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:

- Analyze the dataset in the given CSV file
- Clean the given dataset
- Load the dataset into sqlite database engine
- Execute different 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.

Data Preparation

As a data scientist for BestDeal retailer, you have been tasked with improving their revenue and the effectiveness of the marketing campaign of their electronic products. The given dataset has 10,000 records for the purchases of their customers and is used to predict customers shopping patterns and to provide answers for ad-hoc queries. The dataset DirtyData4BestDeal10000.csv is drawn from its database of customers.

In [70]:
```
import pandas as pd  # panda's nickname is pd

import numpy as np  # numpy as np

from pandas import DataFrame, Series     # for convenience

import sqlalchemy

from sqlalchemy import create_engine

from sqlalchemy import inspect
```

Lets ead the dirtydata4bestdeal CSV and load into a dataframe object

In [71]:
```
dirtydata4bestdeal=pd.read_csv('DirtyData4BestDeal10000.csv')
```

In [72]:
```
# Do you see NaN values below?

dirtydata4bestdeal.head()
```

Out[72]:

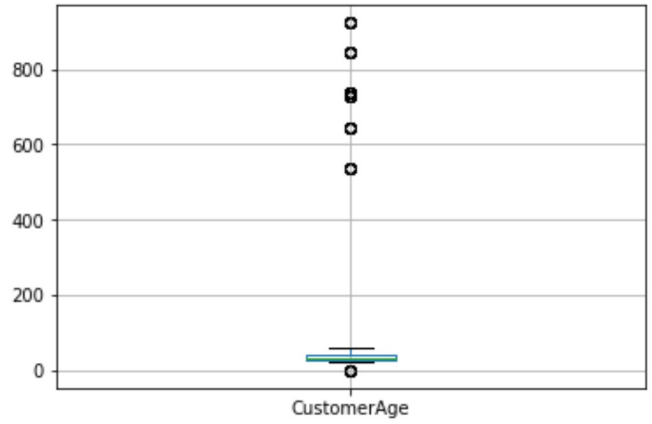
	ZipCode	CustomerAge	SamsungTV46LED	SonyTV42LED	XBOX360	DellLaptop	BoseSoundSystem	BoseHea
0	30134.0	35.0	1	1	1	0	0	
1	62791.0	43.0	0	1	0	0	1	
2	60611.0	23.0	1	NaN	0	1	0	
3	60616.0	56.0	0	1	1	1	0	
4	30303.0	25.0	1	NaN	0	NaN	1	

5 rows × 34 columns

Lets use boxplot to visualize the data and get an idea if there are dirty/messy /invalid data

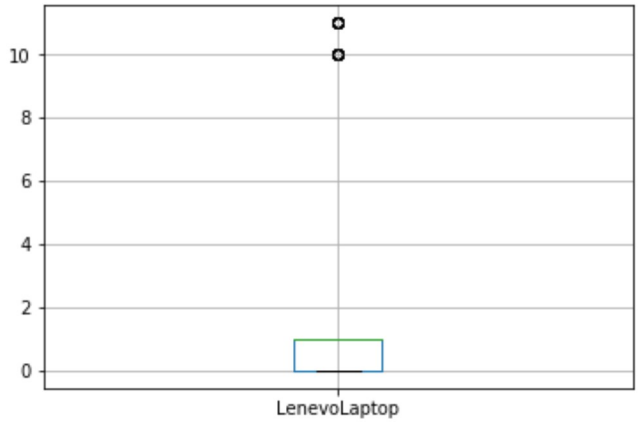
In [73]:
```
dirtydata4bestdeal.boxplot(column='CustomerAge')
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x22461f21f28>



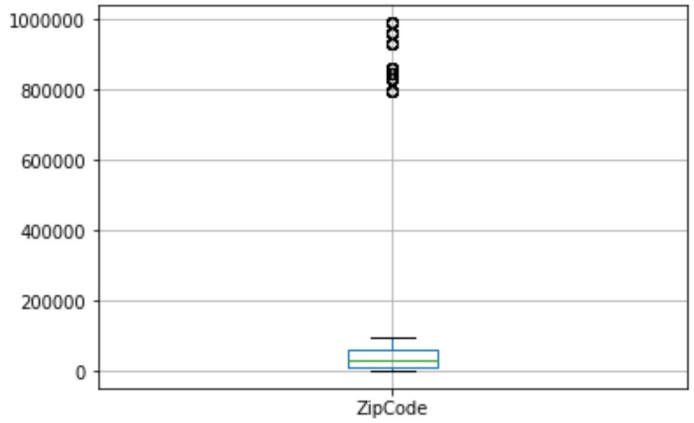
In [74]:
```
dirtydata4bestdeal.boxplot(column='LenevoLaptop')
```

Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x22462675240>



In [75]:
```
dirtydata4bestdeal.boxplot(column='ZipCode')
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x22462052cc0>



Lets clean the dirty/messy data in the dirtydata4bestdeal dataframe object

```
You need to write your python code such that:
1. rows/records/tuples/transactions in the data frame that have missing values for
fields/columns will be removed
2. rows/records/tuples/transactions in the data frame that have invalid/abnormal
values for fields/columns will be removed



Examples of invalid/dirty/messy data:
  1. NaN values in the dataframe (Blank/Empty cells in the CSV file)

  2. Every product has a value 1 which means bought or 0 which means NOT bought;
values like 11, 10, 9 are examples of invalid data

  3. CustomerAge value range could be from 18 to 150; values like 723, 634 are
examples of invalid data
```

In [76]:
```
# Drop the NaN values

cleandata4bestdeal=dirtydata4bestdeal.dropna()
cleandata4bestdeal.head()

# Do you see NaN values dropped below?
```

Out[76]:

	ZipCode	CustomerAge	SamsungTV46LED	SonyTV42LED	XBOX360	DellLaptop	BoseSoundSystem	BoseHea
0	30134.0	35.0	1	1	1	0	0	
1	62791.0	43.0	0	1	0	0	1	
3	60616.0	56.0	0	1	1	1	0	
5	2108.0	55.0	1	1	1	1	10	
6	90033.0	44.0	1	1	1	1	0	

5 rows × 34 columns

In [77]:
```
# Add the rest of your code here to clean the data
```

Lets store the cleaned data into the Database

In [78]:
```
engine=create_engine('sqlite:///bestdeal.db')
```

In [79]:
```
cleandata4bestdeal.to_sql('trans4cust', engine)
```

Sanity Test: Did it create the table in bestdeal.db? Check!!

In [80]:
```
insp=inspect(engine)
```

In [81]:
```
insp.get_table_names()
```

Out[81]: ['trans4cust']

In [82]:
```
pd.read_sql_table('trans4cust', engine).columns
```

Out[82]:
```
Index(['index', 'ZipCode', 'CustomerAge', 'SamsungTV46LED', 'SonyTV42LED',
       'XBOX360', 'DellLaptop', 'BoseSoundSystem', 'BoseHeadSet',
       'SonyHeadSet', 'iPod', 'iPhone', 'Panasonic50LED', 'SonyPS4', 'WiiU',
       'WDexternalHD', 'SamsungTV55LED', 'SonyTV60LED', 'SandiskMemoryCard',
       'SonySoundSystem', 'SonyCamera', 'PanasonicCamera', 'HPPrinter',
       'SonyDVDplayer', 'ToshibaDVDplayer', 'GalaxyTablet', 'SurfaceTablet',
       'HPLaptop', 'HDMICable', 'SpeakerCable', 'CallOfDutyGame',
       'GrandTheftAutoGame', 'ASUSLaptop', 'LenevoLaptop', 'TVStandWallMount'],
      dtype='object')
```

```
should produce the columns of the DataFrame you wrote to the db
```

Now we are ready to query the Database

Query example #1: get the transactions for the customers in zipCode 60616

In [83]:
```
resultsForBestDealCustTrans=pd.read_sql_query("SELECT * FROM trans4cust WHERE ZipCode=
```

In [84]:
```
resultsForBestDealCustTrans.head()
```

Out[84]:

	index	ZipCode	CustomerAge	SamsungTV46LED	SonyTV42LED	XBOX360	DellLaptop	BoseSoundSystem	B
0	3	60616.0	56.0	0	1	1	1	0	
1	16	60616.0	43.0	0	1	1	0	1	
2	18	60616.0	54.0	1	0	0	1	0	
3	23	60616.0	43.0	1	1	1	0	1	
4	34	60616.0	31.0	0	1	1	1	0	

5 rows × 35 columns

Query example #2: get the transactions for ALL customers

In [85]:
```
resultsForBestDealCustTrans=pd.read_sql_query("SELECT * FROM trans4cust", engine)
```

In [86]:
```
resultsForBestDealCustTrans.head()
```

Out[86]:

	index	ZipCode	CustomerAge	SamsungTV46LED	SonyTV42LED	XBOX360	DellLaptop	BoseSoundSystem	B
0	0	30134.0	35.0	1	1	1	0	0	
1	1	62791.0	43.0	0	1	0	0	1	
2	3	60616.0	56.0	0	1	1	1	0	
3	5	2108.0	55.0	1	1	1	1	10	
4	6	90033.0	44.0	1	1	1	1	0	

5 rows × 35 columns

Query example #3: get the number of customers in every ZipCode sorted by ZipCode

In [87]:
```
resultsForBestDealCustTrans=pd.read_sql_query("SELECT ZipCode , COUNT(*) as 'num_custo
```

In [88]:
```
resultsForBestDealCustTrans
```

Out[88]:

	ZipCode	num_customers
0	2108.0	632
1	2109.0	955
2	2110.0	224
3	10065.0	788
4	30134.0	1173
5	30303.0	1001
6	33129.0	554
7	33130.0	280
8	44114.0	526
9	60532.0	243
10	60585.0	248
11	60603.0	240
12	60611.0	62
13	60616.0	960
14	62791.0	3
15	90024.0	144
16	90033.0	665
17	94102.0	166
18	94158.0	512
19	794158.0	8
20	830134.0	8
21	844114.0	8
22	860616.0	8
23	930134.0	8
24	960616.0	8
25	990033.0	8

Query example #4: get the number of customers for every Age Group in ZipCode 60616 sorted by CustomerAge

In [89]:
```
resultsForBestDealCustTrans=pd.read_sql_query(
"SELECT CustomerAge , COUNT(*) as 'num_customers' FROM trans4cust WHERE ZipCode=60616
```

In [90]:
```
resultsForBestDealCustTrans
```

Out[90]:

	CustomerAge	num_customers
0	21.0	56
1	22.0	32
2	23.0	40
3	25.0	88
4	26.0	48
5	27.0	32
6	28.0	32
7	29.0	56
8	31.0	16
9	32.0	16
10	34.0	96
11	35.0	72
12	37.0	64
13	38.0	24
14	39.0	8
15	43.0	48
16	44.0	88
17	45.0	24
18	46.0	24
19	51.0	8
20	54.0	48
21	56.0	32
22	727.0	8

Query example #5: Plot in a stacked-bar figure the number of customers who bought SonyTV60LED and/or BoseSoundSystem in every zipcode that has more than 400 customers who bought these two products(either bought one of these products or the two products)

In [91]:
```
SonyTV60LEDCustTrans=pd.read_sql_query(
"SELECT ZipCode , COUNT(*) as 'num_customers' FROM trans4cust WHERE SonyTV60LED=1  GRO

BoseSoundSystemCustTrans=pd.read_sql_query(
"SELECT ZipCode , COUNT(*) as 'num_customers' FROM trans4cust WHERE BoseSoundSystem=1
```

In [92]:
```
SonyTV60LEDCustTrans
```

Out[92]:

	ZipCode	num_customers
0	2108.0	416
1	2109.0	611
2	10065.0	467
3	30134.0	774
4	30303.0	524
5	60616.0	697

In [93]:
```
BoseSoundSystemCustTrans
```

Out[93]:

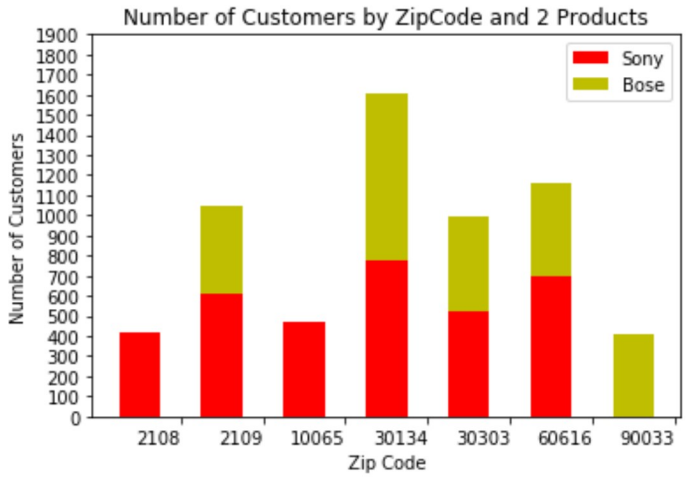
	ZipCode	num_customers
0	2109.0	436
1	30134.0	832
2	30303.0	472
3	60616.0	467
4	90033.0	406

In [94]:
```
SonyTV60LEDCustTrans.ZipCode
```

Out[94]:
```
0     2108.0
1     2109.0
2    10065.0
3    30134.0
4    30303.0
5    60616.0
Name: ZipCode, dtype: float64
```

In [95]:
```
import numpy

#   There are zipcodes that Sony got bought but not Bose
#   but there are also zipcodes that Bose got bought but not Sony
#
#   AND we need to use stacked-bar graph and we have a potentially asymmetrical set  c
#   So, we need to do somework to create the symmteric set of zipcode values for Sony


sonyZipCodeTuples=tuple(SonyTV60LEDCustTrans.ZipCode.astype(numpy.int))
sony_num_customersTuples=tuple(SonyTV60LEDCustTrans.num_customers.astype(numpy.int))

boseZipCodeTuples=tuple(BoseSoundSystemCustTrans.ZipCode.astype(numpy.int))
bose_num_customersTuples=tuple(BoseSoundSystemCustTrans.num_customers.astype(numpy.int




sony_dict = dict(zip(sonyZipCodeTuples, sony_num_customersTuples))
bose_dict = dict(zip(boseZipCodeTuples, bose_num_customersTuples))

for key in bose_dict.keys():
    if ((key in sony_dict.keys()) == False): sony_dict[key]=0


for key in sony_dict.keys():
    if ((key in bose_dict.keys()) == False): bose_dict[key]=0



bose_zip= sorted(bose_dict.keys())

sony_zip= sorted(sony_dict.keys())

bose_zip_tuple=tuple(bose_zip)

sony_zip_tuple=tuple(sony_zip)

bose_customer_list=[]

for bose in bose_zip_tuple:
    bose_customer_list.append(bose_dict[bose])

sony_customer_list=[]

for sony in sony_zip_tuple:
    sony_customer_list.append(sony_dict[sony])

bose_customer_tuple=tuple(bose_customer_list)
sony_customer_tuple=tuple(sony_customer_list)
```

In [96]:
```
# See docs for bar_stack at the URL
# http://matplotlib.org/examples/pylab_examples/bar_stacked.html

import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline


ind = np.arange(len(sony_customer_tuple))


# the width of the bars: can also be len(x) sequence
width = .5


p1 = plt.bar(ind, sony_customer_tuple, width,  color='r')
p2 = plt.bar(ind, bose_customer_tuple, width, color='y', bottom=sony_customer_tuple)


plt.ylabel('Number of Customers')
plt.xlabel('Zip Code')

plt.title('Number of Customers by ZipCode and 2 Products')

plt.xticks(ind + width, sony_zip_tuple, horizontalalignment='right')

plt.yticks(np.arange(0, 2000, 100))
plt.legend((p1[0], p2[0]), ('Sony', 'Bose'))

plt.show()
```



Requirements :

1. (Use SQL/SQlite): get the number of customers who bought DellLaptop and HPPrinter for every Age group sorted by CustomerAge
2. (Use SQL/SQlite): Get the list of ZipCodes where no customer bought XBOX360 (this query means NOT even a single csutomer in that zip code bought XBOX360)
3. (Use SQL/SQlite/Matplotlib): Plot in a stacked-bar figure the number of customers who bought HPLaptop and/or HPPrinter but did NOT buy WDexternalHD for every CustomerAge group that has more than 100 customers who bought these two products(either bought one of these products or the two products but didn't buy WDexternalHD)

In [97]:
```
# Write your python code that meets the above requirements in this cell
```