Dt:- 21/12/22
5:50pm

* Data Preprocessing / Data Preparation

⇒ Feature Engineering

Data cleaning
- Missing Values
- Outliers

+

Data Wrangling
↳ (Data Transformation)
- Encoding
- Scaling
- Transformation.

Us.
(Car Insurance)

Data cleaning (In spyder)

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

Why two Dataset
Because to check old(
and new dataset.

⇒ Load Data set

```
df_raw = pd.read_csv("claimants.csv")
# df = pd.read_csv("claimants.csv")
```

Out:- df_raw | dataFrame | 1340,7 | column name : CASENUM,
df | dataframe | 1340,7 | column names : "

⇒ Check is there null values.

```
# df.isnull().sum()
```

Out :-

CASENUM	0
CLMSEX	12
CLMINSUR	41
SEATBELT	48
CLMAGE	189
LOSS	0
ATTORNEY	0

MISSING Values.

Data Understanding

Case Num :- Number of case (insurance)

Discrete clm sex :- 0, 1 [Male] [Female]

Discrete clm insur :- 1(Yes), 0 [NO] # having Insurance or not

Discrete Seat belt :- 0 [NO], 1 [Yes] # wearing seat belt

Continuoy clm age :- Age of the person

Loss :- Loss / Damage happened

attorney :- Whether they hire a lawyer (or) not, [0] No, [1] Yes.

Person who is Claming Isurance is called "claimants" During the accident

There are various methods for Filling null values

* For Continous Data
  - mean (no outliers)
  - median (outliers)
* For Discrete Data → Mode

Because Mean is Heavily Impacted with Outliers [Less affected]

Replacing null values For discrete Variables

So, we have to Replace Them with "MODE"

```
# df["CLMSEX"].value_counts()
```

(MODE)

Out :- 1.0 742 → Highest, so we Replace with "1"
0.0 586

```
# df["CLMSEX"].fillna(1, inplace=True)
```

↓ changing in orginal DaTa = "TRUE"

Out Filled with "1"

```
# df["CLMINSUR"].value_counts()
```

Out :- 1.0 1220 → MODE
0.0 120

```
# df["CLMINSUR"].fillna(1, inplace=True)
```

Out Filled with "1", so, replaced with "1" value

df["SEATBELT"].valuecounts()

Out :- 0.0 1270 → Mode.
1.0 22

df["SEATBELT"].fillna(0, inplace = True)

Out :- Filled with "0"

Time 11:30pm

Replacing null Values For Continous Variables By #Pandas

df["CLMAGE"].mean()

Out 28.414

df["CLMAGE"].median()

Out 30.00

df["CLMAGE"].fillna(28.414, inplace True)

Out Filled with 28.414

If we ask again what is median value.

df["CLMAGE"].median()

Out 28.414 , so it changes 30.00 To 28.414
Because we Filled with null values 28.414.

2nd method

Sklearn (Scikit Learn) (One stop shop For ML)

* SimpleImputer

Load data set again

Liberary → (module) (Function)

```
From Sklearn.impute import SimpleImputer
```

Ex: missing values = "???"

Calculating mean value

```
# mean_imputer = SimpleImputer (strategy = "mean") → E/n
```

(Function) → (Argument)

name as

Function. (or) Store in mean_imputer

Ex:- Core Python [Keyword argument]

If we don't write this line it gives

Same : a = 4

Converting array To data Frame

To Remove array. From mean value.

```
# df ["CLMAGE"] = Pd.DataFrame (mean_imputer.fit_trans
```

storing

```
_Trans Form (df [["CLMAGE"]]))
```

```
array ([50.],
       [18.],
       [5.],
       ⋮
```

Fill in This column (or) Replacing

Calculate The mean value

Out

Filled with 28.414

For Median only change mean To median

```
# median_imputer = SimpleImputer (strategy = "Median")
```

```
# df["CLMAGE"] = pd.DataFrame(median_imputer.fit_
                Transform(df[["CLMAGE"]])
```

For Mode [Discrete Data]

```
mode_imputer = SimpleImputer(strategy="mode")

#df["CLMSEX"] = pd.DataFrame(mode_imputer.fit_transform(df[[
                                    "CLMSEX"]])

# df["CLMINSUR"] = pd.DataFrame(mode_imputer.fit_transfo
                        (df[["CLMINSUR"]])

#df("SEATBELT"] = pd.Data Frame( mode_imputer.fit_trans
                        m(df[["SEATBELT"]])
```

Output — Fills Directly The mode Value in to the dataset.

MODE = it is for Discrete DaTa.

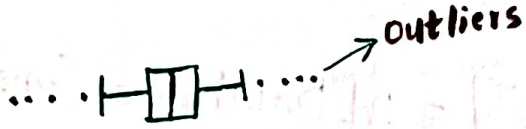
MEAN
MEDIAN
} → Continous DaTa

3/4/22 2:08 AM

Outliers → # Applicable Only For Continous Data

1Que :- What is outlier ?

Ans :- An Outlier is a data point in a data set. that is distant From all other observation which is significantly different From The Remaing Data



(or)

- The data point that lies outside the over all Distribution of the dataset.

2 Que :- What are Impacts having Outliers in a DataSet ?

Ans :- it will cause Various Problems in Statistical Analysis. (it may cause a significant impact on mean and Standard deviation)

Ex :- $\bar{x}$ = mean (affect)

$\frac{\Sigma[x-\bar{x}]}{n-1}$ = varience → $\sqrt{\frac{\Sigma(x-\bar{x})^2}{n-1}}$ (Standard deviation)

all are Effected by Outliers

* In addition, Some Machine learning models are Sensitive to outliers, which may Decrease There Performance.

3 Que : Reasons For Outliers ?

Ans :
1. Data Entry Errors (Ex:- Salary = 1,00,000, But 10,000 Entering)
2. Measurement Errors (Ex :- Measuring in meters instead of KM)
3. Instrumental Error

4 Que :- Types of Outliers ?

Ans :- Univariate Outlier ----> Identifying Outlier For Single Variable

* Bivariate Outlier ----> Identifying as Outlier by Analyzing 2 Variables at a Time

Importing Libraries

```
# Import numpy as np
# Import pandas as pd
# Import matplotlib as plt.
# Import Seaborn as Sns.
```

Real Time Case study

boston - House Price Dataset

```
# boston = pd.read_csv("D:\\datascience\\shivaramakrishna class\\6. Data cleaning\\boston.csv")
```

```
boston.head()
```

out

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRTIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
# boston.info()
```

506 rows, 14 columns

out

0	CRIM	Float 64
1	ZN	Float 64
2	INDUS	Float 64
3	CHAS	→ int 64
4	NOX	Float 64
5	RM	Float 64
6	AGE	Float 64
7	DIS	
8	RAD	→ Int 64
9	TAX	→ Int 64
10	PTRTIO	Float 64
11	B	Float 64
12	LSTAT	Float 64
13	PRICE	Float 64

(Float 64 (11), int 64 (3))

Various ways of finding The Outlier

Changing Every Value to Zscore.

Ex:-

X	Zscore
6	$\frac{6-5.6}{1}$
4	$\frac{4-5.6}{1}$
9	$\frac{9-5.6}{1}$
7	$\frac{7-5.6}{1}$
2	$\frac{2-5.6}{1}$

$Z = \frac{X-\mu}{\sigma}$

$\therefore \mu = \frac{6+4+9+7+2}{5}$

$\mu = 28/5 = 5.6$

Ex:- $\sigma = 1$

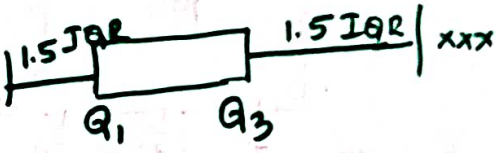
Option (1) **Z score**

$|Z\text{score}| > 2$

$[\because Z < -2, \; Z > +2]$ |modulus|

Option (2) **IQR** (Inter Quartile Range)

- any Value $< Q_1 - 1.5\,IQR$
- any value $> Q_3 + 1.5\,IQR$



any value Less than $Q_1 - 1.5\,IQR$ is Outlier.

any value More Than $Q_3 - 1.5\,IQR$ is Outlier

Option (3) **visualization**

* Box plot, Histogram For Univarte Outliers
* Scatter plot For bivariate Outliers.

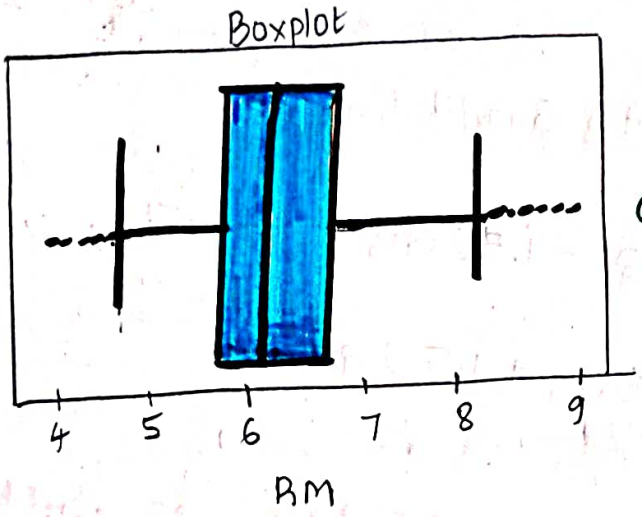
Option ③

Detection of Outliers of "RM" column (Based on Boxplot)

```
# sns.boxplot(boston.RM)
  plt.title("Boxplot")
  plt.show()
```

* Over all view of Outlier

Out :



Outliers in Both Tails of "RM".

Option ①

Detection of Outliers of "RM" column (Based on Zscore)

* For Exact Outliers in Data

$$Z\text{score} = \frac{X - \mu}{\sigma}$$

```
# boston["RM"].mean(), boston["RM"].std()
```

Out (6.284, 0.702)

```
# boston["RM_Zscore"] = (boston["RM"] - boston["RM"].mean()) / boston["RM"].std()
```

creating new column and storing Zscores

X → boston["RM"]; μ → boston["RM"].mean(); σ → boston["RM"].std()

Output :-

Clarity :-

$$\frac{\text{boston["RM"]} - \text{boston["RM"].mean()}}{\text{boston["RM"].std()}}$$

(x − μ) / σ

E Answers Ammi

boston["RM_Zscore"] = ↑ Endulo store ayethaye.

Zscore

original

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	...	RM SCORE
0	0.0632	18.0	2.31	.	.	6.575	.	.	.	.	.	.		0.413263
1	0.0273	0.0	7.07	.	.	6.421	.	.	.	.	.	.		0.194082
2	0.0272	0.0	2.18	.	.	7.185	.	.	.	.	.	.		1.281446
...														
505	0.0474	0.0	11.93	.	.	6.030	.	.	.	.	.	.		-0.362408

506 x 15 columns

Now, Identify Outliers in this column ↑

Outlier = boston[(boston["RM_Zscore"] > +2) OR (boston["RM_Zscore"] < -2)]

2 conditions

Greater Than +2

Less Than -2

Out :- Outlier

	CRIM	...	RM SCORE
97			2.539
98			2.185
162			3.47
163			-2.127
66			-3.055

3/4/22 4:30 Am

Time
1:00pm

Outlier. Shape

Out :: (31, 15)

option (2)
Detection of Outlier of RM (based on IQR)

- Calculate First (q_1) and third quartile (q_3)
- Find interquartile range ($q_3 - q_1$)
- Find Lower bound $q_1 1.5$ & Find upper bond $q_3 1.5$

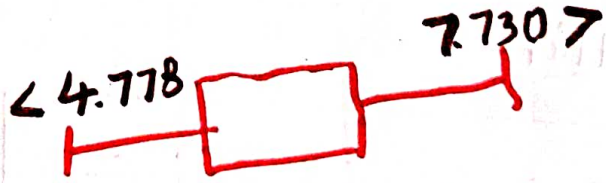
for Exact no. of range + - of outliers

```
# Q3 = boston["RM"].quantile(0.75)
  Q1 = boston["RM"].quantile(0.25)

  IQR = Q3 - Q1
  Lower_limit = Q1 - (IQR * 1.5)
  Upper_limit = Q3 + (IQR * 1.5)

  Lower_limit, upper_Limit
```

$Q_1 = 0.25$ (ol)
$Q_3 = 0.75$ (ol)

$IQR = Q_3 - Q_1$

$\Rightarrow Q_3 + IQR * 1.5$
$\Rightarrow Q_1 - IQR * 1.5$



Out :- 4.778 , 7.730
Min Max

* Which is having less than 4.778 is outlier
* which is having Greater than 7.730 is Outlier

Methods of deal The Outliers

3R Techinque
* Remove → Deleting The outliers
* Replace → changing values / Data Manipulation
* Retain → Treat Them Seperately.
Work without outlier / with outlier "Work" seperately.

Outliers should be detected and "REMOVED" Only From Training data set, NOT from The Test set. So we Should first divide our data set into train and tests. and remove Outliers in the train set, but keep those in test set, and Measure how well our model is doing.

option (1) Removing * (Let's trimm The dataset) → 4.77

```
# boston_trimmed = boston[(boston["RM"] > Lower_limit) &
                          (boston["RM"] < upper_limit)]
                                          → 7.73
```

```
boston_trimmed
```

Out:

Which ever Data is Greater than 4.77 and Less Than 7.73 Consider That data Only. Remove Remaining data.

		RM		RM_Zsco
0		6.575		0.413263
1		6.421		0.194082
2		7.185		1.28 14416
...		...	...	...
504		6.794		0.724955
505		6.030		-0.362408

476 x 15 columns

∴ From 506 records, Removed 30 records

That Means 476 Having (or) there.

⇒ it's about 6% of the data is Removed

⇒ it is only in One Variabe (1 column) = 6%.

⇒ If Total Variable (1,2,3,4..... all columns (14) → [10] continous Data) → Remove → May Delete 100 rows ↓ 20% of Data

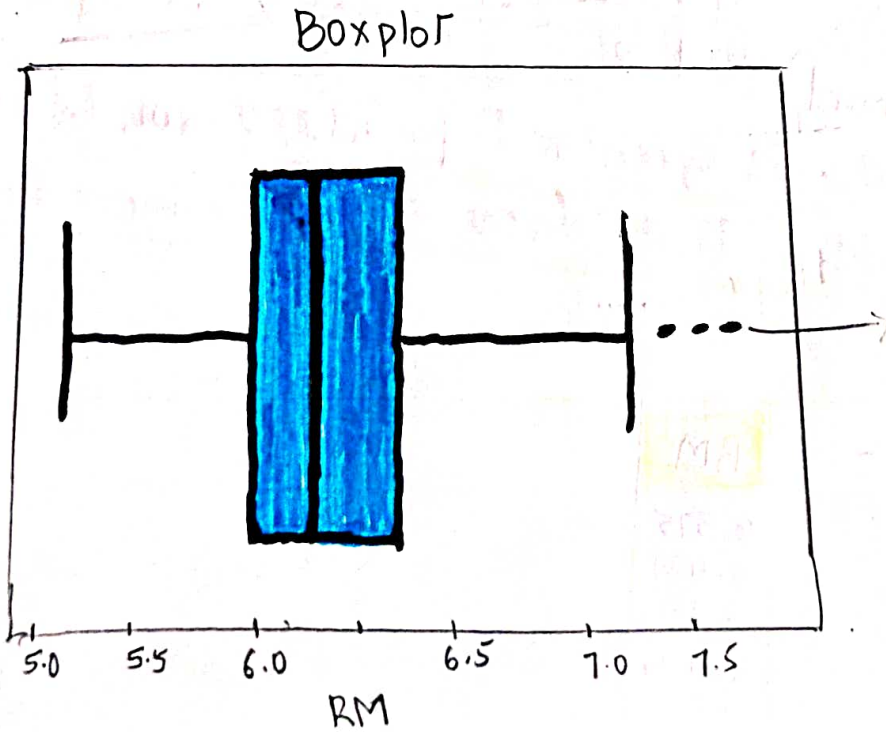
So, In real Life, we don't use outliers "Remove"

Even, Though, If we want Remove, still it contains Outliers.

⇒ Outliers in The Trimmed dataset

```
# sns.boxplot (boston_trimmed . RM)
plt. title ("Boxplot")
plt. show ().
```

Out



still, we have Outliers, after Removing The Outlier records.

Q:- Why ? it showing Outliers after Removing records of outliers

A:- When we remove datapoints From Our dataset, all the Parameters of The distribution are re-calculated.

For 476 records,

Again, it's calculating (Q_1, Q_3, IQR)

for 506

$Q_1 = 4.77$, Q_1 = New Value (again)

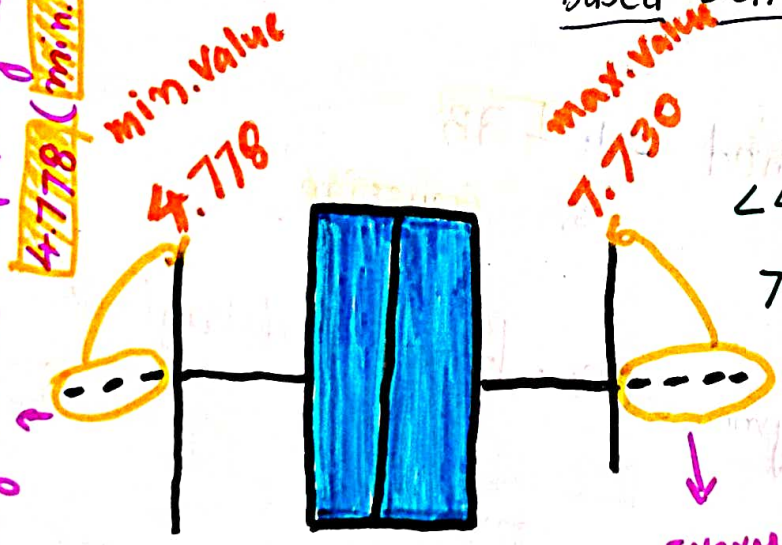
$Q_3 = 7.730$, Q_3 = New value (again)

* Therefore, In The new-trimmed variables values that before were not considered Outliers.

That's why, it showing new Outliers after removing some records also.

option (2). **Replace**

Replace with Upper Limit & Lower Limit (Calculated Based IQR)



$< 4.778 = Q_1 - [IQR * 1.5]$

$> 7.730 = Q_3 + [IQR * 1.5]$

Every Outiler Replacing with 7.730 (max.value)

⇒ More DeTailness

Replacing with Q_1, Q_3 Values

	CRIM	.....	RM	..... RAD
97		More Than 7.730, so, They are outliers. Reduce with Q_3 = 7.730	8.069 7.730	
98			7.820 7.730	
162			7.802 7.730	
163			8.375 7.730	

Q_3 - IQR * 1.5 < [4.778] (Lower_value) → less Than 4.778 = outlier

Q_1 IQR * 1.5 > [7.730] (Upper_value) → greater Than 7.730 = outlier

* Winsorizer

PIP install feature_engine

Winsorzier

From feature_engine.Outliers (Module) Import Winsozier Function

(Library)

In Winsozier Function (capping method argument)

* capping method = 1. IQR
2. "Gaussian"

In feature_engine.Outliers Import ArbitaryOutlierCapper Function

(Library) (module)

We can give our own Lower, upper Values.

1. By IQR method

In Depth of IQR.

Firstly,

 # We Calculate Q_1 Value

```
# Q1 = boston["RM"].quantile(0.25)
```

Out :- 5.855

 # We calculate Q_3 value.

```
# Q3 = boston["RM"].quantile(0.75)
```

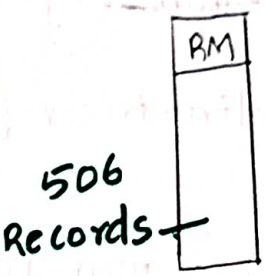
Out :- 6.623

 # IQR Value

```
# IQR = Q3 - Q1
```

IQR

Out 0.737



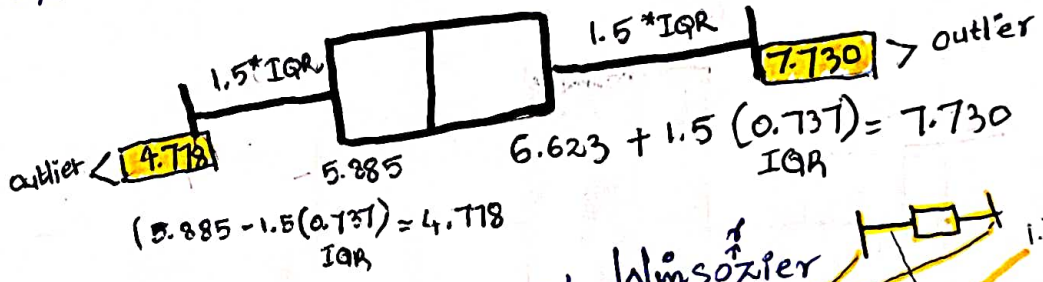
$\therefore Q_1 = 5.885$

$Q_3 = 6.623$

$IQR = Q_3 - Q_1$

$= 0.737$

Distance between Q_3 and Q_1



CODE :-

```
# From feature_engine.Outliers Import Winsozier
# Win = Winsozier(capping_method="iqr", tail='both', fold=1.5, Variables=["RM"])
# boston_t = Win.fit_transform(boston[["RM"]])
# Print(win.left_tail_caps_, Win.right_tail_caps_)
# Sns.boxplot(boston_t.RM)
# Plt.title("boxplot")
# Plt.show()
```

Win → Random name

store in → boston_t

boston[["RM"]] → original data

4.778 ← win.left_tail_caps_

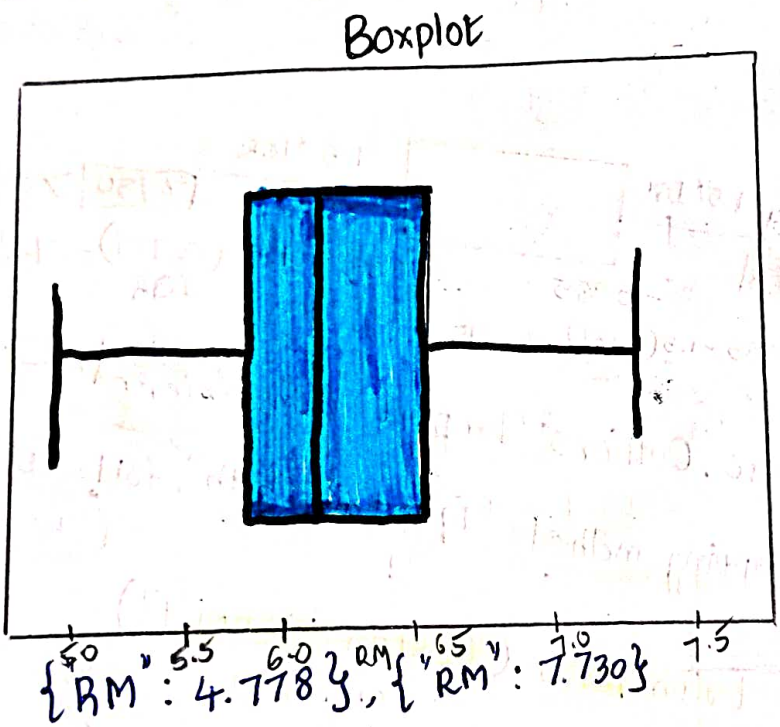
7.730 ← Win.right_tail_caps_

Edited (Replaced IQR).RM

again

```
From feature_engine.Outliers import Winsorizer

win = Winsorizer (capping_method = "iqr", fold = 1.5, Tail = "both",
                  Variables = ["RM"])
boston_t = win.fit_transform (boston [["RM"]]

Print (win.left_tail_caps_ , win.right_tail_caps_)

Sns.boxplot (boston_t.RM)
plt.title ("boxplot")
plt.show()
```

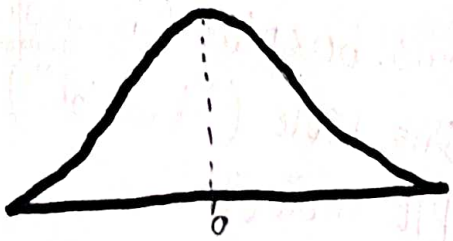
Out



2) Replacing using Winsorizer (min, max, automatically Taken by "Gaussian method")

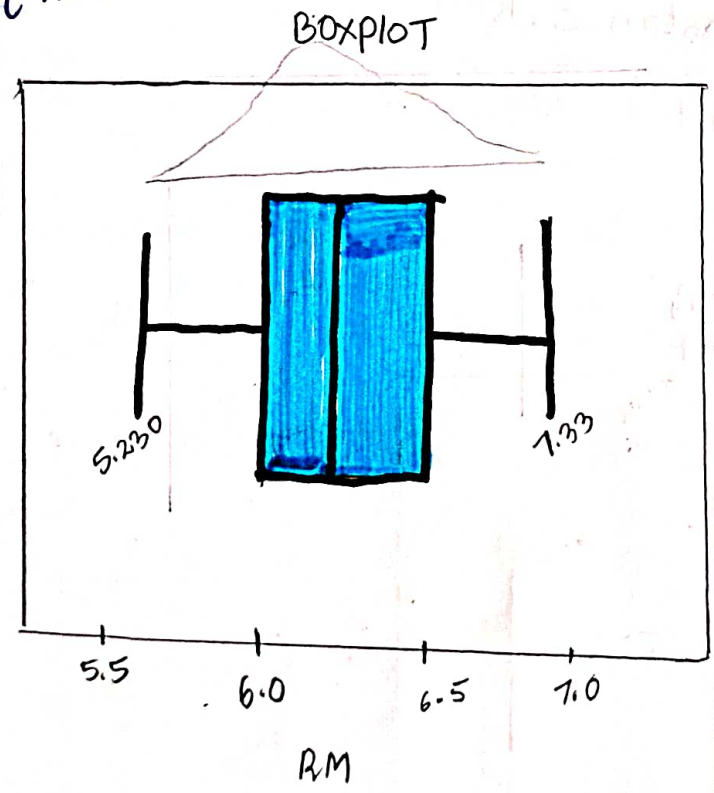
Gaussian method = "Normal Distribution method"

same



```
# from feature_engine.Outliers import Winsorizer

# win = Winsozier (capping method = "gaussian", tail = "both",
                   fold = 1.5, variables = ['RM'])

boston_t = win.fit_transform (boston[["RM"]])

print (win.left_tail_caps_, win.right_tail_caps)

sns.boxplot (boston_t, RM)
plt.title ("Boxplot")
plt.show()
```

Out {'RM': 5.230} {"RM": 7.33}



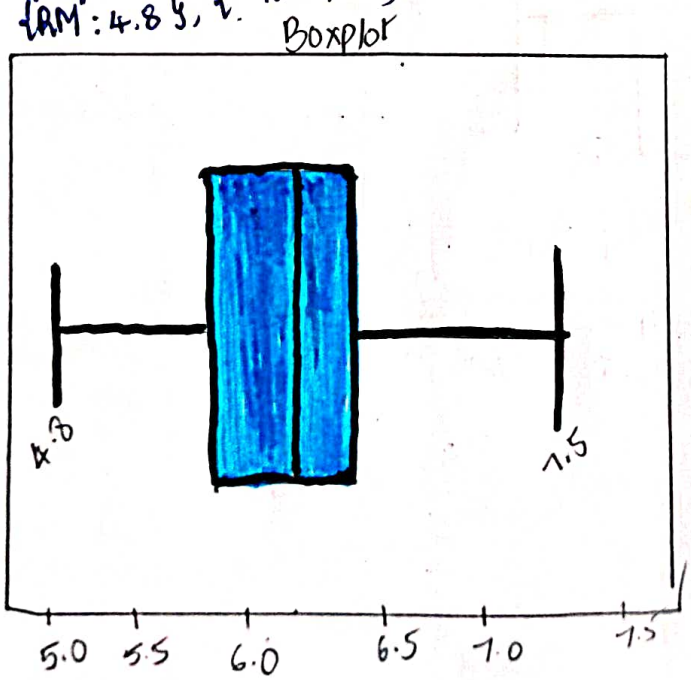
3. Replace arbitary Outlier Capper (The min, max, by user)

we get the min and max values based on "Domain Expert"
(or) by our Own "Research"

```
# From feature_engine.Outliers Import Arbitrary Outlier Capper

Capper = ArbitraryOutlierCapper (max_capping_dict = {"RM": 7.5},
                                 min_capping_dict = {"RM": 4.8})

boston_c = Capper.fit_transform (boston[["RM"]])

Print (capper.right_tail_caps_, Capper.left_tail_caps_)

Sns.boxplot (boston_c.RM)
plt.title ("Boxplot")
plt.show()
```

Out

{'RM': 4.8}, {'RM': 7.5}



3/4/22 5:20 pm

5/4/22
10:30pm

3 * Discretization *

↓

Converting Continous Data To Discrete Data

↓

also called as "Binning"

Code :-

```
Import pandas as pd

# stroke = pd.read_csv("stroke prediction.csv")

# stroke.head()
```

Out

id	gend	age	hyper Tension	heart_ disease	Ever_married	Work type	Residence Type	avg_ glucose Level	bmi	Smok status	Stroke
30669	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
30468	Male	58.0	1	0	Yes	Private	urban	87.96	39.2	Never	0
16523	Female	8.0	0	0	No	Private	urban	110.89	17.6	NAN	0
56543	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	Formerly smoked	0
46136	male	14.0	0	0	No	Never_ Worked	Rural	161.28	19.1	NAN	0

discrete (cat) | continous | discrete (const) | discrete (count) | discrete | discrete | discrete | continous | Contin | discrete | discrete

```
# stroke.shape
```

Out : (43400, 12)

```
# stroke.info()
```

Out : information about stroke data.

⇒ Create Bins

intervals = [0, 12, 19, 30, 60, 90] → #bins ⇒ Each interval

categories = ["child", "teenager", "young adult", "middle aged", "Senior_citizen"]

#The Age column is Continous Data. So, We converting into Discrete Variable

intervals → categorised
* 0-12 → child
* 12-19 → teenager
* 19-30 → young adult
* 30-60 → Middle aged
* 60-90 → Senior citizen

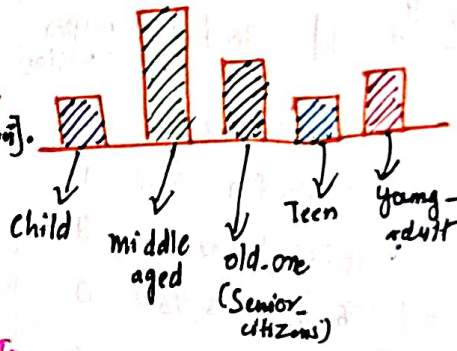
Creating New column & Storing the data of "Age"

Storke ["Age_category"] = Pd.cut [X =] (Divide.)

X = Stroke ['Age'], bins = intervals,

Labels = categories]

Storke ["Age category"].hist()



stroke.head()

Out

Changed Continous Data To Discrete Categorical data

Newly-added

Id	gender	age	hyper Tension	Heart-disease	Ever married	work type	Residence Type	avg. glucose Level	bmi	Smoke	Stroke	Age_Category
30669	Male	3.0	-	-	-	-	-	-	-	-	-	child
30468	Male	58.0	-	-	-	-	-	-	-	-	-	middle-aged
16523	Female	8.0	-	-	-	-	-	-	-	-	-	child
56543	Female	70.0	-	-	-	-	-	-	-	-	-	Senior_citizen
46143	Male	14.0	-	-	-	-	-	-	-	-	-	teen

4. * Encoding *

Applicable For Discrete Categorical Variables

* Machine Can't Understand Text Data. So, We Convert the "discrete categorical" to "discrete count" Data

* There are two types of Categorical Data
  1. Nominal (No Natural Sequence)
  2. Ordinal (Natural sequence)

Ex: Grades

O	10
A+	9
A	8
B+	7
B	6
F	0

⇒ O > A+ > ... > 10 > 9

Ordinal Data

citynames	
Hyd	0
Delhi	1
Mumbai	2
Kerala	3
Gujarat	4

In this case, we can't Decide by numbering which is greater given To state.

we should Treat Every variable Equally.

⇒ To convert Categorical Data To Numeric:

Nominal Data → (dummies)
1. get Dummies (Pandas)
2. One hot Encoding (sklearn)

Odinal Data
1. map (pandas)
2. Label Encoder (sklearn)

# One Hot Encoding	# Label Encoding
* Nominal	* Odinal

```
import pandas as pd
import Numpy as np
import matplotlib.pyplot as plt
% matplotlib inline
```

```
# df = pd.read_csv("homeprices.csv")
# df.head()
```

Out :-

spt

	Town	area	Price
0	chennai	2600	5500000
1	chennai	3000	5650000
2	chennai	3200	6100000
3	chennai	3600	6800000
4	Banglore	2600	5850000

Town

1. Chennai
2. Banglore
3. hyderabad

Discrete → Categorical Data

Nominal Data

⇒ **Pd.get_dummies** [Nominal Variable Encoding using pandas]

```
# dummies = pd.get_dummies(df["Town"])
# dummies
```

Out :-

	Banglore	Chennai	Hyderabad
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	1	0	0
5	1	0	0
6	1	0	0
7	1	0	0
8	0	0	1
9	0	0	1
10	0	0	1

Out :-

	Banglore	Chennai	Hyderabad
0	0	1	0
1	0	1	0
2	0	1	0
3	0	1	0
4	1	0	0
5	1	0	0
6	1	0	0
7	1	0	0
8	0	0	1
9	0	0	1
10	0	0	1
11	0	0	1

→ Dummies

which Ever Record is there it shows = 1

* For other columns it shows = 0

Adding → original → converted one [by]

```
# df_dummies = pd.Concat([df, dummies], axis = "columns")
```

```
df_dumies
```

Out :-

	Town	Area	Price	Banglore	Chennai	Hyderabad
0	Chennai	2600	5500000	0	1	0
1	Chennai	3000	5650000	0	1	0
2	Chennai	3200	6100000	0	1	0
3	Chennai	3600	6800000	0	1	0
4	Banglore	3600	5850000	1	0	0
5	Banglore	2600	6150000	1	0	0
6	Banglore	2800	6500000	1	0	0
7	Banglore	3300	6100000	1	0	0
8	Hyderabad	3600	7100000	0	0	1
9	Hyderabad	2600	5750000	0	0	1
10	Hyderabad	2900	6000000	0	0	1
11	Hyderabad	3100	6200000	0	0	1

Delete The Original column

```
# df_dummies.drop("town", axis = "columns", inplace = True)

df_dummies
```

Out:

	Area	Price	Banglore	chennai	Hyderabad
0	2600	5500000	0	1	0
1	3000	5650000	0	1	0
2	3200	6100000	0	1	0
3	3800	6800000	0	1	0
4	2600	5850000	1	0	0
5	2800	6150000	1	0	0
6	3300	7100000	1	0	0
7	3600	5750000	1	0	0
8	2600	6000000	0	0	1
9	2900	6200000	0	0	1
10	3100	6900000	0	0	1
11	3600	6500000	0	0	1

after deleting original column in New Data (ie. df_dummies) Old_data (df) is still remains same.

Dummy Variable Trap

⇒

Bang	chenn	Hyder
0	1	0
0	1	0
1	0	0
1	0	0
0	0	1
0	0	1

→

Bang	chenn
0	1
0	1
1	0
1	0
0	0
0	0

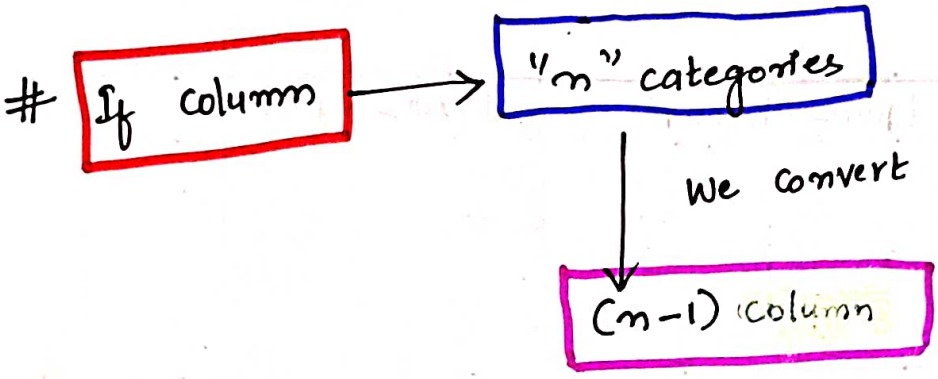
→ hyderabad

If we Remove One of The column also, still we can Identify, col name by (0,0)

Town — 3 categories

↓ Dummies

3 columns [Multi-collinearity / collerations Problem]




```
# df_dummies.drop("chennai", axis="columns", inplace=True)

# df_dummies
```

out

	Area	Price	Banglore	Hyderabad
0	2600	5500000	0	0
1	—	—	0	0
2	—	—	0	0
3	—	—	0	0
4	—	—	0	0
5	—	—	1	0
6	—	—	1	0
7	—	—	1	0
8	—	—	1	0
9	—	—	0	1
10	—	—	0	1
11	36 000	6500000	0	1

(Same) as old data

where ever Both column Having Equal Zero's, it i... another column's data.

Chennai = 0 0

We Can Write One line of Code To do all these

```
# df_dum = pd.get_dummies (df, drop_first = True)

# df_dum
```

Out:

	Area	Price	town_chennai	town_hyderabad
0			1	0
1				
2			1	0
...				
10			1	0
11				

This Diagram is Exact of (previous) Last page diagram. But deleted banglore Replaced chennai"

town_banglore [deleted] → Because, B ✓ (Alphabetical order)
chennai
hyd

Replaced with (0 0)

⇒ One Hot Encoding → Nominal using sklearn

```
From sklearn.Preprocessing import OneHotEncoder

# enc = OneHotEncoder (drop = "First")
```

↑ stores in enc

```
# enc_df = pd.DataFrame (enc.fit_transform
                          (df [["town"]]).
                          toarray ())
```

↑ converting

↓ output

enc_df

Out

	0	1
0	1.0	0.0
1	1.0	0.0
2	1.0	0.0
3	1.0	0.0
4	0.0	0.0
5	0.0	0.0
6	0.0	0.0
7	0.0	0.0
8	0.0	1.0
9	0.0	1.0
10	0.0	1.0
11	0.0	1.0

merge with main df

df_ohe = df.join(enc_df)

df_ohe.drop("town", axis="columns", inplace=True)

df_ohe

Out

	area	Price	0	1
0	2600	5500000	1.0	0.0
1	3000	5650000	1.0	0.0
2	-	-	1.0	0.0
3	-	-	0.0	0.0
4	-	-	0.0	0.0
5	-	-	0.0	0.0
6	-	-	0.0	0.0
7	-	-	0.0	1.0
8	-	-	0.0	1.0
9	-	-	0.0	1.0
10	-	-	0.0	1.0
11	-	-		

Alphabetical order

⇒ **Label Encoder** → **Ordinal Variable** [sklearn]

Used For binary category Variable

dfnew = df.copy() → creating "copy" From original Data.

dfnew

Out

	town	Area	price
0	Chennai	2600	5500000
1	chennai	3000	5650000
2	Chemai	3200	61000000
.	.	.	.
.	.	.	.
.	.	.	.
11	Hyderabad	3600	6950000

From sklearn.preprocessing Import LabelEncoder

Le = LabelEncoder()

dfnew.town = Le.fit_transform(dfnew.town)

dfnew

Out:-

	town	Area	price
0	1		
1	1		
2	1		
3	1		
4	0		
5	0		
6	0		
7	0		
8	2		
9	2		
10	2		
11	2		

Alphabetical Order

Banglore [B] — 0
Chennai [C] — 1
Hyderabad [H] — 2

↓ User Desired values we use this.

⇒ map() → Ordinal Variable [pandas]

map function.

```
# df_m = df.copy()
# df_m ["town"] = df_m ["town"]. map ({ "Chennai :0,
                                        "Bangalore ":2 , "hyderabad": 1})
# df_m
```

Taken by our choices

Here

Chennai = 0, 20

Banglore = 2, 15

hyderabad = 1, 25

We can Replace with any Value

Out :-

	town	Area	Price
0	0		
1	0		
2	0		
3	0		
4	2		
5	2		
6	2		
7	2		
8	1		
9	1		
10	1		
11	1		

Which Ever Order we write it Takes in The Order ↑

⇒ Ordinal Encoder → Ordinal Variable (Ascending Order)

From Sklearn.Preprocessing import OrdinalEncoder

```
# df_new1 = df.copy()
# oe = OrdinalEncoder (categories = [["Banglore", "Hyderabad", "Chennai"]])
#                                     0           1            2
# df_new1.town = oe.fit_transform (df_new1 [["town"]])
# df_new1
```

Out :-

	Town
0	2.0
1	2.0
2	2.0
3	2.0
4	0.0
5	0.0
6	0.0
7	0.0
8	1.0
9	1.0
10	1.0
11	1.0

6/4/22 3:00AM

Dt:-3/4/22
11:30pm

"1" ⇒ Feature Transformation → Only For Continous Data

Feature processing : Transformation.

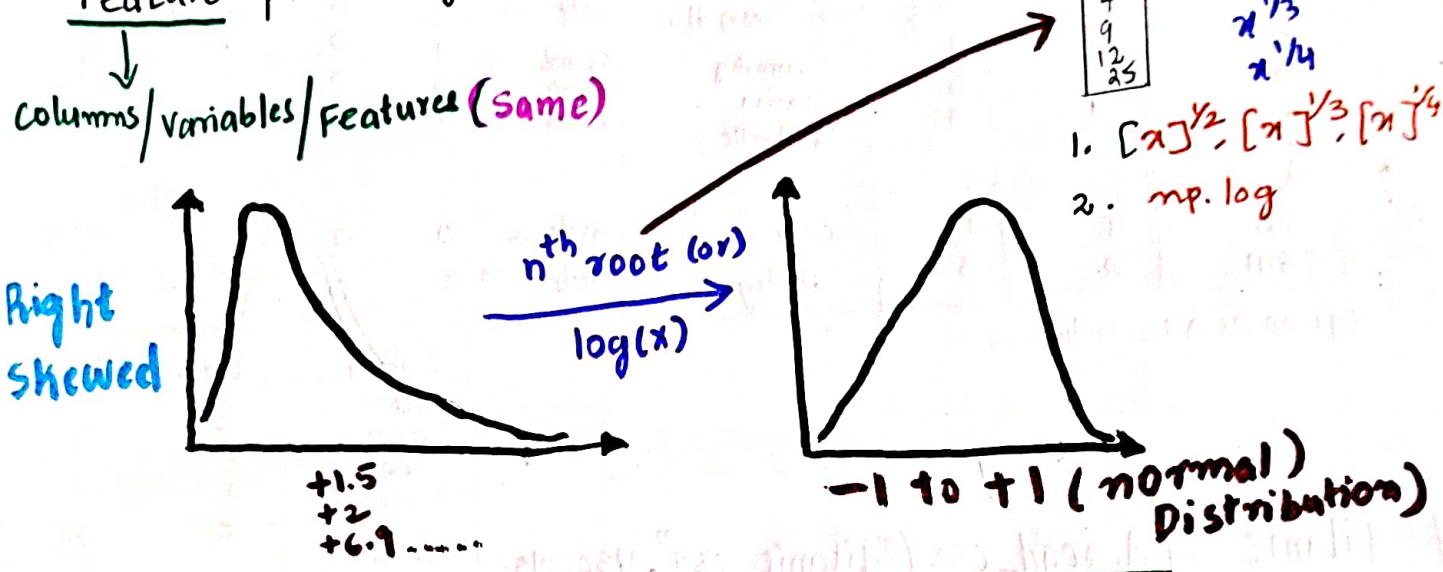
Feature ↓
Columns / Variables / Features (same)

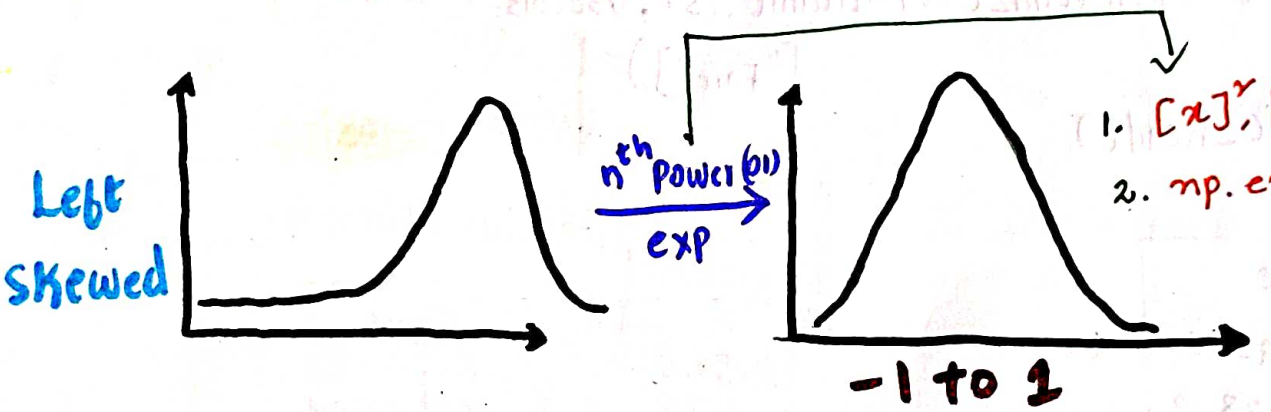
Ex:

x
2
4
9
12
25

$x^{-0.5}$ ✓
$x^{1/2}$
$x^{1/3}$
$x^{1/4}$

1. $[x]^{1/2}$, $[x]^{1/3}$, $[x]^{1/4}$
2. np.log





1. $[x]^{r}$, $[x]^{3}$, $[x]^{4}$.
2. np.exp

import Liberaris / packages.

```
# import numpy as np
# Import pandas as pd
# Import matplotlib.pyplot as plt
# % matplotlib inline
# Import Seaborn as sns
```

1. Root Transformations
2. log Transformations
3. Reciprocal Transformat
4. Boxcox Transformatio

titanic = pd.read_csv("titanic_csv")

location of csv

titanic

	Passenger Id	Survied	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Owen Harris	male	22.0	1	0	A/52771	7.2500	NAN	S
1	2	1	1	cumng	Female	38.0	1	0	.	71.2833	C85	C
2	3	1	3	Laina	Female	26.0	0	0	.	7.9250	NAN	S
3	4	1	1	Futrelle	Female	350	1	0		53.1000	C123	S
	⋮										⋮	
889	890	1	1	Behr	male	26.0	0	0	.	30,0000	C148	C
890	891	0	3	Dooley	male	32.0	0	0		7.7500		

891 rows x 12 columns.

titanic = pd.read_csv("titanic.csv", usecols=["Fare"])

Particular column

usecols = ["Fare"]

col. Name.

titanic.head()

Out

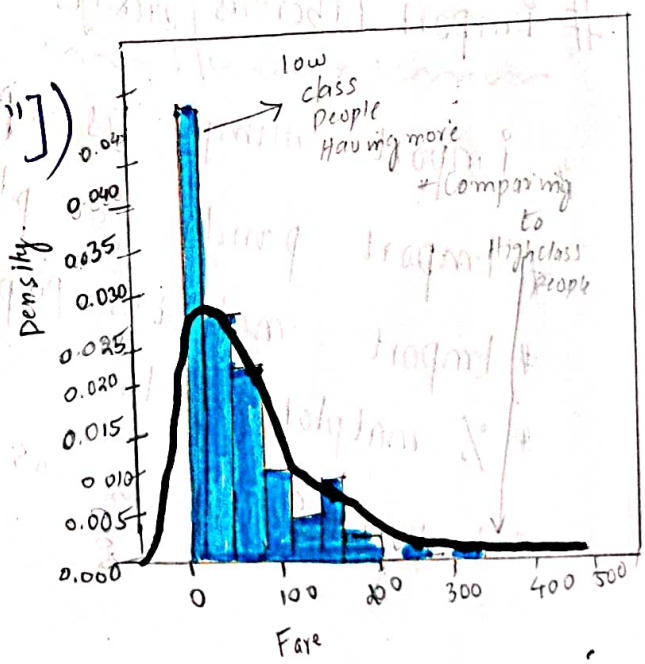
	Fare
0	7.2500
1	71.2833
2	7.9250
3	53.1000
4	8.0500

sns.distplot(titanic["Fare"])

plt.show()

titanic["Fare"].skew()

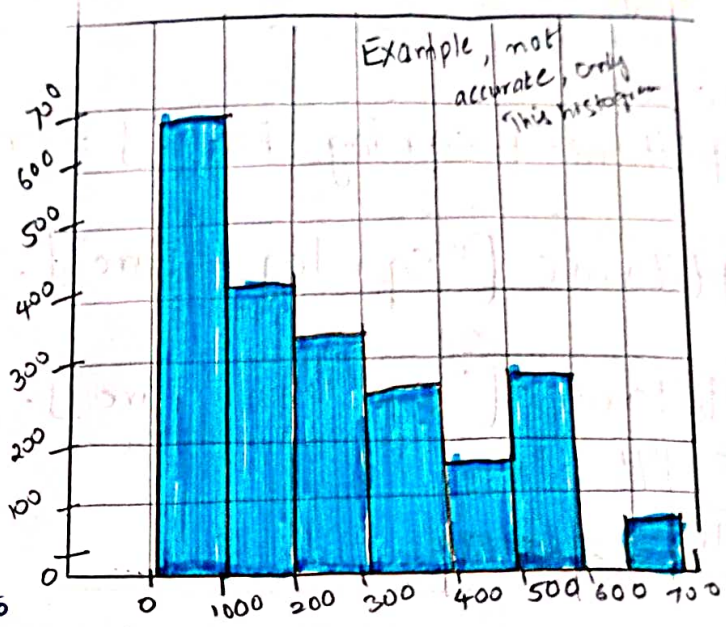
Out: + 4.7873



another way

```
t - ["Fare"].hist()
plt.show()
titanic["Fare"].skew()
```

Out :- +4.7873



Values

Skew = +2.085 [2nd root]
Skew = +0.5 (4th root)
Skew = +0.21 (5th root) ← Consider, This which is close To "0"
Skew = -0.95 (6th root)

* Root Transformation # For Right skewed

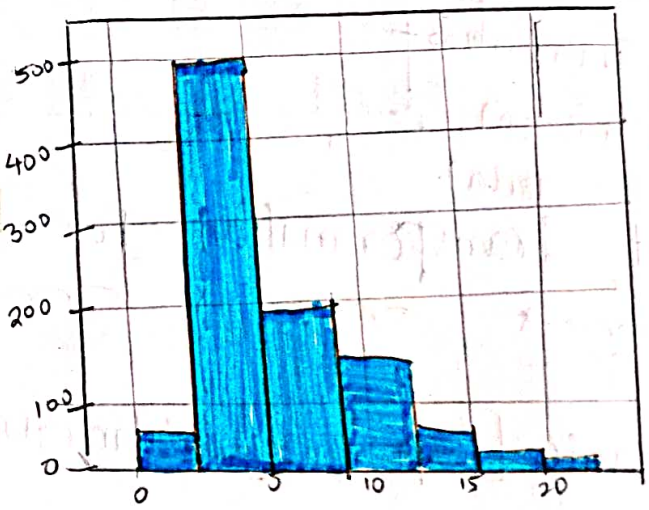
```
# titanic["sqr_Fare"] = titanic["Fare"] ** (1/2)
# titanic["sqr-Fare"].skew()
# titanic["sqr-Fare"].hist()
```

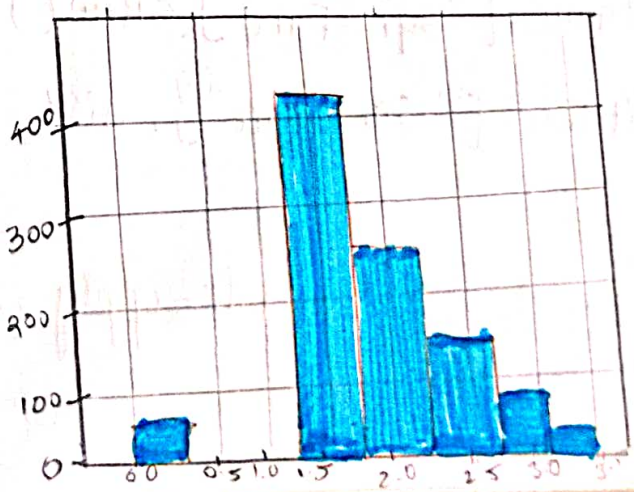
→ For Right skewed we use n^{th} root $\sqrt{}, \sqrt[3]{}, \sqrt[4]{}, \sqrt[5]{}$

Out :- +2.085 More Than +1 or -1

* Try with [1/5], [1/6]...

```
# titanic["sqr_Fare"] = titanic["Fare"] ** (1/5)
# titanic["sqr_Fare"].hist()
# titanic["sqr_Fare"].skew()
```





Out :- -0.212676 Final

less Than -1 to +1

→ log Transformation

Q. Is, This mandatory, To write Every time?
- No, Only if we "0" in The Data.

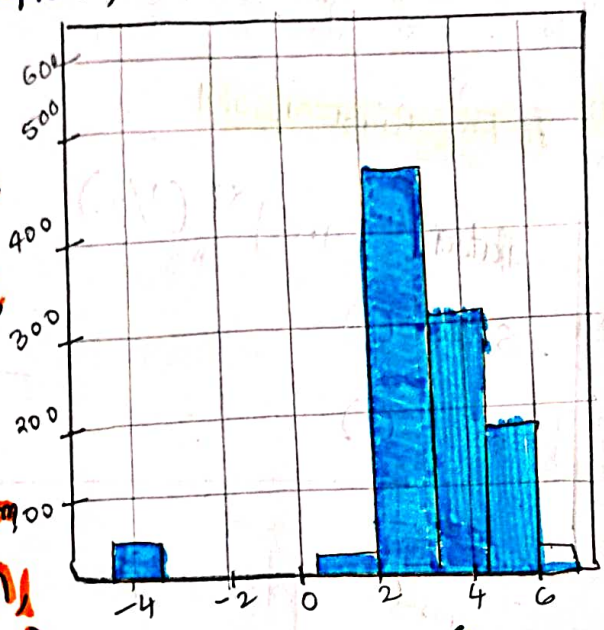
```
#titanic["sqr log_"Fare"] = np.log(titanic["Fare"]+0.01)
#titanic["sqr log_"Fare"].skew()
#titanic["sqr log_"Fare"].hist()
# plt.show()
```

Why?

Because In Data "Fare" column, we have some records "0" Values. So, If we calculate with log. it give a (infinite) Value, so, we add +0.01 to "0" column.

Out

-2.41004



Here, The Value, in This Have more, so we don't use log Transform For This col, Data)

Root Transformation For (Left Skewed) → n^{th} power → $(x)^2, (x)^3, (x)^4$ → left skewed

→ n^{th} root $(x)^{1/2}, (x)^{1/3}, (x)^{1/4}$ → Right skewed

we use "n" power

Fare data is not Left skewed, But To understand, Left skewed

```
titanic["sqr_Fare"] = titanic[Fare]**(2)
titanic["sqr_Fare"].skew()
titanic["sqr_Fare"].hist()
```

it can be any Value (2) (3) (4) ....

* when The Data is left skewed.

Out:

* We apply n^{th} power To The Data (or) column.

* Exponential

```
titanic["sqr_exp Fare"] = np.exp(titanic["Fare"])
titanic["sqr_exp Fare"].skew()
titanic["sqr_exp Fare"].hist()
plt.show
```

→ another method For Left Skewed Distibution

* Evaluates $e^1 x$ For Each Element in The given Input.

* Reciprocal Transformation

```
titanic["Rec_Fare"] = 1 / (titanic["Fare"]+0.01)
titanic["Rec-Fare"].skew()
titanic["Rec-Fare"].hist()
plt.show()
```

"Reciprocal" is that We divide Every value with (1)/[column] +0.0 to Elimi "0" value In The Data.

of The data .

* Box Cox Transformation

Fare
7.250
71.283
–
–
–
–

$\rightarrow \frac{7.250^{\lambda}-1}{\lambda}$

$\rightarrow \frac{71.283^{\lambda}-1}{\lambda}$

$$T[x] = \frac{x^{\lambda}-1}{\lambda}$$

∴ λ [lamba] Varies From -5 to +

∴ Where x is The response Variable and

⇒ In This Transformation, all Values o

"λ" are considered and The optimal Value for a given Variables are Selected.

```
from scipy import stats
# stats.boxcox (titanic["Fare"]+0.01)
```

If we don't write this value
Error: Data must be +ve
+0.01 (or) +1 (user defined)

Out :- (array([2.3844558, 6.43357655, 2.51273973, ...
2.607914, 5.7648427, 4.06763781, ...
...]),
which ever gives best result

From help
we see Two return values
1. array values (boxcox)
2. optimal Lamda parameter (max log)

```
# Fitting data, Fitting Lambda
# titanic["Fare_boxcox"], param = stats.boxcox
                                  (titanic["Fare"]+1)
```

Multiple Variable assignment
Out :- a, b = (1) (2)
stores

```
# print("λ=", param) (or) param
```

Out :- -0.09

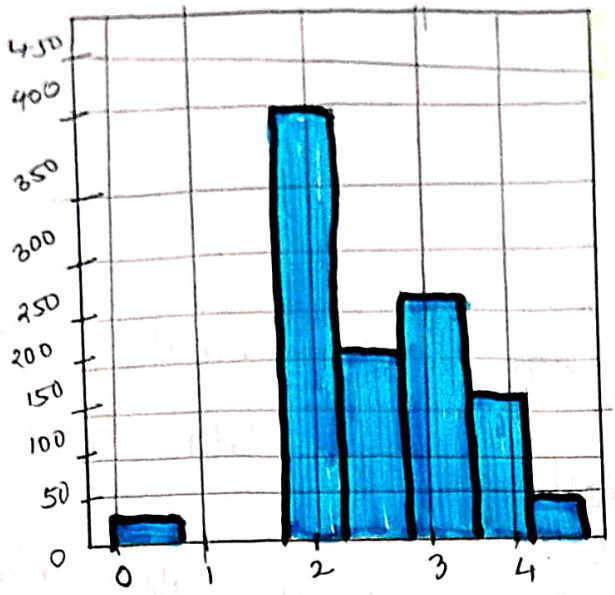
```
# titanic["Fare_boxcox"].skew()
```

Out :- -0.04 → lowest values while Compare with other Transformations

Code

```
# titanic["Fare_boxcox], param = stats.boxcox (titanic.Fare+1
# titanic["Fare_boxcox].hist()
# titanic["Fare_boxcox].skew()
```

Out :- -0.04



4/4/22
5:30pm.

5/4/22
11:00 AM

2 ⇒ Feature Scaling ⟹ Applicable only For Continous data

Feature Scaling ?

Ans:- it refers (or) Techniques used to normalize the ranges of independent Variables (input Variables) in our data. (or)

* The methods to set the Features value range within a similar Scale. (same scale)

* Variables with bigger magnitude / larger value range dominate over those with smaller magnitude / value range.

Ex:- 10,00,00 , 10 (x)
If we take
10 ——— 10 → Both are Equal Import

* Scale of the features is an important consideration when building Machine Learning models

* Feature scaling is generally the last step in the data pre processing pipeline, PerFormed Just Before Training the machine Learning algorithms.

5. Gradient descent converges Faster when Features are on Similar scales

* Feature Scaling *

Example :-

C.C	Milage
1600	14
1800	15
2200	16.5
2100	18
2600	22.5
1800	19.4
1900	25.4

of a car

Continous Data

Q. When we ask Machine which is important In The both columns ?

* Machine is Giving Importance To C.C Because it is Having Large Values.

* But, We know both are importa

So, we have train model in Initially such that, it should consider "Both The columns Similar

So, we Reduce The Value with 200

$= \frac{800}{1000} = \frac{80}{100} = \frac{8}{10} = \frac{4}{5}$ (Every one is Same)

with 100

We are scaling down Dividing Every Value with "10" In this case.

* To Reduce The Value.

* Feature Scaling Importance in Some ML Algorithms

1. Linear Regression & logistic Regression

* The regression coefficients of linear models are Directly influenced by Scale of The Variable.

2. Support Vector Machines :-

* Feature scaling helps Decrease The Time to Find Support Vectors For SVM's

3. K-means clustering

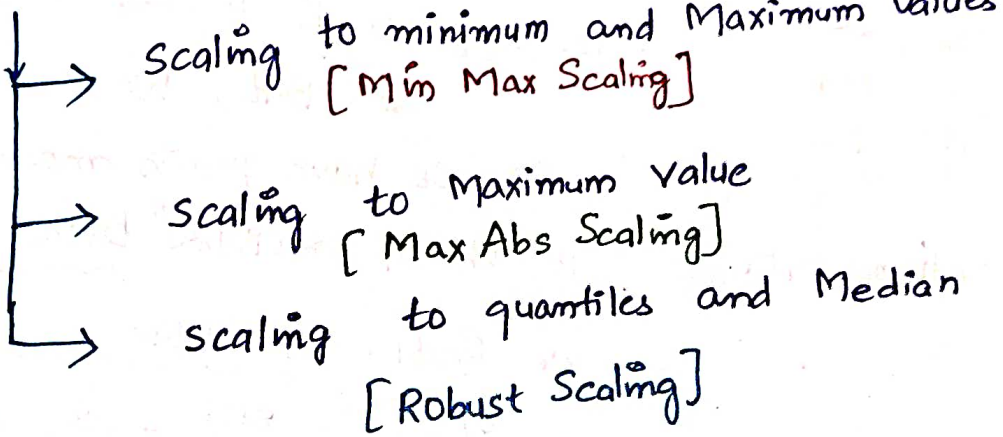
* Euclidean distances are Sensitive to Feature magnitude

4. Principal component analysis (PCA) :-

* PCA requires The Features to be Centered at "0".

* Various Feature Scaling Techniques :-

1. Standardisation
2. Normalisation Technique




```
# import pandas as pd
# import matplotlib.pyplot as plt
  %. matplotlib inline.
```

→ Using (titanic Data set)

code:

```
# titanic = pd.read_csv("titanic.csv", usecols=["Age"
  titanic.head()
```

Out

Age
22.0
38.0
26.0
35.0
35.0

check Null values in "Age" column.

```
# titanic.isnull().sum()
```

Out Age 177 → Null values

Replacing with "Median Value" For Null values

```
# titanic["Age"].fillna(titanic["Age"].median(),
                        inplace = True)
# titanic["Age"].isnull().sum()
```

Out : Age 0 → Zero Null values

I. **Standardisation**

↓

Converting Each Value To Zscore

Ex :-

X	$\frac{x-\mu}{\sigma}$ = Zscore
23	$\frac{23-60}{2}$ = -18.5
45	$\frac{45-60}{2}$ = -7.5
67	$\frac{67-60}{2}$ = 3.5
89	$\frac{89-60}{2}$ = 14.5
78	$\frac{78-60}{2}$ = 9

These Five values converting To Zscores is called as "Standard" Scaling

Avg/mean = 60.

(Std) $\sigma = \underset{Exp}{2}$

Code :-

```
From sklearn.Preprocessing import StandardScaler (SC)

# sc = StandardScaler ()        # call The Function (Short Cut)
                  ↓ main (don't Forget)
# titanic ["Age_sc"] = sc. fit_tranform (titanic [["Age"]
```

creating new column

fit_transform → calculation → Converting The value → stores in

```
# titanic ["Age_sc"]
```

Out :-

	Age_sc
0	-0.5657
1	0.6638
2	-0.2583
3	0.43312
⋮	⋮
890	0.20272

Here Every value is Converted To Zscore.

$$\left[\frac{X-\mu}{\sigma}\right] = Z\ score.$$

Ex :- Age [22] # First record X = 22

```
# titanic.mean()
μ = 29.361
# titanic.std ()
σ = 13.01
```

$$\left[\frac{X-\mu}{\sigma}\right] = \left[\frac{22-29.361}{13.01}\right]$$

"Age_sc" → [-0.5657]

```
# titanic
```

Out

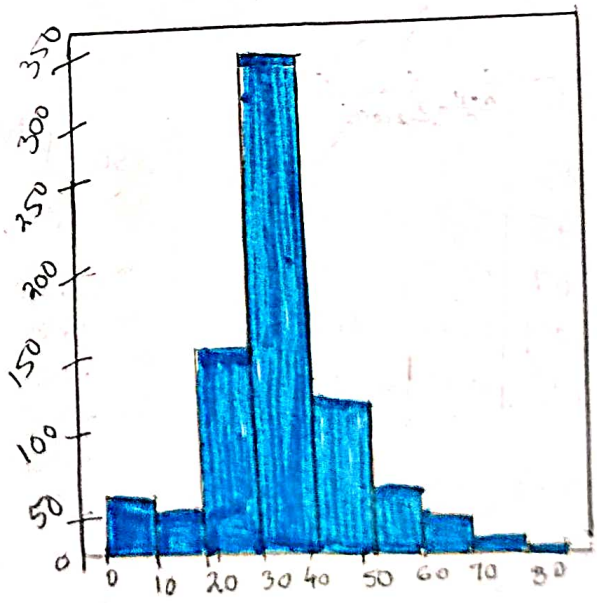
	Age	Age_sc
0	22.0	-0.5657
1	38.0	-0.6638
2	25.0	-0.2583
⋮	⋮	⋮
889	26.0	-0.2583
890	32.0	0.20272

Original Data

Converted To Z_Score

histogram of Original ["Age"]

```
# titanic ["Age"]. hist()
plt. show
```

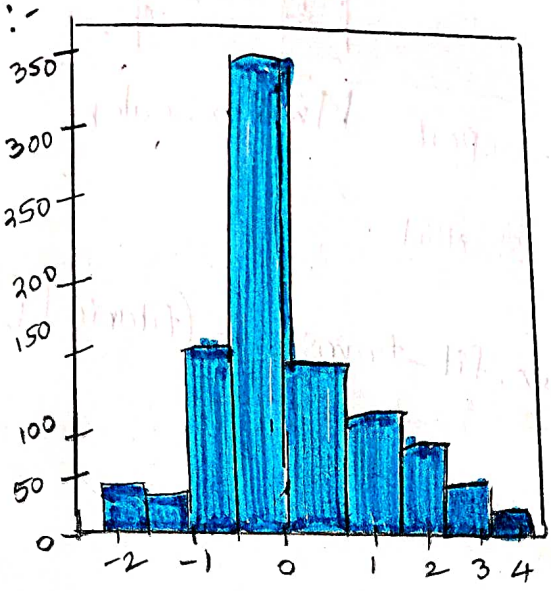


Histogram of Converted "Age"

titanic["Age_sc"].hist()

plt.show()

Out :-



Here, we see, Histogram is not going to change, But, Values does.

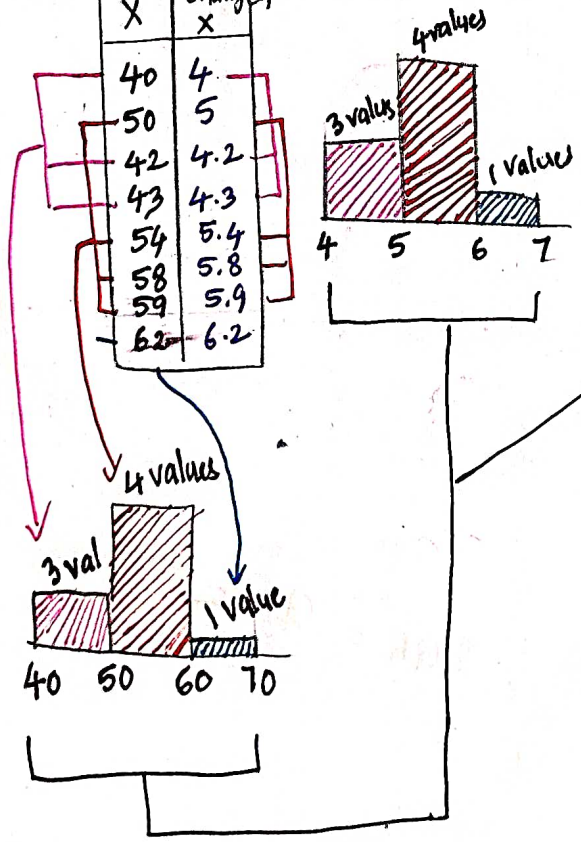
⇒ Example :-

* How ? Histogram Remains Same, But changed Values

Ans :-

Ex :-

X	Changed x
40	4
50	5
42	4.2
43	4.3
54	5.4
58	5.8
59	5.9
62	6.2



Here only "x" values Only changes But Histogram Remains Same.

II # Normalisation

* MinMax Scaling

⇒ MinMax Scaling Scales, The Values between "0" to "1" (Every Thing is (+ve) value No -ve value)

Ex:-

X	
65	$\frac{65-34}{53}$
45	$\frac{45-34}{53}$
34 ← min. value	$\frac{34-34}{53}=0$
67	$\frac{67-34}{53}$
67	
87 ← Max. value	$\frac{87-34}{53}=1$

$$\frac{X - X_{min}}{X_{max} - X_{min}}$$

MinMax Scaler

Code: From sklearn.Preprocessing import

```
# min_max = MinMaxScaler()  # Shortcut
# titanic ["Age_mm"] = min_max.fit_transform(titanic [["Age"]])
```

fit_transform

creating new column

```
# titanic   (calling/Show Data)
```

Out :-

	Age	Age_SC	Age_mm
0	22.0	-0.5653	0.2711
1	38.0	0.4337	0.4722
2	26.0	-0.1858	0.322
⋮	⋮	⋮	⋮
888	26.0	-0.25	0.32
890	32.0	0.20	0.39

* Robust Scaling

$$\frac{X - X_{median}}{IQR}$$

$\therefore IQR = Q_3 - Q_1$

Code:

```
From sklearn.preprocessing import RobustScaler

# rs = RobustScaler()  → shortcut()
# titanic["Age_rs"] = rs.fit_transform(titanic[["Age"]])
```

titanic["Age_rs"] (or) = RobustScaler().fit_transform(...)

If not using any shortcut like rs = RobustScaler()

```
# titanic
```

Out

Age	Age_SC	Age_Mm	Age_RS
22.0	0.565	0.461	0.4615
38.0	0.66	0.769	0.769
26.0	0.25	-0.153	-0.153
⋮			
26.0	-0.25	0.153	0.153
32.0	0.2	0.396	0.3076

* **Max Abs Scaling**

$$\frac{X}{X_{max}}$$

Ex:-

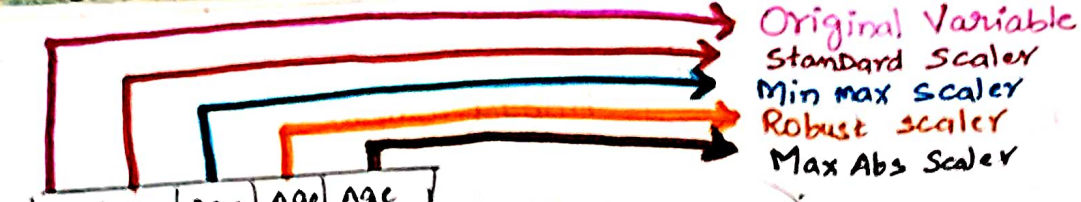
X	
68	68/80
44	44/80
52	52/80
69	69/80
80	80/80 = 1

Max value

code

```
From sklearn.preprocessing import MaxAbsScaler

# mas = MaxAbsScaler()

# titanic["Age_mas"] = mas.fit_transform(titanic[["Age"]]

# titanic
```

Out :-



	Age	Age_sc	Age_mm	Age_rs	Age_mas
0	.	.	.	.	0.2750
1	.	.	.	.	0.4150
2	.	.	.	.	0.3250
					⋮
889	.	.	.	.	0.3250
890	.	.	.	.	0.4000

891 x 5 col

5/04/22
4:58 pm.

Example :- Discretization

Continous Data ⇒ Milage of a Car (Discrete, categorical) Data

12.5
22.8
25.2
18.3
12.4
15.8
23.3

- <15 → low. milage
- 15-20 → Avg. milage
- 20-25 → good. milage
- >25 → Very good

→ By Our Own choice To Divide the into parts