PANDAS:

- open source library
- has its own data structure known as dataframe.
- Dataframe stores data in tabular format in form of rows and columns.
- Data can be loaded from different file formats or SQL databases.
- Fast and efficient
- Data Alignment
- Can handle missing data here.
- Reshaping and pivoting of datasets.
- Label based indexing, slicing and subsetting.
- Mutable Data structure as allows add, update and delete functionality
- Group by functionality for data aggregation.
- High performance merging and joining of data.
- Direct data plotting.
- Functionality to handle time-series.
- Can work with mixed data such as timestamp,numerics etc.

In [2]:
```
#Working with PANDAS:
#import pandas
import pandas as pd

# Dataframe creation with lists:
print('''Dataframe creation with lists: Steps
1. convert list to dictionary
2. Add to dataframe
''')
dl1=[1,2,3,4]
dl2=['A','B','C','D']
dl3=[True,False,False,True]

print(f"\n dl1 is \n {dl1}\n\n dl2 is \n {dl2}\n\n  dl3 is \n {dl3}")
# Convert to dictionary:
pdict={"dl1":dl1,"dl2":dl2,"dl3":dl3}
print(f"\n Dictionary is \n{pdict}")

#Add to dataframe:
df1=pd.DataFrame(pdict)
print(f"\nDataframe type is: {type(df1)}")
df1
```

```
Dataframe creation with lists: Steps
1. convert list to dictionary
2. Add to dataframe


 dl1 is
 [1, 2, 3, 4]

 dl2 is
 ['A', 'B', 'C', 'D']

  dl3 is
 [True, False, False, True]

 Dictionary is
{'dl1': [1, 2, 3, 4], 'dl2': ['A', 'B', 'C', 'D'], 'dl3': [True, False, False, Tru
e]}
```

```
Dataframe type is: <class 'pandas.core.frame.DataFrame'>
```

Out[2]:

	dl1	dl2	dl3
0	1	A	True
1	2	B	False
2	3	C	False
3	4	D	True

In [3]:
```
# Reading CSV or excel
#r is for raw string, skiprows for skipping rows while loading and header=None to sp
df1=pd.read_csv(r'C:\Users\sneha\Downloads\carscsv.csv',skiprows=1,header=None)
df2=pd.read_excel(r'C:\Users\sneha\Downloads\cars.xlsx',skiprows=1)
df1
```

Out[3]:

	0	1	2	3	4
0	CarName	CarModel	CarOwner	Price	Date bought on
1	A	Cm1	O1	2344	01-01-2021
2	B	cm2	O2	34221	02-01-2021
3	C	cm3	O3	36456	03-01-2021
4	A	cm4	O4	34232	04-01-2021
5	A	cm5	O5	5657578	05-01-2021
6	A	Cm1	O1	24232	01-01-2021
7	A	cm2	O2	432533	05-02-2021
8	B	cm5	O5	34564	06-02-2021
9	B	NaN	NaN	NaN	NaN

In [4]:
```
# To get column info:
df2.columns
```

Out[4]:
```
Index(['CarName', 'CarModel', 'CarOwner', 'Price', 'Date bought on'], dtype='objec
t')
```

In [5]:
```
# Rename headers: number of columns should match
df2.columns=['CarNameReanmed','CarModelRenamed','CarOwner','Price', 'Date bought on'
df2.columns
```

Out[5]:
```
Index(['CarNameReanmed', 'CarModelRenamed', 'CarOwner', 'Price',
       'Date bought on'],
      dtype='object')
```

In [6]:
```
#To get index: gives range of index
df2.index
```

Out[6]:
```
RangeIndex(start=0, stop=9, step=1)
```

In [7]:
```
# If want to define index while loading the file:
df3=pd.read_csv(r'C:\Users\sneha\Downloads\carscsv.csv',skiprows=1,index_col=0)
df3.index
```

Out[7]:
```
Index(['A', 'B', 'C', 'A', 'A', 'A', 'A', 'B', 'B'], dtype='object', name='CarName')
```

In [8]:
```
#Renaming the index name from 'CarName' to 'CarnameIndex'
df3.index.name='CarnameIndex'
df3.index
```

Out[8]:
```
Index(['A', 'B', 'C', 'A', 'A', 'A', 'A', 'B', 'B'], dtype='object', name='CarnameIn
dex')
```

In [9]:
```
#Delete index name
df3.index.name=None
df3
```

Out[9]:

	CarModel	CarOwner	Price	Date bought on
A	Cm1	O1	2344.0	01-01-2021
B	cm2	O2	34221.0	02-01-2021
C	cm3	O3	36456.0	03-01-2021
A	cm4	O4	34232.0	04-01-2021
A	cm5	O5	5657578.0	05-01-2021
A	Cm1	O1	24232.0	01-01-2021
A	cm2	O2	432533.0	05-02-2021
B	cm5	O5	34564.0	06-02-2021
B	NaN	NaN	NaN	NaN

In [50]:
```
import numpy as np
dfadd1=df3.copy()
dfadd1.reset_index(inplace=True)
dfadd1
d11={'index':["B"]}
dfadd2=pd.DataFrame(d11)
dfadd2
dfadd3=pd.concat([dfadd1,dfadd2],axis=0)
dfadd3
```

Out[50]:

	index	CarModel	CarOwner	Price	Date bought on
0	A	Cm1	O1	2344.0	01-01-2021
1	B	cm2	O2	34221.0	02-01-2021
2	C	cm3	O3	36456.0	03-01-2021
3	A	cm4	O4	34232.0	04-01-2021
4	A	cm5	O5	5657578.0	05-01-2021
5	A	Cm1	O1	24232.0	01-01-2021
6	A	cm2	O2	432533.0	05-02-2021
7	B	cm5	O5	34564.0	06-02-2021
8	B	NaN	NaN	NaN	NaN
0	B	NaN	NaN	NaN	NaN

In [55]:
```
(dfadd3.isnull().sum()>0).sum()
```

Out[55]:
```
4
```

In [9]:
```
#Set hierarchical index:
df2.set_index(['CarNameReanmed','CarModelRenamed'],inplace=True)
```

In [10]:
```
df2
```

Out[10]:

CarNameReanmed	CarModelRenamed	CarOwner	Price	Date bought on
A	**Cm1**	O1	2344.0	2021-01-01
B	**cm2**	O2	34221.0	2021-01-02
C	**cm3**	O3	36456.0	2021-01-03
A	**cm4**	O4	34232.0	2021-01-04
	cm5	O5	5657578.0	2021-01-05
	Cm1	O1	24232.0	2021-01-01
	cm2	O2	432533.0	2021-02-05
B	**cm5**	O5	34564.0	2021-02-06
	NaN	NaN	NaN	NaT

In [11]:
```
# write dataframe to csv:it creates automatically
df2.to_csv(r'C:\Users\sneha\Downloads\carscsv1.csv')
```

In [12]:
```
# to check first 3 rows:
df3.head(3)
```

Out[12]:

	CarModel	CarOwner	Price	Date bought on
A	Cm1	O1	2344.0	01-01-2021
B	cm2	O2	34221.0	02-01-2021
C	cm3	O3	36456.0	03-01-2021

In [13]:
```
# to check last 3 rows:
df3.tail(3)
```

Out[13]:

	CarModel	CarOwner	Price	Date bought on
A	cm2	O2	432533.0	05-02-2021
B	cm5	O5	34564.0	06-02-2021
B	NaN	NaN	NaN	NaN

In [14]:
```
# Display column info:
df3.info()#use verbose=true parameter when more number of columns
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9 entries, A to B
Data columns (total 4 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   CarModel        8 non-null      object 
 1   CarOwner        8 non-null      object 
 2   Price           8 non-null      float64
 3   Date bought on  8 non-null      object 
```

```
dtypes: float64(1), object(3)
memory usage: 360.0+ bytes
```

In [15]:
```
# Get statistical info:
#importing numpy to use around function.
import numpy as np
np.around(df3.describe())
```

Out[15]:

	Price
count	8.0
mean	782020.0
std	1975142.0
min	2344.0
25%	31724.0
50%	34398.0
75%	135475.0
max	5657578.0

Connecting to sql to read data from there:

In [16]:
```
import sqlalchemy
```

In [17]:
```
sqlalchemy.create_engine('mysql://root:root@localhost/market_star_schema')
```

Out[17]:
```
Engine(mysql://root:***@localhost/market_star_schema)
```

In [18]:
```
%load_ext sql
```

In [19]:
```
%sql mysql://root:root@localhost/market_star_schema
```

In [20]:
```
#storing in variable
res = %sql select * from market_fact_full
```

```
 * mysql://root:***@localhost/market_star_schema
8399 rows affected.
```

Back to Pandas

In [21]:
```
#converting to dataframe:
dfsql=res.DataFrame()
dfsql.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8399 entries, 0 to 8398
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Market_fact_id       8399 non-null   int64 
 1   Ord_id               8399 non-null   object
 2   Prod_id              8399 non-null   object
 3   Ship_id              8399 non-null   object
 4   Cust_id              8399 non-null   object
 5   Sales                8399 non-null   object
 6   Discount             8399 non-null   object
 7   Order_Quantity       8399 non-null   int64 
```

```
 8   Profit               8399 non-null   object
 9   Shipping_Cost        8399 non-null   object
 10  Product_Base_Margin  8336 non-null   object
dtypes: int64(2), object(9)
memory usage: 721.9+ KB
```

In [22]:
```
print(f"Another datatype of pandas: \n{type(dfsql['Ord_id'])}")
```

```
Another datatype of pandas:
<class 'pandas.core.series.Series'>
```

Indexing, Slicing and Subsetting:

In [23]:
```
# One column indexing:
dfsql['Ord_id']
```

Out[23]:
```
0       Ord_5446
1       Ord_5406
2       Ord_5446
3       Ord_5456
4       Ord_5485
          ...
8394    Ord_5353
8395    Ord_5411
8396    Ord_5388
8397    Ord_5348
8398    Ord_5459
Name: Ord_id, Length: 8399, dtype: object
```

In [24]:
```
# Multiple column indexing:
dfsql[['Ord_id','Order_Quantity']]
```

Out[24]:

	Ord_id	Order_Quantity
0	Ord_5446	23
1	Ord_5406	13
2	Ord_5446	26
3	Ord_5456	43
4	Ord_5485	35
...	...	...
8394	Ord_5353	28
8395	Ord_5411	20
8396	Ord_5388	39
8397	Ord_5348	23
8398	Ord_5459	47

8399 rows × 2 columns

In [25]:
```
# Row indexing:
# When want to see data for particular index values only - use loc accessor
df3.loc[['A','C']]['CarModel']
```

Out[25]:
```
A    Cm1
A    cm4
A    cm5
A    Cm1
A    cm2
```

```
C    cm3
Name: CarModel, dtype: object
```

In [26]:
```
#Accessing via row number/ column number:
#use iloc accessor
df3.iloc[3,3] #row,col value gives that particular element at that position
```

Out[26]:
```
'04-01-2021'
```

In [27]:
```
#Slicing: - remember .loc or .iloc are not functions
df3.loc[:,['CarModel','CarOwner']] # want all rows and specific columns
df3.loc[['A','C'],:] # want all columns and specific rows
```

Out[27]:

	CarModel	CarOwner	Price	Date bought on
A	Cm1	O1	2344.0	01-01-2021
A	cm4	O4	34232.0	04-01-2021
A	cm5	O5	5657578.0	05-01-2021
A	Cm1	O1	24232.0	01-01-2021
A	cm2	O2	432533.0	05-02-2021
C	cm3	O3	36456.0	03-01-2021

In [28]:
```
#slicing -cont. - when want specif rows and specific columns
df3.loc[['A','C'],['CarModel','CarOwner']]
```

Out[28]:

	CarModel	CarOwner
A	Cm1	O1
A	cm4	O4
A	cm5	O5
A	Cm1	O1
A	cm2	O2
C	cm3	O3

In [29]:
```
##slicing -cont.- using indexes
df3.iloc[:,[0,1,2]] # want all rows and specific columns
df3.iloc[0:2,:] # want all columns and specific rows
df3.iloc[0:2,[1,2]] #when want specif rows and specific columns
```

Out[29]:

	CarOwner	Price
A	O1	2344.0
B	O2	34221.0

In [30]:
```
#Subsetting: basically filtering out based on certain condition
# ~ -> used for complement
dfsql[(dfsql['Cust_id'].isin (['Cust_1818'])) & (~dfsql['Ord_id'].isnull())]
```

Out[30]:

	Market_fact_id	Ord_id	Prod_id	Ship_id	Cust_id	Sales	D
0	1	Ord_5446	Prod_16	SHP_7609	Cust_1818	136.810000000000000000000000000000	
1	2	Ord_5406	Prod_13	SHP_7549	Cust_1818	42.270000000000000000000000000000	

	Market_fact_id	Ord_id	Prod_id	Ship_id	Cust_id	Sales	D
2	3	Ord_5446	Prod_4	SHP_7610	Cust_1818	4701.690000000000000000000000000000	
3	4	Ord_5456	Prod_6	SHP_7625	Cust_1818	2337.890000000000000000000000000000	
4	5	Ord_5485	Prod_17	SHP_7664	Cust_1818	4233.150000000000000000000000000000	
5	6	Ord_5446	Prod_6	SHP_7608	Cust_1818	164.020000000000000000000000000000	

Operations on Dataframe:

In [31]:
```
# creating new columns:
# Int64 converts astype to integer with null values.
df3['Year']=(pd.DatetimeIndex(df3['Date bought on']).year).astype('Int64') # Working
df3['Year']
```

Out[31]:
```
A    2021
B    2021
C    2021
A    2021
A    2021
A    2021
A    2021
B    2021
B    <NA>
Name: Year, dtype: object
```

In [32]:
```
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9 entries, A to B
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   CarModel        8 non-null      object 
 1   CarOwner        8 non-null      object 
 2   Price           8 non-null      float64
 3   Date bought on  8 non-null      object 
 4   Year            8 non-null      object 
dtypes: float64(1), object(4)
memory usage: 752.0+ bytes
```

In [33]:
```
# creating new columns: using lambda function -> useful incase of complex scenarios
# Use 'and' instead of '&' when using lambda functions:
# axis required when more than 1 column used in logic creation
# when carmodel=Cm1 and owner = O1 then the value should be billgates else others
lm1=lambda x:"billgates" if x['CarModel']=="Cm1" and x['CarOwner']=="O1" else "Other
df3['NameLambda']=df3.apply(lm1, axis=1)
df3.head()
```

Out[33]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda
A	Cm1	O1	2344.0	01-01-2021	2021	billgates
B	cm2	O2	34221.0	02-01-2021	2021	Others
C	cm3	O3	36456.0	03-01-2021	2021	Others
A	cm4	O4	34232.0	04-01-2021	2021	Others
A	cm5	O5	5657578.0	05-01-2021	2021	Others

In [34]:
```
# creating new columns: using functions(): -> specifying axis is important
# Use 'and' instead of '&' when using functions:
def func1(x):
    if (x['CarModel']=="Cm1" and x['CarOwner']=="O1"):
        return "Billgates"
    else:
        return "Others"
df3['NameFunc']=df3.apply(func1,axis=1)
df3.head()
```

Out[34]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates
B	cm2	O2	34221.0	02-01-2021	2021	Others	Others
C	cm3	O3	36456.0	03-01-2021	2021	Others	Others
A	cm4	O4	34232.0	04-01-2021	2021	Others	Others
A	cm5	O5	5657578.0	05-01-2021	2021	Others	Others

Group By, aggregate Functions and order functions

In [35]:
```
#Groupby:single column grouping
dfsql.groupby(by=['Cust_id']).mean()
dfsql.groupby(by=['Cust_id']).count()
# instead of groupby for count we can use value_counts()
dfsql[['Cust_id','Ord_id']].value_counts(normalize=True)
```

Out[35]:
```
Cust_id    Ord_id
Cust_188   Ord_542     0.000714
Cust_1376  Ord_4025    0.000595
Cust_466   Ord_1234    0.000595
Cust_812   Ord_2164    0.000595
Cust_1682  Ord_4946    0.000595
                          ...
Cust_266   Ord_809     0.000119
           Ord_817     0.000119
Cust_267   Ord_814     0.000119
Cust_268   Ord_824     0.000119
Cust_1     Ord_1       0.000119
Length: 5820, dtype: float64
```

In [36]:
```
#Groupby:multiple column grouping -> no [[]], only [] for mentioning the columns
dfsql.groupby(by=['Cust_id','Ord_id']).mean()
```

Out[36]:

		Market_fact_id	Order_Quantity
Cust_id	**Ord_id**		
Cust_1	**Ord_1**	5859.000000	6.000000
Cust_10	**Ord_10**	1771.000000	15.000000
Cust_100	**Ord_163**	1504.000000	1.000000
	Ord_219	1505.000000	31.000000
	Ord_227	1506.000000	47.000000
...	**...**	...	...
Cust_999	**Ord_2700**	7048.000000	24.000000
	Ord_2742	7057.000000	14.000000

		Market_fact_id	Order_Quantity
Cust_id	**Ord_id**		
	Ord_2784	7040.000000	33.000000
	Ord_2810	7045.333333	38.666667
	Ord_2827	7049.000000	19.000000

5820 rows × 2 columns

In [37]:
```
#Groupby:multiple column grouping and mentioning the column to get the aggregate val
dfsql.groupby(by=['Cust_id','Ord_id'])[['Market_fact_id','Order_Quantity']].mean()
```

Out[37]:

		Market_fact_id	Order_Quantity
Cust_id	**Ord_id**		
Cust_1	**Ord_1**	5859.000000	6.000000
Cust_10	**Ord_10**	1771.000000	15.000000
Cust_100	**Ord_163**	1504.000000	1.000000
	Ord_219	1505.000000	31.000000
	Ord_227	1506.000000	47.000000
...	**...**	...	...
Cust_999	**Ord_2700**	7048.000000	24.000000
	Ord_2742	7057.000000	14.000000
	Ord_2784	7040.000000	33.000000
	Ord_2810	7045.333333	38.666667
	Ord_2827	7049.000000	19.000000

5820 rows × 2 columns

In [38]:
```
#Groupby:multiple column grouping and mentioning the column to get the aggregate val
#value provided in dictionary
dfsql.groupby(by=['Cust_id','Ord_id'])[['Market_fact_id','Order_Quantity']].agg({'Ma
```

Out[38]:

		Market_fact_id	Order_Quantity
Cust_id	**Ord_id**		
Cust_1	**Ord_1**	5859.000000	6.0
Cust_10	**Ord_10**	1771.000000	15.0
Cust_100	**Ord_163**	1504.000000	1.0
	Ord_219	1505.000000	31.0
	Ord_227	1506.000000	47.0
...	**...**	...	...
Cust_999	**Ord_2700**	7048.000000	24.0
	Ord_2742	7057.000000	14.0
	Ord_2784	7040.000000	33.0

Cust_id	Ord_id	Market_fact_id	Order_Quantity
	Ord_2810	7045.333333	37.0
	Ord_2827	7049.000000	19.0

5820 rows × 2 columns

In [39]:
```
#SortBy:
dfsql.sort_values(['Order_Quantity','Market_fact_id'],ascending=False)
```

Out[39]:

	Market_fact_id	Ord_id	Prod_id	Ship_id	Cust_id	Sale
8281	8282	Ord_1770	Prod_5	SHP_2454	Cust_595	387.030000000000000000000000000000(
8207	8208	Ord_3433	Prod_8	SHP_4759	Cust_1180	3601.070000000000000000000000000000(
8176	8177	Ord_5211	Prod_13	SHP_7282	Cust_1755	264.050000000000000000000000000000(
8122	8123	Ord_4813	Prod_4	SHP_6707	Cust_1655	4872.667500000000000000000000000000(
8091	8092	Ord_2422	Prod_5	SHP_3323	Cust_966	230.630000000000000000000000000000(
...	...	...	...	...	...	..
290	291	Ord_297	Prod_17	SHP_399	Cust_53	171.710000000000000000000000000000(
58	59	Ord_4499	Prod_6	SHP_6264	Cust_1474	67.490000000000000000000000000000(
52	53	Ord_4755	Prod_13	SHP_6628	Cust_1579	3.410000000000000000000000000000(
45	46	Ord_4768	Prod_12	SHP_6650	Cust_1579	3.850000000000000000000000000000(
43	44	Ord_1189	Prod_6	SHP_1641	Cust_452	49.610000000000000000000000000000(

8399 rows × 11 columns

In [40]:
```
#Sortby: combined with groupby
dfsql.groupby(by=['Cust_id','Ord_id'])[['Market_fact_id','Order_Quantity']].mean().s
```

Out[40]:

Cust_id	Ord_id	Market_fact_id	Order_Quantity
Cust_1618	**Ord_4730**	207.0	50.0
Cust_1516	**Ord_4428**	1957.0	50.0
Cust_1618	**Ord_4784**	205.0	50.0
Cust_1137	**Ord_3085**	3568.0	50.0
Cust_585	**Ord_1766**	5656.0	50.0
...	**...**	...	...
Cust_471	**Ord_1274**	4160.0	1.0
Cust_990	**Ord_2618**	8133.0	1.0
Cust_1661	**Ord_4801**	458.0	1.0
Cust_576	**Ord_2942**	5538.0	1.0

		Market_fact_id	Order_Quantity
Cust_id	**Ord_id**		
Cust_67	**Ord_248**	865.0	1.0

5820 rows × 2 columns

Merging Dataframes:

- inner join
- left outer join
- right outer join
- full outer join
- Various ways - merge(), join(), concat(), append()
- Merge and join are usually used when both dataframes have several different columns against a common index.

In [41]:
```
#Join()
print('''
- Join uses index for joining two dataframes,
even if you provide a column it will still join on the basis of index.
- Index is preserved.
- suffix is specified in order to differentiate in case of columns of same name.
\n''')
df3
df4=df3.iloc[0:4,:]
df4['cnt']=1
df5=df3.iloc[4:8,:]
df5
```

```
- Join uses index for joining two dataframes,
even if you provide a column it will still join on the basis of index.
- Index is preserved.
- suffix is specified in order to differentiate in case of columns of same name.
```

```
<ipython-input-41-3c5fdf96d7b1>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df4['cnt']=1
```

Out[41]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	cm5	O5	5657578.0	05-01-2021	2021	Others	Others
A	Cm1	O1	24232.0	01-01-2021	2021	billgates	Billgates
A	cm2	O2	432533.0	05-02-2021	2021	Others	Others
B	cm5	O5	34564.0	06-02-2021	2021	Others	Others

In [42]:
```
print("\n Joined dataframe -> note 'cnt' has no suffix")
df4.join(df5, on='CarModel', lsuffix='_',how="outer")
```

```
 Joined dataframe -> note 'cnt' has no suffix
```

Out[42]:

	CarModel_	CarOwner_	Price_	Date bought on_	Year_	NameLambda_	NameFunc_	cnt	CarModel
A	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates	1.0	NaN
B	cm2	O2	34221.0	02-01-2021	2021	Others	Others	1.0	NaN
C	cm3	O3	36456.0	03-01-2021	2021	Others	Others	1.0	NaN
A	cm4	O4	34232.0	04-01-2021	2021	Others	Others	1.0	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	B

In [43]:
```
#Merge():
print('''
- Merge is almost same as join and serves the same purpose as join
- with additional advantage of adding columns as keys.
- Index is not preserved.
- Therefore number of rows are more compared to merge.
\n''')
df4.merge(df5,on="CarModel",how="outer")
```

```
- Merge is almost same as join and serves the same purpose as join
- with additional advantage of adding columns as keys.
- Index is not preserved.
- Therefore number of rows are more compared to merge.
```

Out[43]:

	CarModel	CarOwner_x	Price_x	Date bought on_x	Year_x	NameLambda_x	NameFunc_x	cnt	CarOwner_
0	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates	1.0	O
1	cm2	O2	34221.0	02-01-2021	2021	Others	Others	1.0	O
2	cm3	O3	36456.0	03-01-2021	2021	Others	Others	1.0	Na
3	cm4	O4	34232.0	04-01-2021	2021	Others	Others	1.0	Na
4	cm5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	O
5	cm5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	O

In [44]:
```
#Concat():
pd.concat([df4,df5]) # row wise merge
pd.concat([df4.reset_index(),df5.reset_index()],axis=1) # index reset because earlie
```

Out[44]:

	index	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc	cnt	index	CarM
0	A	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates	1	A	
1	B	cm2	O2	34221.0	02-01-2021	2021	Others	Others	1	A	
2	C	cm3	O3	36456.0	03-01-2021	2021	Others	Others	1	A	
3	A	cm4	O4	34232.0	04-01-2021	2021	Others	Others	1	B	

In [45]:
```
#Concat():
pd.concat([df4,df5],join="inner") # row wise merge with join type specified -> note
```

Out[45]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates
B	cm2	O2	34221.0	02-01-2021	2021	Others	Others
C	cm3	O3	36456.0	03-01-2021	2021	Others	Others
A	cm4	O4	34232.0	04-01-2021	2021	Others	Others
A	cm5	O5	5657578.0	05-01-2021	2021	Others	Others
A	Cm1	O1	24232.0	01-01-2021	2021	billgates	Billgates
A	cm2	O2	432533.0	05-02-2021	2021	Others	Others
B	cm5	O5	34564.0	06-02-2021	2021	Others	Others

In [46]:
```
#Concat():# to get continous index 0-7, earlier was car names
pd.concat([df4,df5], ignore_index=True)
```

Out[46]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc	cnt
0	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates	1.0
1	cm2	O2	34221.0	02-01-2021	2021	Others	Others	1.0
2	cm3	O3	36456.0	03-01-2021	2021	Others	Others	1.0
3	cm4	O4	34232.0	04-01-2021	2021	Others	Others	1.0
4	cm5	O5	5657578.0	05-01-2021	2021	Others	Others	NaN
5	Cm1	O1	24232.0	01-01-2021	2021	billgates	Billgates	NaN
6	cm2	O2	432533.0	05-02-2021	2021	Others	Others	NaN
7	cm5	O5	34564.0	06-02-2021	2021	Others	Others	NaN

In [47]:
```
#concat(): Assign keys to each dataframe -> to be used without ignore_index=True
```

```
print('''
can be accessed using dfname.loc['x'] or dfname.loc['y']
''')

pd.concat([df4,df5],keys=['x', 'y'])
```

```
can be accessed using dfname.loc['x'] or dfname.loc['y']
```

Out[47]:

		CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc	cnt
x	**A**	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates	1.0
	B	cm2	O2	34221.0	02-01-2021	2021	Others	Others	1.0
	C	cm3	O3	36456.0	03-01-2021	2021	Others	Others	1.0
	A	cm4	O4	34232.0	04-01-2021	2021	Others	Others	1.0
y	**A**	cm5	O5	5657578.0	05-01-2021	2021	Others	Others	NaN
	A	Cm1	O1	24232.0	01-01-2021	2021	billgates	Billgates	NaN
	A	cm2	O2	432533.0	05-02-2021	2021	Others	Others	NaN
	B	cm5	O5	34564.0	06-02-2021	2021	Others	Others	NaN

Pivot Tables:

- used in case of multivariate analysis

In [48]:

```
#Pivot table creation:
print('''
index and column can take multiple values.
''')
dfsql.head(10).pivot_table(index=['Cust_id','Ord_id'],columns=['Prod_id','Discount']
```

```
index and column can take multiple values.
```

Out[48]:

		Prod_id	Prod_12	Prod_13	Prod_16	Prod_17	Prod_4			Prod_6	
		Discount	**0.01**	**0.01**	**0.01**	**0.08**	**0.00**	**0.10**	**0.03**	**0.07**	**0.09**
Cust_id	**Ord_id**										
Cust_1641	**Ord_4725**		NaN	33.0	NaN	NaN	NaN	48.0	NaN	8.0	NaN
Cust_1818	**Ord_5406**		NaN	13.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	Ord_5446		NaN	NaN	23.0	NaN	26.0	NaN	23.0	NaN	NaN
	Ord_5456		NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	43.0
	Ord_5485		NaN	NaN	NaN	35.0	NaN	NaN	NaN	NaN	NaN
Cust_26	**Ord_31**		5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Pandas column Types:

- Categorical:
  1. Ordered - which has a order ex. Ratings(good, bad, poor)
  2. Nominal - No order ex. cities, gender
- Continous - ex. speed, weight

Miscellaneous:

In [71]:
```
print("Get unique values \n")
dfsql['Cust_id'].unique()
```

```
Get unique values

array(['Cust_1818', 'Cust_26', 'Cust_1641', ..., 'Cust_851', 'Cust_1519',
       'Cust_1798'], dtype=object)
```

In [72]:
```
print("Get count of unique values \n")
dfsql['Cust_id'].nunique()
```

```
Get count of unique values

1832
```

In [50]:
```
df3
```

Out[50]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	01-01-2021	2021	billgates	Billgates
B	cm2	O2	34221.0	02-01-2021	2021	Others	Others
C	cm3	O3	36456.0	03-01-2021	2021	Others	Others
A	cm4	O4	34232.0	04-01-2021	2021	Others	Others
A	cm5	O5	5657578.0	05-01-2021	2021	Others	Others
A	Cm1	O1	24232.0	01-01-2021	2021	billgates	Billgates
A	cm2	O2	432533.0	05-02-2021	2021	Others	Others
B	cm5	O5	34564.0	06-02-2021	2021	Others	Others
B	NaN	NaN	NaN	NaN	<NA>	Others	Others

In [54]:
```
print("Change datatype to datetime \n")
print('''
when pandas not able to decode the date format then declare format explicitly
https://strftime.org/
pd.to_datetime(df3['Date bought on'],format='%d-%m-%y')
''')
df3['Date bought on']=pd.to_datetime(df3['Date bought on'])
```

```
Change datatype to datetime


when pandas not able to decode the date format then declare format explicitly
https://strftime.org/
pd.to_datetime(df3['Date bought on'],format='%d-%m-%y')
```

In [58]:
```
#Extract date,month,year: -> when dtype id datetime
print("can use date,month,year")
df3['Date bought on'].dt.year
```

```
uses date,month,year
```

Out[58]:
```
A    2021.0
B    2021.0
C    2021.0
A    2021.0
A    2021.0
A    2021.0
```

```
A    2021.0
B    2021.0
B       NaN
Name: Date bought on, dtype: float64
```

In [59]:
```
df3
```

Out[59]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	2021-01-01	2021	billgates	Billgates
B	cm2	O2	34221.0	2021-02-01	2021	Others	Others
C	cm3	O3	36456.0	2021-03-01	2021	Others	Others
A	cm4	O4	34232.0	2021-04-01	2021	Others	Others
A	cm5	O5	5657578.0	2021-05-01	2021	Others	Others
A	Cm1	O1	24232.0	2021-01-01	2021	billgates	Billgates
A	cm2	O2	432533.0	2021-05-02	2021	Others	Others
B	cm5	O5	34564.0	2021-06-02	2021	Others	Others
B	NaN	NaN	NaN	NaT	<NA>	Others	Others

In [60]:
```
#Drop columns/rows: -> has inplace parameter
df3.drop(['A'],axis=0) # for rows
```

Out[60]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
B	cm2	O2	34221.0	2021-02-01	2021	Others	Others
C	cm3	O3	36456.0	2021-03-01	2021	Others	Others
B	cm5	O5	34564.0	2021-06-02	2021	Others	Others
B	NaN	NaN	NaN	NaT	<NA>	Others	Others

In [61]:
```
#Drop columns/rows: -> has inplace parameter
df3.drop(['CarModel'],axis=1) # for columns
```

Out[61]:

	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	O1	2344.0	2021-01-01	2021	billgates	Billgates
B	O2	34221.0	2021-02-01	2021	Others	Others
C	O3	36456.0	2021-03-01	2021	Others	Others
A	O4	34232.0	2021-04-01	2021	Others	Others
A	O5	5657578.0	2021-05-01	2021	Others	Others
A	O1	24232.0	2021-01-01	2021	billgates	Billgates
A	O2	432533.0	2021-05-02	2021	Others	Others
B	O5	34564.0	2021-06-02	2021	Others	Others
B	NaN	NaN	NaT	<NA>	Others	Others

In [64]:
```
# Drop null records:
df3.dropna(axis=0) # drop entire row if any value is null -> by default axis=0
```

Out[64]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	2021-01-01	2021	billgates	Billgates
B	cm2	O2	34221.0	2021-02-01	2021	Others	Others
C	cm3	O3	36456.0	2021-03-01	2021	Others	Others
A	cm4	O4	34232.0	2021-04-01	2021	Others	Others
A	cm5	O5	5657578.0	2021-05-01	2021	Others	Others
A	Cm1	O1	24232.0	2021-01-01	2021	billgates	Billgates
A	cm2	O2	432533.0	2021-05-02	2021	Others	Others
B	cm5	O5	34564.0	2021-06-02	2021	Others	Others

In [65]:
```
# Drop null records:
df3.dropna(axis=1) # drop entire column if any value is null -> by default axis=0
```

Out[65]:

	NameLambda	NameFunc
A	billgates	Billgates
B	Others	Others
C	Others	Others
A	Others	Others
A	Others	Others
A	billgates	Billgates
A	Others	Others
B	Others	Others
B	Others	Others

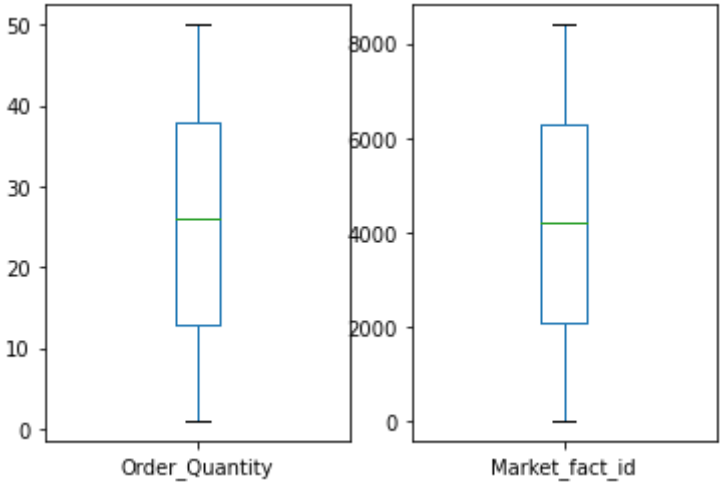
In [68]:
```
# Drop null records:
df3.dropna(axis=0,how='all') # drop entire column if all values are null -> by defau
```

Out[68]:

	CarModel	CarOwner	Price	Date bought on	Year	NameLambda	NameFunc
A	Cm1	O1	2344.0	2021-01-01	2021	billgates	Billgates
B	cm2	O2	34221.0	2021-02-01	2021	Others	Others
C	cm3	O3	36456.0	2021-03-01	2021	Others	Others
A	cm4	O4	34232.0	2021-04-01	2021	Others	Others
A	cm5	O5	5657578.0	2021-05-01	2021	Others	Others
A	Cm1	O1	24232.0	2021-01-01	2021	billgates	Billgates
A	cm2	O2	432533.0	2021-05-02	2021	Others	Others
B	cm5	O5	34564.0	2021-06-02	2021	Others	Others
B	NaN	NaN	NaN	NaT	<NA>	Others	Others

In [69]:
```
#Plotting data from columns:
dfsql[['Order_Quantity','Market_fact_id']].plot(kind='box',subplots=True)
```

Out[69]:
```
Order_Quantity      AxesSubplot(0.125,0.125;0.352273x0.755)
Market_fact_id    AxesSubplot(0.547727,0.125;0.352273x0.755)
dtype: object
```



In [ ]:

In [ ]:

In [ ]: