Noé Flores

Introduction:
Our data consists of three distinct customer datasets on information collected by xyz company. These datasets will be utilized to create new succinct and concise data-tables based on factors such as buying habits, credit cards used, gender, and activity of the customer.

Through the use of the following code, "**psql -h spspostgresql - "** in a GNOME terminal, we connect to locate our files labeled item, mail, and customer, and copy the files to our specified directory. Finally, we use Cyberduck to copy the files to our local PC where they are now ready to be extracted for use.

Part 1:Import csv files downloaded from the SSCC into a pandas DataFrame:
We use the pandas .csv reader to download the files

```
####Readin Files###
item_df = pd.read_csv('/Users/Noe/Desktop/)
mail_df = pd.read_csv('/Users/Noe/Desktop/)
customer_df = pd.read_csv('/Users/Noe/Desktop/)
```

We can now verify the contents of the "item_df "to ensure our data was imported correctly.

```
item_df.columns
Out[43]:
Index(['acctno', 'qty', 'trandate', 'tran_channel', 'price', 'totamt',
       'orderno', 'deptdescr'],
      dtype='object')
item_df.head(4)
Out[44]:
   acctno  qty    trandate tran_channel   price  totamt       orderno     deptdescr
0   GGGSD    1  2009-11-18           RT   93.60   93.60  CCXUKCIXXXKI     Portable Electronics
1   GGGSD    1  2009-11-19           RT  213.60  213.60  CCXURVCXXXKI     Portable Electronics
2   GGGSD    1  2009-11-19           RT   93.60   93.60  CCXURVCXXXKI     Portable Electronics
3  WGDQLA    1  2009-06-09           RT  599.85  599.85  CCXXNNXXXXUX     Home Audio
```

The data located in the item_df data-frame consists of objects that contains account information, transaction dates, item descriptions, price information, and total amount spent on a particular "type" of item.

Part 2: Write DataFrames to a local SQLite DB named xyz.db

We want to get the corresponding DataFrames of active customers only. We begin the process by creating a temporary DataFrame called "active_customer". This was initiated to make a clean DataFrame from which to extract and merge the data for the other DataFrames.

```
###Get the corresponding data frames for active customers across all three csv's
active_customer = customer_df[['acctno','buyer_status']]
```

From active_customer, we can use pd.merge to create the new DataFrames of most active buyers.

```
###Create new DataFrame for specific active_customers.
active_customer_df = active_customer[active_customer.buyer_status == "ACTIVE"]
active_customer_df.shape
active_customer_df.head(4)
###Create new DataFrame for specific active_mail.
active_mail_df = pd.merge(active_customer_df,mail_df,on=["acctno"])
active_mail_df.shape
active_mail_df.head(4)
###Create new DataFrame for specific active_item
active_item_df = pd.merge(active_customer_df,item_df,on=["acctno"])
active_item_df.shape
active_item_df.head(4)
```

Write active_customer, active_item and active_mail to a local SQLite DB named db:
In order to write the nex active customer DataFrames to SQLite, we import "sqlalchemy" to help us write the tables to the new xyz.db local SQLite DB.

```
import sqlalchemy
```

```
from sqlalchemy import create_engine
###Write active customer mail and item to SQLite DB
engine=create_engine('sqlite:///xyz.db')
conn=engine.connect()
active_customer_df.to_sql('active_customer',conn,index=False)
active_mail_df.to_sql('active_mail',conn,index=False)
active_item_df.to_sql('active_item',conn,index=False)
```

Once that process is complete, we can use "**ls**" **to help us identify if the process was successful**.

```
ls
 Volume in drive C has no label.
 Volume Serial Number is C6B1-E803
 Directory of C:\Users\Noe
10/21/2017  07:45 PM    <DIR>          .matplotlib
10/29/2017  02:23 PM         7,915,520  xyz.db
```

We can now query the table_names in the xyz.db as a final verification. as you can see below, the three new active customer tables have been successfully saved to the xyz.db

```
engine.table_names()
Out[37]: ['active_customer', 'active_item', 'active_mail']
```

Part 3:Create a new DataFrame, custSum, as follows:

To create a summary DataFrame of all the most active buyers, we begin by creating a test DataFrame called "custsum1". This DataFrame is created through a pd.merge of the new "active_customer" DataFrame and the Original "customer_df" DataFrame, selecting only those customer accounts with "ACTIVE" status.

```
###For the second part of the progress report create a new DataFrame, custSum, as follows
custsum1 = pd.merge(customer_df,active_customer_df[active_customer.buyer_status == "ACTIVE"])
```

We need to add a new column, called 'heavy_buyer' to our "**custsum1**" DataFrame with 'Y' and 'N' indicator for 'heavy buyer'status. There the definition of a 'heavy buyer' is a customer whose YTD purchasing in 2009 ('**ytd_sales_2009**') is greater than 90% of the 2009 YTD purchasing of all customers who are 'ACTIVE' buyers. We begin by creating the function to differentiate the heavy buyers through the use of "If" then "return" rules.

```
###Create new function
def encode(ytd_sales_2009):
    if ytd_sales_2009 >= custsum1.ytd_sales_2009.quantile(.90):
        return 'Y'
    if ytd_sales_2009 <= custsum1.ytd_sales_2009.quantile(.90):
        return 'N'
###Check
custsum1.ytd_sales_2009.map(encode).head(14)
```

Once the function is complete, we can insert the new column to our "custsum1" DataFrame

```
###Add new heavy buyer column
custsum1.insert(9,'heavy_buyer',custsum1.ytd_sales_2009.map(encode))
pd.crosstab(custsum1.heavy_buyer,custsum1.ytd_sales_2009)
```

After adding the column, we can cross-tabulate to ensure that our process worked correctly

```
pd.crosstab(custsum1.heavy_buyer,custsum1.ytd_sales_2009, margins=True)
Out[20]:
ytd_sales_2009  3   6   9  12  15  18  21  24  27   30  ...    12393  12684  \
heavy_buyer                                              ...
N               2  10  15  19  41  39  39  32  75  170  ...        0      0
Y               0   0   0   0   0   0   0   0   0    0  ...        1      1
All             2  10  15  19  41  39  39  32  75  170  ...        1      1

ytd_sales_2009  13149  14448  14817  15273  17460  28002  351000    All
heavy_buyer
N                   0      0      0      0      0      0       0  15737
Y                   1      1      1      1      1      1       1   1754
All                 1      1      1      1      1      1       1  17491
```

We similarly want to add columns that indicates if a buyer has an American Express (AMEX), Discover Card (DISC), Visa Card (VISA), or a Master Cars (MC). In our custsum1 database, we have columns that indicate if a customer is in possession of a regular issue credit card, a premium card, or both. We will identify the customers who are in possession of either card. And assign a "Y" for yes if they are in possession of the specified card, or a "U" for no or unknown if they are not in possession of the specific card. The identification is made through the use of np.where in pandas.

```
###Create new function for AMEXPREM
custsum1['has_AMEX'] = np.where((custsum1.amex_reg == 'Y') | (custsum1.amex_prem == 'Y'), 'Y', 'U')
###Create new function for DISCOVER
custsum1['has_DISC'] = np.where((custsum1.disc_reg == 'Y') | (custsum1.disc_prem == 'Y'), 'Y', 'U')
###Create new function for MCREG
custsum1['has_MC'] = np.where((custsum1.mc_reg == 'Y') | (custsum1.mc_prem == 'Y'), 'Y', 'U')
###Create new function for VISAREG
custsum1['has_VISA'] = np.where((custsum1.visa_reg == 'Y') | (custsum1.visa_prem == 'Y'), 'Y', 'U')
```

We can no verify that the values were assigned correctly to our new has_AMEX, has_DISC, has_MC, and has_VISA columns inserted into the "custsum1" DataFrame.

```
custsum1[['has_AMEX','has_DISC','has_MC','has_VISA']].head(25)
Out[50]:
   has_AMEX has_DISC has_MC has_VISA
0         U        Y      Y        Y
1         U        U      U        U
2         U        U      U        U
3         U        U      Y        Y
4         U        U      Y        Y
5         U        U      U        U
6         U        U      U        Y
7         U        U      Y        U
8         U        U      U        U
9         U        U      U        U
10        U        U      U        U
```

Now, we can work towards creating our final "custsum" DataFrame. Since this is meant to be a “summary” DataFrame, we will select only the columns we’ve added in the previous steps, along with the account number (acctno), estimated HH income (med_inc), genders of adults "1" and "2" (adult1_g and adult2_g), ZIP code (zip) and ZIP+4 code (zip4) for our new and final "**custsum**" dataset.

```
 #####Create final Custsum DataFrame
custsum =
custsum1[['acctno','heavy_buyer','has_AMEX','has_DISC','has_MC','has_VISA','med_inc','adult1_g','adult2_g','zip','
zip4']]

custsum.shape
```

We can check the contents of our new "custsum" DataFrame to ensure we selected to correct columns and the values match the work we previously completed.

```
custsum.shape
Out[25]: (17491, 11)

custsum.head(5)
Out[26]:
      acctno heavy_buyer has_AMEX has_DISC has_MC has_VISA  med_inc adult1_g  adult2_g      zip    zip4
0  SSYSPDHQH           N        U        Y      Y        Y    73436        F         M  60068.0  1752.0
1  WQSDDAWQS           N        U        U      U        U    63157        M         F  60068.0  1148.0
2    HHSSPSD           N        U        U      U        U    85576        F         M  60068.0  5322.0
3  PHLHSYLWP           N        U        U      Y        Y    68499        F         M  60068.0  1570.0
4  PSDSQYYDW           N        U        U      U        Y    61570        F         M  60068.0  5447.0
```

Finally, we write **custSum** as a table to local xyz.db. We use sql-alchemy once again to help us write to the local SQLite DB xyz.db databases. There should be a total of 4 (four) tables in the database now.

```
######Write table to DB
conn=engine.connect()
custsum.to_sql('custsum',conn,index=False)

engine.table_names()
Out[51]: ['active_customer', 'active_item', 'active_mail', custsum']
```

We can now verify that the we have written the table correctly to our SQLite DB by querying the database for a count of the number of records in the table.

```
count = pd.read_sql_query("SELECT COUNT(*) FROM custsum",conn)
count
Out[4]:
   COUNT(*)
0     17491
```

```
count = pd.read_sql_query("SELECT COUNT(*) FROM active_mail",conn)
count
Out[6]:
   COUNT(*)
0     13714
count = pd.read_sql_query("SELECT COUNT(*) FROM active_item",conn)
count
Out[8]:
   COUNT(*)
0     77121
count = pd.read_sql_query("SELECT COUNT(*) FROM active_customer",conn)
count
Out[10]:
   COUNT(*)
0     17491
```

We can also read the table in from the database into a new DataFrame to verify the contents.

```
custsum_df2=pd.read_sql('custsum',conn)

custsum_df2.head(5)
Out[9]:
      acctno heavy_buyer has_AMEX has_DISC has_MC has_VISA  med_inc adult1_g  adult2_g      zip    zip4
0  SSYSPDHQH           N        U        Y      Y        Y    73436        F         M  60068.0  1752.0
1  WQSDDAWQS           N        U        U      U        U    63157        M         F  60068.0  1148.0
2    HHSSPSD           N        U        U      U        U    85576        F         M  60068.0  5322.0
3  PHLHSYLWP           N        U        U      Y        Y    68499        F         M  60068.0  1570.0
4  PSDSQYYDW           N        U        U      U        Y    61570        F         M  60068.0  5447.0
```

Part 4.Create a new DataFrame of data that will be used for target maketing and write it out to a headered csv file:

We will be creating a new DataFrame that only has one row per customer account. Those customer accounts should include 'ACTIVE' and 'LAPSED' customer, the total dollar amount of the purchases they have made from XYZ.

We begin by extracting the 'acctno','buyer_status', and 'ltd_sales'(total dollar amount) from the original "customer_df' Dataframe.

```
###Create ACTIVE + LAPSED + total dollar amount of customers purchases
lapsed_cust = customer_df[['acctno','buyer_status','ltd_sales']]
lapsed_cust.shape
lapsed_cust.head(5)
```

To extract only the 'ACTIVE' and 'LAPSED' records, we make use of .isin and create a separate DataFrame called "lapsed_cust2".

```
###Create dataframe with only ACTIVE and LAPSED customers
lapsed_cust2 = lapsed_cust.loc[lapsed_cust['buyer_status'].isin(['ACTIVE','LAPSED'])]
lapsed_cust2.shape
lapsed_cust2.head(5)
```

Since we only want one row per customer account, we must use .group on the original item_df to group by 'acctno' and 'deptdescr' and create a our new "market_item" DataFrame. Since this DataFrame is to be used as an indicator of the purchase of a particular product, we must also assign a '1' or '0' as indicators to our newly created DataFrame by applying (**lambda** *x*: np.where(x>0,1,0).

```
###Create new Dataframe for Item_df
market_item = item_df.groupby('acctno').deptdescr.value_counts()

###Change all value counts to indicate 1
market_item = market_item.apply(lambda x: np.where(x>0,1,0))
market_item.head(5)
```

We can now use .unstack to create a new DataFrame, "market_item2" separated by individual accounts while also resetting the index to make for a cleaner merger in the next step.

```
market_item2 = market_item.unstack(fill_value=0)
market_item2.reset_index(inplace=True)

market_item2.head(5)
```

The results produced by (market_item2.head(5)) were extracted and placed into a table for simplification.

	acctno	Appliances	Cameras & Camcorder Accessories	Home Audio	Mobile Electronic Accessories	Mobile Electronics	Portable Electronics	Small Appliances
0	AAAAPSSYY	0	0	0	0	0	1	0
1	AAAASDQWP	0	0	0	1	0	0	1
2	AAAASYHQW	0	0	0	0	1	0	0
3	AAAASYLYG	0	0	0	0	1	0	0
4	AAAASYPHD	0	0	0	1	0	0	0

Finally, we can merge , "market_item2" and "lapsed_cust2"by using pd.merge. Also of note, we exported the results of target_marketing_df.head(5) and target_marketing_df.tail(5) for simplification once again.

```
target_marketing_df=pd.merge(lapsed_cust2, market_item2, on='acctno', how='left').fillna(0)
target_marketing_df.shape
Out[21]: (27840, 10)

target_marketing_df.head(5)
```

	acctno	buyer_status	ltd_sales	Appliances	Cameras & Camcorder Accessories	Home Audio	Mobile Electronic Accessories	Mobile Electronics	Portable Electronics	Small Appliances
0	WHSDSLYQQ	LAPSED	354	0	0	0	0	0	0	0
1	SQWSAPYAY	LAPSED	564	0	0	0	0	0	0	0
2	SSYSPDHQH	ACTIVE	132	0	0	0	0	1	0	1
3	PDLGQDQQQ	LAPSED	1626	0	0	0	0	0	0	0
4	WQSDDAWQS	ACTIVE	1461	0	0	0	1	0	0	0

```
target_marketing_df.tail(5)
```

	acctno	buyer_status	ltd_sales	Appliances	Cameras & Camcorder Accessories	Home Audio	Mobile Electronic Accessories	Mobile Electronics	Portable Electronics	Small Appliances
0	WHSDSLYQQ	LAPSED	354	0	0	0	0	0	0	0
1	SQWSAPYAY	LAPSED	564	0	0	0	0	0	0	0
2	SSYSPDHQH	ACTIVE	132	0	0	0	0	1	0	1
3	PDLGQDQQQ	LAPSED	1626	0	0	0	0	0	0	0
4	WQSDDAWQS	ACTIVE	1461	0	0	0	1	0	0	0

To preserve this new "target_marketing_df' we will write it to a csv file and preserve it in a shelve database.

```
###Save to CVS
target_marketing_df.to_csv("target_marketing_df.csv",index=False)

##import shelve
import shelve
target_marketing = shelve.open('target_marketing_df')
```

We can use **ls to verify that the csv file and shelve database were written correctly.**

```
ls

 Directory of C:\Users\Noe
10/28/2017   08:58 PM         1,450,717 target_marketing_df.csv
10/29/2017   06:31 PM                 0 target_marketing_df.dat
10/29/2017   04:06 PM         8,705,024 xyz.db
              14 File(s)      15,361,725 bytes
              17 Dir(s)   372,555,542,528 bytes free
```

<u>Part 5 .Using the active customers' data For each gender code for adult_1,calculate and report the number of adults with this gender code, most frequently purchased products by gender, total spent in dollars on each category and the total number of products purchased in these categories.</u>

First thing we find the total number of adults in each gender category. We start by creating a new dataset from the original "item_df" dataset, selecting 'acctno','qty','price','totamt','deptdescr'.

```
###new item_df2
item_df2=item_df[["acctno","qty","price","totamt","deptdescr"]]
item_df2.head(4)
```

We can now use pd.merge to unite "item_df1" and our active customer dataset, "custsum" based on 'acctno'.

```
###Merg Custsum and new item_df
gender_df=pd.merge(item_df2, custsum, on='acctno', how='inner').fillna(0)
gender_df.head(3)
```

Now, we define a function to add a "gender" variable to our dataset in order to calculate the total number of adults per gender category. We can then add this new variable to the "gender_df".

```
def encode(adult1_g):
    if adult1_g == 'F':
        return 'F'
    if adult1_g == 'M':
        return 'M'
    if adult1_g == 'U':
        return 'U'
    if adult1_g == 'B':
        return 'B'

gender_df.insert(9,'gender',gender_df.adult1_g.map(encode))
```

Finally, we can use a crosstabs fucntion to find the total number of adults per gender category.

```
pd.crosstab(gender_df.gender,gender_df.adult1_g)
Out[55]:
adult1_g   B      F     M     U
gender
B         13      0     0     0
F          0  12629     0     0
M          0      0  3268     0
U          0      0     0  1581
```

Now, we are going to use groupby to find the total spent in each category by gender and the total quantity in each category as well.

```
gender_spent = gender_df.groupby(['adult1_g','deptdescr']).agg({'totamt': [sum]})
gender_spent
Out[91]:
                                               qty      totamt
                                               sum         sum
adult1_g deptdescr
B        Cameras & Camcorder Accessori           1      104.97
         Home Audio                              2      719.70
         Mobile Electronic Accessories          10      472.47
         Mobile Electronics                     14      471.87
         Portable Electronics                    3      834.00
         Small Appliances                       10      848.22
F        Appliances                              2     1739.85
         Cameras & Camcorder Accessori        2639   321097.20
         Home Audio                           7088  2171664.39
         Mobile Electronic Accessories       18255   760369.65
         Mobile Electronics                  13974   800757.84
         Portable Electronics                 6104   745500.27
         Small Appliances                    14622   956301.03
M        Appliances                              2     2100.00
         Cameras & Camcorder Accessori         666    89132.16
         Home Audio                           1948   655876.92
         Mobile Electronic Accessories        3544   150226.02
         Mobile Electronics                   2299   144819.54
         Portable Electronics                 1440   206839.29
         Small Appliances                     3417   236165.61
U        Appliances                              1     2997.00
         Cameras & Camcorder Accessori        3647   390842.25
         Home Audio                            952   318808.20
         Mobile Electronic Accessories        2103    86722.92
         Mobile Electronics                   1647    91468.41
         Portable Electronics                  938   115611.87
         Small Appliances                     1958   125988.45
```

finally, we can return the top six most frequently purchased product categories by gender, using groupby in pandas.

```
gender_spent2 = gender_df.groupby(['adult1_g','deptdescr']).size()

gender_spent2
Out[51]:
adult1_g  deptdescr
B         Cameras & Camcorder Accessori        1
          Home Audio                           2
          Mobile Electronic Accessories       10
          Mobile Electronics                  14
          Portable Electronics                 3
```

```
          Small Appliances                10
F         Appliances                       2
          Cameras & Camcorder Accessori  2343
          Home Audio                     6336
          Mobile Electronic Accessories 17164
          Mobile Electronics            12443
          Portable Electronics           5155
          Small Appliances              14106
M         Appliances                       2
          Cameras & Camcorder Accessori   574
          Home Audio                     1771
          Mobile Electronic Accessories  3379
          Mobile Electronics             2124
          Portable Electronics           1212
          Small Appliances               3342
U         Appliances                       1
          Cameras & Camcorder Accessori   320
          Home Audio                      845
          Mobile Electronic Accessories  1981
          Mobile Electronics             1413
          Portable Electronics            687
          Small Appliances               1881
dtype: int64
```