Pandas: Most frequent used methods

- 1.Introduction to pandas
- 2.Top 5 or bottom 5 records
- 3.renaming the column names
- 4.statistical description
- 5.adding new column
- 6.dropping column
- 7.Setting any column as index
- 8.resettig index
- 9.selecting row with index position
- 10.selecting subsection of the dataset with LOC and iloc
- 11.how to drop the row
- 12.conditional filtering using "&" and "|"
- 13.apply function on single column
- 14.apply func on multiple column using lambda func
- 15.sorting method
- 16.min , max and their index position
- 17.value_counts/unique/nunique/replace/map function
- 18.treatment of duplicate values
- 19.nlargest/nsmallest to get highest/lowest records
- 20.sample of the dataset (by numbers or percentage)
- 21.handling missing data
- 22.isnull/notnull
- 23.dropping missing values
- 24.filling missing values
- 25.group by operation on pandas
- 26.combining dataframe-->concatenation
- 27.combining dataframe--> merging-->join--> inner/left/right/outer
- 28.text method on string data
- 29.cleaning the data

1.Introduction to pandas

Pandas Library

- it is useful for data processing and Analysis

Pandas Dataframe

- pandas dataframe it is two dimentional tabular data structure with labelled axis (rows and columns)

- Dataframe is table of columns and rows in pandas that we can easily restructure and filter
- Formal Defination:=A group of pandas series Object that share the same index

In [218]:

```
# ls ===> to get the content(or the list of the all the files in same folder) in the dire
```

In [2]:

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

In [3]:

```
my_Data=np.random.randint(0,101,(4,3))
```

In [4]:

```
my_Data
```

Out[4]:

```
array([[ 2, 79, 18],
       [34, 90, 80],
       [ 9, 61,  2],
       [67, 87, 77]])
```

In [5]:

```
my_index=["india","Japan","Australia","newziland"]
```

In [6]:

```
column=["jan","feb","mar"]
```

In [7]:

```
df=pd.DataFrame(my_Data,index=my_index,columns=column)
df
```

Out[7]:

	jan	feb	mar
india	2	79	18
Japan	34	90	80
Australia	9	61	2
newziland	67	87	77

In [219]:

```
df=pd.read_csv("tips.csv")
```

2.Top 5 or bottom 5 records

In [220]:

```
df.head()
```

Out[220]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732

In [226]:

```
df.tail()
```

Out[226]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	C
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	52960686
240	27.18	2.00	Female	Yes	Sat	Dinner	2	13.59	Monica Sanders	35068061
241	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	60118916
242	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	43752
243	18.78	3.00	Female	No	Thur	Dinner	2	9.39	Michelle Hardin	35114516

3.To get the name of all the columns and how to rename the name of the columns

In [10]:

```
df.columns
```

Out[10]:

```
Index(['total_bill', 'tip', 'sex', 'smoker', 'day', 'time', 'size',
       'price_per_person', 'Payer Name', 'CC Number', 'Payment ID'],
      dtype='object')
```

```
#to get index
df.index
```

Out[11]:

```
RangeIndex(start=0, stop=244, step=1)
```

In [228]:

```
df=df.rename(columns={"Payer Name":"payer_name","CC Number":"cc_number"})
```

In [229]:

```
df.head()
```

Out[229]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	payer_name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	356032
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	447807
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	467613
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	483273

4.Statistical description

In [12]:

```
df.describe()
```

Out[12]:

	total_bill	tip	size	price_per_person	CC Number
count	244.000000	244.000000	244.000000	244.000000	2.440000e+02
mean	19.785943	2.998279	2.569672	7.888197	2.563496e+15
std	8.902412	1.383638	0.951100	2.914234	2.369340e+15
min	3.070000	1.000000	1.000000	2.880000	6.040679e+10
25%	13.347500	2.000000	2.000000	5.800000	3.040731e+13
50%	17.795000	2.900000	2.000000	7.255000	3.525318e+15
75%	24.127500	3.562500	3.000000	9.390000	4.553675e+15
max	50.810000	10.000000	6.000000	20.270000	6.596454e+15

In [13]:

```
df.describe().transpose()
```

Out[13]:

	count	mean	std	min	25%	5
total_bill	244.0	1.978594e+01	8.902412e+00	3.070000e+00	1.334750e+01	1.779500e+
tip	244.0	2.998279e+00	1.383638e+00	1.000000e+00	2.000000e+00	2.900000e+
size	244.0	2.569672e+00	9.510998e-01	1.000000e+00	2.000000e+00	2.000000e+
price_per_person	244.0	7.888197e+00	2.914234e+00	2.880000e+00	5.800000e+00	7.255000e+
CC Number	244.0	2.563496e+15	2.369340e+15	6.040679e+10	3.040731e+13	3.525318e+

To extract out single column or multiple column

In [223]:

```
df[["total_bill"]].head(3)
```

Out[223]:

	total_bill
0	16.99
1	10.34
2	21.01

In [224]:

```
df[["total_bill","tip"]].head(3)
```

Out[224]:

	total_bill	tip
0	16.99	1.01
1	10.34	1.66
2	21.01	3.50

5. Adding new column

In [17]:

```
df["tip_percentage"]=round(df['tip']/df["total_bill"]*100,2)
```

In [18]:

```
df["tip_percentage"]
```

Out[18]:

```
0       5.94
1      16.05
2      16.66
3      13.98
4      14.68
       ...
239    20.39
240     7.36
241     8.82
242     9.82
243    15.97
Name: tip_percentage, Length: 244, dtype: float64
```

In [19]:

```
df.head()
```

Out[19]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732

6.How to drop the column

In [20]:

```
df.drop("tip_percentage",axis=1,inplace=True)
```

In [1]:

```
# in case of deleting multiple column
# df.drop(['CC Number','tip_percentage'].axis=1)
```

In [21]:

```
df.head()
```

Out[21]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732

7.How to set any column as index

In [23]:

```
df=df.set_index("Payment ID").head()  # in order to save this permanantly you have to sav
```

In [24]:

```
df.head(2)
```

Out[24]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker

8.In order to reset index

In [26]:

```
df=df.reset_index()
```

In [27]:

```
df.head(2)
```

Out[27]:

	Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Paye Nam
0	Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christ Cunningha
1	Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Dougla Tucke

9.To select any specific rows with index postion

In [28]:

```
df.iloc[0]
```

Out[28]:

```
Payment ID                    Sun2959
total_bill                      16.99
tip                              1.01
sex                            Female
smoker                             No
day                               Sun
time                           Dinner
size                                2
price_per_person                 8.49
Payer Name          Christy Cunningham
CC Number             3560325168603410
Name: 0, dtype: object
```

10. For selecting some subsection of the dataframe

In [29]:

```
df.iloc[1:5]
```

Out[29]:

	Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name
1	Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker
2	Sun4458	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters
3	Sun5260	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris
4	Sun2251	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter

In [30]:

```
df=df.set_index("Payment ID")
```

In [31]:

```
df.head(2)
```

Out[31]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker

In [32]:

```
df.loc[["Sun2959","Sun4608"]]
```

Out[32]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker

In [33]:

```
df.head(2)
```

Out[33]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name
Sun2959	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker

11.How to drop the row

In [34]:

```
df.drop("Sun2959",axis=0)
```

Out[34]:

Payment ID	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
Sun4608	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	447
Sun4458	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	60
Sun5260	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	467
Sun2251	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	483

12. Condtional Filtering

- typically in data analysis our datasets are large enough that we dont filter based on positon but based on some condition
- conditional formating allows us to select row based condtion on the column
- this leads to discussion on organizing the data

- condtion filtering
- filter by single condition
- filter by multiple condtion
- check against multiple values

In [35]:

```
new_df=pd.read_csv("tips.csv")
new_df.head(2)
```

Out[35]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071

In [36]:

```
#let filter out for the bill values greater than 40 dollars
new_df[new_df["total_bill"]>40]
```

Out[36]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	6596
95	40.17	4.73	Male	Yes	Fri	Dinner	4	10.04	Aaron Bentley	180
102	44.30	2.50	Female	Yes	Sat	Dinner	3	14.77	Heather Cohen	379
142	41.19	5.00	Male	No	Thur	Lunch	5	8.24	Eric Andrews	43565
156	48.17	5.00	Male	No	Sun	Dinner	6	8.03	Ryan Gonzales	35231
170	50.81	10.00	Male	Yes	Sat	Dinner	3	16.94	Gregory Clark	5473
182	45.35	3.50	Male	Yes	Sun	Dinner	3	15.12	Jose Parsons	41122
184	40.55	3.00	Male	Yes	Sun	Dinner	2	20.27	Stephen Cox	3547
197	43.11	5.00	Female	Yes	Thur	Lunch	4	10.78	Brooke Soto	55449
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	6

new_df["total_bill"]>40 ==> will only give the boolean values, that is true and false where as outer function will filter out ouly true value of the dataframe

In [37]:

```
# number of male customer
new_df[new_df["sex"]=="Male"].count()
```

Out[37]:

```
total_bill          157
tip                 157
sex                 157
smoker              157
day                 157
time                157
size                157
price_per_person    157
Payer Name          157
CC Number           157
Payment ID          157
dtype: int64
```

In [38]:

```
# how to filter out with multiple condition

# and --> & ==> where both condition need to be true

# OR --> | ==> where either of condition must be true
```

In [39]:

```
# show me the male who have paid the total bill more than 30

new_df[(new_df["total_bill"]>30) & (new_df["sex"]=="Male")].head(5)
```

Out[39]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	CC
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	35425840
39	31.27	5.00	Male	No	Sat	Dinner	3	10.42	Mr. Brandon Berry	60115258
44	30.40	5.60	Male	No	Sun	Dinner	4	7.60	Todd Cooper	5038
47	32.40	6.00	Male	No	Sun	Dinner	4	8.10	James Barnes	35520025
56	38.01	3.00	Male	Yes	Sat	Dinner	4	9.50	James Christensen DDS	3497936

In [40]:

```
new_df[(new_df["total_bill"]>30) | (new_df["sex"]=="Male")].tail(5)
```

Out[40]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	C
237	32.83	1.17	Male	Yes	Sat	Dinner	2	16.42	Thomas Brown	42847226
238	35.83	4.67	Female	No	Sat	Dinner	3	11.94	Kimberly Crane	6761
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	52960686
241	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	60118916
242	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	43752

In [41]:

```
# another way of filtering is with isin function
```

In [42]:

```
option=["Sat","Sun"]
```

In [43]:

```
new_df[new_df["day"].isin(option)]
```

Out[43]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	35603
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	44780
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	46761
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	48327
...	...	...	...	...	...	...	...	...	...	
238	35.83	4.67	Female	No	Sat	Dinner	3	11.94	Kimberly Crane	6
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	52960
240	27.18	2.00	Female	Yes	Sat	Dinner	2	13.59	Monica Sanders	35068
241	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	60118
242	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	43

163 rows × 11 columns

13. Use of apply function

In [44]:

```
#what if i want to grab last four digit number of credit card number
#this can be acchieved by .apply function
```

In [45]:

```
def Last_Four(num):
    return str(num)[-4:]
```

In [46]:

```
Last_Four(945723487)
```

Out[46]:

```
'3487'
```

In [47]:

```
new_df["Last Four"]=new_df["CC Number"].apply(Last_Four)
```

In [48]:

```
new_df.head(2)
```

Out[48]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071

In [49]:

```
# based on price assing dollar sign== <10 -- "$",10-30--"$$",>30="$$$"
```

In [50]:

```
def yelp(price):
    if price<10:
        return "$"
    elif price >=10 and price <30:
        return "$$"
    else:
        return "$$$"
```

In [51]:

```
new_df["Yelp"]=new_df["total_bill"].apply(yelp)
```

In [52]:

```
new_df.head(4)
```

Out[52]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137

14. Apply function with multiple columns

In [53]:

```
# problem statement is ==> based on the ratio totalbill and tip categories customer wheth
```

In [54]:

```
def quality(total_bill,tip):
    if total_bill/tip>0.25:
        return "Generous"
    else:
        return "other"
```

In [55]:

```
quality(16.00,1.01)
```

Out[55]:

```
'Generous'
```

In [86]:

```
new_df["Quality"]=new_df[["total_bill","tip"]].apply(lambda new_df:quality(new_df["total_
```

In [ ]:

```
# same result can be obtained by vectorize function
```

In [88]:

```
new_df["Quality"]=np.vectorize(quality)(new_df["total_bill"],new_df["tip"])
```

In [89]:

```
#both will give the same result but vectorize is lil easy to remember and it works faster
```

15. Sorting method in pandas

In [91]:

```
df=pd.read_csv("tips.csv")
```

In [92]:

```
df.head(2)
```

Out[92]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071

In [93]:

```
# describe we already know we will directly see the sorting
```

In [96]:

```
# let sort the values by tip value
df.sort_values("tip",ascending=False).head(5)
```

Out[96]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
170	50.81	10.00	Male	Yes	Sat	Dinner	3	16.94	Gregory Clark	5473850
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	676
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	3542584
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	659645
141	34.30	6.70	Male	No	Thur	Lunch	6	5.72	Steven Carlson	352651

In [98]:

```
# we can do sorting with multiple column as well
df.sort_values(["tip","size"]).head()
```

Out[98]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
67	3.07	1.00	Female	Yes	Sat	Dinner	1	3.07	Tiffany Brock	43594
111	7.25	1.00	Female	No	Sat	Dinner	1	7.25	Terri Jones	3559
92	5.75	1.00	Female	Yes	Fri	Dinner	2	2.88	Leah Ramirez	3508
236	12.60	1.00	Male	Yes	Sat	Dinner	2	6.30	Matthew Myers	35436
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560

16. min max and there index position

In [99]:

```
#what is max value of total bill and what is its index position
df["total_bill"].max()
```

Out[99]:

```
50.81
```

In [100]:

```
df["total_bill"].idxmax()
```

Out[100]:

```
170
```

- max amount of total bill is 50.81 and its index position is 170

In [101]:

```
#what is min value of total bill and what is its index position
df["total_bill"].min()
```

Out[101]:

```
3.07
```

In [102]:

```
#what is min value of total bill and what is its index position
df["total_bill"].idxmin()
```

Out[102]:

```
67
```

In [103]:

```
df.iloc[67]
```

Out[103]:

```
total_bill                      3.07
tip                              1.0
sex                           Female
smoker                           Yes
day                              Sat
time                          Dinner
size                               1
price_per_person                3.07
Payer Name              Tiffany Brock
CC Number           4359488526995267
Payment ID                   Sat3455
Name: 67, dtype: object
```

In [104]:

```
df.corr()
```

Out[104]:

	total_bill	tip	size	price_per_person	CC Number
total_bill	1.000000	0.675734	0.598315	0.647554	0.104576
tip	0.675734	1.000000	0.489299	0.347405	0.110857
size	0.598315	0.489299	1.000000	-0.175359	-0.030239
price_per_person	0.647554	0.347405	-0.175359	1.000000	0.135240
CC Number	0.104576	0.110857	-0.030239	0.135240	1.000000

17. value counts/unique/nunique/replace/map func

In [105]:

```
df["sex"].value_counts()
```

Out[105]:

```
Male      157
Female     87
Name: sex, dtype: int64
```

In [106]:

```
df["day"].unique()
```

Out[106]:

```
array(['Sun', 'Sat', 'Thur', 'Fri'], dtype=object)
```

In [107]:

```
df["day"].nunique()
```

Out[107]:

```
4
```

In [110]:

```
# how to use replace method==> replace female with "F" and Male with "M"
df.head(2)
```

Out[110]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071

In [111]:

```
df["sex"].replace(["Female","Male"],["F","M"])
```

Out[111]:

```
0      F
1      M
2      M
3      M
4      F
      ..
239    M
240    F
241    M
242    M
243    F
Name: sex, Length: 244, dtype: object
```

- Another way of doing the same thing is mapping

In [113]:

```
mymap={"Female":"F","Male":"M"}
```

In [114]:

```
df["sex"].map(mymap)
```

Out[114]:

```
0      F
1      M
2      M
3      M
4      F
      ..
239    M
240    F
241    M
242    M
243    F
Name: sex, Length: 244, dtype: object
```

18. How to treat duplicate values

In [117]:

```
df.duplicated()
```

Out[117]:

```
0      False
1      False
2      False
3      False
4      False
       ...
239    False
240    False
241    False
242    False
243    False
Length: 244, dtype: bool
```

```
df.drop_duplicates().head(5)   # to drop the duplicates
```

Out[120]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560325
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478071
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	601181
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676137
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832732

In [123]:

```
# use of between function
df[df["total_bill"].between(10,30,inclusive=True)]
```

```
C:\Users\SSRVC\AppData\Local\Temp\ipykernel_14472\1154451165.py:2: FutureW
arning: Boolean inputs to the `inclusive` argument are deprecated in favou
r of `both` or `neither`.
  df[df["total_bill"].between(10,30,inclusive=True)]
```

Out[123]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Name	
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	3560
1	10.34	1.66	Male	No	Sun	Dinner	3	3.45	Douglas Tucker	4478
2	21.01	3.50	Male	No	Sun	Dinner	3	7.00	Travis Walters	6011
3	23.68	3.31	Male	No	Sun	Dinner	2	11.84	Nathaniel Harris	4676
4	24.59	3.61	Female	No	Sun	Dinner	4	6.15	Tonya Carter	4832
...	...	...	...	...	...	...	...	...	...	
239	29.03	5.92	Male	No	Sat	Dinner	3	9.68	Michael Avila	5296
240	27.18	2.00	Female	Yes	Sat	Dinner	2	13.59	Monica Sanders	3506
241	22.67	2.00	Male	Yes	Sat	Dinner	2	11.34	Keith Wong	6011
242	17.82	1.75	Male	No	Sat	Dinner	2	8.91	Dennis Dixon	4
243	18.78	3.00	Female	No	Thur	Dinner	2	9.39	Michelle Hardin	3511

195 rows × 11 columns

19. nlargest and nsmallest

In [125]:

```
#show me the 5 largest tip
df.nlargest(5,"tip")
```

Out[125]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Name	
170	50.81	10.00	Male	Yes	Sat	Dinner	3	16.94	Gregory Clark	5473850
212	48.33	9.00	Male	No	Sat	Dinner	4	12.08	Alex Williamson	676
23	39.42	7.58	Male	No	Sat	Dinner	4	9.86	Lance Peterson	3542584
59	48.27	6.73	Male	No	Sat	Dinner	4	12.07	Brian Ortiz	6596453
141	34.30	6.70	Male	No	Thur	Lunch	6	5.72	Steven Carlson	3526515

In [126]:

```
# show me the 5 smallest tips
df.nsmallest(5,"tip")
```

Out[126]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Name	
67	3.07	1.00	Female	Yes	Sat	Dinner	1	3.07	Tiffany Brock	43594
92	5.75	1.00	Female	Yes	Fri	Dinner	2	2.88	Leah Ramirez	3508
111	7.25	1.00	Female	No	Sat	Dinner	1	7.25	Terri Jones	35592
236	12.60	1.00	Male	Yes	Sat	Dinner	2	6.30	Matthew Myers	35436
0	16.99	1.01	Female	No	Sun	Dinner	2	8.49	Christy Cunningham	35603

20. Random sample of the dataset by the number or percentage

In [128]:

```
df.sample(5)   # any five random rows will be selected
```

Out[128]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	C
51	10.29	2.60	Female	No	Sun	Dinner	2	5.14	Jessica Ibarra	49997
81	16.66	3.40	Male	No	Thur	Lunch	2	8.33	William Martin	45505490
38	18.69	2.31	Male	No	Sat	Dinner	3	6.23	Brandon Bradley	44276015
108	18.24	3.76	Male	No	Sat	Dinner	2	9.12	Steven Grant	41128104
208	24.27	2.03	Male	Yes	Sat	Dinner	2	12.14	Jason Carter	42689429

In [129]:

```
df.sample(frac=0.05)   # 5% of the data will be selected ramdomly
```

Out[129]:

	total_bill	tip	sex	smoker	day	time	size	price_per_person	Payer Name	
201	12.74	2.01	Female	Yes	Thur	Lunch	2	6.37	Abigail Parks	358664
188	18.15	3.50	Female	Yes	Sun	Dinner	3	6.05	Glenda Wiggins	57
138	16.00	2.00	Male	Yes	Thur	Lunch	2	8.00	Jason Burgess	356146
218	7.74	1.44	Male	Yes	Sat	Dinner	2	3.87	Nicholas Archer	34051
69	15.01	2.09	Male	Yes	Sat	Dinner	2	7.50	Adam Hall	470092
154	19.77	2.00	Male	No	Sun	Dinner	4	4.94	James Smith	21316
205	16.47	3.23	Female	Yes	Thur	Lunch	3	5.49	Carly Reyes	478
198	13.00	2.00	Female	Yes	Thur	Lunch	2	6.50	Katherine Bond	492
200	18.71	4.00	Male	Yes	Thur	Lunch	3	6.24	Jason Conrad	458
132	11.17	1.50	Female	No	Thur	Lunch	2	5.58	Taylor Gonzalez	601199
228	13.28	2.72	Male	No	Sat	Dinner	2	6.64	Glenn Jones	50
126	8.52	1.48	Male	No	Thur	Lunch	2	4.26	Mario Bradshaw	452440

21. How to handle missing data

- real world data will ofter be missing data for variety of reason
- many machine learning models and statistical methods cant work with missing data in such case we need to decide what to do with missing data
- when reading missing values pandas will display them as NaN values

Option for missing data

- keep it
- remove it
- replace it

Keeping the missing data

- pros --> Does not manipulate or change the true data

- Cons --> Many method or model do not support NaN values

Dropping or Removing the missing data

- pros --> easy/ can be based on rules
- Cons --> chance of lossing lot of data or usefull information can also be loss

Filling the missing data

- pros --> potential to save lot of data for use of training a model
- Cons --> Hardest to do and somewhat arbitrary --> potentail to lead false conclusion

In [132]:

```
df=pd.read_csv("movie_scores.csv")
```

In [133]:

```
df.head()
```

Out[133]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

22. isnull and notnull

In [134]:

```
df.isnull()   # wherever there is missing value it shows true
```

Out[134]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	False	False	False	False	False	False
1	True	True	True	True	True	True
2	False	False	False	False	True	True
3	False	False	False	False	False	False
4	False	False	False	False	False	False

In [135]:

```
df.notnull()   # reverse of isnull will indicate by false wherever there is missing value
```

Out[135]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	True	True	True	True	True	True
1	False	False	False	False	False	False
2	True	True	True	True	False	False
3	True	True	True	True	True	True
4	True	True	True	True	True	True

In [ ]:

```
#show me the rows where all the promovie score is given
```

In [137]:

```
df[df["pre_movie_score"].notnull()]
```

Out[137]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [ ]:

```
#show me the rows where all the promovie score is not given
```

In [139]:

```
df[df["pre_movie_score"].isnull()]
```

Out[139]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
1	NaN	NaN	NaN	NaN	NaN	NaN
2	Hugh	Jackman	51.0	m	NaN	NaN

In [ ]:

```
#show me the rows where all the promovie score is not given but first name is given
```

In [140]:

```
df[(df["pre_movie_score"].isnull()) & (df["first_name"].notnull())]
```

Out[140]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
2	Hugh	Jackman	51.0	m	NaN	NaN

In [225]:

```
#help(df.dropna)
```

23.Dropping

In [146]:

```
# this will remove the row which contail even one missing value and will not consider oth
df.dropna()
```

Out[146]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [148]:

```
# to prevent from this we can use threshold==> it means only drop the row if it contain 1
df.dropna(thresh=1)
```

Out[148]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
2	Hugh	Jackman	51.0	m	NaN	NaN
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

24. Filling Na values

In [ ]:

```
#help(df.fillna)
```

In [151]:

```
df.fillna("xjd")
```

Out[151]:

	first_name	last_name	age	sex	pre_movie_score	post_movie_score
0	Tom	Hanks	63.0	m	8.0	10.0
1	xjd	xjd	xjd	xjd	xjd	xjd
2	Hugh	Jackman	51.0	m	xjd	xjd
3	Oprah	Winfrey	66.0	f	6.0	8.0
4	Emma	Stone	31.0	f	7.0	9.0

In [155]:

```
# fill with mean values
df["pre_movie_score"].fillna(df["pre_movie_score"].mean())
```

Out[155]:

```
0    8.0
1    7.0
2    7.0
3    6.0
4    7.0
Name: pre_movie_score, dtype: float64
```

- there is one more method of filling the missing values it is interpolate when the category arrange in linear order and there is missing values this function find interpolate value considering its linear order

25. Group By operation on Pandas

- A groupby() operation allows us to examine data per category basis
- group by is applicable on either categorical or discrete(when the column contain numerical column) column

In [157]:

```
df=pd.read_csv("mpg.csv")
df.head()
```

Out[157]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	na
0	18.0	8	307.0	130	3504	12.0	70	1	chevr chev ma
1	15.0	8	350.0	165	3693	11.5	70	1	b sky
2	18.0	8	318.0	150	3436	11.0	70	1	plymc sate
3	16.0	8	304.0	150	3433	12.0	70	1	rebel
4	17.0	8	302.0	140	3449	10.5	70	1	to

In [158]:

```
df["model_year"].value_counts()
```

Out[158]:

```
73    40
78    36
76    34
82    31
75    30
70    29
79    29
80    29
81    29
71    28
72    28
77    28
74    27
Name: model_year, dtype: int64
```

In [159]:

```
df.groupby(df["model_year"]).mean()
```

Out[159]:

model_year	mpg	cylinders	displacement	weight	acceleration	origin
70	17.689655	6.758621	281.413793	3372.793103	12.948276	1.310345
71	21.250000	5.571429	209.750000	2995.428571	15.142857	1.428571
72	18.714286	5.821429	218.375000	3237.714286	15.125000	1.535714
73	17.100000	6.375000	256.875000	3419.025000	14.312500	1.375000
74	22.703704	5.259259	171.740741	2877.925926	16.203704	1.666667
75	20.266667	5.600000	205.533333	3176.800000	16.050000	1.466667
76	21.573529	5.647059	197.794118	3078.735294	15.941176	1.470588
77	23.375000	5.464286	191.392857	2997.357143	15.435714	1.571429
78	24.061111	5.361111	177.805556	2861.805556	15.805556	1.611111
79	25.093103	5.827586	206.689655	3055.344828	15.813793	1.275862
80	33.696552	4.137931	115.827586	2436.655172	16.934483	2.206897
81	30.334483	4.620690	135.310345	2522.931034	16.306897	1.965517
82	31.709677	4.193548	128.870968	2453.548387	16.638710	1.645161

In [161]:

```
df.groupby(["model_year","cylinders"]).mean()
```

Out[161]:

model_year	cylinders	mpg	displacement	weight	acceleration	origin
70	4	25.285714	107.000000	2292.571429	16.000000	2.285714
	6	20.500000	199.000000	2710.500000	15.500000	1.000000
	8	14.111111	367.555556	3940.055556	11.194444	1.000000
71	4	27.461538	101.846154	2056.384615	16.961538	1.923077
	6	18.000000	243.375000	3171.875000	14.750000	1.000000
	8	13.428571	371.714286	4537.714286	12.214286	1.000000
72	3	19.000000	70.000000	2330.000000	13.500000	3.000000
	4	23.428571	111.535714	2382.642857	17.214286	1.928571
	8	13.615385	344.846154	4228.384615	13.000000	1.000000
73	3	18.000000	70.000000	2124.000000	13.500000	3.000000
	4	22.727273	109.272727	2338.090909	17.136364	2.000000
	6	19.000000	212.250000	2917.125000	15.687500	1.250000
	8	13.200000	365.250000	4279.050000	12.250000	1.000000
74	4	27.800000	96.533333	2151.466667	16.400000	2.200000
	6	17.857143	230.428571	3320.000000	16.857143	1.000000
	8	14.200000	315.200000	4438.400000	14.700000	1.000000
75	4	25.250000	114.833333	2489.250000	15.833333	2.166667
	6	17.583333	233.750000	3398.333333	17.708333	1.000000
	8	15.666667	330.500000	4108.833333	13.166667	1.000000
76	4	26.766667	106.333333	2306.600000	16.866667	1.866667
	6	20.000000	221.400000	3349.600000	17.000000	1.300000
	8	14.666667	324.000000	4064.666667	13.222222	1.000000
77	3	21.500000	80.000000	2720.000000	13.500000	3.000000
	4	29.107143	106.500000	2205.071429	16.064286	1.857143
	6	19.500000	220.400000	3383.000000	16.900000	1.400000
	8	16.000000	335.750000	4177.500000	13.662500	1.000000
78	4	29.576471	112.117647	2296.764706	16.282353	2.117647
	5	20.300000	131.000000	2830.000000	15.900000	2.000000
	6	19.066667	213.250000	3314.166667	16.391667	1.166667
	8	19.050000	300.833333	3563.333333	13.266667	1.000000
79	4	31.525000	113.583333	2357.583333	15.991667	1.583333
	5	25.400000	183.000000	3530.000000	20.100000	2.000000
	6	22.950000	205.666667	3025.833333	15.433333	1.000000
	8	18.630000	321.400000	3862.900000	15.400000	1.000000

model_year	cylinders	mpg	displacement	weight	acceleration	origin
80	3	23.700000	70.000000	2420.000000	12.500000	3.000000
	4	34.612000	111.000000	2360.080000	17.144000	2.200000
	5	36.400000	121.000000	2950.000000	19.900000	2.000000
	6	25.900000	196.500000	3145.500000	15.050000	2.000000
81	4	32.814286	108.857143	2275.476190	16.466667	2.095238
	6	23.428571	184.000000	3093.571429	15.442857	1.714286
	8	26.000000	350.000000	3725.000000	19.000000	1.000000
82	4	32.071429	118.571429	2402.321429	16.703571	1.714286
	6	28.333333	225.000000	2931.666667	16.033333	1.000000

In [164]:

```
#describe function with group by
#df.groupby("model_year").describe().transpose()
```

26. Combining dataframe --> Concatenation

- often the data you need exist in two separate sources , fortunately pandas makes it easy to combine these together
- the simplest combination is if both sources already in the same format then concatenation through pd.concat() call is that all needed

In [165]:

```
data_one={"A":["A0","A1","A2","A3"],"B":["B0","B1","B2","B3"]}
```

In [166]:

```
data_Two={"C":["C0","C1","C2","C3"],"D":["D0","D1","D2","D3"]}
```

In [168]:

```
df1=pd.DataFrame(data_one)
```

In [169]:

```
df2=pd.DataFrame(data_Two)
```

In [170]:

```
df1
```

Out[170]:

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

In [171]:

```
df2
```

Out[171]:

	C	D
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

In [173]:

```
# concate them along column
pd.concat([df1,df2],axis=1)
```

Out[173]:

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

In [174]:

```
# concate them along column
pd.concat([df1,df2],axis=0)
```

Out[174]:

	A	B	C	D
0	A0	B0	NaN	NaN
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	A3	B3	NaN	NaN
0	NaN	NaN	C0	D0
1	NaN	NaN	C1	D1
2	NaN	NaN	C2	D2
3	NaN	NaN	C3	D3

In [175]:

```
# we can know this not better approch to join the table
```

In [ ]:

```
# so to join the two table along the rows column name of both column must be matching
```

In [180]:

```
df2.columns=df1.columns
```

In [181]:

```
df2
```

Out[181]:

	C	D
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

In [182]:

```
df1
```

Out[182]:

	C	D
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

In [184]:

```
pd.concat([df1,df2],axis=0)
```

Out[184]:

	C	D
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3
0	C0	D0
1	C1	D1
2	C2	D2
3	C3	D3

27. Combining Dataframe --> Merging

- this is analogous to Join in sql
- the merge method take key argument labelled how
- there are three way of merging how= inner,outer , left or right

In [186]:

```
registrations = pd.DataFrame({'reg_id':[1,2,3,4],'name':['Andrew','Bobo','Claire','David'
logins = pd.DataFrame({'log_id':[1,2,3,4],'name':['Xavier','Andrew','Yolanda','Bobo']})
```

In [187]:

```
registrations
```

Out[187]:

	reg_id	name
0	1	Andrew
1	2	Bobo
2	3	Claire
3	4	David

In [188]:

```
logins
```

Out[188]:

	log_id	name
0	1	Xavier
1	2	Andrew
2	3	Yolanda
3	4	Bobo

In [189]:

```
#Lets first try with inner join
```

In [190]:

```
pd.merge(registrations,logins,how="inner",on="name")
```

Out[190]:

	reg_id	name	log_id
0	1	Andrew	2
1	2	Bobo	4

- left and right Merge

In [191]:

```
#Left
pd.merge(registrations,logins,how="left",on="name")
```

Out[191]:

	reg_id	name	log_id
0	1	Andrew	2.0
1	2	Bobo	4.0
2	3	Claire	NaN
3	4	David	NaN

In [192]:

```
#Right
pd.merge(registrations,logins,how="right",on="name")
```

Out[192]:

	reg_id	name	log_id
0	NaN	Xavier	1
1	1.0	Andrew	2
2	NaN	Yolanda	3
3	2.0	Bobo	4

In [193]:

```
# outer merge
pd.merge(registrations,logins,how="outer",on="name")
```

Out[193]:

	reg_id	name	log_id
0	1.0	Andrew	2.0
1	2.0	Bobo	4.0
2	3.0	Claire	NaN
3	4.0	David	NaN
4	NaN	Xavier	1.0
5	NaN	Yolanda	3.0

28. Text method on string data

- often text data needs to be cleaned or manipulated for processing
- while we can always use a custome apply(), function for these task ,pandas comes with built in string method calls

In [194]:

```
#split
email="milindgaur@gmail.com"
email.split("@")
```

Out[194]:

```
['milindgaur', 'gmail.com']
```

In [195]:

```
name="Milind"
name.isdigit()
```

Out[195]:

```
False
```

In [199]:

```
"7".isdigit()
```

Out[199]:

```
True
```

In [201]:

```
names=pd.Series(["Milind","Kanchan","Rohit",'Snehal'])
```

In [202]:

```
names
```

Out[202]:

```
0     Milind
1    Kanchan
2      Rohit
3     Snehal
dtype: object
```

In [203]:

```
names.str.upper()
```

Out[203]:

```
0     MILIND
1    KANCHAN
2      ROHIT
3     SNEHAL
dtype: object
```

In [204]:

```
names.str.capitalize()
```

Out[204]:

```
0     Milind
1    Kanchan
2      Rohit
3     Snehal
dtype: object
```

In [205]:

```
names.str.lower()
```

Out[205]:

```
0     milind
1    kanchan
2      rohit
3     snehal
dtype: object
```

29. How to clean the data

In [215]:

```
messy_names=pd.Series(["SaChin ","Kanchan","Rohit",'SNehal  '])
```

In [216]:

```
messy_names
```

Out[216]:

```
0      SaChin
1      Kanchan
2        Rohit
3     SNehal
dtype: object
```

In [217]:

```
messy_names.str.strip().str.capitalize()
```

Out[217]:

```
0     Sachin
1    Kanchan
2      Rohit
3     Snehal
dtype: object
```