Value count with percentage

df['al'].value_counts(normalize=True)

give percentage of distribution

df[''al"].apply(preprocessor)

call preprocessor method

Ploting distribution

ls = [10, 12, 40, 11, 30, 20, 20, 20, 20, , 11, 10]
plt.hist[ls, bins=30]

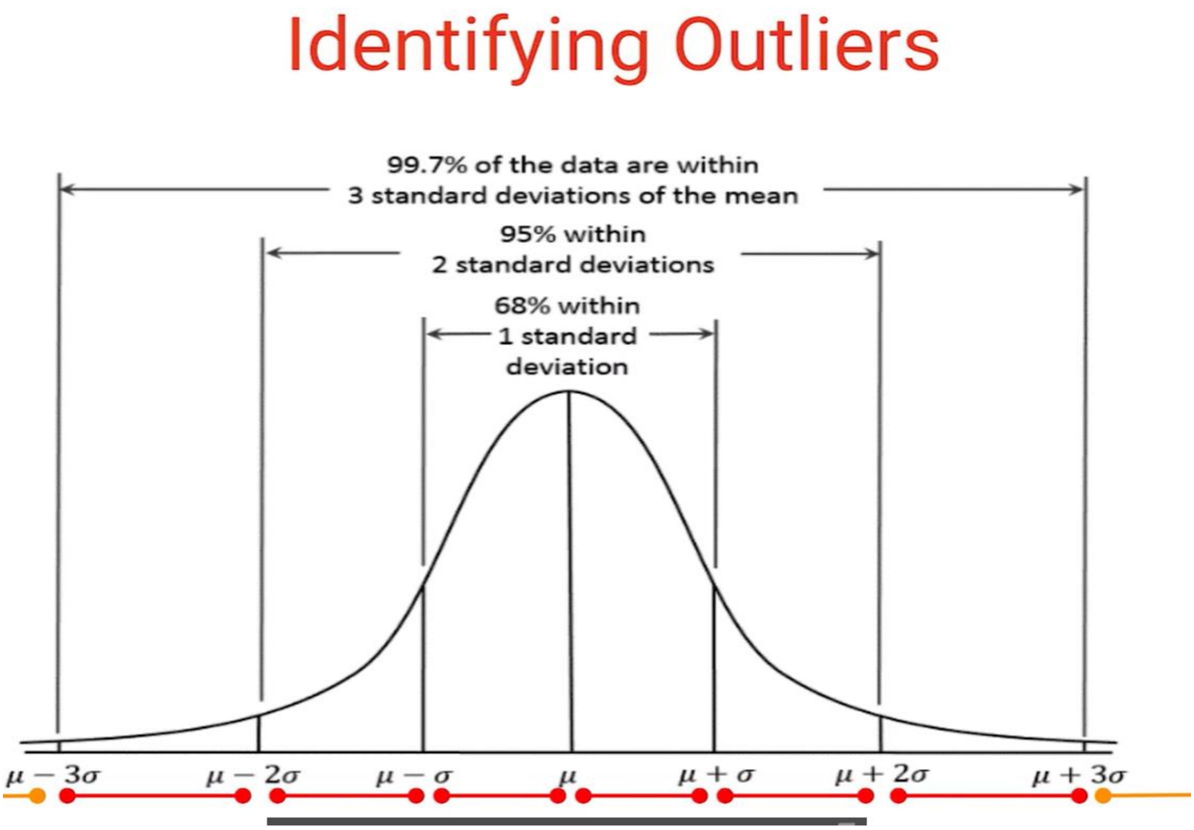
this will draw a distribution and show 10 - 2 times and 20 4 times and so on.

Univariate missing values

df.sum().isnull() : give individual count for missing values

Emperical rules

Any data outside 3 standard deviation is outlier

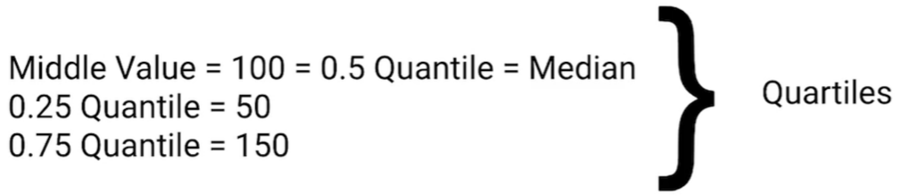


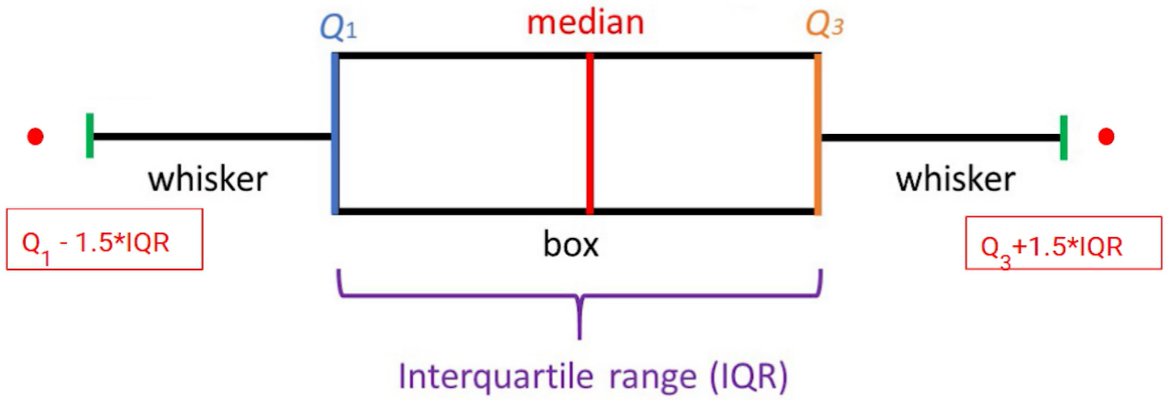
Data with Yellow points are outliers.

Rule only work if data is noraml distributed.but most of the cases data is not normally.

Quantiles

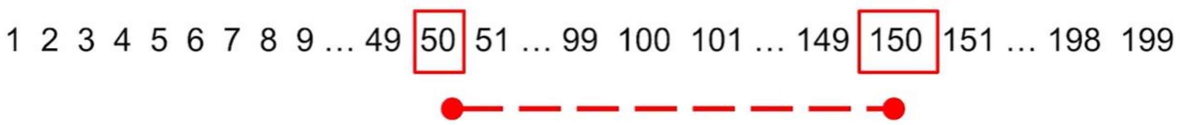
Similar to percanttile

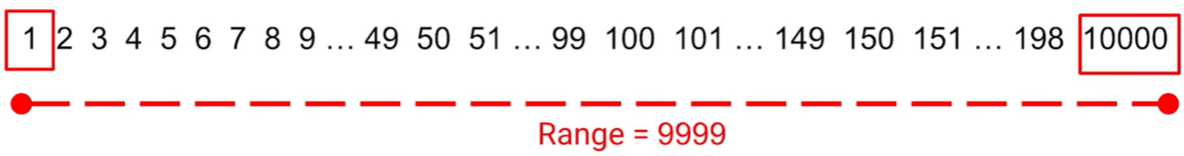




Work with any data set irrespective of its distribution.

Iterquartile Range vs Range





Random number creation

Import random

Random_ random = [random.randit(1, 2) fori in range(1,20)]

Panda data frame indexing

1. data.index = random_list

2. data.set_index('col1', drop=True, inplace =True)

   inplace = true is used to make chanes in the original dataframe

   drop = true is used to drop the colums that's set as index

3. data.reset_index(inplace=True)
   reset data frame to original form whatever experients we performed on index columns it change all

Subsetting data

data[data['col1']=='milk']

return all columns having milk as value

data[10:15:2]

it will skip every next row and return 10,12,14

data[-10:0]

return last 10 row

data.iloc[[1,2,3,4,5,6]]

select specific rows by index number

data.ilox[[1,4,5,2],[1,3,5]]

select 1,4,5,2 rows and 1,3,5 columns

data[] and data.loc[] same

iloc vs loc

- loc gets rows (and/or columns) with particular **labels.**
- iloc gets rows (and/or columns) at integer **locations.**

iloc is positional index(check underneath position instead of particular row)

loc is label based indexing(check what is value for particular row which we can see)

id	gender	grade	marks
A103	M	B	12
A104	M	B	14
A105	F	A	20
A102	F	A	21
A101	M	A	22

Iloc work with categorical data as well because it check for underneath position.

```
sample_df.iloc[2:4]
```

id	gender	grade	marks
A105	F	A	20
A102	F	A	21

```
sample_df.loc[2:4]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-62-a0b024ef820d> in <module>
----> 1 sample_df.loc[2:4]
```

Data.loc['A104':'A102'] : this will give right results.

isin

Data[data.year==2017 and data.year ==2018]

Data[data.year.isin([2017,2018])

Both will give same results.

Data.dtype

Data.select_dtypes['object']

Df[] not work properly some time

Df.loc work with label and categorical indexing

isna

data.isna().sum()

check all the null values

isna() gives true and false

isna only true when value is missing

sum() counts that true and false

mean

replace all missing item with mean value.

data.loc[data.item.isna()==True), 'item'] = data.item.mean()

value_counts

data.item.value_counts()

number of values in category

filllna

data.item.fillna('Medium' , inplace=True)

mode

data.item.mode()

give most frequent category as output

mapping

can change big categories name to smaller

```
# Create a new mapping (dictionary)
mapping = {
    'Low Fat' : 'LF',
    'Regular' : 'R',
    'LF' : 'LF',
    'reg': 'R',
    'low fat' : 'LF'
}
```

```
data.Item_Fat_Content.map(mapping)
```

```
# use the  map function to update the values
data.Item_Fat_Content = data.Item_Fat_Content.map(mapping)
```

```
# Count of new categories in the column Item_Fat_Content
data.Item_Fat_Content.value_counts()
```

Last line out put will be based on maping changes.

Apply

Create new columns in data frame

Inside apply we can use lambda

Data['item_usd'] = data.item.apply(lambda x : x/74)

Get dummies

Convert categorical variables into mumerical variables

```
df_train[df_train["text"].notnull()]
```
take only non null

Sort data

data.sort_values(by=[roll_no, marks], ascending=[True, False])

inplace = True update in original dataframe

now new dataframe have unsorted index. Lets use

data.reset_index(inplace=True, drop = True)

it will remove old unsorted index and add new index

Row wise merging of data

```
frames = [df_train, merged_df]
df_train = pd.concat(frames, axis =0)
```

Column wise merging of dataframe

```
frames = [df_train, merged_df]
df_train = pd.concat(frames, axis =1)
```

Join

Df2 = pd.DataFrame({

'roll_no':[102, 103]

})

df.merge(Df2, how= left, on='roll_no') #merge similarly as in sql

mean

data.item.mean()

describe

#get the summary

data.describe()

```
# get the summary
data.describe()
```

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

Agregation

Groupby

```
d1 = df.groupby('Animal')
```

Pivot

Pd.pivot_table(data, index = ‘type’ values=’mrp’ , aggfuc=’mean’)

Aggregate type columns and apply mean operation on mrp columns

Cross tab

Used to aggregate frequency of two or more factors.

Return give count

```
pd.crosstab(data['Outlet_Size'], data['Item_Type'])
```

Item_Type Outlet_Size	Baking Goods	Breads	Breakfast	Canned	Dairy	Frozen Foods	Fruits and Vegetables	Hard Drinks	Health and Hygiene	Household	Meat	Others	Seafood	Snack Foods
High	73	25	13	65	80	92	142	23	61	103	41	16	5	125
Medium	203	83	36	217	218	274	413	75	170	289	149	52	21	408
Small	187	71	30	189	198	249	328	50	136	257	119	55	20	335

Add mean in one columns

1st way

Data[avg_item’] = data.groupby([‘item_iden’])[‘item_vis’].transform(‘mean’)

Group by item_iden and take mean of item_vis and add new column.

2nd way

Calculate avg

```
average_item_visibility = data.groupby(['Item_Identifier'])['Item_Visibility'].mean().reset_index()
average_item_visibility
```

	Item_Identifier	Item_Visibility
0	DRA12	0.031956
1	DRA24	0.048062
2	DRA59	0.134718

Make a method and call method by apply

```
def get_item_visibility(x) :
    return average_item_visibility.loc[(average_item_visibility.Item_Identifier == x), 'Item_Visibility'].values[0]
```

```
# let's test it on the sample Item_Identifier
get_item_visibility('DRA24')
```

```
0.04806226414285714
```

Now, use the apply function to create the new feature. You just need to access the Item_Identifier column and use the apply method and pass the function that we have defined.

```
data['average_item_visibility'] = data.Item_Identifier.apply(get_item_visibility)
```

```
data.head()
```

Day time library

to_datetime

Convert object type to DateTime

day_name

2012-08-25 to Monday

month_name

2012-08-25 to August

dayofweek

dayofyear

import datetime

datetime.date.today()

Day time library directive

Directive	Meaning
%a	Weekday as locale's abbreviated name.
%A	Weekday as locale's full name.
%d	Day of the month as a zero-padded decimal number.
%b	Month as locale's abbreviated name.
%B	Month as locale's full name.
%m	Month as a zero-padded decimal number.
%y	Year without century as a zero-padded decimal number.
%Y	Year with century as a decimal number.
%H	Hour (24-hour clock) as a zero-padded decimal number.

tz_convert

```
data['utc_timezone'] = data.asia_timezone.dt.tz_convert('UTC')
```

timestamp

```
data_with_unix_ts.timestamp = pd.to_datetime(data_with_unix_ts.timestam
p, unit='s')
```

dropna(how="any")

```
# drop the null values
data_BM = data_BM.dropna(how="any")
```

plot

```
# draw the plot
plt.plot(calories_burnt,marker= 'o')
plt.plot(weight,'y--', marker='*')

# add legend in the lower right part of the figure
plt.legend(labels=['Calories Burnt', 'Weight'], loc='lower right')

# set labels for each of these persons
plt.xticks(ticks=[0,1,2,3], labels=['p1', 'p2', 'p3', 'p4']);
```

subplots

```
# create 2 plots
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12,6), sharex=True, s
harey=True)


# plot on 0 row and 0 column
ax[0].plot(calories_burnt,'go')

# plot on 0 row and 1 column
ax[1].plot(weight)

# set titles for subplots
ax[0].set_title("Calories Burnt")
ax[1].set_title("Weight")

# set ticks for each of these persons
ax[0].set_xticks(ticks=[0,1,2,3]);
ax[1].set_xticks(ticks=[0,1,2,3]);

# set labels for each of these persons
ax[0].set_xticklabels(labels=['p1', 'p2', 'p3', 'p4']);
ax[1].set_xticklabels(labels=['p1', 'p2', 'p3', 'p4']);
```

Line Chart

```
plt.plot(x, y, marker = 'o');
```

Bar Chart

```
plt.bar(x, y, color=['red', 'orange', 'magenta']);
```

Histogram

```
plt.hist(data_BM['Item_MRP'], bins=20, color='lightblue');
```

Box Plot

```
red_diamond = dict(markerfacecolor='y', marker='D')


plt.boxplot(data.values, labels=['Item Weight', 'Item MRP (price)'], fl
ierprops=red_diamond);
```

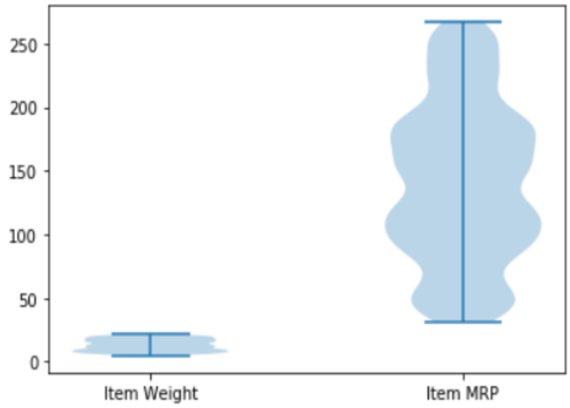
Scatter Plot

```
plt.scatter(data_BM["Item_Weight"][:200], data_BM["Item_Visibility"][:2
00]);
```

Violin Plots

```
# add labels to x axis
plt.xticks(ticks=[1,2], labels=['Item Weight', 'Item MRP'])

# make the violinplot
plt.violinplot(data.values);
```



scatter

```
plt.scatter(data_BM["Item_Weight"][:200], data_BM["Item_Visibility"][:2
00]);
```

https://courses.analyticsvidhya.com/courses/take/applied-machine-learning-beginner-to-professional/lessons/12903622-data-visualization-with-matplotlib

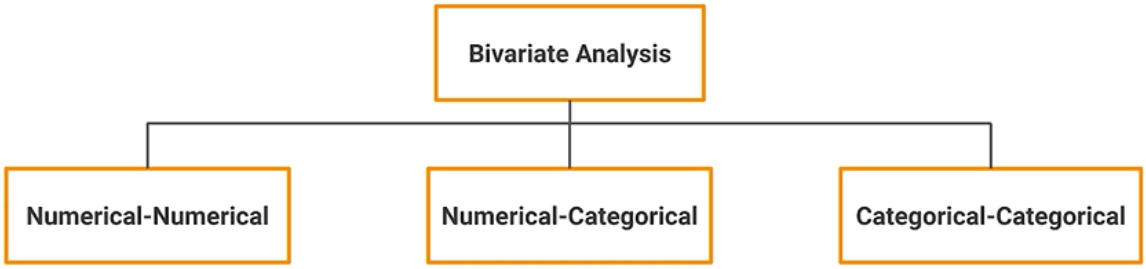
Bivariate analysis

Relation between two vairables

Hypothesis analysis

Ideas ofr features engineering and feature selection

Introduction to Bivariate Analysis



Variance : average squared difference of values from mean(only used for single variables)

Standard deviataion : square root of vairance

Coveriance

Only give the sense of direction

two variables (-inf to +inf)

Positive negative and zero covariance.

$$\text{Cov(X,Y)} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N}$$

Difficult to work for 0.0000045 and 30000000000000000

So to resolve this issue a concept comes into **correlations.**

Correlation

Relationship btn two sets of variables used to describes the direction and strength of the relationship.

Pearson Correlation:

$$Correlation = \frac{Cov\,(x,y)}{\sigma x * \sigma y} \qquad r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

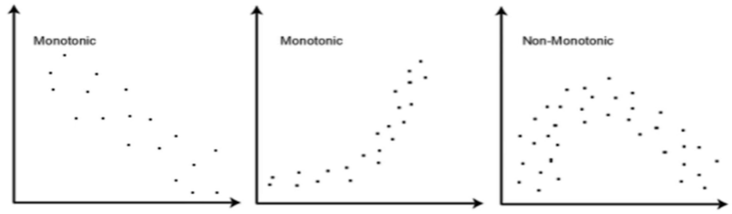
Covariance divided by the product of standard deviation of the two variables.

Values between -1 to +1

Strength of linear relation

Monotonic relationship

values of one vairables increases other decrases or vice versa,



Spearman Rho Correlation

Determines the strength and direction of the monotonic relationship

-1 to +1

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

How to calculated d^2:

Hours	Marks	Rank_Hours	Rank_Marks	difference	d_squared
16	66	9	4	5	25
35	70	3	2	1	1
5	40	10	10	0	0
31	60	4	7	3	9
22	65	6	5	1	1
24	56	5	9	4	16
18	59	8	8	0	0
40	77	1	1	0	0
36	67	2	3	1	1

Σd_squared = 54

Rho: greek letter

https://courses.analyticsvidhya.com/courses/take/applied-machine-learning-beginner-to-professional/lessons/12906590-spearman-s-correlation-kendall-s-tau

Kendall's Tau

C = Σ Concordant = 34

D = Σ Discordant = 11

τ = (C-D)/(C+D)

Hours	Marks	Hours_Rank	Marks_Rank	Concordant	Discordant
5	40	1	1	9	0
16	66	2	7	3	5
18	59	3	3	6	1
21	63	4	5	4	2
22	65	5	6	3	2
24	56	6	2	4	0
31	60	7	4	3	0
35	70	8	9	1	1
36	67	9	8	1	0
40	77	10	10	0	0

$C = \Sigma$ Concordant = 34

$D = \Sigma$ Discordant = 11

How many are below a particular rank and vice versa.

KT prefered over SR

SR is sensitive to outlier, KT is robust

SR based on difference btn in rank and KT based on concordnant and discordant pairs.

Small dataset -> KT

Big dataset -> SR

Usage

PCC

Linear relationship

Normally distributed(Assumption)

Continous data

No Outliers

SR and KT:

Monotic relationship

Outliers -.> KT

Not assumption of normal distribution of data

Only oridnal variables

Causation

cause and effect, is when one variables outcomes casue the other variable outocomes.

"Correlation is not causation"

In many cases correlation, are just because of the concidences.

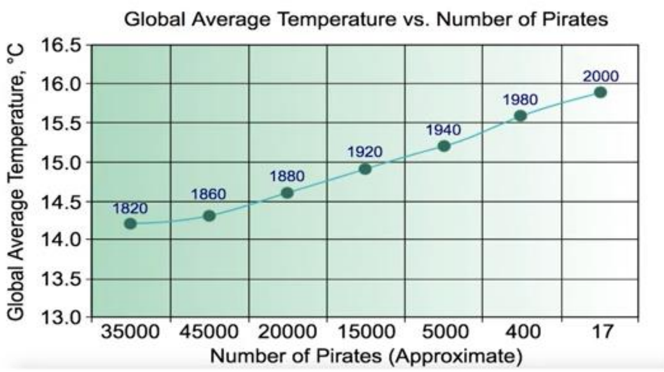
There is presence of one other factoe which is affecting the two variables.

This phenomina is also known as Spuriousness and spurioius coorelation

Population is compounding factor

So we have to take care not to make the direct conclusion.

Ex. Global Warming and Pirates



Increased Population → More fuel Consumption → Global Warming

Increased Population → More Pirates

Heat map : good coorelated variables btn data.

Scaterplot : two continous variables are vary and interact with each other.

KDE: too many data with unable to find the pattern or dense data