In [48]:
```
import pandas as pd
import numpy as np
import os
```

In [ ]:
```

```

In [3]:
```
# import
os.getcwd()
os.chdir("C:\\Users\\admin\\pandas\\DataSets")

stores = pd.read_csv("stores.csv")

# file name` or file path

# Excel sheet - sheetname or sheetindex

# rows =

```

In [49]:
```
myxlfile = pd.ExcelFile('stores.xlsx')

sh = myxlfile.sheet_names  # see all sheet names
sh

storesxls = pd.read_excel(myxlfile,'stores') # or pd.read_excel(myxlfile, 2)
```

Out[49]: [u'Sheet3', u'Sheet4', u'stores', u'Sheet1', u'Sheet2']

In [50]:
```
type(stores)
```

Out[50]: pandas.core.frame.DataFrame

In [5]:
```
# Indexes
# 1. Every row has a default number - sno

# 2. A list of names can be assigned to the rows

# 3. A column can be promoted as an index
```

In [6]:
```
stores.head(5)
stores.tail(10)
```

Out[6]:

	StoreCode	StoreName	StoreType	Location	OperatingCost	Staff_Cnt	TotalSales	Total_Custo
22	STR123	Fashion Bazar	Apparel	Mumbai	15.2	80	304.0	
23	STR124	Digital Bazar	Electronincs	Mumbai	13.3	80	350.0	
24	STR125	Electronics Zone	Electronincs	Kolkata	19.2	80	400.0	
25	STR126	Apparel Zone	Apparel	Kolkata	27.3	40	79.0	
26	STR127	Super Bazar	Super Market	Kolkata	26.0	40	120.3	
27	STR128	Super Market	Super Market	Kolkata	30.4	40	95.1	
28	STR129	Central Store	Super Market	Kolkata	15.8	80	351.0	
29	STR130	Apparel Zone	Apparel	Kolkata	19.7	60	145.0	
30	STR131	Fashion Bazar	Apparel	Kolkata	15.0	80	301.0	
31	STR132	Digital Bazar	Electronincs	Kolkata	21.4	40	121.0	

In [5]:
```
cols = stores.columns.tolist()
```

In [15]:
```
cols[0:5]
```

Out[15]: ['StoreCode', 'StoreName', 'StoreType', 'Location', 'OperatingCost']

In [16]:
```
stores.shape # givews me a tuple (nrow, ncol)
```

Out[16]: (32, 15)

In [9]:
```
# Describing a dataframe
stores.shape # givews me a tuple (nrow, ncol)

stores.shape[0] # no of rows
stores.shape[1] # no of cols
```

Out[9]: 15

In [ ]:
```
stores.dtypes
```

In [ ]:
```
stores.ndim
```

In [ ]:
```
stores.info()
```

In [ ]:
```
stores.get_dtype_counts()
```

In [ ]:
```
stores.head()
stores.tail()
```

In [ ]:
```
stores.columns
stores.columns.tolist()
```

In [ ]:
```
stores.index.tolist()
```

In [ ]:
```
stores.values
```

In [ ]:
```
stores.describe() # all the numeric columns in my data
```

In [10]:
```
# 1. Fetching Columns and rows

# 1.1 Use the . operator
Location = stores.Location
Sales = stores.TotalSales
```

In [11]:
```
stores.Location
```

Out[11]:
```
0       Delhi
1       Delhi
2       Delhi
3       Delhi
4       Delhi
5       Delhi
6       Delhi
7       Delhi
8     Chennai
9     Chennai
10    Chennai
11    Chennai
12    Chennai
13    Chennai
14    Chennai
15    Chennai
16     Mumbai
17     Mumbai
18     Mumbai
19     Mumbai
20     Mumbai
21     Mumbai
22     Mumbai
23     Mumbai
24    Kolkata
25    Kolkata
26    Kolkata
27    Kolkata
28    Kolkata
29    Kolkata
30    Kolkata
31    Kolkata
Name: Location, dtype: object
```

In [14]:
```
print stores.TotalSales.mean()
print stores.TotalSales.median()
print stores.TotalSales.kurtosis()
print stores.TotalSales.sum()
```

```
230.721875
196.3
-1.06752340014
7383.1
```

In [ ]:
```
stores.Location.dtypes

print stores.OperatingCost.dtype
```

In [ ]:
```
# 1.2 Use the  [] notation

# Motive is to get a list of columns by index or by values
stores["Location"] # Good for one column
# gives a Series

#"Total Sales"
#stores.Total Sales
#stores["Total Sales"]
```

In [ ]:
```
# for multiple columns, pass a list [] to the dataFrame[]

stores["Location"]
stores[["Location","TotalSales"]]

# stores[c("Location","TotalSales")]
# gives a new DataFrame
```

In [ ]:
```
# Using [] and indexes
# iloc attribute - to fetch by index location
stores.iloc[:,] # one column gives series
```

In [ ]:
```
stores.iloc[:,:]
```

In [ ]:
```
stores.iloc[:,1:5] # multiple columns gives a new data frame

stores.iloc[:,2:14:2]
# 2,4,6..14
```

In [ ]:
```

# use loc attribute to get names by column names
stores.loc[:,"StoreName"]
```

In [ ]:
```
tt = (stores.dtypes == "int64") | (stores.dtypes == "float64")
```

In [ ]:
```
# Fetcing rows along with columns
stores.iloc[1:3,:]
```

In [ ]:
```
stores.iloc[1:3,1:4] # 2nd 3rd rows, 2nd, 3rd, 4th columns
```

In [ ]:
```
stores.iloc[[1,2,10,11],[0,1,5,8]]

#stores[c(1,2,10,11),c(1,5,8)]
```

In [ ]:
```
# 2. Data types of columns

# Numbers

# To strings
stores1 = stores
```

In [ ]:
```
stores1.TotalSales = stores1.TotalSales.astype(str)
```

In [ ]:
```
# to bool
stores1.TotalSales.astype(bool) # all non 0 values will be True
```

In [ ]:
```
# Strings to numeric

# Logically correct only if the text has no characters or symbols
pd.to_numeric(stores.TotalSales)

#pd.to_numeric(stores.Location,errors='coerce')
# Any alphanumeric text can be coerced to NaN
```

In [ ]:
```
s1 = pd.Series([2434,56778,1234,"A","C","12","678"])

print s1.dtypes

pd.to_numeric(s1,errors = 'coerce')
```

In [51]:
```
# Handling Dates

import datetime as dt

# Sample Data from excel with dates
dict1 = {"SNo":[1,2,3,4,5],
         "Empname" : ["Whitmann","May","Hammond","Clarkson","She"],
         "DoB" : ["16May1971","21April1970","8June1961","12April1970","14July1966
         "Sal" : [23445,45651,83235,32452,33565],
         "DoJ" : [20111201,20120908,20120202,20150110,20140919]
}
```

In [52]:
```
dict1
```

Out[52]:
```
{'DoB': ['16May1971', '21April1970', '8June1961', '12April1970', '14July1966'],
 'DoJ': [20111201, 20120908, 20120202, 20150110, 20140919],
 'Empname': ['Whitmann', 'May', 'Hammond', 'Clarkson', 'She'],
 'SNo': [1, 2, 3, 4, 5],
 'Sal': [23445, 45651, 83235, 32452, 33565]}
```

In [53]:
```
EmpSample = pd.DataFrame(dict1)
```

In [54]:
```
EmpSample
```

Out[54]:

	DoB	DoJ	Empname	SNo	Sal
0	16May1971	20111201	Whitmann	1	23445
1	21April1970	20120908	May	2	45651
2	8June1961	20120202	Hammond	3	83235
3	12April1970	20150110	Clarkson	4	32452
4	14July1966	20140919	She	5	33565

In [55]:
```
EmpSample = EmpSample.iloc[:,[3,2,4,0,1]]
EmpSample
```

Out[55]:

	SNo	Empname	Sal	DoB	DoJ
0	1	Whitmann	23445	16May1971	20111201
1	2	May	45651	21April1970	20120908
2	3	Hammond	83235	8June1961	20120202
3	4	Clarkson	32452	12April1970	20150110
4	5	She	33565	14July1966	20140919

In [ ]:
```
```

In [57]:
```
EmpSample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
SNo        5 non-null int64
Empname    5 non-null object
Sal        5 non-null int64
DoB        5 non-null object
DoJ        5 non-null int64
dtypes: int64(3), object(2)
memory usage: 272.0+ bytes
```

In [58]:
```
EmpSample.DoB
```

Out[58]:
```
0      16May1971
1    21April1970
2      8June1961
3    12April1970
4     14July1966
Name: DoB, dtype: object
```

In [61]:
```
EmpSample["DoB"] = pd.to_datetime(EmpSample.DoB,format = ["%d%B%Y","%m%d%Y"])

#pd.to_datetime(EmpSample.DoB,format = "%d%B%Y")
```

In [62]:
```
EmpSample["DoB"] # the only way how proper dates are displayed - YYYY-MM-DD
```

Out[62]:
```
0   1971-05-16
1   1970-04-21
2   1961-06-08
3   1970-04-12
4   1966-07-14
Name: DoB, dtype: datetime64[ns]
```

In [67]:
```
EmpSample.DoJ
```

Out[67]:
```
0    20111201
1    20120908
2    20120202
3    20150110
4    20140919
Name: DoJ, dtype: object
```

In [68]:
```
EmpSample["DoJ"] = EmpSample.DoJ.astype(str)
EmpSample["DoJ"] = pd.to_datetime(EmpSample.DoJ, format = "%Y%m%d")
EmpSample["DoJ"]
```

Out[68]:
```
0   2011-12-01
1   2012-09-08
2   2012-02-02
3   2015-01-10
4   2014-09-19
Name: DoJ, dtype: datetime64[ns]
```

In [ ]:
```
EmpSample.DoB.dt.date
EmpSample.DoB.dt.day
EmpSample.DoB.dt.month
EmpSample.DoB.dt.year
EmpSample.DoB.dt.weekday_name
```

In [ ]:
```
EmpSample.DoB.dt.strftime("%b, %d, %Y")
# Oct, 21, 1990
```

In [ ]:
```
pd.to_datetime(EmpSample.DoJ, format = "%Y%m%d")
```

In [ ]:
```
s = "1700-01-26"
pd.to_datetime(s,format = "%Y-%m-%d")
```

In [ ]:
```
EmpSample.DoJ.astype(str)
DoJ = pd.to_datetime(EmpSample.DoJ,format = "%Y%m%d")
DoJ
```

```
import datetime as dt # dt is a package
age = dt.datetime.now().date()- EmpSample.DoB
# datetime here is a module from package datetime or dt
age
```

In [ ]:
```
age/365.25
```

In [71]:
```
EmpSample.DoB
```

Out[71]:
```
0   1971-05-16
1   1970-04-21
2   1961-06-08
3   1970-04-12
4   1966-07-14
Name: DoB, dtype: datetime64[ns]
```

In [72]:
```
# 1971-05-16 - "Weekday, 16 May 1971"
EmpSample.DoB.dt.strftime("%A, %d %B %Y")
```

Out[72]:
```
0       Sunday, 16 May 1971
1    Tuesday, 21 April 1970
2    Thursday, 08 June 1961
3     Sunday, 12 April 1970
4    Thursday, 14 July 1966
Name: DoB, dtype: object
```

In [73]:
```
from datetime import datetime as dt
```

In [76]:
```
Today = dt.now().date()
Age = Today - EmpSample.DoB
```

In [77]:
```
Age
```

Out[77]:
```
0   17077 days
1   17467 days
2   20706 days
3   17476 days
4   18844 days
Name: DoB, dtype: timedelta64[ns]
```

Till Here...

In [ ]:
```
# stores1.rename(index)
```

In [ ]:
```
# Renaming Columns

# df = df.rename(columns={'oldName1': 'newName1', 'oldName2': 'newName2'})
stores1 = stores
stores1.head(2)

# Location to City
# OnlinePresence to EComm

stores1 = stores1.rename(columns={"Location":"City","OnlinePresence":"EComm"})
stores1.head(2)
```

In [ ]:
```
print stores1.columns,
```

In [ ]:
```
stores1.head(2)
```

In [ ]:
```
# Adding a new column

# GrandTotalSales = TotalSales * Total_Customers

res = stores1.TotalSales * stores1.Total_Customers
stores1['GrandTotalSales'] = res
```

In [ ]:
```
stores1['NetProfit'] = stores1.TotalSales - (stores1.OperatingCost + stores1.AcqC
```

In [ ]:
```
stores['NetExpense'] = stores.OperatingCost + stores.AcqCostPercust
```

In [ ]:
```
stores1.head(2)
```

In [ ]:
```
stores = stores.assign(GrandTotalSales = stores.TotalSales * stores.Total_Custome
```

In [ ]:
```
stores1.head(2)
```

In [ ]:
```
# Rearranging
stores1.iloc[1:6,[0,1,2,3,5,6,7,15,8,4,16,9,10,11,12,13,14]]
```

In [ ]:
```
[]
```

In [ ]:
```
# Deleting or dropping a column
# del df['column_name']

del stores1['NetProfit2']

# del would permanently delete a column - be careful!
```

In [ ]:
```
stores1.head(2)
```

In [ ]:
```
# Applying conditions on a dataframe

# Select all columns where TotalSales > 250

#df.loc[<condition>,<column name>]

#stores.loc[stores.TotalSales >= 250]

stores.loc[stores.Location == "Delhi"]
```

In [ ]:
```
stores.loc[stores.Location == "Delhi",["StoreType","StoreName"]]
```

In [ ]:
```
stores.loc[((stores.TotalSales > 150) & (stores.TotalSales < 300)) & ((stores.Loc
```

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```
# Select StoreName,Location,TotalSales and OnlinePresence from stores where Totals

stores.loc[(stores.TotalSales >= 150) & (stores.TotalSales <= 250),["StoreName","
```

In [ ]:
```
# Sorting

# df.sort_values(by='column', ascending=False)

# Location asc order
Sort1 = stores.sort_values(by = "Location")
```

In [ ]:
```
# TotalSales desc order
Sort2 = stores.sort_values(by = "TotalSales",ascending=False)
```

In [ ]:
```
# Location in asc and TotalSales in desc
Sort4 = stores.sort_values(by = ["Location","TotalSales"],ascending=[True,False])
Sort4
```

In [ ]:
```
# Location in asc and TotalSales in desc
Sort4 = stores.sort_index(by = ["Location","TotalSales"],ascending=[1,0])
Sort4
```

In [ ]:
```
# Missing Values
stores.AcqCostPercust.isnull()
#stores.AcqCostPercust[stores.AcqCostPercust.isnull()]
#stores.AcqCostPercust[stores.AcqCostPercust.isnull()] = 0
```

In [ ]:
```
# Missing values denoted by NaN
```

In [ ]:
```
# Create a dummy data
raw_data = {'first_name': ['Jason',"", 'Tina', 'Jake', 'Amy'],
        'last_name': ['Miller', np.nan, 'Ali', 'Milner', 'Cooze'],
        'age': [42, np.nan, 36, 24, 73],
        'gender': ['m', np.nan, 'f', 'm', 'f'],
        'preTestScore': [4, np.nan, np.nan, 2, 3],
        'postTestScore': [25, np.nan, np.nan, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['first_name', 'last_name', 'age', 'gender'
```

In [ ]:
```
# Drop missing observations

df_no_missing = df.dropna()

# Drop rows where all cells in that row is NA

df_cleaned = df.dropna(how='all')
```

In [ ]:
```
# Fill in missing data with zeros

#MissingWithZero = df.fillna(0)
#MissingWithZero

#stores["AcqCostPercust"] =
stores.AcqCostPercust.fillna(0)
```

In [ ]:
```
# Fill in missing in preTestScore with the mean value of preTestScore

MissingWithMean = df["preTestScore"].fillna(df["preTestScore"].mean())

stores['AcqCostPercust'] = stores["AcqCostPercust"].fillna(stores["AcqCostPercust
MissingStores
```

In [ ]:
```
# Count Missing values
df.isnull()
```

In [ ]:
```
df.preTestScore.isnull()

df.loc[df.last_name.isnull()]
```

In [ ]:
```
df.preTestScore.isnull().sum()
```

In [ ]:
```
# stores.isnull().sum()
stores.AcqCostPercust.isnull().sum()
```

In [ ]:
```
# Get column names that have missing values
stores.columns
```

In [ ]:
```
ColList = stores.columns

ColListMissing = ColList[stores.isnull().any()]
ColListMissing
type(ColListMissing)
ColListMissing.tolist()
```

In [ ]:
```
stores.AcqCostPercust
```

In [ ]:
```
# Checking for duplicates
```

In [ ]:
```
raw_data = {'first_name': ['Hector', 'CapnJack', 'Will', 'Davy', 'Tia','Hector',
        'test1': [70, 42, 26, 62, 45,70, 42, 26, 62, 45],
        'test2': [4, 8, 31, 2, 3,4, 9, 31, 2, 3],
        'test3': [25, 20, 57, 62, 70,25, 20, 57, 62, 70]}
df = pd.DataFrame(raw_data, columns = ['first_name', 'test1', 'test2', 'test3'])
```

In [31]:
```
df = pd.read_csv("Score.csv")
df
```

Out[31]:

	Student	Section	Test1	Test2	Final
0	Capalleti	1	94	91	87
1	Dubose	2	51	65	91
2	Engles	1	95	97	97
3	Grant	2	63	75	80
4	Krupski	2	80	76	71
5	Lundsford	1	92	40	86
6	Mcbane	1	75	78	72
7	Capalleti	1	94	65	87
8	Dubose	2	51	65	91
9	Engles	1	95	97	97
10	Grant	2	63	75	80
11	Krupski	2	80	76	71
12	Lundsford	1	92	40	86
13	Mcbane	1	75	78	72

In [25]:
```
# Identifying duplicate columns
df.duplicated()
```

Out[25]:
```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8      True
9      True
10     True
11     True
12     True
13     True
dtype: bool
```

In [26]:
```
DuplicatedSubset = df.loc[df.duplicated(),]
```

Out[26]:

	Student	Section	Test1	Test2	Final
8	Dubose	2	51	65	91
9	Engles	1	95	97	97
10	Grant	2	63	75	80
11	Krupski	2	80	76	71
12	Lundsford	1	92	40	86
13	Mcbane	1	75	78	72

In [33]:
```
UniqueRecords = df.loc[~df.duplicated()]
```

In [ ]:
```
ss1 = pd.Series([1,2,5,3,1,5,1,2,3,6,8,9,10])
```

In [ ]:
```
ss1[-ss1.duplicated()]
```

In [ ]:
```
df.loc[-df.duplicated()] # All unique values
```

In [ ]:
```
# Drop duplicates

df.drop_duplicates() # remove all duplicates or give all unique values
```

In [ ]:
```
df.drop_duplicates(["first_name"])
```

In [ ]:
```
df.drop_duplicates(['Student'], keep='last')

# keep = "first" or "last" or False
# 0,100,120,500,550

# keep = "first" row no 0
# keep = "last" row no 550
# keep = False will delete all the 5 rows
```

In [ ]:
```
df
```

In [30]:
```
df.Student.duplicated()
df.loc[-df.Student.duplicated(),:]
```

Out[30]:

	Student	Section	Test1	Test2	Final
0	Capalleti	1	94	91	87
1	Dubose	2	51	65	91
2	Engles	1	95	97	97
3	Grant	2	63	75	80
4	Krupski	2	80	76	71
5	Lundsford	1	92	40	86
6	Mcbane	1	75	78	72

In [ ]:
```
Emp.loc[(Emp.EMPID.duplicated()) & (Emp.EMail.duplicated()) & (Emp.Phone.duplicat
```

In [ ]:
```
# Binning
```

In [ ]:
```
# Chosing the bins

print "min: ",stores.TotalSales.min()," max : ",stores.TotalSales.max()
```

In [ ]:
```
stores.TotalSales.min()
```

In [ ]:
```
stores.TotalSales.max()
```

In [ ]:
```
q1 = stores.TotalSales.quantile([0,0.05,0.1,0.25,0.5,0.75,1])
q1[0.25]
```

In [ ]:
```
q2 = stores.TotalSales.quantile([0.3,0.6,0.9,])
q2
q2[0.3]
```

In [ ]:
```
stores.StoreType.value_counts()
```

In [ ]:
```
# Create a column StorePerformance based on TotalSales

# if TotalSales <= 30% - "Low Performance"
# if TotalSales > 30%  and <= 60%- "Average Performance"
# if TotalSales > 60% and <= 90%  - "Good Performance"
# if TotalSales > 90% - "High Performance"
```

In [ ]:
```
Category = np.where(stores.TotalSales <= q2[0.3],"Low","High")
Category
```

In [ ]:
```
q = q2
stores["Category"] = np.where(stores.TotalSales <= q[0.3],"Low",np.where((stores.
Category
```

In [ ]:
```
# the where() function from numpy

# where(<condition>,True,False)
# ifelse(c,T,F)
stores['test1'] = np.where((stores['TotalSales'] >= 250) & (stores['TotalSales'] 
stores.test1.value_counts()
```

In [ ]:
```
# where(<condition>,TRUE = <s1>,FALSE = <s2 for false>)

result = "High" if stores.TotalSales > 90 else "Not High"
```

```
stores['OfficeCode'] = np.where(stores['Location'] == "Delhi",
'D',np.where(stores['Location']== "Mumbai", 'BOM',np.where(stores['Location']==
"Kolkata", 'KOLOffice',np.where(stores['Location']== "Chennai",
'MASOffice',''))))
```

```
stores["OfficeCode"] = np.where(stores['Location'] ==
"Delhi","D",np.where(stores["Location"] == "Chennai","C"))
```

```
test
```

In [ ]:
```
stores.loc[:,['Location','test']]
```

```
TotalSales

"StorePerformance"
< 100 - "Low Performing"
> 100- <= 250 - "Average Performing"
> 250 - <= 350 - "Good Performing"
> 350 - "High Performing"
```

```
stores["StoresPerformance"] = np.where(stores['TotalSales'] < 100,"Low",
                                       np.where(stores['TotalSales'] ...
```

```
np.where(stores['TotalSales'] < 100,"Low",np.where((stores['TotalSales'] > 100)
and (stores["TotalSales"] <= 250,"Average",np.where((stores['TotalSales'] >
250..)))
```

In [ ]:

In [85]:
```
# Merging

txn = pd.read_csv("Transaction_Summary.csv")
demographic = pd.read_csv("Demographic_Data.csv")

demographic
```

Out[85]:

	CustName	Gender	Age	Location	Salary	Education	Mobile
0	Alex	M	21	UK	19159	PhD	8834777722
1	Tom	M	34	USA	10461	X	8812237772
2	Michel	M	25	India	19961	XII	8834777232
3	Michael	M	28	Belgium	821	B.Com	8831234222
4	Patrik	M	29	Australia	13743	MBA	8823427722
5	Hans	M	34	Japan	25000	MS	8834772342
6	Biliana	F	26	Russia	3000	MS	8837427722
7	Raj	M	29	India	2000	MS	9892877722
8	Laila	F	26	India	4000	MBA	7867277722
9	Prabhas	F	35	India	20000	MBA	8834772321

In [86]:
```
txn
```

Out[86]:

	CustomerName	Total_Transaction_value	No_of_holding_prods	No_of_visits	No_of_channels
0	Alex	19159	3	3	1
1	Tom	10461	2	7	2
2	Michel	19961	4	8	3
3	Hans	821	1	2	2
4	Biliana	13743	6	9	3
5	Rajesh	25000	2	12	2
6	Laila	3000	1	6	1
7	Prabhas	2000	2	4	2
8	Ramu	30000	6	2	3

In [87]:
```
InnerJoin = pd.merge(left = txn,right = demographic,left_on = "CustomerName",righ
```

In [89]:
```
InnerJoin
```

Out[89]:

	CustomerName	Total_Transaction_value	No_of_holding_prods	No_of_visits	No_of_channels	Cu
0	Alex	19159	3	3	1	
1	Tom	10461	2	7	2	
2	Michel	19961	4	8	3	
3	Hans	821	1	2	2	
4	Biliana	13743	6	9	3	
5	Laila	3000	1	6	1	
6	Prabhas	2000	2	4	2	

In [ ]:
```
RightJoin = pd.merge(left = txn,right = demographic,right_on = "CustName", left_o
```

In [ ]:
```
RightJoin
```

In [90]:
```
LeftJoin = pd.merge(left = txn,right = demographic,right_on = ["CustName"], left_
LeftJoin
```

Out[90]:

	CustomerName	Total_Transaction_value	No_of_holding_prods	No_of_visits	No_of_channels	Cu
0	Alex	19159	3	3	1	
1	Tom	10461	2	7	2	
2	Michel	19961	4	8	3	
3	Hans	821	1	2	2	
4	Biliana	13743	6	9	3	
5	Rajesh	25000	2	12	2	
6	Laila	3000	1	6	1	
7	Prabhas	2000	2	4	2	
8	Ramu	30000	6	2	3	

In [ ]:
```
```

In [ ]:
```
OuterJoin = pd.merge(left = txn,right = demographic,right_on = "CustName", left_o
OuterJoin
```

In [ ]:
```
# Set indicator option to True
OuterJoin = pd.merge(left = txn,right = demograhics,right_on = "CustName", left_o
```

In [ ]:
```
OuterJoin
```

In [ ]:
```
# Group By in pandas
```

In [78]:
```
# SELECT Location, avg(TotalSales) FROM stores GROUP BY Location

temp = stores.groupby(["Location"])
temp.TotalSales.mean()
```

Out[78]:
```
Location
Chennai    279.4250
Delhi      222.2125
Kolkata    201.5500
Mumbai     219.7000
Name: TotalSales, dtype: float64
```

In [ ]:
```
temp
```

In [80]:
```
#temp.TotalSales.agg({"SumOfTotalSales" : sum})
# key -> the name of the coulumn SumOfTotalSales
# value -> the aggragation - sum, "mean", "median", len, "std", "var"

#SELECT Location, sum(TotalSales) as 'SumOfTotalSales' FROM stores GROUP BY Locat


# mean, median, std, len, count, var

# use agg option to create a dict of various options
# use reset_index() to get a proper dataframe
#temp['TotalSales'].agg(sum)

temp = stores.groupby("Location")
Result1 = temp["TotalSales"].agg({"AverageOfTotalSales":"mean"})
Result1 = Result1.reset_index()
Result1
```

```
C:\Users\admin\Anaconda2\lib\site-packages\ipykernel\__main__.py:15: FutureWarn
ing: using a dict on a Series for aggregation
is deprecated and will be removed in a future version
```

Out[80]:

	Location	AverageOfTotalSales
0	Chennai	279.4250
1	Delhi	222.2125
2	Kolkata	201.5500
3	Mumbai	219.7000

In [ ]:
```
temp = stores.groupby("Location")
Result = temp.TotalSales.agg({"Avg of TotalSales":"mean"})
Result.reset_index()
```

In [81]:
```
temp = stores.groupby(["Location","StoreType"])
Result2 = temp['TotalSales'].agg({"SumOfTotalSales":sum,"CountOfValues":len,"Avgo
Result2 = Result2.reset_index()
Result2
```

```
C:\Users\admin\Anaconda2\lib\site-packages\ipykernel\__main__.py:2: FutureWarni
ng: using a dict on a Series for aggregation
is deprecated and will be removed in a future version
  from ipykernel import kernelapp as app
```

Out[81]:

	Location	StoreType	CountOfValues	AvgofTotalSales	SumOfTotalSales
0	Chennai	Apparel	3.0	305.133333	915.4
1	Chennai	Electronincs	2.0	300.400000	600.8
2	Chennai	Super Market	3.0	239.733333	719.2
3	Delhi	Apparel	3.0	248.333333	745.0
4	Delhi	Electronincs	2.0	153.350000	306.7
5	Delhi	Super Market	3.0	242.000000	726.0
6	Kolkata	Apparel	3.0	175.000000	525.0
7	Kolkata	Electronincs	2.0	260.500000	521.0
8	Kolkata	Super Market	3.0	188.800000	566.4
9	Mumbai	Apparel	3.0	233.566667	700.7
10	Mumbai	Electronincs	2.0	395.000000	790.0
11	Mumbai	Super Market	3.0	88.966667	266.9

In [ ]:
```
#a = stores.groupby("Location")["TotalSales","OperatingCost"].agg({"Count":len,
                                                                   #"Mean":'mean',
                                                                    # "Sum" :sum })
temp = stores.groupby("Location")
temp["TotalSales","OperatingCost"].agg({"Sum_Col":sum}).reset_index()
```

In [ ]:
```
temp = stores.groupby("Location")
a = temp["TotalSales","OperatingCost"].agg({"Size":len,"Sum":sum,"Mean":"mean"})
a.reset_index()
```

In [ ]:
```
a.Count;a.Mean;a.Sum
```

In [ ]:
```
# Sum and Count on TotalSales and only average on OperatingCost

b = stores.groupby(["Location","StoreType"])["TotalSales","OperatingCost"].agg({"
```

In [ ]:
```
temp = stores.groupby(["Location","StoreType"])
temp[["TotalSales","OperatingCost"]].agg({"Sum":sum}).reset_index()
```

In [ ]:
```
cust = pd.read_csv("Customers.csv")
import datetime as dt # dt is a package
# datetime here is a module from package datetime or dt
```

In [ ]:
```
d1 = cust["recent date"][1]
```

In [ ]:
```
d1_date = pd.to_datetime(d1,format = "%Y%m%d")
```

In [ ]:
```
d1_date
```

In [ ]:
```
d1
```

In [ ]:
```
pd.Series(d1_date)
```

In [ ]:
```
cust["recent date"] = pd.to_datetime(cust["recent date"],format = "%Y%m%d")
```

In [ ]:
```
recdt = cust["recent date"][1:10]
```

In [ ]:
```
# Time differnece

td = dt.datetime.today() - recdt
td = td/np.timedelta64(1, 'D')
td/365.25
```

In [ ]:
```
from __future__ import division
((len(stores.AcqCostPercust)-stores.AcqCostPercust.count())/len(stores.AcqCostPer
```

In [ ]:
```
import seaborn as sb
```

In [ ]:
```

```

In [ ]:
```
def functionName(arg1,arg2....):
    ...
    ...
    ...
    ...

    return(val)
```

In [ ]:
```
def AddNum(x,y):
    if (type(x) == str) | (type(y) == str):
        return "You have entered strings"
    else:
        return(x + y)
```

In [ ]:
```
AddNum(x = 12, y = 4)
```

In [ ]:
```
AddNum(x = 12, y = "s")
```

In [ ]:
```
a = raw_input("Enter a number")
print a

# the object that is passed in raw_input is always a string

a = int(a)
print a * 1000
```

In [ ]:
```
AddNum()
```

In [ ]:
```
# args
# kwargs

MyFunc(*arguments)
MyFunc(**arguments)
```

In [ ]:
```
def AddNum1(*args):
    print args
```

In [ ]:
```
def AddNum2(**kwargs):
    print kwargs
```

In [ ]:
```
AddNum1(10,50,14,26,78)
```

In [ ]:
```
AddNum2(first = 21,sec = 52,third = 100,fourth = 300)
```

In [ ]:
```
def AddNum(x,y,*args,**kwargs):
    print x
    print y
    print args
    print kwargs
```

In [ ]:
```
AddNum(x=12,y = 20)
```

In [ ]:
```
stores.to_csv()

stores.to_excel()

stores.to_html()
```

In [ ]:
```

```

In [ ]:
```
from __future__ import print_function
```

In [ ]:
```
# The print statements
my_name = 'Zed A. Shaw'
my_age = 35 # not a lie
my_height = 74 # inches
my_weight = 180 # lbs
my_eyes = 'Blue'
my_teeth = 'White'
my_hair = 'Brown'

print("Let's talk about %s." % my_name)
```

In [ ]:
```
l1 = [10, 14, 9, 666, 34, 58, 23, 32, 100, 5]
ListLen = len(l1)
for i in range(0,ListLen):
    print(l1[i],end = " ")
```

In [36]:
```
import pymysql as sql
```

In [37]:

```
# In[2]:

# Establish a connection with DBMS
# Either specify a DSN or the user credentials
db = sql.connect("localhost","testuser","password","regrecords")


# In[3]:

db


# In[4]:

import pandas as pd


# In[5]:

CarsData_df = pd.read_sql("SELECT * FROM cars_data",con = db)


# In[6]:

CarsData_df.head(3)
CarsData_df.tail(3)
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
<ipython-input-37-f16a67be24c8> in <module>()
      6 # Establish a connection with DBMS
      7 # Either specify a DSN or the user credentials
----> 8 db = sql.connect("localhost","testuser","password","regrecords")
      9
     10

C:\Users\admin\Anaconda2\lib\site-packages\pymysql\__init__.pyc in Connect(*arg
s, **kwargs)
     88     """
     89     from .connections import Connection
---> 90     return Connection(*args, **kwargs)
     91
     92 from pymysql import connections as _orig_conn

C:\Users\admin\Anaconda2\lib\site-packages\pymysql\connections.pyc in __init__
(self, host, user, password, database, port, unix_socket, charset, sql_mode, re
ad_default_file, conv, use_unicode, client_flag, cursorclass, init_command, con
nect_timeout, ssl, read_default_group, compress, named_pipe, no_delay, autocomm
it, db, passwd, local_infile, max_allowed_packet, defer_connect, auth_plugin_ma
p, read_timeout, write_timeout)
    686             self._sock = None
    687         else:
--> 688             self.connect()
    689
```

```
    690     def _create_ssl_ctx(self, sslp):

C:\Users\admin\Anaconda2\lib\site-packages\pymysql\connections.pyc in connect(s
elf, sock)
    935                 exc.traceback = traceback.format_exc()
    936                 if DEBUG: print(exc.traceback)
--> 937                 raise exc
    938
    939             # If e is neither DatabaseError or IOError, It's a bug.

OperationalError: (2003, "Can't connect to MySQL server on 'localhost' ([Errno
 10061] No connection could be made because the target machine actively refused
 it)")
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
b = stores.groupby(["Location","StoreType"])["TotalSales","OperatingCost"].agg({"
```

In [ ]:
```
temp = stores.groupby(["Location","StoreType"])

Res = temp[["TotalSales","OperatingCost"]].agg({"TotalSales":sum,"OperatingCost":
Res = Res.rename(columns={"TotalSales":"SumOfTotalSales","OperatingCost":"AvgOfOp
```

In [ ]:
```
Res
```

In [42]:
```
pd.merge(left=txn,right=demographic,left_on="CustomerName",right_on="CustName",ho
```

Out[42]:

	CustomerName	Total_Transaction_value	No_of_holding_prods	No_of_visits	No_of_channels	Cu
0	Alex	19159	3	3	1	
1	Tom	10461	2	7	2	
2	Michel	19961	4	8	3	
3	Hans	821	1	2	2	
4	Biliana	13743	6	9	3	
5	Rajesh	25000	2	12	2	
6	Laila	3000	1	6	1	
7	Prabhas	2000	2	4	2	I
8	Ramu	30000	6	2	3	

```
SELECT Location, sum(TotalSales) as 'SumOfTotalSales' from stores GROUP BY
Location
```

In [50]:
```
# step 1
# getting a grouped object

t = stores.groupby(["Location"])
t

# an exact copy of the DF - data and column names

# Step 2
# aggregations
# Specify the continious var TotalSales
# adn specify what calculations
# mean
# sum
# std
# var
# len
# median
Result1 = t.TotalSales.agg({"SumOfTotalSales" : sum })
# I'll pass a dictionary
# {key : value }

#{"SumOfTotalSales" : sum } # sum, mean, var, len, std, median
# when typing in the function name -> font color = green but if font color = blac

Result1.reset_index()
```

```
C:\Users\admin\Anaconda2\lib\site-packages\ipykernel\__main__.py:19: FutureWarn
ing: using a dict on a Series for aggregation
is deprecated and will be removed in a future version
```

Out[50]:

	Location	SumOfTotalSales
0	Chennai	2235.4
1	Delhi	1777.7
2	Kolkata	1612.4
3	Mumbai	1757.6

In [51]:
```
Cust_demo = pd.read_csv("Cust_demo.csv")
```

In [ ]:
```
"SELECT Martial_Status, count(*) as 'Freq' FROM Cust_demo GROUP BY Martial_Status
```

In [54]:
```
t = Cust_demo.groupby(["Martial_Status"])
t
```

Out[54]: <pandas.core.groupby.DataFrameGroupBy object at 0x000000000954CB00>

In [53]:
```
Res = t.Martial_Status.agg({"CountOfMartialSatus":len})
Res = Res.reset_index()
Res
```

```
C:\Users\admin\Anaconda2\lib\site-packages\ipykernel\__main__.py:1: FutureWarni
ng: using a dict on a Series for aggregation
is deprecated and will be removed in a future version
  if __name__ == '__main__':
```

Out[53]:

	Martial_Status	CountOfMartialSatus
0	Married	42845
1	Single	107111

In [55]:
```
t = stores.groupby(["Location","StoreType"])
```

In [59]:
```
Result = t[["TotalSales","OperatingCost"]].agg({"AvgOfTotalSales":"mean","SumOfTo
```

```
C:\Users\admin\Anaconda2\lib\site-packages\ipykernel\__main__.py:1: FutureWarni
ng: using a dict on a Series for aggregation
is deprecated and will be removed in a future version
  if __name__ == '__main__':
```

In [60]:
```
Result.reset_index()
```

Out[60]:

	Location	StoreType	AvgOfTotalSales	SumOfTotalSales
0	Chennai	Apparel	305.133333	915.4
1	Chennai	Electronincs	300.400000	600.8
2	Chennai	Super Market	239.733333	719.2
3	Delhi	Apparel	248.333333	745.0
4	Delhi	Electronincs	153.350000	306.7
5	Delhi	Super Market	242.000000	726.0
6	Kolkata	Apparel	175.000000	525.0
7	Kolkata	Electronincs	260.500000	521.0
8	Kolkata	Super Market	188.800000	566.4
9	Mumbai	Apparel	233.566667	700.7
10	Mumbai	Electronincs	395.000000	790.0
11	Mumbai	Super Market	88.966667	266.9

In [ ]: