Deliverables:

- Submit a single zip-compressed file that has the name: YourLastName_Exercise_1 that has the following files:

    1. Your **PDF document** that has your Source code and output
    2. Your **ipynb script** that has your Source code and output

Objectives:

In this exercise, you will:

- Perform data analysis tasks on data read from a CSV file and loaded into a DataFrame object
- Use sqlalchemy to load data stored in a DatFrame object into sqlite database engine
- Use sqlalchemy to connect to sqlite database engine to execute SQL queries

Submission Formats :

Create a folder or directory with all supplementary files with your last name at the beginning of the folder name, compress that folder with zip compression, and post the zip-archived folder under the assignment link in Canvas. The following files should be included in an archive folder/directory that is uploaded as a single zip-compressed file. (Use zip, not StuffIt or any 7z or any other compression method.)

1. Complete IPYNB script that has the source code in Python used to access and analyze the data. The code should be submitted as an IPYNB script that can be be loaded and run in Jupyter Notebook for Python
2. Output from the program, such as console listing/logs, text files, and graphics output for visualizations. If you use the Data Science Computing Cluster or School of Professional Studies database servers or systems, include Linux logs of your sessions as plain text files. Linux logs may be generated by using the script process at the beginning of your session, as demonstrated in tutorial handouts for the DSCC servers.
3. List file names and descriptions of files in the zip-compressed folder/directory.

Formatting Python Code When programming in Python, refer to Kenneth Reitz' PEP 8: The Style Guide for Python Code: http://pep8.org/ (http://pep8.org/) (Links to an external site.)Links to an external site. There is the Google style guide for Python at https://google.github.io/styleguide/pyguide.html (https://google.github.io/styleguide/pyguide.html) (Links to an external site.)Links to an external site. Comment often and in detail.

In [129]:
```
import os

import  pickle

import pandas as pd  # panda's nickname is pd

import numpy as np  # numpy as np

from pandas import DataFrame, Series     # for convenience
```

In [130]:
```
xyzcust10=pd.read_csv('xyzcust10.csv')
```

In [131]:
```
(xyzcust10).dtypes
```

Out[131]:
```
ACCTNO                  object
ZIP                      int64
ZIP4                     int64
LTD_SALES              float64
LTD_TRANSACTIONS         int64
YTD_SALES_2009         float64
YTD_TRANSACTIONS_2009    int64
CHANNEL_ACQUISITION     object
BUYER_STATUS            object
ZIP9_Supercode           int64
ZIP9_SUPERCODE           int64
dtype: object
```

In [132]:
```
type(xyzcust10)
```

Out[132]:
```
pandas.core.frame.DataFrame
```

In [133]:
```
pickle.dump(xyzcust10,open('xyzcust10.p','wb'))
```

In [134]:
```
xyzcust10=pickle.load(open('xyzcust10.p','rb'))

xyzcust10red = xyzcust10.copy() # by default makes a “deep” copy

xyzcust10rev1=xyzcust10.copy()  # by default makes a “deep” copy
```

The above assumes that xyzcust10.p is in your default directory. Otherwise, you'll need to include a path specification, of course. xyzcust10 should be a pandas DataFrame:

In [135]:
```
type(xyzcust10)
```

Out[135]:
```
pandas.core.frame.DataFrame
```

In [136]:
```
xyzcust10.head()
```

Out[136]:

	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	YTD_TRANSA
0	WDQQLLDQL	60084	5016	90.0	1	0.0	
1	WQWAYHYLA	60091	1750	4227.0	9	1263.0	
2	GSHAPLHAW	60067	900	420.0	3	129.0	
3	PGGYDYWAD	60068	3838	6552.0	6	0.0	
4	LWPSGPLLS	60090	3932	189.0	3	72.0	

xyzcust10 appears to have two nine-digit ZIP “supercode” columns with slightly different column labels or names. To see them, try entering xyzcust10.columns or xyzcust10.dtypes at the command prompt. Are the values in these two columns the same? If so, we can get rid of one of them. There are different ways we can figure out whether they are the same, but a simple way is to test each pair of values to see if they are equal or not, and then to total up the results, the number of equal pairs or not equal pairs:

In [137]:
```
xyzcust10.columns
```

Out[137]:
```
Index(['ACCTNO', 'ZIP', 'ZIP4', 'LTD_SALES', 'LTD_TRANSACTIONS',
       'YTD_SALES_2009', 'YTD_TRANSACTIONS_2009', 'CHANNEL_ACQUISITIO
N',
       'BUYER_STATUS', 'ZIP9_Supercode', 'ZIP9_SUPERCODE'],
      dtype='object')
```

In [138]:
```
xyzcust10.dtypes
```

Out[138]:
```
ACCTNO                  object
ZIP                      int64
ZIP4                     int64
LTD_SALES              float64
LTD_TRANSACTIONS         int64
YTD_SALES_2009         float64
YTD_TRANSACTIONS_2009    int64
CHANNEL_ACQUISITION     object
BUYER_STATUS            object
ZIP9_Supercode           int64
ZIP9_SUPERCODE           int64
dtype: object
```

In [139]:
```
(xyzcust10.ZIP9_Supercode!=xyzcust10.ZIP9_SUPERCODE).sum()
```

Out[139]:
```
0
```

which will return zero if the values in the two columns are the same. What result do you get? Note that what's going on here is that what's in the parentheses is a logical test of inequality between the two columns of the DataFrame (which are also pandas Series objects), which results in a Series of true or false Boolean values. The post-pended .sum() function adds up over the Series by treating the Trues as 1's, and the Falses as 0's. So if the result is zero, the two Series are identical, except for their names, of course. We could have also expressed the logical comparison in the parens as (~(xyzcust10.ZIP9_Supercode==xyzcust10.ZIP9_SUPERCODE)) to get the same result, since the the “twidddle,” the ~, works in some pandas contexts as “not.” What kind of result do you think you'd get with the following variation: ~(xyzcust10.ZIP9_Supercode==xyzcust10.ZIP9_SUPERCODE).sum() Why might it be different? Note that we could have referred to the columns differently, for example: xyzcust10['ZIP9_Supercode'] Columns in DataFrames can be referred to in different ways. We'll see more of them going forward.

In [140]:
```
 xyzcust10['ZIP9_Supercode']
```

Out[140]:
```
0        600845016
1        600911750
2        600670900
3        600683838
4        600903932
           ...
30466    600983951
30467    600989681
30468    600983858
30469    600987927
30470    600984160
Name: ZIP9_Supercode, Length: 30471, dtype: int64
```

So, Oops! Someone included the same column in the data twice, but with slightly different names. Why waste the space? Why risk confusion? Let's get rid of one of them: We could do:

In [141]:
```
del xyzcust10['ZIP9_Supercode']
del xyzcust10rev1['ZIP9_Supercode']
```

or

In [142]:
```
xyzcust10red.drop('ZIP9_Supercode',axis=1,inplace=True)
```

Next we're going to shift gears and gobble up some transaction data for XYZ's customers. They are in a table in a SQLite3 relational database ("RDB") file that's called xyz.db. This file is available to you in the compressed file that you download from Canvas. At this point you might want to pickle xyzcust10rev1 in case you need to end your session and start again later. Remember that things in a Python session are not permanent. To make things simple you'll want to put the xyz.db file in a place where you can find it easily from in Anaconda. Your default directory would be a good bet. Remember what it is? See what os.getcwd() tells you.

In [143]:
```
os.getcwd()
```

Out[143]:
```
'/Users/zachtsouprakos/Documents/MSDS/MSDS-420/Module-5/Exercise_3'
```

If you installed the sqlite3 client, you can take a look at this database ("DB") using it and without using Python. sqlSQLite3 is a very simple and easy to use RDB, and it doesn't require a server. Assuming that you've installed it and that you're in the directory were you put xyztrans.db, using the command from your OS command prompt: c:\bader\nu\420\ExercisePractices\ExercisePractice3>sqlite3 xyz.db SQLite version 3.8.8.3 2015-02-25 13:29:11 Enter ".help" for usage hints. sqlite> will start sqlite3 and open the db file. You can see the tables in this db with the sqlite3 command .tables . (That's a period, "." before tables. "Help" in sqlSQLite3 is .help .) sqlite> .tables xyztrans sqlite> There are a couple of different ways to read and write data to RDBs using Python, but the most flexible and easiest may be by using what's in pandas. pandas will make use of the SQLAlchemy package, which is available for installation within Anaconda. (Did you install it in Session 1?) SQLAlchemy provides a consistent interface with different RDBs, SQLite being one of them. Let's get SQLAlchemy into our IPython session:

In [144]:
```
import sqlalchemy
```

Now if you do the sqlalchemy. trick from the command prompt, you'll be able to see SQLAlchemy's various (and many) attributes and functions. To simplify things, let's get a function out of SQLAlchemy that we'll use to define the SQLite3 db we'll be working with:

In [145]:
```
from sqlalchemy import create_engine
```

Now let's specify the xyz db as the SQLite3 RDB we want to work with:

In [146]:
```
engine=create_engine('sqlite:///xyz.db')
```

This assumes that you have xyz.db in your current working directory. There are different valid syntaxes, e.g. sqlite:///:memory: (or, sqlite://) sqlite:///relative/path/to/file.db sqlite:////absolute/path/to/file.db We used the second syntax, above. Be sure to use the correct number of slashes for the version you want to use. You need the enclosing single quotes, too. There's only one table in this RDB. It's called "xyztrans." Let's read it into a DataFrame:

In [147]:
```
xyztrans=pd.read_sql('xyztrans', engine)
```

xyztrans is a DataFrame. This defaults to reading all records from the db. What columns have been read from the table xyztrans? Try:

In [148]:
```
xyztrans.dtypes
```

Out[148]:
```
index             int64
ACCTNO           object
QTY               int64
TRANDATE         object
TRAN_CHANNEL     object
PRICE           float64
TOTAMT          float64
ORDERNO          object
DEPTDESCR        object
dtype: object
```

or

In [149]:
```
xyztrans.columns
```

Out[149]:
```
Index(['index', 'ACCTNO', 'QTY', 'TRANDATE', 'TRAN_CHANNEL', 'PRICE',
'TOTAMT',
       'ORDERNO', 'DEPTDESCR'],
      dtype='object')
```

This db has only one table in it. What if it had more than one, and you didn't know their names? How would you know? Well, one way is to read some “metadata” from it:

In [150]:
```
from sqlalchemy import schema
```

In [151]:
```
xyzMetaData=schema.MetaData(bind=engine)
xyzMetaData.reflect()
```

In [152]:
```
xyzMetaData.tables
```

Out[152]:
```
immutabledict({'xyztrans': Table('xyztrans', MetaData(bind=Engine(sqlit
e:///xyz.db)), Column('index', BIGINT(), table=<xyztrans>), Column('ACC
TNO', TEXT(), table=<xyztrans>), Column('QTY', BIGINT(), table=<xyztran
s>), Column('TRANDATE', TEXT(), table=<xyztrans>), Column('TRAN_CHANNE
L', TEXT(), table=<xyztrans>), Column('PRICE', FLOAT(), table=<xyztrans
>), Column('TOTAMT', FLOAT(), table=<xyztrans>), Column('ORDERNO', TEXT
(), table=<xyztrans>), Column('DEPTDESCR', TEXT(), table=<xyztrans>), s
chema=None)})
```

xyzMetaData.tables will be a dict that contains information about the db. Tables will be keys in this dict:

In [153]:
```
xyzMetaData.tables.keys()
```

Out[153]:
```
dict_keys(['xyztrans'])
```

At this point there's only one table name, 'xyztrans,” in xyz.db. You'll see another method for inspecting DB's below. We're going to write the xyz customer records into a new table in the sqlite3 RDB, but before we do that let's make sure that the records are unique, that is, that no customer has more than one record. We can do this with some pandas DataFrame methods. Using the customer DataFrame xyzcust10rev1

In [154]:
```
xyzcust10rev1.duplicated().sum()
```

Out[154]:
```
292
```

will return a zero if all records are unique, or the number of rows in xyzcust10rev1 that are duplicates. The reason is that the duplicated() method for the DataFrame returns a Series of Trues and Falses, a Boolean Series. Summing over the Series forces the values to be cast as numeric. Oops. There are some duplicates. How many duplicates do you find in xyzcust10rev1? To rid a DataFrame of unduplicated rows,

In [155]:
```
xyzcustUnDup=xyzcust10rev1.drop_duplicates()

xyzcustUnDup.duplicated().sum()
```

Out[155]:
```
0
```

How many unique customer records do you now have? By the way, note that you could have limited your examination to just one or more columns, for example just ACCTNO, customer account number, by providing ACCTNO as an argument or by using it to define a Series:

In [156]:
```
xyzcust10rev1.duplicated('ACCTNO').sum()
```

Out[156]:
```
292
```

In [157]:
```
xyzcust10rev1.ACCTNO.duplicated().sum()
```

Out[157]:
```
292
```

When there are duplicates of a record, which of them do you think .drop_duplicates() retains? Now that we've checked for, and have removed, duplicate customer records, from the customer records, let's write them into a new table in xyztrans.db.

In [158]:
```
xyzcustUnDup.to_sql('xyzcust', engine)
```

Did it create the table in xyz.db? Check:

In [159]:
```
pd.read_sql_table('xyzcust', engine).columns
```

Out[159]:
```
Index(['index', 'ACCTNO', 'ZIP', 'ZIP4', 'LTD_SALES', 'LTD_TRANSACTION
S',
       'YTD_SALES_2009', 'YTD_TRANSACTIONS_2009', 'CHANNEL_ACQUISITIO
N',
       'BUYER_STATUS', 'ZIP9_SUPERCODE'],
      dtype='object')
```

should produce the columns of the DataFrame you wrote to the db. Remember that “engine” refers to the SQLite3 DB by way of defining the connection using SQLAlchemy's create_engine method. How many tables are there now in xyz.db? And, what are their names? Do

In [160]:
```
xyzMetaData.tables.keys()
```

Out[160]:
```
dict_keys(['xyztrans'])
```

Another way to look at the metadata of an RDB using SQLAlchemy is by using the “inspect” method:

In [161]:
```
xyzMetaData
```

Out[161]:
```
MetaData(bind=Engine(sqlite:///xyz.db))
```

In [162]:
```
from sqlalchemy import inspect
```

In [163]:
```
insp=inspect(engine)
```

In [164]:
```
insp.get_table_names()
```

Out[164]:
```
['xyzcust', 'xyztrans']
```

Do you think there are any duplicates in the order transaction data? If so, what would you make of them? You can use SQLAlchemy to query a DB so as to import selected records from an RDB. You can also append records to existing tables in an RDB, create various kinds of DB indexes, and pretty much do everything you would do using standard SQL while interacting with an RDB using a client for it. As a query example, suppose we wanted to select from the xyz tranaction data in the xyztrans.db all transactions made in XYZ's retail stores. These are coded as RT in the table's TRAN_CHANNEL. We could do:

In [165]:
```
rttrans=pd.read_sql_query("SELECT * FROM xyztrans WHERE TRAN_CHANNEL='R
T'", engine)
```

A last point about SQLAlchemy: it has its own declarative language that provides means of interacting with DB's that is more “object oriented” than traditional SQL is. You can find lots of documentation about SQLAlchemy at http://www.sqlalchemy.org.

In [166]:
```
rttrans
```

Out[166]:

	index	ACCTNO	QTY	TRANDATE	TRAN_CHANNEL	PRICE	TOTAMT	ORDE
0	0	WGDQLA	1	09JUN2009	RT	599.85	599.85	CCXXNNXX
1	1	WGDQLA	1	09JUN2009	RT	39.00	39.00	CCXXNNXX
2	2	WGDQLA	1	28NOV2009	RT	15.00	15.00	CCXNXXK)
3	3	WGDQLA	1	28NOV2009	RT	69.00	69.00	CCXNXXK)
4	4	WGDQLA	1	28NOV2009	RT	84.00	84.00	CCXNXXK)
...	...	...	...	...	...	...	...	
53806	62376	GYLYSQQSG	1	14NOV2009	RT	45.00	45.00	CCXCXIK)
53807	62377	GYLYSQQSG	1	14NOV2009	RT	15.00	15.00	CCXCXIK)
53808	62378	GYLYSQQSG	1	29NOV2009	RT	42.00	42.00	CCXCRZE)
53809	62379	GYLYSQQSG	1	29NOV2009	RT	74.85	74.85	CCXCRZI)
53810	62381	GYGWWHQWW	1	24OCT2009	RT	1199.90	1199.85	CCXKXKR)

53811 rows × 9 columns

In [167]:
```
custtrans=pd.read_sql_query("SELECT * FROM xyzcust", engine)
```

In [168]:
```
custtrans.head()
```

Out[168]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	YTD_T
0	0	WDQQLLDQL	60084	5016	90.0	1	0.0	
1	1	WQWAYHYLA	60091	1750	4227.0	9	1263.0	
2	2	GSHAPLHAW	60067	900	420.0	3	129.0	
3	3	PGGYDYWAD	60068	3838	6552.0	6	0.0	
4	4	LWPSGPLLS	60090	3932	189.0	3	72.0	

In [169]:
```
allrttrans=pd.read_sql_query("SELECT * FROM xyztrans", engine)
```

In [170]:
```
allrttrans.head()
```

Out[170]:

	index	ACCTNO	QTY	TRANDATE	TRAN_CHANNEL	PRICE	TOTAMT	ORDERNO	DEPT
0	0	WGDQLA	1	09JUN2009	RT	599.85	599.85	CCXXNNXXXXUX	Hom
1	1	WGDQLA	1	09JUN2009	RT	39.00	39.00	CCXXNNXXXXUX	Ap
2	2	WGDQLA	1	28NOV2009	RT	15.00	15.00	CCXNXXKXXXRI	Ap
3	3	WGDQLA	1	28NOV2009	RT	69.00	69.00	CCXNXXKXXXRI	Ap
4	4	WGDQLA	1	28NOV2009	RT	84.00	84.00	CCXNXXKXXXRI	Ap

Requirements :

1. Get a list of all records in xyzcust table where YTD_SALES_2009 > 1000
2. Get a list of all records in xyzcust table where YTD_SALES_2009 > 1000 and CHANNEL_ACQUISITION = 'RT'
3. What is the total number of records in in xyzcust table where YTD_SALES_2009 > 1000, CHANNEL_ACQUISITION = 'RT', and ZIP = 60056

In [179]:
```
# Write your python code that meets the above requirements in this cell
# 1. Get a list of all records in xyzcust table where YTD_SALES_2009 > 1000
# Python Version
custtrans[custtrans['YTD_SALES_2009'] > 1000]
```

Out[179]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009
1	1	WQWAYHYLA	60091	1750	4227.0	9	1263.0
12	12	WLDAYHQLW	60091	2813	3240.0	7	2064.0
24	24	ASDHAYAW	60062	6077	3411.0	19	1875.0
31	31	HDWAWLH	60069	3402	25476.0	93	1623.0
40	40	GSHLHGHWW	60070	2352	3576.0	10	1398.0
...	...	...	...	...	...	...	...
30066	30358	LWWAWAPQD	60098	8091	21030.0	20	5322.0
30087	30379	AYQWWQLHY	60098	7943	4092.0	9	2625.0
30114	30406	WWQYYPSA	60098	3133	2100.0	3	1800.0
30116	30408	WLLWDLLYD	60098	7807	1827.0	2	1827.0
30162	30454	LLQLHHQYP	60098	7108	2184.0	3	1248.0

1633 rows × 11 columns

In [180]:
```
# Write your python code that meets the above requirements in this cell
# 1. Get a list of all records in xyzcust table where YTD_SALES_2009 > 1
000
# SQL Version
pd.read_sql_query('''SELECT *
                     FROM xyzcust
                     WHERE YTD_SALES_2009 > 1000''', engine)
```

Out[180]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	Y
0	1	WQWAYHYLA	60091	1750	4227.0	9	1263.0	
1	12	WLDAYHQLW	60091	2813	3240.0	7	2064.0	
2	24	ASDHAYAW	60062	6077	3411.0	19	1875.0	
3	31	HDWAWLH	60069	3402	25476.0	93	1623.0	
4	40	GSHLHGHWW	60070	2352	3576.0	10	1398.0	
...	...	...	...	...	...	...	...	
1628	30358	LWWAWAPQD	60098	8091	21030.0	20	5322.0	
1629	30379	AYQWWQLHY	60098	7943	4092.0	9	2625.0	
1630	30406	WWQYYPSA	60098	3133	2100.0	3	1800.0	
1631	30408	WLLWDLLYD	60098	7807	1827.0	2	1827.0	
1632	30454	LLQLHHQYP	60098	7108	2184.0	3	1248.0	

1633 rows × 11 columns

In [181]:
```
# 2.Get a list of all records in xyzcust table where YTD_SALES_2009 > 10
00 and CHANNEL_ACQUISITION = 'RT'
# Python Version
custtrans[(custtrans['YTD_SALES_2009'] > 1000) & (custtrans['CHANNEL_ACQ
UISITION'] == 'RT')]
```

Out[181]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009
1	1	WQWAYHYLA	60091	1750	4227.0	9	1263.0
12	12	WLDAYHQLW	60091	2813	3240.0	7	2064.0
24	24	ASDHAYAW	60062	6077	3411.0	19	1875.0
31	31	HDWAWLH	60069	3402	25476.0	93	1623.0
76	77	LGDGQPGDH	60061	4540	2364.0	17	1359.0
...	...	...	...	...	...	...	...
30044	30336	PLHHGGQYH	60098	8075	6681.0	16	2985.0
30066	30358	LWWAWAPQD	60098	8091	21030.0	20	5322.0
30087	30379	AYQWWQLHY	60098	7943	4092.0	9	2625.0
30116	30408	WLLWDLLYD	60098	7807	1827.0	2	1827.0
30162	30454	LLQLHHQYP	60098	7108	2184.0	3	1248.0

1207 rows × 11 columns

In [182]:
```
# 2.Get a list of all records in xyzcust table where YTD_SALES_2009 > 10
00 and CHANNEL_ACQUISITION = 'RT'
# SQL Version
pd.read_sql_query('''SELECT *
                     FROM xyzcust
                     WHERE YTD_SALES_2009 > 1000
                     AND  CHANNEL_ACQUISITION = 'RT' ''', engine)
```

Out[182]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	Y
0	1	WQWAYHYLA	60091	1750	4227.0	9	1263.0	
1	12	WLDAYHQLW	60091	2813	3240.0	7	2064.0	
2	24	ASDHAYAW	60062	6077	3411.0	19	1875.0	
3	31	HDWAWLH	60069	3402	25476.0	93	1623.0	
4	77	LGDGQPGDH	60061	4540	2364.0	17	1359.0	
...	...	...	...	...	...	...	...	
1202	30336	PLHHGGQYH	60098	8075	6681.0	16	2985.0	
1203	30358	LWWAWAPQD	60098	8091	21030.0	20	5322.0	
1204	30379	AYQWWQLHY	60098	7943	4092.0	9	2625.0	
1205	30408	WLLWDLLYD	60098	7807	1827.0	2	1827.0	
1206	30454	LLQLHHQYP	60098	7108	2184.0	3	1248.0	

1207 rows × 11 columns

In [183]:
```
# 3. What is the total number of records in in xyzcust table where YTD_S
ALES_2009 > 1000,
## CHANNEL_ACQUISITION = 'RT', and ZIP = 60056
# Python Version
custtrans[(custtrans['YTD_SALES_2009'] > 1000) & (custtrans['CHANNEL_ACQ
UISITION'] == 'RT') & (custtrans['ZIP'] == 60056)]
```

Out[183]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009
1002	1012	AGDDLWSWL	60056	2137	1806.0	5	1806.0
1249	1263	GLAPQGYWQ	60056	3610	2559.0	5	1164.0
3815	3847	WAAYQSSWL	60056	3122	5895.0	38	1863.0
3880	3912	WGQHYLAWY	60056	3217	3753.0	6	1926.0
4553	4592	ADGGWWHHL	60056	3707	5958.0	5	1677.0
5539	5586	APSGALYPL	60056	4343	2619.0	6	2271.0
5847	5896	SWYDPWSWH	60056	3657	1776.0	7	1536.0
7162	7235	LQQLHSHAQ	60056	3245	1689.0	4	1347.0
9490	9588	WGSLDLWL	60056	2120	1968.0	8	1467.0
10378	10484	PSGHWQADH	60056	2509	2319.0	2	1923.0
10802	10913	LGLGHPDGH	60056	1023	1248.0	4	1119.0
10857	10968	LAHHYYGSA	60056	0	5019.0	19	2151.0
11730	11850	ASHDGLSWP	60056	4344	5547.0	16	1776.0
11746	11866	WQLHLAYSG	60056	4309	13305.0	31	5094.0
12250	12376	ALQYDGSPH	60056	3425	1845.0	4	1692.0
12289	12415	GSWSSGWHD	60056	3258	1116.0	1	1116.0
12557	12686	LHLGAYDWA	60056	3614	8304.0	27	2643.0
12584	12714	SPGDQQQWA	60056	1705	1764.0	1	1764.0
12625	12755	WHDPHDLAA	60056	2946	3024.0	10	1107.0
13488	13630	WYGYHGLDL	60056	2936	5292.0	22	1575.0
13990	14137	SWQSGYPPP	60056	3500	15912.0	21	2706.0
14026	14173	WLSYAQPWQ	60056	3515	4668.0	19	1311.0
14028	14175	SAGHLDPAQ	60056	2872	2685.0	23	1272.0
14431	14580	AWSWGGWSL	60056	2543	3165.0	6	1863.0
14948	15099	LQQGYGQSL	60056	2050	29448.0	104	2475.0
14979	15131	PYSAQAGPP	60056	3746	1326.0	1	1326.0
16497	16665	SLGHWGWLD	60056	2932	3402.0	11	1941.0
17080	17252	GSQHWPHAS	60056	6068	1440.0	1	1440.0
17324	17498	WHHQSDDLL	60056	3037	1347.0	8	1008.0
17752	17930	ASYHLAAHS	60056	4157	5160.0	19	1638.0
17984	18164	GADHALQQ	60056	3432	6300.0	8	5559.0
18938	19124	WPWSQGQWD	60056	2940	7788.0	29	2334.0
19203	19390	WLQSYDPSL	60056	2312	3786.0	21	1149.0
20671	20872	GLASHHDLS	60056	2553	1281.0	1	1281.0

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009
21380	21590	PHYSLSAH	60056	4568	1587.0	6	1188.0
21905	22119	AQQQHSLAG	60056	3427	3123.0	19	2022.0
22043	22259	GYLQQHLQA	60056	3278	1302.0	2	1302.0
23295	23522	GGPGSGQDD	60056	3615	3384.0	4	3384.0
23534	23762	HALADGHQ	60056	4022	3786.0	12	1512.0
23573	23801	WLGPQAGAQ	60056	1508	2307.0	9	1053.0
23602	23832	PAAYSPGYD	60056	3540	1209.0	4	1074.0
24347	24587	AWHYDGDGY	60056	1669	2757.0	10	1926.0
25026	25272	PPQWGWDAL	60056	3743	2682.0	7	2451.0
25039	25286	PAQSHYALD	60056	2209	3048.0	7	2130.0
25560	25810	WQWAAYADP	60056	3659	3249.0	8	2298.0
26154	26412	LWAQGAQQW	60056	1028	2265.0	11	1362.0
26671	26936	LGGGQYYGL	60056	3807	4305.0	11	1428.0
26970	27239	WWHDQYLYA	60056	3013	2862.0	8	2259.0
28203	28482	LHHSLAASQ	60056	3426	10995.0	60	4683.0

In [184]:
```
# 3. What is the total number of records in in xyzcust table where YTD_S
ALES_2009 > 1000,
## CHANNEL_ACQUISITION = 'RT', and ZIP = 60056
# SQL Version
pd.read_sql_query(''' SELECT *
                  FROM xyzcust
                  WHERE YTD_SALES_2009 > 1000
                  AND  CHANNEL_ACQUISITION = 'RT'
                  AND ZIP = 60056 ''', engine)
```

Out[184]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	YTI
0	1012	AGDDLWSWL	60056	2137	1806.0	5	1806.0	
1	1263	GLAPQGYWQ	60056	3610	2559.0	5	1164.0	
2	3847	WAAYQSSWL	60056	3122	5895.0	38	1863.0	
3	3912	WGQHYLAWY	60056	3217	3753.0	6	1926.0	
4	4592	ADGGWWHHL	60056	3707	5958.0	5	1677.0	
5	5586	APSGALYPL	60056	4343	2619.0	6	2271.0	
6	5896	SWYDPWSWH	60056	3657	1776.0	7	1536.0	
7	7235	LQQLHSHAQ	60056	3245	1689.0	4	1347.0	
8	9588	WGSLDLWL	60056	2120	1968.0	8	1467.0	
9	10484	PSGHWQADH	60056	2509	2319.0	2	1923.0	
10	10913	LGLGHPDGH	60056	1023	1248.0	4	1119.0	
11	10968	LAHHYYGSA	60056	0	5019.0	19	2151.0	
12	11850	ASHDGLSWP	60056	4344	5547.0	16	1776.0	
13	11866	WQLHLAYSG	60056	4309	13305.0	31	5094.0	
14	12376	ALQYDGSPH	60056	3425	1845.0	4	1692.0	
15	12415	GSWSSGWHD	60056	3258	1116.0	1	1116.0	
16	12686	LHLGAYDWA	60056	3614	8304.0	27	2643.0	
17	12714	SPGDQQQWA	60056	1705	1764.0	1	1764.0	
18	12755	WHDPHDLAA	60056	2946	3024.0	10	1107.0	
19	13630	WYGYHGLDL	60056	2936	5292.0	22	1575.0	
20	14137	SWQSGYPPP	60056	3500	15912.0	21	2706.0	
21	14173	WLSYAQPWQ	60056	3515	4668.0	19	1311.0	
22	14175	SAGHLDPAQ	60056	2872	2685.0	23	1272.0	
23	14580	AWSWGGWSL	60056	2543	3165.0	6	1863.0	
24	15099	LQQGYGQSL	60056	2050	29448.0	104	2475.0	
25	15131	PYSAQAGPP	60056	3746	1326.0	1	1326.0	
26	16665	SLGHWGWLD	60056	2932	3402.0	11	1941.0	
27	17252	GSQHWPHAS	60056	6068	1440.0	1	1440.0	
28	17498	WHHQSDDLL	60056	3037	1347.0	8	1008.0	
29	17930	ASYHLAAHS	60056	4157	5160.0	19	1638.0	
30	18164	GADHALQQ	60056	3432	6300.0	8	5559.0	
31	19124	WPWSQGQWD	60056	2940	7788.0	29	2334.0	
32	19390	WLQSYDPSL	60056	2312	3786.0	21	1149.0	
33	20872	GLASHHDLS	60056	2553	1281.0	1	1281.0	

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SALES_2009	YTI
34	21590	PHYSLSAH	60056	4568	1587.0	6	1188.0	
35	22119	AQQQHSLAG	60056	3427	3123.0	19	2022.0	
36	22259	GYLQQHLQA	60056	3278	1302.0	2	1302.0	
37	23522	GGPGSGQDD	60056	3615	3384.0	4	3384.0	
38	23762	HALADGHQ	60056	4022	3786.0	12	1512.0	
39	23801	WLGPQAGAQ	60056	1508	2307.0	9	1053.0	
40	23832	PAAYSPGYD	60056	3540	1209.0	4	1074.0	
41	24587	AWHYDGDGY	60056	1669	2757.0	10	1926.0	
42	25272	PPQWGWDAL	60056	3743	2682.0	7	2451.0	
43	25286	PAQSHYALD	60056	2209	3048.0	7	2130.0	
44	25810	WQWAAYADP	60056	3659	3249.0	8	2298.0	
45	26412	LWAQGAQQW	60056	1028	2265.0	11	1362.0	
46	26936	LGGGQYYGL	60056	3807	4305.0	11	1428.0	
47	27239	WWHDQYLYA	60056	3013	2862.0	8	2259.0	
48	28482	LHHSLAASQ	60056	3426	10995.0	60	4683.0	