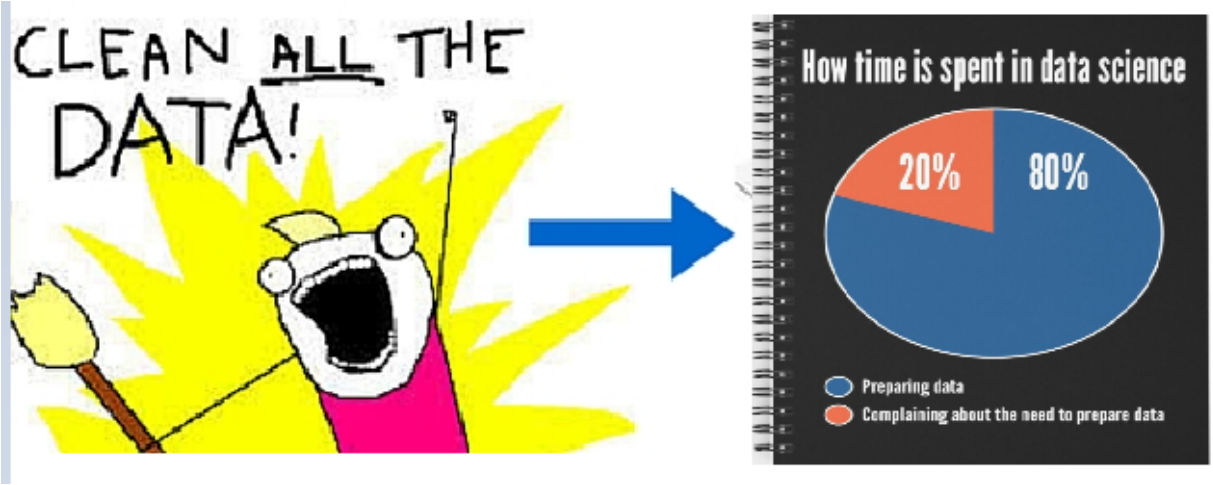
Data Preparation with Python!!

- How to deal with raw data with help of pandas and numpy libraries of python?
- What is a numerical and categorical variable in data and how to handle it.
- You will know how to find duplicate values , handle missing values and outliers.
- You will know how to scale the data and why it is important with its visualization impact.



Step 1: Load the data set and storing it in data-frame

We will be covering an example to read the data of type .csv and .xlsx to dataframe.

```
import pandas as pd

# Set ipython's max row display
pd.set_option('display.max_row', 1000)
# Set iPython's max column width to 50
pd.set_option('display.max_columns', 50)

#Loading .xlsx type data set to data-frame with meaningful name.
df_train = pd.read_excel('Data_Train.xlsx',sheet_name="Sheet1")
df_test = pd.read_excel('Data_Test.xlsx',sheet_name="Sheet1")

#Loading .csv type data set ,considering the data is in same folder, if not then you can refer here for more details
df_csvData = pd.read_csv('horror-train.csv')
```

Step 2: Handling missing data

Missing values are a common occurrence data and if not handled in the training data set , it can reduce the model fit performance or can lead to a biased model. It can lead to wrong prediction or classification. A missing value signifies a number of different things in your data.

Perhaps the data was not available or not applicable or the event did not happen. It could be that the person who entered the data did not know the right value, or missed filling in. Data mining methods vary in the way they treat missing values. There should be a strategy to treat missing values, let's see how we can do it.

1. Remove the missing data

#Method 1: List-wise deletion , is the process of removing the entire data which contains the missing value. Although it's a simple process, its disadvantage is reduction of power of the model as the sample size decreases.

Number	City	Gender	Age	Income	Illness
1	New York	Male	41	40367	No
2	Los Angele	Male	54	45084	No
3	New York	Male	42	52483	No
4	Los Angele	Male	40	40941	No
5	New York	Male	46	50289	No
6	Dallas	Female		50786	No
7	Dallas	Female	32	33155	No
8	Los Angele	Male	39	30914	No
9	Los Angele	Male	51	68667	No
10	Los Angele	Female	30		No
11	Dallas	Female	48	41524	Yes
12	New York	Male	47	54777	No
13	New York	Male	46	62749	No
14	Dallas		42	50894	No
15	Boston	Female	61	38429	No
16	Boston	Male	43	34074	No
17	Dallas	Male	27	50398	No
18	Dallas	Male		46373	Yes
19	New York	Male	47	51137	No
20	New York	Female	35	23688	No
21	New York	Male	57	17378	No

Number	City	Gender	Age	Income	Illness
1	New York	Male	41	40367	No
2	Los Angele	Male	54	45084	No
3	New York	Male	42	52483	No
4	Los Angele	Male	40	40941	No
5	New York	Male	46	50289	No
7	Dallas	Female	32	33155	No
8	Los Angele	Male	39	30914	No
9	Los Angele	Male	51	68667	No
11	Dallas	Female	48	41524	Yes
12	New York	Male	47	54777	No
13	New York	Male	46	62749	No
15	Boston	Female	61	38429	No
16	Boston	Male	43	34074	No
17	Dallas	Male	27	50398	No
19	New York	Male	47	51137	No
20	New York	Female	35	23688	No
21	New York	Male	57	17378	No

Remove all the rows with missing data

#Method 2: Pairwise deletion , is the process of removing only specific variables with missing values from the analysis and continuing to analyze all other variables without missing values, variables chosen will vary from analysis to analysis based on missingness. One of the disadvantages of this method, it uses different sample sizes for different variables.

Number	City	Gender	Age	Income	Illness
1	New York	Male	41	40367	No
2	Los Angele	Male	54	45084	No
3	New York	Male	42	52483	No
4	Los Angele	Male	40	40941	No
5	New York	Male	46	50289	No
6	Dallas	Female		50786	No
7	Dallas	Female	32	33155	No
8	Los Angele	Male	39	30914	No
9	Los Angele	Male	51	68667	No
10	Los Angele	Female	30	45919	No
11	Dallas	Female	48	41524	Yes
12	New York	Male	47	54777	No
13	New York	Male	46	62749	No
14	Dallas		42	50894	No
15	Boston	Female	61	38429	No
16	Boston	Male	43	34074	No
17	Dallas	Male	27	50398	No
18	Dallas	Male		46373	Yes
19	New York	Male	47	51137	No
20	New York	Female	35	23688	No
21	New York	Male	57	17378	No

The grey column is excluded for data analysis

In the above example,for pairwise deletion while performing a correlation we will only perform correlation between city, income and illness and ignore correlation between gender and age but while list-wise deletion of the missing row would have been done and analysis can be carried out on all three features. Let's see it with some pandas code as well.

```
#df is the data frame name where csv data is loaded.
#drop id and use inplace = True only when you want to modify the original dataframe, otherwise you can make a copy and use it.
df.drop(['Number'],axis=1, inplace=True)
```

```
#Removing column data
df.drop(columns=['City'], inplace=True)

#Remove rows which contain null values.
df_1 = df.dropna( subset =['Gender','Age'] )
```

#Method 3: Retain the Data through imputation

The imputation overcomes the problem of removal of missing records and produces a complete dataset that can be used for machine learning.

#Method 4: Mean , Mode and Median imputation

Imputation is a way to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. **For numeric data type Mean / Mode / Median imputation** is one of the most frequently used methods while for categorical mode is preferred. To know when to go for Mean/Median/Mode refer to descriptive statistics.

Number	City	Gender	Age	Income	Illness
1	New York	Male	41	40367	No
2	Los Angele	Male	54	45084	No
3	New York	Male	42	52483	No
4	Los Angele	Male	40	40941	No
5	New York	Male	46	50289	No
6	Dallas	Female		50786	No
7	Dallas	Female	32	33155	No
8	Los Angele	Male	39	30914	No
9	Los Angele	Male	51	68667	No
10	Los Angele	Female	30		No
11	Dallas	Female	48	41524	Yes
12	New York	Male	47	54777	No
13	New York	Male	46	62749	No
14	Dallas	Male	42	50894	No
15	Boston	Female	61	38429	No
16	Boston	Male	43	34074	No
17	Dallas	Male	27	50398	No
18	Dallas	Male		46373	Yes
19	New York	Male	47	51137	No
20	New York	Female	35	23688	No
21	New York	Male	57	17378	No

→

Number	City	Gender	Age	Income	Illness
1	New York	Male	41	40367	No
2	Los Angele	Male	54	45084	No
3	New York	Male	42	52483	No
4	Los Angele	Male	40	40941	No
5	New York	Male	46	50289	No
6	Dallas	Female	mean/median/ mode	50786	No
7	Dallas	Female	32	33155	No
8	Los Angele	Male	39	30914	No
9	Los Angele	Male	51	68667	No
10	Los Angele	Female	30	mean/median/ mode	No
11	Dallas	Female	48	41524	Yes
12	New York	Male	47	54777	No
13	New York	Male	46	62749	No
14	Dallas	Male	42	50894	No
15	Boston	Female	61	38429	No
16	Boston	Male	43	34074	No
17	Dallas	Male	27	50398	No
18	Dallas	Male	mean/median/ mode	46373	Yes
19	New York	Male	47	51137	No
20	New York	Female	35	23688	No
21	New York	Male	57	17378	No

Fill missing data with mean/median/mode using below code.

```
#Filling with mean value
df['Income'] = df['Income'].fillna((df['Income'].mean()))

#Filling with median value
df['Age'] = df['Age'].fillna((df['Age'].median()))

#Filling with mode value
df['Age'] = df['Age'].fillna((df['Age'].mode()))
```

#Method 5: Forward filling

Also commonly known as Last observation carried forward (LOCF). It is the process of replacing a missing value with the last observed record. Its widely used imputed method in time series data . This method is advantageous as it's easy to communicate , but it is based on the assumption that the value of outcome remains unchanged by the missing data, which seems very unlikely.

```
import pandas as pd
df = pd.DataFrame([[1, 2, 3], [None, None, 6], [None, 9, None]])
print(df)
'''
    0    1    2
0  1.0  2.0  3.0
1  NaN  NaN  6.0
2  NaN  9.0  NaN
'''

df1 = df[1].fillna(method='ffill')
print(df1)
'''
    0    1    2
0  1.0  2.0  3.0
1  NaN  2.0  6.0
2  NaN  9.0  NaN
'''

df1 = df.fillna(method='ffill')
print(df1)
'''
    0    1    2
0  1.0  2.0  3.0
1  1.0  2.0  6.0
2  1.0  9.0  6.0
'''
```

```
df[0].fillna(method='ffill',limit=1 , inplace=True)
print(df)
'''
    0    1    2
0  1.0  2.0  3.0
1  1.0  NaN  6.0
2  NaN  9.0  NaN
'''
```

#Method 6: Backward filling

As the name suggests, its exact opposite of forward filling and also commonly known as **Next Observation Carried Backward (NOCB)**. It takes the first observation after the missing value and carries it backward.

For backward filling, you can replace imputation method to “bfill” in Forward fill example.

#Method 7: Linear Interpolation

Interpolation is a mathematical method that adjusts a function to data and uses this function to extrapolate the missing data. In simple words using logic to fill the missing values. The simplest type of interpolation is the linear interpolation, that makes a mean between the values before the missing data and the value after. Of course, we could have a pretty complex pattern in data and linear interpolation could not be enough. There are several different types of interpolation. Just in Pandas we have the following options like : ‘linear’, ‘time’, ‘index’, ‘values’, ‘nearest’, ‘zero’, ‘slinear’, ‘quadratic’, ‘cubic’, ‘polynomial’, ‘spline’, ‘piecewise polynomial’ and many more .

Step 3: Check for duplicate value

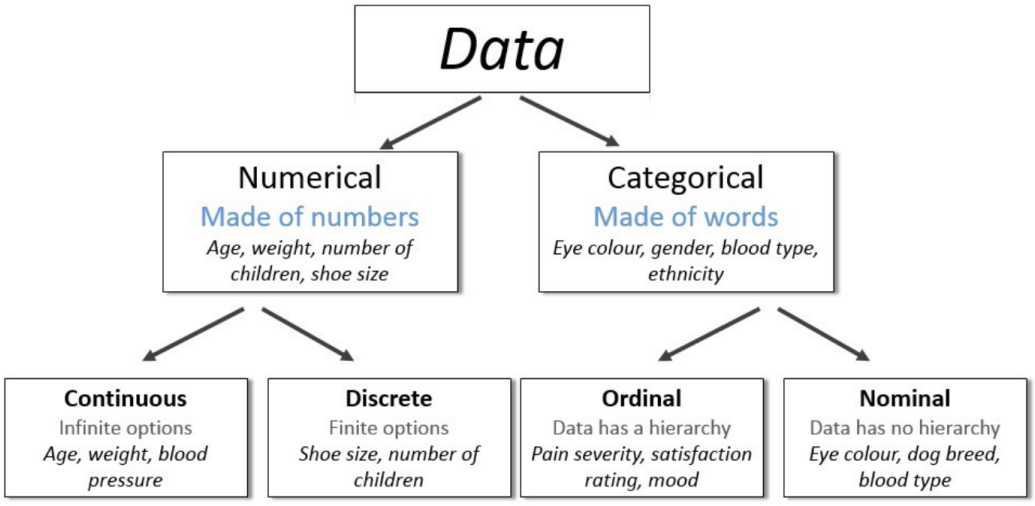
One another reason why your model performance might not be accurate can be because of the duplicated data, which can make the data bias and results corrupted. Make sure you take care of such data as well.

```
import pandas as pd
df = pd.DataFrame(['a','b','c','d','a','b','e'])
df[df.duplicated(keep=False)]
'''
Out[1]:
   0
0  a
1  b
4  a
5  b
'''
```

Keep : {‘first’, ‘last’, False}, default ‘first’

- first : Mark Duplicates as True Except for the first Occurance.
- Second : Mark Duplicates as True Except for the last Occurance.
- False : Mark all Duplicates as True.

Step 4: Separating categorical and numerical data.



In general you can process the whole data using steps 1–3. But posting this numeric data and categorical data needs different kinds of treatment because of their nature.

```
#Getting categorical data
df_cat = df.loc[:,df.dtypes==np.object]
#Getting Numeric data
df_num = df.loc[:,df.dtypes!=np.object]
```

Step 5 : Handling the numerical data

1. **Scaling :-**

   Well let me first explain why scaling the data is important. You understand how 1- kg is same as 1000 gm or 1- km is same as 1000 meters but your machine doesn’t, suppose a feature is denoted in Kg while another in gm, your system just sees them as number and do its processing on it, it’s our job to make machine understand units importance . Since most of our algorithm uses distance calculation, scaling will give the correct direction for computation. Post this topic go back to your data and see if it is scaled and if not, scale it and see performance difference in algorithm. Let’s see types of scaling with python code.

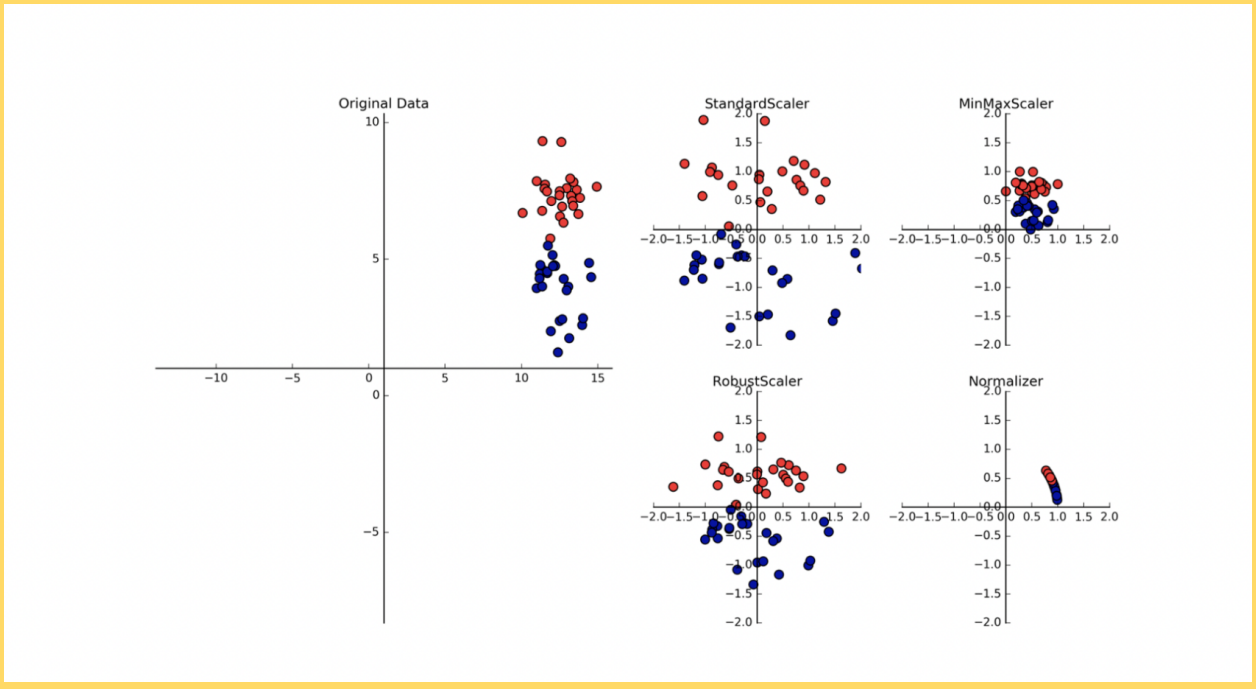


Fig. Data comparison using various scales.

With theory let's also practice below scaling technique on data, this data is about top 50 popular songs , and have features like Beats.Per.Minute and ranges from 85 to 190 while feature like Acousticness ranges from 1 to 75, and if are going to study these two features together , a better comparison be on same scale.

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

In [5]:
```
df_train = pd.read_csv('top50_sportify_songs.csv', encoding='latin-1', index_col=0)
df_train.shape
```

Out[5]:
```
(50, 13)
```

In [6]:
```
df_train.head(10)
```

Out[6]:

Genre	Beats.Per.Minute	Energy	Danceability	Loudness..dB..	Liveness	Valence.	Length.	Acousticness..	Speechiness.	Popularity
nadian pop	117	55	76	-6	8	75	191	4	3	79
gaeton flow	105	81	79	-4	8	61	302	8	9	92
ce pop	190	80	40	-4	16	70	186	12	46	85
pop	93	65	64	-8	8	55	198	12	19	86
fw rap	150	65	58	-4	11	18	175	45	7	94
pop	102	68	80	-5	9	84	220	9	4	84
music	180	64	75	-6	7	23	131	2	29	92
pop	111	68	48	-5	8	35	202	15	9	90
ountry rap	136	62	88	-6	11	64	157	5	10	87
tropop	135	43	70	-11	10	56	194	33	38	95

#Method 1 : Standardization

Standardization is the process of transforming the data that centers the data by removing the mean value of each feature and then scale it by dividing (non-constant) features by their standard deviation. It assumes your data is normally distributed within each feature, with a mean 0 and standard deviation of 1.

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

σ = Standard Deviation

We can check data's distribution using seaborn library but there are many other ways as well, which you are free to explore. Below is a plot with a seaborn library.

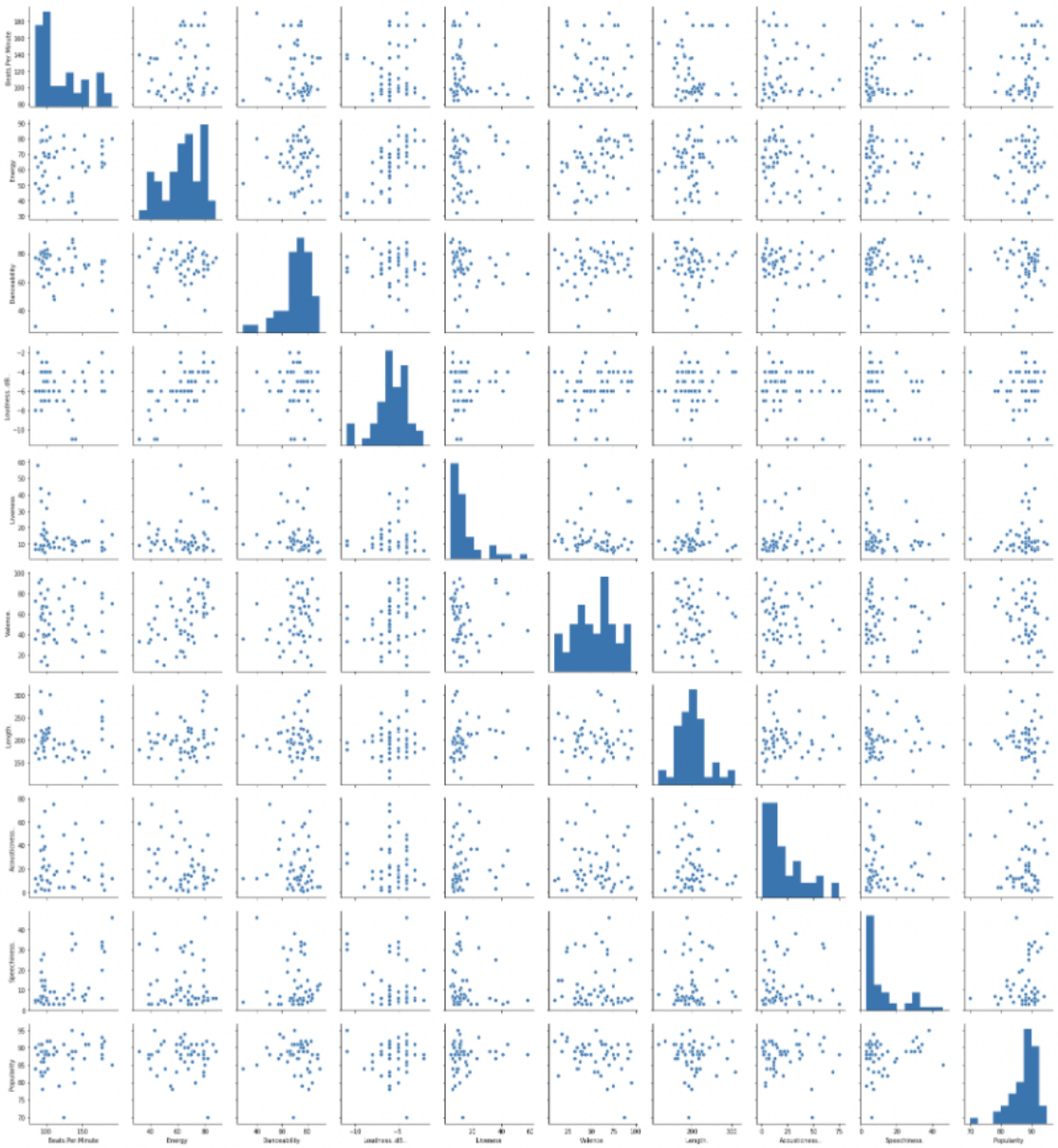
In [7]:
```
import pandas as pd
import seaborn as sns
```

In [5]:
```
df_train = pd.read_csv('top50_sportify_songs.csv', encoding='latin-1', index_col=0)
df_train.shape
```

Out[5]:
```
(50, 13)
```

In [8]:
```
sns.pairplot(df_train)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x22191b6bb70>
```



The diagonal data shows a histogram distribution plot, most of the data doesn't look normalized, you can use log on the data which sometimes helps reshape data, please do explore other options to reshape distribution of data and make it normalized before you standardized it. For now just Standardized the numeric data.

In [11]:
```
import pandas as pd
import numpy as np
```

In [12]:
```
df_train = pd.read_csv('top50_sportify_songs.csv', encoding='latin-1', index_col=0)
df_train.shape
```

Out[12]:
```
(50, 13)
```

In [13]:
```
#Getting Numeric data
df_num = df_train.loc[:,df_train.dtypes!=np.object]
```

In [15]:
```
from sklearn.preprocessing import StandardScaler
# Create the standard Scaler object
scaler = StandardScaler()
scaler = scaler.fit(df_num)
```

In [17]:
```
# standardization the dataset and print the first 5 rows
normalized = scaler.transform(df_num)
for i in range(5):
    print(normalized[i])
```

```
[-0.10003973 -0.64306057  0.3911946  -0.16701216 -0.6050936   0.92259536
 -0.25702919 -0.96571922 -0.85796387 -1.91168174]
[-0.4923524   1.20236711  0.64521708  0.81541229 -0.6050936   0.28944168
  2.6074527  -0.75300573 -0.31494876  1.0120668 ]
[ 2.28652901  1.13138912 -2.65707504  0.81541229  0.12174556  0.69646905
 -0.3860599  -0.54029225  3.03364442 -0.56225933]
[-0.88466507  0.06671931 -0.62489528 -1.14943661 -0.6050936   0.01809011
 -0.07638618 -0.54029225  0.59007642 -0.3373556 ]
[ 0.97882011  0.06671931 -1.13294022  0.81541229 -0.33252891 -1.65524462
 -0.66992748  1.21459399 -0.4959538   1.46187427]
```

In [18]:
```
# inverse transform and print the first 5 rows
inversed = scaler.inverse_transform(normalized)
for i in range(5):
    print(inversed[i])
```

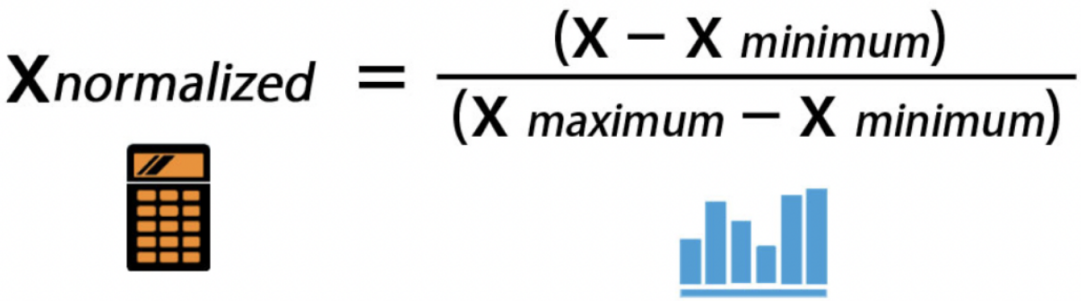
```
[117.  55.  76.  -6.   8.  75. 191.   4.   3.  79.]
[105.  81.  79.  -4.   8.  61. 302.   8.   9.  92.]
[190.  80.  40.  -4.  16.  70. 186.  12.  46.  85.]
[ 93.  65.  64.  -8.   8.  55. 198.  12.  19.  86.]
[150.  65.  58.  -4.  11.  18. 175.  45.   7.  94.]
```

Fig. Standard scaling

#Method 2: Normalization

Normalization is the process of transforming the data by scaling individual samples to have unit norm. It is often called Min-Max scaling which basically shrinks the range of data values between 0 and 1. It works well when distribution is not Gaussian or Standard deviation is quite small.

It is highly sensitive to outliers, so make sure to use this scaling technique your data is recommended to be outlier free.
We can also use the mean way of normalizing by just replacing X-minimum in numerator to X-mean.



Formula : Normalization using min-max

Let's normalize the data using python using normalize and MinMaxScaler library of sklearn.preprocessing.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('top50_sportify_songs.csv', encoding='latin-1', index_col=0)
df_train.shape
Out[2] - (50, 13)

#Getting Numeric data
df_num = df_train.loc[:,df_train.dtypes!=np.object]

from sklearn.preprocessing import normalize
# Create the normal object
normal = normalize(df_num,norm='l2')
#nomr can be l1, l2, max which is basically way to
#assigns a strictly positive length or size to each vector
#in a vector space.
normal
#gives a array in range of -1 to 1
# you can use it in pipeline effectively
```

Out[9]:

```
array([[ 0.43921073,  0.20646658,  0.28529928, -0.02252363,  0.0300315 ,
         0.28154534,  0.71700213,  0.01501575,  0.01126181,  0.29656109],
       [ 0.29412688,  0.22689788,  0.22129546, -0.01120483,  0.02240967,
         0.17087371,  0.84596493,  0.02240967,  0.02521088,  0.25771117],
       [ 0.62185547,  0.26183388,  0.13091694, -0.01309169,  0.05236678,
         0.22910465,  0.60876378,  0.03927508,  0.15055448,  0.2781985 ],
       [ 0.35868057,  0.25069072,  0.24683394, -0.03085424,  0.03085424,
         0.21212292,  0.76364251,  0.04628136,  0.07327883,  0.33168311],
       [ 0.55868731,  0.24209783,  0.21602576, -0.01489833,  0.0409704 ,
         0.06704248,  0.65180186,  0.16760619,  0.02607207,  0.35011071],
       [ 0.35163425,  0.23442284,  0.27579157, -0.01723697,  0.03102655,
         0.28958115,  0.75842682,  0.03102655,  0.01378958,  0.28958115],
       [ 0.68423261,  0.24328271,  0.28509692, -0.02280775,  0.02660905,
         0.08742972,  0.49796929,  0.00760258,  0.11023748,  0.34971889],
       [ 0.42022248,  0.25743359,  0.18171783, -0.01892894,  0.0302863 ,
         0.13250258,  0.7647292 ,  0.05678682,  0.03407209,  0.34072093],
       [ 0.52667307,  0.24010096,  0.34078846, -0.02323558,  0.04259856,
         0.24784615,  0.60799759,  0.01936298,  0.03872596,  0.33691586],
       [ 0.48485612,  0.15443565,  0.25140688, -0.03950679,  0.03591527,
         0.2011255 ,  0.6967562 ,  0.11852038,  0.13647802,  0.34119505],
       [ 0.51691201,  0.182094  ,  0.179157  , -0.014685  ,  0.070488  ,
         0.070488  ,  0.73718701,  0.17622   ,  0.091047  ,  0.27314101],
       [ 0.37619588,  0.2782282 ,  0.32133398, -0.01567483,  0.05878061,
         0.14891087,  0.72496081,  0.1097238 ,  0.02743095,  0.33700881],
       [ 0.43157171,  0.16085854,  0.19616896, -0.02354027,  0.04315717,
         0.17655206,  0.714055  ,  0.29425344,  0.01177014,  0.34525736],
       [ 0.47151955,  0.21164798,  0.19557345, -0.00535818,  0.01607453,
         0.20361071,  0.77157744,  0.01875362,  0.05358177,  0.23308068],
       [ 0.38048516,  0.18835899,  0.31267593, -0.01506872,  0.04520616,
         0.0376718 ,  0.77227187,  0.0376718 ,  0.0188359 ,  0.34658055],
       [ 0.30780362,  0.14893724,  0.23168014, -0.02316801,  0.05295546,
         0.04633603,  0.86383597,  0.0397166 ,  0.04964575,  0.27139674],
       [ 0.52771775,  0.19489576,  0.22487972, -0.01799038,  0.03298236,
         0.12893104,  0.7286103 ,  0.04497594,  0.09594868,  0.26985567],
       [ 0.3645428 ,  0.19442283,  0.30783615, -0.02430285,  0.02835333,
         0.36859328,  0.63997514,  0.22682663,  0.02025238,  0.36859328],
       [ 0.52087581,  0.15626274,  0.33656591, -0.0320539 ,  0.05609432,
         0.20033685,  0.64508465,  0.07212127,  0.0320539 ,  0.35259285],
       [ 0.57973162,  0.22748962,  0.2641815 , -0.01100756,  0.04403025,
         0.15043669,  0.63476943,  0.04036106,  0.04036106,  0.33389606],
       [ 0.50100294,  0.29898563,  0.27474355, -0.02828242,  0.02828242,
         0.25454182,  0.61817299,  0.01616139,  0.01212104,  0.36767152],
       [ 0.64330494,  0.24646098,  0.29241133, -0.02506383,  0.05012766,
         0.20051063,  0.48039005,  0.14202836,  0.03341844,  0.38013474],
       [ 0.25781133,  0.22138147,  0.22698606, -0.01120919,  0.02522067,
         0.16253323,  0.86590978,  0.03923216,  0.01961608,  0.23259065],
       [ 0.27783406,  0.23555497,  0.24159484, -0.01207974,  0.13287716,
         0.24159484,  0.80330283,  0.10871768,  0.01207974,  0.27481413],
       [ 0.48619378,  0.16206459,  0.24129617, -0.03961579,  0.04321722,
         0.24489761,  0.7022799 ,  0.09003588,  0.10804306,  0.32052775],
       [ 0.44276397,  0.29279553,  0.24637672, -0.01428271,  0.0464188 ,
         0.31064891,  0.68199934,  0.17496318,  0.02142406,  0.2499474 ],
       [ 0.34881188,  0.20999899,  0.291863  , -0.02135583,  0.06406749,
         0.19220246,  0.74745403,  0.24559204,  0.03559305,  0.29542231],
       [ 0.3544898 ,  0.16985969,  0.26955995, -0.02584821,  0.07015944,
         0.14770408,  0.78652424,  0.13662628,  0.01107781,  0.32864158],
       [ 0.35081466,  0.32095809,  0.27617324, -0.01119621,  0.02239242,
         0.24631667,  0.72028967,  0.04105278,  0.02239242,  0.33961844],
       [ 0.31534847,  0.26787665,  0.217014  , -0.01356337,  0.12207037,
         0.31873931,  0.75276731,  0.01017253,  0.08477109,  0.30178509],
       [ 0.3374637 ,  0.20247822,  0.11513468, -0.03176129,  0.03970161,
         0.1429258 ,  0.83373386,  0.04764193,  0.01588064,  0.33349354],
       [ 0.51983582,  0.11881962,  0.28962281, -0.04084424,  0.03341802,
         0.12253273,  0.66464723,  0.21907367,  0.12253273,  0.33046706],
       [ 0.37586126,  0.27798073,  0.30538728, -0.01957611,  0.03523699,
         0.26623506,  0.68907899,  0.08613487,  0.1096262 ,  0.34845471],
       [ 0.37532469,  0.22596078,  0.3063875 , -0.02680891,  0.04978797,
         0.06893719,  0.76596875,  0.00765969,  0.05744766,  0.34085609],
       [ 0.34352904,  0.30230556,  0.26451736, -0.01717645,  0.10992929,
         0.13397633,  0.76950505,  0.06527052,  0.02061174,  0.30574085],
       [ 0.46940218,  0.29646454,  0.27175916, -0.02882294,  0.05764588,
         0.13176202,  0.67528033,  0.07411613,  0.02470538,  0.36646311],
       [ 0.54007044,  0.21480074,  0.23014365, -0.01534291,  0.0337544 ,
         0.19025209,  0.69349955,  0.04296015,  0.10433179,  0.27924097],
       [ 0.54209363,  0.29244525,  0.25678119, -0.01783203,  0.1283906 ,
         0.3245429 ,  0.57775769,  0.04636327,  0.01783203,  0.31027728],
       [ 0.48847492,  0.25839615,  0.29733256, -0.01769837,  0.03893641,
         0.33626897,  0.64068087,  0.01415869,  0.02123804,  0.28317387],
       [ 0.35538922,  0.20949259,  0.30301607, -0.02244564,  0.02244564,
         0.25438387,  0.7444469 ,  0.17956508,  0.02618657,  0.29179326],
       [ 0.36386999,  0.23791499,  0.24141374, -0.02449125,  0.0349875 ,
         0.13994999,  0.79771497,  0.0069975 ,  0.01049625,  0.30788999],
       [ 0.34540576,  0.27632461,  0.31289698, -0.02438158,  0.02844518,
         0.2966426 ,  0.69487512,  0.0040636 ,  0.02031799,  0.36572375],
       [ 0.35983081,  0.14618127,  0.21364954, -0.02248943,  0.08620946,
         0.1199436 ,  0.80961932,  0.13868479,  0.01124471,  0.32609667],
       [ 0.49334982,  0.14510289,  0.3264815 , -0.03264815,  0.02176543,
         0.12696503,  0.71825929,  0.01813786,  0.04715844,  0.30471606],
```

```
from sklearn.preprocessing import MinMaxScaler
# Create the scalar object
scaler = MinMaxScaler()
scaler.fit(df_num)
print(scaler.data_max_)
scaler.transform(df_num)
```

```
[190.  88.  90.  -2.  58.  95. 309.  75.  46.  95.]
Out[14]: array([[0.3047619 , 0.41071429, 0.7704918 , 0.55555556, 0.05660377,
                 0.76470588, 0.39175258, 0.04054054, 0.        , 0.36      ],
                [0.19047619, 0.875     , 0.81967213, 0.77777778, 0.05660377,
                 0.6       , 0.96391753, 0.09459459, 0.13953488, 0.88      ],
                [1.        , 0.85714286, 0.18032787, 0.77777778, 0.20754717,
                 0.70588235, 0.36597938, 0.14864865, 1.        , 0.6       ],
                [0.07619048, 0.58928571, 0.57377049, 0.33333333, 0.05660377,
                 0.52941176, 0.42783505, 0.14864865, 0.37209302, 0.64      ],
                [0.61904762, 0.58928571, 0.47540984, 0.77777778, 0.11320755,
                 0.09411765, 0.30927835, 0.59459459, 0.09302326, 0.96      ],
                [0.16190476, 0.64285714, 0.83606557, 0.66666667, 0.0754717 ,
                 0.87058824, 0.54123711, 0.10810811, 0.02325581, 0.56      ],
                [0.9047619 , 0.57142857, 0.75409836, 0.55555556, 0.03773585,
                 0.15294118, 0.08247423, 0.01351351, 0.60465116, 0.88      ],
                [0.24761905, 0.64285714, 0.31147541, 0.66666667, 0.05660377,
                 0.29411765, 0.44845361, 0.18918919, 0.13953488, 0.8       ],
                [0.48571429, 0.53571429, 0.96721311, 0.55555556, 0.11320755,
                 0.63529412, 0.21649485, 0.05405405, 0.1627907 , 0.68      ],
                [0.47619048, 0.19642857, 0.67213115, 0.        , 0.09433962,
                 0.54117647, 0.40721649, 0.43243243, 0.81395349, 1.        ],
                [0.86666667, 0.53571429, 0.52459016, 0.66666667, 0.35849057,
                 0.16470588, 0.70103093, 0.7972973 , 0.65116279, 0.92      ],
                [0.1047619 , 0.69642857, 0.86885246, 0.77777778, 0.18867925,
                 0.32941176, 0.36082474, 0.36486486, 0.09302326, 0.64      ],
                [0.23809524, 0.16071429, 0.3442623 , 0.55555556, 0.11320755,
                 0.41176471, 0.34536082, 1.        , 0.        , 0.72      ],
                [0.86666667, 0.83928571, 0.72131148, 1.        , 0.01886792,
                 0.77647059, 0.89175258, 0.08108108, 0.39534884, 0.68      ],
                [0.15238095, 0.32142857, 0.8852459 , 0.77777778, 0.13207547,
                 0.        , 0.46391753, 0.12162162, 0.04651163, 0.88      ],
                [0.07619048, 0.23214286, 0.67213115, 0.44444444, 0.20754717,
                 0.04705882, 0.75257732, 0.14864865, 0.27906977, 0.48      ],
                [0.86666667, 0.58928571, 0.75409836, 0.55555556, 0.11320755,
                 0.38823529, 0.65979381, 0.18918919, 0.6744186 , 0.8       ],
                [0.04761905, 0.28571429, 0.7704918 , 0.55555556, 0.03773585,
                 0.95294118, 0.22164948, 0.74324324, 0.04651163, 0.84      ],
                [0.42857143, 0.125     , 0.90163934, 0.33333333, 0.16981132,
                 0.47058824, 0.2371134 , 0.22972973, 0.11627907, 0.72      ],
                [0.6952381 , 0.53571429, 0.70491803, 0.88888889, 0.13207547,
                 0.36470588, 0.29896907, 0.13513514, 0.18604651, 0.84      ],
                [0.37142857, 0.75      , 0.63934426, 0.44444444, 0.03773585,
                 0.62352941, 0.19587629, 0.04054054, 0.        , 0.84      ],
                [0.65714286, 0.48214286, 0.67213115, 0.55555556, 0.13207547,
                 0.44705882, 0.        , 0.44594595, 0.11627907, 0.84      ],
                [0.06666667, 0.83928571, 0.85245902, 0.77777778, 0.0754717 ,
                 0.56470588, 1.        , 0.17567568, 0.09302326, 0.52      ],
                [0.06666667, 0.82142857, 0.83606557, 0.77777778, 0.73584906,
                 0.82352941, 0.77835052, 0.47297297, 0.02325581, 0.84      ],
                [0.47619048, 0.23214286, 0.62295082, 0.        , 0.13207547,
                 0.68235294, 0.41237113, 0.32432432, 0.62790698, 0.76      ],
                [0.37142857, 0.89285714, 0.6557377 , 0.77777778, 0.1509434 ,
                 0.90588235, 0.39175258, 0.64864865, 0.06976744, 0.        ],
                [0.12380952, 0.48214286, 0.86885246, 0.55555556, 0.24528302,
                 0.51764706, 0.48969072, 0.91891892, 0.1627907 , 0.52      ],
                [0.1047619 , 0.25      , 0.72131148, 0.44444444, 0.26415094,
                 0.35294118, 0.50515464, 0.48648649, 0.        , 0.76      ],
                [0.08571429, 0.96428571, 0.73770492, 0.88888889, 0.01886792,
                 0.65882353, 0.40206186, 0.13513514, 0.06976744, 0.84      ],
                [0.07619048, 0.83928571, 0.57377049, 0.77777778, 0.58490566,
                 0.98823529, 0.55154639, 0.02702703, 0.51162791, 0.76      ],
                [0.        , 0.33928571, 0.        , 0.33333333, 0.09433962,
                 0.30588235, 0.48969072, 0.14864865, 0.02325581, 0.56      ],
                [0.52380952, 0.        , 0.80327869, 0.        , 0.0754717 ,
                 0.27058824, 0.32989691, 0.78378378, 0.69767442, 0.76      ],
                [0.1047619 , 0.69642857, 0.80327869, 0.66666667, 0.0754717 ,
                 0.68235294, 0.31443299, 0.28378378, 0.58139535, 0.76      ],
```

#Method 3: Robust Scaling

Its scaling is similar to normalization but it instead uses the interquartile range, to make it robust to outliers. Because of interquartile property it does not take median into account and only focuses on the part where the bulk data is.

Robust Scaler	$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$

Python implementation for robust-Scaler.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('top50_sportify_songs.csv', encoding='latin-1', index_col=0)
df_train.shape

Out[2]: (50, 13)
#Getting Numeric data
df_num = df_train.loc[:,df_train.dtypes!=np.object]

from sklearn.preprocessing import RobustScaler
# Create the scalar object
scaler = RobustScaler()
scaler.fit(df_num)
scaler.transform(df_num)
```

Out[15]:

```
array([[ 0.30120482, -0.58974359,  0.19607843,  0.        , -0.38709677,
         0.624     , -0.17177914, -0.43137255, -0.4       , -1.89473684],
       [ 0.01204819,  0.74358974,  0.43137255,  0.72727273, -0.38709677,
         0.176     ,  2.55214724, -0.2745098 ,  0.2       ,  0.84210526],
       [ 2.06024096,  0.69230769, -2.62745098,  0.72727273,  0.64516129,
         0.464     , -0.29447853, -0.11764706,  3.9       , -0.63157895],
       [-0.27710843, -0.07692308, -0.74509804, -0.72727273, -0.38709677,
        -0.016     ,  0.        , -0.11764706,  1.2       , -0.42105263],
       [ 1.09638554, -0.07692308, -1.21568627,  0.72727273,  0.        ,
        -1.2       , -0.56441718,  1.17647059,  0.        ,  1.26315789],
       [-0.06024096,  0.07692308,  0.50980392,  0.36363636, -0.25806452,
         0.912     ,  0.5398773 , -0.23529412, -0.3       , -0.84210526],
       [ 1.81927711, -0.12820513,  0.11764706,  0.        , -0.51612903,
        -1.04      , -1.64417178, -0.50980392,  2.2       ,  0.84210526],
       [ 0.15662651,  0.07692308, -2.        ,  0.36363636, -0.38709677,
        -0.656     ,  0.09815951,  0.        ,  0.2       ,  0.42105263],
       [ 0.75903614, -0.23076923,  1.1372549 ,  0.        ,  0.        ,
         0.272     , -1.00613497, -0.39215686,  0.3       , -0.21052632],
       [ 0.73493976, -1.20512821, -0.2745098 , -1.81818182, -0.12903226,
         0.016     , -0.09815951,  0.70588235,  3.1       ,  1.47368421],
       [ 1.72289157, -0.23076923, -0.98039216,  0.36363636,  1.67741935,
        -1.008     ,  1.3006135 ,  1.76470588,  2.4       ,  1.05263158],
       [-0.20481928,  0.23076923,  0.66666667,  0.72727273,  0.51612903,
        -0.56      , -0.3190184 ,  0.50980392,  0.        , -0.42105263],
       [ 0.13253012, -1.30769231, -1.84313725,  0.        ,  0.        ,
        -0.336     , -0.39263804,  2.35294118, -0.4       ,  0.        ],
       [ 1.72289157,  0.64102564, -0.03921569,  1.45454545, -0.64516129,
         0.656     ,  2.20858896, -0.31372549,  1.3       , -0.21052632],
       [-0.08433735, -0.84615385,  0.74509804,  0.72727273,  0.12903226,
        -1.456     ,  0.17177914, -0.19607843, -0.2       ,  0.84210526],
       [-0.27710843, -1.1025641 , -0.2745098 , -0.36363636,  0.64516129,
        -1.328     ,  1.54601227, -0.11764706,  0.8       , -1.26315789],
       [ 1.72289157, -0.07692308,  0.11764706,  0.        ,  0.        ,
        -0.4       ,  1.10429448,  0.        ,  2.5       ,  0.42105263],
       [-0.34939759, -0.94871795,  0.19607843,  0.        , -0.51612903,
         1.136     , -0.98159509,  1.60784314, -0.2       ,  0.63157895],
       [ 0.61445783, -1.41025641,  0.82352941, -0.72727273,  0.38709677,
        -0.176     , -0.90797546,  0.11764706,  0.1       ,  0.        ],
       [ 1.28915663, -0.23076923, -0.11764706,  1.09090909,  0.12903226,
        -0.464     , -0.61349693, -0.15686275,  0.4       ,  0.63157895],
       [ 0.46987952,  0.38461538, -0.43137255, -0.36363636, -0.51612903,
         0.24      , -1.10429448, -0.43137255, -0.4       ,  0.63157895],
       [ 1.19277108, -0.38461538, -0.2745098 ,  0.        ,  0.12903226,
        -0.24      , -2.03680982,  0.74509804,  0.1       ,  0.63157895],
       [-0.30120482,  0.64102564,  0.58823529,  0.72727273, -0.25806452,
         0.08      ,  2.72392638, -0.03921569,  0.        , -1.05263158],
       [-0.30120482,  0.58974359,  0.50980392,  0.72727273,  4.25806452,
         0.784     ,  1.66871166,  0.82352941, -0.3       ,  0.63157895],
       [ 0.73493976, -1.1025641 , -0.50980392, -1.81818182,  0.12903226,
         0.4       , -0.07361963,  0.39215686,  2.3       ,  0.21052632],
       [ 0.46987952,  0.79487179, -0.35294118,  0.72727273,  0.25806452,
         1.008     , -0.17177914,  1.33333333, -0.1       , -3.78947368],
       [-0.15662651, -0.38461538,  0.66666667,  0.        ,  0.90322581,
        -0.048     ,  0.29447853,  2.11764706,  0.3       , -1.05263158],
       [-0.20481928, -1.05128205, -0.03921569, -0.36363636,  1.03225806,
        -0.496     ,  0.36809816,  0.8627451 , -0.4       ,  0.21052632],
       [-0.25301205,  1.        ,  0.03921569,  1.09090909, -0.64516129,
         0.336     , -0.12269939, -0.15686275, -0.1       ,  0.63157895],
       [-0.27710843,  0.64102564, -0.74509804,  0.72727273,  3.22580645,
         1.232     ,  0.58895706, -0.47058824,  1.8       ,  0.21052632],
       [-0.46987952, -0.79487179, -3.49019608, -0.72727273, -0.12903226,
        -0.624     ,  0.29447853, -0.11764706, -0.3       , -0.84210526],
       [ 0.85542169, -1.76923077,  0.35294118, -1.81818182, -0.25806452,
        -0.72      , -0.46625767,  1.7254902 ,  2.6       ,  0.21052632],
       [-0.20481928,  0.23076923,  0.35294118,  0.36363636, -0.25806452,
         0.4       , -0.5398773 ,  0.2745098 ,  2.1       ,  0.21052632],
       [-0.15662651, -0.38461538,  0.50980392, -0.36363636,  0.25806452,
        -1.2       ,  0.04907975, -0.50980392,  0.8       ,  0.21052632],
```

Step 6: Outliers removal

Outlier is one of the most faced issues which can easily give misleading statistical results and as well knock down the model performance if not taken care. It is one of the essential parts of data seen frequently by a data scientist and most of them aren't sure how to deal with it.

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.
An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set. An outlier can cause serious problems in statistical analyses.
Types of outliers

1. **Global outlier :** A data can be considered anomalous with respect to the entire data, if its value is far outside the entire dataset . For example, Intrusion detection in computer networks.
2. **Contextual outlier :** When an individual data instance is anomalous in a specific context or condition (but not otherwise). e.g., time & location, temperature.

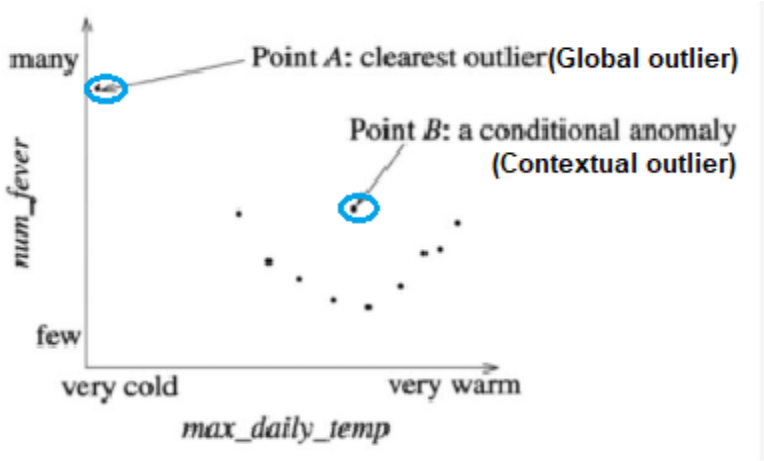


Fig. Global and contextual outlier

3. Collective outlier : When a collection of data points is anomalous with respect to the entire data set, it is termed as a collective outlier.

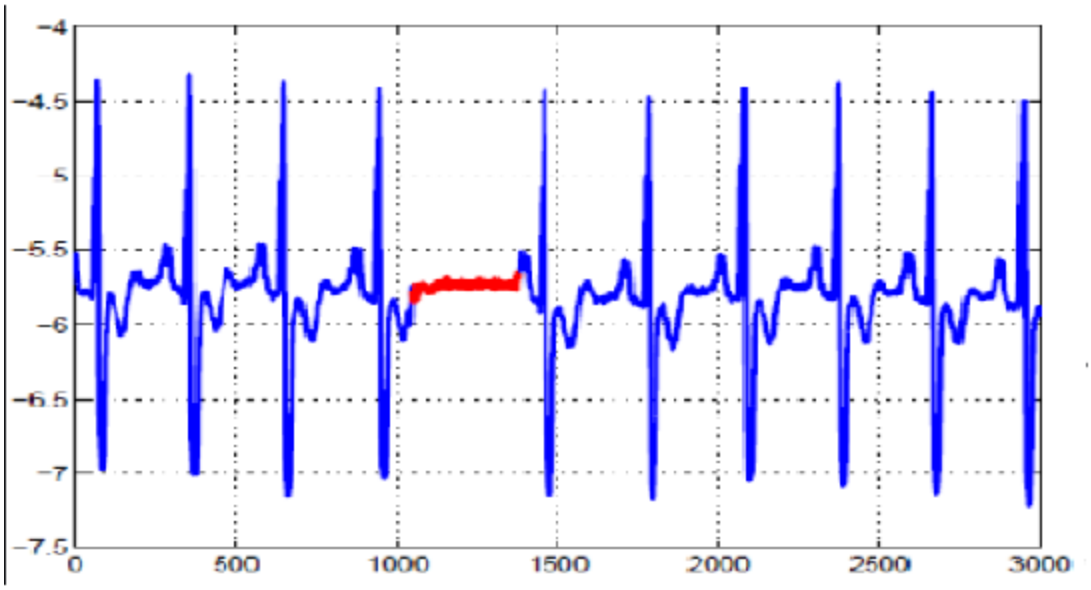


Fig. A human ECG report showing collective outlier

Outliers detection methods

1. **Statistical methods**
    - **Gaussian distribution method**
      If we know that the distribution of data is Gaussian or Gaussian like in simple words is your data normally distributed? If ans is yes then we can use the standard deviation of the data values to identify the outliers by applying the simple property of normal distribution.

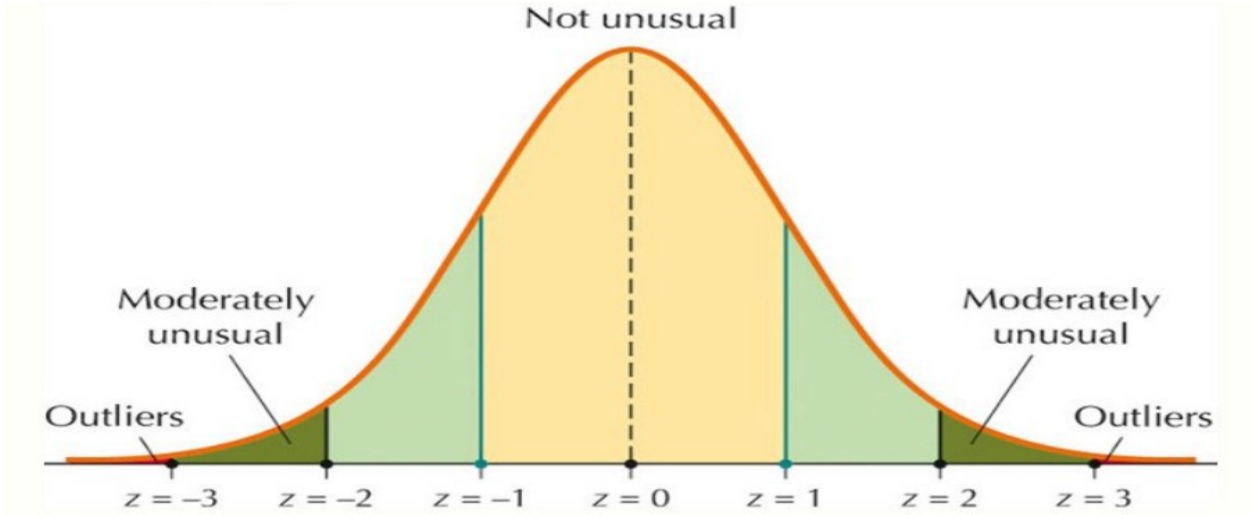


Fig. Gaussian distribution or normal distribution

For recap of normal distribution data coverage :

- 1 Standard Deviation(1 SD)from the Mean: 68%
- 2 Standard Deviations(2 SD) from the Mean: 95%
- 3 Standard Deviations(3 SD) from the Mean: 99.7%

Most of the sample values are covered in 3 SD, if value falls outside it can be considered an outlier which is an unlikely or rare event at approximately 1 of 400 samples. The 3 SD is a general rule it can be increased or decreased based on the type of problem worked, suppose if 99.9% data coverage is needed consider 4 SD instead. Let's also see how we can code it

```
# identify outliers with standard deviation
import numpy as np
from numpy.random import randn

# seed the random number generator
np.random.seed(42)

# generate univariate observations
sample = 8 * randn(90000) + 100

# calculate summary statistics
sample_mean, sample_std = np.mean(sample), np.std(sample)

# identify outliers
cut_off = sample_std * 3
lower, upper = sample_mean - cut_off, sample_mean + cut_off

# identify outliers
outliers = [x for x in sample if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))

# remove outliers
outliers_removed = [x for x in sample if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

OUTPUT:

```
Identified outliers: 232
Non-outlier observations: 89768
```

This is a one-dimensional data we saw our implementation, but what if data in is hyper-plane (n-dimensional data)? Well ans is simple we will follow same approach, imagine a 2-D data showing eclipse shape, consider a cut-off for boundary and any points lying outside the boundary will be considered an outlier.

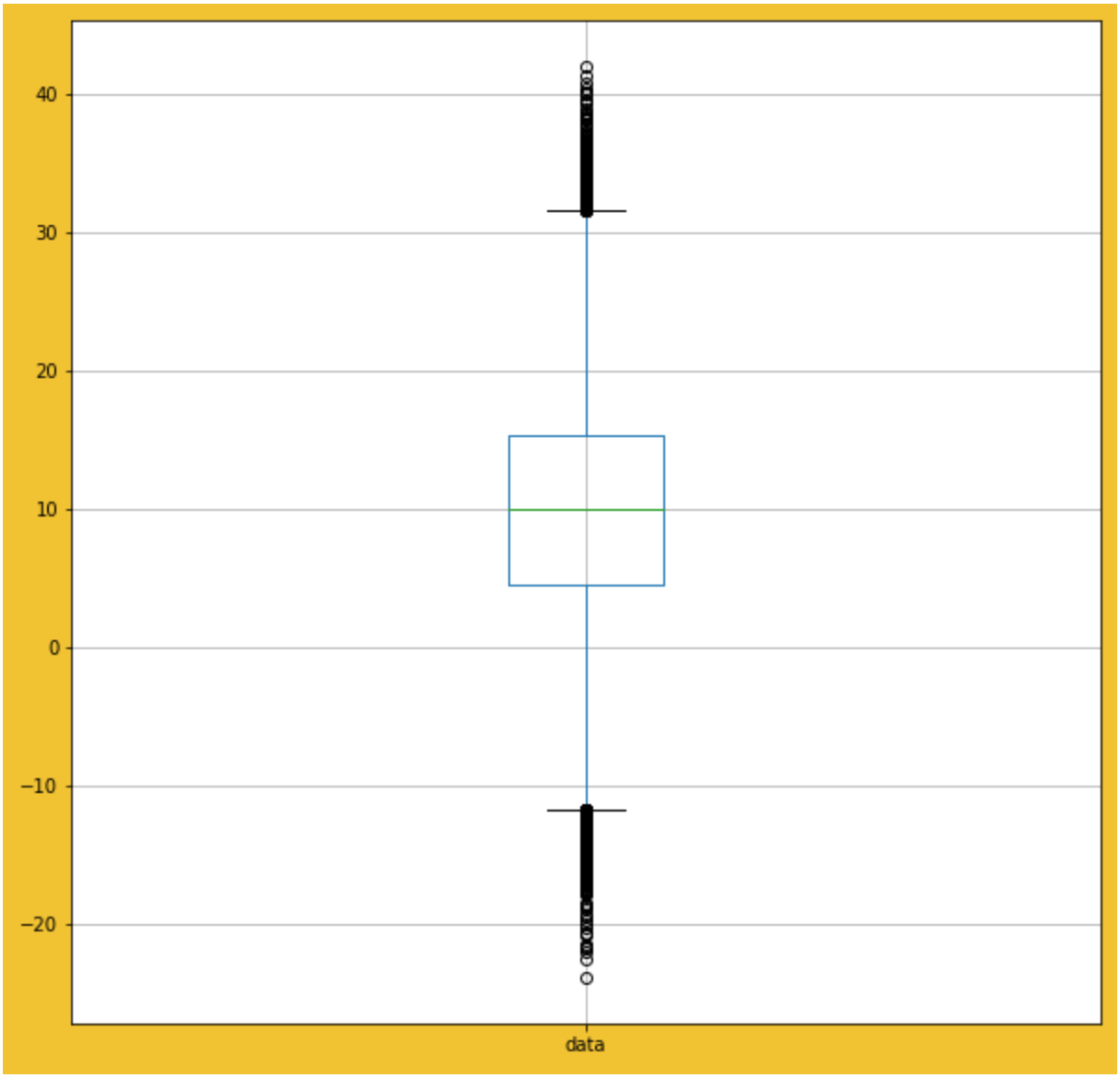
- Box-plot or Interquartile range(IQR) method

Not all data follow normal distribution, in that case we can use IQR method for the sample data. Let's see a python implementation example as well.

```
# identify outliers with interquartile range
import numpy as np
from numpy.random import randn
import pandas as pd
import matplotlib.pyplot as plt

# generate univariate observations
data =  8 * randn(90000) + 10

# calculate interquartile range
q25, q75 = np.percentile(data, 25), np.percentile(data, 75)
iqr = q75 - q25
print(f'Percentiles: 25th={np.round(q25)}, 75th={np.round(q75)}, IQR={np.round(iqr)}')

# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off

# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))

# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))

df = pd.DataFrame(data, columns=['data'])
boxplot = df.boxplot(column=['data'], figsize=(10,10), return_type='axes')
plt.show()
```

OUTPUT:

```
Percentiles: 25th=5.0, 75th=15.0, IQR=11.0
Identified outliers: 627
Non-outlier observations: 89373
```



2. Machine learning algorithm

- **DBSCAN**
  It is a density-based clustering approach ,very commonly used in unsupervised data, and not an outlier detection method per-se but due to algorithm approach used .It forms clusters based on radius and neighbor count and any point outside cluster is considered outlier. Core points -points that have a minimum of points in their surrounding- and points that are close enough to those core points together form a cluster.
- The algorithm has two parameters (epsilon: length scale, and min_samples: the minimum number of samples required for a point to be a core point). Finding a good epsilon is critical.

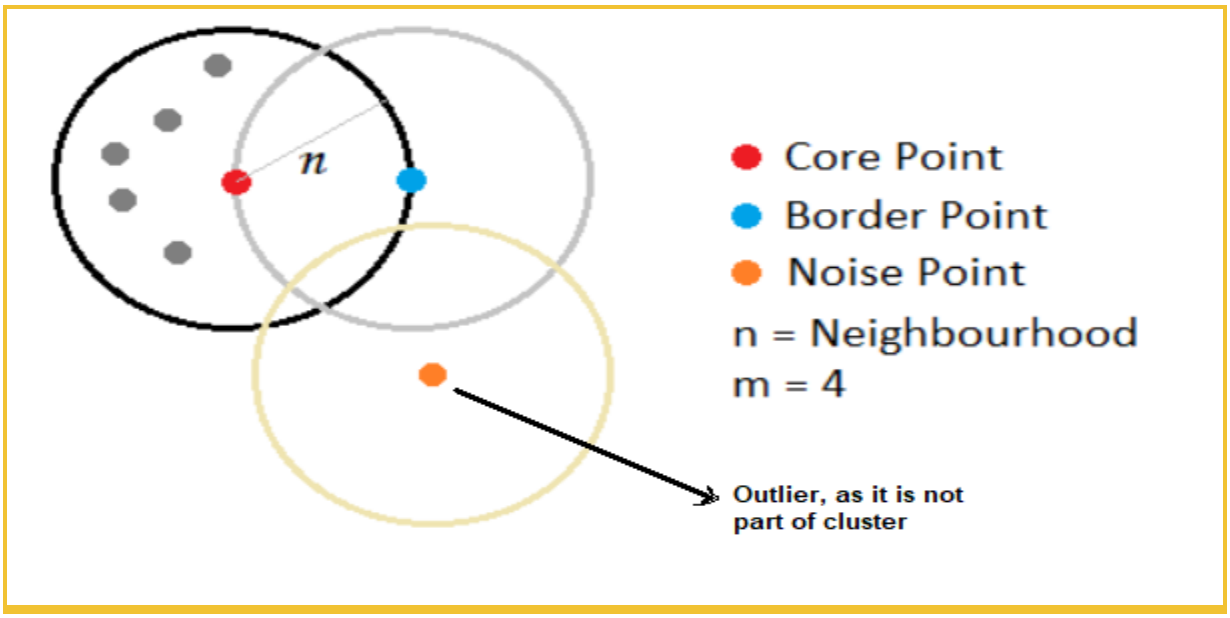
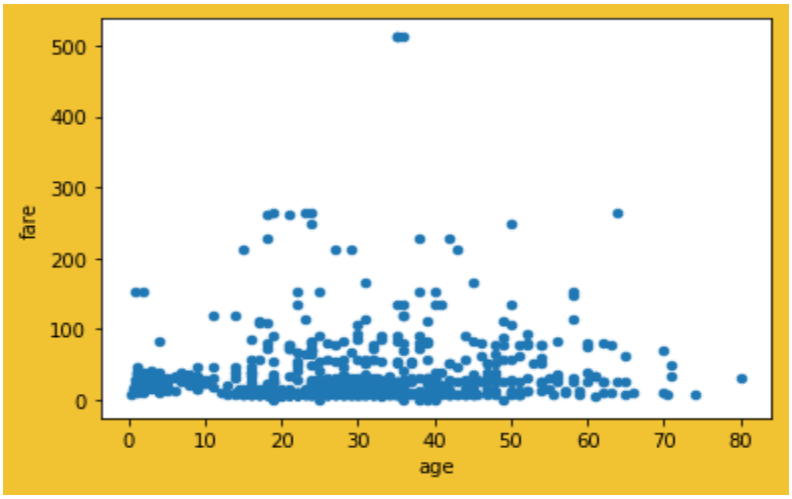


Fig. DBSCAN algorithm pictorial representation

```
import seaborn as sns
import pandas as pd
import numpy as np
titanic = sns.load_dataset('titanic')

ageAndFare = titanic[["age", "fare"]]
ageAndFare.plot.scatter(x = "age", y = "fare")
```

```
Out[8]:
<matplotlib.axes._subplots.AxesSubplot at 0x280b808cfd0>
```



```
ageAndFare.isnull().sum()
```

```
Out[16]:
age     177
fare      0
dtype: int64
```

```
mean = np.mean(ageAndFare.age)
ageAndFare.age.fillna(value=mean, inplace =True)
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
ageAndFare = scaler.fit_transform(ageAndFare)
ageAndFare = pd.DataFrame(ageAndFare, columns = ["age", "fare"])
```

```
from sklearn.cluster import DBSCAN
outlier_detection = DBSCAN(
  eps = 0.5,
  metric="euclidean",
  min_samples = 3,
  n_jobs = -1)
clusters = outlier_detection.fit_predict(ageAndFare)
clusters
```

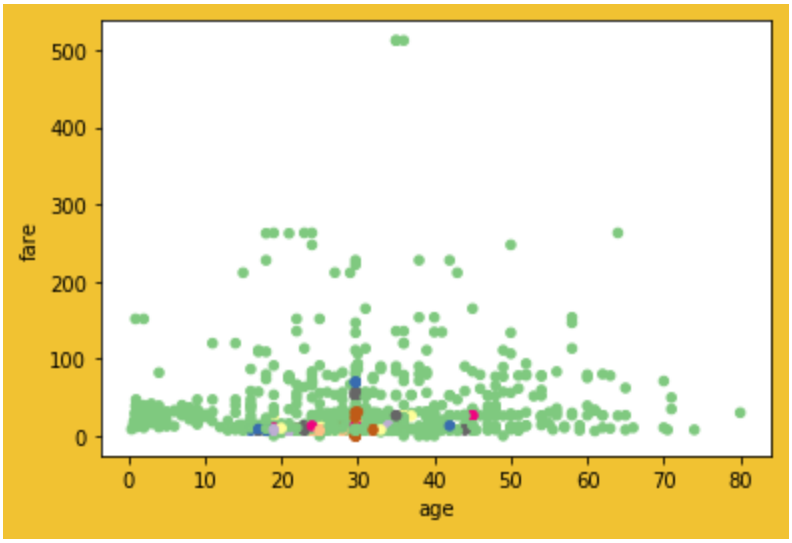
```
Out[25]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

As we see from output of cluster a

```
from matplotlib import cm
cmap = cm.get_cmap('Accent')
ageAndFare.plot.scatter(
  x = "age",
  y = "fare",
  c = clusters,
  cmap = cmap,
  colorbar = False
)
```

```
Out[23]:
<matplotlib.axes._subplots.AxesSubplot at 0x280b97284a8>
```



Step 6: Encoding the categorical data

Categorical encoding is a process of converting categories to numbers. It is very common to see categorical data which can be of type nominal or ordinal in your data set. Since Machine algorithm works on numbers only hence, it's important to transform it, the technique used is called encoding.

Let's study different kinds of encoding and its implementation on a Kaggle problem using python. You can find the dataset here. For example, I will consider First 20 rows for demonstrating.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)
print(df_train.shape)
df_train.head(20)
```

OUTPUT: (20, 25)

Out[11]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_9	ord_0	ord_1	ord_2	ord_3
0	0	0	0	0	T	Y	Green	Triangle	Snake	Finland	...	2f4cb3d51	2	Grandmaster	Cold	h
1	1	0	1	0	T	Y	Green	Trapezoid	Hamster	Russia	...	f83c56c21	1	Grandmaster	Hot	a
2	2	0	0	0	F	Y	Blue	Trapezoid	Lion	Russia	...	ae6800dd0	1	Expert	Lava Hot	h
3	3	0	1	0	F	Y	Red	Trapezoid	Snake	Canada	...	8270f0d71	1	Grandmaster	Boiling Hot	i
4	4	0	0	0	F	N	Red	Trapezoid	Lion	Canada	...	b164b72a7	1	Grandmaster	Freezing	a
5	5	0	1	1	T	N	Blue	Polygon	Lion	Costa Rica	...	51e27c16d	1	Novice	Freezing	j
6	6	0	1	1	T	N	Green	Trapezoid	Cat	China	...	7e3d79a0d	2	Grandmaster	Lava Hot	g
7	7	1	0	1	T	Y	Red	Triangle	Dog	Russia	...	feb72ecc2	1	Novice	Lava Hot	j
8	8	1	0	1	T	Y	Blue	Square	Hamster	Canada	...	34a7273bf	2	Novice	Boiling Hot	e
9	9	0	0	0	F	Y	Red	Trapezoid	Lion	China	...	0ece7a511	1	Expert	Freezing	h
10	10	0	1	0	T	Y	Green	Star	Snake	China	...	7f5e74721	1	Grandmaster	Boiling Hot	g
11	11	0	1	0	T	Y	Blue	Polygon	Snake	Russia	...	01dd99dc0	1	Novice	Freezing	i
12	12	0	1	0	F	N	Green	Polygon	Cat	China	...	e82ec9b46	3	Novice	Boiling Hot	g
13	13	1	1	1	F	Y	Blue	Trapezoid	Cat	India	...	cba64770e	1	Novice	Boiling Hot	i
14	14	0	0	1	T	Y	Blue	Triangle	Snake	India	...	3c915f463	1	Grandmaster	Boiling Hot	d
15	15	0	0	1	F	N	Green	Trapezoid	Lion	Russia	...	d445baa4d	1	Contributor	Hot	i
16	16	0	0	0	T	N	Red	Circle	Hamster	Russia	...	21ef6824e	1	Contributor	Boiling Hot	j
17	17	0	1	0	T	Y	Green	Star	Lion	Finland	...	332ab0429	1	Master	Cold	i
18	18	0	0	1	T	N	Blue	Square	Lion	China	...	2533756c6	1	Contributor	Freezing	g
19	19	0	0	1	F	N	Red	Polygon	Lion	Russia	...	a1f8ae96a	3	Grandmaster	Boiling Hot	b

20 rows × 25 columns

```
#Filtering Categorical data
df_cat = df_train.loc[:,df_train.dtypes==np.object]

df_cat.isnull().sum()
```

```
Out[13]:
bin_3    0
bin_4    0
nom_0    0
nom_1    0
nom_2    0
nom_3    0
nom_4    0
nom_5    0
nom_6    0
nom_7    0
```

```
nom_8    0
nom_9    0
ord_1    0
ord_2    0
ord_3    0
ord_4    0
ord_5    0
dtype: int64
```

```
df_cat.head(20)
```

Out[12]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	T	Y	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000ab	2f4cb3d51	Grandı
1	T	Y	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd920251	f83c56c21	Grandı
2	F	Y	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9f684	ae6800dd0	
3	F	Y	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6ab69	8270f0d71	Grandı
4	F	N	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab175	b164b72a7	Grandı
5	T	N	Blue	Polygon	Lion	Costa Rica	Oboe	46cab09da	29a854620	ff5b35098	b7e6f8e6f	51e27c16d	I
6	T	N	Green	Trapezoid	Cat	China	Piano	be5592604	3393a0f78	c6587685d	06f5ae149	7e3d79a0d	Grandı
7	T	Y	Red	Triangle	Dog	Russia	Oboe	72f8028dc	55eed5058	2dd9daf45	98addc2c9	feb72ecc2	I
8	T	Y	Blue	Square	Hamster	Canada	Bassoon	4604905e7	3e44d44eb	3f0057c9b	a2d110837	34a7273bf	I
9	F	Y	Red	Trapezoid	Lion	China	Piano	ad95dc0ee	8ed6221ae	4fbfe4a84	2c15d0173	0ece7a511	
10	T	Y	Green	Star	Snake	China	Piano	4604905e7	153316f52	b23e16a87	59bd5621f	7f5e74721	Grandı
11	T	Y	Blue	Polygon	Snake	Russia	Oboe	2ff007c26	6ea52a806	d31f87228	dcf843eef	01dd99dc0	I
12	F	N	Green	Polygon	Cat	China	Oboe	a35c346aa	1795ef28b	89f4255e4	7e01eda84	e82ec9b46	I
13	F	Y	Blue	Trapezoid	Cat	India	Oboe	dbfb714a4	628f3170f	5a99b7d9f	6e6d08b6f	cba64770e	I
14	T	Y	Blue	Triangle	Snake	India	Theremin	e1558b071	628f3170f	5ee5d882b	b80b1b531	3c915f463	Grandı
15	F	N	Green	Trapezoid	Lion	Russia	Piano	39647c92a	08b282a6c	00994f749	8a8ccfe81	d445baa4d	Contı
16	T	N	Red	Circle	Hamster	Russia	Oboe	ee55b9d67	395941181	91c26901a	078133e43	21ef6824e	Contı
17	T	Y	Green	Star	Lion	Finland	Theremin	416a8f3ab	d44f7245a	d2d8eabdb	19a7677f3	332ab0429	I
18	T	N	Blue	Square	Lion	China	Bassoon	91bde92fa	78500847e	5c843e08f	e588cff32	2533756c6	Contı
19	F	N	Red	Polygon	Lion	Russia	Piano	3aa9329e2	9448b8e3b	22ff79d81	dd3966979	a1f8ae96a	Grandı

#Method 1 : Label encoding

It's a very simple technique which converts each value in a column/feature of categorical type to number. Each category is assigned a unique label starting from 0 and going on till n_categories — 1 per feature. Label encoders are suited for a ordinal data as data denotes some order and encoding consider order in alphabetical way by default, you can also change order using dictionary/map which we will see in custom mapping , but using it for nominal data can be

misleading for example while converting a name feature to numeric can you say my name is numerically greater than yours? No right , One-hot encoding or dummy encoding will be a more appropriate approach for later which we will be discussing next. Let's consider a label encoding example.
Let’s say a feature talks about an animal's name which contains 5 animals: cat, cow, dog, horse, zebra. We are studying the height relation of these 5 animals. So given the order problem we can encode horse -> 0 , zebra ->1, cow -> 2 , dog ->3 and cat -> 4. Below is Label encoding using python

```
import pandas as pd
import numpy as np
df_train = pd.read_csv('train.csv',nrows=20)
print(df_train.shape)
```

OUTPUT: (20, 25)

```
df_cat.head(20)
```

Out[26]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	T	Y	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000ab	2f4cb3d51	Grandı
1	T	Y	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd920251	f83c56c21	Grandı
2	F	Y	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9f684	ae6800dd0	
3	F	Y	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6ab69	8270f0d71	Grandı
4	F	N	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab175	b164b72a7	Grandı
5	T	N	Blue	Polygon	Lion	Costa Rica	Oboe	46cab09da	29a854620	ff5b35098	b7e6f8e6f	51e27c16d	I
6	T	N	Green	Trapezoid	Cat	China	Piano	be5592604	3393a0f78	c6587685d	06f5ae149	7e3d79a0d	Grandı
7	T	Y	Red	Triangle	Dog	Russia	Oboe	72f8028dc	55eed5058	2dd9daf45	98addc2c9	feb72ecc2	I
8	T	Y	Blue	Square	Hamster	Canada	Bassoon	4604905e7	3e44d44eb	3f0057c9b	a2d110837	34a7273bf	I
9	F	Y	Red	Trapezoid	Lion	China	Piano	ad95dc0ee	8ed6221ae	4fbfe4a84	2c15d0173	0ece7a511	
10	T	Y	Green	Star	Snake	China	Piano	4604905e7	153316f52	b23e16a87	59bd5621f	7f5e74721	Grandı
11	T	Y	Blue	Polygon	Snake	Russia	Oboe	2ff007c26	6ea52a806	d31f87228	dcf843eef	01dd99dc0	I
12	F	N	Green	Polygon	Cat	China	Oboe	a35c346aa	1795ef28b	89f4255e4	7e01eda84	e82ec9b46	I
13	F	Y	Blue	Trapezoid	Cat	India	Oboe	dbfb714a4	628f3170f	5a99b7d9f	6e6d08b6f	cba64770e	I
14	T	Y	Blue	Triangle	Snake	India	Theremin	e1558b071	628f3170f	5ee5d882b	b80b1b531	3c915f463	Grandı
15	F	N	Green	Trapezoid	Lion	Russia	Piano	39647c92a	08b282a6c	00994f749	8a8ccfe81	d445baa4d	Conti
16	T	N	Red	Circle	Hamster	Russia	Oboe	ee55b9d67	395941181	91c26901a	078133e43	21ef6824e	Conti
17	T	Y	Green	Star	Lion	Finland	Theremin	416a8f3ab	d44f7245a	d2d8eabdb	19a7677f3	332ab0429	I
18	T	N	Blue	Square	Lion	China	Bassoon	91bde92fa	78500847e	5c843e08f	e588cff32	2533756c6	Conti
19	F	N	Red	Polygon	Lion	Russia	Piano	3aa9329e2	9448b8e3b	22ff79d81	dd3966979	a1f8ae96a	Grandı

```
from sklearn.preprocessing import LabelEncoder

%%time
label=LabelEncoder()

for col in  df_cat.columns:
    df_lable[col]=label.fit_transform(df_cat[col])

df_lable.head(20)
```

OUTPUT: Wall time: 3.51 ms

Out[25]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	ord_1	ord_2	ord_3	ord_4	or
0	1	1	1	5	4	3	0	7	8	10	14	4	2	1	5	1	
1	1	1	1	4	2	5	2	13	18	4	5	18	2	3	0	0	
2	0	1	0	4	3	5	3	1	1	13	18	13	1	4	5	8	
3	0	1	2	4	4	0	1	18	10	18	4	11	2	0	6	1	
4	0	0	2	4	3	0	1	8	4	1	15	14	2	2	0	8	
5	1	0	0	1	3	2	1	6	5	19	12	8	4	2	7	2	
6	1	0	1	4	0	1	2	14	6	15	0	9	2	4	4	6	
7	1	1	2	5	1	5	1	9	11	3	10	19	4	4	7	4	
8	1	1	0	2	2	0	0	5	9	5	11	6	4	0	3	9	
9	0	1	2	4	3	1	2	12	15	6	3	1	1	2	5	7	
10	1	1	1	3	4	1	2	5	2	14	6	10	2	0	4	8	
11	1	1	0	1	4	5	1	0	13	17	16	0	4	2	6	10	
12	0	0	1	1	0	1	1	11	3	11	8	17	4	0	4	8	
13	0	1	0	4	0	4	1	15	12	7	7	15	4	0	6	9	
14	1	1	0	5	4	4	3	16	12	9	13	7	2	0	2	5	
15	0	0	1	4	3	5	2	2	0	0	9	16	0	3	6	7	
16	1	0	2	0	2	5	1	17	7	12	1	2	0	0	7	3	
17	1	1	1	3	3	3	3	4	17	16	2	5	3	1	6	7	
18	1	0	0	2	3	1	0	10	14	8	19	3	0	2	4	9	
19	0	0	2	1	3	5	2	3	16	2	17	12	2	0	1	5	

#Method 2 : Replace or Custom Mapping

This encoding is particularly useful for ordinal variables where the order of categories is important and Label encoder will not help as it is only considered by alphabetical order but here we need our own order. Like the size of shirts from large to small will be XL>L>M>S>XS .To make sure that the learning algorithm interprets the ordinal variables correctly, we can map the categorical values to integer values manually. There are two ways to do so — one is to use map function and another is to use replace function as shown below.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)
df_cat = df_train.loc[:,df_train.dtypes==np.object]
print(df_train.shape)
```

OUTPUT: (20, 25)

```
df_cat.head(20)
```

Out[3]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	T	Y	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000ab	2f4cb3d51	Grandı
1	T	Y	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd920251	f83c56c21	Grandı
2	F	Y	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9f684	ae6800dd0	
3	F	Y	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6ab69	8270f0d71	Grandı
4	F	N	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab175	b164b72a7	Grandı
5	T	N	Blue	Polygon	Lion	Costa Rica	Oboe	46cab09da	29a854620	ff5b35098	b7e6f8e6f	51e27c16d	I
6	T	N	Green	Trapezoid	Cat	China	Piano	be5592604	3393a0f78	c6587685d	06f5ae149	7e3d79a0d	Grandı
7	T	Y	Red	Triangle	Dog	Russia	Oboe	72f8028dc	55eed5058	2dd9daf45	98addc2c9	feb72ecc2	I
8	T	Y	Blue	Square	Hamster	Canada	Bassoon	4604905e7	3e44d44eb	3f0057c9b	a2d110837	34a7273bf	I
9	F	Y	Red	Trapezoid	Lion	China	Piano	ad95dc0ee	8ed6221ae	4fbfe4a84	2c15d0173	0ece7a511	
10	T	Y	Green	Star	Snake	China	Piano	4604905e7	153316f52	b23e16a87	59bd5621f	7f5e74721	Grandı
11	T	Y	Blue	Polygon	Snake	Russia	Oboe	2ff007c26	6ea52a806	d31f87228	dcf843eef	01dd99dc0	I
12	F	N	Green	Polygon	Cat	China	Oboe	a35c346aa	1795ef28b	89f4255e4	7e01eda84	e82ec9b46	I
13	F	Y	Blue	Trapezoid	Cat	India	Oboe	dbfb714a4	628f3170f	5a99b7d9f	6e6d08b6f	cba64770e	I
14	T	Y	Blue	Triangle	Snake	India	Theremin	e1558b071	628f3170f	5ee5d882b	b80b1b531	3c915f463	Grandı
15	F	N	Green	Trapezoid	Lion	Russia	Piano	39647c92a	08b282a6c	00994f749	8a8ccfe81	d445baa4d	Conti
16	T	N	Red	Circle	Hamster	Russia	Oboe	ee55b9d67	395941181	91c26901a	078133e43	21ef6824e	Conti
17	T	Y	Green	Star	Lion	Finland	Theremin	416a8f3ab	d44f7245a	d2d8eabdb	19a7677f3	332ab0429	I
18	T	N	Blue	Square	Lion	China	Bassoon	91bde92fa	78500847e	5c843e08f	e588cff32	2533756c6	Conti
19	F	N	Red	Polygon	Lion	Russia	Piano	3aa9329e2	9448b8e3b	22ff79d81	dd3966979	a1f8ae96a	Grandı

```
print(df_cat.nom_0.unique())
print(df_cat.nom_1.unique())
print(df_cat.nom_2.unique())
print(df_cat.nom_3.unique())
print(df_cat.nom_4.unique())
```

OUTPUT:

```
['Green' 'Blue' 'Red']
['Triangle' 'Trapezoid' 'Polygon' 'Square' 'Star' 'Circle']
['Snake' 'Hamster' 'Lion' 'Cat' 'Dog']
['Finland' 'Russia' 'Canada' 'Costa Rica' 'China' 'India']
['Bassoon' 'Piano' 'Theremin' 'Oboe']
```

```
%%time
map_nom0 = {
    "Red":0,
    "Blue":1,
    "Green":2
}
map_nom1 = {
    "Triangle":0,
    "Trapezoid":1,
    "Polygon":2,
    "Square":3,
    "Star":4,
    "Circle":5
}
map_nom4 = {
    "Bassoon":5,
    "Piano":6,
    "Theremin":7,
    "Oboe":8
}

replace_num_2_3 = {"nom_2": {"Snake": 2, "Hamster": 4, "Lion": 6, "Cat": 8, "Dog": 10},
                   "nom_3": {"Finland": 1, "Russia": 3, "Canada": 5, "Costa Rica": 7, "China": 9, "India":
11}}

# # Using Map
df_cat.nom_0 = df_cat.nom_0.map(map_nom0)
df_cat.nom_1 = df_cat.nom_1.map(map_nom1)
df_cat.nom_4 = df_cat.nom_4.map(map_nom4)
```

```
# Using Replace
df_cat.replace(replace_num_2_3, inplace=True)
```

```
Wall time: 80.3 ms
C:\D\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:5096:
SettingWithCopyWarning:
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:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self[name] = value
C:\D\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:6517:
SettingWithCopyWarning:
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:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  regex=regex)
```

```
df_cat.iloc[:,2:7].head(20)
```

Out[6]:

	nom_0	nom_1	nom_2	nom_3	nom_4
0	2	0	2	1	5
1	2	1	4	3	6
2	1	1	6	3	7
3	0	1	2	5	8
4	0	1	6	5	8
5	1	2	6	7	8
6	2	1	8	9	6
7	0	0	10	3	8
8	1	3	4	5	5
9	0	1	6	9	6
10	2	4	2	9	6
11	1	2	2	3	8
12	2	2	8	9	8
13	1	1	8	11	8
14	1	0	2	11	7
15	2	1	6	3	6
16	0	5	4	3	8
17	2	4	6	1	7
18	1	3	6	9	5
19	0	2	6	3	6

#Method 3 : One-hot encoding and dummy variables

One-hot encoding is one of the most widely used encoding schemes. It works by creating a new column for each category present in the feature and assigning a 1 or 0 to indicate the presence of a category in the data. A separate column is created for each possible value. A value of 1 in a column represents the presence of that level in the original data.

Pandas get_dummies method can be applied to a data frame and will only convert string columns into numbers and leave all others as it is. An alternative to get_dummies method is sklearn's OneHotEncoder function which returns an array. This data frame should be concatenated with the original data frame.

One hot encoding performs better when the category cardinality of a feature is not too high. Because for features with high cardinality, one hot encoding would create a large number of columns which could lead to memory problems.

Note : pd.get_dummies results in a Pandas DataFrame whereas OneHotEncoder results in a SciPy sparse matrix. Let's go through implementation of both.

```
import pandas as pd
import numpy as np
df_train = pd.read_csv('train.csv',nrows=20)
df_cat = df_train.loc[:,df_train.dtypes==np.object]
print(df_train.shape)
```

OUTPUT: (20, 25)

```
df_cat.head(20)
```

Out[3]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	T	Y	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000ab	2f4cb3d51	Grandı
1	T	Y	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd920251	f83c56c21	Grandı
2	F	Y	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9f684	ae6800dd0	
3	F	Y	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6ab69	8270f0d71	Grandı
4	F	N	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab175	b164b72a7	Grandı
5	T	N	Blue	Polygon	Lion	Costa Rica	Oboe	46cab09da	29a854620	ff5b35098	b7e6f8e6f	51e27c16d	I
6	T	N	Green	Trapezoid	Cat	China	Piano	be5592604	3393a0f78	c6587685d	06f5ae149	7e3d79a0d	Grandı
7	T	Y	Red	Triangle	Dog	Russia	Oboe	72f8028dc	55eed5058	2dd9daf45	98addc2c9	feb72ecc2	I
8	T	Y	Blue	Square	Hamster	Canada	Bassoon	4604905e7	3e44d44eb	3f0057c9b	a2d110837	34a7273bf	I
9	F	Y	Red	Trapezoid	Lion	China	Piano	ad95dc0ee	8ed6221ae	4fbfe4a84	2c15d0173	0ece7a511	
10	T	Y	Green	Star	Snake	China	Piano	4604905e7	153316f52	b23e16a87	59bd5621f	7f5e74721	Grandı
11	T	Y	Blue	Polygon	Snake	Russia	Oboe	2ff007c26	6ea52a806	d31f87228	dcf843eef	01dd99dc0	I
12	F	N	Green	Polygon	Cat	China	Oboe	a35c346aa	1795ef28b	89f4255e4	7e01eda84	e82ec9b46	I
13	F	Y	Blue	Trapezoid	Cat	India	Oboe	dbfb714a4	628f3170f	5a99b7d9f	6e6d08b6f	cba64770e	I
14	T	Y	Blue	Triangle	Snake	India	Theremin	e1558b071	628f3170f	5ee5d882b	b80b1b531	3c915f463	Grandı
15	F	N	Green	Trapezoid	Lion	Russia	Piano	39647c92a	08b282a6c	00994f749	8a8ccfe81	d445baa4d	Conti
16	T	N	Red	Circle	Hamster	Russia	Oboe	ee55b9d67	395941181	91c26901a	078133e43	21ef6824e	Conti
17	T	Y	Green	Star	Lion	Finland	Theremin	416a8f3ab	d44f7245a	d2d8eabdb	19a7677f3	332ab0429	I
18	T	N	Blue	Square	Lion	China	Bassoon	91bde92fa	78500847e	5c843e08f	e588cff32	2533756c6	Conti
19	F	N	Red	Polygon	Lion	Russia	Piano	3aa9329e2	9448b8e3b	22ff79d81	dd3966979	a1f8ae96a	Grandı

```
#Using one hot encoding
from sklearn.preprocessing import OneHotEncoder
%%time

OHE=OneHotEncoder()
OHE.fit(df_cat)
df_ohe = OHE.transform(df_cat)

print('df_cat data set has got {} rows and {} columns'.format(df_cat.shape[0],df_cat.shape[1]))
print('df_ohe data set has got {} rows and {} columns'.format(df_ohe.shape[0],df_ohe.shape[1]))
```

OUTPUT:

```
df_cat data set has got 20 rows and 17 columns
df_ohe data set has got 20 rows and 175 columns
Wall time: 3.11 ms
```

```
# using pandas inbuilt function , get_dummies
df_dummies = pd.get_dummies(data=df_cat, columns = ['nom_0','ord_3'], drop_first=False)
df_dummies.iloc[:,15:]
```

Out[23]:

	nom_0_0	nom_0_1	nom_0_2	ord_3_a	ord_3_b	ord_3_d	ord_3_e	ord_3_g	ord_3_h	ord_3_i	ord_3_j
0	0	0	1	0	0	0	0	0	1	0	0
1	0	0	1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	0	0	1	0
4	1	0	0	1	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0	0	1
6	0	0	1	0	0	0	0	1	0	0	0
7	1	0	0	0	0	0	0	0	0	0	1
8	0	1	0	0	0	0	1	0	0	0	0
9	1	0	0	0	0	0	0	0	1	0	0
10	0	0	1	0	0	0	0	1	0	0	0
11	0	1	0	0	0	0	0	0	0	1	0
12	0	0	1	0	0	0	0	1	0	0	0
13	0	1	0	0	0	0	0	0	0	1	0
14	0	1	0	0	0	1	0	0	0	0	0
15	0	0	1	0	0	0	0	0	0	1	0
16	1	0	0	0	0	0	0	0	0	0	1
17	0	0	1	0	0	0	0	0	0	1	0
18	0	1	0	0	0	0	0	1	0	0	0
19	1	0	0	0	1	0	0	0	0	0	0

#Method 4 : Feature Hashing

It's an alternative to one-hot encoding and overcomes the major disadvantage of increasing feature size. It represents data as a sparse matrix but with much lower dimensions. In feature hashing we apply a hashing function to the category and then represent it by its indices. Example, if we choose a dimension of 5 to represent “London” we will calculate Hash(London) mod 5 = 3 (for example) so London representation will be (0,0,1,0,0). Let’s see its implementation using sklearn.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)
df_cat = df_train.loc[:,df_train.dtypes==np.object]
print(df_train.shape)
```

OUTPUT: (20, 25)

```
#Using FeatureHasher
from sklearn.feature_extraction import FeatureHasher

df_hash = df_cat
for c in df_cat.columns:
    df_hash[c]=df_cat[c].astype('str')

hashing = FeatureHasher(input_type='string')
df_hash = hashing.transform(df_hash.values)

print('df_hash data set has got {} rows and {}
columns'.format(df_hash.shape[0],df_hash.shape[1]))
```

```
C:\D\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:6:
SettingWithCopyWarning:
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:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

OUTPUT: df_hash data set has got 20 rows and 1048576 columns

```
df_hash
```

Out[36]:
<20x1048576 sparse matrix of type '<class 'numpy.float64'>'
with 340 stored elements in Compressed Sparse Row format>

#Method 5 : Encoding categories with dataset statistics
Statistics basically gives a count/frequency summary of a value or variable, the same concept is extended for this encoding.
It replaces every category with the number of times that we saw it in the dataset. For instance ,if "Paris" and "London" are both big cities, they will probably both appear many times in our dataset and the model will know that they are similar.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)
df_cat = df_train.loc[:,df_train.dtypes==np.object]
print(df_train.shape)
```

OUTPUT: (20, 25)

```
df_cat.head(20)
```

Out[40]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	T	Y	Green	Triangle	Snake	Finland	Bassoon	50f116bcf	3ac1b8814	68f6ad3e9	c389000ab	2f4cb3d51	Grandı
1	T	Y	Green	Trapezoid	Hamster	Russia	Piano	b3b4d25d0	fbcb50fc1	3b6dd5612	4cd920251	f83c56c21	Grandı
2	F	Y	Blue	Trapezoid	Lion	Russia	Theremin	3263bdce5	0922e3cb8	a6a36f527	de9c9f684	ae6800dd0	
3	F	Y	Red	Trapezoid	Snake	Canada	Oboe	f12246592	50d7ad46a	ec69236eb	4ade6ab69	8270f0d71	Grandı
4	F	N	Red	Trapezoid	Lion	Canada	Oboe	5b0f5acd5	1fe17a1fd	04ddac2be	cb43ab175	b164b72a7	Grandı
5	T	N	Blue	Polygon	Lion	Costa Rica	Oboe	46cab09da	29a854620	ff5b35098	b7e6f8e6f	51e27c16d	I
6	T	N	Green	Trapezoid	Cat	China	Piano	be5592604	3393a0f78	c6587685d	06f5ae149	7e3d79a0d	Grandı
7	T	Y	Red	Triangle	Dog	Russia	Oboe	72f8028dc	55eed5058	2dd9daf45	98addc2c9	feb72ecc2	I
8	T	Y	Blue	Square	Hamster	Canada	Bassoon	4604905e7	3e44d44eb	3f0057c9b	a2d110837	34a7273bf	I
9	F	Y	Red	Trapezoid	Lion	China	Piano	ad95dc0ee	8ed6221ae	4fbfe4a84	2c15d0173	0ece7a511	
10	T	Y	Green	Star	Snake	China	Piano	4604905e7	153316f52	b23e16a87	59bd5621f	7f5e74721	Grandı
11	T	Y	Blue	Polygon	Snake	Russia	Oboe	2ff007c26	6ea52a806	d31f87228	dcf843eef	01dd99dc0	I
12	F	N	Green	Polygon	Cat	China	Oboe	a35c346aa	1795ef28b	89f4255e4	7e01eda84	e82ec9b46	I
13	F	Y	Blue	Trapezoid	Cat	India	Oboe	dbfb714a4	628f3170f	5a99b7d9f	6e6d08b6f	cba64770e	I
14	T	Y	Blue	Triangle	Snake	India	Theremin	e1558b071	628f3170f	5ee5d882b	b80b1b531	3c915f463	Grandı
15	F	N	Green	Trapezoid	Lion	Russia	Piano	39647c92a	08b282a6c	00994f749	8a8ccfe81	d445baa4d	Cont
16	T	N	Red	Circle	Hamster	Russia	Oboe	ee55b9d67	395941181	91c26901a	078133e43	21ef6824e	Cont
17	T	Y	Green	Star	Lion	Finland	Theremin	416a8f3ab	d44f7245a	d2d8eabdb	19a7677f3	332ab0429	I
18	T	N	Blue	Square	Lion	China	Bassoon	91bde92fa	78500847e	5c843e08f	e588cff32	2533756c6	Cont
19	F	N	Red	Polygon	Lion	Russia	Piano	3aa9329e2	9448b8e3b	22ff79d81	dd3966979	a1f8ae96a	Grandı

```
%%time

df_stat=df_cat.copy()
for col in df_stat.columns:
    df_stat[col]=df_stat[col].astype('category')
    counts=df_stat[col].value_counts()
    counts=counts.sort_index()
    counts=counts.fillna(0)
    counts += np.random.rand(len(counts))/1000
    df_stat[col].cat.categories=counts
```

Wall time: 60.9 ms

```
df_stat.head(20)
```

Out[41]:

	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	nom_7	nom_8	nom_9	
0	12.000327	12.000243	7.000724	3.000835	5.000499	2.000804	3.000655	1.000698	1.000015	1.000482	1.000693	1.000930	8
1	12.000327	12.000243	7.000724	8.000105	3.000514	7.000607	6.000544	1.000020	1.000691	1.000641	1.000175	1.000056	8
2	8.000704	12.000243	7.000958	8.000105	8.000825	7.000607	3.000310	1.000554	1.000656	1.000195	1.000934	1.000371	2
3	8.000704	12.000243	6.000535	8.000105	5.000499	3.000752	8.000144	1.000061	1.000262	1.000220	1.000570	1.000309	8
4	8.000704	8.000400	6.000535	8.000105	8.000825	3.000752	8.000144	1.000724	1.000374	1.000591	1.000105	1.000807	8
5	12.000327	8.000400	7.000958	4.000535	8.000825	1.000411	8.000144	1.000091	1.000508	1.000856	1.000988	1.000345	6
6	12.000327	8.000400	7.000724	8.000105	3.000888	5.000678	6.000544	1.000524	1.000215	1.000924	1.000823	1.000616	8
7	12.000327	12.000243	6.000535	3.000835	1.000658	7.000607	8.000144	1.000276	1.000676	1.000617	1.000482	1.000550	6
8	12.000327	12.000243	7.000958	2.000834	3.000514	3.000752	3.000655	2.000916	1.000877	1.000988	1.000457	1.000255	6
9	8.000704	12.000243	6.000535	8.000105	8.000825	5.000678	6.000544	1.000615	1.000815	1.000763	1.000826	1.000290	2
10	12.000327	12.000243	7.000724	2.000630	5.000499	5.000678	6.000544	2.000916	1.000863	1.000576	1.000424	1.000768	8
11	12.000327	12.000243	7.000958	4.000535	5.000499	7.000607	8.000144	1.000947	1.000671	1.000673	1.000149	1.000726	6
12	8.000704	8.000400	7.000724	4.000535	3.000888	5.000678	8.000144	1.000870	1.000631	1.000273	1.000323	1.000592	6
13	8.000704	12.000243	7.000958	8.000105	3.000888	2.000637	8.000144	1.000843	2.000656	1.000158	1.000027	1.000247	6
14	12.000327	12.000243	7.000958	3.000835	5.000499	2.000637	3.000310	1.000884	2.000656	1.000939	1.000121	1.000973	8
15	8.000704	8.000400	7.000724	8.000105	8.000825	7.000607	6.000544	1.000715	1.000387	1.000885	1.000807	1.000516	3
16	12.000327	8.000400	6.000535	1.000392	3.000514	7.000607	8.000144	1.000237	1.000056	1.000284	1.000155	1.000911	3
17	12.000327	12.000243	7.000724	2.000630	8.000825	2.000804	3.000310	1.000473	1.000060	1.000533	1.000214	1.000609	1
18	12.000327	8.000400	7.000958	2.000834	8.000825	5.000678	3.000655	1.000035	1.000625	1.000595	1.000984	1.000067	3
19	8.000704	8.000400	6.000535	4.000535	8.000825	7.000607	6.000544	1.000473	1.000870	1.000619	1.000625	1.000806	8

#Method 6 : Encoding cyclic features

Data like time-series, sales, seasonal related , and contains features like time, months, days, weekdays, hours, minutes, seconds etc.

Here simple trigonometry will help us to transform the data, a common method for encoding cyclical data is to transform the data into two dimensions using a sine and cosine transformation.

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)

df_train.head(20)
```

Out[6]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_9	ord_0	ord_1	ord_2	ord_3
0	0	0	0	0	T	Y	Green	Triangle	Snake	Finland	...	2f4cb3d51	2	Grandmaster	Cold	h
1	1	0	1	0	T	Y	Green	Trapezoid	Hamster	Russia	...	f83c56c21	1	Grandmaster	Hot	a
2	2	0	0	0	F	Y	Blue	Trapezoid	Lion	Russia	...	ae6800dd0	1	Expert	Lava Hot	h
3	3	0	1	0	F	Y	Red	Trapezoid	Snake	Canada	...	8270f0d71	1	Grandmaster	Boiling Hot	i
4	4	0	0	0	F	N	Red	Trapezoid	Lion	Canada	...	b164b72a7	1	Grandmaster	Freezing	a
5	5	0	1	1	T	N	Blue	Polygon	Lion	Costa Rica	...	51e27c16d	1	Novice	Freezing	j
6	6	0	1	1	T	N	Green	Trapezoid	Cat	China	...	7e3d79a0d	2	Grandmaster	Lava Hot	g
7	7	1	0	1	T	Y	Red	Triangle	Dog	Russia	...	feb72ecc2	1	Novice	Lava Hot	j
8	8	1	0	1	T	Y	Blue	Square	Hamster	Canada	...	34a7273bf	2	Novice	Boiling Hot	e
9	9	0	0	0	F	Y	Red	Trapezoid	Lion	China	...	0ece7a511	1	Expert	Freezing	h
10	10	0	1	0	T	Y	Green	Star	Snake	China	...	7f5e74721	1	Grandmaster	Boiling Hot	g
11	11	0	1	0	T	Y	Blue	Polygon	Snake	Russia	...	01dd99dc0	1	Novice	Freezing	i
12	12	0	1	0	F	N	Green	Polygon	Cat	China	...	e82ec9b46	3	Novice	Boiling Hot	g
13	13	1	1	1	F	Y	Blue	Trapezoid	Cat	India	...	cba64770e	1	Novice	Boiling Hot	i
14	14	0	0	1	T	Y	Blue	Triangle	Snake	India	...	3c915f463	1	Grandmaster	Boiling Hot	d
15	15	0	0	1	F	N	Green	Trapezoid	Lion	Russia	...	d445baa4d	1	Contributor	Hot	i
16	16	0	0	0	T	N	Red	Circle	Hamster	Russia	...	21ef6824e	1	Contributor	Boiling Hot	j
17	17	0	1	0	T	Y	Green	Star	Lion	Finland	...	332ab0429	1	Master	Cold	i
18	18	0	0	1	T	N	Blue	Square	Lion	China	...	2533756c6	1	Contributor	Freezing	g
19	19	0	0	1	F	N	Red	Polygon	Lion	Russia	...	a1f8ae96a	3	Grandmaster	Boiling Hot	b

20 rows × 25 columns

```
%%time

df_cyclic = df_train.copy()
columns=['day','month']
for col in columns:
    df_cyclic[col+'_sin']=np.sin((2*np.pi*df_cyclic[col])/max(df_cyclic[col]))
    df_cyclic[col+'_cos']=np.cos((2*np.pi*df_cyclic[col])/max(df_cyclic[col]))
df_cyclic = df_cyclic.drop(columns,axis=1)

df_cyclic[['day_sin','day_cos']].head(20)
```

```
Wall time: 5.95 ms
```

Out[8]:

	day_sin	day_cos
0	9.749279e-01	-0.222521
1	-2.449294e-16	1.000000
2	-2.449294e-16	1.000000
3	9.749279e-01	-0.222521
4	-2.449294e-16	1.000000
5	9.749279e-01	-0.222521
6	-9.749279e-01	-0.222521
7	-4.338837e-01	-0.900969
8	4.338837e-01	-0.900969
9	4.338837e-01	-0.900969
10	4.338837e-01	-0.900969
11	-2.449294e-16	1.000000
12	4.338837e-01	-0.900969
13	4.338837e-01	-0.900969
14	-2.449294e-16	1.000000
15	7.818315e-01	0.623490
16	9.749279e-01	-0.222521
17	-4.338837e-01	-0.900969
18	4.338837e-01	-0.900969
19	7.818315e-01	0.623490

#Method 7 : Target encoding

Above methods have some drawbacks like , Label encoding doesn't work well with non-ordinal categorical features. One-hot encoding leads to an immense number of added features when your data contains a large number of categories. Entity embedding can only be used with neural network models.

To suppress some of the above disadvantages, and consider the relationship of target to features, we use target encoding also known as "mean encoding". In this method, we replace each category of the categorical variable with its corresponding probability of the target (if categorical) or average of the target (if numerical). The main drawbacks of this method are its dependency on the distribution of the target,its lower predictability power compared to the binary encoding method and also naively applying target encoding can allow data leakage, leading to overfitting and poor predictive performance..

for example,

Country	Target
India	1
China	0
India	0
China	1
India	1

Encoding for India = [Number of true targets under the label India/ Total Number of targets under the label India] which is 2/3 = 0.66

Country	Target
India	0.66
China	0.5

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)

df_train.head(20)
```

Out[6]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_9	ord_0	ord_1	ord_2	ord_3
0	0	0	0	0	T	Y	Green	Triangle	Snake	Finland	...	2f4cb3d51	2	Grandmaster	Cold	h
1	1	0	1	0	T	Y	Green	Trapezoid	Hamster	Russia	...	f83c56c21	1	Grandmaster	Hot	a
2	2	0	0	0	F	Y	Blue	Trapezoid	Lion	Russia	...	ae6800dd0	1	Expert	Lava Hot	h
3	3	0	1	0	F	Y	Red	Trapezoid	Snake	Canada	...	8270f0d71	1	Grandmaster	Boiling Hot	i
4	4	0	0	0	F	N	Red	Trapezoid	Lion	Canada	...	b164b72a7	1	Grandmaster	Freezing	a
5	5	0	1	1	T	N	Blue	Polygon	Lion	Costa Rica	...	51e27c16d	1	Novice	Freezing	j
6	6	0	1	1	T	N	Green	Trapezoid	Cat	China	...	7e3d79a0d	2	Grandmaster	Lava Hot	g
7	7	1	0	1	T	Y	Red	Triangle	Dog	Russia	...	feb72ecc2	1	Novice	Lava Hot	j
8	8	1	0	1	T	Y	Blue	Square	Hamster	Canada	...	34a7273bf	2	Novice	Boiling Hot	e
9	9	0	0	0	F	Y	Red	Trapezoid	Lion	China	...	0ece7a511	1	Expert	Freezing	h
10	10	0	1	0	T	Y	Green	Star	Snake	China	...	7f5e74721	1	Grandmaster	Boiling Hot	g
11	11	0	1	0	T	Y	Blue	Polygon	Snake	Russia	...	01dd99dc0	1	Novice	Freezing	i
12	12	0	1	0	F	N	Green	Polygon	Cat	China	...	e82ec9b46	3	Novice	Boiling Hot	g
13	13	1	1	1	F	Y	Blue	Trapezoid	Cat	India	...	cba64770e	1	Novice	Boiling Hot	i
14	14	0	0	1	T	Y	Blue	Triangle	Snake	India	...	3c915f463	1	Grandmaster	Boiling Hot	d
15	15	0	0	1	F	N	Green	Trapezoid	Lion	Russia	...	d445baa4d	1	Contributor	Hot	i
16	16	0	0	0	T	N	Red	Circle	Hamster	Russia	...	21ef6824e	1	Contributor	Boiling Hot	j
17	17	0	1	0	T	Y	Green	Star	Lion	Finland	...	332ab0429	1	Master	Cold	i
18	18	0	0	1	T	N	Blue	Square	Lion	China	...	2533756c6	1	Contributor	Freezing	g
19	19	0	0	1	F	N	Red	Polygon	Lion	Russia	...	a1f8ae96a	3	Grandmaster	Boiling Hot	b

20 rows × 25 columns

```
%%time

df_target = df_train.copy()
df_target['day'] = df_target['day'].astype('object')
df_target['month'] = df_target['month'].astype('object')
for col in df_target.columns:
    if (df_target[col].dtype=='object'):
        target= dict (
df_target.groupby(col)['target'].agg('sum')/df_target.groupby(col)['target'].agg('count'))
        df_target[col] = df_target[col].replace(target).values

df_target[['day','month']].head(10)
```

```
Wall time: 133 ms
```

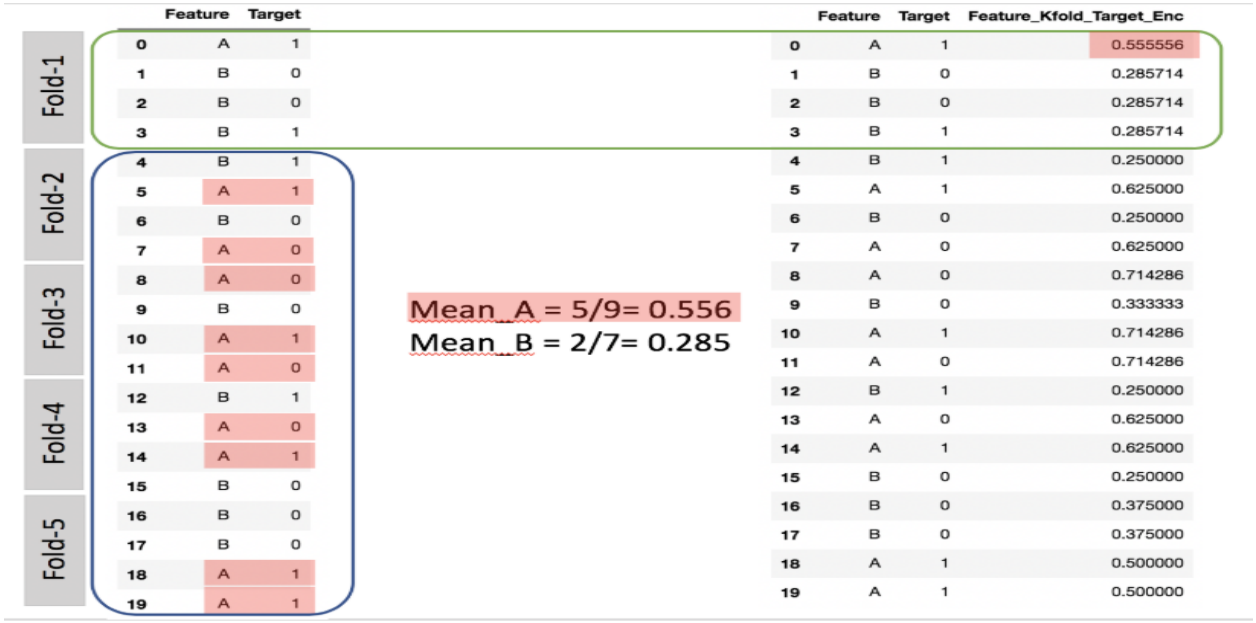
Out[12]:

	day	month
0	0.250000	0.125
1	0.000000	0.000
2	0.000000	0.125
3	0.250000	1.000
4	0.000000	0.000
5	0.250000	0.125
6	0.000000	0.000
7	0.500000	0.125
8	0.333333	0.000
9	0.333333	0.125

#Method 8 : K-fold/Cross-Fold target encoding

To clamp down the over-fitting problem from target encoding, we split the data into N-folds and compute the mean of each category to i-th fold using data in all other folds.

k-fold or cross fold target encoding method, we divide the dataset into the k-folds, here we consider 5 folds. Fig below shows the first round of the 5 fold cross-validation. We calculate mean-target for fold 2, 3, 4 and 5 and we use the calculated values, mean_A = 0.556 and mean_B = 0.285 to estimate mean encoding for the fold-1.



		Feature	Target
Fold-1	0	A	1
	1	B	0
	2	B	0
	3	B	1
Fold-2	4	B	1
	5	A	1
	6	B	0
Fold-3	7	A	0
	8	A	0
	9	B	0
	10	A	1
	11	A	0
Fold-4	12	B	1
	13	A	0
	14	A	1
	15	B	0
Fold-5	16	B	0
	17	B	0
	18	A	1
	19	A	1

Mean_A = 5/9= 0.556
Mean_B = 2/7= 0.285

	Feature	Target	Feature_Kfold_Target_Enc
0	A	1	0.555556
1	B	0	0.285714
2	B	0	0.285714
3	B	1	0.285714
4	B	1	0.250000
5	A	1	0.625000
6	B	0	0.250000
7	A	0	0.625000
8	A	0	0.714286
9	B	0	0.333333
10	A	1	0.714286
11	A	0	0.714286
12	B	1	0.250000
13	A	0	0.625000
14	A	1	0.625000
15	B	0	0.250000
16	B	0	0.375000
17	B	0	0.375000
18	A	1	0.500000
19	A	1	0.500000

Fig. Referred example from kaggle

```
import pandas as pd
import numpy as np

df_train = pd.read_csv('train.csv',nrows=20)

df_train.head(20)
```

Out[6]:

	id	bin_0	bin_1	bin_2	bin_3	bin_4	nom_0	nom_1	nom_2	nom_3	...	nom_9	ord_0	ord_1	ord_2	ord_3
0	0	0	0	0	T	Y	Green	Triangle	Snake	Finland	...	2f4cb3d51	2	Grandmaster	Cold	h
1	1	0	1	0	T	Y	Green	Trapezoid	Hamster	Russia	...	f83c56c21	1	Grandmaster	Hot	a
2	2	0	0	0	F	Y	Blue	Trapezoid	Lion	Russia	...	ae6800dd0	1	Expert	Lava Hot	h
3	3	0	1	0	F	Y	Red	Trapezoid	Snake	Canada	...	8270f0d71	1	Grandmaster	Boiling Hot	i
4	4	0	0	0	F	N	Red	Trapezoid	Lion	Canada	...	b164b72a7	1	Grandmaster	Freezing	a
5	5	0	1	1	T	N	Blue	Polygon	Lion	Costa Rica	...	51e27c16d	1	Novice	Freezing	j
6	6	0	1	1	T	N	Green	Trapezoid	Cat	China	...	7e3d79a0d	2	Grandmaster	Lava Hot	g
7	7	1	0	1	T	Y	Red	Triangle	Dog	Russia	...	feb72ecc2	1	Novice	Lava Hot	j
8	8	1	0	1	T	Y	Blue	Square	Hamster	Canada	...	34a7273bf	2	Novice	Boiling Hot	e
9	9	0	0	0	F	Y	Red	Trapezoid	Lion	China	...	0ece7a511	1	Expert	Freezing	h
10	10	0	1	0	T	Y	Green	Star	Snake	China	...	7f5e74721	1	Grandmaster	Boiling Hot	g
11	11	0	1	0	T	Y	Blue	Polygon	Snake	Russia	...	01dd99dc0	1	Novice	Freezing	i
12	12	0	1	0	F	N	Green	Polygon	Cat	China	...	e82ec9b46	3	Novice	Boiling Hot	g
13	13	1	1	1	F	Y	Blue	Trapezoid	Cat	India	...	cba64770e	1	Novice	Boiling Hot	i
14	14	0	0	1	T	Y	Blue	Triangle	Snake	India	...	3c915f463	1	Grandmaster	Boiling Hot	d
15	15	0	0	1	F	N	Green	Trapezoid	Lion	Russia	...	d445baa4d	1	Contributor	Hot	i
16	16	0	0	0	T	N	Red	Circle	Hamster	Russia	...	21ef6824e	1	Contributor	Boiling Hot	j
17	17	0	1	0	T	Y	Green	Star	Lion	Finland	...	332ab0429	1	Master	Cold	i
18	18	0	0	1	T	N	Blue	Square	Lion	China	...	2533756c6	1	Contributor	Freezing	g
19	19	0	0	1	F	N	Red	Polygon	Lion	Russia	...	a1f8ae96a	3	Grandmaster	Boiling Hot	b

20 rows × 25 columns

```
columns = df_train.drop(['target','id'],axis=1).columns
columns
```

Out[15]:

```
Index(['bin_0', 'bin_1', 'bin_2', 'bin_3', 'bin_4', 'nom_0', 'nom_1', 'nom_2',
       'nom_3', 'nom_4', 'nom_5', 'nom_6', 'nom_7', 'nom_8', 'nom_9', 'ord_0',
       'ord_1', 'ord_2', 'ord_3', 'ord_4', 'ord_5', 'day', 'month'],
      dtype='object')
```

```
from sklearn.model_selection import KFold
%%time
df_kfold = df_train.copy()
df_kfold[['ord_0','day','month']] = df_kfold[['ord_0','day','month']].astype('object')
df_kfold[['bin_3','bin_4']] = df_kfold[['bin_3','bin_4']].replace({'Y':1,'N':0,'T':1,"F":0})
kf = KFold(n_splits = 5, shuffle = False)
for train_ind,val_ind in kf.split(df_kfold):
    for col in columns:
        if(df_kfold[col].dtype=='object'):
            replaced=dict(df_kfold.iloc[train_ind][[col,'target']].groupby(col)['target'].mean())
            df_kfold.loc[val_ind,col] = df_kfold.iloc[val_ind][col].replace(replaced).values
```

```
df_kfold[['bin_3','bin_4','day','month','target']].head(10)
```

```
Wall time: 75.4 ms
```

Out[21]:

	bin_3	bin_4	day	month	target
0	1	1	0	0.166667	0
1	1	1	0	0	0
2	0	1	0	0.166667	0
3	0	1	0	1	1
4	0	0	0	8	0
5	1	0	0	0.25	0
6	1	0	5	0	0
7	1	1	1	0.25	0
8	1	1	0.666667	4	0
9	0	1	0.666667	0.5	0