In [1]:
```
import pandas as pd
```

In [69]:
```
import numpy as np
import statistics as s
```

pandas provides two data structures

1. series(1 dimesional array with indexes that can be named)
2. dataframes(2 dimensional array with rows and columns that can be named

A series can be create using pd.series() thee input can be a simple dictionary or list

In [11]:
```
s1=pd.Series([88,99,75,93,95,93,85])
print('series created using list as input:\n',s1)
s2=pd.Series({'a':91,'b':88,'c':95,'d':79,'e':94})
print('series using dictionary as input:\n',s2)
```

```
series created using list as input:
 0    88
1    99
2    75
3    93
4    95
5    93
6    85
dtype: int64
series using dictionary as input:
 a    91
b    88
c    95
d    79
e    94
dtype: int64
```

A dataframe can also be created uing simple dictionary with values as a series object(list,tuple) and key(string,charecter,number) keys are shown as column names and the values are the column values respectively or list of lists or list of tuples or tuple of lists or tuple of tuples in the above four cases rows and columns are indexed with number below two cases using dictionary and one of the four cases is shown

In [23]:
```
data1={'name':['rohith','maruthi','gangadhar','barghav','abhay','sudeb','tharun','abhiram'],
       'rollno':[12,3,13,16,63,57,35,33],
       'branch':['csd','csm','csm','csd','csd','csd','csm','csd']}
```

In [24]:
```
df=pd.DataFrame(data1)
df.head(4)
```

Out[24]:

	name	rollno	branch
0	rohith	12	csd
1	maruthi	3	csm
2	gangadhar	13	csm
3	barghav	16	csd

In [26]:
```
data2=(('rohith','maruthi','gangadhar','barghav','abhay','sudeb','tharun','abhiram'),
      (12,3,13,16,63,57,35,33),
      ('csd','csm','csm','csd','csd','csd','csm','csd'))
```

In [28]:
```
df1=pd.DataFrame(data2)
df1
```

Out[28]:

	0	1	2	3	4	5	6	7
0	rohith	maruthi	gangadhar	barghav	abhay	sudeb	tharun	abhiram
1	12	3	13	16	63	57	35	33
2	csd	csm	csm	csd	csd	csd	csm	csd

DATA HANDLING

In [29]:
```
print('names:\n',df['name'])
print('rollno:\n',df.rollno)
```

```
names:
 0       rohith
1      maruthi
2    gangadhar
3      barghav
4        abhay
5        sudeb
6       tharun
7      abhiram
Name: name, dtype: object
rollno:
 0    12
1     3
2    13
3    16
4    63
5    57
6    35
7    33
Name: rollno, dtype: int64
```

data(dataframe) can be created using 'xlsx' , 'csv' files by:
pd.read_excel('path of the excel file in your system')
pd.read_csv('path of the csv file in your system')

In [46]:
```
#current working directory is downloads and the excel workbook is also in downloads (relative path)
d1=pd.read_excel('Car Sales Dataset.xlsx')
d1.head()
```

Out[46]:

	name	brand	year	selling_price	km_driven	Number of Owners	seats	fuel	seller_type	transmission	engine	max_power	torque
0	Mahindra Bolero Pik-Up	Mahindra	2020	679000	5000	1	2	Diesel	Individual	Manual	2523 CC	70 bhp	200Nm@ 1400-2200rpm
1	Mahindra Bolero Pik-Up CBC 1.7T	Mahindra	2019	722000	80000	1	2	Diesel	Individual	Manual	2523 CC	70 bhp	200Nm@ 1400-2200rpm
2	Tata Nano Cx	Tata	2011	45000	10000	3	4	Petrol	Individual	Manual	624 CC	35 bhp	48Nm@ 3000rpm
3	Maruti 800 Std	Maruti	2002	40000	80000	3	4	Petrol	Individual	Manual	796 CC	37 bhp	59Nm@ 2500rpm
4	Tata Nano Cx BSIV	Tata	2010	55000	50000	3	4	Petrol	Individual	Manual	624 CC	35 bhp	48@ 3,000+/-500(NM@ rpm)

In [50]:
```
d2=pd.read_csv(r"C:\Users\rohit\Desktop\Data sets\Toy-Sales-dataset - Training.csv")
d2.head()
```

Out[50]:

	Month	Sales	PromExp	Price	AdExp
0	1	73959	61.13	8.75	50.04
1	2	71544	60.19	8.99	50.74
2	3	78587	59.16	7.50	50.14
3	4	80364	60.38	7.25	50.27
4	5	78771	59.71	7.40	51.25

FILTERING THE DATA

filtering whole data with branch either csd or csm

In [30]:
```
df[df['branch'] == 'csd']
```

Out[30]:

	name	rollno	branch
0	rohith	12	csd
3	barghav	16	csd
4	abhay	63	csd
5	sudeb	57	csd
7	abhiram	33	csd

In [31]:
```
df[df['branch'] == 'csm']
```

Out[31]:

	name	rollno	branch
1	maruthi	3	csm
2	gangadhar	13	csm
6	tharun	35	csm

filtering only names based on the branch

In [32]:
```
df.name[df['branch'] == 'csd']
```

Out[32]:
```
0     rohith
3    barghav
4      abhay
5      sudeb
7    abhiram
Name: name, dtype: object
```

MISSING DATA HANDLING.
pandas provide functions like
-> isna()
-> notna()
-> fillna()
-> dropna()
to handle with missing data
null values are represented as "np.nan" or "NaN" and it is a float data type

In [52]:
```
df['age']=[21,21,23,20,np.nan,21,np.nan,20]
```

In [57]:
```
df[df.age.isna()]
```

Out[57]:

	name	rollno	branch	age
4	abhay	63	csd	NaN
6	tharun	35	csm	NaN

In [58]:
```
df[df.age.notna()]
```

Out[58]:

	name	rollno	branch	age
0	rohith	12	csd	21.0
1	maruthi	3	csm	21.0
2	gangadhar	13	csm	23.0
3	barghav	16	csd	20.0
5	sudeb	57	csd	21.0
7	abhiram	33	csd	20.0

fillna has two parameter which decides the value to be filled

1. value ->you can fill it with a variable or output of an statistical measure(mean,median,mode)
2. method ->ffill,bfill
3. it has parameter 'limit' to limit no of values to be filled consecutively

In [75]:
```
df.age.fillna(s.mode(df.age))
#if u want to fill a specic value use ->fillna(value) i.e  fillna(21)
```

Out[75]:
```
0    21.0
1    21.0
2    23.0
3    20.0
4    21.0
5    21.0
6    21.0
7    20.0
Name: age, dtype: float64
```

In [80]:
```
df.fillna(method='ffill',limit=1)
```

```
C:\Users\rohit\AppData\Local\Temp\ipykernel_23908\393035141.py:1: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill',limit=1)
```

Out[80]:

	name	rollno	branch	age
0	rohith	12	csd	21.0
1	maruthi	3	csm	21.0
2	gangadhar	13	csm	23.0
3	barghav	16	csd	20.0
4	abhay	63	csd	20.0
5	sudeb	57	csd	21.0
6	tharun	35	csm	21.0
7	abhiram	33	csd	20.0

In [82]:
```
df.ffill(limit=1)
```

Out[82]:

	name	rollno	branch	age
0	rohith	12	csd	21.0
1	maruthi	3	csm	21.0
2	gangadhar	13	csm	23.0
3	barghav	16	csd	20.0
4	abhay	63	csd	20.0
5	sudeb	57	csd	21.0
6	tharun	35	csm	21.0
7	abhiram	33	csd	20.0

In [73]:
```
df.dropna()
```

Out[73]:

	name	rollno	branch	age
0	rohith	12	csd	21.0
1	maruthi	3	csm	21.0
2	gangadhar	13	csm	23.0
3	barghav	16	csd	20.0
5	sudeb	57	csd	21.0
7	abhiram	33	csd	20.0

->in the above examples only a single feature has null values
->all the four functions are applicable even if the data has null values in multiple columns
->you can filter the data by a particular columns null values
->fillna fills data based on that particular column values

MERGING,JOINING AND CONCATENATING DATAFRAMES

->merge and join are almost same in crating output/result
-megre takes separate key parameter
-join considers index as key parameter
these features are used to combine two dataframes with different features(columns)

In [118…
```
m1=pd.DataFrame({'key':['A','B','C'],'value':[10,20,30]})
m2=pd.DataFrame({'key':['B','C','D'],'value':[35,45,55]})
m3=pd.merge(m1,m2,on='key',how='inner')
m4=pd.merge(m1,m2,on='key',how='left')
m5=pd.merge(m1,m2,on='key',how='outer')
```

In [119…
```
m3
```

Out[119…

	key	value_x	value_y
0	B	20	35
1	C	30	45

In [120…
```
m4
```

Out[120…

	key	value_x	value_y
0	A	10	NaN
1	B	20	35.0
2	C	30	45.0

In [121…
```
m5
```

Out[121…

	key	value_x	value_y
0	A	10.0	NaN
1	B	20.0	35.0
2	C	30.0	45.0
3	D	NaN	55.0

->concatenating two dataframes

In [126…
```
df['scores']=[94,89,84,98,80,79,93,90]
```

In [127…
```
newdata=pd.DataFrame({'name':['teja','sathwik','madhu','lokesh'],
                      'rollno':[49,6,51,30],
                      'branch':['csm','csd','csm','csd'],
                      'age':[21,21,22,21],
                      'scores':[89,96,93,90]})
```

In [135…
```
df=pd.concat([df,newdata],ignore_index=True)
df
```

Out[135…

	name	rollno	branch	age	scores
0	rohith	12	csd	21.0	94
1	maruthi	3	csm	21.0	89
2	gangadhar	13	csm	23.0	84
3	barghav	16	csd	20.0	98
4	abhay	63	csd	NaN	80
5	sudeb	57	csd	21.0	79
6	tharun	35	csm	NaN	93
7	abhiram	33	csd	20.0	90
8	teja	49	csm	21.0	89
9	sathwik	6	csd	21.0	96
10	madhu	51	csm	22.0	93
11	lokesh	30	csd	21.0	90
12	teja	49	csm	21.0	89
13	sathwik	6	csd	21.0	96
14	madhu	51	csm	22.0	93
15	lokesh	30	csd	21.0	90

DATA ANALYSIS WITH PANDAS
->pivottable for data aggreagation
->groupby for data grouping
these two features are vastly used for eploratory data analysis(EDA)
consider car sales dataset for pivottable and df dataframe for groupby

In [88]:
```
d1.pivot_table(index='brand',columns='Number of Owners',values=['selling_price','km_driven'],aggfunc='mean').fillna(0)
```

Out[88]:

	km_driven					selling_price				
Number of Owners	0	1	2	3	4	0	1	2	3	4
brand										
Ambassador	0.0	0.000000	83333.333333	80000.000000	0.000000	0.0	0.000000e+00	9.866667e+04	2.000000e+05	0.000000
Ashok	0.0	0.000000	200000.000000	0.000000	0.000000	0.0	0.000000e+00	3.000000e+05	0.000000e+00	0.000000
Audi	14300.0	53109.357143	58271.428571	120000.000000	98000.000000	6223000.0	2.544786e+06	1.818571e+06	1.025000e+06	810000.000000
BMW	0.0	19110.247525	95680.000000	60000.000000	110000.000000	0.0	4.641733e+06	1.335000e+06	8.300000e+05	480000.000000
Chevrolet	0.0	72047.088496	82621.172840	92400.000000	109727.272727	0.0	2.885002e+05	2.754691e+05	2.135600e+05	248818.090909
Daewoo	0.0	81317.000000	0.000000	0.000000	0.000000	0.0	7.700000e+04	0.000000e+00	0.000000e+00	0.000000
Datsun	0.0	35478.701754	43571.428571	35000.000000	0.000000	0.0	3.102807e+05	3.575713e+05	2.600000e+05	0.000000
Fiat	0.0	75974.916667	86333.333333	110000.000000	0.000000	0.0	3.700000e+05	2.220832e+05	2.919998e+05	0.000000
Force	0.0	16500.000000	133639.500000	0.000000	0.000000	0.0	9.800000e+05	7.025000e+05	0.000000e+00	0.000000
Ford	0.0	63805.357143	88934.376147	90061.727273	72798.800000	0.0	6.125714e+05	3.618990e+05	3.035000e+05	321400.000000
Honda	24857.0	48728.764706	86674.300000	99400.000000	161000.000000	2000000.0	6.737147e+05	4.056683e+05	3.091470e+05	81000.000000
Hyundai	0.0	51645.801587	85182.809392	86419.910256	95567.567568	0.0	5.470793e+05	3.598204e+05	2.829807e+05	234864.810811
Isuzu	0.0	45560.000000	0.000000	0.000000	0.000000	0.0	1.942000e+06	0.000000e+00	0.000000e+00	0.000000
Jaguar	0.0	29578.571429	70000.000000	0.000000	0.000000	0.0	2.914257e+06	3.000000e+06	0.000000e+00	0.000000
Jeep	0.0	37704.100000	20000.000000	0.000000	0.000000	0.0	2.157267e+06	1.920000e+06	0.000000e+00	0.000000
Kia	0.0	10000.000000	0.000000	0.000000	0.000000	0.0	1.504500e+06	0.000000e+00	0.000000e+00	0.000000
Land	0.0	29757.600000	0.000000	77500.000000	0.000000	0.0	3.930000e+06	0.000000e+00	2.000000e+06	0.000000
Lexus	0.0	20000.000000	0.000000	0.000000	0.000000	0.0	5.150000e+06	0.000000e+00	0.000000e+00	0.000000
MG	0.0	12366.666667	0.000000	0.000000	0.000000	0.0	1.783333e+06	0.000000e+00	0.000000e+00	0.000000
Mahindra	0.0	81442.209446	97647.697115	138749.081633	139963.357143	0.0	7.116488e+05	4.989326e+05	4.490816e+05	371428.571429
Maruti	0.0	52256.656785	80818.193126	91023.966480	93222.868852	0.0	4.841598e+05	2.990833e+05	2.277031e+05	161901.622951
Mercedes-Benz	0.0	44310.162162	62664.642857	100000.000000	0.000000	0.0	2.899054e+06	1.552786e+06	1.466667e+06	0.000000
Mitsubishi	0.0	105081.833333	186000.000000	110000.000000	0.000000	0.0	1.258333e+06	5.958333e+05	1.600000e+05	0.000000
Nissan	0.0	65943.264151	82748.428571	90000.000000	91000.000000	0.0	5.123396e+05	4.009047e+05	3.008333e+05	320000.000000
Opel	0.0	0.000000	0.000000	110000.000000	0.000000	0.0	0.000000e+00	0.000000e+00	6.800000e+04	0.000000
Renault	0.0	50693.383333	90733.750000	88825.000000	0.000000	0.0	4.690166e+05	4.596000e+05	3.337499e+05	0.000000
Skoda	0.0	51234.970588	106035.035714	117500.000000	0.000000	0.0	7.378088e+05	4.028571e+05	2.450000e+05	0.000000
Tata	0.0	69604.760776	98427.124352	110089.142857	102084.615385	0.0	4.432909e+05	2.281826e+05	1.729219e+05	151807.615385
Toyota	0.0	82989.461538	118886.405405	171454.545455	181791.000000	0.0	1.183630e+06	8.003986e+05	5.418636e+05	430777.555556
Volkswagen	5400.0	60387.425926	88630.839286	91714.285714	89333.333333	1350000.0	5.641296e+05	3.915000e+05	3.407142e+05	276666.500000
Volvo	0.0	13287.878788	72500.000000	0.000000	0.000000	0.0	3.303409e+06	1.200000e+06	0.000000e+00	0.000000

In [136…
```
df.groupby(by=['branch','name']).size() #size represents the count
```

Out[136…
```
branch  name
csd     abhay        1
        abhiram      1
        barghav      1
        lokesh       2
        rohith       1
        sathwik      2
        sudeb        1
csm     gangadhar    1
        madhu        2
        maruthi      1
        teja         2
        tharun       1
dtype: int64
```

In [137…
```
df.groupby(by=['age','name']).size()
```

Out[137…
```
age   name
20.0  abhiram      1
      barghav      1
21.0  lokesh       2
      maruthi      1
      rohith       1
      sathwik      2
      sudeb        1
      teja         2
22.0  madhu        2
23.0  gangadhar    1
dtype: int64
```

In [138…
```
df.groupby(by=['branch'])['scores'].mean()
```

Out[138…
```
branch
csd    90.333333
csm    90.000000
Name: scores, dtype: float64
```

PANDAS (SERIES,DATAFRAME) VS REGULAR PYTHON DATA STRUCTURES

1.pandas numerical computation speed is faster than regular datastructures
2.no use of lopp control statements for iterating throung a series object
3.-pandas vast inbuit functions to peform larger tasks(agrregation)
-using regular data structures new code with logic must be developed to perform such tasks
4.large data is easy to use and handle using pandas

PANDAS APPLICATIONS IN DATA SCIENCE

pandas is built on top of the numpy package,pandaas provides even more vast features including the features of numpy
pandas provide features to handle categorical data unlike numpy
pandas provide these applications in data science
->handling missing values(step of data pre processing)
->for summary ststistics using pivot tables and grouping (EDA)
->data preparation for machine learning model fillting(PCA,kmeans,etc)