A single PDF document that has the following 7 screen-shots: 1. sqlite3 command-window shows only one table before you run the Ipython Notebook script 2. First Run for the Ipython Notebook script with no error 3. sqlite3 command-windows shows two tables after you run the Ipython Notebook script 4. sqlite3 command-window shows only one table before you run the Ipython Notebook script 5. Second Run for the Ipython Notebook script that shows error 6. sqlite3 command-window shows drop table for xyzcust 7. Third Run for the Ipython Notebook script that shows NO error

Week #3 - Python Practice

Where we left off in the last session is with some questions about XYZ's customer data. We pickled the data we had input from a csv file. (We did, right?) Let's get started again in Canopy and load our pickled data. The first thing to do is to import pandas, numpy, DataFrame from pandas, and cPickle into Python. Remember the nicknames we gave some of them when we imported them? I'm talking about pd, np, and pickle. They're optional, but they're convenient and also used by many Python programmers. Anyway, assuming that you stored your XYZ customer data in a pickle file called xyzcust10.p, you can unpickle it into Python like:

In [83]:
```
import os

import cPickle as 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 [84]:
```
xyzcust10=pd.read_csv('xyzcust10.csv')
```

In [85]:
```
(xyzcust10).dtypes
```

Out[85]:
```
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 [86]:
```
type(xyzcust10)
```

Out[86]:
```
pandas.core.frame.DataFrame
```

In [87]:
```
pickle.dump(xyzcust10,open('xyzcust10.p','wb'))
```

In [88]:
```
xyzcust10=pickle.load(open('xyzcust10.p','rb')) # note the cPickle nic
kname

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 [89]:
```
type(xyzcust10)
```

Out[89]:
```
pandas.core.frame.DataFrame
```

How many records are there in xyzcust10? In the last session we noted that 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 [90]:
```
xyzcust10.columns
```

Out[90]:
```
Index([u'ACCTNO', u'ZIP', u'ZIP4', u'LTD_SALES', u'LTD_TRANSACTIONS'
,
       u'YTD_SALES_2009', u'YTD_TRANSACTIONS_2009', u'CHANNEL_ACQUIS
ITION',
       u'BUYER_STATUS', u'ZIP9_Supercode', u'ZIP9_SUPERCODE'],
      dtype='object')
```

In [91]:
```
xyzcust10.dtypes
```

Out[91]:
```
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 [92]:
```
(xyzcust10.ZIP9_Supercode!=xyzcust10.ZIP9_SUPERCODE).sum()
```

Out[92]:
```
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 [93]:
```
xyzcust10['ZIP9_Supercode']
```

Out[93]:
```
0        600845016
1        600911750
2        600670900
3        600683838
4        600903932
5        600858670
6        600913447
7        600911613
8        600683668
9        600911759
10       600818325
11       600562960
12       600912813
13       600673528
14       600603209
```

```
15       600891326
16       600692129
17       600911453
18       600682219
19       600624628
20       600912346
21       600614527
22       600612123
23       600894622
24       600626077
25       600818248
26       600932706
27       600623210
28       600933840
29       600905705
           ...
30441    600987410
30442    600987615
30443    600988020
30444    600988426
30445    600988550
30446    600987893
30447    600987977
30448    600987805
30449    600988014
30450    600988671
30451    600988128
30452    600988760
30453    600988093
30454    600987108
30455    600987552
30456        60098
30457    600989172
30458    600988958
30459    600989029
30460    600987869
30461    600982556
30462    600980142
30463    600982857
30464    600983342
30465    600987858
30466    600983951
30467    600989681
30468    600983858
30469    600987927
30470    600984160
Name: ZIP9_Supercode, 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 [94]:
```
del xyzcust10['ZIP9_Supercode']
del xyzcust10rev1['ZIP9_Supercode']
```

or

In [95]:
```
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 on 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 Canopy. Your default directory would be a good bet. Remember what it is? See what os.getcwd() tells you.

In [96]:
```
os.getcwd()
```

Out[96]:
```
'/Users/Zeeshan/Desktop/PREDICT 420/Week 3/Sync Session 3-5/Exercise
Practice3'
```

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 Canopy. (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 [97]:
```
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 [98]:
```
from sqlalchemy import create_engine
```

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

In [99]:
```
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 [100]:
```
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 [101]:
```
xyztrans.dtypes
```

Out[101]:
```
index             int64
ACCTNO           object
QTY               int64
TRANDATE         object
TRAN_CHANNEL     object
PRICE           float64
TOTAMT          float64
ORDERNO          object
DEPTDESCR        object
dtype: object
```

or

In [102]:
```
xyztrans.columns
```

Out[102]:
```
Index([u'index', u'ACCTNO', u'QTY', u'TRANDATE', u'TRAN_CHANNEL', u'
PRICE',
       u'TOTAMT', u'ORDERNO', u'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 [103]:
```
from sqlalchemy import schema
```

In [104]:
```
xyzMetaData=schema.MetaData(bind=engine, reflect=True)
```

```
/Users/Zeeshan/Library/Enthought/Canopy_64bit/User/lib/python2.7/sit
e-packages/ipykernel/__main__.py:1: SADeprecationWarning: reflect=Tr
ue is deprecate; please use the reflect() method.
  if __name__ == '__main__':
```

In [105]:
```
xyzMetaData.tables
```

Out[105]:
```
immutabledict({u'xyztrans': Table('xyztrans', MetaData(bind=Engine(s
qlite:///xyz.db)), Column('index', BIGINT(), table=<xyztrans>), Colu
mn('ACCTNO', TEXT(), table=<xyztrans>), Column('QTY', BIGINT(), tabl
e=<xyztrans>), Column('TRANDATE', TEXT(), table=<xyztrans>), Column(
'TRAN_CHANNEL', TEXT(), table=<xyztrans>), Column('PRICE', FLOAT(),
table=<xyztrans>), Column('TOTAMT', FLOAT(), table=<xyztrans>), Colu
mn('ORDERNO', TEXT(), table=<xyztrans>), Column('DEPTDESCR', TEXT(),
table=<xyztrans>), schema=None)})
```

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

```
In [106]: xyzMetaData.tables.keys()
```

```
Out[106]: [u'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 [107]: xyzcust10rev1.duplicated().sum()
```

```
Out[107]: 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 [108]: xyzcustUnDup=xyzcust10rev1.drop_duplicates()

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

```
Out[108]: 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 [109]: xyzcust10rev1.duplicated('ACCTNO').sum()
```

```
Out[109]: 292
```

```
In [110]: xyzcust10rev1.ACCTNO.duplicated().sum()
```

```
Out[110]: 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 [111]: xyzcustUnDup.to_sql('xyzcust', engine)
```

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

In [112]:
```
pd.read_sql_table('xyzcust', engine).columns
```

Out[112]:
```
Index([u'index', u'ACCTNO', u'ZIP', u'ZIP4', u'LTD_SALES', u'LTD_TRA
NSACTIONS',
       u'YTD_SALES_2009', u'YTD_TRANSACTIONS_2009', u'CHANNEL_ACQUIS
ITION',
       u'BUYER_STATUS', u'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 [113]:
```
xyzMetaData.tables.keys()
```

Out[113]:
```
[u'xyztrans']
```

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

In [114]:
```
xyzMetaData
```

Out[114]:
```
MetaData(bind=Engine(sqlite:///xyz.db))
```

In [115]:
```
from sqlalchemy import inspect
```

In [116]:
```
insp=inspect(engine)
```

In [117]:
```
 insp.get_table_names()
```

Out[117]:
```
[u'xyzcust', u'xyztrans']
```

Do you think there are any duplicates in the order transaction data? If so, what would you make of them? We're done for now, so save everything you need to save. You'll be using some of the data you worked with in this Practice in the next session's practice, where we'll be reading data in yet another format, merging data sets in various ways, and continuing to clean up various and sundry problems with XYZ's data. We'll also be starting to look at some descriptive statistics about the dataYou 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 [118]:
```
rttrans=pd.read_sql_query("SELECT * FROM xyztrans WHERE TRAN_CHANNEL='
RT'", engine)
```

to get a DataFrame, rttrans, that has only the retail transactions in xyztrans. How many retail transactions did you find? What proportion of the total number of transactions are they? 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 [119]:
```
rttrans
```

Out[119]:

	index	ACCTNO	QTY	TRANDATE	TRAN_CHANNEL	PRICE	TOTAMT
0	0	WGDQLA	1	09JUN2009	RT	599.85	599.85
1	1	WGDQLA	1	09JUN2009	RT	39.00	39.00
2	2	WGDQLA	1	28NOV2009	RT	15.00	15.00
3	3	WGDQLA	1	28NOV2009	RT	69.00	69.00
4	4	WGDQLA	1	28NOV2009	RT	84.00	84.00
5	5	WGDQLA	1	28NOV2009	RT	69.00	69.00
6	6	WGDQLA	1	28NOV2009	RT	89.85	89.85
7	7	WGDQLA	1	28NOV2009	RT	119.85	119.85
8	8	APSYYW	1	07JUN2009	RT	22.50	22.50
9	9	APSYYW	1	07JUN2009	RT	44.85	44.85
10	10	APSYYW	1	07JUN2009	RT	30.00	30.00
11	11	APSYYW	1	07JUN2009	RT	30.00	30.00
12	13	GGDWGY	1	14SEP2009	RT	239.85	239.85
13	14	GGDWGY	1	18DEC2009	RT	234.00	234.00
14	15	HHSSAL	1	13SEP2009	RT	66.00	66.00

15	16	HHSSAL	1	13SEP2009	RT	66.00	66.00
16	17	HHSSAL	1	13SEP2009	RT	38.25	38.25
17	18	HHSSAL	1	13SEP2009	RT	28.50	28.50
18	19	HHSSAL	1	13SEP2009	RT	43.50	43.50
19	20	HHSSAL	1	13SEP2009	RT	24.00	24.00
20	21	HHSSAL	1	13SEP2009	RT	42.00	42.00
21	22	HHSSAL	1	13SEP2009	RT	38.85	38.85
22	23	HHSSAL	1	13SEP2009	RT	105.00	105.00
23	24	HHSSAL	1	13SEP2009	RT	30.00	30.00
24	25	HHSSAL	1	13SEP2009	RT	32.85	32.85
25	26	HHSSAL	1	13SEP2009	RT	84.00	84.00
26	27	HHSSAL	1	18DEC2009	RT	28.50	28.50
27	28	HHSSAL	1	18DEC2009	RT	43.50	43.50

28	29	HHSSAL	1	18DEC2009	RT	27.00	27.00
29	30	HHSSAL	1	18DEC2009	RT	31.50	31.50
...	...	...	...	...	...	...	...
53781	62350	GYLAPPYPQ	1	11OCT2009	RT	59.85	59.85
53782	62351	GYLAPPYPQ	1	11OCT2009	RT	126.00	126.00
53783	62352	GYLAPPYPQ	1	11OCT2009	RT	81.00	81.00
53784	62353	GYLAPPYPQ	1	11OCT2009	RT	36.00	36.00
53785	62354	GYLAPPYYW	1	10OCT2009	RT	31.50	31.50
53786	62355	GYLPADYQL	1	14OCT2009	RT	59.85	59.85
53787	62356	GYLPADYQL	1	14OCT2009	RT	36.00	36.00
53788	62357	GYLPADYQL	1	14OCT2009	RT	72.00	72.00
53789	62358	GYLPADYQL	1	14OCT2009	RT	72.00	72.00
53790	62359	GYLPADYQL	1	14OCT2009	RT	27.00	27.00
53791	62360	GYLPADYQL	1	14OCT2009	RT	48.00	48.00
53792	62361	GYLPADYQL	1	14OCT2009	RT	66.00	66.00
53793	62362	GYLPADYQL	1	14OCT2009	RT	57.00	57.00
53794	62364	GYLHWWQGW	1	21NOV2009	RT	36.00	36.00

53795	62365	GYLHWWQGW	1	21NOV2009	RT	30.00	30.00
53796	62366	GYLHWWQGW	1	21NOV2009	RT	28.50	28.50
53797	62367	GYLHWWQGW	1	21NOV2009	RT	54.00	54.00
53798	62368	GYLHWWQGW	1	21NOV2009	RT	28.50	28.50
53799	62369	GYLYSQQSG	1	27NOV2009	RT	27.00	27.00
53800	62370	GYLYSQQSG	1	27NOV2009	RT	45.00	45.00
53801	62371	GYLYSQQSG	1	27NOV2009	RT	74.85	74.85
53802	62372	GYLYSQQSG	1	21NOV2009	RT	62.64	62.64
53803	62373	GYLYSQQSG	1	21NOV2009	RT	299.85	299.85
53804	62374	GYLYSQQSG	1	29OCT2009	RT	299.85	299.85
53805	62375	GYLYSQQSG	1	14NOV2009	RT	32.85	32.85
53806	62376	GYLYSQQSG	1	14NOV2009	RT	45.00	45.00
53807	62377	GYLYSQQSG	1	14NOV2009	RT	15.00	15.00
53808	62378	GYLYSQQSG	1	29NOV2009	RT	42.00	42.00
53809	62379	GYLYSQQSG	1	29NOV2009	RT	74.85	74.85
53810	62381	GYGWWHQWW	1	24OCT2009	RT	1199.90	1199.85

53811 rows × 9 columns

In [120]:
```
custtrans=pd.read_sql_query("SELECT * FROM xyzcust", engine)
```

In [121]:
```
custtrans
```

Out[121]:

	index	ACCTNO	ZIP	ZIP4	LTD_SALES	LTD_TRANSACTIONS	YTD_SA
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
5	5	LQGYDGSYQ	60085	8670	4278.0	12	102.0
6	6	WGQWQDPDA	60091	3447	1869.0	5	495.0
7	7	LPASPGYLS	60091	1613	33.0	1	33.0
8	8	GPGDSHGL	60068	3668	735.0	2	0.0
9	9	PQHSWQSDQ	60091	0	468.0	2	0.0
10	10	AGDDPGGQP	60081	8325	804.0	8	57.0
11	11	WDSYWHWDP	60056	0	219.0	4	0.0
12	12	WLDAYHQLW	60091	2813	3240.0	7	2064.0
13	13	AYWWADPYG	60067	3528	180.0	1	0.0
14	14	SPGWSWDGP	60060	3209	423.0	4	99.0
15	15	ADQAPGPYH	60089	1326	306.0	2	0.0
16	16	PLWAYQHQL	60069	2129	1002.0	2	0.0
17	17	HWPPYQWS	60091	1453	1155.0	4	36.0
18	18	SQLYGSPQD	60068	2219	612.0	3	162.0
19	19	APAHLSLPD	60062	4628	633.0	4	345.0
20	20	WPDYADQLS	60091	2346	114.0	1	0.0
21	21	PPLQHQWDG	60061	4527	294.0	1	294.0
22	22	WYQGYGYWY	60061	2123	849.0	4	0.0
23	23	WGPQDWASS	60089	4622	72.0	1	0.0

24	24	ASDHAYAW	60062	6077	3411.0	19	1875.0
25	25	PDAYLGDGY	60081	8248	1023.0	6	663.0
26	26	WYHDHLDLY	60093	2706	873.0	4	0.0
27	27	PQDAAPPDQ	60062	3210	2778.0	13	726.0
28	28	WWGAQQHH	60093	3840	2676.0	13	678.0
29	29	SALLQHHGA	60090	5705	528.0	5	0.0
...	...	...	...	...	...	...	...
30149	30441	GWDSDHQAD	60098	7410	861.0	2	84.0
30150	30442	WSSGDWWLP	60098	7615	837.0	5	198.0
30151	30443	PHSWLADGL	60098	8020	2478.0	12	96.0
30152	30444	GLHAQQGLL	60098	8426	84.0	1	84.0
30153	30445	ADWLADSW	60098	8550	2877.0	9	735.0
30154	30446	AQQPHADGH	60098	7893	1611.0	3	0.0
30155	30447	WGSLSYSW	60098	0	1860.0	8	0.0
30156	30448	SLYWGGSDL	60098	7805	48.0	1	0.0
30157	30449	WHSLPGWSG	60098	0	195.0	1	195.0
30158	30450	LQWSLYSDP	60098	8671	60.0	1	0.0
30159	30451	SHASPYYPH	60098	0	252.0	2	0.0
30160	30452	LGPALDLDG	60098	8760	594.0	3	0.0
30161	30453	GDDAPHS	60098	8093	1272.0	7	0.0
30162	30454	LLQLHHQYP	60098	7108	2184.0	3	1248.0
30163	30455	PSQLLSAQG	60098	7552	759.0	2	231.0
30164	30456	LGPYGQAAD	60098	0	756.0	1	0.0
30165	30457	DSHAHSSH	60098	9172	1365.0	6	213.0
30166	30458	WSWASDDH	60098	8958	2490.0	14	435.0
30167	30459	GYPSYHAAY	60098	9029	438.0	2	438.0
30168	30460	SGDPPYQLL	60098	7869	549.0	4	0.0
30169	30461	LSQSHQYPQ	60098	2556	150.0	1	150.0
30170	30462	GGQDYSHPS	60098	142	93.0	1	93.0

30171	30463	LYWPPPGSL	60098	0	834.0	2	0.0
30172	30464	WAPPQLYQP	60098	0	147.0	2	0.0
30173	30465	AYDSQWQWA	60098	0	816.0	4	0.0
30174	30466	SYDQYLSWH	60098	3951	2736.0	9	96.0
30175	30467	SAPDQHQLP	60098	9681	2412.0	8	108.0
30176	30468	SASYAPDSY	60098	0	429.0	1	0.0
30177	30469	PWQPDWHA	60098	7927	651.0	1	0.0
30178	30470	SQQHDYHWH	60098	4160	4527.0	16	672.0

30179 rows × 11 columns

In [122]:
```
allrttrans=pd.read_sql_query("SELECT * FROM xyztrans", engine)
```

In [123]:
```
allrttrans
```

Out[123]:

	index	ACCTNO	QTY	TRANDATE	TRAN_CHANNEL	PRICE	TOTAMT
0	0	WGDQLA	1	09JUN2009	RT	599.85	599.85
1	1	WGDQLA	1	09JUN2009	RT	39.00	39.00
2	2	WGDQLA	1	28NOV2009	RT	15.00	15.00
3	3	WGDQLA	1	28NOV2009	RT	69.00	69.00
4	4	WGDQLA	1	28NOV2009	RT	84.00	84.00
5	5	WGDQLA	1	28NOV2009	RT	69.00	69.00
6	6	WGDQLA	1	28NOV2009	RT	89.85	89.85
7	7	WGDQLA	1	28NOV2009	RT	119.85	119.85
8	8	APSYYW	1	07JUN2009	RT	22.50	22.50
9	9	APSYYW	1	07JUN2009	RT	44.85	44.85

10	10	APSYYW	1	07JUN2009	RT	30.00	30.00
11	11	APSYYW	1	07JUN2009	RT	30.00	30.00
12	12	SDHLPH	1	18JAN2009	CB	477.00	477.00
13	13	GGDWGY	1	14SEP2009	RT	239.85	239.85
14	14	GGDWGY	1	18DEC2009	RT	234.00	234.00
15	15	HHSSAL	1	13SEP2009	RT	66.00	66.00
16	16	HHSSAL	1	13SEP2009	RT	66.00	66.00
17	17	HHSSAL	1	13SEP2009	RT	38.25	38.25
18	18	HHSSAL	1	13SEP2009	RT	28.50	28.50
19	19	HHSSAL	1	13SEP2009	RT	43.50	43.50
20	20	HHSSAL	1	13SEP2009	RT	24.00	24.00
21	21	HHSSAL	1	13SEP2009	RT	42.00	42.00
22	22	HHSSAL	1	13SEP2009	RT	38.85	38.85
23	23	HHSSAL	1	13SEP2009	RT	105.00	105.00

24	24	HHSSAL	1	13SEP2009	RT	30.00	30.00
25	25	HHSSAL	1	13SEP2009	RT	32.85	32.85
26	26	HHSSAL	1	13SEP2009	RT	84.00	84.00
27	27	HHSSAL	1	18DEC2009	RT	28.50	28.50
28	28	HHSSAL	1	18DEC2009	RT	43.50	43.50
29	29	HHSSAL	1	18DEC2009	RT	27.00	27.00
...	...	...	...	...	...	...	...
62365	62365	GYLHWWQGW	1	21NOV2009	RT	30.00	30.00
62366	62366	GYLHWWQGW	1	21NOV2009	RT	28.50	28.50
62367	62367	GYLHWWQGW	1	21NOV2009	RT	54.00	54.00
62368	62368	GYLHWWQGW	1	21NOV2009	RT	28.50	28.50
62369	62369	GYLYSQQSG	1	27NOV2009	RT	27.00	27.00
62370	62370	GYLYSQQSG	1	27NOV2009	RT	45.00	45.00
62371	62371	GYLYSQQSG	1	27NOV2009	RT	74.85	74.85

62372	62372	GYLYSQQSG	1	21NOV2009	RT	62.64	62.64
62373	62373	GYLYSQQSG	1	21NOV2009	RT	299.85	299.85
62374	62374	GYLYSQQSG	1	29OCT2009	RT	299.85	299.85
62375	62375	GYLYSQQSG	1	14NOV2009	RT	32.85	32.85
62376	62376	GYLYSQQSG	1	14NOV2009	RT	45.00	45.00
62377	62377	GYLYSQQSG	1	14NOV2009	RT	15.00	15.00
62378	62378	GYLYSQQSG	1	29NOV2009	RT	42.00	42.00
62379	62379	GYLYSQQSG	1	29NOV2009	RT	74.85	74.85
62380	62380	GYGWWHQWW	1	19OCT2009	IB	1199.90	1199.85
62381	62381	GYGWWHQWW	1	24OCT2009	RT	1199.90	1199.85
62382	62382	GYGSGDLAW	1	21OCT2009	IB	899.85	899.85
62383	62383	GYGSGDLAW	1	10NOV2009	IB	899.85	899.85
62384	62384	GYGPWHDQY	1	22OCT2009	IB	89.85	89.85
62385	62385	GYGHSSSWA	1	24OCT2009	IB	59.85	59.85
62386	62386	GYGYWSHGH	1	26OCT2009	IB	78.00	78.00
62387	62387	GYGYWSHGH	2	24NOV2009	IB	99.00	198.00
62388	62388	GYHAAPHYA	1	30OCT2009	IB	300.00	300.00
62389	62389	GYHAAPHYA	1	30OCT2009	IB	225.00	225.00

62390	62390	GYHAAPHYA	1	15DEC2009	IB	300.00	300.00
62391	62391	GYHAAPHYA	1	15DEC2009	IB	300.00	300.00
62392	62392	GYHAAPHYA	1	15DEC2009	IB	300.00	300.00
62393	62393	GYHPWGWPD	1	31OCT2009	IB	150.00	150.00
62394	62394	GYHPAWAPY	1	31OCT2009	IB	96.00	96.00

62395 rows × 9 columns

Deliverable:

In [ ]: