
alone           0
dtype: int64
```

In [10]:
```
#removing all null values of  dataframe
kashti= kashti.dropna()
```

In [11]:
```
#finding null again
kashti.isnull().sum()
```

Out[11]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [12]:
```
kashti.shape
```

Out[12]:
```
(182, 15)
```

In [13]:
```
kashti.head()
```

Out[13]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	
10	1	3	female	4.0	1	1	16.7000	S	Third	child	False	
11	1	1	female	58.0	0	0	26.5500	S	First	woman	False	

In [14]:
```
ks1= kashti
ks2= kashti
```

In [15]:
```
#finding an average(mean)
mean=ks1["age"].mean()
mean
```

Out[15]:
```
35.62318681318681
```

In [16]:
```
#replacing nan with mean of the data (updating as well)
ks1["age"]= ks1['age'].replace(np.nan,mean)
```

In [17]:
```
ks1.isnull().sum()
```

Out[17]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [18]:
```
#replacing nan with mean of the data (updating as well)
ks1["embark_town"]= ks1['embark_town'].replace(np.nan,mean)
ks1["embarked"]= ks1['embarked'].replace(np.nan,mean)
```

In [19]:
```
ks1.isnull().sum()
```

Out[19]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
```

```
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [20]:
```
kashti.dropna(subset=["deck"], axis=0, inplace=True)
```

In [21]:
```
ks1.isnull().sum()
```

Out[21]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

Data formating

1. standardize the data
2. understandable and consistent
3. in one same standard unit

In [22]:
```
kashti.dtypes
```

Out[22]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
dtype: object
```

In [23]:
```
ks1['deck']= ks1['deck'].astype('object')
kashti.dtypes
```

Out[23]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck             object
embark_town      object
alive            object
alone              bool
dtype: object
```

In [24]:
```
#converting age into days lived
ks1["age"]=ks1["age"]*365
ks1.head()
```

Out[24]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
1	1	1	female	13870.0	1	0	71.2833	C	First	woman	False
3	1	1	female	12775.0	1	0	53.1000	S	First	woman	False
6	0	1	male	19710.0	0	0	51.8625	S	First	man	True
10	1	3	female	1460.0	1	1	16.7000	S	Third	child	False
11	1	1	female	21170.0	0	0	26.5500	S	First	woman	False

In [25]:
```
#finding types of columns
ks1.dtypes
```

Out[25]:
```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck             object
embark_town      object
alive            object
alone              bool
dtype: object
```

In [26]:
```
ks1['age']=  ks1['age'].apply(np.int64)
ks1.dtypes
```

Out[26]:
```
survived          int64
pclass            int64
```

```
sex              object
age               int64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck             object
embark_town      object
alive            object
alone              bool
dtype: object
```

In [27]:
```
ks1.head()
```

Out[27]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	d
1	1	1	female	13870	1	0	71.2833	C	First	woman	False	
3	1	1	female	12775	1	0	53.1000	S	First	woman	False	
6	0	1	male	19710	0	0	51.8625	S	First	man	True	
10	1	3	female	1460	1	1	16.7000	S	Third	child	False	
11	1	1	female	21170	0	0	26.5500	S	First	woman	False	

In [28]:
```
#rename the column
ks1.rename(columns={'age': 'age in days'},inplace=True)
ks1.head()
```

Out[28]:

	survived	pclass	sex	age in days	sibsp	parch	fare	embarked	class	who	adult_male	d
1	1	1	female	13870	1	0	71.2833	C	First	woman	False	
3	1	1	female	12775	1	0	53.1000	S	First	woman	False	
6	0	1	male	19710	0	0	51.8625	S	First	man	True	
10	1	3	female	1460	1	1	16.7000	S	Third	child	False	
11	1	1	female	21170	0	0	26.5500	S	First	woman	False	

Data normalization

1. uniform the data
2. make sure they have same impact
3. also for comutational reasons

In [29]:
```
kashti.head()
```

Out[29]:

	survived	pclass	sex	age in days	sibsp	parch	fare	embarked	class	who	adult_male	d
1	1	1	female	13870	1	0	71.2833	C	First	woman	False	
3	1	1	female	12775	1	0	53.1000	S	First	woman	False	
6	0	1	male	19710	0	0	51.8625	S	First	man	True	
10	1	3	female	1460	1	1	16.7000	S	Third	child	False	
11	1	1	female	21170	0	0	26.5500	S	First	woman	False	

In [30]:
```
#subsetting for the big df
ks2= kashti[['age in days','fare']]
ks2.head()
```

Out[30]:

	age in days	fare
1	13870	71.2833
3	12775	53.1000
6	19710	51.8625
10	1460	16.7000
11	21170	26.5500

- the above data is in wide range ,we need to normalize the data
- normalizing will help in comparison
- normalization will change values in range 0-1
- variables will have same influence on the model

Methods of Normalization

1. simple feature scaling

   > x(new)= x(old) / x(max)

2. min-max method
3. z-score (standard score method, range is -3 to +3)
4. log transformation

In [31]:
```
#1. simple feature scaling
#for fare
ks2['fare']= ks2['fare']/ ks2['fare'].max()
#for age
ks2['age in days']= ks2['age in days']/ ks2['age in days'].max()
#Lets see what it came out!
ks2
```

```
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/959847662.py:3: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks2['fare']= ks2['fare']/ ks2['fare'].max()
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/959847662.py:5: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks2['age in days']= ks2['age in days']/ ks2['age in days'].max()
```

Out[31]:

	age in days	fare
1	0.4750	0.139136
3	0.4375	0.103644
6	0.6750	0.101229
10	0.0500	0.032596
11	0.7250	0.051822
...	...	...
871	0.5875	0.102579
872	0.4125	0.009759
879	0.7000	0.162314
887	0.2375	0.058556
889	0.3250	0.058556

182 rows × 2 columns

In [32]:

```
#subsetting the data
ks3= kashti[['age in days','fare']]
ks3.head()
```

Out[32]:

	age in days	fare
1	13870	71.2833
3	12775	53.1000
6	19710	51.8625
10	1460	16.7000
11	21170	26.5500

In [33]:

```
#2. min-max method
#for fare
ks3['fare']= (ks3['fare']-ks3['fare'].min()) / (ks3['fare'].max()-ks3['fare'].min())
#for age
ks3['age in days']= (ks3['age in days']-ks3['age in days'].min()) / (ks3['age in day
#lets see what it came out!
ks3
```

```
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/2888301978.py:3: SettingWithCopyWarn
```

```
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks3['fare']= (ks3['fare']-ks3['fare'].min()) / (ks3['fare'].max()-ks3['fare'].min
())
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/2888301978.py:5: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks3['age in days']= (ks3['age in days']-ks3['age in days'].min()) / (ks3['age in d
ays'].max()-ks3['age in days'].min())
```

Out[33]:

	age in days	fare
1	0.468907	0.139136
3	0.430972	0.103644
6	0.671228	0.101229
10	0.038975	0.032596
11	0.721808	0.051822
...	...	...
871	0.582713	0.102579
872	0.405682	0.009759
879	0.696518	0.162314
887	0.228651	0.058556
889	0.317166	0.058556

182 rows × 2 columns

In [34]:

```
#subsetting the data
ks4= kashti[['age in days','fare']]
ks4.head()
```

Out[34]:

	age in days	fare
1	13870	71.2833
3	12775	53.1000
6	19710	51.8625
10	1460	16.7000
11	21170	26.5500

In [35]:

```
#3. z score / std method
#for fare
ks4['fare']= (ks4['fare']-ks4['fare'].mean()) / ks4['fare'].std()
#for age
ks4['age in days']= (ks4['age in days']-ks4['age in days'].mean()) / ks4['age in day
```

```
#Lets see what it came out!
ks4
```

```
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/1767079994.py:3: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['fare']= (ks4['fare']-ks4['fare'].mean()) / ks4['fare'].std()
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/1767079994.py:5: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks4['age in days']= (ks4['age in days']-ks4['age in days'].mean()) / ks4['age in d
ays'].std()
```

Out[35]:

	age in days	fare
1	0.151666	-0.099835
3	-0.039763	-0.337554
6	1.172618	-0.353732
10	-2.017859	-0.813428
11	1.427856	-0.684654
...	...	...
871	0.725951	-0.344689
872	-0.167382	-0.966388
879	1.300237	0.055413
887	-1.060716	-0.639551
889	-0.614049	-0.639551

182 rows × 2 columns

In [36]:

```
#subsetting the data
ks5= kashti[['age in days','fare']]
ks5.head()
```

Out[36]:

	age in days	fare
1	13870	71.2833
3	12775	53.1000
6	19710	51.8625
10	1460	16.7000
11	21170	26.5500

In [37]:

```
#4. log transformation
```

```
#for fare
ks5['fare']=np.log(ks5['fare'])
#for age
ks5['age in days']=np.log(ks5['age in days'])
ks5.head()
```

```
C:\Users\Azka\anaconda3\lib\site-packages\pandas\core\arraylike.py:364: RuntimeWarni
ng: divide by zero encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/847960995.py:3: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks5['fare']=np.log(ks5['fare'])
C:\Users\Azka\AppData\Local\Temp/ipykernel_6180/847960995.py:5: SettingWithCopyWarni
ng:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  ks5['age in days']=np.log(ks5['age in days'])
```

Out[37]:

	age in days	fare
1	9.537484	4.266662
3	9.455245	3.972177
6	9.888881	3.948596
10	7.286192	2.815409
11	9.960340	3.279030

Binning

- grouping of values into smaller number values(Bins)
- convert numeric into catagory (1-16, 17-30, 31-75 as jawan, borhay, bachy)
- to have better understanding of groups (low vs mid vs high price)

In [38]:

```
bins = np.linspace(min(kashti['age in days']),max(kashti['age in days']),4)
age_groups= ['bachy','jawan','borhy']
kashti['age in days']= pd.cut(kashti['age in days'],bins,labels=age_groups, include_
kashti['age in days']
```

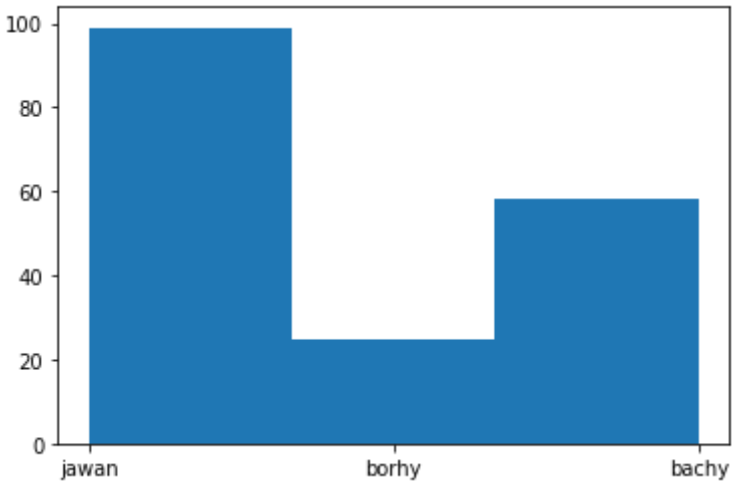
Out[38]:

```
1      jawan
3      jawan
6      borhy
10     bachy
11     borhy
       ...
871    jawan
872    jawan
879    borhy
887    bachy
889    bachy
```

```
Name: age in days, Length: 182, dtype: category
Categories (3, object): ['bachy' < 'jawan' < 'borhy']
```

In [39]:
```
import matplotlib.pyplot as plt
plt.hist(kashti['age in days'], bins=3)
```

Out[39]:
```
(array([99., 25., 58.]),
 array([0.        , 0.66666667, 1.33333333, 2.        ]),
 <BarContainer object of 3 artists>)
```



converting catagories into dummies

- easy to use in computation
- male / female (0/1)

In [40]:
```
pd.get_dummies(ks1['sex'])
```

Out[40]:

	female	male
1	1	0
3	1	0
6	0	1
10	1	0
11	1	0
...	...	...
871	1	0
872	0	1
879	1	0
887	1	0
889	0	1

182 rows × 2 columns

In [46]:
```
# Assignment
```

In [41]:
```
sex_dummies = pd.get_dummies(ks1.sex, prefix='sex')
```

In [42]:
```
ks1_with_dummies = pd.concat([ks1, sex_dummies], axis='columns')
ks1_with_dummies.head()
```

Out[42]:

	survived	pclass	sex	age in days	sibsp	parch	fare	embarked	class	who	adult_male	d
1	1	1	female	jawan	1	0	71.2833	C	First	woman	False	
3	1	1	female	jawan	1	0	53.1000	S	First	woman	False	
6	0	1	male	borhy	0	0	51.8625	S	First	man	True	
10	1	3	female	bachy	1	1	16.7000	S	Third	child	False	
11	1	1	female	borhy	0	0	26.5500	S	First	woman	False	

In [43]:
```
# now we need to remove age column which is text data, and already placed with dummi
ks1_with_dummies.drop('sex', axis='columns', inplace=True)
```

In [45]:
```
ks1_with_dummies.head()
```

Out[45]:

	survived	pclass	age in days	sibsp	parch	fare	embarked	class	who	adult_male	deck	em
1	1	1	jawan	1	0	71.2833	C	First	woman	False	C	
3	1	1	jawan	1	0	53.1000	S	First	woman	False	C	Sc
6	0	1	borhy	0	0	51.8625	S	First	man	True	E	Sc
10	1	3	bachy	1	1	16.7000	S	Third	child	False	G	Sc
11	1	1	borhy	0	0	26.5500	S	First	woman	False	C	Sc